

Bright Cluster Manager 9.0

Administrator Manual

Revision: a0ced4f

Date: Wed Apr 23 2025



©2020 Bright Computing, Inc. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of Bright Computing, Inc.

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of Novell, Inc. PGI is a registered trademark of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, PBS Pro, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

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

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

The 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

Table of Contents	i
0.1 Quickstart	xv
0.2 About This Manual	xv
0.3 About The Manuals In General	xv
0.4 Getting Administrator-Level Support	xvi
0.5 Getting Professional Services	xvi
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
2.1.6 Configuration Overlay	8
2.2 Modules Environment	9
2.2.1 Adding And Removing Modules	9
2.2.2 Using Local And Shared Modules	9
2.2.3 Setting Up A Default Environment For All Users	10
2.2.4 Creating A Modules Environment Module	11
2.2.5 Lua Modules Environment (LMod)	11
2.3 Authentication	12
2.3.1 Changing Administrative Passwords On The Cluster	12
2.3.2 Logins Using ssh	13
2.3.3 Certificates	14
2.3.4 Profiles	15
2.4 Bright View GUI	15
2.4.1 Installing The Cluster Management GUI Service	15
2.4.2 Navigating The Cluster With Bright View	17
2.5 Cluster Management Shell	20
2.5.1 Invoking cmsh	21
2.5.2 Levels, Modes, Help, And Commands Syntax In cmsh	23
2.5.3 Working With Objects	27

2.5.4	Accessing Cluster Settings	37
2.5.5	Advanced <code>cmsh</code> Features	38
2.6	Cluster Management Daemon	48
2.6.1	Controlling The Cluster Management Daemon	48
2.6.2	Configuring The Cluster Management Daemon	49
2.6.3	CMDaemon Versions	49
2.6.4	Configuring The Cluster Management Daemon Logging Facilities	50
2.6.5	Configuration File Modification, And The <code>FrozenFile</code> Directive	51
2.6.6	Configuration File Conflicts Between The Standard Distribution And Bright Cluster Manager For Generated And Non-Generated Files	52
2.6.7	CMDaemon Lite	52
3	Configuring The Cluster	55
3.1	Main Cluster Configuration Settings	56
3.1.1	Cluster Configuration: Various Name-Related Settings	56
3.1.2	Cluster Configuration: Some Network-Related Settings	57
3.1.3	Miscellaneous Settings	58
3.1.4	Limiting The Maximum Number Of Open Files	60
3.2	Network Settings	61
3.2.1	Configuring Networks	62
3.2.2	Adding Networks	66
3.2.3	Changing Network Parameters	66
3.2.4	Tools For Viewing Cluster Connections	77
3.3	Configuring Bridge Interfaces	80
3.4	Configuring VLAN interfaces	81
3.4.1	Configuring A VLAN Interface Using <code>cmsh</code>	81
3.4.2	Configuring A VLAN Interface Using Bright View	82
3.5	Configuring Bonded Interfaces	82
3.5.1	Adding A Bonded Interface	82
3.5.2	Single Bonded Interface On A Regular Node	83
3.5.3	Multiple Bonded Interface On A Regular Node	83
3.5.4	Bonded Interfaces On Head Nodes And HA Head Nodes	84
3.5.5	Tagged VLAN On Top Of a Bonded Interface	84
3.5.6	Further Notes On Bonding	84
3.6	Configuring InfiniBand And Omni-Path Interfaces	85
3.6.1	Installing Software Packages	85
3.6.2	Subnet Managers	85
3.6.3	InfiniBand Network Settings	86
3.6.4	Verifying Connectivity	87
3.7	Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces	88
3.7.1	BMC Network Settings	89
3.7.2	BMC Authentication	90
3.7.3	Interfaces Settings	91
3.7.4	Identification With A BMC	92
3.8	Configuring Switches And PDUs	93
3.8.1	Configuring With The Manufacturer's Configuration Interface	93
3.8.2	Configuring SNMP	93

3.8.3	Uplink Ports	95
3.8.4	The showport MAC Address to Port Matching Tool	95
3.8.5	Disabling Port Detection	96
3.8.6	The switchoverview Command	97
3.9	Disk Layouts: Disked, Semi-Diskless, And Diskless Node Configuration	97
3.9.1	Disk Layouts	98
3.9.2	Disk Layout Assertions	98
3.9.3	Changing Disk Layouts	98
3.9.4	Changing A Disk Layout From Disked To Diskless	98
3.10	Configuring NFS Volume Exports And Mounts	99
3.10.1	Exporting A Filesystem Using Bright View And cmsh	101
3.10.2	Mounting A Filesystem Using Bright View And cmsh	104
3.10.3	Mounting A Filesystem Subtree For A Diskless Node Over NFS	106
3.10.4	Mounting The Root Filesystem For A Diskless Node Over NFS	108
3.10.5	Configuring NFS Volume Exports And Mounts Over RDMA With OFED Drivers	110
3.11	Managing And Configuring Services	111
3.11.1	Why Use The Cluster Manager For Services?	111
3.11.2	Managing And Configuring Services—Examples	112
3.12	Managing And Configuring A Rack	115
3.12.1	Racks	115
3.12.2	Assigning Devices To A Rack	117
3.12.3	Assigning Devices To A Chassis	118
3.13	Configuring GPU Settings	121
3.13.1	GPUs And GPU Units	121
3.13.2	Configuring GPU Settings	121
3.14	Configuring Custom Scripts	127
3.14.1	custompowerscript	128
3.14.2	custompingscript	128
3.14.3	customremoteconsolescript	128
3.15	Cluster Configuration Without Execution By CMDaemon	129
3.15.1	Cluster Configuration: The Bigger Picture	129
3.15.2	Making Nodes Function Differently By Image	130
3.15.3	Making All Nodes Function Differently From Normal Cluster Behavior With FrozenFile	132
3.15.4	Adding Functionality To Nodes Via An initialize Or finalize Script	132
3.15.5	Examples Of Configuring Nodes With Or Without CMDaemon	133
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	137
4.1.3	Combining PDU- and IPMI-Based Power Control	138
4.1.4	Custom Power Control	138
4.1.5	Hewlett Packard iLO-Based Power Control	139
4.1.6	Dell drac-based Power Control	140
4.1.7	Redfish-Based and CIMC-Based Power Control	140
4.2	Power Operations	140

4.2.1	Power Operations Overview	140
4.2.2	Power Operations With Bright View	141
4.2.3	Power Operations Through cmsh	141
4.3	Monitoring Power	145
4.4	Switch Configuration To Survive Power Downs	145
5	Node Provisioning	147
5.1	Before The Kernel Loads	147
5.1.1	PXE Booting	147
5.1.2	iPXE Booting From A Disk Drive	151
5.1.3	iPXE Booting Using InfiniBand	151
5.1.4	Using PXE To Boot From The Drive	151
5.1.5	Network Booting Without PXE On The ARMv8 Architecture	152
5.1.6	Network Booting Protocol	152
5.1.7	The Boot Role	152
5.2	Provisioning Nodes	153
5.2.1	Provisioning Nodes: Configuration Settings	153
5.2.2	Provisioning Nodes: Role Setup With cmsh	154
5.2.3	Provisioning Nodes: Role Setup With Bright View	155
5.2.4	Provisioning Nodes: Housekeeping	157
5.3	The Kernel Image, Ramdisk And Kernel Modules	161
5.3.1	Booting To A “Good State” Software Image	161
5.3.2	Selecting Kernel Driver Modules To Load Onto Nodes	162
5.3.3	InfiniBand Provisioning	163
5.3.4	Omni-Path Provisioning	165
5.3.5	VLAN Provisioning	166
5.4	Node-Installer	166
5.4.1	Requesting A Node Certificate	167
5.4.2	Deciding Or Selecting Node Configuration	169
5.4.3	Starting Up All Network Interfaces	180
5.4.4	Determining Install-mode Type And Execution Mode	181
5.4.5	Running Initialize Scripts	187
5.4.6	Checking Partitions, RAID Configuration, Mounting Filesystems	187
5.4.7	Synchronizing The Local Drive With The Software Image	188
5.4.8	Writing Network Configuration Files	192
5.4.9	Creating A Local /etc/fstab File	192
5.4.10	Booting From The Local Hard Drive	192
5.4.11	Running Finalize Scripts	195
5.4.12	Unloading Specific Drivers	195
5.4.13	Switching To The Local init Process	196
5.5	Node States	196
5.5.1	Node States Icons In Bright View	196
5.5.2	Node States Shown In cmsh	196
5.5.3	Node States Indicating Regular Start Up	197
5.5.4	Node States That May Indicate Problems	198
5.6	Updating Running Nodes	200
5.6.1	Updating Running Nodes: Configuration With excludelistupdate	200

5.6.2	Updating Running Nodes: With <code>cmsh</code> Using <code>imageupdate</code>	207
5.6.3	Updating Running Nodes: With Bright View Using the <code>Update node</code> Option	207
5.6.4	Updating Running Nodes: Considerations	207
5.7	Adding New Nodes	208
5.7.1	Adding New Nodes With <code>cmsh</code> And Bright View Add Functions	208
5.7.2	Adding New Nodes With The Node Creation Wizard	208
5.8	Troubleshooting The Node Boot Process	209
5.8.1	Node Fails To PXE Boot	210
5.8.2	Node-installer Logging	213
5.8.3	Provisioning Logging	214
5.8.4	Ramdisk Fails During Loading Or Sometime Later	214
5.8.5	Ramdisk Cannot Start Network	214
5.8.6	Node-Installer Cannot Create Disk Layout	215
5.8.7	Node-Installer Cannot Start BMC (IPMI/iLO) Interface	218
6	User Management	223
6.1	Managing Users And Groups With Bright View	223
6.2	Managing Users And Groups With <code>cmsh</code>	225
6.2.1	Adding A User	225
6.2.2	Saving The Modified State	226
6.2.3	Editing Properties Of Users And Groups	227
6.2.4	Reverting To The Unmodified State	230
6.2.5	Removing A User	230
6.3	Using An External LDAP Server	231
6.3.1	External LDAP Server Replication	232
6.3.2	High Availability	235
6.4	Tokens And Profiles	235
6.4.1	Modifying Profiles	237
6.4.2	Creation Of Custom Certificates With Profiles, For Users Managed By Bright Cluster Manager's Internal LDAP	238
6.4.3	Creation Of Custom Certificates With Profiles, For Users Managed By An External LDAP	240
6.4.4	Logging The Actions Of <code>CMDDaemon</code> Users	241
7	Workload Management	243
7.1	Workload Managers Choices	243
7.2	Forcing Jobs To Run In A Workload Management System	244
7.2.1	Disallowing User Logins To Regular Nodes Via <code>cmsh</code>	244
7.2.2	Disallowing User Logins To Regular Nodes Via Bright View	245
7.2.3	Disallowing Other User Processes Outside Of Workload Manager User Processes	246
7.2.4	High Availability By Workload Managers	246
7.3	Installation Of Workload Managers	246
7.3.1	Running <code>cm-wlm-setup</code> In CLI Mode	246
7.3.2	Running <code>cm-wlm-setup</code> In <code>Ncurses</code> Mode	250
7.3.3	Prolog And Epilog Scripts	251
7.4	Enabling, Disabling, And Monitoring Workload Managers	253
7.4.1	Enabling And Disabling A WLM With Bright View	254

7.4.2	Enabling And Disabling A Workload Manager With <code>cmsh</code>	256
7.4.3	Monitoring The Workload Manager Services	260
7.5	Configuring And Running Individual Workload Managers	263
7.5.1	Configuring And Running Slurm	264
7.5.2	Installing, Configuring, And Running UGE	272
7.5.3	Configuring And Running PBS	278
7.5.4	Installing, Configuring, And Running LSF	284
7.6	Using Bright View With Workload Management	290
7.6.1	Jobs Display And Handling In Bright View	290
7.6.2	Queues Display And Handling In Bright View	291
7.7	Using <code>cmsh</code> With Workload Management	292
7.7.1	The <code>jobs</code> Submode In <code>cmsh</code>	294
7.7.2	Job Queue Display And Handling In <code>cmsh: jobqueue</code> Mode	298
7.7.3	Nodes Drainage Status And Handling In <code>cmsh</code>	300
7.8	Examples Of Workload Management Assignment	302
7.8.1	Setting Up A New Category And A New Queue For It	302
7.8.2	Setting Up A Prejob Check	306
7.9	Power Saving With <code>cm-scale</code>	307
7.10	Cgroups	307
7.10.1	Cgroups Settings For Workload Managers	307
7.11	Custom Node Parameters	312
8	Bright Cluster Manager Auto Scaler	315
8.1	Introduction	315
8.1.1	Use Cases	315
8.1.2	Restrictions	316
8.1.3	Setup	317
8.1.4	Comparison With <code>cm-scale-cluster</code>	317
8.2	Configuration	318
8.2.1	The ScaleServer Role	318
8.2.2	Resource Providers	319
8.2.3	Time Quanta Optimization	322
8.2.4	Fairsharing Priority Calculations And Node Management	324
8.2.5	Engines	324
8.2.6	Trackers	325
8.3	Examples Of <code>cm-scale</code> Use	330
8.3.1	Simple Static Node Provider Usage Example	330
8.3.2	Simple Dynamic Node Provider Usage Example	333
8.4	Further <code>cm-scale</code> Configuration And Examples	340
8.4.1	Dynamic Nodes Repurposing	340
8.4.2	Pending Reasons	340
8.4.3	Locations	342
8.4.4	Azure Storage Accounts Assignment	344
8.4.5	Mapping HPC Jobs To Particular Nodes	344
8.4.6	How To Exclude Unused Nodes From Being Stopped	346
8.4.7	Queue Node Placeholders	346

9	Containerization	349
9.1	Docker Engine	349
9.1.1	Docker Setup	350
9.1.2	Integration With Workload Managers	351
9.1.3	DockerHost Role	351
9.1.4	Iptables	354
9.1.5	Storage Backends	354
9.1.6	Docker Monitoring	358
9.1.7	Docker Setup For NVIDIA	359
9.2	Docker Registries	360
9.2.1	Docker And Harbor Registries: Introduction	360
9.2.2	Docker And Harbor Registries: Setup And Configuration	360
9.3	Kubernetes	363
9.3.1	Reference Architecture	363
9.3.2	Kubernetes Setup	364
9.3.3	Using GPUs With Kubernetes: NVIDIA GPUs	370
9.3.4	Using GPUs With Kubernetes: AMD GPUs	374
9.3.5	Kubernetes Configuration Overlays	378
9.3.6	Removing A Kubernetes Cluster	378
9.3.7	Kubernetes Cluster Configuration Options	379
9.3.8	EtcdCluster	382
9.3.9	Kubernetes Roles	382
9.3.10	Security Model	393
9.3.11	Addition Of New Kubernetes Users And Kubernetes Role Bindings Configuration	393
9.3.12	Enabling And Disabling Pod Security Policies for Kubernetes	397
9.3.13	Providing Access To External Users	397
9.3.14	Networking Model	399
9.3.15	Kubernetes Monitoring	400
9.3.16	Setup Of A Storage Class For Ceph	400
9.3.17	Integration With Harbor	402
9.4	Singularity	402
9.4.1	Use Cases	402
9.4.2	Package <code>cm-singularity</code>	403
9.4.3	MPI Integration	403
9.5	Kubernetes Apps	404
9.6	Spark On Kubernetes	406
9.6.1	Deploying Spark On Kubernetes	406
9.6.2	Providing Custom Docker Images	410
9.7	Kubernetes On Edge	410
9.7.1	Flags For Edge Installation	410
10	BeeGFS	413
10.1	BeeGFS Introduction	413
10.1.1	BeeGFS Concepts	413
10.1.2	BeeGFS Installation Notes And Options	413
10.2	Deployment And Uninstallation Of BeeGFS With <code>cm-beegfs-setup</code>	414
10.2.1	Deployment Of BeeGFS	414

10.2.2	Uninstalling BeeGFS	419
10.3	Managing The Deployed BeeGFS Instance	419
10.3.1	Setup	420
10.3.2	BeeGFS Objects	420
10.3.3	Usage	437
10.3.4	Admon Interface	437
11	Post-Installation Software Management	441
11.1	Bright Cluster Manager Packages And Their Naming Convention	441
11.1.1	The packages Command	443
11.2	Managing Packages On The Head Node	444
11.2.1	Managing RPM Or .deb Packages On The Head Node	444
11.2.2	Installation Of Packages On The Head Node That Are Not .deb And Not .rpm Packages	447
11.3	Kernel Management On A Head Node Or Image	447
11.3.1	Installing A Standard Distribution Kernel Into An Image Or On A Head Node	448
11.3.2	Excluding Kernels And Other Packages From Updates	448
11.3.3	Updating A Kernel In A Software Image	450
11.3.4	Setting Kernel Options For Software Images	451
11.3.5	Kernel Driver Modules	451
11.4	Managing A Package In A Software Image And Running It On Nodes	453
11.4.1	Installing From Head Into The Image: Changing The Root Directory Into Which The Packages Are Deployed	453
11.4.2	Installing From Head Into The Image: Updating The Node	455
11.4.3	Installing From Head Into The Image: Possible Issues When Using rpm --root, yum --installroot Or chroot	455
11.4.4	Managing A Package In The Node-Installer Image	456
11.5	Managing Non-RPM Software In A Software Image And Running It On Nodes	457
11.5.1	Managing The Software Directly On An Image	457
11.5.2	Managing The Software Directly On A Node, Then Syncing Node-To-Image	458
11.6	Creating A Custom Software Image	461
11.6.1	Creating A Base Distribution Archive From A Base Host	461
11.6.2	Creating The Software Image With cm-create-image	463
11.6.3	Configuring Local Repositories For Linux Distributions, And For The Bright Cluster Manager Package Repository, For A Software Image	468
11.6.4	Creating A Custom Image From The Local Repository	470
11.7	Creating Images For Other Distributions And Architectures (Multidistro And Multiarch)	470
11.7.1	The cm-image Tool	471
11.7.2	Multidistro Examples: Provisioning From CentOS 7 Head Node To Ubuntu 18.04 Regular Nodes	473
11.7.3	Multiarch Example: Creating An Image From A Centos 7 Head Node For ARMv8 Architecture Regular Nodes	474
12	Cluster Monitoring	477
12.1	A Basic Monitoring Example And Action	477
12.1.1	Synopsis Of Basic Monitoring Example	477
12.1.2	Before Using The Basic Monitoring Example—Setting Up The Pieces	478

12.1.3	Using The Monitoring Basic Example	479
12.2	Monitoring Concepts And Definitions	482
12.2.1	Measurables	482
12.2.2	Enummetrics	485
12.2.3	Metrics	485
12.2.4	Health Check	487
12.2.5	Trigger	488
12.2.6	Action	488
12.2.7	Severity	489
12.2.8	AlertLevel	490
12.2.9	Flapping	490
12.2.10	Data Producer	490
12.2.11	Conceptual Overview: The Main Monitoring Interfaces Of Bright View	493
12.3	Monitoring Visualization With Bright View	494
12.3.1	The Monitoring Window	495
12.4	Monitoring Configuration With Bright View	496
12.4.1	Monitoring Configuration: Data Producers	497
12.4.2	Monitoring Configuration: Measurables	499
12.4.3	Monitoring Configuration: Consolidators	501
12.4.4	Monitoring Configuration: Actions	504
12.4.5	Monitoring Configuration: Triggers	506
12.4.6	Monitoring Configuration: Health status	509
12.4.7	Monitoring Configuration: All Health Checks	510
12.4.8	Monitoring Configuration: Standalone Monitored Entities	511
12.5	The monitoring Mode Of cmsd	511
12.5.1	The action Submode	512
12.5.2	The consolidator Submode	514
12.5.3	The measurable Submode	516
12.5.4	The setup Submode	520
12.5.5	The standalone Submode	526
12.5.6	The trigger Submode	526
12.6	Obtaining Monitoring Data Values	529
12.6.1	Getting The List Of Measurables For An Entity: The measurables, metrics, healthchecks And enummetrics Commands	529
12.6.2	On-Demand Metric Sampling And Health Checks	530
12.6.3	The Latest Data And Counter Values—The latest*data And latestmetriccounters Commands	533
12.6.4	Data Values Over A Period—The dumpmonitoringdata Command	535
12.6.5	Monitoring Data Health Overview—The healthoverview Command	541
12.6.6	Monitoring Data About The Monitoring System—The monitoringinfo Command	542
12.6.7	Dropping Monitoring Data With The monitoringdrop Command	543
12.6.8	Monitoring Suspension And Resumption—The monitoringsuspend And monitoringresume Commands	544
12.6.9	Monitoring Pickup Intervals	545
12.7	The User Portal	547
12.7.1	Accessing The User Portal	547

12.7.2	Setting A Common Username/Password For The User Portal	548
12.7.3	User Portal Access	549
12.7.4	User Portal Home Page	549
12.8	Job Monitoring	550
12.8.1	Job Metrics Introduction	550
12.8.2	Job Information Retention	551
12.8.3	Job Metrics Sampling Configuration	551
12.8.4	Job Monitoring In cmsh	554
12.9	Job Accounting	557
12.9.1	Labeled Entities	557
12.9.2	PromQL Queries	559
12.9.3	Parameterized PromQL Queries	567
12.9.4	Job Accounting In Bright View	568
12.9.5	PromQL Query Modes And Specification In Bright View	571
12.9.6	Access Control For Workload Accounting And Reporting	573
12.9.7	Drilldown Queries For Workload Accounting And Reporting	575
12.10	Cloud Job Tagging	576
12.11	Event Viewer	577
12.11.1	Viewing Events In Bright View	577
12.11.2	Viewing Events In cmsh	577
12.11.3	Using The Event Bucket From The Shell For Events And For Tagging Device States	579
12.11.4	InfoMessages	580
13	Day-to-day Administration	583
13.1	Parallel Shells: pdsh And pexec	583
13.1.1	pdsh In The OS Shell	583
13.1.2	pexec In cmsh	587
13.1.3	pexec In Bright View	587
13.1.4	Using The -j --join Option Of pexec In cmsh	588
13.1.5	Other Parallel Commands	588
13.2	Getting Support With Cluster Manager Issues, And Notifications For Release Updates	589
13.2.1	Support Via E-mail	589
13.2.2	Reporting Cluster Manager Diagnostics With cm-diagnose	590
13.2.3	Requesting Remote Support With request-remote-assistance	591
13.2.4	Requesting Remote Support With A Shared Screen Utility	592
13.2.5	Getting Notified About Updates	593
13.3	Backups	593
13.3.1	Cluster Installation Backup	593
13.3.2	Local Database And Data Backups And Restoration	594
13.4	Revision Control For Images	596
13.4.1	Btrfs: The Concept And Why It Works Well In Revision Control For Images	596
13.4.2	Btrfs Availability And Distribution Support	597
13.4.3	Installing Btrfs To Work With Revision Control Of Images In Bright Cluster Manager	597
13.4.4	Using cmsh For Revision Control Of Images	599
13.5	BIOS Configuration And Updates	601
13.5.1	Updating BIOS	601
13.5.2	Booting DOS Image	602

13.6	Hardware Match Check With The <code>hardware-profile</code> Data Producer	602
13.7	Serial Over LAN Console Access	603
13.7.1	Background Notes On Serial Console And SOL	604
13.7.2	SOL Console Configuration With Bright View	606
13.7.3	SOL Console Configuration And Access With <code>cmsh</code>	606
13.7.4	The <code>conman</code> Serial Console Logger And Viewer	607
13.8	Managing Raw Monitoring Data	611
13.8.1	Monitoring Subsystem Disk Usage With The <code>monitoringinfo --storage</code> Option	611
13.8.2	Estimating The Required Size Of The Storage Device	611
13.8.3	Moving Monitoring Data Elsewhere	612
13.8.4	Reducing Monitoring Data By Reducing Samples	612
13.8.5	Deleting All Monitoring Data	612
13.9	Node Replacement	614
14	High Availability	615
14.0	Introduction	615
14.0.1	Why Have High Availability?	615
14.0.2	High Availability Is Possible On Head Nodes, And Also On Regular Nodes	615
14.0.3	High Availability Usually Uses Shared Storage	615
14.0.4	Organization Of This Chapter	615
14.1	HA Concepts	616
14.1.1	Primary, Secondary, Active, Passive	616
14.1.2	Monitoring The Active Head Node, Initiating Failover	616
14.1.3	Services In Bright Cluster Manager HA Setups	616
14.1.4	Failover Network Topology	617
14.1.5	Shared Storage	619
14.1.6	Guaranteeing One Active Head At All Times	620
14.1.7	Automatic Vs Manual Failover	621
14.1.8	HA And Cloud Nodes	622
14.1.9	HA Using Virtual Head Nodes	622
14.2	HA Setup Procedure Using <code>cmha-setup</code>	622
14.2.1	Preparation	623
14.2.2	Failover Cloning (Replacing A Passive Head)	626
14.2.3	Shared Storage Setup	628
14.2.4	Automated Failover And Relevant Testing	630
14.3	Running <code>cmha-setup</code> Without Ncurses, Using An XML Specification	631
14.3.1	Why Run It Without Ncurses?	631
14.3.2	The Syntax Of <code>cmha-setup</code> Without Ncurses	631
14.3.3	Example <code>cmha-setup</code> Run Without Ncurses	632
14.4	Managing HA	633
14.4.1	Changing An Existing Failover Configuration	633
14.4.2	<code>cmha</code> Utility	633
14.4.3	States	636
14.4.4	Failover Action Decisions	637
14.4.5	Keeping Head Nodes In Sync	638
14.4.6	High Availability Parameters	640
14.4.7	Viewing Failover Via Bright View	642

14.4.8	Re-cloning A Head Node	642
14.5	HA For Regular Nodes	643
14.5.1	Why Have HA On Regular Nodes?	643
14.5.2	Comparing Head And Regular Node HA	643
14.5.3	Setting Up A Regular Node HA Service	644
14.5.4	The Sequence Of Events When Making Another HA Regular Node Active	648
14.6	HA And Workload Manager Jobs	648
15	The Jupyter Notebook Environment Integration	651
15.1	Introduction	651
15.2	Jupyter Environment Installation	652
15.2.1	Jupyter Setup	653
15.2.2	Jupyter Architecture	653
15.2.3	Verifying Jupyter Installation	655
15.2.4	Login Configuration	656
15.2.5	JupyterHub Screen After Login	657
15.3	Jupyter Notebook Examples	658
15.4	Jupyter Kernels	659
15.4.1	Jupyter Enterprise Gateway Kernels	660
15.5	Jupyter Kernel Creator Extension	661
15.5.1	Running Jupyter Kernels With Kubernetes	666
15.5.2	Running Jupyter Kernels With Workload Managers	667
15.6	Jupyter Kernel Creator Extension Customization	667
15.6.1	Kernel Template Parameters Definition	667
15.6.2	Kernel Template Parameters Usage	670
15.6.3	Kernel Template Creation Example	671
15.6.4	Extending Kubernetes Kernel Templates	672
15.6.5	Extending Workload Managers Kernel Templates	674
15.7	Jupyter VNC Extension	676
15.8	Jupyter WLM Magic Extension	678
15.9	Jupyter Environment Removal	681
16	Dell BIOS Management	683
16.1	Introduction	683
16.2	Prerequisites For BIOS Management	683
16.3	BIOS settings	684
16.3.1	Initializing The BIOS Settings Via <code>cmsh</code>	684
16.3.2	Managing The BIOS Settings Via Bright View	684
16.3.3	Applying Dell Settings And Firmware Updates Via Bright View	687
16.3.4	Managing The BIOS Settings Via <code>cmsh</code>	687
16.4	Frequently Asked Questions	690
A	Generated Files	693
A.1	System Configuration Files Created Or Modified By <code>CMDDaemon</code> On Head Nodes	693
A.2	System Configuration Files Created Or Modified By <code>CMDDaemon</code> In Software Images:	695
A.3	Files Created On Regular Nodes By The Node-Installer	695
A.4	Files Not Generated, But Installed In RHEL	696

B Bright Computing Public Key	701
C CMDaemon Configuration File Directives	703
D Disk Partitioning And RAID Configuration	731
D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema File	731
D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File	738
D.3 Example: Default Node Partitioning	741
D.4 Example: Hardware RAID Configuration	742
D.4.1 RAID level 0 And RAID 10 Example	742
D.5 Example: Software RAID	744
D.6 Example: Software RAID With Swap	745
D.7 Example: Logical Volume Manager	746
D.8 Example: Logical Volume Manager With RAID 1	747
D.9 Example: Diskless	749
D.10 Example: Semi-diskless	749
D.11 Example: Preventing Accidental Data Loss	750
D.12 Example: Using Custom Assertions	751
E Example initialize And finalize Scripts	753
E.1 When Are They Used?	753
E.2 Accessing From Bright View And cmsh	753
E.3 Environment Variables Available To initialize And finalize Scripts	754
E.4 Using Environment Variables Stored In Multiple Variables	757
E.5 Storing A Configuration To A Filesystem	758
E.5.1 Storing With Initialize Scripts	758
E.5.2 Ways Of Writing A Finalize Script To Configure The Destination Nodes	758
E.5.3 Restricting The Script To Nodes Or Node Categories	761
F Workload Managers Quick Reference	763
F.1 Slurm	763
F.2 Univa Grid Engine	765
F.3 PBS Pro	766
G Metrics, Health Checks, Enummetrics, And Actions	767
G.1 Metrics And Their Parameters	767
G.1.1 Regular Metrics	767
G.1.2 NFS Metrics	772
G.1.3 Monitoring System Metrics	773
G.1.4 GPU Metrics	774
G.1.5 Job Metrics	775
G.1.6 Prometheus Metrics	780
G.1.7 Parameters For Metrics	783
G.2 Health Checks And Their Parameters	785
G.2.1 Health Checks	785
G.2.2 Parameters For Health Checks	791
G.3 Enummetrics	792
G.4 Actions And Their Parameters	793

G.4.1	Actions	793
G.4.2	Parameters For A Monitoring Action	793
H	Workload Manager Configuration Files Updated By CMDaemon	795
H.1	Slurm	795
H.2	Univa Grid Engine (UGE)	795
H.3	PBS Pro	795
I	Changing The LDAP Password	797
I.1	Setting A New Password For The LDAP Server	797
I.2	Setting The New Password In <code>cmd.conf</code>	797
I.3	Checking LDAP Access	798
J	Tokens	799
K	Understanding Consolidation	807
K.1	Introduction	807
K.2	What Is Consolidation?	807
K.3	Raw Data And Consolidation	807
K.4	A Demonstration Of The Output	808
L	Node Execution Filters And Execution Multiplexers	811
L.1	Data Producers: Default Configuration For Running And Sampling	812
L.1.1	Nodes That Data Producers Are Running On By Default—The <code>nodes</code> Command	812
L.1.2	Nodes That Data Producers Target By Default—The <code>samplenow</code> Command	812
L.2	Data Producers: Configuration For Running And Targeting	813
L.2.1	Custom Metrics From The <code>fm.sh</code> Custom Script	813
L.3	Replacing A Resource With An Explicit Node Specification	815
L.4	Excessive Sampling	816
L.5	Not Just For Nodes	817
L.6	Lua Node Execution Filters	817
M	A Tree View Of <code>cmsh</code>	821
M.1	Modes	821

Preface

Welcome to the *Administrator Manual* for the Bright Cluster Manager 9.0 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 9.0 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at <http://support.brightcomputing.com/manuals>.

- The *Upgrade Manual* describes upgrading from earlier versions of Bright Cluster Manager.
- The *Installation Manual* describes installation procedures.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to program with Bright Cluster Manager.
- The *OpenStack Deployment Manual* describes how to deploy OpenStack 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.

There is also a feedback form available via Bright View, via the Account icon, , following the clickpath:

Account→Help→Feedback

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 13.2 has more details on working with support.

0.5 Getting Professional Services

Bright Computing normally differentiates between professional services (customer asks Bright Computing to do something or asks Bright Computing to provide some service) and support (customer has a specific question or problem that requires an answer or resolution). Professional services can be provided after consulting with the reseller, or the Bright account manager.

1

Introduction

1.1 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 (network) boot off it and provision themselves from it with a default image, without requiring a Linux distribution DVD themselves. The network boot and provisioning process for the regular nodes is described in detail in Chapter 5.

In brief, provisioning installs an operating system and files on a node. This kind of installation to a regular node differs from a normal Linux installation in several ways. An important difference is that content that is put on the filesystem of the regular node is normally overwritten by provisioning when the regular node reboots.
- **Post-installation software installation:** The installation of software to a cluster that is already configured and running Bright Cluster Manager is described in detail in Chapter 11 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, OpenStack, Edge, 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 (Bright View, 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:

Chapter, Title	Description
2 Cluster Management With Bright Cluster Manager	Introduction to main concepts and tools of Bright Cluster Manager. Lays down groundwork for the remaining chapters
3 Configuring The Cluster	Further configuration and set up of the cluster after software installation of Bright Cluster Manager on the head node.
4 Power Management	How power management within the cluster works
5 Node Provisioning	Node provisioning in detail
6 User Management	Account management for users and groups
7 Workload Management	Workload management implementation and use
8 The <code>cm-scale</code> Service	A Bright Cluster Manager service to dynamically scale the cluster according to need
9 Containerization	Using Docker and Kubernetes with Bright Cluster Manager
11 Post-Installation Software Management	Managing, updating, modifying Bright Cluster Manager software and images
12 Cluster Monitoring	Cluster monitoring, conditional triggering of actions, user portal and job monitoring
13 Day-To-Day Administration	Miscellaneous administration
14 High Availability	Background details and setup instructions to build a cluster with redundant head nodes
15 The Jupyter Notebook Environment Integration	Installing and using the Jupyter notebook environment
16 Dell BIOS Management	BIOS management with Bright Cluster Manager for certain Dell hardware

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*)
- Edge deployment (*Edge Manual*)
- Developer topics (*Developer Manual*)

1.3 Outside The Direct Scope Of The Administrator Manual

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

- **Use by the end user:** This is covered very peripherally in this manual. The user normally interacts with the cluster by logging into a custom Linux user environment to run jobs. Details on running jobs from the perspective of the user are given in the *User Manual*.
- **The knowledge base** at <http://kb.brightcomputing.com> often supplements the *Administrator Manual* with discussion of the following:
 - Obscure, or complicated, configuration cases
 - Procedures that are not really within the scope of 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 13.2 describes how to get further support.

2

Cluster Management With Bright Cluster Manager

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 (Bright View) and cluster management shell (`cmsh`) respectively. These are the primary applications that interact with the cluster management daemon.

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

2.1 Concepts

In this section some concepts central to cluster management with 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
- GPU Unit
- Chassis
- Ethernet Switch
- InfiniBand Switch

- Lite Node
- Myrinet Switch
- Power Distribution Unit
- Rack Sensor Kit
- Generic Device

A device can have a number of properties (e.g. rack position, hostname, switch port) which can be set in order to configure the device. Using 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:

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

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

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

RHEL7/CentOS7/SL7 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 11.

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 Bright View (section 2.4) or `cmsh` (section 2.5) front end tools, and the configurations are managed internally with a database.

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

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

A regular node is in exactly one category at all times, which is default by default. The default category can be changed by accessing the base object of partition mode (page 70), and setting the value of `defaultcategory` to another, existing, category.

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

A node inherits values from the category it is in. Each value is treated as the default property value for a node, and can be overruled by specifying the node property value for a particular node.

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

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

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

Example

Node Category	Description
<code>nodes-ib</code>	nodes with InfiniBand capabilities
<code>nodes-highmem</code>	nodes with extra memory
<code>login</code>	login nodes
<code>storage</code>	storage nodes

2.1.4 Node Groups

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

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

Example

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

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

2.1.5 Roles

A *role* is a task that can be performed by a node. By assigning a certain role to a node, an administrator activates the functionality that the role represents on this node. For example, a node can be turned into provisioning node, or can be turned into a storage node, by assigning the corresponding roles to the node.

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

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

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

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

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

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

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

2.1.6 Configuration Overlay

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

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

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

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

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

priority	assigned to node from
-1	<i>configuration overlay not assigned</i>
250	category
500	configuration overlay with default priority
750	node

2.2 Modules Environment

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

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

2.2.1 Adding And Removing Modules

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

Modules currently installed are listed with:

```
module list
```

The modules available for loading are listed with:

```
module avail
```

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

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

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

Example

```
module add shared
module add gcc/9.2.0
module add openmpi/gcc/64/1.10.7
module add openblas
module add openblas/dynamic/0.2.20
mpicc -o myapp myapp.c
```

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

2.2.2 Using Local And Shared Modules

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

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

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

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

```
module initadd shared
```

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

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

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

2.2.3 Setting Up A Default Environment For All Users

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

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

Example

```
[fred@bright90 ~]$ module list
Currently Loaded Modulefiles:
1) shared
```

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

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

```
[root@bright90 ~]# cat /etc/profile.d/userdefaultmodules.sh
module load shared
module load slurm
module load gdb
```

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

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

```
[root@bright90 ~]# cat /cm/shared/modulefiles/default-environment
#%Module1.0#####
## default modulefile
##
proc ModulesHelp { } {
```

```

        puts stderr "\tLoads default environment modules for this cluster"
    }
module-whatis "adds default environment modules"

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

```

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

```

[root@bright90 ~]# cat /etc/profile.d/userdefaultmodules.sh
module load -s default-environment

```

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

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

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

2.2.4 Creating A Modules Environment Module

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

Cluster administrators can use the existing modules files as a guide to creating and installing their own modules for module environments, and can copy and modify a file for their own software if there is no environment provided for it already by 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*. Lua modules, or *LMod*, provide an alternative modules environment, where the files are typically written in Lua. LMod can be used as a replacement for TMod.

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

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

On a 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 for a node can be done by setting an environment variable, `$ENABLE_LMOD` in the `cm-lmod-init.sh` shell script.

