Bright Cluster Manager 7.3

Administrator Manual

Revision: 9293115
Date: Wed Dec 6 2023
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. PGI is a registered trademark of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. ScaleMP is a registered trademark of ScaleMP, 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 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 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 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 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

0.1 Quickstart ........................................... xiii
0.2 About This Manual ................................. xiii
0.3 About The Manuals In General ................. xiii
0.4 Getting Administrator-Level Support .......... xiv
0.5 Getting Professional Services .................. xiv

1 Introduction ........................................ 1
  1.1 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 Bright Cluster Manager Tools And Applications .................. 2
  1.3 Outside The Direct Scope Of The Administrator Manual ...... 2

2 Cluster Management With 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
  2.2 Modules Environment ............................... 8
      2.2.1 Adding And Removing Modules .................. 8
      2.2.2 Using Local And Shared Modules ................ 9
      2.2.3 Setting Up A Default Environment For All Users ...... 9
      2.2.4 Creating A Modules Environment Module ........... 10
      2.2.5 Lua Modules Environment (LMod) ................ 10
  2.3 Authentication ..................................... 11
      2.3.1 Changing Administrative Passwords On The Cluster .... 11
      2.3.2 Logins Using ssh ............................... 12
      2.3.3 Certificates .................................... 13
      2.3.4 Profiles ....................................... 13
  2.4 Cluster Management GUI ............................ 14
      2.4.1 Installing Cluster Management GUI On The Desktop ...... 14
      2.4.2 Navigating The Cluster Management GUI .............. 19
      2.4.3 Advanced cmgui Features ....................... 22
  2.5 Cluster Management Shell .......................... 25
      2.5.1 Invoking cmsh .................................. 26
      2.5.2 Levels, Modes, Help, And Commands Syntax In cmsh ........ 27
      2.5.3 Working With Objects ........................... 31
# Table of Contents

2.5.4 Accessing Cluster Settings .................................. 39
2.5.5 Advanced cmsh Features .................................. 40
2.6 Cluster Management Daemon .................................. 47
   2.6.1 Controlling The Cluster Management Daemon ............... 47
   2.6.2 Configuring The Cluster Management Daemon ............... 48
   2.6.3 Configuring The Cluster Management Daemon Logging Facilities ............... 48
   2.6.4 Configuration File Modification, And The FrozenFile Directive .......... 49
   2.6.5 Configuration File Conflicts Between The Standard Distribution And Bright Cluster Manager For Generated And Non-Generated Files .......... 50

3 Configuring The Cluster .................................. 51
   3.1 Main Cluster Configuration Settings .............................. 52
      3.1.1 Cluster Configuration: Various Name-Related Settings ............... 53
      3.1.2 Cluster Configuration: Some Network-Related Settings ............... 53
      3.1.3 Miscellaneous Settings .................................. 54
      3.1.4 Limiting The Maximum Number Of Open Files ............... 55
   3.2 Network Settings .................................. 56
      3.2.1 Configuring Networks .................................. 57
      3.2.2 Adding Networks .................................. 60
      3.2.3 Changing Network Parameters .................................. 60
   3.3 Configuring Bridge Interfaces .................................. 71
   3.4 Configuring VLAN interfaces .................................. 73
      3.4.1 Configuring A VLAN Interface Using cmsh ............... 73
      3.4.2 Configuring A VLAN Interface Using cmgui ............... 73
   3.5 Configuring Bonded Interfaces .................................. 74
      3.5.1 Adding A Bonded Interface .................................. 74
      3.5.2 Single Bonded Interface On A Regular Node ............... 75
      3.5.3 Multiple Bonded Interface On A Regular Node ............... 75
      3.5.4 Bonded Interfaces On Head Nodes And HA Head Nodes ............... 76
      3.5.5 Tagged VLAN On Top Of a Bonded Interface ............... 76
      3.5.6 Further Notes On Bonding .................................. 76
   3.6 Configuring InfiniBand And Omni-Path Interfaces ............... 77
      3.6.1 Installing Software Packages .................................. 77
      3.6.2 Subnet Managers .................................. 77
      3.6.3 InfiniBand Network Settings .................................. 78
      3.6.4 Verifying Connectivity .................................. 79
   3.7 Configuring BMC (IPMI/iLO) Interfaces .................................. 80
      3.7.1 BMC Network Settings .................................. 81
      3.7.2 BMC Authentication .................................. 82
      3.7.3 Interfaces Settings .................................. 83
   3.8 Configuring Switches And PDUs .................................. 83
      3.8.1 Configuring With The Manufacturer’s Configuration Interface ............... 83
      3.8.2 Configuring SNMP .................................. 83
      3.8.3 Uplink Ports .................................. 84
      3.8.4 The showport MAC Address to Port Matching Tool ............... 84
      3.8.5 Disabling Port Detection .................................. 85
      3.8.6 The switchoverview Command .................................. 86
### Table of Contents

3.9 Disk Layouts: Disked, Semi-Diskless, And Diskless Node Configuration .......................... 86
   3.9.1 Disk Layouts ........................................... 86
   3.9.2 Disk Layout Assertions ................................. 87
   3.9.3 Changing Disk Layouts .................................. 87
   3.9.4 Changing A Disk Layout From Disked To Diskless .................................................. 87
3.10 Configuring NFS Volume Exports And Mounts ......................................................... 88
   3.10.1 Exporting A Filesystem Using cmgui And cmsh ................................................ 90
   3.10.2 Mounting A Filesystem Using cmgui And cmsh .................................................. 91
   3.10.3 Mounting A Filesystem Subtree For A Diskless Node Over NFS ........................... 94
   3.10.4 Mounting The Root Filesystem For A Diskless Node Over NFS ........................... 96
   3.10.5 Configuring NFS Volume Exports And Mounts Over RDMA With OFED Drivers .......... 98
3.11 Managing And Configuring Services ............................................................................ 99
   3.11.1 Why Use The Cluster Manager For Services? ...................................................... 99
   3.11.2 Managing And Configuring Services—Examples .................................................... 100
3.12 Managing And Configuring A Rack .............................................................................. 104
   3.12.1 Racks .......................................................... 104
   3.12.2 Rack View ..................................................... 106
   3.12.3 Assigning Devices To A Rack ............................................ 109
   3.12.4 Assigning Devices To A Chassis .................................................. 110
   3.12.5 An Example Of Assigning A Device To A Rack, And Of Assigning A Device To A Chassis .......................................................... 121
3.13 Configuring A GPU Unit, And Configuring GPU Settings ........................................... 122
   3.13.1 GPUs And GPU Units ........................................... 122
   3.13.2 GPU Unit Configuration Example: The Dell PowerEdge C410x ........................... 122
   3.13.3 Configuring GPU Settings ........................................ 124
3.14 Configuring Custom Scripts ...................................................................................... 127
   3.14.1 custompowerscript .................................................. 127
   3.14.2 custompingscript ............................................... 128
   3.14.3 customremoteconsolescript ...................................... 128
3.15 Cluster Configuration Without Execution By CMDaemon ............................................ 128
   3.15.1 Cluster Configuration: The Bigger Picture ...................................................... 128
   3.15.2 Making Nodes Function Differently By Image .................................................. 129
   3.15.3 Making All Nodes Function Differently From Normal Cluster Behavior With FrozenFile .......................................................... 131
   3.15.4 Adding Functionality To Nodes Via An initialize Or finalize Script ........................ 131
   3.15.5 Examples Of Configuring Nodes With Or Without CMDaemon ........................... 132
4 Power Management .................................................................................................. 135
   4.1 Configuring Power Parameters .................................................................................. 135
      4.1.1 PDU-Based Power Control ........................................ 136
      4.1.2 IPMI-Based Power Control ........................................ 138
      4.1.3 Combining PDU- and IPMI-Based Power Control .................. 138
      4.1.4 Custom Power Control ............................................ 139
      4.1.5 Hewlett Packard iLO-Based Power Control ......................... 140
   4.2 Power Operations ................................................................................................. 141
      4.2.1 Power Operations With cmgui ........................................ 141
      4.2.2 Power Operations Through cmsh ..................................... 143
# Table of Contents

4.3 Monitoring Power ................................................. 145

4.4 CPU Scaling Governors ........................................ 145

4.4.1 The Linux Kernel And CPU Scaling Governors .......... 145

4.4.2 The Governor List According To `sysinfo` ............... 145

4.4.3 Setting The Governor ........................................ 146

5 Node Provisioning .................................................. 149

5.1 Before The Kernel Loads ....................................... 149

5.1.1 PXE Booting .................................................. 149

5.1.2 iPXE Booting From A Disk Drive ......................... 153

5.1.3 iPXE Booting Using InfiniBand ......................... 153

5.1.4 Booting From The Drive .................................. 153

5.1.5 The Boot Role ................................................ 153

5.2 Provisioning Nodes .............................................. 154

5.2.1 Provisioning Nodes: Configuration Settings .......... 154

5.2.2 Provisioning Nodes: Role Setup With `cmsh` .......... 155

5.2.3 Provisioning Nodes: Role Setup With `cmgui` ........ 156

5.2.4 Provisioning Nodes: Housekeeping ..................... 157

5.3 The Kernel Image, Ramdisk And Kernel Modules .......... 160

5.3.1 Booting To A “Good State” Software Image .......... 160

5.3.2 Selecting Kernel Driver Modules To Load Onto Nodes .. 161

5.3.3 InfiniBand Provisioning .................................. 162

5.4 Node-Installer .................................................... 164

5.4.1 Requesting A Node Certificate ......................... 165

5.4.2 Deciding Or Selecting Node Configuration ............ 166

5.4.3 Starting Up All Network Interfaces .................... 177

5.4.4 Determining Install-mode Type And Execution Mode .. 178

5.4.5 Running Initialize Scripts ............................... 184

5.4.6 Checking Partitions, RAID Configuration, Mounting Filesystems ............................................. 184

5.4.7 Synchronizing The Local Drive With The Software Image .................................................. 185

5.4.8 Writing Network Configuration Files ................... 189

5.4.9 Creating A Local `/etc/fstab` File ....................... 190

5.4.10 Installing GRUB Bootloader ............................... 190

5.4.11 Running Finalize Scripts ................................. 191

5.4.12 Unloading Specific Drivers ............................... 191

5.4.13 Switching To The Local `init` Process ................. 192

5.5 Node States ....................................................... 192

5.5.1 Node States Icons In `cmgui` ............................ 192

5.5.2 Node States Shown In `cmsh` ............................. 192

5.5.3 Node States Indicating Regular Start Up .............. 193

5.5.4 Node States That May Indicate Problems .............. 193

5.6 Updating Running Nodes ....................................... 196

5.6.1 Updating Running Nodes: Configuration With `excludelistupdate` .............................................. 196

5.6.2 Updating Running Nodes: With `cmsh` Using `imageupdate` .................................................. 201

5.6.3 Updating Running Nodes: With `cmgui` Using The `Update node` Button ........................................ 202

5.6.4 Updating Running Nodes: Considerations ................ 202

5.7 Adding New Nodes ............................................... 202
7.5.3 Installing, Configuring, And Running UGE ........................................... 256
7.5.4 Configuring And Running Torque ....................................................... 262
7.5.5 Configuring And Running PBS Pro ...................................................... 265
7.5.6 Installing, Configuring And Running OpenLava ................................... 266
7.5.7 Installing, Configuring, And Running LSF ........................................... 268

7.6 Using cmgui With Workload Management ........................................... 273
7.6.1 Jobs Display And Handling In cmgui .................................................. 273
7.6.2 Queues Display And Handling In cmgui .............................................. 274
7.6.3 Nodes Display And Handling In cmgui ............................................... 277

7.7 Using cmsh With Workload Management ............................................. 278
7.7.1 Jobs Display And Handling In cmsh: jobs Mode .................................... 278
7.7.2 Job Queues Display And Handling In cmsh: jobqueue Mode ................. 280
7.7.3 Nodes Drainage Status And Handling In cmsh ....................................... 282
7.7.4 Launching Jobs With cm-launcher ..................................................... 284

7.8 Examples Of Workload Management Assignment ................................... 285
7.8.1 Setting Up A New Category And A New Queue For It ............................ 285
7.8.2 Setting Up A Prejob Health Check ..................................................... 287

7.9 Power Saving Features ........................................................................... 289
7.9.1 Slurm ................................................................................................. 289
7.9.2 The cm-scale-cluster Utility ............................................................... 290

7.10 Cgroups ............................................................................................... 305
7.10.1 Cgroups Settings For Workload Managers ......................................... 305
7.10.2 Managed Cgroups ............................................................................. 310

8 Containerization ......................................................................................... 315
8.1 Docker Engine ......................................................................................... 315
8.1.1 Docker Setup ..................................................................................... 316
8.1.2 Integration With Workload Managers ............................................... 317
8.1.3 DockerHost Role ............................................................................... 318
8.1.4 Iptables ............................................................................................ 320
8.1.5 Storage Backends ............................................................................. 320
8.1.6 The Container Daemon Service And RunC Runtime ......................... 323
8.1.7 Docker Monitoring ............................................................................ 324

8.2 Kubernetes ............................................................................................. 325
8.2.1 Kubernetes Setup ............................................................................. 325
8.2.2 Disabling Kubernetes ....................................................................... 331
8.2.3 Kubernetes Cluster .......................................................................... 332
8.2.4 Kubernetes Roles ............................................................................. 334
8.2.5 Security Model .................................................................................. 341
8.2.6 Networking Model ............................................................................ 343
8.2.7 Kubernetes Monitoring ................................................................. 345
8.2.8 Files Managed By CMDaemon ......................................................... 345

8.3 Singularity ............................................................................................... 346
8.3.1 Use Cases ......................................................................................... 346
8.3.2 Package cm-singularity ................................................................. 346
8.3.3 MPI Integration ................................................................................. 347
9 Post-Installation Software Management

9.1 Bright Cluster Manager RPM Packages And Their Naming Convention

9.2 Managing Packages On The Head Node

9.2.1 Managing RPM Packages On The Head Node

9.2.2 Managing Non-RPM Software On The Head Node

9.3 Kernel Management On A Head Node Or Image

9.3.1 Installing A Standard Distribution Kernel

9.3.2 Excluding Kernels And Other Packages From Updates

9.3.3 Updating A Kernel In A Software Image

9.3.4 Setting Kernel Options For Software Images

9.3.5 Kernel Driver Modules

9.4 Managing An RPM Package In A Software Image And Running It On Nodes

9.4.1 Installing From Head Via chroot: Installing Into The Image

9.4.2 Installing From Head Via chroot: Updating The Node

9.4.3 Installing From Head Via rpm --root, yum --installroot Or chroot: Possible Issues

9.5 Managing Non-RPM Software In A Software Image And Running It On Nodes

9.5.1 Managing The Software Directly On An Image

9.5.2 Managing The Software Directly On A Node, Then Syncing Node-To-Image

9.6 Creating A Custom Software Image

9.6.1 Creating A Base Distribution Archive From A Base Host

9.6.2 Creating The Software Image With cm-create-image

9.6.3 Configuring Local Repositories For Linux Distributions, And For The Bright Cluster Manager Package Repository, For A Software Image

9.6.4 Creating A Custom Image From The Local Repository

10 Cluster Monitoring

10.1 A Basic Example Of How Monitoring Works

10.1.1 Before Using The Framework—Setting Up The Pieces

10.1.2 Using The Framework

10.2 Monitoring Concepts And Definitions

10.2.1 Metric

10.2.2 Action

10.2.3 Threshold

10.2.4 Health Check

10.2.5 Conceptual Overview: Health Checks Vs Threshold Checks

10.2.6 Severity

10.2.7 AlertLevel

10.2.8 InfoMessages

10.2.9 Flapping

10.2.10 Transition

10.2.11 Conceptual Overview: cmgui's Main Monitoring Interfaces

10.3 Monitoring Visualization With cmgui

10.3.1 The Monitoring Window

10.3.2 The Graph Display Pane

10.3.3 Using The Grid Wizard

10.3.4 Zooming In With Mouse Gestures
10.3.5 The Graph Display Settings Dialog ........................................... 393
10.4 Monitoring Configuration With cmgui ........................................... 394
10.4.1 The Overview Tab ................................................................. 394
10.4.2 The Metric Configuration Tab .................................................. 395
10.4.3 Health Check Configuration Tab ................................................ 400
10.4.4 Metrics Tab ............................................................................. 403
10.4.5 Health Checks Tab .................................................................... 407
10.4.6 Actions Tab .............................................................................. 408
10.5 Overview Of Monitoring Data For Devices ........................................ 408
10.6 Event Viewer ................................................................................. 409
10.6.1 Viewing Events In cmgui .......................................................... 409
10.6.2 Viewing Events In cmsh ............................................................. 410
10.6.3 Using The Event Bucket From The Shell For Events And For Tagging Device States .................................................. 413
10.7 The monitoring Modes Of cmsh ....................................................... 414
10.7.1 The monitoring actions Mode In cmsh ........................................ 415
10.7.2 The monitoring healthchecks Mode In cmsh ............................... 417
10.7.3 The monitoring metrics Mode In cmsh ........................................ 420
10.7.4 The monitoring setup Mode In cmsh .......................................... 421
10.8 Obtaining Monitoring Data Values .................................................. 428
10.8.1 The metrics and healthchecks Commands .................................... 428
10.8.2 On-Demand Metric Sampling And Health Checks ......................... 429
10.8.3 The Latest Data Values—The latest*data Commands .................... 430
10.8.4 Filtering Monitoring Data Values With monitoringdatafilter ........ 431
10.8.5 The showhealth Command For An Overview Of Health State .......... 432
10.8.6 Data Values Over Time—The dump* Commands .......................... 432
10.8.7 Setting Fictional Data Values—The fakedata Command ................ 436
10.9 The User Portal .............................................................................. 437
10.9.1 Accessing The User Portal .......................................................... 437
10.9.2 Setting A Common Username/Password For The User Portal ........... 438
10.9.3 User Portal Home Page ............................................................. 439
10.10 Job Monitoring ............................................................................ 440
10.10.1 Job Metrics .............................................................................. 440
10.10.2 Job Metrics Collection Setup ..................................................... 446
10.10.3 Job Information Retention ........................................................ 446
10.10.4 Job Metrics Sampling Configuration .......................................... 447
10.10.5 Job Monitoring In cmsh ............................................................ 448
10.10.6 Job Metrics Monitoring With cmgui ......................................... 449

11 Day-to-day Administration ................................................................ 453
11.1 Parallel Shells: pdsh And pexec .................................................... 453
11.1.1 pdsh In The OS Shell ............................................................... 453
11.1.2 pexec In cmsh ......................................................................... 457
11.1.3 pexec In cmgui ....................................................................... 457
11.1.4 Using The -j|--join Option Of pexec ......................................... 458
11.1.5 Other Parallel Commands ......................................................... 459
11.2 Getting Support With Cluster Manager Issues .................................. 459
11.2.1 Support Via E-mail ................................................................... 459
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.2.2 Reporting Cluster Manager Diagnostics With cm-diagnose</td>
<td>460</td>
</tr>
<tr>
<td>11.2.3 Requesting Remote Support With request-remote-assistance</td>
<td>461</td>
</tr>
<tr>
<td>11.2.4 Requesting Remote Support With A Shared Screen Utility</td>
<td>462</td>
</tr>
<tr>
<td>11.3 Backups</td>
<td>462</td>
</tr>
<tr>
<td>11.3.1 Cluster Installation Backup</td>
<td>462</td>
</tr>
<tr>
<td>11.3.2 Local Database Backups And Restoration</td>
<td>463</td>
</tr>
<tr>
<td>11.4 Revision Control For Images</td>
<td>466</td>
</tr>
<tr>
<td>11.4.1 Btrfs: The Concept And Why It Works Well In Revision Control For Images</td>
<td>466</td>
</tr>
<tr>
<td>11.4.2 Btrfs Availability And Distribution Support</td>
<td>466</td>
</tr>
<tr>
<td>11.4.3 Installing Btrfs To Work With Revision Control Of Images In Bright Cluster Manager</td>
<td>467</td>
</tr>
<tr>
<td>11.4.4 Using cmsh For Revision Control Of Images</td>
<td>468</td>
</tr>
<tr>
<td>11.5 BIOS Configuration And Updates</td>
<td>471</td>
</tr>
<tr>
<td>11.5.1 BIOS Configuration</td>
<td>471</td>
</tr>
<tr>
<td>11.5.2 Updating BIOS</td>
<td>472</td>
</tr>
<tr>
<td>11.5.3 Booting DOS Image</td>
<td>472</td>
</tr>
<tr>
<td>11.6 Hardware Match Check</td>
<td>473</td>
</tr>
<tr>
<td>11.7 Serial Over LAN Console Access</td>
<td>474</td>
</tr>
<tr>
<td>11.7.1 Background Notes On Serial Console And SOL</td>
<td>474</td>
</tr>
<tr>
<td>11.7.2 SOL Console Configuration And Access With cmgui</td>
<td>476</td>
</tr>
<tr>
<td>11.7.3 SOL Console Configuration And Access With cmsh</td>
<td>477</td>
</tr>
<tr>
<td>11.7.4 The conman Serial Console Logger And Viewer</td>
<td>477</td>
</tr>
<tr>
<td>12 MIC Configuration</td>
<td>483</td>
</tr>
<tr>
<td>12.1 Introduction</td>
<td>483</td>
</tr>
<tr>
<td>12.2 MIC Software Installation</td>
<td>483</td>
</tr>
<tr>
<td>12.2.1 MIC Software Packages</td>
<td>484</td>
</tr>
<tr>
<td>12.2.2 MIC Environment MIC Commands</td>
<td>486</td>
</tr>
<tr>
<td>12.2.3 Bright Computing MIC Tools</td>
<td>486</td>
</tr>
<tr>
<td>12.2.4 MIC OFED Installation</td>
<td>486</td>
</tr>
<tr>
<td>12.3 MIC Configuration</td>
<td>488</td>
</tr>
<tr>
<td>12.3.1 Using cm-mic-setup To Configure MICs</td>
<td>488</td>
</tr>
<tr>
<td>12.3.2 Using cmsh To Configure Some MIC Properties</td>
<td>490</td>
</tr>
<tr>
<td>12.3.3 Using cmgui To Configure Some MIC Properties</td>
<td>491</td>
</tr>
<tr>
<td>12.3.4 Using MIC Overlays To Place Software On The MIC</td>
<td>495</td>
</tr>
<tr>
<td>12.4 MIC Card Flash Updates</td>
<td>497</td>
</tr>
<tr>
<td>12.5 Other MIC Administrative Tasks</td>
<td>499</td>
</tr>
<tr>
<td>12.5.1 How CMDaemon Manages MIC Cards</td>
<td>500</td>
</tr>
<tr>
<td>12.5.2 Using Workload Managers With MIC</td>
<td>500</td>
</tr>
<tr>
<td>12.5.3 Mounting The Root Filesystem For A MIC Over NFS</td>
<td>501</td>
</tr>
<tr>
<td>12.5.4 MIC Metrics</td>
<td>502</td>
</tr>
<tr>
<td>12.5.5 User Management On The MIC</td>
<td>502</td>
</tr>
<tr>
<td>13 High Availability</td>
<td>505</td>
</tr>
<tr>
<td>13.0 Introduction</td>
<td>505</td>
</tr>
<tr>
<td>13.0.1 Why Have High Availability?</td>
<td>505</td>
</tr>
<tr>
<td>13.0.2 High Availability Is Possible On Head Nodes, And Also On Regular Nodes</td>
<td>505</td>
</tr>
<tr>
<td>13.0.3 High Availability Usually Uses Shared Storage</td>
<td>505</td>
</tr>
</tbody>
</table>
## Table of Contents

### 13.0.4 Organization Of This Chapter .......................................................... 505
### 13.1 HA Concepts ................................................................. 506
  - 13.1.1 Primary, Secondary, Active, Passive .................................................. 506
  - 13.1.2 Monitoring The Active Head Node, Initiating Failover .......................... 506
  - 13.1.3 Services In Bright Cluster Manager HA Setups .................................... 506
  - 13.1.4 Failover Network Topology ............................................................... 507
  - 13.1.5 Shared Storage ..................................................................................... 509
  - 13.1.6 Guaranteeing One Active Head At All Times ........................................ 510
  - 13.1.7 Automatic Vs Manual Failover ............................................................ 511
  - 13.1.8 HA And Cloud Nodes ......................................................................... 512
### 13.2 HA Setup Procedure Using cmha-setup ............................................. 512
  - 13.2.1 Preparation ......................................................................................... 513
  - 13.2.2 Failover Cloning ................................................................................ 514
  - 13.2.3 Shared Storage Setup ......................................................................... 518
  - 13.2.4 Automated Failover And Relevant Testing ........................................... 519
### 13.3 Running cmha-setup Without Ncurses, Using An XML Specification ....... 520
  - 13.3.1 Why Run It Without Ncurses? .............................................................. 520
  - 13.3.2 The Syntax Of cmha-setup Without Ncurses ......................................... 521
  - 13.3.3 Example cmha-setup Run Without Ncurses ........................................ 521
### 13.4 Managing HA ................................................................. 522
  - 13.4.1 Changing An Existing Failover Configuration ..................................... 522
  - 13.4.2 cmha Utility ....................................................................................... 522
  - 13.4.3 States ................................................................................................. 526
  - 13.4.4 Failover Action Decisions .................................................................... 527
  - 13.4.5 Keeping Head Nodes In Sync ............................................................... 527
  - 13.4.6 High Availability Parameters .............................................................. 529
  - 13.4.7 Handling And Viewing Failover Via cmgui ......................................... 530
  - 13.4.8 Re-cloning A Head Node .................................................................... 531
### 13.5 HA For Regular Nodes ........................................................................ 532
  - 13.5.1 Why Have HA On Regular Nodes? ....................................................... 532
  - 13.5.2 Comparing Head And Regular Node HA .............................................. 533
  - 13.5.3 Setting Up A Regular Node HA Service .............................................. 533
  - 13.5.4 The Sequence Of Events When Making Another HA Regular Node Active .......................... 537
### 13.6 HA And Workload Manager Jobs ..................................................... 538

### 14 Puppet ................................................................. 541
  - 14.1 Puppet Basics ....................................................................................... 541
  - 14.1.1 Puppet Resource ............................................................................... 541
  - 14.1.2 Puppet Class ...................................................................................... 541
  - 14.1.3 Puppet Declaration ........................................................................... 541
  - 14.1.4 Puppet Manifests ............................................................................... 541
  - 14.1.5 Puppet Modules ................................................................................. 541
  - 14.2 Puppet Configuration With cmsh .......................................................... 542
    - 14.2.1 Configuration Of Puppet ................................................................. 542
    - 14.2.2 Installing Puppet Modules From The Shell ...................................... 542
    - 14.2.3 Installing Puppet Modules From cmsh ............................................. 543
    - 14.2.4 Assignment Of Puppet Roles ........................................................... 545
# Table of Contents

14.2.5 Applying Puppet With `apply` ......................................................... 548
14.3 Puppet Configuration With `cmgui` .................................................. 550
  14.3.1 Overview Tab ................................................................. 550
  14.3.2 Settings Tab ................................................................. 551
  14.3.3 Classes Tab ................................................................. 553
  14.3.4 Modules (PuppetForge) Tab: Installing Puppet Modules From `cmgui` .... 553
  14.3.5 Configuration Tab: Puppet Hello World Example ........................ 554

15 Dell BIOS Management ................................................................. 561
  15.1 Introduction ........................................................................... 561
  15.2 Prerequisites For BIOS Management ...................................... 561
  15.3 BIOS settings ...................................................................... 562
    15.3.1 Initializing The BIOS Settings Via `cmsh` ......................... 562
    15.3.2 Managing The BIOS Settings Via `cmgui` ......................... 562
    15.3.3 Applying Dell Settings And Firmware Updates Via `cmgui` .... 567
    15.3.4 Managing The BIOS Settings Via `cmsh` ......................... 569
  15.4 Frequently Asked Questions ................................................... 572

A Generated Files ............................................................................. 575
  A.1 Files Generated Automatically On Head Nodes ...................... 575
  A.2 Files Generated Automatically In Software Images: ............. 578
  A.3 Files Generated Automatically On Regular Nodes ................ 579
  A.4 Files Not Generated, But Installed. ......................................... 579

B Bright Computing Public Key ......................................................... 583

C CMDaemon Configuration File Directives ........................................ 585

D Disk Partitioning And RAID Configuration .................................... 613
  D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema File ........................................................................ 613
  D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File ................................................................. 619
  D.3 Example: Default Node Partitioning ...................................... 622
  D.4 Example: Hardware RAID Configuration .............................. 623
    D.4.1 RAID level 0 And RAID 10 Example ................................. 624
  D.5 Example: Software RAID ....................................................... 625
  D.6 Example: Software RAID With Swap ..................................... 626
  D.7 Example: Logical Volume Manager ....................................... 627
  D.8 Example: Logical Volume Manager With RAID 1 ................. 628
  D.9 Example: Diskless ................................................................. 630
  D.10 Example: Semi-diskless ....................................................... 630
  D.11 Example: Preventing Accidental Data Loss ....................... 631
  D.12 Example: Using Custom Assertions ..................................... 632

E Example `initialize` And `finalize` Scripts ...................................... 635
  E.1 When Are They Used? ............................................................ 635
  E.2 Accessing From `cmgui` And `cmsh` ...................................... 635
  E.3 Environment Variables Available To `initialize` And `finalize` Scripts ..................................................................................... 636
  E.4 Using Environment Variables Stored In Multiple Variables ..... 639
Welcome to the *Administrator Manual* for the Bright Cluster Manager 7.3 cluster environment.

### 0.1 Quickstart

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

The *Administrator Manual* covers administration topics which are specific to the Bright 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 Bright Cluster Manager 7.3 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](http://support.brightcomputing.com/manuals).

- 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 Bright Cluster Manager.
- The *OpenStack Deployment Manual* describes how to deploy OpenStack with Bright Cluster Manager.
- The *Big Data Deployment Manual* describes how to deploy Big Data with Bright Cluster Manager.
- The *UCS Deployment Manual* describes how to deploy the Cisco UCS server with Bright Cluster Manager.
- The *Machine Learning Manual* describes how to install and configure machine learning capabilities with Bright 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 Bright 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.

### 0.4 Getting Administrator-Level Support

If the reseller from whom Bright 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 11.2 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.
Introduction

1.1 Bright Cluster Manager Functions And Aims

Bright Cluster Manager contains tools and applications to facilitate the installation, administration, and monitoring of a cluster. In addition, Bright 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 Bright 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 Bright 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 Bright 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 Bright Cluster Manager for the first time. The Administrator Manual can be referred to as the main reference resource once the head node has had Bright Cluster Manager installed on it.

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

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

• Post-installation software installation: The installation of software to a cluster that is already configured and running Bright Cluster Manager is described in detail in Chapter 9 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 Bright Cluster Manager. This is described in detail in the Installation Manual.

• Cloudbursting, Big Data, OpenStack, UCS, and Machine Learning: these are integrated by Bright Computing in various ways. These have their own deployment procedures and have separate manuals.
1.2.2 Configuration, Management, And Monitoring Via Bright Cluster Manager Tools And Applications

The administrator normally deals with the cluster software configuration via a front end to the Bright Cluster Manager. This can be GUI-based (cmgui, section 2.4), or shell-based (cmsh, section 2.5). Other tasks can be handled via special tools provided with Bright Cluster Manager, or the usual Linux tools. The use of Bright 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 Bright Cluster Manager</td>
<td>Introduction to main concepts and tools of Bright 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 Bright 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 Containerization</td>
<td>Using Docker and Kubernetes with Bright Cluster Manager</td>
</tr>
<tr>
<td>9 Post-Installation Software Management</td>
<td>Managing, updating, modifying Bright Cluster Manager software and images</td>
</tr>
<tr>
<td>10 Cluster Monitoring</td>
<td>Cluster health, metrics, and actions</td>
</tr>
<tr>
<td>11 Day-To-Day Administration</td>
<td>Miscellaneous administration</td>
</tr>
<tr>
<td>12 MIC Configuration</td>
<td>Intel MIC architecture integration with Bright Cluster Manager</td>
</tr>
<tr>
<td>13 High Availability</td>
<td>Background details and setup instructions to build a cluster with redundant head nodes</td>
</tr>
<tr>
<td>14 Puppet</td>
<td>Puppet configuration and integration with Bright Cluster Manager</td>
</tr>
<tr>
<td>15 Dell BIOS Management</td>
<td>BIOS management with Bright Cluster Manager for certain Dell hardware</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 Bright 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)
- OpenStack deployment (OpenStack Deployment Manual)
- Big Data deployment (Big Data Deployment Manual)
- Cisco UCS deployment (Cisco UCS Deployment 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](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 Bright 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 11.2 describes how to get further support.
This chapter introduces cluster management with Bright Cluster Manager. A cluster running Bright 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 Bright 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 (cmgui) 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 Bright Cluster Manager are introduced.

2.1.1 Devices

A device in the Bright Cluster Manager cluster management infrastructure represents components of a cluster. A device can be any of the following types:

- Head Node
- Physical Node
- Virtual Node
- Cloud Node
- Virtual SMP Node
- GPU Unit
- MIC
- Chassis
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 Bright 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 Bright Cluster Manager has a device state associated with it. The table below describes the most important states for devices:

<table>
<thead>
<tr>
<th>state</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>UP/CLOSED</td>
<td>UP</td>
<td>mostly ignored</td>
<td>tracked</td>
</tr>
<tr>
<td>DOWN/CLOSED</td>
<td>DOWN</td>
<td>mostly ignored</td>
<td>tracked</td>
</tr>
</tbody>
</table>

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

DOWN and DOWN/CLOSED 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 DOWN/CLOSED, the device is down, but is intended to be unavailable, and thus typically indicates that the administrator 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 Bright 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.

RHEL6/CentOS6/SL6 mixtures are completely compatible with each other on the head and regular nodes. On the other hand, SLES may need some effort to work in a mixture with RHEL/CentOS/SL.

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

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

It is also possible to “lock” a software image so that no node is able to pick up the image until the software image is unlocked. (section 5.4.7).
Software images can be changed using regular Linux tools and commands (such as `rpm` and `chroot`). More details on making changes to software images and doing image package management can be found in Chapter 9.

### 2.1.3 Node Categories

The collection of settings in Bright Cluster Manager that can apply to a node is called the configuration of the node. The administrator usually configures nodes using the `cmgui` (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 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 64), 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 is overruled by specifying the node property value for the 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 one, variable, image may be used in the same node category.

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

<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 (if any) 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
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 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 of the category.

A configuration overlay is a feature introduced in Bright Cluster Manager 7.1. A configuration overlay is a group of roles that can be assigned to designated groups of nodes within a cluster. In the case of Big Data, the overlays are called Hadoop, Cassandra, or Spark configuration groups (section 3.1.9 of the Big Data Deployment Manual), and are applied to nodes within a Hadoop, Cassandra, or Spark instance.

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.

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

2.2 Modules Environment

The modules environment is a third-party software (section 7.1 of the Installation Manual) that 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 <MODULENAME1> [<MODULENAME2> ...]
```

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 to be compiled with OpenBLAS optimizations:
Example

module add shared
module add gcc/6.1.0
module add openmpi/gcc/64/1.10.1
module add openblas/dynamic/0.2.18
mpicc -o myapp myapp.c

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:

Example

[fred@bright73 ~]$ module list
Currently Loaded Modulefiles:
1) gcc/6.1.0 2) slurm/16.05.2

The Torque and Maui modules can then be set up by the administrator as a default for all users in the following 2 ways:
1. Defining part of a `.profile` to be executed for login shells. For example:

   ```
   [root@bright73 ~]# cat /etc/profile.d/userdefaultmodules.sh
   module load shared
   module load torque
   module load maui
   ```

   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@bright73 ~]# 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 torque maui
   ```

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

   ```
   [root@bright73 ~]# cat /etc/profile.d/userdefaultmodules.sh
   module load default-environment
   ```

   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` trees. 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 Bright Cluster Manager.

2.2.5 Lua Modules Environment (LMod)

By default, Bright Cluster Manager uses traditional TCL scripts for its module files, or TMod. Using Lua modules, or LMod, is an alternative modules environment, where the files are typically written in Lua. LMod can be used as a replacement for TMod.

Conceptually LMod works in the same way as TMod, but provides some extra features and commands.
For LMod, the module files are typically written in Lua, but LMod is also capable of reading Tcl module files. It is therefore not necessary to convert all existing Tcl modules manually to the Lua language.

On a 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@bright73 ~]# cat /etc/sysconfig/modules/lmod/cm-lmod-init.sh
export ENABLE_LMOD=1

In the preceding example, LMod is enabled, and TMod is disabled if $ENABLE_LMOD is set to 1.

Example

[root@bright73 ~]# cat /etc/sysconfig/modules/lmod/cm-lmod-init.sh
export ENABLE_LMOD=0

In the preceding example, LMod is disabled, and TMod is enabled if $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@bright73 ~]# cm-change-passwd
With this utility you can easily change the following passwords:

\* root password of head node
\* root password of slave images
\* root password of node-installer
\* root password of mysql

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

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

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

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

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

For a high-availability—also called a failover—configuration, the passwords are copied over automatically to the other head node when a change is made in the software image root password (case 2 on page 11).

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 cmgui front-end, then the password should be modified there too (figure 2.3).

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 are set by default to be passwordless:

- For non-root users, an ssh passwordless login works if the /home directory that contains the authorized keys for these users is mounted. The /home directory is mounted by default.

- For the root user, an ssh passwordless login should always work since the authorized keys are stored in /root.

Users can be restricted from ssh logins
• on regular nodes using the `usernode/login` (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 a Bright Cluster Manager 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.

Bright 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 Bright 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 Bright 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 233.

Using `cmgui` and authenticating to the Bright Cluster Manager: When an administrator uses the `cmgui` front end, the same certificate pair as used by `cmsh` is used. Running `cmgui` from a desktop, that is, a location other than the head node, is described in section 2.4.1, and being prompted for authentication when doing so is an expected security requirement. In theory, authentication is not a security requirement if the user is already logged into the head node and running `cmgui` from there as that user. However, for consistency reasons, the `cmgui` front end always prompts for user authentication to take place, unless the `Password` field and `Connect at start-up` checkbox in the dialog of figure 2.3 are both already filled in.

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.

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 Cluster Management GUI

This section introduces the basics of the cluster management GUI (cmgui). This is the graphical interface to cluster management in Bright Cluster Manager. It can be run from the head node or on a login node of the cluster using X11-forwarding:

Example

user@desktop:~> ssh -X root@mycluster cmgui

However, typically it is installed and run on the administrator’s desktop computer. This saves user-discriminal lag time if the user is hundreds of kilometers away from the head node.

2.4.1 Installing Cluster Management GUI On The Desktop

Installation packages are available for Linux, for Windows XP/Vista/Windows 7/8/10, and for Mac OS.

To install cmgui on a desktop computer a Firefox browser, version 10 or greater, must first be installed onto the operating system. The cmgui installation package corresponding to the CMDaemon version must then be installed to the desktop. The package can be downloaded from from the internet or from the head node.

Downloading From The Internet

The latest cmgui packages are publicly available at http://support.brightcomputing.com/cmgui-download. Mac OS, MS Windows, or Linux packages are available. The administrator should use these with up-to-date Bright Cluster Manager releases.

Downloading From The Head Node

For a default repository configuration, doing a yum update, or zypper up for SLES-based distributions, on the head node ensures the head node has the latest updates.

If no internet access is available, then installation packages for the desktop, that may be out of date, can still be copied over from the head node, from under the directory:

/cm/shared/apps/cmgui/dist/

Installing cmgui On Mac OS, MS Windows, And Linux

The installation package can be placed in a convenient and appropriately accessible location on the desktop.

On the Mac OS desktop, cmgui is installed from the Mac OS install.cmgui.macosx.7.3.r8788.pkg file by clicking on it and following the installation procedure. The cmgui front end is then run by clicking on the cmgui icon.

On the MS Windows desktop, cmgui is installed from the install.cmgui.7.3.r8788.exe installer file by running it and following the installation procedure.

If there is an antivirus application preventing its installation, then the file and folders into which cmgui is installed should be excluded from the antivirus checks as suggested by the antivirus software vendor.

After installation, cmgui is started through the Start menu or through the desktop shortcut.

For Linux, the installation of a 64-bit Firefox browser on the desktop, and not a 32-bit version, is mandatory, before cmgui is installed. The version of Firefox on the desktop can be checked and replaced if necessary. The binary can be checked in a manner similar to the following (some output elided):
Example

root@work:~# file firefox /usr/lib32/firefox/firefox-bin: ELF 32-bit LSB executable, Intel 80386, ...
root@work:~# apt-get install firefox:amd64 ...
root@work:~# exit

If the output to the file command is something like POSIX shell script, ASCII text executable, then the file being checked is a wrapper script to the real firefox. The binary in that case is probably called firefox-bin, which means the command locate firefox-bin is likely to show its location, and the file command can then be run on that binary to check its bitsize.

For the Linux desktop, cmgui can then be installed and run by:

- copying over the .tar.bz2 file
- untarring the .tar.bz2 file
- running cmgui from inside the directory that the untar process created.

The install and run procedure may look similar to (some output elided):

Example

me@work:~$ scp root@bright73:/cm/shared/apps/cmgui/dist/cmgui.7.3.r8788.tar.bz2 . ...
me@work:~$ tar -xjf cmgui.7.3.r8788.tar.bz2 ...
me@work:~/cmgui-7.3-r8788$ ./cmgui

Bright Cluster Manager cmgui Welcome Screen

![Welcome Screen](image)

Figure 2.1: Cluster Manager GUI welcome screen

When cmgui is started for the first time, the welcome screen (figure 2.1) is displayed.

To configure cmgui for connections to a new Bright Cluster Manager cluster, the cluster is added to cmgui by clicking the □ button in the welcome screen. More clusters can be added within cmgui as needed.
After a cluster is added, the screen displays the connection parameters for the cluster (figure 2.2).

![Figure 2.2: Connecting to a cluster](image)

**Bright Cluster Manager cmgui Connection Parameters Dialog Window**

Figure 2.3 shows the dialog window in which the connection parameters can be entered.

![Figure 2.3: Editing The Cluster Connection Parameters](image)

The dialog window in figure 2.3 has several options:

- The Host field can be a name or an IP address. If the port on the host is not specified, then port 8081 is added automatically.

- The “Connect at start-up” checkbox option offers convenience by attempting to connect to the cluster right away when starting up cmgui, without waiting for the administrator to click on the connect button of figure 2.2.

- The “Username” field accepts the name of the user that cmgui is to run as. So, cmgui can be run by a non-root user as user root as well as run the other way round.

© Bright Computing, Inc.
• The Password field affects connection behavior when starting up cmgui, or when clicking on the connect button of figure 2.2. The behavior is generally obvious, and depends on whether a password has been saved in the Password field:

1. With a correct password set, and all other fields in the dialog set correctly, a connection to the cluster is established when starting up cmgui if the “Connect at start-up” state has been set. The cluster overview screen (figure 2.4) for the cluster is then displayed.

2. With a blank password, but all other fields in the dialog set correctly, and if continuing in a current cmgui session where a previous successful connection to the cluster has been disconnected, then clicking on the connect button of figure 2.2 establishes a successful connection to the cluster, and the cluster overview screen (figure 2.4) for the cluster is then displayed.

3. With a blank password, but all other fields in the dialog set correctly, and if the cluster is being connected to for the first time during a new cmgui session, then a password prompt dialog is displayed if:

   - starting cmgui and if the “Connect at start-up” state has been set
   - or
   - if cmgui is already running and the connect button in the screen shown in figure 2.2 is clicked.

   The cluster overview screen (figure 2.4) for the cluster in this case (case number 3) is only displayed after a successful authentication.

The administrator should be aware that storing the password is unsafe if untrusted people can access the cmgui files in which the password is stored. These are kept on the machine that runs the cmgui process, which may not be the machine that displays cmgui on its screen.

Avoiding Lag With cmgui

The cmgui front-end is often run on the head node of the cluster via an ssh -X login, with the -X option passing the X11 protocol instructions. With such a connection, more changes being tracked on the front end means more X11 protocol instructions, and therefore a more sluggish interface. Additionally, the transport lag for X11 instructions is greater if the X-display server, where the administrator is viewing cmgui, is some distance away from the head where cmgui is actually running. This can lead to an irritating, or even an unusable cmgui in extreme cases.

Then there is the common case when there is a bastion host or a similar “jump” host in the way, which prevents running cmgui from the head node directly via ssh -X. A way to deal with the bastion/jump host problem is to carry out an ssh -X login to the bastion host first, and then to carry out an ssh -X login to the head node to run the cmgui session. This double-ssh -X connection is easy enough to implement. However it has several problems.

• The X11-protocol overhead and lag becomes larger because there are now two jumps between the cluster and the remote display server. Viewing cmgui in this way is therefore likely to be even more irritating than before.

• Microsoft windows cmgui clients will need to be displayed over an X-display server too, which means installing an X-display server for MS windows. This is often undesirable.

Because of these problems, alternatives to double-ssh -X are therefore often used:

• One workaround is to run cmgui over an ssh -X connection from the bastion host, and then use VNC (a remote desktop viewing software) from the remote display-server to the bastion host. This is usually more pleasant for interactive use than double-ssh -X because

   - the bastion host is usually close to the cluster, thus minimizing the X11 lag from the cluster head node to the bastion host.
using VNC in the connection to transport the desktop image from the bastion host to the remote display avoids the X11 protocol transport lag issue entirely for that part of the connection.

For implementing VNC, an administrator should note that:

* most modern standard versions of VNC can connect with compression
* encryption and authentication for VNC must be enabled in order to avoid unauthorized VNC access. If enabling these is a problem, then the well-trusted OpenVPN utility can be used to provide these in a separate layer instead with good results.

- Usually the best workaround to set up cmgui to display on the desktop is to run cmgui as the desktop client on the remote display, and set up ssh port-forwarding on the bastion host. This again avoids the transport overhead and lag of X11 over double-ssh -X, and indeed avoids transporting the X11 protocol entirely. Instead, the ssh connection transports the CMDaemon REST calls directly. These REST calls run encrypted over the SSL port 8081 already, so that all that is needed is to forward them, which is what the ssh port-forwarding feature is able to do quite easily. The additional encryption from ssh on top of the SSL encryption already operating on the REST calls does no harm. It is unlikely to be a performance bottleneck, and in any case can be switched off or speeded up in some implementations or compiler options of ssh.

The ssh port-forwarding connection can be configured by following these steps:

- For failover clusters, to connect to the active node, cmgui needs to be aware that it should not follow redirection. To arrange this, the cmgui settings file ~/.cm/cmgui/clusters.dat should be modified on the machine where the cmgui desktop client is running. The safest way to carry out the modification is to first make sure that the file is created by starting up the desktop cmgui, then stopping it. The followRedirect line in the clusters.dat file should then be changed to:

```plaintext
followRedirect = false;
```

Single-head clusters require no such change.

- Port-forwarding can then be set up from the cluster. If these parameters are used:
  * `cluster-ip`: the cluster address. This is the shared alias IP address for a failover cluster
  * `bast`: the bastion host.
  * `user`: the user name that the administrator is using
  * `admin-desktop`: the remote machine

Then:

* If the bastion host allows no inbound ssh connections from the administrator’s remote desktop, then:
  · a remote-port-forwarding tunnel can be set up on it by running:
    ```plaintext
    bast:$ ssh -R 8081:cluster-ip:8081 user@admin-desktop
    ```
    A login to `admin-desktop` is prompted for, as part of the execution of the preceding command, and the login should be carried out.
  · on the remote desktop, after cmgui is started up, a connection to `localhost:8081` should be made. The value `localhost:8081` can be set in the cmgui cluster connection parameters dialog shown in figure 2.3.

**Bright Cluster Manager cmgui Default Display On Connection**

Clicking on the Connect button establishes a connection to the cluster, and cmgui by default then displays a tabbed pane overview screen of the cluster. This default screen presents summarized data for the entire cluster (figure 2.4):
2.4 Cluster Management GUI

2.4.2 Navigating The Cluster Management GUI

Aspects of the cluster can be managed by administrators using cmgui (figure 2.4).

The resource tree, displayed on the left side of the window, consists of hardware resources such as nodes and switches as well as non-hardware resources such as Users & Groups and Workload Management. Selecting a resource opens an associated tabbed pane on the right side of the window that allows tab-related parameters to be viewed and managed.

The number of tabs displayed and their contents depend on the resource selected. The following standard tabs are available for most resources:

- **Overview**: provides an overview containing the most important status details for the resource.
- **Tasks**: accesses tasks that operate on the resource.
- **Settings**: allows configuration of properties of the resource.

The Overview tab of the cluster resource (figure 2.4) displays properties intended to convey at a glance the overall state of the cluster to the administrator.

The CPU Usage display is typically the item of greatest interest to a cluster administrator. Its indicator bar is divided into 4 sections, and marked with u, s, o, i according to the scheme described in the following table:
20 Cluster Management With Bright Cluster Manager

<table>
<thead>
<tr>
<th>Color</th>
<th>Bright state</th>
<th>top state</th>
<th>CPU time spent by applications in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>user</td>
<td>us</td>
<td>user space</td>
</tr>
<tr>
<td>blue</td>
<td>system</td>
<td>sy</td>
<td>kernel space</td>
</tr>
<tr>
<td>orange</td>
<td>other</td>
<td>sum of:</td>
<td>sum of these other active states:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wa</td>
<td>i/o wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ni</td>
<td>nice user processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hi</td>
<td>hardware IRQ servicing/handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>si</td>
<td>software IRQ servicing/handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>st</td>
<td>“stolen time”—forced wait for a virtual CPU while hypervisor services another processor</td>
</tr>
<tr>
<td>none</td>
<td>idle</td>
<td>id</td>
<td>idling</td>
</tr>
</tbody>
</table>

The Bright Cluster Manager CPU States are based on metrics (section G.1) gathered from all nodes and made into a summary. The states are based on the CPU states defined for the standard top (1) command, and which are described briefly in the table.

![Node Settings](image)

Figure 2.5: Node Settings

The Settings tab of the node resource (figure 2.5) displays properties, such as the hostname of the resource, that can be changed. The Save button on the bottom of the tab makes the changes active and permanent, while the Revert button undoes all unsaved changes.

© Bright Computing, Inc.
Figure 2.6: Node Tasks

Figure 2.6 shows the Tasks tab of the node001 resource. The tab displays operations that can be performed on the node001 resource. Details on setting these up, their use, and meaning are provided in the remaining chapters of this manual.

It is also possible to select a resource folder (rather than a resource item) in the tree. For example: Node Categories, Nodes, and Networks. Selecting a resource folder in the tree by default opens an Overview tab, which displays a list of resource items inside the folder. These are displayed in the resource tree and in the tabbed pane. Resource items in the tabbed pane can be selected, and operations carried out on them by clicking on the buttons at the bottom of the tabbed pane. For example, for Nodes, one or more nodes can be selected, and the Open, Add, Clone and Remove buttons can be clicked to operate on the selection (figure 2.7).

Figure 2.7: Nodes Overview
2.4.3 Advanced cmgui Features

This section describes some advanced features of cmgui. It may be skipped on first reading.

Within cmgui the right mouse button often brings up convenient options. Two of these are labor-saving options that synchronize properties within resource items.

Synchronize Values To Other Items In Current Resource

“Synchronize values to other <items in current resource>” is a menu option that synchronizes the properties of the item that is currently selected to other items within the same resource.

Some examples:

1. Synchronize values to other fsmounts: Within the current category of Node Categories, for a mount point within the FS Mounts tab, synchronize the mount point values for the selected mount point to other mount points under the same FS Mounts tab.

2. Synchronize values to other network interfaces: Synchronize network interface values of the selected network interface to other network interfaces, within the Network Setup tab.
   
   (a) For a head node, this is done within the selected head node in the Head Nodes resource.
   
   (b) For a regular node, this is done within the selected node in the Nodes resource.

3. Synchronize values to other physicalnodes: For the selected node in the Nodes resource, synchronize the values of the node to other nodes.

   Item 2b can be regarded as a subset of item 3.

   Item 3 in the list is a good example to elaborate upon, to illustrate what is meant:

   Within the Nodes resource in the resources menu, right-clicking on a node item brings up a menu (figure 2.8),
Figure 2.8: Right Click Menu Options

Selecting **Synchronize Values to other physical nodes** brings up a Clone Wizard. The wizard helps the administrator choose the node property values to be synchronized (figure 2.9), and also to decide to which of the other nodes (figure 2.10) in the Nodes resource these property values are to be synchronized to.
A final overview (figure 2.11) explains what is going to be synchronized when the Finish button of the wizard is clicked.
Before clicking Finish button, the operation can also optionally be saved to a file.

- To save the file: The file location should be specified as a full path. Typically the directory /cm/shared/apps/cmgui/ is used if cmgui is run from the head node.
- To load the file: A previously-saved operation can be re-executed by selecting it from the Tools menu.

Export To Other Items Outside Current Resource

“Export to other <items outside current resource>” is a menu option that exports the properties of the current item to other items outside the parent resource providing the item. So it synchronizes, but to items outside the direct parent resource. For example:

1. Export to other Categories: Within the current category of Node Categories, for a mount point within the FS Mounts tab, export the mount point values for the selected mount point to other mount points outside the original FS Mounts tab.

2. Export network interface values of the selected network interface to other network interfaces, outside the current Network Setup tab.

   (a) Export to other masternodes is the option used for a head node. It exports to the other head node items in the Head Nodes resource.

   (b) Export to other physicalnodes is the option used for a regular node. It exports to the other node items in the Nodes resource.

The numbering of the items listed here corresponds to the items in the Synchronize... list from earlier (page 22), except that item 3 in the that earlier list has no corresponding export menu option.

Just like with the Synchronize... menu option from earlier (page 22), selecting the Export... menu option brings up a Clone Wizard. The wizard helps the administrator choose the values to be exported, where to export them to, and allows the operation to be saved for later re-execution from the Tools menu.

2.5 Cluster Management Shell

This section introduces the basics of the cluster management shell, cmsh. This is the command-line interface to cluster management in Bright Cluster Manager. Since cmsh and cmgui 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 cmgui may therefore safely skip this section.

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

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

### 2.5.1 Invoking cmsh

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

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

Running cmsh without arguments starts an interactive cluster management session. To go back to the unix shell, a user enters quit:

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

**Batch Mode And Piping In cmsh**

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

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

Alternatively, commands can be piped to cmsh:

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

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

In a similar way to unix shells, cmsh sources dotfiles, if they exist, upon start-up in both batch and interactive mode. In the following list of dotfiles, a setting in the file that is in the shorter path will override a setting in the file with the longer path (i.e.: “shortest path overrides”):

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

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

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
```

© Bright Computing, Inc.
2.5 Cluster Management Shell

Built-in Aliases In cmsh

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

```bash
[bright73]# alias alias - goto -
alias .. exit
alias / home
alias ? help
alias ds device status
```

Options Usage For cmsh

The options usage information for cmsh is obtainable with cmsh -h (figure 2.12).

```
Usage:
  cmsh [options] ................ Connect to localhost using default port
  cmsh [options] <--certificate|-i certfile> <--key|-k keyfile> <host[:port]>
      Connect to a cluster using certificate and key in PEM format
  cmsh [options] <--certificate|-i certfile> [--password|-p password] <uri[:port]>
      Connect to a cluster using certificate in PEX format

Valid options:
--help|-h ..................... Display this help
--noconnect|-u ................ Start unconnected
--controlflag|-z .............. ETX in non-interactive mode
--nocolor|-o .................. Do not use misc. colors in the text output
--noredirect|-r ............... Do not follow redirects
--norc|-n ..................... Do not load cmshrc file on start-up
--noquitconfirmation|-Q ....... Do not ask for quit confirmation
--command|-c <"c1; c2; ..."> .. Execute commands and exit
--file|-f <filename> .......... Execute commands in file and exit
--echo|-x ..................... Echo all commands
--quit|-q ..................... Exit immediately after error
--disabledmultiline|-m ....... Disable multiline support
```

Figure 2.12: Usage information for cmsh

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 modes. Modes and their levels are a hierarchy available below the top-level, and therefore to perform cluster management functions, a user switches and descends into the appropriate mode.

Figure 2.13 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>Alias</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Set aliases</td>
</tr>
<tr>
<td>bigdata</td>
<td>Enter bigdata mode</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>cmsub</td>
<td>Enter cmsub 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>delimiter</td>
<td>Display/set delimiter</td>
</tr>
<tr>
<td>device</td>
<td>Enter device 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>filesyncconfig</td>
<td>Enter filesyncconfig 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>hadoop</td>
<td>Enter hadoop mode</td>
</tr>
<tr>
<td>help</td>
<td>Display this help</td>
</tr>
<tr>
<td>history</td>
<td>Display command history</td>
</tr>
<tr>
<td>jobqueue</td>
<td>Enter jobqueue mode</td>
</tr>
<tr>
<td>jobs</td>
<td>Enter jobs mode</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>openstack</td>
<td>Enter openstack 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>puppet</td>
<td>Enter puppet 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>user</td>
<td>Enter user mode</td>
</tr>
<tr>
<td>watch</td>
<td>Execute a command periodically, showing output</td>
</tr>
<tr>
<td>zookeeper</td>
<td>Enter zookeeper mode</td>
</tr>
</tbody>
</table>

Figure 2.13: 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

```bash
[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, --quit
    Exit immediately after error
```

In the general case, invoking `help` at any level, 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 the `session` level and entering `help` gives firstly, output with a title of Top, and secondly, output with a title of `session` (some output ellipsized):

```bash
[myheadnode] % session
[myheadnode->session] % help
```

#### Navigation Through Modes And Levels In cmsh

- 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 get to an object level within a mode, the `use` command is used with the object name. The prompt then changes to indicate that an object within that 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. 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 `path` command at any mode depth displays a path to the current mode depth, in a form that is convenient for copying and pasting.
- The `home` command, which is aliased to `/`, takes the user from any mode depth to the top level.

© Bright Computing, Inc.
[bright73]% device
[bright73->device]% list

<table>
<thead>
<tr>
<th>Type</th>
<th>Hostname (key)</th>
<th>MAC</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>EthernetSwitch</td>
<td>switch01</td>
<td>00:00:00:00:00:00</td>
<td></td>
</tr>
<tr>
<td>HeadNode</td>
<td>bright73</td>
<td>00:0C:29:5D:55:46</td>
<td></td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>node001</td>
<td>00:0C:29:7A:41:78</td>
<td>default</td>
</tr>
<tr>
<td>PhysicalNode</td>
<td>node002</td>
<td>00:0C:29:CC:4F:79</td>
<td>default</td>
</tr>
</tbody>
</table>

[bright73]->device]% exit
[bright73]% device
[bright73->device]% use node001
[bright73->device[node001]]% path
home;device;use node001;
[bright73->device[node001]]% home
[bright73]% home;device;use node001 #copy-pasted from path output
[bright73->device[node001]]%

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

A cmsh input line has the following syntax:

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

where <mode> and <arg> are optional. ¹

Example

```
[bright73->network]% device status bright73; list
bright73 ............ [ UP ]
Name (key) Type Netmask bits Base address Domain name
----------------- --------- -------------- -------------- -------------------
externalnet External 16 192.168.1.0 brightcomputing.com
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 mode is entered, so that the list displays a list of nodes (some output truncated here for convenience):

Example

```
[bright73->network]% device; status bright73; list
bright73 ............ [ UP ]
Type Hostname (key) MAC Category
---------------------- ------------------ ----------------
HeadNode bright73 FA:16:3E:C8:06:D1
PhysicalNode node001 FA:16:3E:A2:9C:87 default
```

¹ A more precise synopsis is:

```
<mode> <cmd> [ <arg> ... ] ; ... ; <mode> <cmd> [ <arg> ... ]
```
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>
<tr>
<td>refresh</td>
<td>Undo local changes done to the object</td>
</tr>
<tr>
<td>list</td>
<td>List all objects at current level</td>
</tr>
<tr>
<td>sort</td>
<td>Sort the order of display for the list command</td>
</tr>
<tr>
<td>format</td>
<td>Set formatting preferences for list output</td>
</tr>
<tr>
<td>foreach</td>
<td>Execute a set of commands on several objects</td>
</tr>
<tr>
<td>show</td>
<td>Display all properties of the object</td>
</tr>
<tr>
<td>swap</td>
<td>Swap (exchange) the names of two objects</td>
</tr>
<tr>
<td>get</td>
<td>Display specified property of the object</td>
</tr>
<tr>
<td>set</td>
<td>Set a specified property of the object</td>
</tr>
<tr>
<td>clear</td>
<td>Set default value for a specified property of the object.</td>
</tr>
<tr>
<td>append</td>
<td>Append a value to a property of the object, for a multi-valued property</td>
</tr>
<tr>
<td>removefrom</td>
<td>Remove a value from a specific property of the object, for a multi-valued property</td>
</tr>
<tr>
<td>modified</td>
<td>List objects with uncommitted local changes</td>
</tr>
<tr>
<td>usedby</td>
<td>List objects that depend on the object</td>
</tr>
<tr>
<td>validate</td>
<td>Do a validation check on the properties of the object</td>
</tr>
<tr>
<td>exit</td>
<td>Exit from the current object or mode level</td>
</tr>
</tbody>
</table>

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

**Working With Objects:** `use, exit`

**Example**

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

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

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

Example

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

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

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

`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 or 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.

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

The \texttt{--wait} option to remove: The commit command by default does not wait for the 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 \texttt{cmsh} for cloning (discussed shortly) a software image (section 2.1.2). The \texttt{--wait} option to the commit command works around this issue by waiting for the cloning of the software image to be completed before making the prompt available.

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

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

Usage:
   add <type> <hostname>
   add <type> <hostname> <ip>

Arguments:
type  
   cloudnode, physicalnode, virtualnode, virtualsmpnode, headnode, mic,
   ethernetswitch, ibswitch, myrinetswitch, powederdistributionunit,
   genericdevice, racksensor, chassis, gpuunit
```

Working With Objects: clone, modified, swap

Continuing on with the node object \texttt{node100} that was created in the previous example, it can be cloned to \texttt{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
```

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

The modified command corresponds roughly to the functionality of the List of Changes menu option under the View menu of \texttt{cmgui}’s main menu bar.

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

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

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
[bright73->device[mycluster]->interfaces]% swap eth0 eth1; commit
[bright73->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
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 the `modified` command confirms that no local changes exist. Finally the `get` command reconfirms that no local changes exist.

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

© Bright Computing, Inc.
2.5 Cluster Management Shell

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.

Some properties are Booleans. For these, 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 all device objects. The -f flag 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, displaying the hostname, ethernetswitch and ip properties for each object.

Example

[bright73->device]% list -f hostname:14,ethernetswitch:15,ip
hostname (key) ethernetswitch ip
----------------- ------------------ ---------------
apc01 10.142.254.1
bright73 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
[bright73->device]%

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

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

Example

[bright73->device]% format
Current list printing format:
-----------------------------
hostname:[10-14]
Valid fields:
-------------
Revision : Entity revision
Type : The type of the device
activation : Date on which node was defined
additionalhostnames : List of additional hostnames that should resolve to the interfaces IP address
banks : Number of banks
bmcpassword : Password used to send ipmi/ilo commands
bmcusername : Username used to send ipmi/ilo commands

To change the current format string, a new format string can be passed as an argument to format. 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

[bright73->device]% format hostname:10
[bright73->device]% list
hostname (key)
--------------
apc01
bright73
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

[bright73->device]% format hostname:[10-14]
[bright73->device]% list
hostname (key)
--------------
apc01
bright73
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. Multiple parameters can be specified as a comma-separated list (with a colon-delimited width specification for each parameter). For example:

Example

[bright73->device]% format hostname:[10-14],ethernetswitch:14,ip:20
[bright73->device]% list
hostname (key) ethernetswitch ip
-------------- ------------------------------
apc01 10.142.254.1
The default parameter settings can be restored with the \texttt{-r|--reset} option:

\textbf{Example}

\begin{verbatim}
[bright73->device]\% format -r
[bright73->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]
[bright73->device]%
\end{verbatim}

The \texttt{sort} command sets the alphabetical sort order for the output of the \texttt{list} command according precedence of the parameters specified.

\textbf{Example}

\begin{verbatim}
[bright73->device]\% sort type mac
[bright73->device]\% list -f type:15,hostname:15,mac
type hostname (key) mac
--------------- --------------- --------------------
HeadNode bright73 08:0A:27:BA:B9:43
PhysicalNode node002 00:00:00:00:00:00
PhysicalNode log001 52:54:00:DE:E3:6B
[bright73->device]\% sort type hostname
[bright73->device]\% list -f type:15,hostname:15,mac
type hostname (key) mac
--------------- --------------- --------------------
HeadNode bright73 08:0A:27:BA:B9:43
PhysicalNode log001 52:54:00:DE:E3:6B
PhysicalNode node002 00:00:00:00:00:00
[bright73->device]\% sort mac hostname
[bright73->device]\% list -f type:15,hostname:15,mac
type hostname (key) mac
--------------- --------------- --------------------
PhysicalNode node002 00:00:00:00:00:00
HeadNode bright73 08:0A:27:BA:B9:43
PhysicalNode log001 52:54:00:DE:E3:6B
\end{verbatim}

The preceding \texttt{sort} commands can alternatively be specified with the \texttt{--sort} option to the \texttt{list} command:

\begin{verbatim}
[bright73->device]\% list -f type:15,hostname:15,mac --sort type,mac
[bright73->device]\% list -f type:15,hostname:15,mac --sort type,hostname
[bright73->device]\% list -f type:15,hostname:15,mac --sort mac,hostname
\end{verbatim}

\textbf{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 \texttt{append} and \texttt{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 \texttt{set} command

© Bright Computing, Inc.
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 apc01:1:0 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 193) 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
```

**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**

```bash
[mycluster->device]% use node001
[mycluster->device[node001]]% clear category
[mycluster->device[node001]]% commit
```

© Bright Computing, Inc.
### 2.5 Cluster Management Shell

<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
[mycluster->device[node001]]%
```

**Working With Objects:** *show*

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

```
[mycluster->device[node001]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation</td>
<td>Thu, 08 Aug 2013 16:57:38 CEST</td>
</tr>
<tr>
<td>BMC Password</td>
<td>&lt; not set &gt;</td>
</tr>
<tr>
<td>BMC User ID</td>
<td>-1</td>
</tr>
<tr>
<td>BMC User name</td>
<td></td>
</tr>
<tr>
<td>Burn config</td>
<td>&lt;0 bytes&gt;</td>
</tr>
<tr>
<td>Burning</td>
<td>no</td>
</tr>
<tr>
<td>Category</td>
<td>default</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Software image</td>
<td>default-image</td>
</tr>
<tr>
<td>Start new burn</td>
<td>no</td>
</tr>
<tr>
<td>Type</td>
<td>PhysicalNode</td>
</tr>
<tr>
<td>Userdefined 1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></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 the main `category` mode or the main `device` mode. Within `roles`, `assign` is used for assigning tasklike `roles` objects.

### 2.5.4 Accessing Cluster Settings

The management infrastructure of Bright 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 Bright Cluster Manager, its design implications do decide how some global cluster properties are accessed through `cmsh`.

© Bright Computing, Inc.
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>BMC Password</td>
<td>***</td>
</tr>
<tr>
<td>BMC User ID</td>
<td>4</td>
</tr>
<tr>
<td>BMC User name</td>
<td>bright</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 MIC category</td>
<td>default</td>
</tr>
<tr>
<td>Default burn configuration</td>
<td>default</td>
</tr>
<tr>
<td>Default category</td>
<td>default</td>
</tr>
<tr>
<td>Default software image</td>
<td>default-image</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>myheadnode</td>
</tr>
<tr>
<td>Lustre settings</td>
<td>&lt;submode&gt;</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>Search domains</td>
<td>example.com</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>
<tr>
<td>UCS Profiles</td>
<td>&lt;0 in submode&gt;</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 \textless\text{ctrl}\textgreater -r, or using the up- and down-arrow keys

**History And Timestamps**

The \texttt{history} command within \texttt{cmsh} explicitly displays the \texttt{cmsh} command history as a list.

The \texttt{--timestamps|-t} option to the \texttt{history} command displays the command history with timestamps.

**Example**

```bash
[bright73->device[node001]]% history | tail -3
162 use node001
163 history
164 history | tail -3
[bright73->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
```

**Mixing \texttt{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, \texttt{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 \texttt{!} command by itself will start an interactive login sub-shell. By exiting the sub-shell, the user will return to the \texttt{cmsh} prompt.

Besides simply executing commands from within \texttt{cmsh}, the output of operating system shell commands can also be used within \texttt{cmsh}. This is done by using the “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, \texttt{cmsh} also supports output redirection to the shell through common operators such as \texttt{>}, \texttt{>>} and \texttt{<}.  

**Example**

```bash
[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>

© Bright Computing, Inc.
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**

```
[bright73->device[node001]]% time check deviceisup
Health Check        Value     Age (sec.) Info Message
------------------------------- -------------- --------------- 
DeviceIsUp                  PASS        0
```

time: 0.276s

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**

```
[bright73->device]% watch newnodes
screen clears  
Every 2.0s: newnodes Thu Dec 3 13:01:45 2015
No new nodes currently available.
```

```
[bright73->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.: “()”. 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  ..........  [ UP ]
node002  ..........  [ UP ]
```

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.

© Bright Computing, Inc.
Advanced options for the *foreach* command: The *foreach* command advanced options can be viewed from the help page:

```
[root@bright73 ~]# cmsh -c "device help foreach"
```

The options can be classed as: grouping options, adding options, conditional options, and looping options.

- **Grouping options:** `-n|--nodes`, `-g|--group`, `-g|--category`, `-r|--rack`, `-h|--chassis`, `-e|--overlay`, `-l|--role`

  In the device mode of cmsh, for example, the *foreach* command has grouping options for selecting devices. These options can be used to select the nodes to loop over, instead of passing a list of objects to *foreach* directly. For example, the --category (-c) option takes a node category argument, while the --node (-n) option takes a node-list argument. Node-lists (specification on page 44) can use the following syntax:

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

**Example**

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

- **Adding options:** `--clone`, `--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**

```
[bright73->device]% foreach --clone node001 -n node022..node024 ()
[bright73->device*]% list | grep node
Type Hostname (key) Ip
---------- -------------- ----------
PhysicalNode node001 10.141.0.1
PhysicalNode node022 10.141.0.22
PhysicalNode node023 10.141.0.23
PhysicalNode node024 10.141.0.24
[bright73->device*]% commit
```

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

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

- **Conditional options:** `--status`, `--quitonunknown`

  The --status (-s) option allows nodes to be filtered by the device status (section 2.1.1).
Example

[bright73->device]% foreach -n node001..node004 --status UP (get IP)
10.141.0.1
10.141.0.3

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

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

Example

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

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

Example

[bright73->category]% foreach * (get name; listnodes)
default

<table>
<thead>
<tr>
<th>Type</th>
<th>Hostname</th>
<th>MAC</th>
<th>Category</th>
<th>Ip</th>
<th>Network</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhysicalNode node001</td>
<td>FA:16:3E:79:4B:77</td>
<td>default</td>
<td>10.141.0.1 internalnet</td>
<td>[ UP ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhysicalNode node002</td>
<td>FA:16:3E:41:9E:A8</td>
<td>default</td>
<td>10.141.0.2 internalnet</td>
<td>[ UP ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhysicalNode node003</td>
<td>FA:16:3E:C0:1F:E1</td>
<td>default</td>
<td>10.141.0.3 internalnet</td>
<td>[ UP ]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

• adhoc (with a comma):
  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

© Bright Computing, Inc.
• sequential extended expansion (only for square brackets):
  example: node[001-002]s[001-005]
  which is:
  node001s001 node001s002 node001s003 node001s004 node001s005
  node002s001 node002s002 node002s003 node002s004 node002s005

• rack-based:
  This is intended to hint which rack a node is located in. Thus:
  - example: r[1-2]n[01-03]
    which is: r1n01,r1n02,r1n03,r2n01,r2n02,r2n03
    This might hint at two racks, r1 and r2, with 3 nodes each.
  - example: rack[1-2]node0[1-3]
    which is: rack1node01,rack1node02,rack1node03,rack2node01,
    rack2node02,rack2node03
    Essentially the same as the previous one, but for nodes that were named more verbosely.

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

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

• mixed list:
  The square brackets and the two dots input specification cannot be used at the same time in one argument. Other than this, specifications can be mixed:
  - example: r1n001..r1n003,r2n003
    which is: r1n001,r1n002,r1n003,r2n003
  - example: r2n003,r[3-5]n0[01-03]
    which is: r2n003,r3n001,r3n002,r3n003,r4n001,r4n002
    r4n003,r5n001,r5n002,r5n003
  - example: node[001-100],-node[004-100:4]
    which is: every node in the 100 nodes, except for every fourth node.

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@bright73 ~]# cat .cm/cmsh/.cmshrc
groupingsyntax auto

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.

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 26) 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@bright73 ~]# cat .cm/cmsh/.cmshrc
bookmark -x mysaved
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 cmgui 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. Monitoring data is also stored in a MySQL database.

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:

<table>
<thead>
<tr>
<th>Service Operation For cmd</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>restart</td>
<td>restart the cluster management daemon</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 (<em>expert only</em>)</td>
</tr>
<tr>
<td>debugon</td>
<td>enable debug logging (<em>expert only</em>)</td>
</tr>
<tr>
<td>debugoff</td>
<td>disable debug logging (<em>expert only</em>)</td>
</tr>
</tbody>
</table>

**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@bright73 etc]# service cmd full-status
CMDaemon version 1.7 is running (active).

Current Time: Thu, 03 Dec 2015 15:43:20 CET  
Startup Time: Thu, 03 Dec 2015 15:42:30 CET
```

© Bright Computing, Inc.
Cluster Management With Bright Cluster Manager

Uptime: 50 seconds
CPU Usage: 4.20251u 1.17037s (10.7%)
Memory Usage: 98.9MB

Sessions Since Startup: 7
Active Sessions: 7

Number of occupied worker-threads: 1
Number of free worker-threads: 11

Connections handled: 119
Requests processed: 119
Total read: 324.7KB
Total written: 42.8KB

Average request rate: 2.38 requests/s
Average bandwidth usage: 6.5KB/s

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 11.1).

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 Configuring The Cluster Management Daemon Logging Facilities

CMDaemon generates log messages 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 Global Debug Mode

It is possible to enable a global debug mode in CMDaemon using cmdaemonctl:

Example

[root@bright73 ~]# cmdaemonctl -h
cmdaemonctl [OPTIONS...] COMMAND ...

Query or send control commands to the cluster manager daemon.

-h --help Show this help

© Bright Computing, Inc.
2.6 Cluster Management Daemon

Commands:
- `debugon` Turn on CMDaemon debug
- `debugoff` Turn off CMDaemon debug
- `full-status` Display CMDaemon status

```
[root@bright73 ~]# 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 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), `OPENSTACK` (for OpenStack), and so on.

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

**Example**

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

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

The logging configuration can be reloaded without having to restart CMDaemon, by restarting the `rsyslogd` system messages logger service with:

**Example**

```
[root@bright73 etc]# systemctl restart rsyslog.service
```

### 2.6.4 Configuration File Modification, And The FrozenFile Directive

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

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

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

Such a file will be entirely overwritten, unless the `FrozenFile` configuration file directive (Appendix C, page 596) is used to keep it frozen.

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

```
# 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:

© Bright Computing, Inc.
Example

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

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

While Bright 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 Bright 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 Bright Cluster Manager for LDAP and PAM.

In such a case the configuration file generated by Bright 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 2.6.4) allows a workaround. Otherwise, the functionality of the Bright 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 2.6.4 or in the lists in Appendixes A.1, A.2 and A.3), are given in Appendix A.4.
Configuring The Cluster

After 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 9). This chapter however goes through a number of basic cluster configuration aspects that are required to get all the hardware up and running on the head and regular nodes.

Section 3.1 explains how some of the main cluster configuration settings can be changed.
Section 3.2 details how the internal and external network parameters of the cluster can be changed.
Section 3.3 describes the setting up of network bridge interfaces.
Section 3.4 describes VLAN configuration.
Section 3.5 describes the setting up of network bond interfaces.
Section 3.6 covers how InfiniBand and Omni-Path are set up.
Section 3.7 describes how Baseboard Management Controllers such as IPMI and iLO are set up.
Section 3.8 describes how switches are set up.
Section 3.9 explains how disk layouts are configured, as well as how diskless nodes are set up.
Section 3.10 describes how NFS volumes are exported from an NFS server and mounted to nodes using the integrated approach of Bright Cluster Manager.
Section 3.11 describes how services can be run from Bright Cluster Manager.
Section 3.12 describes how a rack can be configured and managed with Bright Cluster Manager.
Section 3.13 describes how a GPU unit such as the Dell PowerEdge C410x can be configured with Bright 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.
Puppet configuration is covered in Chapter 14.
3.1 Main Cluster Configuration Settings

In cmgui, the Settings tab for the cluster resource (figure 3.1), which represents the overall cluster object, allows changes to be made to some of the main cluster settings related to names, the administrator e-mail address, networks, and some miscellaneous items.
3.1 Main Cluster Configuration Settings

3.1.1 Cluster Configuration: Various Name-Related Settings
In the tab, the following settings are used for names throughout the cluster.

- Cluster name: (default: Bright 7.3-stable Cluster).
- Default category: (default: default)
- Default software image: (default: default-image)
- How the nodes of the cluster are named:
  - Node name: the base prefix (default prefix: node)
  - Node digits size: number of digits in suffix of node name (default size: 3)

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
The tab 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.

Instead of using the GUI, the changes to the nameserver and searchdomain values can instead be carried out using cmsh in partition mode (page 64).

The names of the search domains used by the cluster have a limit of 6 search domains. This is a hardcoded limit imposed by the Linux operating system. Use of FQDNs is advised as a workaround instead of trying to set more search domains.

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.

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

Time Zone
The time zone setting for the cluster is applied to the time that the cluster uses.

- In cmgui, the time zone can be selected from a menu displayed by clicking on the Timezone button in the Settings tab of the cluster resource (figure 3.1).
- In cmsh, the time zone can be selected in partition mode, using the base object. Tab-completion prompting after entering “set timezone” displays a list of possible time zones, from which one can be chosen:

Example

[bright73]% partition use base
[bright73->partition[base]]% set timezone america/los_angeles
[bright73->partition*[base*]]% commit
### 3.1.3 Miscellaneous Settings

#### BMC (IPMI) Settings

The BMC (Baseboard Management Controller) access settings can be configured here:

- **BMC User ID:** (default: 4)
- **BMC Username:** (default: `bright`)
- **BMC Password:** (default: random string generated during head node installation)

#### Administrator E-mail Setting

By default, the distribution which Bright 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 Bright Cluster Manager so that these e-mails are received elsewhere.

- In cmgui, an e-mail address can be set in the Administrator e-mail section in the Settings tab of the cluster resource (figure 3.1).
- In cmsh, the e-mail address (or space-separated addresses) can be set in partition mode, using the `base` object as follows:

  **Example**

  ```bash
  [bright73]# partition use base
  [bright73->partition[base]]% set administratore-mail goldstein@oceania.com
  obrien@oceania.com
  [bright73->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 cmgui: the SMTP relay host value can be set in the Settings tab for the cluster resource
- in cmsh: the `relayhost` property can be set for the `base` object within partition mode:

  **Example**

  ```bash
  [root@bright73 ~]# cmsh
  [bright73]# partition use base
  [bright73->partition[base]]% set relayhost mail.example.com
  [bright73->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.

  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.
3.1 Main Cluster Configuration Settings

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 *.conf files under \texttt{/etc/sysctl.d/}. Further information on these files can be found via the man page, \texttt{man(5) sysctl.conf}. Bright 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 Bright Cluster Manager is 65536. 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 \texttt{ulimit}.

By default the value set by Bright Cluster Manager is 65536.

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 *.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 Bright Cluster Manager. However, it is set to 65536 by default for a system managed by Bright Cluster Manager. The \texttt{nofile} value is defined by Bright Cluster Manager in:

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

The values set in \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 65536 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 testbed, 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 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.

© Bright Computing, Inc.
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 Bright 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 65 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 62 under the subheading “Changing External Network Parameters For The Cluster”. This is the only network for which Bright 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 69 under the subheading “Changing The Global Network Parameters For The Cluster”.

For a Type 1 cluster (section 3.3.6 of the Installation Manual) the internal and external networks are illustrated conceptually by figure 3.2.

![Network Settings Concepts](image)

Figure 3.2: Network Settings Concepts

The configuration of network settings is completed when, after having configured the general network settings, specific IP addresses are then also assigned to the interfaces of devices connected to the networks.

- Changing the configuration of the head node external interface is described on page 63 under the subheading “The IP address of the cluster”.

© Bright Computing, Inc.
3.2 Network Settings

- Changing the configuration of the internal network interfaces is described on page 67 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 207.

- Changing the configuration of `globalnet` is described on page 69 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 `cmgui`, a network can be configured by selecting the Networks item in the resource tree (figure 3.3).

![Figure 3.3: Networks](image)

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 its tabbed pane. By default, the tab displayed is the Overview tab. This gives a convenient overview of the IP addresses of items assigned in the selected network, grouped as nodes, switches, Power Distribution Units, GPU units, and other devices (figure 3.4).
Selecting the **Settings** tab allows a number of network properties to be changed (figure 3.5).
### 3.2 Network Settings

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>Management network</td>
<td>Switch to treat the network as having nodes managed by the head node.</td>
</tr>
<tr>
<td>network type</td>
<td>Menu options to set the network to be an Internal, External, Global, Tunnel, Cloud, or NetMap network.</td>
</tr>
<tr>
<td>Base address</td>
<td>Base address of the network (also known as the network address)</td>
</tr>
<tr>
<td>Gateway</td>
<td>Default route IP address</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>MTU</td>
<td>Maximum Transmission Unit. The maximum size of an IP packet transmitted without fragmenting.</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.</td>
</tr>
<tr>
<td>start/end</td>
<td>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>Allow node booting</td>
<td>It can be set in cmgui via the Networks resource, selecting a network item, and then choosing Allow node booting from the Settings tabbed pane.</td>
</tr>
<tr>
<td>Don’t allow new nodes to be booted from this network</td>
<td>It can be set in cmsh via the network mode, selecting a network, and then setting nodebooting to yes.</td>
</tr>
</tbody>
</table>

Don’t allow new nodes to be booted from this network

When set to true, 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 Bright Cluster Manager has no MAC addresses associated with the node.

- It can be set in cmsh via the network mode, selecting a network, and then setting lockdowndhcpd to yes.
- It can be set in cmgui via the Networks resource, selecting a network item, and then choosing the Settings tabbed pane.

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**

```
user@bright73:~$ 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 `cm.cluster`, which means that the hostname `node001.cm.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 cmgui the Add button in the networks overview tab of figure 3.3 can be used to add a new network. After the new network has been added, the Settings tab (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 cmgui in the Settings tab 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@bright73 ~]# cmsh
[bright73]# partition use base
[bright73->partition[base]]% set managementnetwork internalnet; commit
[bright73->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.
3.2 Network 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 cmgui, the device listed under Head Nodes in the resource tree is selected and its Settings tab selected from the tabbed pane (figure 3.6). The hostname is changed by modifying the Hostname property and clicking on Save. When setting a hostname, a domain is not included. After the change, as suggested by cmgui, the head node must be rebooted.

![Figure 3.6: Head Node Settings](image)

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

  **Example**

  ```
  [root@bright73 ~]# cmsh
  [bright73]# device use bright73
  [bright73->device[bright73]]% 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@bright73 ~]# sleep 30; hostname -f foobar.cm.cluster
  [root@bright73 ~]#
  ```
Configuring The Cluster

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 Bright 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

[bright73] device use bright73
[bright73->device[bright73]] interfaces
[bright73->device[bright73]->interfaces] use eth1
[bright73->device[bright73]->interfaces[eth1]] set additionalhostnames test
[bright73->device[bright73]->interfaces[eth1]] commit
Fri Oct 12 16:48:47 2012 [notice] bright73: Service named was restarted
[bright73->device[bright73]->interfaces[eth1]] !ping test
PING test.cm.cluster (10.141.255.254) 56(84) bytes of data.
...

Multiple hostnames can be added as space-separated entries.

The named service automatically restarts within about 20 seconds after committal, implementing the configuration changes. This kind of restart is a feature (section 3.11.1) of changes made to service configurations by cmgui 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 the Settings tab of the item chosen from the Networks resource of cmgui. Figure 3.5 shows a settings tab 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.
     - 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.
3.2 Network Settings

- **Gateway**: the default route for the external network.
  - **the network name** (what the external network itself is called), by default this is `externalnet` 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 cmgui (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 3.3.6 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 Bright 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 Bright 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 cmgui or cmsh:

**With cmgui**: The associated cluster network settings tabs are accessed as shown in figure 3.7 for the external network object, and as shown in figure 3.1 for the cluster object.
Setting the static IP address of the head node can be done by selecting the head node from the Head Nodes resources tab, then selecting the Network Setup tabbed pane, then selecting the interface for the address. Clicking on the Edit button opens up an editor that allows the IP address to be changed (figure 3.8).

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

**Example**

```
[bright73]% network use externalnet
[bright73->network[externalnet]]% set baseaddress 192.168.1.0
[bright73->network*[externalnet*]]% set netmaskbits 24
[bright73->network*[externalnet*]]% set gateway 192.168.1.1
[bright73->network*[externalnet*]]% commit
[bright73->network[externalnet]]% partition use base
[bright73->partition[base]]% set nameservers 192.168.1.1
[bright73->partition*[base*]]% set searchdomains x.com y.com
[bright73->partition*[base*]]% append timeservers ntp.x.com
[bright73->partition*[base*]]% commit
[bright73->partition[base]]% device use bright73
[bright73->device[bright73]]% interfaces
[bright73->device[bright73]->interfaces]% use eth1
[bright73->device[bright73]->interfaces[eth1]]% set ip 192.168.1.176
[bright73->device[bright73]->interfaces*[eth1*]]% commit
[bright73->device[bright73]->interfaces[eth1]]%
```

After changing the external network configurations, a reboot of the head node is necessary to activate the changes.

© Bright Computing, Inc.
3.2 Network Settings

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 cmgui, the DHCP checkbox of figure 3.8 can be ticked
- Alternatively, in cmsh, within interfaces mode for the head node interface, the value of the parameter DHCP can be set:

  ```
  [bright73->device[bright73]->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 3.28 in Chapter 3 of the Installation Manual). The time servers can be changed using cmgui 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 66)
2. the cluster network for the internal network (page 67)
3. the individual device network interface (page 67)

4. the node categories network-related items for the device (page 68), 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 \texttt{internalnet} interface, or a managed switch on the internal network. In \texttt{cmgui}, the settings can be configured in the **Settings** tab for the \texttt{internalnet} item selected in the **Networks** resource (figure 3.5). In \texttt{cmsh} the settings can be configured by changing the values in the \texttt{internalnet} object within \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{10.141.128.0} to \texttt{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.
     - Allow 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 one of the possibilities (section 5.1.3). Only if the Allow node booting option is ticked does ticking the “Don’t allow new nodes to be booted from this network” checkbox have any effect, and stop new nodes from booting. 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 \texttt{cm.cluster}.
   
   - the network name, or what the internal network is called. By default, set to \texttt{internalnet}.
   
   - The \texttt{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 tab of the cluster object resource in cmgui (figure 3.1). The cmsh equivalents can be configured from the base object in partition mode. Values that can be set include:

- the “Management network”. This is the network over which CMDaemon manages the nodes. Management means that CMDaemon on the head node communicates with CMDaemons on the other nodes. The communication includes CMDaemon calls and monitoring data. By default, the management network is set to internalnet for Type 1 and Type 2 networks, and managementnet in Type 3 networks. It is a partition-level cluster-wide setting by default. Partition-level settings can be overridden by the category level setting, and node-level settings can override category level or partition level settings.

- the “Node name” can be set to decide the prefix part of the node name. By default, set to node.

- the “Node digits” can be set to decide the possible size of numbers used for suffix part of the node name. By default, set to 3.

- the “Default category”. This sets the category the nodes are in by default. By default, it is set to default.

- the “Default software image”. This sets the image the nodes use by default, for new nodes that are not in a category yet. By default, it is set to default-image.

- the “Name server”. This sets the name server used by the cluster. By default, it is set to the head node. The default configuration uses the internal network interface and serves only internal hosts. Internal hosts can override using this value at category level (page 68). 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: The “Network Setup” tab can be chosen for a node selected from the Nodes resource in cmgui. The tab then allows network configuration to be done for 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:
### 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 67 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**

```
[bright73] device interfaces node001
[bright73->device[node001]->interfaces]% add physical eth1
[bright73->...->interfaces*[eth1*]]% set network externalnet
[bright73->...->interfaces*[eth1*]]% set additionalhostnames extra01
[bright73->...->interfaces*[eth1*]]% set ip 10.141.1.1
[bright73->...->interfaces*[eth1*]]% commit
[bright73->...->interfaces[eth1]]% ..
[bright73->...->interfaces]% ..
[bright73->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 cmgui, using the Settings tab for the default node category under Node Categories,
- or via the category mode in cmsh, for the default node category.

If there are individual node settings that have been configured in cmgui 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 cmgui 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 65, 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 Bright 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 a SYN 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 Bright Cluster Manager. It is required in Bright 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 5.4.1 of the Cloudbursting Manual.

Setting Static Routes

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 3.3.6 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:
then an example session for setting the static route is as follows:

**Example**

```
[root@bright73 ~]# cmsh
[bright73 ]% device use node001
[bright73->device[node001]]% staticroutes
[bright73->device[node001]->staticroutes]% list
Name (key) Gateway Destination
-------------------------------- ---------------- --------------
[bright73->device[node001]->staticroutes% add printerroute
[bright73->...[printerroute*]]% set gateway 10.141.1.1
[bright73->...[printerroute*]]% set destination 192.168.0.3
[bright73->...[printerroute*]]% set networkdevicename bootif
[bright73->...[printerroute*]]% commit
[bright73->...[printerroute*]]% show
Parameter Value
-------------------------------- -------------------------------
Destination 192.168.0.3/32
Gateway 10.141.1.1
Name printerroute
Network Device Name bootif
Revision
[bright73->device[node001]->staticroutes[printerroute]]% exit
[bright73->device[node001]->staticroutes]% list
Name (key) Gateway Destination
-------------------------------- ---------------- --------------
printerroute 10.141.1.1 192.168.0.3/32
[bright73->device[node001]->staticroutes% exit; exit
[bright73->device]% reboot node001
```
In the preceding example, the regular node node001, with IP address 10.141.0.1 can connect to a host 192.168.0.3 outside the regular node subnet using a gateway with the IP addresses 10.141.1.1 and 192.168.0.1. The route is given the arbitrary name printerroute for administrator and CMDaemon convenience, because the host to be reached in this case is a print server. The networkdevicename is given the interface name bootif. If another device interface name is defined in CMDaemon, then that can be used instead. If there is only one interface, then networkdevicename need not be set.

After a reboot of the node, the route that packets from the node follow can be seen with a traceroute or similar. The ping utility with the -R option can often track up to 9 hops, and can be used to track the route:

Example

```
[root@bright73 ~]# ssh node001 ping -c1 -R 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(124) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=62 time=1.41 ms
RR: 10.141.0.1
   10.141.1.1
   192.168.0.1
   192.168.0.3
   192.168.0.1
   10.141.1.1
   10.141.0.1

--- 192.168.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 1ms
rtt min/avg/max/mdev = 1.416/1.416/1.416/0.000 ms
```

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).

### 3.3 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**

```
[bright73->device[node001]->interfaces]% add bridge br0
[bright73->...->interfaces*[br0*]]% set network internainet
```
A bridge interface is composed of one or more physical interfaces. The IP and network fields of the member interfaces must be empty:

**Example**

```
[bright73->...->interfaces*[br0*]]% set interfaces eth1 eth2; exit
[bright73->...->interfaces*]% clear eth1 ip; clear eth1 network
[bright73->...->interfaces*]% clear eth2 ip; clear eth2 network
[bright73->...->interfaces*]% use br0; commit
```

The **BOOTIF** interface is also a valid interface option.

Currently, the following bridge properties can be set:

- **SpanningTreeProtocol**: sets the spanning tree protocol to be on or off. The default is off.
- **Forward Delay**: sets the delay for forwarding Ethernet frames, in seconds. The default is 0.

Additional properties, if required, can be set manually using the `brctl` command in the OS shell.

When listing interfaces in `cmsh`, if an interface is a member of a bond (or bridge) interface, then the corresponding bond or bridge interface name is shown in parentheses after the member interface name:

**Example**

```
[headnode->device[node001]->interfaces]% list

<table>
<thead>
<tr>
<th>Type</th>
<th>Network device name</th>
<th>IP</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>bond</td>
<td>bond0 [prov]</td>
<td>10.141.128.1</td>
<td>internalnet</td>
</tr>
<tr>
<td>bridge</td>
<td>br0</td>
<td>10.141.128.2</td>
<td>internalnet</td>
</tr>
<tr>
<td>physical</td>
<td>eth0</td>
<td>10.141.0.1</td>
<td>internalnet</td>
</tr>
<tr>
<td>physical</td>
<td>eth1 (bond0)</td>
<td>0.0.0.0</td>
<td></td>
</tr>
<tr>
<td>physical</td>
<td>eth2 (bond0)</td>
<td>0.0.0.0</td>
<td></td>
</tr>
<tr>
<td>physical</td>
<td>eth3 (br0)</td>
<td>0.0.0.0</td>
<td></td>
</tr>
<tr>
<td>physical</td>
<td>eth4 (br0)</td>
<td>0.0.0.0</td>
<td></td>
</tr>
</tbody>
</table>
```

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.

© Bright Computing, Inc.
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 Bright Cluster Manager using `cmsh` or `cmgui`.

### 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@bright73 ~]# cmsh
[bright73]# device interfaces node001
[bright73->device[node001]->interfaces]% list
Type Network device name IP Network
---------- -------------------- -------------- ------------
physical BOOTIF [prov] 10.141.0.1 internalnet

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

[bright73->device[node001]->interfaces]% add vlan BOOTIF.1
[bright73->device[node001]->interfaces[BOOTIF.1]]% commit

[bright73->device[node001]->interfaces[BOOTIF.1]]% show
Parameter Value
------------------------- ------------------------
Additional Hostname        
DHCP                      no
IP                        0.0.0.0
MAC                      00:00:00:00:00:00
Network                   BOOTIF.1
Reorder HDR               no
Revision
Type                      vlan

[bright73->device[node001]->interfaces[BOOTIF.1]]% set network internalnet
[bright73->device[node001]->interfaces[BOOTIF.1]]% set ip 10.141.2.1; commit
```

### 3.4.2 Configuring A VLAN Interface Using `cmgui`

Within `cmgui` a VLAN interface can be configured by selecting its node from the resources, selecting the **Network Setup** tabbed pane, and clicking on the **Add** button. The interface type is then selected and a dialog opens up allowing an IP address, network, and other options to be set (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 68) 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

```
[bright73->device[node001]->interfaces]% add bond bond0
[...->device*[node001*]->interfaces*[bond0*]]% set network internalnet
[...->device*[node001*]->interfaces*[bond0*]]% set ip 10.141.128.1
[...->device*[node001*]->interfaces*[bond0*]]% set mode 3
[...->device*[node001*]->interfaces*[bond0*]]% set interfaces eth1 eth2
```

Each bonded interface has a unique set of object properties, called bonding options, that specify how the bonding device operates. The bonding options are a string containing one or more options formatted as `option=value` pairs, with pairs separated from each other by a space character.

A bonded interface also always has a mode associated with it. By default it is set to 0, corresponding to a balanced round-robin packet transmission.
3.5 Configuring Bonded Interfaces

The 7 bonding modes are:

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

Technically, outside of cmgui or cmsh, the bonding mode is just another bonding option specified as part of the options string. However in Bright 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 72.

- 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 13), 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 13.1) 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 13.1) 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 187.

**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[headnode1]->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`.

© Bright Computing, Inc.
When listing interfaces in cmsh, if an interface is a member of a bond or bridge interface, then the corresponding bonded or bridge interface name is shown in parentheses after the member interface name. Section 3.3, on configuring bridge interfaces, shows an example of such a listing from within cmsh on page 72.

More on bonded interfaces (including a detailed description of bonding options and modes) can be found at http://www.kernel.org/doc/Documentation/networking/bonding.txt.

3.6 Configuring InfiniBand And Omni-Path Interfaces

On clusters with an InfiniBand interconnect, the InfiniBand Host Channel Adapter (HCA) in each node must be configured before it can be used. A similar requirement applies for Omni-Path and its fabric controller. Bright 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 Bright 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 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 Bright 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 Bright 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 Bright Cluster Manager, an administrator sets a subnet manager service\footnote{usually opensm, but opensmd in SLES} to start at boot-time with a command such as: “chkconfig opensm on”. However, for clusters managed by Bright Cluster Manager, a subnet manager is best set up using CMDaemon.
There are two ways of setting CMDaemon to start up the subnet manager on a node at boot time:

1. by assigning a role.
   
   In cmsh this can be done with:
   
   ```
   [root@bright73 ~]# cmsh -c "device roles <node>; assign subnetmanager; commit"
   ```
   
   where `<node>` is the name of a node on which it will run, for example: `bright73, node001, node002`...

   In cmgui, the subnet manager role is assigned by selecting a head node or regular node from the resources tree, and assigning it the “Subnet Manager Role” from the “Roles” tab.

2. by setting the service up. Services are covered more generally in section 3.11.
   
   In cmsh this is done with:
   
   **Example**
   
   ```
   [root@bright73 ~]# cmsh
   [bright73]# device services node001
   [bright73->device[node001]->services]% add opensm
   [bright73->device[node001]->services*[opensm*]]% set autostart yes
   [bright73->device[node001]->services*[opensm*]]% set monitored yes
   [bright73->device[node001]->services*[opensm*]]% commit
   [bright73->device[node001]->services[opensm]]%
   ```

   In cmgui the subnet manager service is configured by selecting a head node or regular node from the resources tree, and adding the service to it.

   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. It can be configured to run in connected mode by setting the `connectedmode` 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@bright73 ~]# 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@bright73 ~]# 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:

```
[bright73->device[bright73]->interfaces]% add physical ib0
[bright73->device[bright73]->interfaces*[ib0*]]% set network ibnet
[bright73->device[bright73]->interfaces*[ib0*]]% set ip 10.149.255.254
[bright73->device[bright73]->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 Verifying Connectivity

After all nodes have been restarted, the easiest way to verify connectivity is to use the `ping` utility

Example

Pinging node015 while logged in to node014 through the InfiniBand interconnect:

```
[root@node014 ~]# ping node015.ib.cluster
PING node015.ib.cluster (10.149.0.15) 56(84) bytes of data.
64 bytes from node015.ib.cluster (10.149.0.15): icmp_seq=1 ttl=64
time=0.086 ms
...
```

If the ping utility reports that ping replies are being received, the InfiniBand is operational. The `ping` utility is not intended to benchmark high speed interconnects. For this reason it is usually a good idea to perform more elaborate testing to verify that bandwidth and latency are within the expected range.

The quickest way to stress-test the InfiniBand interconnect is to use the Intel MPI Benchmark (IMB), which is installed by default in `/cm/shared/apps/imb/current`. The `setup.sh` script in this directory can be used to create a template in a user’s home directory to start a run.

Example

Running the Intel MPI Benchmark using `openmpi` to evaluate performance of the InfiniBand interconnect between `node001` and `node002`:
[root@bright73 ~]# su - cmsupport
[cmsupport@bright73 ~]$ cd /cm/shared/apps/imb/current/
[cmsupport@bright73 current]$ ./setup.sh
[cmsupport@bright73 current]$ cd ~/BenchMarks/imb/3.2.2
[cmsupport@bright73 3.2.2]$ module load openmpi/gcc
[cmsupport@bright73 3.2.2]$ module initadd openmpi/gcc
[cmsupport@bright73 3.2.2]$ make -f make_mpi2
[cmsupport@bright73 3.2.2]$ mpirun -np 2 -machinefile ../nodes IMB-MPI1 PingPong
#---------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#---------------------------------------------------
#bytes #repetitions t[usec] Mbytes/sec
0 1000 0.78 0.00
1 1000 1.08 0.88
2 1000 1.07 1.78
4 1000 1.08 3.53
8 1000 1.08 7.06
16 1000 1.16 13.16
32 1000 1.17 26.15
64 1000 1.17 52.12
128 1000 1.20 101.39
256 1000 1.37 177.62
512 1000 1.69 288.67
1024 1000 2.30 425.34
2048 1000 3.46 564.73
4096 1000 7.37 530.30
8192 1000 11.21 697.20
16384 1000 21.63 722.24
32768 1000 42.19 740.72
65536 640 70.09 891.69
131072 320 125.46 996.35
262144 160 238.04 1050.25
524288 80 500.76 998.48
1048576 40 1065.28 938.72
2097152 20 2033.13 983.71
4194304 10 3887.00 1029.07
# All processes entering MPI_Finalize

To run on nodes other than node001 and node002, the ../nodes file must be modified to contain different hostnames. To perform other benchmarks, the PingPong argument should be omitted.

### 3.7 Configuring BMC (IPMI/iLO) Interfaces

Bright 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 (section 3.3.7 of the Installation Manual), or it can be set after installation as described in this section. The IPMI, iLO, or UCS 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 Bright Cluster Manager.
3.7 Configuring BMC (IPMI/iLO) Interfaces

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, or ipminet</td>
</tr>
<tr>
<td>Domain name</td>
<td>bmc.cluster, ilo.cluster, or ipmi.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 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:

```bash
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
```

The preceding loop can conveniently be replaced with the addinterface command, run from within the device mode of cmsh:

```bash
echo "device addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1 commit" | cmsh -x
```

The help text for addinterface gives more details on how to use it.

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:

```
device[device]->interfaces[% add alias eth0:0
device[device]->interfaces*[eth0:0*] set network bmcnet
device[device]->interfaces*[eth0:0*] set ip 10.148.255.254
device[device]->interfaces*[eth0:0*] commit
```
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.

### 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 in the parameter `bmcpassword`, which can be managed from `cmsh` from within the `base` object of `partition` mode. 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**

```bash
[bright73]\% partition use base
[bright73->partition[base]]\% get bmcusername
bright
[bright73->partition[base]]\% get bmcpassword
Za4ohnliohMa2zew
[bright73->partition[base]]\% set bmcusername bmcadmin
[bright73->partition*[base*]]\% set bmcpassword
enter new password: ******
retype new password: ******
[bright73->partition*[base*]]\% commit
[bright73->partition[base]]\%
```

In cmgui, selecting the cluster item in the resources pane, and then using the Settings tabbed pane, allows the BMC settings to be edited.

It is possible to set the BMC authentication credentials 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</td>
</tr>
<tr>
<td>BMC Password</td>
<td>Password for specified user name</td>
</tr>
</tbody>
</table>

The Bright Cluster Manager stores the configured BMC username and password. These are used:

- 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.

© Bright Computing, Inc.
3.7.3 Interfaces Settings

Interface Name

It is recommended that the network device name of a BMC interface start with `ipmi` or `ilo`, according to whether the BMC is running with IPMI or iLO. 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 Bright Cluster Manager. A node with a DHCP-assigned BMC IP address, requires custom BMC power management (section 4.1.4) due to its dynamic nature.

3.8 Configuring Switches And PDUs

3.8.1 Configuring With The Manufacturer’s Configuration Interface

Network switches and PDUs that will 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 `cmgui`, this is done by selecting the `Switches` resource, selecting the particular switch from within that resource, then selecting the associated `Settings` tab (figure 3.11). The IP address can then be set and saved.

- In `cmsh` this can be done in `device` mode, with a `set` command:

  Example

  ```
  [root@bright73 ~]# cmsh
  [bright73]# device
  [bright73->device]$ set switch01 ip 10.141.253.2
  [bright73->device*]$ commit
  ```

3.8.2 Configuring SNMP

Moreover, in order to allow the cluster management software to communicate with the switch or PDU, SNMP must be enabled and the SNMP community strings should be configured correctly. By default, the SNMP community strings for switches and PDUs are set to `public` and `private` for 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

```
[bright73]$ device use switch01
[bright73->device[switch01]]$ get readstring
public
[bright73->device[switch01]]$ get writestring
private
[bright73->device[switch01]]$ set readstring public2
[bright73->device[switch01*]]$ set writestring private2
[bright73->device[switch01*]]$ commit
```

Alternatively, these properties can also be set via `cmgui`, from the `Switches` resource, selecting the switch, and then modifying the string settings in the `Settings` tab.
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, cmgui or cmsh are used:

- In cmgui, the switch is selected from the Switches folder, and the Settings tabbed pane is opened (figure 3.11). The port number corresponding to uplink port number is filled in the blank field beside the “Uplink:” label. More uplinks can be appended by clicking on the + widget. The state is saved with the Save button.

![Figure 3.11: Notifying CMDaemon About Uplinks With cmgui](image)

- In cmsh, the switch is accessed from the device mode. The uplink port numbers can be appended one-by-one with the append command, or set in one go by using space-separated numbers.

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# device
[bright73->device]# set switch01 uplinks 15 16
[bright73->device*]# set switch02 uplinks 01
[bright73->device*]# commit
successfully committed 3 Devices
```

3.8.4 The showport MAC Address to Port Matching Tool

The showport command can be used in troubleshooting network topology issues, as well as checking and setting up new nodes (section 5.4.2).

**Basic Use Of showport**

In the device mode of cmsh is the showport command, which works out which ports on which switch are associated with a specified MAC address.

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# device
[bright73->device]# showport 00:30:48:30:73:92
[bright73->device]# switch01:12
```
When running `showport`, CMDaemon on the head node queries all switches until a match is found. If a switch is also specified using the “-s” option, then the query is carried out for that switch first. Thus the preceding example can also be specified as:

```
[bright73->device]% showport -s switch01 00:30:48:30:73:92
[bright73->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:

**Example**

```bash
#!/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
00:30:48:30:73:92 bright73: 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 `ethernetswitch` setting of a node, a category, or a group. For example, in `cmsh`, for a node:

**Example**

```
[root@bright73 ~]# cmsh
[bright73]% device use node001
[bright73->device[node001]]% clear ethernetswitch
[bright73->device[node001++]]% commit
[bright73->device[node001]]% 
```

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

**Example**

```
[root@bright73 ~]# cmsh
[bright73]% device
[bright73->device]]% foreach -c default {clear ethernetswitch}; 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):

```
[bright73->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>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>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</td>
<td>node34</td>
<td>false</td>
<td>5C:F9:DD:F5:78:3D, node34</td>
</tr>
<tr>
<td>54</td>
<td>UP</td>
<td>node35</td>
<td>false</td>
<td>5C:F9:DD:F5:45:AC, node35</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 3.3.17 of the Installation Manual). For regular nodes, the disk layout can also be re-configured by Bright Cluster Manager once the cluster is running. For a head node, however, the disk layout cannot be re-configured after installation by Bright 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 cmgui 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 cmgui or cmsh as follows:

Changing A Disk Layout Using cmgui

To change a disk layout with cmgui, the current disk layout is accessed by selecting a node category or a specific node from the resource tree. The “Disk Setup” tab is then selected. Clicking on the Load 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, 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@bright73 ~]# cmsh
[bright73]# device use node001
[bright73->device[node001]]# set disksetup
```

After editing and saving the XML configuration, the change is then committed to CMDaemon with the commit command.

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

```
[bright73->device[node001]]% set disksetup /cm/shared/apps/cmgui/disk-setup/slave-diskless.xml
```

In these preceding cmgui 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 Bright 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.

- For RHEL6, the client module and server module for NFS over RDMA are provided by default as a combined kernel module. The module is built with the compile option:
  
  `– CONFIG_SUNRPC_XPRTEST_RDMA`

- In RHEL7 only the client module is provided by default. The module is built with the compile option:
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_CLIENT

However, this is currently (January 2015) unstable and unsupported by Red Hat. The server module may be usable for custom-built kernel versions beyond 3.17. The Bright Computing suggestion for an administrator who needs a very fast network filesystem, is Lustre (section 7.8 of the Installation Manual) 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 cmgui via the “Services” tab (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@bright73 ~]# cmsh
[bright73]% device use bright73
[bright73->device[bright73]]% services
[bright73->device[bright73]->services]% add autofs
[bright73->device[bright73]->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>
</tbody>
</table>

[bright73->device[bright73]->services*{autofs}%]% set autostart yes
[bright73->device[bright73]->services*{autofs}%]% commit
```

Filesystems imported via an auto-mount operation must explicitly be excluded in excludelistupdate by the administrator, as explained in section 5.6.1, page 199.

The rest of this section describes the configuration of NFS for static mounts, using cmgui 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 cmgui 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.

© Bright Computing, Inc.
3.10.1 Exporting A Filesystem Using cmgui And cmsh

Exporting A Filesystem Using cmgui

As an example, if an NFS volume exists at "bright73:/modeldata" it can be exported using cmgui as follows:

The head node is selected from under the "Head Nodes" resource, and the "FS Exports" tab is then selected. This shows the list of exports (figure 3.13):

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

Using the Add button, a new entry (figure 3.14) can be configured with values as shown:

![Figure 3.14: Setting Up An NFS Export Using cmgui](image)

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 Allowed hosts is set from the selection menu to allowing access to internalnet (by default 10.141.0.0/16 in CIDR notation).

By setting the Write option to Disabled, read-only access is kept.
3.10 Configuring NFS Volume Exports And Mounts

Saving this preceding configuration means the NFS server now provides NFS access to this filesystem for `internalnet`.

The network can be set to other network values using CIDR notation. It can also be set to particular hosts such as just node001 and node002, by specifying a value of "node001 node002" instead. Other settings and options are also possible and are given in detail in the man pages for `exports(5)`.

Exporting A Filesystem Using cmsh

The equivalent to the preceding `cmgui` NFS export procedure can be done in `cmsh` by using the `fsexports` submode on the head node (some output elided):

Example

```
[root@bright73 ~]# cmsh
[bright73] % device use bright73
[bright73->device[bright73]] % 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Hosts</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluid Model Data</td>
<td>/modeldata</td>
<td>10.141.0.0/16</td>
<td>no</td>
</tr>
</tbody>
</table>
```

3.10.2 Mounting A Filesystem Using cmgui And cmsh

Continuing on with the export example from the preceding section, the administrator decides to mount the remote filesystem over the default category of nodes. Nodes can also mount the remote filesystem individually, but that is usually not a common requirement in a cluster. The administrator also decides not to re-use the exported name from the head node. That is, the remote mount name `modeldata` is not used locally, even though NFS allows this and many administrators prefer to do this. Instead, a local mount name of `/modeldatagpu` is used, perhaps because it avoids confusion about which filesystem is local to a person who is logged in, and perhaps to emphasize the volume is being mounted by nodes with GPUs.

Mounting A Filesystem Using cmgui

Thus, in `cmgui`, values for the remote mount point (`bright73:/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):
Figure 3.15: Setting Up NFS Mounts On A Node Category Using cmgui

Saving the configuration using the Ok button 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 cmgui NFS mount procedure can be done in cmsh by using the fsmounts submode, for example on the default category. The add method under the fsmounts submode sets the mountpoint path, in this case /modeldatagpu (some output elided):

**Example**

```
[root@bright73 ~]# cmsh
[bright73]% category use default
[bright73->category[default]]% fsmounts
[bright73->category[default]->fsmounts]% add /modeldatagpu
[bright73->...[/modeldatagpu*]]% set device bright73:/modeldata
[bright73->...[/modeldatagpu*]]% set filesystem nfs
[bright73->category[default*]->fsmounts*[/modeldatagpu*]]% commit
[bright73->category[default]->fsmounts[/modeldatagpu]]
Device                      Mountpoint (key)        Filesystem
-------------------------- -------------------------- ------------
...                        bright73:/modeldata      nfs
[bright73->category[default]->fsmounts[/modeldatagpu]]% show
Parameter            Value
-------------------------- ------------
Device               bright73:/modeldata
Dump                  no
```
3.10 Configuring NFS Volume Exports And Mounts

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>nfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filesystem Check</td>
<td>0</td>
</tr>
<tr>
<td>Mount options</td>
<td>defaults</td>
</tr>
<tr>
<td>Mountpoint</td>
<td>/modeldatagpu</td>
</tr>
</tbody>
</table>

Values can be set for “Mount options” other than default. For example, the “noac” flag can be added as follows:

[bright73->...[/modeldatagpu]]% set mountoptions defaults,noac; commit

Mounting a CIFS might use:

[bright73->...[/modeldatagpu]]% set mountoptions gid,users,file_mode=0666,dir_mode=0777,\iocharset=iso8859-15,credentials=/path/to/credential
[bright73->...[/modeldatagpu*]]% 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, Bright 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 option, then CMDaemon changes to /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 11.1), 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 cmgui, viewing category-inherited mount points for an individual node requires checking the checkbox for “Display category inherited mounts” in the FS Mounts tabbed view for that node (figure 3.16).
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@bright73 ~]# 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>$localnfsserv+</td>
<td>/cm/shared</td>
<td>nfs</td>
</tr>
<tr>
<td>[default] bright73:/home</td>
<td>/home</td>
<td>nfs</td>
</tr>
<tr>
<td>bright73:/cm/shared/eva+</td>
<td>/home/examples</td>
<td>nfs</td>
</tr>
<tr>
<td>[root@bright73 ~]#</td>
<td></td>
<td></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, or by using the arrow buttons at the bottom of the FS Mounts tabbed pane.

### 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 Bright Cluster Manager 7.3, the administrator should be aware that mounting certain paths such as \( /bin \) is not possible.

When cmgui 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@bright73 ~]# cmsh
[bright73]% device use bright73; fsexports
[bright73->device[bright73]->fsexports]% add defaultimage
[bright73...defaultimage*]% set path /cm/images/default-image
[bright73...defaultimage*]% set hosts 10.141.0.0/16
[bright73...defaultimage*]% commit
[bright73...defaultimage*]% list | grep defaultimage

Name (key) Path Hosts Write
------------------- ------------------------ ------------- -----
defaultimage /cm/images/default-image 10.141.0.0/16 no
```

As the output to list shows, the NFS export should be kept read-only, which is the default. Appropriate parts of the export can then be mounted by a node or node category. The mount is defined by setting the mount point, the nfs filesystem property, and the export device. For example, for a node category (some output elided):

```
[br...defaultimage*]% category use default
[bright73->category[default]]% fsmounts
[bright73->category[default]->fsmounts]% add /usr/share/doc
[bright73->category[default]->fsmounts]% set device bright73:/cm/images/default-image/user/share/doc
[bright73->category[default]->fsmounts]% set filesystem nfs
[bright73->category[default]->fsmounts]% commit
[bright73->category[default]->fsmounts]% list

Device Mountpoint (key) Filesystem
--------------------- ------------------ ----------
... ... ...
bright73:/cm/images/usr/share/doc /usr/share/doc nfs
```

2by default \(<image>\) is default-image on a newly-installed cluster

© Bright Computing, Inc.
Other mount points can be also be added according to the judgment of the system administrator. Some consideration of mount order may be needed, as discussed on page 94 under the subheading “Mount Order Considerations”.

**An Example Of Several NFS Subtree Mounts**

The following mounts save about 500MB from tmpfs on a diskless node with CentOS 6, as can be worked out from the following subtree sizes:

```
[root@bright73 ~]# cd /cm/images/default-image/
[root@bright73 default-image]# du -sh usr/share/locale usr/java usr/share/doc usr/src
262M  usr/share/locale
78M    usr/java
107M   usr/share/doc
45M    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@bright73 default-image]# cmsh
[bright73]% category use default; fsmounts
[bright73->category[default]->fsmounts]% list -f device:53,mountpoint:17
```

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 6 or 7 system:

1. The full diskless disk setup is partitioned first:

   ```
   [root@bright73 ~]# cmsh
   [bright73]% category use default
   ```
### 3.10 Configuring NFS Volume Exports And Mounts

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

```
[bright73->category[default]]% device use master; fsexports
[bright73->device[bright73]->fsexports]% add /cm/images/default-image
[bright73->device[bright73]->fsexports]% set hosts internalnet
[bright73->device[bright73]->fsexports]% set name default@internal
[bright73->device[bright73]->fsexports]% commit #exports default image
[bright73->category[default]]% quit
```

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

```
[root@bright73 ~]# cmsh -c "category use default; fsmounts; add /; set
device master:/cm/images/default-image; set mountoptions defaults,ro;
set filesystem nfs; commit" #adds readonly root via nfs
```

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

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

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

```
[root@bright73 ~]# cat mountfiles #for centos 6
/var/cache
/var/log
/var/lock
/var/run
/var/tmp
/var/spool
/tmp
/dev
/cm/local/apps/cmd/etc
/cm/local/apps/openldap/etc
/cm/local/apps/sge/var
/cm/local/apps/torque/var
/cm/local/apps/slurm/var
/cm/local/apps/pbspro/var
/cm/local/apps/openlava/var
/etc
```

For CentOS 7 the subtrees are slightly different:

```
[root@bright73 ~]# cat mountfiles #for centos 7
/var/cache
/var/log
/var/tmp
/var/spool
```

© Bright Computing, Inc.
The /cm/local/apps/openlava/var directory does not exist by default, and so it should only be placed in mountfiles if the OpenLava software has been installed (section 7.5.6). The various workload manager /var directories for SGE/Torque/Slurm/PBS Pro/OpenLava are only needed if these workload managers are to be set up and used.

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

```
[root@bright73 ~]# (echo "category use default; fsmounts"
for i in $(<mountfiles)
    do
        echo "add $i; set device tmpfs; set filesystem tmpfs; exit"
    done
echo "commit" ) | cmsh
```

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

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

### 3.10.5 Configuring NFS Volume Exports And Mounts Over RDMA With OFED Drivers

If running NFS over RDMA, then at least NFS version 4.0 is recommended. NFS version 3 will also work with RDMA, but uses IPoIB encapsulation instead of native verbs. NFS version 4.1 uses the RDMA Connection Manager (`librdmacm`), instead of the InfiniBand Connection Manager (`ib_cm`) and is thereby also able to provide pNFS.

The administrator can set the version of NFS used by the cluster by setting the value of `Nfsvers` in the file `/etc/nfsmount.conf` on all the nodes, including the head node.

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

The use of the RDMA protocol (section 3.6) to provide NFS, by installing updated Bright 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.

However, NFS over RDMA does work within Bright Cluster Manager with the OFED drivers that the parent distribution provides, in the case of RHEL6/SL6/CENTOS6. For the remaining distributions, this 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.
3.11 Managing And Configuring Services

Exporting With cmgui And cmsh Using NFS Over RDMA

With the drivers installed, a volume export can be carried out using NFS over RDMA.

The procedure using cmgui is much the same as done in section 3.10.1 (“Exporting A Filesystem Using cmgui”), except for that the ibnet network should be selected instead of the default internalnet, and the “NFS over RDMA” checkbox should be ticked.

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 cmgui And cmsh Using NFS Over RDMA

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

The procedure using cmgui is largely like that in section 3.10.2, (“Mounting A Filesystem Using cmgui”), except that the Device entry must point to master.ib.cluster so that it resolves to the correct NFS server address for RDMA, and the checkbox for NFS over RDMA must be ticked.

The procedure using cmsh is similar to that in section 3.10.2, (“Mounting A Filesystem Using cmsh”), except that device must be mounted to the ibnet, and the rdma option must be set, as shown:

Example

(based on the example in section 3.10.2)

... 
[bright73->category*[default*]->fsmounts*]% add /modeldatagpu 
[bright73->...*[/modeldatagpu*]]% set device bright73.ib.cluster:/modeldata 
[bright73->...*[/modeldatagpu*]]% set filesystem nfs 
[bright73->...*[/modeldatagpu*]]% set rdma yes 
[bright73->category*[default*]->fsmounts*][/modeldatagpu*]]% commit 
...

3.11 Managing And Configuring Services

3.11.1 Why Use The Cluster Manager For Services?

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

  chkconfig

and

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

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

Already installed services can also be brought under Bright Cluster Manager control, and started and stopped with cmdgui 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 Bright Cluster Manager, then no attempts are made to restart the service. Attempted autostarts become possible again only after Bright Cluster Manager starts the service again.
  - If **autostart** is set to **on**, and if a service is removed using Bright 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 62.

• **runif:** (only honored for nodes used as part of a high-availability configuration (chapter 13)) 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. Only honored for head nodes.

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

• Thus Bright 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 example can be found within the file `/usr/share/doc/initscripts-<version>/sysvinitfiles` in RHEL6 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.

Bright 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 13.4.6). 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@bright73 ~]# chkconfig cups on
[root@bright73 ~]# service cups start
```
3.11 Managing And Configuring Services

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):

```
[bright73]% device services node001
[bright73->device[node001]->services]% add cups
[bright73->device*[node001*]->services*[cups*]]% show
Parameter                  Value
----------------------------------  -------------------------
Autostart                   no
Belongs to role             no
Monitored                   no

Run if                      ALWAYS
Service                     cups

...[bright73->device*[node001*]->services*[cups*]]% set monitored on
[bright73->device*[node001*]->services*[cups*]]% set autostart on
[bright73->device*[node001*]->services*[cups*]]% commit
[bright73->device[node001]->services[cups]]%
Apr 14 14:02:16 2017 [notice] node001: Service cups was started
[bright73->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:

```
[bright73->device[node001]->services[cups]]% stop
Fri Apr 14 14:03:40 2017 [notice] node001: Service cups was stopped
Successfully stopped service cups on: node001
[bright73->device[node001]->services[cups]]%
```

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

```
[bright73->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 cmgui” on page 103.

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

```
[bright73->device[node001]->services[cups]]% category
[bright73->category]% services default
[bright73->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
[bright73->category*[default*]->services*[cups*]]% show ...
[bright73->category*[default*]->services*[cups*]]% set autostart yes
[bright73->category*[default*]->services*[cups*]]% set monitored yes
[bright73->category*[default*]->services*[cups*]]% commit
[bright73->category*[default*]->services*[cups*]]%
Fri Apr 14 14:06:27 2017 [notice] node002: Service cups was started
Fri Apr 14 14:06:27 2017 [notice] node005: Service cups was started
Fri Apr 14 14:06:27 2017 [notice] node004: Service cups was started
Fri Apr 14 14:06:27 2017 [notice] node003: Service cups was started
[bright73->category*[default*]->services*[cups*]]% status
[bright80->category*[default*]->services*[cups*]]% status
node001.................. cups [STOPPED ]
node002.................. cups [  UP ]
node003.................. cups [  UP ]
node004.................. cups [  UP ]
node005.................. cups [  UP ]

Managing The Service From cmgui
Using cmgui, a service can be managed from a Services tab. The tab is accessible from a “Head Nodes” item, from a “Node Categories” item, or from a Nodes item.

Figure 3.17 shows the Services tab accessed from the default software image, which is an item within the “Node Categories” folder.

![CMGUI Services Tab](image)

Figure 3.17: A Services Tab In cmgui

The service `<service name> start|stop|restart...` command options start, stop, restart, and so on, are displayed as buttons in the Services tab.

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

Figure 3.18 shows CUPS being set up from an Add dialog in the Services tab, accessed by clicking on the Add button in the Services tab of figure 3.17.
Managing And Configuring Services

3.11 Managing And Configuring Services

3.11.1 Managing And Configuring Services

Managing And Configuring Services

For a service in the Services tab, clicking on the Status button in figure 3.17 displays a grid of the state of services on a running node as either Up or Down (figure 3.19). If the node is not running, it shows blank service entries.

Monitoring A Service With cmsh And cmgui

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 cmgui) to clear a FAILING state of a service, reset the attempted auto-starts count to zero, and attempt a restart of the service.

The monitoring system sets the ManagedServicesOk health check (Appendix G.2.1) to a state of FAIL if any of the services it monitors is in the FAILING state. In cmsh, the statuses of the services are listed by running the latesthealthdata command (section 10.8.3) from device mode. In cmgui the failed services themselves are listed from the Overview tab, and the statuses of the services are listed in the Services tab.

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 Bright 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 Bright 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 Bright Cluster Manager control while autostart is set to on stops the service on the nodes:

```
[bright73->category[default]->services]% add cups
[bright73->category*[default*]->services*[cups*]]% set monitored on
[bright73->category*[default*]->services*[cups*]]% set autostart on
```

© Bright Computing, Inc.
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.

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 cmgui: Selecting the Racks resource in cmgui opens up the Overview tabbed pane (figure 3.20). Racks can then be added, removed, or edited from that pane.

Within the Overview tabbed pane:

- a new rack item can be added with the Add button.
- an existing rack item can be edited by selecting it and clicking on the Open button, or by double clicking on the item itself in the tabbed pane.

An existing rack item can also be edited by simply selecting its name in the resource tree pane. This brings up its Settings tabbed pane (figure 3.21).

Racks overview in cmsh: The rack mode in cmsh allows racks defined in the cluster manager to be listed.
3.12 Managing And Configuring A Rack

Rack Configuration Settings

**Rack configuration settings in cmgui:** The Settings tabbed pane for editing a rack item selected from the Racks resource is shown in figure 3.21.

![Rack Configuration Settings Using cmgui](image)

**Figure 3.21: Rack Configuration Settings Using cmgui**

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. In the rack view visualization (section 3.12.2), the racks are displayed in one row, with nodes ordered so that the lowest x-value is the leftmost node for nodes with the same y-value. 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. The Notes tabbed pane 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 displays the numbering layout accordingly for all racks in Rackview (section 3.12.2), 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**

© Bright Computing, Inc.
The configuration settings for a particular rack obviously match with the parameters associated with and discussed in figure 3.21. The only slightly unobvious match is the Boolean parameter inverted in cmsh, which simply corresponds directly to “Bottom of rack is position 1” in cmgui.

Setting the values can be done as in this example:

Example

```
[bright73->rack]% use rack2
[bright73->rack[rack2]]% set room "skonk works"
[bright73->rack[rack2%]]% set x-coordinate 2
[bright73->rack[rack2%]]% set y-coordinate 0
[bright73->rack[rack2%]]% set inverted no
[bright73->rack[rack2%]]% commit
```

3.12.2 Rack View

The Rackview tab is available within cmgui after selecting the cluster resource item (figure 3.22).

![Rack View Using cmgui](image)

**Figure 3.22: Rack View Using cmgui**

**Using Rack View**

The Rackview pane of cmgui is a visualization of the layout of the racks, and also a visualization of the layout of devices such as nodes, switches, and chassis within a rack.

Any device can be visually located in the rack that it has been assigned to (section 3.12.3) by clicking on the “Locate in rack” button in the Tasks tabbed pane for that device. Clicking the button opens up the Rackview pane, with the device highlighted.

Some of the Racks configuration settings of figure 3.21 can also be visualized within rack view.

Visual metrics, in the form of bars that are color-calibrated with metric values, can be assigned via the Setup button from within the Rackview pane. The details for this are given on page 108.

In the Rackview pane, the color bar associated with a metric value is displayed within the device (a node or a switch), which is in turn within the rack. This can be useful in getting an idea of the spread of values across a cluster at a glance. In some cases patterns can become obvious when related to the physical positioning of devices in the rack, and related to the physical positioning of the racks.

For example, if ventilation is not working well in one part of the room, then metrics for the rack devices in that part may show a higher temperature compared to devices that are better ventilated. Mapped out as colors in the Rackview pane, the discrepancy would be noticed at a glance.
Rack View Configuration

In rack view:

- The rack name is displayed at the top of the rack.
- An item such as a node can be assigned a particular position within a rack space (section 3.12.3).
- A cloud region, for example, is given a rack space, and takes the region name as its “rack” name. For example `us-east-1`.
- A cloud region can be visualized as a rack for cloud nodes. That is, within the Rackview pane, cloud nodes are assigned to their cloud region in the same way as nodes are assigned to racks.
- The remaining items that are not assigned to a rack are kept in a space labeled “Unassigned”.
- Tooltip text above a node shows useful information about that particular node, and any rack-view-associated cluster metrics configured using the “Rack Setup” window (figure 3.24, described on page 108).
- At the bottom of the Rackview tabbed pane, the Size: field button allows a drop-down selection of rack sizes to set four kinds of rack views. These options are Large, Normal, Small, and Tiny.
  - Large shows the rack names, the node names, the rows of the racks, visual metrics (colored bars) next to the nodes, and a color calibration legend for the visual metrics along the bottom.
  - Normal shows the same as Large, except that racks displayed are scaled down to half the Large size. (figure 3.23).
  
  ![Figure 3.23: Rack View Normal Size](image)

  - Small shows the same as Normal, except that the node names are dropped, and the racks are scaled down to half the Normal size.
  - Tiny shows the same as Small, except that the racks are scaled down to half the Small size. This is really usually just to discern color patterns—the text associated with the racks is unreadable even with cmgui running in full-screen mode on a 22” monitor.
- The Refresh button refreshes the rack view.
- The Setup button opens up the Rack Setup window (figure 3.24).
The Rack Setup window allows the racks to be configured with metrics. Rack view then displays the metrics in the Large, Normal, Small, and Tiny rack views according to an automatically calibrated color scale.

In the Rack Setup window:

- The metrics can be configured by drag-and-drop.
- The Apply button applies a change.
- The Ok button applies any changes, and closes the “Rack Setup” screen.
- The Refresh Rate selection sets the time interval between metric measurement retrieval.
- The Show empty boxes if no data displays empty bars at the device if there are no data values, instead of the color that corresponds to a zero value.
- A check-mark in the Live sampling checkbox instructs that the selected metrics be measured and displayed at the refresh interval. Scripts that take longer than about a second to run can be used, but are best avoided because the display is not very “live” in that case.

• Select devices allows devices to be selected in groups according to a menu. Menu choices are
  - none
  - hostname
  - MAC
  - tag
  - category

© Bright Computing, Inc.
Managing And Configuring A Rack

3.12 Managing And Configuring A Rack

The selected menu items can then be filtered with regexes applied with Boolean AND or OR. The Job filter button allows selective filtering by clicking on jobs from a window.

The resultant filtered items are devices such as nodes or switches, and these are then highlighted in the rack view display.

- The color scale is auto-calibrated by default. It can be user-calibrated via a dialog box, by clicking on the existing minimum (blue end) or maximum (red end) values.

3.12.3 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. All devices are thus originally shown as part of the “Unassigned” grouping in the Rackview tabbed pane.

Devices can be assigned to a particular rack and to a particular position within the rack as follows:

Assigning Devices To A Rack Using cmgui

Using cmgui, the assignments of a device such as a node to a rack can be done from the Settings tabbed pane of the device, within the Rack section (figure 3.25):

![Figure 3.25: Rack Assignment Using cmgui](image)

Assigning Devices To A Rack Using cmsh

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

```
[bright73->device]% foreach -n node001..node003 (set deviceheight 1; set
deviceposition 2; set rack rack2)
[bright73->device] % commit
Successfully committed 3 Devices
[bright73->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 cmgui, 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 3.12.4), 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.

### 3.12.4 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 cmgui:** General chassis configuration in cmgui is done from the **Chassis** resource. Within the *Chassis* resource is the **Chassis Members** tabbed pane, which allows assignment of nodes to a chassis, and which is described in detail soon (page 112).

If the **Chassis** resource is selected from the resource tree, then a new chassis item can be added, or an existing chassis item can be opened.

The chassis item can then be configured from the available tabs, some of which are:

- The **Tasks** tab (figure 3.26).

![Figure 3.26: Chassis Tasks Tab](image)

The **Tasks** tab for a chassis item allows:

- Individual member nodes of the chassis to be powered on and off, or reset.
- All the member nodes of the chassis to be powered on and off, or reset.

Miscellaneous tasks in the **Tasks** tab are:

- The ability to start a telnet session, if possible, with the chassis operating system.
- The ability to start an SSH session, if possible, with the chassis operating system.
- The ability to locate the chassis in the visual representation of rack view.

© Bright Computing, Inc.
- The ability to open or close the chassis state. States are described in section 5.5.

- The **Settings** tab (figure 3.27).

![Figure 3.27: Chassis Settings Tab](image)

Amongst other options, the **Settings** tab for a chassis item allows selection of:

- A rack from the set of defined racks.
- A position within the rack (typically between 1 and 42 units).
- The height of the chassis (typically between 6 and 10 units). Because the typical numbering of a rack is from position 1 at the top to position 42 at the bottom, a chassis of height 6 at position 1 will take up the space from position 1 to 6. Effectively the base of the chassis is then at position 6, and the space from 1 to 6 can only be used by hardware that is to go inside the chassis.

The settings become active when the **Save** button is clicked.

- The **Members** tab (figure 3.28).
112 Configuring The Cluster

Figure 3.28: Chassis Members Tab

The Members tab for a chassis item allows the following actions:

- Nodes can be selected, then added to or removed from the chassis.
- Multiple nodes can be selected at a time for addition or removal.
- The selection can be done with standard mouse and keyboard operations, or using the filter tool.
- The filter tool allows selection based on some useful node parameters such as the hostname, the MAC address, the category. Several rules can made to operate together.

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

[bright73->device[chassis1]]% set
containerindex ip powerdistributionunits
custompingscript layout rack
custompingscriptargument mac revision
custompowerscript members slots
custompowerscriptargument model tag
defaultgateway network userdefined1
deviceheight notes userdefined2
deviceposition partition username
ethernetswitch password
hostname powercontrol
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

[bright73->device[chassis1]]% set rack rack2
[bright73->device*[chassis1*]]% set deviceposition 1; set deviceheight 6
[bright73->device*[chassis1*]]% commit

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

Example

[bright73->device[chassis1]]% append members bright73 node001..node005
[bright73->device*[chassis1*]]% commit

Advanced Chassis Configuration And Node Assignment

This section can be ignored in a first reading. Looking at the example of section 3.12.5 is recommended before reading about these features.

Advanced chassis configuration features are slots, containerindex, and layout. These can be useful for some less common requirements.

Slots In cmsh, slots in the chassis can be assigned an arbitrary value. A chassis may physically have a series of slots, with a node per slot. The labels used can be in the node naming order, in which case setting labels of 1 to the first node, 2 to the second node, and so on, is probably a good start. If the numbering is not right, or if there is no numbering, then administrators can set their own arbitrary values:

Example

[bright73->device[chassis1]]% set slots bright73 "leftmost top"
[bright73->device*[chassis1*]]% set slots node001 "leftmost bottom"
[bright73->device*[chassis1*]]% set slots node002 "red cable tie top"
[bright73->device*[chassis1*]]% set slots node003 "red cable tie bottom"

The values set are saved with the commit command. Looking up the slot values via cmsh is possible, but a more convenient way to use such labels is via the environment variable $CMD_CHASSIS_SLOT in scripts.

Container index A container is an object such as a node or a chassis. The container index is the index of the position of the container in rack view. The container index can be used to reposition containers within rack view, so that they align according to preference.

By default, nodes take up the first empty space in the container displayed by the rack view, from left to right, then on the next line, like the sequence followed when reading words in lines on a page. The position of each node in this sequence is given by the value of container index, which indicates the physical position in the sequence for a non-zero value. For a zero value the node simply takes up its default position (the first empty space in the container). So depending on the physical positioning, the nodes that are displaying incorrectly by default can be moved to a corrected location by setting their container index value. For example, the following layout is displayed for a chassis with a position specified as 5, a height specified as 10U, with 16 nodes as members, and with the default of containerindex=0 for all members (figure 3.29):
The chassis text label is represented rather like a node in the chassis, and takes up position 0 in the chassis by default. The next node in the sequence then takes up the first free position by default, which is position 1, and which would correspond to containerindex=1 if the position were set explicitly. Similarly, node001 takes up position 2 by default, which would correspond to containerindex=2 if set explicitly, node002 takes up position 3 (containerindex=3), and so on.

The last node can be shifted one position further as follows:

```
[bright73->device]% set node016 containerindex 18; commit
Successfully committed 1 Devices
[bright73->device]%
```

The display is then as shown by figure 3.30:
Moving `node001` to the place vacated by `node016` can be done with the following command:

```
[bright73->device]% set node001 containerindex 17; commit
Successfully committed 1 Devices
[bright73->device]%
```

The display is then as shown in figure 3.31:
As can be seen, the remaining nodes all shift to take up the first empty space available in the container.

**Layout** The administrator can make the rack view display match the physical layout, by displaying nodes in a chassis in a custom vertical or horizontal layout, instead of in the default horizontal layout.

In this section, a way of getting to a particular custom vertical layout is first explained, via successive approximations. After that, an example of a custom horizontal layout is described. Finally, how to remove the chassis text entirely so that only the nodes are displayed is described.

- **Custom vertical layouts** can make sense for blades in a chassis. Thus, for the chassis in figure 3.29, with 16 nodes as members, a chassis with 8 blades across and 2 nodes vertically might be desired.

  - **First approximation.**
    A starting point to get to the desired layout can be specified by setting the `layout` field of the container (the chassis) to:
    
    |8, 2

    This can be read as:
    “vertical text in the node box, space for 8 nodes along the x-axis, space for 2 nodes along the y-axis”

    This makes room in the container for 8 nodes across by 2 nodes up, although this space is only filled to the extent that members exist for the chassis.

    The “|”, or vertical sign, makes the text associated with the member nodes orient vertically, or more accurately, rotates the text clockwise a quarter turn.

    Conversely, the “-”, or horizontal sign, keeps the text oriented the standard way.

    - In cmgui the layout field for the container can be set via the Chassis resource, then selecting the Settings tab, and scrolling down to the Rack layout field.

© Bright Computing, Inc.
In cmsh, the layout field for the container can be set with a session such as:

```
[root@bright73 ~]# cmsh
[bright73]# device use chassis1
[bright73->device[chassis1]]# set layout "|8,2"
[bright73->device*[chassis1*]]# commit
```

Figure 3.32 shows the resulting first approximation of the desired vertical layout:

Figure 3.32: Vertical Blades Display, Approximation 1

- **Second approximation.**

Since the chassis text takes up 1 nodesworth of space inside the chassis container in rack view, 1 unit of extra room should be specified either along the x- or y-axis. With this extra unit specified, for example as:

```
|9,2
```

an improved second approximation of the layout is reached (figure 3.33).
Third approximation.

Then, applying the principles of containerindex (page 113), a third adjustment of the layout can be carried out on the second row with the following short bash script:

```bash
[root@bright73 ~]# for i in {009..016}
do j=$((10#$i+2))
cmsh -c "device use node$i; set containerindex $j; commit"
done
```

The 10#$i bit in the bash script simply makes each zero-padded value that $i loops through get treated as a base 10 number instead of an octal number when assigned to $j. In other words, the script is like running (6 lines elided):

```bash
cmsh -c "device use node009; set containerindex 11; commit"
...  
cmsh -c "device use node016; set containerindex 18; commit"
```

On commit, this third approximation shifts each node from node009 onwards to a position further to the right. The layout of the third approximation is now a sufficiently human-friendly display for most system administrators, with the nodes aligned in pairs in the way they would typically be physically (figure 3.34):
Custom horizontal layouts can be made, instead of a default horizontal layout of the kind shown in figure 3.29. A session can be continued from that layout, or from the layout reached at the end of figure 3.34, as follows:

The container (which is the chassis) index values can first be cleared:

```bash
[bright73->device[chassis1]]% !
...
[root@bright73 ~]# for i in {001..016}
do
  cmsh -c "device use node$i; clear containerindex; commit"
done
[root@bright73 ~]# exit
```

The layout can then be set to: -3, 10:

```bash
[bright73->device[chassis1]]% set layout "-3,10"; commit
```

This can be read as:

“set layout to have horizontal text in the node box, space for 3 nodes along the x-axis, and 10 nodes along the y-axis, for this container”.

Finally, the value of containerindex can be offset by 3 for each node, in order to place the nodes in a position that starts in a fresh row, separate from the chassis text:

```bash
[bright73->device[chassis1]]% !
...
[root@bright73 ~]# for i in {001..016}
do
  j=$((10#$i+3))
  cmsh -c "device use node$i; set containerindex $j; commit"
done
...
exit
[bright73->device[chassis1]]%
```
The resulting layout is shown in figure 3.35.

![Figure 3.35: Horizontal Nodes Custom Display](image)

- **Removing the container text label and box from the layout entirely** is also possible. The container text label is the box marked `chassis1` in figures 3.29-3.35. Sane system administrators would prefer to remove it because it is unpleasant to use a bash shell to carry out arithmetic to rearrange node layouts to work around the space taken up by the box, as is done in some of the preceding explanations. Removing the container text label and its box from the chassis container allows the container space to be used only by the nodes assigned to it, so that bash arithmetic is not needed.

Removing the container text label and box can be done by setting the container index of the chassis container to be greater than the layout dimension. The layout dimension value for a layout field is: \( x \)-axis value \( \times \) \( y \)-axis value. For the examples discussed, this value is 16 during the first approximation (page 116), 18 in the second and third approximations (pages 117, 118), and 30 for the custom horizontal layout (page 120).

Thus, for all these examples, a value of \( 31 \) or greater for `containerindex` for the chassis removes the container label box and the text itself.

Continuing with the custom horizontal layout from before:

**Example**

```
[bright73->device[chassis1]]% set containerindex 31
```

The resulting layout is shown in figure 3.36:
3.12 Managing And Configuring A Rack

Figure 3.36: Container Label Text And Box Removed

The offset by 3 for the nodes is a legacy from the earlier custom horizontal layout (page 119). It can be cleared as follows:

Example

[bright73->device[chassis1]]% ..
[bright73->device]% foreach -n node001..node016 (clear containerindex)
[bright73->device*]% commit

3.12.5 An Example Of Assigning A Device To A Rack, And Of Assigning A Device To A Chassis

The following illustrative case explains how to set up a blade server in a cluster. This breaks down naturally into 2 steps: firstly configuring the chassis (the blade server device) for the rack, and secondly configuring the nodes (the blade devices) for the chassis.

1. **Assigning a chassis to a position within a rack:** A position within a rack of the cluster is first chosen for the chassis.

   To assign a chassis to a rack in the cluster manager, the rack must first exist according to the cluster manager. If the rack is not yet known to the cluster manager, it can be configured as described in section 3.12.

   Assuming racks called *rack1*, *rack2*, and *rack3* exist, it is decided to place the chassis in *rack2*. If the blade server is 10 units in height, with 20 nodes, then 10 units of room from the rack must be available. If positions 5 to 14 are free within *rack2*, then suitable positional settings are:

   - *rack2* for the rack value,
   - 5 for the position value,
   - 10 for the height value.

   These values can then be set as in the example of figure 3.27, and come into effect after saving them.

© Bright Computing, Inc.
2. **Assigning nodes to the chassis**: The nodes of the chassis can then be made members of the chassis, as is done in the example of figure 3.28, and saving the members’ settings.

   The rack, position within the rack, and height values of the chassis are inherited by nodes that are members of the chassis.

   After the nodes rack, position within the rack, and height values match those held by the chassis, they display correctly within the chassis space within rack view, as shown in figure 3.37.

3.13 Configuring A GPU Unit, And 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 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.

3.13.2 GPU Unit Configuration Example: The Dell PowerEdge C410x

   An example of a GPU unit is the Dell PowerEdge C410x, which comes in a 3U chassis size, has up to 16 Tesla M-series GPUs (in the form of cards in slots) inside it, and can allocate up to 4 GPUs per node.

   It can be configured to work with Bright Cluster Manager as follows:

   1. The GPUs in the unit are assigned to nodes using the direct web interface provided by the Baseboard Management Controller (BMC) of the C410x. This configuration is done outside of Bright
Cluster Manager. The assignment of the GPUs can be done only according to the fixed groupings permitted by the web interface.

For example, if the unit has 16 GPUs in total (1, 2,..., 16), and there are 4 nodes in the cluster (node001, node002,..., node004), then an appropriate GPU indices mapping to the node may be:

- node001 is assigned 1, 2, 15, 16
- node002 is assigned 3, 4, 13, 14
- node003 is assigned 5, 6, 11, 12
- node004 is assigned 7, 8, 9, 10

This mapping is decided by accessing the C410x BMC with a browser, and making choices within the “Port Map” resource (figure 3.38). 4 mappings are displayed (Mapping 1, 2, 3, 4), with columns displaying the choices possible for each mapping. In the available mapping choices, the iPass value indicates a port, which corresponds to a node. Thus iPass 1 is linked to node001, and its corresponding PCIE values (1, 2, 15, 16) are the GPU indices here. Each iPass port can have 0 (N/A), 2, or 4 PCIE values associated with it, so that the GPU unit can supply up to 4 GPUs to each node.

2. The GPUs can be assigned to the nodes in Bright Cluster Manager as follows:

First, the GPU unit holding the GPUs can be assigned to the GPU unit resource. This can be carried out by adding the hostname of the GPU unit, for example in `cmsh` with:

```bash
```

Figure 3.38: Assigning GPUs To Nodes With The C410x web interface

The GPU indices (PCIE values) (that is, the numbers 1, 2,..., 16) are used to identify the card in Bright Cluster Manager. The lowest GPU index associated with a particular node—for example, 5 (rather than 6) for node003—is used to make Bright Cluster Manager aware in the next step that an association exists for that node, as well as aware which the first GPU card associated with it is.
This drops the administrator into the gpuunit object, which has the hostname schwartz here. The equivalent addition in cmgui can be done via the GPU Units resource folder, and clicking on the Add button in the Overview tabbed pane.

Next, the BMC user name, password, IP address, and network values can be set to appropriate values:

**Example**

```
[bright73->device[schwartz]]% set bmcusername darkhelmet
[bright73->device[schwartz]]% set bmcpassword 12345
[bright73->device[schwartz]]% set ip 10.148.0.10
[bright73->device[schwartz]]% set network bmcnet
```

Here, it is assumed that a BMC network that the BMC network interface is connected to has already been created appropriately. If it is not already set up, then the network can be configured as described in section 3.7.

The value of owners must be set to the list of nodes that have access to GPUs.

Also, the reserved variable, userdefined1 must be set to a key=value pair, so that each key (the node in owner) has a value (the PCIE values), assigned to it:

**Example**

```
[bright73...]% set owners node001 node002 node003 node004
[bright73...]% set userdefined1 node001=1,2,15,16 node002=3,4,13,14 node003=5,6,11,12 node004=7,8,9,10
[bright73->device[schwartz]]% commit
```

As a convenience, if the indices are all separated by 1, as in the case of node004 here, only the lowest index need be set for that node. The line setting the userdefined1 value can thus equivalently be carried out with:

**Example**

```
[bright73...]% set userdefined1 node001=1,2,15,16 node002=3,4,13,14 node003=5,6,11,12 node004=7
```

Once these values are committed, Bright Cluster Manager is able to track GPU assignment consistently and automatically.

3. Finally, CUDA drivers should be installed on the relevant nodes so that they can make use of the GPUs. The details on how to do this are given in section 7.5 of the *Installation Manual*.

### 3.13.3 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 range to be specified for a range of GPUs:
3.13 Configuring A GPU Unit, And Configuring GPU Settings

Example

[root@bright73 ~]# cmsh
[bright73]% device use node001
[bright73->device[node001]]% gpusettings
[bright73->device[node001]->gpusettings]% add 1-3 ; commit

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 range to be set for the range of GPUs:

Example

[root@bright73 ~]# cmsh
[bright73]% category use default
[bright73->category[default]]% gpusettings
[bright73->category[default]->gpusettings]% list

GPU range (key) Power limit ECC mode Compute mode Clock speeds
-----------------------------------------------

[bright73->category[default]->gpusettings]% add 1-3 ; commit
[bright73->category[default]->gpusettings[1-3]]% show

Parameter Value
-----------------------------------------------

Clock speeds
Compute mode
ECC mode
...

As usual, GPU settings for a node override those for a category (section 2.1.3).

After a name has been set, the following GPU settings may be specified, if supported, from within the gpusettings submode:

- **clockspeeds**: The clock speed (frequency in MHz) choices are:
  - max#max: maximum for both memory and graphics
  - min#min: minimum for both memory and graphics
  - `<number for memory>#<number for graphics>`: other possible combinations for memory and graphics, as suggested by tab-completion

- **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 651), and corrected automatically.

Double bit errors are also detected, using the EccDBitGPU metric (page 650), but cannot be corrected.

The gpureset and gpuclearecc commands (page 126) are relevant for this setting.

- gpureset: After changing the eccmode setting, the GPU must be reset to make the new value active.
- gpuclearecc: This is used to clear ECC counts for the GPU.

- **name**: 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

- **operationmode**: The operation mode values for the GPU are:
  - All on: All of the GPU node is made available for calculations
  - Compute: Only the computing section is made available for calculations. Graphics section calculations are not possible.
  - Low Double precision: A mode that spares resources and power if little double precision use is expected

- **persistencemode**:
  - Enabled: If the mode is enabled, then the NVIDIA driver is kept loaded, even if nothing is actively using it. This reduces loading latency when running CUDA programs.

- **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.

- **powermode**:
  - Supported: Allows power management functions.

If no value is specified for a GPU setting, then the hardware default is used.

Within cmsh, the gpureset and gpuclearecc commands can be executed at a node or category level, and also executed for specified nodes. The cmsh help text for these commands (help gpureset and help gpuclearecc) gives details, including node range specifications.

- The gpureset command carries out a power reset. It is executed only if there are no processes using the GPU. Possible processes using the GPU can be CUDA applications, X, monitoring applications, or other calculations. The device is all done with the reset in about 2 seconds.

**Example**

[bright73->device] % gpureset -c default 1,3–6
This resets GPUs 1, 3, 4, 5, and 6 in the default category. The command can be used to reset GPUs without rebooting the node, or to reset some hung GPUs.

- The `gpuclearecc` command clears the ECC counts for the GPU. The ECC counts are of two types:
  - volatile: Resets when the driver unloads. Resets on power cycle.
  - aggregate: Persists when the driver unloads. Resets on power cycle.

By default, both types are cleared when the command is run, but each type can be cleared separately.

**Example**

```
[bright73->device]# gpuclearecc -c default 1,3-6 -a
```

This clears the aggregate ECC counts for the GPUs 1, 3, 4, 5, and 6 in the default category.

The `gpusettings` Submode In cmgui

In cmgui the GPU settings can be accessed by selecting a node from the **Nodes** resource, and then selecting the **GPU Settings** tabbed pane for that node (figure 3.39). Similarly, GPU settings can also be accessed within the **Category** resource, selecting a category item, and then selecting the **GPU Settings** tabbed pane.

Clicking on the **Status** button in the **GPU Settings** pane opens up a new window to show what properties and values are supported for the GPU.

### 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`
- `custompingscript` and `custompingscriptargument`
- `customremoteconsolescript` and `customremoteconsolescriptargument`

The environment variables of CMDaemon (section 2.6 of the **Developer Manual**) can be used in the scripts. Successful scripts, as is the norm, return 0 on exit.

#### 3.14.1 custompowerscript

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

The following example script:

Example

```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 a type of PING called TCP SYN ping. This results in the statuses that can be seen on running the list command of cmsh in device mode. An example output, formatted for convenience, is:

Example

```
[root@bright73]# cmsh -c "device; format hostname:15, status:15; list"
```

<table>
<thead>
<tr>
<th>hostname (key)</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>bright73</td>
<td>[ UP   ]</td>
</tr>
<tr>
<td>node001</td>
<td>[ UP   ]</td>
</tr>
<tr>
<td>node002</td>
<td>[ UP   ]</td>
</tr>
</tbody>
</table>

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 replace 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 to its KVM switch.

3.15 Cluster Configuration Without Execution By CMDaemon

3.15.1 Cluster Configuration: The Bigger Picture

The configurations carried out in this chapter so far are based almost entirely on configuring nodes, via a CMDaemon front end (cmsh or cmgui), 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 cmgui or cmsh.
2. **Replacing the node image:** The image on a node can be replaced by an entirely different one, so that the node can function in another way. This is covered in section 3.15.2. It can be claimed that since it is CMDaemon that selects the image, this technique should perhaps be classed as under item 1. However, since the execution of the change is really carried out by the changed image without CMDaemon running on the image, and because changing the entire image to implement a change of functionality is rather extreme, this technique can be given a special mention outside of CMDaemon execution.

3. **Using a FrozenFile directive:** Applied to a configuration file, this directive prevents CMDaemon from executing changes on that file for nodes. During updates, the frozen configuration may therefore need to be changed manually. The prevention of CMDaemon acting on that file prevents the standard cluster functionality that would run based on a fully CMDaemon-controlled cluster configuration. The FrozenFile directive is introduced in section 2.6.4, and covered in the configuration context in section 3.15.3.

4. **Using an initialize or finalize script:** This type of script is run during the initrd stage, much before CMDaemon on the regular node starts up. It is run if the functionality provided by the script is needed before CMDaemon starts up, or if the functionality that is needed cannot be made available later on when CMDaemon is started on the regular nodes. CMDaemon does not execute the functionality of the script itself, but the script is accessed and set on the initrd via a CMDaemon front end (Appendix E.2), and executed during the initrd stage. It is often convenient to carry out minor changes to configuration files inside a specific image in this way, as shown by the example in Appendix E.5. The initialize and finalize scripts are introduced in section 3.15.4.

5. **A shared directory:** Nodes can be configured to access and execute a particular software stored on a shared directory of the cluster. CMDaemon does not execute the functionality of the software itself, but is able to mount and share directories, as covered in section 3.10.

Finally, outside the stricter scope of cluster configuration adjustment, but nonetheless a broader way to modify how a cluster functions, and therefore mentioned here for more completeness, is:

6. **Software management:** the installation, maintenance, and removal of software packages. Standard post-installation software management based on repositories is covered in sections 9.2–9.6. Third-party software management from outside the repositories, for software that is part of Bright Cluster Manager is covered in Chapter 7 of the *Installation Manual*.

Third-party software that is not part of Bright Cluster Manager can be managed on the head node usually like to any other Linux system, under /opt or other recommended locations. If required by the other nodes, then the software should typically be set up by the administrator so that it can be accessed via a shared filesystem.

### 3.15.2 Making Nodes Function Differently By Image

#### Making All Nodes Function Differently By Image

To change the name of the image used for an entire cluster, for example after cloning the image and modifying it (section 3.15.2), the following methods can be used:

- in cmgui from within the Cluster resource, in the Settings tab
- or in cmsh from within the base object of partition mode

A system administrator more commonly sets the software image on a per-category or per-node basis (section 3.15.2).
Making Some Nodes Function Differently By Image

For minor changes, adjustments can often be made to node settings via initialize and finalize scripts so that nodes or node categories function differently (section 3.15.4).

For major changes on a category of nodes, it is usually more appropriate to have nodes function differently from each other by simply carrying out image changes per node category with CMDaemon. Carrying out image changes per node is also possible. As usual, node settings override category settings.

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@bright73 ~]# cmsh
   [bright73]% softwareimage
   [bright73->softwareimage]% clone default-image imagetwo
   images/default-image -> /cm/images/imagetwo
   [bright73->softwareimage*][imagetwo*]]% commit
   [bright73->softwareimage][imagetwo]%
   ```

2. After cloning, the settings can be modified in the new object. For example, if the kernel needs to be changed to suit nodes with different hardware, kernel modules settings are changed (section 5.3.2) and committed. This creates a new image with a new ramdisk.

   Other ways of modifying and committing the image for the nodes are also possible, as discussed in sections 9.2–9.6 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:

   ```
   [bright73->softwareimage*][imagetwo*]% category
   [bright73->category]% clone default categorytwo
   [bright73->category*][categorytwo*]% set softwareimage imagetwo
   [bright73->category*][categorytwo*]% commit
   [bright73->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):

   ```
   [bright73->category][categorytwo]% device
   [bright73->device]% use node099
   [bright73->device][node099]% set category categorytwo
   [bright73->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

   ```
   [bright73->device]% foreach -n node100..node200 (set category categorytwo)
   [bright73->device*]% commit
   ```
5. Rebooting restarts the nodes that are assigned to the new category with the new image.

Similarly, using cmgui, a default image can be cloned with the “Clone” button in the “Software Images” resource operating on an existing image. The new image is modified from the “Software Images” tab (section 9.3.4) or using the other image modifying methods in this chapter, to make nodes function in the way required, and any appropriate node or node category is assigned this image.

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

Configuration changes carried out by cmgui or cmsh often generate, restore, or modify configuration files (Appendix A).

However, sometimes an administrator may need to make a direct change (without using cmgui 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.

### 3.15.4 Adding Functionality To Nodes Via An initialize Or finalize Script

CMDaemon can normally be used to allocate different images per node or node category as explained in section 3.15.2. However, some configuration files do not survive a reboot (Appendix A), sometimes hardware issues can prevent a consistent end configuration, and sometimes drivers need to be initialized before provisioning of an image can happen. In such cases, an initialize or finalize script (sections 5.4.5, 5.4.11, and Appendix E.5) can be used to initialize or configure nodes or node categories.

These scripts are also useful because they can be used to implement minor changes across nodes:

**Example**

Supposing that some nodes with a particular network interface have a problem auto-negotiating their network speed, and default to 100Mbps instead of the maximum speed of 1000Mbps. Such nodes can be set to ignore auto-negotiation and be forced to use the 1000Mbps speed by using the ETHTOOL_OPTS configuration parameter in their network interface configuration file: /etc/sysconfig/network-scripts/ifcfg-eth0 (or /etc/sysconfig/network/ifcfg-eth0 in SUSE).

The ETHTOOL_OPTS parameter takes the options to the “ethtool -s <device>” command as options. The value of <device> (for example eth0) is specified by the filename that is used by the configuration file itself (for example /etc/sysconfig/network-scripts/ifcfg-eth0). The ethtool package is installed by default with Bright 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:

```bash
#!/bin/bash

## node010..node014 get forced to 1000 duplex full

```

© Bright Computing, Inc.
if $[\text{if} \ \&\& \ $\text{CMD\_HOSTNAME} = \text{node01}\{0-4\}] \text{] then}

\text{echo} '{ETHTOOL\_OPTS="speed 1000 duplex full"} >>/\text{localdisk}/\text{etc}/\text{sysconfig/\text{network-scripts/i}fcfg-\text{eth0}}

\text{fi}

### 3.15.5 Examples Of Configuring Nodes With Or Without CMDaemon

A node or node category can often have its software configured in CMDaemon via \text{cmgui} or \text{cmsh}:

\textbf{Example}

**Configuring a software for nodes using \text{cmgui} or \text{cmsh}:** If the software under consideration is CUPS, then a node or node category can manage it from the Services tab with \text{cmgui} or \text{cmsh} as outlined in section 3.11.2.

A counterexample to this is:

\textbf{Example}

**Configuring a software for nodes without using \text{cmgui} or \text{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 \text{cmgui} or \text{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 Bright 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 \text{cmgui} Or \text{cmsh}: Category Settings

When configuring nodes using \text{cmgui} or \text{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 \text{cmgui} or \text{cmsh} acting on CMDaemon.

\textbf{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.

\textbf{Example}

**Setting many nodes to a category:** If there are many nodes that are to be set to run CUPS, then a separate, new category can be created (cloning it from the existing one is easiest) and those many nodes are moved into that category, while the image is kept unchanged. The CUPS service setting is then set at category level to the appropriate value for the new category.

In contrast to these two examples, the software image method used in section 3.15.2 to implement a functionality such as CUPS would load up CUPS as configured in an image, and would not handle

\footnote{except to link nodes to their appropriate image via the associated category}
it via CMDaemon. So, in section 3.15.2, software images prepared by the administrator are set for a node category. Since, by design, images are only selected for a category, a node cannot override the image used by the category other than by creating a new category, and using it with the new image. The administrative overhead of this can be inconvenient.

Administrators would therefore normally prefer letting CMDaemon track software functionality across nodes as in the last two examples, rather than having to deal with tracking software images manually. Indeed, the roles assignment option (section 2.1.5) is just a special pre-configured functionality toggle that allows CMDaemon to set categories or regular nodes to provide certain functions, typically by enabling services.
Aspects of power management in 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 13)
- 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 Bright 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)
- Cisco UCS-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 Settings tab of each device object plugged into the PDU is then used to configure the PDU ports that control the device. Figure 4.1 shows the Settings tab for a head node.

Each device plugged into the PDU can have PDU ports added and removed with the + and - buttons in their Settings tab. For the APC brand of PDUs, the “Power controlled by” property in the Settings tab 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 Bright Cluster Manager. Administrators wishing to use Bright Cluster Manager to record that a dumb PDU is assigned to a node can deal with it as follows:

- in cmgui the Notes tab, or the “User defined 1”/“User defined 2” options in the...
Settings tab for that node can be used.

- 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 cmgui, the Overview tab of a PDU (figure 4.2) provides an overview of the state of PDU ports and devices that have been associated with each port.

![PDU Overview](image)

Figure 4.2: PDU Overview

The power status of a node can be seen by selecting the node from the Nodes resource, then selecting the Overview tabbed pane. The first tile of the Overview tab displays the power state and assignment of any PDUs for the node.

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

  **Example**

  ```bash
  [bright73]\% device use node001
  [bright73->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**

  ```bash
  [bright73->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/UCS interfaces.

To carry out IPMI-based power control operations, the Power controlled by 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 controlled by property is not apc.

**Example**

Configuring power parameters settings for all the nodes using cmsh, with IPMI interfaces that are called ipmi0:

```
[mycluster]# device
[...device]# foreach allphysicalnodes (set powercontrol ipmi0; commit)
```

**Example**

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

```
[mycluster]# device use node001
[...device[node001]]# set powerdistributionunits apc01:6 apc01:7 apc01:8
[...device[node001]]# get powerdistributionunits
apc01:6 apc01:7 apc01:8
[...device[node001]]# removefrom powerdistributionunits apc01:7
[...device[node001]]# get powerdistributionunits
apc01:6 apc01:8
[...device[node001]]# set powercontrol apc
[...device[node001]]# get powercontrol
apc
[...device[node001]]# commit
```

### 4.1.3 Combining PDU- and IPMI-Based Power Control

By default when nodes are configured for IPMI Based Power Control, any configured PDU ports are ignored. However, it is sometimes useful to change this behavior.
For example, in the CMDaemon configuration file directives in /cm/local/apps/cmd/etc/cmd.conf (introduced in section 2.6.2 and listed in Appendix C), the default value of PowerOffPDUOutlet is false. It can be set to true on the head node, and CMDaemon restarted to activate it.

With PowerOffPDUOutlet set to true it means that CMDaemon, after receiving an IPMI-based power off instruction for a node, and after powering off that node, also subsequently powers off the PDU port. Powering off the PDU port shuts down the BMC, which saves some additional power—typically a few watts per node. When multiple nodes share the same PDU port, the PDU port only powers off when all nodes served by that particular PDU port are powered off.

When a node has to be started up again the power is restored to the node. It is important that the node BIOS is configured to automatically power on the node when power is restored.

### 4.1.4 Custom Power Control

For a device which cannot be controlled through any of the standard existing power control options, it is possible to set a custom power management script. This is then invoked by the cluster management daemon on the head node whenever a power operation for the device is done.

Power operations are described further in section 4.2.

**Using custompowerscript**

To set a custom power management script for a device, the powercontrol attribute is set by the administrator to custom using either cmgui 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:

```bash
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 cmgui. 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
```

© Bright Computing, Inc.
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 2.6 of the Developer Manual). This is explained next.

Using Environment Variables With `custompowerscript`

Simplification of the steps needed for custom scripts in CMDaemon is often possible because there are values in the CMDaemon environment already available to the script. A line such as:

```
env > /tmp/env
```

added to the start of a custom script dumps the names and values of the environment variables to `/tmp/env` for viewing.

One of the names is `$CMD_MAC`, and it holds the MAC address string of the node being considered.

So, it is not necessary to retrieve a MAC value for `custompowerscriptargument` with a bash script as shown in the previous section, and then pass the argument via `$3` such as done in the command

```
wakeonlan $3
```

Instead, `custompowerscript` can simply call

```
"wakeonlan $CMD_MAC"
```

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 3.3.5 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". Bright 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
   ```
4.2 Power Operations

Power operations may be carried out on devices from either cmgui 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

4.2.1 Power Operations With cmgui

In cmgui, buttons for executing On/Off/Reset operations are to be found under the Tasks tab of a device. Figure 4.3 shows the Tasks tab for a head node.

![Figure 4.3: Head Node Tasks](image)

The Overview tab of a device can be used to check its power status information (figure 4.4).

© Bright Computing, Inc.
The green LEDs indicate that all three PDU ports are turned on. Red LEDs would indicate power ports that have been turned off, while gray LEDs would indicate an unknown power status for the device.

Performing power operations on multiple devices at once is possible through the Tasks tabs of node categories and node groups. It is also possible to do power operations on selected node through the Nodes folder in the resource tree. The nodes can be selected using the Overview tab. The selected group can then be operated on by a task chosen from the Tasks tab.

When doing a power operation on multiple devices, CMDaemon ensures a 1 second delay occurs by default between successive devices, to avoid power surges on the infrastructure. The delay period can be modified from within the device mode of cmsh, by using the -d|--delay option of the power command.

The Overview tab of a PDU object (figure 4.5), allows power operations on PDU ports by the administrator directly.
All ports on a particular PDU can have their power state changed, or a specific PDU port can have its state changed.

### 4.2.2 Power Operations Through cmsh

All power operations in cmsh are done using the `power` command in device mode. Some examples of usage are now given:

- **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
  ...
  ```

- **Powering off all nodes in the default category with a 100ms delay between nodes** (some output elided):

  **Example**

  ```
  [mycluster]\% device power -c default -d 0.1 off
  apc01:1 ............. [ OFF ] node001
  apc01:2 ............. [ OFF ] node002
  ...
  apc23:8 ............. [ OFF ] node953
  ```

- **Retrieving power status information for a group of nodes:**
Example

[mycluster]# device power -g mygroup status
apc01:3 ............. [ ON ] node003
apc01:4 ............. [ OFF ] node004

The help text for the `power` command is:

Name:
`power` - Manipulate or retrieve power state of devices

Usage:
```
power [OPTIONS] status
power [OPTIONS] on
power [OPTIONS] off
power [OPTIONS] reset
power [OPTIONS] cancel
```

Options:
- `-n, --nodes <node>`
  List of nodes, e.g. node001..node015,node20..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

- `-l, --role <role>`
  Filter all nodes that have the given role

- `-p, --powerdistributionunitport <pdu:port>(list)`
  Perform power operation directly on power distribution units. Use port ‘*’ for all ports

- `-b, --background`
  Run in background, output will come as events

- `-d, --delay <seconds>`

© Bright Computing, Inc.
4.3 Monitoring Power

Wait \(<seconds>\) between executing two sequential power commands.
This option is ignored for the status command

\(-s, --status <states>\)
Only run power command on nodes in specified states, e.g. UP,
"CLOSED|DOWN", "INST.*"

\(-f, --force\)
Force power command on devices which have been closed

Examples:
power status Display power status for all devices or current device
power on -n node001 Power on node001

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 Bright 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 10 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 Bright Cluster Manager:

\- in cmgui, by viewing it under <hostname>\’s System Information tab
\- in cmsh, by using the sysinfo <hostname> command, in device mode

© Bright Computing, Inc.
The list can be displayed under $\texttt{cmsh}$ in a form such as:

**Example**

```
[bright73->device[bright73]]% 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 cmgui:**
  - if the `Node Categories` resource is chosen, and a particular category is selected, then a `Settings` tabbed pane 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` tabbed pane 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 Bright 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

```
[bright73->)% device use node001
[bright73->)device[node001]]% sysinfo
System Information

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Core 0</th>
<th>Core 1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Six-Core AMD</td>
<td>Six-Core AMD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>performance</td>
<td>performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(ondemand,</td>
<td>(ondemand,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>userspace)</td>
<td>userspace)</td>
<td></td>
</tr>
</tbody>
</table>

[bright73->)device[node001]]% category use default
[bright73->)category[default]]% set scalinggovernor ondemand,userspace,performance
[bright73->)category[default]]% device use node001
[bright73->)device[node001]]% sysinfo
System Information

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Core 0</th>
<th>Core 1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Six-Core AMD</td>
<td>Six-Core AMD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ondemand</td>
<td>ondemand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(userspace,</td>
<td>(userspace,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>performance)</td>
<td>performance)</td>
<td></td>
</tr>
</tbody>
</table>
```

© Bright Computing, Inc.
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 cmgui 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 Bright Cluster Manager. This is called a network boot, or sometimes a PXE boot. It is recommended as a BIOS setting for nodes. The head node runs a tftp server from within xinetd, which 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.

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
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**

```bash
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

---

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**

```bash
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
5.1 Before The Kernel Loads

PXE boot menu is displayed, so that the associated kernel image and options can conveniently be selected by user intervention.

The PXE DEFAULT And MENU DEFAULT Instructions

**DEFAULT:** If the PXE menu module is not used and if no MENU instructions are used, and if there is no user intervention, then setting the same text that follows a LABEL tag immediately after the DEFAULT instruction, results in the associated kernel image and its options being run by default after the timeout.

By default, as already explained, the PXE menu module is used. In particular it uses the setting: DEFAULT menu.c32 to enable the menu.

**MENU DEFAULT:** If the PXE menu module is used and if MENU instructions are used, and if there is no user intervention, then setting a MENU DEFAULT tag as a line within the multiline LABEL statement results in the kernel image and options associated with that LABEL statement being loaded by default after the timeout.

The CMDaemon PXE Label Setting For Specific Nodes
The MENU DEFAULT value by default applies to every node using the software image that the PXE menu configuration file under pxelinux.cfg/ is loaded from. To override its application on a per-node basis, the value of PXE Label can be set for each node.

- Some simple examples of overriding the default MENU DEFAULT value are as follows:

  For example, using cmsh:

  **Example**

  ```
  [root@bright73 ~]# cmsh
  [bright73]% device use node001
  [bright73->device[node001]]% set pxelabel MEMTEST ; commit
  ```

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

  **Example**

  ```
  [root@bright73 ~]# cmsh
  [bright73]% device
  [bright73->device]% foreach -c default (set pxelabel MEMTEST)
  [bright73->device*]% commit
  ```

  The value of pxelabel can be cleared with:

  **Example**

  ```
  [root@bright73 ~]# cmsh -c "device; foreach -c default (clear pxelabel); commit"
  ```

  In cmgui, the PXE label can be set from the Settings tab 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 cmgui or cmsh to localdrive. This will then set the node to boot from the first local drive and not the node-installer. This is a setting
that is discouraged since it usually makes node management harder, but it can be used by administrators who do not wish to answer any prompt during node boot, and also want the node drives to have no risk of being overwritten by the actions of the node-installer, and also want the system to be up and running quite fully, even if not necessarily provisioned with the latest image from the head node.

Here, the overwriting-avoidance method relies on the nodes being associated with a configuration file under `pxelinux.cfg` at the time that the `localdrive` setting is done. However, nodes that are unidentified, or are identified later on, will have their `MENU DEFAULT` value still set to a default `pxelabel` value set in the files under `/tftpboot/pxelinux.cfg/`, which is the value `linux` by default, and which is associated with a code block in that file with the label `LABEL linux`. To make such (as yet) unidentified nodes boot to a `localdrive` setting instead, requires modifying the files under `/tftpboot/pxelinux.cfg/`, so that the `MENU DEFAULT` line is associated with the code block of `LABEL localdrive` rather than the code block of `LABEL linux`.

There are two methods other than using the preceding `pxelabel` method to deal with the risk of overwriting. Unlike the `pxelabel` method however, these methods can interrupt node booting, so that the node does not progress to being fully up until the administrator takes further action:

1. If it is acceptable that the administrator manually enters a confirmation as part of the boot process when a possible overwrite risk is found, then the `datanode` method (section 5.4.4) can be used.
2. If it is acceptable that the boot process halts on detecting a possible overwrite risk, then the XML assertions method (Appendix D.11) is recommended.

### Changing The Install Mode Or Default Image Offered To Nodes

The selections offered by the PXE menu are pre-configured by default so that the `AUTO` menu option by default loads a kernel, runs the `AUTO` install mode, and eventually the `default-image` software image is provisioned.

Normally administrators should not be changing the install mode, kernel, or kernel options in the PXE menu configuration files under `pxelinux.cfg/`.

More on changing the install mode is given in section 5.4.4. More on changing software images, image package management, kernels, and kernel options, is to be found in Chapter 9.
5.1 Before The Kernel Loads

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 Bright Cluster Manager aware that nodes are to be booted over InfiniBand. This can be done during the initial installation on the head node by marking the option “Allow booting over InfiniBand” (figure 3.11 of the Installation Manual). Alternatively, if the cluster is already installed, then node booting (section 3.2.3, page 66) can be set from cmsh or cmgui 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 cmgui’s “Settings” tab from the “Networks” resource (figure 3.5): The network item selected must be the InfiniBand network, “ibnet” by default, and the “Allow node booting” option is then set and saved.

If the InfiniBand network does not yet exist, then it must be created (section 3.2.2). The recommended default values used are described in section 3.6.3. The MAC address of the interface in CMDaemon defaults to using the GUID of the interface.

The administrator should also be aware that the interface from which a node boots, (conveniently labeled BOOTIF), must not be an interface that is already configured for that node in CMDaemon. For example, if BOOTIF is the device ib0, then ib0 must not already be configured in CMDaemon. Either BOOTIF or the ib0 configuration should be changed so that node installation can succeed.

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.

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 Booting From The Drive

Besides network boot, a node can also be configured to start booting and get to the stage of loading up its kernel entirely from its drive, just like a normal standalone machine, by setting PXE LABEL to localdrive (page 149).

5.1.5 The Boot Role

The action of providing a PXE boot image via DHCP and TFTP is known as providing node booting.
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 PXE boot attempts of a large number of regular nodes from the head node, it turns out that random different regular nodes are unable to boot, apparently 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 5.1.5), 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:
5.2 Provisioning Nodes

<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>maxProvisioningNodes</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.

© Bright Computing, Inc.
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

```
[bright73]% category add misc
[bright73]->category[misc*]->roles% assign provisioning
[bright73...]->roles[provisioning*] % set allimages no
[bright73...]->roles[provisioning*] % set localimages default-image
[bright73...]->roles[provisioning*] % set maxprovisioningnodes 20
[bright73...]->roles[provisioning*] % show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Images</td>
<td>no</td>
</tr>
<tr>
<td>Include revisions of local images</td>
<td>yes</td>
</tr>
<tr>
<td>Local images</td>
<td>default-image</td>
</tr>
<tr>
<td>Name</td>
<td>provisioning</td>
</tr>
<tr>
<td>Nodegroups</td>
<td></td>
</tr>
<tr>
<td>Provisioning associations</td>
<td>&lt;0 internally used&gt;</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Shared images</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>ProvisioningRole</td>
</tr>
<tr>
<td>maxProvisioningNodes</td>
<td>20</td>
</tr>
</tbody>
</table>

```
[bright73]->category[misc*]->roles[provisioning*] % commit
[bright73]->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

```
[bright73]% device use node001
[bright73]->device[node001] % roles
[bright73]->device[node001]->roles% assign provisioning
[bright73]->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, Bright Cluster Manager runs the updateprovisioners command described in section 5.2.4 automatically, so that regular images are propagated to the provisioners. The propagation can be done by provisioners themselves if they have up-to-date images. CMDaemon tracks the provisioning nodes role changes, as well as which provisioning nodes have up-to-date images available, so that provisioning node configurations and regular node images propagate efficiently. Thus, for example, image update requests by provisioning nodes take priority over provisioning update requests from regular nodes.

### 5.2.3 Provisioning Nodes: Role Setup With cmgui

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

- A misc category can be added by clicking on the Add button in the Overview tabbed pane in the Node Categories resource (figure 5.3).
5.2 Provisioning Nodes

Clicking on the misc category in the resource tree on the left hand side (or alternatively, double-clicking on the misc category in the Overview tabbed pane of the Node Categories right hand side pane) opens the category up (figure 5.4).

Figure 5.4: cmgui: Configuring A provisioning Role

Selecting the Roles tab in this category displays roles that are part of the misc category. Ticking the checkbox of a role assigns the role to the category, and displays the settings that can be configured for this role. The Provisioning slots setting (maxProvisioningNodes in cmsh) decides how many images can be supplied simultaneously from the provisioning node, while the Software images settings (related to the images and allimages attributes of cmsh) decides what images the provisioning node supplies.

The Software image in the Roles tab should not be confused with the Software image selection possibility within the Settings tab, which is the image the provisioning node requests for itself.

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 maxProvisioningNodes (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 cmdgui (figure 5.5):

![Figure 5.5: cmgui: A Button To Update Provisioning Nodes](image)

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 (cmdgui 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, there is also a scheduled time for it to run to ensure that it runs at least once every 24 hours.

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**

```
[bright73]# softwareimage updateprovisioners
Provisioning nodes will be updated in the background.

Sun Dec 12 13:45:09 2010 bright73: Starting update of software image(s)
  provisioning node(s). (user initiated).
[bright73]# softwareimage updateprovisioners [bright73]#
Sun Dec 12 13:45:41 2010 bright73: Updating image default-image on prov
  visioning node node001.
[bright73]#
Sun Dec 12 13:46:00 2010 bright73: Updating image default-image on prov
  visioning node node001 completed.
Sun Dec 12 13:46:00 2010 bright73: Provisioning node node001 was updated
Sun Dec 12 13:46:00 2010 bright73: Finished updating software image(s) \n  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**

```
[bright73->softwareimage]# drain -n master
Nodes drained
[bright73->softwareimage]# provisioningstatus
Provisioning subsystem status
Pending request:     node001, node002
Provisioning node status:
  + bright73
    Slots:     1 / 10
    State:    draining
    Active nodes: node003
    Up to date images: default-image
[bright73->softwareimage]# provisioningstatus
Provisioning subsystem status
Pending request:     node001, node002
Provisioning node status:
  + bright73
    Slots:     0 / 10
    State:    drained
```

© Bright Computing, Inc.
Active nodes: none
Up to date images: default-image

To drain all nodes at once, the \texttt{--role} option can be used, with \texttt{provisioning} role as its value. All pending requests then remain in the queue, until the nodes are undrained again.

\textbf{Example}

\begin{verbatim}
[bright73->softwareimage]$ drain --role provisioning
... Time passes. Pending requests stay in the queue. Then admin undrains it...
[bright73->softwareimage]$ undrain --role provisioning
\end{verbatim}

\textbf{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.

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 this is the case an update is triggered. The update is disabled if the timeout is set to zero (\texttt{false}).

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

\textbf{Example}

\begin{verbatim}
[root@bright73 ]# cmsh
[bright73]$ partition use base
[bright73->partition[base]]% get provisioningnodeautoupdatetimeout
[bright73->partition[base]]% set provisioningnodeautoupdatetimeout 0
[bright73->partition*[base*]]% commit
\end{verbatim}

Within \texttt{cmgui} it is accessible via the \texttt{Settings} tab for the cluster resource (figure 3.1).

To prevent provisioning an image to the nodes, the \texttt{locked} state (section 5.4.7) can be used. A \texttt{locked} state defers the provisioning request, until the image is unlocked once more.

\section{The Kernel Image, Ramdisk And Kernel Modules}

A software image is a complete Linux filesystem that is to be installed on a non-head node. Chapter 9 describes images and their management in detail.

The head node holds the head copy of the software images. Whenever files in the head copy are changed using CMDaemon, the changes automatically propagate to all provisioning nodes via the \texttt{updateprovisioners} command (section 5.2.4).

\subsection{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 \texttt{/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.

© Bright Computing, Inc.
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 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.

Whenever a change is made via the kernelmodules submode to the kernel module selection of a software image, CMDaemon automatically runs the creatoramdisk command. The creatoramdisk 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 creatoramdisk command can also be run manually from within the softwareimage mode.

**Kernel Driver Modules With cmgui**

In cmgui the selection of kernel modules is carried out by selecting the Software Images resource, and then choosing the “Kernel Config” tabbed pane (figure 5.6).

![Figure 5.6: cmgui: Selecting Kernel Modules For Software Images](image)

The order of module loading can be rearranged by selecting a module and clicking on the arrow keys. Clicking on the “Recreate Initrd” button runs the creatoramdisk command manually.

**Manually Regenerating A Ramdisk**

Regenerating a ramdisk manually via cmsh or cmgui is useful if the kernel or modules have changed without using CMDaemon. For example, after running a YUM update which has modified the kernel or modules of the nodes (section 9.3). In such a case, the distribution would normally update the ramdisk on the machine, but this is not done for the extended ramdisk for nodes in Bright Cluster Manager. Not regenerating the Bright 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 9.3.4.
5.3.3 InfiniBand Provisioning

On clusters that have InfiniBand hardware, it is normally used for data transfer as a service after the nodes have fully booted up (section 3.6). It can also be used for PXE booting (section 5.1.3) and for node provisioning (described here), but these are not normally a requirement. This section (about InfiniBand node provisioning) may therefore safely be skipped in almost all cases when first configuring a cluster.

During node start-up on a setup for which InfiniBand networking has been enabled, the init process runs the rdma script. For SLES and distributions based on versions prior to Red Hat 6, 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. Enabling InfiniBand is normally set by configuring the InfiniBand network when installing the head node, during the Additional Network Configuration screen (figure 3.11 of the Installation Manual).

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 Bright Cluster Manager is done via Ethernet.

Provisioning at the faster InfiniBand speeds rather than Ethernet speeds is however a requirement for some clusters. To get the cluster to provision using InfiniBand requires both of the following two configuration changes to be carried out:

1. configuring InfiniBand drivers for the ramdisk image that the nodes first boot into, so that provisioning via InfiniBand is possible during this pre-init stage
2. defining the provisioning interface of nodes that are to be provisioned with InfiniBand. It is assumed that InfiniBand networking is already configured, as described in section 3.6.

The administrator should be aware that the interface from which a node boots, (conveniently labeled BOOTIF), must not be an interface that is already configured for that node in CMDaemon. For example, if BOOTIF is the device ib0, then ib0 must not already be configured in CMDaemon. Either BOOTIF or the ib0 configuration should be changed so that node installation can succeed.

How these two changes are carried out is described next:

InfiniBand Provisioning: Ramdisk Image Configuration

An easy way to see what modules must be added to the ramdisk for a particular HCA can be found by running rdma (or openibd), and seeing what modules do load up on a fully booted regular node.

One way to do this is to run the following lines as root:

```
[root@bright73 ~]# { service rdma stop; lsmod | cut -f1 -d" "; }>/tmp/a
[root@bright73 ~]# { 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
As suggested by the output, the modules `ib_ipoib`, `rdma_ucm` and so on are the modules loaded when `rdma` starts, and are therefore the modules that are needed for this particular HCA. Other HCAs may cause different modules to be loaded.

For a default Red Hat from version 7 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@bright73 ~]# cmsh
[bright73]% softwareimage use default-image
[bright73]->softwareimage[default-image]>% kernelmodules
[bright73...]->softwareimage[default-image]->kernelmodules]% add mlx4_ib
[bright73...]->softwareimage[default-image]->kernelmodules[mlx4_ib]% add ib_ipoib
[bright73...]->softwareimage[default-image]->kernelmodules[ib_ipoib]% add ib_umad
[bright73...]->softwareimage[default-image]->kernelmodules[ib_umad]% commit
[bright73]->softwareimage[default-image]->kernelmodules[ib_umad]]%
Tue May 24 03:45:35 2011 bright73: 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 3.15.2).

**InfiniBand Provisioning: Network Configuration**

It is assumed that the networking configuration for the final system for InfiniBand is configured following the general guidelines of section 3.6. If it is not, that should be checked first to see if all is well with the InfiniBand network.

The provisioning aspect is set by defining the provisioning interface. An example of how it may be set up for 150 nodes with a working InfiniBand interface `ib0` in `cmsh` is:

© Bright Computing, Inc.
Example

[root@bright73~]# cmsh
[bright73]% device
[bright73->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[bright73->device*]% 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 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 /cm/node-installer/scripts/node-installer.conf, and illustrated on page 597.

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 Bright 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).

The certificate is stored on the head node in /cm/node-installer/certificates/ by MAC address.

Null Cipher Certificates

By default, a null cipher is used on internal networks such as internalnet, to keep communications speedy. Using encryption on even these networks is sometimes a requirement in very unusual situations. In that case, setting the advanced configuration flag AllowNullCipherNetwork=0 in cmd.conf (Appendix C, page 611) forces encryption on after CMDaemon is restarted. By default, its value is 1.

Certificate Auto-signing

By default, certificate auto-signing means the cluster management daemon automatically issues a certificate to any node that requests a certificate.

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 can then be done by issuing the autosign off command from cert mode in cmsh.

Section 2.3 has more information on certificate management in general.

Example

Disabling certificate auto-sign mode:

[bright73]% cert autosign
on
[bright73]% cert autosign off
off
[bright73]% cert autosign
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.
2. The node is new, and an Ethernet switch port is detected. A node configuration associated with the port is not found. The node-installer then displays a dialog that allows the administrator to either retry Ethernet switch port detection (figure 5.10) or to drop into a sub-dialog to manually select a node configuration (figure 5.9). By default, port detection is retried after a timeout.
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.

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 ethernetswitch"
  ```
- 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>

Figure 5.13: Scenarios: Port Unset Dialog
In these scenarios, whenever the user manually selects a node configuration in the prompt dialog, an attempt to detect an Ethernet switch port is repeated. If a port mismatch still occurs, it is handled by the system as if the user has not made a selection.

**Summary Of Behavior During Hardware Changes**

The logic of the scenarios means that an unpreconfigured node always boots to a dialog loop requiring manual intervention during a first install (scenarios 2 and 3). For subsequent boots the behavior is:

- If the node MAC hardware has changed (scenarios 1, 2, 3):
  - if the node is new and the detected port has a configuration, the node automatically boots to that configuration (scenario 1).
  - else manual intervention is needed (scenarios 2, 3)

- If the node MAC hardware has not changed (scenarios 4, 5, 6, 7):
  - if there is no port mismatch, the node automatically boots to its last configuration (scenarios 4, 7).
  - else manual intervention is needed (scenarios 5, 6).

**The newnodes Command**

newnodes basic use: New nodes that have not been configured yet can be detected using the `newnodes` command from within the device mode of cmsh. A new node is detected when it reaches the node-installer stage after booting, and contacts the head node.

**Example**

```
[bright73->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 `ethernetswitch` property for the specified device can be set to the port and switch values.

**Example**

```
[bright73->device]# showport 00:0C:29:01:0F:F8
switch01:8
[bright73->device]# set node003 ethernetswitch switch01:8
[bright73->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.

© Bright Computing, Inc.
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:**
  
  - `-n|--nodes`
  - `-w|--write`
  - `-s|--save`

  Usually the most straightforward way to assign the nodes is to use the `-n` option, which accepts a list of nodes, together with a `-w` or `-s` option. The `-w` (**--write**) option sets the order of nodes to the corresponding order of listed MAC addresses, and is the same as setting an object in `cmsh`. The `-s` (**--save**) option is the same as setting and committing an object in `cmsh`, so `-s` implies a `-w` option is run at the same time.

  So, for example, if 8 new nodes are discovered by the node-installer on a cluster with no nodes so far, then:

  **Example**

  ```
  [bright73->device]% newnodes -w -n node001..node008
  ```

  assigns (but does not commit) the sequence `node001` to `node008` the new MAC address according to the sequence of MAC addresses displaying on the list.

- **The grouping options:**

  - `-g|--group`
  - `-c|--category`
  - `-h|--chassis`
  - `-r|--rack`

  The "`help newnodes`" command in **device mode** shows assignment options other than `-n` for a node range are possible. For example, the assignments can also be made for a group (`-g`), per category (`-c`), per chassis (`-h`), and per rack (`-r`).

- **The miscellaneous options:**

  - `-f|--force`
  - `-o|--offset`
By default, the `newnodes` command fails when it attempts to set a node name that is already taken. The `-f` (`--force`) option forces the new MAC address to be associated with the old node name. When used with an assignment grouping, (node range, group, category, chassis, or rack) all the nodes in the grouping lose their node-to-MAC assignments and get new assignments. The `-f` option should therefore be used with care.

The `-o` (`--offset`) option takes a number `<number>` and skips `<number>` nodes in the list of detected unknown nodes, before setting or saving values from the assignment grouping.

Examples of how to use the advanced options follow.

`newnodes` **advanced use—range assignment behavior example**: For example, supposing there is a cluster with nodes assigned all the way up to `node022`. That is, CMDaemon knows what node is assigned to what MAC address. For the discussion that follows, the three nodes `node020`, `node021`, `node022` can be imagined as being physically in a rack of their own. This is simply to help to visualize a layout in the discussion and tables that follow and has no other significance. An additional 3 new, that is unassigned, nodes are placed in the rack, and allowed to boot and get to the node-installer stage.

The `newnodes` command discovers the new MAC addresses of the new nodes when they reach their node-installer stage, as before (the switch port column is omitted in the following text for convenience):

```
Example
[bright73->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
[bright73->device]$ newnodes -s -n node023..node025
MAC First appeared Hostname
-------------------- ----------------------------- --------
00:0C:29:EF:40:2A Tue, 01 Nov 2011 11:42:31 CET node023
00:0C:29:95:D3:5B Tue, 01 Nov 2011 11:46:25 CET node024
00:0C:29:65:9A:3C Tue, 01 Nov 2011 11:47:13 CET node025
```

Once this is done, the node-installer is able to stop looping, and to go ahead and install the new nodes with an image.

The physical layout in the rack may then look as indicated by this:

```
before after
node020 node020
node021 node021
node022 node022
     node023 ...A
     node024 ...B
     node025 ...C
```

Here, `node023` is the node with the MAC address ending in `A`.

If instead of the previous `newnodes` command, an offset of 1 is used to skip assigning the first new node:

```
Example
```
Example

[bright73->device] % newnodes -s -o 1 node024..node025

then 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>unassigned</td>
<td>...A</td>
<td></td>
</tr>
<tr>
<td>node024</td>
<td>...B</td>
<td></td>
</tr>
<tr>
<td>node025</td>
<td>...C</td>
<td></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

[bright73->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 cmgui, assigning node names to a rack can be done from the Nodes resource and selecting the Settings tab. Within the tab, the Rack values can be set appropriately, and saved for each node (figure 5.14).
In cmsh, the assignment of multiple node names to a rack can conveniently be done with a `foreach` loop from within `device` mode:

**Example**

```
[bright73->device]% foreach -n node020..node029 (set rack rack02)
[bright73->device]% commit
[bright73->device]% foreach -n node030..node039 (set rack rack03)
[bright73->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**

```
[bright73->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**

```
[bright73->device]% foreach -r rack02 ( clear mac ) ; commit
[...removes MAC association with nodes from CMDaemon...]
[...now reboot nodes in rack02 in sequence (not with Bright Cluster Manager)...]
[bright73->device]% newnodes
[...shows sequence as the nodes come up..]
[bright73->device]% newnodes -s -r rack02
[...assigns sequence in boot order...]
```
**newnodes advanced use—assignment forcing example:** The `--force` option can be used in the following case: Supposing that node022 fails, and a new node hardware comes in to replace it. The new regular node has a new MAC address. So, as explained by scenario 3 (section 5.4.2), if there is no switch port assignment in operation for the nodes, then the node-installer loops around, waiting for intervention.\(^1\)

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

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

  ```bash
  [bright73->device]# newnodes -s -f -n node022
  ```

**Node Identification Wizard**

The node identification wizard can be accessed from the tabbed pane under the Nodes resource (figure 5.15). It can also be accessed by double-clicking, in the event viewer, on the event message “Nodes waiting to be identified. Double click event to assign”.

The node identification wizard is roughly the cmgui equivalent to the `newnodes` command of cmsh. Like `newnodes`, the wizard lists the MAC address and switch port of any unassigned node that the head node detects. Also, like `newnodes`, it can help assign a node name to the node, assuming the node object exists. After assignment is done, a prompt to save the new status appears.

![Node Identification Wizard](image)

**Figure 5.15: Node Identification Wizard**

The most useful way of using the wizard is for node assignment in large clusters. To do this, it is assumed that the node objects have already been created for the new nodes. The creation of the node objects means that the node names exist, and so assignment to the node names is able to take place. An easy way to create 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 node022 without further intervention.

\(^1\)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
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 configuration (section 3.7) with an IP address, netmask and user/password settings so that layer 3 (TCP/IP) networking works for it. BMC networking runs independently of node networking.

Bringing up non-BMC, non-provisioning network interfaces: Non-provisioning interfaces are inactive unless they are explicitly brought up. Bright 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 180) 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 182) 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 four values: `AUTO`, `FULL`, `MAIN` and `NOSYNC`. It should be understood that the term “install-mode” implies that these values operate only during the node-installer phase.\(^2\)

- If the install-mode is set to `FULL`, 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`, 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

\(^2\)For example, `imageupdate` (section 5.6.2), which is run by CMDaemon, ignores these settings, which is as expected. This means that, for example, if `imageupdate` is run with `NOSYNC` set, then the head node image is still synchronized over as usual to the regular node while the node is up. It is only during node boot, during the installer stage, that setting NOSYNC prevents synchronization.
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`, the node-installer halts in 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 config-
  uration, 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 182).

`NOSYNC` is useful during mass planned node reboots when set with the `nextinstallmode` option
of device mode. This sets the nodes to be provisioned, during the next boot only, without an
image sync:

**Example**

```
[bright73]\% device foreach -n node001..node999 (set nextinstallmode nosync)
[bright73]\% device commit
```

**Install-mode Logging**

The decision that is made is normally logged to the node-installer file, `/var/log/node-installer` on the head node.

**Example**

```
08:40:58 node001 node-installer: Installmode is: AUTO
08:40:58 node001 node-installer: Fetching disks setup.
08:40:58 node001 node-installer: Setting up environment for initialize scripts.
08:40:58 node001 node-installer: Initialize script for category default is empty.
08:40:59 node001 node-installer: Checking partitions and filesystems.
08:40:59 node001 node-installer: Updating device status: checking disks
08:40:59 node001 node-installer: Detecting device '/dev/sda': found
08:41:00 node001 node-installer: Number of partitions on sda is ok.
08:41:00 node001 node-installer: Size for /dev/sda1 is ok.
08:41:00 node001 node-installer: Checking if /dev/sda1 contains ext3 filesystem.
08:41:01 node001 node-installer: fsck.ext3 -a /dev/sda1
08:41:02 node001 node-installer: Size for /dev/sda1: recovering journal
08:41:02 node001 node-installer: /dev/sda1: clean, 129522/1250928 files, 886932/5000000 blocks
08:41:02 node001 node-installer: Filesystem check on /dev/sda1 is ok.
08:41:02 node001 node-installer: Size for /dev/sda2 is wrong.
08:41:02 node001 node-installer: Partitions and/or filesystems are missing/corrupt. (Exit code 18, signal 0)
08:41:03 node001 node-installer: Creating new disk layout.
```

© Bright Computing, Inc.
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 cmgui (figure 5.16):

   ![Figure 5.16: cmgui Install-mode Settings Under Node Category](image)

   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 5.17). 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 cmgui (figure 5.18):

![CMGUI Install-mode Settings](image)

Figure 5.18: cmgui Install-mode Settings For The Node

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 `cmgui` (figure 5.18), or using `cmsh` with a one-liner such as:

   ```bash
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 `cmgui` (figure 5.16), or using `cmsh` with a one-liner such as:

   ```bash
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 5.19) 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`:

   ![Cluster Manager Node Installer](image)

   Figure 5.19: Install-mode Setting Option During Node-Installer Run

**FULL Install Confirmation**

Related to execution mode values is the “Do a FULL install only after explicit confirmation” value. 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.

**FULL install confirmation checkbox**: The property can be set in the node configuration with `cmgui` (figure 5.20):

© Bright Computing, Inc.
5.4 Node-Installer

FULL install confirmation via datanode setting: Alternatively, the parameter datanode can be set using a cmsh one-liner as follows:

[root@bright73 ]# 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 cmgui or cmsh will also work.
- From cmsh, within device mode, using the installerinteractions command (some output elided):

  Example

  [bright73->device]# installerinteractions -w -n node001 --confirm
  Hostname     Action
  -----------------  ---------------------------------------------
  node001 Requesting FULL Install (partition mismatch)
  [bright73->device]#
  ...07:57:36 [notice] bright73: node001 [ INSTALLER_CALLINGINIT ]...
  [bright73->device]#
  ...07:58:20 [notice] bright73: node001 [ UP ]

© Bright Computing, Inc.
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.

Further details on the command can be viewed by running `help installerinteractions`.

An alternative way to avoid overwriting node storage: There is an alternative method to prevent data loss on nodes due to a FULL install. The alternative does not require setting the FULL install checkbox in cmgui, nor does it require setting datanode in cmsh. Instead, 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:

```
[bright73->device[node001]]% append blockdevicesclearedonnextboot /dev/sda /dev/sdb ; commit
```

The value of `blockdevicesclearedonnextboot` 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 3.15.4). For example, to initialize some not explicitly supported hardware, or to do a RAID configuration lookup for a particular node. In such cases the custom commands are added to an initialize script. How to edit an initialize script is described in Appendix E.2.

An initialize script can be added to both a node’s category and the node configuration. The node-installer first runs an initialize script, if it exists, from the node’s category, and then an initialize script, if it exists, from the node’s configuration.

The node-installer sets several environment variables which can be used by the initialize script. Appendix E contains an example script documenting these variables.

Related to the initialize script is the finalize script (section 5.4.11). This may run after node provisioning is done, but just before the init process on the node runs.

### 5.4.6 Checking Partitions, RAID Configuration, Mounting Filesystems

**Behavior As Decided By The Install-Mode Value**

In section 5.4.4 the node-installer determines the install-mode value, along with when to apply it to a node.

**AUTO:** The install-mode value is typically set to default to AUTO. If AUTO applies to the current node, it means the node-installer then checks the partitions of the local drive and its filesystems and recreates them in case of errors. Partitions are checked by comparing the partition layout of the local drive(s) against the drive layout as configured in the node’s category configuration and the node configuration.

After the node-installer checks the drive(s) and, if required, recreates the layout, it mounts all filesystems to allow the drive contents to be synchronized with the contents of the software image.

**FULL or MAIN:** If install-mode values of FULL or MAIN 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: Bright Cluster Manager supports hardware RAID levels 0, 1, 5, 10, and 50, and supports the following options:

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

- stripe size:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cached</td>
<td></td>
</tr>
<tr>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

- cache policy:

<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>

- read policy:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WT</td>
<td>Write Through</td>
</tr>
<tr>
<td>WB</td>
<td>Write Back</td>
</tr>
</tbody>
</table>

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 NOSYNC is set, and takes place if install-mode values of 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 software image that is requested is available to nodes by default. It can however be set to a locked state, which means that the request is held in the queue until it is no longer in a locked state. This is sometimes useful for making changes to an image when nodes are booting.

Example

```bash
[root@bright73 ~]# cmsh
[bright73]% softwareimage use default-image
[bright73->softwareimage[default-image]]% set locked yes
[bright73->softwareimage*[default-image*]]% commit
```

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 Bright 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:

```
cmsh -c "category use default; set excludelistfullinstall; commit"
```

where a text-editor opens up to allow changes to be made to the list. Figure 5.21 illustrates how the setting can be modified via cmgui.

![Figure 5.21: Setting up exclude lists with cmgui for provisioning](image)

Image synchronization is done using rsync, and the syntax of the items in the exclude lists conforms to the “INCLUDE/EXCLUDE PATTERN RULES” section of the rsync(1) man page, which includes patterns such as “**”, “?”, and “[[:alpha:]]”.

The `excludelistfullinstall` and `excludelistsyncinstall` properties decide how a node synchronizes to an image during boot. For a node that is already fully up, the related `excludelistupdate` property decides how a running node synchronizes to an image without a reboot event, and is discussed in section 5.6.

**Interface Used To Receive Image Data:** `provisioninginterface`

For regular nodes with multiple interfaces, one interface may be faster than the others. If so, it can be convenient to receive the image data via the fastest interface. Setting the value of `provisioninginterface`, which is a property of the node configuration, allows this.

By default it is set to `BOOTIF` for regular nodes. Using `BOOTIF` is not recommended for node configurations with multiple interfaces.

When listing the network interfaces in cmsh, the provisioning interface has a `[prov]` flag appended to its name.

**Example**

```
[bright73]->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 13), 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 13.1).
Transport Protocol Used For Image Data: \texttt{provisioningtransport}

The \texttt{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 \texttt{device} mode for the receiving node to one of these values:

- \texttt{rsyncdaemon}, which sends the data unencrypted
- \texttt{rsyncssh}, which sends the data encrypted

The \texttt{provisioningtransport} value can be set for all nodes, including provisioning nodes, head nodes, and cloud-director (section 3.2 of the \textit{Cloudbursting Manual}) nodes. Because encryption severely increases the load on the provisioning node, using \texttt{rsyncssh} is only suggested if the users on the network cannot be trusted. By default, \texttt{provisioningtransport} is set to \texttt{rsyncdaemon}. If high availability (chapter 13) is set up with the head nodes exposed to the outside world on the external network, the administrator should consider setting up \texttt{rsyncssh} for the head nodes.

The \texttt{rsyncssh} transport requires passwordless root access via \texttt{ssh} from the provisioner to the node being provisioned. This is configured by default in the default Bright Cluster Manager nodes. However, if a new image is created with the \texttt{--exclude} options for \texttt{cm-create-image} as explained in (section 9.6.2), the keys must be copied over from \texttt{/root/.ssh/} on the existing nodes.

Tracking The Status Of Image Data Provisioning: \texttt{provisioningstatus}

The \texttt{provisioningstatus} command within the \texttt{softwareimage} mode of \texttt{cmsh} displays an updated state of the provisioning system. As a one-liner, it can be run as:

```
bright73:~ # 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 bright73:
  Max number of provisioning nodes: 10
  Nodes provisioning: 0
  Nodes currently being provisioned: <none>
```

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

The \texttt{cmgui} equivalent to \texttt{provisioningstatus} is accessed from the “Provisioning Status” tabbed pane in the “Software Images” resource (figure 5.5). By default, it displays basic status information on provisioning requests, while the “Request Details” button allows details to be seen of provisioning requests.

Tracking The Provisioning Log Changes: \texttt{synclog}

For a closer look into the image file changes carried out during provisioning requests, the \texttt{synclog} command from \texttt{device} mode can be used (lines elided in the following output):

Example

```
[bright73->device]# synclog node001
Tue, 11 Jan 2011 13:27:17 CET - Starting rsync daemon based provisioning. Mode is SYNC.

sending incremental file list
./
...
deleting var/lib/ntp/etc/ localtime
```

© Bright Computing, Inc.
5.4 Node-Installer

var/lib/ntp/var/run/ntp/
...
sent 2258383 bytes  received 6989 bytes  156232.55 bytes/sec
total size is 1797091769  speedup is 793.29


In cmgui, the equivalent output to cmsh’s synclog is displayed by selecting a specific device or a specific category from the resource tree. Then, within the Tasks tabbed pane that opens up, the "Provisioning Log" button at the bottom right is clicked (figure 5.22):

![Figure 5.22: cmgui: Provisioning Log Button For A Device Resource](image)

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:

```
bright3:~ # cmsh -c "softwareimage cancelprovisioningrequest -a"
```

The provisioningstatus command of cmsh, or the “Provisioning Status” tab of cmgui can be used to find request IDs. Individual request IDs, for example 10 and 13, can then be specified in the cancelprovisioningrequest command, as:

```
bright3:~ # cmsh -c "softwareimage cancelprovisioningrequest 10 13"
```

The help page for cancelprovisioningrequest shows how to run the command on node ranges, groups, categories, racks, and chassis.

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

© Bright Computing, Inc.
for each configured network interface. These are files like:

/etc/sysconfig/network-scripts/ifcfg-eth0

for Red Hat, Scientific Linux, and CentOS, 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 cmgui 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 9.6.1), 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 9.6.2) is impractical.

Providing a custom working image from a standalone node that has been customized after direct installation from the parent distribution, ensures the GRUB boot configuration layout of the custom image is as expected by the parent distribution. This then allows a standalone GRUB boot on the node to run properly.

With a working custom software image, to set the GRUB bootloader in cmgui, 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
   cmsh -c "device use node001; set installbootrecord yes; commit"
```

or

```cmsh
   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.

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.

As suggested by the Bright Cluster Manager iPXE boot prompt, setting network booting to work from the BIOS (regular “PXE” booting) is preferred to iPXE booting from the disk.

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 3.15.4) it is used when custom commands need to be executed after the preceding mounting, provisioning, and housekeeping steps, but before handing over control to the node’s local init process. For example, custom commands may be needed to:

- initialize some not explicitly supported hardware before init takes over
- supply a configuration file for the software image that cannot simply be added to the software image and used by init because it needs node-specific settings
- load a slightly altered standard software image on particular nodes, typically with the change depending on automatically detecting the hardware of the node it is being loaded onto. While this could also be done by creating a full new software image and loading it on to the nodes according to the hardware, it usually turns out to be better for simplicity’s sake (future maintainability) to minimize the number of software images for the cluster.

The custom commands used to implement such changes are then added to the finalize script. How to edit a finalize script is described in Appendix E.2.

A finalize script can be added to both a node’s category and the node configuration. The node-installer first runs a finalize script, if it exists, from the node’s category, and then a finalize script, if it exists, from the node’s configuration.

The node-installer sets several environment variables which can be used by the finalize script. Appendix E contains an example script which documents these variables.

5.4.12 Unloading Specific Drivers

Many kernel drivers are only required during the installation of the node. After installation they are not needed and can degrade node performance.

Baseboard Management Controllers (BMCs, section 3.7) that use IPMI drivers are an egregious example of this. The IPMI drivers are required to have the node-installer configure the IP address of any IPMI cards. Once the node is configured, these drivers are no longer needed, but they continue to consume significant CPU cycles and power if they stay loaded, which can affect job performance.

To solve this, the node-installer can be configured to unload a specified set of drivers just before it hands over control to the local init process. This is done by editing the removeModulesBeforeInit setting in the node-installer configuration file /cm/node-installer/scripts/node-installer.conf. 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 cmgui

In the node icons used by cmgui (figure 5.23):

- Nodes in the **UP** state are indicated by an up-arrow.
  - If all health checks (section 10.2.4) for the node are successful, the up-arrow is green.
  - If there is a health check that fails or if the node requires a reboot, the up-arrow is red.
- Nodes in the **DOWN** state are indicated by a blue down-arrow.
- Nodes in all other states are indicated by a white horizontal dash inside a solid red circle.

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**

```
[bright73->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 35) command:

**Example**

```
[bright73->device]$ list -f "hostname:10, status:48"
hostname ( status
---------- ------------------------------------------------
apc01     [ UP ]
bright73 [ 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).