Switching For The Head Node

For example, for the head node:

Example

```

[root@bright90 ~]# 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@bright90 ~]# 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 in the file on the node is effective after having logged out, then logged into the shell again.

Switching For The Regular Nodes

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

2.3 Authentication

2.3.1 Changing Administrative Passwords On The Cluster

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

Amongst the administrative passwords associated with the cluster are:

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

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

```
[root@bright90 ~]# cm-change-passwd
With this utility you can easily change the following passwords:
* root password of head node
* root password of slave images
* root password of node-installer
* root password of mysql
```

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

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

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

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

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

For a high-availability—also called a failover—configuration, the passwords are copied over automatically to the other head node when a change is made in the software image root password (case 2 on page 12).

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

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

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

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

2.3.2 Logins Using `ssh`

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

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

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

Users can be restricted from `ssh` logins

- on regular nodes using the `usernodelogin` (section 7.2.1) or `User node login` (section 7.2.2) settings

- on the head node by modifying the `sshd` configuration on the head node. For example, to allow only root logins, the value of `AllowUsers` can be set in `/etc/ssh/sshd_config` to `root`. The man page for `sshd_config` has details on this.

2.3.3 Certificates

PEM Certificates And CMDaemon Front-end Authentication

While nodes in 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 240.

Using Bright View and authenticating to the Bright Cluster Manager: When an administrator uses the Bright View front end, a login to the cluster is carried out with username password authentication (figure 2.1), unless the authentication has already been stored in the browser, or unless certificate-based authentication is used.

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

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

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

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

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

Certificate generation in general, including the generation and use of non-administrator certificates, is described in greater detail section 6.4.

Replacing A Temporary Or Evaluation License

In the preceding section, if a license is replaced, then regular user certificates need to be generated again. Similarly, if a temporary or evaluation license is replaced, regular user certificates need to be generated again. This is because the old user certificates are signed by a key that is no longer valid. The generation of non-administrator certificates and how they function is described in section 6.4.

2.3.4 Profiles

Certificates that authenticate to CMDaemon contain a *profile*.

A profile determines which cluster management operations the certificate holder may perform. The administrator certificate is created with the `admin` profile, which is a built-in profile that allows all cluster management operations to be performed. In this sense it is similar to the `root` account on unix systems. Other certificates may be created with different profiles giving certificate owners access to a pre-defined subset of the cluster management functionality (section 6.4).

2.4 Bright View GUI

This section introduces the basics of the cluster management GUI (Bright View). Bright View is the web application front end to cluster management in Bright Cluster Manager. It has been tested to run without problems on the following desktop browsers:

Browser	Version
Chrome	77
Chrome	78
Edge	17
Edge	18
Firefox	69
Firefox	70
Safari	12.1
Safari	13

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

Bright View should run on other recent browsers without issues too, but this is not supported. Mobile browsers are also not supported.

2.4.1 Installing The Cluster Management GUI Service

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

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

The Bright Cluster Manager package that provides the service is `bright-view`, and is installed by default with Bright Cluster Manager. The service can be disabled by removing the package.

Bright Cluster Manager Bright View Login Window

Figure 2.1 shows the login dialog window for Bright View

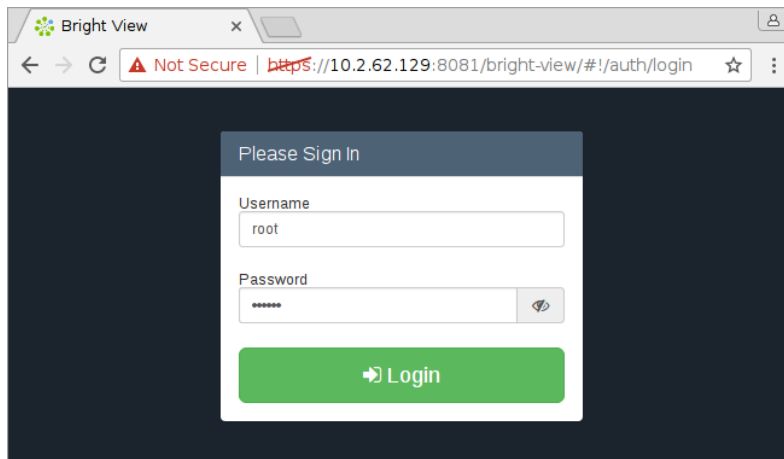


Figure 2.1: Bright View Login via `https://<host name or IP address>:8081/bright-view`

Bright Cluster Manager Bright View Default Display On Connection

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

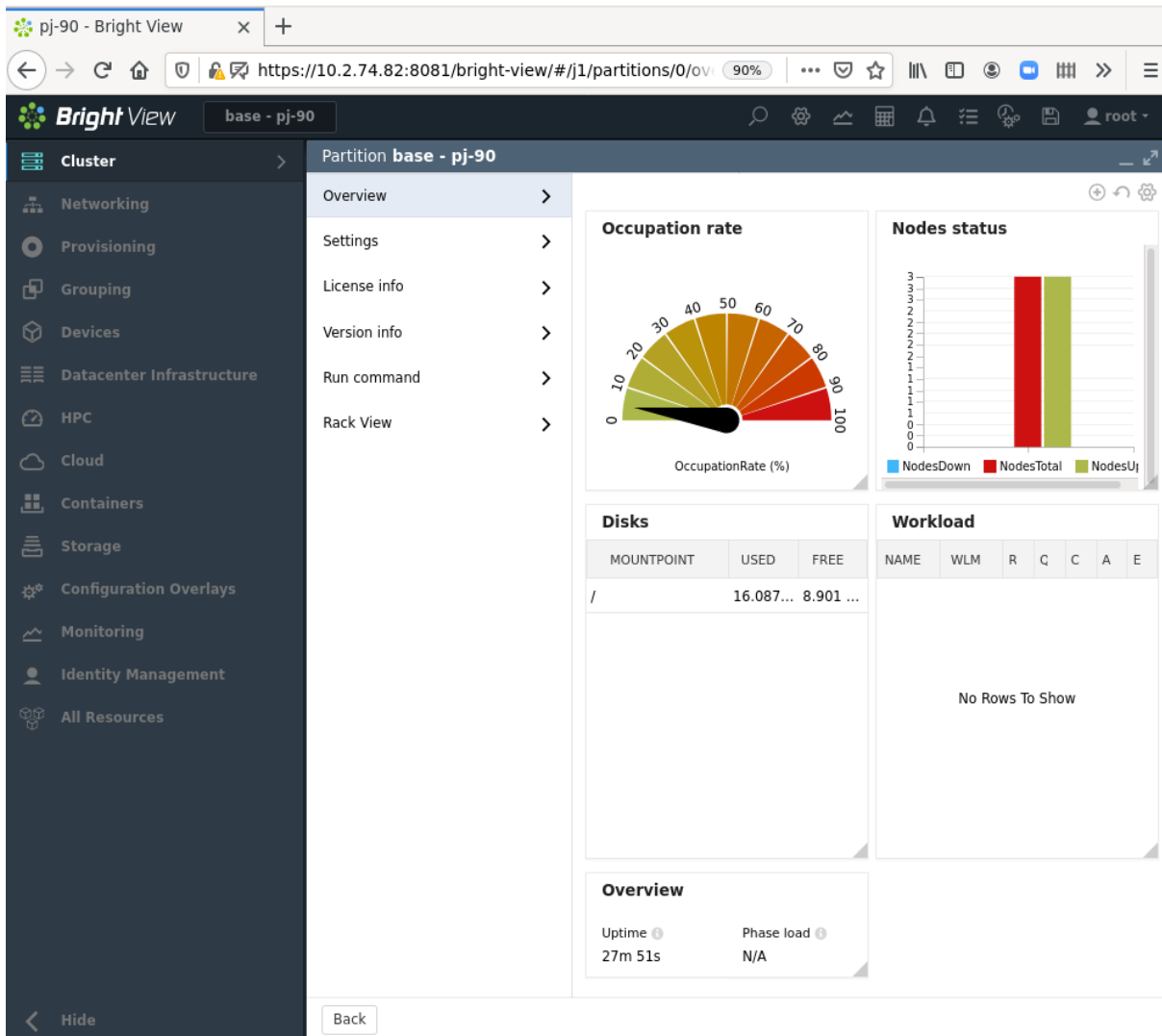


Figure 2.2: Cluster Overview

2.4.2 Navigating The Cluster With Bright View

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

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

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

As an example, the Cluster resource can be selected. This opens up the so-called Partition base window, which is essentially a representation of the cluster instance.¹

The options within the Partition base window are mapped out in figure 2.3 and summarily described next.

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

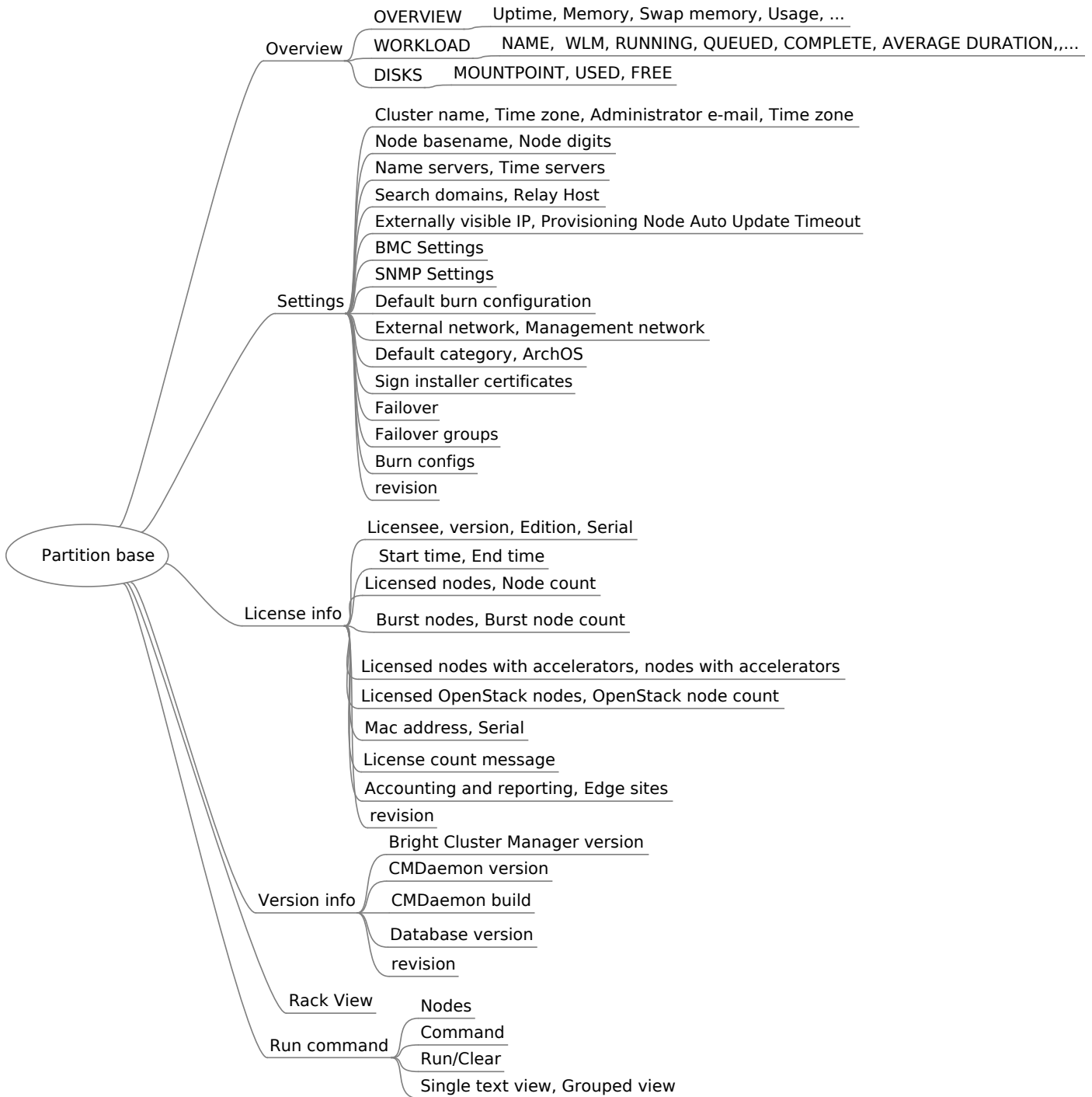


Figure 2.3: Cluster Overview Navigation

Overview

The Overview window has three scrollable overview panes that show current data for the cluster as a whole:

- The top pane, OVERVIEW, shows some system resource information, such as uptime, memory, swap memory.

The Usage display is typically an item of great interest to a cluster administrator, and displays CPU usage. Its indicator bar is divided into 4 sections. Hovering the mouse cursor over the bar

(figure 2.4) shows the usage segregation:

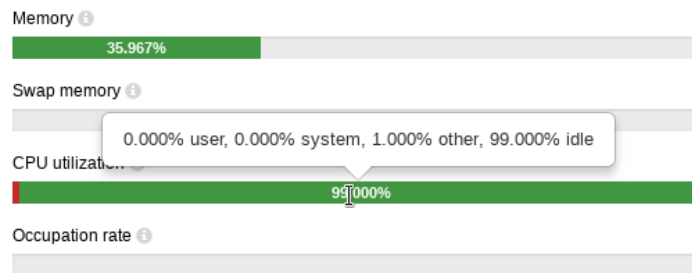


Figure 2.4: Hovering The Mouse Cursor Over The CPU Utilization Bar

The segregation is according to the scheme described in the following table:

Bright state	top state	CPU time spent by applications in:
user	us	user space
system	sy	kernel space
other	sum of:	sum of these other active states:
	wa	i/o wait
	ni	niced user processes
	hi	hardware IRQ servicing/handling
	si	software IRQ servicing/handling
	st	“stolen time”—forced wait for a virtual CPU while hypervisor services another processor
idle	id	idling

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

- The middle pane, *WORKLOAD*, shows workloads that the cluster is managing.
- The bottom pane, *DISKS*, shows disk mount points and their use.

Settings

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

- Cluster name, Administrator e-mail, Time zone
- Node basename, Node digits
- Name servers, Time servers
- Search domains, Relay Host
- Externally visible IP, Provisioning Node Auto Update Timeout

- `BMC Settings`: Opens up a window to manage BMC settings
- `SNMP Settings`: Opens up a window to manage SNMP settings
- `Default burn configuration`
- `External network, Management network`
- `Default category, ArchOS`: Sets the default category and can open up a window for ArchOS settings.
- `Sign installer certificates`
- `Failover`: Opens up a window to manage failover properties.
- `Failover groups`: Opens up a window to manage failover groups properties.
- `Burn configs`: Opens up a window to manage burn configuration settings.
- `revision`

License info

The `License info` window shows information to do with cluster licensing. A slightly obscure property within this window is `version`, which refers to the version type of the license. The license for Bright Cluster Manager version 7.0 and above is of a type that is compatible with versions all the way up to 9.0. Bright Cluster Manager license versions from before 7.0 are not compatible. In practice it means that an upgrade from before 7.0, to 7.0 or beyond, requires a license upgrade. Bright Computing must be contacted to arrange the license upgrade.

Version info

The `Version info` window shows version information for important cluster software components, such as the `CMDaemon` version, and the Bright Cluster Manager version.

Run command

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

Rack View

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

2.5 Cluster Management Shell

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

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

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

2.5.1 Invoking `cmsh`

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

Example

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

By default it connects to the IP address of the local management network interface, using the default Bright Cluster Manager port. If it fails to connect as in the preceding example, but a connection takes place using `cmsh localhost`, then the management interface is most probably not up. In that case, bringing the management interface up allows `cmsh` to connect to `CMDaemon`.

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

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

Batch Mode And Piping In `cmsh`

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

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

Alternatively, commands can be piped to `cmsh`:

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

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

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

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

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

Defining Command Aliases In `cmsh`

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

Example

```
alias lv device list virtualnode
```

Aliases in `cmsh` can be listed with the `alias` command.

Built-in Aliases In `cmsh`

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

```
[bright90]% alias
alias - goto -
alias .. exit
alias / home
alias ? help
alias ds device status
```

The meanings are:

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

Options Usage For `cmsh`

The options usage information for `cmsh` is obtainable with `cmsh -h` (figure 2.5).

```
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

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
--color <yes/no> ..... Define usage of colors
--spool <directory> ..... Alternative /var/spool/cmd
--tty|t ..... Pretend a TTY is available
--noredirect|-r ..... Do not follow redirects
--norc|-n ..... Do not load cmshrc file on start-up
--noquitconfirmation|-Q ..... Do not ask for quit confirmation
--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
--disablemultiline|-m ..... Disable multiline support
--hide-events ..... Hide all events by default
--disable-events ..... Disable all events by default
```

Figure 2.5: 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 mode` levels. Mode levels and associated *objects* for a level make up a hierarchy available below the top-level.

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

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

Figure 2.6 shows the top-level commands available in `cmsh`. These commands are displayed when `help` is typed in at the top-level of `cmsh`:

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

Figure 2.6: Top level commands in cmsh

All levels inside cmsh provide these top-level commands.

Passing a command as an argument to help gets details for it:

Example

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

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

Options:   -x, --echo
           Echo all commands

           -q, --quit
           Exit immediately after error

[myheadnode]%
```

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

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

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

Example

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

Navigation Through Modes And Objects In `cmsh`

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

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

- The home command, which is aliased to /, takes the user from any mode depth to the top level.

Example

```
[bright90]% device
[bright90->device]% list
Type                Hostname (key)  MAC                Category  Ip                Network           Status
-----
EthernetSwitch     switch01        00:00:00:00:00:00          10.141.0.50   internalnet [ UP ]
HeadNode           bright90        00:0C:29:5D:55:46          10.141.255.254 internalnet [ UP ]
PhysicalNode       node001         00:0C:29:7A:41:78  default   10.141.0.1       internalnet [ UP ]
PhysicalNode       node002         00:0C:29:CC:4F:79  default   10.141.0.2       internalnet [ UP ]
[bright90->device]% exit
[bright90]% device
[bright90->device]% use node001
[bright90->device[node001]]% path
home;device;use node001;
[bright90->device[node001]]% home
[bright90]% home;device;use node001 #copy-pasted from path output earlier
[bright90->device[node001]]%
```

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

A cmsh input line has the following syntax:

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

where *<mode>* and *<arg>* are optional. ²

Example

```
[bright90->network]% device status bright90; list
bright90 ..... [ UP ]
Name (key)      Type      Netmask bits  Base address  Domain name      Ipv6
-----
externalnet    External  16            192.168.1.0  brightcomputing.com no
globalnet      Global    0             0.0.0.0      cm.cluster
internalnet    Internal  16            10.141.0.0   eth.cluster
[bright90->network]%
```

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

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

Example

```
[bright90->network]% device; status bright90; list
bright90 ..... [ UP ]
Type                Hostname (key)  MAC                Category  Ip                Network           Status
```

² A more precise synopsis is:

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

```
-----
HeadNode      bright90      FA:16:3E:C8:06:D1      10.141.255.254 internalnet [ UP ]
PhysicalNode  node001      FA:16:3E:A2:9C:87  default  10.141.0.1      internalnet [ UP ]
[bright90->device]%

```

2.5.3 Working With Objects

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

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

Command	Description
<code>use</code>	Use the specified object. I.e.: Make the specified object the <i>current object</i>
<code>add</code>	Create the object and use it
<code>assign</code>	Assign a new object
<code>unassign</code>	Unassign an object
<code>clear</code>	Clear the values of the object
<code>clone</code>	Clone the object and use it
<code>remove</code>	Remove the object
<code>commit</code>	Commit local changes, done to an object, to CMDaemon
<code>refresh</code>	Undo local changes done to the object
<code>list</code>	List all objects at current level
<code>sort</code>	Sort the order of display for the <code>list</code> command
<code>format</code>	Set formatting preferences for <code>list</code> output
<code>foreach</code>	Execute a set of commands on several objects
<code>show</code>	Display all properties of the object
<code>swap</code>	Swap (exchange) the names of two objects
<code>get</code>	Display specified property of the object
<code>set</code>	Set a specified property of the object
<code>clear</code>	Set default value for a specified property of the object.
<code>append</code>	Append a value to a property of the object, for a multi-valued property
<code>removefrom</code>	Remove a value from a specific property of the object, for a multi-valued property
<code>modified</code>	List objects with uncommitted local changes
<code>usedby</code>	List objects that depend on the object
<code>validate</code>	Do a validation check on the properties of the object
<code>exit</code>	Exit from the current object or mode level

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

Working With Objects: `use`, `exit`

Example

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

```

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

Working With Objects: `add`, `commit`, `remove`

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

Example

```
[mycluster->device]% add physicalnode node100 10.141.0.100
[mycluster->device*[node100*]]% commit
[mycluster->device[node100]]% category add test-category
[mycluster->category*[test-category*]]% commit
[mycluster->category[test-category]]% remove test-category
[mycluster->category*]% commit
Successfully removed 1 Categories
Successfully committed 0 Categories
[mycluster->category]% device remove node100
[mycluster->category]% device
[mycluster->device*]% commit
Successfully removed 1 Devices
Successfully committed 0 Devices
[mycluster->device]%
```

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

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

The `add` command—syntax notes:

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

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

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

Arguments:
```

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

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

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

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

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

The `-d|--data` option to `remove`:

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

The `-a|--all` option to `remove`:

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

Working With Objects: `clone`, `modified`, `swap`

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

Example

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

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

The modified command corresponds roughly to the functionality of the Unsaved entities icon in figure 12.5.

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

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

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

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

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

```
[mycluster->device]% use mycluster
[mycluster->device[mycluster]]% interfaces
[mycluster->device[mycluster]->interfaces]% list
Type      Network device name  IP                Network
-----
physical  eth0 [dhcp]           10.150.4.46       externalnet
physical  eth1 [prov]           10.141.255.254    internalnet
[bright90->device[mycluster]->interfaces]% swap eth0 eth1; commit
[bright90->device[mycluster]->interfaces]% list
Type      Network device name  IP                Network
-----
physical  eth0 [prov]           10.141.255.254    internalnet
physical  eth1 [dhcp]           10.150.4.46       externalnet
[mycluster->device[mycluster]->interfaces]% exit; exit
```

Working With Objects: get, set, refresh

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

Example


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

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

The refresh command undoes the changes made, and corresponds roughly to the Revert button in Bright View when viewing Unsaved entities (figure 12.5). The modified command then confirms that no local changes exist. Finally the get command reconfirms that no local change took place.

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

Example

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

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

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
[mycluster->device*]%
```

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

Working With Objects: list, format, sort

The list command is used to list 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, ethernet switch and ip properties for each object.

Example

```
[bright90->device]% list -f hostname:14,ethernet switch:15,ip
hostname (key) ethernet switch ip
-----
apc01                               10.142.254.1
bright90      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
[bright90->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

```
[bright90->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 : Additional hostnames that should resolve to the interfaces IP address
banks        : Number of banks
...
```

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

```
[bright90->device]% format hostname:10
[bright90->device]% list
hostname (
-----
apc01
bright90
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

```
[bright90->device]% format hostname:[10-14]
[bright90->device]% list
hostname (key)
-----
apc01
bright90
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

```
[bright90->device]% format hostname:[10-14],ethernetswitch:14,ip:20
[bright90->device]% list
hostname (key) ethernetswitch ip
-----
apc01                               10.142.254.1
bright90      switch01:46           10.142.255.254
node001       switch01:47           10.142.0.1
node002       switch01:45           10.142.0.2
switch01                               10.142.253.1
```

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

Example

```
[bright90->device]% format -r
[bright90->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]
[bright90->device]%
```

The `sort` command sets the alphabetical sort order for the output of the `list` according precedence of the parameters specified.

Example

```
[bright90->device]% sort type mac
[bright90->device]% list -f type:15,hostname:15,mac
type          hostname (key) mac
-----
HeadNode      bright90      08:0A:27:BA:B9:43
PhysicalNode   node002        00:00:00:00:00:00
PhysicalNode   log001         52:54:00:DE:E3:6B
[bright90->device]% sort type hostname
[bright90->device]% list -f type:15,hostname:15,mac
type          hostname (key) mac
-----
HeadNode      bright90      08:0A:27:BA:B9:43
PhysicalNode   log001        52:54:00:DE:E3:6B
```

```
PhysicalNode    node002          00:00:00:00:00:00

[bright90->device]% sort mac hostname
[bright90->device]% list -f type:15,hostname:15,mac
type           hostname (key)  mac
-----
PhysicalNode   node002          00:00:00:00:00:00
HeadNode       bright90         08:0A:27:BA:B9:43
PhysicalNode   log001           52:54:00:DE:E3:6B
```

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

```
[bright90->device]% list -f type:15,hostname:15,mac --sort type,mac
[bright90->device]% list -f type:15,hostname:15,mac --sort type,hostname
[bright90->device]% list -f type:15,hostname:15,mac --sort mac,hostname
```

Working With Objects: `append`, `removefrom`

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

Example

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

Working With Objects: `usedby`

Removing a specific object is only possible if other objects do not have references to it. To help the administrator discover a list of objects that depend on (“use”) the specified object, the `usedby` command may be used. In the following example, objects depending on device `apc01` are requested. The `usedby` property of `powerdistributionunits` indicates that device objects `node001` and `node002` contain references to (“use”) the object `apc01`. In addition, the `apc01` device is itself displayed as being in the up state, indicating a dependency of `apc01` on itself. If the device is to be removed, then the 2 references to it first need to be removed, and the device also first has to be brought to the CLOSED state (page 198) by using the `close` command.

Example

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

Working With Objects: validate

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

Example

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

Working With Objects: show

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

```
[mycluster->device[node001]]% show
Parameter                                     Value
-----
Activation                                     Thu, 03 Aug 2017 15:57:42 CEST
BMC Settings                                   <submode>
Block devices cleared on next boot
Category                                       default
...
Data node                                     no
Default gateway                               10.141.255.254 (network: internalnet)
...
Software image                                 default-image
Static routes                                  <0 in submode>
...
```

Working With Objects: assign, unassign

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

For example,

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

The only place where the `assign` command is currently used within `cmsh` is within the `roles` sub-mode, available under the main category mode or the main device mode. Within `roles`, `assign` is used for assigning tasklike roles objects.

Working With Objects: import For Roles

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

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% device roles node001
[bright90->device[node001]->roles]% list
Name (key)
-----
[category:default] cgroupsupervisor
[category:default] slurmclient
```

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

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

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

Example

```
[bright90->device[node001]->roles]% import<TAB><TAB>
beegfs::admon cephmnds cloudgateway elasticsearch
...and other available roles including slurmclient...
[bright90->device[node001]->roles]% import --overlay slurm-client slurmclient
[bright90->device*[node001*]->roles*]% list
Name (key)
-----
[category:default] cgroupsupervisor
slurmclient
[bright90->device*[node001*]->roles*]% set slurmclient queues node1q
[bright90->device*[node001*]->roles*]% commit
```

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

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

2.5.4 Accessing Cluster Settings

The management infrastructure of 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`.

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

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% partition use base
[myheadnode->partition[base]]% show
Parameter                                     Value
-----
Administrator e-mail
ArchOS                                         <0 in submode>
BMC Settings                                 <submode>
Burn configs                                 <2 in submode>
Cluster name                                 my-cluster
Default burn configuration                   default-destructive
Default category                             default
Default software image                       default-image
External network                             externalnet
Externally visible IP
Failover                                     not defined
Failover groups                              <0 in submode>
Headnode                                     my-headnode
Management network                           internalnet
Name                                          base
Name servers                                192.168.101.1
Name servers from dhcp                       4.2.2.4
No zero conf                                 no
Node basename                               node
Node digits                                  3
Notes                                        <0 bytes>
Provisioning Node Auto Update Timeout       300
Relay Host
Revision
SNMP Settings
Search domains                               example.com
Sign installer certificates                 AUTO
Time servers                                0.pool.ntp.org,1.pool.ntp.org,2.pool.ntp.org
Time zone                                    America/Los_Angeles
UCS Profiles                                 <0 in submode>
```

2.5.5 Advanced `cmsh` Features

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

Command Line Editing

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

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

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

History And Timestamps

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

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

Example

```
[bright90->device[node001]]% history | tail -3
162 use node001
163 history
164 history | tail -3
[bright90->device[node001]]% history -t | tail -3
163 Thu Dec  3 15:15:18 2015 history
164 Thu Dec  3 15:15:43 2015 history | tail -3
165 Thu Dec  3 15:15:49 2015 history -t | tail -3
```

This history is saved in the file `.cm/.cmshhistory` in the `cmsh` user's directory.

Mixing `cmsh` And Unix Shell Commands

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

Example

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

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

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

Example

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


Output Redirection

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

Example

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

The ssh Command

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

Example

```
[bright90]% device ssh node001
<screen is cleared>
<some MOTD text and login information is displayed>
[root@node001 ~]# exit
Connection to node001 closed.
[bright90]% device use bright90
[bright90->device[bright90]]% #now let us connect to the head node from the head node object
[bright90->device[bright90]]% ssh
<screen is cleared>
<some MOTD text and login information is displayed>
[root@bright90 ~]# exit
logout
Connection to bright90 closed.
[bright90->device[bright90]]%
```

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

The time Command

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

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

Example

```
[bright90->device]% time ds node001
node001 ..... [ UP ]
time: 0.108s
```

The watch Command

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

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

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

Example

```
[bright90->device]% watch newnodes
screen clears
Every 2.0s: newnodes                               Thu Dec  3 13:01:45 2015
No new nodes currently available.
```

Example

```
[bright90->device]% watch -n 3 status -n node001,node002
screen clears
Every 3.0s: status -n node001,node002             Thu Jun 30 17:53:21 2016
node001 .....[ UP ]
node002 .....[ UP ]
```

Looping Over Objects With `foreach`

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

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

```
foreach <object1> <object2> ... ( <command1>; <command2> ... )
```

Example

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

With the `foreach` command it is possible to perform `set` commands on groups of objects simultaneously, or to perform an operation on a group of objects. The `range` command (page 44) provides an alternative to it in many cases.

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

```
[root@bright90 ~]# cmsh -c "device help foreach"
```

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

- **Grouping options:**

- `-n|--nodes`, `-g|--group`, `-c|--category`, `-r|--rack`, `-h|--chassis`, `-e|--overlay`, `-l|--role`, `-m|--image`, `-u|--union`, `-i|--intersection`
- `-t|--type` `virtualnode|cloudnode|node|physicalnode|switch|ibswitch|ethernetswitch|racksensor|chassis|powerdistributionunit`

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

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

For example, with the list form, the `--category` (`-c`) option takes a node category argument (or several categories), while the `--node` (`-n`) option takes a node-list argument. Node-lists (specification on page 42) can also use the following, more elaborate, syntax:

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

Example

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

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

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

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

- **Adding options:** `-o|--clone`, `-a|--add`

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

Example

```
[bright90->device]% foreach --clone node001 -n node022..node024 ()
[bright90->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
[bright90->device*]% commit
```

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

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

- **Conditional options:** `-s|--status`, `-q|--quitionunknown`

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

Example

```
[bright90->device]% foreach -n node001..node004 --status UP (get IP)
10.141.0.1
10.141.0.3
```

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

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

- **Looping options:** `*`, `-v|--verbose`

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

Example

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

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

Example

```
[bright90->category]% foreach * (get name; listnodes)
default
Type          Hostname  MAC                Category  Ip          Network      Status
-----
PhysicalNode  node001  FA:16:3E:79:4B:77  default   10.141.0.1  internalnet  [ UP ]
PhysicalNode  node002  FA:16:3E:41:9E:A8  default   10.141.0.2  internalnet  [ UP ]
PhysicalNode  node003  FA:16:3E:C0:1F:E1  default   10.141.0.3  internalnet  [ UP ]
```

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

Node List Syntax

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

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

- **sequential extended expansion (only for square brackets):**

example: `node[001-002]s[001-005]`

which is:

```
node001s001,node001s002,node001s003,node001s004,node001s005,\
node002s001,node002s002,node002s003,node002s004,node002s005
```

- **rack-based:**

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

- example: `r[1-2]n[01-03]`

which is: `r1n01,r1n02,r1n03,r2n01,r2n02,r2n03`

This might hint at two racks, `r1` and `r2`, with 3 nodes each.

- example: `rack[1-2]node0[1-3]`

which is: `rack1node01,rack1node02,rack1node03,rack2node01,`

`rack2node02,rack2node03`

Essentially the same as the previous one, but for nodes that were named more verbosely.

- **sequential exclusion (negation):**

example: `node001..node005,-node002..node003`

which is: `node001,node004,node005`

- **sequential stride (every <stride> steps):**

example: `node00[1..7:2]`

which is: `node001,node003,node005,node007`

- **mixed list:**

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

- example: `r1n001..r1n003,r2n003`

which is: `r1n001,r1n002,r1n003,r2n003`

- example: `r2n003,r[3-5]n0[01-03]`

which is: `r2n003,r3n001,r3n002,r3n003,r4n001,r4n002,r4n003,r5n001,r5n002,r5n003`

- example: `node[001-100],-node[004-100:4]`

which is: every node in the 100 nodes, except for every fourth node.

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@bright90 ~]# cat .cm/cmsh/.cmshrc
groupingsyntax auto
```

The range Command

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

Example

```
[bright90->device]% range -n node0[01-24]
[bright90->device{-n node001..024}]%
```

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

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

Example

```
[bright90->device{-n node001..024}]% range -c default
[bright90->device{-c default}]% ds
node001 ..... [  UP  ] state flapping
node002 ..... [  UP  ]
node003 ..... [  UP  ]
```

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

Example

```
[bright90->device{-c default}]% get revision
```

```
[bright90->device{-c default}]% set revision test
[bright90->device{-c default}]% get revision
test
test
test
```

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

Example

```
[bright90->device{-c default}]% roles; assign pbsproclient; commit
```

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

Example

```
[bright90->device{-c default}]% get ip
10.141.0.1
10.141.0.2
10.141.0.3
[bright90->device{-c default}]% ..
[bright90->device]% foreach -c default (get ip)
10.141.0.1
10.141.0.2
10.141.0.3
```

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

Example

```
[bright90->device]% range -n node[001-100]
[bright90->device{-n node[001-100]}]% pexec -n node[001-100] hostname
```

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

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

Example

```
[root@bright90 ~]# cmsh -c "device help range"
Name:      range - Set a range of several devices to execute future commands on

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

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

```

The bookmark **And** goto **Commands**

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

A bookmark can be

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

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

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

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

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

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 21) 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@bright90 ~]# cat .cm/cmsh/.cmshrc
bookmark -x mysaved
```

Renaming Nodes With The `rename` Command

Nodes can be renamed globally from within partition mode, in the Node `basename` field associated with the prefix of the node in Bright View (section 3.1.1) or in `cmsh` (section 2.5.4, and also page 73).

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

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

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

Example

```
[bright90->device]% rename node001..node003 test
Renamed: node001 to test1
Renamed: node002 to test2
Renamed: node003 to test3
```

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

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

Example

```
[bright90->device]% rename node[001-010] test
Renamed: node001 to test01
Renamed: node002 to test02
...
Renamed: node009 to test09
Renamed: node010 to test10
```

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

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

Example

```
[bright90->device]% rename node[001-003] test%04d
Renamed: node001 to test0001
Renamed: node002 to test0002
Renamed: node003 to test0003
```

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

Example

```
[bright90->device]% rename node[001-005] test0[1,2,5-7]
Renamed: node001 to test01
Renamed: node002 to test02
Renamed: node003 to test05
Renamed: node004 to test06
Renamed: node005 to test07
```

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

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

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% rename node[001-004] alice,bob,charlie,dave
Renamed: node001 to alice
Renamed: node002 to bob
Renamed: node003 to charlie
Renamed: node004 to dave
[bright90->device]% commit
```

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% rename node[001-100:2] left[001-050]
Renamed: node001 to left001
Renamed: node003 to left002
...
Renamed: node097 to left049
Renamed: node099 to left050
[bright90->device]% rename node[002-100:2] right[001-050]
Renamed: node002 to right001
Renamed: node004 to right002
...
Renamed: node098 to right049
Renamed: node100 to right050
[bright90->device]% commit
```

2.6 Cluster Management Daemon

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

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

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

On the head node, the cluster management daemon uses a MySQL database server to store all of its internal data. Raw monitoring data, on the other hand, is stored as binary data outside of the MySQL database (section 13.8).

2.6.1 Controlling The Cluster Management Daemon

It may be useful to shut down or restart the cluster management daemon. For instance, a restart may be necessary to activate changes when the cluster management daemon configuration file is modified. The cluster management daemon operation can be controlled through the following init script arguments to `service cmd`. The `cmdaemonctl` command also works with some arguments:

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

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

Example

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

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

Example

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

```
[root@bright90 etc]# service cmd full-status
CMDaemon version 1.9 is running (active).
Running locally

Current Time: Fri, 20 Sep 2019 12:25:21 CEST
Startup Time: Sun, 08 Sep 2019 22:37:13 CEST
Uptime: 1w 4d

CPU Usage: 3308.37u 1087.5s (0.4%)
Memory Usage: 872MB

Sessions Since Startup: 749
Active Sessions: 5

Number of occupied worker-threads: 1
Number of free worker-threads: 17

Connections handled: 51673
Requests processed: 221655
Total read: 96.1MB
Total written: 267MB

Average request rate: 0.2requests/s
Average bandwidth usage: 267.498B/s
```

Example

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

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

This uses `pdsh`, the parallel shell command (section 13.1).

2.6.2 Configuring The Cluster Management Daemon

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

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

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

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

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

2.6.3 CMDaemon Versions

Updating CMDaemon

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

```
yum update cmdaemon
```

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

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

Updating software on the cluster is covered in greater detail in Chapter 11.

CMDaemon Version Extraction

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

Example

```
[bright90->device]% cmdaemonversions
Hostname          Version index Version hash
-----
bright90          146,315      dba359716a
node001           146,315      dba359716a
node002           146,315      dba359716a
```

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

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

```
[bright90->device]% cmdaemonversions --join
Version index Version hash Count      Hostnames
-----
146,315      dba359716a  3          node001,node002,bright90
```

2.6.4 Configuring The Cluster Management Daemon Logging Facilities

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

CMDaemon Logging Configuration Global Debug Mode

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

Example

```
[root@bright90 ~]# cmdaemonctl -h
cmdaemonctl [OPTIONS...] COMMAND ...
```

Query or send control commands to the cluster manager daemon.

```
-h --help          Show this help
```

Commands:

```
debugon           Turn on CMDaemon debug
debugoff          Turn off CMDaemon debug
...
```

```
[root@bright90 ~]# cmdaemonctl debugon
CMDaemon debug level on
```

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

CMDaemon Subsystem Logging Configuration Debug Mode

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

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

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

CMDaemon Subsystem Logging Configuration Severity Levels

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

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

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

Example

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

CMDaemon Subsystem Logging Configuration Deployment

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

Example

```
[root@bright90 etc]# service cmd restart
```

or by reloading the logging configuration:

Example

```
[root@bright90 etc]# service cmd logconf
```

2.6.5 Configuration File Modification, And The `FrozenFile` Directive

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

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

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

Such a file will be entirely overwritten, unless the `FrozenFile` configuration file directive (Appendix C, page 715) 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:

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).