© Bright Computing, Inc.
5.5 Node States

5.5.3 Node States Indicating Regular Start Up
During a successful boot process the node goes through the following states:

- **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 cmgui and cmsh are normally, in sequence:
  1. node-installer started
  2. Optionally, the following two messages:
     (a) waiting for user input
     (b) installation was resumed
  3. checking disks
  4. recreating partitions and filesystems
  5. mounting disks
  6. One of these following two messages:
     (a) waiting for FULL provisioning to start
     (b) waiting for SYNC provisioning to start
  7. provisioning started, waiting for completion
  8. provisioning complete
  9. initializing SELinux

Between steps 1 and 3 in the preceding, these optional messages can also show up:

- If **burn** mode is entered or left:
  running burn-in tests
  burn-in test completed successfully
- If **maintenance** mode is entered:
  entered maintenance mode

- **INSTALLER_CALLINGINIT.** This state is entered as soon as the node-installer has handed over control to the local init process. The associated message normally seen with it in cmgui and cmsh 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@b52 ~]# cmsh
[b52]\% device
[b52->device]\% close -m "fan dead" -n node001,node009,node020
Mon May 2 16:32:01 [notice] b52: node001 ...[ DOWN/CLOSED ] (fan dead)
Mon May 2 16:32:01 [notice] b52: node009 ...[ DOWN/CLOSED ] (fan dead)
Mon May 2 16:32:01 [notice] b52: node020 ...[ DOWN/CLOSED ] (fan dead)
```

The `--noping` option for the `close` command of `cmsh` causes the state ping tracking to cease. The additional text `state ping disabled` is then used alongside the `CLOSED` state message to indicate when the regular node state is no longer tracked by CMDaemon on the head node.

The `CLOSED` state can also be set from `cmgui`. This is done via the Tasks tab of the node item in the `Nodes` resource, or from the category item in the `Node Categories` resource (figure 5.24).

![Figure 5.24: Acting On A CLOSED State From cmgui](image)

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 the restart-required flag (section 5.5.2). The reason that set the restart-required flag is not solved by the reset.
  - `--failbeforedown <count>`: Specifies the number of failed pings before a device is marked as down (default is 1). This is actually a special case of the `Threshold duration` parameter in a metric (page 397) or a health check (page 402).
- `drain` and `undrain` (Appendix G.3.1)
- For nodes that have power control\(^3\):

\(^3\)Power control mechanisms such as PDUs, BMCs using IPMI/HP iLO, and custom power scripts are described in Chapter 4
5.5 Node States

* power -f on
* power -f off
* power -f reset

The cmgui equivalents in figure 5.24 are:
- The Open button from the Watch row
- The Drain and Undrain buttons from the Workload row
- The On, Off, and Reset buttons from the PDU Power row

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. Executing the close command with the option --noping in cmsh disables even state tracking for the device by CMDaemon. Executing the close command without the --noping option then enables status tracking again.

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 or cmgui “open” options discussed earlier. The node, as far as CMDaemon is concerned, then switches from the CLOSED state to the OPEN state. Whether the node listens or not does not matter—the head node records it as being in an OPENING state for a short time, and during this time the next OPEN state (UP/OPEN, DOWN/OPEN, etc) is agreed upon by the head node and the node.

When querying the state of a node, an OPEN tag is not displayed in the response, because it is the “standard” state. For example, UP is displayed rather than UP/OPEN. In contrast, a CLOSED tag is displayed when it is active, because it is a “special” state.

The CLOSED state is normally set to take a node that is unhealthy out of the cluster management system. The node can then still be in the UP state, displaying UP/CLOSED. It can even continue running workload jobs in this state, since workload managers run independent of CMDaemon. So, if the workload manager is still running, the jobs themselves are still handled by the workload manager, even if CMDaemon is no longer aware of the node state until the node is re-opened. For this reason, draining a node is often done before closing a node, although it is not obligatory.

- OPENING. This transitional state is entered as soon as the CMDaemon of the node rescinds the CLOSED state with an “open” command from cmsh or cmgui. The state usually lasts no more than about 5 seconds, and never more than 30 seconds in the default configuration settings of Bright 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

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).

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 cmgui, and is queued and delegated to a provisioning node, just like a regular provisioning request. The properties that apply to the regular provisioning 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), while in cmgui, it is submitted using the “Update node” button (section 5.6.3). The imageupdate command and “Update node” button use a configuration file called exclodelistupdate, which is, as its name suggests, a list of exclusions to the update.

The imageupdate command and “Update node” button update what is on a running node from a stored image. 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 (cmgui), and involves further exclude lists exclodelistgrab or exclodelistgrabnew. The grabimage command and Synchronize option are covered in detail in section 9.5.2.

#### 5.6.1 Updating Running Nodes: Configuration With exclodelistupdate

The exclude list exclodelistupdate 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 exclodelistupdate list that is being discussed here, in contrast with the exclodelistsyncinstall/exclodelistfullinstall lists which are discussed in section 5.4.7, and also in contrast with the exclodelistgrab/exclodelistgrantnew lists of section 9.5.2.

So, for the imageupdate command the exclodelistupdate list concerns an update to a running system, while for installation sync or full provisioning, the corresponding exclude lists (exclodelistsyncinstall and exclodelistfullinstall) 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 exclodelistupdate exclusion list it uses is actually analogous to the exclodelistsyncinstall exclusion list used in the sync case of section 5.4.7, rather than being analogous to the exclodelistfullinstall list.

Similarly, the exclodelistgrab/exclodelistgrabnew lists of section 9.5.2 are about a grab from the running node to the image.

• The exclodelistgrab list here is intended for the case of synchronizing the existing image with the running node, and is thus analogous to the exclodelistsyncinstall exclusion list.
• The `excludelistgrabnew` list here is intended for the case of copying a full image from the running node, and is thus analogous to the `excludelistfullinstall` list.

The following table summarizes this:

<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></td>
<td>full, image to starting node</td>
</tr>
<tr>
<td></td>
<td><code>excludelistsyncinstall</code></td>
<td>sync, image to starting node</td>
</tr>
<tr>
<td>grab</td>
<td><code>excludelistgrabnew</code></td>
<td>full, running node to image</td>
</tr>
<tr>
<td></td>
<td><code>excludelistgrab</code></td>
<td>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:

<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>eg: imageupdate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>install</td>
<td><code>excludelistfullinstall</code></td>
<td>full, image to starting node</td>
</tr>
<tr>
<td>eg: node-provisioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>process during pre-init stage depending</td>
<td></td>
<td></td>
</tr>
<tr>
<td>on installmode decision</td>
<td></td>
<td>sync, image to starting node</td>
</tr>
<tr>
<td>eg: node provisioning AUTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>with healthy partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grab</td>
<td><code>excludelistgrabnew</code></td>
<td>full, running node to image</td>
</tr>
<tr>
<td>eg: <code>grabimage</code> (cmsh), <code>grabimage -i</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronize to image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(cmgui) and Grab to image (cmgui)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>image (cmgui)</td>
<td><code>grabimage/Synchronize image</code></td>
<td></td>
</tr>
</tbody>
</table>

**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.

© Bright Computing, Inc.
• 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 5.25.

Figure 5.25: Exclude list logic
The files and directories on the node that are not in the sub-lists lose their original contents, and are copied over from the software image. So, content not covered by the sub-lists at that time is normally not protected from deletion.

Thus, the provisioning system excludes paths described according to the `excludelistupdate` property.

The provisioning system also excludes a statically-imported filesystem on a node if the filesystem is a member of the following special list: NFS, Lustre, FUSE, CephFS, CIFS, PanFS, FhGFS, BeeGFS, ClusterFS, 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 Bright Cluster Manager via the `fsmounts` mode can be excluded from being mounted on the nodes in the first place, by removing them from the listed mounts within the `fsmounts` mode.

Imported filesystems not on the special list can have their data wiped out during provisioning or sync updates, if the statically-imported filesystems are placed in the image manually—that is, if the filesystems are mounted manually into the image on the head node via `/etc/fstab` without using `cmsh` or `cmgui`.

Filesystems mounted dynamically, that is, with an auto-mounter cannot have their appearance or disappearance detected reliably. Any filesystem that may be imported via an auto-mount operation must therefore explicitly be excluded by the administrator manually adding the filesystem to the exclude list. This is to prevent an incorrect execution of `imageupdate`. Neglecting to do this may wipe out the filesystem, if it happens to be mounted in the middle of an `imageupdate` operation.

### Editing An Exclude List

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

```
cmsh -c "category use default; set excludelistupdate; commit"
```

Similarly, the exclude list for updates can also be edited in `cmgui` as indicated in figure 5.26.

![Figure 5.26: Setting up exclude lists with cmgui for node updates](image)

### Provisioning Modifications Via `excludelistmanipulatescript`

Sometimes the administrator has a need to slightly modify the execution of exclude lists during provisioning. The `excludelistmanipulatescript` file takes as an input the exclude list inherited from a category, modifies it in some way, and then produces a new exclude list. Conceptually it is a bit like how an administrator might use `sed` if it worked without a pipe. As usual, setting it for node level overrides the category level.
A script that manipulates the exclude lists of a node can be specified as follows within cmsh:

```bash
#!/bin/bash

echo "- *
no-new-files: - *"
```

The script can be as simple as:

**Example**

```bash
#!/bin/bash

echo "- *
no-new-files: - *
```

If provisioning a node from the head node, then the script modifies the node-provisioning exclude lists—`excludelistfullinstall`, `excludelistsyncinstall`, and `excludelistupdate`—so that they appear to contain these items only:

- *
no-new-files: - *

The provisioning node then excludes everything during provisioning.

Careful editing and testing of the script is advised. Saving a script with just a single whitespace, for example, usually has undesirable results.

A more useful script template is the following:

**Example**

```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:

```
[bright73]\% device use node001
[bright73\->device[node001]]\% get excludelistmanipulatescript
  (the script is put in)
[bright73\->device[node001\*]]\% commit; imageupdate
  Performing dry run (use synclog command to review result, then pass -w to perform real update)...  
  - yes, no data changes!  
[bright73\->device[node001]]
```

```
[bright73]\% device use node001
[bright73\->device[node001]]\% get excludelistmanipulatescript
  (the script is put in)
[bright73\->device[node001\*]]\% commit; imageupdate
  Performing dry run (use synclog command to review result, then pass -w to perform real update)...  
  - yes, no data changes!  
imageupdate [ COMPLETED ]
```
An excerpt from the sync log, after running the `synclog` command, then shows output similar to (some output elided):

```
...  
- /cm/shared/*  
- /cm/shared/  
- /home/*  
- /home/  
- /cm/shared/apps/torque/*  
- /cm/shared/apps/torque/  
# This and next line added by category excludelistmanipulatescript.  
# The command line arguments were: update /
```

Rsync output:
```
sending incremental file list
```
```
cm/local/apps/cmd/scripts/healthchecks/configfiles/
```
```
...  
```

Here, the sync mode is `update` and the destination path is “/”. Which of the exclude lists is being modified can be determined by the `excludelistmanipulatescript` by parsing the sync mode.

The bash variable that accepts the exclude list text is set to a safely-marked form using curly braces. This is done to avoid expansion surprises, due to wild card characters in the exclude lists. For example, if `$REPLY` were used instead of `${REPLY}`, and the script were to accept an exclude list line containing “- /proc/*”, then it would give quite confusing output.

Two further exclude list files that modify standard provisioning behavior are `excludelistfailover` and `excludelistnormal`. These are discussed in section 13.4.8.

### 5.6.2 Updating Running Nodes: With cmsh Using `imageupdate`

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

**Example**

```
[bright73->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 bright73: Provisioning started on node node001
[bright73->device]# imageupdate -n node001: image update in progress ...
[bright73->device]#
Tue Jan 11 12:13:44 2011 bright73: 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**

```
[bright73->device]# imageupdate -n node001 -w
Provisioning started on node node001
node001: image update in progress ...
[bright73->device]# Provisioning completed on node node001
```

© Bright Computing, Inc.
5.6.3 Updating Running Nodes: With cmgui Using The Update node Button

In cmgui an image update can be carried out by selecting the specific node or specific category from the resource tree. Then, within the tasks tabbed pane that opens up, the “Update node” button is clicked (figure 5.27). This opens up a dialog which has a dry-run checkbox marked by default.

The dry-run can be reviewed by clicking on the “Provisioning Log” button further down the same tabbed pane. The update can then be done again with the dry-run check mark off to actually implement the update.

5.6.4 Updating Running Nodes: Considerations

Updating an image via cmsh or cmgui automatically updates the provisioners first via the updateprovisioners command (section 5.2.4) if the provisioners have not been updated in the last 5 minutes. The conditional update period can be set with the provisioningnodeautoupdatetimeout parameter (section 5.2.4).

So, with the default setting of 5 minutes, if there has been an update within the last 5 minutes, then provisioners do not get an updated image when doing the updates. Running the updateprovisioners command just before running the imageupdate command therefore usually makes sense.

Also, when updating services, the services on the nodes may not restart since the init process may not notice the replacement.

For these reasons, especially for more extensive changes, it can be safer for the administrator to simply reboot the nodes instead of using imageupdate to provision the images to the nodes. A reboot by default ensures that a node places the latest image with an AUTO install (section 5.4.7), and restarts all services.

The “Reinstall node” button (figure 5.27) 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 cmgui Add Functions

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

Example
5.7 Adding New Nodes

[bright73->device]% add physicalnode node002 10.141.0.2
[bright73->device*][node002*]% commit

The cmgui equivalent of this is to go within the Nodes resource, and after the Overview tabbed pane for the Nodes resource comes up, to click on the Add button (figure 5.28)

[Image of cmgui interface with nodes and add button highlighted]

Figure 5.28: Buttons To Add, Remove And Set Up Nodes

When adding the node objects in cmsh and cmgui, 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 cmgui by the administrator.

Adding new node objects as “placeholders” can also be done from cmsh or cmgui. 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 cmgui as in the previous section, there is also a cmgui 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 from the Nodes resource, by selecting the Overview tabbed pane and then the “Create Nodes” button (figure 5.28).

This wizard should not be confused with the closely related node identification wizard described earlier in section 5.4.2, which identifies unassigned MAC addresses and switch ports, and helps assign them node names.

The node creation wizard instead 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” (figure 5.29).
The MAC addresses can be assigned to a node via the node identification wizard. However, leaving nodes in a “placeholder” state, where the MAC address entry is left unfilled, means that any new node with an unassigned MAC address that is started up is offered a choice out of the created node names by the provisioning system at its console. This happens 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.

The node creation wizard can set IP addresses for the nodes. At one point in the dialog a value for IP-offset can also be set (figure 5.30).
The default setting for IP-offset is 0.0.0.0, and means the default IP address is suggested for assignment to each node in the range. The default IP address is based on the node name, with node001 having the value 10.141.0.1, and so on. An offset of x implies that the xth IP address after the default is suggested for assignment to each node in the range. Some care must be taken when setting IP addresses using the wizard, since no duplicate IP address checking is done.

**Example**

A node001 has its default IP address 10.141.0.1. The node005 is then added.

- If IP-offset=0.0.0.0, then 10.141.0.5 is suggested for assignment to node005, because, by default, the node name is parsed and its default IP address suggested.
- If IP-offset=0.0.0.2, then 10.141.0.7 is suggested for assignment to node005, because it is 2 IP addresses after the default.

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.

#### 5.8.1 Node Fails To PXE Boot

Possible reasons to consider if a node is not even starting to PXE boot 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@bright73 ~]# 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 in cmgui the Allow node booting checkbox in figure 3.5 (page 58) needs to be ticked. 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 yes in cmsh, or “Don’t allow new nodes to be booted from this network” is unset in cmgui.

• 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 PXE boot to happen. If a normal PXE 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.

• 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.

• 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.

• The cable may be connected to the wrong interface. By default, eth0 is normally assigned the internal network interface, and eth1 the external network interface on the head node for a type 1 network. However:

  - 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.
  - If the node is booting from iPXE (that is from a hard drive), for the very first time, and there is more than one interface on the node, then the interface assignment is not guaranteed in Red Hat and derivatives before version 7.

The node running a version before RHEL7 can therefore end up unable to pick up the provisioning image from the interface that is in use for provisioning. Moving the provisioning cable to the correct interface on the node at this very first boot resolves this issue.
Interface Naming Conventions Post-RHEL7

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/ch-Consistent_Network_Device_Naming.html describes the persistent naming convention for interfaces in RHEL7. This convention 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.

Reverting To The Pre-RHEL7 Interface Naming Conventions

To revert to the pre-RHEL7 behavior, the text:

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

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

- The `biosdevname` parameter only works if the dev helper is installed. The dev helper is available from the `biosdevname` RPM package. The parameter also requires that the system supports SMBIOS 2.6 or ACPI DSM.
- The `net.ifnames` parameter is needed if `biosdevname` is not installed.

**Example**

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

Then the GRUB configuration should be generated with:

```
[root@bright73 ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

For a regular node, the text `net.ifnames=0 biosdevname=0` can be appended to the `kernelparameters` property, for an image selected from `softwareimage` mode.

**Example**

```
[bright73->softwareimage]% list
Name (key) Path
-------------------- ------------------------------
default-image /cm/images/default-image
openstack-image /cm/images/openstack-image
[bright73->softwareimage]% use default-image
[bright73->softwareimage[default-image]]% append kernelparameters " net.ifnames=0 biosdevname=0"
```

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 connections should be checked to eliminate these possibilities.

- 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@bright73 ~]# ps ax | grep [t]ftp
7512 ? Ss 0:03 in.tftpd --maxthread 500 /tftpboot
```

© Bright Computing, Inc.
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@bright73 ~]# chkconfig --list
```

should include in its output the line:

```
tftp: on
```

if TFTP is running under xinet.d.

- The xinetd server on the head node that manages the TFTP service in RHEL6 may be hung. If so, then it should be restarted.
- The switchover process from TFTP to HTTP may have hung. During a normal provisioning run, assuming that CMDaemon uses the default AdvancedConfig setting of PXEBootOverHTTP=1 (Appendix C), 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 PXEBootOverHTTP=0 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.
- 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 PXE 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 Bright 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**

```
[bright73->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@bright73 ~]# cmsh -c "softwareimage foreach * (get name; get locked)"
default-image
yes
[root@bright73 ~]# cmsh -c "softwareimage; foreach * (set locked no); commit"
[root@bright73 ~]# cmsh -c "softwareimage foreach * (get name; get 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.

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 11.7), to allow the viewing of console messages directly. A further depth in logging can be achieved by setting the kernel option loglevel=N, where N is a number from 0 (KERN_EMERG) to 7 (KERN_DEBUG).

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 or the Provisioning Log button in cmgui (section 5.4.7, page 188). 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.

5.8.4 Ramdisk Fails During Loading Or Sometime Later

One issue that may come up after a software image update via yum or zypper (section 9.4), is that the ramdisk stage may fail during loading or sometime later, for a node that is rebooted after the update. This occurs if there are instructions to modify the ramdisk by the update. In a normal machine the ramdisk would be regenerated. In a cluster, the extended ramdisk that is used requires an update, but Bright Cluster Manager is not aware of this. Running the createramdisk command from cmsh or cmgui (section 5.3.2) generates an updated ramdisk for the cluster, and solves the failure for this case.

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 5.31.
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]]% add e1000
[mc->softwareimage[default-image]]% commit
```

Initial ramdisk for image default-image was regenerated successfully

```
[mc->softwareimage[default-image]]% mkinitrd
```

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 5.32. 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: Partitions and/or filesystems are missing/corrupt. (Exit code 4, signal 0)
```
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.

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.

One of the most common reasons is that the correct storage driver is not being loaded. To solve this issue, the correct kernel module should be added to the software image’s kernel module configuration (section 5.3.2).

Experienced system administrators work out what drivers may be missing by checking the results of hardware probes. For example, going into the node-installer shell using Alt + F2, and then looking at the output of lspci, shows a list of hardware detected in the PCI slots and gives the chipset name of the storage controller hardware in this case:

**Example**

```
[<installer> root@node001 ~]# lspci | grep SCSI
00:10.0 Serial Attached SCSI controller: LSI Logic / Symbios Logic SAS2\ 008 PCI-Express Fusion-MPT SAS-2 [Falcon] (rev 03)
```

© Bright Computing, Inc.
The next step is to Google with likely search strings based on that output. The Linux Kernel Driver DataBase (LKDDb) is a hardware database built from kernel sources that lists driver availability for Linux. It is available at http://cateee.net/lkddb/. Using the Google search engine’s “site” operator to restrict results to the cateee.net web site only, a likely string to try might be:

**Example**

SAS2008 site:cateee.net

The search result indicates that the mpt2sas kernel module needs to be added to the node kernels. A look in the modules directory of the software image shows if it is available:

**Example**

```
find /cm/images/default-image/lib/modules/ -name "*mpt2sas*"
```

If it is not available, the driver module must then be obtained. If it is a source file, it will need to be compiled. By default, nodes run on standard distribution kernels, so that only standard procedures need to be followed to compile modules.

If the module is available, it can be added to the default image, by using cmsh in softwareimage mode to create the associated object. The object is given the same name as the module, i.e. mpt2sas in this case:

**Example**

```
[bright73]\% softwareimage use default-image
[bright73->softwareimage[default-image]]\% kernelmodules
[bright73->softwareimage[default-image]->kernelmodules]\% add mpt2sas
[bright73->softwareimage[default-image]->kernelmodules[mpt2sas]]\% commit
```

Thu May 19 16:54:52 2011 [notice] bright73: Initial ramdisk for image de\ fault-image is being generated

Thu May 19 16:55:43 2011 [notice] bright73: Initial ramdisk for image de\ fault-image was regenerated successfully.

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 5.33.
5.8 Troubleshooting The Node Boot Process

Figure 5.33: 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 in the node-installer configuration file `/cm/node-installer/scripts/node-installer.conf` to `false`. 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 setting `failOnMissingBmc = true` in the node-installer configuration file `/cm/node-installer/scripts/node-installer.conf`. Toggling this to `false` skips BMC network device detection, and lets the node-installer continue past the BMC detection and configuration stage. This can be convenient, for example, if the BMC is not yet configured and the aim is to get on with setting up the rest of the cluster.

**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`.

© Bright Computing, Inc.
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@bright73 ~]# module load ipmitool
[root@bright73 ~]# 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.

### 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@bright73 ~]# service ipmi status
Redirecting to /bin/systemctl status ipmi.service

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@bright73 ~]# 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@bright73 ~]# 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 bright73 systemd[1]: Starting IPMI Driver...
Dec 19 14:34:27 bright73 openipmi-helper[8930]: Startup failed.
Dec 19 14:34:27 bright73 systemd[1]: ipmi.service: main process exited, code=exited, status=1/FAILURE

Dec 19 14:34:27 bright73 systemd[1]: Failed to start IPMI Driver.
Dec 19 14:34:27 bright73 systemd[1]: Unit ipmi.service entered failed state.
Dec 19 14:34:27 bright73 systemd[1]: ipmi.service failed.
```
5.8 Troubleshooting The Node Boot Process

Further details can be found in the journal:

Example

[root@bright73 ~]# journalctl -xe | grep -i ipmi
... -- Unit ipmi.service has begun starting up.
Dec 19 14:34:27 bright73 kernel: ipmi message handler version 39.2
Dec 19 14:34:27 bright73 kernel: IPMI System Interface driver.
Dec 19 14:34:27 bright73 kernel: ipmi_si: Unable to find any System Interface(s)
Dec 19 14:34:27 bright73 openipmi-helper[8930]: Startup failed.
...

In the preceding session, the failure is due to a missing BMC interface (Unable to find any System Interface(s). A configured BMC interface should show an output status similar to:

Example

[root@bright73 ~]# 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 bright73 systemd[1]: Starting IPMI Driver...
Dec 19 14:37:10 bright73 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:

Example

[root@bright73 ~]# module load ipmitool
[root@bright73 ~]# 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

© Bright Computing, Inc.
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 Bright 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@bright73 ~]# module load ipmitool
[root@bright73 ~]# ipmitool sensor list all
    # ipmitool sensor list
Ambient Temp | 22.000 | degrees C | ok | na | na | na | 38.000 | 41.000 | 45.000
AVG Power | 300.000 | Watts | ok | na | na | na | na | na | na | na
Fan 1 Tach | 4125.000 | RPM | ok | na | 750.000 | na | na | na | na | na
...
User Management

Users and groups for the cluster are presented to the administrator in a single system paradigm. That is, if the administrator manages them with the Bright Cluster Manager, then the changes are automatically shared across the cluster (the single system).

Bright 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 Bright Cluster Manager.

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

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

6.1 Managing Users And Groups With cmgui

Selecting “Users & Groups” from the Resources tree within cmgui (figure 6.1) by default lists the LDAP object entries for regular users. These entries are clickable and can be managed further.

There is already one user on a newly installed Bright 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. Bright 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.

There are five buttons, Add, Save, Edit, Revert, and Remove, available in the Users & Groups resource tabbed pane:

1. Add: allows users to be added via a dialog (figure 6.2). These additions can be committed via the Save button.

An explanation of the less obvious items in the dialog follows:
• **Expiration warning**: The number of days, before the password expires, that the user is warned of the expiry

• **Shadow max**: The maximum number of days the password is valid

• **Shadow min**: The minimum number of days required between password changes. A value of zero means the user may change their password at any time

• **Inactivity**: The number of days of inactivity allowed for the user before the account is blocked. A value of zero means the user is never blocked

• **Management profile**: The preconfigured capability that the user is assigned. Available settings are:
  - **admin**: Allows the user to run `cmgui` with the same privileges as user `admin`
  - **cloudjob**: Allows the user to run `cmsub`, the cloud job submission utility (section 4.7 of the *User Manual*)
  - **portal**: Allows the user to access the user portal
  - **readonly**: Allows the user to run `cmgui` without the ability to modify settings.
  - **none**: (default). Prevents the user from using `cmgui`

A profile setting only takes effect if the certificate for the user is used. User certificates are only persistent for a cluster with a permanent license (page 66 of the *Installation Manual*), so the administrator should check the license is not a temporary license before attempting to use this feature. Section 6.4 explains the concepts of capabilities, profiles, certificates, and tokens.

2. **Save**: saves the as-yet-uncommitted Add or Edit operations. When saving an addition:
• User and group ID numbers are automatically assigned from UID and GID 1000 onwards. Normally Red Hat and similar distributions assign from 500 onwards, while SUSE assigns from 1000 onwards.
• A home directory is created and a login shell is set. Users with unset passwords cannot log in.

3. **Edit**: allows user attributes to be modified via a dialog similar to the Add dialog of figure 6.2.

4. **Revert**: discards unsaved edits that have been made via the **Edit** button. The reversion goes back to the last save.

5. **Remove**: removes selected rows of users. By default, along with their home directories.

Group management in **cmgui** is started by selecting the **Groups** tab in the **Users & Groups** pane. 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 button functions as for user management.

### 6.2 Managing Users And Groups With **cmsh**

User management tasks as carried out by **cmgui** 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 **cmgui** of section 6.1. These functions are run from within the **user** mode of **cmsh**:

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# user
[bright73->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**

```
[bright73->user]# add maureen
[bright73->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**

```
[bright73->user* [maureen*]]% modified
State Type Name
-------- ------------------------------ ------------------------------
+ User maureen
```

This corresponds roughly to the functionality of the **List of Changes** menu option under the **View** menu of the main menu bar of **cmgui**.

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

[bright73->user*[maureen*]]% show
Parameter Value
------------------------------- --------------------------
Common name
Expiration date 2038/1/1
Group ID
Home directory
Inactive 0
Last change 1970/1/1
Login shell
Password < not set >
Profile
Revision
Shadow max 999999
Shadow min 0
Shadow warning 7
User ID
User name maureen
email

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

[bright73->user*[maureen*]]% commit
[bright73->user[maureen]]% show
Parameter Value
------------------------------- ---------------------------
Common name maureen
Expiration date 2038/1/1
Group ID 1002
Home directory /home/maureen
Inactive 0
Last change 2011/5/30
Login shell /bin/bash
Password ********
Profile
Revision
Shadow max 999999
Shadow min 0
Shadow warning 7
User ID 1002
User name maureen
email

If, however, commit were to be run at the user mode level without dropping into the maureen object level, then instead of just that modified user, all modified users would be committed.

When the commit is done, all the empty fields for the user are automatically filled in with defaults based the underlying Linux distribution used. Also, as a security precaution, if an empty field (that is, a “not set”) password entry is committed, then a login into 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 button 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**

```
[bright73->user[maureen]]% set
Name:
    set - Set specific user property
Usage:
    set [user] <parameter> <value> [<value> ...]
Arguments:
    user
        userID of the user, omit if current is set
Parameters:
    Revision ............ Object revision
    commonname .......... Full user name
    email ............... email
    expirationdate ...... Indicates the date on which the user login will be disabled
    groupid ............. Base group of this user
    hadoopdfsaccess .... Home directories on specified Hadoop HDFS
    homedirectory ...... Home directory
    inactive ............ Indicates the number of days of inactivity allowed for the user
    loginshell .......... Login shell
    password ............ Password
    profile ............. Profile for Authorization
    shadowmax ........... Indicates the maximum number of days for which the user password remains valid.
    shadowmin ........... Indicates the minimum number of days required between password changes
    shadowwarning ...... The number of days of advance warning given to the user before the user password expires
    surname ............. Surname
    userid .............. User id number
    username ............ User name
```

© Bright Computing, Inc.
Continuing the session from the end of section 6.2.2, the password can be set at the user context prompt like this:

**Example**

```
[bright73->user[maureen]]% set password setecaStr0n0my
[bright73->user*[maureen*]]% commit
[bright73->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**

```
[bright73->user[maureen]]% clear password; commit
```

**Editing Groups With append And removefrom**

While the above commands `set` and `clear` also work with groups, there are two other commands available which suit the special nature of groups. These supplementary commands are `append` and `removefrom`. They are used to add extra users to, and remove extra users from a group.

For example, it may be useful to have a `printer` group so that several users can share access to a printer. For the sake of this example (continuing the session from where it was left off in the preceding), tim and fred are now added to the LDAP directory, along with a group `printer`:

**Example**

```
[bright73->user[maureen]]% add tim; add fred
[bright73->user*[fred*]]% exit; group; add printer
[bright73->group*[printer*]]% exit; exit; user
[bright73->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**

```
[bright73->user*[maureen*]]% append groupmembers maureen tim fred; commit
[bright73->group*[printer*]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group ID</td>
<td>1003</td>
</tr>
<tr>
<td>Group members</td>
<td>maureen tim fred</td>
</tr>
<tr>
<td>Group name</td>
<td>printer</td>
</tr>
</tbody>
</table>

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:
6.2 Managing Users And Groups With cmsh

```cmsh%
removefrom groupmembers tim fred; commit
show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group ID</td>
<td>1003</td>
</tr>
<tr>
<td>Group members</td>
<td>maureen</td>
</tr>
<tr>
<td>Group name</td>
<td>printer</td>
</tr>
</tbody>
</table>

The `clear` command can also be used to clear members—but it also clears all of the extras from the group:

**Example**

```cmsh%
clear groupmembers
show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group ID</td>
<td>1003</td>
</tr>
<tr>
<td>Group members</td>
<td></td>
</tr>
<tr>
<td>Group name</td>
<td>printer</td>
</tr>
</tbody>
</table>

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 button 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**

```cmsh%
refresh
show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group ID</td>
<td>1003</td>
</tr>
<tr>
<td>Group members</td>
<td>maureen</td>
</tr>
<tr>
<td>Group name</td>
<td>printer</td>
</tr>
</tbody>
</table>

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 Remove button 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**

```
[bright73->group[printer]]% user use maureen
[bright73->user[maureen]]% remove -d; commit
Successfully removed 1 Users
Successfully committed 0 Users
[bright73->user]% !ls -d /home/*| grep maureen #no maureen left behind
[bright73->user]%
```

### 6.3 Using An External LDAP Server

Sometimes, an external LDAP server is used to serve the user database. If, instead of just using the database for authentication, the user database is also to be managed, then its LDAP schema must match the Bright Cluster Manager LDAP schema.

By default, Bright 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 Bright Cluster Manager’s `ldap` health check object that is used to monitor the LDAP service. An example of a health check object is shown on page 427.

- 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@bright73 ~]# cmsh
[bright73]# user
[bright73->user]# add ldapcheck; commit
[bright73->user[ldapcheck]]% monitoring setup healthconf headnode
[bright73->monitoring->setup*[HeadNode*]->healthconf*]# commit
[bright73->monitoring->setup[HeadNode*]->healthconf*% use ldap:ldapcheck
...*[HeadNode*]->healthconf*[ldap:ldapcheck*]]% commit
```

**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 `cmgui` or `cmsh`.

- `cmgui`: the `ldap` health check is disabled as follows: Within the Monitoring Configuration resource the Health Check Configuration tabbed pane is selected. All Head Nodes is then selected from the dropdown menu, and the `ldap` health check is selected from the list of health checks that are displayed. Clicking the edit button then opens Edit Health Check Configuration” dialog (figure 10.29), which has a Disabled checkbox. Ticking this checkbox disables the health check.

© Bright Computing, Inc.
• 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@bright73 ~]# cmsh -c "monitoring setup healthconf headnode;\nset ldap:ldapcheck 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 prior to RHEL 6: the URIs in /etc/ldap.conf, and in the image file /cm/images/default-image/etc/ldap.conf are set to point to the external LDAP server.
    
    • derived from the RHEL 6.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/pam_ldap.conf. To implement the changes, the nslcd daemon must then be restarted, for example with service nslcd restart.
    
    • derived from RHEL 7.x series: the file /etc/ldap.conf does not exist. The file in which the changes then need to be made are /etc/nslcd.conf. To implement the changes, the nslcd daemon must then be restarted, for example with service nslcd restart.
    
    - the updateprovisioners command (section 5.2.4) is run to update any other provisioners.
  
  • Then, to update configurations on the regular nodes so that they are able to do LDAP lookups:
    
    - They 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 cmgui or cmsh, then altering cmd.conf is not required, and Bright Cluster Manager’s user management capabilities do nothing in any case.
    
    - If, however, system users and groups are to be managed via cmgui 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 Bright Cluster Manager, and so it is very unlikely to match the Bright 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 Bright 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 Bright Cluster Manager cluster. The LDAP TCP ports 389 and 689 may therefore need to be made accessible between the consumer and the provider by changing firewall settings.

If a provider LDAP server is already configured then the following synchronization directives must be in the `slapd.conf` file to allow replication:

- `index entryCSN eq`
- `index entryUUID eq`
- `overlay syncprov`
- `syncprov-checkpoint <ops> <minutes>`
- `syncprov-sessionlog <size>`

The openldap documentation (http://www.openldap.org/doc/) has more on the meanings of these directives. If the values for `<ops>`, `<minutes>`, and `<size>` are not already set, typical values are:

- `syncprov-checkpoint 1000 60`
- `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 and the server can be restarted using service ldap restart, or for RHEL 7.x: service slapd restart.

External LDAP Server Replication: Configuring The Consumer(s)
The consumer is an LDAP server on a Bright 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 Bright Cluster Manager nodes including the head node(s). If the ldap.conf file does not exist, then the note on page 225 about RHEL versions 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 Bright Cluster Manager is set to its high availability configuration, with its LDAP servers running internally, on its own head nodes—then by default LDAP services are provided from both the active and the passive node. The high-availability setting ensures that CMDaemon takes care of any changes needed in the `slapd.conf` file when a head node changes state from passive to active or vice versa, and also ensures that the active head node propagates its LDAP database changes to the passive node via a `syncprov/syncrepl` configuration in `slapd.conf`.

#### External LDAP Server With No Replication Locally Case

In the case of an external LDAP server being used, but with no local replication involved, no special high-availability configuration is required. The LDAP client configuration in `/etc/ldap.conf` simply remains the same for both active and passive head nodes, pointing to the external LDAP server. The file `/cm/images/default-image/etc/ldap.conf`, in each software image also point to the same external LDAP server. If the `ldap.conf` files referred to here in the head and software images do not exist, then the note on page 225 about RHEL versions 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 225 about RHEL versions applies.

2. For high availability configuration done after LDAP replication configuration, the initial LDAP configurations and database are propagated to the passive node. To set replication to the passive node from the active node, and not to the passive node from an external server, the provider option in the `syncrepl` directive on the passive node must be changed to point to the active node, and the suffix in `/cm/local/apps/openldap/etc/slapd.conf` on the passive node must be set identical to the head node.

The high availability replication event occurs once only for configuration and database files in Bright 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 cmgui 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 Authorization resource of cmgui.

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>Bright Cluster Manager PAM tasks</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 using tab-completion prompting, as follows:

Example

[root@bright73 ~]# cmsh
[bright73]% profile
[bright73->profile]% show <TAB> <TAB>
admin cloudjob cmhealth cmpam node portal power readonly

The tokens, and other properties of a particular profile can be seen with cmsh within profile mode as follows:

Example

[bright73->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
A profile can be set with \texttt{cmsh} for a user within \texttt{user} mode as follows:

\textbf{Example}

\begin{verbatim}
[root@bright73 ~]# cmsh
[bright73]% user use conner
[bright73->user[conner]]% get profile

[bright73->user[conner]]% set profile readonly; commit
\end{verbatim}

\subsection{6.4.1 Modifying Profiles}

A profile can be modified by adding or removing appropriate tokens to it. For example, the \texttt{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 \texttt{cmsh} the removal can be done from within \texttt{profile} mode as follows:

\begin{verbatim}
[root@bright73 ~]# cmsh
[bright73]% profile use readonly
[...[readonly]]% removefrom tokens burn_status_token get_burn_log_token
[bright73]%->profile*[readonly*]% commit
\end{verbatim}

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

In \texttt{cmgui} (figure 6.3), the removal can be done by selecting the \textit{Authorization} item from the resources tree. In the display pane, the \texttt{BURN_STATUS_TOKEN} and \texttt{GET_BURN_LOG_TOKEN} can have their ticks removed from their checkboxes in the \texttt{readonly profile group} (in the \texttt{CMDevice subgroup}). The changed settings can then be saved.

\subsection{6.4.2 Creation Of Custom Certificates With Profiles, For Users Managed By Bright Cluster Manager's Internal LDAP}

Custom profiles can be created to include a custom collection of capabilities in \texttt{cmsh} and \texttt{cmgui}. Cloning of profiles is also possible from \texttt{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 \texttt{cmsh} (page 232) or \texttt{cmgui} (page 234) is described later on. After creating such a certificate, the \texttt{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:

\begin{verbatim}
[root@bright73]# openssl x509 -in peterfile.pem -text -noout
\end{verbatim}

\begin{verbatim}
Data:  
   Not After : Sep 21 13:18:27 2014 GMT
   Subject: ... CN=democert
   Public-Key: (2048 bit)
\end{verbatim}

\begin{verbatim}
X509v3 extensions:
\end{verbatim}
However, using the openssl utility for managing certificates is rather inconvenient. Bright 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@bright73 ~]# cmsh -c "cert; listcertificates"

Serial num Days left Profile Country Name Revoked
---------- ---------- -------- -------- ---------------
   1       36451   admin   US     Administrator      Yes
   10       36451 readonly ef     democert      Yes
   11       36496   node     NL      52-54-00-de-e3-6b     No
   12       36496   node     NL      52-54-00-44-fb-85     No
   13       36496   admin     UK      otheradmin            No

...  
```

**Listing certificates with cmgui:** The cmgui equivalent for listing certificates is via the Authentication resource, in the Certificates tabbed pane (figure 6.4):

---
© Bright Computing, Inc.
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
- `cmgui`: within the `Users & Groups` resource, within the `Users` tabbed pane, using the Add dialog to set the Management profile.

Creating a new certificate for `cmsh` users: Creating a new certificate in `cmsh` is done from `cert` mode using the `createcertificate` command, which has the following help text:

```
[bright73->cert]% help createcertificate
Name:
createcertificate - Create a new certificate

Usage:
createcertificate <key-length> <common-name> <organization> <organizational-unit> <locality> <state> <country> <profile> <sys-login> <days> <key-file> <cert-file>

Arguments:
key-file
    Path to key file that will be generated

cert-file
    Path to pem file that will be generated

Accordingly, as an example, a certificate file with a read-only profile set to expire in 30 days, to be run with the privileges of user `peter`, can be created with:

Example

createcertificate 2048 democert a b c d e f readonly peter 30 /home/peter\ /peterfile.key /home/peter/peterfile.pem
```
Fri Aug 22 06:18:27 2014 [notice] bright73: New certificate request with ID: 1
[bright73->cert]% createcertificate 2048 democert a b c d ef readonly p
eter 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 root was run-
ing cmsh. This means that user peter cannot use them until their ownership is changed to user peter:

Example

[root@bright73 ~]# cd /home/peter
[root@bright73 peter]# ls -l peterfile.*
-rw------- 1 root root 1704 Aug 22 06:18 peterfile.key
-rw------- 1 root root 1107 Aug 22 06:18 peterfile.pem
[root@bright73 peter]# chown peter:peter peterfile.*

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@bright73 ~] cmsh -i peterfile.pem -k peterfile.key

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}
   (c) .cm/cmsh/admin.{pem,key}
   (d) .cm/cmsh/cert.{pem,key}

   For example, putting the key pair in the path in (a) overrides any key pairs placed in (b), (c), or
   (d). The order is based on the path with the lowest directory depth having a higher priority, then
   the path that is earlier in alphabetical order having a higher priority. Other than priority, there is
   nothing special about choosing a prefix of admin.{key,cert} instead of cert.{key,cert}.

   For example, continuing the preceding session, here is how the user peter can have his default
   certificate set in his home directory:

   Example

   [peter@bright73 ~]$ mkdir -p /home/peter/.cm/cmsh
   [peter@bright73 ~]$ mv peterfile.* /home/peter/.cm/cmsh/
   [peter@bright73 cmsh]$ cd /home/peter/.cm/cmsh
   [peter@bright73 cmsh]$ mv peterfile.key admin.key
   [peter@bright73 cmsh]$ mv peterfile.pem admin.pem
   [peter@bright73 cmsh]$ cd /home/peter
   [peter@bright73 cmsh]$ cmsh #now uses custom certificate by default
   [bright73]%
Creating a custom certificate for cmgui users: As in the case of cmsh, a cmgui 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 Management profile from the Add or Edit dialog for the user (figure 6.2).

The certificate files, cert.pem and cert.key, can then be copied over to the following paths and names, under $HOME:

- .cm/admin.{pem,key}
- .cm/cert.{pem,key}
- .cm/cmgui/admin.{pem,key}
- .cm/cmgui/cert.{pem,key}

Users that authenticate with their user name and password when running cmgui use this certificate for their cmgui clients, and are then restricted to the set of tasks allowed by their associated profile. Certificates are given priority according to the shallowest directory depth order first, then alphabetical order next. This is analogous to the logic of item 2 for cmsh keys.

### 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 Bright 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 Bright Cluster Manager. The external-user-cert.py script, available under /cm/local/apps/cmd/scripts/, does this, embedding the user and profile in the certificate during the process. It has the following usage:

Usage:

For a single profile:

```
external-user-cert.py <profile> <user> [<user> ...] --home=<home-prefix> [-g <group>] [-p <port>]
```

For several profiles:

```
external-user-cert.py --home=<home-prefix> --file=<inputfile> [-g <group>] [-p <port>]
```

where lines of <inputfile> have the syntax

```
<profile> <user> [<user> ...]
```

The -p <port> option must be set if CMDaemon is using a non-standard SSL port.

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
- <port> is a port number, such as the CMDaemon SSL port 8081

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.py readonly spongebob --home=/home
```
If the home directory of spongebob is /home/spongebob, then the key files that are generated are /home/spongebob/.cm/cert.key and /home/spongebob/.cm/cert.pem.

Assuming no other keys are used by cmsh, a cmsh session that runs as user spongebob with readonly privileges can now be launched:

```
$ module load cmsh
$ cmsh
```

If other keys do exist, then they may be used according to the logic explained in item 2 on page 233.

### 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 management 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 workload manager is used, the user submits a batch (i.e. non-interactive) job to it. The workload manager 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 workload manager then schedules and executes the job based on the assignment criteria that the administrator has set for the workload management 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 7.5 of the Installation Manual.

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 workload manager up and running. Sections 7.6 –7.7 describe how cmgui and cmsh are used to view and handle jobs, queues and node drainage.

Section 7.8 shows examples of workload manager assignments handled by Bright Cluster Manager. Section 7.9 ends the chapter by describing the power saving features of workload managers.

7.1 Workload Managers Choices

Some workload manager packages are installed by default, others require registration from the distributor before installation.

During cluster installation, a workload manager can be chosen (figure 3.22 of the Installation Manual) for setting up. The choices are:

- None
- Slurm v16.05.2 (default)
- Grid Engine 2011.11p1. An open source development of Sun Grid Engine (SGE)
  A flavor of SGE is Univa Grid Engine (UGE)(section 7.5.2) developed by Univa. Bright Cluster Manager 7.3 supports integration with UGE versions 8.2.0 and higher at the time of writing (May 2015).
- Torque v6.0.2 and its built-in scheduler
- Torque v6.0.2 and the Maui scheduler
- Torque v6.0.2 and the Moab scheduler
• PBS Pro v13.1.0

These workload managers can also be chosen and set up later using the wlm-setup tool (section 7.3).

Besides the preceding workload managers, the installation of the following workload managers is also possible, and described in their own sections:

• OpenLava v4.0.0 (section 7.5.6)
• Load Sharing Facility (LSF) v9, v10 (section 7.5.7)

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 workload management system. Having jobs run via a workload manager is normally a best practice.

For convenience, the Bright Cluster Manager defaults to allowing users to login 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 outside the workload management system without restriction. 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 workload management system, means that jobs are then distributed to the nodes only according to the planning of the workload manager. If planning is based on sensible assignment criteria, then resources use is optimized—which is the entire aim of a workload management system 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 workload manager, 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@bright73 ~]# cmsh
[bright73]% category use default
[bright73->category[default]]% set usernodelogin onlywhenjob
[bright73->category+[default*]]% commit

The attributes for usernodelogin are:

• always (the default): This allows all users to ssh directly into a node at any time.
• never: This allows no user other than root to directly ssh into the node.
• onlywhenjob: This allows the user to ssh directly into the node when a job is running on it. It typically also prevents other users from doing a direct ssh into the same node during the job run, since typically the workload manager 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.

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 button in cmgui (section 5.6.3).

7.2.2 Disallowing User Logins To Regular Nodes Via cmgui

In cmgui, user node login access is set from the Settings tab for a category selected in the Node Categories resource, in the section labeled “User node login” (figure 7.1):

![Figure 7.1: Disallowing User Logins To Nodes Via cmgui](image)

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 workload management system 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 workload manager, as part of a policy, or for general administrative hygiene. These are Epilog scripts and are part of the workload manager.

The workload management system documentation has more on configuring these options.

7.3 Installation Of Workload Managers

Normally the administrator selects a workload manager to be used during Bright Cluster Manager installation (figure 3.22 of the Installation Manual). A workload manager may however also be added and configured after Bright Cluster Manager has been installed, using wlm-setup.

7.3.1 Setting Up, Enabling, And Disabling The Workload Manager With wlm-setup

The following options from wlm-setup must be chosen to set up, enable or disable a workload manager for Bright Cluster Manager:

- `-w` or `--wlmanager`: the workload manager option, together with `<name>`, the workload manager name. The value of `<name>` can be one of
  - slurm
· sge
· uge (This has a prerequisite that the packages must be installed from the UGE site first, as explained in section 7.5.3)
· torque (Torque with its built-in scheduler)
· torque-maui (Torque with the Maui scheduler)
· torque-moab (Torque with the Moab scheduler)
· pbspro (PBS Pro commercial edition)
· pbspro-ce (PBS Pro Community Edition)
· lsf
· openlava (This has a prerequisite that the packages must be installed first, as explained in section 7.5.6)

and for the named workload manager, either:

- **-s or --setup**: the setup option. This does a basic setup of the workload manager, initializing the database for the workload manager, setting up default queues, and enabling the workload manager clients for the default category. It can optionally take the further options of section 7.3.2 or

- **-e or --enable**: the enable option. Enabling assigns workload manager client roles and default queues to the applicable node categories. This option need not be run if the setup option has just run. CMDaemon attempts to run workload managers that have been enabled.

- **-p or --powersave** option will also allow power management, for slurm only, for Slurm clients only, for non-head nodes only. Independently of wlm-setup, power management can also be allowed by using Slurm roles (section 7.9.1) without using wlm-setup. When using Slurm roles, the powersavingallowed parameter in the slurmclient role allows power management for head nodes too if set for a head node, in the unlikely case that the administrator finds it is acceptable to power off a head node that is not running Slurm jobs.

If power saving is allowed for a node, then the powersavingenabled parameter in the slurmserver role is what actually enables power management for that node.

or

- **-d or --disable**: the disable option, which disables the workload manager

- **-p or --powersave** option will disallow power management only, for slurm only, for Slurm clients only.

or

- **-i or --image**: the image option, which places the job execution daemons in software images, for example when a new node category is added to the cluster. The default image directory is /cm/images/default-image.

The --setup option of wlm-setup installs a workload manager package along with its scheduler. Roles, queues, and databases used by the workload manager are initialized on the head. Software images stored on the head node are also set up. However, the nodes themselves are only updated after a reboot, or manually running imageupdate (section 5.6.2)—the idea being to avoid an automatic update so that an administrator is encouraged to check via a dry-run if unintended consequences would happen.

The enable and disable options of wlm-setup are the same as those that take place during role assignment (sections 7.4.1 and 7.4.2). These options therefore make CMDaemon enable or disable the workload manager as a service.

For example, setting up SGE can be done as follows:
Example

[root@bright73 ~]# wlm-setup -w sge -s

Disabling sge services ..... [ OK ]
Initializing sge setup ..... [ OK ]
Installing sge qmaster ..... [ OK ]
Updating image services ..... [ OK ]
Updating qmaster config ..... [ OK ]
Creating default sge setup ..... [ OK ]
Setting permissions ..... [ OK ]
Enabling sge services ..... [ OK ]

For example, Slurm job daemons can be installed in the node image new-image as follows:

Example

wlm-setup -w slurm -i /cm/images/new-image

If there are provisioning nodes, the updateprovisioners command (section 5.2.4) should be run after the software image is changed. The nodes can then simply be rebooted to pick up the new image, or alternatively, to avoid rebooting, the imageupdate command (section 5.6.2) places a new image on the node from the provisioner.

7.3.2 Other Options With wlm-setup

Other options exist for wlm-setup, which are run as further options to the -s|--setup option. These are:

- **included images**
  The included images option (-u or --updateimages <image1[,image2, ...]> includes the comma-separated list of images

- **excluded images**
  The excluded images option (-x or --excludeimages <image1[,image2, ...]> excludes the comma-separated list of images

- **offload**
  The offload option (-o or --offload) deals with setting up, enabling or disabling a workload manager server running on another specified node, other than the default head node:

Example

wlm-setup -w slurm -s -o node003

- **head as a compute node**
  This option (-m or --mcompute) sets the head node to join in as a compute node for job execution tasks in a workload management system. This can be significantly worthwhile on smaller clusters:

Example

wlm-setup -w slurm -s -m

- **slots**
  The slots option (-n or --slots) sets the maximum allowed value of slots, and is typically set to the number of cores per node:

Example
wlm-setup -w torque -s -n 4

For workload managers, slots is the number of logical CPUs set per node.

- It is not recommended to set the number of slots to be greater than the number of cores for non-threading applications. That is because in such a case, when all the slots are running jobs, the maximum speed at which the jobs can run is usually reduced in comparison with the one-slot-per-core case, and the efficiency at which they run is also less.

- However increasing the slots value to be greater than the number of cores may improve performance for applications that are optimized to use threading, if they are running on cores that have hyperthreading enabled. Because enabling hyperthreading can increase power consumption and can result in unexpected performance issues, benchmarking is advised.

The number of slots can also be adjusted outside of `wlm-setup`, after setup, by using Bright Cluster Manager to set it in the workload manager client role.

**SGE and slots:** In SGE, `wlm-setup` sets the number of slots in a complex (`man (5) complex`) configuration file.

In practice, for SGE this means that the system administrator manually sets the number of slots to the number of cores per node during the initial Bright Cluster Manager installation, or during `wlm-setup`, or using `cmsh` or `cmgui`, just like in the other workload managers. The complex configuration file value is set automatically according to this value.

If an end user explicitly requests slots for a job `myjob` with:

**Example**

```bash
qsub -l slots=2 myjob
```

then SGE tries to allocate a node with 2 free slots. If there are no nodes with more than one slot, then `myjob` will not run.

The difference in comparison with the other workload managers occurs when a user needs to use a parallel engine driver such as Open MPI. In that case, the requested number of slots in SGE can be used by MPI processes as well as regular non-MPI processes. Thus, the user submission request:

**Example**

```bash
qsub -l slots=2 -pe openmpi 32 myjob
```

means that enough nodes with 2 free slots each should be made available to run all the 32 MPI processes being requested. Again, if there are no nodes with more than one slot, the job will not run. If `slots=2` is not specified in the `qsub` line, then the MPI processes are spread across whatever cores are available, if there are nodes with one or more free slots.

- **archives** The archives option (-a or `--archives-location`) sets the directory path to the archives files used for software installation. This is valid for UGE or LSF only.

**Example**

```bash
wlm-setup -w lsf -s -a /root/lsf
```

- **categories** The categories option (-c or `--categories`) assigns the client role to a comma-separated list of categories instead of to the default category. The default category takes the value `default` by default. The default value for a category can be changed to another existing category by setting the value for the `defaultcategory` attribute for the `base` object in `partition` mode:
Example

[bright73]\% partition use base
[bright73->partition[base]]\% get defaultcategory
default
[bright73->partition[base]]\% set defaultcategory fast; commit

The \textit{-c|--categories} option to \texttt{wlm-setup} is available from the 7.3-9 release onwards, as part of the \texttt{cluster-tools} package r9918.

Example

\texttt{wlm-setup \texttt{-w slurm \texttt{-s \texttt{-c fast,superfast,ludicrous}}}

7.3.3 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 SGE, LSF, or OpenLava, by setting such scripts in the \texttt{cmgui} or \texttt{cmsh} front ends. Default epilogs and prologs are created automatically when a new queue is created.

Example

[bright73->jobqueue]\% add lsf newq
[bright73->jobqueue\{lsf\}->newq\} show | grep . | grep -i epilog
Epilog /cm/local/apps/cmd/scripts/epilog
Prolog/Epilog user root

For Torque/PBS Pro 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 edit the scripts directly for their own needs, outside of the \texttt{cmgui} or \texttt{cmsh} front ends. A more detailed explanation of how the prolog scripts work therefore follows:

The prolog scripts have names and function according to their locations. The scripts are placed in two directories:

1. In the \texttt{main scripts directory}, at:

   /cm/local/apps/cmd/scripts/

   In this directory, a main prolog script, \texttt{prolog}, can call a sequence of \texttt{rc.d}-style prolog scripts for a particular workload manager in an \texttt{rc.d}-style directory.
2. rc.d-style prolog directory, at:

/cm/local/apps/<workload manager>/var/prologs/

In this directory, prolog scripts, if they exist, are stored in a directory path associated with the workload manager name. The names of the scripts have suffixes and prefixes associated with them that make them run in special ways, as follows:

- **suffixes used in the rc.d-style directory:**
  - `-prejob` script runs prior to all jobs
  - `-cmsub`: script runs prior to job run in a cloud
  - `-mic`: script runs prior to job run in a MIC
  - `-michost`: script runs prior to job started on a MIC host

- **prefixes used in the rc.d-style directory:**
  - `00-` to `99-`
    
    Number prefixes determine the order of script execution. This is like for SysV-style rc.d names, where scripts with the lower number are run earlier. Hence the terminology “rc.d-style” associated with these prolog scripts.

The script names can therefore look like:

**Example**

- `00-prolog-prejob`
- `10-prolog-cmsub`
- `15-prolog-mic`

Return values for the rc.d-style 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 rc.d-style directory.

Often, the script in an rc.d-style prolog directory is not a real script but a symlink, with the symlink going to a general script located in the main scripts directory. In that case, this general script is then able to take care of what is expected of the symlink. The name of the symlink, and destination file, usually hints at what the script is expected to do.

For example, the Torque workload manager uses the symlink `10-prolog-prejob` within the rc.d-style directory `/cm/local/apps/torque/var/prologs/`. The symlink links to the script `prolog-prejob` within the main scripts directory `/cm/local/apps/cmd/scripts/`. In this case, the script is expected to run prior to the job.

Epilog script paths follow the same pattern as prolog scripts, with “epilog” substituted in the path name for “prolog”. However, the only suffix form allowed in the rc.d-style directory is `-cmsub`. The Torque/PBS Pro and Slurm prolog and epilog scripts are global in effect.

© Bright Computing, Inc.
7.4 Enabling, Disabling, And Monitoring Workload Managers

Workload Manager Package Configuration For Prolog And Epilog Scripts

Each workload manager package configures prolog- and epilog-related scripts or links during installation, as follows:

- **Slurm**
  - prolog-prejob: in the main scripts directory, is assigned to PrologSlurmctld in \(/\text{etc}/\text{slurm}/\text{slurm.conf}\). It runs by default during job execution, and is executed with slurm user permissions.
  - prolog: in the main scripts directory, is assigned to the variable Prolog in \(/\text{etc}/\text{slurm}/\text{slurm.conf}\). The script executes the \(<\text{number}>\)-prolog\{-\text{cmsub}|\text{mic}|\text{michost}|\text{prejob}\} scripts located in the rc.d-style directory, if they exist. By default, none exist. The epilog script in the main scripts directory follows the same pattern, with the appropriate name changes.
  - slurm.epilog.clean: in the \(/\text{etc}/\text{slurm}/\) directory, is an example epilog script. To enable it, it can be placed in the rc.d-style \(/\text{cm/local/apps/slurm/var/epilogs/}\) directory and given a name with a suitable prefix and suffix.

- **SGE, LSF/OpenLava**
  - prolog: in the main scripts directory, executes any \(<\text{number}>\)-prolog\{-\text{cmsub}|\text{mic}|\text{michost}|\text{prejob}\} scripts located in the rc.d-style directory, if they exist. It is set by default to execute 10-prolog-prejob in the rc.d-style directory. This in turn is configured as a symlink to prolog-prejob in the main scripts directory, which is able to handle the execution of scripts in a general manner. Any further scripts in the rc.d-style directories, whether for prologs or epilogs, are then also processed.

- **PBS Pro/Torque**
  - mom_priv/prologue: (with the -ue ending) in the rc.d-style directory, is a symlink to prolog in the main scripts directory. This in turn executes any \(<\text{number}>\)-prolog\{-\text{cmsub}|\text{mic}|\text{michost}|\text{prejob}\} scripts located in the rc.d-style directory, if they exist. Similar to this are the epilogue scripts in the rc.d-style directory.

7.4 Enabling, Disabling, And Monitoring Workload Managers

After a workload manager package is installed and initialized with \texttt{wlm-setup} (section 7.3), it can also be enabled or (if already enabled) disabled, with \texttt{wlm-setup}. Enabling and disabling means the workload management services start or stop.

Alternatively, a workload manager can be enabled or disabled by the administrator with \texttt{cmgui} or \texttt{cmsh}. This is described further on in this section.

In Bright Cluster Manager 7.3, workload managers can even run concurrently. For example, Slurm, SGE, and Torque can run at the same time in the cluster, for example with one workload manager assigned to one category. However, this can easily lead to confusion. Running only one workload manager is a more sane practice, and is strongly urged for production environments.

From the \texttt{cmgui} or \texttt{cmsh} point of view a workload manager consists of

- a workload manager server, usually on the head node
- workload manager 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 workload manager server or client role on the head or compute nodes, as deemed appropriate.

© Bright Computing, Inc.
The administrator typically also sets up an appropriate workload manager environment module (slurm, sge, torque, pbspro, or openlava) so that it is loaded up for the end user (section 2.2.3).

7.4.1 Enabling And Disabling A Workload Manager With cmgui

A particular workload manager package may be set up, but the workload manager may not be enabled. This can happen, for example, if using wlm-setup (section 7.3) to install the package without enabling it. It may also happen, for example, if disabling a workload manager that was previously enabled.

The workload manager client and server can be enabled from cmgui by assigning the appropriate role to it. Within the role, the properties of the workload manager may be further configured.

Workload Manager Role Assignment To An Individual Node With cmgui

Workload Manager Server

Enabling the server on a node can be done by clicking on the “Head Nodes” or Nodes folder, selecting the node item, and selecting the Roles tab to display the possible roles. A workload manager server role is then chosen and its options set. For example, for Torque, the Maui or Moab schedulers must be set as options instead of Torque’s built-in scheduler, when enabling Torque with Maui or Moab. The workload manager server role is then saved with the selected options (figure 7.2). 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.

Figure 7.2: Workload Management Role Assignment On A Head Node

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 in the role. Some basic options can be set for the client role right away in the tab (figure 7.3), and some advanced options can be set by clicking on the Advanced button for that role (section 7.5.1, figure 7.6).

Saving the Roles pane, and then running imageupdate (section 5.6.2), automatically starts up the client process with the options chosen, and managed by CMDaemon.
While workload manager role assignment can be done as described in the preceding text for individual non-head nodes, it is usually more efficient to assign roles using categories due to the large number of compute nodes in typical clusters.

All non-head nodes are by default placed in the default category. This means that by default roles in the category are automatically assigned to all non-head nodes, unless by way of exception an individual node configuration overrides the category setting and uses its own role setting instead.

Viewing the possible workload manager roles for the category default is done by clicking on the “Node Categories” folder, selecting the default category, and selecting the Roles tab. The appropriate workload manager role is then configured.

For compute nodes, the role assigned is usually a workload manager client. The assigned role for a compute node allows queues and GPUs to be specified, and other parameters depending on the workload manager used.

The workload manager server role can also be assigned to a non-head node. For example, a Torque server role can be carried out by a non-head node. This is the equivalent to the offload option of wlm-setup. As in the individual node assignment case, for Torque with Maui or Torque with Moab, the Maui scheduler or the Moab scheduler options must be set if these schedulers are to be used.

Saving the roles with their options and then running imageupdate (section 5.6.2) automatically starts up the newly-configured workload manager.

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 Bright Cluster Manager to:
  - the CPUs setting (a NodeName parameter) in Slurm’s slurm.conf
  - the np setting in Torque and PBS Pro,

and is normally set to the number of cores per node.

In LSF and OpenLava, 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 cmgui) and 7.7.2 (using cmsh).
Workload Manager Client Advanced Role Options With \texttt{cmgui}

The roles tabbed pane displays advanced role options for Workload manager clients when the \texttt{Advanced} button for the client role (figure 7.3) is clicked. These advanced options, as well as the basic options described previously, are managed by CMDaemon and should not be configured separately outside of CMDaemon control. For other options, changes beyond the scope of Bright Cluster Manager are required. Such changes are then typically managed by setting the options in a workload manager configuration file, and freezing the configuration. Freezing a configuration is done using a CMDaemon directive of the form \texttt{FreezeChangesTo<workloadmanager>Config} (page 595).

Workload Manager Server Basic Role Options With \texttt{cmgui}

Various schedulers can be selected from a menu when configuring the Slurm and Torque workload managers in the basic server role options.

When the checkbox for the role is ticked, an \texttt{Advanced} button appears in the basic workload manager server role options. More role options are shown when the \texttt{Advanced} button is clicked.

Workload Manager Server Advanced Role Options With \texttt{cmgui}

- Most of the options displayed on clicking the \texttt{Advanced} button are installation path or spool path settings.

- All server roles also provide the option to toggle the \texttt{External Server} checkbox. A ticked checkbox means that the server is no longer managed by Bright Cluster Manager, but provided by an external device.

An external Torque server role on a ticked node also means that additional scheduler services—Maui or Moab—are not started, nor are they monitored, on that node. For the ticked node, using the head node can still be a practical and convenient option for an external server.

The \texttt{cmsh} equivalent of this is described on page 251.

Overriding Category Settings Per Node With \texttt{cmgui}

If the role for the individual non-head node is set and saved then it overrides its corresponding category role. In \texttt{cmgui} this is done by selecting the particular node device from the \texttt{Nodes} folder, then selecting the \texttt{Roles} tab. The appropriate workload manager client role can then be configured (figure 7.4).

A useful feature of \texttt{cmgui} is that the role displayed for the individual node can be toggled between the category setting and the individual setting by clicking on the role checkbox (figure 7.5). Clicking on the \texttt{Save} button of the tabbed pane saves the displayed setting.

![Figure 7.5: Workload Management Role Assignment Toggle States For An Individual Node](image)

7.4.2 Enabling And Disabling A Workload Manager With \texttt{cmsh}

A particular workload manager package may be set up, but not enabled. This can happen, for example, if using \texttt{wlm-setup} (section 7.3) on the package without enabling it. The workload manager client and server can be enabled from \texttt{cmsh} by assigning it from within the \texttt{roles} submode. Within the assigned role, the properties of the workload manager may be further configured.
7.4 Enabling, Disabling, And Monitoring Workload Managers

Workload Manager Role Assignment To A Category With cmsh

Workload manager role assignment of a node category is done using category mode, using the category name, and assigning a role from the roles submode:

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# category
[bright73->category]# use default
[bright73->category[default]]# roles
[bright73->category[default]->roles]# assign torqueclient
[bright73->category[default]->roles[torqueclient]]# commit
[bright73->category[default]->roles[torqueclient]]#
```

After 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.

Workload Manager Role Assignment To An Individual Node With cmsh

In cmsh, assigning a workload manager role to a head node is done in device mode, using master as the device, and assigning the workload manager role from the roles submode:

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# device
[bright73->device]# use bright73
[bright73->device[bright73]]# roles
[bright73->device[bright73]->roles]# assign torqueserver
```

© Bright Computing, Inc.

Figure 7.4: Workload Management Role Assignment For An Individual Node

![Image](image.png)
For regular nodes, role assignment is done via device mode, using the node name, and assigning a role from the roles submode:

Example

```
[root@bright73 ~]# cmsh
[bright73]% device
[bright73]->device% use node001
[bright73]->device[node001]% roles
[bright73]->device[node001]->roles% assign torqueclient
[bright73]->device[node001]->roles[torqueclient]% commit
[bright73]->device[node001]->roles[torqueclient]%
```

Role assignment values set in device mode have precedence over any role assignment values set in category mode for that node. This means, for example, that if a node is originally in a node category with a torqueclient role and queues set, then when the node is assigned a torqueclient role from device mode, its queue properties are empty by default.

Setting Options For Workload Manager Settings With cmsh

In the preceding text, it is explained how the workload manager client or server is assigned a role (such as torqueclient or torqueserver) within the roles submode. It is done from within a main mode of category or devices.

Options for workload managers in general: Whatever the main mode used, the workload manager settings can then be handled with the usual object commands introduced in section 2.5.3. For example, the number of slots can be set for Torque clients as follows:

Example

```
[bright73]->category[default]->roles[torqueclient]% show
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Queues</td>
<td>yes</td>
</tr>
<tr>
<td>GPUs</td>
<td>0</td>
</tr>
<tr>
<td>Name</td>
<td>torqueclient</td>
</tr>
<tr>
<td>Queues</td>
<td>shortq longq</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Slots</td>
<td>4</td>
</tr>
<tr>
<td>Type</td>
<td>TorqueClientRole</td>
</tr>
</tbody>
</table>
```

```
[bright73]->category[default]->roles[torqueclient]% set slots 5
[bright73]->category[default]->roles[torqueclient]% commit
```

Options for the Torque variants: In particular, the Scheduler parameter can be set from an assigned torqueserver role:

Example

```
[bright73]->device[bright73]->roles% set torqueserver scheduler backfill builtin maul moab torque
[bright73]->device[bright73]->roles% use torqueserver
[bright73]->device[bright73]->roles[torqueserver]% show
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
</table>
```
## 7.4 Enabling, Disabling, And Monitoring Workload Managers

In the preceding example, the available schedulers list can be displayed using tab-completion, and the Moab scheduler is then enabled. Running `imageupdate` (section 5.6.2) for non-head nodes ensures that the built-in Torque scheduler stops and the Moab scheduler starts. It is possible to switch to the Maui scheduler with the same approach. The Moab or Maui schedulers do however need to be installed before starting them in this manner.

In cmgui, when the Torque server is selected as a role for the node, a scheduler service can then be selected from a drop-down menu. Selecting a new scheduler and saving it enables the new scheduler. Running `imageupdate` (section 5.6.2) for non-head nodes stops the previous scheduler and starts the newly-selected and enabled scheduler.

### 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**

```
[bright73->device[bright73]->roles[torqueserver]]% set externalsearcher on
[bright73->device[bright73]->roles[torqueserver*]]% commit
```

For convenience, setting it on the head node is recommended.

If an external server is set for the `torqueserver` role, and uses the schedulers Maui or Moab, then these scheduler services are not started or monitored on the device. The device is the head node, if the earlier recommendation for convenience is followed. The default setting of `torque` remains the scheduler.

The cmgui equivalent of configuring `externalsearcher` is described on page 248.

### 7.4.3 Monitoring The Workload Manager Services

By default, the workload manager services are monitored. The Bright 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 (using `cmsh` or cmgui). Workload manager roles and corresponding services can be disabled using `wlm-setup` (section 7.3.1), cmgui (section 7.4.1), or cmsh (section 7.4.2).

The daemon service states can be viewed for each node via the shell, `cmsh`, or cmgui (section 3.11). Queue submission and scheduling daemons normally run on the head node. From cmgui their states are viewable by clicking on the node folder in the resources tree, then on the node name item, and selecting the Services tab (figures 3.17 and 10.5).

The job execution daemons run on compute nodes. Their states are viewable by clicking on the Nodes folder, then on the node name item, and selecting the Services tab.

From cmsh the services states are viewable from within device mode, using the `services` command. One-liners from the shell to illustrate this are (output elided):

**Example**

```
[root@bright73 ~]# cmsh -c "device services node001; status"
sgeexecd[ UP ]
[root@bright73 ~]# cmsh -c "device services bright73; status"
...  
sge[ UP ]
```
7.5 Configuring And Running Individual Workload Managers

Bright 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 Bright Cluster Manager should normally not be changed directly because Bright Cluster Manager overwrites them. However, overwriting by CMDaemon is prevented on setting the directive:

\[
\text{FreezeChangesTo<workload manager>Config = <true|false>}
\]

in `cmd.conf` (Appendix C), where `<workload manager>` takes the value of Slurm, SGE, Torque, PBSPro, or OpenLava, 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 Bright Cluster Manager 7.3 system is given in Appendix F.

7.5.1 Configuring And Running Slurm

Configuring Slurm

After package setup is done with `wlm-setup` (section 7.3), Slurm software components are installed in `/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). For example, the advanced options can be set in `cmgui` (figure 7.6).

![Advanced Role Settings](image)

Figure 7.6: Workload Manager Role Assignment By Node For A Compute Node—Dialog Example For Advanced Options, For Slurm

Using `cmsh`, advanced option parameters can be set under the `slurmclient` role:
For example, the number of cores per socket can be set:

**Example**

```
[bright73->category[default]->roles[slurmclient]]% set corespersocket 2
[bright73->category[default]->roles[slurmclient]]% commit
```

or some GPUs can be added to Slurm:

**Example**

```
[bright73->category[default]->roles[slurmclient]]% set gpus 2
[bright73->...->roles[slurmclient]]% append gpudevices /dev/nvidia0
[bright73->...->roles[slurmclient]]% append gpudevices /dev/nvidia1
[bright73->category[default]->roles[slurmclient]]% commit
```

The option parameter values are used unless they have the value: 0.

A parameter value of 0 means that the default values of Slurm are used. These usually have the value: 1.

The advanced options that CMDaemon manages for Slurm are:

<table>
<thead>
<tr>
<th>cmgui 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 slurm.conf</td>
<td>Arbitrary strings can be entered to indicate some characteristics of a node, one string per entry. For example: text1 text2 and so on. These become part of the: Feature=text1,text2... attribute to the NodeName=&lt;node name&gt; entry line in slurm.conf, as indicated in man(5) slurm.conf. The strings also become added attributes to the GresTypes entry of that file. Default: blank.</td>
</tr>
<tr>
<td>GPU devices</td>
<td>File=&lt;device&gt; entries in the file gres.conf</td>
<td>This is only implemented if the number of GPUs has been set by the administrator using Bright Cluster Manager, by setting the basic Slurm client option GPUs. The GPUs option is seen, for example, in cmgui, in figures 7.4, and 7.5. Each device detected on the host node can be added as an entry. For example: /dev/nvidia0 /dev/nvidia1 and so on. These become part of the: Name=gpu File=&lt;device&gt; lines in gres.conf, as indicated in man(5) gres.conf. Default: blank</td>
</tr>
<tr>
<td>cmgui Option</td>
<td>Slurm Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Slots</td>
<td>CPU</td>
<td>Number of logical processors on the node. Default: 1</td>
</tr>
<tr>
<td>Sockets</td>
<td>Sockets</td>
<td>Processor chips on node. Default: 1</td>
</tr>
<tr>
<td>Cores per socket</td>
<td>CoresPerSocket</td>
<td>Number of cores per socket. Default: 1</td>
</tr>
<tr>
<td>Threads per core</td>
<td>ThreadsPerCore</td>
<td>Number of logical threads for a single core. Default: 1</td>
</tr>
<tr>
<td>Boards</td>
<td>Boards</td>
<td>Number of baseboards in a node. Default: 1</td>
</tr>
<tr>
<td>Sockets per board</td>
<td>SocketsPerBoard</td>
<td>Number of processor chips on baseboard. Default: 1</td>
</tr>
<tr>
<td>Real memory</td>
<td>RealMemory</td>
<td>Size of real memory on the node, MB. Default: 1</td>
</tr>
<tr>
<td>Node hostname</td>
<td>NodeHostname</td>
<td>Default: as defined by Slurm’s NodeName parameter.</td>
</tr>
<tr>
<td>Node address</td>
<td>NodeAddr</td>
<td>Default: as set by Slurm’s NodeHostname parameter.</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
<td>State of the node with user jobs. Possible Slurm values are: DOWN, DRAIN, FAIL, FAILING, and UNKNOWN. Default: UNKNOWN</td>
</tr>
<tr>
<td>Weight</td>
<td>Weight</td>
<td>The priority of the node for scheduling. Default: 1</td>
</tr>
<tr>
<td>TCP port</td>
<td>Port</td>
<td>Port that slurmd listens to on the compute node. Default: as defined by Slurm’s Port parameter. If Slurm’s Port is not specified during build: Default: 6818</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 /usr/share/doc/slurm-16.05.2/html.

Slurm is set up with reasonable defaults, but administrators familiar with Slurm can reconfigure the configuration file /cm/shared/apps/slurm/var/etc/slurm.conf using the JavaScript-based configuration generator in /usr/share/doc/slurm-16.05.2/html/configurator.html in a web browser. If the 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 wlm-setup.

Running Slurm

Slurm can be disabled and enabled with the wlm-setup tool (section 7.3) during package installation itself. Alternatively, role assignment and role removal also enables and disables Slurm from cmgui (section 7.4.1) or cmsh (section 7.4.2).

The Slurm workload manager runs these daemons:

1. as servers:
   (a) slurmdbd: The database that tracks job accounting. It is part of the slurmdbd service.
   (b) slurmctld: The controller daemon. Monitors Slurm processes, accepts jobs, and assigns resources. It is part of the slurm service.
   (c) munged: The authentication (client-and-server) daemon. It is part of the munge service.

2. as clients:
   (a) slurmd: The compute node daemon that monitors and handles tasks allocated by slurmctld to the node. It is part of the slurm service.
   (b) slurmstepd: A temporary process spawned by the slurmd compute node daemon to handle Slurm job steps. It is not initiated directly by users or administrators.
   (c) munged: The authentication (client-and-server) daemon. It is part of the munge service.

Logs for the daemons are saved on the node that they run on. Accordingly, the locations are:

- /var/log/slurmdbd
- /var/log/slurmd
- /var/log/slurmctld
- /var/log/munge/munged.log

© Bright Computing, Inc.
7.5.2 Configuring And Running SGE

There are several flavors of SGE available since around 2010.

The reference implementation for Bright Cluster Manager is Open Grid Scheduler (OGS), and is provided by the Bright Computing repository. References to SGE in the manuals therefore imply OGS. If other flavors of SGE are meant, then it is stated explicitly. For example, in the section for Univa Grid Engine (UGE, section 7.5.3), UGE means UGE, and not SGE.

Configuring SGE

After installation and initialization, SGE has reasonable defaults, with $SGE_ROOT set to /cm/shared/apps/sge/current.

Administrators familiar with SGE can reconfigure it using the template files in /cm/shared/apps/sge/var/cm/templates/, which define the queues, host-groups and parallel environments that the wlm-setup utility uses. To configure the head node for use with SGE, the install_qmaster wrapper script under $SGE_ROOT is run. To configure a software image for use with SGE the install_execd wrapper script under $SGE_ROOT is run.

By default, the SGE application is installed in /cm/shared/apps/sge/current, the SGE documentation in /cm/shared/docs/sge/current, and job examples in /cm/shared/examples/workload/sge/jobscripts/.

Running SGE

SGE can be disabled and enabled with the wlm-setup tool (section 7.3) during package installation itself. Alternatively, role assignment and role removal also enables and disables SGE from cmgui (section 7.4.1) or cmsh (section 7.4.2).

The SGE 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 under:

/cm/shared/apps/sge/current/default/spool/

On the associated compute nodes the execution log messages exists (alongside other job tracking files and directories) at:

/cm/local/apps/sge/var/spool/node<number>/messages

where node<number> is the node name, for example:

node001, node002 ...

SGE Submission And Execution Hosts

The login, submission, and execution role assignment and behavior for SGE follow the corresponding UGE examples for submission and execution hosts, as described on page 259.

7.5.3 Installing, Configuring, And Running UGE

The workload manager, Univa Grid Engine (UGE), is a flavor of SGE (section 7.5.2) developed by Univa. Bright Cluster Manager 7.3 supports integration with UGE versions 8.2.0 and higher at the time of writing (May 2015). Older versions can also be supported, as detailed in the Bright Computing Knowledge Base (https://kb.brightcomputing.com).
UGE should be picked up directly from the Univa website at http://www.univa.com.

The installation and integration of UGE version 8.2.0 into Bright Cluster Manager 7.3 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 bundle
       `ge-<uge_ver>-bin-lx-amd64.tar.gz`
     - or the 32-bit bundle
       `ge-<uge_ver>-bin-lx-x86.tar.gz`
   - Common files: `ge-<uge_ver>-common.tar.gz`

   Here `<uge_ver>` is the UGE version, for example: 8.2.0

   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:

   **Example**

   - For RHEL-based distributions:
   
     `[root@bright73 ~]# yum install cm-uge`

   - For SLES distributions:

     `[root@bright73 ~]# zypper install cm-uge`

   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 258).

   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@bright73 ~]# cd /cm/shared/apps/uge/var/cm
   [root@bright73 cm]# cp inst_template.conf.template inst_template.conf
   [root@bright73 cm]# vi inst_template.conf

© Bright Computing, Inc.
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 Bright Cluster Manager utility `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, %. Such flagged values should not be modified, since they are replaced by Bright Cluster Manager values by `wlm-setup` during installation.

Values not enclosed by percentage signs are not replaced by `wlm-setup`. Such unflagged values can, if needed, be tuned in the copied file by the administrator. These values are then kept by `wlm-setup` during installation, and used when UGE is run.

3. `wlm-setup` is run. The directory where the downloaded UGE files are located is specified with the `-a` option.

**Example**

```
[root@bright73 ~]# wlm-setup -w uge -s -a /root/uge
```

The `wlm-setup` step is actually a wrapper here. It runs the `inst_sge` script that comes with UGE, as one of its actions. The `inst_sge` script should not normally be run directly in a cluster that runs Bright Cluster Manager.

4. The nodes are rebooted. The UGE command `qhost` then displays an output similar to:

**Example**

```
[root@bright73 ~]# module load uge
[root@bright73 ~]# 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 under:

/cm/shared/apps/uge/current/default/spool/

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 ...

Submission And Execution Hosts

In UGE terminology, a submission host, or submit host, is a node that is allowed to submit jobs. An execution host is a node on which the computational part of the job is executed.

Standard Bright Cluster Manager Use

In a standard Bright Cluster Manager configuration, a submission host or execution host in UGE are configured as follows:

Submission host: If one of the following 2 conditions is met, in node or category properties for the host:

1. Both the ugeserver and login roles are assigned to the host
   or
2. Both the ugeclient and login roles are assigned to the host

then the host is managed by Bright Cluster Manager automatically as a submit host in UGE.

Execution host: A submit host is typically not configured by administrators to run jobs. Thus, in Bright Cluster Manager, a submit host cannot normally be assigned a ugeclient role, in order to run as an execution host. For unusual situations, the advanced configuration directive AdditionalExecHosts described next, can be used to allow the submit host to run as an execution host.

Non-standard Bright Cluster Manager Use

For non-standard configurations, where the UGE service or login service is managed outside of Bright 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 Bright Cluster Manager, then the submit hosts that are to be execution hosts can be specified in the advanced configuration directive, AdditionalExecHosts, in the cmd.conf file, as follows:

Example

AdvancedConfig = { "AdditionalExecHosts=node002,login01"}

2. If the host has a ugeclient role, but the cluster is configured so that no login role can be added to a node that is be a submit host, for instance due to the login service being managed outside of Bright Cluster Manager, then the execution hosts that are to be submit hosts can be specified in the advanced configuration directive, AdditionalSubmitHosts, in the cmd.conf file, as follows:

© Bright Computing, Inc.
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 `wlm-setup` installs UGE.

The CMDaemon front ends, i.e. `cmsh` or `cmgui`, can be used to add, remove, or modify the existing PE. Each node or node category with a `ugeserver` role has a `Parallel Environments` submode. In the `Parallel Environments` submode, each PE is represented as a separate object with associated properties. For example, the following session displays Open MPI properties (some text elided):

Example

```
[root@bright73 ~]# cmsh
[bright73]% device roles bright73
[bright73->device[bright73]->roles]% assign ugeserver; commit
[bright73->...->roles[ugeserver]]% parallelenvironments
[bright73->...->roles[ugeserver]->parallelenvironments]% show openmpi
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting Summary</td>
<td>yes</td>
</tr>
<tr>
<td>Allocation Rule</td>
<td>$round_robin</td>
</tr>
<tr>
<td>Control Slaves</td>
<td>yes</td>
</tr>
<tr>
<td>Daemon Forks Slaves</td>
<td>yes</td>
</tr>
<tr>
<td>Extra Parameter</td>
<td></td>
</tr>
<tr>
<td>Job Is First Task</td>
<td>no</td>
</tr>
<tr>
<td>Master Forks Slaves</td>
<td>no</td>
</tr>
<tr>
<td>Name</td>
<td>openmpi</td>
</tr>
<tr>
<td>Slots</td>
<td>999999</td>
</tr>
<tr>
<td>Start Procedure Arguments</td>
<td>NONE</td>
</tr>
<tr>
<td>Stop Procedure Arguments</td>
<td>NONE</td>
</tr>
<tr>
<td>Urgency Slots</td>
<td>min</td>
</tr>
<tr>
<td>User Lists</td>
<td>NONE</td>
</tr>
<tr>
<td>User Lists</td>
<td>NONE</td>
</tr>
</tbody>
</table>

The values shown in the preceding example are defaults.

Database Recovery

By default, Bright 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:
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) 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. Bright 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.
- **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</th>
<th>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>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>C</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>Hardware Thread</td>
<td>T</td>
<td>t</td>
<td></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.
- **SCcCSccCC**: 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@bright73 ~]# cmsh
  [bright73]# category roles default
  [bright73->device[bright73]->roles]$ assign ugeclient; commit

© Bright Computing, Inc.
The RSMAP attribute value is then:
GPU=2 (gpu0 gpu1)

- Allow GPUs to use only different sockets:

```
[bright73->device[bright73]->roles[ugeclient]]% show | grep -i gpu
GPU devices                gpu0:SCCCScccc gpu1:SccccSCCCCC
GPUs                      0
```

The RSMAP attribute value is then:
GPU=2 (gpu0:SCCCScccc gpu1:SccccSCCCCC)

- 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:

```
[bright73->device[bright73]->roles[ugeclient]]% show | grep mic
MIC devices            mic0:SCccc mic1:ScCcc mic2:SccCc mic3:ScccC
```

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:
MIC=4 (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*.

### 7.5.4 Configuring And Running Torque

Torque is a resource manager controlling the jobs and compute nodes it talks with. Torque has its own built-in scheduler, but since this is quite basic, the open source Maui and the proprietary Moab schedulers are recommended alternatives.

#### Configuring Torque

The Torque package is installed, but not set up by default on Bright Cluster Manager 7.3. If it is not set up during installation (figure 3.22 of the *Installation Manual*), it can be set up later on using the `wlm-setup` tool (section 7.3).

The execution daemon, `pbs_mom` is already in the software images by default and does not need to be installed, even if Maui or Moab are added.

The Torque services can be enabled and disabled via role assignment as described in section 7.4. Resources such as the number of GPUs are configured in that section too for the node or node category in order to set up the corresponding resource definitions in the Torque configuration files.

Torque software components are installed in `/cm/shared/apps/torque/current`, also referred to as the `PBS_HOME`. The `torque` environment module, which sets `$PBS_HOME` and other environment
variables, must be loaded in order to submit jobs to Torque. The man pages for Torque are then accessible, with $MANPATH set to $PBS_HOME/man.

Torque documentation is available at the Adaptive Computing website at http://www.adaptivecomputing.com/resources/docs/, including various versions of the Torque administrator manual.

Torque examples are available under /cm/shared/examples/workload/torque/jobscripts/

Installing The Maui Scheduler Package
If Maui is to be installed, the Maui scheduler source version 3.3.1 must be picked up from the Adaptive Computing website at http://www.adaptivecomputing.com/support/download-center/maui-cluster-scheduler/ (registration required). The source file must be installed over the zero-sized placeholder file on the head node at /usr/src/redhat/SOURCES/maui-3.3.1.tar.gz.


The RPM file is built from the source and patches are applied, by running the rpmbuild command on the head node using the Bright Maui spec file as follows:

```
rpmbuild -bb /usr/src/redhat/SPECS/maui.spec
```

The installation can then be done with:

```
rpm -i /usr/src/redhat/RPMS/x86_64/maui-3.3.1-58_cm7.3.x86_64.rpm
```

The exact version of the rpm file to install may differ from the version shown here. The version freshly generated by the rpmbuild process is what should be used.

CMDaemon needs to be aware the scheduler is Maui for nodes in a Torque server role. This can be configured using wlm-setup to enable the torquemaui option (as shown in section 7.3.1), or using cmgui to set the scheduler from the Roles tab (as shown in section 7.4.1), or using cmsh to assign the scheduler from within the assigned torqueserver role (as shown in section 7.4.2).

Installing The Moab Scheduler Package
Moab is not installed by default in Bright Cluster Manager 7.3. Moab and related products must be purchased from Adaptive Computing. Bright Cluster Manager 7.3 expects the init script for Moab to be located in the file /etc/init.d/moab. Other files such as the Moab binaries and libraries can be installed in /cm/shared/apps/moab/<moab.version> or any other preferred location.

The Moab install documentation should be followed carefully. Issues needing attention during the Moab installation, at least for the Moab version 7 series, include the following: if using Torque and if the --with-torque=<(path) option is not specified, then library paths for Torque and Moab must be set in Moab's ldconfig configuration file.

This can be by appending them with, for example:

```
appsdir="/cm/shared/apps"
ldsdire="/etc/ld.so.conf.d"
echo $appsdir/torque/current/lib >> $ldsdire/moab.conf
echo $appsdir/moab/<version>/lib >> $ldsdire/moab.conf
ldconfig
```

Alternatively, the following lines can be added at the beginning of the Moab init.d script:

```
export LD_LIBRARY_PATH=/cm/shared/apps/torque/current/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/cm/shared/apps/moab/<version>/lib:$LD_LIBRARY_PATH
```

CMDaemon needs to be aware the scheduler is Moab for nodes in a Torque server role. This can be configured using wlm-setup to enable the torquemoab option (section 7.3.1), or using cmgui to set the scheduler role (as shown in section 7.4.1) or using cmsh to assign the scheduler from within the assigned torqueserver role (as shown in section 7.4.2).
Running Torque And Schedulers
The Torque resource manager runs the following daemons:

1. a **pbs_server** daemon. This handles submissions acceptance, and talks to the execution daemons on the compute nodes when sending and receiving jobs. It writes logs to the `/cm/shared/apps/torque/var/spool/server_logs` directory on its node. Queues for this service are configured with the `qmgr` command.

2. a **pbs_mom** execution daemon running on the nodes that are assigned the compute role. This accepts, manages, and returns the results of jobs on the compute nodes. It writes logs to the `/cm/local/apps/torque/var/spool/mom_logs` directory on the compute nodes.

3. a **trqauthd** authentication daemon, in version 4.0 and above. This service is used by Torque clients to authorize user connections to the pbs_server server daemon. When `torqueclient` or `torqueserver` roles are assigned in `cmsh/cmgui`, the authentication service is added automatically to relevant nodes so that user commands can be executed on those nodes. This allows the schedulers health check to execute Torque user utilities on compute nodes.

   If the `torqueserver` is running as an external server (pages 248, 251), then `trqauthd` must run on the external server too.

Job submission from a non-computing node: The **trqauthd** daemon is also needed by Torque users on a job submit node, if the submit node does not have a Torque role (`torqueclient` or `torqueserver`) assigned to it already. For example, some customized clusters use a category of node that is a login node. So, there may be login nodes named `<login01>`, `<login02>`, and so on. The login node does no computation itself, but users simply log into it to submit jobs. In that case, a minimal way to change the configuration of the node is as follows:

   (a) The submit hosts are added to the Torque configuration database so that Torque knows about them:

   **Example**

   ```
   qmgr -c 'set server submit_hosts = <login01>'
   qmgr -c 'set server submit_hosts += <login02>'
   qmgr -c 'set server submit_hosts += <login03>'
   ...
   ```

   The names become part of the `qmgr` database configuration. This configuration can be viewed by running:

   ```
   qmgr -c "p s"
   ```

   (b) The `/etc/init.d/trqauthd` script is copied from the head node to the login/submit node software image. Running `imageupdate` then places the script on the running login/submit nodes. Adding the `trqauthd` service to the list of services being monitored is recommended. Adding a service so that it is monitored and automatically restarted if needed is explained in section 3.11.

Jobs are however not be executed unless the scheduler daemon is also running. This typically runs on the head node and schedules jobs for compute nodes according to criteria set by the administrator. The possible scheduler daemons for Torque are:

- **pbs_sched** if Torque’s built-in scheduler itself is used. It writes logs to the `/cm/shared/apps/torque/var/spool/sched_logs` directory.

- **maui** if the Maui scheduler is used. It writes logs to `/cm/shared/apps/maui/var/spool/log`.  

© Bright Computing, Inc.
• moab if the Moab scheduler is used. Log files are written to the spool directory. For example: /cm/shared/apps/moab/moab.version/spool/moab.log if Moab is installed in /cm/shared/apps/moab/<moab.version>.

7.5.5 Configuring And Running PBS Pro

PBS Pro Versions

Bright Cluster Manager provides two versions of PBS Pro:

• PBS Pro. This is the commercial version. The PBS Pro commercial version is available under a 90-day trial license. This version can be installed:
  – as a selection option during Bright Cluster Manager 7.3 installation, at the point when a workload manager must be selected (figure 3.22 of the Installation Manual)
  – it can also be installed later on, when the cluster has already been set up.

• PBS Pro Community Edition (or CE). This is the edition with community support (http://pbspro.org). This version can be installed only after the cluster has already been set up.

When no particular version is specified in the Bright Cluster Manager documentation, then the text is valid for both editions.

If Bright Cluster Manager has already been set up without PBS Pro, then the wlm-setup tool (section 7.3) should be used to install and initialize the PBS Pro versions.

Installing PBS Pro

Both the PBS Pro commercial and CE version can be installed and initialized via wlm-setup:

Example

[root@bright73 ~]# wlm-setup -w pbspro -s

or

[root@bright73 ~]# wlm-setup -w pbspro-ce -s

Installing one version automatically disables the other version.

The software components are installed and initialized by default under the PBS_HOME directory, which is either:
/cm/shared/apps/pbspro/current
or
/cm/shared/apps/pbspro-ce/current

as appropriate.

Users must load the pbspro environment module, which sets PBS_HOME and other environment variables, in order to use either version of PBS Pro.

PBS Pro Configuration


By default, PBS Pro examples are available under the directory /cm/shared/examples/workload/pbspro/jobscripts/

Some PBS Pro configuration under Bright Cluster Manager can be done using roles. The roles are the same for both the commercial and the CE versions:

• In cmgui, the Roles tabbed pane allows the configuration of PBS Pro client and server roles.
  – For the PBS Pro server role, ticking and saving the check box state enables the role.
    • Using the Advanced button for the server role, its installation path and server spool path can be specified.
For the PBS Pro client role, ticking and saving the check box state enables the role. The number of slots and GPUs can be specified, and all or just some of the queues can be selected.

- Using the Advanced button for the client role, its client spool path, properties, and GPU devices can be specified.

- 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.

**Example**

```
[root@bright73 ~]# cmsh
[bright73]% device roles bright73
[bright73->device[bright73]->roles]% assign pbsproserver; commit
[bright73->device[bright73]->roles[pbsproserver]]% show
Parameter Value
----------------------------------------------
External Server no
Name pbsproserver
Provisioning associations <0 internally used>
Revision
Type PbsProServerRole
Prefix /cm/shared/apps/pbspro/current
Spool /cm/shared/apps/pbspro/var/spool
...
```

Further configuration of PBS Pro is done using its qmgr command and is covered in the PBS Pro documentation.

Cloudbursting with cluster extension may need special handling for PBS Pro. If this feature is needed, then the administrator should contact Bright Computing support via the website https://support.brightcomputing.com.

**Running PBS Pro**

PBS Pro runs the following three 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/server_logs/` directory that is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-pe` on the nodes it is associated with. 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/sched_logs/` directory under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-pe`.

3. A `pbs_mom` execution daemon running on each compute node. This accepts, manages, and returns the results of jobs on the compute nodes. It writes logs on the head node to the `var/spool/mom_logs/` directory under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-pe`, and to `/cm/local/apps/pbspro/var/spool/mom_logs/` on the compute nodes.

**7.5.6 Installing, Configuring And Running OpenLava**

*At the time of writing (November 2016), OpenLava from version 3 onward was subject to a copyright infringement claim by IBM. At the request of the OpenLava founder, its packages are now no longer distributed with Bright Cluster Manager. Text in this section that refers to such packages is now invalid until the situation is resolved.*
7.5 Configuring And Running Individual Workload Managers

Prerequisites To Installing OpenLava
The OpenLava version 4.0.0 packages are available from the Bright Cluster Manager repositories, and can be installed on the head node, or active head node in a failover setup, with:

Example

[root@bright73 ~]# yum install openlava

Similarly, the client package can be installed in a software image such as default-image with:

Example

[root@bright73 ~]# yum --installroot=/cm/images/default-image install openlava-client

The updateprovisioners (section 5.2.4) and imageupdate (section 5.6.2) commands can be used to implement the changes on all running nodes.

Installing OpenLava
After package installation has been carried out, OpenLava installation can be done on the cluster with wlm-setup (section 7.3):

wlm-setup -w openlava -s

Configuring OpenLava
After installation of OpenLava using wlm-setup (section 7.3), OpenLava software components are installed in /cm/shared/apps/openlava/current.

Documentation for OpenLava is available via man pages under /cm/shared/apps/openlava/current/share/man. Further documentation is available online at: http://www.openlava.org/documentation/documentation.html.

Reasonable defaults are used for OpenLava, but administrators familiar with OpenLava can reconfigure it by modifying the configuration files under /cm/shared/apps/openlava/var/etc/.

Within the OpenLava module environment, running the OpenLava command bhosts displays something like:

[root@bright73 ~]# bhosts
HOST_NAME  STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
node001    ok     -     2    0     0     0     0     0
node002    ok     -     2    0     0     0     0     0
node003    unavail  -     1    0     0     0     0     0
node004    closed  -     1    0     0     0     0     0
node005    ok     -     1    0     0     0     0     0
bright73   ok     -     0    0     0     0     0     0

Running OpenLava
OpenLava can be disabled and enabled with the wlm-setup tool (section 7.3) during package installation itself. Alternatively, role assignment and role removal also enables and disables OpenLava from cmgui (section 7.4.1) or cmsh (section 7.4.2).

All compute nodes (sometimes called passive OpenLava masters in OpenLava terminology) have the same set of daemons. One node (it can be a regular node or it can be a head node) takes on the role of active OpenLava master. It is enough to assign only the openlavaserver role to each node or node category that is running the daemons. The nodes used to submit an OpenLava job take on the openlavaclient role, but do not have the daemons running on them.

The OpenLava workload manager daemons are:

© Bright Computing, Inc.
• daemons that can run on the active OpenLava master node:
  
  – mbatchd: The active OpenLava master batch daemon. Started by the batch daemon on a passive OpenLava master. Responsible for the overall state of OpenLava jobs. Receives job submissions, and information query requests. Manages jobs held in queues. Dispatches jobs to hosts as determined by mbschd. It is a part of the openlava service.
  

• daemons that can run on both on the active and passive OpenLava master nodes):
  
  – sbatchd: The standard batch openlava daemon. It is a part of the openlava service.
  
  – lim: The load information manager daemon. Collects built-in load indices that reflect the load situations of CPU, memory, disk space, I/O, and interactive activities on individual hosts. It is a part of the openlava service.
  
  – res: The remote execution manager daemon. Executes user tasks on remote hosts. It is a part of the openlava service.
  
  – pim: The process information manager daemon. Is responsible for monitoring all jobs and monitoring every process created for all jobs running on the node.

lim, res, and sbatchd daemons maintain log files in:

/cm/local/apps/openlava/var/log

Batch accounting files are shared by all potential active OpenLava master nodes. The spools are stored in the work directory:

/cm/shared/apps/openlava/var/work/logdir.

7.5.7 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 as an option in the installation carried out by Bright Cluster Manager 7.3 in figure 3.22 of the Installation Manual.

Installing LSF

The workload manager LSF versions 9 or 10 are installed and integrated into Bright Cluster Manager 7.3 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: lsf<$lsf_ver>_documentation.tar.Z
   
   • License file: platform_lsf_<lic_type>_entitlement.dat

Here:

• $lsf_ver$ is the LSF version, for example: 9.1.1

• $kern_ver$ is the Linux kernel version, for example: 2.6

• $glibc_ver$ is the glibc library version, for example: 2.3

• $cpu_arch$ is the CPU architecture, for example: x86_64

© Bright Computing, Inc.
7.5 Configuring And Running Individual Workload Managers

- `<lic_type>` is the license type, for example: std

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 4 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.

2. The `cm-lsf` package must be installed from the Bright Computing repository:

```
[root@bright73 ~]# yum install cm-lsf  # RHEL-based distributions
```

or

```
[root@bright73 ~]# zypper install cm-lsf  # SLES distributions.
```

The `cm-lsf` package contains a template environment module file and an installation configuration file. The installation configuration file may be tuned by administrator if required. It is passed to the `lsfinstall` script distributed with LSF, which is be executed by `wlm-setup` during setup. To change the default values in the installation configuration file, the administrator should copy the template file first and then change the copied file:

```
[root@bright73 ~]# cd /cm/shared/apps/lsf/var/cm/
[root@bright73 cm]# cp install.config.template install.config
[root@bright73 cm]# vi install.config
```

The `install.config` file can be changed by the administrator to decide the location of LSF software and some other LSF configuration options, if needed. The changed file overrides the settings suggested by the `.template` file, when the Bright Cluster Manager utility `wlm-setup` runs in the next step of the installation procedure.

A few of the values in the key-value pairs in the file are enclosed by percentage signs, %. Such “flagged” values should not be modified, since they are replaced by Bright Cluster Manager values by `wlm-setup` during installation.

Values not enclosed by percentage signs are left alone by `wlm-setup`. Such “unflagged” values can, if needed, be tuned in the file copy by the administrator. These values are then kept by `wlm-setup` during installation, and used when LSF is run.

3. `wlm-setup` is run. The directory where the LSF files were downloaded is specified with the `-a` option.

**Example**

```
[root@bright73 ~]# wlm-setup -w lsf -s -a /root/lsf
```

4. The nodes are then rebooted, and the LSF command `bhosts` then displays an output similar to:

**Example**

```
[root@bright73 ~]# module load lsf
[root@bright73 ~]# bhosts

<table>
<thead>
<tr>
<th>HOST_NAME</th>
<th>STATUS</th>
<th>JL/U</th>
<th>MAX</th>
<th>NJOBS</th>
<th>RUN</th>
<th>SSUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>bright73</td>
<td>ok</td>
<td>-</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>head2</td>
<td>ok</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node001</td>
<td>ok</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node002</td>
<td>ok</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node003</td>
<td>unavail</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node004</td>
<td>closed</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>node005</td>
<td>ok</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
The output in the preceding example has been truncated for this manual, for convenience.

The installation status can be checked with:

[root@bright73 ~]# /etc/init.d/lsf status
Show status of the LSF subsystem
lim (pid 21138) is running...
res (pid 17381) is running...
sbatchd (pid 17383) is running...

while default queues can be seen by running:

[root@bright73 ~]# module load lsf
[root@bright73 ~]# bqueues

<table>
<thead>
<tr>
<th>QUEUE_NAME</th>
<th>PRIO</th>
<th>STATUS</th>
<th>MAX</th>
<th>JL/U</th>
<th>JL/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>owners</td>
<td>43</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>priority</td>
<td>43</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>night</td>
<td>40</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>chkpnt_rerun_qu</td>
<td>40</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>short</td>
<td>35</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>license</td>
<td>33</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>normal</td>
<td>30</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>interactive</td>
<td>30</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>idle</td>
<td>20</td>
<td>Open:Active</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The output in the preceding example has been truncated for this manual, for convenience.

**Configuring LSF**

**LSF server configuration:** After installation, the following CMDaemon settings can be specified for the LSF server role:

- **prefix:** this sets the path to the root of the LSF installation.
- **var:** this sets the path to the var directory of LSF

These settings can be specified as follows:

- **Within cmgui:** For example, for a particular category in the Node Category resource, or for a particular regular node in the Nodes resource, or for a head node in the Head nodes resource, the Roles tabbed pane can be selected. Within the Roles tabbed pane, in the LSF Server Role tile, clicking the Advanced button brings up a pop up window. This allows the prefix and var options to be specified. An example can be seen for a regular node002 in the LSF server role in figure 7.7.

![Figure 7.7: cmgui access to LSF path settings via roles tab](https://www.brightcomputing.com/support/images/cmgui-access-to-lsf-path-settings-via-roles-tab.png)
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` parameter can be set. The following example shows how to set the LSF prefix parameter for the default category.

**Example**

```
[root@bright73 ~]# cmsh
[bright73]% category use default; roles
[bright73->category[default]->roles]% assign lsfserver; commit
[bright73->...[lsfserver]]% set prefix /cm/shared/apps/lsf/current2
[bright73->...[lsfserver*]]% 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:

  ```
  set queues <qname1> [...<qname2> ...]
  ```

  Alternatively, these, and other queues can be added or deleted using `cmgui` (section 7.6.2) or `cmsh` (section 7.7.2).

- **Options** The options that can be set for a regular node `node001`, via figure 7.8 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.

  From `cmsh` these properties are accessible from within the appropriate node or category `roles` submode (section 7.4.2).
Data-Aware Scheduling With `cmsub`: The ability to use `cmsub` to submit jobs to cloud nodes, as described in section 4.7 of the *User Manual*, is called *data-aware scheduling*. Data-aware scheduling is enabled for LSF as follows:

1. If LSF is not under the control of CMDaemon because the directive `FreezeChangesToLSFConfig` (page 595) is active, and if the `cloudtransfers` queue does not exist in LSF, then the queue should be added manually into the following configuration file:

   ```
   /cm/shared/apps/openlava/var/etc/lsb.queues
   ```

   The default value for `$LSF_ENVDIR` is

   ```
   /cm/shared/apps/openlava/var/etc/
   ```

2. In order to run the cloud prolog script as the root user, `/etc/lsf.sudoers` must be configured. By default this file does not exist. Its ownership must be root. It must also have read-write permissions for root only, i.e. its unix permissions can be set with `chmod 600 /etc/lsf.sudoers`. To execute the `PRE_EXEC` script as root, `LSB_PRE_POST_EXEC_USER=root` should be set in `lsf.sudoers`:

   ```bash
   [root@bright73 ~]$ echo "LSB_PRE_POST_EXEC_USER=root" >> /etc/lsf.sudoers
   [root@bright73 ~]$ chmod 600 /etc/lsf.sudoers
   ```

3. The changes are applied by restarting the hosts:

   ```bash
   [root@bright73 ~]$ badmin hrestart all
   ```

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 `cmgui` (sections 7.4.1) or `cmsh` (section 7.4.2).

An active LSF master (typically, but not necessarily on a head node) has the following LSF-related processes or services 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.

Logs for LSF processes and services are kept under `/cm/shared/apps/lsf/log/` (or `$LSF_TOP/log/` if the value of `$LSF_TOP` during installation is other than the recommended `/cm/shared/apps/lsf/`).

© Bright Computing, Inc.
7.6 Using cmgui With Workload Management

Viewing the workload manager services from cmgui is described in section 7.4.3. Selecting the Bright Cluster Manager workload manager item from the resources tree displays tabs that let a cluster administrator change the states of:

- jobs
- queues
- nodes

These tabs are described next.

7.6.1 Jobs Display And Handling In cmgui

Selecting the Jobs tab displays a list of job IDs, along with the scheduler, user, queue, and status of the job (figure 7.9).

![Figure 7.9: Workload Manager Jobs](image)

Within the tabbed pane:

- The Show button allows further details of a selected job to be listed. For example, the details of a Slurm job are displayed as in figure 7.10:
• The **Remove** button removes selected jobs from the queue.

• The **Hold** button stops selected queued jobs from being considered for running by putting them in a *Hold* state.

• The **Release** button releases selected queued jobs in the *Hold* state so that they are considered for running again.

• The **Suspend** button suspends selected running jobs.

• The **Resume** button allows selected suspended jobs to run again.

• The **Refresh** button refreshes the screen so that the latest available jobs list is displayed.

### 7.6.2 Queues Display And Handling In cmgui

Selecting the **Queues** tab displays a list of queues available, their associated scheduler, and the list of nodes that use each queue (figure 7.11).
Within the tabbed pane:

- The **Edit** button allows an existing job queue of a workload manager to be edited. The particular values that can be edited for the queue depend upon the workload manager used (figures 7.12, 7.13 and 7.14).

![Figure 7.11: Workload Manager Queues](image1)

![Figure 7.12: Workload Management Queues Edit Dialog For SGE](image2)
In the edit dialog:

- the generic names “Minimum wallclock” and “Maximum wallclock” correspond respectively to the soft and hard walltimes allowed for the jobs in the queue. Specifically, these are $s_{rt}$ and $h_{rt}$ in SGE, or $\text{resources_default.walltime}$, and $\text{resources_max.walltime}$ in Torque and PBS Pro. The Maximum time parameter for Slurm corresponds to Maximum wallclock and sets Slurm’s MaxTime value in $/etc/slurm/slurm.conf$.

- The Prolog and Epilog files that can be specified in the dialog for SGE are CMDaemon-managed scripts run before and after the job is executed. A default global Prolog script is used by Bright Cluster Manager if no CMDaemon-managed script exists. The global script ensures that a health check script flagged as a prejob script (section 10.4.3), is run before a job runs. Administrators creating their own Prolog script can refer to the
global Prolog script (/cm/local/apps/cmd/scripts/prolog), to see how it calls the prolog-healthchecker script, which, in turn, executes a flagged health check script in its call to cmprejobcheck. Only one cmprejobcheck runs at a time on a particular node, in order to avoid health check diagnostics interfering with themselves.

The Prolog and Epilog scripts for Torque and PBS Pro are set up for the software images, and their paths cannot be altered via Bright Cluster Manager.

- The Add button allows a new job queue to be added to a workload manager.
- The Remove button removes a job queue from the workload manager.
- The Revert button reverts the Queues tabbed pane to its last saved state.
- The Save button saves the modified Queues tabbed pane.

### 7.6.3 Nodes Display And Handling In cmgui

Selecting the Nodes tab displays a list of nodes, along with their schedulers, queues, and whether they are in a status of Drained or Undrained (figure 7.15).

Figure 7.15: Node Drainage

- The Drain button acts on the selected node, scheduler, and queue combination. It opens up a dialog (figure 7.16).
If the default options are accepted, then clicking on the Ok button sets the node to a Drained state. This means that jobs no longer run on that node, for that workload manager.

- If actions are appended to the drain command, from the set of actions described in Appendix G.3.1, then these are executed in order after the default drain action takes place for that node, for that workload manager.

- If there are any existing actions, then ticking the Clear all existing actions checkbox means that they are cleared after the OK button is clicked.

- The Undrain button unsets a Drained state, allowing jobs to start running for that combination.

- The Refresh button refreshes the screen so that the latest available state is displayed.

The cmsh commands for this functionality are described in section 7.7.3.

### 7.7 Using cmsh With Workload Management

#### 7.7.1 Jobs Display And Handling In cmsh: jobs Mode

**jobs Mode In cmsh: Top Level**

At the top level of jobs mode, the administrator can view all jobs regardless of scheduler type with the list command:

**Example**

```
[bright73->jobs] % list
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Job ID</th>
<th>User</th>
<th>Queue</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGEJob</td>
<td>620</td>
<td>maud</td>
<td>all.q</td>
<td>r</td>
</tr>
<tr>
<td>SGEJob</td>
<td>621</td>
<td>maud</td>
<td>qw</td>
<td></td>
</tr>
<tr>
<td>TorqueJob</td>
<td>90.bright73+</td>
<td>maud</td>
<td>hydroq</td>
<td>R</td>
</tr>
</tbody>
</table>

Also within the jobs mode, the hold, release, suspend, resume, show, and remove commands act on jobs when used with a specified scheduler type and job ID. Continuing with the example:
Using cmsh

With Workload Management

7.7 Using cmsh

With Workload Management

> suspend torque 90.bright73.cm.cluster

Success

> list

<table>
<thead>
<tr>
<th>Type</th>
<th>Job ID</th>
<th>User</th>
<th>Queue</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGEJob</td>
<td>620</td>
<td>maud</td>
<td>all.q</td>
<td>r</td>
</tr>
<tr>
<td>SGEJob</td>
<td>621</td>
<td>maud</td>
<td>qw</td>
<td></td>
</tr>
<tr>
<td>TorqueJob</td>
<td>90.bright73+</td>
<td>maud</td>
<td>hydroq</td>
<td>S</td>
</tr>
</tbody>
</table>

While at the jobs mode top level, the suspended job here can be made to resume using suspend’s complementary command—resume. However, resume along with the other commands can also be executed within a scheduler submode, as is shown shortly.

**jobs Mode in cmsh: The scheduler Submode**

Setting the scheduler type sets the scheduler submode, and can be done thus (continuing with the preceding example):

> scheduler torque

> jobs (torque) %

The submode restriction can be unset with: scheduler "".

The top level job mode commands executed within the scheduler submode then only apply to jobs running under that scheduler. The list and resume commands, for example, then only apply only to jobs running under torque (continuing with the example):

> list; !#no sge jobs listed now - only torque

<table>
<thead>
<tr>
<th>Type</th>
<th>Job ID</th>
<th>User</th>
<th>Queue</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TorqueJob</td>
<td>90.bright73+</td>
<td>maud</td>
<td>hydroq</td>
<td>S</td>
</tr>
</tbody>
</table>

> resume 90.bright73.cm.cluster; !#torque job

Success

> list; !#only torque jobs

<table>
<thead>
<tr>
<th>Type</th>
<th>Job ID</th>
<th>User</th>
<th>Queue</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TorqueJob</td>
<td>90.bright73+</td>
<td>maud</td>
<td>hydroq</td>
<td>R</td>
</tr>
</tbody>
</table>

**jobs Mode in cmsh: The show Command**

The show command for a particular scheduler and job lists details of the job. Continuing with the preceding example:

> show 90.bright73.cm.cluster;

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments</td>
<td>-q hydroq /home/maud/sleeper.sh</td>
</tr>
<tr>
<td>Executable</td>
<td></td>
</tr>
<tr>
<td>In queue</td>
<td></td>
</tr>
<tr>
<td>Job ID</td>
<td>90.bright73.cm.cluster</td>
</tr>
<tr>
<td>Job name</td>
<td>sleeper.sh</td>
</tr>
<tr>
<td>Mail list</td>
<td></td>
</tr>
<tr>
<td>Mail options</td>
<td>a</td>
</tr>
<tr>
<td>Maximum wallclock time</td>
<td>02:00:00</td>
</tr>
<tr>
<td>Memory usage</td>
<td>0</td>
</tr>
<tr>
<td>Nodes</td>
<td>node001</td>
</tr>
<tr>
<td>Notify by mail</td>
<td>yes</td>
</tr>
<tr>
<td>Number of processes</td>
<td>1</td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
</tr>
<tr>
<td>Queue</td>
<td>hydroq</td>
</tr>
</tbody>
</table>

© Bright Computing, Inc.
Run directory /home/maud
Running time 809
Status R
Stderr file bright73.cm.cluster:/home/maud/sleeper.sh.e90
Stdout file bright73.cm.cluster:/home/maud/sleeper.sh.o90
Submission time Fri Feb 18 12:49:01 2011
Type TorqueJob
User maud

7.7.2 Job Queues Display And Handling In cmsh: jobqueue Mode

Properties of scheduler job queues can be viewed and set in jobqueue mode.

jobqueue Mode In cmsh: Top Level

If a scheduler submode is not set, then the list, qstat, and listpes commands operate, as is expected, on all queues for all schedulers.

At the top level of jobqueue mode:

- list lists the queues associated with a scheduler.

Example

     [root@bright73 ~]# cmsh
     [bright73]% jobqueue
     [bright73->jobqueue]% list
     Type Name
        ------------ ------------------------
        sge all.q
        torque default
        torque hydroq
        torque longq
        torque shortq

- qstat lists statistics for the queues associated with a scheduler.

Example

     [bright73->jobqueue]% qstat
     Queue Load Total Used Available
        ------------ --------- --------- --------- ---------
        all.q 0.1 1 0 1

     Queue Running Queued Held Waiting
        ------------ --------- --------- --------- ---------
        default 0 0 0 0
        hydroq 1 0 0 0
        longq 0 0 0 0
        shortq 0 0 0 0

- listpes lists the parallel environment available for schedulers.

Example
(some details elided)

[bright73->jobqueue]$ listpes
Scheduler Parallel Environment
----------- ------------------------
sge make
sge mpich
...
sge openmpi_ib

• scheduler sets the scheduler submode

Example

[bright73->jobqueue]$ scheduler torque
Working scheduler is torque
[bright73->jobqueue(torque)]$

The submode can be unset using: scheduler ""

jobqueue Mode In cmsh: The scheduler Submode
If a scheduler submode is set, then commands under jobqueue mode operate only on the queues for that particular scheduler. For example, within the torque submode of jobqueue mode, the list command shows only the queues for torque.

Example

[bright73->jobqueue]$ list
Type Name
----------- ------------------------
sge all.q
torque default
torque longq
torque shortq
[bright73->jobqueue]$ scheduler torque
Working scheduler is torque
[bright73->jobqueue(torque)]$ list
Type Name
----------- ------------------------
torque default
torque longq
torque shortq

jobqueue Mode In cmsh: Other Object Manipulation Commands
The usual object manipulation commands of section 2.5.3 work at the top level mode as well as in the scheduler submode:

Example

[bright73->jobqueue]$ list torque
Type Name
----------- ------------------------
torque default
torque longq
torque shortq
[bright73->jobqueue]$ show torque longq

© Bright Computing, Inc.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximal runtime</td>
<td>23:59:59</td>
</tr>
<tr>
<td>Minimal runtime</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Queue type</td>
<td>Execution</td>
</tr>
<tr>
<td>Routes</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>torque</td>
</tr>
<tr>
<td>name</td>
<td>longq</td>
</tr>
<tr>
<td>nodes</td>
<td>node001.cm.cluster node002.cm.cluster</td>
</tr>
</tbody>
</table>

[bright73->jobqueue] % get torque longq maximalruntime
23:59:59

[bright73->jobqueue]%
[bright73->jobqueue] % scheduler torque
Working scheduler is torque
[bright73->jobqueue(torque)] % list

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>torque</td>
<td>default</td>
</tr>
<tr>
<td>torque</td>
<td>longq</td>
</tr>
<tr>
<td>torque</td>
<td>shortq</td>
</tr>
</tbody>
</table>

[bright73->jobqueue(torque)] % show longq

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximal runtime</td>
<td>23:59:59</td>
</tr>
<tr>
<td>Minimal runtime</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Queue type</td>
<td>Execution</td>
</tr>
<tr>
<td>Routes</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>torque</td>
</tr>
<tr>
<td>name</td>
<td>longq</td>
</tr>
<tr>
<td>nodes</td>
<td>node001.cm.cluster node002.cm.cluster</td>
</tr>
</tbody>
</table>

[bright73->jobqueue(torque)] % get longq maximalruntime
23:59:59

[bright73->jobqueue(torque)] % use longq
[bright73->jobqueue(torque)->longq] % show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximal runtime</td>
<td>23:59:59</td>
</tr>
<tr>
<td>Minimal runtime</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Queue type</td>
<td>Execution</td>
</tr>
<tr>
<td>Routes</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>torque</td>
</tr>
<tr>
<td>name</td>
<td>longq</td>
</tr>
<tr>
<td>nodes</td>
<td>node001.cm.cluster node002.cm.cluster</td>
</tr>
</tbody>
</table>

[bright73->jobqueue(torque)->longq] % get maximalruntime
23:59:59

### 7.7.3 Nodes Drainage Status And Handling In cmsh

The node drainage displayed using cmgui (section 7.6.3) has cmsh equivalents that are described in this section.

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@bright73 ~]# cmsh
Using cmsh With Workload Management

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:

Example

The help text for each command indicates the syntax:

Example

Name: drain - Drain jobs (not data) on a set of nodes
Usage: drain [OPTIONS/node]

Options: -n, --nodes node(list)
List of nodes, e.g.
node001..node015,node020..node028,node030 or
"/some/file/containing/hostnames"

-g, --group group(list)
Include all nodes that belong to the node group, e.g.
testnodes or test01,test03

-c, --category category(list)
Include all nodes that belong to the category, e.g. default or default,gpu

-r, --rack rack(list)
Include all nodes that are located in the given rack, e.g.
rack01 or rack01..rack04

-h, --chassis chassis(list)
Include all nodes that are located in the given chassis, e.g.
chassis01 or chassis03..chassis05

-e, --overlay overlay(list)
Include all nodes that are part of the given overlay, e.g.
overlay1 or overlayA,overlayC
-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.*"

-t, --setactions <actions>
  set drain actions, actions already set will be removed (comma-separated)

-a, --appendactions <actions>
  append drain actions to already existing drain actions (comma-separated)

-e, --removeactions <actions>
  remove drain actions

-l, --clearactions
  remove all drain actions

-i, --listactions
  list all drain actions

Examples: drain  Drain the current node
drain node001  Drain node001
drain -r rack01  Drain all nodes in rack01
drain -a reboot  Drain the current node, and append reboot when all jobs are completed

7.7.4 Launching Jobs With cm-launcher

Some MPI distributions occasionally leave processes behind that cannot be killed from the workload manager. To prevent this situation from occurring, Bright Cluster Manager provides the cm-launcher wrapper for the mpirun command, which tracks and kills such processes. The tracking relies on knowing what processes the workload manager launches, so it can only run from within a suitable workload manager. Currently, suitable workload managers are Torque or SGE.

In the following job script examples, instead of having users use:

Example

mpirun <...>

which may not have a job clean up properly after it ends, users can use:

Example

cm-launcher mpirun <...>

which cleans up properly after the job ends. Here, <...> is used to indicate the mix of options, programs and arguments used with mpirun.

For Torque  cm-launcher can be used if the default Torque epilogue script provided by the Bright Cluster Manager Torque package is present, at /cm/local/apps/torque/var/spool/mom_priv/epilogue.
For SGE the procedure is as follows:

- A symlink to cm-launcher is created to the `<arch>` SGE functions directory library
  
  ```bash
  ln -s /cm/shared/apps/sge/var/cm/cm-launcher /cm/shared/apps/sge/current/bin/<arch>
  ```

- SGE's epilog (spelled without the “-ue” ending) script is set either for a particular queue, or globally.
  
  To set it for a particular queue, for example, for the default queue all.q, the following cmsh commands can be run:

  Example

  ```bash
  [root@bright73 ~]# cmsh
  [bright73]% jobqueue
  [bright73->jobqueue]% scheduler sge
  Working scheduler is sge
  [bright73->jobqueue(sge)% set all.q epilog /cm/shared/apps/sge/var/cm/epilog
  [bright73->jobqueue*(sge)% commit
  Successfully committed 1 JobQueues
  [bright73->jobqueue(sge)%]
  
  To set it globally for all queues, that is, not just the queue all.q but all the other queues as well, the following SGE configuration command is used:

  ```bash
  qconf -mconf global
  ```

  This starts up an editor acting on the global configuration setting. The epilog line entry is modified to:

  ```bash
  epilog /cm/shared/apps/sge/var/cm/epilog
  ```

### 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 cmgui**

To create a new queue, the Workload Management item is selected, and the Queues tab selected. The Add button is used to associate a newly created queue with a scheduler and add it to the workload manager. The modification is then saved (figure 7.17).

![Figure 7.17: Adding A New Queue Via cmgui](image-url)
A useful way to create a new category is to simply clone the old default category over to a new category, and then change parameters in the new category to suit the new machine (figure 7.18).

![Figure 7.18: Cloning A New Category Via cmgui](image)

Having cloned and saved the category, called shornodes in the example of figure 7.18, the configuration of the category may be altered to suit the new machine, perhaps by going into the settings tab and altering items there.

Next, the queue is set for this new category, shornodes, by going into the Roles tabbed pane of that category, selecting the appropriate workload manager client role and queues, and saving the setting (figure 7.19).

![Figure 7.19: Setting A Queue For A New Category Via cmgui](image)

Finally, a node in the Nodes folder that is to be placed in the new shornodes category must be placed there by changing the category value of that node in its Settings tab (figure 7.20).
A New Category And A New Queue With cmsh

The preceding example can also be configured in cmsh as follows:

The new queue can be added from within jobqueue mode, for the workload manager. For example, if Slurm is enabled:

```
[bright73]\% jobqueue add shorq
[bright73->jobqueue*(slurm)->shorq\%]% commit
```

The new category, shornodes, can be created by cloning an old category, such as default:

```
[bright73->jobqueue(slurm)->shorq]\% category
[bright73->category]% clone default shornodes
[bright73->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:

```
[bright73->category[shornodes]]% roles
[bright73->category[shornodes]->roles]% assign slurmcclient; commit
[bright73->category[shornodes]->roles[slurmcclient]]% append queues shorq
[bright73->category[shornodes*]->roles*]% commit
```

The nodes belonging to the shornodes category can then be placed by going into device mode to select the nodes to be placed in that category. For example:

```
[bright73->category[shornodes]->roles]% device use node002
[bright73->device[node002]]% set category shornodes
[bright73->device*[node002*]]% commit
```

### 7.8.2 Setting Up A Prejob Health Check

**How It Works**

Health checks (section 10.2.4) by default run as scheduled tasks over regular intervals. They can optionally be configured to run as prejob health checks, that is, before a job is run. If the response to a prejob health check is **PASS**, then it shows that the node is displaying healthy behavior for that particular health aspect.

If the response to a prejob health check is **FAIL**, then it implies that the node is unhealthy, at least for that aspect. A consequence of this is 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. 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, if the node has just been flagged as unhealthy by a health check. So for a cluster in the default configuration, the action (section 10.2.2) taken by the prejob health check defaults to putting the node in a Drained state (sections 7.6.3 and 7.7.3), with Bright Cluster Manager arranging a rescheduling of the job so that the job runs only on nodes that are believed to be healthy.

A node that has been put in a Drained state with a health check is not automatically undrained. The administrator must clear such a state manually.

The failed prejob health check (page 661) is enabled by default, and logs prejob health check passes and failures.

Configuration Using cmgui
To configure the monitoring of nodes as a prejob health check in cmgui, the Monitoring Configuration resource item is selected, and the Health Check Configuration tabbed pane is opened. The default resource is chosen as a value for Health Check Configuration, and the Add button is clicked on to add the health check via a dialog (figure 7.21). In the dialog, the Health Check script value is set to the chosen health check. If the health check is already listed, then it can be edited as needed. The Sampling interval is set to prejob, which automatically sets the Fail action to the Drain node action, when the configuration is saved. After saving these settings, any node that is not in the Drained state in the default resource gets a pre-job check when a job is scheduled for the node, and the pre-job check puts the node in a Drained state if it is unhealthy.

![Figure 7.21: Configuring A Prejob Health Check Via cmgui](image)

Configuration Using cmsh
To configure a prejob health check with cmsh, the healthconf submode (section 10.7.4) is entered, and the prejob health script object used. In the following example, where some text has been elided, the object is the smart script:

Example

```
[bright73% monitoring setup healthconf default
[bright73%monitoring->setup[default]->healthconf]% use smart
```
The `failactions` value automatically switches to "enter: Drain node()" when the value for the `checkinterval` parameter of the health check is set to `prejob`.

### 7.9 Power Saving Features

This section discusses power saving options for workload managers:

- a mechanism specifically for Slurm (section 7.9.1)
- a more general mechanism, `cm-scale-cluster` for nodes in a cluster managed by Bright Cluster Manager (section 7.9.2)

#### 7.9.1 Slurm

Slurm provides a power saving mechanism that allows nodes in a client role to go into a power save mode. Power saving is disallowed by default for the head node. If power saving is allowed, then it must be enabled on the nodes where it is to function.

**Allowing And Enabling Power Saving On Slurm**

Power saving can be allowed and enabled at the category or node level as follows:

**Using cmgui:** If using cmgui, then within the Nodes resource or the Node Categories resource, the Roles tabbed pane is selected.

- To allow power saving, the `Power save allowed` checkbox can be ticked within the Slurm client role, and the setting saved. For example, figure 7.22 illustrates this for node001 specified within the Nodes resource.

![Figure 7.22: Allowing Power Save In Slurm With cmgui](image)

- To enable power saving, the `Power save enabled` checkbox can then be ticked within the corresponding Slurm server role Advanced role settings (figure 7.23), and the setting saved. The Slurm server role is typically assigned on the head node.
Using cmsh: If using cmsh, then within category or device mode, the slurmclient role is chosen from the roles submode.

- To allow power saving, the value for powersavingallowed is set to yes within the slurmclient role, and the setting is committed.

  Example

  ```
  [root@bright73 ~]# cmsh
  [bright73]% device roles node001
  [bright73->device[node001]->roles]% set slurmclient powersavingallowed yes; commit
  ```

- To enable power saving, the powersavingenabled parameter can be set to yes in the slurmserver role, as shown in the following example:

  Example

  ```
  [root@bright73 ~]# cmsh
  [bright73]% device roles bright73
  [bright73->device[bright73]->roles]% set slurmserver powersavingenabled yes; commit
  ```

  The slurmserver role is typically assigned to the head node.

  Additional power saving configuration file parameters are described in the Slurm power saving guide at http://slurm.schedmd.com/power_save.html

7.9.2 The cm-scale-cluster Utility

Introduction

The cm-scale-cluster command 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, it automatically scales a cluster up or down on-demand, powering up physical nodes, cloud nodes, or virtual nodes according to a specification file. The cm-scale-cluster utility works
by collecting information about jobs, and knowledge of the queues they are to be assigned to.\footnote{SGE/UGE jobs have no queue assignment by default, and are therefore ignored by default. Adding a line such as: \texttt{-q all.q} to the \texttt{var/default/common/sge_request} file, under \texttt{/cm/shared/apps/sge/ (or /cm/shared/apps/uge/)} assigns SGE (or UGE) jobs to the all.q queue by default, so that \texttt{cm-scale-cluster} considers them.} Based on that it clones and starts compute nodes when the jobs are ready to start. The utility also stops or terminates compute nodes when no queued or running jobs remain on the managed nodes.

The \texttt{cm-scale-cluster} utility is placed in a local directory when the administrator installs the \texttt{cm-scale-cluster} package. Therefore, if it is needed on a non-head node, the package should be installed in the corresponding software image. The administrator also typically configures the default user environment for users of the utility so that the \texttt{cm-scale-cluster} modules environment automatically loads up (section 2.2.3).

A compute node managed by \texttt{cm-scale-cluster} can be defined either via configuring its node group or via configuring its template node within the configuration file of the utility. The definitions in the configuration file are set with key=value options. By default, a file at \texttt{/cm/local/apps/cm-scale-cluster/etc/default.conf} is used, while the command line option \texttt{-c} allows any other file to be specified. The default configuration file has helpful commented-out configuration examples and explanations in it.

An example of a configuration file is:

**Example**

```bash
QUEUE=cloudq NODEGROUP=nodegroup1 TEMPLATE=cnode001 EXTRA_NODES=us-west-1-director
QUEUE=localq NODEGROUP=nodegroup2 JOBS_PER_NODE=4
TEMPLATE=cnode001 NODEGROUP=cnode[002..128] START=YES STOP=YES REMOVE=YES
EXTRA_NODE=eu-west-1-director IDLE_TIMEOUT=1800 START=YES STOP=YES
WORKLOAD_MANAGER = slurm
POLICY_MODULE = /cm/local/apps/cm-scale-cluster/lib/default-policy.py
ACTION_MODULE = /cm/local/apps/cm-scale-cluster/lib/default-action.py
PARALLEL_REQUESTS = 10
LOG_FILE = /var/log/cm-scale-cluster.log
LOCK_FILE = /var/lock/subsys/cm-scale-cluster
SPOOL_DIR = /var/spool/cmd
DEBUG = YES
```

The preceding file is explained very briefly on page 299. The details of the command line options to \texttt{cm-scale-cluster} are described in the man page for the utility (\texttt{man(8) cm-scale-cluster}), along with the configuration file parameters and their key=value options, and also with a crontab example of how it may be run. A more extended explanation is given in the section on the meaning of configuration parameters beginning on page 294.

**Simple cm-scale-cluster example:** A simplified example of how \texttt{cm-scale-cluster} can be used with a simple configuration file is given in this section.

The example here considers a default cluster with a head node and 5 regular nodes, 3 of which are powered down at the start of the run. The power control for the nodes must be functioning properly, or otherwise \texttt{cm-scale-cluster} 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 \texttt{roleoverview} command should show something like:

**Example**
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 3.3.3 of the User Manual) can be built:

Example

[fred@bright73 ~]$ module add shared openmpi/gcc/64/1.10.1 slurm
[fred@bright73 ~]$ 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@bright73 ~]$ 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

The user fred can now flood the default queue, defq, with the batch file:

Example

[fred@bright73 ~]$ while (true); do sbatch slurmhello.sh; done

After putting enough jobs into the queue (a few thousand should be enough) the flooding can be stopped with a ctrl-c.

The activity in the queue can be watched:

Example

[root@bright73 ~]# watch "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:

© Bright Computing, Inc.
Example

[root@bright73 ~]# cmsh -c "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-cluster being used.
The next steps are to modify the behavior by bringing in cm-scale-cluster. A copy of the default configuration file can be prepared as follows:

Example

[root@bright73 ~]# cp /cm/local/apps/cm-scale-cluster/etc/default.conf /root/cmsc.conf

The configuration file is originally a fully-commented-out file. The line

```
QUEUE=defq NODEGROUP=pool
```

can be inserted into the file. This means that jobs in the default jobs queue, defq, in Slurm are handled by nodes in the nodegroup pool.

Example

[root@bright73 ~]# grep -v ^# /root/cmsc.conf | grep .

```
QUEUE=defq NODEGROUP=pool
```

The preceding `grep`s filter out the commented lines and the blank lines, so that the only active line of the file is displayed.
The nodegroup pool mentioned the configuration file should be created and the pool of nodes that cm-scale-cluster is to consider are added to it.

Example

[bright73->nodegroup]% add pool
[bright73->nodegroup*([pool*])]% append nodes node001..node005 ; commit

The pool of 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-cluster, depending on the number of jobs that are pending.
The cm-scale-cluster command can be run, with the configuration specified, as follows:

Example

[root@bright73 ~]# cm-scale-cluster -c /root/cmsc.conf

Following the log that this generates shows what it is doing when it runs:

Example

[root@bright73 ~]# tail -f /var/log/cm-scale-cluster.log

(Sep 07 2016 13:00:30) INFO: Job 5011 (PENDING): q=defq, nodes=1, slots=1, ppn=1, nodelist=[]
(Sep 07 2016 13:00:30) INFO: Job 5012 (PENDING): q=defq, nodes=1, slots=1, ppn=1, nodelist=[]
(Sep 07 2016 13:00:30) INFO: Total nodes number that cannot be found for all jobs [defq]: 1549
(Sep 07 2016 13:00:30) INFO: node003: Powering on, see events for result
(Sep 07 2016 13:00:30) INFO: node004: Powering on, see events for result
(Sep 07 2016 13:00:30) INFO: node005: Powering on, see events for result

© Bright Computing, Inc.
The log shows that cm-scale-cluster cluster, on seeing the number of jobs queued up, decides it makes sense to power on the spare compute nodes that are in the pool.

After a while, node003, node004, and node005 are up. Once up, they can start to process the jobs in the queue too.

Meanwhile, cm-scale-cluster should be run at intervals, for example, from a cronjob. Each time it is run, it checks if the nodes are being kept busy. If the nodes are idle, then they are powered down.

Watching the running jobs will show something like:

Example

```
[root@bright73 ~]# cmsh -c "jobs ; watch filter --running"
```

<table>
<thead>
<tr>
<th>Every 2.0s: filter --running</th>
<th>Thu Sep 8 13:01:59 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID Job name User Queue Submit time Start time Nodes Exit code</td>
<td>-------------------</td>
</tr>
<tr>
<td>16498 slurmhello.sh fred defq 07/09/2016 12:44:20 07/09/2016 13:20:58 node003 0</td>
<td></td>
</tr>
</tbody>
</table>

Eventually, a run of cm-scale-cluster finds that all jobs have been dealt with, and the nodes are then powered down.

Example

```
[root@bright73 ~]# tail -14 /var/log/cm-scale-cluster.log
```

A further explanation of the configuration file parameters is now given. Referring back to the example configuration file on page 291 should be helpful in understanding the parameters and their options when going through the explanations that follow:

- **QUEUE=<queue> [key=value]>…

  The QUEUE configuration file parameter sets, as an option, a queue that is to be watched. One queue is set per QUEUE parameter. Node groups, extra nodes, and additional options, are defined as key=value options for each queue.

    - NODEGROUP: The value of the NODEGROUP option to QUEUE is a convenient way to specify compute nodes using node groups (section 2.1.4), so that they can be started, stopped, or terminated according to jobs in the associated queue. 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, virtualnode, headnode,
mic, ethernetswitch and others, and are shown in the first column of the output of the default list command in device mode. Node group types currently (September 2016) consist only of Normal and Storage pair.

Only node groups of type Normal can currently be managed by cm-scale-cluster. These can be regular local compute nodes (nodes), cluster extension cloud compute nodes (cnodes), and Bright-managed OpenStack virtual compute nodes (vnodes). For the purposes of this section on cm-scale-cluster, these compute nodes can conveniently be called nodes, cnodes, and vnodes. Since compute nodes are typically the most common cluster nodes, significant resources can typically be saved by having the cm-scale-cluster utility decide on whether to bring them up or down according to demand.

- cnodes and vnodes 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.

The NODEGROUP option to QUEUE is mandatory if the TEMPLATE option to QUEUE is not set.

- TEMPLATE: The node specified for the TEMPLATE option of the QUEUE configuration file parameter must be one of the nodes specified for the TEMPLATE configuration file parameter. The TEMPLATE option defines a template node as a QUEUE configuration file parameter setting option, and then uses it to clone new nodes for that queue when the nodes specified by NODEGROUP are not enough. The following restrictions apply:
  - 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 node group defined for the queue.
  - A specific template node is restricted to a specific queue.

The TEMPLATE option to QUEUE is mandatory if the NODEGROUP option to QUEUE is not set.

- EXTRA_NODES: A node specified for the EXTRA_NODES option of the QUEUE configuration file parameter must be one of the extra nodes specified for the EXTRA_NODE configuration file parameter. A list of extra nodes can be assigned using commas to separate the extra nodes.

The EXTRA_NODES option are extra nodes that are associated with the queue besides compute nodes. The cm-scale-cluster utility:
  - starts the extra nodes before the first job is started
  - stops the extra nodes 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.

- NEVER_TERMINATE: Number of cloud nodes which will never be terminated even if no jobs need them. If there are this number or fewer cloud nodes, then cm-scale-cluster no longer terminates them. Cloud nodes that cannot be terminated can, however, still be powered off, allowing them to remain configured in Bright Cluster Manager.

As an aside, local nodes that are under cm-scale-cluster control are powered off automatically when no jobs need them, regardless of the NEVER_TERMINATE value. Default: 0.

- JOBS_PER_NODE: How many jobs can use a node at once. Default: 1.

- PRIMARY_OVERLAYS and SECONDARY_OVERLAYS: These options are comma-separated lists of configuration overlays. Only if PRIMARY_OVERLAYS has been set, is SECONDARY_OVERLAYS a possible, but non-mandatory option.
• When a job is in the queue that has \texttt{PRIMARY\_OVERLAYS} set, then the pool of \texttt{cm-scale-cluster}-managed nodes is checked.

The check is to decide on if a node is to be made available for the job.

If the node is appended to the \texttt{PRIMARY\_OVERLAYS} already and is not running jobs, then \texttt{cm-scale-cluster} simply hands over a job from the job queue to run on the node.

If the node is not appended to the \texttt{PRIMARY\_OVERLAYS} overlays already, but is in other overlays (which can also be the ones in \texttt{SECONDARY\_OVERLAYS}), and is not running jobs, then \texttt{cm-scale-cluster} prepares and boots the node as follows:

\begin{itemize}
\item the node is rebooted if it is up, or
\item the node is merely booted if it is not up
\end{itemize}

The node is removed from any previous overlays that it was with, before booting up, and it is appended to the new overlays of \texttt{PRIMARY\_OVERLAYS}.

• When there are no jobs in the queue that has \texttt{PRIMARY\_OVERLAYS}, then the pool of \texttt{cm-scale-cluster}-managed nodes is checked.

The check is to decide on if a node in the overlays of \texttt{PRIMARY\_OVERLAYS} can be made available for use elsewhere. This then becomes a decision on if the node should be appended to \texttt{SECONDARY\_OVERLAYS}.

If the node is not appended to the \texttt{SECONDARY\_OVERLAYS} overlays already, and is not running jobs, then \texttt{cm-scale-cluster} prepares and boots the node as follows:

\begin{itemize}
\item the node is rebooted if it is up, or
\item the node is merely booted if it is not up
\end{itemize}

The node is removed from \texttt{PRIMARY\_OVERLAYS} before booting up, and it is appended to the overlays of \texttt{SECONDARY\_OVERLAYS}.

\textbf{TEMPLATE=\langle template node\rangle \langle key=value\rangle...}

The \texttt{TEMPLATE} configuration file parameter sets a template node. One template node is set per \texttt{TEMPLATE} parameter. The template node set for this parameter is a node defined by Bright Cluster Manager.

Further \texttt{key=value} options specify the nodes that use the template node, along with other settings relevant to the template node.

The \texttt{cm-scale-cluster} utility creates new cloud nodes, by cloning the template if there are not enough nodes configured and running under CMDaemon control. The nodes that are cloned are listed in the \texttt{NODES} key to the \texttt{TEMPLATE} configuration file parameter.

A template node only has to exist as an object in the Bright 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 \texttt{START} and \texttt{STOP} values apply.

The \texttt{TEMPLATE} configuration file parameter takes the following \texttt{key=value} options:

\begin{itemize}
\item \texttt{START}: Is the template node allowed to start automatically? Default: \texttt{NO}
\item \texttt{STOP}: Is the template node allowed to stop automatically? Default: \texttt{NO}
\item \texttt{NODES}: The range of the new nodes that can be cloned from the template. The ways in which the range can be specified are given in \texttt{man(8)} \texttt{cm-scale-cluster}. Default: \texttt{<blank>}
\item \texttt{REMOVE}: Should the new node be removed from Bright Cluster Manager when the node terminates? If the node is not going to be terminated, but just stopped, then it will never be removed. Default: \texttt{NO}
\item \texttt{INTERFACE}: The network interface. Its IP address will be increased by 1 (IPv4) when the node is cloned. Default: \texttt{tun0}
\end{itemize}
7.9 Power Saving Features

- **LEAVE_FAILED_NODES**: This setting decides if nodes discovered to be in a state of INSTALLER_FAILED or INSTALLER_UNREACHABLE (section 5.5.4) can be left alone, so that the administrator can decide what do with them later on. Otherwise the cloud nodes are terminated automatically. Default: NO

- **EXTRA_NODE=**<extra node> [<key=value>...]

  The EXTRA_NODE configuration file parameter sets, as an option, an extra node. One extra node is set per EXTRA_NODE parameter. This extra node is a node defined by Bright Cluster Manager. It is a non-head node that is always needed for regular or cloud nodes to run. Typically, for cloud nodes it is a cloud director, while for regular nodes it is a provisioning node or a licensing server.

  Each extra node can take the following key=value options:

  - **IDLE_TIMEOUT**: The maximum time, in seconds, that extra nodes can remain unused. The cm-scale-cluster utility checks for the existence of queued and active jobs using the extra node when the time elapsed since the last check reaches IDLE_TIMEOUT. If there are jobs using the extra node, then the time elapsed is reset to zero and a time stamp is written into the file cm-scale-cluster.state under the directory set by the SPOOL_DIR parameter. The time stamp is used to decide when the next check is to take place. Setting IDLE_TIMEOUT=0 means the extra node is stopped whenever it is found to be idle, and started again whenever jobs require it, which may result in a lot of stops and starts. Default: 3600.
  
  - **START**: Automatically start extra node before the first compute node starts. Default: YES.
  
  - **STOP**: Automatically stop extra node after an idle period (as specified by IDLE_TIMEOUT). Default: YES.

- **WORKLOAD_MANAGER=**<workload manager>

  The WORKLOAD_MANAGER configuration file parameter sets the current workload manager as an option. Possible values are: slurm, pbspro, torque, uge, sge, lsf, openlava. Default: slurm

- **FILTER_MODULES=**<path>...

  The FILTER_MODULES file parameter is a configuration parameter that specifies comma-delimited paths to Python files that nodes that are used by cm-scale-cluster to be filtered. Such a filter allows, for example, a node to be excluded from switching its overlays if a virtual machine runs on the node.

  The filter file should implement these two functions:

  1. **setup(conf)**: this is called once when the module is loaded.

     The only parameter, conf, is a dictionary that keeps the global parameters of cm-scale-cluster. The setup function can add more parameters to the dictionary if needed, and it carries out some preparations. For example, in this function it makes sense to get information about the nodes from CMDaemon and to cache it in the global dictionary. For example:

     ```python
def setup(conf):
    hosts = get_os_hosts_info(conf)
    conf["MY_OPENSTACK_FILTER"] = hosts
```

     Here `get_os_hosts_info(conf)` should be implemented as one more function in the filter file.

  2. **allow_to_use(hostname, conf)**: this is called each time when cm-scale-cluster checks to see if is allowed to manipulate the node or not.

     For example if the node node003 is going to be moved to another overlay, then cm-scale-cluster calls allow_to_use("node003") and tests the result.

© Bright Computing, Inc.
If the result is False, then the node is skipped and is not moved to another overlay. The function should return quickly, because it can be called many times per `cm-scale-cluster` run.

A basic example of such a filter that does not allow `cm-scale-cluster` to manipulate `node003` is the following:

```python
#!/usr/bin/env python
import sys

sys.path.append("/cm/local/apps/cm-scale-cluster/lib")
import cmutils

def setup(conf):
    pass

def allow_to_use(hostname, conf):
    return hostname != "node003"
```

A simple example of a `cm-scale-cluster` configuration file may then look as follows:

```
QUEUE=defq NODEGROUP=pool PRIMARY_OVERLAYS=slurm
SECONDARY_OVERLAYS=OpenStackHypervisors NEVER_TERMINATE=10
FILTER_MODULES=/cm/local/apps/cm-scale-cluster/filters/openstack-filter.py
DEBUG=YES

• DEBUG=\<YES\|NO\>
  The DEBUG parameter allows `cm-scale-cluster` to append debug information to the log file. Default: NO

• PARALLEL_REQUESTS=\<number\>
  The PARALLEL_REQUESTS parameter sets a limit to the number of simultaneous requests to CMDaemon from `cm-scale-cluster` when requesting information about the changes being applied. Default: 10

• CMD_TIMEOUT=\<number\>
  The CMD_TIMEOUT parameter sets the maximum time, in seconds, that `cm-scale-cluster` waits for a response from CMDaemon. Default: 5

• LOG_FILE=\<file\>
  The LOG_FILE parameter sets the full path to the log file for `cm-scale-cluster`. Default: /var/log/cm-scale-cluster.log.

• LOCK_FILE=\<file\>
  The LOCK_FILE parameter prevents the simultaneous execution of several copies of `cm-scale-cluster` when the lock file is defined. Default: /var/lock/subsys/cm-scale-cluster

• SPOOL_DIR=\<directory\>
  The SPOOL_DIR parameter defines the directory where `cm-scale-cluster` creates temporary files. Default: /var/spool/cmd
```

© Bright Computing, Inc.
• **POLICY_MODULE=<file>**

The POLICY_MODULE parameter defines a path to a Python script containing policy rules. The policy rules define what kind of nodes should start, and how many, based on the current number of queued jobs requests. Default: /cm/local/apps/cm-scale-cluster/lib/default-policy.py

• **ACTION_MODULE=<file>**

Set a Python script containing action rules. The action rules define how nodes start, stop, or terminate. Default: /cm/local/apps/cm-scale-cluster/lib/default-action.py

• **AUTO_SLOTS=<YES\|NO>**

(LSF only). Use the number of node slots (slots number) dynamically retrieved from the workload manager. This can be useful when nodes change their slots number frequently. The slots number is retrieved only when the node is running, which means that a manual slots setting in the workload manager client role must be set by the administrator for use at other times. Default: NO

• **CLOUD_PROVIDER=<cloud provider>**

Set a cloud provider name. If both CLOUD_PROVIDER and CLOUD_REGION are set, then cm-scale-cluster tags each instance and its volumes in the cloud with the tag bright-cluster=<cluster name>. The cloud provider name must match the one in Bright Cluster Manager. Default: Not defined.

• **CLOUD_REGION=<cloud region>**

Set a cloud region name. If both CLOUD_REGION and CLOUD_REGION are set, then cm-scale-cluster tags each instance and its volumes in the cloud with the tag bright-cluster=<cluster name>. Default: Not defined.

**Configuration file parameters—in action:**

In the example configuration on page 291 two queues are defined: cloudq and localq.

- For **cloudq**: When all nodes from node group nodegroup1 are busy and new jobs are queued in cloudq, then cm-scale-cluster will use the template of cnode001 to clone the nodes cnode[002..128] to satisfy queued jobs needs. Before the first node from the cloudq queue is started, cm-scale-cluster makes sure that us-west-1-director extra node is running.

- For **localq**: In the case of the localq queue, no nodes will be created automatically, but they will be started (or stopped) nodes from the nodes of node group nodegroup2. Also, for jobs from localq, it will be taken into account that each node can run up to 4 jobs.

**Time Quanta Optimization**

*Time quanta optimization* is an additional feature that cm-scale-cluster can use for further cost-saving with certain cloud providers such as Amazon.

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 Bright 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

© Bright Computing, Inc.
Time quanta optimization is implemented with some guidance from the administrator for its associated parameters. These parameters are:

- **WHOLE_TIME**
- **RUN_INTERVAL**
- **STOPPING_ALLOWANCE_PERIOD**

These are explained next. Figure 7.24 illustrates a time line with the parameters used in time quanta optimization.

**WHOLE_TIME**: Currently (April 2015) Amazon’s time quantum is 60 minutes. The number of minutes per time quantum can be stored by the administrator, in the **WHOLE_TIME** setting of cm-scale-cluster for the cloud provider. By default, Bright Cluster Manager uses a value of **WHOLE_TIME=0**, which is a special value that means **WHOLE_TIME** is ignored. Ignoring **WHOLE_TIME** means that Bright 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.

**RUN_INTERVAL**: The cm-scale-cluster utility is run every few minutes, typically as a cron job. A value of every 5 minutes is recommended.

**Example**

```
*/5 * * * * /bin/bash -c "module add cm-scale-cluster; /cm/local/apps/cm-scale-cluster/bin/cm-scale-cluster -c /home/fred/cmsc.conf.sh >> /tmp/cron-cm-scale-cluster 2>&1"
```

The interval used between runs is copied over manually, and set as a value, in minutes for **RUN_INTERVAL**. By default, Bright Cluster Manager is set to a value of **RUN_INTERVAL=5**.

The algorithm that cm-scale-cluster follows, with and without time quanta optimization, can now be described using the two parameters explained so far:
1. cm-scale-cluster as part of its normal working, checks every RUN_INTERVAL minutes 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-cluster utility 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-cluster utility calculates the time period until the next time quantum from Amazon starts. This time period is the current closing time period. Its value changes each time that cm-scale-cluster 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-cluster run but also running within the current time quantum—is not long enough for the node to stop cleanly before the next quantum starts

      then the current closing time period starts at a time called the last call.

In drinking bars, the last call time by a bartender allows some time for a drinker to place the final orders. This allows a drinker to finish drinking in a civilized manner. The drinker is meant to stop drinking before closing time. If the drinker is still drinking beyond that time, then a vigilant law enforcement officer will fine the bartender.

Similarly, the last call time in a scaling cluster allows some time for a node to place its orders to stop running. It allows the node to finish running cleanly. The node is meant to stop running before the next time quantum starts. If the node is still running beyond that time, then a vigilant cloud provider will charge for the next whole-time period.

The last call time is the last time that cm-scale-cluster can run during the current whole-time period and still have the node stop cleanly within that current whole-time period, and before the next whole-time period starts. Thus, when whole-time has been set to a non-zero time:

   i. If the node is at the last call time, then the node begins with stopping

   ii. If the node is not at the last call time, then the node does not begin with stopping

The algorithm goes back again to step 1.

So, with whole-time set, a node that is idle but that cannot be stopped before the start of the next whole-time period, stays up for most of that next whole-time period, until the last call time for that whole-time period is reached again. At that time, if the node is still idle and with nothing scheduled to run on it, then the node is stopped.

The algorithm, when used over many nodes, can result in substantial cost savings.

STOPPING_ALLOWANCE_PERIOD: A third 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=10. 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-cluster stop the node.

**Dynamic Nodes Repurposing**

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. Different settings, or a different software image, then run on the re-assigned node after reprovisioning.

The feature is enabled using ASSIGNED CATEGORY parameter in the configuration file.

For example: Assuming two queues chem_q and phys_q. 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 for some reason the softwares cannot run on the node at the same time. Then, the nodes can be repurposed 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 queues configuration may then look as follows:

**Example**

```
QUEUE=chem_q ASSIGN_CATEGORY=chem_cat
QUEUE=phys_q ASSIGN_CATEGORY=phys_cat
```

Assuming that initially there are two nodes, node001 and node002, both in category chem_cat. Then, when cm-scale-cluster 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.

**Pending Reasons**

If cm-scale-cluster 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-cluster/pending_reasons/WLM.exclude, where WLM is a name of workload manager specified in the configuration file of the utility.

In the pending reasons exclude file, the pending reason texts are listed as one reason per line. The reasons are grouped in two sublists, with headers:

- [IGNORE_ALWAYS]
- [IGNORE_NO_NODE]

The [IGNORE_ALWAYS] sublist lists the type 1 pending reason texts. If a job has only this group of reasons, then cm-scale-cluster considers the job as ready to start, and attempts to create or boot compute nodes for it.
The [IGNORE_NO_NODE] sublist lists the type 2 pending reason texts. If the reason does not specify the hostname of a new free node at the end of a pending reason after the colon (":"). then the job can start on the node. If the reason does specify the hostname of a new free node after the colon, and if the hostname is owned by one of the managed nodes—nodes that can be stopped/started/created by cm-scale-cluster—then the job is considered as one that is not to start, when nodes become available.

If a job has a pending reason text that is not in the pending reasons exclude file, then it is assumed to be a type 3 reason. New free nodes for such a job do not get the job started.

If there are several pending reason texts for a job, then cm-scale-cluster checks all the pending reasons one by one. If all reasons are from the IGNORE_ALWAYS or IGNORE_NO_NODE sublists, and if a pending reason text matched in the IGNORE_NO_NODE sublist does not include hostnames for the managed nodes, only then will the job be considered as one that can be started just with new nodes.

Custom pending reasons: If the workload manager supports them, then custom pending reason texts are also supported. The administrator can add a pending reason text to one of the sections in the pending reasons exclude file.

The cm-scale-cluster utility checks only if the pending reason text for the job starts with a text from the pending reasons file. It is therefore enough to specify just a part of the text of the reason in order to make cm-scale-cluster take it into account. Regular expressions are also supported. For example, the next two pending reason expressions are equivalent when used to match the pending reason text Not enough job slot(s):

Example

• Not enough
• Not enough [a-z]* slot(s)

The workload manager statistics utility can be used to find out what custom pending reason texts there are, and to add them to the pending reasons file. To do this, some test job can be forced to have such a pending reason, and the output of the job statistics utility can then be copy-pasted. For example, LSF shows custom pending reasons that look like this:

Example

Customized pending reason number <integer>

Here, <integer> is an identifier (an unsigned integer) for the pending reason, as defined by the administrator.

Queue Node Placeholders

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-cluster is used. When cm-scale-cluster 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=FUTURE. Bright Cluster Manager allows Slurm to add such “fake” nodes to Slurm queues dynamically, when not enough real nodes have yet been added. Bright 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 server role it is possible to set a list of placeholder objects within the placeholder submode of cmsh. Each placeholder allows the following values to be set:

- **queue**: the maximum number of nodes
- **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
- **templatename**: 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 slurmserver role 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@bright73 ~]# scontrol show part defq | grep " Nodes="
Nodes=node001
[root@bright73 ~]# cmsh
[bright73]% device roles master
[bright73->device[bright73]->roles]% use slurmserver
[bright73->...]->roles[slurmserver]]% placeholders
[bright73->...->mserver]->placeholders[]% add defq
[bright73->...->placeholders[defq]]% set maxnodes 32
[bright73->...->placeholders[defq]]% set basenodename placeholder
[bright73->...->placeholders[defq]]% set templatenode node001
[bright73->...->placeholders[defq]]% commit
[bright73->...->placeholders[defq]]% show defq
[bright73->...->placeholders[defq]]% show 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 cmgui in the Slurm server role advanced options window.
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 behaviour 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 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-v1/cgroups.txt.

As far as workload management is concerned, it makes sense to distinguish between workload manager cgroup settings and system-wide cgroup parameters. The workload manager cgroup settings allow the administrator to configure a workload manager to use cgroups in a particular way, whereas the system-wide cgroup settings allow the administrator to manage cgroups whether a workload manager is used or not.

7.10.1 Cgroups Settings For Workload Managers

If the workload manager allows cgroups usage, then Bright 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 cgroups support:

1. /etc/slurm/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 using the AllowedDevicesFile option within the cgroups submode of the SlurmServer role.

For Slurm, the administrator can manage cgroups parameters using cmsh by going into the cgroups submode of the SlurmServer role. 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 a cgroup subsystems if they are not mounted yet</td>
<td>CgroupAutomount</td>
</tr>
<tr>
<td>Mount Point</td>
<td>Where cgroups 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 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
### 7.10 Cgroups

...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.

**UGE**

Univa Grid Engine allows cgroups settings to be defined at two levels.

- **Globally**: The parameters can be set globally in the global configuration. Bright 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. Bright Cluster Manager defines cgroups host settings in the `cgroups` submode of the UGEClient 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>
<tr>
<td>Freezer</td>
<td>If true, then it enables the cgroup freezer subsystem for job suspension and resumption</td>
<td>freezer</td>
</tr>
</tbody>
</table>

*continues

© Bright Computing, Inc.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<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 -mbind 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>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_hard</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>h_mem_free_soft</td>
</tr>
<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>
</tbody>
</table>

...continues
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.

**OGS**
The Open Grid Scheduler version currently-supported by Bright Cluster Manager does not support Linux cgroups.

**PBS Pro**
PBS Professional supports cgroups through a set of special python hooks. The account manager at Altair must be contacted by the administrator in order to install these hooks.

**Torque**
The Torque versions distributed with Bright Cluster Manager are built with cgroups support. In particular, --enable-cpuset is used with configure script. No special cgroups-related parameters are provided by Torque.

**LSF**
LSF allows resource enforcement to be controlled with the Linux cgroup memory and cpuset subsystems. By default, when LSF is set up with 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 cmgui, and can be found in the cgroups submode of the LSFServer role:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
<th>In lsf.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Enforce</td>
<td>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)</td>
<td>LSB_RESOURCE_ENFORCE</td>
<td>lsf.conf</td>
</tr>
</tbody>
</table>

...continues
**Workload Management**

...continued

<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</td>
<td>LSF_PROCESS_TRACKING</td>
</tr>
<tr>
<td></td>
<td>functions such as termination, suspension, resume, and other signals, on Linux systems which support the cgroups freezer subsystem</td>
<td></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 cpuacct subsystems</td>
<td>LSF_LINUX_CGROUP_ACCT</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

The options are updated in LSF only when cgroups is enabled in the LSFServer role. To enable cgroups, enabled must be set to true in the cgroups submode of the role. By default cgroups is set to enabled in LSF.

OpenLava

OpenLava allows batch jobs to enforce CPU and memory usage during runtime. The cgroups-related configuration options are available in cmsh or cmgui, and can be found in the cgroups submode of the OpenLavaServer role:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bind CPU</td>
<td>Enable this parameter to bind processes to CPU cores (default: true)</td>
<td>SBD_BIND_CPU</td>
</tr>
<tr>
<td>Mount Point</td>
<td>Where cgroups root is mounted (enables memory usage enforcement)</td>
<td>OL_CGROUP_ROOT</td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

By default the CPU and memory enforcement mentioned here are enabled in OpenLava.

### 7.10.2 Managed Cgroups

In order to control cgroups behaviour Bright Cluster Manager introduces a new node role: CgroupSupervisor. The role is used for two purposes:

- to help collect job metrics based on cgroups
- to allow the configuration of permanent cgroups, managed by CMDaemon

When a cgroup is created manually (for example, using the mkdir command, then by default it disappears after the node reboots. An administrator may wish to have persistent cgroups. The current method of creating permanent cgroups is using the cgconfig service. The service allows the creation of cgroup hierarchies, attaches the cgroup subsystems to the hierarchies, and manages the subsystems and the cgroups.

When the CgroupSupervisor role is assigned to a node or node category, then the cgconfig service is started and monitored by CMDaemon on that node or node category. CMDaemon manages the autogenerated section in the main configuration file of the service, /etc/cgconfig.conf. This file contains three types of entries—mount, group and template. CMDaemon manages the group and template entries on all Linux systems and in addition manages the mount entries on non-systemd systems.

© Bright Computing, Inc.
Each group entry in `/etc/cgconfig.conf` describes one hierarchy of cgroups. A template entry, described further on, can be helpful when cgroup rules are defined. A mount entry defines where cgroup controllers should be mounted.

In the CgroupSupervisor role, the administrator can add a new permanent cgroup within the `cgroups` submode of the role. Configuration of each cgroup defined in the role is written by CMDaemon to `/etc/cgconfig.conf`, and is managed by the cgconfig service. The cgroup object that can be accessed from `cmsh` or `cmgui` allows a list of cgroup controllers (accessible from the `controllers` submode in the cgroup object) to be defined. In turn, each controller object in the `controllers` submode has a list of properties that depend on the controller type. Available controller types are the following:

- `cgroupcontrollerblkio`: blkio controller
- `cgroupcontrollercpuset`: cpuset controller
- `cgroupcontrollerhugetlb`: hugetlb controller
- `cgroupcontrollernetprio`: net_prio controller
- `cgroupcontrollercpu`: cpu controller
- `cgroupcontrollerdevices`: devices controller
- `cgroupcontrollermemory`: memory controller
- `cgroupcontrollerns`: ns (namespaces) controller
- `cgroupcontrollercpuacct`: cpuacct controller
- `cgroupcontrollerfreezer`: freezer controller
- `cgroupcontrollernetcls`: net_cls controller
- `cgroupcontrollerperf`: perf_event controller

In `cmsh`, the permanent (managed) cgroup is created and configured from within the CgroupSupervisor role:

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# device roles master
[bright73->device[bright73]->roles]# assign cgroupsupervisor; commit
[bright73->device[bright73]->roles[cgroupsupervisor]]# show
Parameter          Value
-------------------------------- ----------------------------------
Cgroups             <2 in submode>
Collect Metrics     no
Name                 cgroupsupervisor
Root                 /sys/fs/cgroup
Type                 CgroupSupervisorRole

[bright73->device[bright73]->roles[cgroupsupervisor]]# cgroups
[bright73->device[bright73]->roles[cgroupsupervisor->cgroups]]# add cg1
[bright73->device[bright73]->roles[cgroupsupervisor->cgroups* [cg1*]]]# show
Parameter          Value
-------------------------------- ----------------------------------
Lookup              cg1
Controllers         <0 in submode>
Is Template         no
Managed              yes
```

© Bright Computing, Inc.
Different cgroup controller types have different set of properties. When a new controller is added into the managed cgroup, then the controller name is required, but it can be any string. If the Enabled parameter for a controller is set to false, then cgroup configuration will not include the settings of that controller.

Each Group entry in /etc/cgconfig.conf creates a new cgroups hierarchy, sets permissions for the hierarchy and for the tasks virtual files, and configures subsystems attached to the hierarchy. The cgroup that is created in the example above would lead to the following content in /etc/cgconfig.conf:

Example

```
[root@bright73 ~]# cat /etc/cgconfig.conf

# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
group cg1
    perm
        admin
            uid = root;
            gid = root;

        task
            uid = root;
            gid = root;
```
For operating systems that run without systemd (RHEL/SL/CentOS < 7.0 and SLES < 12) a managed cgroup also allows rules to be configured that allow processes to be moved into this cgroup automatically. For this functionality the cgred service is used. This starts the cgrulesengd daemon. For systemd systems this functionality is not available, because systemd prevents the cgred service from doing its job. At the time of writing (January 2016) systemd does not as yet provide functionality similar to the cgred service.

The cgrulesengd daemon (started by the cgred service) reads the process movement rules from /etc/cgrules.conf, which can be managed by CMDaemon. In order to define a new rule, the administrator can add it in the rules submode in cgroup mode as follows:

**Example**

```bash
[bright73->device[bright73]->roles[cgroupsupervisor]->cgroups[cg1]]% rules
[bright73->device[bright73]->roles[cgroupsupervisor]->cgroups[cg1]->rules]% add alice
[bright73->...[cgroupsupervisor*]->cgroups*[cg1*]->rules*[alice*]]% set controllernames cpuset
[bright73->...[cgroupsupervisor*]->cgroups*[cg1*]->rules*[alice*]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>cpuset</td>
</tr>
<tr>
<td>User</td>
<td>alice</td>
</tr>
</tbody>
</table>

The rule name is the name of the user the rule will be applied to. It is possible to use an asterisk to define a rule for all users. The command in the rule can be set in order to apply rules only for processes spawned by that command. For example if Command is set to "ftp", then when the user runs the ftp command, the FTP process will automatically be moved to the specified cgroup, and restrictions set for the cgroup will automatically be applied to that FTP process.

The rule that was created in the example above would lead to the following content of the cgred configuration file:

**Example**

```bash
[root@bright73 ~]# cat /etc/cgrules.conf
```

```
# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
alice cpuset cg1
# END AUTOGENERATED SECTION -- DO NOT REMOVE
```

It is also possible to define a template cgroup that will not be created in a filesystem until a process of the specified user is started. There are several variables that can be used for specifying cgroup paths in such templates:
• %u – is replaced with the name of the user who owns the current process. If name resolution fails, UID is used instead.

• %U – is replaced with the UID of the specified user who owns the current process.

• %g – is replaced with the name of the user group that owns the current process, or with the GID if name resolution fails.

• %G – is replaced with the GID of the cgroup that owns the current process.

• %p – is replaced with the name of the current process. PID is used in case of name resolution failure.

• %P – is replaced with the PID of the current processes.

For example if the cgroup name is users/%u, then setting a rule for user alice in the cgroup will lead to creation of a new cgroup users/alice and place the new processes of the user into that cgroup automatically.

The man pages for cgconfig.conf and cgred.conf have more details on cgroup templates.
Containerization

Containerization is a technology that allows processes to be isolated by combining cgroups, Linux namespaces, and 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 component of a container is the container image, which 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.

Cgroup, 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 8.1), and Kubernetes (section 8.2). Docker manages containers on individual hosts, while Kubernetes manages containers across a cluster. Bright Cluster Manager integrates both of these solutions, so that setup, configuration and monitoring of containers becomes an easily-managed part of Bright Cluster Manager.

8.1 Docker Engine

Docker integration for Docker version 1.11.1 is supported by Bright Cluster Manager 7.3 for RHEL/SL/CentOS versions 7.0 and above, and for SLES versions 12 and above.

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. It does not allow containers to spawn on several hosts, and is responsible only for the host where the Docker daemon runs. Docker provides a utility called docker, and two daemons called docker and containerd.

Additional functionality includes pulling the container image from a specific image registry, configuring the containers network, setting systemd limits, and attaching volumes.

Docker Hub Remote Registry

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. By default, Docker searches for images in Docker Hub, which is a cloud-hosted public and private registry service. 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, then Docker verifies if the latest version of the image has already been downloaded. If it has not, then Docker 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 Services

Besides using Docker Hub for the registry service, the administrator can also install a local registry service on the cluster. There are two types of such registry services provided by Docker Inc. The first one is the open source version of the registry, and can be useful if the registry is used by trusted users. This version can be installed with the Bright Cluster Manager utility `cm-docker-registry-setup`.

The second registry that can be installed is the Docker Trusted Registry, which is a proprietary, commercial version supported by Docker Inc. The commercial version provides extended functionality. The administrator can download and install this manually according to the official Docker Inc. instructions.

#### 8.1.1 Docker Setup

Bright Cluster Manager provides the following Docker packages:

- `cm-docker-engine`: docker binaries
- `cm-containerd`: tools to control runC
- `cm-runc`: CLI tool for containers
- `cm-python-docker-py`: a python library that provides python hooks to the Docker API

These packages can be installed by the administrator with the `yum` or `zypper` commands.

Typically, however, the administrator is expected to simply run the Bright Cluster Manager `cm-docker-setup` utility. The utility takes care of the installation of `cm-docker-engine` (docker binaries) and `cm-python-docker-py` packages and also takes care of Docker setup. For Bright Cluster Manager 7.3 the utility must be run with the `-e|--engine` option.

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, which volume backend to configure, and so on. If `cm-docker-setup` is run with the `-y` option, then the default values are used.

When the questions have been answered, the utility:

- installs the `cm-docker-engine`, `cm-containerd`, `cm-runc` and `cm-python-docker-py` packages, if they are not yet installed
- then assigns the DockerHost and Containerd roles to the node categories or head nodes that were specified
- adds healthchecks to the Bright Cluster Manager monitoring configuration
- performs the initial configuration of Docker.

**Example**

```
[root@bright73 ~]# cm-docker-setup -e -y
Specify additional docker registries (default: none):
Specify node categories (default: "default"): 
  default,
  none,
```
vnodes
Which storage backend will be used (default: devicemapper):
  custom, devicemapper
Will block device be used (default: no):
  no, yes
Specify loopback file size in GB for containers data (default: 100)
Specify loopback file size in GB for metadata (default: 2):
Run docker on head node (default: "no"):
  no, yes

done

The regular nodes on which Docker is to run should then be restarted. The restart operation provisions the updated images from the image directory onto the nodes.

The cm-docker-engine 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@bright73 ~]# ssh node001
[root@node001 ~]# module load docker
[root@node001 ~]# docker info
Containers: 0
Images: 0
...
Docker Root Dir: /var/lib/docker
[root@node001 ~]#

Or, for example, importing a container with Docker may result in the following output:

Example
[root@node001 ~]# module load docker
[root@node001 ~]# docker ps --format ".ID: .Image"
270f80462a79: perl
c422ed0dc9dc: perl
99bf78eb18de: busybox
ff2d0a29ce9f: gcr.io/google_containers/pause:0.8.0
7833a3df45b: gcr.io/google_containers/pause:0.8.0
ce487ea2a9ec: gcr.io/google_containers/pause:0.8.0
[root@node001 ~]#

Regular users should however use Kubernetes instead of Docker commands. After Docker has been installed, Kubernetes can be set up, as covered in section 8.2.

8.1.2 Integration With Workload Managers
Bright 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.
- OpenLava – OpenLava provides a job starter script. If the script is configured as a job starter, then all jobs start in Docker containers. Details on this are available at https://github.com/openlava/openlava/blob/master/examples/docker/docker_integration.txt.
• 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.

8.1.3 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.

The parameters that CMDaemon can configure in the DockerHost role, and the corresponding Docker options, are shown in table 8.1:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option To docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug*</td>
<td>Enable debug mode (default: no)</td>
<td>-D</td>
</tr>
<tr>
<td>Default Ulimits</td>
<td>Set the default ulimit options for all containers</td>
<td>--default-ulimit</td>
</tr>
<tr>
<td>Enable SELinux*</td>
<td>Enable selinux support in Docker daemon (default: yes)</td>
<td>--selinux-enabled</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>
<td>-l</td>
</tr>
<tr>
<td>Spool</td>
<td>Root of the Docker runtime (default: /var/lib/docker)</td>
<td>-g</td>
</tr>
<tr>
<td>Tmp dir</td>
<td>Location for temporary files. Default: $&lt;spool&gt;/tmp, where $&lt;spool&gt; is replaced by the path to the Docker runtime root directory</td>
<td>$DOCKER_TMPDIR</td>
</tr>
<tr>
<td>API Sockets</td>
<td>Daemon socket(s) to connect to (default: unix:///var/run/docker.sock)</td>
<td>-H</td>
</tr>
<tr>
<td>Bridge</td>
<td>Attach containers to a network bridge (not defined by default)</td>
<td>-b</td>
</tr>
<tr>
<td>Iptables*</td>
<td>Enable Iptables rules (default: no)</td>
<td>--iptables</td>
</tr>
<tr>
<td>MTU</td>
<td>Set the containers network MTU, in bytes (default: 1500)</td>
<td>--mtu</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>
<td>--users-remap</td>
</tr>
<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>
<td>--insecure-registry</td>
</tr>
<tr>
<td>Certificates Path</td>
<td>Path to Docker certificates (default: /etc/docker)</td>
<td>$DOCKER_CERT_PATH</td>
</tr>
<tr>
<td>Enable TLS*</td>
<td>Use TLS (default: no)</td>
<td>--tls</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>TLS CA</td>
<td>Trust only certificates that are signed by this CA</td>
<td>--tlscacert</td>
</tr>
<tr>
<td></td>
<td>(not defined by default)</td>
<td></td>
</tr>
<tr>
<td>TLS Certificate</td>
<td>Path to TLS certificate file (not defined by default)</td>
<td>--tlscert</td>
</tr>
<tr>
<td>TLS Key</td>
<td>Path to TLS key file (not defined by default)</td>
<td>--tlskey</td>
</tr>
<tr>
<td>Verify TLS*</td>
<td>Use TLS and verify the remote (default: no)</td>
<td>--tlsverify</td>
</tr>
<tr>
<td>Storage Backends</td>
<td>Docker storage backends</td>
<td></td>
</tr>
<tr>
<td>Containerd Socket</td>
<td>Path to containerd socket (default: /run/</td>
<td>--containerd</td>
</tr>
<tr>
<td></td>
<td>containerd/containerd.sock)</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>Additional parameters for docker daemon</td>
<td></td>
</tr>
</tbody>
</table>

* Boolean (takes yes or no as a value)

Table 8.1: DockerHost role and docker options

8.1.4 Iptables

Iptables rules can be added to nodes that function as a Docker host, to let network traffic go from the containers to outside the pods network. For example, if the docker::host has already been assigned to the nodes via the default category, then the Iptables rules that are set can be enabled by setting the iptables parameter in the Docker::Host role to yes:

Example

```
[root@bright73 ~]# cmsh
[bright73]# category use default
[bright73->category[default]]# roles
[bright73->category[default]->roles]# use docker::host
[bright73->category[default]->roles[Docker::Host]]# set iptables yes
[bright73->category*[default*]->roles*[Docker::Host*]]# commit
```

8.1.5 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 backends, also called storage drivers, that rely heavily on various filesystem features in the kernel or volume manager. Some storage backends perform better than others, depending on the circumstances.

The default storage backend configured by cm-docker-setup is devicemapper. Storage backends supported by Docker are listed in table 8.2:

<table>
<thead>
<tr>
<th>Technology</th>
<th>Description</th>
<th>Backend Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFS</td>
<td>Very simple fallback backend that has no copy-on-write support. Creating</td>
<td>vfs</td>
</tr>
<tr>
<td></td>
<td>a new layer based on another layer is done by making a deep copy of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>base layer into a new directory. Not recommended for production usage.</td>
<td></td>
</tr>
</tbody>
</table>

...continues

© Bright Computing, Inc.
Device Mapper  
This 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. 

Btrfs  
It is included in the mainline Linux kernel and its on-disk-format is now considered stable. However, many of its features are still under heavy development. 

AUFS  
This was the first first storage backend that Docker used. It includes the following features: fast container startup time, efficient use of storage, and efficient use of memory. AUFS is not included in the mainline Linux kernel, and some Linux distributions also do not support AUFS. 

ZFS  
This is a filesystem that supports many advanced storage technologies, including: volume management, snapshots, checksumming, compression, deduplication, and replication. ZFS can be installed on Linux, however, for now the ZFS Docker storage driver is not recommended for production use. 

OverlayFS  
This is a modern union filesystem that is similar to AUFS, but with a simpler design, and also potentially faster. It has been in the mainline Linux kernel since version 3.18. OverlayFS is still relatively young, and administrators should therefore be cautious before using it in production Docker environments. 

Table 8.2: Docker storage backends

To find out which storage driver is set on the daemon, the `docker info` command can be used:

Example

```
[root@bright73 ~]# module load docker
[root@bright73 ~]# docker info
Containers: 74
Images: 18
Storage Driver: devicemapper
Pool Name: docker-253:3-2819682-pool
Pool Blocksize: 65.54 kB
Backing Filesystem: extfs
Data file: /dev/loop0
Metadata file: /dev/loop1
Data Space Used: 2.687 GB
Data Space Total: 107.4 GB
Data Space Available: 25.44 GB
Metadata Space Used: 6.55 MB
Metadata Space Total: 2.147 GB
Metadata Space Available: 2.141 GB
Udev Sync Supported: true
Deferred Removal Enabled: false
Data loop file: /var/lib/docker/devicemapper/devicemapper/data
Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
...>
```
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 backends is available at https://docs.docker.com/engine/userguide/storagedriver.

**Device Mapper Driver Settings Support**

Bright Cluster Manager supports only device mapper driver settings, but the settings of other backends can be added by using the Options parameter in the DockerHost role. By default the device mapper storage backend is added automatically, and can be configured in the storagebackends submode of the DockerHost role:

**Example**

```
[bright73->device[bright73]->roles[dockerhost]]% storagebackends
[bright73->device[bright73]->roles[dockerhost]->storagebackends]% use devicemapper
[bright73->device[bright73]->roles[dockerhost]->storagebackends[devicemapper]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blk Discard</td>
<td>yes</td>
</tr>
<tr>
<td>Block Size</td>
<td>64K</td>
</tr>
<tr>
<td>Filesystem</td>
<td>ext4</td>
</tr>
<tr>
<td>Loop Data Size</td>
<td>10</td>
</tr>
<tr>
<td>Loop Device Size</td>
<td>100GB</td>
</tr>
<tr>
<td>Loop Metadata Size</td>
<td>1</td>
</tr>
<tr>
<td>Mkfs Arguments</td>
<td></td>
</tr>
<tr>
<td>Mount Options</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>devicemapper</td>
</tr>
<tr>
<td>Pool Device</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>DockerStorageDeviceMapperBackend</td>
</tr>
</tbody>
</table>

The device mapper backend parameters are described in table 8.3:

<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: ext4)</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 (default: 100GB)</td>
<td>dm.basesize</td>
</tr>
</tbody>
</table>

...continues

© Bright Computing, Inc.
8.1 Docker Engine

...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to docker</th>
</tr>
</thead>
<tbody>
<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.mkfsargs</td>
</tr>
<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 8.3: Device mapper backends and docker options

8.1.6 The Container Daemon Service And RunC Runtime

ContainerdHost Role

When cm-docker-setup is executed, the ContainerdHost role is assigned to nodes or categories together with the DockerHost role. The ContainerdHost role runs the container daemon, as a service called containerd. The daemon controls the runC runtime, which is the default execution component of Docker Engine.

The RunC Runtime

The runC runtime handles OCI (Open Container Initiative) images and manages their lifecycle according the Open Containers runtime specification. Users can replace the runC binary on their system with an alternative runtime—for example with execution components that have a preferred security or performance, as they become available—but still reap the benefits of using the Docker API.

Parameters And Configuration For ContainerdHost And RunC

The parameters that CMDaemon can configure in the ContainerdHost role, and the corresponding containerd options, are shown in table 8.4:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option To containerd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug*</td>
<td>Enable debug mode (default: no)</td>
<td>--debug</td>
</tr>
<tr>
<td>Listen Address</td>
<td>Address on which GRPC API will listen (default: /run/containerd/containerd.sock)</td>
<td>--listen</td>
</tr>
<tr>
<td>Runtime</td>
<td>Name of the OCI-compliant runtime to use when executing containers (default: runc)</td>
<td>--runtime</td>
</tr>
<tr>
<td>Runtime Arguments</td>
<td>Additional runtime arguments</td>
<td>--runtime-args</td>
</tr>
<tr>
<td>Spool</td>
<td>Runtime state directory (default: /run/containerd)</td>
<td>--state-dir</td>
</tr>
</tbody>
</table>

...continues
Parameter | Description | Option To
---|---|---
Start Timeout | Timeout duration (in seconds) for waiting on a container to start before it is killed (default: 60) | --start-timeout
Retain Count | Number of past events to keep in the event log (default: 500) | --retain-count
Options | Additional parameters for containerd daemon |

* Boolean (takes yes or no as a value)

Table 8.4: ContainerdHost role settings And containerd options

The root directory path for Container daemon setup takes the following format:
/cm/local/apps/containerd/<version>.

Packages And Installation

The cm-containerd RPM package: is available from the Bright repository. It is installed automatically by the cm-docker-setup package, and it provides or sets up:
- the containerd service
- the containerd environment module file (section 2.2 of this manual, and 7.1 of the Installation Manual). Users must load this when accessing the containerd service.
- the ctr command-line utility. The ctr utility allows users to interact with the Container daemon. If Docker is used, then the ctr utility is not normally needed during regular use, but it can be used for debugging purposes.

Example

[root@bright73 ~]# module load containerd
[root@bright73 ~]# ctr containers
ID PATH STATUS PROCESSES
964f59[...] /var/run/docker/libcontainerd/964f59[...] running init
f2107a[...] /var/run/docker/libcontainerd/f2107a[...] running init
0483b3[...] /var/run/docker/libcontainerd/0483b3[...] running init
b48daf[...] /var/run/docker/libcontainerd/b48daf[...] running init
%

The cm-runc RPM package: is available from the Bright repository. It is installed automatically by cm-docker-setup, and it provides or sets up:
- the default Docker runtime, which is the runc binary. The runc binary can create and manage OCI containers. A user can also create a container manually with it, but typically users don’t use it directly if Docker is installed.
- the runc environment module file. Users must load this before using runC.

8.1.7 Docker Monitoring

When cm-docker-setup runs, it configures and runs the following Docker healthchecks:

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
8.2 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

Bright 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 http://kubernetes.io/docs/.

Bright Cluster Manager 7.3 at the time of writing (August 2016) supports Kubernetes v1.3.0 integration on RHEL/SL/CentOS versions 7.0 and higher, and on SLES version 12 and higher.

8.2.1 Kubernetes Setup

Bright Cluster Manager provides four Kubernetes-related packages:

1. cm-kubernetes-master
2. cm-kubernetes-node
3. cm-etcd
4. cm-flannel

The packages cm-kubernetes-master, cm-etcd, and cm-flannel, are installed on the head node. The packages cm-kubernetes-node and cm-flannel are installed on all Kubernetes cluster nodes. The package cm-etcd contains distributed key-value storage binaries in etcd that are required by Kubernetes and Flannel. The package cm-flannel is used to provide a virtual network that gives a subnet to each host for use with container runtimes. Pods then have reachable IP addresses across the Kubernetes cluster (section 8.2.6).

The packages are installed automatically from the repository when the administrator runs cm-kubernetes-setup from the command line, or uses the Kubernetes setup wizard from cmgui.

Kubernetes Setup From the Command Line

The command line utility has the following usage synopsis:

[root@bright73 ~]# cm-kubernetes-setup -h

optional arguments:
-h, --help show this help message and exit
-y, --yes Use default answers to the question (batch mode)
-d, --disable Disable Kubernetes setup
-c, --certificates Only regenerate kubeca and kube certificates
-u USER, --user USER Generate and sign and deploy user certificate for specified username
-l LOCATION, --location LOCATION Put generated certificates into this directory
--dry-run Dry run

The cm-kubernetes-setup utility should be executed on the console, or alternatively the administrator can run the Kubernetes setup wizard in cmgui.

© Bright Computing, Inc.
Dealing With A Pre-existing Docker Installation: Docker is required for Kubernetes configured by Bright Cluster Manager. The setup wizard checks if Docker has been installed (page 316), 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 disabled with the `cm-docker-setup -d` command.

In both Kubernetes installation methods (`cm-kubernetes-setup` or `cmgui` wizard), the administrator answers several questions. These include questions about the node categories which should be configured to run the Kubernetes services, the service and pod networks parameters, the port numbers that will be configured for the daemons and so on. The `-y|--yes` option to `cm-kubernetes-setup` uses the default values. After the questions have been answered, appropriate roles will be assigned, the new Kubernetes cluster will be added, health checks will be added to the monitoring configuration, and Kubernetes daemons certificates will be generated:

**Example**

```
[root@bright73 ~]# cm-kubernetes-setup -y
Installing cm-kubernetes-node (root=/cm/images/default-image)
Installing cm-flannel (root=/cm/images/default-image)
Installing cm-etcd (root=/)
Installing cm-flannel (root=/)
Installing cm-kubernetes-master (root=/)
Installing cm-kubernetes-node (root=/)
Configuring monitoring ...
Kubernetes has been setup successfully.
The compute nodes where pods will run must be rebooted.

[root@bright73 ~]#
```

**Kubernetes Setup From cmgui**

A GUI version of Kubernetes setup can be run using the Kubernetes setup wizard from `cmgui`.

**Kubernetes setup from cmgui—accessing the wizard:** The wizard can be accessed via the Kubernetes resource, within in the left hand panel of `cmgui` (figure 8.1). If Kubernetes has not yet been configured, then a Setup Wizard button is displayed.

![Figure 8.1: Kubernetes Setup Button](image)

Clicking on the button starts up the wizard, and brings up a main overview screen.
**Kubernetes setup from cmgui—main overview screen:** The main overview screen (figure 8.2) provides an overview of how the wizard is to run, and asks for the following input:

- Should the wizard run in step-by-step mode, or in express mode?
  - Step-by-step node asks many explicit configuration options, and can be used by the administrator to become familiar with the configuration options.
  - Express mode asks for no configuration options, and uses default settings. It can be used by an administrator who would like to try out a relatively standard configuration.

![Kubernetes Setup Wizard Overview](image)

**Figure 8.2: Kubernetes Setup Wizard Overview**

During the wizard procedure, buttons are available at the bottom of the screen to go forward to the next screen or back to a previous screen.

On clicking the **Next** button in the overview screen:

- If the express mode has been chosen, then the wizard skips the in-between steps, and jumps straight ahead to the summary screen (page 329)

- Otherwise, if the step-by-step mode has been chosen, then each time the **Next** button is clicked, the wizard goes to the next screen in the series of in-between steps. Each screen allows options to be configured.

**Kubernetes setup from cmgui—cluster settings screen:** Following the path of the step-by-step mode, the next screen after overview mode is the cluster settings screen. This allows the administrator to set a Kubernetes cluster name (figure 8.3).

![Kubernetes Cluster Settings Screen](image)

**Figure 8.3: Kubernetes Cluster Settings Screen**

**Kubernetes setup from cmgui—node settings screen:** The node settings screen allows the administrator to set the categories within which Kubernetes can run jobs and pods nodes. The administrator
can also specify if the pods may run on the head node (figure 8.4).

![Kubernetes Nodes Settings Screen](image)

**Figure 8.4: Kubernetes Nodes Settings Screen**

**Kubernetes setup from cmgui—Docker settings screen:** The Docker settings screen allows the administrator to set up the Docker deployments on the nodes previously selected. (figure 8.5). The administrator can add additional registries and select the storage backend: default or devicemapper, with support for loopback or block devices.

![Docker Settings Screen](image)

**Figure 8.5: Kubernetes Docker Settings Screen**

**Kubernetes setup from cmgui—networks screen:** The Kubernetes networks screen allows the administrator to set parameters for pod and service networks (figure 8.3). The networks will be created in Bright Cluster Manager (section 8.2.6). Besides the networks settings, the screen allows the administrator to set the individual ports for Kubernetes daemons.
**Kubernetes setup from cmgui—summary screen:** The summary screen displays a summary of the configuration. The configuration may still be changed with cmgui if the administrator goes back through the screens to adjust settings.
Kubernetes setup from cmgui—XML configuration screen: The full configuration is kept in an XML file. The XML file can be viewed by clicking on the Show Configuration button, or it can be saved with the Save Configuration button (figure 8.7). The resulting read-only view is shown in figure 8.8.

The XML file can be used as the default configuration settings for the text-based cm-kubernetes-setup utility, if it is run as:
8.2 Kubernetes

[root@bright73 ~]# cm-kubernetes-setup -x <XML file>

Kubernetes setup from cmgui—deployment screen: Clicking on the Deploy button that appears in figure 8.7 sets up a Kubernetes cluster in the background. The direct background progress relies on the text-based cm-kubernetes-setup script, and is hidden from the administrator during a cmgui wizard run. Some log excerpts from the script output are displayed within the cmgui deployment screen (figure 8.9). The deployment can be done in dry-mode (the default) to see if all the parameters validate correctly.

![Kubernetes Cluster Deployment Screen](image)

Figure 8.9: Kubernetes Cluster Deployment Screen

At the end of its run, the cluster should have Kubernetes set up and running in an integrated manner with Bright Cluster Manager. The administrator should reboot the compute nodes on which pods are to run, in order to provision the installed Kubernetes files.

The administrator can now configure the cluster to suit the particular site requirements.

8.2.2 Disabling Kubernetes

The Kubernetes configuration can be disabled using the text-based cm-kubernetes-setup utility, or using cmgui.

With cm-kubernetes-setup the -d option is used:

[root@bright73 ~]# cm-kubernetes-setup -d
Disabling Kubernetes ...

Kubernetes has been successfully disabled.

[root@bright73 ~]#

Disabling Kubernetes removes the Kubernetes cluster configuration from Bright Cluster Manager, unassigns Kubernetes-related roles—including the EtcdHost role—and removes Kubernetes health checks. The command does not remove packages that were installed with a cm-kubernetes-setup command before that.

A similar procedure can be performed in cmgui using the Tasks tabbed window in the Kubernetes resource panel (figure 8.10).

© Bright Computing, Inc.
Clicking on the **Disable** button starts Kubernetes cluster disabling process in the background. The direct background progress is hidden from the administrator, and relies on the text-based `cm-kubernetes-setup` script. At the end of its run, the cluster has no Kubernetes configured and running.

### 8.2.3 Kubernetes Cluster

Kubernetes allows several Kubernetes clusters to be configured. These are separated sets of hosts with different certificates, users and other global settings. Bright Cluster Manager 7.3 supports only one Kubernetes cluster. During the Kubernetes setup the 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 resource of `cmgui`.

**Example**

```
[root@bright73 ~]# cmsh
[bright73]# kubernetes list
Name (key)------------------
default
[bright73]# kubernetes use default
[bright73->kubernetes[default]]# show
Parameter Value
              -----------------------------------------------
Notes
Name default
Authorization Mode ABAC
Authorization Policy File /cm/local/apps/kubernetes/var/etc/auth-policy.jsonl
User Policies <2 in submode>
Kube Config /cm/local/apps/kubernetes/var/etc/kubeconfig
Kube Config Template <371 bytes>
CA /cm/local/apps/kubernetes/var/etc/kubeca.pem
```

© Bright Computing, Inc.
Kubernetes Certificate: /cm/local/apps/kubernetes/var/etc/kube.pem
Kubernetes Key: /cm/local/apps/kubernetes/var/etc/kube.key
Insecure API Port: 8085
Secure API Port: 6443
Kubelet Port: 10250
Health Check Port: 10251
Controller Port: 10252
Scheduler Port: 10253

The preceding Kubernetes mode parameters are described in table 8.5:

<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: ABAC (Attribute-Based Access Control))</td>
</tr>
<tr>
<td>Authorization Policy File</td>
<td>File with authorization policy</td>
</tr>
<tr>
<td>User Policies</td>
<td>User authorization policies</td>
</tr>
<tr>
<td>Kube Config</td>
<td>Path to a kubeconfig file, specifying how to authenticate to API server</td>
</tr>
<tr>
<td>Kube Config Template</td>
<td>Template for system kubeconfig file</td>
</tr>
<tr>
<td>CA</td>
<td>If set, any request presenting a client certificate signed by one of the authorities in the --client-ca-file option used by the Kubernetes API server is authenticated with an identity corresponding to the CommonName of the client certificate</td>
</tr>
<tr>
<td>Kubernetes Certificate</td>
<td>File containing x509 certificate used by the Kubernetes daemons</td>
</tr>
<tr>
<td>Kubernetes Key</td>
<td>File containing x509 private key used by the Kubernetes daemons</td>
</tr>
<tr>
<td>Insecure API Port</td>
<td>The port on which to serve unsecured, unauthenticated access</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.</td>
</tr>
<tr>
<td>Kubelet Port</td>
<td>Port that the HTTP service of the node runs on</td>
</tr>
<tr>
<td>Health Check Port</td>
<td>The port to bind the health check server (use 0 to disable)</td>
</tr>
<tr>
<td>Controller Port</td>
<td>Port that the controller-manager runs on</td>
</tr>
</tbody>
</table>

© Bright Computing, Inc.
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduler Port</td>
<td>Port that the HTTP service of the scheduler runs on</td>
</tr>
</tbody>
</table>

Table 8.5: kubernetes mode parameters

8.2.4 Kubernetes Roles

Kubernetes roles include the following roles:

- EtcdHost (page 334)
- KubernetesApiServer (page 335)
- KubernetesController (page 336)
- KubernetesScheduler (page 338)
- KubernetesProxy (page 338)
- KubernetesNode (page 339)

When nodes are configured using Kubernetes roles, then settings in these roles may sometimes use the same pointer variables—for example the certificates path, or the Kubernetes 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.

EtcdHost Role

The EtcdHost role is used to configure and manage the etcd service. 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 8.6:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to etcd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Name</td>
<td>The human-readable name for this etcd member ($hostname will be replaced by the node hostname)</td>
<td>--name</td>
</tr>
<tr>
<td>Spool</td>
<td>Path to the data directory</td>
<td>--data-dir</td>
</tr>
<tr>
<td>Advertise Client URLs</td>
<td>List of client URLs for this member to advertise publicly</td>
<td>--advertise-client-urls</td>
</tr>
<tr>
<td>Advertise Peers URLs</td>
<td>List of peer URLs for this member to advertise to the rest of the cluster</td>
<td>--initial-advertise-peer-urls</td>
</tr>
<tr>
<td>Listen Client URLs</td>
<td>List of URLs to listen on for client traffic</td>
<td>--listen-client-urls</td>
</tr>
</tbody>
</table>

...continues
Parameter Description Option to etcd
---
Listen Peer URLs List of URLs to listen on for peer traffic --listen-peer-urls
Election Timeout Time (in milliseconds) for an election to timeout --election-timeout
Heart Beat Time (in milliseconds) for a heartbeat interval --heartbeat-interval
Interval
Snapshot Count Number of committed transactions that trigger a snapshot to disk --snapshot-count
Debug* Drop the default log level to DEBUG for all subpackages --debug
Options Additional parameters for the etcd daemon

* Boolean (takes yes or no as a value)

Table 8.6: EtcdHost role parameters and etcd options

The etcd settings are updated by Bright Cluster Manager in /cm/local/apps/etcd/current/etc/cm-etcd.conf.
Further details on the etcd store can be found at http://kubernetes.io/docs/admin/etcd/.

KubernetesApiServer Role
The Kubernetes API 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 Kubernetes API objects. The API objects include pods, services, and replicationcontrollers. 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 8.7:

Parameter Description Option to kube-apiserver
---
Kubernetes Cluster The Kubernetes cluster instance (pointer)
Insecure Bind Address IP address to serve on (set to 0.0.0.0 for all interfaces) --insecure-bind-address
Secure Bind Address The IP address on which to serve the read-only and secure ports --bind-address
Etcd Prefix The prefix for all resource paths in etcd --etcd-prefix
Etcd Servers List of etcd servers to watch (format: http://<IP address>:<port number>) --etcd-servers
Service Network Network from which the service cluster IP addresses will be assigned. Must not overlap with any IP address ranges assigned to nodes for pods. --service-cluster-ip-range

...continues

© Bright Computing, Inc.
Parameter | Description | Option to kube-apiserver
--- | --- | ---
Admission Control | Ordered list of plug-ins to control the admission of resources into cluster | --admission-control
Allowed Privileged* | If true, allow privileged containers | --allow-privileged
Event TTL | Time period that events are retained. Default, and a format example: 1h0m0s | --event-ttl
Kubelet HTTPS* | Use HTTPS for kubelet connections | --kublet-https
Kubelet Timeout | Kubelet port timeout. Default value 5s | --kublet-timeout
Log Level | Log level | --v
Log To StdErr* | Logging to stderr means it goes into the systemd journal (default yes) | --logtostderr
Options | Additional parameters for the kube-apiserver daemon | 

* Boolean (takes yes or no as a value)

Table 8.7: KubernetesApiServer role parameters and kube-apiserver options

Further details on the Kubernetes API Server can be found at http://kubernetes.io/docs/admin/kube-apiserver.html.

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 (October 2016) are the replication controller, endpoints controller, namespace controller, and serviceaccounts controller.

The KubernetesController role parameters are described in table 8.8:

Parameter | Description | Option to kube-controller-manager
--- | --- | ---
Kubernetes Cluster | The Kubernetes cluster instance (pointer) | 
Pod Network | Network where pod IP addresses will be assigned from | --cluster-cidr
API Server | IP address of the Kubernetes API server | --master
Address | IP address to serve on (set to 0.0.0.0 for all interfaces) | --address

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent Endpoint Syncs</td>
<td>Number of endpoint syncing operations that will be done concurrently.</td>
<td>--concurrent-endpoint-syncs</td>
</tr>
<tr>
<td>Concurrent Rc Syncs</td>
<td>The number of replication controllers that are allowed to sync concurrently.</td>
<td>--concurrent-rc-syncs</td>
</tr>
<tr>
<td>Namespace Sync Period</td>
<td>Period for syncing namespace life-cycle updates</td>
<td>--namespace-sync-period</td>
</tr>
<tr>
<td>Node Monitor Period</td>
<td>Period the running Node is allowed to be unresponsive before marking it unhealthy</td>
<td>--node-monitor-period</td>
</tr>
<tr>
<td>Node Startup Grace Period</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>Node Sync Period</td>
<td>Period for syncing nodes from cloud-provider</td>
<td>--node-sync-period</td>
</tr>
<tr>
<td>Node Monitor Grace Period</td>
<td>Period for syncing NodeStatus in NodeController</td>
<td>--node-monitor-grace-period</td>
</tr>
<tr>
<td>Pod Eviction Timeout</td>
<td>Grace period for deleting pods on failed nodes</td>
<td>--pod-eviction-timeout</td>
</tr>
<tr>
<td>Pv Claim Binder Sync Period</td>
<td>Period for syncing persistent volumes and persistent volume claims</td>
<td>--pvclaimbinder-sync-period</td>
</tr>
<tr>
<td>Register Retry Count</td>
<td>Number of retries for initial node registration</td>
<td>--register-retry-count</td>
</tr>
<tr>
<td>Resource Quota Sync Period</td>
<td>Period for syncing quota usage status in the system</td>
<td>--resource-quota-sync-period</td>
</tr>
<tr>
<td>Log Level</td>
<td>Log level</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>
</tbody>
</table>

Options Additional parameters for the kube-controller-manager daemon

* Boolean (takes yes or no as a value)

Table 8.8: KubernetesController role parameters and kube-controller-manager options

Further details on the Kubernetes controller manager can be found at http://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 8.9:

<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>API Server</td>
<td>IP Address of the Kubernetes API server</td>
<td>--master</td>
</tr>
<tr>
<td>Address</td>
<td>IP address to serve on (set to 0.0.0.0 for all interfaces)</td>
<td>--address</td>
</tr>
<tr>
<td>Algorithm Provider</td>
<td>The scheduling algorithm provider to use.</td>
<td>--algorithm-provider</td>
</tr>
<tr>
<td>Policy Config</td>
<td>File with scheduler policy configuration</td>
<td>--policy-config-file</td>
</tr>
<tr>
<td>Log Level</td>
<td>Log level</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>
</tbody>
</table>

* Boolean (takes yes or no as a value)

Table 8.9: KubernetesScheduler role parameters and kube-scheduler options

Further details on the Kubernetes scheduler can be found at http://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 backends.

The KubernetesProxy role parameters are described in table 8.10:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Configuration Parameter Passed To kube-proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes Cluster</td>
<td>The Kubernetes cluster instance (pointer)</td>
<td></td>
</tr>
<tr>
<td>API Server</td>
<td>Addresses of the Kubernetes API server</td>
<td>--master</td>
</tr>
</tbody>
</table>

...continues
Further details on the Kubernetes network proxy can be found at [http://kubernetes.io/docs/admin/kube-proxy](http://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 8.11:

<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>--master</td>
</tr>
<tr>
<td>API Server</td>
<td>Addresses of the Kubernetes API server</td>
<td>--master</td>
</tr>
<tr>
<td>Address</td>
<td>IP address to serve on (set to 0.0.0.0 for all interfaces)</td>
<td>--address</td>
</tr>
<tr>
<td>Enable Server*</td>
<td>Enable Kubelet’s server mode</td>
<td>--enable-server</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Option to <code>kubelet</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Network Sources</td>
<td>List of sources from which the Kubelet allows pods to use of host network</td>
<td><code>--host-network-sources</code></td>
</tr>
<tr>
<td>Hostname Override</td>
<td>If non-empty, will use this string as identification instead of the actual hostname</td>
<td><code>--hostname-override</code></td>
</tr>
<tr>
<td>Manifests Path</td>
<td>Path to the config file or directory of files</td>
<td><code>--config</code></td>
</tr>
<tr>
<td>Spool</td>
<td>Directory path for managing kubelet files</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</td>
<td><code>--docker-endpoint</code></td>
</tr>
<tr>
<td>Docker Spool</td>
<td>Absolute path to the Docker state root directory</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</td>
<td><code>--resource-container</code></td>
</tr>
<tr>
<td>Allowed Privileged*</td>
<td>If true, allow privileged containers</td>
<td><code>--allow-privileged</code></td>
</tr>
<tr>
<td>Create Bridge*</td>
<td>If true, kubelet will automatically create and configure bridge cbr0</td>
<td><code>--configure-cbr0</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</td>
<td><code>--register-node</code></td>
</tr>
<tr>
<td>Max Dead Containers</td>
<td>Maximum number of old instances of a container to retain globally</td>
<td><code>--maximum-dead-containers</code></td>
</tr>
<tr>
<td>Max Dead Containers Per Container</td>
<td>Maximum number of old instances of a container to retain per container</td>
<td><code>--maximum-dead-containers-per-container</code></td>
</tr>
<tr>
<td>Max Pods</td>
<td>Number of pods that can run on this node</td>
<td><code>--max-pods</code></td>
</tr>
<tr>
<td>Min Container</td>
<td>Minimum age for a finished container before garbage collection is done on it</td>
<td><code>--minimum-container-\ ttl-duration</code></td>
</tr>
<tr>
<td>TTL Duration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster DNS</td>
<td>IP address for a cluster DNS server</td>
<td><code>--cluster-dns</code></td>
</tr>
<tr>
<td>Cluster Domain</td>
<td>Domain for this cluster</td>
<td><code>--cluster-domain</code></td>
</tr>
</tbody>
</table>

...continues
Table 8.11: KubernetesNode role parameters and kubelet options

Further details on the kubelet daemon can be found at http://kubernetes.io/docs/admin/kubelet/.

8.2.5 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 CA model is the model that Bright Cluster

The Kubernetes CA should not be confused with the Bright Cluster Manager CA. While it is conceivable to use the Bright Cluster Manager CA for this purpose, it makes more sense to use a separate CA for Kubernetes. This is because the Kubernetes public certificates are meant for regular users, unlike Bright Cluster Manager public certificates.
Manager configures by default for Kubernetes, so that certificate use is forced for any communication within Kubernetes. During Kubernetes setup, and also when the Bright Cluster Manager license is updated, a Kubernetes CA is generated and signed by the CMDaemon master node certificate. The certificates are put into `/cm/local/apps/kubernetes/var/etc/`, and `/etc/kubernetes/` is made a link to this directory.

In Kubernetes terminology a user is a unique identity accessing the Kubernetes API server, which may be a human or an automated process. For example an administrator or a pod 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. Bright Cluster Manager configures client certificate usage by default. The authentication is performed by the API server by validating the user certificate, where the common name of the subject is used as the user name for the request.

In Kubernetes, authorization happens as a separate step from authentication. Authorization applies to all HTTP accesses on the main (secure) API server port. The authorization check for any request compares attributes of the context of the request (such as user, resource, and namespace) with access policies. Any API call must be allowed by a policy in order to proceed. By default the policy file is `/cm/local/apps/kubernetes/var/etc/auth-policy.jsonl`, which represents policies in a special format (JSONL, json lines). The file is updated by CMDaemon automatically when user policies are changed.

The user policies definitions are managed in a Kubernetes cluster using `cmsh` or `cmgui`:

**Example**

```
[root@bright73 ~]# cmsh
[root@bright73 ~]# cmsh
[bright73]% kubernetes use default
[bright73->kubernetes[default]]% userpolicies
[bright73->kubernetes[default]->userpolicies]% list
  Name (key) Resources Namespaces
  all    readonly
[bright73->kubernetes[default]->userpolicies]% show readonly
  Parameter  Value
  Is Read Only   yes
  Name           readonly
  Namespaces     readonly
  Resources
```

By default only 2 user policies are defined:

- `all`: allows a user to have read/write access for all resources
- `readonly`: allows a user only read access for all resources

The configuration parameter that is put in the policy file (policy file setting value), is associated with the parameter (key) described in table 8.12:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Policy file setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>...continues</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Bright Computing, Inc.
...continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Policy file setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is Read Only</td>
<td>When true, means that the policy only applies to</td>
<td>readonly</td>
</tr>
<tr>
<td>GET operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namespaces</td>
<td>Namespaces the policy is applied to</td>
<td>namespace</td>
</tr>
<tr>
<td>Resources</td>
<td>Resources from a URL, such as pods, that the policy</td>
<td>resource</td>
</tr>
<tr>
<td></td>
<td>is applied to</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.12: userpolicy mode and policy file setting options

The policies are applied to particular users when Authentication Mode is set to ABAC (Attribute-Based Access Control) for a Kubernetes cluster entity. The Authentication mode can be set using cmsh or cmgui. The policy can be applied to a user via cmsh via the user mode, and adding one or several policies to the Kubernetes Policies string list.

Example

[root@bright73 ~]# cmsh
[bright73]% user
[bright73->user]% add newuser
[bright73->user*[newuser*]]% set kubernetespolicies all
[bright73->user*[newuser*]]% commit
[bright73->user*[newuser*]]%

When the policy configuration file is updated by CMDaemon, kube-apiserver is restarted automatically in order to apply the new user policies. By default, the policy file allows the user kube to have read/write access, because this user is set as a common name in the Kubernetes daemon certificates, and allows the Kubernetes daemons to authenticate against the API server.

If the administrator applies the policies to a new user, then CMDaemon generates a new user certificate for that user and puts the certificate into the $HOME/.cm/ directory. At the same time the $HOME/.kubeconfig file is generated based on the KubeConfigTemplate parameter in the Kubernetes cluster object. If the $HOME/.kubeconfig file already exists then CMDaemon does not change it.

8.2.6 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. Bright Cluster Manager 7.3 by default sets up Flannel, a network manager, for this purpose.

Flannel is a virtual network manager that assigns a subnet to each host for use with the container runtime. Bright Cluster Manager allows the IP address range of the pod networks to be configured. This IP address range is split by Flannel into subnetworks. Each subnetwork is assigned to one, and only one, host in the cluster at a time. Within the host subnetwork, one IP address for each pod within the host. All IP addresses that Flannel manages are reachable across the cluster.

Flannel comes with the cm-flannel package, available from the Bright Computing repository. The package is automatically installed by cm-kubernetes-setup on the head node and software images. The flanneld daemon should run on all nodes of the Kubernetes cluster. When the KubernetesProxy role is assigned to a node or to its category, then Bright Cluster Manager configures and starts a flanneld instance. Its configuration is kept in the etcd key-value store, and is updated by CMDaemon. The administrator can get a Flannel JSON configuration from etcd by executing the following command:

© Bright Computing, Inc.
Example

[root@bright73 ~]# module load etcd
[root@bright73 ~]# etcdctl -endpoint=http://master:5001 get /coreos.com/network/config
"Network": "172.29.0.0/16", "SubnetLen": 24, "Backend": "Type":"udp"

In Bright Cluster Manager 7.3 the Flannel resource name set in etcd is /coreos.com/network/config by default. The resource name can be changed using the advanced configuration flag EtcdFlanneldResource (page 610). Flannel parameters updated in etcd are defined by CMDaemon as follows:

- **Network**: IPv4 network in CIDR format used by the entire Flannel network. The CIDR is defined by a network, specified in KubernetesController as the podnetwork parameter. By default cm-kubernetes-setup creates the network with CIDR 172.29.0.0/16.

- **SubnetLen**: This number sets the size of the subnetwork allocated to each host. By default it is 24, but can be changed with the advanced configuration flag FlannelSubnetLen (page 610).

- **Backend**: Type of backend to use, and configurations for that backend. By default Bright Cluster Manager 7.3 uses a UDP backend. The default port that is used for the backend is 7890. The entire backend section in the Flannel configuration can be changed using the advanced configuration parameter FlannelBackend.

  For example, `FlannelBackend={"Type":"udp", "Port":7900}` specifies a UDP backend that uses port 7900.

With the default setup Flannel can create up to 256 subnetworks, and allocate 254 IP addresses in each of the subnetworks. This means that by default the cluster can run up to 65024 pods, each with a unique IP address. If the number of allocated IP addresses need to be modified, then the administrator can change the pod network mask using cmsh or cmgui, and can also change the SubnetLen value via the advanced configuration flag.

By default, pods get their IP addresses from Docker. When Flannel is used, the Docker daemon gets its parameters from the file /run/flannel/docker created by flanneld. The parameters specify how Docker should configure network interfaces for pods. One of the parameters is the IPv4 subnetwork, in CIDR format, that is allocated by Flannel for a particular host. Flannel guarantees that IP addresses allocated from that subnetwork will never conflict with the IP address allocated from the subnetworks defined for Docker running on other hosts.

The path that a TCP/IP packet takes between two pods running on different hosts is illustrated by figure 8.11.

![Figure 8.11: Path of TCP/IP packet passed from a pod running on one host to a pod running on another host](image)

An explanation of this figure follows:
• The process running inside a pod connects to a second pod running on different host. The packet goes from the process to the local network interface visible from inside the pod (usually eth0).

• The packet goes from the eth0 interface of the first pod to the bridge created by Docker, because eth0 has been added to it here. The bridge is usually called docker0.

• The packet is transferred from the docker0 bridge to the flannel0 interface.

• The packet gets transferred from the flannel0 interface to the flanneld daemon. The virtual network device /dev/net/tun is used by flanneld for this.

• flanneld sends the packet through the network using the IP address of the remote, second, pod. If the UDP backend of Flannel is used, then the packets are transferred between flanneld daemons using the UDP protocol.

• The packet is received by flanneld running on the remote host, and goes to the second pod in the reverse order that it followed when leaving the first pod. That is, it goes through the second host via the flannel0 network interface and the docker0 bridge network interface before reaching the second pod.

### 8.2.7 Kubernetes Monitoring

When cm-kubernetes-setup runs it configures the following Kubernetes-related healthchecks:

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

### 8.2.8 Files Managed By CMDaemon

All the Kubernetes files that are managed by CMDaemon are located under /cm/local/apps/kubernetes/var/etc/, and are described in table 8.13:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiserver</td>
<td>kube-apiserver parameters</td>
</tr>
<tr>
<td>auth-policy.jsonl</td>
<td>Authentication policy file</td>
</tr>
<tr>
<td>controller-manager</td>
<td>kube-controller-manager parameters</td>
</tr>
<tr>
<td>kubeca.key</td>
<td>Kubernetes CA key</td>
</tr>
</tbody>
</table>

...continues
Table 8.13: Kubernetes files managed by CMDaemon

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubeca.pem</td>
<td>Kubernetes CA certificate</td>
</tr>
<tr>
<td>kube.key</td>
<td>Kubernetes infrastructure user key</td>
</tr>
<tr>
<td>kube.pem</td>
<td>Kubernetes infrastructure user certificate</td>
</tr>
<tr>
<td>kubeconfig</td>
<td>Configuration file shared among Kubernetes daemons of a single Kubernetes cluster</td>
</tr>
<tr>
<td>kubelet</td>
<td>Kubelet parameters</td>
</tr>
<tr>
<td>proxy</td>
<td>kube-proxy parameters</td>
</tr>
<tr>
<td>scheduler</td>
<td>kube-scheduler parameters</td>
</tr>
<tr>
<td>scheduler-policy.json</td>
<td>kube-scheduler policies</td>
</tr>
</tbody>
</table>

8.3 Singularity

Bright 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.

8.3.1 Use Cases

Adding Singularity to Bright 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 `singularity` command can also be used to apply special options to the container, when executing the image file in the jobscript or shell.

8.3.2 Package cm-singularity

Singularity is packaged for SLES12 and RHEL7 and derivatives. It is available from the YUM or Zypper repositories from version 7.3 of Bright Cluster Manager onwards, and is distributed as a package called cm-singularity. The package should be installed in the software image for each node. The user
8.3 Singularity

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.

8.3.3 MPI Integration

Because of the nature of Singularity, all MPI implementations should work fine inside a Singularity container. Nevertheless, 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. In some cases, starting an MPI process may not be as optimal as execution outside the container. Thus, 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 start up 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.

© Bright Computing, Inc.
Post-Installation Software Management

Some time after 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 Bright 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 Red Hat enterprise distributions require the purchase of their license in order to access their repositories.

Packages managed by the Bright 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).

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 the filesystem that a node picks up from a provisioner (a head node or a provisioning node) during 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 9.1 describes the naming convention for a Bright Cluster Manager RPM package.

Section 9.2 describes how an RPM package is managed for the head node.

Section 9.3 describes how a kernel RPM package can be managed on a head node or image.

Section 9.4 describes how an RPM package can be managed on the software image.

Section 9.5 describes how a software other than an RPM package can be managed on a software image.

Section 9.6 describes how custom software images are created that are completely independent of the existing software image distribution and version.

9.1 Bright Cluster Manager RPM Packages And Their Naming Convention

Like the distributions it runs on top of, Bright Cluster Manager uses RPM (RPM Package Manager) packages. An example of a Bright Cluster Manager RPM package is:

```
mpich-ge-gcc-64-1.2.7-116_cm7.3.x86_64.rpm
```

The file name has the following structure:

```
package-version-revision_cm.x.y.architecture.rpm
```

where:
• *package* (mpich-ge-gcc-64) is the name of the package

• *version* (1.2.7) is the version number of the package

• *revision* (116) is the revision number of the package

• *x.y* (7.3) is the version of Bright Cluster Manager for which the RPM was built

• *architecture* (x86_64) is the architecture for which the RPM was built

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 identified using “rpm -qf”.

**Example**

```
[root@bright73 bin]# rpm -qf /usr/bin/zless
gzip-1.3.12-18.el6.x86_64
[root@bright73 bin]# rpm -qf /cm/local/apps/cmd/sbin/cmd
cmdaemon-7.3-13383_cm7.3.x86_64
```

In the example, /usr/bin/zless is supplied by the distribution, while /cm/local/apps/cmd/sbin/cmd is supplied by Bright Computing, as indicated by the “_cm” in the nomenclature.

As an aside, it should be noted that using a distribution package means less work for Bright Computing developers. System administrators should therefore be aware that the Bright Computing version of a package is provided and used for a reason. Replacing the Bright Computing version with a distribution 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.

More information about the RPM Package Manager is available at [http://www.rpm.org](http://www.rpm.org).

### 9.2 Managing Packages On The Head Node

#### 9.2.1 Managing RPM Packages On The Head Node

Once Bright Cluster Manager has been installed, distribution packages and Bright Cluster Manager software packages can be managed using the `rpm` command-line utility.

A more convenient way of managing RPM packages is to use either YUM or zypper. Both of these tools are repository and package managers. The zypper tool is recommended for use by the SUSE 11 distribution, while YUM is recommended for use by the other distributions that Bright Cluster Manager supports. YUM is not installed by default in SUSE 11, and it is better not to install and use it with SUSE 11 unless the administrator is familiar with configuring YUM.

For YUM and zypper, the following commands list all available packages:

```
yum list
```

or

```
zypper packages
```

For zypper, the short command option *pa* can also be used instead of *packages*.

The following commands can install a new package called `<package name>`:

```
yum install <package name>
```

or

```
zypper in <package name>
```

All installed packages can be updated with:

```
yum update
```
or
zypper refresh; zypper up   #refresh mostly not needed

Bright Computing maintains YUM and zypper repositories at:

http://updates.brightcomputing.com/yum

and updates are fetched by YUM and zypper for Bright Cluster Manager packages from there by default, to overwrite older package versions by default.

Accessing the YUM repositories manually (i.e. not through YUM or zypper) requires a username and password. Authentication credentials are provided upon request. For more information on this, support@brightcomputing.com should be contacted.

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 is done with:

yum clean all
or
zypper clean -a

As an extra protection to prevent Bright Cluster Manager installations from receiving malicious updates, all Bright 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, Scientific Linux, and CentOS packages. 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 RPM 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

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 installation purposes. These are installed mostly on the head node, but some packages work as a server and client process, with the clients installed and running on the regular nodes. The packages are described in Chapter 7 of the Installation Manual:

• Modules (section 7.1)
• Shorewall (section 7.2)
• Compilers (section 7.3):
  – GCC (section 7.3.1)
  – Intel Compiler Suite (section 7.3.2)
  – PGI High-Performance Compilers (section 7.3.3)
  – AMD Open64 Compiler Suite (section 7.3.4)
  – FLEXlm License Daemon (section 7.3.5)
• Intel Cluster Checker (section 7.4)

© Bright Computing, Inc.
• CUDA (section 7.5)
• Lustre (section 7.8)
• ScaleMP (section 7.10)

Exclusion of packages on the head node can be carried out as explained in section 9.3.2, where the kernel package is used as an example for exclusion.

Gnome And KDM Installation—Disabling network-manager

NetworkManager interferes with the custom network management of Bright Cluster Manager and should never be enabled.

When installing the cluster from bare metal using the Bright Cluster Manager installer, if the X11 graphical user installation option checkbox is ticked, then NetworkManager is disabled by using a finalize script, so that Bright 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, for example with:

\[ \text{yum groupinstall "GNOME Desktop"} \]

The Network Manager package is a dependency of the Gnome and KDE desktops, and is therefore automatically installed by the \texttt{yum} command. As a result, the NetworkManager service is enabled, and the distribution defaults are to have it run. However, because this interferes with the custom network management of Bright Cluster Manager, the NetworkManager should be disabled.

Disabling Network Manager can be done as follows:

• On Red Hat 6 and derivatives:

  Example

  [root@bright73 ~]# service NetworkManager status
  NetworkManager (pid 1527) is running...
  [root@bright73 ~]# chkconfig NetworkManager off
  [root@bright73 ~]# service NetworkManager stop

• On Red Hat 7 and derivatives:

  Example

  [root@bright73 ~]# systemctl status NetworkManager.service
  [root@bright73 ~]# systemctl disable NetworkManager.service

• On SLES-based systems:

  Example

  YAST can be used to disable NetworkManager by going to Network Devices → Network Settings → Global Options → Network Setup Method, and setting the option: Traditional Method with ifup.

9.2.2 Managing Non-RPM Software On The Head Node

Sometimes a package is not packaged as an RPM package by Bright Computing or by the distribution. In that case, the software can usually be treated as for installation onto a standard distribution. There may be special considerations on placement of components that the administrator may feel appropriate due to the particulars of a cluster configuration.
For example, for compilation and installation of the software, some consideration may be made of the options available on where to install parts of the software within the default shared filesystem. A software may have a compile option, say `--prefix`, that places an application `<application>` in a directory specified by the administrator. If the administrator decides that `<application>` should be placed in the shared directory, so that everyone can access it, the option could then be specified as: “`--prefix=/cm/shared/apps/<application>`”.

Other commonly provided components of software for the applications that are placed in shared may be documentation, licenses, and examples. These may similarly be placed in the directories `/cm/shared/docs`, `/cm/shared/licenses`, and `/cm/shared/examples`. The placement may be done with a compiler option, or, if that is not done or not possible, it could be done by modifying the placement by hand later. It is not obligatory to do the change of placement, but it helps with cluster administration to stay consistent as packages are added.

Module files (section 2.2 of this manual, and 7.1 of the *Installation Manual*) may sometimes be provided by the software, or created by the administrator to make the application work for users easily with the right components. The directory `/cm/shared/modulefiles` is recommended for module files to do with such software.

To summarize the above considerations on where to place software components, the directories under `/cm/shared` that can be used for these components are:

```
/cm/shared/
|-- apps
|-- docs
|-- examples
|-- licenses
|-- modulefiles
```

### 9.3 Kernel Management On A Head Node Or Image

Care should be taken when updating a head node or software image. This is particularly true when custom kernel modules compiled against a particular kernel version are being used.

#### 9.3.1 Installing A Standard Distribution Kernel

A standard distribution kernel is treated almost like any other package in a distribution. This means that for head nodes, installing a standard kernel is done according to the normal procedures of managing a package on a head node (section 9.2), while for regular nodes, installing a standard distribution kernel onto a regular node is done according to the normal procedures of managing an RPM package inside an image (section 9.4).

An example standard kernel package name is “`kernel-2.6.18-274.3.1.el5`”. The actual one suited to a cluster varies according to the distribution used. 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.

When upgrading a kernel, an extra consideration for a head or software image is that third-party drivers may need to be re-installed or rebuilt for that kernel. This is necessary in the case of OFED drivers used instead of the standard distribution drivers. Details on re-installing or rebuilding such OFED drivers are given in section 7.6 of the *Installation Manual*.

When installing a kernel, extra considerations for software images are:

- The kernel must also be explicitly set in CMDaemon (section 9.3.3) before it may be used by the regular nodes.

- Some GRUB-related errors with a text such as “`grubby fatal error: unable to find a suitable template`” typically show up during the installation of a kernel package in the software image. These occur due to a failure to find the partitions when running the post-install
scripts in the chroot environment. They can simply be ignored because the nodes do not boot from partitions configured by GRUB.

- **Warnings such as:** warning: Failed to read auxiliary vector, /proc not mounted? may come up, due to /proc not being in use in the software image. These can also simply be ignored.

- The ramdisk must be regenerated using the `createramdisk` command (section 9.4.3).

As is standard for Linux, both head or regular nodes must be rebooted to use the new kernel.

### 9.3.2 Excluding Kernels And Other Packages From Updates

#### Specifying A Kernel Or Other Package For Update Exclusion

Sometimes it may be desirable to exclude the kernel from updates on the head node.

- When using `yum`, to prevent an automatic update of a package, the package is listed after using the `--exclude` flag. So, to exclude the kernel from the list of packages that should be updated, the following command can be used:

```bash
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:

```ini
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, then the `update` command is run, and finally the kernel package is unlocked again using the `removelock` command:

```bash
zypper addlock kernel
zypper update
zypper removelock kernel
```

#### 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, a construction like 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`:

```bash
[root@bright73 ~]# yum repolist
... repo id repo name status base CentOS-6 - Base 6,356+11 cm-rhel6-7.3 Cluster Manager 7.3 316+13 cm-rhel6-7.3-updates Cluster Manager 7.3 514 extras CentOS-6 - Extras 14 updates CentOS-6 - Updates 391 repolist: 7,591
[root@bright73 ~]# yum --disablerepo=* --enablerepo=cm* update
```
9.3 Kernel Management On A Head Node Or Image

Blocking Major OS Updates With The cm-dist-limit-<DIST> Package

Sometimes Bright Cluster Manager may install the package:

    cm-dist-limit-<DIST>

where <DIST> is rhel6.3, sl6.3, centos6.3, and higher numbered versions.

When the package is installed, it deliberately prevents yum update from working if the update is going to install packages that require a higher version of cm-dist-limit-<DIST>. That is, it is a locking mechanism.

For example, when trying to upgrade from SL6.3 to SL6.4, and if cm-dist-limit-sl6.3 is installed, then the only way to update the operating system to SL6.4, is to remove the cm-dist-limit-sl6.3 package. Any dependent packages will also be removed automatically by YUM. A subsequent yum update installs new versions of the removed packages, and the cm-dist-limit-sl6.4 package is installed automatically.

The reason for the cm-dist-limit-<DIST> packages is to prevent the OS from upgrading. This is helpful if some packages are not compatible with upgrades.

Updates blocked Bright Cluster Manager with cm-dist-limit-<DIST> package include intel-mic-* package updates. The intel-mic-* packages, used by MICs, are not compatible with a yum update from an OS based on RHEL6.3 that updates the OS to RHEL6.4.

9.3.3 Updating A Kernel In A Software Image

A kernel is typically updated in the software image by carrying out a package installation using the chroot environment (section 9.4).

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 derived from RHEL6 and above are by default configured, by the distribution itself, so that only up to 3 kernel images are kept during YUM updates. The distribution default value is however overridden by a Bright Cluster Manager default value, so that kernel images are never removed during YUM updates, 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 cmgui or cmsh.

- To enable it in cmgui, the Software Images resource is selected, and the specific image item is selected. The Settings pane for that particular software image is opened, the new kernel version is selected from the "Kernel version" drop-down menu, and the Save button is clicked. Saving the version builds a new initial ramdisk containing the selected kernel (figure 9.1).

- 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@bright73 ~]# cmsh
[bright73]# softwareimage
[bright73]->softwareimage% use default-image
[bright73->softwareimage[default-image]]# set kernelversion 2.6.32-131.2.1.el6.x86_64
[bright73->softwareimage*[default-image*]]# commit -w
```

Tab-completion suggestions for the set kernelversion command will display the available values for the kernel version.