2.6.6 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 2.6.5) 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 2.6.5 or in the lists in Appendixes A.1, A.2 and A.3), are given in Appendix A.4.

2.6.7 CMDaemon Lite

Introduction

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

It can be installed on a device where the administrator considers the option of installing a regular, full-featured, CMDaemon to be overkill, but still wants an alternative so that some basic monitoring can be done on the device.

CMDaemon Lite is a Python service, and can be run on a device such as a standalone desktop, running Windows, Linux, or MacOS. It uses up one node license per node that it is run on. It requires Python 3.6 or higher.

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

Deployment

A zipped package can be picked up on the head node from the Bright Cluster Manager repositories with:

Example

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

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

```
[root@bright90 ~]# scp /cm/shared/apps/cm-lite-daemon-dist/cm-lite-daemon.zip \
  root@lite01:/opt/cm-lite-daemon.zip
[root@bright90 ~]# ssh root@lite01
[root@lite01 ~]# cd /opt
[root@lite01 opt]# unzip cm-lite-daemon.zip
[root@lite01 opt]# cd cm-lite-daemon
[root@lite01 cm-lite-daemon]# ls -ll --group-directories-first
total 44
drwxr-xr-x 7 root root 4096 Nov 11 13:22 cm_lite_daemon
```

```

drwxr-xr-x 2 root root 104 Nov 11 13:23 etc
drwxr-xr-x 2 root root 76 Oct 19 16:43 examples
drwxr-xr-x 2 root root 6 Oct 19 16:43 log
drwxr-xr-x 2 root root 78 Oct 19 16:43 service
-rwxr-xr-x 1 root root 4986 Oct 19 16:43 cm-lite-daemon
-rwxr-xr-x 1 root root 740 Oct 19 16:43 cm-lite-daemon_ctl
-rwxr-xr-x 1 root root 2469 Oct 19 16:43 connection_test
-rwxr-xr-x 1 root root 445 Oct 19 16:43 install-required-pip-packages
-rwxr-xr-x 1 root root 245 Oct 19 16:43 install-required-pip-packages.bat
-rwxr-xr-x 1 root root 5401 Oct 19 16:43 register_node
-rwxr-xr-x 1 root root 2808 Oct 19 16:43 request_certificate
-rwxr-xr-x 1 root root 3907 Oct 19 16:43 unregister_node

```

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

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

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

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

Example

```

[bright90->cert]% listrequests
Request ID  Client type  Session ID  Name
-----
1           Lite node    ....       ....

```

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

```
bright90->cert)% issuecertificate --days 10000 1
```

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

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

```

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

```

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

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

Example

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

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

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

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

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

To put things in perspective: so far the options described have been:

1. CMDaemon running on the device
2. CMDaemon Lite running on the device

A third option that can be considered, is to have

3. no CMDaemon at all running on the device and to register the device as a generic node with the regular CMDaemon on the head node. A generic node is a generic network device (page 75) that happens to be a node.

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

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

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

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

Example

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


3

Configuring The Cluster

After 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 11). This chapter however goes through a number of basic cluster configuration aspects that are required to get all the hardware up and running on the head and regular nodes.

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

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

Section 3.3 describes the setting up of network bridge interfaces.

Section 3.4 describes VLAN configuration.

Section 3.5 describes the setting up of network bond interfaces.

Section 3.6 covers how InfiniBand and Omni-Path are set up.

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

Section 3.8 describes how switches are set up.

Section 3.9 explains how disk layouts are configured, as well as how diskless nodes are set up.

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

3.1 Main Cluster Configuration Settings

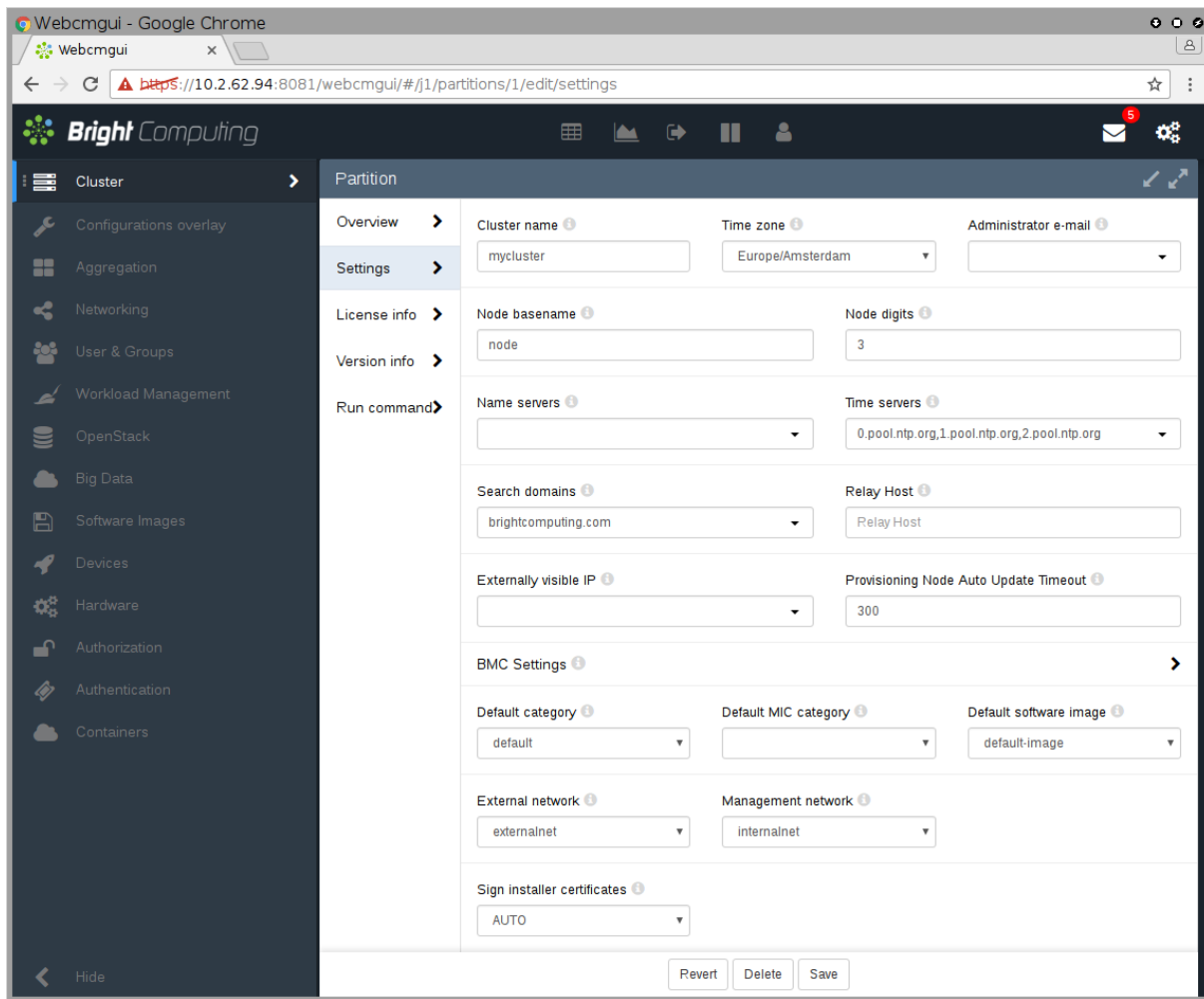


Figure 3.1: Cluster Settings

In Bright View, the Settings window 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. The values can in some cases be overridden by more specific configuration levels, such as category-level or node-level configuration. The main cluster settings in figure 3.1 are related to the cluster name-related settings, cluster networking, and some miscellaneous global cluster settings.

3.1.1 Cluster Configuration: Various Name-Related Settings

In the Settings window, the following settings are used for names throughout the cluster.

- Cluster name: (default name: Bright 9.0-stable Cluster).
- Default category: (default name: default)
- Default software image: (default name: default-image)
- How the nodes of the cluster are named:
 - Node name: the base prefix, also called *basename* (default prefix name: node)
 - Node digits size: number of digits in suffix of node name (default size: 3)

Node names can be managed globally in this window, or in the corresponding partition mode options (page 73) of the cluster object in `cmsh`. More specific renaming of node names is possible using the `rename` command from `device` mode of `cmsh`, (section 2.5.5, page 46).

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 settings window can be used to set the IP address of the nameserver and the names of the search domains for the cluster.

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

Instead of using the GUI, the changes to the `nameserver` and `searchdomain` values can instead be carried out using `cmsh` in `partition` mode (page 70).

The number of names that can be set as search domains used by the cluster has a maximum limit of 6 by default. This is a hardcoded limit imposed by the Linux operating system in older versions.

More recent versions of `glibc` (`glibc > 2.17-222.el7` in RHEL7) no longer set a limit. However using more than 6 search domains currently requires the use of the `GlobalConfig` directive, `MaximalSearchDomains`, in the `CMDDaemon` configuration file (Appendix C). For example, to set 30 domains, the directive setting would be: `GlobalConfig = { "MaximalSearchDomains=30" }`

Changing The Order In `resolv.conf`

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

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

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

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

Example

```
[bright90]% network use ibnet
[bright90->network[ibnet]]% show
...
Exclude from search domain      no
Search domain index             0
```

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

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

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

Example

```
[bright90]% partition
[bright90->partition[base]]% get searchdomains
brightcomputing.com:1
domain.test:6
```

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

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

Externally Visible IP Address

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

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 Bright View, the time zone can be selected from a menu displayed by clicking on the Time zone menu selection (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

```
[bright90]% partition use base
[bright90->partition[base]]% set timezone america/los_angeles
[bright90->partition*[base*]]% commit
```

3.1.3 Miscellaneous Settings

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

The BMC (Baseboard Management Controller) access settings can be configured by clicking on the BMC Settings option in Bright View. This opens up a subwindow with the following settings:

- User name: (default: bright)
- Password: (default: random string generated during head node installation)
- User ID: (default: 4)
- Power reset delay: During a reset, this is the time, in seconds, that the machine is off, before it starts powering up again (default: 0)

BMC configuration is discussed in more detail in section 3.7.

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 Bright View, an e-mail address can be set in the Administrator e-mail field (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

```
[bright90]% partition use base
[bright90->partition[base]]% set administratore-mail alf@example.com beth@example.com
[bright90->partition*[base*]]% commit
```

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

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

SMTP Relay Host Mailserver Setting

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

- in Bright View: the Relay Host field sets the SMTP relay host for the cluster resource
- in `cmsh`: the `relayhost` property can be set for the base object within partition mode:

Example

```
[root@bright90 ~]# cmsh
[bright90]% partition use base
[bright90-> partition[base]]% set relayhost mail.example.com
[bright90-> partition[base*]]% commit
```

Postfix on the regular nodes is configured to use the head node as a relay host and is normally left untouched by the administrator. If the regular node configuration for Postfix is changed in partition mode, then a node reboot deploys the change for the node. Further Postfix changes can be done directly to the configuration files as is done in the standard distribution. The changes must be done after the marked auto-generated sections, and should not conflict with the auto-generated sections.

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

```
[root@bright90 ~]# mailq; ls -al /var/spool/mail/root
Mail queue is empty
-rw----- 1 root mail 0 Sep  8 11:11 /var/spool/mail/root
[root@bright90 ~]# cmsh -c "main; testemail"
Mail delivered to postfix
You have new mail in /var/spool/mail/root
[root@bright90 ~]# mailq; ls -al /var/spool/mail/root
Mail queue is empty
-rw----- 1 root mail 749 Sep  8 11:12 /var/spool/mail/root
```

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

Failover Settings

To access the high availability (HA) feature of the cluster for head nodes, the administrator can click on the `Failover` option in Bright View. This opens up a subwindow that can be used to configure HA-related parameters (section 14.4.6).

Failover Groups Settings

To access the high availability feature of the cluster for groups of regular nodes, the administrator can click on the `Failover groups` option in Bright View. This opens up a subwindow that can be used to configure failover-groups-related parameters (section 14.5).

Burn Configs

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

- `Cluster [Partition base] → Settings → Default burn configuration`
This allows the `Default burn configuration` for a node burn run to be modified.
- `Cluster [Partition base] → Settings → Burn configs`
This lists the possible burn configuration settings in a subwindow, and allows changes to some of their properties of each item of the list.

3.1.4 Limiting The Maximum Number Of Open Files

Configuring The System Limit On Open Files: The `/proc/sys/fs/file-max` Setting

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

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

Configuring The User Limit On Open Files: The `nofile` Setting

The maximum number of open files allowed for a user can be seen on running `ulimit -n`. The value is defined by the `nofile` parameter of `ulimit`.

By default the value set by Bright Cluster Manager is 131072.

`Ulimit` limits are limits to restrict the resources used by users. If the `pam_limits.so` module is used to apply `ulimit` limits, then the resource limits can be set via the `/etc/security/limits.conf` file and `*.conf` files in the `/etc/security/limits.d` directory. Further information on these files can be found via the man page, `man(5) limits.conf`.

Resource limits that can be set for user login sessions include the number of simultaneous login sessions, the number of open files, and memory limits.

The maximum number of open files for a user is unlimited by default in an operating system that is not managed by Bright Cluster Manager. However, it is set to 131072 by default for a system managed by Bright Cluster Manager. The `nofile` value is defined by Bright Cluster Manager in:

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

The values set in `91-cm-limits.conf` are typically sufficient for a user session, unless the user runs applications that are resource hungry and consume a lot of open files.

Deciding On Appropriate Ulimit, Limit, And System Limit Values

Decreasing the `nofile` value in `/etc/security/limits.d/91-cm-limits.conf` (but leaving the `/proc/sys/fs/file-max` untouched), or increasing `/proc/sys/fs/file-max` (but leaving the `nofile` value of 131072 per session as is), may help the system stay under the maximum number of open files allowed.

In general, users should not be allowed to use the head node as a compilation server, or as a 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 server, and the load on it, the administrator may wish to increase the resource limit values.

A very rough rule-of-thumb that may be useful as a first approximation to set `file-max` optimally is suggested in the kernel source code. The suggestion is to simply multiply the system memory (in MB) by 10 per MB, and make the resulting number the `file-max` value. For example, if the node has 128 GB of memory, then 1280000 can be set as the `file-max` value.

Fine-tuning to try and ensure that the operating system no longer runs out of file handles, and to try and ensure the memory limits for handling the load are not exceeded, is best achieved via an empirical trial-and-error approach.

3.2 Network Settings

A simplified quickstart guide to setting up the external head node network configuration on a vendor-prepared cluster is given in Chapter 6 of the *Installation Manual*. This section (3.2) covers network configuration more thoroughly.

After the cluster is set up with the correct license as explained in Chapter 4 of the *Installation Manual*, the next configuration step is to define the networks that are present (sections 3.2.1 and 3.2.2).

During 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 72 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 68 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 75 under the subheading “Changing The Global Network Parameters For The Cluster”.

For a Type 1 cluster (section 3.3.9 of the *Installation Manual*) the internal and external networks are illustrated conceptually by figure 3.2.

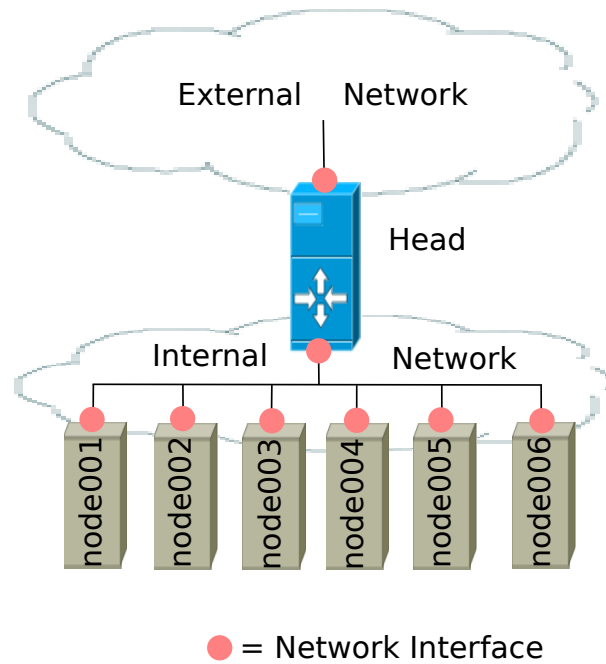


Figure 3.2: Network Settings Concepts

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

- Changing the configuration of the head node external interface is described on page 69 under the subheading “The IP address of the cluster”.
- Changing the configuration of the internal network interfaces is described on page 73 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 211.
- Changing the configuration of `globalnet` is described on page 75 under the subheading “Changing The Global Network Parameters For The Cluster”. IP addresses are not configured at the `globalnet` network level itself.

3.2.1 Configuring Networks

The `network` mode in `cmsh` gives access to all network-related operations using the standard object commands. Section 2.5.3 introduces `cmsh` modes and working with objects.

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

Networks					
NAME	NETMASK BITS	▲ BASE ADDRESS	DOMAIN NAME	TYPE	OPTIONS
<input type="checkbox"/> globalnet	0	0.0.0.0	cm.cluster	GLOBAL	Edit ▾
<input type="checkbox"/> internalnet	16	10.141.0.0	eth.cluster	INTERNAL	Edit ▾
<input type="checkbox"/> externalnet	24	192.168.200.0	openstacklocal	EXTERNAL	Edit ▾

Figure 3.3: Networks

In the context of the OSI Reference Model, each network object represents a layer 3 (i.e. Network Layer) IP network, and several layer 3 networks can be layered on a single layer 2 network (e.g. routes on an Ethernet segment).

Selecting a network such as `internalnet` or `externalnet` in the resource tree displays, for example, by double-clicking the row, opens up the `Overview` window of that network. The `Overview` window by default lists the device names and device properties that are in the network. For example the `internal` network typically has some nodes, switches, and other devices attached to it, each with their IP addresses and interface (figure 3.4).

Network internalnet				
Overview	TYPE	HOSTNAME	INTERFACE	IP
Settings	Node	node005	BOOTIF	10.141.0.5
	Node	node004	BOOTIF	10.141.0.4
	Node	node003	BOOTIF	10.141.0.3
	Node	node002	BOOTIF	10.141.0.2
	Node	node001	BOOTIF	10.141.0.1
	Node	pj-tr	eth0	10.141.255.254

Figure 3.4: Network Overview

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

The screenshot shows the 'Network internalnet' settings page. On the left, there is a sidebar with 'Overview' and 'Settings' (selected). The main area contains the following settings:

- name:** internalnet
- Domain Name:** eth.cluster
- Type:** Internal (dropdown menu)
- MTU:** 1500
- Node booting:** Enabled (radio button)
- Lock down dhcpd:** Disabled (radio button)
- Management allowed:** Enabled (radio button)
- Base address:** 10.141.0.0
- Broadcast address:** 10.141.255.255
- Dynamic range start:** 10.141.160.0
- Dynamic range end:** 10.141.167.255
- Netmask bits:** 16 (dropdown menu)
- Gateway:** 0.0.0.0

At the bottom, there are four buttons: Back, Revert, Delete, and Save.

Figure 3.5: Network Settings

The properties of figure 3.5 are introduced in table 3.2.1.

Property	Description
name	Name of this network.
Domain Name	DNS domain associated with the network.
Type	Menu options to set the network type. Options are Internal, External, Tunnel, Global, Cloud, or NetMap network.
MTU	Maximum Transmission Unit. The maximum size of an IP packet transmitted without fragmenting.
Node booting	Enabling means nodes are set to boot from this network (useful in the case of nodes on multiple networks). For an internal subnet called <i><subnet></i> , when node booting is set, CMDaemon adds a subnet configuration <code>/etc/dhcpd.<subnet>.conf</code> on the head node, which is accessed from <code>/etc/dhcpd.conf</code> . <ul style="list-style-type: none"> • It can be set in Bright View via the Networking resource, selecting a network, and then setting Node booting from within the network settings. • It can be set in cmsh via the network mode, selecting a network, and then setting nodebooting.

...continues

...continued

Property	Description
Lock down dhcpd	<p>Enabling means new nodes are not offered a PXE DHCP IP address from this network, i.e. DHCPD is “locked down”. A DHCP “deny unknown-clients” option is set by this action, so no new DHCP leases are granted to unknown clients for the network. Unknown clients are nodes for which Bright Cluster Manager has no MAC addresses associated with the node.</p> <ul style="list-style-type: none"> • It can be set in Bright View via the Networking resource, selecting a network, and then setting Lock down dhcpd from within the network settings. • It can be set in cmsh via the network mode, selecting a network, and then setting lockdowndhcpd.
Management allowed	Enabling means that the network has nodes managed by the head node.
Base address	Base address of the network (also known as the <i>network address</i>).
Broadcast address	Broadcast address of the network.
Dynamic range start/end	Start/end IP addresses of the DHCP range temporarily used by nodes during PXE boot on the internal network. These are addresses that do not conflict with the addresses assigned and used by nodes during normal use.
Netmask bits	Prefix-length, or number of bits in netmask. The part after the “/” in CIDR notation.
Gateway	Default route IP address

Table 3.2.1: 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@bright90:~$ sipcalc 192.168.0.1/28
-[ipv4 : 192.168.0.1/28] - 0

[CIDR]
Host address           - 192.168.0.1
Host address (decimal) - 3232235521
Host address (hex)     - C0A80001
Network address       - 192.168.0.0
Network mask          - 255.255.255.240
Network mask (bits)   - 28
Network mask (hex)    - FFFFFFF0
Broadcast address     - 192.168.0.15
Cisco wildcard        - 0.0.0.15
Addresses in network  - 16
```

Network range	- 192.168.0.0 - 192.168.0.15
Usable range	- 192.168.0.1 - 192.168.0.14

Every network has an associated DNS domain which can be used to access a device through a particular network. For `internalnet`, the default DNS domain is set to `eth.cluster`, which means that the hostname `node001.eth.cluster` can be used to access device `node001` through the primary internal network. If a dedicated storage network has been added with DNS domain `storage.cluster`, then `node001.storage.cluster` can be used to reach `node001` through the storage network. Internal DNS zones are generated automatically based on the network definitions and the defined nodes on these networks. For networks marked as external, no DNS zones are generated.

3.2.2 Adding Networks

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

In Bright View the Add button in the Networks subwindow (figure 3.3) can be used to add a new network. After the new network has been added, the Settings pane (figure 3.5) can be used to further configure the newly-added network.

In `cmsh`, a new network can be added from within `network` mode using the `add` or `clone` commands.

The default assignment of networks (`internalnet` to Management network and `externalnet` to External network) can be changed using Bright View, via the Settings window of the cluster object (figure 3.1).

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% partition use base
[bright90->partition[base]]% set managementnetwork internalnet; commit
[bright90->partition[base]]% set externalnetwork externalnet; commit
```

3.2.3 Changing Network Parameters

After both internal and external networks are defined, it may be necessary to change network-related parameters from their original or default installation settings.

Changing The Head Or Regular Node Hostname

To reach the head node from inside the cluster, the alias `master` may be used at all times. Setting the hostname of the head node itself to `master` is not recommended.

The name of a cluster is sometimes used as the hostname of the head node. The cluster name, the head node hostname, and the regular node hostnames, all have their own separate names as a property of their corresponding objects. The name can be changed in a similar manner for each.

For example, to change the hostname of the head node, the device object corresponding to the head node must be modified.

- In Bright View, the click path for modifying the host name is `Devices→Head Nodes→Edit→Settings→Hostname→Save`. That is, under the `Devices` resource, the `Head Nodes` option is selected. The `Edit` button can then be used to edit the host. This opens up a pane, and the `Settings` option can then be selected. The `Hostname` record can then be modified (figure 3.6), and saved by clicking on the `Save` button. When setting a hostname, a domain is not included.

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

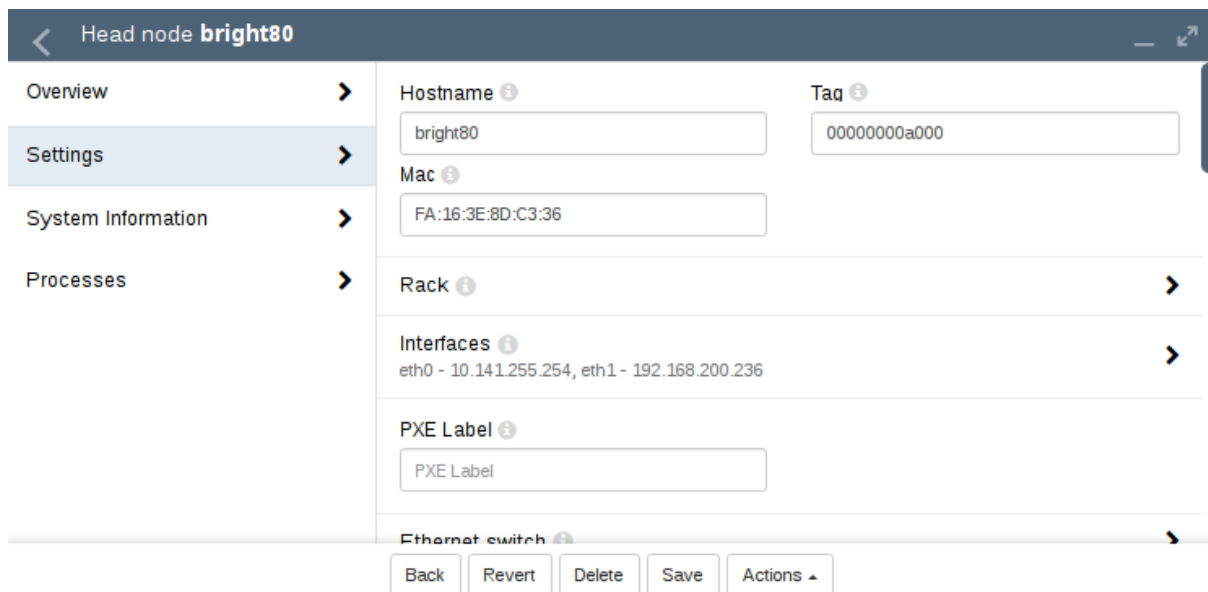


Figure 3.6: Head Node Settings

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use bright90
[bright90->device[bright90]]% 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@bright90 ~]# sleep 30; hostname -f foobar.cm.cluster
[root@bright90 ~]#
```

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

```
[bright90]% device use bright90
[bright90->device[bright90]]% interfaces
[bright90->device[bright90]->interfaces]% use eth1
[bright90->device[bright90]->interfaces[eth1]]% set additionalhostnames test
[bright90->device*[bright90*]->interfaces*[eth1*]]% commit
[bright90->device[bright90]->interfaces[eth1]]%
Fri Oct 12 16:48:47 2012 [notice] bright90: Service named was restarted
[bright90->device[bright90]->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 Bright View or `cmsh`.

Changing External Network Parameters For The Cluster

The external network parameters of the cluster: When a cluster interacts with an external network, such as a company or a university network, its connection behavior is determined by the settings of two objects: firstly, the external network settings of the `Networks` resource, and secondly, by the cluster network settings.

1. **The external network object** contains the network settings for all objects configured to connect to the external network, for example, a head node. Network settings are configured in Bright View under the `Networking` resource, then under the `Networks` subwindow, then within the `Settings` option for a selected network. Figure 3.5 shows a settings window for when the `internalnet` item has been selected, but in the current case the `externalnet` item must be selected instead. The following parameters can then be configured:

- the IP network parameters of the cluster (but not the IP address of the cluster):
 - `Base address`: the network address of the external network (the “IP address of the external network”). This is not to be confused with the IP address of the cluster, which is described shortly after this.
 - `Broadcast address`: the broadcast address of the external network. This is not to be confused with the IP address of the internal network (page 65).
 - `Dynamic range start` and `Dynamic range end`: Not used by the external network configuration.
 - `Netmask bits`: the netmask size, or prefix-length, of the external network, in bits.
 - `Gateway`: the default route for the external network.
- the `network name` (what the external network itself is called), by default this is defined as `externalnet` in the base partition on a newly installed Type 1 cluster,
- the `Domain Name`: the network domain (LAN domain, i.e. what domain machines on the external network use as their domain),
- the `External network` checkbox: this is checked for a Type 1 cluster,
- and `MTU size` (the maximum value for a TCP/IP packet before it fragments on the external network—the default value is 1500).

If the cluster is running the Bright Cluster Manager for OpenStack edition then some OpenStack parameters are also set:

- `OpenStack network type`: the OpenStack network type. Choices are: `None`, `Flat`, `VLAN Host`, `VLAN`, `Other`, `VXLAN Host`, `FloatingIps`

- `OpenStack DNS Format`: the format for instances that have a floating IP address assigned to them.
- `OpenStack network is shared`: If enabled, the OpenStack network is a shared, snoopable network.
- `OpenStack alloc pool start/end`: the start/end addresses of the OpenStack allocation.
- `OpenStack VLAN Range/ID`: The VLAN range and ID (if using VLANs).
- `OpenStack Host network`: Defaults to `vlanhostnet` (for VLANs) or `vxlanhostnet` (for VXLANs)
- `OpenStack Physical network name`: Name of the OpenStack physical network, defaults to `internalnet`

2. **The cluster object** contains other network settings used to connect to the outside. These are configured in the `Settings` options of the cluster object resource in Bright View (figure 3.1):

- e-mail address(es) for the cluster administrator,
- any additional external name servers used by the cluster to resolve external host names. As an aside: by default, only the head node name server is configured, and by default it only serves hosts on the internal network via the internal interface. Enabling the `PublicDNS` directive (Appendix C) allows the head node name server to resolve queries about internal hosts from external hosts via any interface, including the external interface.
- the DNS search domain (what the cluster uses as its domain),
- and NTP time servers (used to synchronize the time on the cluster with standard time) and time zone settings.

These settings can also be adjusted in `cmsh` in the base object under `partition` mode.

Changing the networking parameters of a cluster (apart from the IP address of the cluster) therefore requires making changes in the settings of the two preceding objects.

The IP address of the cluster: The cluster object itself does not contain an IP address value. This is because it is the cluster network topology type that determines whether a direct interface exists from the cluster to the outside world. Thus, the IP address of the cluster in the Type 1, Type 2, and Type 3 configurations (section 3.3.9 of the *Installation Manual*) is defined by the cluster interface that faces the outside world. For Type 1, this is the interface of the head node to the external network (figure 3.2). For Type 2 and Type 3 interfaces the cluster IP address is effectively that of an upstream router, and thus not a part of 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 Bright View or `cmsh`:

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

The external network object:

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

The screenshot shows the 'Network Settings' page for a network named 'externalnet'. The interface is divided into several sections:

- Overview/Settings:** A sidebar on the left with 'Settings' selected.
- name:** Input field containing 'externalnet'.
- Domain Name:** Input field containing 'eth.cluster'.
- Type:** Dropdown menu set to 'Internal'.
- MTU:** Input field containing '1500'.
- Node booting:** Toggle switch set to 'Enabled'.
- Lock down dhcpd:** Toggle switch set to 'Disabled'.
- Management allowed:** Toggle switch set to 'Enabled'.
- Base address:** Input field containing '10.141.0.0'.
- Broadcast address:** Input field containing '10.141.255.255'.
- Dynamic range start:** Input field containing '10.141.160.0'.
- Dynamic range end:** Input field containing '10.141.167.255'.
- Netmask bits:** Dropdown menu set to '16'.
- Gateway:** Input field containing '0.0.0.0'.
- OpenStack network type:** Dropdown menu set to 'None'.
- OpenStack DNS Format:** Input field containing '\$(uuid)'.
- OpenStack network is shared:** Toggle switch set to 'Disabled'.

At the bottom of the form are four buttons: 'Back', 'Revert', 'Delete', and 'Save'.

Figure 3.7: Network Settings For External Network

The cluster object:

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

The head node IP address:

For a head node `bright90`, where the IP address is used by the interface `eth0`, the static IP address can be set via the clickpath `Devices` → `Head Nodes` → `Edit [bright90]` → `Settings` → `Interfaces[eth0]` → `IP` (figure 3.8).

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

Example

```
[bright90]% network use externalnet
[bright90->network[externalnet]]% set baseaddress 192.168.1.0
[bright90->network*[externalnet*]]% set netmaskbits 24
[bright90->network*[externalnet*]]% set gateway 192.168.1.1
[bright90->network*[externalnet*]]% commit
[bright90->network[externalnet]]% partition use base
[bright90->partition[base]]% set nameservers 192.168.1.1
[bright90->partition*[base*]]% set searchdomains x.com y.com
[bright90->partition*[base*]]% append timeservers ntp.x.com
```

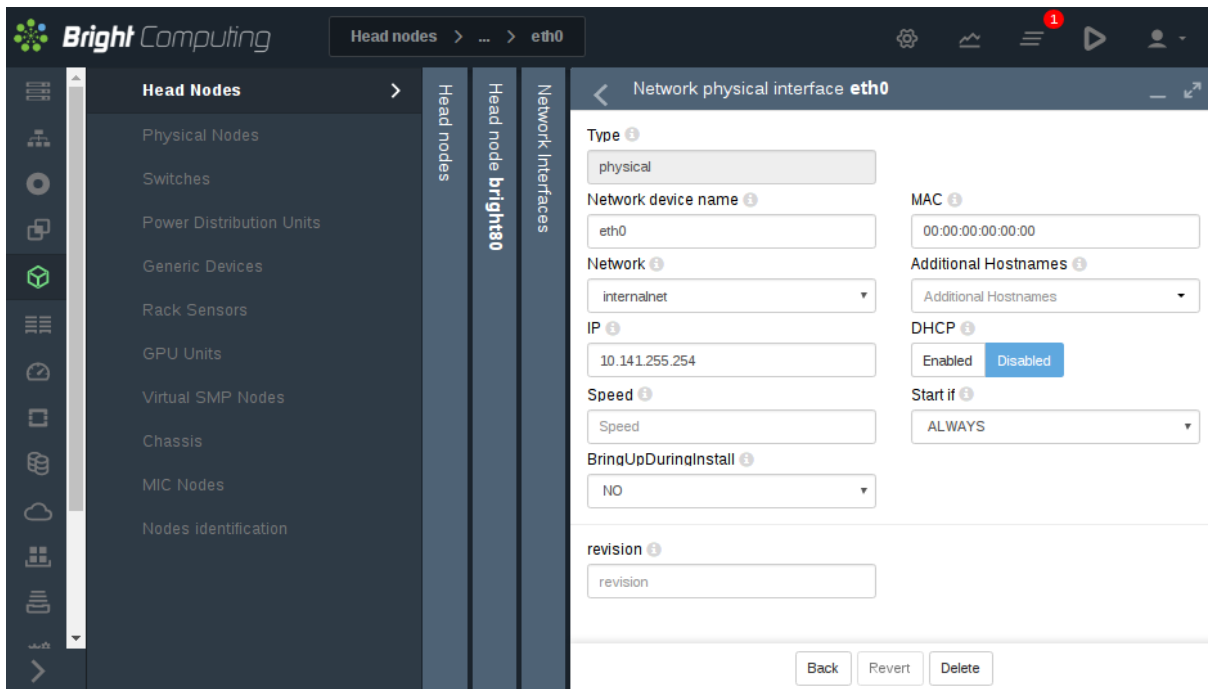



Figure 3.8: Setting The IP Address On A Head Node In Bright View

```
[bright90->partition*[base*]]% commit
[bright90->partition[base]]% device use bright90
[bright90->device[bright90]]% interfaces
[bright90->device[bright90]->interfaces]% use eth1
[bright90->device[bright90]->interfaces[eth1]]% set ip 192.168.1.176
[bright90->device[bright90]->interfaces*[eth1*]]% commit
[bright90->device[bright90]->interfaces[eth1]]%
```

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

Using DHCP to supply network values for the external interface: Connecting the cluster via DHCP on the external network is not generally recommended for production clusters. This is because DHCP-related issues can complicate networking troubleshooting compared with using static assignments.

For a Type 1 network, the cluster and head node can be made to use some of the DHCP-supplied external network values as follows:

- In Bright View, the DHCP setting of figure 3.8 can be set to Enabled
- Alternatively, in `cmsh`, within `interfaces` mode for the head node interface, the value of the parameter `DHCP` can be set:

```
[bright90->device[bright90]->interfaces[eth0]]% set dhcp yes
```

The gateway address, the name server(s), and the external IP address of the head node are then obtained via a DHCP lease. Time server configuration for `externalnet` is not picked up from the DHCP server, having been set during installation (figure 3.9 in Chapter 3 of the *Installation Manual*). The time servers can be changed using Bright View as in figure 3.1, or using `cmsh` in `partition` mode as in the preceding example. The time zone can be changed similarly.

It is usually sensible to reboot after implementing these changes in order to test the changes are working as expected.

Changing Internal Network Parameters For The Cluster

When a cluster interacts with the internal network that the regular nodes and other devices are on, its connection behavior with the devices on that network is determined by settings in:

1. the internal network of the `Networks` resource (page 72)
2. the cluster network for the internal network (page 73)
3. the individual device network interface (page 73)
4. the node categories network-related items for the device (page 74), in the case of the device being a regular node.