© Bright Computing, Inc.
9.3.4 Setting Kernel Options For Software Images

A standard kernel can be booted with special options that alter its functionality. For example, a kernel can boot with `apm=off`, to disable Advanced Power Management, which is sometimes useful as a workaround for nodes with a buggy BIOS that may crash occasionally when it remains enabled.

To enable booting with this kernel option setting in cmgui, the “Software Images” resource is selected, and the specific image item is selected (figure 9.1). The Settings tabbed pane for that particular software image is opened, and the “Kernel parameters” value is set to `apm=off`.

In cmsh the value of “kernel parameters” in softwareimage mode for the selected image can be modified as follows:

```
[root@bright73 ~/]# cmsh
[bright73]# softwareimage
[bright73]# use default-image
[bright73]# append kernelparameters " apm=off"
[bright73]# commit
```

Often kernel options load up modules and their parameters. Making module loading persist after reboot and setting module loading order is covered in section 5.3.2

Some kernel options may require changes to be made in the BIOS settings in order to function.

9.3.5 Kernel Driver Modules

Bright Computing provides some packages which install new kernel drivers or update kernel drivers. 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**

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.

To check the head node and software images for the installation status of the `kernel-devel` package, the Bright Cluster Manager utility `kernel-devel-check` is run from the head node:

**Example**
9.3 Kernel Management On A Head Node Or Image

[root@mycluster ~]# kernel-devel-check
Head node: mycluster
   Found kernel development rpm package kernel-devel-2.6.32-131.2.1.el6.x86_64

Software image: default-image
   No kernel development directories found, probably no kernel development package installed.
   Kernel development rpm package kernel-devel-2.6.32-131.2.1.el6.x86_64 not found
   If needed, try to install the kernel development package with:
   # chroot /cm/images/default-image yum install kernel-devel-2.6.32-131.2.1.el6.x86_64

Software image: default-image1
   No kernel development directories found, probably no kernel development package installed.
   Kernel development rpm package kernel-devel-2.6.32-131.2.1.el6.x86_64 not found
   If needed, try to install the kernel development package with:
   # chroot /cm/images/default-image1 yum install kernel-devel-2.6.32-131.2.1.el6.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-2.6.32-131.2.1.el6.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 9.5.2), 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.

© Bright Computing, Inc.
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 9.3.5).

If the regular nodes have the kernel-devel package installed, then the following yum commands are issued on the head node, to install the package on the head node and in the default-image:

**Example**

```
[root@mycluster ~]# yum install intel-wired-ethernet-drivers
[root@mycluster ~]# chroot /cm/images/default-image
[root@mycluster /]# yum install intel-wired-ethernet-drivers
```

For SUSE, the equivalent zypper commands are used (“zypper in” instead of “yum install”).

**Kernel Driver Modules: CUDA Driver Installation**

CUDA drivers are drivers the kernel uses to manage GPUs. These are compiled on the fly for nodes with GPUs in Bright Cluster Manager. The details of how this is done is covered in the CUDA software section (section 7.5 of the Installation Manual).

**Kernel Driver Modules: OFED And OPA Stack Installation**

By default, the distribution provides the OFED stack used by the kernel to manage the InfiniBand or RDMA interconnect. Installing a Bright Cluster Manager repository OFED stack to replace the distribution version is covered in section 7.6 of the Installation Manual. Some guidance on placement into initrd for the purpose of optional InfiniBand-based node provisioning is given in section 5.3.3.

Installing a Bright Cluster Manager repository Omni-Path (OPA) stack is covered in section 7.7 of the Installation Manual.

### 9.4 Managing An RPM Package In A Software Image And Running It On Nodes

#### 9.4.1 Installing From Head Via chroot: Installing Into The Image

Managing RPM packages (including the kernel) inside a software image is most easily done while on the head node, using a chroot mechanism with rpm, yum, or zypper.

The `rpm` command supports the `--root` flag. To install an RPM package inside the default software image while on the head node, the command is used as follows:

**Example**

```
rpm --root /cm/images/default-image -ivh /tmp/libxml2-2.6.16-6.x86_64.rpm
```

YUM and zypper implement the same functionality in slightly different ways. For example, all packages in the default image are updated with:

```
yum --installroot=/cm/images/default-image update
```

or
9.4 Managing An RPM Package In A Software Image And Running It On Nodes

zypper --root /cm/images/default-image up

With the chroot command, the same result is accomplished by first chrooting into an image, and subsequently executing rpm, yum, or zypper commands without --root or --installroot arguments.

Excluding Packages And Repositories From The Image
Sometimes it may be desirable to exclude a package 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:

```
xclude = <package 1> <package 2> ...
```

- The zypper command can also carry out the task of excluding a package from getting updated when during update. To do this, the package is first locked (prevented from change) using the addlock command, then the update command is run, and finally the package is unlocked again using the removelock command. For example, for the kernel package:

```
zypper --root /cm/images/default-image addlock kernel
zypper --root /cm/images/default-image update
zypper --root /cm/images/default-image removelock kernel
```

- 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 Bright Cluster Manager updates to be pulled into the image. In that case, a construction like the following may be used to specify that, for example, from the repositories listed in /cm/images/default-image/etc/yum.repos.d/, only the repositories matching the pattern cm* are used:

```
[root@bright73 ~]# cd /cm/images/default-image/etc/yum.repos.d/
[root@bright73 yum.repos.d]# yum --installroot=/cm/images/default-image --disablerepo=* --enablerepo=cm* update
```

9.4.2 Installing From Head Via chroot: Updating The Node
Rebooting the nodes that use the software images then has 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.

9.4.3 Installing From Head Via rpm --root, yum --installroot Or chroot: Possible Issues

- The update process with YUM or zypper on an image 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 setting the image to a locked state (section 5.4.7), before running the update process. The image can then be updated. The administrator normally unlocks the image after the update, to allow image maintenance by the provisioners again.
Example

[root@bright73 ~]# cmsh -c "softwareimage use default-image; set locked yes; commit"
[root@bright73 ~]# yum --installroot /cm/images/default-image update
[root@bright73 ~]# cmsh -c "softwareimage use default-image; set locked no; commit"

- 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 5 head node to a Red Hat 6 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. 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 packages may cause similar problems.

Bright Computing’s RPM packages are designed to install under chroot without issues. However packages from other repositories may cause the issues described. To deal with that, the cluster manager runs the `chrootprocess` health check, which alerts the administrator if there is a daemon process running in the image. The `chrootprocess` also checks and kills the process if it is a `crond` process.

- For some package updates, the distribution package management system attempts to modify the ramdisk image. This is true for kernel updates, many kernel module updates, and some other packages. Such a modification is designed to work on a normal machine. For a regular node on a cluster, which uses an extended ramdisk, the attempt does nothing.

In such cases, a new ramdisk image must nonetheless be generated for the regular nodes, or the nodes will fail during the ramdisk loading stage during start-up (section 5.8.4).

The ramdisk image for the regular nodes can be regenerated manually, using the `createramdisk` command (section 5.3.2).

- Trying to work out what is in the image from under chroot must be done with some care.

For example, under chroot, running "`uname -a`" returns the kernel that is currently running—that is the kernel outside the chroot. This is typically not the same as the kernel that will load on the node from the filesystem under chroot. It is the kernel in the filesystem under chroot that an unwary administrator may wrongly expect to detect on running the `uname` command under chroot.

To find the kernel version that is to load from the image, the software image kernel version property (section 9.3.3) can be inspected using the cluster manager with:

Example

```
  cmsh -c "softwareimage; use default-image; get kernelversion"
```

### 9.5 Managing Non-RPM Software In A Software Image And Running It On Nodes

Sometimes, packaged software is not available for a software image, but non-packaged software is. This section describes the installation of non-packaged software onto a software image in these two cases:

1. copying only the software over to the software image (section 9.5.1)
2. placing the software onto the node directly, configuring it until it is working as required, and syncing that back to the software image using Bright Cluster Manager’s special utilities (section 9.5.2)

As a somewhat related aside, completely overhauling the software image, including changing the base files that distinguish the distribution and version of the image is also possible. How to manage that kind of extreme change is covered separately in section 9.6.

However, this current section (9.5) is about modifying the software image with non-RPM software while staying within the framework of an existing distribution and version.

In all cases of installing software to a software image, it is recommended that software components be placed under appropriate directories under /cm/shared (which is actually outside the software image).

So, just as in the case for installing software to the head node in section 9.2.2, appropriate software components go under:

/cm/shared/
|-- apps
|-- docs
|-- examples
|-- licenses
|-- modulefiles

### 9.5.1 Managing The Software Directly On An Image

The administrator may choose to manage the non-packaged software directly in the correct location on the image.

For example, the administrator may wish to install a particular software to all nodes. If the software has already been prepared elsewhere and is known to work on the nodes without problems, such as for example library dependency or path problems, then the required files can simply be copied directly into the right places on the software image.

The `chroot` command may also be used to install non-packaged software into a software image. This is analogous to the `chroot` technique for installing packages in section 9.4:

**Example**

```
cd /cm/images/default-image/usr/src
| tar -xvzf /tmp/app-4.5.6.tar.gz
| chroot /cm/images/default-image
| cd /usr/src/app-4.5.6
| ./configure --prefix=/usr
| make install
```

Whatever method is used to install the software, after it is placed in the software image, the change can be implemented on all running nodes by running the `updateprovisioners` (section 5.2.4) and `imageupdate` (section 5.6.2) commands.

### 9.5.2 Managing The Software Directly On A Node, Then Syncing Node-To-Image

#### Why Sync Node-To-Image?

Sometimes, typically if the software to be managed is more complex and needs more care and testing than might be the case in section 9.5.1, the administrator manages it directly on a node itself, and then makes an updated image from the node after it is configured, to the provisioner.

For example, the administrator may wish to install and test an application from a node first before placing it in the image. Many files may be altered during installation in order to make the node work with the application. Eventually, when the node is in a satisfactory state, and possibly after removing any temporary installation-related files on the node, a new image can be created, or an existing image updated.
Administrators should be aware that until the new image is saved, the node loses its alterations and reverts back to the old image on reboot.

The node-to-image sync can be seen as the converse of the image-to-node sync that is done using imageupdate (section 5.6.2).

The node-to-image sync discussed in this section is done using the “Grab to image” or “Synchronize image” button in cmgui, or using the “grabimage” command with appropriate options in cmsh. The sync automatically excludes network mounts and parallel filesystems such as Lustre and GPFS, but includes any regular disk mounted on the node itself.

Some words of advice and a warning are in order here:

- The cleanest, and recommended way, to change an image is to change it directly in the node image, typically via changes within a chroot environment (section 9.5.1).

- Changing the deployed image running on the node can lead to unwanted changes that are not obvious. While many unwanted changes are excluded because of the excludelistgrab* lists during a node-to-image sync, there is a chance that some unwanted changes do get captured. These changes can lead to unwanted or even buggy behavior. The changes from the original deployed image should therefore be scrutinized with care before using the new image.

- For scrutiny, the bash command:

\[
\text{vimdiff }<\text{(cd image1; find . | sort)} <\text{(cd image2; find . | sort)}
\]

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 cmgui

In cmgui, saving the node state to a new image is done by selecting a node from the Nodes resource, and then selecting the Tasks tab. The “Software image” section of the tab has these two buttons that can carry out the sync from node to software image (figure 9.2):

1. The “Grab to image” button. This opens up a dialog offering a new image to sync to. This button creates a new image, wiping 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 the “Node Categories” resource, under the Settings tab. The exclude list is known as excludelistgrabnew (page 364) in cmsh.

2. The “Synchronize image” button. 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 the “Node Categories” resource, under the Settings tab. This exclude list is known as excludelistgrab (page 364) in cmsh.

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.

Both node-to-image sync options ask for reconfirmation of the action before going ahead with it. The “Grab to image” option in addition also allows selection of the image to be used for syncing. Also,
in the dialog for both options, a dry-run checkbox is marked by default to allow the administrator to see what would happen during the node-to-image sync, without actually having the files written over to the image.

Logs of the run are viewable by clicking on the “Provisioning Log” button, also in the Tasks tab. If the administrator is sure that the run works as wanted, then the node-to-image sync can be done with the dry-run checkbox unmarked.

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 is created into which the node state can be written. This is easiest to do by using the “Software Images” resource to clone a new image. The node state with the software installed on it would then be saved using the “Grab to image” button, and choosing the cloned image name as the image to save it to.

2. A new category is then cloned from the old category, and within the new category the old image is changed to the new image. This is easiest to do from the Overview tab of the “Node Categories” resource. Here the original category of the node is selected and cloned. The cloned category is then selected, opened and the old image within it—in the “Software image” section—is changed to the new, cloned, image, and the result is saved.

3. Some nodes are set to the new category to test the new image. This is done from the Nodes resource, selecting the node, and from its Settings tab, changing its category to the new one, and saving the change.

4. The nodes in the new category are made to pick up and run their new images. This can be done with a reboot.

5. After sufficient testing, all remaining nodes can be moved to using the new image, and the old image is removed if no longer needed.

**Node-To-Image Sync Using cmsh**

The preceding cmgui 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@bright73 ~]# cmsh
[bright73]# softwareimage
[bright73->softwareimage]% clone default-image default-image1
[bright73->softwareimage[default-image1]]% commit
[bright73->softwareimage[default-image1]]% category
[bright73->category]% clone default default1
[bright73->category[default1]]% commit
[bright73->category[default1]]% set softwareimage default-image1
[bright73->category[default1]]% commit
[bright73->category[default1]]% device
[bright73->device]% grabimage -w -i default-image1 node001
[bright73->device]%
Mon Jul 18 16:13:00 2011 [notice] bright73: Provisioning started on node node001
[bright73->device]%
Mon Jul 18 16:13:04 2011 [notice] bright73: Provisioning completed on node node001
```

The `grabimage` command without the `-w` option simply does a dry-run so that the user can see in the provisioning logs what should be grabbed, without having the changes actually carried out. Running `grabimage -w` instructs CMDaemon to really write the image.

When writing out the image, two exclude lists may be used:

- 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 362) in cmgui.

- The `excludelistgrab` object. This is used with the `grabimage` command, run without the `-i` option. The list can be accessed and edited under cmsh, in its category mode for a node image, as the `excludelistgrab` object. It corresponds to the “Exclude list image grab” exclusion list associated with the Synchronize image option (page 362) in cmgui.

### 9.6 Creating A Custom Software Image

By default, the software image used to boot non-head nodes is based on the same version and release of the Linux distribution as used by the head node. However, sometimes an image based on a different distribution or a different release from that on the head node may be needed.

A custom software image is created typically by building an entire filesystem image from a regular node. The node, which is never a head node, is then called the base host, with the term “base” used to indicate that it has no additional cluster manager packages installed. The distribution on the base host, is called the base distribution and is a selection of packages derived from the parent distribution (Red Hat, Scientific Linux etc). A base distribution package is a package or rpm that is directly provided by the vendor of the parent distribution which the base distribution is based on, and is not provided by Bright Cluster Manager.

Creating a custom software image consists of two steps. The first step (section 9.6.1) is to create a base (distribution) archive from an installed base host. The second step (section 9.6.2) is to create the image from the base archive using a special utility, `cm-create-image`.

#### 9.6.1 Creating A Base Distribution Archive From A Base Host

**Structure Of The Base Distribution Archive**

The step of creating the base distribution archive is done by creating an archive structure containing the files that are needed by the non-head node.
The filesystem that is archived in this way can differ from the special way that a Linux distribution unpacks and installs its filesystem on to a machine. This is because the distribution installer often carries out extra changes, for example in GRUB boot configuration. The creation of the base distribution archive is therefore a convenience to avoid working with the special logic of a distribution installer, which will vary across distributions and versions. Instead, the filesystem and contents of a node on which this parent distribution is installed—i.e. the end product of that logic—is what is dealt with.

The archive can be a convenient and standard tar.gz file archive (sometimes called the “base tar”), or, taking the step a little further towards the end result, the archive can be a fully expanded archive file tree.

Repository Access Considerations When Intending To Build A Base Distribution Archive
For convenience, the archive should be up-to-date. So, the base host used to generate the base distribution archive should ideally have updated files. If, as is usual, the base host is a regular node, then it should ideally be up to date with the repositories that it uses. Therefore running yum update or zypper up on the base host image, and then provisioning the image to the base host, is recommended in order to allow the creation of an up-to-date base distribution archive.

However sometimes updates are not possible or desirable for the base host. This means that the base host archive that is put together from the base host filesystem is an un-updated archive. The custom image that is to be created from the archive must then be also be created without accessing the repositories, in order to avoid dependency problems with the package versions. Exclusion of access to the repositories is possible by specifying options to the cm-create-image command, and is described in section 9.6.2.

Examples Of How To Build A Base Distribution Archive
In the following example, a base distribution tar.gz archive /tmp/BASEDIST.tar.gz is created from the base host basehost64:

Example

```bash
ssh root@basehost64 "
  "tar -cz --acls --xattrs 
  --exclude /etc/HOSTNAME --exclude /etc/localtime 
  --exclude /proc --exclude /lost+found --exclude /sys 
  --exclude /root/.ssh --exclude /var/lib/dhcpd/* 
  --exclude /media/floppy --exclude /etc/motd 
  --exclude /root/.bash_history --exclude /root/CHANGES 
  --exclude /etc/udev/rules.d/*persistent*.rules 
  --exclude /var/spool/mail/* --exclude /rhn 
  --exclude /etc/sysconfig/rhn/systemid --exclude /tmp/* 
  --exclude /var/spool/up2date/* --exclude /var/log/* 
  --exclude /etc/sysconfig/rhn/systemid.save 
  --exclude /root/mbox --exclude /var/cache/yum/* 
  --exclude /etc/cron.daily/rhn-updates /" > /tmp/BASEDIST.tar.gz
```

Or alternatively, a fully expanded archive file tree can be created from basehost64 by rsyncing to an existing directory (here it is /cm/images/new-image):

Example

```bash
rsync -av --acls --xattrs --hard-links --numeric-ids 
  --exclude="/etc/HOSTNAME" --exclude="/etc/localtime" --exclude="/proc" 
  --exclude="/lost+found" --exclude="/sys" --exclude="/root/.ssh" 
  --exclude="/var/lib/dhcpd/*" --exclude="/media/floppy" 
  --exclude="/etc/motd" --exclude="/root/.bash_history" 
  --exclude="/root/CHANGES" --exclude="/var/spool/mail/*"
```

© Bright Computing, Inc.
The --acls, --xattrs, and --hard-links options are only explicitly required—if they are valid—for earlier distribution versions of RHEL and SLES and their derivatives. This is indicated by the following table:

<table>
<thead>
<tr>
<th>Used by default?</th>
<th>--acls</th>
<th>--xattrs</th>
<th>--hard-links</th>
</tr>
</thead>
<tbody>
<tr>
<td>before RHEL7</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>before SLES12</td>
<td>*</td>
<td>*</td>
<td>no</td>
</tr>
<tr>
<td>RHEL7 and beyond</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SLES 12 and beyond</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

* The --acls, --xattrs options in both examples are invalid before SLES12, so can be dropped.

The defaults can be modified by adjusting the AdvancedConfig options RsyncHardLinks (page 604), RsyncXattrs (page 605), and RsyncAcls (page 605).

For distribution versions RHEL7 and its derivatives, and also for SLES12, extended attributes and ACLs are used as rsync options by default. This is expected to continue for future distribution versions.

For versions of RHEL and derivatives prior to RHEL7, the --acls and --xattrs options should be specified explicitly. For versions of SLES prior to SLES12, the --acls and --xattrs options are invalid, so can be dropped.

Having built the archive by following the examples suggested, the first step in creating the software image is now complete.

### 9.6.2 Creating The Software Image With cm-create-image

The second step, that of creating the image from the base archive, now needs to be done. This uses the cm-create-image utility, which is part of the cluster-tools package.

The cm-create-image utility uses the base archive as the base for creating the image. By default, it expects that the base distribution repositories be accessible just in case files need to be fetched from a repository package.

Thus, when the cm-create-image utility is run with no options, the image created mostly picks up the software only from the base archive. However, the image picks up software from the repository packages:

- if it is required as part of a dependency, or
- if it is specified as part of the package selection file (page 370).

If a repository package file is used, then it should be noted that the repository package files may be more recent compared with the files in the base archive. This can result in an image with files that are perhaps unexpectedly more recent in version than what might be expected from the base archive, which may cause compatibility issues. To prevent this situation, the --exclude option (section 9.2) can be used to exclude updates for the packages that are not to be updated.

Repository access can be directly to the online repositories provided by the distribution, or it can be to a local copy. For RHEL, online repository access can be activated by registering with the Red Hat
9.6 Creating A Custom Software Image

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 9.6.3 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 | --excludemrepo` 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 daemon 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

© Bright Computing, Inc.
-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 | --excludewvvendor <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/RHEL6.tar.gz -n rhel6-image
2. cm-create-image -a /tmp/RHEL6.tar.gz -n rhel6-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/RHEL6.tar.gz -n rhel6-image -i /cm/images/new-image -r
6. cm-create-image -a /tmp/RHEL6.tar.gz -n rhel6-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/rhel6-updates.repo
9. cm-create-image -d /cm/images/new-image -n bio-image -s -c /tmp/cm-rhel6.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/RHEL6.tar.gz, is written out to a software image named rhel6-image:

   cm-create-image --fromarchive /tmp/RHEL6.tar.gz --imagename rhel6-image

   The image with the name rhel6-image is created in the CMDaemon database, making it available for use by cmsh and cmgui. 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/rhel6-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/RHEL6.tar.gz is written out to the /cm/images/test-image directory. The software image is given the name rhel6-image:

   cm-create-image --fromarchive /tmp/RHEL6.tar.gz --imagename rhel6-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 --fromndir 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 9.5.2) in cmsh. It requires passwordless access to the node in order to work. Generic nodes, that is nodes outside the cluster, 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/RHEL6.tar.gz` and then placed under `/cm/images/new-image/`. Also, the software image is given the name `rhel6-image`:

```
cm-create-image --fromarchive /tmp/RHEL6.tar.gz --imagename rhel6-image --imagedir \
/cm/images/new-image --recreate
```

If the `--recreate`, option is not used, then the contents are simply overwritten, that is, the existing directory contents are copied over by the source content. It also means that old files on the destination directly 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`. The option 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/RHEL6.tar.gz`, is used to update the contents under the directory `/cm/images/new-image/`. The name of the image is also set to `rhel6-image`.

```
cm-create-image --fromarchive /tmp/RHEL6.tar.gz --imagename rhel6-image --imagedir \
/cm/images/new-image --updateimage
```

7. With the default Bright 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 Bright 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 Bright Cluster Manager packages are updated to the image bio-image in the directory /cm/images/bio-image.

8. The --basedistrorepo 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 Bright 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

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 CentOS6, then the configuration file used is:

/cm/local/apps/cluster-tools/config/CENTOS6-config-dist.xml

The package selection file is made up of a list of XML elements, specifying the name of the package, architecture and image type. For example:

```
...<package image="slave" name="apr" arch="x86_64"/>
<package image="slave" name="apr-util" arch="x86_64"/>
<package image="slave" name="atk-devel" arch="x86_64"/>
<package image="slave" name="autoconf" arch="noarch"/>
...```

The minimal set of packages in the list defines the minimal distribution that works with Bright Cluster Manager, and is the base-distribution set of packages, which may not work with some features of the distribution or Bright 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 Bright 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.

© Bright Computing, Inc.
9.6 Creating A Custom Software Image

- 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 Bright Cluster Manager. For example, outside the main Red Hat channel, but still within the parent distribution, there is a supplementary packages channel (Red Hat 5) or an optional packages channel (Red Hat 6). Packages from these optional/supplementary channels can be added to the base-distribution to enhance the capabilities of the image or resolve dependencies of Bright Cluster Manager. Section 7.5.1 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 Bright Cluster Manager, such as CUDA, cannot work correctly or cannot work at all.

The package selection file also contains entries for the packages that can be installed on the head (image="master") node. Therefore non-head node packages must have the image="slave" attribute.

Kernel Module Selection By cm-create-image
For an image created by cm-create-image, with a distribution <dist>, the default list of kernel modules to be loaded during boot are read from the file /cm/local/apps/cluster-tools/config/<dist>-slavekernelmodules. 

<dist> can take the value CENTOS6u7, CENTOS6u8, CENTOS7u1, CENTOS7u2, RHEL6u7, RHEL6u8, RHEL7u1, RHEL7u2, SL6u7, SL6u8, SL7u1, SL7u2, SLES11sp3, SLES11sp4, SLES12, SLES12sp1.

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 Bright Cluster Manager repository files, installing distribution package, finalizing image services, and installing Bright 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.<image name>, 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:

- In cmgui in the “Software Images” resource, by filling in the box for Path in the Settings tabbed pane for the image.
- In cmsh within softwareimage mode, for example:

  [bright73->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.

Workload Manager Reconfiguration On The Custom Regular Node Image
After a custom regular node image has been created by cm-create-image, and the workload manager on the custom node image is to be managed by Bright Cluster Manager, then the manager typically needs to be reconfigured. This can be done by running the wlm-setup utility using the --image option with the path to the custom node image (section 7.3).
9.6.3 Configuring Local Repositories For Linux Distributions, And For The Bright 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 9.6.2, using local base distribution repositories.

The administrator can choose to access an online repository provided by the distribution itself via a subscription as described in Chapter 5 of the Installation Manual. Another way to set up a repository is to set it up as a local repository, which may be offline, or perhaps set up as a locally-controlled proxy with occasional, restricted, updates from the distribution repository.

In the 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 Bright 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 372)
- Setting up a local repository for RHEL (page 373)
- Setting up a local repository for CentOS and Bright from the Bright Cluster Manager ISO for CentOS (page 373)

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 adrepo /root/repo1 "SLES11SP1-SDK"
zypper adrepo /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:

```
[based]
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:

```
imagedir=$imagedir
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/cm7.3-dvd.repo`

  can be created, for example, for a release tagged with a `<subminor>` number tag, with the content:

  ```
  [bright-repo]
  name=Bright Cluster Manager DVD Repo
  baseurl=file:///mnt1/data/cm-rpms/7.3-<subminor>
  enabled=1
  gpgcheck=1
  exclude = slurm* pbspro* sge* torque* cm-hwloc
  ```

- **for the regular node image:** A repository configuration file
  
  `$imagedir/etc/yum.repos.d/cm6.1-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@bright73 ~]# yum clean all
[root@bright73 ~]# 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.

9.6.4 Creating A Custom Image From The Local Repository

After having created the local repositories for SLES, RHEL or CentOS (section 9.6.3), a custom software image based on one of these can be created. For example, for CentOS, in a directory given the arbitrary name `offlineimage`:

```
cm-create-image -d $imagedir -n offlineimage -e -s
```

The `-e` option prevents copying the default cluster manager repository files on top of the image being created, since they may have been changed by the administrator from their default status. The `-s` option prevents installing additional base distribution packages that might not be required.
Cluster Monitoring

The Bright Cluster Manager monitoring framework lets a cluster administrator:

- inspect monitoring data to the required level for existing resources;
- configure gathering of monitoring data for new resources;
- see current and past problems or abnormal behavior;
- notice trends that help the administrator predict likely future problems;
- handle current and likely future problems by
  - triggering alerts;
  - taking action if necessary to try to improve the situation or to investigate further.

Powerful features are accessible within an intuitive monitoring framework, and customized complex setups can be constructed to suit the requirements of the administrator.

In this chapter, the monitoring framework 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 are acted on when a threshold is exceeded.

2. With this easy-to-understand example as the base, the various features and associated functionality of the Bright Cluster Manager monitoring framework are described and discussed in depth. These include visualization of data, concepts, configuration, monitoring customization and cmsh use.

10.1 A Basic Example Of How Monitoring Works

In this section, a minimal basic example of monitoring a process is set up. The aim is to present a simple overview that covers a part of what the monitoring framework is capable of handling. The overview gives the reader a structure to keep in mind, around which further details are fitted and filled in during the coverage in the rest of this chapter.

In the example, a user runs a large number of pointless CPU-intensive processes on a head node which is normally very lightly loaded. An administrator would then want to monitor user mode CPU load usage, and stop such processes automatically when a high load is detected (figure 10.1).

The basic example illustrates a (very contrived) way for the Bright Cluster Manager monitoring framework to be used to do that.
10.1.1 Before Using The Framework—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 usually used to answer prompts for confirmation.

8 subshell processes are run 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, and which is reported as CPUUser in the Bright Cluster Manager metrics, is close to 100% 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” is used. It is made a part of a script killallyes:

```bash
#!/bin/bash
killall yes
```

and made executable with a `chmod 700 killallyes`. For convenience, it may be placed in the `/cm/local/apps/cmd/scripts/actions` directory where other action scripts also reside.

10.1.2 Using The Framework

Now that the pieces are in place, cmgui’s monitoring framework is used to add the action to its action list, and then set up a threshold level that triggers the action:

Adding The Action To The Actions List

From the resources tree of cmgui, Monitoring Configuration is selected, and then the Actions tab is selected. A list of currently available actions is displayed. A new action is added by entering the following values in the Add dialog (figure 10.2):

- **action name:** killallyes
- **description:** kill all yes processes
- **command:** /cm/local/apps/cmd/scripts/actions/killallyes
10.1 A Basic Example Of How Monitoring Works

The **Save** button adds the action killallyes to the list of possible actions, which means that the action can now be used throughout the monitoring framework.

**Setting Up The Threshold Level For CPUUser On The Head Node(s)**

Continuing on, the **Metric Configuration** tab is selected. Then within the selection box options for Metric Configuration, All Head Nodes is selected to confine the metrics being measured to the head node(s). The metric CPUUser, which is a measure of the user mode CPU usage as a percentage, is selected. The **Thresholds** button is clicked on to open a Thresholds dialog. Within the “New Threshold” dialog (figure 10.3), these values are set:

- **threshold name**: killallyesthreshold
- **(upper) bound**: 50
- **action name (first selection box in the action option)**: killallyes
- **action state option (next selection box in the action option)**: Enter

Clicking on **Ok** exits the “New Threshold” dialog, clicking on **Done** exits the Thresholds dialog, and clicking on **Save** saves the threshold setting associated with CPUUser on the head node.

© Bright Computing, Inc.
Figure 10.3: cmgui Monitoring Configuration: Setting A Threshold
The Result
In the preceding section, an action was added, and a threshold was set up with the monitoring framework.

With a default installation on a newly installed cluster, the measurement of CPUUser is done every 120s (the edit dialog of figure 10.23 shows how to edit this value). The basic example configured with the defaults thus monitors if CPUUser on the head node has crossed the bound of 50% every 120s.

If CPUUser is found to have entered, that is crossed over from below the value and gone into the zone beyond 50%, then the framework runs the killallyes script, killing all running yes processes. Assuming the system is trivially loaded apart from these yes processes, the CPUUser metric value then drops to below 50%.

After an Enter threshold condition has been met for a sample, the first sample immediately after that does not ever meet the Enter threshold 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 threshold 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 being illustrated here, and is of no real concern.

10.2 Monitoring Concepts And Definitions
A discussion of the concepts of monitoring, along with definitions of terms used, is appropriate at this point. The features of the monitoring framework covered later on in this chapter will then be understood more clearly.

10.2.1 Metric
In the basic example of section 10.1, the metric value considered was CPUUser, measured at regular time intervals of 120s.

A metric is a property of a device that can be monitored. It has a numeric value and can have units, unless it is unknown, i.e. has a null value. Examples are:

- temperature (value in degrees Celsius, for example: 45.2 °C);
- load average (value is a number, for example: 1.23);
- free space (value in bytes, for example: 12322343).

A metric can be a built-in, which means it is an integral part of the monitoring framework, or it can be a standalone script.

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.

10.2.2 Action
In the basic example of section 10.1, the action script is the script added to the monitoring system to kill all yes processes. The script runs when the condition is met that CPUUser crosses 50%.

An action is a standalone script or a built-in command that is executed when a condition is met. This condition can be:

- health checking (section 10.2.4);
- threshold checking (section 10.2.3) associated with a metric (section 10.2.1);
- state flapping (section 10.2.9).
10.2.3 Threshold

In the basic example of section 10.1, a threshold is set to 50% of CPUUser, and an action set so that crossing this threshold runs the killallyes script.

A threshold is a particular value in a sampled metric. A sample can cross the threshold, thereby entering or leaving a zone that is demarcated by the threshold.

A threshold can be configured to launch an action (section 10.2.2) according to threshold crossing conditions. The “New Threshold” dialog of cmgui (figure 10.3) has three action launch configuration options:

1. **Enter:** if the sample has entered into the zone and the previous sample was not in the zone
2. **Leave:** if the sample has left the zone and the previous sample was in the zone
3. **During:** if the sample is in the zone, and the previous sample was also in the zone.

A threshold zone also has a settable severity (section 10.2.6) associated with it. This value is processed for the AlertLevel metric (section 10.2.7) when an action is triggered by a threshold event.

10.2.4 Health Check

A health check value is a state. It is the response to running a health check script at a regular time interval, with as possible response values: PASS, FAIL, or UNKNOWN. The state is recorded in the CMDaemon database, and in cmgui an overview can be seen in the Overview tab for the device, in the Health Status section.

Examples of health checks are:

- checking if the hard drive still has enough space left on it and returning PASS if it has;
- checking if an NFS mount is accessible, and returning FAIL if it is not;
- checking if CPUUser is below 50%, and returning PASS if it is;
- checking if the cmsh binary is found, and returning UNKNOWN if it is not.

A health check has a settable severity (section 10.2.6) associated with a FAIL or UNKNOWN response. The severity can be set by appending a value to the response. For example, FAIL 40 or UNKNOWN 10, as is done in the hpraid health check (/cm/local/apps/cmd/scripts/healthchecks/hpraid).

Severity values are processed for the AlertLevel metric (section 10.2.7) when the health check runs.

A health check can also launch an action based on any of the response values, similar to the way that an action is launched by a metric with a threshold condition.

10.2.5 Conceptual Overview: Health Checks Vs Threshold Checks

A health check is quite similar to a threshold state check with a metric. Conceptually, however, they are intended to differ as follows:

- A threshold state check works with numeric values.
- A health check on the other hand works with a response state of PASS, FAIL, or UNKNOWN.

- Threshold-checking does not specifically store a direct history of whether the threshold condition was met or not—it just calls the action script right away as its response. Admittedly, the associated metric data values are still kept by the monitoring database that CMDaemon records data into, so that establishing if a threshold has been crossed historically is always possible with a little effort.

- A health check on the other hand stores its PASS/FAIL/UNKNOWN responses for the monitoring framework, making it easily accessible for viewing by default.
• The threshold-checking mechanism is intended to be limited to doing a numerical comparison of
a metric value with a threshold value

A health check on the other hand has more general checking capabilities.

With some inventiveness, a health check can be made to do the function of a metric’s threshold/action
sequence (as well as the other way round).

The appropriate monitoring tool for the job can thus be either a health check, or a metric check,
depending on what makes more sense to the administrator.

10.2.6 Severity

Severity is a positive integer value that the administrator assigns to a threshold-crossing event or to a
health check status event. It takes one of these 5 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>info</td>
<td><img src="image" alt="Info Icon" /></td>
<td>informational message</td>
</tr>
<tr>
<td>10</td>
<td>notice</td>
<td><img src="image" alt="Warning Icon" /></td>
<td>normal, but significant, condition</td>
</tr>
<tr>
<td>20</td>
<td>warning</td>
<td><img src="image" alt="Warning Icon" /></td>
<td>warning conditions</td>
</tr>
<tr>
<td>30</td>
<td>error</td>
<td><img src="image" alt="Error Icon" /></td>
<td>error conditions</td>
</tr>
<tr>
<td>40</td>
<td>alert</td>
<td><img src="image" alt="Alarm Icon" /></td>
<td>action must be taken immediately</td>
</tr>
</tbody>
</table>

By default the value is 10. Severity levels are used in the AlertLevel metric (section 10.2.7). They
can also be set by the administrator in the return values of health check scripts (section 10.2.4) and during
event bucket processing (section 10.6.3). In the showhealth command (section 10.8.5), they give the
administrator an overview of cluster health.

10.2.7 AlertLevel

AlertLevel is a special metric. It is not sampled, but it is re-calculated when an event with an associated
Severity (section 10.2.6) occurs. There are two types of AlertLevel metrics:

1. 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.

2. AlertLevel (sum): the sum of the latest severity values of all the events. The aim of this metric
   is to alert the administrator to the overall severity of issues.

10.2.8 InfoMessages

InfoMessages are optional messages that inform the administrator of the reason for a health status change,
or give a reason for a particular metric value in the cluster. These show up in the Overview tab of nodes,
in the Health Status section for metrics and for health checks.

Metric or health check scripts can use file descriptor 3 within their scripts to write an InfoMessage:

Example

```
example echo "Drive speed unknown: Reverse polarity" >&3
```

The AdvancedConfig directives InfoMessageCacheSize, MaxInfoMessageSize, MaxInfoMessagesPerMetric, can be used to control their behavior with metric and healthcheck
values. These directives are described further starting on page 605.
10.2.9 Flapping

Flapping, or State Flapping, is when a state transition (section 10.2.10) occurs too many times over a number of samples. In the basic example of section 10.1, if the CPUUser metric crossed the threshold zone 7 times within 12 samples (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. Flapping configuration for cmgui is covered for thresholds crossing events in section 10.4.2, when the metric configuration tab’s Edit and Add dialogs are explained; and also covered for health check state changes in section 10.4.3, when the health check configuration tab’s Edit and Add dialogs are explained.

10.2.10 Transition

A state transition is:

- a health check state change (for example, changing from PASS to FAIL, or from FAIL to UNKNOWN);
- a metric threshold (section 10.2.3) crossing event. This is only valid for values that Enter or Leave the threshold zone.

10.2.11 Conceptual Overview: cmgui’s Main Monitoring Interfaces

Monitoring information is presented in several places in cmgui for convenience during everyday use. The conceptual overview in figure 10.4 covers a commonly seen layout in cmgui, showing 4 monitoring-related viewing areas for the cluster administrator. These are:

1. **Visualization**

Figure 10.4: cmgui Conceptual Overview - Monitoring Types
Visualization of monitoring data is made available from cmgui’s monitoring menu, and launches a new window. Graphs are generated from metrics and health checks data, and these graphs are viewed in various ways within window panes.

The use of the visualization tool is covered in section 10.3 using typical data from CPUUser from the basic example of section 10.1.

2. Monitoring Configuration

Selecting the Monitoring Configuration resource in cmgui from the Resources list on the left hand side of the Bright Cluster Manager displays the monitoring configuration pane on the right hand side. Within this pane, the following tabs show up:

- **Overview**: an overview of enabled actions
- **Metric Configuration**: allows configuration of device categories with metrics
- **Health Check Configuration**: allows configuration of device categories with health checks
- **Metrics**: allows configuration of metrics for devices
- **Health Checks**: allows configuration of health checks for devices
- **Actions**: allows actions to be set to run from metric thresholds and health check results

Some parts of Monitoring Configuration were used in the basic example of section 10.1 to set up the threshold for CPUUser, and to assign the action. It is covered more thoroughly in section 10.4.

3. Event Viewer

The Event Viewer displays events that are seen on the cluster(s) within a pane of cmgui. How the events are presented is configurable, with tools that allow filtering based on dates, clusters, nodes or a text string; and widgets that allow rearranging the sort order or detaching the pane.

4. Overview Of Monitored Data

A dashboard in a car conveys the most important relevant information at a glance and attracts attention to items that are abnormal and merit further investigation.

The same idea lies behind the Overview tab of Bright Cluster Manager. This gives a dashboard view based on the monitored data for a particular device such as a switch, a cluster (probably the most useful overview, and therefore also the default when first connecting to the cluster with cmgui), a node, a GPU unit, and so on.

Neighboring tabs often allow a closer look at issues noticed in the Overview, and also sometimes a way to act on them.

For example, if jobs are not seen in the Overview tab, then the administrator may want to look at the neighboring Services tab (figure 10.5), and see if the workload manager is running. The Services tab (section 3.11.2) allows the administrator to manage a service such as the workload manager.
10.3 Monitoring Visualization With cmgui

The Monitoring option in the menu bar of cmgui (item 1 in figure 10.4) launches an intuitive visualization tool that should be the main tool for getting a feel of the system’s behavior over periods of time. With this tool the measurements and states of the system are viewed. Graphs for metrics and health checks can be looked at in various ways: for example, 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 cmgui’s visualization tool is the command-line cmsh. This has the same functionality in the sense that data values can be selected and studied according to configurable parameters with it (section 10.8). The data values can even be plotted and displayed on graphs with cmsh with the help of unix pipes and graphing utilities. However, the strengths of monitoring with cmsh lie elsewhere: cmsh is more useful for scripting or for examining pre-decided metrics and health checks rather than a quick visual check over the system. This is because cmsh needs more familiarity with options, and is designed for text output instead of interactive graphs. Monitoring with cmsh is discussed in sections 10.7 and 10.8.

How cmgui is used for visualization is now described.

10.3.1 The Monitoring Window

The Monitoring menu is selected from the menu bar of cmgui and a cluster name is selected. The Monitoring window opens (figure 10.6). The resources in the cluster are shown on the left side of the window. Clicking on a resource opens or closes its subtree of metrics and health checks.

The subsequent sections describe ways of viewing and changing resource settings. After having carried out such modifications, saving and loading a settings state can be done from options in the File menu.

Figure 10.6 shows the different resources of the head node, with the CPU resource subtree opened up in turn to show its metrics and health checks. Out of these, the CPUUser metric (for user CPU usage) is shown selected for further display.

To display this metric, the selection is drag-and-dropped onto one of the 3 panes which has the text “drop sensor here”.

10.3.2 The Graph Display Pane

Figure 10.7 shows the monitoring window after such a drag-and-drop. The graph of the metric CPUUser is displayed over 20 minutes (10th November 2010 08:04 to 08:24). On the y-axis the unit used by the metric is shown (0% to about 100%). This example is actually of data gathered when the basic example
of 10.1 was run, and shows CPUUser rising as a number of yes processes are run, and falling when they end.

Features of graph display panes are (figure 10.8):

1. **The close widget** which erases all graphs on the drawing pane when it is clicked. (Individual graphs are removed in the settings dialog discussed in section 10.3.5.)

2. **The (time, measurement) data values** in the graph are displayed on the graph toolbar by hovering the mouse cursor over the graph.

3. **The graph view adjustment buttons** are:
   - **play/pause**: by default the graph is refreshed with new data every 2 minutes. This is disabled and resumed by clicking on the pause/play button on the graph toolbar.
   - **zoom-out/zoom-in**: Clicking on one of the magnifying glasses zooms-in or zooms-out on the graph in time. This way data values can be shown, even from many months ago. Zooming in with mouse gestures is also possible and is discussed in section 10.3.4.
Figure 10.8: Graph Display Pane: Features

- **broadcast**: A time-scale synchronizer. Toggling this button to a pressed state for one of the graphs means that scale changes carried out via magnifying glass zooms (preceding bullet point) or via mouse gestures (section 10.3.4) are done over all the other graph display panes too so that their x-ranges match. This is useful for large numbers of nodes.

- **settings**: Clicking on this button opens a dialog window to modify certain aspects of the graph. The settings dialog is discussed in section 10.3.5.

4. A **grid of graph display panes** can be laid out by using the Grid menu option of the main Monitoring Pane (figure 10.6). Among the menu options of the Grid menu (figure 10.9) are:

![Figure 10.9: Grid Menu Options](image)

(a) **Layout**: a grid of dimensions $x \times y$ can be selected or specified with the Layout option. With the Layout option, metrics need to be added manually to each grid unit.

(b) **Grid plot wizard**: For larger grids it is tedious to allocate devices to a grid and manually fill in the grid units with metrics. The Grid plot wizard can take care of the tedious aspects, and is described in section 10.3.3.
5. **Multiple graphs** are drawn in a single graph display pane by repeating the drag and drop for different metrics. For example, adding the CPUIdle metric with a drag-and-drop to the CPUUser graph of figure 10.7 gives a result as seen in figure 10.10, where both graphs lie on the same axis in the top pane.

![Figure 10.10: Graph Display Pane: Multiple Graphs On One Pane](image)

6. **Boxplots** (sometimes called box-and-whisker plots) are used to draw graphs for samples carried out over the interval for a particular time on a resource item that has multiple devices in it. For example, in the resource item `node001`, CPUUser is sampled and plotted for a single node for set time intervals, and so it is displayed as a line plot. However, for the resource item “All nodes”, CPUUser is sampled and plotted for all nodes for set time intervals, and so it is displayed as a boxplot (figure 10.11):

![Figure 10.11: Graph Display Pane: Boxplots](image)

Resource items that have multiple devices are indicated by starting with the text “All”. For example: “All nodes”, “All head nodes”, and so on.

The boxplot displayed shows the lowest and highest values at the whisker ends. Typically, the
lower and upper quartile values are the box borders, and the median value is the middle divider of the box. In addition, outlier points can be displayed. If there are only two outlier points, they are the same as the lowest and highest values. Outlier points and how to set them are described in section 10.3.5.

The `dumpstatistics` command in `cmsh` is somewhat similar in function to boxplots in `cmgui`.

### 10.3.3 Using The Grid Wizard

Within the Monitoring window (section 10.3.1), the Grid plot wizard sets up a grid for devices selected by the administrator, and allows metrics to be added automatically to each grid unit.

The first screen of the wizard allows devices to be selected from the group of all devices, and placed in a group of devices that are to have their metrics plotted (figure 10.12).

![Grid plot wizard](image)

Figure 10.12: Grid Wizard: Devices Selection

The next screen of the wizard allows metrics to be drag-and-dropped from the available metrics into a group of metrics that are to be displayed for the devices in the previous screen (figure 10.13).
The last screen of the wizard allows several display options to be set for the selected devices and their metrics (figure 10.14).
One of these options is the specification of the layout width and height for the displayed grid of selected devices. For example, four nodes could be laid out in a grid of 4 wide by 1 high, 2 wide by 2 high, or 1 wide by 4 high. The meanings of the remaining display options are described in section 10.3.5.

Once the Finish button of the last screen is clicked, a graph display pane is shown with a grid of graphs (figure 10.15).
10.3 Monitoring Visualization With cmgui

10.3.4 Zooming In With Mouse Gestures

Besides using a magnifying glass button there are two other ways to zoom in on a graph, based on intuitive mouse gestures:

X-Axis Zoom

The first way to zoom in is to draw a horizontal line across the graph by holding the left mouse button down on the graph. A guide line shows up while doing this (figure 10.16):

The x-axis range covered by this line is zoomed in on when the mouse button is released (figure 10.17):

© Bright Computing, Inc.
Box Zoom

The second way to zoom in is to draw a box instead of a line across the graph by holding the left mouse button down and drawing a line diagonally across the data instead of horizontally. A guide box shows up (figure 10.18):

This is zoomed into when the mouse button is released (figure 10.19):
10.3 Monitoring Visualization With cmgui

10.3.5 The Graph Display Settings Dialog
Clicking on the settings button in the graph display pane (figure 10.8) opens up the graph display pane settings dialog (figure 10.20):

This allows the following settings to be modified:

- the Title shown at the top of the graph;
- over When the x-range is displayed;
- The existence and positioning of the Legend;
- the Intervals value. This is the number of intervals (by default 200) used to draw the graph. For example, although there may be 2000 data points available during the selected period, by
default only 200 are used, with each of the 200 an average of 10 real data points. This mechanism is especially useful for smoothing out noisier metrics to give a better overview of behavior.

- The **Refresh Interval**, which sets how often the graph is recreated;
- the visual layout of the graphs, which can be adjusted so that:
  - Color aspects of each graph are changed in the row of settings for that graph;
  - Each graph is deleted from its pane with the \( \ominus \) button at the end of the row of settings for that graph;
  - The number of points representing Outliers can be set for each boxplot (page 387) displayed. The points are chosen based on their distance being a maximum from the median, with the order in which they are chosen being alternately above and below the median.

### 10.4 Monitoring Configuration With cmgui

This section is about the configuration of monitoring for health checks and metrics, along with setting up the actions which are triggered from a health check or a metric threshold check.

Selecting Monitoring Configuration from the resources section of cmgui makes the following tabs available (figure 10.21):

![Figure 10.21: cmgui Monitoring Configuration Tabs](image)

- **Overview** (displays as the default)
- **Metric Configuration**
- **Health Check Configuration**
- **Metrics**
- **Health Checks**
- **Actions**

The tabs are now discussed in detail.

#### 10.4.1 The Overview Tab

The **Overview** tab of figure 10.21 shows an overview of custom threshold actions and custom health check actions that are active in the system. Each row of conditions in the list that decides if an action is launched is called a rule. Only one rule is on display in figure 10.21, showing an overview of the metric threshold action settings which were set up in the basic example of section 10.1.

© Bright Computing, Inc.
The Add rule button runs a convenient wizard that guides an administrator in setting up a condition, and thereby avoids having to go through the other tabs separately.

The Remove button removes a selected rule.

The Edit button edits aspects of a selected rule. It opens a dialog that edits a metric threshold configuration or a health check configuration. These configuration dialog options are also accessible from within the Metric Configuration and Health Check Configuration tabs.

The Revert button reverts a modified state of the tab to the last saved state.

The Save button saves a modified state of the tab.

### 10.4.2 The Metric Configuration Tab

The Metric Configuration tab allows device categories to be selected for the sampling of metrics. Properties of metrics related to the taking of samples can then be configured from this tab for the selected device category. These properties are the configuration of the sampling parameters themselves (for example, frequency and length of logs), but also the configuration of related properties such as thresholds, consolidation, actions launched when a threshold is crossed, and actions launched when a metric state is flapping.

The Metric Configuration tab is initially a blank tab until the device category is selected by using the Metric Configuration selection box. The selection box selects the device category from a list of built-in categories and user-defined node categories (node categories are introduced in section 2.1.3). On selection, the metrics of the selected device category are listed in the Metric Configuration tab. Properties of the metrics related to sampling are only available for configuration and manipulation after the metrics list displays. Handling metrics in this manner, via groups of devices, is slightly awkward for just a few machines, but for larger clusters it keeps administration scalable and thus manageable.

Figure 10.22 shows an example of the Metric Configuration tab after All master nodes is chosen as the device category. This corresponds to the basic example of section 10.1, where All master nodes was the device category chosen because it was the CPUUser metric on a master node that was to be monitored. Examples of other device categories that could be chosen are All ethernet switches, if Ethernet switches are to have their metrics configured; or All Power Distribution Units, if power distribution units are to have their metrics configured.

![Figure 10.22: cmgui Monitoring: Metric Configuration Display After Category Selection](image-url)
With the screen displaying a list of metrics as in figure 10.22, the metrics in the Metric Configuration tab can now be configured and manipulated. The buttons used to do this are: Edit, Add, Remove, Thresholds, Consolidators, Revert and Save.

The Save button saves as-yet-uncommitted changes made via the Add or Edit buttons.

The Revert button discards unsaved edits made via the Edit button. The reversion goes back to the last save.

The Remove button removes a selected metric from the metrics listed.

The remaining buttons, Edit, Add, Thresholds and Consolidators, open up options dialogs. These options are now discussed.

Metric Configuration Tab: Edit And Add Options

The Metric Configuration tab of figure 10.22 has Add and Edit buttons. The Add button opens up a dialog to add a new metric to the list, and the Edit button opens up a dialog to edit a selected metric from the list. The dialogs allow logging options for a metric to be set or adjusted. For example, a new metric could be set for sampling by adding it to the device category from the available list of all metrics, or the sampling frequency could be changed on an existing metric, or an action could be set for a metric that has a tendency to flap.

The Edit and Add dialogs for a metric have the following options (figure 10.23):

- **Metric**: The name of the metric.
- **Parameter**: Values that the metric script is designed to handle. For example:
  - the metric `FreeSpace` tracks the free space left in a filesystem, and is given a mount point such as / or /var as a parameter;
  - the metric `BytesRecv` measures the number of bytes received on an interface, and takes an interface name such as eth0 or eth1 as a parameter.

  For CPUUser, the parameter field is disallowed in the Metric tab, so values here are ignored.
- **Log length**: The maximum number of raw data samples that are stored for the metric. 3000 by default.

![Figure 10.23: cmgui Monitoring: Metric Configuration Tab Edit Dialog](image)

© Bright Computing, Inc.
10.4 Monitoring Configuration With cmgui

- **Sampling interval**: The time between samples. 120s by default.

- **Gap size**: The number of missing samples allowed before a null value is stored as a sample value. 2 by default.

- **Threshold duration**: Number of samples in the threshold zone before a threshold event is decided to have occurred. 1 by default. The `failbeforedown` option to the open command (page 194) is actually a special use of this option.

- **Options checkboxes**:
  - **Store**: If ticked, the metric data values are saved to the database. Note that any threshold checks are still done, whether the samples are stored or not.
  - **Disabled**: If ticked, the metric script does not run, and no threshold checks are done for it. If Store is also ticked, no value is stored.
  - **Only when idle**: If ticked, the metric script is only run when the system is idling. A resource-hungry metric burdens the system less this way.

- **State Flapping**: An action can be set for the metric, with 3 parameters that can be set per action.
  1. The first parameter can be set from a menu selection option, which decides what action to launch if state flapping is detected. Possible actions are:
     - Drain node
     - Power off
     - Power on
     - Power reset
     - Reboot
     - Restart service
     - SendEmail: sends an email to a user specified in the second box
     - Shutdown
     - Start service
     - Stop service
     - Undrain node
     - Update from image
     - Killprocess: kills a process, if given process IDs as comma-separated parameters in the second box
     - remount: tries to remount all defined mounts
     - testaction: a test script

     * Scripts for these are located under `/cm/local/apps/cmd/scripts/actions`

  2. The second parameter is a plain text-entry box that allows a parameter to be passed to the action.

  3. The third parameter is a menu selection option again, which decides when to launch the action, depending on which of these following states is set:
     - **Enter**: if the flapping has just started. That is, the current sample is in a flapping state, and the previous sample was not in a flapping state.
     - **During**: if the flapping is ongoing. That is, the current and previous flapping sample are both in a flapping state.
     - **Leave**: if the flapping has just stopped. That is, the current sample is not in a flapping state, and the previous sample was in a flapping state.
Metric Configuration Tab: Thresholds Options

The Metric Configuration tab of figure 10.22 also has a Thresholds button associated with a selected metric.

Thresholds are defined and their underlying concepts are discussed in section 10.2.3. The current section describes the configuration of thresholds.

In the basic example of section 10.1, CPUUser was configured so that if it crossed a threshold of 50%, it would run an action (the killallyyes script). The threshold configuration was done using the Thresholds button of cmgui.

Clicking on the Thresholds button launches the Thresholds display window, which lists the thresholds set for that metric. Figure 10.24, which corresponds to the basic example of section 10.1, shows a Thresholds display window with a threshold named killallyesthreshold configured for the metric CPUUser.

The Edit, and Remove buttons in this display edit and remove a selected threshold from the list of thresholds, while the Add button adds a new threshold to the list.

The Edit and Add dialogs for a threshold prompt for the following values (figure 10.25):

- **Name**: the threshold's name.
10.4 Monitoring Configuration With cmgui

Figure 10.26: cmgui Metric Configuration: Consolidators Display

- **Bound**: the metric value which demarcates the threshold.
- **Bound type**: If checked, the radio button for
  - upper bound: places the threshold zone above the bound;
  - lower bound: places the threshold zone below the bound.
- **Severity**: A value assigned to indicate the severity of the situation if the threshold is crossed. It is 10 by default. Severity is discussed in section 10.2.6.
- **Action**: The action field types decide how the action should be triggered and run. The field types are, from left to right:
  - script: a script selected from a drop-down list of available actions;
  - parameter: [optional] what parameter value to pass to the action;
  - when: when the action is run. It is selected from a drop-down choice of Enter, During or Leave, where:
    - Enter runs the action if the sample has entered the zone;
    - Leave runs the action if the sample has left the zone;
    - During runs the action if the sample is in the zone, and the previous sample was also in the zone.

Metric Configuration Tab: Consolidators Options

The Metric Configuration tab of figure 10.22 also has a Consolidators button associated with the selected metric.

Consolidators decide how the data values are handled once the initial log length quantity for a metric is exceeded. Data points that have become old are gathered and, when enough have been gathered, they are processed into consolidated data. Consolidated data values present fewer data values than the original raw data values over the same time duration. The aim of consolidation is to increase performance, save space, and keep the basic information still useful when viewing historical data.

The Consolidators button opens a window that displays a list of consolidators that have been defined for the selected metric (figure 10.26).

The Edit and Remove buttons in this display edit and remove a selected consolidator from the list of consolidators while the Add button in this display adds a new consolidator to the list of consolidators.

The Edit and Add dialogs for a consolidator prompt for the following values (figure 10.27):

- **Name**: the consolidator’s name. By default Day, Hour, and Month are already set up, with appropriate values for their corresponding fields.
400 Cluster Monitoring

Figure 10.27: cmgui Metric Configuration: Consolidators Edit Dialog

- **Length**: the number of intervals that are logged for this consolidator. Not to be confused with the metric log length.

- **Interval**: the time period (in seconds) associated with the consolidator. Not to be confused with the metric interval time period. For example, the default consolidator with the name Hour has a value of 3600.

- **Time Offset**: the time offset from the default consolidation time.

  To understand what this means, consider the Log length of the metric, which is the maximum number of raw data points that the metric stores. When this maximum is reached, the oldest data point is removed from the metric data when a new data point is added. Each removed data point is gathered and used for data consolidation purposes.

  For a metric that adds a new data point every Sampling interval seconds, the time $t_{raw\ gone}$, which is how many seconds into the past the raw log data point is removed, is given by:

  $$t_{raw\ gone} = (\text{Log length})_{\text{metric}} \times (\text{Sampling interval})_{\text{metric}}$$

  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 Time Offset has its default, zero value.

  If however the Time 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:

  $$t_{consolidation} = t_{raw\ gone} + \text{Time Offset}$$

  The monitoring visualization graphs then show consolidated data from $t_{consolidation}$ seconds into the past, to further into the past.

- **Kind**: the kind of consolidation done on the raw data samples. 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.

  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.

10.4.3 Health Check Configuration Tab

The Health Check Configuration tab behaves in a similar way to the Metric Configuration tab of section 10.4.2, with some differences arising due to working with health checks instead of metric values.

---

© Bright Computing, Inc.
The Health Check Configuration tab allows device categories to be selected for the evaluating the states of health checks. Properties of health checks related to the evaluating these states can then be configured from this tab for the selected device category. These properties are the configuration of the state evaluation parameters themselves (for example, frequency and length of logs), but also the configuration of related properties such as severity levels based on the evaluated state, the actions to launch based on the evaluated state, or the action to launch if the evaluated state is flapping.

The Health Check Configuration tab is initially a blank tab until the device category is selected by using the Health Check Configuration selection box. The selection box selects a device category from a list of built-in categories and user-defined node categories (node categories are introduced in section 2.1.3). On selection, the health checks of the selected device category are listed in the Health Check Configuration tab. Properties of the health checks related to the evaluation of states are only available for configuration and manipulation after the health checks list is displayed. Handling health checks in this manner, via groups of devices, is slightly awkward for just a few machines, but for larger clusters it keeps administration scalable and thus manageable.

![cmgui Monitoring: Health Check Configuration Display After Category Selection](image)

Figure 10.28 shows an example of the Health Check Configuration tab after All master nodes is chosen as the category. Examples of other categories that could be chosen to have health checks carried out on them are All ethernet switches and All Power Distribution Units.

With the screen displaying a list of health checks as in figure 10.28, the health checks in the Health Check Configuration tab can now be configured and manipulated. The buttons used to do this are: Edit, Add, Remove, Revert and Save.

These Health Configuration tab buttons behave just like the corresponding Metric Configuration tab buttons of section 10.4.2, that is:

- The Save button saves as-yet-uncommitted changes made via the Add or Edit buttons.
- The Revert button discards unsaved edits made via the Edit button. The reversion goes back to the last save.
- The Remove button removes a selected health check from the health checks listed.
- The remaining buttons, Edit and Add, open up options dialogs. These are now discussed.

Health Check Configuration Tab: Edit And Add Options

The Health Check Configuration tab of figure 10.28 has Add and Edit buttons. The Add button opens up a dialog to add a new health check to the list, and the Edit button opens up a dialog to edit a selected health check from the list. The dialogs are very similar to those of the Add and Edit options of...
Metric Configuration in section 10.4.2. The dialogs for the Health Check Configuration tab are as follows (figure 10.29):

- **Health Check**: The name of the health check.
- **Parameter**: The values that the health check script is designed to handle. For example:
  - The health check `ldap` checks if the ldap service is running. It tests the ability to look up a user on the LDAP server using `cmsgsupport` as the default user. If a value is specified for the parameter, it uses that value as the user instead;
  - The health check `portchecker` takes parameter values such as `192.168.0.1 22` to check if host 192.168.0.1 has port 22 open.
- **Log length**: The maximum number of samples that are stored for the health check. 3000 by default.
- **Sampling interval**: The time between samples. 120s by default.
- **Prejob**: Clicking on this button sets the health check to run before a new job is run from the scheduler of the workload management system, instead of running at regular intervals.
- **Gap size**: The number of missing samples allowed before a null value is stored as a sample value. 2 by default.
- **Threshold duration**: Number of samples in the threshold zone before a health check state is decided to have changed. 1 by default. The `failbeforedown` option to the `open` command (page 194) is actually a special use of this option.
- **Fail severity**: The severity value assigned to a FAIL response for a health check. 10 by default.
- **Unknown severity**: The severity value assigned to an UNKNOWN response for a health check. 10 by default.
- **Options checkboxes**:
  - **Store**: If ticked, the health check state data values are saved to the database. Note that health state changes and actions still take place, even if no values are stored.
  - **Disabled**: If ticked, the health state script does not run, and no health check state changes or actions associated with it occur. If **Store** is ticked, the value it stores while **Disabled** is ticked for this health check configuration is an **UNKNOWN** value.
  - **Only when idle**: If ticked, the health check script is only run when the system is idling. This burdens a system less, and is useful if the health check is resource-hungry.
- **Pass action, Fail action, Unknown action, State Flapping**: These are all action launchers, which launch an action for a given health state (PASS, FAIL, UNKNOWN) or for a flapping state, depending on whether these states are true or false. Each action launcher is associated with three input option types:
  1. The first input option type is a menu selection to set what action to launch if the state is true. Possible actions are the same as the actions previously listed for the metric configuration tab for state flapping (page 397).
  2. The second input option type is a plain text-entry box that allows a parameter to be passed to the action.
  3. The third input option type is a menu selection that sets when to launch the action, depending on which of the following conditions is met:
- **Enter**: if the state has just started being true. That is, the current sample is in that state, and the previous sample was not in that state.
- **During**: if the state is true, and ongoing. That is, the current and previous state sample are both in the same state.
- **Leave**: if the state has just stopped being true. That is, the current sample is not in that state, and the previous sample was in that state.

**Figure 10.29: cmgui Monitoring: Health Check Configuration Edit Dialog**

### 10.4.4 Metrics Tab

The **Metrics** tab displays the list of metrics that can be set in the cluster. Some of these metrics are built-ins, such as `CPUUser` in the basic example of section 10.1. Other metrics are standalone scripts. New custom metrics can also be built and added as standalone commands or scripts. A useful template for such a script is the `testmetric` script (Appendix G.1.1).

Metrics can be manipulated and configured. The **Save** button saves as-yet-uncommitted changes made via the **Add** or **Edit** buttons.

The **Revert** button discards unsaved edits made via the **Edit** button. The reversion goes back to the last save.

The **Remove** button removes a selected metric from the list.

The remaining buttons, **Edit** and **Add**, open up options dialogs. These are now discussed.

**Metrics Tab: Edit And Add Options**

The **Metrics** tab of figure 10.30 has **Add** and **Edit** buttons. The **Add** button opens up a dialog to add a new metric to the list, and the **Edit** button opens up a dialog to edit a selected metric from the list. Both dialogs have the following options (figure 10.31):

- **Name**: the name of the metric.
- **Description**: the description of the metric.
Figure 10.30: cmgui Monitoring: Metrics Tab

Figure 10.31: cmgui Monitoring: Metrics Tab, Edit Dialog
• **Command**: the command that carries out the script, or the full path to the executable script.

• **Command timeout**: After how many seconds the script should stop running, in case of no response.

• **Parameter**: a value passed to the script, with the following conditions as a possible setting:
  - Optional (default)
  - Disallowed
  - Required

• **Cumulative**: whether the metric value is cumulative (for example, based on the bytes-received counter for an Ethernet interface, which accumulates over time), or non-cumulative (for example, based on the temperature, which does not accumulate over time). The value for a cumulative metric is the difference between the current measurement and the preceding measurement, divided by the time interval between the two measurements. The value of the `BytesRecv` metric (a cumulative metric) is thus in bytes/second and ideally indicates the number of bytes per second received through the interface at that time, although it is merely a simple interpolation. Similarly, the value of the `Uptime` metric (a cumulative metric) is thus ideally a unitless 1 with this definition.

• **Unit**: the unit in which the measurement associated with the metric is done. For a cumulative metric, the unit is applied to the two measurements that occurred over the time that the metric was monitored. So the bytes-received measurement has units of bytes, in contrast to the `BytesRecv` metric, which has units of bytes/second. Similarly, the uptime measurement has units of seconds, in contrast to the `Uptime` metric, which is unitless.

• **When to run**:
  - **Disabled**: if ticked, the metric script does not run.
  - **Only when idle**: if ticked, the metric script only runs when the system is idling. This burdens the system less if the metric is resource-hungry.

• **Sampling Method**: the options are:
  - **Sampling on master**: The head node samples the metric on behalf of a device. For example: the head node may do this for a PDU, since a PDU does not have the capability to run the cluster management daemon at present, and so cannot itself pass on data values directly when `cmsh` or `cmgui` need them. Similarly, the IPMI health check—not classed as a metric, but classed as a health check, and using the same kind of dialog (section 10.4.5) as here—is also sampled via the head node, since IPMI diagnostics run on the node even when the node is down.
  - **Sampling on node**: The non-head node samples the metric itself. The administrator should ensure that the script is accessible from the non-head node.

• **Class**: An option selected from:
  - CPU
  - GPU
  - Disk
  - Memory
  - Network
  - Environmental
  - Operating System
Figure 10.32: cmgui Monitoring: Metrics Tab, Add collection Dialog

- Internal
- Workload
- Cluster
- Amazon EC2
- Cloud
- Misc
- several Hadoop components, applicable if the cluster is managing Big Data
- several OpenStack components, applicable if the cluster is managing OpenStack
- Dell NSS
- Ceph
- Ceph
- Lustre
- Prototype

• Retrieval Method:
  - cmdaemon: Metrics retrieved internally using CMDaemon (default).
  - snmp: Metrics retrieved internally using SNMP.

• State flapping count (default value 7): How many times the metric value must cross a threshold within the last 12 samples (a default setting, set in cmd.conf) before it is decided that it is in a flapping state.

• Absolute range: The range of values that the metric takes. A range of 0–0 implies no constraint is imposed.

• Notes: Notes can be made here.

Metrics Tab: Add Collection Option

The Add Collection button opens a dialog which is used to create a metric collection (figure 10.32). A metric collection is a special metric script, with the following properties:

• It is able to return several metrics of different types when it is run, not just one metric of one type like a normal metric script does—hence the name, “metric collection”.

© Bright Computing, Inc.
10.4 Monitoring Configuration With cmgui

- It autodetects if its associated metrics are able to run, and to what extent, and presents the metrics accordingly. For example, if the metric collection is run on a node which only has 3 CPUs running rather than a default of 4, it detects that and presents the results for just the 3 CPUs.

Further details on metric collections scripts are given in Chapter 2 of the Developer Manual.

Because handling metric collections is just a special case of handling a metric, the Add Collection button dialog is actually just a restricted version of the Add button dialog. Setting up a metric collection is therefore simplified by having most of the metric fields pre-filled and kept hidden. For example, the Class field for a metric collection would have the value Prototype in the Add button dialog, while this value is pre-filled and invisible in the Add Collection dialog. A metric collection can be created with the Add dialog, but it would be a little more laborious.

Whatever the method used to create the metric collection, it can always be edited with the Edit button, just like any other metric.

Viewing visualizations of a metric collection in cmgui is only possible through selection and viewing the separate graphs of its component metrics.

The device category resource on which the metric can be run can be chosen from the following:

- Head nodes
- Ethernet switches
- Myrinet switches
- InfiniBand switches
- Power distribution units
- Rack sensors
- Generic devices
- Chassis
- GPU Units
- default
- vnodes

In cmsh, the collections script can be associated with the resource within the monitoring setup mode (section 10.7.4).

For many metrics collections run on a node, only the checkboxes for Node metric and Master Node metric are set, in which case the collection is valid for both head node and regular nodes.

10.4.5 Health Checks Tab

The Health Checks tab lists available health checks (figure 10.33). These can be set to run from the system by configuring them from the Health Check Configuration tab of section 10.4.3.

What the listed health checks on a newly installed system do are described in Appendix G.2.1.

The remove, revert and save buttons work for health checks just like they do for metrics in section 10.4.4.

Also, the edit and add buttons start up dialogs to edit and add health checks. The dialog options for health checks are the same as for editing or adding metrics, with a few exceptions. The exceptions are for options that are inapplicable for health checks, and are elaborated on in Appendix G.2.2.
10.4.6 Actions Tab

The Actions tab lists available actions (figure 10.34) that can be set to run on the system from metrics thresholds configuration, as explained in section 10.4.2, and as was done in the basic example of section 10.1. Actions can also be set to run from health check configuration action launcher options as described in section 10.4.3.

What the listed actions on a newly installed system do are described in Appendix G.3.1. The remove, revert, and save buttons work as described for metrics in section 10.4.4.

The edit and add buttons start up dialogs to edit or add options to action parameters. Action parameters are described in Appendix G.3.2.

10.5 Overview Of Monitoring Data For Devices

These views are set up under the Overview tab for various devices that are items under the resource tree in the cluster.

They are a miscellany of monitoring views based on the monitored data for a particular device. The views are laid out as part of an overview tab for that device, which can be a switch, cluster, node, GPU unit, and so on.

When first connecting to a cluster with cmgui, the Overview tab of the cluster is the default view. The Overview tab is also the default view first time a device is clicked on in a cmgui session.

Of the devices, the cluster(s), head node(s) and regular nodes have a relatively extensive Overview tab, with a pre-selected mix of information from monitored data. For example, in figure 10.4, a head node is shown with an Overview tab presenting memory used, CPU usage, disk usage, network statistics, running processes, and health status. Some of these values are presented with colors and histograms to make the information easier to see.
10.6 Event Viewer

This is a view of events on the cluster(s). It is accessible from the View menu of the main window of cmgui. By default, there is no log file, but a log file can be activated with the EventLogger directive (Appendix C).

The events can be handled and viewed in several ways.

10.6.1 Viewing Events In cmgui

Double clicking on an event row starts up an Event Details dialog (figure 10.35), with buttons to:

- Acknowledge or Unacknowledge the event, as appropriate. Clicking on Acknowledge removes the event from the event view unless the Show Acknowledged checkbox has been checked. Any visible acknowledged events have their acknowledged status removed when the Unacknowledge button is clicked.

- Report to cluster vendor. The report option is used for sending an e-mail about the selected event to the cluster vendor in case troubleshooting and support is needed.
The event viewer toolbar (figure 10.35) offers icons to handle events:

- **detach event viewer**: Detaches the event viewer pane into its own window. Reattachment is done by clicking on the reattachment event viewer icon that becomes available in the detached window.

- **new event viewer filter dialog**: Loads or defines filters (figure 10.36). Filters can be customized according to acknowledgement status, time periods, cluster, nodes, severity (section 10.2.6), or message text. The filter settings can be saved for later reloading. If the dialog opened by this button is simply given an arbitrary name, and the Ok button clicked on to accept the default values, then a default event viewer tabbed pane is added to cmgui.

![Figure 10.36: cmgui Monitoring: Event Viewer Filter Dialog](image)

- **set event viewer filter dialog**: Adjusts an existing filter with a similar dialog to the new event viewer filter dialog.

- **acknowledge event**: Sets the status of one or more selected events in the display pane to an acknowledged state. To set all events in the pane to an acknowledged state, the button is shift-clicked. Acknowledged events are no longer seen, unless the filter setting for the show acknowledged checkbox is checked in the set event filter option.

### 10.6.2 Viewing Events In cmsh

The `events` command allows events to be viewed at several severity levels (section 10.2.6). The usage and synopsis of the `events` command is:

**Usage:**
```
events
events on [broadcast|private]
events off [broadcast|private]
events level <level>
```
10.6 Event Viewer

```
    events clear
    events details <id>
    events <number> [level]
    events follow
```

Without arguments:

Displays settings of events command for:

- private (whether on or off)
- broadcast (whether on or off)
- level (whether info, notice, warning, error, or alert)

Also shows events since last time the command was run.

With arguments:

```
on [broadcast|private]      allow event messages
off [broadcast|private]     block event messages
level <info|notice|warning|error|alert> display events at that level and above
clear  clear logged event details
details <id> show event detail for that <id>
<number> [info|notice|warning|error|alert> show <number> events for [level]
follow  follow event messages as they come in
         (ctrl-c to interrupt)
```

Running the command without any option shows event settings, and displays any event messages that have not been displayed yet in the session:

**Example**

```
[bright73->device]% events
Private events: off
Broadcast events: on
Level: notice
custom .........................[  RESET  ] node001
```

Running the command with options allows the viewing and setting of events as follows:

- **on [broadcast|private]**: event messages are displayed as they happen in a session, with `cmsh` prompts showing in between messages:
  
  - If *only* `on` is set, then all event messages are displayed as they happen:
    
    - either to all open `cmsh` sessions, and also all `cmgui` 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 all `cmgui` 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.

© Bright Computing, Inc.
• 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:
  
  ```bash
  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**

```
[bright73->softwareimage]% events on private
Private events: on
[bright73->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[bright73->softwareimage]% Tue Apr 29 01:19:12 2014 [notice] bright73: Provisioning started: sendi...
[bright73->softwareimage]% Tue Apr 29 01:19:52 2014 [notice] bright73: Provisioning completed: sen...
updateprovisioners [ COMPLETED ]
[bright73->softwareimage] % !#events were indeed seen in cmsh session
[bright73->softwareimage] % !#now block the events and rerun update:
[bright73->softwareimage] % events off private
Private events: off
[bright73->softwareimage] % updateprovisioners
Provisioning nodes will be updated in the background.
[bright73->softwareimage] % !#let this 2nd update run for a while
[bright73->softwareimage] % !#(time passes)
[bright73->softwareimage] % !#nothing seen in cmsh session.
[bright73->softwareimage] % !#show a 2nd update did happen:
[bright73->softwareimage] % events 4 | grep -i provisioning
Tue Apr 29 01:19:12 2014 [notice] bright73: Provisioning started: sendi...
```

© Bright Computing, Inc.
10.6 Event Viewer

Tue Apr 29 01:19:52 2014 [notice] bright73: Provisioning completed: sen...
Tue Apr 29 01:25:37 2014 [notice] bright73: Provisioning started: sendi...
Tue Apr 29 01:26:01 2014 [notice] bright73: Provisioning completed: sen...

10.6.3 Using The Event Bucket From The Shell For Events And For Tagging Device States

Event Bucket Default Behavior
The Bright 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@bright73 ~]# echo "Some text" > /var/spool/cmd/eventbucket

This adds an event with, by default, the info severity level, to the event pane.

Event Bucket Severity Levels
To write events at specific severity levels (section 10.2.6), 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:
EVENT_SEVERITY_EMERGENCY:

Example

echo "EVENT_SEVERITY_ERROR:An error line" > /var/spool/cmd/eventbucket

The preceding example displays an output in the cmgui event viewer like in figure 10.37:

![Figure 10.37: cmgui Monitoring: Event Bucket Message Example](image)

Event Bucket Filter
Regex expressions can be used to conveniently filter out the user-defined messages that are about to go into the event bucket from the shell. The filters used are placed in the event bucket filter, located by default at /cm/local/apps/cmd/etc/eventbucket.filter.

© Bright Computing, Inc.
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:

```
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, `bright73`, 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:

```
bright73 .................. (too hot) [ UP ]
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
```

10.7  The monitoring Modes Of cmsh

This section covers how to use cmsh to configure monitoring. The monitoring mode in cmsh is how metrics and health checks are configured from the command line, and corresponds to the configuration carried out by cmgui in section 10.4.

Visualization of data similar to how cmgui does it in section 10.3 can also be done from cmsh’s command line, via its device mode. Graphs can be obtained from cmsh by piping values returned by device mode commands such as `latestmetricdata` (section 10.8.3) and `dumpmetricdata` (section 10.8.6) into graphing utilities. These techniques are not covered in this chapter.

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 these objects—the details of the monitoring settings—are the parameters and values which are accessed and manipulated from the monitoring mode hierarchy within cmsh.

The monitoring “mode” of cmsh gives access to 4 modes under it.

Example
These 4 modes are regarded as being the top level monitoring related modes:

- monitoring actions
- monitoring healthchecks
- monitoring metrics
- monitoring setup

The word monitoring is therefore merely a grouping label prefixed inseparably to these 4 modes. The syntax of the 4 bulleted commands above is thus consistent with that of the other top level cmsh modes.

The sections 10.7.1, 10.7.2, 10.7.3, and 10.7.4 give examples of how objects are handled under these 4 monitoring modes. To avoid repeating similar descriptions, section 10.7.1 is relatively detailed, and is often referred to by the other sections.

10.7.1 The monitoring actions Mode In cmsh

The monitoring actions mode of cmsh corresponds to the cmgui actions tab of section 10.4.6.

The monitoring actions mode handles actions 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.

This section continues the cmsh session started above, giving examples of how the monitoring actions mode is used.

The monitoring actions Mode In cmsh: list, show, And get

The list command by default lists the names and command scripts available in monitoring actions mode:

Example

```
[myheadnode]# monitoring actions
[myheadnode->monitoring->actions]# list
Name (key)            Command
------------------------ ----------------------------------------
Drain node             <built-in>
Power off              <built-in>
Power on               <built-in>
Power reset            <built-in>
Reboot                 <built-in>
SendEmail              <built-in>
Shutdown               <built-in>
Undrain node           <built-in>
killprocess            /cm/local/apps/cmd/scripts/actions/killprocess.+
remount                /cm/local/apps/cmd/scripts/actions/remount
testaction             /cm/local/apps/cmd/scripts/actions/testaction
```
The above shows the actions available on a newly installed system. The details of what they do are covered in Appendix G.3.1.

The `show` command of `cmsh` displays the parameters and values of a specified action:

**Example**

```
[myheadnode->monitoring->actions]% show poweroff
Parameter Value
-------------------------------------------- --------------------------------------------
Command <built-in>
Description Power off the device
Name Power off
Revision
Run on master
Timeout 5
isCustom no
```

The meanings of the parameters are covered in Appendix G.3.2.

Tab-completion suggestions with the `show` command suggest arguments corresponding to names of action objects:

**Example**

```
[myheadnode->monitoring->actions]% show
```

A double-tap on the tab key to get tab-completion suggestions for `show` in the preceding displays the following:

**Example**

```
drainnode killprocess poweron reboot sendemail testaction
killallyes poweroff powerreset remount shutdown undrainnode
```

The `Power off` action name, for example, corresponds to the argument `poweroff`. By default, the arguments are the action names in lower case, with the spaces removed. However, they are space- and case-insensitive, so typing in `show "Power off"` with the quotes included to pass the space on is also valid.

The `get` command returns the value of an individual parameter of the action object:

**Example**

```
[myheadnode->monitoring->actions]% get poweroff runon master
[myheadnode->monitoring->actions]%
```

The monitoring actions Mode in cmsh: add, use, remove, commit, refresh, modified, set, clear, And validate

In the basic example of section 10.1, in “Adding The Action To The Actions List”, the name, description and command for an action were added via a dialog in the Actions tab of cmgui.

The equivalent is done in cmsh with `add` and `set`. 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.

If there is no `killallyes` action already, then the name is added in the actions mode with the `add` command as follows:
Example

[myheadnode->monitoring->actions]\% add killallyes
[myheadnode->monitoring->actions*[killallyes*]]\%

The converse to the add command is the remove command, which removes the action.

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 parameter displayed by a show command:

Example

[myheadnode->monitoring->actions*[killallyes*]]\% set description "kill all yes processes"

The clear command is the converse of set, and removes any value for a given parameter.

Example

[myheadnode->monitoring->actions*[killallyes*]]\% clear command

The validate command checks if the object has all required values set to sensible values. The commands refresh, modified and commit work as expected from the introduction to working with objects (section 2.5.3). So, for example, commit only succeeds if the killallyes object passes validation.

Example

[myheadnode->monitoring->actions*[killallyes*]]\% validate

<table>
<thead>
<tr>
<th>Code</th>
<th>Field</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>command</td>
<td>command should not be empty</td>
</tr>
</tbody>
</table>

Here validation fails because the parameter Command has no value set for it yet. This is remedied with set acting on the parameter (some prompt text elided for display purposes):

Example

[...\% set command "/cm/local/apps/cmd/scripts/actions/killallyes"
[...\% commit
[...\%

Validation then succeeds and the commit successfully saves the killallyes object.

Note that validation does not check if the script itself exists. It solely does a sanity check on the values of the parameters of the object, which is another issue. If the killallyes script does not yet exist in the location given by the parameter, it can be created as suggested in the basic example of section 10.1, in “Setting Up The Kill Action”.

10.7.2 The monitoring healthchecks Mode in cmsh

The monitoring healthchecks mode of cmsh corresponds to the cmgui Health Checks tab of section 10.4.5.

The monitoring healthchecks mode handles health check objects in the way described in the introduction to working with objects (section 2.5.3). A typical reason to handle health check objects—the properties associated with an health check script or health check built-in—might be to view the health checks already available, or to add a health check for use by a device resource.

This section goes through a cmsh session giving some examples of how this mode is used and to illustrate what it looks like.
The monitoring healthchecks Mode in cmsh: list, show, And get

In monitoring healthchecks mode, the list command by default lists the names of the health check objects along with their command scripts:

Example

[bright73->monitoring->healthchecks]% format name:18 command:55
[bright73->monitoring->healthchecks]% list

<table>
<thead>
<tr>
<th>name (key)</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceIsUp</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>ManagedServicesOk</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>chrootprocess</td>
<td>/cm/local/apps/cmd/scripts/healthchecks/chrootprocess</td>
</tr>
<tr>
<td>cmsh</td>
<td>/cm/local/apps/cmd/scripts/healthchecks/cmsh</td>
</tr>
<tr>
<td>diskspace</td>
<td>/cm/local/apps/cmd/scripts/healthchecks/diskspace</td>
</tr>
</tbody>
</table>

... The format command, introduced in section 2.5.3, is used here with the given column width values to avoid truncating the full path of the commands in the display.

The above example shows a truncated list of health checks that can be set for sampling on a newly installed system. The details of what these health checks do is covered in Appendix G.2.1.

The show command of cmsh displays the parameters and values of a specified health check:

Example

[myheadnode->monitoring->healthchecks]% show deviceisup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class of healthcheck</td>
<td>internal</td>
</tr>
<tr>
<td>Command</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>Returns PASS when device is up, closed or insta+</td>
</tr>
<tr>
<td>Disabled</td>
<td>no</td>
</tr>
<tr>
<td>Extended environment</td>
<td>no</td>
</tr>
<tr>
<td>Name</td>
<td>DeviceIsUp</td>
</tr>
<tr>
<td>Notes</td>
<td>&lt;0 bytes</td>
</tr>
<tr>
<td>Only when idle</td>
<td>no</td>
</tr>
<tr>
<td>Parameter permissions</td>
<td>disallowed</td>
</tr>
<tr>
<td>Required resources</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Sampling method</td>
<td>samplingonmaster</td>
</tr>
<tr>
<td>State flapping count</td>
<td>7</td>
</tr>
<tr>
<td>Timeout</td>
<td>5</td>
</tr>
</tbody>
</table>

[myheadnode->monitoring->healthchecks]%

The meanings of the parameters are covered in Appendix G.2.2.

As detailed in section 10.7.1, tab-completion suggestions for the show command suggest arguments corresponding to names of objects that can be used in this mode. For show in healthchecks mode, tab-completion suggestions give the following as possible health check objects:

Example

[myheadnode->monitoring->healthchecks]% show

chrootprocess failover mysql ssh2node
cmsh hardware-profile ntp swraid
deviceisup interfaces portchecker testhealthcheck
diskspace ldap rogueprocess
The get command returns the value of an individual parameter of a particular health check object:

Example

```
[myheadnode->monitoring->healthchecks]% get deviceisup description
Returns PASS when device is up, closed or installing
```

The monitoring healthchecks Mode in cmsh: add, use, remove, commit, refresh, modified, set, clear, and validate

The remaining commands in monitoring healthchecks mode: add, 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 Cmsh Monitoring Actions (section 10.7.1).

In the basic example of section 10.1, a metric script was set up from cmgui to check if thresholds were exceeded, 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 cmgui in the basic example. A start is made on the task by creating a health check object and setting its values using the monitoring healthchecks mode of cmsh. The task is completed in the section on the monitoring setup mode in section 10.7.4.

To start the task, cmsh's add command is used to create the new health check object:

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% monitoring healthchecks
[myheadnode->monitoring->healthchecks]% add cpucheck
[myheadnode->monitoring->healthchecks*[cpucheck*]]%
```

The set command sets the value of each parameter displayed by a show command (some prompt text elided for layout purposes):

Example

```
[...]% set command /cm/local/apps/cmd/scripts/healthchecks/cpucheck
[...]% set description "CPUuser under 50%?"
[...]% set parameterpermissions disallowed
[...]% set samplingmethod samplingonmaster
[...]% commit
```

Since the cpucheck script does not yet exist in the location given by the parameter command, it needs to be created:

```
#!/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
```

© Bright Computing, Inc.
if ($(bc <<< "$comparisonstring") )); then
   echo PASS
else
   echo FAIL
fi

The script should be placed in the location suggested by the object, /cm/local/apps/cmd/scripts/healthchecks/cpucheck, and made executable with a chmod 700.

The cpucheck object is handled further within the cmsh monitoring setup mode in section 10.7.4 to produce a fully configured health check.

10.7.3 The monitoring metrics Mode In cmsh

The monitoring metrics mode of cmsh corresponds to the cmgui metrics tab of section 10.4.4.

The monitoring metrics mode of cmsh handles metrics objects in the way described in the introduction to working with objects (section 2.5.3). A typical reason to handle metrics objects—the properties associated with a metrics script or metrics built-in—might be to view the configuration metrics already being used for sampling by a device category, or to add a metric for use by a device category.

This section goes through a cmsh session giving some examples of how this mode is used and to illustrate its behavior.

The monitoring metrics Mode In cmsh: list, show, And get

In metrics mode, the list command by default lists the names and command scripts available for setting for device categories:

Example

[root@myheadnode ~]# cmsh
[myheadnode]$ monitoring metrics
[myheadnode->monitoring->metrics]$ list

<table>
<thead>
<tr>
<th>Name (key)</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>AvgExpFactor</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

The above shows a truncated list of the metrics that may be used for sampling on a newly installed system. What these metrics do is described in Appendix G.1.1.

The show command of cmsh displays the parameters and values of a specified metric:

Example

[myheadnode->monitoring->metrics]$ show cpuuser

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class of metric</td>
<td>cpu</td>
</tr>
<tr>
<td>Command</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>Cumulative</td>
<td>yes</td>
</tr>
<tr>
<td>Description</td>
<td>Percent of node-wide core time spent in user mode</td>
</tr>
<tr>
<td>Disabled</td>
<td>no</td>
</tr>
<tr>
<td>Extended environment</td>
<td>no</td>
</tr>
<tr>
<td>Maximum</td>
<td>&lt;range not set&gt;</td>
</tr>
<tr>
<td>Metric Unit</td>
<td>%</td>
</tr>
<tr>
<td>Minimum</td>
<td>&lt;range not set&gt;</td>
</tr>
</tbody>
</table>
The meanings of the parameters above are explained in Appendix G.1.2.

Tab-completion suggestions for the show command suggest arguments corresponding to names of objects (the names returned by the list command) that may be used in a monitoring mode. For metrics mode, show, followed by a double-tap on the tab key, displays a large number of possible metrics objects:

Example

[myheadnode->monitoring->metrics]% show
Display all 130 possibilities? (y or n)
alertlevel
droprecv
ipoutrequests
avgexpfactor
dropsent
ipreasmoks
avgjobduration
errossrecv
ipreasmreqds
await_sda
errosssent
loadfifteen...

The get command returns the value of an individual parameter of a particular metric object:

Example

[myheadnode->monitoring->metrics]% get CPUUser description
Percent of node-wide core time spent in user mode

The monitoring metrics Mode In cmsh: add, use, remove, commit, refresh, modified, set, clear, and validate.
The remaining commands in monitoring metrics mode: add, 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 Cmsh Monitoring Actions (section 10.7.1).

Adding a metric collections script to the framework is possible from this point in cmsh too. Details on how to do this are given in Chapter 2 of the Developer Manual.

10.7.4 The monitoring setup Mode In cmsh

The cmsh monitoring setup mode corresponds to the cmgui Metric Configuration and Health Check Configuration tabs of sections 10.4.2 and 10.4.3.

The monitoring setup mode of cmsh, like the Metric Configuration and the Health Check Configuration tabs of cmgui, is used to select a device category. Properties of metrics or of health checks can then be configured for the selected device category. These properties are the configuration of the sampling parameters themselves (for example, frequency and length of logs), but also the configuration of related properties such as thresholds, consolidation, actions launched when a metric threshold is crossed, and actions launched when a metric or health state is flapping.

The setup mode only functions in the context of metrics or health checks, and therefore these contexts under the setup mode are called submodes. On a newly installed system, a list command from
the monitoring setup prompt displays an account of metrics and health checks that are in use by device categories:

Example

[root@myheadnode ~]# cmsh
[myheadnode]# monitoring setup
[myheadnode->monitoring->setup]# list

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric configuration</th>
<th>Health configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chassis</td>
<td>2 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>EthernetSwitch</td>
<td>13 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>GenericDevice</td>
<td>2 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>GpuUnit</td>
<td>3 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>HeadNode</td>
<td>104 in submode</td>
<td>24 in submode</td>
</tr>
<tr>
<td>IBSwitch</td>
<td>2 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>MIC</td>
<td>3 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>MyrinetSwitch</td>
<td>2 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>myheadnode</td>
<td>575 in submode</td>
<td>19 in submode</td>
</tr>
<tr>
<td>PowerDistributionUnit</td>
<td>5 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>RackSensor</td>
<td>2 in submode</td>
<td>1 in submode</td>
</tr>
<tr>
<td>default</td>
<td>40 in submode</td>
<td>17 in submode</td>
</tr>
<tr>
<td>node001</td>
<td>30 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>node002</td>
<td>30 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>node003</td>
<td>38 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>node004</td>
<td>30 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>switch01</td>
<td>0 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>vnode001</td>
<td>14 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>vnode002</td>
<td>0 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>vnode003</td>
<td>0 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>vnode004</td>
<td>0 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>vnode005</td>
<td>0 in submode</td>
<td>0 in submode</td>
</tr>
<tr>
<td>vnodes</td>
<td>38 in submode</td>
<td>17 in submode</td>
</tr>
</tbody>
</table>

A device category must always be used when handling the properties of the metrics and health checks configurable under the submodes of monitoring setup. The syntax of a configuration submode, metricconf or healthconf, therefore requires the device category as a mandatory argument, and tab-completion suggestions become quite helpful at this point.

Examples are now given of how the metric configuration metricconf and health check configuration healthconf submodes are used:

The monitoring setup Mode in cmsh: metricconf

Continuing with the session above, the metricconf option can only be used with a device category specified. Tab-completion suggestions for metricconf suggest the following possible device categories:

Example

[myheadnode->monitoring->setup]# metricconf

chassis ibswitch node004 vnode002
default mic myheadnode vnode003
ethernetswitch myrinetswitch powerdistributionunit vnode004
genericdevice node001 racksensor vnode005
gpuunit node002 switch01 vnodes
headnode node003 vnode001
A category can be chosen with the `use` command, and `show` shows the properties of the category. With a category selected, the `metricconf` or `healthconf` submodes can then be invoked:

**Example**

```
[myheadnode->monitoring->setup]% use headnode
[myheadnode->monitoring->setup[HeadNode]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>HeadNode</td>
</tr>
<tr>
<td>Health configuration</td>
<td>&lt;14 in submode&gt;</td>
</tr>
<tr>
<td>Metric configuration</td>
<td>&lt;90 in submode&gt;</td>
</tr>
<tr>
<td>Normal pickup interval</td>
<td>180</td>
</tr>
<tr>
<td>Scrutiny pickup interval</td>
<td>60</td>
</tr>
</tbody>
</table>

```
[myheadnode->monitoring->setup[HeadNode]]% metricconf
```

As an aside, data values are picked up by CMDaemon on the head node from the devices that run metrics or health checks at certain pickup intervals. The interval value used for the pickups is set with these category-level parameters:

- **Normal pickup interval**: used when all metrics are below threshold levels, and all health checks succeed.
- **Scrutiny pickup interval**: used when at least one of the following is true:
  - a metric is above a threshold level
  - a health check is in a failed state

The idea behind this is that the more frequent pickup lets the administrator get up-to-date values sooner during abnormal conditions. Increasing the value of these intervals reduces the amount of data values picked up over time for the category. To stop data values being picked up completely, a category, or other node grouping, can be set to a `CLOSED` state (section 5.5.4).

Dropping into a submode—in the example given, the `metricconf` submode—could also have been done directly in one command: `metricconf headnode`. The synopsis of the command in the example is actually `[[[monitoring] setup] metricconf] headnode`, where the optional parts of the command are invoked depending upon the context indicated by the prompt. The example below clarifies this (some prompt text elided for display purposes):

**Example**

```
[...->monitoring->setup[HeadNode]->metricconf]% exit; exit; exit
[...]% monitoring setup metricconf headnode
[...->monitoring->setup[HeadNode]->metricconf]% exit; exit
[...->monitoring]% setup metricconf headnode
[...->monitoring->setup[HeadNode]->metricconf]% exit; exit
[...->monitoring->setup]% metricconf headnode
[...->monitoring->setup[HeadNode]->metricconf]% exit
[...->monitoring->setup[HeadNode]]% metricconf
[...->monitoring->setup[HeadNode]->metricconf]%
```

A list of metrics that have been set to do sampling for the device category `headnode` is obtained with `list`. Since there are many of these, only 10 lines are displayed in the list shown below by piping it through `head`:

**Example**

```
```
Besides `list`, an alternative way to get a list of metrics that are set to sample for `headnode` is to use the tab-completion suggestions to the `use` command.

The `use` command is normally used to drop into the configuration properties of the metric so that parameters of the metric object can be configured:

**Example**

```
[...metricconf]%
```

The `add` command adds a metric to be set for sampling for the device category. The list of all possible metrics that can be added to the device category can be seen with the command `monitoring metrics list`, or more conveniently, simply with tab-completion suggestions to the `add` command at the `[...metricconf]%` prompt in the above example.

The above example indicates that there are two submodes for each metric configuration: Consolidators and Thresholds. Running the `consolidators` or `thresholds` commands brings cmsh into the chosen submode.

Consolidation and threshold manipulation only make sense in the context of a metric configuration, so at the `metricconf` prompt in the example above (before `use cpuuser` is executed), the commands `thresholds cpuuser` or `consolidators cpuuser` can be executed as more direct ways of getting to the chosen submode.

**The thresholds submode** If, continuing on from the above example, the thresholds submode is entered, then the `list` command lists the existing thresholds. If the basic example of section 10.1 has already been carried out on the system, then a threshold called `killallyesthreshold` is already there with an assigned action `killallyes`. The properties of each threshold can be shown (some prompt text elided for layout purposes):
10.7 The monitoring Modes Of cmsh

Example

```
[...metricconf]% thresholds
[...metricconf[CPUUser]]% thresholds
[...metricconf[CPUUser]]->thresholds]% list
Name (key) Bound  Severity
---------------------------------  ---------  ------------
killallyesthreshold  50  10
```

The meanings of the parameters are explained in the GUI equivalent of the above example in section 10.4.2 in the section labeled “Metric Configuration: Thresholds Options”. The object manipulation commands introduced in section 2.5.3 work as expected at this cmsh prompt level: add and remove add and remove a threshold; set, get, and clear set and get values for the parameters of each threshold; refresh and commit revert and commit changes; use “uses” the specified threshold, making it the default for commands; validate applied to the threshold checks if the threshold object has sensible values; and append and removefrom append an action to, and remove an action from, a specified threshold.

The append and removefrom commands correspond to the ⨂ and ⨂ widgets of cmgui in figure 10.25 and work with parameters that can have multiple values. For example, a sendemail action with a parameter root can be appended to the Actions parameter, which already has the killallyes action as a value. This sends an e-mail to the root mail account. A get command can be run to see the values for the threshold actions:

Example

```
[...->thresholds*]% append killallyesthreshold actions sendemail root
[...->thresholds*]% get killallyesthreshold actions
enter: killallyes()
enter: SendEmail(root)
```

When using the sendemail() command, the e-mail is sent to the parameter value. If no parameter is specified, then the e-mail is sent to root by default, unless a value for administratore-mail has been set for the cluster as described in the text accompanying figure 3.1. If multiple e-mail addresses are used, then they must be space-separated from each other.

The actions command takes flags that define when the action is taken on crossing the threshold. These are:

- `-e|--enter`: Run on entering the zone. This is also the implied default when the flag is omitted.
- `-l|--leave`: Run on leaving the threshold zone.
- `-d|--during`: Run during the time the value is within the threshold zone.

In the example, the “Actions” parameter now has the value of the built-in action name, sendemail, as well as the value of the action script name, killallyes. This means that both actions run when the threshold condition is met.

© Bright Computing, Inc.
The **consolidators submode** If, continuing on with the preceding example, the consolidators submode is entered, then the `list` command lists the consolidators running on the system. On a newly installed system there are three consolidators by default for each metric set for a device category. Each consolidator has an appropriately assigned time `Interval`, in seconds. The `show` command shows the parameters and values of a specific consolidator:

**Example**

```
[...metricconf[CPUUser]->thresholds*]% exit
[...metricconf[CPUUser]% consolidators
[...metricconf[CPUUser]->consolidators]% list
Name (key) Length Interval
------------------------ ---------- ----------
Day 1000 86400
Hour 2000 3600
Week 1000 604800
[...metricconf[CPUUser]->consolidators]% show day
Parameter Value
------------------------------ -------------------------------
Interval 86400
Kind AVERAGE
Length 1000
Name Day
Offset 0
Revision
```

The meanings of the parameters are explained in the GUI equivalent of the above example in section 10.4.2 in the section labeled “Metric Configuration: Consolidators Options”.

The object manipulation commands introduced in section 2.5.3 work as expected at this `cmsh` prompt level: add and remove add and remove a consolidator; `set`, `get`, and `clear` set and get values for the parameters of each consolidator; `refresh` and `commit` revert and commit changes; use “uses” the specified consolidator, making it the default for commands; and `validate` applied to the consolidator checks if the consolidator object has sensible values.

The **monitoring setup Mode in cmsh: healthconf**

The `healthconf` submode is the alternative to the `metricconf` submode under the main `monitoring setup` mode. Like the `metricconf` option, `healthconf` too can only be used with a device category specified.

If the session above is continued, and the device category `headnode` is kept unchanged, then the `healthconf` submode can be invoked with:

```
[...metricconf[CPUUser]->consolidators]% exit; exit; exit
[myheadnode->monitoring->setup[HeadNode]]% healthconf
[...healthconf]%
```

Alternatively, the `healthconf` submode with the `headnode` device category could also have been reached from `cmsh`’s top level prompt by executing `monitoring setup healthconf headnode`.

The health checks set to do sampling in the device category `headnode` are listed:

**Example**

```
[myheadnode->monitoring->setup[HeadNode]->healthconf]% list
HealthCheck HealthCheck Param Check Interval
-----------------------------------------------
```
The use command would normally be used to drop into the health check object. However use can also be an alternative to the list command, since tab-completion suggestions to the use command get a list of currently configured health checks for the headnode too.

The add command adds a health check into the device category. The list of all possible health checks that can be added to the category can be seen with the command monitoring healthchecks list, or more conveniently, simply with tab-completion suggestions to the add command.

At the end of section 10.7.2 a script called cpucheck was built. This script was part of a task to use health checks instead of metric threshold actions to set up the functional equivalent of the behavior of the basic example of section 10.1. In this section the task is continued and completed, and on the way how to use the health checks configuration object methods to do this is shown.

First, the script is added, and as usual when using add, the prompt drops into the level of the added object. The show command acting on the object displays the following default values for its parameters (some prompt text elided for display purposes):

**Example**

```
[...[HeadNode]->healthconf]% add cpucheck
[...*[HeadNode*]->healthconf*[cpucheck*]]% show
Parameter       Value
----------------- -------------------------------
Check interval   120
Check interval offset 0
Disabled no
Fail Actions
Fail severity 10
GapThreshold 2
HealthCheck cpucheck
HealthCheckParam
LogLength 3000
Only when idle no
Pass Actions
Revision
Stateflapping Actions
Store yes
ThresholdDuration 1
Unknown Actions
Unknown severity 10
[...*[HeadNode*]->healthconf*[cpucheck*]]%
```
The details of what these parameters mean is covered in section 10.4.3 where the edit and add dialog options for a health check state shown in figure 10.29 are explained.

The object manipulation commands introduced in section 2.5.3 work as expected at the `healthconf` prompt level in the example above: `add` and `remove` add and remove a health check; `set`, `get`, and `clear` set and get values for the parameters of each health check; `refresh` and `commit` revert and commit changes; use “uses” the specified health check, making it the default for commands; and `validate` applied to the health check checks if the health check object has sensible values; and `append` and `removefrom` append an action to, and remove an action from, a specified health check action parameter.

The `append` and `removefrom` commands correspond to the ⍺ and ⍵ widgets of cmgui in figure 10.29 and work with parameters that can have multiple values.

The action `killallyes` was set up to be carried out with the metric `CPUUser` in the basic example of section 10.1. The action can also be carried out with a `FAIL` response for the `cpucheck` health check by using `append` command:

**Example**

```
 [...]healthconf*[cpucheck*] ]% append failactions killallyes
 [...]healthconf*[cpucheck*] ]% 
```

Sending an e-mail to root can be done by appending further:

**Example**

```
 [...]healthconf*[cpucheck*] ]% append failactions sendemail root
 [...]healthconf*[cpucheck*] ]% get failactions
 enter: SendEmail(root)
 enter: killallyes()
 [...]healthconf*[cpucheck*] ]% 
```

### 10.8 Obtaining Monitoring Data Values

The monitoring data values that are logged by devices can be used to generate graphs using the methods in section 10.3. However, sometimes an administrator would like to have the data values that generate the graphs instead, perhaps to import them into a spreadsheet for further direct manipulation, or to pipe them into a utility such as `gnuplot`.

#### 10.8.1 The metrics and healthchecks Commands

The types of monitoring data values that can be obtained can be seen with the `metrics` or `healthchecks` commands.

- **metrics**: The `metrics` command in `device` mode lists the metrics that are currently configured to be monitored for a specified device. These correspond to the metrics shown in cmgui in the “Metric Configuration” tab for a specific device.

- **healthchecks**: The `healthchecks` command in `device` mode lists the health checks that are currently configured to be monitored for a device. These correspond to the health checks shown in cmgui in the “Health Check Configuration” tab for a device.

**Using The metrics And healthchecks Commands**

When using the `metrics` or `healthchecks` command, the device must be specified (some output elided):

**Example**

```
The values for metric samples and health checks can be obtained from within device mode in various ways.

10.8.2 On-Demand Metric Sampling And Health Checks

The sample Command For On-Demand Metric Samples

An administrator can run a metric sample on demand, by using the `sample` command. With it, a particular metric sample can be carried out over for a range of devices:

Example

```
[bright73->device]% sample -n node001..node002 loadone
Device Metric Value Age (sec.) Info Message
-------- --------------- ------ ---------- ------------
node001 LoadOne 0.02 1
node002 LoadOne 0.5 1
```

All metric samples can be run on demand over a selection of nodes, using `sample *`.

Example

```
[bright73->device]% sample -n node001..node002 *
Device Metric Value Age (sec.) Info Message
-------- --------------- ------ ---------- ------------
node001 AlertLevel:max -nan 1
node001 AlertLevel:sum -nan 1
node001 Baseboard_P_Vtt 1.22 1
node001 Baseboard_Temp1 26 1
...
```

The latestmetricdata command (section 10.8.3) displays the results from the latest metric samples that have been run by the cluster, rather than running metric samples on demand.

The check Command For On-Demand Health Checks

The administrator can run a health check on demand, by using the `check` command. With it, a particular health check that is to be run can be specified for a range of devices:

Example

```
[bright73->device]% check -n node001..node002 ib
Device Health Check Value Age (sec.) Info Message
----------- --------------------- --------- ---------- ------------
node001 ib no data 0 Node down
node002 ib PASS 0
```

All the health checks can be run using `check *` as follows:
Example

```
[device]\% check -n node001..node002 *
```

<table>
<thead>
<tr>
<th>Device</th>
<th>Health Check</th>
<th>Value</th>
<th>Age (sec.)</th>
<th>Info Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>ManagedServicesOk</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>mounts</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>rogueprocess</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>smart</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>interfaces</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>dmesg</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>ib</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>diskspace:2% 10% 20%</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>ntp</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node001</td>
<td>schedulers</td>
<td>no data</td>
<td>1</td>
<td>Node down</td>
</tr>
<tr>
<td>node002</td>
<td>ManagedServicesOk</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>mounts</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>rogueprocess</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>smart</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>interfaces</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>dmesg</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>ib</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>diskspace:2% 10% 20%</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>ntp</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>node002</td>
<td>schedulers</td>
<td>PASS</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The `latesthealthdata` command (section 10.8.3) displays the results from the latest health checks that have been run by the cluster, rather than running health checks on demand.

### 10.8.3 The Latest Data Values—The `latest*data` Commands

Within `device` mode, the values obtained by the latest metrics and health checks can be displayed with the `latestmetricdata` and `latesthealthdata` commands:

- **latestmetricdata**: The `latestmetricdata` command for a particular device displays the most recent value that has been obtained by the monitoring system for each item in the list of active metrics. For displaying a metric response on demand in `cmsh`, the `sample` command (page 429) can be used.

- **latesthealthdata**: The `latesthealthdata` command for a particular device displays the most recent value that has been obtained by the monitoring system for each item in the list of active health checks. Its `cmgui` equivalent is the **Health** button in the **Tasks** tabbed pane for the node (figure 2.6). For displaying a health check response on demand in `cmsh`, the `check` command (page 429) can be used.

**Using The `latestmetricdata` And `latesthealthdata` Commands**

When using the `latestmetricdata` or `latesthealthdata` commands, the device must be specified (some output elided):

Example

```
[device]\% use node001
[device[node001]]\% latestmetricdata
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Age (sec.)</th>
<th>Info Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel:max</td>
<td>30</td>
<td>76</td>
<td>FAIL schedulers</td>
</tr>
<tr>
<td>AlertLevel:sum</td>
<td>30</td>
<td>76</td>
<td>FAIL schedulers</td>
</tr>
</tbody>
</table>
Valid device grouping options and other options can be seen in the help text for the `latestmetricdata` and `latesthealthdata` commands.

**Example**

```
[bright73->device]# help latestmetricdata
Name:    latestmetricdata - Get latest metric data

Usage:   latestmetricdata [OPTIONS] [device]

Options:
    -n,  --nodes node(list)
          List of nodes, e.g. node001..node015,node20..node028,node030
          or ^/some/file/containing/hostnames

    -g,  --group group(list)
          Include all nodes that belong to the node group, e.g. testnodes
          or test01,test03
```

In addition, the `latestmetricdata` command can take the following options:

- **--raw**
  Displays data in the `Values` column without units. Default for Bright Cluster Manager up to version 7.0.

- **--human**
  Displays data in the `Values` column with units. Default for Bright Cluster Manager version 7.1 onwards.

**10.8.4 Filtering Monitoring Data Values With `monitoringdatafilter`**

The `monitoringdatafilter` command takes the latest known metric values from specified nodes and filters them according to conditions. Its usage is:

```
Usage: monitoringdatafilter [OPTIONS] <filter>
```

The available options are displayed when `help monitoringdatafilter` is run in device mode. The options are mainly ways of grouping the nodes to be filtered. Other options include `--raw|--human`, and `--precision`, as described in the table of options to the `dumpmetricdata` command (page 434).

The filter is a logical conditional expression. If it is evaluated to be true, then output is displayed for the metrics used in the expression. The expression can take the following operators, which have their usual meanings:

```
<, <=, >, >=, ||, OR, &&, AND, =, !=, +, -, *, /, ()
```

**Example**

© Bright Computing, Inc.
432 Cluster Monitoring

[bright73->device]% monitoringdatafilter -n node001 "System_Fan_1<8000"
No matching nodes / no data
[bright73->device]% monitoringdatafilter -n node001 "System_Fan_1>8000"
Device System_Fan_1
--------- --------------------------------
node001 8434.55
[bright73->device]% monitoringdatafilter -n node001 "System_Fan_1>9000"
No matching nodes / no data
[bright73->device]% monitoringdatafilter -n node001 "(System_Fan_1+System_Fan_2) > 9000"
Device System_Fan_1 System_Fan_2
--------- -------------------------------- -----------------------
node001 8289.12 8434.55
[bright73->device]% monitoringdatafilter -n node001 "System_Fan_1/System_Fan_2 < 1"
Device System_Fan_1 System_Fan_2
--------- -------------------------------- -----------------------
node001 8289.12 8434.55

10.8.5 The showhealth Command For An Overview Of Health State

The latesthealthdata command (section 10.8.3) displays the results from the latest health checks that have been run by the cluster for specified nodes. An administrator is however usually only interested in nodes in the cluster that have a health problem or are exceeding a metric threshold. Such nodes raise the AlertLevel (section 10.2.7) metric.

The showhealth command by default lists devices and their last recorded AlertLevel:max values, for metrics or health checks that have raised the AlertLevel values.

By default, failed health checks are listed in the Failed column, and metrics that have exceeded their threshold are listed in the Thresholds column along with the threshold value that they have crossed. If a health check has no last-recorded value, then it is listed in the Unknown column.

Example

[bright73->device]% showhealth
Device AlertLevel Failed Thresholds Unknown
--------- ----------- ----------------------- ----------- --------
chassis1 20 DeviceIsUp
node001 40 DeviceIsUp, nanchecker
node002 40 DeviceIsUp, nanchecker
node003 10 coredump random>0 logcheck

The help text for showhealth presents further options, including the ability to specify results for: all devices, a list of devices, and devices by status (sections 5.5.3 and 5.5.4).

10.8.6 Data Values Over Time—The dump* Commands

Within device mode, the following commands display monitoring data values over a specified period:

- dumpmetricdata: The dumpmetricdata command for a particular device displays the values of metrics obtained over a specific period by the monitoring system for a particular metric.

- dumphealthdata: The dumphealthdata command for a particular device displays the values of health checks obtained over a specific period by the monitoring system for a particular health check.

- dumpstatistics: The dumpstatistics command for a particular device displays statistics obtained over a specific period by the monitoring system for a particular metric item in the list of checks.
Using The `dumpmetricdata` And `dumphealthdata` Commands

A concise overview of the `dumpmetricdata` or `dumphealthdata` commands can be displayed by, for example, typing in “help dumpmetricdata” in the device mode of cmsh.

The usage of the `dumpmetricdata` and `dumphealthdata` commands is:

```
dumpmetricdata [OPTIONS] <start-time> <end-time> <metric> [device]
```

The mandatory arguments for the times, the metric being dumped, and the device or devices being sampled, have values that are specified as follows:

- The metric item `<metric>` for which the data values are being gathered must always be given. Metrics currently in use can conveniently be listed by running the `metrics` command (section 10.8.3).

- If `[device]` is not specified when running the `dumpmetricdata` or `dumphealthdata` command, then it must either be set by specifying the object from the device mode of cmsh (for example, with “use node001”), or by specifying it in the options. The options allow specifying, amongst others, a list, a group, or a category of devices.

- 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 `dumpmetricdata` or `dumphealthdata` 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>

An example of how the preceding mandatory arguments to the `dumpmetricdata` command might be used is:

**Example**

```
[bright73->device]% dumpmetricdata -10m now cpuidle node001
Time           Value
------------------------
```

© Bright Computing, Inc.
The options applied to the samples are specified as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>-s, --status</td>
<td>&lt;state&gt;</td>
<td>for nodes in state UP, OPENING, DOWN, and so on</td>
</tr>
<tr>
<td>-i, --intervals</td>
<td>&lt;number&gt;</td>
<td>number of samples to show</td>
</tr>
<tr>
<td>-k, --kind</td>
<td>average, min, max</td>
<td>show the average (default), minimum, or maximum of set of stored values, out of the list of device samples</td>
</tr>
<tr>
<td>-m, --sum</td>
<td></td>
<td>sum over specified devices</td>
</tr>
<tr>
<td>-x, --max</td>
<td></td>
<td>maximum over specified devices</td>
</tr>
<tr>
<td>--min</td>
<td></td>
<td>minimum over specified devices</td>
</tr>
<tr>
<td>--avg</td>
<td></td>
<td>average over specified devices</td>
</tr>
<tr>
<td>-u, --unix</td>
<td>&quot;&lt;string&gt;&quot;</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>--precision</td>
<td>&quot;&lt;number&gt;&quot;</td>
<td>Use at most &lt;number&gt; significant digits for floating point values</td>
</tr>
<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>--raw</td>
<td></td>
<td>show the metric value without units</td>
</tr>
<tr>
<td>--human</td>
<td></td>
<td>show the metric value with units (default)</td>
</tr>
</tbody>
</table>

The -s|--status option selects only for nodes in that state. States are explained in section 5.5.

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

[bright73->device] % dumpmetricdata -i 0 -3m now loadone node001
# From Thu Oct 20 17:22:12 2011 to Thu Oct 20 17:59:12 2011
Thu Oct 20 17:22:12 2011 0
Thu Oct 20 17:48:00 2011 0
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 for performance reasons.

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).

The \(-m|--sum\) option sums a specified metric for specified nodes, for a set of specified times. For 2 nodes, over a period 2 hours ago, with values interpolated over 3 time intervals, the option can be used as follows:

**Example**

```
[bright73->device] % dumpmetricdata -2h now -i 3 loadone -n node00[1-2] --sum
```

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu Sep 19 14:54:19 2013</td>
<td>1.330446</td>
</tr>
<tr>
<td>Thu Sep 19 15:34:19 2013</td>
<td>1.944914</td>
</tr>
<tr>
<td>Thu Sep 19 16:14:19 2013</td>
<td>2.529622</td>
</tr>
</tbody>
</table>
```

Each entry in the values column in the preceding table is the sum of \(\text{loadone}\) displayed by node001, and by node002, at that time, as can be seen from the following corresponding table:

**Example**

```
[bright73->device] % dumpmetricdata -2h now -i 3 loadone -n node00[1-2]
```

```
<table>
<thead>
<tr>
<th>Device</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>Thu Sep 19 14:54:19 2013</td>
<td>0.715523</td>
</tr>
<tr>
<td>node001</td>
<td>Thu Sep 19 15:34:19 2013</td>
<td>1.03349</td>
</tr>
<tr>
<td>node001</td>
<td>Thu Sep 19 16:14:19 2013</td>
<td>1.264811</td>
</tr>
<tr>
<td>node002</td>
<td>Thu Sep 19 14:54:19 2013</td>
<td>0.614923</td>
</tr>
<tr>
<td>node002</td>
<td>Thu Sep 19 15:34:19 2013</td>
<td>0.911424</td>
</tr>
<tr>
<td>node002</td>
<td>Thu Sep 19 16:14:19 2013</td>
<td>1.264811</td>
</tr>
</tbody>
</table>
```

Each \(\text{loadone}\) 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.
Using The `dumpstatistics` Commands

The usage of the `dumpstatistics` command is:

```bash
dumpstatistics [OPTIONS] <start-time> <end-time> <metric> [device]
```

and it follows the pattern of earlier (page 433) for the `dumpmetricdata` and `dumphealthdata` commands.

There are two significant differences:

- The `-p` option sets percentile boundaries. By default, the statistics are divided into percentiles of 25%.
- The `-i` option is set by default to a value of 20. A setting of `0` (which displays non-interpolated values when used with `dumpmetricdata` and `dumphealthdata`) is not a valid value for `dumpstatistics`.

Example

```bash
[bright73->device]# dumpstatistics -i 5 -1h -p 100 now loadone -n node001..node003
```

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fri Oct 21 17:04:30 2011</td>
<td>Fri Oct 21 17:16:30 2011</td>
<td>0.11</td>
</tr>
<tr>
<td>Fri Oct 21 17:16:30 2011</td>
<td>Fri Oct 21 17:28:30 2011</td>
<td>0.08</td>
</tr>
<tr>
<td>Fri Oct 21 17:28:30 2011</td>
<td>Fri Oct 21 17:40:30 2011</td>
<td>0.09</td>
</tr>
<tr>
<td>Fri Oct 21 17:40:30 2011</td>
<td>Fri Oct 21 17:52:30 2011</td>
<td>0.27</td>
</tr>
<tr>
<td>Fri Oct 21 17:52:30 2011</td>
<td>Fri Oct 21 18:04:30 2011</td>
<td>0.22</td>
</tr>
</tbody>
</table>

```bash
[bright73->device]# dumpstatistics -i 5 -u -1h now loadone -n node001..node003
```

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>0%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1319444362</td>
<td>1319445082</td>
<td>0</td>
<td>0.03</td>
<td>0.0475</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>1319445082</td>
<td>1319445802</td>
<td>0</td>
<td>0</td>
<td>0.0025</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>1319445802</td>
<td>1319446522</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>1319446522</td>
<td>1319447242</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>1319447242</td>
<td>1319447962</td>
<td>0</td>
<td>0</td>
<td>0.015</td>
<td>0.05</td>
<td></td>
</tr>
</tbody>
</table>

10.8.7 Setting Fictional Data Values—The `fakedata` Command

Sometimes an administrator would like to test how the cluster behaves when a monitored metric value changes beyond a threshold, (page 425), or when a monitored health check changes.

For example, if a node goes down, the administrator may have configured the cluster to send an e-mail alert to root. However, actually bringing the node down on a production system, to test such an alert works, is often unwanted.

So instead, a quick way to test that the alert works as expected for the node is to use the `fakedata` command. Thus, the `DeviceIsUp` health check value of the node, which is really PASS, is manually set to a fictional value of FAIL:

Example

```bash
[head1->device]# check -n node001 deviceisup
Health Check Value Age (sec.) Info Message
----------------- ----- --------- ----------------
DeviceIsUp PASS 0

[head1->device]# fakedata -n node001 deviceisup fail
Fake data count: 1
```

[436] Cluster Monitoring

© Bright Computing, Inc.
After some time, within the check interval period, the monitoring system detects the fake FAIL state.

If the node has been configured in Bright Cluster Manager to send an e-mail alert for this state to the root mailbox using the SendEmail action (page 397), then the mailbox can be checked. After testing that the mail has reached and the alert system is working as expected, the fake value can be removed with the -l option to fakedata:

**Example**

```
[head1->device]# fakedata -l -n node001
Fake data count: 0

[head1->device]# check -n node001 deviceisup
Fri Jan 27 10:55:57 2017 [notice] node001: Check ‘DeviceIsUp’ is in state PASS on node001
```

10.9 The User Portal

10.9.1 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 bright73, it is accessible to users for a login via a browser at the URL http://bright73, or more directly via https://bright73:8081/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 standard CA, if that is preferred.
10.9.2 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.

However, a possible workaround is to set a shared (common) username/password for all users. This requires that some parts at the head and tail of the file /cm/local/apps/cmd/etc/htdocs/userportal/app/controllers/AuthController.js be edited.

Example

[root@bright73 ~]# cd /cm/local/apps/cmd/etc/htdocs/userportal/app/controllers
[root@bright73 controllers]# head AuthController.js

'strict';
app.controller('AuthController', function($scope, $log, $location, $window, $timeout, AUTH,
  AUTH_EVENTS, AuthService) {
  $scope.credentials = {
    username: '',
    password: ''
  };

  $scope.pwFieldAttr = 'password';
  $scope.pwOpenEyeIcon = true;

[root@bright73 controllers]# tail AuthControllers.js

function init() {
  doKeepAlive();
}

init();
});
[root@bright73 controllers]# vi AuthControllers.js

Within the editor that opens up, the changes that should be made can be as follows:

- The empty values for username and password seen in the first few lines of the file should be changed to a user name and password that all users will use. In the example that follows, the username and password are set to the same string (userportalreadaccess), but they can also be set to differ from each other.

- The line immediately after the init() line, seen in the last few lines, should have a new “bypass” line inserted.

After the changes have been saved and the editor is exited, the result the looks like (altered parts shown in bold):

[root@bright73 controllers]# head AuthController.js

'use strict';
app.controller('AuthController', function($scope, $log, $location, $window, $timeout, AUTH,
  AUTH_EVENTS, AuthService) {
  $scope.credentials = {
    username: 'userportalreadaccess',

© Bright Computing, Inc.
password: ‘userportalreadaccess’
}

$scope.pwFieldAttr = ‘password’;
$scope.pwOpenEyeIcon = true;
[root@bright73 controllers]# tail AuthControllers.js
  $timeout(doKeepAlive, AUTH.keepAlive);
});

function init() {
  doKeepAlive();
};

init();
  $scope.doLogin($scope.credentials);
});

The administrator should create the user userportalreadaccess, and set a password for it. All users can then only access the user portal via the common username/password combination.

If the cm-webportal package is updated, then the changes in AuthControllers.js will be overwritten.

10.9.3 User Portal Home Page
The default user portal home page allows a quick glance to convey the most important cluster-related information for users (figure 10.38):

![Figure 10.38: User Portal: Overview Page](https://example.com/figure10.38)

The following items are displayed on the default home page, which is the overview page:
• a Message Of The Day. This can be edited in /cm/local/apps/cmd/etc/htdocs/userportal/app/partials/messageOfTheDay.html

• links to the documentation for the cluster

• contact information. This can be edited in /cm/local/apps/cmd/etc/htdocs/userportal/app/partials/contact.html

• an overview of the cluster state, displaying some cluster parameters.
  
  - This can conveniently be set to refresh automatically every 60 seconds, for example, by inserting an HTML meta attribute:

  ```html
  <meta http-equiv="refresh" content="60">
  ```
  in the line immediately after the `<body ng-controller="ApplicationController">` line in `/cm/local/apps/cmd/etc/htdocs/userportal/index.html`

Other user-accessible pages are available by default, and are described further in Chapter 12 of the User Manual. The default pages that are available can be configured from the `/cm/local/apps/cmd/etc/htdocs/userportal/index.html` page.

The user portal is designed to serve files only, and will not run executables such as PHP or similar CGI scripts.

### 10.10 Job Monitoring

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 the other sections of this chapter.

Users can also select a job that is currently running, or that has recently run, and get metrics for nodes, jobs, and accumulated metrics per user. This is known as job monitoring. It is covered in this section (section 10.10), and uses job metrics.

#### 10.10.1 Job Metrics

Bright Cluster Manager job metrics setup is carried out using the `cm-jobmetrics-setup` script (section 10.10.2), while job metrics collection uses cgroups (section 7.10). Thus each job should be associated by a workload manager with a unique cgroup automatically. Bright Cluster Manager configures the following workload managers to run jobs in cgroups:

- Slurm
- UGE
- LSF
- OpenLava
- Torque

Notes:

1. OGS for now does not support cgroups.

2. PBS Pro can be configured to run with cgroups, but this configuration is not provided by Bright Cluster Manager 7.3 and job metrics are not supported with PBS Pro. To enable cgroups in PBS Pro, Altair should be asked for a special hook script for cgroups configuration.
Different workload managers use different cgroup controllers when creating a cgroup for a job. The controllers enable different metric collections to be carried out by the Linux kernel. In order to have the same set of collected metrics for each of the workload managers, Bright Cluster Manager configures `systemd` to mount the following cgroup controllers to the same directory:

- `blkio`
- `cpu`
- `cpuacct`
- `freezer`
- `hugetlb`
- `memory`

For example: Slurm job metrics collection uses the following cgroup directories structure:

**Example**

```
[root@node001 ~]# tree -d --charset=C -L 2 /sys/fs/cgroup/
/sys/fs/cgroup/
|-- blkio -> blkio,cpu,cpuacct,freezer,hugetlb,memory
|-- blkio,cpu,cpuacct,freezer,hugetlb,memory
|   |-- slurm
|-- cpu -> blkio,cpu,cpuacct,freezer,hugetlb,memory
|-- cpuacct -> blkio,cpu,cpuacct,freezer,hugetlb,memory
|-- cpuset
|-- devices
|-- freezer -> blkio,cpu,cpuacct,freezer,hugetlb,memory
|-- hugetlb -> blkio,cpu,cpuacct,freezer,hugetlb,memory
|-- memory -> blkio,cpu,cpuacct,freezer,hugetlb,memory
|-- net_cls
|-- perf_event
   `-- systemd
      |-- system.slice
      `-- user.slice
```

The directory `blkio,cpu,cpuacct,freezer,hugetlb,memory` is created, the appropriate cgroup controllers are mounted, the directory links are created, and this is all carried out automatically by `systemd`. The trigger for this is when the `JoinControllers` parameter is set by the `cm-jobmetrics-setup` script (section 10.10.2) in a software image, or on a head node for jobs that run on the head node.

**Example**

```
[root@node001 ~]# grep JoinControllers /etc/systemd/system.conf
JoinControllers=blkio,cpu,cpuacct,freezer,hugetlb,memory
[root@node001 ~]#
```

The following table lists some of the most useful job metrics that Bright Cluster Manager can monitor and visualize. In the table, the text `<device>` denotes a block device name, such as `sda`.

© Bright Computing, Inc.
## Metric Name | Description | Cgroup Source File
---|---|---
blkio.time:<device> | Time job had I/O access to device | blkio.time_recursive
blkio.sectors:<device> | Sectors transferred to or from specific devices by a cgroup | blkio.sectors_recursive
blkio.io_serviced_read:<device> | Read I/O operations | blkio.io_serviced_recursive
blkio.io_serviced_write:<device> | Write I/O operations | blkio.io_serviced_recursive
blkio.io_serviced_sync:<device> | Synchronous I/O operations | blkio.io_serviced_recursive
blkio.io_serviced_async:<device> | Asynchronous I/O operations | blkio.io_serviced_recursive
blkio.io_service_read:<device> | Bytes read | blkio.io_service_bytes_recursive
blkio.io_service_write:<device> | Bytes written | blkio.io_service_bytes_recursive
blkio.io_service_sync:<device> | Bytes transferred synchronously | blkio.io_service_bytes_recursive
blkio.io_service_async:<device> | Bytes transferred asynchronously | blkio.io_service_bytes_recursive
cpuacct.usage | Total CPU time consumed by all job processes | cpuacct.usage
cpuacct.stat.user | User CPU time consumed by all job processes | cpuacct.stat
cpuacct.stat.system | System CPU time consumed by all job processes | cpuacct.stat
memory.usage | Total current memory usage | memory.usage_in_bytes
memory.memsw.usage | Sum of current memory plus swap space usage | memory.memsw.usage_in_bytes
memory.memsw.max_usage | Maximum amount of memory and swap space used | memory.memsw.max_usage_in_bytes
memory.failcnt | How often the memory limit has reached the value set in | memory.limit_in_bytes

...continues
...continued

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Cgroup Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory.memsw.failcnt</td>
<td>How often the memory plus swap space limit has reached the value set in memory.memsw.limit_in_bytes</td>
<td>memory.memsw.failcnt</td>
</tr>
<tr>
<td>memory.stat.swap</td>
<td>Total swap usage</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.cache</td>
<td>Total page cache, including tmpfs (shmem)</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.mapped_file</td>
<td>Size of memory-mapped mapped files, including tmpfs (shmem)</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.unevictable</td>
<td>Memory that cannot be reclaimed</td>
<td>memory.stat</td>
</tr>
</tbody>
</table>

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.

The metrics in the preceding table are enabled by default. There are also over 40 other advanced metrics that can be enabled, as described in the following table:

<table>
<thead>
<tr>
<th>Metric Name</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>
<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>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Cgroup Source File</th>
</tr>
</thead>
<tbody>
<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>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>
<tr>
<td>blkio.io_queued_async: &lt;device&gt;</td>
<td>Number of requests queued for I/O asynchronous operations</td>
<td>blkio.io_queued_recursive</td>
</tr>
</tbody>
</table>

...continues
...continued

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Cgroup Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu.stat.nr_periods</td>
<td>Number of period intervals (as specified in cpu.cfs_period_us) that have elapsed</td>
<td>cpu.stat</td>
</tr>
<tr>
<td>cpu.stat.nr_throttled</td>
<td>Number of times processes have been throttled (that is, not allowed to run because they have exhausted all of the available time as specified by their quota)</td>
<td>cpu.stat</td>
</tr>
<tr>
<td>cpu.stat.throttled_time</td>
<td>Total time duration for which processes have been throttled</td>
<td>cpu.stat</td>
</tr>
<tr>
<td>memory.stat.rss</td>
<td>Anonymous and swap cache, not including tmpfs (shmem)</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.pgpgin</td>
<td>Number of pages paged into memory</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.pgpgout</td>
<td>Number of pages paged out of memory</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.active_anon</td>
<td>Anonymous and swap cache on active least-recently-used (LRU) list, including tmpfs (shmem)</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.inactive_anon</td>
<td>Anonymous and swap cache on inactive LRU list, including tmpfs (shmem)</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.active_file</td>
<td>File-backed memory on active LRU list</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.inactive_file</td>
<td>File-backed memory on inactive LRU list</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.hierarchical_memory_limit</td>
<td>Memory limit for the Hierarchy that contains the memory cgroup of job</td>
<td>memory.stat</td>
</tr>
<tr>
<td>memory.stat.hierarchical_memsw_limit</td>
<td>Memory plus swap limit for Hierarchy that contains the memory cgroup of job</td>
<td>memory.stat</td>
</tr>
</tbody>
</table>

If job metrics are set up (section 10.10.2), 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.

© Bright Computing, Inc.
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 in the `sample_cgroup` configuration file (section 10.10.4).

### 10.10.2 Job Metrics Collection Setup

By default Bright Cluster Manager does not collect job metrics. To enable the collection of job metrics, the administrator must run the `cm-jobmetrics-setup` command, and answer several questions that come up. When the command has finished executing, then the compute nodes on which the jobs are to run should be rebooted. After the reboot CMDaemon starts to collect the metrics.

**Example**

```
[root@bright73 ~]# cm-jobmetrics-setup
Workload manager name (default: "slurm"):
  lsf,
  openlava,
  slurm,
  torque,
  uge
> slurm
Node categories for running jobs (default: "default"):
  default,
  none
>
Run jobs on head node (default: "no"):
  no,
  yes
>
Configuring metric collection ...  

Job metric collection has been enabled.
```

### 10.10.3 Job Information Retention

Each job adds a set of metric values to the monitoring database. The longer a job runs, the more data is added to the database. By default, old values are cleaned up from the database in order to limit its size. In Bright Cluster Manager 7.3 there are four advanced configuration flags to control the job data retention, with names and default values as follows:

<table>
<thead>
<tr>
<th>Advanced Configuration Flag</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobInformationKeepDuration</td>
<td>604800</td>
</tr>
<tr>
<td>JobInformationKeepCount</td>
<td>8192</td>
</tr>
<tr>
<td>JobInformationMinimalJobDuration</td>
<td>0</td>
</tr>
<tr>
<td>JobInformationFlushInterval</td>
<td>600</td>
</tr>
</tbody>
</table>

© Bright Computing, Inc.
• **JobInformationKeepDuration**: 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 week ($24 \times 3600 \times 7$ seconds).

• **JobInformationKeepCount**: If the total number of jobs is greater than $(\text{JobInformationKeepCount} + 10\%)$, 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. The default value of JobInformationKeepCount is 8192. If it is set to 0, then none of the job records and content are removed.

• **JobInformationMinimalJobDuration**: 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**: If this interval, in seconds, is set, then the cache is flushed to the database with that interval. Its default value is 10 minutes ($10 \times 60$ seconds). Values of around 30 seconds or less will conflict with the default CMDaemon maintenance timeout value of 30 seconds, and are unlikely to work well.

### 10.10.4 Job Metrics Sampling Configuration
CMDaemon executes the script `/cm/local/apps/cmd/scripts/metrics/sample_cgroup` in order to collect job metrics from the cgroups in which a job runs. The administrator can tune some low level metric collection options for the script in the configuration file for the image used by the node. For example, for the default image the configuration file is located under the images directory on the head node, at `/cm/images/default-image/cm/local/apps/cmd/scripts/metrics/configfiles/sample_cgroup.conf`.

The configuration file has the following JSON-format content by default:

```json

"logger": { "level": "debug", "filename": "/var/log/sample_metrics.log" },
"exclude_devices": ["loop", "sr"],
"include_devices": [],
"enable_advanced": false,
"exclude_metrics": [],
"include_metrics": []
```

The configuration parameters are:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger/level</td>
<td>Log messages level (options: debug, info)</td>
</tr>
<tr>
<td>logger/filename</td>
<td>Log messages file</td>
</tr>
<tr>
<td>exclude_devices</td>
<td>Block devices for which sample_cgroup will not collect job metrics</td>
</tr>
</tbody>
</table>

...continues
Cluster Monitoring

---continued

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

The parameter exclude_metrics can be used to exclude metrics that are currently enabled. For example, if advanced metrics collection is enabled with the enable_advanced flag, then exclude_metrics allows either default or advanced metrics to be excluded by name.

10.10.5 Job Monitoring In cmsh

Using cmsh, the following commands allow job metrics and historical information to be retrieved within jobs mode:

**filter** The command provides filtered historic job-related information. It does not provide metrics data. The command and its options can be use 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

**Example**

```
[bright73->jobs(slurm)]% filter --limit 2
Job ID Job name User Queue Start time End time Nodes Exit code
------- --------- ---- ----- ------------------ ------------------- ------- ---------
10 slurm.job cmsupport defq 02/02/2016 13:58:22 03/02/2016 00:58:24 node001 0
11 slurm.job cmsupport defq 02/02/2016 13:58:22 03/02/2016 00:58:24 node001 0
[bright73->jobs(slurm)]%
```

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 of the options to filter can be seen by running help filter.

**dumpmetricdata** The dumpmetricdata command for job metrics is somewhat similar to the dumpmetric command for device metrics (section 10.8.6), and shows monitoring data for a job ID over a period of time. Options allow metric values to be retrieved and filtered over a period of time. The user can also specify custom periods or select according to the user name of the job owner.

The usage of the dumpmetricdata command for job metrics is:

```
dumpmetricdata [OPTIONS] [<start-time> <end-time>] <metric> <job ID>
```

**Example**

© Bright Computing, Inc.
The data is shown per node if the job uses several nodes.

Further details of the options to dumpmetricdata for job metrics can be seen by running help dumpmetricdata within jobs mode.

**statistics** 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 format of the output can be specified with the --format option. The statistics can be grouped by hour, day, week or a custom interval.

**Example**

```
[bright73->jobs(slurm)]% statistics
Interval User Group Pending Running Finished Error Nodes
--------- ------- ------- -------- -------- --------- ------ ----- 
N/A 0 7 32 0 34
```

Further details of the options to statistics can be seen by running help statistics.

**10.10.6 Job Metrics Monitoring With cmgui**

Collected job metrics can be visualized with cmgui in the same way that device metrics are visualized. The monitoring window is opened from the Monitoring menu item, and within the monitoring window, the job metrics are shown grouped under the Workload resource in the navigation tree, under the RESOURCES panel at the left hand side of the window (figure 10.39):
Figure 10.39: Job metrics monitoring in cmgui

Inside the Workload resource, user names can be seen. Each username represents a group of job metrics associated with a user. If there are no job metrics collected for a particular user, then the name of that user is not displayed.

Two types of subgroups can be found in the username metrics group:

1. **Total**: this is the literal name of the subgroup. The Total subgroup, after data consolidation (page 399) takes place, contains the consolidated metrics for each user. Typically, the administrator views Total to see how a particular user used the cluster.

2. **<date>**: Under each <date> subgroup, the administrator can see a list of job names and their IDs, in the format:
   
   `<job name> (job ID)`
Each of the <date> subgroups represents a set of metrics for particular job. A list of metric classes is seen under a each subgroup. Job-centric metric class names start with CGroup. Both CGroup metrics and node metrics can be displayed.

By default node-centric metrics are not shown and are not displayed under the job metrics group. This restriction can be removed, by choosing the Settings option from the File menu of the monitoring window, and then removing the tickmark from the option List only CGroup metrics for jobs (figure 10.40). Both types of metrics can then be viewed per job. The Monitoring window should be re-opened in order to implement the change after applying it.

Figure 10.40: Job metrics options window
11

Day-to-day Administration

This chapter discusses several tasks that may come up in day-to-day administration of a cluster running Bright Cluster Manager.

11.1 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 11.1.1), runs from within the OS shell. That is, pdsh executes its commands from within bash by default.

- pexec (parallel execute, sections 11.1.2 and 11.1.3), runs from within CMDaemon. That is, pexec executes its commands from within the cmsh or cmgui front ends.

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 or cmgui, 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 or cmgui, 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.

11.1.1 pdsh In The OS Shell

Packages Required For pdsh

By default, the following packages must be installed from the Bright Cluster Manager repositories to the head node for pdsh to function fully:

- pdsh
- genders
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 Bright Cluster Manager in the default configuration associates the following `genders` with a list of nodes:

- **all**: all the nodes in the cluster, head and regular nodes.
- **category=default**: the nodes that are in the default category
- **computenode**: regular nodes
- **headnode**: node or nodes that are head nodes.

In a newly-installed cluster using default settings, the genders `category=default` and `computenode` have the same list of nodes to begin with.

The default `/etc/genders` file has a section that is generated and maintained by CMDaemon, but the file can be altered by the administrator outside the CMDaemon-maintained section. However, it is not recommended to change the file manually frequently. For example, tracking node states with this file is not recommended. Instead, the package `pdsh-mod-cmupdown` provides the `-v` option for node state tracking functionality, and how to use this and other `pdsh` options is described in the next section.

**pdsh Options**

In the OS shell, running `pdsh -h` displays the following help text:

Usage: `pdsh [-options] command` ...

- `-S` return largest of remote command return values
- `-h` output usage menu and quit
- `-V` output version information and quit
- `-q` list the option settings and quit
- `-b` disable ^C status feature (batch mode)
- `-d` enable extra debug information from ^C status
- `-l` user execute remote commands as user
- `-t seconds` set connect timeout (default is 10 sec)
- `-u seconds` set command timeout (no default)
- `-f n` use fanout of n nodes
- `-w host,host,...` set target node list on command line
- `-x host,host,...` set node exclusion list on command line
- `-R name` set rcmd module to name
- `-M name,...` select one or more misc modules to initialize first
- `-N` disable hostname: labels on output lines
- `-L` list info on all loaded modules and exit
- `-v` exclude targets if they are down
- `-g query,...` target nodes using genders query
- `-X query,...` exclude nodes using genders query
- `-F file` use alternate genders file `file`
- `-i` request alternate or canonical hostnames if applicable
- `-a` target all nodes except those with "pdsh_all_skip" attribute
- `-A` target all nodes listed in genders database

available rcmd modules: ssh,exec (default: ssh)
Further options and details are given in `man pdsh(1)`.

Examples Of pdsh Use

For the examples in this section, a cluster can be considered that is set up with two nodes, with the state of node001 being UP and that of node002 being DOWN:

```
[root@bright73 ~]# cmsh -c "device status"
node001 .................. [ UP ]
node002 .................. [ DOWN ]
bright73 ................. [ 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@bright73 ~]# pdsh -A hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@bright73: node002: ssh exited with exit code 255
bright73: bright73
```

**-v:** With this option an attempt is made to run the command only on nodes that are in the state UP:

**Example**

```
[root@bright73 ~]# pdsh -A -v hostname
node001: node001
bright73: bright73
```

**-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@bright73 ~]# pdsh -v -g computenode hostname
node001: node001
```

**-w:** This option allows a node list (man `pdsh(1)`) to be specified on the command line itself:

**Example**

```
[root@bright73 ~]# pdsh -w node00[1-2] hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@bright73: node002: ssh exited with exit code 255
```
-x: This option is the converse of -w, and excludes a node list that is specified on the command line itself:

**Example**

```
[root@bright73 ~]# pdsh -x node002 -w node00[1-2] hostname
node001: node001
```

**The dshbak Command**

The dshbak (distributed shell backend formatting filter) command is a filter that reformats pdsh output. It comes with the pdsh package.

Running dshbak with the -h option displays:

```
[root@bright73 ~]# dshbak -h
Usage: dshbak [OPTION]...
   -h     Display this help message
   -c     Coalesce identical output from hosts
   -d DIR Send output to files in DIR, one file per host
   -f     With -d, force creation of DIR
```

Further details can be found in `man dshbak(1)`.

For the examples in this section, it is assumed that all the nodes in the cluster are now up. That is, `node002` used in the examples of the preceding section is now also up. Some examples to illustrate how dshbak works are then the following:

**Without dshbak:**

**Example**

```
[root@bright73 ~]# pdsh -A ls /etc/services /etc/yp.conf
bright73: /etc/services
bright73: /etc/yp.conf
node001: /etc/services
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright73: node001: ssh exited with exit code 2
node002: /etc/services
node002: /etc/yp.conf
```

**With dshbak, with no dshbak options:**

**Example**

```
[root@bright73 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright73: node001: ssh exited with exit code 2
```

```
----------------
bright73
----------------
/etc/services
/etc/yp.conf
----------------
node001
----------------
```

© Bright Computing, Inc.
11.1 Parallel Shells: \textit{pdsh} And \textit{pexec}

\begin{verbatim}
/etc/services
/etc/yp.conf
[root@bright73 ~]#
\end{verbatim}

With \texttt{dshbak}, with the \texttt{-c} (coalesce) option:

\textbf{Example}

\begin{verbatim}
[root@bright73 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak -c
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright73: node001: ssh exited with exit code 2
----------------
node002,bright73
----------------
/etc/services
/etc/yp.conf
----------------
node001
----------------
/etc/services
[root@bright73 ~]#
\end{verbatim}

The \texttt{dshbak} utility is useful for creating human-friendly output in clusters with larger numbers of nodes.

\subsection*{11.1.2 \texttt{pexec} in \texttt{cmsh}}

In \texttt{cmsh}, the \texttt{pexec} command is run from \texttt{device} mode:

\textbf{Example}

\begin{verbatim}
[bright73->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
\end{verbatim}

\subsection*{11.1.3 \texttt{pexec} in \texttt{cmgui}}

In \texttt{cmgui}, it is executed from the \texttt{Parallel Shell} tab after selecting the cluster from the resource tree (figure 11.1):

\copyright Bright Computing, Inc.
For large numbers of nodes, rendering the node subpanes (little boxes) in the Parallel shell tabbed pane of figure 11.1 can take a long time. To improve the cmgui experience, ticking the checkbox for Only user parallel shell single view (enable for large node counts) in the File menu under settings speeds up the rendering significantly at the cost of removing the borders of the subpanes, while ticking the Hide down button in the Parallel shell tabbed pane hides further messages about nodes that are in the DOWN state (section 2.1.1). If the Join output checkbox is ticked, then output that is the same is grouped into the same subpane.

Running parallel shell commands from cmsh instead is faster in most cases due to less graphics rendering overhead.

11.1.4 Using The -j|--join Option Of pexec

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 trivial. For example, to see if all nodes have the same mounts on a cluster with 10 nodes, of which node002 is down:

**Example**

```
[bright73->device]% pexec -j -c default "mount|sort"
Nodes down: node002
[node002]
Node down
[node001,node003..node010]
/dev/hda1 on / type ext3 (rw,noatime,nodiratime)
/dev/hda2 on /var type ext3 (rw,noatime,nodiratime)
/dev/hda3 on /tmp type ext3 (rw,nosuid,nodev,noatime,nodiratime)
/dev/hda6 on /local type ext3 (rw,noatime,nodiratime)
```
Here, the -j option joins up identical fields (like the standard unix text utility, join). Additional 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.

11.1.5 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:

[bright73-->device]$%help <pcommand>

11.2 Getting Support With Cluster Manager Issues

Bright 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. This section describes how to get support for such issues.

11.2.1 Support Via E-mail

If the reseller from whom Bright 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 goes via standard e-mail.

© Bright Computing, Inc.
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 operated by the Chinese Ministry of Public Security.

As a supplement to e-mail support, Bright Computing also provides the cm-diagnose (section 11.2.2) and the request-remote-assistance (section 11.2.3) utilities to help resolve issues. The use of third-party screen-sharing utilities (section 11.2.4) is also possible.

11.2.2 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@bright73 ~]# cm-diagnose
To be able to contact you about the issue, please provide your e-mail address (or support ticket number, if applicable):
franknfurter@example.com

To diagnose any issue, it would help if you could provide a description about the issue at hand.
Do you want to enter such description? [Y/n]

End input with ctrl-d
I tried X, Y, and Z on the S2464 motherboard. When that didn’t work, I tried A, B, and C, but the florbish is grommicking.
Thank you.

If issues are suspected to be in the cmdaemon process, a gdb trace of that process is useful.
In general such a trace is only needed if 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

© Bright Computing, Inc.
Collecting process information for CMDaemon
Executing CMSH commands
Finished executing CMSH commands

Processing default-image
Processing commands
Processing file contents

Creating log file: /root/bright73_609.tar.gz

Cleaning up

Automatically submit diagnostics to
http://support.brightcomputing.com/cm-diagnose/ ? [Y/n] y

Uploaded file: bright73_609.tar.gz
Remove log file (/root/bright73_609.tar.gz)? [y/N] y

[root@bright73 ~]

11.2.3 Requesting Remote Support With request-remote-assistance
During the process of solving a ticket, a lengthy and time-consuming e-mail exchange between the cluster administrator and Bright Computing may be required before a resolution to the issue is reached. The request-remote-assistance utility can be run after a ticket has been opened up for a case, and speeds up resolution by avoiding this exchange, so is generally recommended to obtain the quickest resolution for such problems.

The request-remote-assistance utility allows a Bright Computing engineer to securely tunnel into the cluster 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.

It is run as follows:

Example

[root@bright73 ~]# 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: bright73.NOFQDN
Connected on port: 7000
ctrl-c to terminate this session

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. 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@bright73 ~]#

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.

11.2.4 Requesting Remote Support With A Shared Screen Utility

If the request-remote-assistance utility (section 11.2.3) 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 (preferred)
• WebEx
• 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.

11.3 Backups

11.3.1 Cluster Installation Backup

Bright Cluster Manager does not include facilities to create backups of a cluster installation. A clone of the head node can however be created by PXE booting the new head node, and following the procedure in section 13.4.8.

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:

© Bright Computing, Inc.
11.3 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](http://www.bacula.org). Bacula requires ports 9101-9103 to be accessible on the head node. Including the following lines in the Shorewall rules file for the head node allows access via those ports from an IP address of 93.184.216.34 on the external network:

```
Example

ACCEPT net:93.184.216.34 fw tcp 9101
ACCEPT net:93.184.216.34 fw tcp 9102
ACCEPT net:93.184.216.34 fw tcp 9103
```

The Shorewall service should then be restarted to enforce the added rules.

- **rsnapshot**: rsnapshot allows periodic incremental filesystem snapshots to be written to a local or remote filesystem. Despite its simplicity, it can be a very effective tool to maintain frequent backups of a system. More information is available at [http://www.rsnapshot.org](http://www.rsnapshot.org). Rsnapshot requires access to port 22 on the head node.

### 11.3.2 Local Database Backups And Restoration

The CMDaemon database is stored in the MySQL `cmdaemon` database, and contains most of the stored settings of the cluster.

Other databases stored separately in MySQL are:

- The CMDaemon monitoring database `cmdaemon_mon`
- The Slurm account database `slurm_acct_db`

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 database, the monitoring database raw data records are not backed up locally, since these can get very large, but the rest of the database is backed up for the last 7 days too.
- The Slurm account database is not backed up.

#### 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 `cmgui` or `cmsh`.

When there is database corruption, InfoMessages (section 10.2.8) 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,

---

© Bright Computing, Inc.

---

1 Actually, more precisely, for failover configurations, there is an automated synchronization of all databases between the active and passive head nodes that allows failover to work. So in that sense the Slurm account database is “backed up”.
Day-to-day Administration

- error numbers from table handlers,
- “Error while executing” a command.

An example of an event message on a head node icarus:

**Example**

```
[icarus] 00:00:00.000 [notice] icarus: Error when reading data from monitoring database. Monitoring data will be saved. (details: Incorrect key file for table './cmdaemon_mon/MonData.MYI'; try to repair it)
```

The associated messages logged in `/var/log/cmdaemon` may show something like:

**Example**

```
Jan 18 10:31:05 icarus CMDaemon: Info: Reconnect command processed.
Jan 18 10:31:05 icarus CMDaemon: Info: MysqlBuffer: starting main mysql buffer thread
Jan 18 10:31:05 icarus CMDaemon: Info: MysqlBuffer: starting mirroring thread
Jan 18 10:31:05 icarus CMDaemon: Info: Preloading mysql key cache using query: LOAD INDEX INT\O CACHE MonMetaData
Jan 18 10:31:05 icarus CMDaemon: Info: MysqlBuffer: starting MysqlBuffer thread
Jan 18 10:31:05 icarus CMDaemon: Info: Database: Mirroring required to remote master
Jan 18 10:31:05 icarus CMDaemon: Info: Database: generalQuery returned: cmdaemon_mon.MonMetaData\preload_keys status OK
Jan 18 10:31:05 icarus CMDaemon: Info: Preloading mysql key cache using query: LOAD INDEX INT\O CACHE MonData INDEX (MonDataIndex) IGNORE LEAVES
Jan 18 10:31:05 icarus CMDaemon: Info: Database: generalQuery returned: cmdaemon_mon.MonData\ preload_keys Error Unexpected eof found when reading file '/var/lib/mysql/cmdaemon_mon/MonData.MYI' (Errcode: 0) cmdaemon_mon.MonData preload_keys status OK
```

Looking through the preceding messages, the conclusion is that the monitoring database has a corrupted MonData table. Being in MyISAM format (.MYI) means the `myisamchk` repair tool can be run on the table, for example as:

```
[root@icarus ~]# service cmd stop
[root@icarus ~]# myisamchk --recover /var/lib/mysql/cmdaemon_mon/MonData.MYI
[root@icarus ~]# 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.

© Bright Computing, Inc.
11.3 Backups

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 13.4.2) should normally provide a CMDaemon database that is current. The option also clones the Slurm database. It does not clone the monitoring database by default. The cm-clone-monitoring-db.sh helper script that comes with CMDaemon can be used to clone the monitoring database as a one-time event.

Cloning extra databases: The file /cm/local/apps/cluster-tools/ha/conf/extradbclone.xml.template can be used as a template to create a file extradbclone.xml in the same directory. The extradbclone.xml file can then be used to define additional databases to be cloned. Running the /cm/local/apps/cmd/scripts/cm-update-mycnf script then updates /etc/my.cnf. The database can then be cloned with this new MySQL configuration by running

```
cmha dbreclone <passive>
```

where <passive> is the hostname of the passive headnode.

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 1593 Oct 10 04:02 backup-monitor-Mon.sql.gz
-rw------- 1 root root 1593 Oct 9 04:02 backup-monitor-Sun.sql.gz
... 
-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 while the monitoring database snapshots are stored as backup-monitor-<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.

A similar procedure for monitoring database backups can be used to restore the table structure and InfoMessages of the data records from local backup, which means that messages and data record schema changes after the snapshot time are gone. Restoring the monitoring database from the backup only...
restores the schema and InfoMessages of the data records. It does not restore the monitoring data records themselves. To restore records, the administrator would need to retrieve a snapshot from the regular backup that the administrator should be running for the cluster.

11.4 Revision Control For Images

Bright Cluster Manager 7 introduces support for the implementations of Btrfs provided by the distributions, and thus makes it possible to carry out revision control for images efficiently.

11.4.1 Btrfs: The Concept And Why It Works Well In Revision Control For Images

Btrfs, often pronounced “butter FS”, is a Linux implementation of a copy-on-write (COW) filesystem.

A COW design for a filesystem follows the principle that, when blocks of old data are to be modified, then the new data blocks are written in a new location (the COW action), leaving the old, now superseded, copy of the data blocks still in place. Metadata is written to keep track of the event so that, for example, the new data blocks can be used seamlessly with the contiguous old data blocks that have not been superseded.

This is in contrast to the simple overwriting of old data that a non-COW filesystem such as Ext3fs carries out.

A result of the COW design means that the old data can still be accessed with the right tools, and that rollback and modification become a natural possible feature.

“Cheap” revision control is thus possible.

Revision control for the filesystem is the idea that changes in the file system are tracked and can be rolled back as needed. “Cheap” here means that COW makes tracking changes convenient, take up very little space, and quick. For an administrator of Bright Cluster Manager, cheap revision control is interesting for the purpose of managing software images.

This is because for a non-COW filesystem such as Ext3fs, image variations take a large amount of space, even if the variation in the filesystem inside the image is very little. On the other hand, image variations in a COW filesystem such as Btrfs take up near-minimum space.

Thus, for example, the technique of using initialize and finalize scripts to generate such image variations on the fly (section 3.15.4) in order to save space, can be avoided by using a Btrfs partition to save the full image variations instead.

“Expensive” revision control on non-COW filesystems is also possible. It is merely not recommended, since each disk image takes up completely new blocks, and hence uses up more space. The administrator will then have to consider that the filesystem may become full much earlier. The degree of restraint on revision control caused by this, as well as the extra consumption of resources, means that revision control on non-COW filesystems is best implemented on test clusters only, rather than on production clusters.

11.4.2 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 is available from Red Hat as a technology preview release in versions 6 and 7 for now. That means it is not officially supported by Red Hat. The Btrfs version used in the 2.6 kernel series, used by RHEL6.x is quite old, which means that it may not be fully stable. Btrfs users who would like significantly more features and stability will prefer the 3.x kernel series, used by RHEL7.x. Btrfs has been deprecated since RHEL7.4.

Btrfs is also available from SUSE. SLES11SP2 and SLES11SP3 both use a 3.0 series kernel. 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 RHEL6 and SLES11, it is highly advisable to keep backups of important software images on a
11.4 Revision Control For Images

non-Btrfs filesystems when Btrfs is used on these Linux distributions.

11.4.3 Installing Btrfs To Work With Revision Control Of Images In Bright 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@bright73 ~]# yum install btrfs-progs
... Resolving Dependencies
---> Running transaction check
--- Package btrfs-progs.x86_64 0:0.20-0.2.git91d9eec.el6 will be installed
... Total download size: 297 k
Installed size: 2.4 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@bright73 ~]# cd /cm/
[root@bright73 cm]# mv images images2
[root@bright73 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@bright73 cm]# mkfs.btrfs /dev/sdc1
[root@bright73 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@bright73 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@bright73 cm]# mkfs.btrfs butter.img
WARNING! - Btrfs Btrfs v0.20-rc1 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using
fs created label (null) on butter.img
    nodesize 4096 leafsize 4096 sectorsize 4096 size 20.00GB
Btrfs Btrfs v0.20-rc1
[root@bright73 cm]# mount -t btrfs butter.img images -o loop
[root@bright73 cm]# mount ...
/cm/butter.img on /cm/images type btrfs (rw,loop=/dev/loop0)
```
Migrating Images With `cm-migrate-images`

The entries inside the `/cm/images/` directory are the software images (file trees) used to provision nodes.

Revision tracking of images in a Btrfs filesystem can be done by making the directory of a specific image a subvolume. Subvolumes in Btrfs are an extension of standard unix directories with versioning. The files in a subvolume are tracked with special internal Btrfs markers.

To have this kind of version tracking of images work, image migration cannot simply be done with a `cp -a` or a `mv` command. That is, moving images from the `images2` directory in the traditional filesystem, over to images in the `images` directory in the Btrfs filesystem command with the standard `cp` or `mv` command is not appropriate. This is because the images are to be tracked once they are in the Btrfs system, and are therefore not standard files any more, but instead extended, and made into subvolumes.

The migration for Bright 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@bright73 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@bright73 cm]# btrfs filesystem show /dev/loop0
Label: none    uuid: 6f94de75-2e8d-45d2-886b-f87326b73474
  Total devices 1 FS bytes used 3.42GB
devid    1 size 20.00GB used 6.04GB path /dev/loop0

Btrfs Btrfs v0.20-rc1
[root@bright73 cm]# btrfs subvolume list /cm/images
ID 257 gen 482 top level 5 path default-image
```

The filesystem can be modified as well as merely viewed with `btrfs`. However, instead of using the utility to modify image revisions directly, it is recommended that the administrator use Bright Cluster Manager to manage image version tracking, since the necessary functionality has been integrated into `cmsh` and `cmgui`.

11.4.4 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 @<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 11.4.3. 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**

```
[bright73->softwareimage]% newrevision default-image "some changes"
[bright73->softwareimage]% newrevision default-image "more changes"
[bright73->softwareimage]% revisions default-image
ID Date Description
----- ----------------------------- -----------
1 Wed, 07 May 2014 05:28:06 PDT some changes
2 Wed, 07 May 2014 05:28:55 PDT more changes
```

```
[bright73->softwareimage]% list -r
Name (key) Path Kernel version
---------------- -------------------------- --------------------------
default-image /cm/images/default-image 2.6.32-431.11.2.el6.x86_64
default-image@1 /cm/images/default-image-1 2.6.32-431.11.2.el6.x86_64
default-image@2 /cm/images/default-image-2 2.6.32-431.11.2.el6.x86_64
```

```
[bright73->softwareimage]% category add storage; commit
```

© Bright Computing, Inc.
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**

   ```bash
   [bright73->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**

   ```bash
   [bright73->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**

   ```bash
   [bright73->category] $ set storage softwareimage default-image@latest
   ```

   then nodes in the storage category use the image default-image@2 if nothing is set at node level. If a new revision of default-image is created later on, with a later tag (@3, @4, @5...) then the property takes the new value for the revision, so that nodes in the category will use the new revision as their image.

**Revision Control For Images—An Example Session**

This section uses a session to illustrate how image revision control is commonly used, with commentary along the way. It assumes a newly installed cluster with Btrfs configured and /cm/images migrated as explained in section 11.4.3.

First, a revision of the image is made to save its initial state:

```bash
[bright73->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:
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:

```
[bright73->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:

```
[bright73->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:

```
[bright73->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.

### 11.5 BIOS Configuration And Updates

Bright Cluster Manager includes a number of tools that can be used to configure and update the BIOS of nodes. All tools are located in the /cm/shared/apps/cmbios/nodebios directory on the head node. The remainder of this section assumes that this directory is the current working directory.

Due to the nature of BIOS updates, it is highly recommended that these tools are used with great care. Incorrect use may render nodes unusable.

Updating a BIOS of a node requires booting it from the network using a specially prepared DOS image. From the autoexec.bat file, one or multiple automated BIOS operations can be performed.

#### 11.5.1 BIOS Configuration

In order to configure the BIOS on a group of nodes, an administrator needs to manually configure the BIOS on a reference node using the conventional method of entering BIOS Setup mode at system boot time. After the BIOS has been configured, the machine needs to be booted as a node. The administrator may subsequently use the cmospull utility on the node to create a snapshot of the reference node’s NVRAM contents.

**Example**

```
ssh node001 /cm/shared/apps/cmbios/nodebios/cmospull > node001.nvram
```

After the NVRAM settings of the reference node have been saved to a file, the settings need to be copied to the generic DOS image so that they can be written to the NVRAM of the other nodes.

The generic DOS image is located in /cm/shared/apps/cmbios/nodebios/win98boot.img. It is generally a good idea to copy the generic image and make changes to the copy only.
Example

cp -a win98boot.img flash.img

To modify the image, it is first mounted:

```
mount -o loop flash.img /mnt
```

When the DOS image has been mounted, the utility that writes out the NVRAM data needs to be combined with the NVRAM data into a single DOS executable. This is done by appending the NVRAM data to the `cmosprog.bin` file. The result is a DOS `.COM` executable.

Example

```
cat cmosprog.bin node001.nvram > cmosprog.com
```

The generated `.COM` is then copied to the image and should be started from the `autoexec.bat` file. Note that DOS text files require a carriage return at the end of every line.

Example

```
cp cmosprog.com /mnt
/bin/echo -e "A:\cmosprog.com\r" >> /mnt/autoexec.bat
```

After making the necessary changes to the DOS image, it is unmounted:

```
umount /mnt
```

After preparing the DOS image, it is booted as described in section 11.5.3.

11.5.2 Updating BIOS

Upgrading the BIOS to a new version involves using the DOS tools that were supplied with the BIOS. Similar to the instructions above, the flash tool and the BIOS image must be copied to the 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 (section 11.5.1).

The next section describes how to boot the DOS image.

11.5.3 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.
11.6 Hardware Match Check

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@bright73 ~]# cmsh -c "category add newbunch; commit"
   [root@bright73 ~]# for i in {129..255}
   > do
   > cmsh -c "device; set node00$i category newbunch; commit"
   > done
   Successfully committed 1 Devices
   Successfully committed 1 Devices
   
   For larger clusters, reducing the time wasted in opening up cmsh can be done by replacing the offending `for` loop with a construction that is more elegant, but less familiar to most people:

   ```bash
   [root@bright73 ~]# for ((i=129; i<=255; i++)) do echo "
   > device
   > set node00$i category newbunch
   > commit"; done | 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 (Appendix G.2.1) as follows:

   ```bash
   [root@bright73 ~]# /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.

3. The frequency with which the health check should run in normal automated periodic use is set as follows (some prompt text elided):

   ```bash
   [root@bright73 ~]# cmsh
   [bright73]% monitoring setup healthconf newbunch
   [...->healthconf]% add hardware-profile
   [...->healthconf*{hardware-profile*}% set checkinterval 600; commit
   ```

4. The `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.

© Bright Computing, Inc.
11.7 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.

2. **Remote shell via CMDaemon.** This is possible if CMDaemon is running on the node and accessible via `cmgui` or `cmsh`.
   - For `cmgui`, for a node in the node settings resource, in the Tasks tabbed pane, clicking the **Root Shell** button launches an interactive **bash** session connected to the node via CMDaemon.
   - 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 `cmgui` (**Remote Console**) or from `cmsh` (**rconsole**).

   Item 3 in the preceding list, SOL access, is very useful for administrators, and is covered next more thoroughly with:
   - some background notes on serial over LAN console access (section 11.7.1)
   - the configuration of SOL with `cmgui` (section 11.7.2)
   - the configuration of SOL with `cmsh` (section 11.7.3)
   - the conman SOL logger and viewer (section 11.7.4)

### 11.7.1 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. It 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 Bright 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 11.7.2 and 11.7.3 explain how SOL access configuration is set up for the node kernel using cmgui 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 Bright Cluster Manager’s cmgui 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 Bright 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 (sections 11.7.2, 11.7.3). 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 Bright Cluster Manager, or vice versa.

### 11.7.2 SOL Console Configuration And Access With cmgui

In cmgui, within the “Software Images” resource, if the “Console over SOL” checkbox (figure 11.2) is checked for the software image that the node is using, 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. Some of the settings for SOL can be set from the same screen.

With SOL correctly configured, an SOL client to access the Linux serial console can be launched using the “Remote Console” button for the node (figure 11.3).
11.7 Serial Over LAN Console Access

11.7.3 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@bright73 ~]# cmsh
[bright73]% softwareimage use default-image

[bright73->softwareimage[default-image]]% set enablesol yes
[bright73->softwareimage*[default-image*]]% commit

The SOL settings for a particular image can be seen with the show command:

[bright73->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 the head node:

[root@bright73 ~]# cmsh
[bright73]% device use node001
[bright73->device[node001]]% rconsole

11.7.4 The conman Serial Console Logger And Viewer

In Bright Cluster Manager, the console viewer and logger service conman is used to connect to an SOL console and log the console output.

If "Console over SOL" in cmgui or enablesol in cmsh are 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 11.4).

![Figure 11.4: Physical Console Freeze During SOL Access](image)

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 11.5) during the freeze of the physical console display.
Finally, just before login is displayed, the physical console once more (figure 11.6) starts to display what is on the remote console (figure 11.7).
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>`. 
12 MIC Configuration

12.1 Introduction

The Intel Many Integrated Core (MIC) architecture combines many Intel CPU cores onto a single chip. Bright Cluster Manager supports the PCIe implementation of the Intel MIC architecture, called the Intel Xeon Phi. In this section the terms MIC, coprocessor, or (MIC) card, mean the Intel Xeon Phi, while a MIC host means a node where a MIC card is installed.

The MIC architecture, k1om, is quite similar to x86_64. Usually an application written for x86_64 CPU can simply be rebuilt using special compiler flags. Thus, Linux can run inside a coprocessor. Bright Cluster Manager provides special packages with names ending in -k1om, which provide executable binary files built to run only inside a MIC. Another set of packages with names beginning with intel-mic- is installed on the MIC host.

To install and configure MICs to work with Bright Cluster Manager, the administrator:

- installs the MIC software using YUM or zypper;
- does the initial configuration of a set of MICs in a cluster using cm-mic-setup or cmgui;

The following can then be done for the MIC, using cmgui or cmsh:

- tuning of coprocessor parameters. These parameters form the mic<INDEX>.conf (section 12.3.1) configuration file.
- monitoring of metrics/health checks of MIC cards.
- configuring workload manager queues for MICs just as for regular nodes. For now, only Slurm supports this feature.
- addition/removal of NFS mount points using cmgui or cmsh.
- enabling/disabling third party software overlays loaded during MIC booting.
- tuning the network interface parameters.

In the preceding list, the value of <INDEX> is a number 0,1,2,...,31, and is called the container index parameter (section 3) of the MIC host. It corresponds to the MIC identification number of the card on the host. Thus the first card on the host has an <INDEX> value of 0, and the card is then identified as mic0.

12.2 MIC Software Installation

In order for MIC cards to function properly on a host hardware, large BAR (Base Address Registers) support (MMIO addressing greater than 4GB) must be enabled. The platform and/or BIOS vendor should be contacted to check if changing this setting is applicable to the platform used.

© Bright Computing, Inc.
12.2.1 MIC Software Packages

As software for the MIC, Intel provides the Intel MIC Platform Software Stack (MPSS). This is made up of system software and libraries for low-level coprocessor management. Distribution-specific builds of the MPSS source packages are used by Bright Cluster Manager to manage MIC cards, and are picked up from the Bright Computing repositories. The Bright Cluster Manager administrator should not pick up the MPSS packages from Intel directly.

MIC Software Packages: Distribution Naming Convention

Linux distributions currently supported by Bright Cluster Manager are those based on RHEL 6.4 (CentOS 6.4 and Scientific Linux 6.4), RHEL 6.5, RHEL 6.6, RHEL 7.0, and SUSE Linux Enterprise Server 11 SP2 and SP3. The packages suffix, which is either -rhel6.4, -rhel6.5, -rhel6.6, -rhel7.0, -suse11.2, or -suse11.3, identifies the MIC package that is to be installed appropriately.

<table>
<thead>
<tr>
<th>Distribution Based On</th>
<th>Suffix On Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL 6.4</td>
<td>-rhel6.4</td>
</tr>
<tr>
<td>RHEL 6.5</td>
<td>-rhel6.5</td>
</tr>
<tr>
<td>RHEL 6.6</td>
<td>-rhel6.6</td>
</tr>
<tr>
<td>RHEL 7.0</td>
<td>-rhel7.0</td>
</tr>
<tr>
<td>SLES 11 SP2</td>
<td>-suse11.2</td>
</tr>
<tr>
<td>SLES 11 SP3</td>
<td>-suse11.3</td>
</tr>
</tbody>
</table>

For example, the MPSS runtime-related packages, together with other RPMs distributed with MPSS, are repackaged by Bright Cluster Manager as .rpm files, and have names that start:

Example

• intel-mic-runtime-rhel6.4
• intel-mic-runtime-rhel6.5
• intel-mic-runtime-rhel6.6
• intel-mic-runtime-rhel7.0
• intel-mic-runtime-suse11.2
• intel-mic-runtime-suse11.3

The suffix is omitted for convenience in this manual.

MIC Software Packages: Names

The full list of packages, without the distribution suffix tags, is:

• intel-mic-driver: Intel MIC driver, built during host boot;
• intel-mic-flash: Intel MIC flash files, to flash the cards (section 12.4);
• intel-mic-ofed: OFED drivers and libraries for MIC. The OFED drivers on the running systems are built during host boot, from source packages packed inside the intel-mic-ofed package. The source packages are:
  • intel-mic-ofed-kmod: Host-side MIC InfiniBand Drivers
  • intel-mic-ofed-libibscif: Intel SCIF Userspace Driver
  • intel-mic-ofed-ibpd: InfiniBand Proxy Daemon

© Bright Computing, Inc.
• **intel-mic-runtime**: Intel MIC runtime environment

• **intel-mic-sdk**: Intel MIC SDK. This includes the Intel MIC GNU tool chain to perform cross compilation (for x68_64 and for k1om), and the Intel MIC kernel development tools and sources.

• **intel-mic-perf**: Benchmarks for measuring the performance of the Intel MIC cards.

• **intel-mic-ldap**: Intel MIC LDAP files.

• **intel-mic-src**: MPSS source archives.

• **intel-mic-doc**: Intel MIC documentation files.

The minimum set of packages to be installed in a host software image—or on a head node, if coprocessors are installed on the head node—consists of **intel-mic-driver**, **intel-mic-ofed**, **intel-mic-runtime**, and **intel-mic-flash**.

The entire set of the packages can be installed on an RHEL-based operating system with a command in the following format:

```bash
yum install intel-mic-* <suffix> --installroot= <image>
```

For example:

```bash
yum install intel-mic-*-rhel6.6 --installroot= /cm/images/michost-image
```

For SLES the set of packages can be installed with a command in the following format:

```bash
zypper install intel-mic-* <suffix> -R <image>
```

If using non-Bright repository OFED stacks, then the following OFED stacks provide their own SCIF drivers:

• the QLogic OFED stack with a version greater than 7.3

• the OFED stack from OFA with version 3.5

These stacks conflict with the Bright-packaged versions. The Bright Computing **intel-mic-ofed** package described earlier on should be removed by the administrator, if one of these non-Bright repository OFED stacks is installed.

Bright Computing also provides a set of k1om pre-built packages, which are installed on the head node. They can be removed if Slurm is not used:

• **slurm-client-k1om**

• **munge-k1om**

An additional k1om pre-built package that is installed is **strace-k1om**. This can typically be used for debugging issues, and may be used by Bright Computing support when tracing problems.

Because the k1om packages are installed in `/cm/shared/`, they are available to each node in the cluster, as well as to the MIC cards. For now, only the Slurm Workload Manager can be started within the MIC in Bright Cluster Manager, and requires all of these k1om packages to be installed on the head node. The other workload managers that Bright Cluster Manager supports are supported with the MIC, but cannot be started from within the MIC.
12.2.2 MIC Environment MIC Commands

The `module` command (section 2.2) can be used to load the MIC environment variables as follows:

```
module add intel/mic/runtime
```

The following Intel MPSS commands, among others, can then be executed from the `intel-mic-runtime` and other `intel-mic-*` packages:

- **micinfo**: retrieves information about a MIC system configuration.
- **micflash**: updates the coprocessor flash, queries the flash version on the coprocessor, and saves a copy of the flash image loaded in the coprocessor to a directory on the host. If executed without a path to the firmware file or directory, then a compatible image is searched for inside `/usr/share/mpss/flash/`. If `-device all` is specified as an option, then the firmware of all MIC cards on a host will be updated. Available options are described in `man(1) micflash`.
- **micsmc**: monitors coprocessors.
- **micctrl**: controls and configures a MIC.
- **miccheck**: verifies a coprocessor system configuration by running a diagnostic test suite.
- **micnativeloadex**: copies a MIC native binary to a specified coprocessor and executes it.
- **micrasd**: logs hardware errors reported by MIC devices.

12.2.3 Bright Computing MIC Tools

The Bright Computing repository also provides:

- **cm-mic-setup**: This tool is provided by the `cluster-tools` package. This configures MICs in a Bright Cluster Manager enabled cluster, and is described further in section 12.3.1
- **cm-mic-postcode**: This tool is provided by the `cluster-tools-slave` package. This returns a human-readable description of the POST code returned by the coprocessor during power on and boot, thus identifying the stage that the card is at during the boot process. A hex representation of the MIC POST code can be found by reading the `/sys/class/mic/mic<INDEX>/post_code` file.

12.2.4 MIC OFED Installation

OFED communication can improve data throughput because there is then no additional copy into the host system memory. Direct data transfers can also take place on the SCIF (Symmetric Communications Interface) between the MIC OFED HCA driver on the host machine and the corresponding MIC OFED HCA driver on the MIC. For MICs that are on the same host, the MIC OFED kernel modules can be used to enable MIC-to-MIC RDMA communication over OFED.

There are 3 widely-used OFED software stack implementations: OFA, QLogic (now known as Intel True Scale), and Mellanox. The latest OFED versions for the HCAs used can be looked up in the Intel MPSS User Guide, available from https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss.

The release notes for the OFED stacks indicate which stack versions are supported by MICs. For example, version OFED-3.5.2-MIC supports the following distribution releases:

- RedHat EL6.2 2.6.32-220.el6
- RedHat EL6.3 2.6.32-279.el6
- RedHat EL6.4 2.6.32-358.el6
- RedHat EL6.5 2.6.32-431.el6
• RedHat EL7.0 3.10.0-123.el7 *
• SLES11 SP2 3.0.13-0.27-default
• SLES11 SP3 3.0.76-0.11-default *

Packages Installation
By default, the parent distribution version of the OFED stack is installed on Bright Cluster Manager. A QLogic (Intel True Scale) OFED stack or Mellanox OFED stack packaged by Bright Computing can also be used, and their installation on the host are described in section 7.6 of the Installation Manual. To use a MIC with OFED, however, requires the installation of a MIC-capable OFED stack, provided by the Intel intel-mic-ofed package listed previously (section 12.2.1).

To install the intel-mic-ofed package on a host requires that the kernel-ib-devel and kernel-ib OFED packages, if they are installed, must first be removed manually. They can be removed as follows:

• For a head node:
  `rpm -e kernel-ib-devel kernel-ib`

• For a regular node with image `<image>`:
  `rpm -e kernel-ib-devel kernel-ib --root=/cm/images/<image>`

After the MIC OFED stack has been installed, the host must be rebooted to implement it. By default the MIC OFED stack is compiled and installed on boot. It is also possible to build and install MIC OFED within a software image, by running the build service outside of MIC host boot, using the `install` argument:

`chroot <image> /etc/init.d/intel-mic-ofed install`

Such an installation may be useful when it is not possible to build drivers during MIC host boot. For example, when the node root filesystem is located on a shared filesystem, such as on NFS, and certain directories are read-only, then a build during MIC host boot fails. However, building outside of MIC host boot means that the installation must be repeated separately whenever the MIC host kernel changes.

Logs for the MIC OFED driver build are found at:

• `/var/log/intel-mic-ofed-driver.log`: This logs the `/var/log/intel-mic-ofed` script

• `/var/log/intel-mic-ofed-driver-build.log`: This logs the gcc compiler and rpmbuild utility messages

The ofed-mic Service
The ofed-mic service running on the MIC host serves the MIC OFED kernel modules. If one of the MICs belonging to that host is not in an UP state, then:

• The ofed-mic service will fail. That is, its return code will not be 0, and the ManagedServicesOK health check (page 659) response is FAIL.

• No attempt to start the ofed-mic service on a host is carried out by Bright Cluster Manager during this FAIL state.

© Bright Computing, Inc.
12.3 MIC Configuration

12.3.1 Using cm-mic-setup To Configure MICs

Introduction

The cm-mic-setup utility is used to set up the initial configuration of MIC hosts and cards in Bright Cluster Manager. Adding new cards is done interactively by default, or otherwise automatically.

- Everything done by cm-mic-setup can be done manually using cmsh. However, using cmsh is not recommended because it is easy for an administrator to forget something and make a mistake. Configuring MICs via cmsh is covered in section 12.3.2.
- A cmgui alternative to cm-mic-setup is to use the Create MICs wizard (page 494).

Running The Command

A command synopsis of cm-mic-setup is:

```
cm-mic-setup -a <MIC>... -m <MICHOSTNAME>... -n <MICNETWORK> [OPTIONS]
```

Here:

- `<MIC>` is the MIC card being added
- `<MICHOSTNAME>` is the MIC host, as specified according to the hostname range format
- `<MICNETWORK>` is a network that the MIC cards are in. Any new network can be set up for this. For a Type 1 or Type 2 network (section 3.3.6 of the Installation Manual), internalnet is the default.

The details of the cm-mic-setup command syntax and hostname range format are given in the manual page, man(8) cm-mic-setup.

When cm-mic-setup is run:

- In interactive mode, questions are asked about the configuration settings of particular card(s) and host(s) (IP addresses of host and card, bridge name and so on).
- In automatic mode, all parameters are configured by default, except that default values can be overwritten using extra options in the command line. The log file of all cm-mic-setup actions is kept in /var/log/cm-mic-setup.log.

When adding a new MIC, cm-mic-setup goes through these steps:

1. The new MIC device is added to Bright Cluster Manager
2. If it has not already been done
   - a network bridge interface is added to a MIC host interface
   - the management physical network interface is added to the bridge
   - the physical network interface IP address is assigned to the bridge
   - and the IP address of the physical network interface is cleared.
3. A range of free IP addresses can be assigned to the MIC card interfaces for a specific MIC host (mic0, mic1, mic2...). The ranges are calculated using
   - a specified network base address (x.y.0.0)
   - network base offset (typically 0.0.z.0, with z ≠ 0. The offset must be specified for netmasks other than a CIDR value of /16)
   - and stride (s).
The dotted quad representation of the range is then as indicated by the series:

\[ x.y.z.(s*n), x.y.z.(s*n+1), x.y.z.(s*n+2), \ldots x.y.z.(s*n+(s-1)) \]

\[ = x.y.z.(s*n), \ldots x.y.z.(s*(n+1)-1) \]

In the series, \( n \) is a number \((0,1,2,3\ldots)\) identified with the MIC host. The stride is thus the number of addresses in the range associated with that MIC host. The first line shows that the range starts here from \( s*n+0 \) and ends at \( s*n+(s-1) \) in the end quad of the IP address. The MIC interfaces are given these IP addresses by default. The stride is thus a way of regularizing the IP addresses of the MIC cards associated with a MIC host.

A worked example of this follows: If the specified network is the internal network, then its base address is \( 10.141.0.0 \) with a netmask of in CIDR notation of /16, and the network base offset defaults to \( 0.0.128.0 \). For a stride of \( 8 \), the IP address range that will be allocated for the second (i.e., \( n=1 \)) node is:

\[ x.y.z.(s*n) \ldots x.y.z(s*(n+1)-1) \]

\[ = 10.141.128.8 \ldots 10.141.128.15 \]

Here, \( 10.141.128.8 \) is the first MIC card IP address at the second node, and \( 10.141.128.9 \) is the next MIC card IP address at the second node.

If the stride is not specified, then a default stride is calculated as the maximum taken from the numbers of MICs per MIC host.

4. The \( \text{mic<INDEX>} \) physical network interface is added to the MIC host interfaces (i.e. on the regular nodes). The value of \( \text{<INDEX>} \) is typically the “number” of the MIC card on the host, starting with \( 0 \) for the first card, \( 1 \) for the second and so on. An example of adding a network interface is shown on page 68.

   [bright73->device[node001]->interfaces]% add physical mic0

5. The \( \text{mic0} \) physical network interface is added to the MIC card interfaces (i.e., to the MIC card nodes). The IP address is picked up from the previously calculated IP range to the interface in the earlier step 3.

   [bright73->device[node001-mic0]->interfaces% add physical mic0

6. Default filesystem mount points are added to the MIC device. These are \( /dev/pts \) for pts, \( /dev/shm \) for tempfs, \( /sys \) for sysfs, \( /proc \) for proc, and \( /cm/shared \) and \( /home \) for NFS.

7. Default overlays are added to the MIC. These are \( \text{cm-mounts} \) and a set of workload manager overlays. For Slurm the overlays \( \text{cm-munge}, \text{cm-munge-key} \) and \( \text{cm-slurm} \) are added.

8. The \( \text{MICHost} \) role is assigned to the MIC host.

   A reboot of the MIC hosts is prompted for in the interactive mode. The non-interactive mode does not prompt for a reboot, and does not carry out a reboot. A reboot of the MIC hosts is however needed after \( \text{cm-mic-setup} \) is run, for the MIC hosts to function correctly.
12.3.2 Using cmsh To Configure Some MIC Properties

A MIC is a first class device in Bright Cluster Manager, which means it is very similar to a regular cluster node, rather than like, for example, a relatively dumb switch. So, when a new card is added, a cluster administrator can manage the MIC card with cmsh like the other nodes in the cluster:

1. MIC network interfaces can be managed from the interfaces submode:

```
[bright73->device[node002-mic0]->interfaces]# list
Type     Network device name   IP          Network
--------- -------------------- ------------- -----------
physical mic0     10.141.128.4  internalnet
```

2. A MIC has a category type, MIC, associated with it. This is similar to nodes, which have a category type of node. MIC categories are analogous to node categories (section 2.1.3), and are therefore accessed from the category mode of cmsh. The idea behind a MIC category is essentially the same as that for a node category: to manage configuration settings in the cluster in a single place (i.e. in cmgui or cmsh) for a group of items of a particular type (i.e. nodes or MIC cards).

By default, the mic-default MIC category is created when the cm-mic-setup script or the MIC wizard from cmgui is run for the first time. Within the mic-default category are defined, among others:

- the default filesystem mounts (fsmounts)
- MIC settings (micsettings)
  - overlays (overlay)
  - powermanagement (powermanagement)
- roles (roles).

Just as for regular node categories, properties are inherited by default from the category. Thus, if some property is set in micsettings, then this property is used by default in the particular MIC item unless it is overwritten.

3. MIC-specific options can be tuned in the micsettings submode of the device mode or category mode:

```
[bright73->category[mic-default]->micsettings]# show
Parameter                      Value
----------------------------------------------
Boot On Start                   yes
Cgroup Memory                   disabled
Console                         hvc0
Crash Dump Dir                  /var/crash/mic/
Crash Dump Limit                16
Extra Command Line              <4 in submode>
Overlays                        cpufreq_on;corec6_on;pc3_on;pc6_on
Power Management                /var/mpss/mic${index}.image.gz
Root Device                     /var/mpss/mic${index}.image.gz
Root Device Type                ramfs
Shutdown Timeout                300
Type                            XeonPhiSettings
User Authentication Type        merge
Verbose Logging                 yes
Base                            /usr/share/mpss/boot/initramfs-knightscorner.cpio.gz
Common Dir                      /var/mpss/common
```
12.3 MIC Configuration

Common Dir File List
Mic Dir /var/mpss/mic${index}
Mic Dir File List
OS Image Address Map /usr/share/mpss/boot/System.map-knightscorner
OS Image Kernel /usr/share/mpss/boot/bzImage-knightscorner
Type Of Base CPIO

In the preceding listing, the text `${index}` is the container index parameter of the MIC node (section 12.1). The text is replaced by the value of when the MIC identification number is set for the card on the host.

Example

[bright73]% device use node001-mic0
[bright73->device[node001-mic0]]% get containerindex
0

4. Overlays which are used to enable and disable third party software in the MIC can be changed from the overlays submode of the micsettings submode.

5. Mount points can be configured just like for regular node mount points, but with one exception: If a directory is to be mounted from a MIC host to its own MIC, then the $michost variable can be used. On MIC boot this variable is substituted with the hostname of the MIC host.

All other MIC node parameters are similar to the parameters of a regular node.

12.3.3 Using cmgui To Configure Some MIC Properties

The cmgui front end treats a MIC very like a regular node. The differences are highlighted in the following:

- The Overview tabbed pane is the default view if the MIC name is clicked upon in the resource tree. The Overview information for a MIC is displayed similar to that for regular nodes. The Running Jobs pane for the MIC lists only jobs executed inside the MIC (that is, for k1om applications, explained in section 12.5.2). Jobs executed on the MIC host, but offloaded to the MIC (offloaded applications, explained in section 12.5.2), are not shown here, but are instead listed in the Running Jobs pane of the MIC host.

- The MIC Tasks tab (figure 12.1) lets the administrator carry out some of the regular tasks operations for a node. This includes power cycling operations, adding or removing the MIC from node groups, draining and undraining (Appendix G.3.1) the device, and accessing the MIC card via ssh or the serial console using minicom running on the host.

![Figure 12.1: cmgui: The MIC Tasks Tab](image)

© Bright Computing, Inc.
The Settings tab (figure 12.2) allows configuration of common node parameters. MIC index here is number of the coprocessor, and should be unique for the host. This is the same as the value of `<INDEX>`, which was introduced in section 12.1 as the MIC identification number set for the card on the host.

![cmgui: The Settings Tab For A MIC](image)

The MIC Settings tab displays MIC-specific parameters (figure 12.3).
When the parameters of this tab are saved, then `/etc/mpss/mic<INDEX>.conf` is updated on the host and the MIC device marked with a restart-required flag. The corresponding message is: “configuration has changed”. CMDaemon never restarts the MPSS service or individual MICs automatically, so to apply changes an administrator must reboot the coprocessor using “power reset” in cmsh, or the Reboot button within the Tasks tab of the MIC in cmgui. The rebooting of a MIC card can take several minutes. During the reboot, the state of the card is DOWN and it is inaccessible using ssh. The reboot process can also be monitored from the command line by following the MPSS daemon log file, `/var/log/mpssd`, on the MIC host.

- The MIC Overlay tab (figure 12.4) allows overlays to be configured, so that third party software can be enabled or disabled.
  
  The source and target entries meanings depend on the overlay type. Overlays and their parameters are explained in greater detail in section 12.3.4.

- The FS Mounts tab allows the mounting of filesystems to the MIC. To apply changes, the MIC card should be rebooted.

- Instead of using the `cm-mic-setup` script (section 12.3.1) to add new pre-configured MIC cards it is also possible to use the cmgui MIC wizard. The wizard is launched by clicking on the "Create
MICs" button in the Overview tab that is displayed when the MIC Nodes resource is selected (figure 12.5).

1. When the "Create MICs" button is clicked, the administrator can select a range of hosts on which the MIC cards are installed. The administrator can also set the name and range of the MIC suffixes to be used for the MICs. The default pattern is mic<INDEX>. A MIC hostname template is suggested using a the MIC host hostnames and MIC card names as parameters.
2. Clicking on the Next button brings the administrator to the second step of the wizard. This allows the administrator to set the host network bridge interface, the network base offset, and stride. A default stride value is calculated as the maximum taken from the numbers of MICs per MIC host. The IP addresses used for the MIC devices are listed. The list of IP addresses in this window change dynamically as changes are made to the base offset or stride.

![MIC Creation Wizard](image1)

Figure 12.7: cmgui: MIC Creation Wizard: Suggested IP Addresses Screen

3. Clicking on the "Next" button brings up the final window. This window shows warnings. If the changes are acceptable, then the Finish button can be clicked. The MICs are then added to Bright Cluster Manager.

### 12.3.4 Using MIC Overlays To Place Software On The MIC

A MIC overlay is used to make software available from a MIC host to a MIC. Usually this is software that is heavily accessed during a code run, such as drivers or parts of applications where access speed is a priority. It can also be software that can only be executed on the MIC, such as init scripts. The memory available to the MIC (8 to 16 GB at the time of writing in December 2014) is much lower than that of a regular node, so that large amounts of data cannot be placed in it. A relatively much slower remote-mounted filesystem is normally what is used instead for access to larger amounts of data.

Ignoring mounts, the MIC filesystem on the MIC is built up in several layers, starting with the fundamental binaries layer, adding some more required layers, and ending with a software overlay that the administrator installs for the end user applications. Each layer on the MIC is configured by using a "layer" source. The source can be:

- a directory
- individual files
- files packed in RPMs

The source depends on the overlay type, on the MIC host, as is described further on. The source used is assigned a corresponding source configuration parameter. The parameter is one of Base, CommonDir, MicDir, and Overlay. Each of these parameters has a default setting, but if needed, the administrator can modify them via the CMDaemon front ends, cmsh and cmgui.

Each overlay has three important parameters: type, source and target. These determine what type of file or files are transferred from a specified source, to a target on the MIC file system. The overlay type parameter can be set to the following values:

- **file** – setting this means that a single k1om RPM file in the directory specified by source is to be copied over to the MIC file system. The file is placed in the /RPMs-to-install directory on the MIC file system. When the MIC card boots, the init process tries to install the RPM within the MIC. No target type can be set for file.
filelist – setting this means that files from the directory specified by source are to be copied over to the MIC file system from the directory that source is set to. The files are placed in the directory specified by target.

rpm – setting this means that a single file specified in source is to be installed, not copied, over to the MIC file system. No target type can be set for rpm.

simple – setting this means that all files found under the directory specified by source are to be copied over to the MIC file system. The file is placed in the directory specified by target and the owner and permission are configured to match the files on the host.

In the case of filelist, each target file, typically given a name ending in .filelist, describes the files, directories, links, that are to be copied over from the layer directory on the MIC host, and laid out in the MIC filesystem on the MIC. The filelist content also includes nods. Nods (with a nod to the mknod(1) command) are used to set, for special devices on the MIC, such as /dev/console:

- the device type: character c or block, b
- the major and minor values
- perms, uid and gid.

The canonical details on these and other parameters at the time of writing (December 2014) can be found in Intel MPSS User Guide, provided with each MPSS release: Intel Xeon Phi Coprocessor Many-core Platform Software Stack (MPSS) Boot Configuration Guide. In revision 3.4 of that guide the details are found in section 16.1: The File System Creation Process.

The layers are laid out in the MIC according to the following sequence of directory configuration parameters:

1. Base: This is a MIC host path to a zipped CPIO archive that contains the basic MIC Coprocessor binaries that were installed during RPM installation.

2. CommonDir: This MIC host directory contains a common filesystem structure that does not change during use of the MIC for compute purposes. Files in this directory are not wiped out during MPSS installation updates.

3. MicDir: This MIC host directory contains per MIC information files. Most of these files are created during MIC configuration. These values do not change if the MIC hardware does not change. However restarting the mpssd daemon will reconfigure the values, so changing them by hand is not advisable.

4. Overlay: Depending on its type, this can be a file, RPM or MIC host directory. It can contain additional sets of files that may be laid over the MIC filesystem that has been built in the layers so far. More than one Overlay parameter can be assigned by an administrator, who can thus use MIC overlays to make application software available to the MIC.

- The default assignment is named cm-mounts. The associated directory and filelist are configured with default values when cm-mic-setup (section 12.3.1, step 7) or the Create MICs wizard (page 494) is used. A MIC using these defaults creates the default mount points for the NFS directories.

- Slurm is offered as a configuration choice when cm-mic-setup or Create MICs are run. If selected, then the names cm-munge, cm-munge-key and cm-slurm are also assigned to the Overlay parameter, and their associated directories and filelists are configured with default values too. Among others, these overlays place slurm and munge init scripts, the munge key file, and also the configuration file gres.conf, inside the MICs.
These overlays have the following default directory and filelist settings:

```
[bright73->category[mic-default]->micsettings->overlay]% list -f name:12,source:47
name (key) source
------------ -----------------------------------------------
cm-mounts /var/mpss/cm-mounts-mic${index}
cm-slurm /cm/shared/apps/slurm/current/k1om-arch
cm-munge /cm/shared/apps/munge/current/k1om-arch
cm-munge-key /etc/munge/munge.key
```

```
[bright73->category[mic-default]->micsettings->overlay]% list -f name:12,target:54
name (key) target
------------ -----------------------------------------------
cm-mounts /var/mpss/cm-mounts-mic${index}.filelist
cm-slurm /cm/shared/apps/slurm/current/k1om-arch/slurm.filelist
cm-munge /cm/shared/apps/munge/current/k1om-arch/munge.filelist
cm-munge-key /etc/munge/munge.key
```

```
[micdev->category[mic-default]->micsettings->overlay]% list -f name:12,type:10
name (key) type
------------ ----------
cm-mounts filelist
cm-slurm filelist
cm-munge filelist
cm-munge-key file
```

The administrator generally only needs to consider the overlay layer(s), because MPSS and Bright Cluster Manager configuration takes care of the other “OS-like” layers with sensible default values. Thus, adding a MIC application as an overlay controlled by Bright Cluster Manager can be done, for example, according to the following procedure:

**Example**

1. A new layer directory can be created on the MIC host:
   ```
   mkdir /cm/local/apps/myapp/k1om/
   ```

2. Files, directories, and links for the MIC application can then be placed inside the layer directory:
   ```
   cp -r k1omapplication /cm/local/apps/myapp/k1om/
   ```

3. A descriptor file, `myapp.filelist`, can be constructed according to the filelist syntax, and placed in the layer directory’s topmost level.

4. A new overlay object is created via `cmsh` or `cmgui`, with the `Overlay` directory configuration parameter set to `/cm/local/apps/myapp/k1om/`, and the filelist descriptor set to `/cm/local/apps/myapp/k1om/myapp.filelist`

5. On restarting the MIC, the MIC application becomes available on the MIC.

**12.4 Microarchitecture Computing Flash Updates**

After all the necessary `intel-mic-*` and `*-k1om` packages have been installed for the first time, the hosts should be rebooted to make the MIC commands available for use. The card flash images should then be updated, and the hosts rebooted after that too.

The flash update procedure on, for example, two MIC hosts `node001` and `node002`, can be carried out as follows:
Backslashes in the following 5-step procedure indicate line splits, which means that any commands running across such line splits should be input as one long line:

1. The MPSS service on the MIC hosts is stopped:

   ```
   [root@bright73 ~]# cmsh
   [bright73] device
   [bright73->device]% foreach -n node001..node002 (services; stop mpss)
   ... 
   ```

2. Using the `micctrl` command with appropriate options, the `ready` state is awaited (-w) on each MIC host after a forced (-f) reset (-r) of the host:

   ```
   [bright73->device]% pexec -n node001..node002 "module load intel/mic/runtime && micctrl -r -f -w"
   
   [node001] :
   mic0: ready
   mic1: ready
   
   [node002] :
   mic0: ready
   mic1: ready
   ```

   For the sake of clarity: The reset option brings a MIC to a `ready` state (ready for boot), which means it is `DOWN` in Bright Cluster Manager terminology. The `-h` option shows an options help text for the `micctrl` command.

3. The firmware can then be updated. Some output has been elided for brevity in the listed output:

   ```
   [bright73->device]% pexec -n node001..node002 "module load intel/mic/runtime\ 
   && micflash -v -update -noreboot -device all"
   
   [node001]
   No image path specified - Searching: /usr/share/mpss/flash
   mic0: Flash image: /usr/share/mpss/flash/EXT_HP2_C0_0390-02.rom.smc
   mic1: Flash image: /usr/share/mpss/flash/EXT_HP2_C0_0390-02.rom.smc
   
   mic0: Updating flash: 10%
   mic1: Updating flash: 10%
   
   mic0: Updating flash: 40%
   mic1: Updating flash: 40%
   
   mic0: Updating flash: 56%
   mic1: Updating flash: 56%
   
   mic0: Updating SMC: 0%
   mic1: Updating SMC: 0%
   
   mic0: Updating SMC: 10%
   mic1: Updating SMC: 10%
   
   mic0: Updating SMC: 22%
   mic1: Updating SMC: 22%
   ```

© Bright Computing, Inc.
12.5 Other MIC Administrative Tasks

After the procedures of installing and configuring the MPSS software, and re-flashing the MIC, the Bright Cluster Manager is in an understandably disturbed state. It should be allowed about a 10 to 15 minute period to settle down, so that it can start populating its metrics and health checks with sensible values. Warnings during the settling down period can be ignored. After the system has settled down to a steady state, any persistent warnings can be taken more seriously.
12.5.1 How CMDaemon Manages MIC Cards

When the michost role is assigned to a host, CMDaemon starts and monitors these services:

- **mpss services.** This is an Intel Manycore Platform Software Stack (Intel MPSS) service which is runs the mpssd daemon. The version of MPSS that Bright Cluster Manager currently (January 2015) supports is 3.4.x.

  The mpssd daemon controls the initialization and booting of coprocessor cards based on a set of configuration files located at /etc/mpss/. The daemon is started and stopped as an operating system service and instructs the cards to boot or shutdown. It supplies the final filesystem image to the cards when requested. Further details about the mpssd daemon and its configuration files are given in the *Intel Xeon Phi Coprocessor Many-core Platform Software Stack (MPSS) Getting Started Guide*.

- **The ofed-mic service.** This provides the OFED interface between a coprocessor and ofed drivers running on its host.

Near the end of a successful boot of the coprocessor, CMDaemon executes the finalize script /cm/local/apps/cmd/scripts/finalize-mic for the MIC. The finalize script prepares the card to be used by users: It clears the IP address of the MIC interface, adds the MIC interface to the network bridge interface, adds the workload manager user (if necessary), and finally executes the prepare-mic script on the card via ssh.

  The prepare-mic script mounts NFS shared directories, starts workload manager daemons, and makes sure that coi_daemon is running with micuser user permissions. The script also forbids all regular users from accessing the MIC via SSH, apart from the user running a job via the job prolog, or a job running as micuser. The root user always has access to the MIC card.

12.5.2 Using Workload Managers With MIC

After a new MIC card is added and the michost role is assigned to its host, CMDaemon can configure the workload manager that is currently in use to use this card as follows:

- For Slurm:
  - The MIC is added to the workload manager as a new computing node.
  - The MIC is added to job queues if the workload manager client role is assigned to the MIC in cmsh or cmgui.
  - Node properties michost and miccard are assigned to the MIC host and the MIC card correspondingly.
  - A special prolog script is enabled for the host, if it has not already been enabled. The prolog script is at /cm/local/apps/cmd/scripts/prolog-michost.

- For all workload managers:
  - A new generic resource type mic is added, and the workload manager daemon on the host is configured to be aware about new available generic resources.

Bright Cluster Manager supports two main use case scenarios for coprocessors:

1. **Native k1om application.**
   
   This currently only works with Slurm.
   
   The user’s application is built to be used on k1om CPU architecture and executed inside MIC card. In this case the user should request nodes with the property miccard on its job submission. The workload manager processes this job just like a regular job executed inside the regular node.
12.5 Other MIC Administrative Tasks

Before the job is executed, the MIC hosts prolog script (/cm/local/apps/cmd/scripts/prolog-michost) is invoked on all hosts which are used by the job, in order to prepare MIC cards for the job.

2. Code offloading by the application.

To carry out code offloading, the user’s application is created to be executed on an x86_64 host, but uses special instructions to offload part of its code to one or more coprocessors installed on the host locally. The user needs to request the michost property or a particular number of MIC generic resources for each host during its job submission.

The application is executed on specified hosts and the workload manager sets an OFFLOAD_DEVICES environment variable, which controls the selection of MICs available to a job’s offloading code.

The offloaded code is executed with the user’s permissions inside the MICs. If no MICs are configured on the host, the OFFLOAD_DEVICES variable is set to -1. This causes the code to ignore the offload directives and run its routines on the host’s CPU(s).

Before the job is executed, the MIC prolog script (/cm/local/apps/cmd/scripts/prolog-mic) is invoked on MIC to prepare the card for the job.

In both cases, prolog scripts eventually execute the prepare-mic script (/cm/local/apps/cmd/scripts/prepare-mic), located on the MIC (the same script as executed within finalize script). This script prepares the card for use by a particular user.

12.5.3 Mounting The Root Filesystem For A MIC Over NFS

In section 3.10.4, the configuration of a diskless node with its root filesystem provided over NFS was described.

This section (section 12.5.3), assumes that the steps described in section 3.10.4 have been carried out, so that a diskless node is now up and running. If a MIC is installed on such a diskless host, then this host, which is now a MIC host, can be configured so that NFS provides it with a root filesystem, as follows:

1. The MIC and OFED MIC drivers should be built and installed in an appropriate software image. This is to reduce the number of directories mounted to the tmpfs filesystem.
   
   The drivers can be built and installed in a software image called root-over-nfs as follows:

   Example

   ```
   [root@bright73 ~]# chroot /cm/images/root-over-nfs
   [root@bright73 /]# chkconfig --del intel-mic-driver
   [root@bright73 /]# chkconfig --del intel-mic-ofed
   [root@bright73 /]# /etc/init.d/intel-mic-driver install
   Building and installing mic driver [ OK ]
   [root@bright73 /]# /etc/init.d/intel-mic-ofed install
   Building and installing intel-mic-ofed-kmod [ OK ]
   Building and installing intel-mic-ofed-ibpd [ OK ]
   Building and installing intel-mic-ofed-libibscif [ OK ]
   
   The intel-mic-ofed command should not be run with a start argument when the drivers are installed inside the software image. This is because the script would then restart the openibd service.
   