In more detail:

1. The internal network object: has the network settings for all devices connecting to the internal network, for example, a login node, a head node via its `internalnet` interface, or a managed switch on the internal network. In Bright View, the settings can be configured under the `Networking` resource, then under the `Networks` subwindow, then within the `Settings` option for the `internalnet` item (figure 3.5). In `cmsh` the settings can be configured by changing the values in the `internalnet` object within `cmsh`'s network mode. Parameters that can be changed include:

- the IP network parameters of the internal network (but not the internal IP address):
 - “Base address”: the internal network address of the cluster (the “IP address of the internal network”). This is not to be confused with the IP address of the internal network interface of the head node. The default value is `10.141.0.0`.
 - “Netmask bits”: the netmask size, or prefix-length, of the internal network, in bits. The default value is `16`.
 - Gateway: the default gateway route for the internal network. If unset, or `0.0.0.0` (the default), then its value is decided by the DHCP server on the head node, and nodes are assigned a default gateway route of `10.141.255.254`, which corresponds to using the head node as a default gateway route for the interface of the regular node. The effect of this parameter is overridden by any default gateway route value set by the value of `Default gateway` in the node category.
 - “Dynamic range start” and “Dynamic range end”: These are the DHCP ranges for nodes. DHCP is unset by default. When set, unidentified nodes can use the dynamic IP address values assigned to the node by the node-installer. These values range by default from `10.141.128.0` to `10.141.143.255`. Using this feature is not generally recommended, in order to avoid the possibility of conflict with the static IP addresses of identified nodes.
 - Node booting: This allows nodes to boot from the provisioning system controlled by `CM-Daemon`. The parameter is normally set for the management network (that is the network over which `CM-Daemon` communicates to manage nodes) but booting can instead be carried out over a separate physical non-management network. Booting over `InfiniBand` or `Omni-Path` is possible (section 5.1.3).
 - `Lock down dhcpd`, if set to `yes`, stops *new nodes* from booting via the network. New nodes are those nodes which are detected but the cluster cannot identify based on `CM-Daemon` records. Details are given in Chapter 5 about booting, provisioning, and how a node is detected as *new*.
- the “domain name” of the network. This is the LAN domain, which is the domain machines on this network use as their domain. By default, set to a name based on the network interface type used on the internal network, for example `eth.cluster`. In any case, the FQDN must be unique for every node in the cluster.

- the network name, or what the internal network is called. By default, set to `internalnet`.
- The MTU size, or the maximum value for a TCP/IP packet before it fragments on this network. By default, set to 1500.

2. The cluster object: has other network settings that the internal network in the cluster uses. These particulars are configured in the Settings option of the cluster object resource in Bright View (figure 3.1). The `cmsh` equivalents can be configured from the base object in partition mode. Values that can be set include:

- the “Management network”. This is the network over which `CMDaemon` manages the nodes. Management means that `CMDaemon` on the head node communicates with `CMDaemon`s 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 75). By default external hosts cannot use this service. To allow external hosts to use the service for resolving internal hosts, the `PublicDNS` directive (Appendix C) must be set to `True`.

3. The internal IP addresses and other internal interface values: In Bright View, the network for a node, such as a physical node `node001`, can be configured via a clickpath `Devices→Nodes[node001]→Interface`. The subwindow that opens up then allows network configuration to be done for such nodes in a similar way to figure 3.8.

In `cmsh` the configuration can be done by changing properties from the `interfaces` submode that is within the device mode for the node.

The items that can be set include:

- the Network device name: By default, this is set to `BOOTIF` for a node that boots from the same interface as the one from which it is provisioned.
- the Network: By default, this is set to a value of `internalnet`.
- the IP address: By default, this is automatically assigned a static value, in the range `10.141.0.1` to `10.141.255.255`, with the first node being given the first address. Using a static IP address is recommended for reliability, since it allows nodes that lose connectivity to the head node to continue operating. The static address can be changed manually, in case there is an IP address or node ID conflict due to an earlier inappropriate manual intervention.

Administrators who want DHCP addressing on the internal network, despite the consequences, can set it via a checkbox.

- `onnetworkpriority`: This sets the priority of DNS resolution queries for the interface on the network. The range of values that it can take is from 0 to 4294967295. Resolution takes place via the interface with the higher value.

The default priority value for a network interface is set according to its type. These defaults are:

Type	Value
Bridge	80
Bond	70
Physical	60
VLAN	50
Alias	40
Tunnel	30
Netmap	20
BMC	10

- **Additional Hostname:** In the case of nodes this is in addition to the default node name set during provisioning. The node name set during provisioning takes a default form of `node<3 digit number>`, as explained earlier on page 73 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

```
[bright90] device interfaces node001
[bright90->device[node001]->interfaces]% add physical eth1
[bright90->...->interfaces*[eth1*]]% set network externalnet
[bright90->...->interfaces*[eth1*]]% set additionalhostnames extra01
[bright90->...->interfaces*[eth1*]]% set ip 10.141.1.1
[bright90->...->interfaces*[eth1*]]% commit
[bright90->...->interfaces[eth1]]% ..
[bright90->...->interfaces]% ..
[bright90->device[node001]]% reboot
```

4. Node category network values: are settings for the internal network that can be configured for node categories. For example, for the default category called `default` this can be carried out:

- via Bright View, using the Settings option for Node categories. This is in the clickpath Grouping→Node categories[default-image]→Settings
- or via the category mode in `cmsh`, for the default node category.

If there are individual node settings that have been configured in Bright View or `cmsh`, then the node settings override the corresponding category settings for those particular nodes.

The category properties involved in internal networking that can be set include:

- **Default gateway:** The default gateway route for nodes in the node category. If unset, or 0.0.0.0 (the default), then the node default gateway route is decided by the internal network object `Gateway` value. If the default gateway is set as a node category value, then nodes use the node category value as their default gateway route instead.

- **Management network:** The management network is the network used by CMDaemon to manage devices. The default setting is a property of the node object. It can be set as a category property.
- **Name server, Time server, Search domain:** The default setting for these on all nodes is set by the node-installer to refer to the head node, and is not configurable at the node level using Bright View or `cmsh`. The setting can however be set as a category property, either as a space-separated list of entries or as a single entry, to override the default value.

Application To Generic Network Devices: The preceding details for the internal network parameters of the cluster, starting from page 72, 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 an ICMP ping (Appendix G.2.1).
- the device can be made to work with the health check and metric framework. The scripts used in the framework will however almost certainly have to be customized to work with the device

After changing network configurations, a reboot of the device is necessary to activate the changes.

Changing The Global Network Parameters For The Cluster

The global network `globalnet` is a unique network used to set up a common name space for all nodes in a cluster in 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 7.5.1 of the *Cloudbursting Manual*.

Setting Static Routes In The `staticroutes` Submode Of `cmsh`

To route via a specific gateway, the `staticroutes` submode can be used. This can be set for regular nodes and head nodes via the device mode, and for node categories via the category mode.

On a newly-installed cluster with a type 1 network (section 3.3.9 of the *Installation Manual*), a node by default routes packets using the head node as the default gateway.

If the administrator would like to configure a regular node to use another gateway to reach a printer on another subnet, as illustrated by figure 3.9:

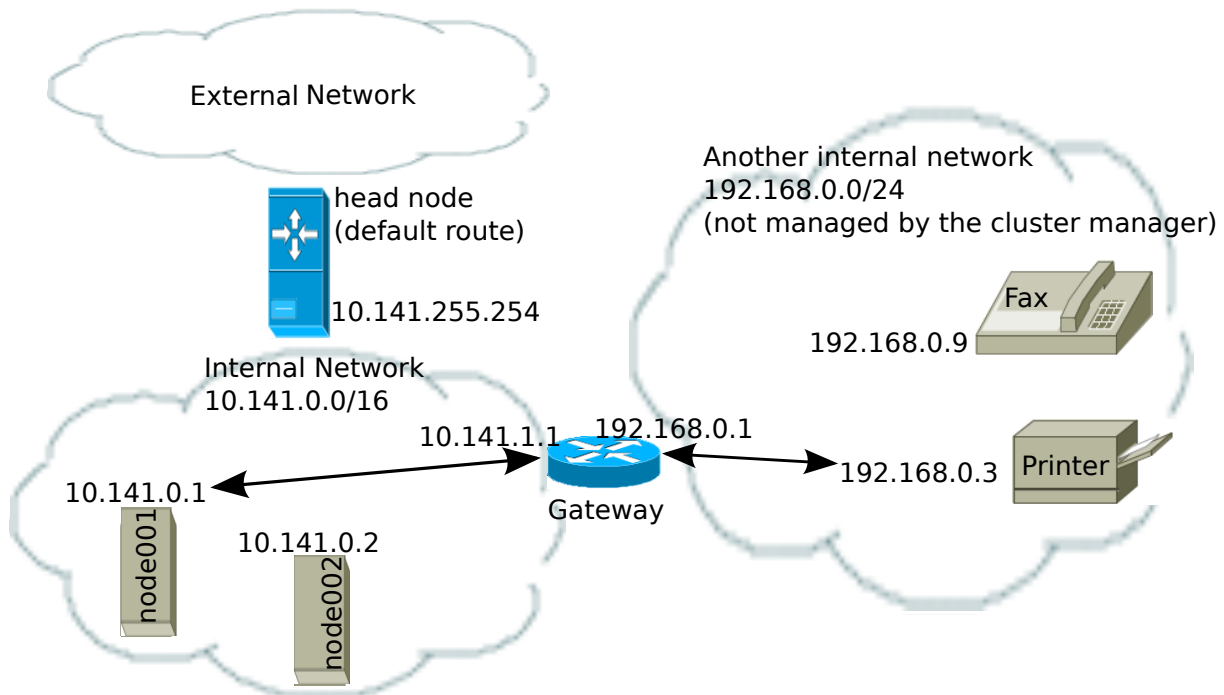


Figure 3.9: Example Of A Static Route To A Printer On Another Subnet

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

Example

```
[root@bright90 ~]# cmsh
[bright90 ]% device use node001
[bright90->device[node001]]%
[bright90->device[node001]]% staticroutes
[bright90->device[node001]->staticroutes]% list
Name (key)                Gateway                Destination
-----
[bright90->device[node001]->staticroutes]% add printerroute
[bright90->...*[printerroute*]]% set gateway 10.141.1.1
[bright90->...*[printerroute*]]% set destination 192.168.0.3
[bright90->...*[printerroute*]]% set networkdevicename bootif
[bright90->...*[printerroute*]]% commit
[bright90->...[printerroute]]% show
Parameter                  Value
-----
Destination                 192.168.0.3/32
Gateway                     10.141.1.1
Name                         printerroute
Network Device Name         bootif
Revision
[bright90->device[node001]->staticroutes[printerroute]]% exit
[bright90->device[node001]->staticroutes]% list
Name (key)                Gateway                Destination
-----
printerroute              10.141.1.1            192.168.0.3/32
[bright90->device[node001]->staticroutes]% exit; exit
[bright90->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 `printeroute` 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@bright90 ~]# ssh node001 ping -c1 -R 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(124) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=62 time=1.41 ms
RR:   10.141.0.1
      10.141.1.1
      192.168.0.1
      192.168.0.3
      192.168.0.3
      192.168.0.1
      10.141.1.1
      10.141.0.1
```

```
--- 192.168.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 1ms
rtt min/avg/max/mdev = 1.416/1.416/1.416/0.000 ms
[root@bright90->device[node001]->staticroutes]%
```

The routing is achieved by `CMDaemon` making sure that whenever the network interface is brought up, the OS of the regular node runs a routing command to configure the routing tables. The command executed for the given example is either:

```
route add -host 192.168.0.3 gw 10.141.1.1
```

or its modern `iproute2` equivalent:

```
ip route add 192.168.0.3 via 10.141.1.1
```

If the device `bootif` is `eth0`—which is often the case—then the command is run from the network interface configuration file: `/etc/sysconfig/network-scripts/ifcfg-eth0` (or `/etc/sysconfig/network/ifcfg-eth0` in SUSE).

3.2.4 Tools For Viewing Cluster Connections

Viewing Node Routes With `routes` In `cmsh`

The `cmsh routes` command is a wrapper around the Linux `route` command, and is designed to run in parallel over nodes. It is intended to get a fast route overview for one or more nodes, and display it for easy comparison.

To get a full overview of the routes for all nodes, the command is run without any options:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% routes
Node      Destination      Gateway          Netwask          Interface
```

```

-----
bright90  0.0.0.0      192.168.200.254  0.0.0.0      eth1
bright90  10.141.0.0     0.0.0.0         255.255.0.0   eth0
bright90  169.254.0.0    0.0.0.0         255.255.0.0   eth1
bright90  169.254.169.254 192.168.200.254 255.255.255.255 eth1
bright90  192.168.200.0  0.0.0.0         255.255.255.0   eth1
node001   0.0.0.0        10.141.255.254  0.0.0.0        eth0
node001   10.141.0.0     0.0.0.0         255.255.0.0    eth0
node001   169.254.0.0    0.0.0.0         255.255.0.0    eth0

```

To select or filter the output, the grouping options of routes, or the text processing utility `awk` can be used. Grouping options are options to select nodes, groups, categories, and so on, and are described in the cluster management chapter, on page 40, while a handy link for `awk` one-liners is <http://tuxgraphics.org/~guido/scripts/awk-one-liner.html>.

Selection or filtering makes it very easy to detect badly configured nodes. For example, for unexpected gateways, when the expected gateway is `<expected-gateway>`, the following command may be used:

Example

```

[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% routes --category default | awk 'if ($3 != "<expected-gateway>") print $0'
...

```

Viewing Connections In `cmsh`

The `connections` command can be run from within the device mode of `cmsh`. It is a parallel wrapper to view active TCP and UDP connections. That is, it runs as a simple command over multiple devices at the same time.

Running `connections` without options displays a full overview of the currently active TCP and UDP ports on all devices (output truncated):

Example

```

[bright90->device]% connections
Node      Type      Source      Port      Destination      Port      State
-----
bright90  TCP       0.0.0.0     22        0.0.0.0          0        Listening
bright90  TCP       0.0.0.0     25        0.0.0.0          0        Listening
bright90  TCP       0.0.0.0     111       0.0.0.0          0        Listening
bright90  TCP       0.0.0.0     636       0.0.0.0          0        Listening
...

```

Filtering with `grep` can then show which nodes are listening on which ports. For example, nodes listening for DNS queries (port 53) can be found with:

Example

```

[bright90->device]% connections | head -2; connections | grep Listen | grep " 53"
Node      Type      Source      Port      Destination      Port      State
-----
bright90  TCP       10.141.255.254  53        0.0.0.0          0        Listening
bright90  TCP       127.0.0.1     53        0.0.0.0          0        Listening
bright90  TCP       192.168.200.162 53        0.0.0.0          0        Listening

```


and shows that only the head node is listening on that port, providing DNS.

Some third party tools require a free port on all nodes for a service to listen on. Filtering and sorting the output of the connections command allows the administrator to find all the existing used ports across all nodes:

Example

```
[bright90->device]% connections | awk 'print $4' | sort -un
Port
22
25
53
67
68
69
...
```

Options to the connections command can be seen by running `help connections`. Options include node grouping options (such as `-n|--nodes`, `-c|--category`, and `-t|--type`), and filtering out TCP6 and UDP6 connections (`--no-tcp6`, `--no-udp6`).

Viewing Connectivity in `cmsh`

The connectivity command can be run from within the device mode of `cmsh`. It runs ICMP pings along each node route on a network of the cluster. By default the network is the management network.

By default the output displays if the connection is OK, the ping sequence ID (counting starts from zero), and the latency between the source and destination:

Example

```
[bright90->device]% connectivity
Source Destination Result ID Latency
-----
bright90 node001 Ok 0 0.4ms
bright90 node002 Ok 0 0.3ms
bright90 node003 Ok 0 0.4ms
bright90 bright90 Ok 0 0.1ms
```

Pings where the source and destination have identical names are carried out via identical interfaces.

Further options to the connectivity command can be seen by running `help connectivity`. Options include the ability to set ping timeouts and the cluster network on which to ping, as well as node grouping options.

For larger clusters, the following type of connectivity check may be a helpful diagnostic:

Example

```
[bright90->device]% connectivity --statistics --count 100 --delay 0.01
Name Value
-----
OK 400
Total 400
Count 2
Average 0.35ms
Minimum 0.35ms
Maximum 0.35ms
Uniformity 100.0%
```

The preceding shows how uniform nodes are in ping timings. A significant spread in uniformity can indicate network problems.

3.3 Configuring Bridge Interfaces

Bridge interfaces can be used to divide one physical network into two separate network segments at layer 2 without creating separate IP subnets. A bridge thus connects the two networks together at layer 3 in a protocol-independent way.

The network device name given to the bridge interface is of the form `br<number>`. The following example demonstrates how to set up a bridge interface in `cmsh`, where the name `br0` is stored by the parameter `networkdevicename`.

Example

```
[bright90->device[node001]->interfaces]% add bridge br0
[bright90->...->interfaces*[br0*]]% set network internalnet
[bright90->...->interfaces*[br0*]]% set ip 10.141.1.1
[bright90->...->interfaces*[br0*]]% show
```

Parameter	Value

Additional Hostnames	
DHCP	no
Forward Delay	0
IP	10.141.1.1
Interfaces	
MAC	00:00:00:00:00:00
Network	internalnet
Network device name	br0
Revision	
SpanningTreeProtocol	no
Type	bridge

A bridge interface is composed of one or more physical interfaces. The IP and network fields of the member interfaces must be empty:

Example

```
[bright90->...->interfaces*[br0*]]% set interfaces eth1 eth2; exit
[bright90->...->interfaces*]% clear eth1 ip; clear eth1 network
[bright90->...->interfaces*]% clear eth2 ip; clear eth2 network
[bright90->...->interfaces*]% use br0; commit
```

The `B00TIF` interface is also a valid interface option.

Currently, the following bridge properties can be set:

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

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

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

Example

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

Type	Network device name	IP	Network

bond	bond0 [prov]	10.141.128.1	internalnet
bridge	br0	10.141.128.2	internalnet

physical	eth0	10.141.0.1	internalnet
physical	eth1 (bond0)	0.0.0.0	
physical	eth2 (bond0)	0.0.0.0	
physical	eth3 (br0)	0.0.0.0	
physical	eth4 (br0)	0.0.0.0	

It is possible to create a bridge interface with no IP address configured, that is, with an IP address of 0.0.0.0. This can be done for security reasons, or when the number of available IP addresses on the network is scarce. As long as such a bridge interface has a network assigned to it, it is properly configured on the nodes and functions as a bridge on the network.

3.4 Configuring VLAN interfaces

A VLAN (Virtual Local Area Network) is an independent logical LAN within a physical LAN network. VLAN tagging is used to segregate VLANs from each other. VLAN tagging is the practice of inserting a VLAN ID tag into a packet frame header, so that each packet can be associated with a VLAN.

The physical network then typically has sections that are VLAN-aware or VLAN-unaware. The nodes and switches of the VLAN-aware section are configured by the administrator to decide which traffic belongs to which VLAN.

A VLAN interface can be configured for an interface within Bright Cluster Manager using `cmsh` or Bright View.

3.4.1 Configuring A VLAN Interface Using `cmsh`

In the following `cmsh` session, a regular node interface that faces a VLAN-aware switch is made VLAN-aware, and assigned a new interface—an alias interface. It is also assigned a network, and an IP address within that network:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device interfaces node001
[bright90->device[node001]->interfaces]% list
Type          Network device name IP          Network
-----
physical     BOOTIF [prov]      10.141.0.1 internalnet
[bright90->device[node001]->interfaces]% list -h | tail -4
Arguments:
  type
    alias, bmc, bond, bridge, netmap, physical, tunnel, vlan

[bright90->device[node001]->interfaces]% add vlan BOOTIF.1
[bright90->device*[node001*]->interfaces*[BOOTIF.1*]]% commit
[bright90->device[node001]->interfaces[BOOTIF.1]]% show
```

Parameter	Value

Additional Hostname	
DHCP	no
IP	0.0.0.0
MAC	00:00:00:00:00:00
Network	
Network device name	BOOTIF.1
Reorder HDR	no
Revision	
Type	vlan

```
[bright90->...[BOOTIF.1]]% set network internalnet
[bright90->...[BOOTIF.1*]]% set ip 10.141.2.1; commit
```

The Reorder HDR setting in a VLAN, if enabled, moves the ethernet header around to make it look exactly like a real ethernet device. This setting controls the REORDER_HDR setting in the file at `/etc/sysconfig/network-scripts/ifcfg-<interface>` on the node.

3.4.2 Configuring A VLAN Interface Using Bright View

Within Bright View a VLAN interface can be configured by selecting the node with that interface, selecting its Settings option, then selecting the Interfaces option. In the Network Interfaces window that opens up, the Network VLAN interface option is selected and a dialog opens up allowing an IP address, network, and other options to be set (figure 3.10).

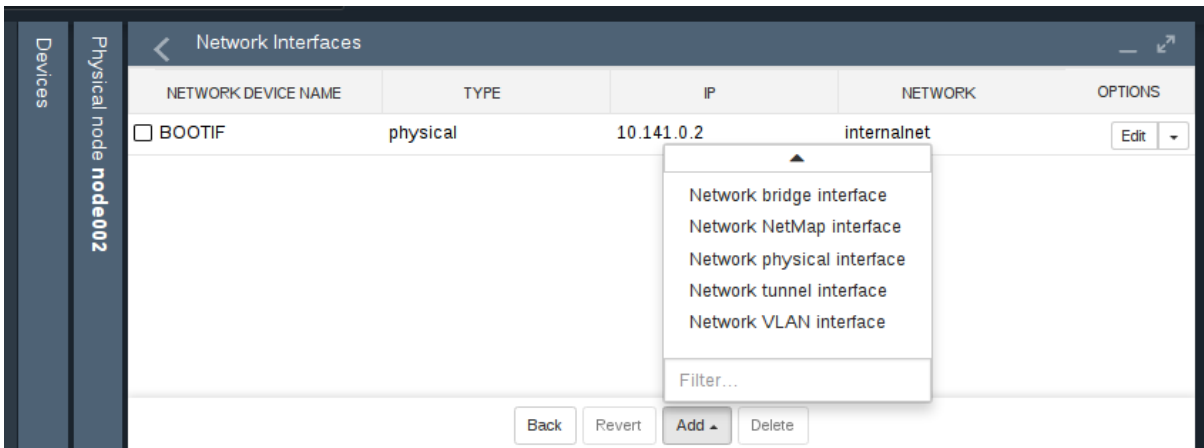


Figure 3.10: Configuring A VLAN via Bright View

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 74) 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

```
[bright90->device[node001]->interfaces]% add bond bond0
[...->device*[node001*]->interfaces*[bond0*]]% set network internalnet
[...->device*[node001*]->interfaces*[bond0*]]% set ip 10.141.128.1
[...->device*[node001*]->interfaces*[bond0*]]% set mode 3
[...->device*[node001*]->interfaces*[bond0*]]% set interfaces eth1 eth2
```

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

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

The 7 bonding modes are:

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

Technically, outside of Bright View 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 80.
- 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 14), 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 14.1) 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 14.1) 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 190.

Example

```
[headnode1->device[headnode1]->interfaces]% list
Type          Network device name  IP                Network
-----
alias         bond0:0              10.141.255.252   internalnet
alias         bond1:0              10.150.57.1      externalnet
bond          bond0 [prov]         10.141.255.254   internalnet
bond          bond1                10.150.57.3      externalnet
physical      eth0 (bond0)         0.0.0.0
physical      eth1 (bond0)         0.0.0.0
physical      eth2 (bond1)         0.0.0.0
physical      eth3 (bond1)         0.0.0.0
```

3.5.5 Tagged VLAN On Top Of a Bonded Interface

It is possible to set up a tagged VLAN interface on top of a bonded interface. There is no requirement for the bonded interface to have an IP address configured in this case. The IP address can be set to `0.0.0.0`, however a network must be set.

Example

```
[headnode1->device[node001]->interfaces]% list
Type          Network device name  IP                Network
-----
bond          bond0                0.0.0.0           internalnet
physical      eth0 [prov]          10.141.0.1        internalnet
physical      eth1 (bond0)         0.0.0.0
physical      eth2 (bond0)         0.0.0.0
vlan          bond0.3              10.150.1.1        othernet
```

3.5.6 Further Notes On Bonding

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

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

name. Section 3.3, on configuring bridge interfaces, shows an example of such a listing from within `cmsh` on page 80.

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 service¹ 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.

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

In `cmsh` this can be done with:

```
[root@bright90 ~]# cmsh -c "device roles <node>; assign subnetmanager; commit"
```

where `<node>` is the name of a node on which it will run, for example: `bright90`, `node001`, `node002`...

In Bright View, the subnet manager role is assigned by selecting a head node or regular node from the Devices resource, and assigning it the "Subnet manager role". The clickpath for this, for a node `node002` for example, is `Devices`→`Nodesnode002`→`Settings`→`Roles`→`Add [Subnet manager role]`.

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

In `cmsh` this is done with:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device services node001
[bright90->device[node001]->services]% add opensm
[bright90->device[node001]->services*[opensm*]]% set autostart yes
[bright90->device[node001]->services*[opensm*]]% set monitored yes
[bright90->device[node001]->services*[opensm*]]% commit
[bright90->device[node001]->services[opensm]]%
```

In Bright View the subnet manager service is configured by selecting a head node or regular node from the resources tree, and adding the service to it. The clickpath for this, for a node `node002` for example, is: `Devices`→`Nodesnode002`→`Settings`→`Services[Service]`.

When the head node in a cluster is equipped with an InfiniBand HCA, it is a good candidate to run as a subnet manager for smaller clusters.

On large clusters a dedicated node is recommended to run the subnet manager.

3.6.3 InfiniBand Network Settings

Although not strictly necessary, it is recommended that InfiniBand interfaces are assigned an IP address (i.e. IP over IB). First, a network object in the cluster management infrastructure should be created. The procedure for adding a network is described in section 3.2.2. The following settings are recommended as defaults:

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

By default, an InfiniBand interface is set to datagram mode, because it scales better than connected mode. It can be configured to run in connected mode by setting the `connectedmode` property:

Example


```
[bright90->device[node001]->interfaces[ib0]]% set connectedmode yes
```

For nodes that are PXE booting or are getting provisioned over InfiniBand, the mode setting in the node-installer script has to be changed accordingly.

Example

```
[root@bright90 ~]# 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@bright90 ~]# echo "device
addinterface -n node001..node150 physical ib0 ibnet 10.149.0.1
commit" | cmsg -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:

```
[bright90->device[bright90]->interfaces]% add physical ib0
[bright90->device[bright90]->interfaces*[ib0*]]% set network ibnet
[bright90->device[bright90]->interfaces*[ib0*]]% set ip 10.149.255.254
[bright90->device[bright90]->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@bright90 ~]# su - cmsupport
[cmsupport@bright90 ~]$ cd /cm/shared/apps/imb/current/
[cmsupport@bright90 current]$ ./setup.sh
[cmsupport@bright90 current]$ cd ~/BenchMarks/imb/2017
[cmsupport@bright90 2017]$ module load openmpi/gcc
[cmsupport@bright90 2017]$ module initadd openmpi/gcc
[cmsupport@bright90 2017]$ make -f make_mpi2
[cmsupport@bright90 2017]$ mpirun -np 2 -machinefile ../nodes IMB-MPI1 PingPong
#-----
# Benchmarking PingPong
# #processes = 2
#-----
      #bytes #repetitions      t[usec]  Mbytes/sec
      0         1000         0.78         0.00
      1         1000         1.08         0.88
      2         1000         1.07         1.78
      4         1000         1.08         3.53
      8         1000         1.08         7.06
     16         1000         1.16        13.16
     32         1000         1.17        26.15
     64         1000         1.17        52.12
    128         1000         1.20       101.39
    256         1000         1.37       177.62
    512         1000         1.69       288.67
   1024         1000         2.30       425.34
   2048         1000         3.46       564.73
   4096         1000         7.37       530.30
   8192         1000        11.21       697.20
  16384         1000        21.63       722.24
  32768         1000        42.19       740.72
  65536          640        70.09       891.69
 131072          320       125.46       996.35
 262144          160       238.04      1050.25
 524288           80       500.76       998.48
1048576           40      1065.28       938.72
2097152           20     2033.13       983.71
4194304           10     3887.00      1029.07

# All processes entering MPI_Finalize

```

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

3.7 Configuring BMC (IPMI/iLO/DRAC/CIMC/Redfish) Interfaces

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 (figure 3.16 of the *Installation Manual*), or it can be set after installation as described in this section. The IPMI, iLO, DRAC, CIMC, or Redfish interface that is exposed by a BMC is treated in the cluster management infrastructure as a special type of network interface belonging to a device. In the most common setup a dedicated network (i.e. IP subnet) is created for BMC communication. The 10.148.0.0/16 network is used by default for BMC interfaces by Bright Cluster Manager.

3.7.1 BMC Network Settings

The first step in setting up a BMC is to add the BMC network as a network object in the cluster management infrastructure. The procedure for adding a network is described in section 3.2.2. The following settings are recommended as defaults:

Property	Value
Name	bmcnet, ilonet, ipminet, dracnet, cimcnet, or rfnet
Domain name	bmc.cluster, ilo.cluster, ipmi.cluster, drac.cluster, cimc.cluster, or rf.cluster
Type	Internal
Base address	10.148.0.0
Netmask bits	16
Broadcast address	10.148.255.255

Once the network has been created, all nodes must be assigned a BMC interface, of type `bmc`, on this network. The easiest method of doing this is to create the interface for one node device and then to clone that device several times.

For larger clusters this can be laborious, and a simple bash loop can be used to do the job instead:

```
[bright90 ~]# 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 | cmsg -x      # -x usefully echoes what is piped into cmsg
```

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

```
[bright90 ~]# echo "
device
addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1
commit" | cmsg -x
```

The help text in `cmsg` gives more details on how to use `addinterface`.

Most administrators are likely to simply run it as an interactive session in `cmsg`, running the `addinterface` command for reference, and then supplying the options for the nodes and interface settings in device mode. For example, as in the following session:

```
[root@bright90 ~]# cmsg
[bright90]% device
[bright90->device]% help addinterface
Name:
    addinterface - Add a network interface to one or more nodes

Usage:
    addinterface [OPTIONS] <type> <devicename> <network> <firstip>

Options:
    ...help text omitted...

Examples:
    addinterface -n node001..node010 physical ib0 ibnet 10.149.0.1
```

```
[bright90->device]% addinterface -n node001..node150 bmc ipmi0 bmcnet 10.148.0.1
[bright90->device]% commit
```

In order to be able to communicate with the BMC interfaces, the head node also needs an interface on the BMC network. Depending on how the BMC interfaces are physically connected to the head node, the head node has to be assigned an IP address on the BMC network one way or another. There are two possibilities for how the BMC interface is physically connected:

- When the BMC interface is connected to the primary internal network, the head node should be assigned an alias interface configured with an IP address on the BMC network.
- When the BMC interface is connected to a dedicated physical network, the head node must also be physically connected to this network. A physical interface must be added and configured with an IP address on the BMC network.

Example

Assigning an IP address on the BMC network to the head node using an alias interface:

```
[bright90->device[bright90]->interfaces]% add alias eth0:0
[bright90->device[bright90]->interfaces*[eth0:0*]]% set network bmcnet
[bright90->device[bright90]->interfaces*[eth0:0*]]% set ip 10.148.255.254
[bright90->device[bright90]->interfaces*[eth0:0*]]% commit
[bright90->device[bright90]->interfaces[eth0:0]]%
Mon Dec 6 05:45:05 bright90: Reboot required: Interfaces have been modified
[bright90->device[bright90]->interfaces[eth0:0]]% quit
[root@bright90 ~]# service network restart
```

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

BMC connectivity from the head node to the IP addresses of the configured interfaces on the regular nodes can be tested with Bash one-liner such as:

Example

```
[root@bright90 ~]# for i in $(cmsh -c "device; foreach -t physicalnode (interfaces; \
use ilo0; get ip)"); do ping -c1 $i; done | grep packet
1 packets transmitted, 0 received, 100% packet loss, time 0ms
1 packets transmitted, 0 received, 100% packet loss, time 0ms
1 packets transmitted, 0 received, 100% packet loss, time 0ms
...
```

In the preceding example the packet loss demonstrates there is a connection problem between the head node and the BMC subnet.

3.7.2 BMC Authentication

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

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

Example

```
[bright90]% partition use base
[bright90->partition[base]]% bmcsettings
[bright90->partition[base]->bmcsettings]% get username
bright
[bright90->partition[base]->bmcsettings]% get password
Za4ohni1ohMa2zew
[bright90->partition[base]->bmcsettings]% set username bmcadmin
[bright90->partition*[base*]->bmcsettings*]% set password
enter new password: *****
retype new password: *****
[bright90->partition*[base*]->bmcsettings*]% commit
```

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

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:

Property	Description
BMC User ID	User type. Normally set to 4 for administrator access.
BMC User Name	User name
BMC Password	Password for specified user name

BMC configuration on a head node is done directly.

For regular nodes 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 after the node has been provisioned again.

If BMC authentication fails, then an explanation for why can often be found in the node-installer log at `/var/log/node-installer`.

3.7.3 Interfaces Settings

Interface Name

It is recommended that the network device name of a BMC interface start with `ipmi`, `ilo`, `drac`, `cimc`, or `rf`, according to whether the BMC is running with IPMI, iLO, DRAC, CIMC, or Redfish. Numbers are appended to the base name, resulting in, for example: `ipmi0`.

Obtaining The IP address

BMC interfaces can have their IP addresses configured statically, or obtained from a DHCP server.

Only a node with a static BMC IP address has BMC power management done by 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.

Dell racadm Installation

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

If Dell is chosen as a hardware option when nodes are configured in the Bright Cluster Manager installer, then the default software image that is used by the node has the Dell OpenManage utilities, located on the head node at `/cm/images/default-image/opt/dell`.

The Dell OpenManage utilities contain the `racadm` binary, which can be used to issue power commands (Chapter 4). Bright Cluster Manager runs commands similar to the following to issue the power commands:

```
/opt/dell/srvadmin/sbin/racadm -r <DRAC interface IP address> -u <bmcusername> -p <bmcpassword>\
serveraction powerstatus
/opt/dell/srvadmin/sbin/racadm -r <DRAC interface IP address> -u <bmcusername> -p <bmcpassword>\
serveraction hardreset
```

The BMC username/password values can be obtained from `cmsh` as follows:

```
[root@bright90 ~]# cmsh
[bright90]% partition use base
[bright90->partition[base]]% bmcsettings
[bright90->partition[base]->bmcsettings]% get password
12345
[bright90->partition[base]->bmcsettings]% get username
tom
[bright90->partition[base]->bmcsettings]%
```

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

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

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

Here it is assumed that the `bright` user has a `userID` 4, and the privilege can be set to 511.

3.7.4 Identification With A BMC

Sometimes it is useful to identify a node using BMC commands. This can be done by, for example, blinking a light via a BMC command on the node:

Example

```
ipmitool -U <bmcusername> -P <bmcpassword> -H <host IP> chassis identify 1
```

The exact implementation may be vendor-dependent, and need not be an `ipmitool` command. Such commands can be scripted and run from `CMDaemon`.

For testing without a BMC, the example script at `/cm/local/examples/cmd/bmc_identify` can be used if the environment variable `$CMD_HOSTNAME` is set. The logical structure of the script can be used as a basis for carrying out an identification task when a physical BMC is in place, by customizing the script and then placing the script in a directory for use.

To have such a custom BMC script run from `CMDaemon`, the `BMCIdentifyScript` advanced configuration directive (page 713) can be used.

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 Bright View, this is done by selecting the **Switches** resource, selecting the particular switch from within that resource (or adding it via the **Add** button), then filling in the values in the associated **Settings** window that comes up (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@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% set switch01 ip 10.141.253.2
[bright90->device*]% commit
```

3.8.2 Configuring SNMP

Configuring SNMP Community Strings

Moreover, in order to allow the cluster management software to communicate with the switch or PDU, SNMP must be enabled 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

```
[bright90]% device use switch01
[bright90->device[switch01]]% snmpsettings
[bright90->device[switch01]->snmpsettings]% get readstring
public
[bright90->device[switch01]->snmpsettings]% get writestring
private
[bright90->device[switch01]->snmpsettings]% set readstring public2
[bright90->device*[switch01*]->snmpsettings*]% set writestring private2
[bright90->device*[switch01*]->snmpsettings*]% commit
```

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

Devices→Switches→Edit→SNMP Settings

Configuring SNMP Settings

SNMP settings can be configured in `cmsh` via the `snmpsettings` submode, which is available under partition mode as well as under device mode.

The submode allows the version to be set to `v1`, `v2c`, or `v3`.

The SNMPv3 settings that can be managed in Bright Cluster Manager are:

Example

```
[bright90]% device use switch01
[bright90->device[switch01]]% snmpsettings
[bright90->device[switch01]->snmpsettings]% show
Parameter                               Value
-----
Authentication key                       < not set >
Authentication protocol                  MD5
Context
Privacy key                              < not set >
Privacy protocol                         DES
Retries                                  -1
Revision
Security level                           Authentication encrypted
Security name
Timeout                                   0s
VLAN Timeout                             0s
version                                   V3
```

The set command can be used, sometimes with tab-completion, to set the SNMP switch parameters. For example, for the SNMPv3 parameters that are set to use cryptographic keys:

Example

```
[bright90->device*[switch01*]->snmpsettings*]% set authenticationprotocol <TAB><TAB>
md5 sha
[bright90->device[switch01]->snmpsettings]% set authenticationprotocol aes
[bright90->device*[switch01*]->snmpsettings*]% set privacyprotocol <TAB><TAB>
aes des
[bright90->device*[switch01*]->snmpsettings*]% set privacyprotocol aes
[bright90->device*[switch01*]->snmpsettings*]% commit
[bright90->device[switch01]->snmpsettings]%
```

SNMP Traps

Bright Cluster Manager can assign an SNMP trap manager role to a node. The snmptrapd daemon is then configured and managed on the assigned node by CMDaemon.

Configuration options include enabling or disabling mailing of the messages, and setting the sender and recipients for the mail. By default an undefined value for Server means that the SNMP server is localhost.

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% roles
[bright90->device[node001]->roles]% assign snmptrap
[bright90->device[node001]->roles*[snmptrap*]]% show
Parameter                               Value
-----
Access                                   public
Add services                             yes
All administrators                       no
Alternative script
Arguments
Event                                     yes
Mail                                      yes
Name                                      snmptrap
```



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

3.8.3 Uplink Ports

Uplink ports are switch ports that are connected to other switches. CMDaemon must be told about any switch ports that are uplink ports, or the traffic passing through an uplink port will lead to mistakes in what CMDaemon knows about port and MAC correspondence. Uplink ports are thus ports that CMDaemon is told to ignore.