   Log entries for the install process can be seen under /var/log, in intel-mic-driver.log, intel-mic-ofed-driver.log, and intel-mic-ofed-driver-build.log.
   ```
2. The new filesystem mount point, accessible under the `fsmounts` submode, should be set to point to the appropriate node category. If the node category name is `root-over-nfs`, then the configuration can be done using `cmsh` as follows:

**Example**

```bash
[root@bright73 ~]# cmsh
[bright73]% category fsmounts root-over-nfs
[bright73->category[root-over-nfs]->fsmounts]% add /var/mpss
[bright73->/var/mpss*]% set device tmpfs
[bright73->/var/mpss*]% set filesystem tmpfs
[bright73->/var/mpss*]% commit
```

3. The MIC host must be rebooted to implement the changes.

**12.5.4 MIC Metrics**

Metrics are listed in Appendix G.1.1. The use of the metric collection `mic` (page 652) means that MIC metrics with working sensors are able to monitor MIC-related values.

The exact metrics that can be monitored vary according to the MIC hardware.

**12.5.5 User Management On The MIC**

A current user in the context of MIC use is a user that starts a job via a workload manager. By default, a current user is added to the `micuser` group within the MIC, in `/etc/group`, and the value:

```
 AllowGroups root micuser
```

is appended to `/etc/ssh/sshd_config` within the MIC at the same time. This then allows only users from `micuser` group and the `root` user to have ssh access to the MIC during job execution.

The default user management behavior can be altered in `cmsh` and `cmgui` by adding the following parameters to `mic<N>.conf`:

- **UserAuthenticationType**: user authentication type used for a MIC. Possible values: Local, None. Default: Local
- **UserAuthenticationLowUID**: lowest ID of a user which will be added. Possible values: any 32-bit positive number. Default: 500
- **UserAuthenticationHighUID**: highest ID of a user which will be added. Possible values: any 32-bit positive number. Default: 65000

Users with user IDs in the range `UserAuthenticationLowUID`–`UserAuthenticationHighUID` will be added to the MIC. Although any 32-bit user ID may be entered, a value of 500 or more is recommended for the lowest value. If `UserAuthenticationType=None` is specified, the `/etc/passwd` file on the card will default to one containing the `root`, `sshd`, and `micuser` accounts.

In `/etc/mpss/cm-mpss.conf`, the parameters that are used by MIC related scripts provided by Bright Cluster Manager are:

- **SSHD_ALLOW_GROUPS**: indicates what will be set as the value to AllowGroups in `/etc/ssh/sshd_config` within all MICs of the host. Possible values are:
- *any quoted string*: Default: "root micuser"
  - *UNUSED*: This value means that
    * AllowGroups will not be appended in /etc/ssh/sshd_config
    * the current user will not be added to the micuser group
    * all users can ssh into the MIC even if they have no job running on it.

- **RESTART_COI_DAEMON**: indicates if coi_daemon will be restarted with the current user permissions. If the value is **YES**, then coi_daemon restarts when the MIC is booted, and also each time when prolog-mic script is executed on MIC or the prolog-michost script is executed on the host. Default: **YES**

If /etc/mpss/cm-mpss.conf is changed, then the MICs should be restarted to apply the changes.
13
High Availability

13.0  Introduction

13.0.1  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.

13.0.2  High Availability Is Possible On Head Nodes, And Also On Regular Nodes
The high availability (HA) feature of Bright 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.

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. For regular nodes, HA is done differently compared with head 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 that is being discussed.

13.0.3  High Availability Usually Uses Shared Storage
HA is typically configured using shared storage (section 13.1.5), such as from an NFS service, which typically provides the /home directory on the active (section 13.1.1) head, and on the regular nodes.

13.0.4  Organization Of This Chapter
The remaining sections of this chapter are organized as follows:

• HA On Head Nodes
  – Section 13.1 describes the concepts behind HA, keeping the Bright Cluster Manager configuration in mind.
  – Section 13.2 describes the normal user-interactive way in which the Bright Cluster Manager implementation of a failover setup is configured.
  – Section 13.3 describes the implementation of the Bright Cluster Manager failover setup in a less user-interactive way, which avoids using the Ncurses dialogs of section 13.2
  – Section 13.4 describes how HA is managed with Bright Cluster Manager after it has been set up.

• HA On Regular Nodes
  – Section 13.5 describes the concepts behind HA for regular nodes, and how to configure HA for them.

© Bright Computing, Inc.
• HA And Workload Manager Jobs
  – Section 13.6 describes the support for workload manager job continuation during HA failover.

13.1 HA Concepts

13.1.1 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.

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.

13.1.2 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.

13.1.3 Services In Bright 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 Bright 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**

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, while if using cmgui, it is done from the Software Images resource, then selecting the Provisioning Status tab, and clicking on the Update Provisioning Nodes button (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 (e.g. SGE, Torque/Maui)

13.1.4 Failover Network Topology
A two-head failover network layout is illustrated in figure 13.1.
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 3.3.6 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 13.1. 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.

**13.1.5 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 Bright Cluster Manager gives the administrator full flexibility on how shared storage is implemented between two head nodes, there are generally three types of storage used: NAS, DAS, and DRBD.

**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.

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 that is usually 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.
DRBD
In a setup with DRBD (Distributed Replicated Block Device), both head nodes mirror a physical block device on each node device over a network interface. This results in a virtual shared DRBD block device. A DRBD block device is effectively a DAS block device simulated via a network. DRBD is a cost-effective solution for implementing shared storage in an HA setup. While a DRBD device itself is not configured by the cluster manager, a DRBD device that is already configured is recreated by the cluster manager. Recreating means that data on the DRBD device is wiped.

The use of DRBD is not recommended by Bright Cluster Manager, and RHEL7 and CentOS7 no longer provide packages for it.

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 cmgui (section 13.4.6).

13.1.6 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 13.1.7. 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 13.2.4).

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 Bright 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.

**13.1.7 Automatic Vs Manual Failover**
Administrators have a choice between creating an HA setup with automatic or manual failover. In case of 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 case of 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, automatic failover is attempted by default. However, it is possible to disable automatic failover. In cmsh this is done by setting the `disableautomaticfailover` property, which is a part of the HA-related parameters (section 13.4.6):

```
[root@bright73 ~]# cmsh
[bright73]% partition failover base
[bright73->partition[base]->failover]% set disableautomaticfailover yes
[bright73->partition[base]->failover]% commit
```
With cmgui it is done by selecting the cluster resource, then selecting the Failover tab. Within the tab, the Disable automatic failover checkbox is ticked, and the change saved with a click on the Save button (figure 13.2).

![Figure 13.2: cmgui High Availability: Disable Automatic Failover](image)

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.

In addition, if automatic failover is enabled, but the active head is still slightly responsive (the so-called mostly dead state, described in section 13.4.2), then the failover sequence must also be initiated manually by the administrator.

### 13.1.8 HA And Cloud Nodes

As far as the administrator is concerned, HA setup remains the same whether a Cluster Extension is configured or not, and whether a Cluster On Demand is configured or not. Behind the scenes, on failover, any networks associated with the cloud requirements are taken care of by Bright Cluster Manager.

### 13.2 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 Bright 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 /cm/local/apps/cmd/etc/cert.{pem,key} 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 Bright Cluster Manager reseller, and then be activated and installed (Chapter 4 of the Installation Manual).
13.2 HA Setup Procedure Using cmha-setup

Existing User Certificates Become Invalid
Installing the new license means that any existing user certificates will lose their validity (page 66 of the Installation Manual) on cmgui session logout. This means:

- If LDAP is managed by Bright Cluster Manager, then on logout, new user certificates are generated, and a new cmgui login session picks up the new certificates automatically.
- For LDAPs other than that of Bright 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 cmgui 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 13.2.1): the configuration parameters are set for the shared interface and for the secondary head node that is about to be installed.
2. Cloning (section 13.2.2): the secondary head node is installed by cloning it from the primary head node.
3. Shared Storage Setup (section 13.2.3): the method for shared storage is chosen and set up.

An optional extra stage is:

4. Automated Failover Setup (section 13.2.4): Power control to allow automated failover is set up.

13.2.1 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 13.2.1-13.2.3, and possibly section 13.2.4) are complete.
1. To start the HA setup, the cmha-setup command is run from a root shell on the primary head node.
2. Setup is selected from the main menu (figure 13.3).
3. Configure is selected from the Setup menu.
4. 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 Bright Cluster Manager reseller, and activated (section 4.3 of the Installation Manual).
5. The virtual shared internal alias interface name and virtual shared internal IP alias address are set.
6. The virtual shared external alias interface name and virtual shared external IP alias address are set. To use DHCP assignments on external interfaces, 0.0.0.0 can be used.
7. The host name of the passive is set.
8. 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.

9. Failover network interfaces have their name and IP address set for the active and passive nodes.

10. 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.

11. The network interfaces of the passive head node are reviewed and can be adjusted as required. DHCP assignments on external interfaces can be set by using an IP address of 0.0.0.0.

12. A summary screen displays the planned failover configuration. If alterations need to be made, they can be done via the next step.

13. The administrator is prompted to set the planned failover configuration. If it is not set, the main menu of cmha-setup is re-displayed.

14. 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.

15. Setup progress for the planned configuration is displayed (figure 13.4).

16. Instructions on what to run on the secondary to clone it from the primary are displayed (figure 13.5).

13.2.2 Failover Cloning
There are actually two kinds of cloning possible. Both types use the /cm/cm-clone-install command described further on in this section. The two cloning types are:

- **Failover cloning**: 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.
13.2 HA Setup Procedure Using cmha-setup

Figure 13.4: cmha-setup Setup Progress For Planned Configuration

Figure 13.5: cmha-setup Instructions To Run On Secondary For Cloning

- **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) the head node.

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 13.4.8.

After the preparation has been done by configuring parameters as outlined in section 13.2.1, 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 13.4.8). In that case the active head node can be either the primary or secondary.

© Bright Computing, Inc.
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, 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 13.6).

```
Creating failover/clone nodes:
  # /cm/cm-clone-install --failover
  # /cm/cm-clone-install --clone --hostname=new-hostname [--reboot]
  # /cm/cm-clone-install --failover [--reboot]

Other useful commands:
  # pmenu "Menu frontend to programs!"
  # dhcpcd dhcpd "Setup wired network connection!"
  # wificonfig "Setup wireless network connection!"
  # msetup "Setup mail and news!"
  # lynx (or) links "WWW browsers!" # rlogin (or) srln "Newsreaders!"

You can use 'backup-mbr' to backup/restore the MBR.

login: root
```

Figure 13.6: Login Screen After Booting Passive Into Rescue Mode From Active

4. The following command is executed (figure 13.7):

```
/cm/cm-clone-install --failover
```

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.
13.2 HA Setup Procedure Using cmha-setup

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 13.2.3) 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@bright73 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright73+ -> master2
  backupting [ OK ]
  mysql [ OK ]
  ping [ OK ]
  status [ OK ]
master2 -> bright73+
  backupting [ OK ]
  mysql [ OK ]
  ping [ OK ]
  status [ OK ]
```

© Bright Computing, Inc.
Here, the mysql, ping and status states indicate that HA setup completed successfully. The backupping (backup ping) state uses the dedicated failover network route for its checks, and starts working as soon as the passive head node has been rebooted.

### 13.2.3 Shared Storage Setup

After cloning the head node (section 13.2.2), the last basic stage of creating an HA setup is setting up shared storage. The available shared storage forms are NAS and DAS. The use of DRBD is not recommended by Bright Cluster Manager, but is still available as a form of shared storage at the time of writing (January 2016), for the distributions that still provide it.

**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 13.1.5. 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 510).

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.

**DRBD**

If a DRBD device is used that already has data on it, then it should be backed up by the administrator,
and restored after the `cmha-setup` procedure is complete. This is because the device content is erased
by re-creation of the device partitions that takes place during the procedure. The procedure is carried
out as follows:

1. In the `cmha-setup` main menu, the “Shared Storage” option is selected.

2. DRBD is selected.

3. The parts of the filesystem that should be placed on DRBD filesystems are selected.

4. The host names of the primary and secondary head nodes and the physical disk partitions to use
on both head nodes are entered.

5. That the contents of the listed partitions can be erased on both head nodes is confirmed. Af-
fer DRBD based filesystems have been created, the current contents of the shared directories are
copied onto the DRBD based filesystems and the DRBD based filesystems are mounted over the
old non-shared filesystems.

6. Once the setup process has completed, “DRBD Status/Overview” is selected to verify the status
of the DRBD block devices.

**13.2.4 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 func-
tional:

**Example**

```
[master1]# device power status -n mycluster1,mycluster2
apc03:21 ............ [ ON ] mycluster1
apc04:18 ............ [ ON ] mycluster2
```
Testing The BMC Interface Is Working

If a BMC (Baseboard Management Controller, section 3.7) such as IPMI or iLO is used for power control, it is possible that a head node is not able to reach its own BMC interface over the network. This is especially true when no dedicated BMC network port is used. In this case, `cmsh -c "device power status"` reports a failure for the active head node. This does not necessarily mean that the head nodes cannot reach the BMC interface of their counterpart. Pinging a BMC interface can be used to verify that the BMC interface of a head node is reachable from its counterpart.

Example

Verifying that the BMC interface of `mycluster2` is reachable from `mycluster1`:

```
[root@mycluster1 ~]# ping -c 1 mycluster2.bmc.cluster
PING mycluster2.bmc.cluster (10.148.255.253) 56(84) bytes of data.
64 bytes from mycluster2.bmc.cluster (10.148.255.253): icmp_seq=1
ttl=64 time=0.033 ms
```

Verifying that the BMC interface of `mycluster1` is reachable from `mycluster2`:

```
[root@mycluster2 ~]# ping -c 1 mycluster1.bmc.cluster
PING mycluster1.bmc.cluster (10.148.255.254) 56(84) bytes of data.
64 bytes from mycluster1.bmc.cluster (10.148.255.254): icmp_seq=1
ttl=64 time=0.028 ms
```

Testing Automated Failover Against A Simulated Crash

A normal (graceful) shutdown of an active head node, does not cause the passive to become active, because HA assumes a graceful failover means there is no intention to trigger a failover. To carry out testing of an HA setup with automated failover, it is therefore useful to simulate a kernel crash on one of the head nodes. The following command crashes a head node instantly:

```
echo c > /proc/sysrq-trigger
```

After the active head node freezes as a result of the crash, the passive head node powers off the machine that has frozen and switches to active mode. A hard crash like this can cause a database replication inconsistency when the crashed head node is brought back up and running again, this time passively, alongside the node that took over. This is normally indicated by a FAILED status for the output of `cmha status` for MySQL (section 13.4). Database administration with the `dbreclone` command (section 13.4) 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 13.4.2). 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.

13.3 Running `cmha-setup` Without Ncurses, Using An XML Specification

13.3.1 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.
13.3.2 The Syntax Of cmha-setup Without Ncurses

The express mode (-x) options are displayed when “cmha-setup -h” is run. The syntax of the -x options is indicated by:

```
cmha-setup [ -x -c <configfile> [-s <type>] <-i|-f[-r]> [-p <mysqlrootpassword>] ]
```

The -x options are:

- `-c|--config <configfile>`: specifies the location of `<configfile>`, which is the failover configuration XML file for cmha-setup. The file stores the values to be used for setting up a failover head node. The recommended location is at `/cm/local/apps/cluster-tools/ha/conf/failoverconf.xml`.

- `-i|--initialize`: prepares a failover setup by setting values in the CMDaemon database to the values specified in the configuration file. This corresponds to section 13.2.1. 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 13.2.2.

- `-f|--finalize`: After the passive node is cloned as described in steps 1 to 10 of section 13.2.2, 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 13.2.2.
  
  - `-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` or `drbd`.

- `-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.

13.3.3 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@bright73 ~]# cd /cm/local/apps/cluster-tools/ha/conf
[root@bright73 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 13.2.1.
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 13.2.2.

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@bright73 conf]# cmha-setup -x -c failoverconf.xml -f -r
Please enter the mysql root password:
```

```
Updating secondary master mac address ..... [ OK ]
Initializing failover setup on master2 ..... [ OK ]
Cloning database ..... [ OK ]
Update DB permissions ..... [ OK ]
Checking for dedicated failover network ..... [ OK ]
```
A reboot has been issued on master2

The preceding corresponds to steps 11 to 13 of section 13.2.2.

Adding Storage:
Continuing with the session on the active, setting up a shared storage could be done with:

```
[root@bright73 conf]# cmha-setup -x -c failoverconf.xml -s nas
```

The preceding corresponds to carrying out the NAS procedure of section 13.2.3.

13.4 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.

13.4.1 Changing An Existing Failover Configuration
Changing an existing failover configuration is usually done most simply by running through the HA setup procedure of section 13.2 again, with one exception. The exception is that the existing failover configuration must be removed by using the “Undo Failover” menu option between steps 2 and 3 of the procedure described in section 13.2.1.

13.4.2 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 Bright 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
```

© Bright Computing, Inc.
[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 cmgui, viewing the status of the head node, and how to make a head node active, are described in section 13.4.7.

- 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 cmgui or cmsh because it requires stopping CM-Daemon.

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
  backing up [ OK ]
  mysql [ OK ]
  ping [ OK ]
  status [ OK ]

mycluster2 -> mycluster1*
  backing up [ 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 nodes:

<table>
<thead>
<tr>
<th>HA Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backing up</td>
<td>the other head node is reachable via the dedicated failover network. This backup ping uses the failover route instead of the internal net route. It is a SYN 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 is a SYN 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, Bright Cluster Manager prepares to carry out the failover sequence (the sequence that includes a STONITH) when all three of ping, backing up, 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 13.2.4).

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 13.1.7), then the failover sequence attempts to complete automatically if power management is working properly, and the cmha status shows ping, backpping 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 from cmgui, as described in section 13.4.7, or it 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.

**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, backpping, 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 511) 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@bright73 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright73* -> head2
  backspacing  [ OK ]
  mysql       [ OK ]
  ping        [ OK ]
  status      [ OK ]

head2 -> bright73*
  backspacing  [ OK ]
  mysql       [FAILED]  (11)
  ping        [ OK ]
  status      [ OK ]

[root@bright73 ~]# cmha dbreclone head2
Proceeding will cause the contents of the cmdaemon state database on he\ad2 to be resynchronized from this node (i.e. bright73 -> head2)

Are you sure ? [Y/N]
Y
Waiting for CMDaemon (3113) to terminate...
   [ OK ]
Waiting for CMDaemon (7967) to terminate...
   [ OK ]

  cmdaemon.dump.8853.sql 100%  253KB 252.9KB/s 00:00
  slurmacctdb.dump.8853.sql 100%  11KB 10.7KB/s 00:00

Waiting for CMDaemon to start...
   [ OK ]
Waiting for CMDaemon to start...[ OK ]

[root@bright73 ~]# cmha status
Node Status: running in active master mode
```

© Bright Computing, Inc.
Failover status:
bright73* -> head2
  backpping [ OK ]
  mysql [ OK ]
  ping [ OK ]
  status [ OK ]
head2 -> bright73*
  backpping [ OK ]
  mysql [ OK ]
  ping [ OK ]
  status [ OK ]

13.4.3 States
The state a head node is in can be determined in three different ways:

1 By looking at the message being displayed at login time.

Example

---------------------------------------------------------------------
| Node Status: running in active master mode |
---------------------------------------------------------------------

2 By executing cmha status.

Example

[root@mycluster ~]# cmha status
Node Status: running in active master mode
...

3 By examining /var/spool/cmd/state.

There are a number of possible states that a head node can be in:

<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.
### 13.4.4 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 “cmha makeactive” on it.</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Nothing</td>
<td>As above.</td>
</tr>
<tr>
<td>Unusably sluggish or freezing system by state pingable with SYN 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 13.4.2).</td>
</tr>
<tr>
<td>Become passive in response to “cmha makeactive” 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>

### 13.4.5 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.

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 160).

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.

- Changes controlled by CMDaemon are synchronized automatically.
  
  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.

- If filesystem changes are made on an active head node without using CMDaemon (`cmsh` or `cmgui`), and if the changes are outside the shared filesystem, then these changes should normally also be made by the administrator on the passive head node. For example:
  
  - RPM installations/updates (section 9.2)
  - Applications installed locally
  - Files (such as drivers or values) placed in the `/cm/node-installer/` directory and referred to by `initialize` (section 5.4.5) and `finalize` scripts (section 5.4.11)
  - Any other configuration file changes outside of the shared filesystems

The reason behind not syncing everything automatically is to guarantee that a change that breaks a head node is not accidentally propagated to the passive. This way there is always a running head node.

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 13.2, 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 13.1.3 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.

© Bright Computing, Inc.
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 13.1.3.

For services that migrate across head nodes during failover, such as NFS and Workload Management, 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 13.4.2), 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.

13.4.6 High Availability Parameters
There are several HA-related parameters that can be tuned. Accessing these via cmgui is described in section 13.4.7. 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
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead time</td>
<td>10</td>
</tr>
<tr>
<td>Disable automatic failover</td>
<td>no</td>
</tr>
<tr>
<td>Failover network</td>
<td>failovernet</td>
</tr>
<tr>
<td>Init dead</td>
<td>30</td>
</tr>
<tr>
<td>Keep alive</td>
<td>1</td>
</tr>
<tr>
<td>Mount script</td>
<td></td>
</tr>
<tr>
<td>Postfailover script</td>
<td></td>
</tr>
<tr>
<td>Prefailover script</td>
<td></td>
</tr>
<tr>
<td>Quorum time</td>
<td>60</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Secondary headnode</td>
<td></td>
</tr>
<tr>
<td>Unmount script</td>
<td></td>
</tr>
<tr>
<td>Warn time</td>
<td>5</td>
</tr>
</tbody>
</table>

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 13.1.7 covers this further.

Failover network

The Failover network setting determines which network is used as a dedicated network for the backupping (backup 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 \texttt{Init dead} seconds is used rather than the \texttt{Dead time} to determine whether the other node is alive.

Keep alive
The \texttt{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 \texttt{Mount script} setting is responsible for bringing up and mounting the shared filesystems.

Postfailover script
The script pointed to by the \texttt{Postfailover script} setting is run by \texttt{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 \texttt{Prefailover script} setting is run by \texttt{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 \texttt{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 \texttt{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 \texttt{Secondary headnode} setting is used to define the secondary head node to the cluster.

Unmount script
The script pointed to by the \texttt{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 \texttt{Warn time} seconds, a warning is logged that the active head node might become unreachable soon.

13.4.7 Handling And Viewing Failover Via \texttt{cmgui}

Accessing \texttt{cmha} Functionality Via \texttt{cmgui}
The equivalent functions of \texttt{cmha} (section 13.4.2) are available under \texttt{cmgui} by selecting a cluster from the resource tree, and then choosing the \texttt{Failover tab} (figure 13.8).

The states of the head nodes are indicated in the first section of the tabbed pane, where the machines are shown along with their modes and states, and with LED lights. Hovering the mouse cursor over the LED lights causes hovertext to appear, indicating which check is associated with which light.

Manual failover can be initiated from \texttt{cmgui} by clicking on the \texttt{Manual Failover button} (figure 13.8).
13.4 Managing HA

Figure 13.8: Accessing HA Cluster Parameters And Functions Via cmgui

Accessing cmsh HA Parameters (partition failover base) Via cmgui

The cmgui equivalents of the cmsh HA parameters in section 13.4.6 are accessed from the same cmgui tab as described earlier in this section (13.4.7).

13.4.8 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 13.2.2. Instead of running the cm-clone-install --failover command as in that section, the following command can be run:

```
[root@bright73 ~]# /cm/cm-clone-install --clone --hostname=<new host name>
```

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).
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 13.2.2).

- **excludelistnormal**: used to exclude files to help generate a clone of the other head node. It is read when running the `cm-clone-install` command without the --failover option.
- **excludelistfailover**: used to exclude files to help generate a passive head node from an active head node. It is read when running the `cm-clone-install --failover` command.

In a default cluster, there is no need to alter these exclude files. However some custom head node configurations may require appending a path to the list.

The cloning that is carried out is logged in `/var/log/clone-install-log`.

Exclude Lists In Perspective

The exclude lists `excludelistfailover` and `excludelistnormal` described in the preceding paragraphs should not be confused with the exclude lists of section 5.6.1. The exclude lists of section 5.6.1:

- **excludelistupdate**
- **excludelistfullinstall**
- **excludelistsyncinstall**
- **excludelistgrabnew**
- **excludelistgrab**
- **excludelistmanipulatescript**

are `cmgui` or `cmsh` options, and are maintained by CMDaemon. On the other hand, the exclude lists introduced in this section (13.4.8):

- **excludelistfailover**
- **excludelistnormal**

are not `cmgui` or `cmsh` options, are not modified with `excludelistmanipulatescript`, are not maintained by CMDaemon, but are made use of when running the `cm-clone-install` command.

13.5 HA For Regular Nodes

HA for regular nodes is available from Bright Cluster Manager version 7.0 onwards.

13.5.1 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.
13.5.2 Comparing Head And Regular Node HA

Many of the features of HA for regular nodes are as for head nodes. These include:

<table>
<thead>
<tr>
<th>Head And Regular Node HA: 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 13.1.7).</td>
</tr>
<tr>
<td>Warn time and dead time parameters can be set (section 13.4.6).</td>
</tr>
<tr>
<td>Mount and unmount scripts (page 510).</td>
</tr>
<tr>
<td>Pre- and post-failover scripts (section 13.4.6).</td>
</tr>
<tr>
<td>Disabling or enabling automatic failover (section 13.1.7).</td>
</tr>
<tr>
<td>A virtual shared IP address that is presented as the virtual node that is always up (section 13.1.4).</td>
</tr>
</tbody>
</table>

Some differences between head and regular node HA are:

<table>
<thead>
<tr>
<th>Head And Regular Node HA: Some Features That Differ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Head Node HA</strong></td>
</tr>
<tr>
<td>Only one passive node.</td>
</tr>
<tr>
<td>Can use the optional failover network and backuping heartbeat.</td>
</tr>
<tr>
<td>Configured within failover submode of partition mode (section 13.4.6).</td>
</tr>
<tr>
<td>Installed with cmha-setup (section 13.2).</td>
</tr>
<tr>
<td>Requires Bright Cluster Manager Advanced Edition license.</td>
</tr>
<tr>
<td>A quorum procedure (section 13.1.6). 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>
</tr>
</tbody>
</table>

| **Regular Node HA**                                  |
| Multiple passive nodes defined by failover groups.  |
| No failover network. Heartbeat checks done via regular node network. |
| Configured within failovergroups submode of partition mode (section 13.5.3). |
| Installed by administrator using a procedure similar to section 13.5.3. |
| Available in both Standard and Advanced Edition. |
| Active head node does checks. If active regular node is apparently dead, it is powered off (STONITH). Another regular node is then made active. |

Failover Groups

Regular nodes use failover groups to identify nodes that are grouped for HA. Two or more nodes are needed for a failover group to function. During normal operation, one member of the failover group is active, while the rest are passive. A group typically provides a particular service.

13.5.3 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
By default, CUPS is provided in the standard image, in a stopped state. In cmgui the failover group can be added via the Failover groups tab. This is available from the cluster item in the resource tree (figure 13.9).

Figure 13.9: cmgui Regular Node Failover Groups Tab

Adding Nodes To The Failover Group
Regular nodes can then be added to a failover group. On adding, Bright 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

$[bright73->partition[base]->failovergroups][]).% set nodes node001..node002
$[bright73->partition[base]->failovergroups][]).% commit
$[bright73->partition[base]->failovergroups][]).% device
foreach -n node001..node002 (services; add cups; \ set runif active; set autostart yes; set monitored yes)
$[bright73->device][]).% commit
Successfully committed 2 Devices

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

$[bright73->partition[base]->failovergroups][]).% device
foreach -n node001..node002 (services; add cups; \ set runif active; set autostart yes; set monitored yes)
$[bright73->device][]).% commit
Successfully committed 2 Devices
Mon Apr  7 08:45:54 2014 [notice] node001: Service cups was started
13.5 HA For Regular Nodes

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

[bright73->partition[base]->failovergroups]% status
Name  State  Active Nodes
-----------------  --------  --------------------------
cupsgroup        ok        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

[bright73->partition[base]->failovergroups]% use cupsgroup
[bright73->...]->failovergroups[cupsgroup]]% makeactive node002
node002 becoming active ...
[bright73->partition[base]->failovergroups[cupsgroup]]% ... Failover group cupsgroup, make node002 become active
...node001: Service cups was stopped
...node002: Service cups was started
...Failover group cupsgroup, failover complete. node002 became active

An alternative is to simply use the cmha utility (section 13.4.2):

Example

[root@bright73 ~]# 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

[bright73->partition[base]->failovergroups]% show cupsgroup
Parameter                                Value
---------------------------------------- ---------------------
Automatic failover after graceful shutdown no
Dead time                                10
Disable automatic failover               no
Mount script                             cupsgroup
Name                                     cupsgroup
Nodes                                    node001, node002
Postfailover script                      
Prefailover script                       
Revision                                 
Unmount script                           
Warn time                                5

Setting Up The Virtual Interface To Make The Server An HA Service

The administrator then assigns each node in the failover group the same alias interface name and IP address dotted quad on its physical interface. The alias interface for each node should be assigned to start up if the node becomes active.

Example

© Bright Computing, Inc.
Successfully committed 2 Devices

```
[...]

Type    Network device name  IP       Network
-------- ---------------------- ------- ------------------
alias    BOOTIF:0             10.141.240.1 internalnet
physical BOOTIF [prov]      10.141.0.1  internalnet

Optionally, each alias node interface can conveniently be assigned a common arbitrary additional
host name, perhaps associated with the server, which is CUPS. This does not result in duplicate names
here because only one alias interface is active at a time. Setting different additional hostnames for the
alias interface to be associated with a unique virtual IP address is not recommended.

Example
```

```
[...]

The preceding can also simply be included as part of the set commands in the foreach statement
earlier when the interface was created.

The nodes in the failover group should then be rebooted.

Only the virtual IP address should be used to access the service when using it as a service. Other
IP addresses may be used to access the nodes that are in the failover group for other purposes, such as
monitoring or direct access.

Service Configuration Adjustments
A service typically needs to have some modifications in its configuration done to serve the needs of the
cluster.

CUPS uses port 631 for its service and by default it is only accessible to the local host. Its default
configuration is modified by changing some directives within the cupsd.conf file. For example, some
of the lines in the default file may be:

```
# Only listen for connections from the local machine.
Listen localhost:631
...
# Show shared printers on the local network.
...
BrowseLocalProtocols
...

<Location />
  # Restrict access to the server...
  Order allow,deny
</Location>
...
```

Corresponding lines in a modified cupsd.conf file that accepts printing from hosts on the internal
network could be modified and end up looking like:
# Allow remote access
Port 631
...
# Enable printer sharing and shared printers.
...
BrowseAddress @LOCAL
BrowseLocalProtocols CUPS dnssd
...
</Location>
  # Allow shared printing...
  Order allow,deny
  Allow from 10.141.0.0/16
</Location>
...

The operating system that ends up on the failover group nodes should have the relevant service modifications running on those nodes after these nodes are up. In general, the required service modifications could be done:

- with an initialize or finalize script, as suggested for minor modifications in section 3.15.4
- by saving and using a new image with the modifications, as suggested for greater modifications in section 3.15.2, page 130.

**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 13.2.4.

**Example**

```bash
ssh node001
echo c > /proc/sysrq-trigger
```

A passive node then takes over.

**13.5.4 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 535):
Sequence Of Events In Making Another HA Regular Node Active

All
- Run pre-failover script

Active
- Stop service
- Run umount script (stop and show `exit>0` on error)
- Stop active IP address
- Start passive IP address
- Start services on passive.
- Active is now passive

Passive
- Stop service
- Run mount script
- Stop passive IP address.
- Start active IP address.
- Start service.
- Passive is now active.

All
- Post-failover script

The actions are logged by CMDaemon.
The following conditions hold for the sequence of actions:

- The remaining actions are skipped if the active `umount` script fails.
- The sequence of events on the initial active node is aborted if a STONITH instruction powers it off.
- The actions for all nodes is done for that particular failover group, for all nodes in that group.

13.6 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 `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>DRBD</th>
<th>DAS</th>
<th>NAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slurm</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>OGS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

...continues
Table 13.6: HA Support For WLM Jobs Vs Shared Filesystems...continued

<table>
<thead>
<tr>
<th>WLM</th>
<th>DRBD</th>
<th>DAS</th>
<th>NAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UGE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Torque</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>PBSPro</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>LSF</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>OpenLava</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

As the table shows, PBS Pro, LSF and OpenLava are not able to support HA for jobs on DRBD or 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 and DRBD 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 Bright Cluster Manager pre-job health check configuration.
Puppet (https://puppetlabs.com/) is a popular tool that system administrators can use to manage IT infrastructure and the state of that infrastructure. Functionally it has some overlap with Bright Cluster Manager because it too deals with the configuration of groups of computers.

Puppet is installed by default on Bright Cluster Manager, and Bright Cluster Manager provides Puppet integration in order to take advantage of the capabilities and strengths of both Puppet and Bright Cluster Manager. So the idea is that Bright Cluster Manager provides the basic configuration for the cluster, while Puppet can then be used to fine-tune it.

This chapter shows how Puppet is integrated and used within Bright Cluster Manager.

14.1 Puppet Basics

Some familiarity with the use of Puppet is assumed. The following is a reminder of the basics of Puppet.

14.1.1 Puppet Resource

A Puppet resource is an item that can be configured on a server. A resource type definition describes a resource and the set of possible parameters associated with it. A definition is abstract, and instantiated by a declaration.

14.1.2 Puppet Class

A Puppet class definition is a collection of resource declarations describing the state of a server. Like the resource definition, it is abstract and also instantiates using a declaration.

14.1.3 Puppet Declaration

A Puppet declaration is maintained as blocks of Puppet code that refer to definitions. Each declaration contains the class or resource type and a set of parameters. For a resource type, a unique name is also needed.

14.1.4 Puppet Manifests

Resources and classes are defined in the form of source code in manifests. Manifests can be bundled into modules.

14.1.5 Puppet Modules

A Puppet module bundles a Puppet manifest with templates and other files. A Puppet module is used to carry out specific kinds of configuration tasks. For example, there are Puppet modules that do ssh configuration. Puppet modules can be obtained from Puppet module repositories, such as, for example, https://forge.puppetlabs.com. Puppet modules should not be confused the properties and syntax of the modules environment of section 2.2.

© Bright Computing, Inc.
When Puppet code is used to provide a configuration, most parameters are predefined in the definitions. They can be overridden if needed.

14.2 Puppet Configuration With cmsh

14.2.1 Configuration Of Puppet

Within cmsh the main configuration is defined in the configuration object within puppet mode. Only one configuration object is allowed at a time. The cmsh shell therefore drops into this object, if it exists, when going into puppet mode.

A new configuration config with default values can be configured with, for example:

Example

```
% puppet
->puppet% add config
->puppet[config%]% commit
```

The configuration properties can be listed and modified. A rescan scans Puppet repositories.

Example

```
->puppet[config%]% rescan; show
Puppet classes <50 in submode>
Puppet modules <1 in submode>
Revision
applyDirectory /var/spool/cmd/puppet/
applyTimeout 3600
baseDirectory /usr/
customModuleDirectory /cm/shared/apps/puppet/custom_modules
forge http proxy
forge http proxy port 0
forge https proxy
forge https proxy port 0
forge port 443
forge url https://forgeapi.puppetlabs.com
forge version v3
moduleDirectory /cm/shared/apps/puppet/modules
name config
on schedule no
onNodeUp no
puppet version
puppetExecutable bin/puppet
rescanExecutable bin/cm-puppet-parse
rescanTimeout 3600
rubyGemDirectory rubygems/
rubyLibraryDirectory ruby/vendor_ruby/
schedule interval 1800
```

14.2.2 Installing Puppet Modules From The Shell

Most system administrators should find it more convenient to install Puppet modules with cmsh (section 14.2.3) or cmgui (section 14.3.4).

However, Puppet modules can be installed using the bash shell. This can be done by first loading the Puppet environment, and then running the puppet command with the appropriate options in the shell. By default, the Puppet Forge module repository (https://forge.puppetlabs.com) is used. A module such as puppetlabs-apache can then be installed as follows:
Example

```
rroot@bright73~# puppet module install --target-dir /cm/shared/apps/puppet/modules/ puppet\nlabsls-apache
```

Notice: Preparing to install into /cm/shared/apps/puppet/modules ...  
Notice: Downloading from https://forgeapi.puppetlabs.com ...  
Notice: Installing -- do not interrupt ...  
/cm/shared/apps/puppet/modules  
puppetlabs-apache (v1.10.0)  
puppetlabs-concat (v2.2.0)  
puppetlabs-stdlib (v4.12.0)

The target directory path used here is the moduleDirectory path that set in the config object within puppet mode (Puppet configuration properties example, page 542).

For a serverless Puppet installation, such as is installed by Bright Cluster Manager, puppet.conf under /cm/shared/apps/puppet is ignored.

In order to access a module repository, the head node typically needs internet access. If the head node uses a web-proxy, then the environment variables http_proxy and https_proxy must be set before accessing the repository. From a bash prompt these are set with:

```
export http_proxy=<http proxy>
export https_proxy=<https proxy>
```

where <http proxy> and <https proxy> are the hostname or IP address of the proxies.

The rescan command in cmsh then makes Bright Cluster Manager aware of the new resources and classes available in installed or updated modules.

Example

```
[bright73]# puppet rescan
puppet rescan done:
rescanned 0
added 125
removed 0
manifests: scan failed 0
```

14.2.3 Installing Puppet Modules From cmsh

Puppet modules can be listed and managed from within the module submode of puppet mode.

Example

```
[root@bright73 ~]# cmsh
[bright73]# puppet
[bright73->puppet[config]]# module
[bright73->puppet[config]->module]# list
Name (key) Version Installed
------------------------------ -------------- ----------
puppet yes
```

Within the module submode, Puppet modules can be managed, amongst others, with the following commands:

© Bright Computing, Inc.
Management Using The cmsh Puppet Module Commands

Command details: The details of the commands can be viewed by running help followed by the command name:

Example

[bright73->puppet[config]->module]% help rescan
Name:  
rescan - Scan puppet repository

Usage:
rescan [OPTIONS]

Options:
-w, --wait

Searching, and supported commands: The search command can be used as in the following example:

Example

[bright73->puppet[config]->module]% search apache
Request has been sent to background
number module installed latest other versions
------- ------------------------------ ---------- ---------- ------------------------------
1 puppetlabs-apache 1.10.0 1.10.0, 1.9.0, 1.8.1, 1.8.0, +

Unsupported modules can be searched for with the --all flag:
14.2 Puppet Configuration With cmsh

Example

[bright73->puppet[config]->module]% search netcat --all
Request has been sent to background

<table>
<thead>
<tr>
<th>number</th>
<th>module</th>
<th>installed</th>
<th>latest</th>
<th>other versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rfletcher-netcat</td>
<td>no</td>
<td>0.0.1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>razorsedge-cloudera</td>
<td>no</td>
<td>3.0.0</td>
<td>2.2.1, 2.2.0, 2.1.1, 2.1.0, 2+</td>
</tr>
</tbody>
</table>

Promp reappears before command completion: By default, the commands return a prompt immediately, but continue to run in the background. The results of these commands can therefore take some time to appear:

Example

[bright73->puppet[config]->module]% install example42-apache
Request has been sent to background
[bright73->puppet[config]->module]% list
Name (key) Version Installed
------------------------ -------------- ----------
puppet yes

[bright73->puppet[config]->module]%
[bright73->puppet[config]->module]% list
Name (key) Version Installed
------------------------ -------------- ----------
example42-apache 2.1.12 yes
example42-firewall 2.1.3 yes
example42-iptables 2.1.13 yes
...

To set the command behavior so that the prompt only appears after the command has finished its run, the -w|--wait option can be used for many of the commands. Running help for the command shows which commands support the option.

Module removal: Modules are often installed as a group of related components due to dependencies. Module removal can be carried out with the uninstall command in Puppet mode:

Example

[bright73->puppet[config]->module]% uninstall example42-apache #then wait a bit
[bright73->puppet[config]->module]% list
Name (key) Version Installed
------------------------ -------------- ----------
example42-apache 2.1.12 no
example42-firewall 2.1.3 yes
example42-iptables 2.1.13 yes
...

14.2.4 Assignment Of Puppet Roles
Puppet roles can be assigned to nodes, categories, and configuration overlays, to provide them with Puppet classes and resources.
The General Puppet Role

To allow specific Puppet roles to be assigned, the general Puppet role must first be assigned to a node, category, or configuration overlay. For example, assigning a Puppet role to `node001` is done in the usual way:

**Example**

```
[bright73]% device use node001
[bright73->device[node001]]% roles
[bright73->device[node001]->roles]% assign puppet; commit
```

General Puppet Settings, Granularity

The general Puppet role can be used to allow some general Puppet behavior to be set:

**Example**

```
[bright73->device[node001]->roles[puppet]]% show
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>puppet</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>PuppetRole</td>
</tr>
<tr>
<td>class granularity</td>
<td>declaration</td>
</tr>
<tr>
<td>granularity</td>
<td>classes: declaration resources: declaration variables: declaration preamble</td>
</tr>
<tr>
<td>preamble granularity</td>
<td>declaration</td>
</tr>
<tr>
<td>resource granularity</td>
<td>declaration</td>
</tr>
<tr>
<td>variable granularity</td>
<td>declaration</td>
</tr>
<tr>
<td>variables</td>
<td>&lt;0 variables defined in submode&gt;</td>
</tr>
</tbody>
</table>

Several of the parameters in the general Puppet mode are about `granularity`, as described next:

Granularity levels:  There are 4 sub-parameters, each corresponding to an individual component level type:

- class granularity
- preamble granularity
- resource granularity
- variable granularity

Each type can have its granularity value set. The main `granularity` parameter value of variables submode cannot itself be set directly, but is set by the 4 components, and becomes a list of key value elements, taking the form:

```
<type>:  <granularity>...
```

as indicated in the preceding example.

There are two possible granularity values that can be set for Puppet component level types: `declaration`, which operates at class level, and `override`, which operates at overlay level. The setting expressed due to these values when combining the Puppet components is as follows:
• **declaration**: Puppet declarations for configuration overlays that are assigned declaration granularity, inherit Puppet declarations from a lower priority overlay to a higher priority overlay by default. However, if a class that exists in a lower priority overlay is also declared in the higher priority overlay, then all the parameters for that class are taken from the higher priority overlay.

• **override**: Puppet declarations for overlays that are assigned override granularity, override Puppet declarations made for lower priority overlays.

**Example**

```
[bright73->device[node001]->roles[puppet]]% set resourcegranularity override; commit
```

**Assigning Specific Puppet Roles**

After the general Puppet role has been assigned, any of the installed Puppet classes and resources are now available to the nodes, categories, or configuration overlays, as a role. Tab auto-completion (\`assign puppet\<TAB\><TAB\>\ License) can help in the selection of the role. For example, assigning an Apache Puppet class:

**Example**

```
[bright73->device[node001]->roles[puppet]]% assign puppet::apache\<TAB\><TAB\> puppet::apache puppet::apache::mod::actions puppet::apache::mod::cache ...
[bright73->device[node001]->roles[puppet]]% assign puppet::apache
[bright73->device[node001]->roles[puppet::apache*]->parameters]%
```

The assignment to categories and configuration overlays is done in a similar manner.

**Parameters Submode: Setting Puppet Class Parameters**

After a Puppet class such as `puppet::apache` has been assigned as a Bright Cluster Manager role, the `parameters` submode can be used to set or unset the parameters of the class:

**Example**

```
[bright73->device[node001]->roles[puppet::apache*]->parameters]% show
Parameter Value
-------------------------------- ------------------------------------------------
Revision className apache
parameters <0 parameters specified in submode>
puppetClassFactoryKey 137438953473
[bright73->device[node001]->roles[puppet::apache*]->parameters]% set\<TAB\><TAB\>
allow_encoded_slashes default_type manage_group server_tokens
apache_name default_vhost manage_user serveradmin
apache_version dev_packages max_keepalive_requests servername
cfg_dir docroot mime_types_additional service_enable
cfg_template error_documents mod_dir service_ensure
cfgd_dir file_mode mod_enable_dir service_manage
default_charset group mpm_module service_name
default.confd_files httpd_dir package_ensure service_restart
default.mods ip pidfile timeout
default_ssl_ca keepalive ports_file trace_enable
default_ssl_cert keepalive_timeout purge_configs use_optional_includes
default_ssl_chain lib_path purge_vdir use_systemd
default_ssl_crl limitreqfieldsize purge_vhost_dir user
default_ssl_crl_check log_formats rewrite_lock vhost_dir
```

© Bright Computing, Inc.
default_ssl_crl_path  log_level  sendfile  vhost_enable_dir
default_ssl_key    logroot    server_root  vhost_include_pattern
default_ssl_vhost  logroot_mode  server_signature

[bright73->device[node001]->roles[puppet::apache*]->parameters]% set default_mods <TAB><TAB>
false true
[bright73->device[node001]->roles[puppet::apache*]->parameters]% set default_mods true; commit
[bright73->device[node001]->roles[puppet::apache]->parameters]% show Parameter Value

Revision
className   apache
parameters   <1 parameters specified in submode>
puppetClassFactoryKey 137438953523

[bright73->device[node001]->roles[puppet::apache]->parameters]% list key type value

------------- ---------- ----------
default_mods Boolean true

Setting the Puppet parameters for categories and configuration overlays is done in a similar manner.
The parameters of the Bright Cluster Manager parameters object:
Revision,className,... should not be confused with the parameters available for the puppet::apache class:
allow_encoded_slashes,apache_name,...
and the available parameters should not be confused with the parameters that are actually set, which in
the preceding example is:
default_mods

Listing Classes, Resources, And BuiltinTypes
The classes and resources of Puppet modules can be listed within the classes submode under puppet
mode:

Example

[bright73]% puppet classes
[bright73->puppet[config]->classes]% list
Name                                  Type               Module               Version exists
---------------------------------------------------------------------------------------
a2mod                                  Resource         puppetlabs-apache
anchor                                 Resource         puppetlabs-stdlib
apache                                 Class             puppetlabs-apache
apache::balancer                       Resource         puppetlabs-apache

14.2.5 Applying Puppet With apply
After assigning Puppet roles to nodes, categories or configuration overlays, Puppet is ready for opera-
tion. The Puppet apply command can be run within cmsh to operate on all nodes with Puppet roles:

Example

[bright73]% puppet apply

The status command in puppet mode shows whether the command is running. After the apply
command has finished running, the status information should show something like:
### 14.2 Puppet Configuration With cmsh

<table>
<thead>
<tr>
<th>node</th>
<th>Status</th>
<th>Start Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>OkWithChanges</td>
<td>2016-Sep-22 15:46:14</td>
<td>66 sec</td>
</tr>
<tr>
<td>node002</td>
<td>OkNoChanges</td>
<td>2016-Sep-22 15:46:14</td>
<td>15 sec</td>
</tr>
</tbody>
</table>

### Retrieving Details On Applying Puppet

#### The listruns command:

This command gives an overview of the last 25 Puppet runs (some text truncated):

```bash
[bright73%] puppet listruns
```

<table>
<thead>
<tr>
<th>run</th>
<th>invoked</th>
<th>nodes</th>
<th>type</th>
<th>success</th>
<th>running</th>
<th>pending</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tue May 26 17:52:55 2015</td>
<td>2 Manual</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Tue May 26 17:51:14 2015</td>
<td>2 Manual</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

#### The showrun command:

This command provides some more details. These are of the last run by default:

```bash
[bright73%] puppet showrun
```

<table>
<thead>
<tr>
<th>property</th>
<th>value</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>run index</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>start time</td>
<td>Tue May 26 17:52:55 2015</td>
<td></td>
</tr>
<tr>
<td>end time</td>
<td>Tue May 26 17:53:06 2015</td>
<td></td>
</tr>
<tr>
<td>longestRunTime</td>
<td>11 sec</td>
<td>headnode</td>
</tr>
<tr>
<td>shortestRunTime</td>
<td>8 sec</td>
<td>node001</td>
</tr>
<tr>
<td>OkNoChanges</td>
<td>1</td>
<td>headnode</td>
</tr>
<tr>
<td>Failures</td>
<td>1</td>
<td>node001</td>
</tr>
</tbody>
</table>

#### The log command:

This command provides even more detail for a particular node, also for the last run by default:

```bash
[bright73%] puppet log node001
```

#### The status command:

This command shows the last results for each node, even if the node was not part of the last apply. It gives an overview of the current state of nodes:

```bash
[bright73%] puppet status
```

<table>
<thead>
<tr>
<th>node</th>
<th>Status</th>
<th>Start Time</th>
<th>Duration</th>
<th>classes/resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>node001</td>
<td>OkNoChanges</td>
<td>Thu Jun 18 16:00:33 2015</td>
<td>24 sec</td>
<td>0 / 1</td>
</tr>
<tr>
<td>node002</td>
<td>Failures</td>
<td>Thu Jun 18 15:32:34 2015</td>
<td>6 sec</td>
<td>0 / 1</td>
</tr>
</tbody>
</table>

**The Run Index Option** `-r|--run` allows information from earlier runs to be displayed for the showrun and log commands. A value of 0 means the information from last run is shown, and is the implied default value for these commands. Thus, details from the second last run can be displayed with:

```bash
[bright73%] puppet showrun -r 1; puppet log -r 1 node001
```
Troubleshooting After Applying Puppet

For troubleshooting purposes, the output of Puppet itself, along with the generated Puppet code, can be useful.

Example

```bash
[cluster%] puppet log --config node001
class 'apache':
...
...
```

14.3 Puppet Configuration With cmgui

If Puppet has not yet been configured, then it can be configured with a default configuration from cmgui by selecting the Puppet resource, and clicking on the Configure button.

Within cmgui, after Puppet is configured, the Puppet resource displays the following tabs:

- Overview (section 14.3.1)
- Settings (section 14.3.2)
- Classes (section 14.3.3)
- Modules (PuppetForge) (section 14.3.4)
- Configuration (section 14.3.5)

14.3.1 Overview Tab

The Overview tab (figure 14.1) displays the results of Puppet apply commands, and some related data. The Puppet states are shown for the nodes. Using menu dropdown selections, the states can be shown for the latest apply run, for all runs, for production runs, or for dry-runs (noop runs).
The Apply buttons can be used to apply Puppet as a production run or noop run. The apply runs can also be applied to all nodes, or just a selection of nodes.

The Get log button shows the Puppet code and output log of apply runs for selected nodes (figure 14.2).

![Puppet code and log file from node node001](image)

**Figure 14.2: Puppet Overview - Get Log Display**

### 14.3.2 Settings Tab

The Settings tab (figure 14.3) displays the current configuration of Puppet.
Parameters here can be edited and saved. Units of time are seconds by default, but minutes and hours can be set as the unit by appending a word starting with an m or an h to the number value. By default:

- the Puppet root directory is set to /usr/
- the Puppet working directory is set to /var/spool/cmd/puppet
- the Puppet modules directory is set to /cm/shared/apps/puppet/modules
- the Puppet custom modules directory is set to /cm/shared/apps/puppet/custom_modules

The other directories in this screen are relative to the Puppet root directory. The default values rarely need to be changed.
14.3 Puppet Configuration With cmgui

14.3.3 Classes Tab

The **Classes** tab (figure 14.4) allows Puppet classes and resources within Puppet modules to be viewed. The view can be expanded or collapsed using the + and - widgets. The installation state of the modules and their parts is shown.

Details of the modules and classes can be viewed using the Details button. This is the cmgui version of the explain command (section 14.2.3) in the puppet mode of cmsh.

The types of class declarations that can be listed are built-ins, resource-like, and class-like. Dependencies and globals are additional classes that Bright Cluster Manager provides, outside of Puppet. Puppet provides a number of built-in types by default.

The classes can be assigned to configuration groups, which are overlays, categories, or nodes.

- If this is done inside the Puppet Configuration tab of cmgui, then Puppet roles are automatically set for the selected configuration group.
- This is in contrast to the case when using cmsh, where the puppet role must be explicitly assigned to the configuration group, from within the configuration group. That is, in configurationoverlay mode, in category mode, or device mode.

The modules that provide the classes can be installed with the Install Puppet Modules button.

14.3.4 Modules (PuppetForge) Tab: Installing Puppet Modules From cmgui

Section 14.2.2 described the text-based version of installing Puppet modules. The cmgui equivalent of installing Puppet modules can be carried out with the Puppet modules tab (figure 14.5).
This is a searchbox-based installer. The only supported modules checkbox restricts the search to official and supported PuppetForge modules. The Search for Updates button finds updates to the existing modules in use, while the Changes button shows any Puppet manifest changes to existing modules.

Modules found are listed and made available for installation and removal (figure 14.6).

The installation itself is carried out on clicking the Install button. Forcing options are available on clicking the eye-and-magnifying-glass widget. Immediately after the installation of the new module via the Install button is complete, a rescan is automatically done, so that CMDaemon becomes automatically aware of what modules are now available. For cases where Puppet modules have been installed outside of CMDaemon, the administrator can use the Rescan button from the Classes tab to manually make CMDaemon aware of what modules are now available.

### 14.3.5 Configuration Tab: Puppet Hello World Example

There is much overlap between Puppet and Bright Cluster Manager in their capabilities to deploy configurations in handy, manageable ways. Puppet integration with Bright Cluster Manager is designed to make it natural to have traditional Bright Cluster Manager continue to carry out the more general, cluster-oriented configuration, and let the Puppet aspects of the integration with Bright Cluster Manager handle finer operating system configuration.

The example that follows in this section is about an administrator who would like to carry out a "hello world" cron-driven mailshot from a regular node, node001, on the cluster. That is, the administrator would like to have the cluster configured with a cron job running on that node, as shown:

**Example**
This sort of fine-tuning modification can be done in the traditional Bright Cluster Manager way of setting node images or finalize scripts, as suggested in section 3.15 on cluster configuration. Indeed, just running it as a cron job from the head node would solve the requirement very easily, and Puppet or Bright Cluster Manager would not even be considered. However, to make the example less idiotic, the contrived condition of running it on a regular node is set. Puppet can then be used in a simple manner to solve the requirement.

**Hello World Puppet Configuration Via cmsh**

If the main Puppet module `puppet` is installed, then the `cron` built-in is available. To configure the `cron` built-in, it can be assigned to a node in cmsh:

```
[root@node001 ~]# crontab -l
* * * * * echo "hello world" | mail -s "hello world!" root@bright73
```

An editor opens up. The text:
```
echo "hello world" | mail -s "hello world!" root@bright73
```

is written and saved by the administrator

The node `node001` now has the “hello world” mailshot script assigned to the cron command. By default, the script is configured to run every minute. The configuration can now be applied:

```
[prompt] % apply
```
After applying this, node001 is seen to have the command in its cron table:

Example

```
[root@bright73 ~]# ssh node001 "crontab -l"
```

```
# HEADER: This file was autogenerated at 2016-09-30 16:52:31 +0200 by puppet.
# HEADER: While it can still be managed manually, it is definitely not recommended.
# HEADER: Note particularly that the comments starting with 'Puppet Name' should
# HEADER: not be deleted, as doing so could cause duplicate cron jobs.
# Puppet Name: pcron
  * * * * * echo "hello world" | mail -s "hello world!" root@bright73
```

and the mail account root@bright73 should be receiving the mailshots accordingly:

Example

```
[root@bright73 ~]# tail -f /var/spool/mail/root
Subject: hello world!
User-Agent: Heirloom mailx 12.5 7/5/10
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Message-Id: <20160930145801.D10B937F8F5@node001.cm.cluster>
From: root@node001.cm.cluster (root)

hello world
```

**Hello World Puppet Configuration Via cmgui**

Configuration groups can be of type nodes, categories, or configuration overlays. The preceding example was about using cmsh for Puppet configuration for a configuration group of type node. In cmgui, Puppet configuration for a configuration group of type configuration overlay can be done as follows:

The configuration tab is selected, and the configuration groups subtab shows an overview of the various configuration groups (nodes, overlays and categories), along with their assigned Puppet roles (figure 14.7).

![Puppet Configuration Groups Subtab](image)

Figure 14.7: Puppet Configuration Groups Subtab

The previous pcron assignment from the cmsh configuration can be seen in the preceding figure too, if the tree is expanded by using the + widget.

A new overlay can be built with a dialog that comes up when the New button is clicked (figure 14.8).
14.3 Puppet Configuration With cmgui

Figure 14.8: Puppet Configuration Groups New Configuration Overlay Dialog

The dialog allows:

- a configuration overlay name to be set for the overlay. A suggestion is made automatically, based on the form PuppetConfiguration_n, where n=1, 2, 3...
- a priority to be set for the overlay
- nodes to be added to or removed from the overlay
- declarations to be set for a Puppet
  - class
  - resource
  - variable
- a preamble code to be written in a text box
- Puppet variables to be added

An existing class or resource, such as cron can be added to the overlay by clicking on the Add Resource/Class button, which opens up a menu. The menu option puppet, then cron, can then be selected, and a dialog for the class then opens up (figure 14.9).
A suggestion is made for the resource title automatically, based on the form `<class>_<n>`, where `<class>` is the name of the Puppet class or resource—in this case `cron`—and where `n=1, 2, 3 ....` For example: `cron_1`

A parameter, such as `command`, can be enabled for the class instance by moving it over from the list of parameters. For `cron` the `command` parameter is expected to be of type string, and it is indeed `string` by default. For other parameters it may make sense to be able to change their type, and it is therefore possible to modify the parameter type.

Nodes can be added to the configuration overlay in the dialogs.

The `Preview` button shows the result of a Puppet apply run.

Once the administrator is satisfied with the configuration overlay that has been constructed, the overlay can be saved with the `Save` button of figure 14.7.

The overlay can then be applied using the `Apply` button from the `Overview` tab (figure 14.1).

If `node001` and `node003` are configured as suggested in figure 14.9, and the `pcron` assignment from the earlier `cmsh` configuration remains in place, then applying means that the overlay works on `node001` to configure the cron table as:

```
[root@node001 ~]# crontab -l
# HEADER: This file was autogenerated at 2016-10-03 16:49:53 +0200 by puppet.
# HEADER: While it can still be managed manually, it is definitely not recommended.
# HEADER: Note particularly that the comments starting with 'Puppet Name' should
# HEADER: not be deleted, as doing so could cause duplicate cron jobs.
# Puppet Name: pcron
  * * * * * echo "hello world" | mail -s "hello world!" root@bright73
# Puppet Name: cron_1
  * * * * * echo "hello world" | mail -s "hello world! via cmgui" root@bright73
```

The overlay works on `node003` to configure the cron table as:

```
[root@node003 ~]# crontab -l
# HEADER: This file was autogenerated at 2016-10-03 16:49:53 +0200 by puppet.
# HEADER: While it can still be managed manually, it is definitely not recommended.
# HEADER: Note particularly that the comments starting with 'Puppet Name' should
# HEADER: not be deleted, as doing so could cause duplicate cron jobs.
# Puppet Name: pcron
  * * * * * echo "hello world" | mail -s "hello world! via cmgui" root@bright73
# Puppet Name: cron_1
  * * * * * echo "hello world" | mail -s "hello world! via cmgui" root@bright73
```
The corresponding messages are sent by mail to root@bright73.

Example

[root@bright73 ~]# tail -f /var/spool/mail/root | egrep ‘(^From: |world)’
Subject: hello world!
From: root@node001.cm.cluster (root)
hello world
Subject: hello world! via cmgui
From: root@node003.cm.cluster (root)
hello world
Subject: hello world! via cmgui
From: root@node001.cm.cluster (root)
hello world
Subject: hello world!
...
15 Dell BIOS Management

15.1 Introduction
Dell BIOS management in Bright Cluster Manager means that for nodes that run on Dell hardware, the BIOS settings and BIOS firmware updates can be managed via the standard Bright front end utilities to CMDaemon, cmgui and cmsh.

In turn, CMDaemon configures the BIOS settings and applies firmware updates to each node via a standard Dell utility called racadm. The racadm utility is part of the Dell OpenManage software stack.

The Dell hardware supported is listed in the following table:

<table>
<thead>
<tr>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>R430, R630, R730, R730XD, R930</td>
</tr>
<tr>
<td>FC430, FC630, FC830</td>
</tr>
<tr>
<td>M630, M830</td>
</tr>
<tr>
<td>C6320</td>
</tr>
</tbody>
</table>

Bright Cluster Manager supports iDRAC Express and iDRAC Enterprise for these models too.

This chapter describes the Dell BIOS management integration features, with:

- An introduction to the integration features (section 15.1)
- The prerequisites for the integration to work (section 15.2)
- A description of settings management in cmgui and cmsh (section 15.3)
- A troubleshooting FAQ (section 15.4)

15.2 Prerequisites For BIOS Management

- The utility racadm must be present on the Bright Cluster Manager head node. The utility is installed on the head node if Dell is selected as the node hardware manufacturer during Bright Cluster Manager installation (section 3.3.5 of the Installation Manual).
- IPMI must be working on all of the servers. This means that it should be possible to communicate out-of-band from the head node to all of the compute nodes, via the IPMI IP address.
- iDRAC management privileges must be enabled for all the nodes for the Bright BMC user (username: bright, userid: 4). This operation has to performed manually. This privilege cannot be enabled during node provisioning by Bright during the BMC configuration step.
15.3 BIOS settings

15.3.1 Initializing The BIOS Settings Via cmsh

Initially, by default, there are no values set for the BIOS settings in the nodes or categories. An administrator therefore typically imports the values for the nodes from the cluster, in order for Bright Cluster Manager to have an initial configuration. An example of how each node in Bright Cluster Manager can have its BIOS values defined is:

Example

[bright73->device]# foreach -n node002..node008 (dellsettings; importlive; commit; )

Similarly, a category can have BIOS settings defined by taking a single node with suitable settings, and running the importlive command with that node being used as the settings for that category.

Example

[bright73->category[default]]% dellsettings; importlive node009; commit

As usual with CMDaemon values, settings at the node level override settings at the category level. To clear settings at the node level for each node within a category default, the clearsettings command can be used:

Example

[bright73->device]# foreach -c default (dellsettings; clearsettings; commit; )

The importlive operation can only be performed from within cmsh at the time of writing (July 2015). The operation will also become available in cmgui updates in Bright Cluster Manager 7.3 later on.

15.3.2 Managing The BIOS Settings Via cmgui

To access the Dell BIOS settings via cmgui, the node or category is first selected from the Nodes or Node Categories resource. If CMDaemon is able to communicate with the racadm utility then

- the Dell tab is available for selection within
  - a tab in a selected node of the Nodes resource (figure 15.3)
  - a tab in a selected category of the Node Categories resource (figure 15.2)

- some extra fields become available in the nodes overview screen in the Nodes resource. The nodes overview screen is seen when a specific node is not selected. The extra fields are BIOS version, Model and System profile (figure 15.1).

![Figure 15.1: Nodes overview folder view](image)

© Bright Computing, Inc.
Within the nodes overview of within the **Node Categories** resource, when a category is selected, the **Dell** tab shows a similar nodes overview screen (figure 15.2).

![Nodes overview folder view within a category](image)

If a node has been selected from the **Nodes** resource, then its **Dell** tab displays some general Dell hardware information near the top (figure 15.3). Below this information are then the following 3 subtabs:

1. **BIOS Settings**: Basic BIOS settings. These contain most of the BIOS settings except for NIC-related BIOS settings and a few other BIOS settings
2. **NIC Integrated Settings**: BIOS settings for the NICs
3. **General Settings**: Some other BIOS settings.

These 3 subtabs are described next.

### The BIOS Settings Subtab

![Selection of a node node004, then Dell tab, with the BIOS Settings subtab on display](image)

© Bright Computing, Inc.
The BIOS Settings subtab groups the basic BIOS options in blocks under the following headings:

- **System Profile**: This block has options to set the system profile. The options are general settings based on the kind of optimization wanted for the nodes. For example: optimization based on a particular kind of performance.

- **Serial Communication Settings**: these decide the serial port characteristics. For example: speed, port, console redirection after POST.

- **Processor Settings**: CPU characteristics. For example: power consumption, cache.

- **Memory Settings**: RAM characteristics. For example: refresh rate, frequency.

- **BIOS Boot Order**: Device boot order settings. For example: which NIC or hard drive to boot from.

For a node in the Nodes resource the Show live button displays the current values of its BIOS alongside the configured values (figure 15.4).

The NIC Integrated Settings Subtab

The NIC Integrated Settings subtab displays the NICs of the node in rows, with the configured property settings of the NICs laid out in columns (figure 15.5).
For a node in the Nodes resource the Show live button displays the current properties of its NICs (figure 15.6) alongside the configured NICs. The Show live button is not available by design within a Dell tab within the Node Categories resource.

![Node NIC settings screenshot](image)

**Figure 15.6: Live NIC settings for a node**

Double-clicking on a row for a configured NIC opens up a new window where the properties of the NIC can be edited (figure 15.7).
Editing a configured NIC via a new window is also possible within a NIC Integrated Settings subtab within Node Categories, and sets NIC properties at a category level.
The **General Settings Subtab**

The **General Settings** subtab (figure 15.8) allows a path to be set to the location on the head node where the firmware has been placed by the administrator. This path is then used when a firmware update is carried out with the [Firmware update button](#) for Nodes or Node Categories.

HTTP, FTP, or TFTP access can also be specified, using IPV4 addresses. These options assume that a firmware catalog file exist at the HTTP, FTP or TFTP servers that host the firmware files. A firmware update is carried out with the `racadm update` command. The `racadm` software only updates particular kinds of firmware. The `racadm` documentation should be referred to for supported firmware file formats.

**Example**

```
/cm/shared/firmwares/BIOS_YKH49_WN64_1.3.6.EXE
```

```
tftp://10.2.4.3
```

```
ftp://10.2.4.3
```

```
http://10.2.4.3
```

For FTP access, a username and password should be set.

15.3.3 Applying Dell Settings And Firmware Updates Via cmgui

The **Dell** tab, within Node and Node Categories, has the following buttons in it:

- Apply: Used to apply Dell BIOS and NIC device settings that have been configured.
- **Firmware update**: Used to apply firmware updates based on the configured firmware path details.

These buttons and the associated Task Manager are now described in more detail.

**Applying Dell Settings**

The **Apply** button launches a confirmation dialog (figure 15.9).

![Confirmation dialog for applying Dell settings](image)

**Figure 15.9: Confirmation dialog for applying Dell settings**

A checkbox in the dialog allows the administrator to run a Task Manager window. Clicking on the **Yes** button of the dialog launches the Task Manager itself (figure 15.10).
The Task Manager displays the progress of the application of the Dell BIOS and NIC settings as a series of tasks. Double-clicking on a task in the Task Manager shows details, while a click on the Log button displays further details.

**Applying Firmware Updates**
Clicking on the Firmware update button of the Dell tab starts a firmware update operation. The parameters specified in the General Settings tab are used. A confirmation dialog and Task Manager are launched in a similar way to the Apply button.

**15.3.4 Managing The BIOS Settings Via cmsh**
In cmsh, the Dell BIOS settings are available under the dellsettings submode. The dellsettings submode is available under a node or category object. If no Dell settings are yet configured, then a new local object is created automatically. The new object is saved using commit.

**BIOS: Listing Properties**
A list of the possible Dell BIOS properties that can be configured is seen in the help text for the get command run within the submode (some text elided):

**Example**

```
[bright73->category[default]->dellsettings]$ get
...
Parameters:
Readonly ............ Mark data as readonly
Revision ............ Entity revision
applyimmediately
biosbootorder
collaborativecpuperfctrl
devicefirmware
efficientturbo
efficiencybiasfailsafebaudrate
l3cache
logicalprocessor
```

© Bright Computing, Inc.
memfrequency
mempatrolscrub
memrefreshrate
mmioabove4gb
modelname
monitormwait
nics
nodeinterleaving
numberofcores
proc1turbocorenum
proc2turbocorenum
procclle
proccstates
procpwrperf
procturbomode
redirectionafterboot
remoteterminal
serialcommunication
serialport
servicetag
snoopmode
sysprofile
turboboost
uncorefrequency

BIOS: Setting Properties
Tab-completion suggestions for the set show the possible properties that can be modified.

Example
[bright73->category[default]->dellsettings]% set <TAB><TAB>
applyimmediately collaborativecpuperfctrl monitormwait ...
...
Tab-completion suggestions for a particular property, such as monitormwait, show the possible values that can be taken by a property with a selection of options. A property can then be set and committed:

Example
[bright73->category[default]->dellsettings]% set monitormwait <TAB><TAB>
nomodify collaborativecpuperfctrl monitormwait ...

BIOS: Importing Properties With importlive
The importlive command imports the Dell settings for a category from an existing node configuration.

Example
[bright73->category*[default*]->dellsettings]% help importlive
importlive - Imports current live dell settings
Usage:
    [-f] importlive node001

Options:
-f, --forcerefresh
Force cmdaemon to refresh its cached live DellSettings copy, before importing

Examples:
importlive Imports current live dell settings

[bright73->category*{default*}]->dellsettings]% importlive node001
[bright73->category*{default*}]->dellsettings]% commit
[bright73->category{default}]->dellsettings]%

**Firmware Updates**
Firmware update configurations can be listed and set within the firmware submode:

**Example**

[bright73->category*{default*}]->dellsettings]% firmware
[bright73->category*{default*}]->dellsettings->firmware]% show

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readonly</td>
<td>no</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>clearConfig</td>
<td>no</td>
</tr>
<tr>
<td>ftpPassword</td>
<td></td>
</tr>
<tr>
<td>ftpUser</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
</tr>
</tbody>
</table>

**Applying Settings For Dell With** applydellsettings
The applydellsettings command of cmsh is used to apply BIOS or firmware configuration changes to a node. The command is available from within device mode.

When applydellsettings is run, a background task is immediately automatically started to apply the settings. The progress can be monitored with the task submode, and the list of current background tasks can be viewed using the task list command.

**Example**

[bright73->device{node001}]% applydellsettings
[bright73->device{node001}]% task list

Details for each task can be shown with the show command, run from within the task submode.

**Example**

[bright73->task]% show 1
...

If applying a Dell settings task has failed, then the Log property of the task should show a detailed error message explaining why.

**Example**

[bright73->task]% show 1
...
Log: ERROR: Unable to connect to RAC at specified IP address.

© Bright Computing, Inc.
Viewing Live BIOS And Firmware Settings With `showlive`

The `showlive` command of `cmsh` displays the BIOS values currently used by the node. These are not necessarily the same as the currently configured values, because the currently configured values may not yet have been pushed to the node. The `showlive` command is available from within the `dellsettings` submode, which is under the node or category object. If live settings have not yet been fetched, then the operation can be forced with the `-f` option.

**Example**

```
[bright73->device[device001]->dellsettings]% showlive
[bright73->device[device001]->dellsettings]% showlive -f
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readonly</td>
<td>no</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>applyImmediately</td>
<td>no</td>
</tr>
<tr>
<td>biosBootOrder</td>
<td>NIC.Embedded.1-1-1, NIC.Mezzanine.1A-1,...</td>
</tr>
<tr>
<td>collaborativeCpuPerfCtrl</td>
<td>no</td>
</tr>
<tr>
<td>devicefirmware</td>
<td>&lt;submode&gt;</td>
</tr>
<tr>
<td>energyEfficientTurbo</td>
<td>no</td>
</tr>
<tr>
<td>energyPerformanceBias</td>
<td>MaxPower</td>
</tr>
<tr>
<td>failsafeBaudRate</td>
<td>115200</td>
</tr>
<tr>
<td>l3Cache</td>
<td>yes</td>
</tr>
<tr>
<td>logicalProcessor</td>
<td>no</td>
</tr>
<tr>
<td>memFrequency</td>
<td>MaxPerf</td>
</tr>
<tr>
<td>memPatrolScrub</td>
<td>Standard</td>
</tr>
<tr>
<td>memRefreshRate</td>
<td>1x</td>
</tr>
<tr>
<td>mmioAbove4Gb</td>
<td>no</td>
</tr>
<tr>
<td>modelName</td>
<td></td>
</tr>
<tr>
<td>monitorMWait</td>
<td>yes</td>
</tr>
<tr>
<td>nics</td>
<td>&lt;2 in submode&gt;</td>
</tr>
<tr>
<td>nodeInterleaving</td>
<td>no</td>
</tr>
<tr>
<td>numberOfCores</td>
<td>All</td>
</tr>
<tr>
<td>proc1TurboCoreNum</td>
<td>All</td>
</tr>
<tr>
<td>proc2TurboCoreNum</td>
<td>All</td>
</tr>
<tr>
<td>procC1E</td>
<td>no</td>
</tr>
<tr>
<td>procCStates</td>
<td>no</td>
</tr>
<tr>
<td>procPwrPerf</td>
<td>MaxPerf</td>
</tr>
<tr>
<td>procTurboMode</td>
<td>no</td>
</tr>
<tr>
<td>redirectionAfterBoot</td>
<td>yes</td>
</tr>
<tr>
<td>remoteTerminal</td>
<td>Vt100Vt220</td>
</tr>
<tr>
<td>serialCommunication</td>
<td>OnConRedirAuto</td>
</tr>
<tr>
<td>serialPort</td>
<td>Serial1Com2Serial2Com1</td>
</tr>
<tr>
<td>serviceTag</td>
<td></td>
</tr>
<tr>
<td>snoopMode</td>
<td>ClusterOnDie</td>
</tr>
<tr>
<td>sysprofile</td>
<td>PerfOptimized</td>
</tr>
<tr>
<td>turboBoost</td>
<td>no</td>
</tr>
<tr>
<td>uncoreFrequency</td>
<td>DynamicUFS</td>
</tr>
</tbody>
</table>

```
[bright73->device[device001]->dellsettings]%
```

15.4 Frequently Asked Questions

1. Where are the log files for the Apply and Firmware update operations?

   All Apply actions are logged in `/var/log/cm-dell-manage.log` on the head node, and all Firmware update actions are logged to `/var/log/cm-dell-manage.log` on the node itself.
2. Why does clicking on the Show Live button in cmgui display nothing?

This is because the required information is not fetched with `racadm` when the node booted for the first time. To force this operation at the time of request, the `showlive -f` command from `cmsh` should be used (page 572).

```
[bright73->device[node001]->dellsettings]% showlive -f
```

3. Why does the cmgui Dell tab for a node show no configured BIOS or NIC device settings?

This is because there are no Dell settings configured for a node. Settings can be initialized from `cmsh` with the `importlive` command. If there is no live information for the node yet, then the command `importlive -f` can be used:

**Example**

```
[bright73->device[node001]->dellsettings]% importlive -f
```

4. Why do the `showlive -f` or `importlive -f` commands fail, saying that the BMC user privileges are insufficient in the reports in `/var/log/cm-dell-manage.log`?

This is because the default BMC user `bright` with user id 4 does not have sufficient privileges to perform this operation. The default BMC user, user ID, and password that the CMDaemon uses for performing this operation, should be changed to a BMC user that has iDRAC management privileges. This can be done from the main Settings tab in cmgui or from each node or base partition submode in cmsh.
This appendix contains lists of all system configuration files which are generated automatically by the
node-installer or CMDaemon. These are files generated 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 2.6.4 describes how system configuration files on all nodes are written out using the Cluster
Management Daemon. The Cluster Management Daemon is introduced in section 2.6.4 and its configu-
ration directives are listed in Appendix C.

All of these configuration files may be listed as Frozen Files in the Cluster Management Dae-
mon configuration file to prevent them from being generated automatically. 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 Files Generated Automatically On Head Nodes

Files generated or modified automatically on the head node by CMDaemon:
<table>
<thead>
<tr>
<th>File</th>
<th>Method</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/aliases</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/dhclient.conf</td>
<td>Entire file</td>
<td>Red Hat only</td>
</tr>
<tr>
<td>/etc/dhcpd.conf</td>
<td>Entire file</td>
<td></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/haproxy/haproxy.cfg</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/hosts</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/hosts.equiv</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/localtime</td>
<td>Entire file</td>
<td>Copied from zoneinfo</td>
</tr>
<tr>
<td>/etc/named.conf</td>
<td>Entire file</td>
<td>For zone additions use /etc/named.conf. include(^1). For options additions, use /etc/named.conf.global.options.include</td>
</tr>
<tr>
<td>/etc/ntp.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/ntp/step-tickers</td>
<td>Section</td>
<td>Red Hat only</td>
</tr>
<tr>
<td>/etc/openstack-dashboard/local_settings</td>
<td>Section</td>
<td></td>
</tr>
</tbody>
</table>
## A.1 Files Generated Automatically On Head Nodes

<table>
<thead>
<tr>
<th>File</th>
<th>Method</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/postfix/canonical</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/postfix/main.cf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/postfix/generic</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/resolv.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/interfaces</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/masq</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/netmap</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/policy</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/shorewall/zones</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/slurm/gres.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/slurm/slurm.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/slurm/slurmdbd.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/slurm/topology.conf</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/bmc.cfg</td>
<td>Entire file</td>
<td>BMC configuration</td>
</tr>
<tr>
<td>/etc/sysconfig/clock</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/dhcpd</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/network</td>
<td>Section</td>
<td>Red Hat only</td>
</tr>
<tr>
<td>/etc/sysconfig/network/routes</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
<tr>
<td>/etc/sysconfig/network/ifcfg-*</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
<tr>
<td>/etc/sysconfig/network/dhcp</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
<tr>
<td>/etc/sysconfig/network-scripts/ifcfg-*</td>
<td>Section</td>
<td>Red Hat only</td>
</tr>
<tr>
<td>/etc/HOSTNAME</td>
<td>Entire file</td>
<td>SUSE only</td>
</tr>
<tr>
<td>/tftpboot/mtu.conf</td>
<td>Entire file</td>
<td>Bright Computing configuration</td>
</tr>
<tr>
<td>/var/named/*.zone(^1)</td>
<td>Entire file</td>
<td>Red Hat only. For custom additions use /var/named/*.zone.include</td>
</tr>
<tr>
<td>/var/lib/named/*.zone(^1)</td>
<td>Entire file</td>
<td>SUSE only. For custom additions use /var/lib/named/*.zone.include</td>
</tr>
</tbody>
</table>

\(^1\) User-added zone files ending in \(*\).zone\(^1\) that are placed for a corresponding zone statement in the include file /etc/named.conf.include are wiped by CMDaemon activity. Another pattern, e.g. \(*\).myzone, must therefore be used instead.

© Bright Computing, Inc.
### A.2 Files Generated Automatically In Software Images:

<table>
<thead>
<tr>
<th>File</th>
<th>Generated By</th>
<th>Method</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/boot/vmlinuz</code></td>
<td>CMDaemon</td>
<td>Symlink</td>
<td></td>
</tr>
<tr>
<td><code>/boot/initrd</code></td>
<td>CMDaemon</td>
<td>Symlink</td>
<td></td>
</tr>
<tr>
<td><code>/boot/initrd-*</code></td>
<td>CMDaemon</td>
<td>Entire file</td>
<td>SUSE only</td>
</tr>
<tr>
<td><code>/etc/aliases</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
<tr>
<td><code>/etc/hosts</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/inittab</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td>only in Red Hat, not in RHEL7.x</td>
</tr>
<tr>
<td><code>/etc/inittab</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td>only in Red Hat, not in RHEL7.x</td>
</tr>
<tr>
<td><code>/etc/securetty</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/sysconfig/clock</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td><code>/etc/sysconfig/init</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td>only in Red Hat, not in RHEL7.x</td>
</tr>
<tr>
<td><code>/etc/sysconfig/network/dhcp</code></td>
<td>CMDaemon</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
</tbody>
</table>

© Bright Computing, Inc.
## A.3 Files Generated Automatically On Regular Nodes

Files generated automatically on regular nodes

<table>
<thead>
<tr>
<th>File</th>
<th>Generated By</th>
<th>Method</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/aliases</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/exports</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/fstab</td>
<td>Node-installer</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/hosts</td>
<td>Node-installer</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/init/serial.conf</td>
<td>CMDaemon</td>
<td>Section</td>
<td>Red Hat only, not in RHEL7.x</td>
</tr>
<tr>
<td>/etc/inittab</td>
<td>CMDaemon</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
<tr>
<td>/etc/mkinitrd_cm.conf</td>
<td>CMDaemon</td>
<td>Section</td>
<td>Red Hat only</td>
</tr>
<tr>
<td>/etc/modprobe.d/bright-cmdaemon.conf</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/pam.d/sshd</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/ntp.conf</td>
<td>Node-installer</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/ntp/step-tickers</td>
<td>Node-installer</td>
<td>Entire file</td>
<td>Red Hat only</td>
</tr>
<tr>
<td>/etc/postfix/main.cf</td>
<td>Node-installer</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/resolv.conf</td>
<td>Node-installer</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/securetty</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/slurm/gres.conf</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/slurm/slurm.conf</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/clock</td>
<td>CMDaemon</td>
<td>Section</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/init</td>
<td>CMDaemon</td>
<td>Section</td>
<td>Red Hat only, not in RHEL7.x</td>
</tr>
<tr>
<td>/etc/sysconfig/network</td>
<td>Node-installer</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/sysconfig/network/dhcp</td>
<td>CMDaemon</td>
<td>Section</td>
<td>SUSE only</td>
</tr>
<tr>
<td>/etc/sysconfig/network/ifcfg-*</td>
<td>Node-installer</td>
<td>Entire file</td>
<td>Red Hat only, not ifcfg-lo</td>
</tr>
<tr>
<td>/etc/sysconfig/network-scripts/ifcfg-*</td>
<td>Node-installer</td>
<td>Entire file</td>
<td></td>
</tr>
<tr>
<td>/etc/HOSTNAME</td>
<td>Node-installer</td>
<td>Entire file</td>
<td></td>
</tr>
</tbody>
</table>

## A.4 Files Not Generated, But Installed.

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 Bright 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.

© Bright Computing, Inc.
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 Bright 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@bright73 ~]# repoquery -l $(repoquery -a | grep -F _cm7.3) | grep nsswitch.conf$
```

The inner `repoquery` displays a list of all the packages. By grepping for the cluster manager version string, for example `_cm7.3` for Bright Cluster Manager 7.3, the list of cluster manager packages is found. The outer `repoquery` displays the list of files within each package in the list of cluster manager packages. By grepping for `nsswitch.conf$`, any file paths ending in `nsswitch.conf` in the cluster manager packages are displayed. The output is:

```
/cm/conf/etc/nsswitch.conf
/cm/conf/etc/nsswitch.conf
/cm/node-installer/etc/nsswitch.conf
```

Files under `/cm/conf` are placed by Bright 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 Bright 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 Bright Cluster Manager packages are updated. During such a Bright 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 Bright Cluster Manager. These are normally the files provided by the distribution:

```
[root@bright73 ~]# repoquery -l $(repoquery -a | grep -F -v _cm7.3) | grep nsswitch.conf$
/etc/nsswitch.conf
/etc/nsswitch.conf
/usr/share/doc/yp-tools-2.9/nsswitch.conf
```

**Which package provides the file in Bright 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
```

This reveals that some Bright 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

```
~# yum whatprovides */etc/nsswitch.conf
```

shows that `glibc` provides the distribution version of the `nsswitch.conf` file, and that there is also a node-installer version of this file available from the Bright Cluster Manager packages.
What are the differences between the Bright 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:

```bash
~# mkdir yumextracted ; cd yumextracted
~# yumdownloader glibc-2.12-1.107.el6.x86_64.rpm
~# rpm2cpio glibc-2.12-1.107.el6.x86_64.rpm | cpio -idmv
~# yumdownloader cm-config-ldap-client-6.0-45_cm7.3.noarch.rpm
~# rpm2cpio cm-config-ldap-client-6.0-45_cm7.3.noarch.rpm | cpio -idmv
~# diff etc/nsswitch.conf cm/conf/etc/nsswitch.conf
```

What are the configuration files in an RPM package? An RPM package allows files within it to be marked as configuration files. Files marked as configuration files can be listed with `rpm -qc <package>`. Optionally, piping the list through “sort -u” filters out duplicates.

**Example**

```bash
~# rpm -qc glibc | sort -u
/etc/ld.so.cache
/etc/ld.so.conf
/etc/localtime
/etc/nsswitch.conf
/usr/lib64/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 Bright 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 Bright 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:

```bash
rpm -q --queryformat '%{FILENAMES}\t%{FILEFLAGS:fflags}\n' glibc | egrep '([^:space:]).*\.(c|n)\.[n]:$' | sort -u
/etc/ld.so.cache cmng
/etc/ld.so.conf cn
/etc/localtime cn
/etc/nsswitch.conf cmng
/usr/lib64/gconv/gconv-modules cn
/var/cache/ldconfig/aux-cache cmng
```

1 This freezing should not be confused with the `FrozenFile` directive (Appendix C), where the file or section of a file is being maintained by CMDaemon, and where freezing the file prevents CMDaemon from maintaining it.
Here, the second column of the output displayed shows which of the files in the package have a configuration (c) flag or a no replace (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.

Bright Cluster Manager should however mark all critical files as configuration files in the Bright 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:

- The file text content should be checked to see if it is a CMDaemon-maintained file (section 2.6.4), or checked against the list of generated files (Appendix A). This is just to make sure to avoid confusion about how changes are occurring in such a file.

  - If it is a CMDaemon-maintained file, then configuration changes put in by the administrator will also not persist in the maintained section of the file unless the FrozenFile directive (section C) is used to freeze the change.

  - If it is only a section that CMDaemon maintains, then configuration changes can be placed outside of the maintained section.

Wherever the changes are placed in such a file, these changes are in any case by default overwritten on RPM updates if the file has no n flag.

- Some regular node updates can effectively be maintained in a desired state with the help of a finalize script (Appendix E).

- Updates can be excluded from YUM/zypper (section 9.3.2), thereby avoiding the overwriting of that file by the excluded package.

A request to change the package build flag may be sent to 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 596.

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.

© Bright Computing, Inc.
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 cmgui 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. During normal running, both SSL and non-SSL ports are listening, but only the SSL port is used. By default, the non-SSL port is only used during CMDaemon start up.

If PXEBootOverHTTP (page 609) is set to 1, then SSLPortOnly must be set to no.

CertificateFile directive
Syntax: CertificateFile = filename
Default: CertificateFile = "/cm/local/apps/cmd/etc/cert.pem"

The CertificateFile directive specifies the PEM-format certificate which is to be used for authentication purposes. On the head node, the certificate used also serves as a software license.

PrivateKeyFile directive
Syntax: PrivateKeyFile = filename
Default: PrivateKeyFile = "/cm/local/apps/cmd/etc/cert.key"

The PrivateKeyFile directive specifies the PEM-format private key which corresponds to the certificate that is being used.

CACertificateFile directive
Syntax: CACertificateFile = filename
Default: CACertificateFile = "/cm/local/apps/cmd/etc/cacert.pem"

The CACertificateFile directive specifies the path to the Bright Cluster Manager PEM-format root certificate. It is normally not necessary to change the root certificate.

RandomSeedFile directive
Syntax: RandomSeedFile = filename
Default: RandomSeedFile = "/dev/urandom"

The RandomSeedFile directive specifies the path to a source of randomness.

DHParamFile directive
Syntax: DHParamFile = filename
The `DHParamFile` directive specifies the path to the Diffie-Hellman parameters.

**SSLHandshakeTimeout directive**

Syntax: `SSLHandshakeTimeout = number`

Default: `SSLHandshakeTimeout = 10`

The `SSLHandshakeTimeout` directive controls the time-out period (in seconds) for SSL handshakes.

**SSLSessionCacheExpirationTime directive**

Syntax: `SSLSessionCacheExpirationTime = number`

Default: `SSLSessionCacheExpirationTime = 300`

The `SSLSessionCacheExpirationTime` directive controls the period (in seconds) for which SSL sessions are cached. Specifying the value 0 can be used to disable SSL session caching.

**DBHost directive**

Syntax: `DBHost = hostname`

Default: `DBHost = "localhost"`

The `DBHost` directive specifies the hostname of the MySQL database server.

**DBPort directive**

Syntax: `DBPort = number`

Default: `DBPort = 3306`

The `DBPort` directive specifies the TCP port of the MySQL database server.

**DBUser directive**

Syntax: `DBUser = username`

Default: `DBUser = cmdaemon`

The `DBUser` directive specifies the username used to connect to the MySQL database server.

**DBPass directive**

Syntax: `DBPass = password`

Default: `DBPass = "<random string set during installation>"`

The `DBPass` directive specifies the password used to connect to the MySQL database server.

**DBName directive**

Syntax: `DBName = database`

Default: `DBName = "cmdaemon"`
The `DBName` directive specifies the database used on the MySQL database server to store CMDaemon related configuration and status information.

**DBMonName directive**

*Syntax:* `DBMonName = database`
*Default:* `DBMonName = "cmdaemon_mon"`

The `DBMonName` directive specifies the database used on the MySQL database server to store monitoring related data.

**DBUnixSocket directive**

*Syntax:* `DBUnixSocket = filename`
*Default:* `DBUnixSocket = "/var/lib/mysql/mysql.sock"`

The `DBUnixSocket` directive specifies the named pipe used to connect to the MySQL database server if it is running on the same machine.

**DBUpdateFile directive**

*Syntax:* `DBUpdateFile = filename`
*Default:* `DBUpdateFile = "/cm/local/apps/cmd/etc/cmdaemon_upgrade.sql"`

The `DBUpdateFile` directive specifies the path to the file that contains information on how to upgrade the database from one revision to another.

**EventBucket directive**

*Syntax:* `EventBucket = filename`
*Default:* `EventBucket = "/var/spool/cmd/eventbucket"`

The `EventBucket` directive (section 10.6.3) specifies the path to the named pipe that is created to listen for incoming events from a user.

**EventBucketFilter directive**

*Syntax:* `EventBucketFilter = filename`
*Default:* `EventBucketFilter = "/cm/local/apps/cmd/etc/eventbucket.filter"`

The `EventBucketFilter` directive (section 10.6.3) 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.

**HomeRoot directive**

Syntax: HomeRoot = path
Default: HomeRoot = "/home"

The HomeRoot directive specifies the default user home directory used by CMDaemon. It is used for automatic mounts, exports, and when creating new users.

**DocumentRoot directive**

Syntax: DocumentRoot = path
Default: DocumentRoot = "/cm/local/apps/cmd/etc/htdocs"

The DocumentRoot directive specifies the directory mapped to the web-root of the CMDaemon. The CMDaemon acts as a HTTP-server, and can therefore in principle also be accessed by web-browsers.
**SpoolDir directive**

Syntax: `SpoolDir = path`
Default: `SpoolDir = "/var/spool/cmd"`

The SpoolDir directive specifies the directory which is used by the CMDaemon to store temporary and semi-temporary files.

**EnableJSON directive**

Syntax: `EnableJSON = true|false`
Default: `EnableJSON = true`

The EnableJSON directive allows cmgui. If the administrator or other user create other JSON-dependent utilities, these require the directive to be true too.

**EnableShellService directive**

Syntax: `EnableShellService = true|false`
Default: `EnableShellService = true`

The EnableShellService directive allows the use of the Root Shell, Remote Console, Telnet Shell, and Ssh Shell buttons in cmgui for connections to devices that allow such shells. The connection runs over CMDaemon, which is running over SSL, which means that between cmgui and the device, the connection is encrypted.

   The directive does not affect cmsh’s rshell, rconsole telnet, and ssh commands.

**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 = filename`
Default: `CMDaemonAuditorFile = "/var/spool/cmd/audit.log"`

The CMDaemonAuditorFile directive sets where the audit logs for CMDaemon actions are logged. The log format 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)

**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 `Authorization` resource of `cmgui`, 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 591).

Whether events are logged in files or not, events are cached and accessible using `cmsh` or `cmgui`. The number of events cached by CMDaemon is determined by the parameter `MaxEventHistory` (page 591). On a related note, `cmgui` users have access to a file in their home directory, with a relative path of `.cm/cmgui/cmgui.settings`. This file contains the `eventCache` parameter, with a default value of 256. Modifying it changes the limit for the number of events viewed by `cmgui`, which is a front end to CMDaemon.

**EventLoggerFile directive**

Syntax: `EventLoggerFile = filename`

Default: `EventLogger = "/var/spool/cmd/events.log"`

The `EventLogger` directive sets where the events seen in the event viewer (section 10.6) are logged.

**MaxEventHistory directive**

Syntax: `AdvancedConfig = {"MaxEventHistory=number", ...}`

Default: `MaxEventHistory=8192`

`MaxEventHistory` is a parameter of the `AdvancedConfig` (page 600) 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 591) to true, to activate logging to a file, is advised instead.

**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.

**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 Bright 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 Bright Cluster Manager, which is contrary to the general principle of segregating the network activity of networks.

The recommended way now to deny letting new nodes boot up is to set the option for specific networks by using `cmsh` or `cmgui` (section 3.2.1, figure 3.5, table 3.2.1). Setting the `cmd.conf` `LockDownDhcpd` directive overrides `lockdowndhcpd` values set by `cmsh` or `cmgui`.

**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.

**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.

**MetricAutoDiscover directive**

**Syntax:** `MetricAutoDiscover = true|false`

**Default:** `MetricAutoDiscover = true`

If this directive is enabled, then when the state of a regular device becomes `UP`, new metrics are scanned for on the device. If new metrics are discovered, then the CMDaemon monitoring configuration is updated to include these, and their data values are monitored with scheduled metrics and health checks.

**AutomaticMetricBlackList directive**

**Syntax:**

AdvancedConfig = {"AutomaticMetricBlackList="<file>"", ...}

or

AdvancedConfig = {"AutomaticMetricBlackList=<list>"", ...}

**Default:** `none`

AutomaticMetricBlackList is a parameter of the `AdvancedConfig` (page 600) directive.

If the `MetricAutoDiscover` directive is enabled, then Bright Cluster Manager automatically makes newly-discovered metrics available when a device state becomes `UP`. Metrics that are to be blacklisted can be specified in a comma-separated inline list of metrics, or they can be placed in a file. The file is specified by an absolute file path, which is preceded by the `^` character to indicate that it is a path rather than a list of inline metrics.

**File Format**

If `<file>` is `/cm/local/apps/cmd/etc/metricsblacklist.conf`, then its contents could look like:

**Example**

```
[root@bright73 ~]# cat /cm/local/apps/cmd/etc/metricsblacklist.conf
# metrics blacklist
# Comment them out to allow use by monitoring configuration
#
# see also MetricAutoDiscover directive
#
# Network interfaces
# Blacklist bytes sent metrics on eth0, nodes nodeXXX only
# ie head gets the bytes sent metrics for eth0
node.*:BytesSent:eth0
#
# Blacklist bytes sent, bytes received metrics on eth1, all devices
.*:Bytes.*:eth1
#
# Disk metrics
# Blacklist these disks metrics
.*:SMART.*
.*:SectorsWritten
.*:IOTime
#
# File systems
```

© Bright Computing, Inc.
Blacklist Entry Format
The syntax for each line of the file is:

```
<device>:<metric>[::<param>]
```

where `<device>` and `<metric>` are evaluated as regular expressions. The allowed devices can be listed from cmsh with:

```
[bright73->device]% list -f hostname
```

The metrics for `<metric>` and parameters for `<param>` can be listed for a given device, for example, for the head node, from cmsh with:

```
[bright73->device]% monitoring setup use headnode
[bright73->monitoring->setup[HeadNode]]% metricconf
[...[HeadNode]->metricconf]% list -d : -f metric:40,metricparam:60
```

Inline List Format
The blacklist entry formats used in the file are also used in the inline list version of the directive.

Example

If `<list>` is `.*:SMART.*,.*:IOTime`, then an example of the directive used inline is:

```
AdvancedConfig = {"AutomaticMetricBlackList=.*:SMART.*,.*:IOTime"}
```

MaxAgeForDataInMemory directive

Syntax: `MaxAgeForDataInMemory = number`
Default: `MaxAgeForDataInMemory = 3600`

Sets the maximum age, in seconds, for keeping monitoring data in memory. After that, the data will still be available, but with slightly more overhead, using MySQL retrieval.

MaxMemoryForBulkInsert directive

Syntax: `MaxMemoryForBulkInsert = number`
Default: `MaxMemoryForBulkInsert = 100000000`

Sets the maximum amount of memory, in bytes, reserved to buffer the new monitoring data temporarily in memory, before it is flushed to the database. The default value set for CMDaemon is 100MiB. If no value is specified in `cmd.conf`, then the value defaults to 500MiB. The flush action is also triggered if the period of `BulkInsertFlushInterval` is reached between flushes.

BulkInsertFlushInterval directive

Syntax: `BulkInsertFlushInterval = number`
Default: `BulkInsertFlushInterval = 300`

Sets the elapsed time, in seconds, before flushing the monitoring data from memory to the database. The flush action is also triggered if the memory buffer setting of `MaxMemoryForBulkInsert` is reached between flushes).
PDUMetricAutoDiscover directive
Syntax: PDUMetricAutoDiscover = true|false
Default: PDUMetricAutoDiscover= true

If this directive is enabled, then when the state of a PDU device becomes UP, new PDU metrics are scanned for on the device. If new PDU metrics are found, then the monitoring configuration is updated to include these, and their data values are monitored with scheduled metrics and health checks.

SensorMetricAutoDiscover directive
Syntax: SensorMetricAutoDiscover = true|false
Default: SensorMetricAutoDiscover = true

If this directive is enabled, then when the state of a rack sensor device becomes UP, new rack sensor metrics are scanned for on the device. If new rack sensor metrics are found, then the monitoring configuration is updated to include these, and their data values are monitored with scheduled metrics and health checks.

DisableBootLogo directive
Syntax: DisableBootLogo = true|false
Default: DisableBootLogo = false

When DisableBootLogo is set to true, the Bright 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:
FreezeChangesToCondorConfig directive
FreezeChangesToPBSPro directive
FreezeChangesToSGEConfig directive
FreezeChangesToUGEConfig directive
FreezeChangesToSlurmConfig directive
FreezeChangesToTorqueConfig directive
FreezeChangesToOpenLavaConfig directive
FreezeChangesToLSFConfig directive
Syntax: FreezeChangesTo<wlm>Config= true|false
Default: FreezeChangesTo<wlm>Config = false

When FreezeChangesTo<wlm>Config is set to true, CMDaemon does not make any modifications to the workload manager configuration. Workload managers for which this value can be set are:

- Condor

© Bright Computing, Inc.
• PBSPro
• SGE
• UGE
• Slurm
• Torque
• OpenLava
• LSF

FrozenFile directive

Syntax: FrozenFile = { filename[filename] ...

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

The FrozenFile directive is used to prevent the CMDaemon-maintained sections of configuration files from being automatically generated. This is useful when site-specific modifications to configuration files have to be made.

To avoid problems, the file that is frozen should not be a symlink, but should be the ultimate destination file. The readlink -f <symlinkname> command returns the ultimate destination file of a symlink called <symlinkname>. This is also the case for an ultimate destination file that is reached via several chained symlinks.

FrozenFile directive for regular nodes

FrozenFile directive for regular nodes for CMDaemon

The FrozenFile directive can be used within the cmd.conf file of the regular node.

Example

To freeze the file /etc/named.conf on the regular nodes running with the image default-image, the file:

/cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf

can have the following directive set in it:

FrozenFile = {"/etc/named.conf"}

The path of the file that is to be frozen on the regular node must be specified relative to the root of the regular node.

The running node should then have its image updated. This can be done with the imageupdate command in cmsh (section 5.6.2), or the Update node button in cmgui (section 5.6.3). After the update, CMDaemon should be restarted within that category of nodes.

Example

[root@bright73 ~]# pdsh -v -g category=default service cmd restart
node002: Waiting for CMDaemon (25129) to terminate...
node001: Waiting for CMDaemon (19645) to terminate...
node002: [ OK ]
node001: [ OK ]
node002: Waiting for CMDaemon to start...[ OK ]
node001: Waiting for CMDaemon to start...[ OK ]
FrozenFile directive for regular nodes for the node-installer

CMDaemon directives only affect files on a regular node after CMDaemon starts up on the node during the init stage. So files frozen by the CMDaemon directive stay unchanged by CMDaemon after this stage, but they may still be changed before this stage.

Freezing files so that they also stay unchanged during the pre-init stage—that is during the node-installer stage—is possible with node-installer directives.

Node-installer freezing is independent of CMDaemon freezing, which means that if a file freeze is needed for the entire startup process as well as beyond, then both a node-installer as well as a CMDaemon freeze are sometimes needed.

Node-installer freezes can be done with the node-installer directives in /cm/node-installer/scripts/node-installer.conf, introduced in section 5.4:

- frozenFilesPerNode
- frozenFilesPerCategory

Example

Per node:

```plaintext
frozenFilesPerNode = "*:localdisk/etc/ntp.conf", "node003:localdisk/etc/hosts"
```

Here, the * wildcard means that no restriction is set. Setting node003 means that only node003 is frozen.

Example

Per category:

```plaintext
frozenFilesPerCategory = "mycategory:localdisk/etc/sysconfig/network-scripts/ifcfg-eth1"
```

Here, the nodes in the category mycategory are prevented from being changed by the node-installer.

The Necessity Of A FrozenFile Directive

In a configuration file after a node is fully up, the effect of a statement earlier on can often be overridden by a statement later in the file. So, the following useful behavior is independent of whether FrozenFile is being used for a configuration file or not: A configuration file, for example /etc/postfix/main.cf, with a configuration statement in an earlier CMDaemon-maintained part of the file, for example:

```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 3.3.6 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 cmd.conf file

Whenever one or more entities have changed, the custom script at filename, specified by a full path, is called 30s later. Python bindings can be used to get information on the current setup.