To inform CMDaemon about what ports are uplink ports, Bright View or cmsh are used:

- In Bright View, the switch is selected from the **Switches** resource option folder, and the **Edit** option opens up a settings window for that switch (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.

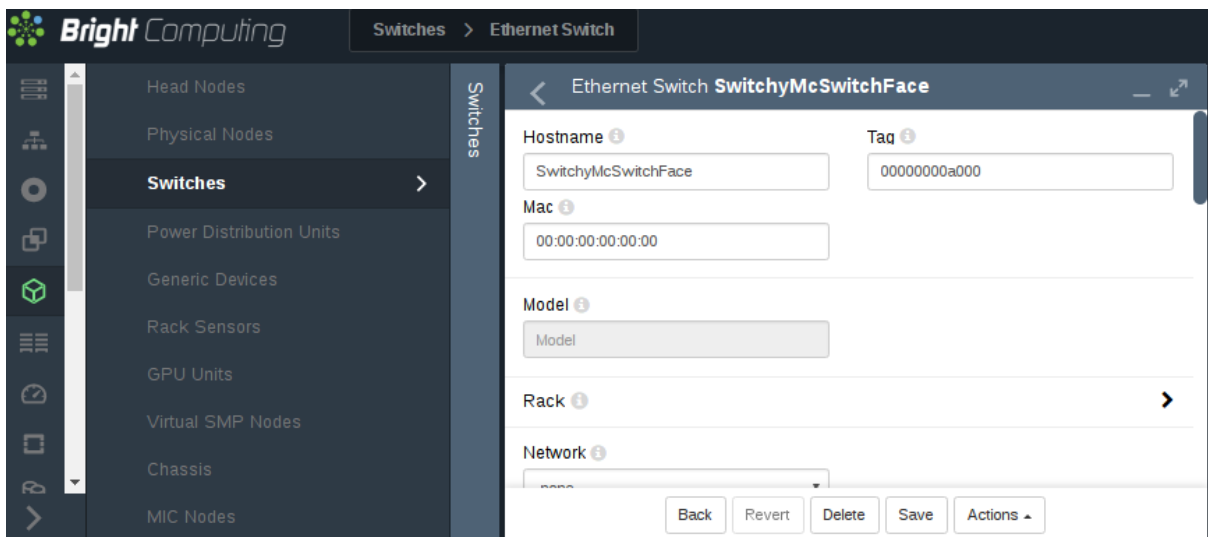


Figure 3.11: Notifying CMDaemon About Uplinks With Bright View

- 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@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% set switch01 uplinks 15 16
[bright90->device*]% set switch02 uplinks 01
[bright90->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@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% showport 00:30:48:30:73:92
switch01:12
```

When running showport, CMDaemon on the head node queries all switches until a match is found.

If a switch is also specified using the “-s” option, then the query is carried out for that switch first. Thus the preceding example can also be specified as:

```
[bright90->device]% showport -s switch01 00:30:49.00:73:92
switch01:12
```

If there is no port number returned for the specified switch, then the scan continues on other switches.

Mapping All Port Connections In The Cluster With showport

A list indicating the port connections and switches for all connected devices that are up can be generated using this script:

Example

```
#!/bin/bash
for nodename in $(cmsh -c "device; foreach * (get hostname)")
do
    macad=$(cmsh -c "device use $nodename; get mac")
    echo -n "$macad $nodename "
    cmsh -c "device showport $macad"
done
```

The script may take a while to finish its run. It gives an output like:

Example

```
00:00:00:00:00:00 switch01: No ethernet switch found connected to this mac address
00:30:49.00:73:92 bright90: 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@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% clear ethernetswitch
[bright90->device*[node001*]]% commit
[bright90->device[node001]]%
```

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

Example

```
[root@bright90 ~]# cmsg
[bright90]% device
[bright90->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):

```
[bright90->device]% switchoverview dell-switch1
Device: dell-switch1
State : [ UP ]
Model : 24G Ethernet Switch

Port Assignment:
```

Port	Status	Assigned	Uplink Speed	Detected
1	UP		false 1 Gb/s	-
2	UP		false 1 Gb/s	-
3	UP		false 1 Gb/s	74:86:7A:AD:3F:2F, node3
4	UP		false 1 Gb/s	74:86:7A:AD:43:E9, node4
...				
11	UP		false 1 Gb/s	74:86:7A:AD:44:D8, node11
12	UP		false 1 Gb/s	74:86:7A:AD:6F:55, node12
...				
23	UP		false 1 Gb/s	74:86:7A:E9:3E:85, node23
24	UP		false 1 Gb/s	74:86:7A:AD:56:DF, node24
49	UP		false 10 Gb/s	74:86:7A:AD:68:FD, node1
50	UP		false 10 Gb/s	74:86:7A:AD:41:A0, node2
53	UP	node34	false 1 Gb/s	5C:F9:DD:F5:79:0D, node34
54	UP	node35	false 1 Gb/s	5C:F9:DD:F5:45:AC, node35
...				
179	UP		false 1 Gb/s	24:B6:FD:F6:20:6F, \ 24:B6:FD:FA:64:2F, 74:86:7A:DF:7E:4C, 90:B1:1C:3F:3D:A9, \ 90:B1:1C:3F:51:D1, D0:67:E5:B7:64:0F, D0:67:E5:B7:61:20
180	UP		false 100 Mb/s	QDR-switch
205	UP		true 10 Gb/s	-
206	DOWN		false 10 Gb/s	-
...				

```
[bright90 ->device]]%
```

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

Configuring the disk layout for head and regular nodes is done as part of the initial setup (section 3.3.16 of the *Installation Manual*). For regular nodes, the disk layout can also be re-configured by 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)
- Hardware RAID setup (Appendix D.4)
- Software RAID setup (Appendix D.5)
- LVM setup (Appendix D.7)
- Diskless setup (Appendix D.9)
- Semi-diskless setup (Appendix D.10)

3.9.2 Disk Layout Assertions

Disk layouts can be set to *assert*

- that particular hardware be used, using XML element tags such as `vendor` or `requiredSize` (Appendix D.11)
- custom assertions using an XML `assert` element tag to run scripts placed in CDATA sections (Appendix D.12)

3.9.3 Changing Disk Layouts

A disk layout applied to a category of nodes is inherited by default by the nodes in that category. A disk layout that is then applied to an individual node within that category overrides the category setting. This is an example of the standard behavior for categories, as mentioned in section 2.1.3.

By default, the cluster is configured with a standard layout specified in section D.3. The layouts can be accessed from Bright View or `cmsh`, as is illustrated by the example in section 3.9.4, which covers changing a node from diskless to diskless mode:

3.9.4 Changing A Disk Layout From Diskless To Diskless

The XML schema for a node configured for diskless operation is shown in Appendix D.9. This can often be deployed as is, or it can be modified during deployment using Bright View or `cmsh` as follows:

Changing A Disk Layout Using Bright View

To change a disk layout with Bright View, the current disk layout is accessed by selecting a node category or a specific node from the resource tree. In the `Settings` pane, the `Disk setup` field can be edited. Clicking on the `Browse` button shows several possible configurations that can be loaded up, and if desired, edited to suit the situation (figure 3.12). To switch from the existing disk layout to a diskless one, 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@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% set disksetup
```

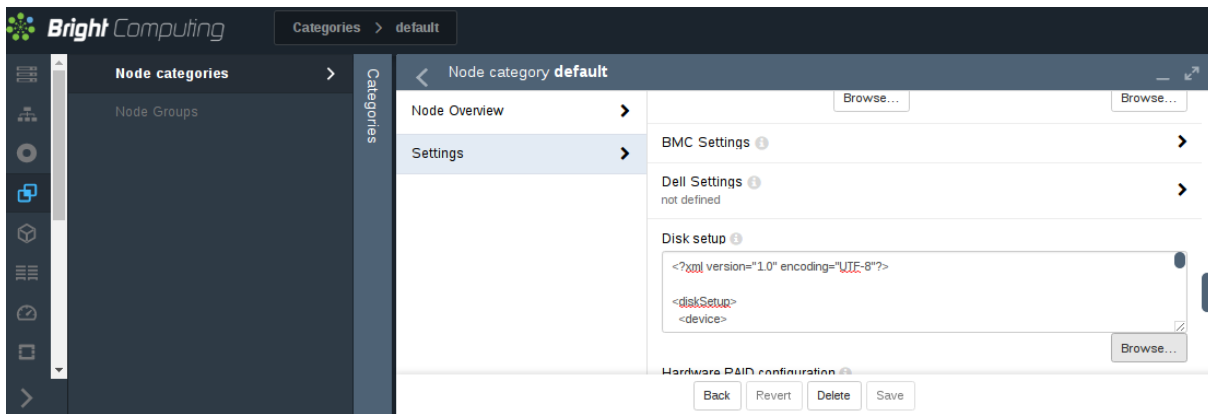


Figure 3.12: Changing A Disked Node To A Diskless Node With Bright View

After editing and saving the XML configuration, the change is then committed to CMDaemon with the `commit` command. It should be understood that a disk layout XML configuration is not stored in a file on the filesystem, but in the CMDaemon database. The XML configurations that exist on a default cluster at

```
/cm/images/default-image/cm/local/apps/cmd/etc/htdocs/disk-setup/
and
```

```
/cm/local/apps/cmd/etc/htdocs/disk-setup/
are merely default configurations.
```

If the `disksetup` setting for a device is deleted, using the `clear` command, then the category level `disksetup` property is used by the device. This is in accordance with the usual behavior for node values that override category values (section 2.1.5).

Instead of editing an existing disk layout, another XML configuration can also be assigned. A diskless configuration may be chosen and set as follows:

Example

```
[bright90->device[node001]]% set disksetup /cm/local/apps/cmd/
etc/htdocs/disk-setup/slave-diskless.xml
```

In these preceding Bright View and `cmsh` examples, after committing the change and rebooting the node, the node then functions entirely from its RAM, without using its own disk.

However, RAM is usually a scarce resource, so administrators often wish to optimize diskless nodes by freeing up the RAM on them from the OS that is using the RAM. Freeing up RAM can be accomplished by providing parts of the filesystem on the diskless node via NFS from the head node. That is, mounting the regular node with filesystems exported via NFS from the head node. The details of how to do this are a part of section 3.10, which covers the configuration of NFS exports and mounts in general.

3.10 Configuring NFS Volume Exports And Mounts

NFS allows unix NFS clients shared access to a filesystem on an NFS server. The accessed filesystem is called an NFS volume by remote machines. The NFS server exports the filesystem to selected hosts or networks, and the clients can then mount the exported volume locally.

An unformatted filesystem cannot be used. The drive must be partitioned beforehand with `fdisk` or similar partitioning tools, and its filesystem formatted with `mkfs` or similar before it can be exported.

In the cluster manager, the head node is typically used to export an NFS volume to the regular nodes, and the regular nodes then mount the volume locally.

- NFS can be made to work at higher speeds with remote direct memory access (RDMA), by bypassing the CPU. If there is RDMA hardware present and if the `rdma-core` package is installed then the RDMA service works automatically in RHEL 7 and 8.

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

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

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

If auto-mounting is used, then the configuration files for exporting should be set up on the NFS server, and the mount configurations set up on the software images. The service “autofs” or the equivalent can be set up using Bright View via the “Services” option (section 3.11) on the head and regular nodes or node categories. With `cmsh` the procedure to configure auto-mounting on the head and regular nodes could be:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use bright90
[bright90->device[bright90]]% services
[bright90->device[bright90]->services]% add autofs
[bright90->device*[bright90*]->services*[autofs*]]% show
Parameter                               Value
-----
Autostart                                no
Belongs to role                           no
Monitored                                 no
Revision
Run if                                    ALWAYS
Service                                   autofs
Sickness check interval                   60
Sickness check script
Sickness check script timeout             10
Timeout                                   -1
[bright90->device*[bright90*]->services*[autofs*]]% set autostart yes
[bright90->device*[bright90*]->services*[autofs*]]% commit
[bright90->device[bright90]->services[autofs]]% category use default
[bright90->category[default]]% services
[bright90->category[default]->services]% add autofs
[bright90->category*[default*]->services*[autofs*]]% set autostart yes
[bright90->category*[default*]->services*[autofs*]]% commit
[bright90->category[default]->services[autofs]]%
```

Filesystems imported to a regular node via an auto-mount operation must explicitly be excluded in `excludelistupdate` by the administrator, as explained in section 5.6.1, page 203.

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

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

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

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

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

3.10.1 Exporting A Filesystem Using Bright View And `cmsh`

Exporting A Filesystem Using Bright View

As an example, if an NFS volume exists at “bright90:/modeldata” it can be exported using Bright View as follows:

The bright90 head node is selected via the clickpath Devices→Head nodes[bright90]→Settings→Filesystem exports. This shows the list of exports (figure 3.13):

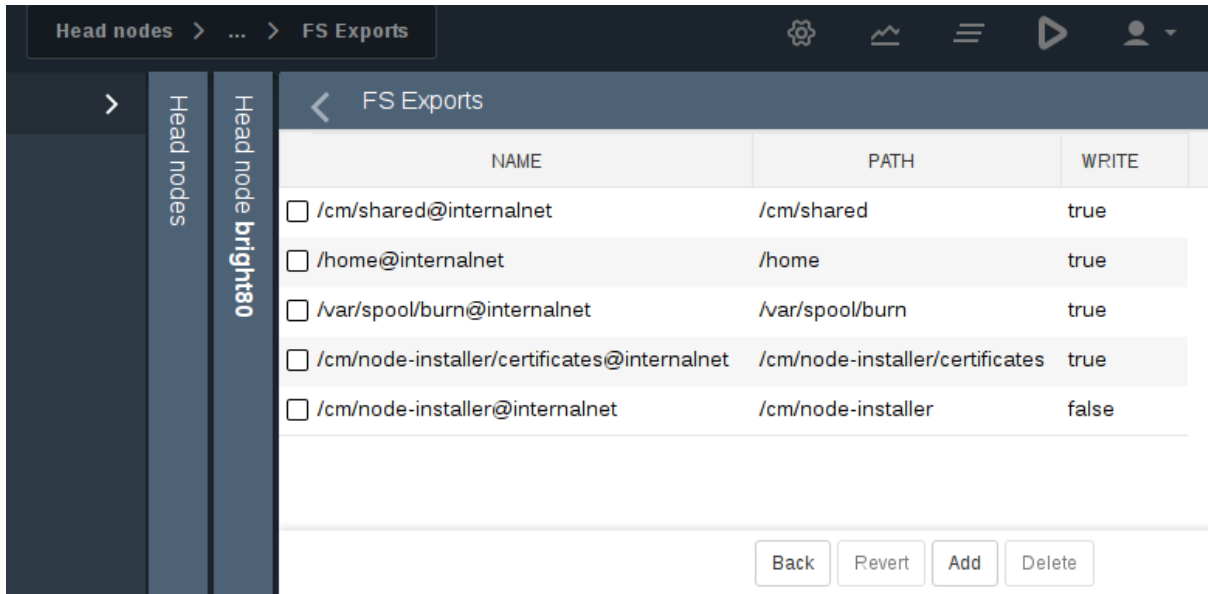


Figure 3.13: NFS Exports From A Head Node Viewed Using Bright View

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

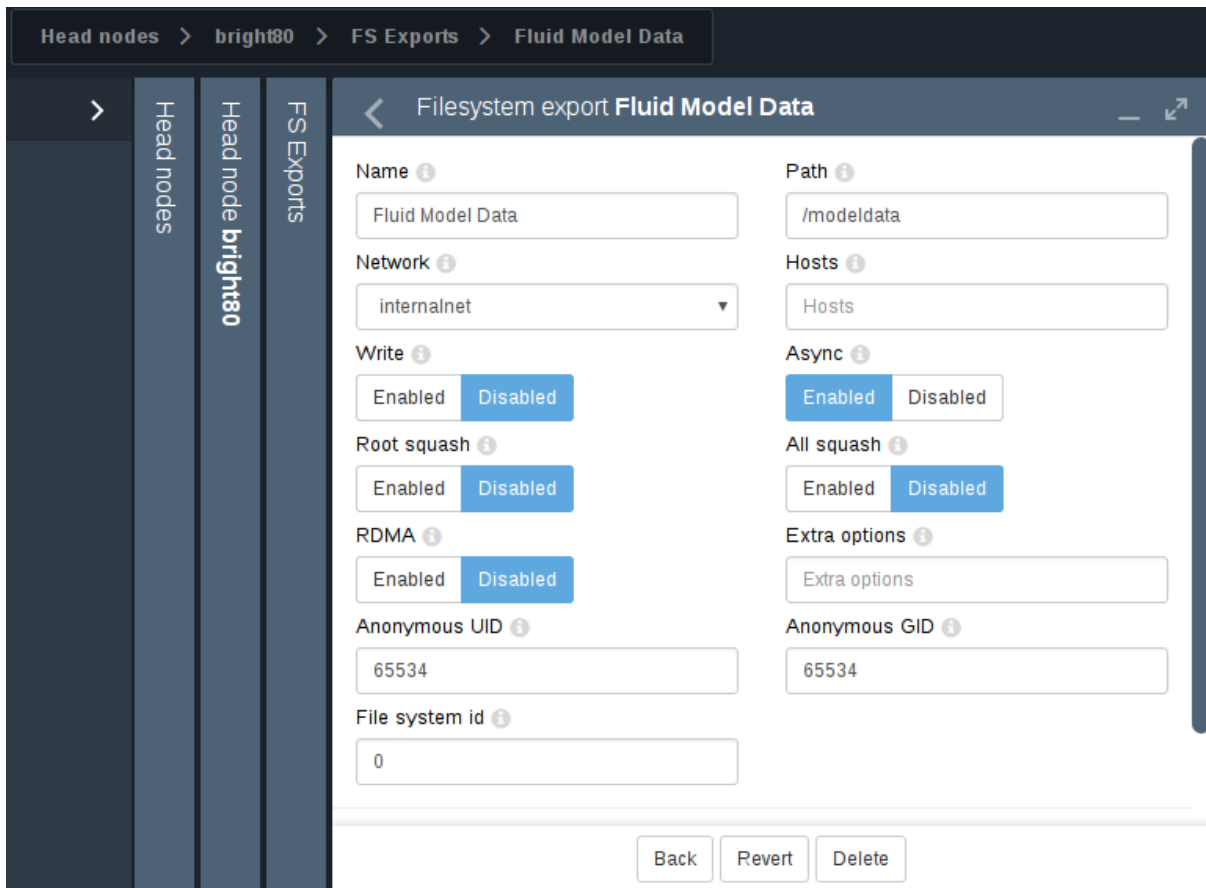


Figure 3.14: Setting Up An NFS Export Using Bright View

For this example, the value for “Name” is set arbitrarily to “Fluid Model Data”, the value for Path is set to /modeldata, and the value for Network is set from the selection menu to allowing access to internalnet (by default 10.141.0.0/16 in CIDR notation).

By setting the Write option to Disabled, read-only access is kept.

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

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

Exporting A Filesystem Using cmsh

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use bright90
[bright90->device[bright90]]% fsexports
[...->fsexports]% add "Fluid Model Data"
[...->fsexports*[Fluid Model Data*]]% set path /modeldata
[...[Fluid Model Data*]]% set hosts 10.141.0.0/16
[...[Fluid Model Data*]]% commit
[...->fsexports[Fluid Model Data]]% list | grep Fluid
Name (key)      Path      Hosts      Write
```



```
-----
Fluid Model Data      /modeldata      10.141.0.0/16      no
```

General Considerations On Exporting A Filesystem

Built-in exports: In versions of Bright Cluster Manager prior to version 9.0, all filesystem exports could be removed from the `fsexports` submode, simply by using the `remove` command with the name of the export.

From version 9.0 onward however, the following filesystem exports:

- `/var/spool/burn`
- `/home`
- `/cm/shared`

are treated as special built-ins.

Head node role and `disableautomaticexports`: Built-ins are exported automatically as part of the headnode role, also introduced in Bright Cluster Manager 9.0, and cannot simply be removed.

To disable export of the built-in file systems, the `disableautomaticexports` command must be run in the headnode role for that node:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use bright90
[bright90->device[bright90]]% roles
[bright90->device[bright90]->roles]% use headnode
[bright90->device[bright90]->roles[headnode]]% show
Parameter                               Value
-----
Name                                     headnode
Revision
Type                                     HeadNodeRole
Add services                             yes
Disable automatic exports                no
Provisioning associations                 <2 internally used>
role use headnode
[bright90->device[bright90]->roles[headnode]]% set disableautomaticexports yes ; commit
```

Disabling exports that are not built-ins: Exports that are not built-ins can still simply be removed. However, also from Bright Cluster Manager 9.0 onwards, they can also simply be disabled in the `fsexports` submode. For example there is an export created by the cluster administrator for `/opt`, it can be disabled as follows:

Example

```
[bright90->device[bright90]->fsexports]% list
Name (key)      Path          Network      Disabled
-----
...
opt             /opt          internalnet  no
[bright90->device[bright90]->fsexports]% set opt disabled yes; commit
```

The reason for automating the export for nodes via a headnode role is that Bright Cluster Manager 9.0 onward has `multidistro` and `multiarch` capabilities (section 11.7), which would make manual management of exports harder for such nodes. The reason for the extra hurdle of `disableautomaticexports` for built-ins is that that disabling these exports can result in an unbootable system.

3.10.2 Mounting A Filesystem Using Bright View 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 Bright View

Thus, in Bright View, values for the remote mount point (`bright90:/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):

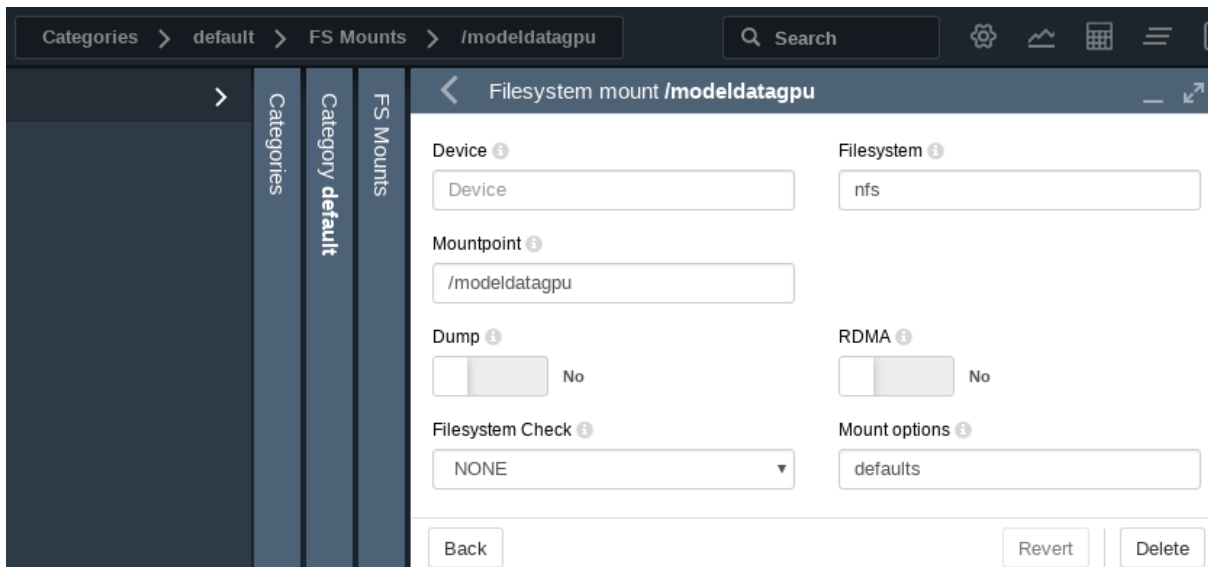


Figure 3.15: Setting Up NFS Mounts On A Node Category Using Bright View

Saving the configuration saves the values, creating the local mount point, and the volume can now be accessed by nodes in that category.

Mounting A Filesystem Using `cmsh`

The equivalent to the preceding Bright View NFS mount procedure can be done in `cmsh` by using the `fsmounts` submode, for example on the `default` category. The `add` method under the `fsmounts` submode sets the mountpoint path, in this case `/modeldatagpu` (some output elided):

Example

```
[root@bright90 ~]# cmsh
[bright90]% category use default
[bright90->category[default]]% fsmounts
[bright90->category[default]->fsmounts]% add /modeldatagpu
[bright90->...[/modeldatagpu*]]% set device bright90:/modeldata
[bright90->...[/modeldatagpu*]]% set filesystem nfs
[bright90->category*[default*]->fsmounts*[/modeldatagpu*]]% commit
[bright90->category[default]->fsmounts[/modeldatagpu]]%
Device           Mountpoint (key)  Filesystem
-----
```

```

...
bright90:/modeldata      /modeldatagpu      nfs
[bright90->category[default]->fsmounts[/modeldatagpu]]% show
Parameter                Value
-----
Device                   bright90:/modeldata
Dump                     no
Filesystem               nfs
Filesystem Check        NONE
Mount options            defaults
Mountpoint               /modeldatagpu

```

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

```
[bright90->...[/modeldatagpu]]% set mountoptions defaults,noac; commit
```

Mounting a CIFS might use:

```
[bright90->...[/modeldatagpu]]% set mountoptions gid,users,file_mode=0666,dir_mode=0777,\
iocharset=iso8859-15,credentials=/path/to/credential
[bright90->...[/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 directive (Appendix C), then CMDaemon changes for `/etc/fstab` are written to the file, but the `mount -a` command is not run by CMDaemon. However, the administrator should be aware that since `mount -a` is run by the distribution during booting, a node reboot implements the mount change.
- While a mount point object may have been removed, `umount` does not take place until reboot, to prevent mount changes outside of the cluster manager. If a `umount` needs to be done without a reboot, then it should be done manually, for example, using the `pdsh` or `pexec` command (section 13.1), to allow the administrator to take appropriate action if unmounting goes wrong.
- When manipulating mount points, the administrator should be aware which mount points are inherited by category, and which are set for the individual node.

- In Bright View, for a node, inheritance by category is indicated in the clickpath Devices→Nodes[*node name*]→Edit→Settings→Filesystem mounts, under the INHERITED column, with the entry (Category).
- In `cmsh`, the category a mount belongs to is displayed in brackets. This is displayed from within the `fsmounts` submode of the device mode for a specified node:

Example

```
[root@bright90 ~]# cmsh -c "device; fsmounts node001; list"
```

Device	Mountpoint (key)	Filesystem
[default] none	/dev/pts	devpts
[default] none	/proc	proc
[default] none	/sys	sysfs
[default] none	/dev/shm	tmpfs
[default] \$localnfs+ /cm/shared		nfs
[default] bright90:/home	/home	nfs
bright90:/cm/shared/exa+	/home/examples	nfs

```
[root@bright90 ~]#
```

To remove a mount point defined at category level for a node, it must be removed from within the category, and not from the specific node.

Mount Order Considerations

Care is sometimes needed in deciding the order in which mounts are carried out.

- For example, if both `/usr/share/doc` and a replacement directory subtree `/usr/share/doc/compat-gcc-34-3.4.6java` are to be used, then the stacking order should be that `/usr/share/doc` is mounted first. This order ensures that the replacement directory subtree overlays the first mount. If, instead, the replacement directory were the first mount, then it would be overlaid, inaccessible, and inactive.
- There may also be dependencies between the subtrees to consider, some of which may prevent the start up of applications and services until they are resolved. In some cases, resolution may be quite involved.

The order in which such mounts are mounted can be modified with the `up` and `down` commands within the `fsmounts` submode of `cmsh`.

3.10.3 Mounting A Filesystem Subtree For A Diskless Node Over NFS

NFS Vs `tmpfs` For Diskless Nodes

For diskless nodes (Appendix D.9), the software image (section 2.1.2) is typically installed from a provisioning node by the node-installer during the provisioning stage, and held as a filesystem in RAM on the diskless node with the `tmpfs` filesystem type.

It can be worthwhile to replace subtrees under the diskless node filesystem held in RAM with subtrees provided over NFS. This can be particularly worthwhile for less frequently accessed parts of the diskless node filesystem. This is because, although providing the files over NFS is much slower than accessing it from RAM, it has the benefit of freeing up RAM for tasks and jobs that run on diskless nodes, thereby increasing the cluster capacity.

An alternative “semi-diskless” way to free up RAM is to use a local disk on the node itself for supplying the subtrees. This is outlined in Appendix D.10.

Moving A Filesystem Subtree Out Of `tmpfs` To NFS

To carry out subtree provisioning over NFS, the subtrees are exported and mounted using the methods outlined in the previous examples in sections 3.10.1 and 3.10.2. For the diskless case, the exported filesystem subtree is thus a particular path under `/cm/images/<image>`² on the provisioning node, and the subtree is mounted accordingly under `/` on the diskless node.

While there are no restrictions placed on the paths that may be mounted in Bright Cluster Manager 9.0, the administrator should be aware that mounting certain paths such as `/bin` is not possible.

When Bright View or `cmsh` are used to manage the NFS export and mount of the subtree filesystem, then `tmpfs` on the diskless node is reduced in size due to the administrator explicitly excluding the subtree from `tmpfs` during provisioning.

An example might be to export `/cm/images/default-image` from the head node, and mount the directory available under it, `usr/share/doc`, at a mount point `/usr/share/doc` on the diskless node. In `cmsh`, such an export can be done by creating an FS export object corresponding to the software image object `defaultimage` with the following indicated properties (some prompt output elided):

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use bright90; fsexports
[bright90->device[bright90]->fsexports]% add defaultimage
[bright90...defaultimage*]]% set path /cm/images/default-image
[bright90...defaultimage*]]% set hosts 10.141.0.0/16
[bright90...defaultimage*]]% commit
[bright90...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
[bright90->category[default]]% fsmounts
[bright90->category[default]->fsmounts]% add /usr/share/doc
[bright90->...*/usr/share/doc*]]% set device bright90:/cm/images/default-image/user/share/doc
[bright90->...*/usr/share/doc*]]% set filesystem nfs
[bright90->category*[default*]->fsmounts*[/usr/share/doc*]]% commit
[bright90->category[default]->fsmounts[/usr/share/doc]]% list
Device                Mountpoint (key)        Filesystem
-----
...
bright90:/cm/images/usr/share/doc /usr/share/doc         nfs
[bright90->category[default]->fsmounts[/usr/share/doc]]% show
Parameter            Value
-----
Device                bright90:/cm/images/default-image/user/share/doc
Dump                  no
Filesystem            nfs
Filesystem Check      0
Mount options         defaults
Mountpoint            /usr/share/doc
```

²by default `<image>` is `default-image` on a newly-installed cluster

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 106 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@bright90 ~]# cd /cm/images/default-image/
[root@bright90 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@bright90 default-image]# cmsh
[bright90]% category use default; fsmounts
[bright90->category[default]->fsmounts]% list -f device:53,mountpoint:17
device                                     mountpoint (key)
-----
...
master:/cm/shared                         /cm/shared
master:/home                              /home
bright90:/cm/images/default-image/usr/share/locale /usr/share/locale
bright90:/cm/images/default-image/usr/java      /usr/java
bright90:/cm/images/default-image/usr/share/doc /usr/share/doc
bright90:/cm/images/default-image/usr/src      /usr/src
[bright90->category[default]->fsmounts]%
```

Diskless nodes that have NFS subtree configuration carried out on them can be rebooted to start them up with the new configuration.

3.10.4 Mounting The Root Filesystem For A Diskless Node Over NFS

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

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

If configuring a node category, it can be configured as follows on a CentOS 6 or 7 system:

1. The full diskless disk setup is partitioned first:

```
[root@bright90 ~]# cmsh
[bright90]% category use default
[bright90->category[default]]% set disksetup /cm/images/default-image/cm/local/apps/cmd/\
etc/htdocs/disk-setup/slave-diskless.xml
[bright90->category*[default*]]% commit #puts full OS in RAM
```

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

```
[bright90->category[default]]% device use master; fsexports
[bright90->device[bright90]->fsexports]% add /cm/images/default-image
...*]->fsexports*[/cm/images/default-image*]]% set hosts internalnet
```

```
...*]->fsexports*[/cm/images/default-image*]]% set name default@internal
...*]->fsexports*[default@internal*]]% commit #exports default image
[bright90->device[bright90]->fsexports[default@internal]]% quit
```

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

```
[root@bright90 ~]# cmsg -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@bright90 ~]# cat mountfiles #for centos 7
/var/cache
/var/log
/var/tmp
/var/spool
/tmp
/dev
/cm/local/apps/cmd/etc
/cm/local/apps/openldap/etc
/cm/local/apps/openpbs/var
/cm/local/apps/sge/var
/cm/local/apps/slurm/var
/cm/local/apps/pbspro/var
/cm/local/apps/pbspro-ce/var
/etc
/run
/var/lib #required by postfix and rpc-statd
```

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

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

```
[root@bright90 ~]# (echo "category use default; fsmounts"
for i in $(<mountfiles)
do
echo "add $i; set device tmpfs; set filesystem tmpfs; exit"
done
echo "commit" ) | cmsg
```

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

The diskless nodes cannot be powered off with a simple `poweroff` or rebooted with a `reboot`. This is because the parts of the filesystem required to carry out these actions are unmounted before they are

called in the diskless configuration. The `-f|--force` option to these commands forces a `poweroff` or `reboot`, but should only be used after first cleanly unmounting any writable shared filesystems, such as `/cm/shared` and `/home`. This is because the forcing options interrupt I/O syncing when invoked, which can result in a corrupted filesystem.

High Availability (Chapter 14) configured on the head nodes, with diskless nodes using a root filesystem mounted over NFS, would not work in a simple configuration such as this because it will result in stale NFS file handles over all the diskless nodes during a failover.

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

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

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

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

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

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

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

Exporting With Bright View And `cmsh` Using NFS Over RDMA

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

The procedure using Bright View is much the same as done in section 3.10.1 (“Exporting A Filesystem Using Bright View”), except for that the `ibnet` network should be selected instead of the default `internalnet`, and the “RDMA” option should be enabled.

The procedure using `cmsh` is much the same as done in section 3.10.1 (“Exporting A Filesystem Using `cmsh`”), except that the `ibnet` network (normally with a recommended value of `10.149.0.0/16`) should be set, and the `rdma` option should be set.

Example

(based on the example in section 3.10.1)

```
...
[...->fsexports*[Fluid Model Data*]]% set path /modeldata
[...[Fluid Model Data*]]% set hosts ibnet
[...[Fluid Model Data*]]% set rdma yes
[...[Fluid Model Data*]]% commit
...
```

Mounting With Bright View And `cmsh` Using NFS Over RDMA

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

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

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

Example

(based on the example in section 3.10.2)

```
...
[bright90->category[default]->fsmounts]% add /modeldatagpu
[bright90->...*/modeldatagpu*]]% set device bright90.ib.cluster:/modeldata
[bright90->...*/modeldatagpu*]]% set filesystem nfs
[bright90->...*/modeldatagpu*]]% set rdma yes
[bright90->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 `mysqld`, `mariabd`, `nfs`, `postfix` and so on.

Already installed services can also be brought under Bright Cluster Manager control, and started and stopped with Bright View and `cmsh` tools.

An additional convenience that comes with the cluster manager tools is that some `CMDaemon` parameters useful for managing services in a cluster are very easily configured, whether on the head node, a regular node, or for a node category. These parameters are:

- **monitored:** checks periodically if a service is running. Information is displayed and logged the first time it starts or the first time it dies
- **autostart:** restarts a failed service that is being monitored.
 - If `autostart` is set to `on`, and a service is stopped using 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 68.
- **runif:** (only honored for nodes used as part of a high availability configuration (chapter 14)) whether the service should run with a state of:
 - `active`: run on the active node only
 - `passive`: run on the passive only
 - `always`: run both on the active and passive
 - `preferpassive`: preferentially run on the passive if it is available. Valid only for head nodes. Invalid for failover groups (section 14.5.3).

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 sample can be found within the file `/usr/share/doc/ini-scripts-<version>/sysvinitfiles` in RHEL7 and derivatives.
- The operating system configuration of the service itself, including its persistence on reboot, remains under control of the operating system, and is not handled by CMDaemon. So, stopping a service within CMDaemon means that by default the service will still start up on reboot. Running `chkconfig` from the command line can be used to configure the service to no longer start up on reboot.

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 14.4.6). 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@bright90 ~]# chkconfig cups on
... symlinks created...
[root@bright90 ~]# service cups start
Redirecting to /bin/systemctl start cups.service
```

Managing The Service From `cmsh`

Starting the service in `cmsh`: The following session illustrates adding the CUPS service from within device mode and the `services` submodule. 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):

```
[bright90]% device services node001
[bright90->device[node001]->services]% add cups
[bright90->device*[node001*]->services*[cups*]]% show
Parameter                               Value
-----
Autostart                               no
Belongs to role                          no
Monitored                                no
...
Run if                                  ALWAYS
Service                                  cups
...
[bright90->device*[node001*]->services*[cups*]]% set monitored on
[bright90->device*[node001*]->services*[cups*]]% set autostart on
[bright90->device*[node001*]->services*[cups*]]% commit
[bright90->device[node001]->services[cups]]%
Apr 14 14:02:16 2017 [notice] node001: Service cups was started
[bright90->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:

```
[bright90->device[node001]->services[cups]]% stop
Fri Apr 14 14:03:40 2017 [notice] node001: Service cups was stopped
Successfully stopped service cups on: node001
[bright90->device[node001]->services[cups]]%
```

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

```
[bright90->device[node001]->services[cups]]% status
cups                [STOPPED]
```

Details on how a state is used when monitoring a service are given in the section “Monitoring A Service With `cmsh` And Bright View” on page 114.