ConfigDumpPath directive

Syntax: ConfigDumpPath = filename
Default: ConfigDumpPath = /var/spool/cmd/cmdaemon.config.dump

The ConfigDumpPath directive sets a dump file for dumping the configuration used by the power control script /cm/local/apps/cmd/scripts/pctl/pctl. The pctl script is a fallback script to allow power operations if CMDaemon is not running.

- If no directive is set (ConfigDumpPath = ""), then no dump is done.
- If a directive is set, then the administrator must match the variable cmdconfigfile in the powercontrol configuration file /cm/local/apps/cmd/scripts/pctl/config.py to the value of ConfigDumpPath. By default, the value of cmdconfigfile is set to /var/spool/cmd/cmdaemon.config.dump.

SyslogHost directive

Syntax: SyslogHost = hostname
Default: SyslogHost = "localhost"

The SyslogHost directive specifies the hostname of the syslog host.

SyslogFacility directive

Syntax: SyslogFacility = facility
Default: SyslogFacility = "LOG_LOCAL6"

The default value of LOG_LOCAL6 is set in:

- /etc/syslog.conf in Red Hat 5 and variants

© Bright Computing, Inc.
• /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. Bright 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

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.

- 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.

- The ResolveToExternalName directive has no effect on the regular nodes unless they query the DNS on the head node, bypassing the /etc/hosts file. What this means is that with ResolveToExternalName enabled or disabled, the regular nodes resolve the hostname of any head node to an IP address located on the internal cluster network by default when resolved using the /etc/hosts file, for example using standard ping with the default node name resolution settings sequence. The regular nodes however resolve according to the DNS server’s configuration if querying the DNS server querying directly.

The following points describe more exhaustively how this directive influences the resolution of hostnames to IP addresses for head and regular nodes in the cluster:

- On the head node of a single-head cluster, if ResolveToExternalName is enabled, the master hostname and the actual hostname of the head node, e.g. head, then both resolve to the external IP address of the head node.

- On head nodes of a failover cluster, if ResolveToExternalName is enabled, the actual head hosts, e.g. head1 and head2, then also both resolve to their external IP address. However, the master hostname stays resolved to the internal shared IP. This is because Bright Cluster Manager uses master within its code base, and there is no need to have the failover configuration have a shared external IP address in the code base.

- On regular nodes in a single-head cluster, if ResolveToExternalName is enabled, then the master hostname or the actual hostname of the head node (e.g. head) both stay resolved to the internal IP address of the head node when using /etc/hosts. However they resolve to the external IP address of the head node when the DNS service is used.

- On regular nodes in a failover cluster, if ResolveToExternalName is enabled, the master hostname then still resolves to the shared internal IP. The actual hostnames of the head nodes, e.g.
head1 and head2, also still resolve to the internal IP address of the respective head nodes if resolved via /etc/hosts. However, they resolve to the external IP addresses of the respective head nodes if resolved by querying the DNS.

The behavior discussed in the preceding points can be summarized by Table C:

<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>ResolveToExternalName = False</td>
<td>I I I I I I I No</td>
<td>I I I I I I I Yes</td>
<td></td>
</tr>
<tr>
<td>ResolveToExternalName = True</td>
<td>E E I E E I No</td>
<td>E E I E E I Yes</td>
<td></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. Torque) 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 Bright Cluster Manager Knowledge Base at http://kb.brightcomputing.com/. A search query for FQDN leads to the relevant entry.

**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.

© Bright Computing, Inc.
GlobalConfig directive

Syntax: GlobalConfig = { "<key1>=<value1>", "<key2>=<value2>", ... }  
Default: not in the default cmd.conf

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 with a boot role (section 5.1.5) to carry out, for example, PXEBootOverHTTP (page 609).

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

The ScriptEnvironment directive sets extra environment variables for CMDaemon and child processes.

For example, if CMDaemon is running behind a web proxy, then the environment variable http_proxy may need to be set for it. If, for example, the proxy is the host brawndo, and it is accessed via port 8080 using a username/password pair of joe/electrolytes, then the directive becomes:

ScriptEnvironment = { "http_proxy=joe:electrolytes@brawndo:8080" }  

BurnSpoolDir directive

Syntax: BurnSpoolDir = path  
Default: BurnSpoolDir = "/var/spool/burn/"

The BurnSpoolDir directive specifies the directory under which node burn log files are placed. 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 metric or health checks that have Only when idle (page 657) set to true will not run. If the metric or health check is sampled on a regular node rather than on the head node, then cmd.conf on the regular node should be modified and restarted.

PhaseLoadMetrics directive

Syntax: AdvancedConfig = {"PhaseLoadMetrics=scriptoutput", ...}  
Default: none
PhaseLoadMetrics is a parameter of the AdvancedConfig (page 600) directive. By default, Bright Cluster Manager sums up the phase load in amps for all APC-compatible PDUs. APC-compatible PDUs are expected to display an SNMP OID with the string PowerNet-MIB, like the PDUs described by APC MIB files at ftp://ftp.apc.com/apc/public/software/pnetmib/mib/.

Other PDUs are not supported by default, but can be added as generic devices, and this is recommended for several reasons (page 69). For these PDUs, their current use in amps can be retrieved by creating a custom script, say load(). So, if the PDUs, say PDU1, PDU2, ..., are sampled by the custom script running from the head node, this means that the script output, which is a list of currents, is:

```
load(PDU1), load(PDU2), ...
```

Then setting:

```
AdvancedConfig = { "PhaseLoadMetrics=load(PDU1),load(PDU2), ...", ...}
```

sums the current for these PDUs, and adds it to the clusterwide PhaseLoad (page 653).

### MaxServiceFailureCount directive

**Syntax:**

```
AdvancedConfig = {"MaxServiceFailureCount=number", ...}
```

**Default:** Implicit value: "MaxServiceFailureCount=10"

MaxServiceFailureCount is a parameter of the AdvancedConfig (page 600) directive. Its value determines the number of times a service failure event is logged (page 103). 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 600) 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 602), 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:
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 CMDaemon restart
```

A more refined approach that avoids a complete CMDaemon restart would be to execute a reset (page 103) on the fhgfs-client from within CMDaemon, as follows:

```
[bright73->category[default]->services[fhgfs-client]]% reset fhgfs-client
Successfully reset service fhgfs-client on: node001,node002
[bright73->category[default]->services[fhgfs-client]]%
```

**UserDefinedMetricClasses directive**

**Syntax:**

```
AdvancedConfig = {"UserDefinedMetricClasses="file with metric scripts", ...}
```

or

```
AdvancedConfig = {"UserDefinedMetricClasses=metric1, metric2, ...", ...}
```

**Default:** *none*

UserDefinedMetricClasses is a parameter of the AdvancedConfig (page 600) directive. The parameter can be assigned metrics created by the user. The assignment can be done as:

- **a file path:** The file is then a text file with a metric name on each line
- **a list of values:** Each list item is then a metric name.

**Example**

```
AdvancedConfig = { "UserDefinedMetricClasses=sfa_pool,sfa_vd,sfa_hc,sfa_sys,GridScalerNodeStats,GridScalerFileSyncSystemStats,ExaScalerNodeStats,ExaScalerOSTStats,ExaScalerMDTStats,ExaScaler\ MGTStats,ExaScalerOSTBKWStats,ExaScalerLNET" }
```

**DeviceIsUpPowerSaveFail directive**

**Syntax:**

```
AdvancedConfig = {"DeviceIsUpPowerSaveFail = 0|1", ...}
```

**Default:** 1

DeviceIsUpPowerSaveFail is a parameter of the AdvancedConfig (page 600) directive. It affects the behavior of the DeviceIsUp (Appendix G.2.1) health check. If the health check is run for a device that is in a DOWN (section 5.5) state, then:

- If DeviceIsUpPowerSaveFail is 1 (the default), the health check returns a FAIL, always.
- If DeviceIsUpPowerSaveFail is 0, the health check returns a PASS if the DOWN state is due to Slurm Power Save.

Slurm Power Save (section 7.9.1) is not active in a default installation of Bright Cluster Manager.
CMDaemon ListenOnInterfaces directive

Syntax: AdvancedConfig = {"CMDaemonListenOnInterfaces=<interfaces>", ...}
Default: all interfaces listening to port 8081

CMDaemon ListenOnInterfaces is a parameter of the AdvancedConfig (page 600) 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 600) directive.

It defines the number of seconds until the cmgui connection to CMDaemon times out, if there is no user activity at the cmgui client side.

OpenStackVXLANGroup directive

Syntax: AdvancedConfig = {"OpenStackVXLANGroup=IP address"}
Default: 224.0.0.1

OpenStackVXLANGroup is a parameter of the AdvancedConfig (page 600) directive.

It defines the multicast address used by OpenStack for broadcast traffic with VXLAN networking (page 29 of the OpenStack Deployment Manual). If an application already uses the default IP address, then another IP address can be specified for VXLAN networking needs with this directive.

MaxSGEJobLimit

Syntax: MaxSGEJobLimit = number
Default: Implicit value: MaxSGEJobLimit = 0

This sets the maximum number of lines that is displayed in the jobs window. By default it is set to 0, which means there is no limit set.

However, if the number of jobs is very large, then parsing them takes time, and can result in a high load on CMDaemon. Setting a limit can help CMDaemon cope with such a case.

RsyncHardLinks directive

Syntax: AdvancedConfig = {"RsyncHardLinks=0|1", ...}
Default: 1

RsyncHardLinks is a parameter of the AdvancedConfig (page 600) directive. It controls, for the rsync command, the hard links preservation option: -H|--hard-links

If RsyncHardLinks is set to:

• 1: then rsync copies hard links
• 0: then rsync does not copy hard links

**RsyncXattrs directive**

**Syntax:** AdvancedConfig = {"RsyncXattrs=-1|0|1", ...}

**Default:** -1

*RsyncXattrs* is a parameter of the AdvancedConfig (page 600) directive. It controls, for the rsync command, the extended attributes command line option: `-X|--xattrs`

If *RsyncXattrs* is set to:

• 1: then rsync copies extended attributes

• 0: then rsync does not copy extended attributes

• -1: then rsync follows the default setting for whether or not to copy extended attributes.

The default setting copies extended attributes only if the distribution version:

- is RHEL7 (and derivatives) and above,
- or SLES12 and above

**RsyncAcls directive**

**Syntax:** AdvancedConfig = {"RsyncAcls=-1|0|1", ...}

**Default:** -1

*RsyncAcls* is a parameter of the AdvancedConfig (page 600) directive. It controls, for the rsync command, the access control list (ACL) command line option: `-A|--acls`

If *RsyncAcls* is set to:

• 1: then rsync copies ACLs

• 0: then rsync does not copy ACLs

• -1: then rsync follows the default setting for whether or not to copy ACLs.

The default setting copies ACLs only if the distribution version:

- is RHEL7 (and derivatives), and above,
- or SLES12 and above

**InfoMessageCacheSize directive**

**Syntax:** AdvancedConfig = {"InfoMessageCacheSize=<number>", ...}

**Default:** 1000

*InfoMessageCacheSize* is a parameter of the AdvancedConfig (page 600) directive. It is the number of lines of InfoMessages (section 10.2.8) that CMDaemon caches for speedier lookup.

**MaxInfoMessageSize directive**

**Syntax:** AdvancedConfig = {"MaxInfoMessageSize=<number>", ...}

**Default:** 1024

*MaxInfoMessageSize* is a parameter of the AdvancedConfig (page 600) directive. It is the maximum size, in bytes, of an InfoMessage. It is 1024 by default.
**MaxInfoMessagesPerMetric directive**

Syntax: AdvancedConfig = {"MaxInfoMessagesPerMetric=<number>", ...}

Default: 1000

MaxInfoMessagesPerMetric is a parameter of the AdvancedConfig (page 600) directive. It is the maximum number of InfoMessages that may display for a particular metric or health check value, until the next value is sampled. This helps prevent InfoMessage flooding.

**<wlm>Timeout directives:**

- CondorTimeout directive
- PBSProTimeout directive
- SGETimeout directive
- SlurmTimeout directive
- TorqueTimeout directive
- OpenLavaTimeout directive
- LSFTimeout directive

Syntax: AdvancedConfig = {"<wlm>Timeout=<number>", ...}

where: <wlm> is one of:
- Condor
- PBSPro
- SGE
- Slurm
- Torque
- OpenLava
- LSF

Default: 120

<wlm>Timeout is a parameter of the AdvancedConfig (page 600) directive. It is the maximum time, in seconds, that CMDaemon will wait for a response from the job scheduler of the workload manager. Increasing this value is sometimes needed for clusters with demanding job loads. Workload managers for which this value can be set are:

- Condor
- PBSPro
- SGE
- Slurm
- Torque
- OpenLava
- LSF

**SlurmDisableAccountingParsing directive**

Syntax: AdvancedConfig = {"SlurmDisableAccountingParsing=0|1", ...}

Default: 0

SlurmDisableAccountingParsing is a parameter of the AdvancedConfig (page 600) directive. If set to 1, it disables collection of accounting information for the Slurm workload manager.

**SampleTimeRandomness directive**

Syntax: AdvancedConfig = {"SampleTimeRandomness=<number>", ...}

Default: 0
SampleTimeRandomness is a parameter of the AdvancedConfig directive. Its value is a time period, in seconds. Sampling is spread over this time period instead of having CMDaemon carrying out the sampling at the regular sampling time instant. This can be useful for reducing the load in larger clusters with a high data throughput due to sampling.

The SampleTimeRandomness time interval is an interval that occurs immediately after the regular sampling time. That is, it is an offset time interval. A random time interval is associated within this offset time interval, with each node that is sampled. Thus each node is associated with an offset time interval which is up to SampleTimeRandomness in value. A health check that is to run on the node at the regular sampling time is run after the additional random offset time interval has also elapsed, with the elapsed time being between 0 and SampleTimeRandomness in value.

The random offset time interval associated per node is random per node, but it does not change over time. This means that for a particular node, the sampling period does not change between one sample and the next sample of the health check being run. The sampling period is just the regular sampling period. It is merely offset by an amount that depends on the node.

An example to illustrate this follows:

Supposing:

- SampleTimeRandomness=10
- node001 has a random offset of 2 seconds
- node002 has a random offset of 7 seconds

If the health checks are normally run at 1:00AM, 1:02AM, 1:04AM . . . , then

- node001 health checks run at 1:00:02AM, 1:02:02AM, 1:04:02AM . . .
- node002 health checks run at 1:00:07AM, 1:02:07AM, 1:04:07AM . . .

JobsSamplingMetricsInterval directive

Syntax: AdvancedConfig = {"JobsSamplingMetricsInterval=<number>", ...}
Default: 60

JobsSamplingMetricsInterval is a parameter of the AdvancedConfig 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 directive. Its value is a time period, in seconds. This time period value elapses between the checks that CMDaemon makes to determine the node states (section 5.5) in a cluster. If the network is very congested, then a larger value can be used to reduce the network load caused by these checks.

AddUserScript directive

Syntax: AdvancedConfig = {"AddUserScript=<path>", ...}
Default: none

1actually pseudo-random, which serves the purpose here

© Bright Computing, Inc.
AddUserScript is a parameter of the AdvancedConfig (page 600) directive. If this parameter is set to a path leading to a script, and if a new user is added using cmsh or cmgui, then the script is automatically run by CMDaemon, with the username of the new user automatically passed as the first argument to the script. The script has a default timeout of 5 seconds.

**AddUserScriptPasswordInEnvironment directive**

Syntax: AdvancedConfig = {"AddUserScriptPasswordInEnvironment=0|1", ...}  
Default: 0

AddUserScriptPasswordInEnvironment is a parameter of the AdvancedConfig (page 600) directive. If this parameter is set to 1, then CMDaemon passes the CMD_USER_PASSWORD environment variable to the script defined by the AddUserScript directive.

**RemoveUserScript directive**

Syntax: AdvancedConfig = {"RemoveUserScript=<path>", ...}  
Default: none

RemoveUserScript is a parameter of the AdvancedConfig (page 600) directive. If this parameter is set to a path leading to a script, and if an existing user is removed using cmsh or cmgui, then the script is automatically run by CMDaemon. The script has a default timeout of 5 seconds.

**AddUserScriptTimeout directive**

Syntax: AdvancedConfig = {"AddUserScriptTimeout=<number>", ...}  
Default: 5

AddUserScriptTimeout is a parameter of the AdvancedConfig (page 600) 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 600) directive. It sets the timeout value in seconds, for the script set by RemoveUserScript.

**AdditionalSubmitHosts directive**

**AdditionalExecHosts directive**

Syntax:
    AdvancedConfig = {"AdditionalSubmitHosts=<host1>,<host2>,...", ...}  
    or  
    AdvancedConfig = {"AdditionalExecHosts=<host1>,<host2>,...", ...}  
Default: none

The AdditionalSubmitHosts and AdditionalExecHosts directives are parameters of the AdvancedConfig (page 600) directive. These directives can be used to make Bright Cluster Manager aware of the existence of a submit host or execution host that is outside of Bright Cluster Manager control. The directives are typically used with SGE and UGE workload managers 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 259.

**MicLDAPBase directive**

Syntax: AdvancedConfig = {"MicLDAPBase=<domain>"}
Default: none

The MicLDAPBase directive is a parameter of the AdvancedConfig (page 600) directive. Setting it to a domain overrides the default value of the LDAP base address on the MIC when its /etc/ldap.conf is initialized.

**MicLDAPServer directive**

Syntax: AdvancedConfig = {"MicLDAPServer=<server>"}
Default: none

The MicLDAPServer directive is a parameter of the AdvancedConfig (page 600) directive. Setting it to a server name overrides the default value of the MIC LDAP server name when its /etc/ldap.conf is initialized.

**MicLDAPForceCustomBase directive**

Syntax: AdvancedConfig = {"MicLDAPForceCustomBase=1|0"}
Default: 0

The MicLDAPForceCustomBase directive is a parameter of the AdvancedConfig (page 600) directive. Setting it to 1 overrides the default globalnetwork base address used for the base LDAP address with the MicLDAPBase address.

**PXEBootOverHTTP directive**

Syntax: GlobalConfig = {"PXEBootOverHTTP=1|0"}
Syntax: AdvancedConfig = {"PXEBootOverHTTP=1|0"}
Default: 1

The PXEBootOverHTTP directive is a parameter of the GlobalConfig (page 601) directive. It is also a parameter of the AdvancedConfig (page 600) directive.

The AdvancedConfig directive is set on the cmd.conf per node image to apply it to the associated nodes. The GlobalConfig directive is set on the cmd.conf file of the head node(s), which applies the value to all nodes.

If the directive is set to a value of 0, then the provisioner sends the initial boot loader, kernel, and ramdisk via TFTP, to the provisioning node.

If the directive is set to a value of 1, which is the default value, then

- The initial boot loader is still loaded up via TFTP, since very few devices support boot loading over HTTP. However, the kernel and ramdisk are now loaded up via HTTP, which is faster and more reliable.
- The administrator must manually set the value of SSLPortOnly (page 586) to no, if needed.
- CMDaemon manages the configuration of /etc/dhcpd.conf and /etc/dhcpd.<network name>.conf on the head node, unless these files are frozen (page 596).
- CMDaemon provides the HTTP server on port 8080, accessible via http://master:8080/tftpboot.

**AutomaticMountAll directive**

**Syntax:** AutomaticMountAll=0|1  
**Default:** 1

If the `AutomaticMountAll` directive is set to the default of 1, then a `mount -a` operation is carried out when a mount change is carried out by CMDaemon.

The `mount -a` operation is to do with attempting to mount devices listed in `/etc/fstab`. It should not be confused with auto-mounting of filesystems, which is to do with mounting an arbitrary device to a filesystem automatically.

If the `AutomaticMountAll` directive is set to 0, then `/etc/fstab` is written, but the `mount -a` command is not run by CMDaemon. However, the administrator should be aware that since `mount -a` is run by the distribution during booting, a node reboot implements the mount change.

**AllowImageUpdateWithAutoMount directive**

**Syntax:** AdvancedConfig = {"AllowImageUpdateWithAutoMount=0|1|2|3"}  
**Default:** 0

The `AllowImageUpdateWithAutoMount` directive is a parameter of the AdvancedConfig (page 600) directive. The values it takes decide how an auto-mounted filesystem should be dealt with during image updates. It must be set per software image.

<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>.</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 Bright Cluster Manager is discussed in section 3.10. The need for adding the automounted filesystem to `excludelistupdate` is discussed on page 199.

**EtcdFlanneldResource directive**

**Syntax:** AdvancedConfig = {"EtcdFlanneldResource=<string>"}  
**Default:** EtcdFlanneldResource=/coreos.com/network/config

The `EtcdFlanneldResource` directive is a parameter of the AdvancedConfig (page 600) directive. The string value it takes—not necessarily a file name—is a key within the `etcd` key-value store, and is used to get a configuration for the Flanneld resource from the store.

**FlannelSubnetLen directive**

**Syntax:** AdvancedConfig = {"FlannelSubnetLen=<number>"}  
**Default:** FlannelSubnetLen=24

© Bright Computing, Inc.
The FlannelSubnetLen directive is a parameter of the AdvancedConfig (page 600) directive. It can take a value from 1 to 31, indicating the Flannel SubnetLen value. SubnetLen is a word from Flannel terminology—in Bright Cluster Manager networking terminology it is normally called prefix-length, or netmask bit size. The default value of 24 corresponds to 256 subnetworks.

**FlannelBackend directive**

**Syntax:**

```
AdvancedConfig = {"FlannelBackend=<string>"}
```

**Default:** FlannelBackend=/coreos.com/network/config

The FlannelBackend directive is a parameter of the AdvancedConfig (page 600) directive. The string value it takes—not necessarily a file name—is a key within the etcd key-value store, and is used to get a configuration for the Flanneld resource from the store.

**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 600) 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 591) 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.

**CassandraDefaultDataCenter directive**

**Syntax:**

```
AdvancedConfig = {"CassandraDefaultDataCenter=<string>"}
```

**Default:** CassandraDefaultDataCenter=DC1

The CassandraDefaultDataCenter directive is a parameter of the AdvancedConfig (page 600) directive. It sets the default datacenter for Cassandra (page 75 of the Big Data Deployment Manual). The default default value is DC1.

**CassandraDefaultRack directive**

**Syntax:**

```
AdvancedConfig = {"CassandraDefaultRack=<string>"}
```

**Default:** CassandraDefaultRack=RAC1

The CassandraDefaultRack directive is a parameter of the AdvancedConfig (page 600) directive. It sets the default rack for Cassandra (page 75 of the Big Data Deployment Manual). The default default value is RAC1.

**AllowNullCipherNetwork directive**

**Syntax:**

```
AdvancedConfig = {"AllowNullCipherNetwork=0|1"}
```

**Default:** AllowNullCipherNetwork=1

The AllowNullCipherNetwork directive is a parameter of the AdvancedConfig (page 600) directive. If it uses the default value of 1, which is recommended, then communication on internal networks is unencrypted. Changing the value to 0 sets encryption on for the internal network, and should only be done for unusual cases.

© Bright Computing, Inc.
CipherList directive

Syntax: AdvancedConfig = {"CipherList=<ciphers>"}
Default: CipherList=ALL:!aNULL:!eNULL

The CipherList directive is a parameter of the AdvancedConfig (page 600) 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@bright73 ~]# 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"}

JobInformationDisabled directive

Syntax: AdvancedConfig = {"JobInformationDisabled=0|1"}
Default: JobInformationDisabled=0

The JobInformationDisabled directive is a parameter of the AdvancedConfig (page 600) directive. If set to 1 it fully disables job-based monitoring (section 10.10).

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.
Disk Partitioning And RAID Configuration

Disk partitioning is initially configured on the head node and regular nodes during installation (section 3.3.17 of the Installation Manual).

For the head node it cannot be changed from within the Bright Cluster Manager after implementation, and the head node is therefore not considered further in this section.

For regular nodes, partitioning can be changed after the initial configuration, by specifying a particular set of values according to the XML partitioning schema described in section D.1.

For example, for regular nodes, changing the value set for the XML tag of:

```
<xs:element name='filesystem'>
```

decides which filesystem type out of ext2, ext3, ext4, or xfs 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 Bright Cluster Manager, regular node partitioning setups have their global structure defined using an XML schema, which is installed on the head node in /cm/node-installer/scripts/disks.xsd. This schema does not include a hardware RAID description. The hardware RAID schema is defined separately in the file raid.xsd (section D.2).

Examples of schemas in use, with and without hardware RAID, are given in sections D.3 and beyond.

An XML file can be validated against a schema using the xmllint tool:

```
Example
[root@bright73 ~]# xmllint --noout --schema /cm/node-installer/scripts/disks.xsd \
/cm/shared/apps/cmgui/disk-setup/slave-diskless.xml
/cm/shared/apps/cmgui/disk-setup/slave-diskless.xml validates
[root@bright73 ~]#
```

© Bright Computing, Inc.
The disks.xsd schema listed next also uses \ as a continuation mark on the printed page, to indicate the line continues. That is not valid XML. The \ marks, at end of the lines associated with the stripeSize attribute, must therefore be removed and those lines joined up into one big line, to make the XML file machine-parseable.

**XML schema for partitioning**

```xml
<?xml version='1.0'?>
<!--
# 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.

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:sequence>
  </xs:complexType>

<xs:key name='partitionAndRaidIds'>
  <xs:selector xpath='.//raid|.//partition'/>
  <xs:field xpath='$id'/>
</xs:key>
```
<xs:complexType name='diskless'>
  <xs:attribute name='maxMemSize' type='memSize' use='required'/>
</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|2K|4K|8K|16K|32K|64K|128K|256K|512K|1024K|1M|2M|4M|8M|16M|32M|64M|128M|256M|512M|1024M|1GB|2GB|4GB|8GB|16GB|32GB|64GB|128GB|256GB|512GB|1024GB|1TB|2TB|4TB|8TB|16TB|32TB|64TB|128TB|256TB|512TB|1024TB|1EB|2EB|4EB|8EB|16EB|32EB|64EB|128EB|256EB|512EB|1024EB|1ZB|2ZB|4ZB|8ZB|16ZB|32ZB|64ZB|128ZB|256ZB|512ZB|1024ZB|1YB|2YB|4YB|8YB|16YB|32YB|64YB|128YB|256YB|512YB|1024YB'/>)
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='size'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='max|auto|([0-9]+[MGT])'/>)
  </xs:restriction>
</xs:simpleType>
<xs: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:group ref="filesystem" minOccurs="0" maxOccurs="1"/>
    <xs:element name="priority" minOccurs="0" maxOccurs="1">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="32767"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="partitiontype" type="xs:string"/>
</xs:complexType>

<xs:group name="filesystem">
  <xs:sequence>
    <xs:element name="filesystem">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ext2"/>
          <xs:enumeration value="ext3"/>
          <xs:enumeration value="ext4"/>
          <xs:enumeration value="xfs"/>
          <xs:enumeration value="btrfs"/>
          <xs:enumeration value="zfs"/>
          <xs:enumeration value="fat32"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="mkfsFlags" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="mountPoint" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="mountOptions" type="xs:string" default="defaults"/>
  </xs:sequence>
</xs:group>
<xs:complexType name='raid'>
  <xs:sequence>
    <xs:element name='member' type='xs:string' minOccurs='2' maxOccurs='unbounded'/>
    <xs:element name='level' type='xs:int'/>
    <xs:choice minOccurs='0' maxOccurs='1'>
      <xs:group ref='filesystem'/>
      <xs:sequence>
        <xs:element name='swap'><xs:complexType /></xs:element>
        <xs:element name='priority' minOccurs='0' maxOccurs='1'>
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="32767"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name='id' type='xs:string' use='required'/>
</xs:complexType>

<xs:complexType name='volumeGroup'>
  <xs:sequence>
    <xs:element name='name' type='xs:string'/>
    <xs:element name='extentSize' type='extentSize'/>
    <xs:element name='physicalVolumes'>
      <xs:complexType>
        <xs:sequence>
          <xs:element name='member' type='xs:string' minOccurs='1' maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name='logicalVolumes'>
      <xs:complexType>
        <xs:sequence>
          <xs:element name='volume' type='logicalVolume' minOccurs='1' maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name='logicalVolume'>
  <xs:sequence>
    <xs:element name='name' type='xs:string'/>
    <xs:element name='size' type='size'/>
    <xs:element name='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:sequence>
</xs:complexType>
D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File

Example 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/node-installer/scripts/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) 2004-2010 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 hardware RAID layout XML file. It can be used by software to validate partitioning XML files.

There are however a few things the schema does not check:
- All of the spans (drive groups) in an raidArray must have the same number of drives.
- There can only be one volume with a 'max' size on a particular array and this must be the last volume in the array.
- If there is only one enclosure defined for a particular RAID controller the actual enclosureID can be omitted by using "auto" instead. Otherwise, the actual enclosureID must be specified.
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="raidLevel">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:restriction pattern="[0-9]+"/>
  </xs:restriction>
</xs:simpleType>
<xs:pattern value="0|1|5|10|50"/>
</xs:restriction>
</xs:simpleType>

<xs:simpleType name="volumeSize">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]1,5[MGT]|max"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="stripeSize">
  <xs:restriction base="xs:string">
    <xs:pattern value="8K|16K|32K|64K|128K|256K|512K|1024K"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="cachePolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="Cached|Direct"/>
  </xs:restriction>
</xs:simpleType>

<!--
  NORA : No Read Ahead
  RA  : Read Ahead
  ADRA : Adaptive Read
-->  
<xs:simpleType name="readPolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="NORA|RA|ADRA"/>
  </xs:restriction>
</xs:simpleType>

<!--
  WT  : Write Through
  WB  : Write Back
-->  
<xs:simpleType name="writePolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="WT|WB"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="enclosureID">
  <xs:restriction base="xs:string">
    <xs:pattern value="auto|[0-9]{1,4}"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="slotNumber">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:pattern value="[0-9]{1,2}"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="raidSetup">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="raidArray" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="level" type="raidLevel"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:choice>
        <xs:element name="device" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="enclosureID" type="enclosureID"/>
              <xs:element name="slotNumber" type="slotNumber"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="span" minOccurs="2" maxOccurs="8">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="device" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="enclosureID" type="enclosureID"/>
                    <xs:element name="slotNumber" type="slotNumber"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs: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>
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="UTF-8"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev mode="cloud">/dev/sdb</blockdev>
    <blockdev mode="cloud">/dev/hdb</blockdev>
    <blockdev mode="cloud">/dev/vdb</blockdev>
    <blockdev mode="cloud">/dev/xvdb</blockdev>
    <blockdev mode="cloud">/dev/xvdf</blockdev>
  </device>
  <partition id="a1">
    <size>20G</size>
    <type>linux</type>
    <filesystem>ext3</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </partition>
  <partition id="a2">
    <size>6G</size>
    <type>linux</type>
    <filesystem>ext3</filesystem>
    <mountPoint>/var</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </partition>
  <partition id="a3">
    <size>20G</size>
    <type>linux</type>
    <filesystem>ext3</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </partition>
</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 mul-
tiple 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 remain-
ing 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 filesystems are specified as ext3. Valid alternatives are ext2, ext4 and xfs.
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@bright73 ~]# cmsh
[bright73]# category use default
[bright73->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.

© Bright Computing, Inc.
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

```xml
<raidSetup>
  <raidArray>
    <level>0</level>
    <raidVolume>
      <stripeSize>64K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>NORA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>40G</size>
    </raidVolume>
    <raidVolume>
      <stripeSize>128K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>RA</readPolicy>
      <writePolicy>WB</writePolicy>
      <size>80G</size>
    </raidVolume>
    <raidVolume>
      <stripeSize>256K</stripeSize>
      <cachePolicy>Cached</cachePolicy>
      <readPolicy>ADRA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>100G</size>
    </raidVolume>
    <device>
      <enclosureID>auto</enclosureID>
      <slotNumber>0</slotNumber>
    </device>
    <device>
      <enclosureID>32</enclosureID>
      <slotNumber>1</slotNumber>
    </device>
  </raidArray>
  <raidArray>
    <level>10</level>
    <raidVolume>
      <stripeSize>64K</stripeSize>
```
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>
</diskSetup>
```
The `level` tag specifies the RAID level used. The following are supported:

- 0 (striping without parity)
- 1 (mirroring)
- 4 (striping with dedicated parity drive)
- 5 (striping with distributed parity)
- 6 (striping with distributed double parity)

The `member` tags must refer to an `id` attribute of a `partition` tag, or an `id` attribute of 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>
  </device>
</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/vdb</blockdev>
    <blockdev mode="cloud">/dev/xvdb</blockdev>
    <blockdev mode="cloud">/dev/xvdf</blockdev>
    <partition id="a1">
      <size>512M</size>
      <type>linux</type>
      <filesystem>ext2</filesystem>
      <mountPoint>/boot</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
    <partition id="a2">
      <size>16G</size>
      <type>linux swap</type>
    </partition>
    <partition id="a3">
      <size>max</size>
    </partition>
  </device>
</diskSetup>
```

© Bright Computing, Inc.
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>
  </device>

  <device>
    <blockdev>/dev/sdb</blockdev>
    <partition id="b1">
      <type>linux raid</type>
    </partition>
  </device>
</diskSetup>
```
<size>512M</size>
<type>linux raid</type>
</partition>
<partition id="b2">
<size>max</size>
<type>linux raid</type>
</partition>
</device>

<raid id="r1">
<member>a1</member>
<member>b1</member>
<level>1</level>
<filesystem>ext3</filesystem>
<mountPoint>/boot</mountPoint>
<mountOptions>defaults,noatime,nodiratime</mountOptions>
</raid>

<raid id="r2">
<member>a2</member>
<member>b2</member>
<level>1</level>
</raid>

<volumeGroup>
<name>vg01</name>
<extentSize>8M</extentSize>
<physicalVolumes>
<member>r2</member>
</physicalVolumes>
</volumeGroup>

<logicalVolumes>
<volume>
<name>lv00</name>
<size>50G</size>
<filesystem>ext3</filesystem>
<mountPoint>/</mountPoint>
<mountOptions>defaults,noatime,nodiratime</mountOptions>
</volume>
<volume>
<name>lv01</name>
<size>25G</size>
<filesystem>ext3</filesystem>
<mountPoint>/tmp</mountPoint>
<mountOptions>defaults,noatime,nodiratime</mountOptions>
</volume>
<volume>
<name>lv02</name>
<size>25G</size>
<filesystem>ext3</filesystem>
<mountPoint>/var</mountPoint>
<mountOptions>defaults,noatime,nodiratime</mountOptions>
</volume>
</logicalVolumes>
</volumeGroup>
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%"/>
</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.

- **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
When nodes operate in semi-diskless mode the node-installer always uses \texttt{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.

\section*{D.11 Example: Preventing Accidental Data Loss}

Optional tags, \texttt{vendor} and \texttt{requiredSize}, can be used to prevent accidentally repartitioning the wrong drive. Such a tag use is shown in the following example.

\textbf{Example}

\begin{verbatim}
<?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>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
</diskSetup>
\end{verbatim}
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 in a multi-drive system is not detected (e.g. due to a hardware failure) 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, as discussed in 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>
</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.
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.

E.1 When Are They Used?

They are sometimes used as an alternative configuration option out of a choice of other possible options (section 3.15.1). As a solution it can be a bit of a hack, but sometimes there is no reasonable alternative other than using an initialize or finalize script.

An initialize script: is used well before the init process starts, to execute custom commands before partitions and mounting devices are checked. Typically, initialize script commands are related to partitioning, mounting, or initializing special storage hardware. Often an initialize script is needed because the commands in it cannot be stored persistently anywhere else.

A finalize script: (also run before init, but shortly before init starts) is used to set a file configuration or to initialize special hardware, sometimes after a hardware check. It is run in order to make software or hardware work before, or during the later init stage of boot. Thus, often a finalize script is needed because its commands must be executed before init, and the commands cannot be stored persistently anywhere else, or it is needed because a choice between (otherwise non-persistent) configuration files must be made based on the hardware before init starts.

E.2 Accessing From cmgui And cmsh

The initialize and finalize scripts are accessible for viewing and editing:

- In cmgui, using the “Node Categories” or Nodes resource, under the Settings tabbed pane for the selected item.
- 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
Example initialize And finalize Scripts

For the initialize and finalize scripts, node-specific customizations can be made from a script using environment variables. The following table shows the available variables with some example values:
### E.3 Environment Variables Available To \texttt{initialize} And \texttt{finalize} Scripts

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CMD_ACTIVE_MASTER_IP}</td>
<td>10.141.255.254</td>
</tr>
<tr>
<td>\texttt{CMDCATEGORY}</td>
<td>default</td>
</tr>
<tr>
<td>\texttt{CMDCHASSIS}</td>
<td>chassis01</td>
</tr>
<tr>
<td>\texttt{CMDCHASSIS_IP}</td>
<td>10.141.1.1</td>
</tr>
<tr>
<td>\texttt{CMDCHASSIS_PASSWORD}</td>
<td>ADMIN</td>
</tr>
<tr>
<td>\texttt{CMDCHASSIS_SLOT}</td>
<td>1</td>
</tr>
<tr>
<td>\texttt{CMDCHASSIS_USERNAME}</td>
<td>ADMIN</td>
</tr>
<tr>
<td>\texttt{CMDCLUSTERNAME}</td>
<td>Bright 7.3 Cluster</td>
</tr>
<tr>
<td>\texttt{CMDDEVICE_HEIGHT}</td>
<td>1</td>
</tr>
<tr>
<td>\texttt{CMDDEVICEPOSITION}</td>
<td>10</td>
</tr>
<tr>
<td>\texttt{CMDDEVICETYPE}</td>
<td>SlaveNode</td>
</tr>
<tr>
<td>\texttt{CMDETHERNETSWITCH}</td>
<td>switch01:1</td>
</tr>
<tr>
<td>\texttt{CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_ALLOWWRITE}</td>
<td>no</td>
</tr>
<tr>
<td>\texttt{CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_HOSTS}</td>
<td>10.141.0.0/16</td>
</tr>
<tr>
<td>\texttt{CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_PATH}</td>
<td>/cm/node-installer</td>
</tr>
<tr>
<td>\texttt{CMD_FSEXPORTS}</td>
<td>_SLASH_cm_SLASH_node-installer</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_cm_SLASH_shared_DEVICE}</td>
<td>master:/cm/shared</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_cm_SLASH_shared_FILESYSTEM}</td>
<td>nfs</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_cm_SLASH_shared_MOUNTPOINT}</td>
<td>/cm/shared</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_cm_SLASH_shared_OPTIONS}</td>
<td>rsize=32768,wsize=32768,hard,intr,async</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_pts_DEVICE}</td>
<td>none</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_pts_FILESYSTEM}</td>
<td>devpts</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_pts_MOUNTPOINT}</td>
<td>/dev/pts</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_pts_OPTIONS}</td>
<td>gid=5,mode=620</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_shm_DEVICE}</td>
<td>none</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_shm_FILESYSTEM}</td>
<td>tmpfs</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_shm_MOUNTPOINT}</td>
<td>/dev/shm</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_dev_SLASH_shm_OPTIONS}</td>
<td>defaults</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_home_DEVICE}</td>
<td>master:/home</td>
</tr>
<tr>
<td>\texttt{CMD_FSMOUNT__SLASH_home_FILESYSTEM}</td>
<td>nfs</td>
</tr>
</tbody>
</table>

...continues
Table E: Environment Variables For The \textit{initialize} And \textit{Finalize} Scripts...continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<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, 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>
<tr>
<td>CMD_FSMOUNT_SLASH_proc_MOUNTPOINT</td>
<td>/proc</td>
</tr>
<tr>
<td>CMD_FSMOUNT_SLASH_proc_OPTIONS</td>
<td>defaults, nosuid</td>
</tr>
<tr>
<td>CMD_FSMOUNT_SLASH_sys_DEVICE</td>
<td>none</td>
</tr>
<tr>
<td>CMD_FSMOUNT_SLASH_sys_FILESYSTEM</td>
<td>sysfs</td>
</tr>
<tr>
<td>CMD_FSMOUNT_SLASH_sys_MOUNTPOINT</td>
<td>/sys</td>
</tr>
<tr>
<td>CMD_FSMOUNT_SLASH_sys_OPTIONS</td>
<td>defaults</td>
</tr>
<tr>
<td>CMD_FSMOUNTS *</td>
<td>_SLASH_dev_SLASH_pts _SLASH_proc _SLASH_sys _SLASH_dev_SLASH_shm _SLASH_cm_SLASH_shared _SLASH_home</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>00:00:00:00:00:01 **</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></td>
</tr>
<tr>
<td>CMD_POWER_CONTROL</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>00000000a000</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\_proc \_SLASH\_sys \_SLASH\_dev\_SLASH\_shm ..."

** The name of this variable varies according to the interfaces available. So, eth0 can be replaced by eth1, eth2, ipmi0, and so on.

© Bright Computing, Inc.
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 code example below shows how they can be used:

Example

```
#!/bin/bash

for interface in $CMD_INTERFACES
  do
    eval type=$CMD_INTERFACE_${interface}_TYPE
    eval ip=$CMD_INTERFACE_${interface}_IP
    eval mask=$CMD_INTERFACE_${interface}_NETMASK

    echo "$interface type=$type"
    echo "$interface ip=$ip"
    echo "$interface netmask=$mask"
  done
```

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__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 cmgui or cmsh (section 3.10).

E.5 Storing A Configuration To A Filesystem

© Bright Computing, Inc.
E.5.1 Storing With Initialize Scripts

The initialize script (section 5.4.5) runs after the install-mode type and execution have been determined (section 5.4.4), but before unloading specific drivers and before partitions are checked and filesystems mounted (section 5.4.6). Data output cannot therefore be written to a local drive. It can however be written by the script to the tmpfs, but data placed there is lost quite soon, namely during the pivot_root process that runs when the node-installer hands over control to the init process running from the local drive. However, if needed, the data can be placed on the local drive later by using the finalize script to copy it over from the tmpfs.

Due to this, and other reasons, a finalize script is easier to use for an administrator than an initialize script, and the use of the finalize script is therefore preferred.

E.5.2 Ways Of Writing A Finalize Script To Configure The Destination Nodes

Basic Example—Copying A File To The Image

For a finalize script (section 5.4.11), which runs just before switching from using the ramdrive to using the local hard drive, the local hard drive is mounted under /localdisk. Data can therefore be written to the local hard drive if needed, but is only persistent until a reboot, when it gets rewritten. For example, predetermined configuration files can be written from the NFS drive for a particular node, or they can be written from an image prepared earlier and now running on the node at this stage, overwriting a node-installer configuration:

Example

#!/bin/bash
cp /etc/myapp.conf.overwrite /localdisk/etc/myapp.conf

This technique is used in a finalize script example in section 3.15.4, except that an append operation is used instead of a copy operation, to overcome a network issue by modifying a network configuration file slightly.

There are two important considerations for most finalize scripts:

1. Running A Finalize Script Without exit 0 Considered Harmful

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. 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.

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).

Copying A File To The Image—Decision Based On Detection

Detection within a basic finalize script is useful extra technique. The finalize script example of section 3.15.4 does detection too, to decide if a configuration change is to be done on the node or not.

A further variation on a finalize script with detection is a script selecting from a choice of possible configurations. A symlink is set to one of the possible configurations based on hardware detection or
detection of an environment variable. The environment variable can be a node parameter or similar, from the table in section E.3. If it is necessary to overwrite different nodes with different configurations, then the previous finalize script example might become something like:

**Example**

```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 Environmental Variables Evaluated In The File**

Sometimes there can be a need to use the CMDaemon environmental 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**

```bash
SPECIALSERVICEPARAMETER=myhost
```

Ideally the placeholder value myhost would be the hostname of the node rather than the fixed value myhost. Conveniently, the CMDaemon environmental 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 environmental variable:

```bash
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/\$\{\([^}\]+\)\}/defined $ENV{$1} ? $ENV{$1} : $&/eg' /localdisk/some/directory/file
```

Here, `/some/directory/file` means that, if for example the final configuration file path for the node is to be `/var/spool/test` then the file name should be set to `/localdisk/var/spool/test` inside the finalize script.

The finalize script replaces all lines within the file that have environmental variable names of the form:

```bash
PARAMETER=${<environmental variable name>}
```

with the value of that environment variable. Thus, if `<environmental 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.
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 cmgui or cmsh front end tools for CMDaemon (section 7.4).

From the command line, direct Slurm commands that may sometimes come in useful include the following:

- **sacct**: used to report job or job step accounting information about active or completed jobs.

  **Example**

  ```
  # sacct -j 43 -o jobid,AllocCPUs,NCPUS,NNodes,NTasks,ReqCPUs
<table>
<thead>
<tr>
<th>JobID</th>
<th>AllocCPUs</th>
<th>NCPUS</th>
<th>NNodes</th>
<th>NTasks</th>
<th>ReqCPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
  
  ```

- **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**

© Bright Computing, Inc.
• **scontrol**: shows the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

**Example**

```bash
[fred@bright73 ~]$ scontrol show nodes
NodeName=bright73 Arch=x86_64 CoresPerSocket=1
    CPUAlloc=0 CPUErr=0 CPUTot=1 CPULoad=0.05 Features=(null)
...
```

• **sinfo**: reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

**Example**

```bash
bright73:~ # sinfo -o "%9P %.5a %.10l %.6D %.6t %C %N"
PARTITION AVAIL TIMELIMIT NODES STATE CPUS(A/I/O/T) NODELIST
defq* up infinite 1 alloc 1/0/0/1 bright73
```

• **squeue**: reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

**Example**

```bash
bright73:~ # squeue -o "%.18i %.9P %.8j %.8u %.2t %.10M %.6D %C %R"
JOBID PARTITION NAME USER ST TIME NODES CPUS NODELIST(REASON)
  43 defq bash fred R 16:22 1 1 bright73
...
```

• **srun**: used to submit a job for execution or initiate job steps in real time. srun has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job’s node allocation.

• **smap**: reports state information for jobs, partitions, and nodes managed by Slurm, but graphically displays the information to reflect network topology.

• **strigger**: used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.

• **sview**: a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

There are man pages for these commands. Full documentation on Slurm is available online at: [http://slurm.schedmd.com/documentation.html](http://slurm.schedmd.com/documentation.html).

### F.2 Sun Grid Engine

Sun Grid Engine (SGE) is a workload management system that was originally made available under an Open Source license by Sun Microsystems. It forked off into various versions in 2010 and its future is unclear at the time of writing, but it remains in widespread use. Bright Cluster Manager 7.3 uses version 6.2 update 5 patch 2 of Grid Scheduler, which is a bugfix-patched version of the last SGE release from Sun Microsystems, and is made available on sourceforge at [http://gridscheduler.sourceforge.net/](http://gridscheduler.sourceforge.net/).

SGE services should be handled using CMDaemon, as explained in section 7.4. However SGE can break in obtuse ways when implementing changes, so the following notes are sometimes useful in getting a system going again:
The `sge_qmaster` daemon on the head node can be started or stopped using
`/etc/init.d/sgemaster.sge1start` | `stop`, or alternatively via `qconf -{s|k}m`.

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 started or stopped via
`/etc/init.d/sgeexecd start|stop`, or alternatively deregistered from `qmaster` via `qconf -{s|k}s`.

To see why a queue is in an error state, `qstat -explain E` shows the reason. Queues in an error state can be cleared with a `qmod -c <queue name>`.

SGE can be configured and managed generally with the command line utility `qconf`, which is what most administrators become familiar with. A GUI alternative, `qmon`, is also provided.

SGE commands are listed below. The details of these are in the `man` page of the command and the SGE documentation.

- `qalter`: modify existing batch jobs
- `qacct`: show usage information from accounting data
- `qconf`: configure SGE
- `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 SGE with an X11 GUI interface
- `qping`: check `sge_qmaster` and `sge_execd` status
- `qquota`: list resource quotas
- `qresub`: create new jobs by copying existing jobs
- `qrdel`: cancel advance reservations
- `qr1s`: 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`)
- `qtcs`: start `csh`-based interactive session with a node
F.3 Torque

The following commands can be used to manage Torque:

Torque resource manager commands:

- `qalter`: alter batch job
- `qdel`: delete batch job
- `qhold`: hold batch jobs
- `qrls`: release hold on batch jobs
- `qstat`: show status of batch jobs
- `qsub`: submit job
- `qmgr`: batch policies and configurations manager
- `qenable`: enable input to a destination
- `qdisable`: disable input to a destination
- `tracejob`: trace job actions and states

These direct commands are capable of modifying properties that CMDaemon does not manage, as well as properties that CMDaemon does manage. CMDaemon however overwrites the properties that are managed by CMDaemon with values recalled from the CMDaemon database, typically within a few minutes, if the values are changed directly via the direct commands.

Further information on these and other commands is available in the appropriate man pages and on-line documentation at http://www.adaptivecomputing.com/resources/docs/. The Torque administrator manual is also available from there.

F.4 PBS Pro

The following commands can be used in PBS Pro to view queues:

```
qstat query queue status
qstat -a alternate form
qstat -r show only running jobs
qstat -q show available queues
qstat -rn only running jobs, w/ list of allocated nodes
qstat -i only idle jobs
qstat -u username show jobs for named user
```

Other useful commands are:

```
tracejob id show what happened today to job id
tracejob -n d id search last d days
qmgr administrator interface to batch system
qterm terminates queues (but cm starts pbs_server again)
pbsnodes -a list available worker nodes in queue
```

The commands of PBS Pro are documented in the PBS Professional 11.0 Reference Guide. There is further extensive documentation for PBS Pro administrators in the PBS Professional 11.0 Administrator’s Guide. Both of these guides are available at the PBS Works website at http://www.pbsworks.com/SupportDocuments.aspx.
F.5 OpenLava

OpenLava is a free, GNU GPLv2 licensed workload management system based on Platform LSF. The OpenLava source is available on github at https://github.com/openlava/openlava/.

OpenLava is a fork of Platform Lava, released under GNU GPLv2. This in turn is based on a stripped-down version of Platform LSF 4.2. Versions beyond 2.2 are at the time of writing (November 2016) subject to an IBM copyright infringement claim. On request of the OpenLava founder, the packages beyond version 2.2 are not redistributed.

OpenLava services and outputs are normally handled using the cmgui or cmsh front end tools for CMDaemon (section 7.4).

From the command line, direct OpenLava commands that may sometimes come in useful include the following:

- **badmin**: the administrative tool used to control and monitor OpenLava. It provides the following commands: ckconfig, reconfig, mbdrestart, qopen, qclose, qact, qinact, hopen, hclose, hrestart, hshutdown, hstartup, hhist, mbdhist, hist, sbdtime, mbdtime and mbddebug.

- **bjobs**: displays information about OpenLava jobs.

- **brun**: used to force a pending OpenLava job to run immediately on specified hosts.

- **bbot**: used to change the queue position of a pending OpenLava job, or a pending job array element, to affect the order in which jobs are considered for dispatch.

- **bchkpnt**: used to checkpoint one or more checkpointable OpenLava jobs.

- **bhosts**: displays information about static and dynamic resources of hosts.

- **bkill**: sends signals to kill, suspend, or resume unfinished openlava jobs.

- **bmgrep**: displays information about host OpenLava groups.

- **bmig**: used to migrate checkpointable or rerunnable OpenLava jobs.

- **bmod**: used to modify job submission options of an OpenLava job.

- **bparams**: displays information about configurable system parameters in lsb.params.

- **bpeek**: displays the stdout and stderr output of an unfinished OpenLava job.

- **bqueues**: displays information about OpenLava queues.

- **brequeue**: used to kill and requeue an OpenLava job.

- **brestart**: used to restart checkpointed OpenLava jobs.

- **bresume**: used to resume one or more suspended OpenLava jobs.

- **bstop**: used to suspend unfinished OpenLava jobs.

- **bsub**: used to submit a batch job to OpenLava.

- **bswitch**: used to switch unfinished OpenLava jobs from one queue to another.

- **btop**: used to move a pending OpenLava job relative to the first job in the queue.

- **bugroup**: displays information about user groups.

- **busers**: displays information about users and user groups.

© Bright Computing, Inc.
• **lshosts**: displays information about resources defined on hosts.

• **lsload**: displays the current state of the host about its ability to run batch jobs and remote tasks.

• **lsinfo**: displays information about load indices.

• **lsid**: displays the current OpenLava version number, the cluster name, and the active OpenLava master node name.

• **lsmon**: displays load information for OpenLava hosts.

• **lsplace**: displays hosts available to execute OpenLava jobs.

• **lsrcp**: used to copy files remotely using OpenLava.

Full documentation on OpenLava is available online at: [http://openlava.org/documentation/documentation.html](http://openlava.org/documentation/documentation.html).
Metrics, Health Checks, And Actions

This appendix describes the metrics (section G.1), health checks (section G.2), and actions (section G.3), along with their parameters, in a newly-installed cluster. Metrics, health checks, 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 2.6 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

G.1.1 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 10.2.7). The lower it is, the better.</td>
</tr>
<tr>
<td>Average</td>
<td>Cluster-wide average of metric of choice</td>
</tr>
<tr>
<td>AvgExpFactor</td>
<td>Average Expansion Factor. This is by what factor, on average, jobs took longer to run than expected. The expectation is according to heuristics based on duration in past and current job queues, as well as node availability.</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>Average Job Duration of current jobs</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>CacheMemory</td>
<td>System memory used for caching</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cimc</strong></td>
<td>CIMC device metrics collection (file: ucsmetrics)</td>
</tr>
<tr>
<td>CMDActiveSessions</td>
<td>Managed active sessions count</td>
</tr>
<tr>
<td>CMDCycleTime</td>
<td>Time spent by head to process picked up data</td>
</tr>
<tr>
<td>CMDMemUsed</td>
<td>Resident memory used by CMDaemon</td>
</tr>
<tr>
<td>CMDState</td>
<td>State in which CMDaemon is running (head: 0, regular: 1, failover: 2)</td>
</tr>
<tr>
<td>CMDStoreQueryTime</td>
<td>Time spent by head node to store monitoring data to database</td>
</tr>
<tr>
<td><strong>CMDsystime</strong></td>
<td>Percent of time spent by CMDaemon in system mode</td>
</tr>
<tr>
<td><strong>CMDUsertime</strong></td>
<td>Percent of time spent by CMDaemon in user mode</td>
</tr>
<tr>
<td>CompletedJobs</td>
<td>Jobs completed</td>
</tr>
<tr>
<td>CPUCoresAvailable</td>
<td>Cluster-wide number of CPU cores</td>
</tr>
<tr>
<td><strong>CPUIdle</strong></td>
<td>Percent of node-wide core time spent in idle mode. Example: The CPUIdle time measurement on a 1-core machine over 1 second can be up to 1 second (no time in this mode = 0s, all of its time in this mode = 1s). The CPUIdle time measurement on a 2-core machine over 1 second can be up to 2 seconds at the most. The derived CPUIdle metric for a 1-core system can be up to 100% at the most. The derived CPUIdle metric for a 2-core system can be up to 200% at the most.</td>
</tr>
<tr>
<td>CPUIrq</td>
<td>Percent of node-wide core time spent in servicing interrupts</td>
</tr>
<tr>
<td><strong>CPUNice</strong></td>
<td>Percent of node-wide core time spent in nice user mode</td>
</tr>
<tr>
<td>CPUSoftIq</td>
<td>Percent of node-wide core time spent in servicing soft interrupts</td>
</tr>
<tr>
<td>CPUSystem</td>
<td>Percent of node-wide core time spent in system mode</td>
</tr>
<tr>
<td><strong>CPUUser</strong></td>
<td>Percent of node-wide core time spent in user mode</td>
</tr>
<tr>
<td>CPUWait</td>
<td>Percent of node-wide core time spent in waiting for I/O to complete</td>
</tr>
<tr>
<td><strong>CtxtSwitches</strong></td>
<td>Context switches/s</td>
</tr>
<tr>
<td><strong>Current_1</strong></td>
<td>First current seen by BMC sensor, in amps (file: sample_ipmi)</td>
</tr>
<tr>
<td><strong>Current_2</strong></td>
<td>Second current seen by BMC sensor, in amps (file: sample_ipmi)</td>
</tr>
<tr>
<td>DevicesUp</td>
<td>Number of devices in status UP. A node (head, regular, cloud, VSMP) 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</td>
<td>Packets/s received and dropped</td>
</tr>
<tr>
<td>DropSent</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</td>
<td>Total number of double bit ECC errors (file: sample_gpu)</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>EccSBitGPU**</td>
<td>Total number of single bit ECC errors (file: sample_gpu)</td>
</tr>
<tr>
<td>ErrorsRecv*</td>
<td>Packets/s received with error</td>
</tr>
<tr>
<td>ErrorsSent*</td>
<td>Packets/s sent with error</td>
</tr>
<tr>
<td>EstimatedDelay</td>
<td>Estimated delay time to execute jobs</td>
</tr>
<tr>
<td>Exhaust_Temp</td>
<td>Exhaust temperature measured by BMC, in Celsius (file: sample_ipmi)</td>
</tr>
<tr>
<td>FailedJobs</td>
<td>Failed jobs</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>FanSpeedPercGPU**</td>
<td>Percent of full fan speed for GPU &lt;number&gt; (file: sample_gpu)</td>
</tr>
<tr>
<td>Forks*</td>
<td>Forks/s</td>
</tr>
<tr>
<td>FrameErrors*</td>
<td>Packet framing errors/s</td>
</tr>
<tr>
<td>FreeSpace</td>
<td>Free space for non-root. Takes mount point as a parameter</td>
</tr>
<tr>
<td>GPUAvailable</td>
<td>Cluster-wide number of GPUs</td>
</tr>
<tr>
<td>gpu**</td>
<td>GPU metrics collection (file: sample_gpu)</td>
</tr>
<tr>
<td>GpuUtilGPU**</td>
<td>Percent Utilization of GPU &lt;number&gt; (file: sample_gpu)</td>
</tr>
<tr>
<td>ilo**</td>
<td>iLO metrics collection (file: sample_ilo)</td>
</tr>
<tr>
<td>Inlet_Temp**</td>
<td>Inlet Temperature, in Celsius (file: sample_ipmi)</td>
</tr>
<tr>
<td>IOInProgress</td>
<td>I/O operations in progress</td>
</tr>
<tr>
<td>IOTime*</td>
<td>I/O operations time in milliseconds/s</td>
</tr>
<tr>
<td>ipForwDatagrams*</td>
<td>Input IP datagrams/s to be forwarded</td>
</tr>
<tr>
<td>ipFragCreates*</td>
<td>IP datagram fragments/s generated</td>
</tr>
<tr>
<td>ipFragFails*</td>
<td>IP datagrams/s which needed to be fragmented but could not</td>
</tr>
<tr>
<td>ipFragOKs*</td>
<td>IP datagrams/s successfully fragmented</td>
</tr>
<tr>
<td>ipInAddrErrors*</td>
<td>Input datagrams/s discarded because the IP address in their header was not a valid address</td>
</tr>
<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>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipmi**</td>
<td>IPMI metrics collection (file: sample_ipmi)</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>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>lustre**</td>
<td>Lustre filesystem metrics collection (file: sample_lustre)</td>
</tr>
<tr>
<td>MajorPageFaults*</td>
<td>Page faults/s that require I/O</td>
</tr>
<tr>
<td>Max</td>
<td>Cluster-wide maximum for metric specified as parameter in metric configuration</td>
</tr>
<tr>
<td>MemoryFreeGPU**</td>
<td>Memory free for GPU &lt;number&gt; (file: sample_gpu)</td>
</tr>
<tr>
<td>MemoryFree</td>
<td>Free system memory</td>
</tr>
<tr>
<td>MemoryUsed</td>
<td>Used system memory</td>
</tr>
<tr>
<td>MemUsedGPU**</td>
<td>Memory used by GPU &lt;number&gt; (file: sample_gpu)</td>
</tr>
<tr>
<td>MemUtilGPUs</td>
<td>Percent of memory utilization by GPU &lt;number&gt; (file: sample_gpu)</td>
</tr>
<tr>
<td>MergedReads*</td>
<td>Merged reads/s</td>
</tr>
<tr>
<td>MergedWrites*</td>
<td>Merged writes/s</td>
</tr>
<tr>
<td>mic</td>
<td>MIC metrics collection (file: sample_mic)</td>
</tr>
<tr>
<td>MICsUP</td>
<td>Number of MICs in status UP</td>
</tr>
<tr>
<td>Min</td>
<td>Cluster-wide minimum for metric specified as parameter in metric configuration</td>
</tr>
<tr>
<td>NetworkBytesRecv</td>
<td>Cluster-wide number of bytes received on all networks</td>
</tr>
<tr>
<td>NetworkBytesSent</td>
<td>Cluster-wide number of bytes transmitted on all networks</td>
</tr>
<tr>
<td>NetworkUtilization</td>
<td>Network utilization percentage</td>
</tr>
<tr>
<td>NodesInQueue</td>
<td>Number of nodes in the 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: LoadOne on each node is mapped to a value, calibrated so that LoadOne=1 corresponds to 100% per node. The maximum allowed for a node in the mapping is 100%. The average of these mappings taken over all nodes is the OccupationRate.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</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>PDBankLoad</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>Cluster-wide phase load current, in amps (depends on PhaseLoad directive)</td>
</tr>
<tr>
<td>PowerDrawGPU</td>
<td>Power used by GPU &lt;number&gt;, in watts (file: sample_gpu)</td>
</tr>
<tr>
<td>price</td>
<td>Price per hour for this cloud node (approximation)</td>
</tr>
<tr>
<td>ProcessCount</td>
<td>Total number of all processes running in the OS</td>
</tr>
<tr>
<td>Pwr_Consumption</td>
<td>Power consumed by BMC, in watts (file: sample_ipmi)</td>
</tr>
<tr>
<td>QueuedJobs</td>
<td>Queued jobs</td>
</tr>
<tr>
<td>RackSensorHumidity</td>
<td>Rack sensor humidity</td>
</tr>
<tr>
<td>RackSensorTemp</td>
<td>Rack sensor Temperature</td>
</tr>
<tr>
<td>Range*</td>
<td>Cluster-wide range of metric of choice (difference between min and max)</td>
</tr>
<tr>
<td>Reads*</td>
<td>Reads/s completed successfully</td>
</tr>
<tr>
<td>ReadTime*</td>
<td>Read time in milliseconds/s</td>
</tr>
<tr>
<td>responsiveness**</td>
<td>Device I/O requests metrics collection (file: sample_responsiveness)</td>
</tr>
<tr>
<td>RunningJobs</td>
<td>Running jobs</td>
</tr>
<tr>
<td>RunningProcesses</td>
<td>Running processes</td>
</tr>
<tr>
<td>sdt**</td>
<td>SuperMicro metrics collection (file: sample_sdt)</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>SensorFanSpeed</td>
<td>System or CPU fan speed detected</td>
</tr>
<tr>
<td>SensorTemp</td>
<td>Temperature (system and CPU) detected</td>
</tr>
<tr>
<td>SensorVoltage</td>
<td>Motherboard voltage detected</td>
</tr>
<tr>
<td>SMARTHDATemp</td>
<td>Temperature of a Hard Disk Assembly</td>
</tr>
<tr>
<td>SMARTReallocSecCnt</td>
<td>SMART reallocated sectors count</td>
</tr>
<tr>
<td>SMARTSeekErrRate</td>
<td>SMART seek errors/s</td>
</tr>
<tr>
<td>SMARTSeekTimePerf</td>
<td>SMART average seek time</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMARTSoftReadErrRate</td>
<td>SMART software read errors/s</td>
</tr>
<tr>
<td>SSLCacheHitRatio</td>
<td>SSL connections that are served by the SSL cache (%). If this is 100, then all connections are cached, and the cluster is serving everything from cache</td>
</tr>
<tr>
<td>SSLCacheUsage</td>
<td>SSL cache used (%). If this is 100, then there are too many SSL connections.</td>
</tr>
<tr>
<td>Sum</td>
<td>Cluster-wide sum of metric of choice</td>
</tr>
<tr>
<td>SwapFree</td>
<td>Free swap space</td>
</tr>
<tr>
<td>SwapUsed</td>
<td>Used swap space</td>
</tr>
<tr>
<td>SwitchBroadcastPackets*</td>
<td>Total number of good packets received and directed to the broadcast address/s</td>
</tr>
<tr>
<td>SwitchCollisions*</td>
<td>Collisions/s on this network segment</td>
</tr>
<tr>
<td>SwitchCPUUsage</td>
<td>Switch CPU utilization estimation (%)</td>
</tr>
<tr>
<td>SwitchDelayDiscardFrames*</td>
<td>Frames discarded/s due to excessive transit delay through the bridge</td>
</tr>
<tr>
<td>SwitchFilterDiscardFrames*</td>
<td>Valid frames received/s but discarded by the forwarding process</td>
</tr>
<tr>
<td>SwitchMTUDiscardFrames*</td>
<td>Number of frames discarded/s due to an excessive size</td>
</tr>
<tr>
<td>SwitchMulticastPackets*</td>
<td>Total number of good packets/s received and directed to a multicast address</td>
</tr>
<tr>
<td>SwitchOverSizedPackets*</td>
<td>Well-received packets/s longer than 1518 octets</td>
</tr>
<tr>
<td>SwitchTemperature</td>
<td>Switch temperature, in Celsius (for Force10 S-series)</td>
</tr>
<tr>
<td>SwitchUnderSizedPackets*</td>
<td>Packets/s received which are less than 64 octets long</td>
</tr>
<tr>
<td>SwitchUptime*</td>
<td>Switch uptime per second. Ie, ideally=1, but in practice has jitter effects</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 retransmitted</td>
</tr>
<tr>
<td>TempGPU**</td>
<td>Temperature of GPU &lt;number&gt;, in Celsius (file: sample_gpu)</td>
</tr>
<tr>
<td>testcollection**</td>
<td>An example of a metric collection script (file: testmetriccollection)</td>
</tr>
<tr>
<td>testmetric**</td>
<td>An example of a metric script (file: testmetric)</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>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TotalCPUUser</td>
<td>Cluster-wide core usage in user mode (sum of all CPUUser metric percentages)</td>
</tr>
<tr>
<td>TotalMemoryUsed</td>
<td>Cluster-wide total of memory used</td>
</tr>
<tr>
<td>TotalNodes</td>
<td>Cluster-wide total number of nodes</td>
</tr>
<tr>
<td>TotalSwapUsed</td>
<td>Cluster-wide total swap used</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>Uptime*</td>
<td>System uptime per second. Ie, ideally=1, but in practice has jitter effects</td>
</tr>
<tr>
<td>UsedSpace</td>
<td>Total used space by a mount point</td>
</tr>
<tr>
<td>Voltage_1**</td>
<td>First voltage seen by BMC sensor, in Volts (file: sample_ipmi)</td>
</tr>
<tr>
<td>Voltage_2**</td>
<td>Second voltage seen by BMC sensor, in Volts (file: sample_ipmi)</td>
</tr>
<tr>
<td>Writes*</td>
<td>Writes/s completed successfully</td>
</tr>
<tr>
<td>WriteTime*</td>
<td>Write time in milliseconds/s</td>
</tr>
</tbody>
</table>

* Cumulative metric. I.e. the metric is derived from cumulative raw measurements taken at two different times, according to:
\[
\text{metric}_{\text{times}_2} = \frac{\text{measurement}_{\text{times}_2} - \text{measurement}_{\text{times}_1}}{\text{times}_2 - \text{times}_1}
\]

** Standalone scripts, not built-ins.

If sampling from a head node, the script is in directory:
/cm/local/apps/cmd/scripts/metrics/

If sampling from a regular node, the script is in directory:
/cm/images/default-image/cm/local/apps/cmd/scripts/metrics/
### G.1.2 Parameters For Metrics

Metrics have the parameters indicated by the left column in the following example:

**Example**

```
[myheadnode->monitoring->metrics] $ show cpuuser
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class of metric</td>
<td>cpu</td>
</tr>
<tr>
<td>Command</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>Cumulative</td>
<td>yes</td>
</tr>
<tr>
<td>Description</td>
<td>Percent of node-wide core time spent in user mode</td>
</tr>
<tr>
<td>Disabled</td>
<td>no</td>
</tr>
<tr>
<td>Extended environment</td>
<td>no</td>
</tr>
<tr>
<td>Maximum</td>
<td>&lt;range not set&gt;</td>
</tr>
<tr>
<td>Metric Unit</td>
<td>%</td>
</tr>
<tr>
<td>Minimum</td>
<td>&lt;range not set&gt;</td>
</tr>
<tr>
<td>Name</td>
<td>CPUUser</td>
</tr>
<tr>
<td>Only when idle</td>
<td>no</td>
</tr>
<tr>
<td>Parameter permissions</td>
<td>disallowed</td>
</tr>
<tr>
<td>Required resources</td>
<td></td>
</tr>
<tr>
<td>Retrieval method</td>
<td>cmdaemon</td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Sampling method</td>
<td>samplingonnode</td>
</tr>
<tr>
<td>State flapping count</td>
<td>7</td>
</tr>
<tr>
<td>Timeout</td>
<td>5</td>
</tr>
</tbody>
</table>

```
[myheadnode->monitoring->metrics] $```

The meanings of the parameters are:

**Class of metric**: A choice assigned to a metric depending on its type. A partial list of the values and what they are related to is:

- **Misc** (default): miscellaneous class of metrics, used if none of the other classes are appropriate, or if none of the other classes are chosen
  - **cpu**: CPU activity
  - **gpu**: GPU activity
  - **disk**: Disk activity
  - **mem**: Memory activity
  - **net**: Network activity
  - **env**: sensor measurements of the physical environment
  - **os**: operating system activity
  - **internal**: Bright Cluster Manager utilities
  - **workload**: workload management
  - **cluster**: clusterwide measurements
  - **prototype**: the metric collections class
  - **ceph**: Ceph
  - **lustre**: Lustre
  - **cgroups**: Cgroups (several kinds)
  - **dellnss**: Dell NSS

© Bright Computing, Inc.
• openstack: OpenStack class (several kinds)
• hadoop: Hadoop (several kinds)

Command: For a standalone metric script, it is the full path. For a built-in, the value cannot be set, and
the command is simply the name of the metric.

Cumulative: If set to yes, then the raw metric sample is cumulative. This actually means that the
raw sample used to calculate the metric is cumulative, like, for example, the bytes-received counter
for an Ethernet interface. If set to no, then the raw sample value is not cumulative (for example,
temperature). For a cumulative metric, the metric value is calculated from the raw values by
taking the difference in raw measurement values, divided by the time period over which they are
sampled. Thus:
• The bytes-received raw measurements, which accumulate as the packets are received, and
are in bytes, 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 a corresponding metric (Uptime) with 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.

Disabled: If set to no (default) then the script runs.

Extended environment: If set to yes, more information about the device is made part of the envi-
ronment to the script. The default is no.

Metric Unit: A unit for the metric. For example B/s (bytes/second) for BytesRecv metric, or
unitless for the Uptime metric. A percent is indicated with %.

Name: The name given to the metric.

Only when idle: If Only when idle is set to yes, then the metric script only runs when the node
is idling below an idle threshold value. It is set to no by default. Setting this to yes for resource-
hungry metrics burdens the node less. The idle value is the value of loadone. The idle threshold
is set by the CMDaemon directive IdleThreshold, and is set by default to 1.0.

Parameter permissions: Decides if a parameter instance type or device can be passed to the metric
script. There are three possible parameter permissions:
• disallowed: parameters are not used
• required: parameters are mandatory
• optional (default): parameters are optional

If parameter permissions are required, then a metric parameter name for a device or instance type
is appended to the metric name with a "::". For example, the metric name BytesRecv has the
device interface name eth0 appended to it as follows:

Example

BytesRecv:eth0

A list of example metric parameter device or instance names can be viewed, for example, for the
head node (headnode), using cmsh as follows:

[bright73 ~]# cmsh -c "monitoring setup use headnode; metricconf;\list -d : -f metric:22,metricparam:30 | grep -v ': *:':''
The truncated output of this one-liner looks like:

```
<table>
<thead>
<tr>
<th>metric</th>
<th>:metricparam</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertLevel</td>
<td>:max</td>
</tr>
<tr>
<td>AlertLevel</td>
<td>:sum</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>:default</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>:longq</td>
</tr>
<tr>
<td>AvgJobDuration</td>
<td>:shortq</td>
</tr>
<tr>
<td>BytesRecv</td>
<td>:eth0</td>
</tr>
<tr>
<td>BytesRecv</td>
<td>:eth1</td>
</tr>
</tbody>
</table>
...```

Retrieval method:
- `cmdaemon` (default): Metrics retrieved internally using CMDaemon
- `snmp`: Metrics retrieved internally using SNMP

Sampling method:
- `samplingonmaster`: The head node samples the metric on behalf of a device. For example, the head node may do this for a PDU because the PDU does not have the capability to run the cluster management daemon at present, and so cannot itself pass on data values directly when `cmsh` or `cmgui` need them.
- `samplingonnode` (default): The non-head node samples the metric itself.

State flapping count: How many times the metric value must cross a threshold within the last 12 samples before it is decided that it is in a flapping state. Default value is 7.

Timeout: After how many seconds the command will give up retrying. Default value is 5 seconds.

Maximum: the default minimum value the y-axis maximum will take in graphs plotted in `cmgui`.¹

Minimum: the default maximum value the y-axis minimum will take in graphs plotted in `cmgui`.¹

---

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 resized to 4, and the y-axis will now span from 0 to 4.
## G.2 Health Checks And Their Parameters

### G.2.1 Health Checks

<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL/UNKNOWN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceIsUp</td>
<td>Is the device up, closed or installing? Determined using TCP SYN pings to: port 2 for nodes, port 1 for GPUs, and port 7 for switches and other devices. <em>This health check does not require a CMDaemon running on the device it is checking. It can therefore be used to check on devices that are registered with CMDaemon, but not provisioned by CMDaemon. Thus a switch, but also a generic node that Bright Cluster Manager did not provision, can be checked with this health check, as long as CMDaemon is aware of them.</em></td>
</tr>
<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. After correcting the problem with the service, a reset of the service is normally carried out (section 3.11, page 103).</td>
</tr>
<tr>
<td>cmsh</td>
<td>Is cmsh 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 disk space available to non-root users than any of the space parameters specified?</td>
</tr>
<tr>
<td></td>
<td><em>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 651) instead.</em></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/UNKNOWN)</th>
</tr>
</thead>
</table>

Examples:

- **diskspace 10%**
  less than 10% space = **FAIL**, severity 10

- **diskspace 10% 20% 30%**
  less than 30% space = **FAIL**, with severity levels as indicated:

<table>
<thead>
<tr>
<th>space left</th>
<th>severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>30</td>
</tr>
<tr>
<td>20%</td>
<td>20</td>
</tr>
<tr>
<td>30%</td>
<td>10</td>
</tr>
</tbody>
</table>

- **diskspace 10GB 20GB**
  less than 20GB space = **FAIL**, severity 10
  less than 10GB space = **FAIL**, severity 20

- **diskspace 10% 20% /var**
  For the filesystem /var:
  less than 20% space = **FAIL**, severity 10
  less than 10% space = **FAIL**, severity 20

...continued
<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL/UNKNOWN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmesg</td>
<td>Is dmesg output ok?</td>
</tr>
</tbody>
</table>
|                  | Regexes to parse the output can be constructed in the configuration file at /
|                  | /cm/local/apps/cmd/scripts/healthchecks/configfiles/dmesg.py |
| exports          | Are all filesystems as defined by the cluster management system exported? |
| failedprejob     | Are there failed prejob health checks (section 7.8.2)? Here: yes = FAIL. By default, the job ID is saved under /cm/shared/apps/<scheduler>/var/cm:/ |
|                  | - On FAIL, in failedprejobs.          |
|                  | - On PASS, in allprejobs              |
|                  | The maximum number of IDs stored is 1000 by default. The maximum period for which the IDs are stored is 30 days by default. Both these maxima can be set with the failedprejob health check script. |
| failover         | Is all well with the failover system? |
| gpuhealth_quick  | Is all well with the GPUs according to a quick check? |
|                  | A configuration file for this is at /
|                  | /cm/local/apps/cmd/scripts/healthchecks/configfiles/gpuhealth.ini |
| hpraid           | Are the HP Smart Array controllers OK? |
| ib               | Is the InfiniBand Host Channel Adapter working properly? |
|                  | A configuration file for this health check is at /
|                  | /cm/local/apps/cmd/scripts/healthchecks/configfiles/ib.py |
| ieel_alerts      | Are there not too many alerts (INFO, WARNING, or ERROR) from IEEL? (A FAIL response happens when there are too many alerts, and a PASS happens if there are not too many alerts) |
| ieel_lnetstate   | If the node is a Lustre server node, then is the Luster network status OK? |
| interfaces       | Are the interfaces all up and OK? |
| ipmihealth       | Is the BMC (IPMI or iLO) health OK? Uses the script sample_ipmi. |
| ldap             | Can the ID of the user be looked up with LDAP? |
| lustre           | Is the Lustre filesystem running OK? |
| ManagedServicesOK| Are the managed services OK? |
| megaraid         | Are the MegaRAID controllers OK? |
|                  | The proprietary MegaCli software from LSI (http://www.lsi.com) is needed for this health check. |

...continued
<table>
<thead>
<tr>
<th>Name</th>
<th>Query (response is PASS/FAIL/UNKNOWN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mounts</td>
<td>Are all mounts defined in the fstab OK?</td>
</tr>
<tr>
<td>mysql</td>
<td>Is the status and configuration of MySQL correct?</td>
</tr>
</tbody>
</table>
| hardware-profile   | Is the specified node’s hardware configuration during health check use unchanged? Uses the script node-hardware-profile. 
The options to this script are described using the “-h” help option. Before this script is used for health checks, the specified hardware profile is usually first saved with the -s option. Eg: “node-hardware-profile -n node001 -s hardwarenode001” |
| ntp                | Is NTP synchronization happening? |
| oomkiller          | Has the oomkiller process run? Yes=FAIL. The oomkiller health check checks if the oomkiller process has run. The configuration file /cm/local/apps/cmd/scripts/healthchecks/configfiles/oomkiller.conf for the oomkiller health check can be configured to reset the response to PASS after one FAIL is logged, until the next oomkiller process runs. The processes killed by the oomkiller process are logged in /var/spool/cmd/save-oomkilleraction. 
A consideration of the causes and consequences of the killed processes is strongly recommended. A reset of the node is generally recommended. |
| portchecker        | Is the specified port on the specified host open for TCP (default) or UDP connections? |
| 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 (info messages), which the killprocess action (page 665) 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” |
Rogue process criteria can be configured in the file /cm/local/apps/cmd/scripts/healthchecks/configfiles/rogueprocess.py within the software image. To implement a changed criteria configuration, the software image used by systems on which the health check is run should be updated (section 5.6). For example, using: cmsh -c "device; imageupdate -c default -w" for the default category of nodes.

### schedulers

Are the queue instances of all schedulers on a node healthy?

### smart

Is the SMART response healthy? The severities can be configured in the file /cm/local/apps/cmd/scripts/healthchecks/configfiles/smart.conf. By default, if a drive does not support the SMART commands and results in a “Smart command failed” info message for that drive, then the healthcheck is configured to give a PASS response. This is because the mere fact that the drive is a non-SMART drive should not be a reason to conclude that the drive is unhealthy.

The info messages can be suppressed by setting a whitelist of the disks to be checked within /cm/local/apps/cmd/scripts/healthchecks/smart.

### ssh2node

Is passwordless ssh root login from head to node working?

Some details of its behavior are:

- If root ssh login to the head node has been disabled, then the health check fails on the head node.
- ssh2node is independent of the CMDaemon that runs on the regular node. That is, the health check may still respond with a PASS, even if the node is not UP according to CMDaemon.

### swraid

Are the software RAID arrays healthy?

### testhealthcheck

A health check script example for creating scripts, or setting a mix of PASS/FAIL/UNKNOWN responses. The source includes examples of environment variables that can be used, as well as configuration suggestions.

* built-ins, not standalone scripts.

If sampling from a head node, a standalone script is in directory:

/cm/local/apps/cmd/scripts/healthchecks/

If sampling from a regular node, a standalone script is in directory:

/cm/images/default-image/cm/local/apps/cmd/scripts/healthchecks/

© Bright Computing, Inc.
G.2.2 Parameters For Health Checks

Health checks have the parameters indicated by the left column in the example below:

Example

```bash
[myheadnode->monitoring->healthchecks]% show cmsh
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class of healthcheck</td>
<td>internal</td>
</tr>
<tr>
<td>Command</td>
<td>/cm/local/apps/cmd/scripts/healthchecks/cmsh</td>
</tr>
<tr>
<td>Description</td>
<td>Checks whether cmsh is available, i.e. can we +</td>
</tr>
<tr>
<td>Disabled</td>
<td>no</td>
</tr>
<tr>
<td>Extended environment</td>
<td>no</td>
</tr>
<tr>
<td>Name</td>
<td>cmsh</td>
</tr>
<tr>
<td>Only when idle</td>
<td>no</td>
</tr>
<tr>
<td>Parameter permissions</td>
<td>optional</td>
</tr>
<tr>
<td>Required resources</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td></td>
</tr>
<tr>
<td>Sampling method</td>
<td>samplingonnode</td>
</tr>
<tr>
<td>State flapping count</td>
<td>7</td>
</tr>
<tr>
<td>Timeout</td>
<td>10</td>
</tr>
</tbody>
</table>

The parameters have the same meaning as for metrics, with the following exceptions due to inapplicability:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Reason For Inapplicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>class: prototype</td>
<td>only applies to metric collections</td>
</tr>
<tr>
<td>cumulative</td>
<td>only sensible for numeric values</td>
</tr>
<tr>
<td>metricunit</td>
<td>only applies to numeric values</td>
</tr>
<tr>
<td>retrievalmethod</td>
<td>all health checks use CMDaemon internally for retrieval</td>
</tr>
<tr>
<td>maximum</td>
<td>only applies to numeric values</td>
</tr>
<tr>
<td>minimum</td>
<td>only applies to numeric values</td>
</tr>
</tbody>
</table>

The remaining parameters have meanings that can be looked up in section G.1.2.
G.3 Actions And Their Parameters

G.3.1 Actions

Table G.3.1: List Of Actions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drain node</td>
<td>Allows no new processes on a compute node from the workload manager. This</td>
</tr>
<tr>
<td></td>
<td>means that already running jobs are permitted to complete. Usage Tip: Plan</td>
</tr>
<tr>
<td></td>
<td>for undrain from another node becoming active</td>
</tr>
<tr>
<td>killprocess*</td>
<td>Kills processes listed in STDIN with KILL (-9) signal. Format:</td>
</tr>
<tr>
<td></td>
<td>killprocess &lt;PID1[,&lt;PID2&gt;...]&gt; This action is designed to work with rogueprocess (page 662)</td>
</tr>
<tr>
<td>Power off</td>
<td>Powers off, hard</td>
</tr>
<tr>
<td>Power on</td>
<td>Powers on, hard</td>
</tr>
<tr>
<td>Power reset</td>
<td>Power reset, hard</td>
</tr>
<tr>
<td>Reboot</td>
<td>Reboot via the system, trying to shut everything down cleanly, and then</td>
</tr>
<tr>
<td></td>
<td>start up again</td>
</tr>
<tr>
<td>SendEmail**</td>
<td>Sends mail using the mailserver that was set up during server configuration.</td>
</tr>
<tr>
<td></td>
<td>Format: sendemail [<a href="mailto:somebody@example.com">somebody@example.com</a>]. Default destination is root@localhost. The e-mail address that it is sent to is specified by a parameter in the monitoring configuration that calls this action.</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Power off via system, trying to shut everything down cleanly</td>
</tr>
<tr>
<td>Restart service**</td>
<td>Restart service. The service is specified by a parameter in the monitoring</td>
</tr>
<tr>
<td></td>
<td>configuration that calls this action.</td>
</tr>
<tr>
<td>Start service**</td>
<td>Start service. The service is specified by a parameter in the monitoring</td>
</tr>
<tr>
<td></td>
<td>configuration that calls this action.</td>
</tr>
<tr>
<td>Stop service**</td>
<td>Stop service. The service is specified by a parameter in the monitoring</td>
</tr>
<tr>
<td></td>
<td>configuration that calls this action.</td>
</tr>
<tr>
<td>test action*</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 that can be used as well as tips on configuring it generally</td>
</tr>
<tr>
<td>Undrain node</td>
<td>Allow processes to run on the node from the workload manager</td>
</tr>
</tbody>
</table>

* standalone scripts, not built-ins.

If running from a head node, the script is in directory: /cm/local/apps/cmd/scripts/actions/
If running from a regular node, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/actions/

** the text “is specified by a parameter in the monitoring configuration that calls this action” means:
for cmsh:
- for metrics: like in append example for thresholds on page 425
- for health checks: like in append example for health configuration on page 428
for cmgui:
- for metrics: as illustrated by figure 10.25
- for health checks: as illustrated by figure 10.29

G.3.2 Parameters For Actions

Actions have the parameters indicated by the left column in the example below:

Example

[myheadnode->monitoring->actions]$ show drainnode

© Bright Computing, Inc.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>&lt;built-in&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>Remove a node from further use by the scheduler+</td>
</tr>
<tr>
<td>Name</td>
<td>Drain node</td>
</tr>
<tr>
<td>Run on</td>
<td>headnode</td>
</tr>
<tr>
<td>Timeout</td>
<td>5</td>
</tr>
<tr>
<td>isCustom</td>
<td>no</td>
</tr>
</tbody>
</table>

The meanings of these parameters are:

**Command:** For a standalone metric script, it is the full path. For a built-in, the value cannot be set, and the command is simply the name of the metric.

**Description:** Description of the metric. Empty by default.

**Name:** The name given to the metric.

**Run on:** The node it will run on. For standalone actions it is usually a choice of head node, or the non-head node. For non-head nodes the action will run from the node that triggered it, if the node has sufficient permission to do that.

**Timeout:** After how many seconds the command will give up retrying. Default value is 5 seconds.

**isCustom:** Is this a standalone script?
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>/etc/slurm/slurm.conf</td>
<td>head node</td>
<td>Add/Remove/Update nodes, hostname change</td>
</tr>
<tr>
<td>/etc/slurm/slurmd.conf</td>
<td>head node</td>
<td>Add/Remove/Update nodes, hostname change</td>
</tr>
<tr>
<td>/etc/slurm/gres.conf</td>
<td>all nodes</td>
<td>Add/Remove/Update nodes</td>
</tr>
<tr>
<td>/etc/slurm/topology.conf</td>
<td>head node</td>
<td>Add/Remove/Update nodes</td>
</tr>
</tbody>
</table>

H.2 Grid Engine (SGE/UGE)

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ROOT/default/common/host_aliases</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$ROOT/default/common/act_qmaster</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
</tbody>
</table>

H.3 Torque

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ROOT/spool/server_name</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$ROOT/spool/torque.cfg</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$ROOT/server_priv/acl_svr/acl_hosts</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$ROOT/spool/server_priv/acl_svr/operators</td>
<td>head node</td>
<td>hostname change, failover</td>
</tr>
<tr>
<td>$ROOT/spool/server_priv/nodes</td>
<td>head node</td>
<td>Add/Remove/Update nodes</td>
</tr>
<tr>
<td>$ROOT/mom_priv/config</td>
<td>software image</td>
<td>hostname change, failover</td>
</tr>
</tbody>
</table>

H.4 PBS Pro

© Bright Computing, Inc.
Workload Manager Configuration Files Updated By CMDaemon

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/pbs.conf</td>
<td>head node, software image</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$ROOT/server_priv/acl_svr/operators</td>
<td>head node</td>
<td>hostname change, failover</td>
</tr>
<tr>
<td>$ROOT/spool/server_priv/nodes</td>
<td>head node</td>
<td>Add/Remove/Update nodes</td>
</tr>
<tr>
<td>$ROOT/server_priv/acl_hosts</td>
<td>head node</td>
<td>Add/Remove/Update nodes</td>
</tr>
</tbody>
</table>

H.5 LSF

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LSF_TOP/conf/lsf.conf</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$LSF_TOP/conf/hosts</td>
<td>cloud-director</td>
<td>add/remove/update cloud nodes</td>
</tr>
<tr>
<td>$LSF_TOP/conf/lsbatch/&lt;clusternamename&gt;/</td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td>configdir/lsb.queues</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$LSF_TOP/conf/lsbatch/&lt;clusternamename&gt;/&lt;</td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td>configdir/lsb.hosts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$LSF_TOP/conf/lsf.cluster.&lt;clusternamename&gt;</td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td>The default value of $LSF_TOP in Bright Cluster Manager is /cm/shared/apps/lsf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

H.6 OpenLava

<table>
<thead>
<tr>
<th>File/Property</th>
<th>Updates What?</th>
<th>Updated During</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LSF_ENVDIR/lsf.conf</td>
<td>head node</td>
<td>hostname/domain change, failover</td>
</tr>
<tr>
<td>$LSF_ENVDIR/hosts</td>
<td>cloud-director</td>
<td>add/remove/update cloud nodes</td>
</tr>
<tr>
<td>$LSF_ENVDIR/lsf.cluster.&lt;clusternamenamedir&gt;</td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td>$LSF_ENVDIR/lsb.queues</td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td>$LSF_ENVDIR/lsb.hosts</td>
<td>head node</td>
<td>add/remove/update nodes</td>
</tr>
<tr>
<td>The default value of $LSF_ENVDIR in Bright Cluster Manager is /cm/shared/apps/openlava/var/etc/</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Bright Computing, Inc.
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@bright73 ~]# module load openldap
[root@bright73 ~]# slappasswd
New password:
Re-enter new password:
SSHAJ/3wyO+1qyAwhh8Q4obl8489CWJHlpLg
```

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@bright73 ~]# grep ^rootpw /cm/local/apps/openldap/etc/slapd.conf
rootpw SSHAJ/3wyO+1qyAwhh8Q4obl8489CWJHlpLg
```

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@bright73 ~]# service ldap restart #Centos 6
[root@bright73 ~]# service slapd restart #Centos 7
```

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@bright73 ~]# grep LDAPPass /cm/local/apps/cmd/etc/cmd.conf
LDAPPass = "Mysecret1dappassw0rd"
```

CMDaemon is then restarted:

```
[root@bright73 ~]# service cmd restart
```
I.3 Checking LDAP Access

For a default configuration with user `cmsupport` and domain `cm.cluster`, the following checks can be run from the head node (some output truncated):

- **anonymous access:**

  ```
  [root@bright73 ~]# ldapsearch -x
  # extended LDIF
  #
  # LDAPv3
  # base <dc=cm,dc=cluster> (default) with scope subtree
  ...
  ```

- **root cn without a password (this should fail):**

  ```
  [root@bright73 ~]# 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@bright73 ~]#
  ```

- **root cn with a password (this should work):**

  ```
  [root@bright73 ~]# ldapsearch -x -D 'cn=root,dc=cm,dc=cluster' -w Mysecretldapassw0rd
  # 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 J: List Of Tokens

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service: CMAuth</strong></td>
<td></td>
</tr>
<tr>
<td>GET_PROFILE_TOKEN</td>
<td>Retrieve list of profiles and profile properties</td>
</tr>
<tr>
<td>ADD_PROFILE_TOKEN</td>
<td>Add a new profile</td>
</tr>
<tr>
<td>MODIFY_PROFILE_TOKEN</td>
<td>Modify existing user profile</td>
</tr>
<tr>
<td>GET_CMSERVICES_TOKEN</td>
<td>Get a list of available CMDaemon services</td>
</tr>
<tr>
<td><strong>Service: CMCeph</strong></td>
<td></td>
</tr>
<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>
<tr>
<td><strong>Service: CMCert</strong></td>
<td></td>
</tr>
<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_INITRD_MD5SUM_TOKEN | Retrieve MD5 sum of initial ramdisk |
| PUT_USERDATA_TOKEN | Set AWS user data in AWS |
| TERMINATE_NODE_TOKEN | Terminate cloud nodes |
| GET_AWS_KEY_TOKEN | Retrieve AWS key |
| 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 |
| SET_CLOUDDERRORS_TOKEN | | |
| 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>
<thead>
<tr>
<th><strong>Service and token name</strong></th>
<th><strong>User can...</strong></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**

<table>
<thead>
<tr>
<th><strong>SNMP_STRING_TOKEN</strong></th>
<th>Get ethernet switch SNMP public string</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHASSIS_USER_PASSWORD_TOKEN</td>
<td>Get/set chassis username and password</td>
</tr>
<tr>
<td>BMC_USERNAME_PASSWORD_TOKEN</td>
<td>View/set BMC (e.g. HP iLO4, IPMI) username and password</td>
</tr>
<tr>
<td>GET_DEVICE_TOKEN</td>
<td>View all device properties</td>
</tr>
<tr>
<td>GET_DEVICE_BY_PORT_TOKEN</td>
<td>View list of devices according to the ethernet switch port that they are connected to</td>
</tr>
<tr>
<td>ADD_DEVICE_TOKEN</td>
<td>Add a new device</td>
</tr>
<tr>
<td>UPDATE_DEVICE_TOKEN</td>
<td>Update device properties</td>
</tr>
<tr>
<td>GET_CATEGORY_TOKEN</td>
<td>Get list of categories</td>
</tr>
<tr>
<td>ADD_CATEGORY_TOKEN</td>
<td>Create new category</td>
</tr>
<tr>
<td>UPDATE_CATEGORY_TOKEN</td>
<td>Update a category property</td>
</tr>
<tr>
<td>GET_NODEGROUP_TOKEN</td>
<td>Get list of nodegroups</td>
</tr>
<tr>
<td>ADD_NODEGROUP_TOKEN</td>
<td>Add a new nodegroup</td>
</tr>
<tr>
<td>UPDATE_NODEGROUP_TOKEN</td>
<td>Update nodegroup properties (e.g. add a new member node)</td>
</tr>
<tr>
<td>GET_DEVICE_STATUS_TOKEN</td>
<td>Get device status (e.g. UP as well as status string e.g. restart required)</td>
</tr>
<tr>
<td>SET_DEVICE_STATUS_TOKEN</td>
<td>Set device status (only via RPC API calls)</td>
</tr>
<tr>
<td>POWER_ON_TOKEN</td>
<td>Power on a device using BMC or PDU power control</td>
</tr>
<tr>
<td>POWER_OFF_TOKEN</td>
<td>Power off a device</td>
</tr>
<tr>
<td>POWER_CYCLE_TOKEN</td>
<td>Power reset a device</td>
</tr>
<tr>
<td>POWER_STATUS_TOKEN</td>
<td>Get power status e.g on or off</td>
</tr>
<tr>
<td>POWER_RESULT_TOKEN</td>
<td>Get the result of the previous power command e.g. failed</td>
</tr>
<tr>
<td>SHUTDOWN_NODE_TOKEN</td>
<td>Shutdown a remote node managed by CMDaemon</td>
</tr>
<tr>
<td>REBOOT_NODE_TOKEN</td>
<td>Reboot a remote a node</td>
</tr>
<tr>
<td>FORCE_RECONNECT_TOKEN</td>
<td>Force remote client to reconnect (RPC API)</td>
</tr>
<tr>
<td>NODE_IDENTIFY_TOKEN</td>
<td>Identify a node (RPC API, used by node installer)</td>
</tr>
<tr>
<td>Service and token name</td>
<td>User can...</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<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 (typically set by CM-Daemon)</td>
</tr>
<tr>
<td>ADD_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN</td>
<td>Add a node-installer interaction (Used by CM-Daemon)</td>
</tr>
<tr>
<td>REMOVE_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN</td>
<td>Remove a node installer interaction</td>
</tr>
<tr>
<td>GET_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN</td>
<td>Get list of pending installer interactions</td>
</tr>
<tr>
<td>UPDATE_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN</td>
<td>Update installer interactions (e.g. confirm full provisioning)</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**

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_CLUSTER_OVERVIEW_TOKEN</td>
<td>Get cluster overview</td>
</tr>
<tr>
<td>GET_NODE_OVERVIEW_TOKEN</td>
<td>Get node overview</td>
</tr>
<tr>
<td>GET_NETSWITCH_OVERVIEW_TOKEN</td>
<td>Get switch overview</td>
</tr>
<tr>
<td>GET_PDU_OVERVIEW_TOKEN</td>
<td>Get PDU overview</td>
</tr>
<tr>
<td>GET_NODE_STATUS_TOKEN</td>
<td>Get node status</td>
</tr>
<tr>
<td>GET_HADOOP_HDFS_OVERVIEW_TOKEN</td>
<td>Get HDFS overview</td>
</tr>
<tr>
<td>GET_CEPH_OVERVIEW_TOKEN</td>
<td>Get Ceph overview</td>
</tr>
<tr>
<td>GET_OPENSTACK_OVERVIEW_TOKEN</td>
<td>Get OpenStack overview</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_OPENSTACK_TENANT_ OVERVIEW_TOKEN</td>
<td>Get OpenStack VM overview</td>
</tr>
</tbody>
</table>

#### Service: CMHadoop

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_HADOOP_HDFS_TOKEN</td>
<td>Get list of HDFS filesystems</td>
</tr>
<tr>
<td>ADD_HADOOP_HDFS_TOKEN</td>
<td>Add a new HDFS filesystem object</td>
</tr>
<tr>
<td>UPDATE_HADOOP_HDFS_TOKEN</td>
<td>Update HDFS object properties</td>
</tr>
<tr>
<td>HADOOP_BALANCER_TOKEN</td>
<td>Set balancer properties</td>
</tr>
<tr>
<td>HADOOP_BALANCER_STATUS_TOKEN</td>
<td>Get status of balancer</td>
</tr>
<tr>
<td>HADOOP_CLUSTER_SERVICES_TOKEN</td>
<td>Start/stop Hadoop services on nodes</td>
</tr>
<tr>
<td>HADOOP_FORMAT_HDFS_TOKEN</td>
<td>Format an HDFS filesystem</td>
</tr>
<tr>
<td>HADOOP_SET_DECOMMISSION_TOKEN</td>
<td>Decommission a Hadoop worker node</td>
</tr>
<tr>
<td>HADOOP_GET_DECOMMISSION_TOKEN</td>
<td>Get commissioning status of Hadoop nodes</td>
</tr>
</tbody>
</table>

#### Service: CMJob

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

#### Service: CMMain

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<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>Headnode 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>
</tbody>
</table>

...continues

© Bright Computing, Inc.
<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

**Service: CMMon**

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<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_THRESHACTION_TOKEN</td>
<td>Get threshold action settings</td>
</tr>
<tr>
<td>UPDATE_THRESHACTION_TOKEN</td>
<td>Update threshold action settings</td>
</tr>
<tr>
<td>ADD_THRESHACTION_TOKEN</td>
<td>Add threshold action settings</td>
</tr>
<tr>
<td>GET_MONITORING_DATA_TOKEN</td>
<td>Get monitoring data settings</td>
</tr>
<tr>
<td>SIGNAL_THRESHOLD_EXCEEDED_TOKEN</td>
<td>See if signal threshold exceeded</td>
</tr>
<tr>
<td>MONITORING_INTERNAL_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>ONDEMAND_RESULT_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**

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_NETWORK_TOKEN</td>
<td>Get network settings</td>
</tr>
<tr>
<td>ADD_NETWORK_TOKEN</td>
<td>Add network settings</td>
</tr>
<tr>
<td>UPDATE_NETWORK_TOKEN</td>
<td>Update network settings</td>
</tr>
</tbody>
</table>

**CMOpenStack**

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_OPENSTACK_TOKEN</td>
<td>Get OpenStack settings</td>
</tr>
<tr>
<td>ADD_OPENSTACK_TOKEN</td>
<td>Add OpenStack settings</td>
</tr>
</tbody>
</table>

...continues
<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE_OPENSTACK_TOKEN</td>
<td>Update OpenStack settings</td>
</tr>
<tr>
<td>OPENSTACK_USERNAMES_PASSWORDS_TOKEN</td>
<td>Get/set username and password settings</td>
</tr>
<tr>
<td>OPENSTACK_MANAGE_USER_INSTANCES_TOKEN</td>
<td>Get/set user instances</td>
</tr>
<tr>
<td>OPENSTACK_VIEW_CONSOLE_LOG_TOKEN</td>
<td>View OpenStack console log</td>
</tr>
<tr>
<td>OPENSTACK_TERMINATE_USER_INSTANCES_TOKEN</td>
<td>Terminate OpenStack user instances</td>
</tr>
<tr>
<td>OPENSTACK_EXECUTE_SETUP_TOKEN</td>
<td>Execute OpenStack setup</td>
</tr>
<tr>
<td>OPENSTACK_GET_SET_FINAL_EXECUTION_TOKEN</td>
<td>Not execute OpenStack setup</td>
</tr>
<tr>
<td>GET_OPENSTACK_VNC_URL_TOKEN</td>
<td>Get OpenStack VNC URL</td>
</tr>
</tbody>
</table>

**Service: CMPart**

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_PARTITION_TOKEN</td>
<td>Get partition settings</td>
</tr>
<tr>
<td>ADD_PARTITION_TOKEN</td>
<td>Add partition settings</td>
</tr>
<tr>
<td>UPDATE_PARTITION_TOKEN</td>
<td>Update partition settings</td>
</tr>
<tr>
<td>GET_RACK_TOKEN</td>
<td>Get rack settings</td>
</tr>
<tr>
<td>ADD_RACK_TOKEN</td>
<td>Add rack settings</td>
</tr>
<tr>
<td>UPDATE_RACK_TOKEN</td>
<td>Update rack settings</td>
</tr>
<tr>
<td>GET_SOFTWAREIMAGE_TOKEN</td>
<td>Get softwareimage settings</td>
</tr>
<tr>
<td>ADD_SOFTWAREIMAGE_TOKEN</td>
<td>Add softwareimage settings</td>
</tr>
<tr>
<td>UPDATE_SOFTWAREIMAGE_TOKEN</td>
<td>Update softwareimage settings</td>
</tr>
<tr>
<td>REMOVE_SOFTWAREIMAGE_TOKEN</td>
<td>Remove softwareimage settings</td>
</tr>
<tr>
<td>UPDATE_PROVISIONERS_TOKEN</td>
<td>Update provisioners settings</td>
</tr>
<tr>
<td>UPDATE_PROVISIONING_NODE_TOKEN</td>
<td>Update provisioning node</td>
</tr>
<tr>
<td>CMDAEMON_FAILOVER_STATUS_TOKEN</td>
<td>Obtain status of failover</td>
</tr>
</tbody>
</table>

**Service: CMProc**

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

...continues

© Bright Computing, Inc.
### Table J: List Of Tokens...continued

<table>
<thead>
<tr>
<th>Service and token name</th>
<th>User can...</th>
</tr>
</thead>
<tbody>
<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 headnode</td>
</tr>
<tr>
<td>NODE_EXEC_COMMAND_TOKEN</td>
<td>Remotely execute a command on a compute node</td>
</tr>
<tr>
<td>NODE_EXEC_COMMAND_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>
<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>
<tr>
<td><strong>Service: CMSession</strong></td>
<td></td>
</tr>
<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>
</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>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**

| GET_USER_TOKEN                          | Retrieve user information                    |
| ADD_USER_TOKEN                          | Add a new LDAP user                          |
| UPDATE_USER_TOKEN                       | Modify an existing LDAP user                 |
| GET_GROUP_TOKEN                         | Retrieve group information                   |
| ADD_GROUP_TOKEN                         | Add a new LDAP group                         |
| UPDATE_GROUP_TOKEN                      | Modify an existing LDAP group                |