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

```
[bright90->device[node001]->services[cups]]% category
[bright90->category]% services default
[bright90->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

```
[bright90->category*[default*]->services*[cups*]]% show
...
[bright90->category*[default*]->services*[cups*]]% set autostart yes
[bright90->category*[default*]->services*[cups*]]% set monitored yes
[bright90->category*[default*]->services*[cups*]]% commit
[bright90->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
[bright90->category[default]->services[cups]]% status
node001..... cups                [STOPPED ]
node002..... cups                [  UP   ]
node003..... cups                [  UP   ]
node004..... cups                [  UP   ]
node005..... cups                [  UP   ]
```

Managing The Service From Bright View

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

- Head Nodes, for example via a clickpath of
Devices → Head Nodes [bright90] → Settings → Services
- Nodes, for example via a clickpath of
Devices → Nodes [node001] → Settings → Services

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

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

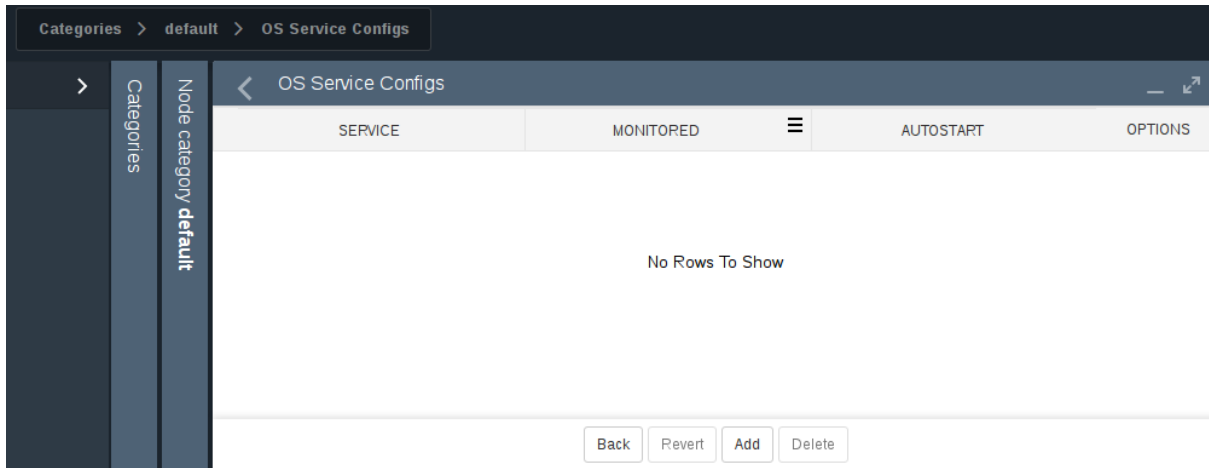


Figure 3.16: Operating System Services Subwindow In Bright View

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

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

Figure 3.17 shows CUPS being set up from an Add dialog in the services subwindow. The subwindow is accessible via the [Add] button of figure 3.16.

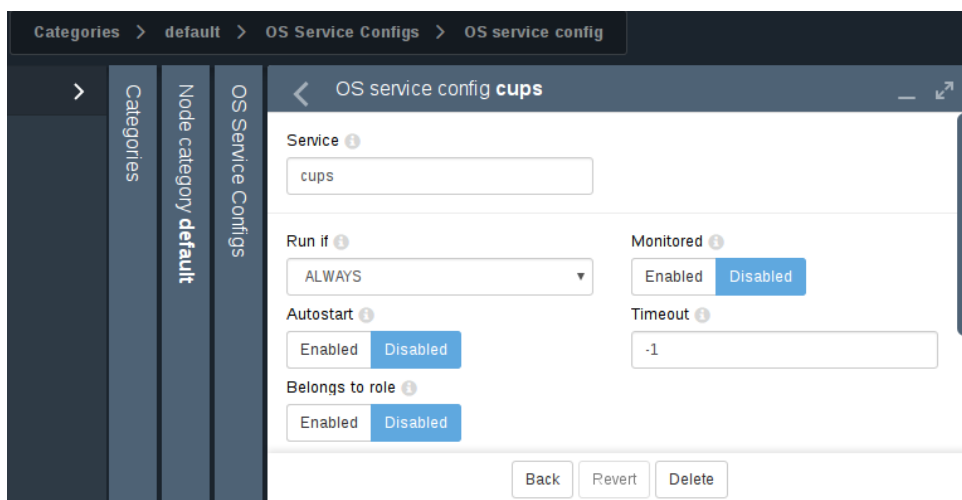


Figure 3.17: Setting Up A Service Using Bright View

For a service in the services subwindow, clicking on the Status button in figure 3.16 displays a grid of the state of services on a running node as either Up or Down.

Monitoring A Service With `cmsh` And Bright View

The service is in a DOWN state if it is not running, and in a FAILING state if it is unable to run after 10 auto-starts in a row. Event messages are sent during these first 10 auto-starts. After the first 10 auto-starts, no

more event messages are sent, but autostart attempts continue.

In case an autostart attempt has not yet restarted the service, the `reset` option may be used to attempt an immediate restart. The `reset` option is not a service option in the regular shell, but is used by `CMDaemon` (within `cmsh` and Bright View) to clear a `FAILING` state of a service, reset the attempted auto-starts count to zero, and attempt a restart of the service.

The monitoring system sets the `ManagedServicesOk` health check (Appendix G.2.1) to a state of `FAIL` if any of the services it monitors is in the `FAILING` state. In `cmsh`, the statuses of the services are listed by running the `latesthealthdata` command (section 12.6.3) from device mode.

Standard `init.d` script behavior is that the script return a non-zero exit code if the service is down, and a zero exit code if the service is up. A non-zero exit code makes 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:

```
[bright90->category[default]->services]% add cups
[bright90->category*[default*]->services*[cups*]]% set monitored on
[bright90->category*[default*]->services*[cups*]]% set autostart on
[bright90->category*[default*]->services*[cups*]]% commit; exit
[bright90->category[default]->services]% remove cups; commit
Wed May 23 12:53:58 2012 [notice] node001: Service cups was stopped
```

In the preceding `cmsh` session, `cups` starts up when the `autostart` parameter is committed, if `cups` is not already up.

The behavior of having the service stop on removal is implemented because it is usually what is wanted.

However, sometimes the administrator would like to remove the service from `CMDaemon` control without it shutting down. To do this, `autostart` must be set to `off` first.

```
[bright90->category[default]->services]% add cups
[bright90->category*[default*]->services*[cups*]]% set monitored on
[bright90->category*[default*]->services*[cups*]]% set autostart off
[bright90->category*[default*]->services*[cups*]]% commit; exit
Wed May 23 12:54:40 2012 [notice] node001: Service cups was started
[bright90->category[default]->services]% remove cups; commit
[bright90->category[default]->services]% !# no change: cups stays up
```

3.12 Managing And Configuring A Rack

3.12.1 Racks

A cluster may have local nodes grouped physically into racks. A rack is 42 units in height by default, and nodes normally take up one unit.

Racks Overview

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

Within the Racks pane:

- a new rack item can be added with the Add button.

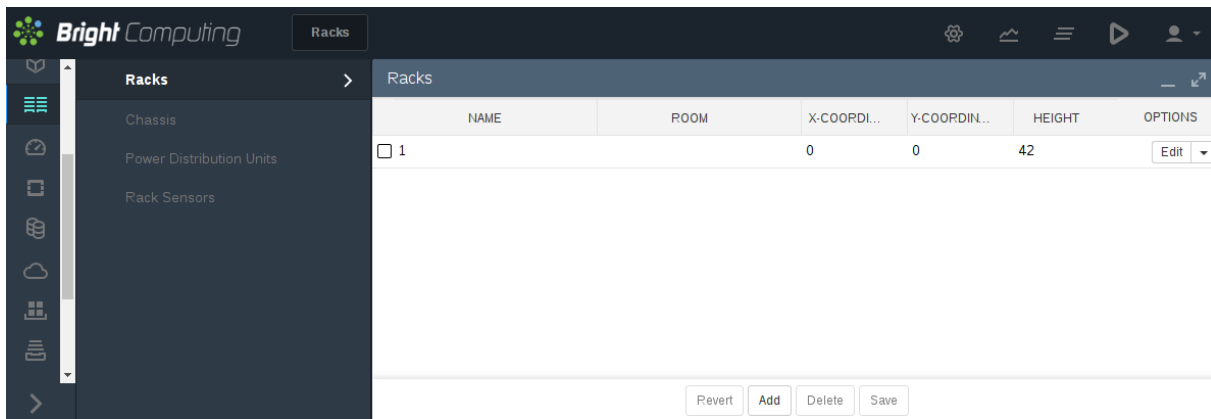


Figure 3.18: Racks Overview Pane Using Bright View

- an existing rack item can be edited with the Edit button, or by double-clicking on the item itself in the pane.

These actions bring up the rack configuration pane (figure 3.19).

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

```
[bright90->rack]% list
Name (key)      Room          x-Coordinate  y-Coordinate  Height
-----
rack2           skonk works   2             0             42
racknroll      1             0             0             42
```

Rack Configuration Settings

Rack configuration settings in Bright View: The Settings pane for editing a rack item selected from the Racks pane is shown in figure 3.19.

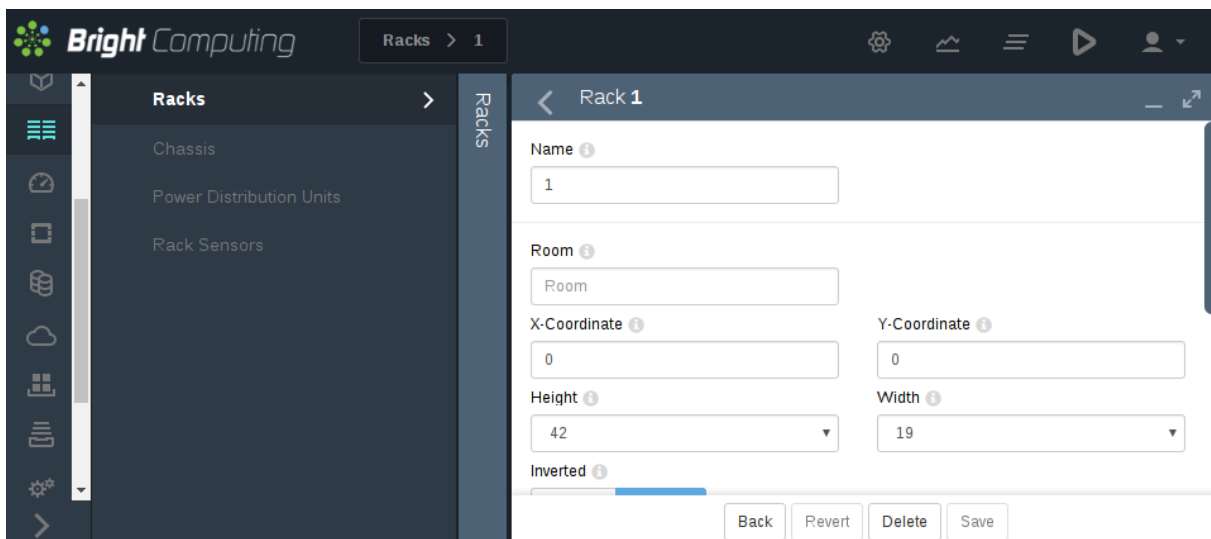


Figure 3.19: Rack Configuration Settings Using Bright View

The rack item configuration settings are:

- Name: A unique name for the rack item. Names such as rack001, rack002 are a sensible choice

- **Room:** A unique name for the room the rack is in.
- **Position:** The x - and y -coordinates of the rack in a room. These coordinates are meant to be a hint for the administrator about the positioning of the racks in the room, and as such are optional, and can be arbitrary. The **Notes** field can be used as a supplement or as an alternative for hints.
- **Height:** by default this is the standard rack size of 42U.
- **Bottom of rack is position 1:** Normally, a rack uses the number 1 to mark the top and 42 to mark the bottom position for the places that a device can be positioned in a rack. However, some manufacturers use 1 to mark the bottom instead. Ticking the checkbox records the numbering layout accordingly for all racks, if the checkboxed rack is the first rack seen in Rackview.

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

Example

```
[bright90->rack]% set rack
rack1 rack2 rack3
[bright90->rack]% set rack2
height      name      revision  width      y-coordinate
inverted    notes     room      x-coordinate
```

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

Setting the values can be done as in this example:

Example

```
[bright90->rack]% use rack2
[bright90->rack[rack2]]% set room "skonk works"
[bright90->rack*[rack2*]]% set x-coordinate 2
[bright90->rack*[rack2*]]% set y-coordinate 0
[bright90->rack*[rack2*]]% set inverted no
[bright90->rack*[rack2*]]% commit
[bright90->rack[rack2]]%
```

3.12.2 Assigning Devices To A Rack

Devices such as nodes, switches, and chassis, can be assigned to racks.

By default, no such devices are assigned to a rack.

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

Assigning Devices To A Rack Using Bright View

Using Bright View, a device such as a node node001 can be assigned to a rack via the clickpath `Devices→Nodes[node001]→Settings→Rack` (figure 3.20):

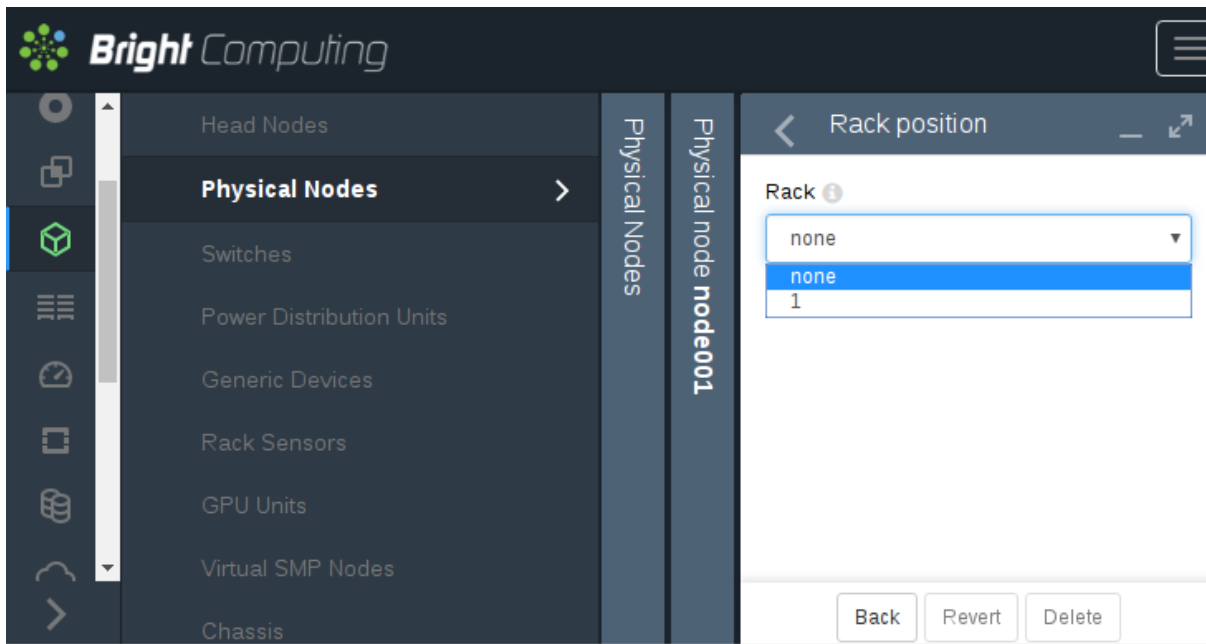


Figure 3.20: Rack Assignment Using Bright View

Assigning Devices To A Rack Using `cmsh`

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

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

can conveniently be done as follows:

```
root@bright90:~# for i in 1..3 ; do cmsh -c "device use node00$i; get hostname; set rack rack2; \
get rack; set deviceposition $i; get deviceposition; commit"; done
```

The Convention Of The Top Of The Device Being Its Position

Since rack manufacturers usually number their racks from top to bottom, the position of a device in a rack (using the parameter `Position` in Bright View, and the parameter `deviceposition` in `cmsh`) is always taken to be where the top of the device is located. This is the convention followed even for the less usual case where the rack numbering is from bottom to top.

A position on a rack is 1U of space. Most devices have a height that fits in that 1U, so that the top of the device is located at the same position as the bottom of the device, and no confusion is possible. The administrator should however be aware that for any device that is greater than 1U in height such as, for example, a blade enclosure chassis (section 3.12.3), the convention means that it is the position of the top of the device that is where the device is considered to be. The position of the bottom of the device is ignored.

3.12.3 Assigning Devices To A Chassis

A Chassis As A Physical Part Of A Cluster

In a cluster, several local nodes may be grouped together physically into a chassis. This is common for clusters using blade systems. Clusters made up of blade systems use less space, less hardware, and less electrical power than non-blade clusters with the same computing power. In blade systems, the blades are the nodes, and the chassis is the blade enclosure.

A blade enclosure chassis is typically 6 to 10U in size, and the node density for server blades is typically 2 blades per unit with 2014 technology.

Chassis Configuration And Node Assignment

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

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

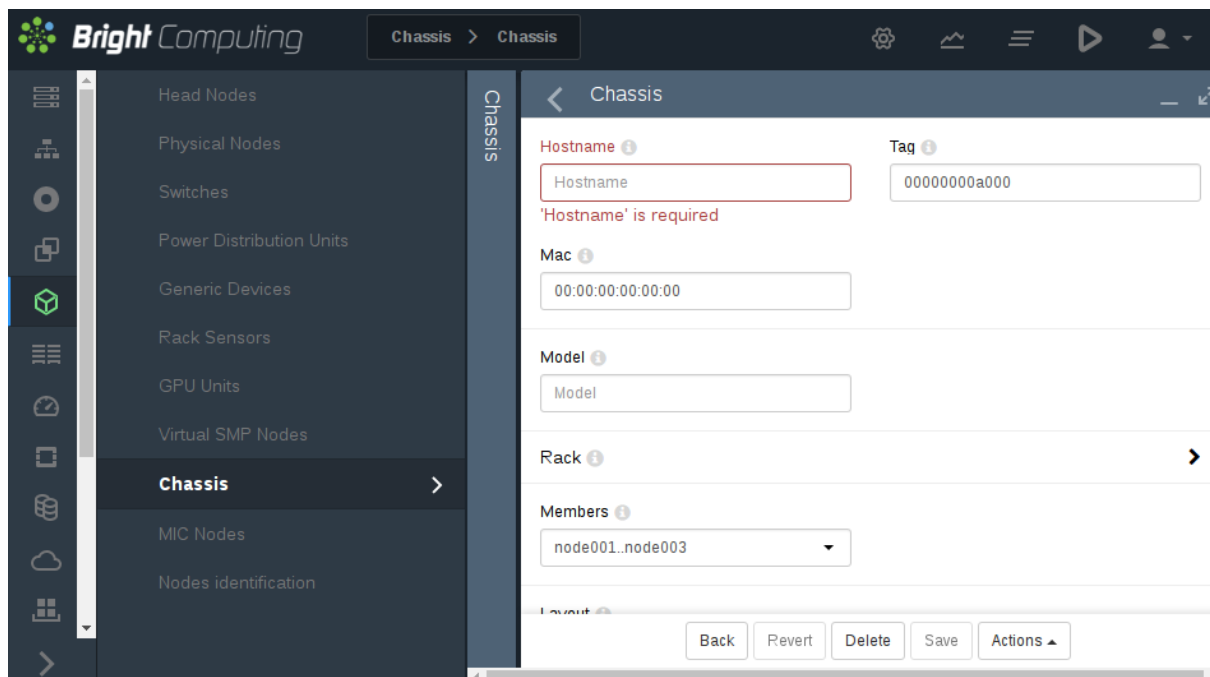


Figure 3.21: Bright View Chassis Subpane

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

- **Hostname:** a name that can be assigned to the chassis operating system
- **Mac:** the MAC address of the chassis
- **Rack:** the rack in which the chassis is placed
- **Members:** the Members menu option allows devices to be assigned to a chassis (figure 3.22). An item within the Device set can be any item from the subsets of Node, Switch, Power Distribution Unit, Generic Device, Rack Sensor, and Gpu Unit. These items can be filtered for viewing, depending on whether they are Assigned (members of the chassis), Not assigned (not members of the chassis), or they can All be viewed (both Assigned and Not assigned items).
- **Layout:** how the nodes in a chassis are laid out visually.
- **Network:** which network the chassis is attached to.
- **Username, Password:** the user name and password to access the chassis operating system
- **Power control, Custom power script, Custom power script argument:** power-related items for the chassis.
- **Userdefined1, Userdefined2:** administrator-defined variables that can be used by CMDaemon.

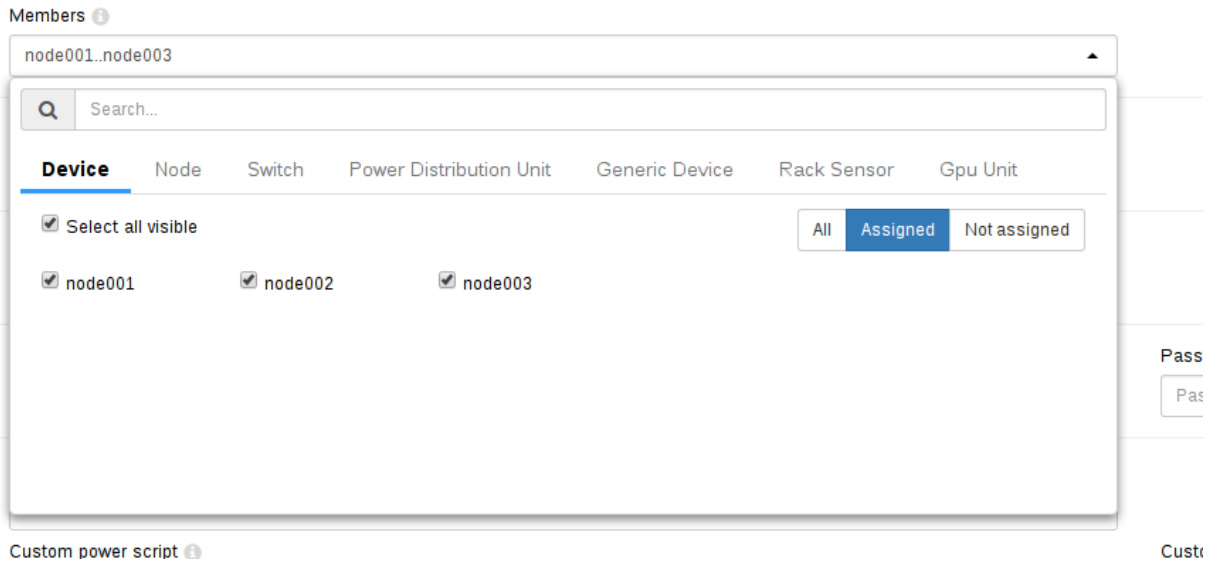


Figure 3.22: Bright View Chassis Members Menu Options

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

```
[bright90->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

```
[bright90->device[chassis1]]% set rack rack2
[bright90->device*[chassis1*]]% set deviceposition 1; set deviceheight 6
[bright90->device*[chassis1*]]% commit
```

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

Example

```
[bright90->device[chassis1]]% append members bright90 node001..node005
[bright90->device*[chassis1*]]% commit
```

3.13 Configuring GPU Settings

3.13.1 GPUs And GPU Units

GPUs (Graphics Processing Units) are processors that are heavily optimized for executing certain types of parallel processing tasks. GPUs were originally used for rendering graphics, and one GPU typically has hundreds of cores. When used for general processing, they are sometimes called General Processing GPUs, or GPGPUs. For convenience, the “GP” prefix for General Processing is not used in this manual.

A GPU is typically placed on a PCIe card. GPUs can be physically inside the node that uses them, or they can be physically external to the node that uses them. As far as the operating system on the node making use of the physically external GPUs is concerned, the GPUs are internal to the node.

If the GPUs are physically external to the node, then they are typically in a *GPU unit*. A GPU unit is a chassis that hosts only GPUs. It is typically able to provide GPU access to several nodes, usually via PCIe extender connections. This ability means that external GPUs typically require more configuration than internal GPUs. GPU units are not covered in this manual because they are not very popular, but there is an example configuration given at <https://community.brightcomputing.com/question/5e16000ef396761939c7ee42/answer> for the Dell PowerEdge C410x.

Configuring GPU settings for GPUs—that is, for devices internal to a node—is covered next.

3.13.2 Configuring GPU Settings

The `gpusettings` Submode In `cmsh`

In `cmsh`, GPUs can be configured for a specified node via `device` mode.

Going into the `gpusettings` submode for that node then allows a type of GPU to be set, from the `amd` or `nvidia` types, and a range to be specified for the GPU slots:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% gpusettings
[bright90->device[node001]->gpusettings]% add nvidia 1-3 ; commit
```

The range can be specified as

- a single number, for example: 3
- a range, for example: 0-2
- all, using:
 - all
 - or
 - *

GPUs can also be configured for a specified category via `category` mode. For example, using the `category default`, then entering into the `gpusettings` submode allows a type (`nvidia` or `amd`) and a range to be set for the range of GPUs:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category use default
[bright90->category[default]]% gpusettings
[bright90->category[default]->gpusettings]% list
GPU range (key)  Power limit  ECC mode      Compute mode  Clock speeds
-----
[bright90->category[default]->gpusettings]% add nvidia 1-3 ; commit
```

```
[bright90->category[default]->gpusettings[nvidia:1-3]]% show
```

```
Parameter          Value
```

```
-----
Clock speeds
Clock sync boost mode
Compute mode
ECC mode
...
```

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

GPU Settings With NVIDIA GPUs

The installation of the NVIDIA GPU software driver packages is covered in section 7.4.1 of the *Installation Manual*. It should be noted that the `cuda-dcgm` package must be installed to access NVIDIA GPU metrics.

The present section is about configuring NVIDIA GPUs in Bright Cluster Manager. The driver itself does not necessarily have to be in place for the configuration to be done, although the configuration only becomes active when the driver is installed.

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

- `clockspeeds`: The pair of clock speeds (frequency in MHz) to be set for this parameter can be selected from the list of available speeds. The available speeds can be seen by running the `status` command. The values are specified in the form: `<number for GPU processor>,<number for memory>`
- `clocksyncboostmode`: GPU boosting. Exceed the maximum core and memory clock speeds if it is safe. Choices are:
 - `enabled`
 - `disabled`
- `computemode`: Contexts can be computed with the following values for this mode:
 - `Default`: Multiple contexts are allowed
 - `Exclusive thread`: Only one context is allowed per device, usable from one thread at a time
 - `Exclusive process`: Only one context is allowed per device, usable from multiple threads at a time. This mode option is valid for CUDA 4.0 and higher. Earlier CUDA versions ran only in this mode.
 - `Prohibited`: No contexts are allowed per device
- `eccmode`: Sets the ECC bit error check, with:
 - `enabled`
 - `disabled`

When ECC is enabled:

- Single bit errors are detected, using the `EccSBitGPU` metric (page 769), and corrected automatically.
 - Double bit errors are also detected, using the `EccDBitGPU` metric (page 768), but cannot be corrected.
- GPU range: range values can be set as follows:
 - `all`: The GPU settings apply to all GPUs on the node.

- *<number>*: The GPU settings apply to an individual GPU, for example: 1
- *<number range>*: The GPU settings apply to a range of GPUs, for example: 1, 3-5
- **powerlimit**: The administrator-defined upper power limit for the GPU. Only valid if powermode is Supported.
 - **min**: The minimum upper power limit that the hardware supports.
 - **max**: The maximum upper power limit that the hardware supports.
 - *<number>*: An arbitrary upper power limit, specified as a number between min and max
 - **default**: Upper power limit based on the default hardware value.

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

GPU Settings With AMD GPUs

GPU settings for AMD Radeon GPUs are accessed via `cmsh` in the same way as NVIDIA GPU settings. The AMD GPU setting parameters do differ from the NVIDIA ones.

The AMD GPUs supported are Radeon cards. A list of cards and operating systems compatible with the Linux driver used is at <https://support.amd.com/en-us/kb-articles/Pages/Radeon-Software-for-Linux-Release-Notes.aspx>

AMD GPU driver installation is described in section 7.5 of the *Installation Manual*.

The Radeon Instinct MI25 shows the following settings in Ubuntu 16_06 running a Linux 4.4.0-72-generic kernel:

Example

```
[bright90->device[node001]->gpusettings]% list
Type GPU range Info
-----
AMD 0          PowerPlay: manual
[bright90->device[node001]->gpusettings]% use amd:0
[bright90->device[node001]->gpusettings[amd:0]]% show
Parameter                               Value
-----
Activity threshold                       1
Fan speed                                255
GPU clock level                           5
GPU range                                  0
Hysteresis down                           0
Hysteresis up                             0
Info                                       PowerPlay: manual
Memory clock level                        3
Minimum GPU clock                          0
Minimum memory clock                       0
Overdrive percentage                       1
PowerPlay mode                             manual
Revision
Type                                       AMD
```

The possible values here are:

- **activitythreshold**: Percent GPU usage at a clock level that is required before clock levels change. From 0 to 100.
- **fanspeed**: Maximum fan speed. From 0 to 255

- `gpuclocklevel`: GPU clock level setting. From 0 to 7.
- `gprange`: The slots used.
- `hysteresisdown`: Delay in milliseconds before a clock level decrease is carried out.
- `hysteresisup`: Delay in milliseconds before a clock level increase is carried out.
- `info`: A compact informative line about the GPU status.
- `memoryclocklevel`: Memory clock speed setting. From 0-3. Other cards can show other values.
- `minimumgpuclock`: Minimum clock frequency for GPU, in MHz. The kernel only allows certain values. Supported values can be seen using the status command.
- `minimummemoryclock`: Minimum clock frequency for the memory, in MHz. The kernel only allows certain values. Supported values can be seen using the status command.
- `overdrivepercentage`: Percent overclocking. From 0 to 20%
- `powerplaymode`: Decides how the performance level power setting should be implemented.
 - `high`: keep performance high, regardless of GPU workload
 - `low`: keep performance low, regardless of GPU workload
 - `auto`: Switch clock rates according to GPU workload
 - `manual`: Use the memory clock level and GPU clock values.

The status command displays supported clock frequencies (some values ellipsized):

Example

```
[bright90->device[node001]->gpusettings[amd:0]]% status
Index Name                Property  Value      Supported
-----
0    Radeon Instinct MI25  Clock    1399Mhz    852Mhz, 991Mhz, ..., 1440Mhz, 1515Mhz
0    Radeon Instinct MI25  Memory   945Mhz     167Mhz, 500Mhz, 800Mhz, 945Mhz
```

The `gpusettings` Submode In Bright View

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

```
Devices->Nodes[node001]->Edit->Settings->GPU Settings
```

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

GPU Configuration For HPC Workload Managers

Slurm: To configure NVIDIA GPUs for Slurm, changes are made in `slurm.conf` when `cm-wlm-setup` configures GPUs in Slurm.

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

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

Example

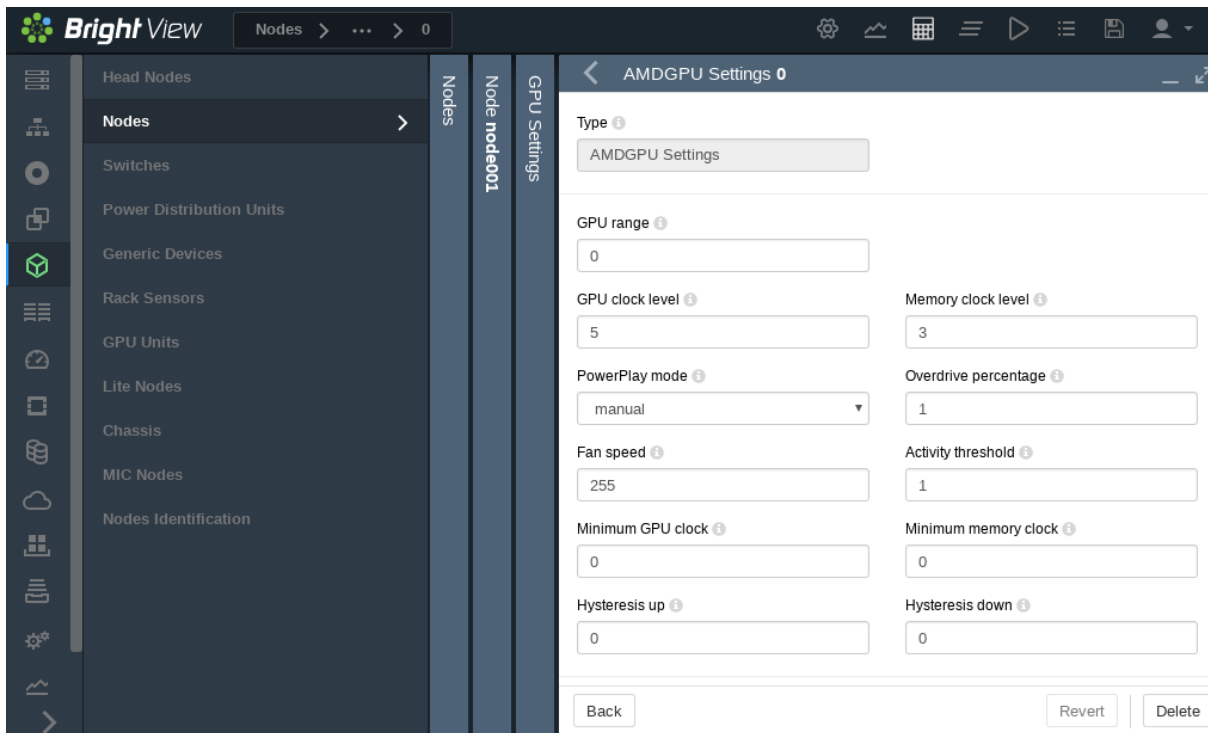


Figure 3.23: GPU Settings Subpane For A Node

```
[fred@bright90 ~]$ sbatch --gres=help
Valid gres options are:
gpu[:type]:count
mps[:type]:count
```

This means that a GPU can be requested in a job script with the Slurm gres option:

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

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

```
#SBATCH --gres=mps:100
```

If adding new parameters manually, care must be taken to avoid duplication of parameters already in the file, because `slurmd` is unlikely to work properly with duplicated parameters.

The Slurm client role can be configured at configuration overlay, category, or node level. If configuring the Slurm client role for GPU gres resources manually, then each GPU can be configured within the role:

Example

```
[bright90->category[dgx]->roles[slurmclient]->genericresources]% add gpu0
[bright90->category*[dgx*]->roles*[slurmclient*]->genericresources*[gpu0*]]% set name gpu
[bright90->category*[dgx*]->roles*[slurmclient*]->genericresources*[gpu0*]]% set file /dev/nvidia0
[bright90->category*[dgx*]->roles*[slurmclient*]->genericresources*[gpu0*]]% commit
[bright90->category[dgx]->roles[slurmclient]->genericresources[gpu0]]%
  (Repeat similar settings for the other GPUs, gpu1..gpu7)
[bright90->category[dgx]->roles[slurmclient]->genericresources]% list
Alias (key) Name      Type      Count      File
-----
gpu0          gpu              /dev/nvidia0
```

```

gpu1      gpu          /dev/nvidia1
gpu2      gpu          /dev/nvidia2
gpu3      gpu          /dev/nvidia3
gpu4      gpu          /dev/nvidia4
gpu5      gpu          /dev/nvidia5
gpu6      gpu          /dev/nvidia6
gpu7      gpu          /dev/nvidia7
[bright90->category[dgx]->roles[slurmclient]->genericresources]%

```

By default, Slurm just allows a single job to be executed per node. To change this behavior, it is necessary to allow oversubscription. For example, to allow 8 jobs per node:

Example

```

[bright90->wlm[slurm]]% jobqueue
[bright90->wlm[slurm]->jobqueue]% use defq
[bright90->wlm[slurm]->jobqueue[defq]]% set oversubscribe yes:8
[bright90->wlm[slurm]->jobqueue*[defq*]]% commit
[bright90->wlm[slurm]->jobqueue[defq]]%

```

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

UGE: GPU configuration is carried out as follows:

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

In Bright Cluster Manager 8.0, 8.1, and 8.2: The command `qconf -mconf <hostname>` is used to add GPU device(s) to the `cgroups_params` parameter for the host.

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

```

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

```

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

Example

```

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

```

The parameter can be made persistent by setting the `AdvancedConfig` directive (page 720) to include the setting `UGEDisableCgroupSettings=1` and restarting `CMDaemon`.

2. GPU devices configuration in UGEClient role (current)

In Bright Cluster Manager 9.0: setting the `AdvancedConfig` directive (page 720) to include the setting `UGEDisableCgroupSettings=1` is not needed. This is because the `qconf -mconf` configuration method is not required.

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

Example

```
[bright90->category[dgx]->roles[ugeclient]]% set gpus 8
[bright90->category*[dgx*]->roles*[ugeclient*]]% set gpudevices gpu0[device=/dev/nvidia0,cuda_id=0]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu1[device=/dev/nvidia1,cuda_id=1]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu2[device=/dev/nvidia2,cuda_id=2]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu3[device=/dev/nvidia3,cuda_id=3]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu4[device=/dev/nvidia4,cuda_id=4]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu5[device=/dev/nvidia5,cuda_id=5]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu6[device=/dev/nvidia6,cuda_id=6]
[bright90->category*[dgx*]->roles*[ugeclient*]]% append gpudevices gpu7[device=/dev/nvidia7,cuda_id=7]
[bright90->category*[dgx*]->roles*[ugeclient*]]% commit
[bright90->category[dgx]->roles[ugeclient]]%.
```

3. UGE port configuration for DCGM

The DCGM port should be set correctly.

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

Example

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

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

In Bright Cluster Manager 9.0 onwards, setting the DCGM port for UGE is a configuration option in the CMDaemon front ends.

LSF: allows GPU devices to be autodetected. To allow it, the administrator should set the following parameter in `lsf.conf`:

Example

```
LSF_GPU_AUTOCONFIG=Y
```

GPU resource enforcement can be configured for LSF as follows:

Example

```
[bright90->wlm[lsf]->cgroups]% append resourceenforce gpu
[bright90->wlm*[lsf*]->cgroups*]% commit
[bright90->wlm[lsf]->cgroups]%
```

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 3.3.1 of the *Developer Manual*) can be used in the scripts. Successful scripts, as is the norm, return 0 on exit.

3.14.1 custompowerscript

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

3.14.2 custompingscript

The following example script:

Example

```
#!/bin/bash
/bin/ping -c1 $CMD_IP
```

can be defined and set for the cases where the default built-in ping script, cannot be used.

By default, the node device states are detected by the built-in ping script (section 5.5) using ICMP ping. This results in the statuses that can be seen on running the list command of cmsh in device mode. An example output, formatted for convenience, is:

Example

```
[root@bright90]# cmsh -c "device; format hostname:15, status:15; list"
hostname (key)  status
-----
bright90       [  UP  ]
node001        [  UP  ]
node002        [  UP  ]
```

If some device is added to the cluster that blocks such pings, then the built-in ping can be replaced by the custom ping of the example, which relies on standard ICMP ping.

However, the replacement custom ping script need not actually use a variety of ping at all. It could be a script running web commands to query a chassis controller, asking if all its devices are up. The script simply has to provide an exit status compatible with expected ping behavior. Thus an exit status of 0 means all the devices are indeed up.

3.14.3 customremoteconsolescript

A custom remote console script can be used to run in the built-in remote console utility. This might be used, for example, to allow the administrator remote console access through a proprietary KVM switch client.

For example, a user may want to run a KVM console access script that is on the head node and with an absolute path on the head node of `/root/kvmaccesshack`. The script is to run on the console, and intended to be used for node `node001`, and takes the argument `1`. This can then be set in `cmsh` as follows:

Example

```
[root@bright90]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% get customremoteconsolescript; get customremoteconsolescriptargument

[bright90->device[node001]]% set customremoteconsolescript /root/kvmaccesshack
[bright90->device[node001]]% set customremoteconsolescriptargument 1
[bright90->device[node001]]% rconsole
      KVM console access session using the 1 argument option is displayed
```

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

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

and

Devices→Nodes↓Edit→Settings→Custom remote console script argument

while the remote console can be launched via the clickpath:

Devices→Nodes↓Edit←Connect→Remote console

3.15 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 Bright View), using CMDaemon to execute the change. Indeed, much of this manual is about this too because it is the preferred technique. It is preferred:

- because it is intended by design to be the easiest way to do common cluster tasks,
- and also generally keeps administration overhead minimal in the long run since it is CMDaemon rather than the system administrator that then takes care of tracking the cluster state.

There are however other cluster configuration techniques besides execution by CMDaemon. To get some perspective on these, it should be noted that cluster configuration techniques are always fundamentally about modifying a cluster so that it functions in a different way. The techniques can then for convenience be separated out into modification techniques that rely on CMDaemon execution and techniques that do not, as follows:

1. **Configuring nodes with execution by CMDaemon:** As explained, this is the preferred technique. The remaining techniques listed here should therefore usually only be considered if the task cannot be done with Bright View or `cmsh`.
2. **Replacing the node image:** The image on a node can be replaced by an entirely different one, so that the node can function in another way. This is covered in section 3.15.2. It can be claimed that since it is CMDaemon that selects the image, this technique should perhaps be classed as under item 1. However, since the execution of the change is really carried out by the changed image without CMDaemon running on the image, and because changing the entire image to implement a change of functionality is rather extreme, this technique can be given a special mention outside of CMDaemon execution.
3. **Using a FrozenFile directive:** Applied to a configuration file, this directive prevents CMDaemon from executing changes on that file for nodes. During updates, the frozen configuration may therefore need to be changed manually. The prevention of CMDaemon acting on that file prevents the standard cluster functionality that would run based on a fully CMDaemon-controlled cluster configuration. The `FrozenFile` directive is introduced in section 2.6.5, and covered in the configuration context in section 3.15.3.
4. **Using an initialize or finalize script:** This type of script is run during the `initrd` stage, much before CMDaemon on the regular node starts up. It is run if the functionality provided by the script is needed before CMDaemon starts up, or if the functionality that is needed cannot be made available later on when CMDaemon is started on the regular nodes. CMDaemon does not execute the functionality of the script itself, but the script is accessed and set on the `initrd` via a CMDaemon front end (Appendix E.2), and executed during the `initrd` stage. It is often convenient to carry out minor changes to configuration files inside a specific image in this way, as shown by the example in Appendix E.5. The `initialize` and `finalize` scripts are introduced in section 3.15.4.
5. **A shared directory:** Nodes can be configured to access and execute a particular software stored on a shared directory of the cluster. CMDaemon does not execute the functionality of the software itself, but is able to mount and share directories, as covered in section 3.10.

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

6. **Software management:** the installation, maintenance, and removal of software packages. Standard post-installation software management based on repositories is covered in sections 11.2–11.6.

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 as on 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 Bright View, via Cluster→Settings→Cluster name
- or in `cmsh` from within the base object of `partition` mode

A system administrator more commonly sets the software image on a per-category or per-node basis (section 3.15.2).

Making Some Nodes Function Differently By Image

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

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

Modifying images via cloning primitives for a node or category: Setting a changed image for a category can be done as follows with `cmsh`:

1. The image on which the new one will be based is cloned. The cloning operation not only copies all the settings of the original (apart from the name), but also the data of the image:

Example

```
[root@bright90 ~]# cmsh
[bright90]% softwareimage
[bright90->softwareimage]% clone default-image imagetwo
[bright90->softwareimage*[imagetwo*]]% commit
Thu Aug 11 15:44:44 2011 [notice] bright90: Started to copy: /cm/\
images/default-image -> /cm/images/imagetwo
[bright90->softwareimage*[imagetwo*]]%
Thu Aug 11 15:53:22 2011 [notice] bright90: Copied: /cm/images/de\
fault-image -> /cm/images/imagetwo
[bright90->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 11.2–11.6 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:

```
[bright90->softwareimage[imagetwo]]% category
[bright90->category]% clone default categorytwo
[bright90->category*[categorytwo*]]% set softwareimage imagetwo
[bright90->category*[categorytwo*]]% commit
[bright90->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):

```
[bright90->category[categorytwo]]% device
[bright90->device]% use node099
[bright90->device[node099]]% set category categorytwo
[bright90->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

```
[bright90->device]% foreach -n node100..node200 (set category categorytwo)
[bright90->device*]% commit
```

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

Modifying images via the `cm/conf` directory, for a category, node, or MAC address: The preceding 5-step method is understandable, but not very elegant. For Bright Cluster Manager 8.2 onwards, there is a more structured and efficient way to make some nodes function differently by image. It relies on adding the files in the image, via a target path that is specified in special configuration locations in the image. It can be configured per node, but also per category and MAC address.

- For a category, the specification takes the form:

```
/cm/images/<image>/cm/conf/category/<category>/<target>
```

Thus, if some file on the node is to be placed so that on a running node it is at `/path/to/some.file`, and this needs to be configured for an image `default-image`, and a category `default`, then it would be placed at this location on the head node:

Example

```
/cm/images/default-image/cm/conf/category/default/path/to/some.file
```

The file on the target node would be placed in the absolute directory `/path/to/some.file`

Multiple categories can be configured per image. Thus, for example, beside the default category, an additional `gpu` category can exist:

Example

```
/cm/images/default-image/cm/conf/category/gpu/path/to/some.file
```

Also, multiple files can be specified per category per image. Thus, beside the file `some.file`, an additional file `some.other.file` could be placed:

Example

```
/cm/images/default-image/cm/conf/category/gpu/path/to/some.file
/cm/images/default-image/cm/conf/category/gpu/path/to/some.other.file
```

- For a node, the configuration form is:

```
/cm/images/<image>/cm/conf/node/<node name>/<target>
```

An example for a node called node001 could then be:

Example

```
/cm/images/default-image/cm/conf/node/node001/path/to/some.file
```

- For a MAC address, the configuration form is:

```
/cm/images/<image>/cm/conf/node/<MAC address>/<target>
```

An example for a node with MAC address 00:aa:bb:cc:dd:ee could then be:

Example

```
/cm/images/default-image/cm/conf/node/00-aa-bb-cc-dd-ee/path/to/some.file
```

The copying of the specified files to the image is done just before the finalize stage of the node-installer (section 5.4.11) during node provisioning.

A common theme in Bright Cluster Manager is that node-level configuration overrides category-level configuration. In keeping with this behavior, a file configuration at category level could be applied to the many nodes in a category. And, a file configuration copy at node level (for a node that is in the category) overrides the category level value for just that particular node.

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

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

However, sometimes an administrator may need to make a direct change (without using Bright View or `cmsh`) to a configuration file to set up a special configuration that cannot otherwise be done.

The `FrozenFile` directive to `CMDeamon` (Appendix C) applied to such a configuration file stops `CMDeamon` 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 `CMDeamon`-based method of configuring nodes, for the sake of administrative maintainability.

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

`CMDeamon` 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:

```
#!/bin/bash

## node010..node014 get forced to 1000 duplex
if [[ $CMD_HOSTNAME = node01[0-4] ]]
then
echo 'ETHTOOL_OPTS="speed 1000 duplex full"'>>/localdisk/etc/sysconfig/network-scripts/ifcfg-eth0
fi
```

The method of enforcing an interface speed just outlined is actually just for educational illustration, and is not a recommended method.

In practice, the recommended way to enforce an interface speed is to simply set it in the CMDaemon database. For example, for the boot interface of `node001` it could be via the Bright View clickpath:

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

3.15.5 Examples Of Configuring Nodes With Or Without CMDaemon

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

Example

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

A counterexample to this is:

Example

Configuring a software for nodes without using Bright View or `cmsh`³, using an image: Software images can be created with and without CUPS configured. Setting up nodes to load one of these two images via a node category is an alternative way of letting nodes run CUPS.

Whether node configuration for a particular functionality is done with CMDaemon, or directly with the software, depends on what an administrator prefers. In the preceding two examples, the first example, that is the one with Bright View or `cmsh` setting the CUPS service, is likely to be preferred over the second example, where an entire separate image must be maintained. A new category must also be created in the second case.

Generally, sometimes configuring the node via 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.

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

Configuring Nodes Using Bright View Or `cmsh`: Category Settings

When configuring nodes using Bright View or `cmsh`, configuring particular nodes from a node category to overrule the state of the rest of its category (as explained in section 2.1.3) is sensible for a small number of nodes. For larger numbers it may not be organizationally practical to do this, and another category can instead be created to handle nodes with the changes conveniently.

The CUPS service in the next two examples is carried out by implementing the changes via Bright View or `cmsh` acting on `CMDDaemon`.

Example

Setting a few nodes in a category: If only a few nodes in a category are to run CUPS, then it can be done by enabling CUPS just for those few nodes, thereby overriding (section 2.1.3) the category settings.

Example

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

In contrast to these two examples, the software image method used in section 3.15.2 to implement a functionality such as CUPS would load up CUPS as configured in an image, and would not handle it via `CMDDaemon`³. So, in section 3.15.2, software images prepared by the administrator are set for a node category. Since, by design, images are only selected for a category, a node cannot override the image used by the category other than by creating a new category, and using it with the new image. The administrative overhead of this can be inconvenient.

Administrators would therefore normally prefer letting `CMDDaemon` 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 `CMDDaemon` to set categories or regular nodes to provide certain functions, typically by enabling services.

4

Power Management

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 power-saving options in workload managers
- ensuring the passive head node can safely take over from the active head during failover (Chapter 14)
- 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.

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

4.1 Configuring Power Parameters

Several methods exist to control power to devices:

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

4.1.1 PDU-Based Power Control

For PDU-based power control, the power supply of a device is plugged into a port on a PDU. The device can be a node, but also anything else with a power supply, such as a switch. The device can then be turned on or off by changing the state of the PDU port.

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

The `PowerDistributionUnits` menu option, accessible via the clickpath `Devices[device]→Settings` pane for each device plugged into the PDU, can then be used to configure the PDU ports that control the device. Figure 4.1 shows the menu option in the `Settings` window for a head node device, `bright90`, as accessed via the clickpath `Devices→Head Nodes[bright90]→Settings`.

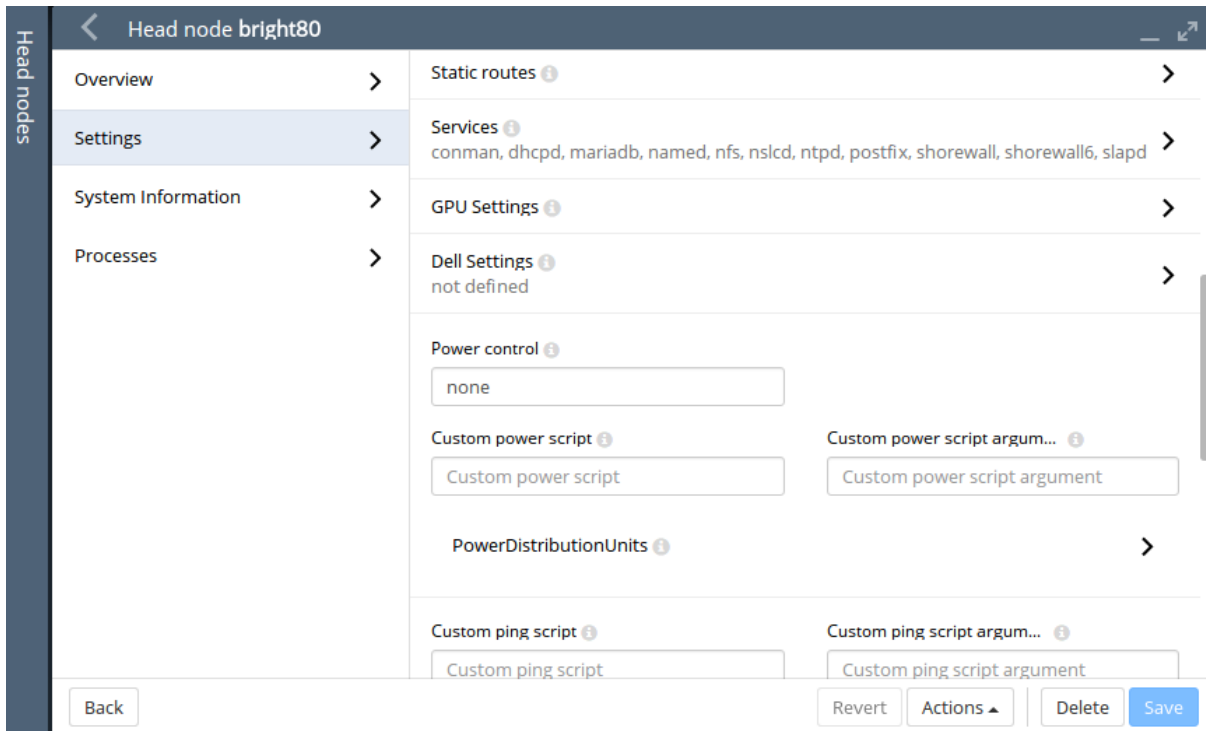


Figure 4.1: Power Section In The Head Node Settings

If the menu option is clicked, then a subpane opens that allows each device plugged into the PDU to have PDU ports added and removed. For the APC brand of PDUs, the `Power control` property in the window of figure 4.1 should be set to `apc`, or the list of PDU ports is ignored by default. Overriding the default is described in section 4.1.3.

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

It is also possible for multiple devices to share the same PDU port. This is the case for example when *twin nodes* are used (i.e. two nodes sharing a single power supply). In this case, all power operations on one device apply to all nodes sharing the same PDU port.

If the PDUs defined for a node are not manageable, then the node's baseboard management controllers (that is, IPMI/iLO and similar) are assumed to be inoperative and are therefore assigned an unknown state. This means that dumb PDUs, which cannot be managed remotely, are best not assigned to nodes in 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 Bright View the `Notes` field or the `Userdefined1/Userdefined2` fields 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 `cmsh`, power-related options can be accessed from device mode, after selecting a device:

Example

```
[bright90]% device use node001
[bright90->device[node001]]% show | grep -i power
Custom power script argument
Ipmitool power reset delay          0
Power control                       apc
PowerDistributionUnits              apc01:6 apc01:7
```

The power status of a node can be accessed with:

Example

```
[bright90->device[node001]]% power status
```

If the node is up and has one or more PDUs assigned to it, then the power status is one of ON, OFF, RESET, FAILED, or UNKNOWN:

Power Status	Description
ON	Power is on
OFF	Power is off
RESET	Shows during the short time the power is off during a power reset. The reset is a hard power off for PDUs, but can be a soft or hard reset for other power control devices.
FAILED	Power status script communication failure.
UNKNOWN	Power status script timeout

4.1.2 IPMI-Based Power Control

IPMI-based power control relies on the baseboard management controller (BMC) inside a node. It is therefore only available for node devices. Blades inside a blade chassis typically use IPMI for power management. Section 3.7 describes setting up networking and authentication for IPMI/iLO/DRAC/CIMC/Redfish interfaces.

To carry out IPMI-based power control operations, the `Power control` property in figure 4.1 must be set to the IPMI interface through which power operations should be relayed. Normally this IPMI interface is configured to be `ipmi0`. Any list of configured APC PDU ports displayed in the GUI is ignored by default when the `Power control` property is not `apc`.

Example

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

```
[mycluster]% device
[...device]% foreach -t physicalnode (set powercontrol ipmi0; commit)
```

Example

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

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

4.1.3 Combining PDU- and IPMI-Based Power Control

By default when nodes are configured for IPMI Based Power Control, any configured PDU ports are ignored. However, it is sometimes useful to change this behavior.

For example, in the `CMDaemon` configuration file directives in `/cm/local/apps/cmd/etc/cmd.conf` (introduced in section 2.6.2 and listed in Appendix C), the default value of `PowerOffPDUOutlet` is `false`. It can be set to `true` on the head node, and `CMDaemon` restarted to activate it.

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

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

4.1.4 Custom Power Control

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

Power operations are described further in section 4.2.

Using `custompowerscript`

To set a custom power management script for a device, the `powercontrol` attribute is set by the administrator to `custom` using either `Bright View` or `cmsh`, and the value of `custompowerscript` is specified by the administrator. The value for `custompowerscript` is the full path to an executable custom power management script on the head node(s) of a cluster.

A custom power script is invoked with the following mandatory arguments:

```
myscript <operation> <device>
```

where `<device>` is the name of the device on which the power operation is done, and `<operation>` is one of the following:

```
ON
OFF
RESET
STATUS
```

On success a custom power script exits with exit code 0. On failure, the script exits with a non-zero exit-code.

Using `custompowerscriptargument`

The mandatory argument values for `<operation>` and `<device>` are passed to a custom script for processing. For example, in bash the positional variables `$1` and `$2` are typically used for a custom power script. A custom power script can also be passed a further argument value by setting the value of `custompowerscriptargument` for the node via `cmsh` or Bright View. This further argument value would then be passed to the positional variable `$3` in bash.

An example custom power script is located at `/cm/local/examples/cmd/custompower`. In it, setting `$3` to a positive integer delays the script via a `sleep` command by `$3` seconds.

An example that is conceivably more useful than a `sleep $3` command is to have a `wakeonlan $3` command instead. If the `custompowerscriptargument` value is set to the MAC address of the node, that means the MAC value is passed on to `$3`. Using this technique, the power operation ON can then carry out a Wake On LAN operation on the node from the head node.

Setting the `custompowerscriptargument` can be done like this for all nodes:

```
#!/bin/bash
for nodename in $(cmsh -c "device; foreach * (get hostname)")
do
    macad=`cmsh -c "device use $nodename; get mac"`
    cmsh -c "device use $nodename; set customscriptargument $macad; commit"
done
```

The preceding material usefully illustrates how `custompowerscriptargument` can be used to pass on arbitrary parameters for execution to a custom script.

However, the goal of the task can be achieved in a simpler and quicker way using the environment variables available in the cluster management daemon environment (section 3.3.1 of the *Developer Manual*). This is explained next.

Using Environment Variables With `custompowerscript`

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

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

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

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

So, it is not necessary to retrieve a MAC value for `custompowerscriptargument` with a bash script as shown in the previous section, and then pass the argument via `$3` such as done in the command `wakeonlan $3`. Instead, `custompowerscript` can simply call `wakeonlan $CMD_MAC` directly in the script when run as a power operation command from within CMDaemon.

4.1.5 Hewlett Packard iLO-Based Power Control

iLO Configuration During Installation

If “Hewlett Packard” is chosen as the node manufacturer during installation (section 3.3.11 of the *Installation Manual*), and the nodes have an iLO management interface, then Hewlett-Packard’s iLO management package, `hponcfg`, is installed by default on the nodes and head nodes.

iLO Configuration After Installation

If “Hewlett Packard” has not been specified as the node manufacturer during installation then it can be configured after installation as follows:

The `hponcfg` rpm package is normally obtained and upgraded for specific HP hardware from the HP website. Using an example of `hponcfg-3.1.1-0.noarch.rpm` as the package downloaded from the

HP website, and to be installed, the installation can then be done on the head node, the software image, and in the node-installer as follows:

```
rpm -iv hponcfg-3.1.1-0.noarch.rpm
rpm --root /cm/images/default-image -iv hponcfg-3.1.1-0.noarch.rpm
rpm --root /cm/node-installer -iv hponcfg-3.1.1-0.noarch.rpm
```

To use iLO on a node, the iLO interface of the node is set up just like the IPMI interfaces as outlined in section 4.1.2. That is, using “set powercontrol ilo0” instead of “set powercontrol ipmi0”. The cluster manager treats HP iLO interfaces just like regular IPMI interfaces, except that the interface names are ilo0, ilo1... instead of ipmi0, ipmi1...

For example, nodes in the default category can be brought under iLO power control as follows:

Example

```
[mycluster]% device foreach -c default (set powercontrol ilo0)
[mycluster]% device commit
```

4.1.6 Dell drac-based Power Control

Dell drac configuration is covered on page 92.

4.1.7 Redfish-Based and CIMC-Based Power Control

Section 3.7 describes setting up networking and authentication for Redfish/CIMC, as well as for IPMI/iLO/DRAC interfaces.

4.2 Power Operations

4.2.1 Power Operations Overview

Main Power Operations

Power operations may be carried out on devices from either Bright View or cmsh. There are four main power operations:

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

Scheduling-related Power Operations

There are also *scheduling-related* power operations, which are currently (December 2018) only accessible via cmsh. Scheduling-related power operations are power operations associated with managing and viewing explicitly-scheduled execution.

Scheduled execution of power operations can be carried out explicitly via the `--at`, `--after`, `-d`, and `-parallel-delay` options. The scheduling-related power operations to manage and view such scheduled power operations are:

- `power wait`: Identifies the devices that have power operations that are in the waiting state, i.e. waiting to be carried out, and also outputs the number of operations that are waiting to be carried out.
- `power cancel`: Cancels an operation in the waiting state. The devices on which they should be cancelled can be specified.
- `power list`: Lists the power operations on the device and the states of the operations. Possible states for operations are:

- waiting: waiting to be executed
- busy: are being executed
- canceled: have been canceled
- done: have been executed

It is possible that power operations without an explicitly-scheduled execution time setting show up very briefly in the output of `power list` and `power wait`. However, the output displayed is almost always about the explicitly-scheduled power operations.

4.2.2 Power Operations With Bright View

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

- via the menu dropdown for a node. For example:
 - for the head node, via the clickpath `Devices→Head Nodes↓Power`
 - for a regular node via the clickpath `Devices→Nodes↓Power`
- via the menu dropdown for a category or group. For example, for the default category, via the clickpath `Grouping→Categories↓Power`
- via the Actions button. The Actions button is available when specific device has been selected. For example, for a head node `bright90` the button can be seen via the clickpath `Devices→Head Nodes [bright90]→Settings`, as illustrated in figure 4.1. Clicking on the button makes power operation buttons available figure 4.2.

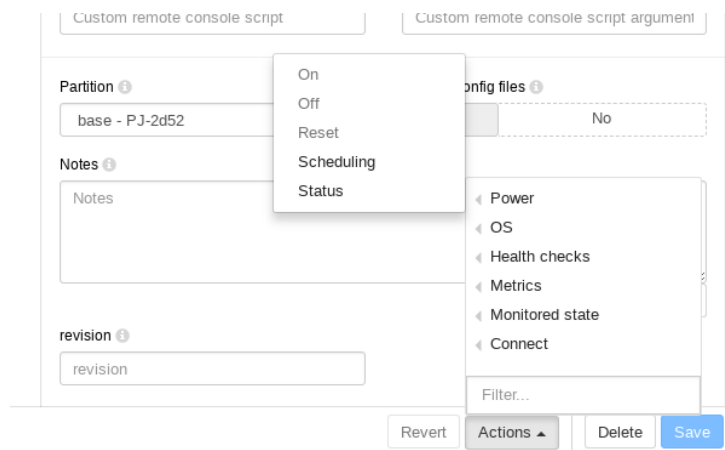


Figure 4.2: Actions Button, Accessing The Power Operations

4.2.3 Power Operations Through `cmsh`

Power operations on nodes can be carried out from within the device mode of `cmsh`, via the `power` command options.

Powering On

Powering on can be carried out on a list of nodes (page 42). Powering on `node001`, and nodes from `node018` to `node033` (output truncated):

Example

```
[mycluster]% device power -n node001,node018..node033 on
apc01:1 ..... [ ON ] node001
apc02:8 ..... [ ON ] node018
```

```
apc02:9 ..... [ ON ] node019
...
```

When a power operation is carried out on multiple devices, CMDaemon ensures that a 1 second delay occurs by default between successive devices. This helps avoid power surges on the infrastructure.

Delay Period Between Nodes

The delay period can be modified from within the device mode of `cmsh`, by using the `-d|--delay` option of the power command. For example, the preceding power command can be run with a shorter, 10ms delay with:

```
[mycluster]% device power -n node001,node018..node033 -d 0.01 on
```

A 0-second delay (`-d 0`) should not be set for larger number of nodes, unless the power surge that this would cause has been taken into consideration.

Powering Up In Batches

Groups of nodes can be powered up “in batches”, according to power surge considerations. For example, to power up 3 racks at a time (“in batches of 3”), the `-p|--parallel` option is used:

Example

```
[mycluster]% device power on -p 3 rack[01-12]
```

By default, there is a delay of 20s between batch commands. So, in the preceding example, there is a 20s pause before the each batch of the next three racks is powered up. For batch operation a delay of `-d 0` is assumed, i.e. the nodes within in the rack are powered up without a built-in delay between the nodes of the rack.

Thread Use During Powering Up

The default number of threads that are started up to handle powering up of all the nodes is 32. If the hardware can cope with it, then it is possible to decrease startup time by increasing the default number of threads used to handle powerup, by editing the `PowerThreadPoolSize` advanced configuration directive in CMDaemon (page 714).

Powering Off Nodes

An example of powering off nodes is the following, where all nodes in the `default` category are powered off, with a 100ms delay between nodes (some output elided):

Example

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

Getting The Power Status

The `power status` command lists the status for devices:

Example

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


Getting The Power History

The `power history` command lists the last few power operations on nodes. By default it lists up to the last 8.

Example

```
[mycluster]% device power history
Device   Time                               Operation   Success
-----
node001  Sat Sep 14 03:35:03 2019 shutdown   yes
node001  Fri Sep 20 14:28:38 2019 on        yes
node002  Sat Sep 14 03:35:03 2019 shutdown   yes
node002  Fri Sep 20 14:28:38 2019 on        yes
node003  Sat Sep 14 03:35:03 2019 shutdown   yes
node003  Fri Sep 20 14:28:38 2019 on        yes
node004  Sat Sep 14 03:35:03 2019 shutdown   yes
```

The power Command Help Text

The help text for the power command is:

```
[bright90->device]% help power
Name:
    power - Manipulate or retrieve power state of devices

Usage:
    power [OPTIONS] status
    power [OPTIONS] on
    power [OPTIONS] off
    power [OPTIONS] reset
    power [OPTIONS] list
    power [OPTIONS] history
    power [OPTIONS] cancel
    power [OPTIONS] wait <index>

Options:
    -n, --nodes <node>
        List of nodes, e.g. node001..node015,node020..node028,node030 or
        ~/some/file/containing/hostnames

    -g, --group <group>
        Include all nodes that belong to the node group, e.g. testnodes or test01,test03

    -c, --category <category>
        Include all nodes that belong to the category, e.g. default or default,gpu

    -r, --rack <rack>
        Include all nodes that are located in the given rack, e.g rack01 or
        rack01..rack04

    -h, --chassis <chassis>
        Include all nodes that are located in the given chassis, e.g chassis01 or
        chassis03..chassis05
```

-e, --overlay <overlay>
Include all nodes that are part of the given overlay, e.g overlay1 or overlayA,overlayC

-m, --image <image>
Include all nodes that have the given image, e.g default-image or default-image,gpu-image

-t, --type <type>
Type of devices, e.g node or virtualnode,cloudnode

-i, --intersection
Calculate the intersection of the above selections

-u, --union
Calculate the union of the above selections

-l, --role role
Filter all nodes that have the given role

-s, --status <status>
Only run command on nodes with specified status, e.g. UP, "CLOSED|DOWN", "INST.*"

-b, --background
Run in background, output will come as events

-d, --delay <seconds>
Wait <seconds> between executing two sequential power commands. This option is ignored for the status command

-f, --force
Force power command on devices which have been closed

-w, --overview
Group all power operation results into an overview

-p, --parallel <number>
Number of parallel option-items to be used per batch, default 0 (disabled)

--at <time>
Execute the operation at the provided time

--after <seconds>
Wait <seconds> before executing the operation

--parallel-delay <seconds>
Wait <seconds> between executing the next batch of parallel commands, default 20s

--parallel-dry-run
Only display the times at which operations will be executed, do not perform any power operations

--retry-count <number>

```

    Number of times to retry operation if it failed the first time (default 0)

--retry-delay <seconds>
    Delay between consecutive tries of a failed power operation (default 3s)

--port <pdu>:<port>
    Do the power operation directly on a pdu port.

```

Examples:

```

power status           Display power status for all devices or current device
power on node001       Power on node001
power on -n node00[1-2] Power on node001 and node002
power list             List all pending power operations
power history          List the last couple of power operations
power wait            List all power operation that can be waited for
power wait 1          Wait for a power operation to be completed
power wait all        Wait for all power operations to be completed
power wait last       Wait for the last given power operation to be completed
power off --after 10m  Power off the current node after 10 minutes
power off --at 23:55   Power off the current node today just before midnight
power cancel node001   Cancel all pending power operations for node001
power on -p 4 rack[01-80] Power on racks 1 to 80 in batches of 4. With a delay of 20s
                        between each batch. And a delay of 0s between nodes.
power on --port pdu1:1 Power on port 1 on pdu1
power on --port pdu1:[1-4] Power on port 1 through 4 on pdu1

```

4.3 Monitoring Power

Monitoring power consumption is important since electrical power is an important component of the total cost of ownership for a cluster. The monitoring system of 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 12 on cluster monitoring has more on metrics and how they can be visualized.

4.4 Switch Configuration To Survive Power Downs

Besides the nodes and the BMC interfaces being configured for power control, it may be necessary to check that switches can handle power on and off network operations properly. Interfaces typically negotiate the link speed down to reduce power while still supporting Wake On Lan and other features. During such renegotiations the switch may lose connectivity to the node or BMC interface. This can happen if dynamic speed negotiation is disabled on the switch. Dynamic speed negotiation should therefore be configured to be on on the switch in order to reduce the chance that a node does not provision from a powered down state.

5

Node Provisioning

This chapter covers *node provisioning*. Node provisioning is the process of how nodes obtain an image. Typically, this happens during their stages of progress from power-up to becoming active in a cluster, but node provisioning can also take place when updating a running node.

Section 5.1 describes the stages leading up to the loading of the kernel onto the node.

Section 5.2 covers configuration and behavior of the provisioning nodes that supply the software images.

Section 5.3 describes the configuration and loading of the kernel, the ramdisk, and kernel modules.

Section 5.4 elaborates on how the node-installer identifies and places the software image on the node in a 13-step process.

Section 5.5 explains node states during normal boot, as well node states that indicate boot problems.

Section 5.6 describes how running nodes can be updated, and modifications that can be done to the update process.

Section 5.7 explains how to add new nodes to a cluster so that node provisioning will work for these new nodes too. The Bright View and `cmsh` front ends for creating new node objects and properties in `CMDaemon` are described.

Section 5.8 describes troubleshooting the node provisioning process.

5.1 Before The Kernel Loads

Immediately after powering up a node, and before it is able to load up the Linux kernel, a node starts its boot process in several possible ways:

5.1.1 PXE Booting

By default, nodes boot from the network when using Bright Cluster Manager. This is called a *network boot*. On the x86 architectures it is known as a *PXE boot* (often pronounced as “pixie boot”). It is recommended as a BIOS setting for nodes. The head node runs a `tftpd` server that is managed by `systemd`. The `tftpd` server supplies the boot loader from within the default software image (section 2.1.2) offered to nodes.

The boot loader runs on the node and displays a menu (figure 5.1) based on loading a menu module within a configuration file. The default configuration files offered to nodes are located under `/tftpboot/pxelinux.cfg/` on the head node. To implement changes in the files, `CMDaemon` may need to be restarted, or the `updateprovisioners` command (page 157) can be run.

The default configuration files give instructions to the menu module of `PXElinux`. The instruction set used is documented at <http://www.syslinux.org/wiki/index.php/Comboot/menu.c32>, and includes the `TIMEOUT`, `LABEL`, `MENU LABEL`, `DEFAULT`, and `MENU DEFAULT` instructions.

The PXE TIMEOUT Instruction

During the display of the PXE boot menu, a selection can be made within a timeout period to boot the node in a several ways. Among the options are some of the install mode options (section 5.4.4). If no

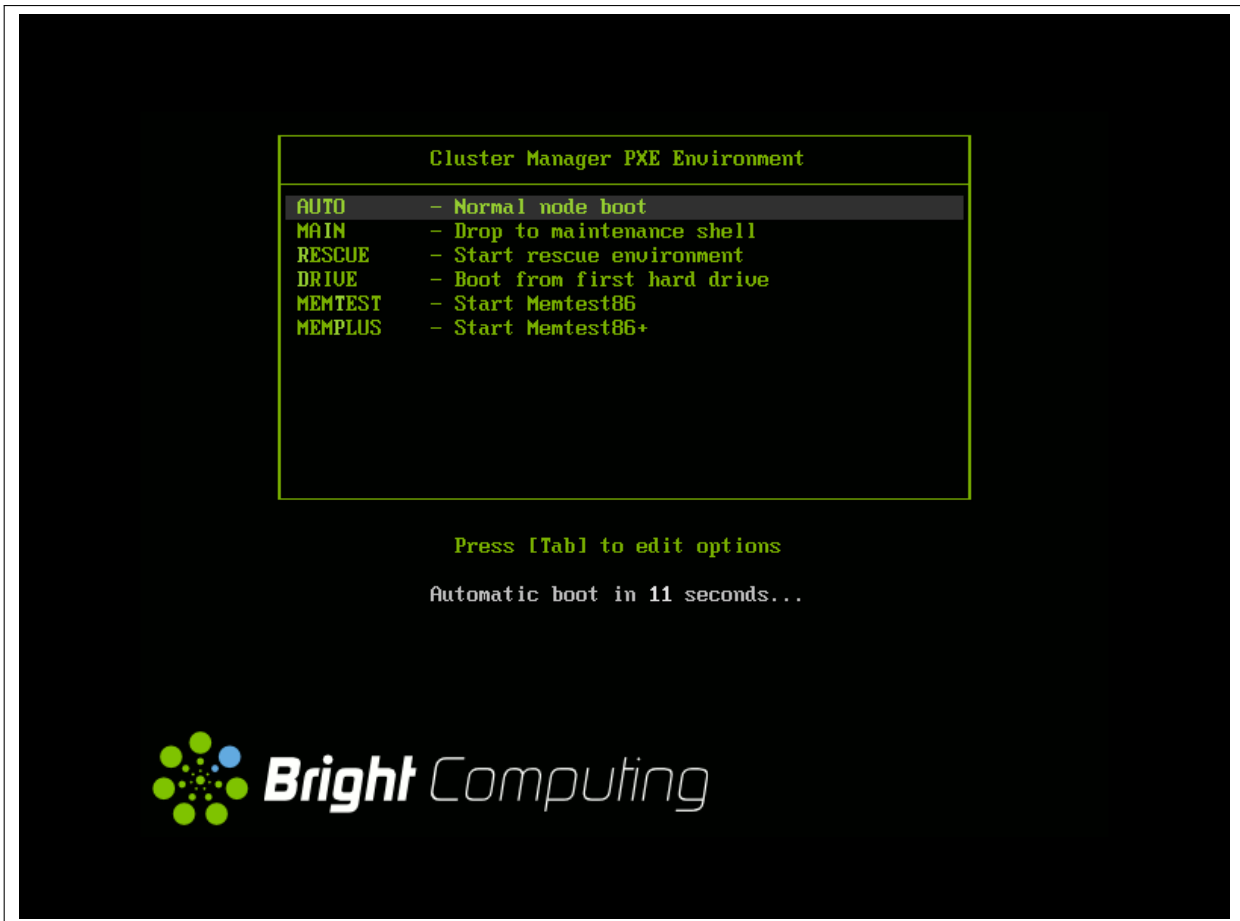


Figure 5.1: PXE boot menu options

selection is made by the user within the timeout period, then the AUTO install mode option is chosen by default.

In the PXE menu configuration files under `pxelinux.cfg/`, the default timeout of 5 seconds can be adjusted by changing the value of the “TIMEOUT 50” line. This value is specified in deciseconds.

Example

```
TIMEOUT 300 # changed timeout from 50 (=5 seconds)
```

The PXE LABEL And MENU LABEL Instructions

LABEL: The menu configuration files under `pxelinux.cfg/` contain several multiline LABEL statements.

Each LABEL statement is associated with a kernel image that can be loaded from the PXE boot menu along with appropriate kernel options.

Each LABEL statement also has a text immediately following the LABEL tag. Typically the text is a description, such as `linux`, `main`, `RESCUE`, and so on. If the PXE menu module is not used, then tab completion prompting displays the list of possible text values at the PXE boot prompt so that the associated kernel image and options can be chosen by user intervention.

MENU LABEL: By default, the PXE menu module is used, and by default, each LABEL statement also contains a MENU LABEL instruction. Each MENU LABEL instruction also has a text immediately following the MENU LABEL tag. Typically the text is a description, such as `AUTO`, `RESCUE` and so on (figure 5.1). Using the PXE menu module means that the list of the MENU LABEL text values is displayed when the PXE boot

menu is displayed, so that the associated kernel image and options can conveniently be selected by user intervention.

The PXE DEFAULT And MENU DEFAULT Instructions

DEFAULT: If the PXE menu module is not used and if no MENU instructions are used, and if there is no user intervention, then setting the same text that follows a LABEL tag immediately after the DEFAULT instruction, results in the associated kernel image and its options being run by default after the timeout.

By default, as already explained, the PXE menu module is used. In particular it uses the setting: `DEFAULT menu.c32` to enable the menu.

MENU DEFAULT: If the PXE menu module is used and if MENU instructions are used, and if there is no user intervention, then setting a MENU DEFAULT tag as a line within the multiline LABEL statement results in the kernel image and options associated with that LABEL statement being loaded by default after the timeout.

The CMDaemon PXE Label Setting For Specific Nodes

The MENU DEFAULT value by default applies to every node using the software image that the PXE menu configuration file under `pxelinux.cfg/` is loaded from. To override its application on a per-node basis, the value of PXE Label can be set for each node.

- Some simple examples of overriding the default MENU DEFAULT value are as follows:

For example, using `cmsh`:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% set pxelabel MEMTEST ; commit
```

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

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% foreach -c default (set pxelabel MEMTEST)
[bright90->device*]% commit
```

The value of `pxelabel` can be cleared with:

Example

```
[root@bright90 ~]# cmsh -c "device; foreach -c default (clear pxelabel); commit"
```

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

- A more complicated example of overriding the default MENU DEFAULT value now follows. Although it helps in understanding how PXE labels can be used, it can normally be skipped because the use case for it is unlikely, and the details are involved.

In this example, `pxelabel` is set by the administrator via Bright View or `cmsh` to `localdrive`. This will then set the node to boot from the first local drive and not the node-installer. This is a setting that is discouraged since it usually makes node management harder, but it can be used by

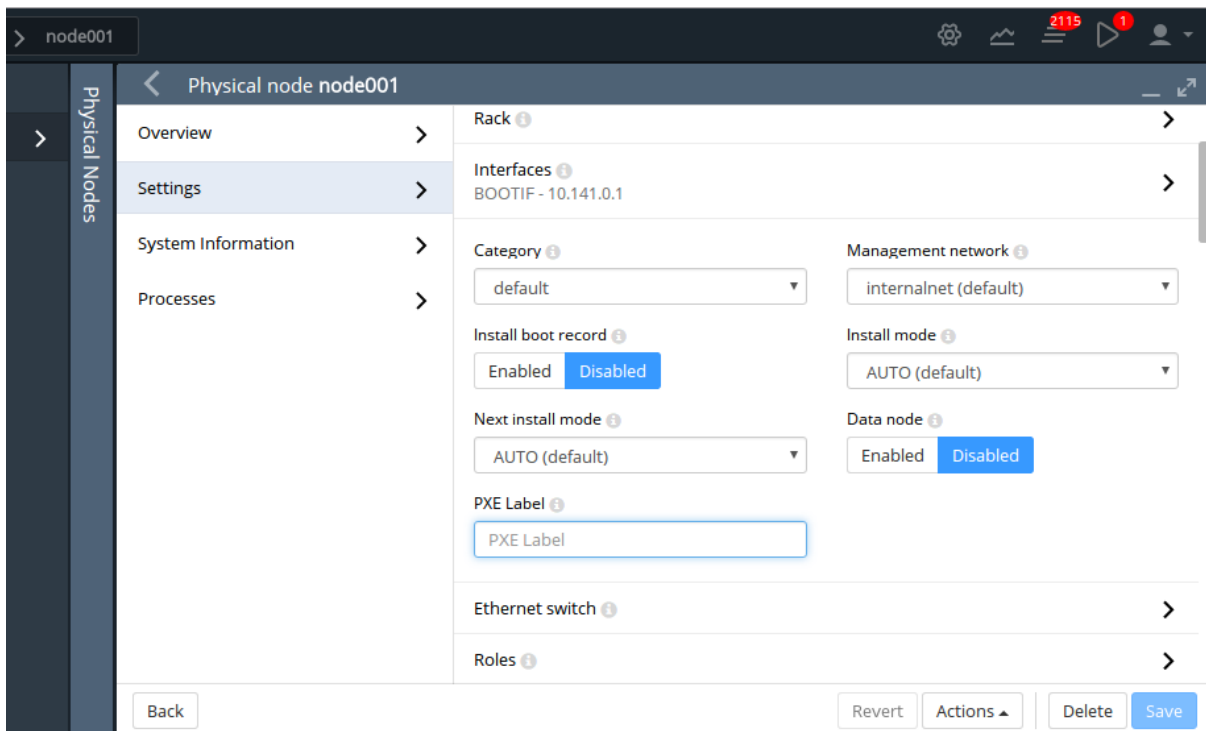


Figure 5.2: Bright View PXE Label option

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 11.

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. Node booting (section 3.2.3, page 72) can be set from `cmsh` or Bright View as follows:
 - (a) From `cmsh`'s network mode: If the InfiniBand network name is `ibnet`, then a `cmsh` command that will set it is:

```
cmsh -c "network; set ibnet nodebooting yes; commit"
```
 - (b) From Bright View: The Settings window for the InfiniBand network, for example `ibnet`, can be accessed from the Networking resource via the clickpath `Networking→Networks[ibnet]→Edit→Settings` (this is similar to figure 3.5, but for `ibnet`). The Node booting option for `ibnet` is then enabled and saved.

If the InfiniBand network does not yet exist, then it must be created (section 3.2.2). The recommended default values used are described in section 3.6.3. The MAC address of the interface in `CMDaemon` defaults to using the GUID of the interface.

The administrator should also be aware that the interface from which a node boots, (conveniently labeled `B00TIF`), must not be an interface that is already configured for that node in `CMDaemon`. For example, if `B00TIF` is the device `ib0`, then `ib0` must not already be configured in `CMDaemon`. Either `B00TIF` or the `ib0` configuration should be changed so that node installation can succeed. It is recommended to set `B00TIF` to `eth0` if the `ib0` device should exist.

2. Flashing iPXE onto the InfiniBand HCAs. (The ROM image is obtained from the HCA vendor).
3. Configuring the BIOS of the nodes to boot from the InfiniBand HCA. For PXE booting over Omni-Path, older Omni-Path cards may need to have UEFI firmware installed on them, and the only supported boot mode is then UEFI.

All MAC addresses become invalid for identification purposes when changing from booting over Ethernet to booting over InfiniBand.

Administrators who enable iPXE booting almost always wish to provision over InfiniBand too. Configuring provisioning over InfiniBand is described in section 5.3.3.

5.1.4 Using PXE To Boot From The Drive

Besides PXE booting from only the network, a node can also be configured via PXE to step over to using its own drive to start booting and get to the stage of loading up its kernel entirely from its drive, just like a normal standalone machine. This can be done by setting `PXE LABEL` to `localdrive` (page 147).

5.1.5 Network Booting Without PXE On The ARMv8 Architecture

ARMv8 nodes use a network boot implementation that differs slightly from the x86 PXE boot implementation. The actual firmware that starts up on ARMv8 nodes depends on the environment the hardware is in. Networking is then started by the firmware and the network requests what to boot. The head node however then sends out a GRUB binary instead of an iPXE binary. The GRUB binary then runs on the regular node, and fetches the kernel and initrd via TFTP. The node installer then runs on the nodes and follows the same steps as in the x86 process.

5.1.6 Network Booting Protocol

The protocol used by network booting is set with the parameter `bootloaderprotocol`. It is set to HTTP by default at category level:

Example

```
[bright90->category[default-centos7-x86_64]]% get bootloaderprotocol
HTTP
```

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

Example

```
[bright90->device[node001]]% get bootloaderprotocol
HTTP (default-centos7-x86_64)
[bright90->device[node001]]% set bootloaderprotocol<TAB><TAB>
http https tftp
[bright90->device[node001]]% set bootloaderprotocol tftp
[bright90->device*[node001*]]% commit
[bright90->device[node001]]% get bootloaderprotocol
TFTP
```

The HTTPS protocol for node booting should almost never be used, because it is rarely implemented in hardware.

5.1.7 The Boot Role

The action of providing a boot image to a node via DHCP and TFTP is known as providing *node booting*. Node provisioning (section 5.2), on the other hand, is about provisioning the node with the rest of the node image.

Roles in general are introduced in section 2.1.5. The *boot role* is one such role that can be assigned to a regular node. The boot role configures a regular node so that it can then provide node booting. The role cannot be assigned or removed from the head node—the head node always has a boot role.

The boot role is assigned by administrators to regular nodes if there is a need to cope with the scaling limitations of TFTP and DHCP. TFTP and DHCP services can be overwhelmed when there are large numbers of nodes making use of them during boot. An example of the scaling limitations may be observed, for example, when, during the powering up and network booting attempts of a large number of regular nodes from the head node, it turns out that random different regular nodes are unable to boot, typically due to network effects.

One implementation of boot role assignment might therefore be, for example, to have a several groups of racks, with each rack in a subnet, and with one regular node in each subnet that is assigned the boot role. The boot role regular nodes would thus take the DHCP and TFTP load off the head node and onto themselves for all the nodes in their associated subnet, so that all nodes of the cluster are then able to boot without networking issues.

5.2 Provisioning Nodes

The action of transferring the software image to the nodes is called *node provisioning*, and is done by special nodes called the *provisioning nodes*. More complex clusters can have several provisioning nodes configured by the administrator, thereby distributing network traffic loads when many nodes are booting.

Creating provisioning nodes is done by assigning a *provisioning role* to a node or category of nodes. Similar to how the head node always has a boot role (section 5.1.7), the head node also always has a provisioning role.

5.2.1 Provisioning Nodes: Configuration Settings

The provisioning role has several parameters that can be set:

Property	Description
<code>allImages</code>	<p>The following values decide what images the provisioning node provides:</p> <ul style="list-style-type: none"> <code>onlocaldisk</code> (the default): all images on the local disk, regardless of any other parameters set <code>onlocaldiskexceptsharedimages</code>: all images on the local disk, except for shared images <code>onsharedstorage</code>: all images on the shared storage, regardless of any other parameters set <code>no</code>: only images listed in the <code>localimages</code> or <code>sharedimages</code> parameters, described next
<code>localimages</code>	A list of software images on the local disk that the provisioning node accesses and provides. The list is used only if <code>allImages</code> is "no".
<code>sharedimages</code>	A list of software images on the shared storage that the provisioning node accesses and provides. The list is used only if <code>allImages</code> is "no"
<code>Provisioning slots</code>	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.
<code>nodegroups</code>	<p>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 <code>nodegroups</code>, can only be provisioned by provisioning nodes that have no <code>nodegroups</code> values set. By default, the <code>nodegroups</code> list is unset in the provisioning nodes.</p> <p>The <code>nodegroups</code> setting is typically used to set up a convenient hierarchy of provisioning, for example based on grouping by rack and by groups of racks.</p>

A provisioning node keeps a copy of all the images it provisions on its local drive, in the same directory as where the head node keeps such images. The local drive of a provisioning node must therefore have enough space available for these images, which may require changes in its disk layout.

5.2.2 Provisioning Nodes: Role Setup With `cmsh`

In the following `cmsh` example the administrator creates a new category called `misc`. The default category `default` already exists in a newly installed cluster.

The administrator then assigns the role called `provisioning`, from the list of available assignable roles, to nodes in the `misc` category. After the `assign` command has been typed in, but before entering the command, tab-completion prompting can be used to list all the possible roles. Assignment creates an association between the role and the category. When the `assign` command runs, the shell drops into the level representing the provisioning role.

If the role called `provisioning` were already assigned, then the `use provisioning` command would drop the shell into the provisioning role, without creating the association between the role and the category.

As an aside from the topic of provisioning, from an organizational perspective, other assignable roles include `monitoring`, `storage`, and `failover`.

Once the shell is within the role level, the role properties can be edited conveniently.

For example, the nodes in the `misc` category assigned the `provisioning` role can have `default-image` set as the image that they provision to other nodes, and have 20 set as the maximum number of other nodes to be provisioned simultaneously (some text is elided in the following example):

Example

```
[bright90]% category add misc
[bright90->category*[misc*]]% roles
[bright90->category*[misc*]->roles]% assign provisioning
[bright90...*]->roles*[provisioning*]]% set allimages no
[bright90...*]->roles*[provisioning*]]% set localimages default-image
[bright90...*]->roles*[provisioning*]]% set provisioningslots 20
[bright90...*]->roles*[provisioning*]]% show
Parameter                               Value
-----
All Images                               no
Include revisions of local images        yes
Local images                             default-image
Name                                      provisioning
Nodegroups
Provisioning associations                 <0 internally used>
Revision
Shared images
Type                                      ProvisioningRole
Provisioning slots                        20
[bright90->category*[misc*]->roles*[provisioning*]]% commit
[bright90->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

```
[bright90]% device use node001
[bright90->device[node001]]% roles
[bright90->device[node001]->roles]% assign provisioning
[bright90->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 Bright View

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

A `misc` category can be added via the clickpath

Grouping→Categories→Add→Settings→<name>

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

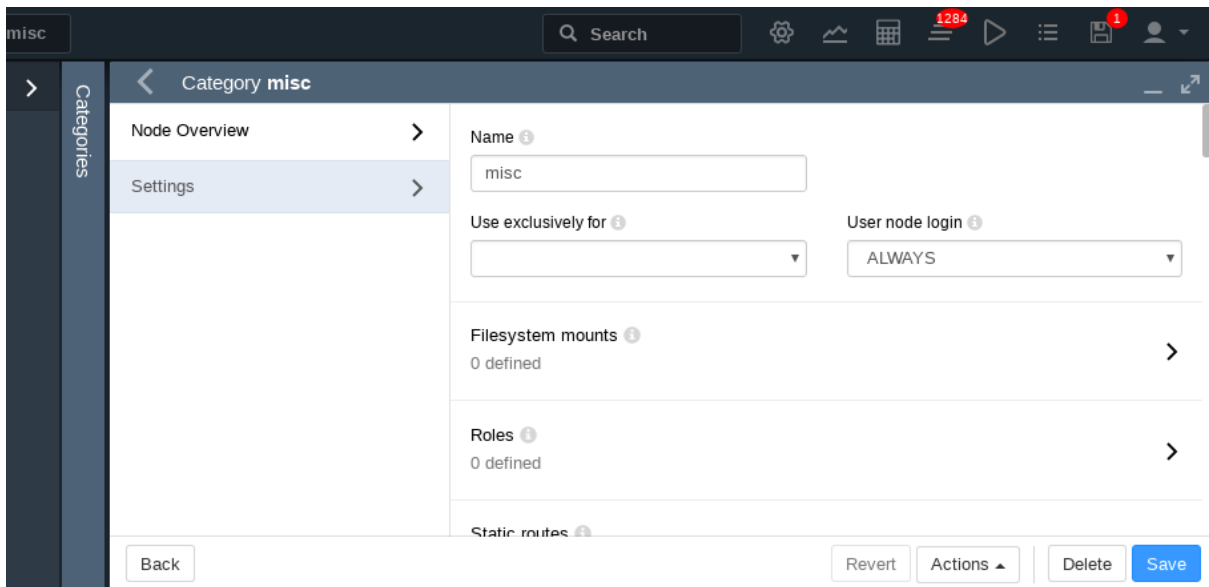


Figure 5.3: Bright View: Adding A `misc` Category

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

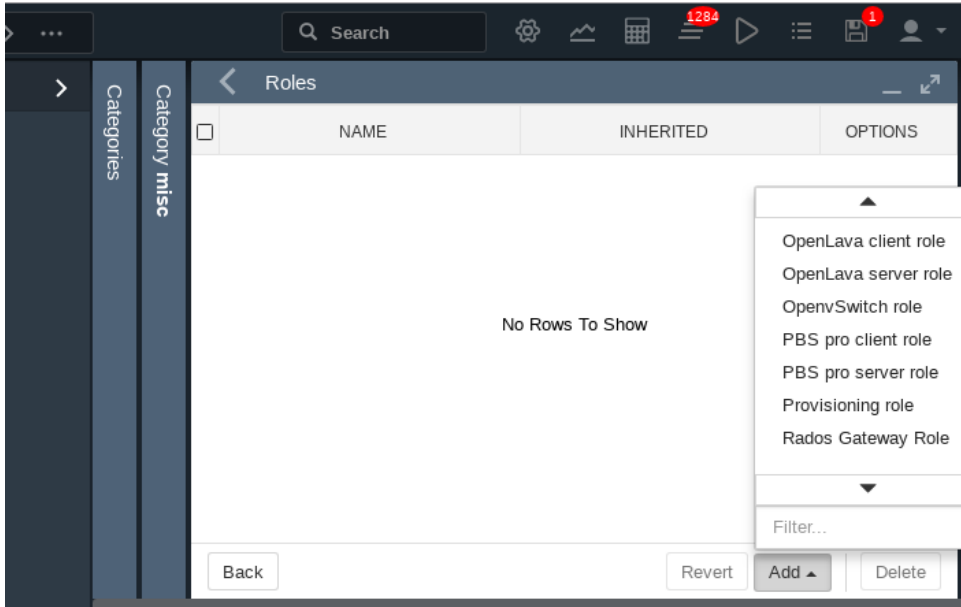


Figure 5.4: Bright View: Setting A provisioning Role

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

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

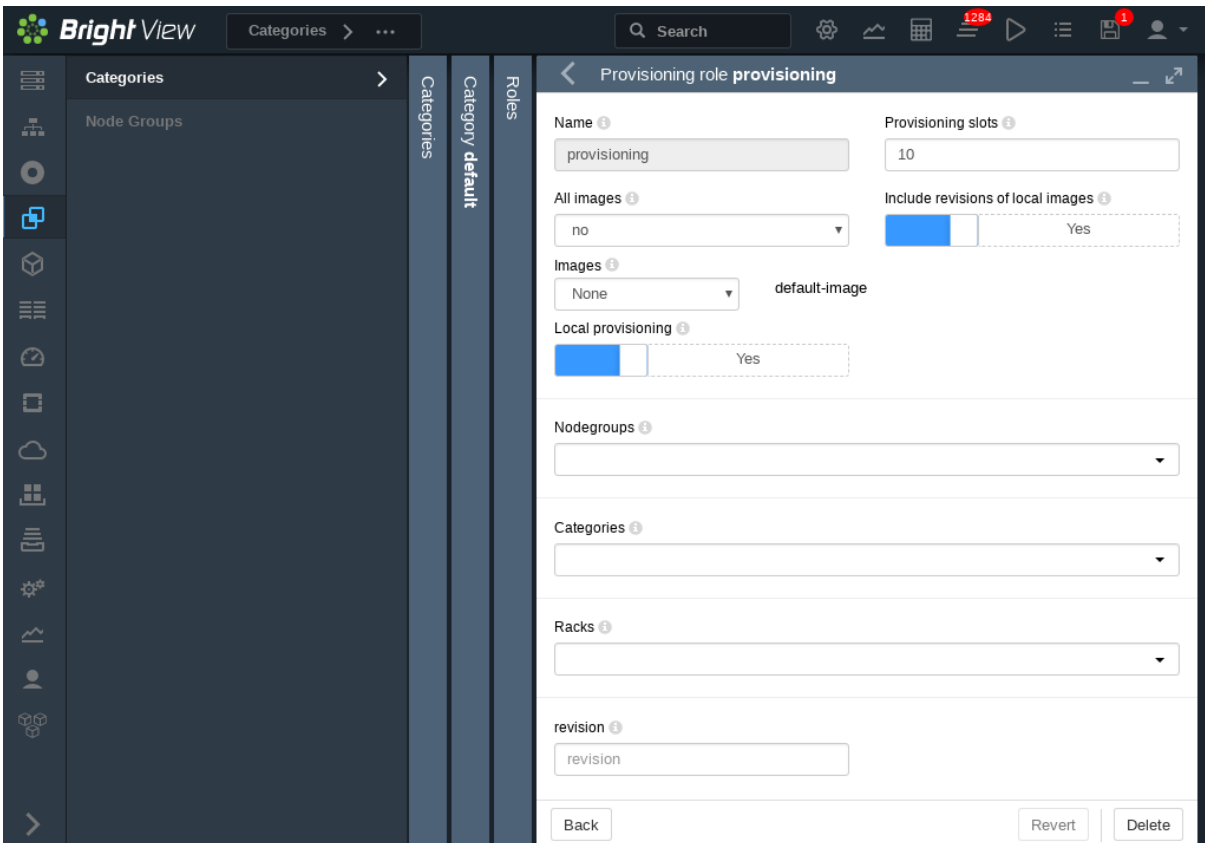


Figure 5.5: Bright View: Configuring A provisioning Role

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

The images offered by the provisioning role should not be confused with the software image setting of the `misc` category itself, which is the image the provisioning node requests for itself from the category.

5.2.4 Provisioning Nodes: Housekeeping

The head node does housekeeping tasks for the entire provisioning system. Provisioning is done on request for all non-head nodes on a first-come, first-serve basis. Since provisioning nodes themselves, too, need to be provisioned, it means that to cold boot an entire cluster up quickest, the head node should be booted and be up first, followed by provisioning nodes, and finally by all other non-head nodes. Following this start-up sequence ensures that all provisioning services are available when the other non-head nodes are started up.

Some aspects of provisioning housekeeping are discussed next:

Provisioning Node Selection

When a node requests provisioning, the head node allocates the task to a provisioning node. If there are several provisioning nodes that can provide the image required, then the task is allocated to the provisioning node with the lowest number of already-started provisioning tasks.

Limiting Provisioning Tasks With `MaxNumberOfProvisioningThreads`

Besides limiting how much simultaneous provisioning per provisioning node is allowed with `Provisioning Slots` (section 5.2.1), the head node also limits how many simultaneous provisioning tasks are allowed to run on the entire cluster. This is set using the `MaxNumberOfProvisioningThreads` directive in the head node's `CMDaemon` configuration file, `/etc/cmd.conf`, as described in Appendix C.

Provisioning Tasks Deferral and Failure

A provisioning request is *deferred* if the head node is not able to immediately allocate a provisioning node for the task. Whenever an ongoing provisioning task has finished, the head node tries to re-allocate deferred requests.

A provisioning request *fails* if an image is not transferred. 5 retry attempts at provisioning the image are made in case a provisioning request fails.

A provisioning node that is carrying out requests, and which loses connectivity, has its provisioning requests remain allocated to it for 180 seconds from the time that connectivity was lost. After this time the provisioning requests fail.

Provisioning Role Change Notification With `updateprovisioners`

The `updateprovisioners` command can be accessed from the `softwareimage` mode in `cmsh`. It can also be accessed from Bright View, via the clickpath `Provisioning`→`Provisioning requests`→`Update provisioning nodes`.

In the examples in section 5.2.2, changes were made to provisioning role attributes for an individual node as well as for a category of nodes. This automatically ran the `updateprovisioners` command.

The `updateprovisioners` command runs automatically if `CMDaemon` is involved during software image changes or during a provisioning request. If on the other hand, the software image is changed outside of the `CMDaemon` front ends (Bright View and `cmsh`), for example by an administrator adding a file by copying it into place from the bash prompt, then `updateprovisioners` should be run manually to update the provisioners.

In any case, if it is not run manually, it is scheduled to run every midnight by default.

When the default `updateprovisioners` is invoked manually, the provisioning system waits for all running provisioning tasks to end, and then updates all images located on any provisioning nodes by using the images on the head node. It also re-initializes its internal state with the updated provisioning role properties, i.e. keeps track of what nodes are provisioning nodes.

The default `updateprovisioners` command, run with no options, updates all images. If run from `cmsh` with a specified image as an option, then the command only does the updates for that particular image. A provisioning node undergoing an image update does not provision other nodes until the update is completed.

Example

```
[bright90]% softwareimage updateprovisioners
Provisioning nodes will be updated in the background.

Sun Dec 12 13:45:09 2010 bright90: Starting update of software image(s) \
provisioning node(s). (user initiated).
[bright90]% softwareimage updateprovisioners [bright90]%
Sun Dec 12 13:45:41 2010 bright90: Updating image default-image on prov \
isioning node node001.
[bright90]%
Sun Dec 12 13:46:00 2010 bright90: Updating image default-image on prov \
isioning node node001 completed.
Sun Dec 12 13:46:00 2010 bright90: Provisioning node node001 was updated
Sun Dec 12 13:46:00 2010 bright90: Finished updating software image(s) \
on provisioning node(s).
```

Provisioning Role Draining And Undraining Nodes With `drain`, `undrain`

The `drain` and `undrain` commands to control provisioning nodes are accessible from within the `softwareimage` mode of `cmsh`.

If a node is put into a drain state, then all currently active provisioning requests continue until they are completed. However, the node is not assigned any further pending requests, until the node is put back into an undrain state.

Example

```
[bright90->softwareimage]% drain -n master
Nodes drained
[bright90->softwareimage]% provisioningstatus
Provisioning subsystem status
Pending request:      node001, node002
Provisioning node status:
+ bright90
  Slots:              1 / 10
  State:              draining
  Active nodes:      node003
  Up to date images:  default-image
[bright90->softwareimage]% provisioningstatus
Provisioning subsystem status
Pending request:      node001, node002
Provisioning node status:
+ bright90
  Slots:              0 / 10
  State:              drained
  Active nodes:      none
  Up to date images:  default-image
```

To drain all nodes at once, the `--role` option can be used, with `provisioning role` as its value. All pending requests then remain in the queue, until the nodes are undrained again.

Example


```
[bright90->softwareimage]% drain --role provisioning
...Time passes. Pending
  requests stay in the queue. Then
  admin undrains it...
[bright90->softwareimage]% undrain --role provisioning
```

Provisioning Node Update Safeguards And provisioningnodeauto-updatetimeout

The `updateprovisioners` command is subject to safeguards that prevent it running too frequently. The minimum period between provisioning updates can be adjusted with the parameter `provisioningnodeauto-updatetimeout`, which has a default value of 300s.

Exceeding the timeout does not by itself trigger an update to the provisioning node.

When the head node receives a provisioning request, it checks if the last update of the provisioning nodes is more than the timeout period. If true, then an update is triggered to the provisioning node. The update is disabled if the timeout is set to zero (`false`).

The parameter can be accessed and set within `cmsh` from `partition` mode:

Example

```
[root@bright90 ]# cmsh
[bright90]% partition use base
[bright90->partition[base]]% get provisioningnodeauto-updatetimeout
[bright90->partition[base]]% 300
[bright90->partition[base]]% set provisioningnodeauto-updatetimeout 0
[bright90->partition*[base*]]% commit
```

Within Bright View the parameter is accessible via the clickpath:

Cluster → Partition[base] → Provisioning Node Auto Update Timeout.

To prevent provisioning an image to the nodes, it can be locked (section 5.4.7). The provisioning request is then deferred until the image is once more unlocked.

Synchronization Of Fspart Subdirectories To Provisioning Nodes

In Bright Cluster Manager, an *fspart* is a subdirectory, and it is a filesystem part that can be synced during provisioning.

The list of `fsparts` can be listed with:

Example

```
[root@bright90 ]# cmsh
[bright90]% fspart
[bright90->fspart]% list
```

Path (key)	Type	Image
/cm/images/default-image	image	default-image
/cm/images/default-image/boot	boot	default-image:boot
/cm/node-installer	node-installer	
/cm/shared	cm-shared	
/tftpboot	tftpboot	
/var/spool/cmd/monitoring	monitoring	

The `updateprovisioners` command (page 157) is used to update image `fsparts` to all nodes with a provisioning role.

The trigger command: is used to update non-image `fsparts` to off-premises nodes, such as cloud directors and edge directors. The directors have a provisioning role for the nodes that they direct.

All of the non-image types can be updated with the `--all` option:

Example

```
[bright90->fspart]% trigger --all
```

The command `help trigger` in `fspart` mode gives further details.

The `info` command: shows the architecture, OS, and the number of inotify watchers that track rsyncs in the `fspart` subdirectory.

```
[bright90->fspart]% info
```

Path	Architecture	OS	Inotify watchers
/cm/images/default-image	x86_64	rhel7	0
/cm/images/default-image/boot	-	-	0
/cm/node-installer	x86_64	rhel7	0
/cm/shared	x86_64	rhel7	0
/tftpboot	-	-	0
/var/spool/cmd/monitoring	-	-	0


```
[bright90->fspart]% info -s
```

(!#with size, takes longer)

Path	Architecture	OS	Inotify watchers	Size
/cm/images/default-image	x86_64	rhel7	0	4.2 GiB
/cm/images/default-image/boot	-	-	0	179 MiB
/cm/node-installer	x86_64	rhel7	0	2.45 GiB
/cm/shared	x86_64	rhel7	0	2.49 GiB
/tftpboot	-	-	0	3.3 MiB
/var/spool/cmd/monitoring	-	-	0	1.02 GiB

The `locked`, `lock`, and `unlock` commands:

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

Example

```
[bright90->fspart]% locked
No locked fsparts
```

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

Example

```
[bright90->fspart]% lock /var/spool/cmd/monitoring
[bright90->fspart]% locked
/var/spool/cmd/monitoring
```

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

Example

```
[bright90->fspart]% unlock /var/spool/cmd/monitoring
[bright90->fspart]% locked
No locked fsparts
```

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

Example

```
[bright90->fspart]% excludelistsnippets /tftpboot

[bright90->fspart[/tftpboot]->excludelistsnippets]% list
Name (key)  Lines  Disabled  Mode sync  Mode full  Mode update  Mode grab  Mode grab new
-----
Default    2      no        yes        yes        yes         no        no

[bright90->fspart[/tftpboot]->excludelistsnippets]% show default
Parameter                                     Value
-----
Lines                                         2
Name                                          Default
Revision
Exclude list                                # no need for rescue on nodes with a boot role,/rescue,/rescue/*
Disabled                                     no
No new files                                no
Mode sync                                   yes
Mode full                                   yes
Mode update                                 yes
Mode grab                                   no
Mode grab new                               no

[bright90->fspart[/tftpboot]->excludelistsnippets]% get default excludelist
# no need for rescue on nodes with a boot role
/rescue
/rescue/*
```

5.3 The Kernel Image, Ramdisk And Kernel Modules

A *software image* is a complete Linux filesystem that is to be installed on a non-head node. Chapter 11 describes images and their management in detail.

The head node holds the head copy of the software images. Whenever files in the head copy are changed using CMDaemon, the changes automatically propagate to all provisioning nodes via the updateprovisioners command (section 5.2.4).

5.3.1 Booting To A “Good State” Software Image

When nodes boot from the network in simple clusters, the head node supplies them with a *known good state* during node start up. The known good state is maintained by the administrator and is defined using a software image that is kept in a directory of the filesystem on the head node. Supplementary filesystems such as /home are served via NFS from the head node by default.

For a diskless node the known good state is copied over from the head node, after which the node becomes available to cluster users.

For a disked node, by default, the hard disk contents on specified local directories of the node are checked against the known good state on the head node. Content that differs on the node is changed to that of the known good state. After the changes are done, the node becomes available to cluster users.

Each software image contains a Linux kernel and a ramdisk. These are the first parts of the image that are loaded onto a node during early boot. The kernel is loaded first. The ramdisk is loaded next, and contains driver modules for the node’s network card and local storage. The rest of the image is loaded after that, during the node-installer stage (section 5.4).

5.3.2 Selecting Kernel Driver Modules To Load Onto Nodes

Kernel modules can be managed in `softwareimage` mode (using an image), in `category` mode (using a category), or in `device` mode (using a node), as indicated by the following `cmsh` tree view of a newly-installed cluster with default values:

```
cmsh
|-- category[default]
|   |-- kernelmodules
|   ...
|-- device] [node001]
|   |-- kernelmodules
|   ...
|-- softwareimage[default-image]
|   |-- kernelmodules
|   ...
```

As is usual in Bright Cluster Manager, if there are values specified at the lower levels in the hierarchy, then their values override the values set higher up in the hierarchy. For example, modules specified at node level override modules specified at category or software image level. Similarly modules specified at the category level override whatever is specified at the software image level. The cluster administrators should be aware that “override” for kernel modules appended to a kernel image means that any kernel modules defined at a higher level are totally ignored—the modules from a lower level exclude the modules at the higher level. A misconfiguration of kernel modules in the lower levels can thus prevent the node from starting up.

Modules are normally just set at `softwareimage` level, using `cmsh` or Bright View.

Kernel Driver Modules With `cmsh`

In `cmsh`, the modules that are to go on the ramdisk can be placed using the `kernelmodules` submode of the `softwareimage` mode. The order in which they are listed is the attempted load order.

Within the `kernelmodules` submode, the `import` command can be used to import the kernel modules list from a software image, from a node, or from a category, replacing the original kernel modules list.

Whenever a change is made via the `kernelmodules` submode to the kernel module selection of a software image, `CMDaemon` automatically runs the `createramdisk` command. The `createramdisk` command regenerates the ramdisk inside the `initrd` image and sends the updated image to all provisioning nodes, to the image directory, set by default to `/cm/images/default-image/boot/`. The original `initrd` image is saved as a file with suffix `.orig` in that directory. An attempt is made to generate the image for all software images that `CMDaemon` is aware of, regardless of category assignment, unless the image is protected from modification by `CMDaemon` with a `FrozenFile` directive (Appendix C).

The `createramdisk` command can also be run manually from within the `softwareimage` mode.

Kernel Driver Modules With Bright View

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

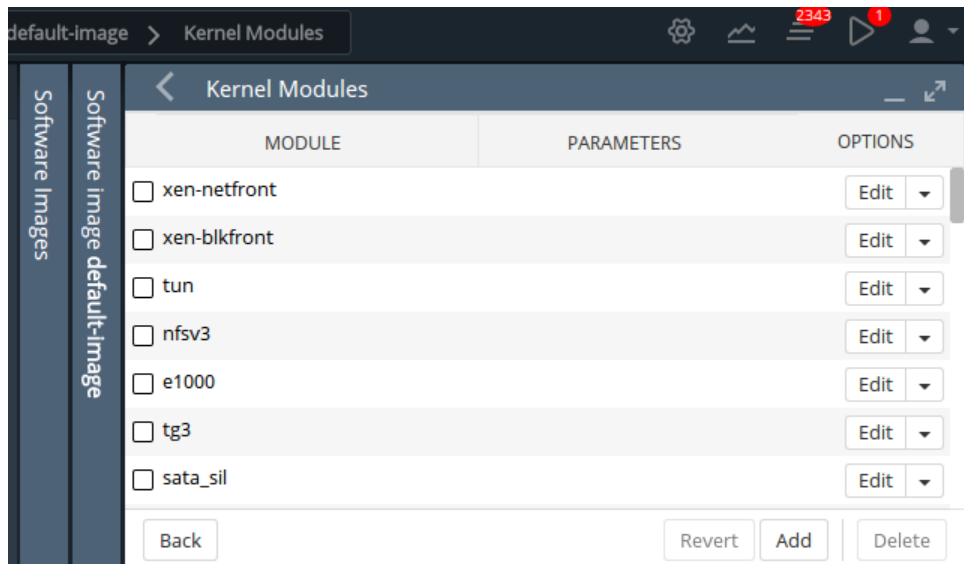


Figure 5.6: Bright View: Selecting Kernel Modules For Software Images

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

Manually Regenerating A Ramdisk

Regenerating a ramdisk manually via `cmsh` or Bright View is useful if the kernel or modules have changed without using `CMDaemon`. For example, after running a YUM update which has modified the kernel or modules of the nodes (section 11.3). 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 11.3.4.

5.3.3 InfiniBand Provisioning

On clusters that have InfiniBand hardware, it is normally used for data transfer as a service after the nodes have fully booted up (section 3.6). It can also be used for PXE booting (section 5.1.3) and for node provisioning (described here), but these are not normally a requirement. This section (about InfiniBand node provisioning) may therefore safely be skipped in almost all cases when first configuring a cluster.

During node start-up on a setup for which InfiniBand networking has been enabled, the `init` process runs the `rdma` script. For SLES the `openib` script is used instead of the `rdma` script. The script loads up InfiniBand modules into the kernel. When the cluster is finally fully up and running, the use of InfiniBand is thus available for all processes that request it.

Provisioning nodes over InfiniBand is not implemented by default, because the `init` process, which handles initialization scripts and daemons, takes place only after the node-provisioning stage launches. InfiniBand modules are therefore not available for use during provisioning, which is why, for default kernels, provisioning in 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@bright90 ~]# { service rdma stop; lsmod | cut -f1 -d" "; }>/tmp/a
[root@bright90 ~]# { service rdma start; lsmod | cut -f1 -d" "; }>/tmp/b
```

The `rdma` service in the two lines should be replaced by `openibd` service instead when using SLES, or distributions based on versions of Red Hat prior to version 6.

The first line stops the InfiniBand service, just in case it is running, in order to unload its modules, and then lists the modules on the node.

The second line starts the service, so that the appropriate modules are loaded, and then lists the modules on the node again. The output of the first step is stored in a file `a`, and the output from the second step is stored in a file `b`.

Running `diff` on the output of these two steps then reveals the modules that get loaded. For `rdma`, the output may display something like:

Example

```
[root@bright90 ~]# diff /tmp/a /tmp/b
1,3c1
< Unloading OpenIB kernel modules:
< Failed to unload ib_core
<
<                                     [FAILED]
---
> Loading OpenIB kernel modules:
>                                     [ OK ]
4a3,14
> ib_ipoib
> rdma_ucm
> ib_ucm
> ib_uverbs
> ib_umad
> rdma_cm
> ib_cm
> iw_cm
> ib_addr
> ib_sa
> ib_mad
> ib_core
```

As suggested by the output, the modules `ib_ipoib`, `rdma_ucm` and so on are the modules loaded when `rdma` starts, and are therefore the modules that are needed for this particular HCA. Other HCAs may cause different modules to be loaded.

For a default Red Hat from version 7 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@bright90 ~]# cmsg
[bright90]% softwareimage use default-image
[bright90->softwareimage[default-image]]% kernelmodules
[bright90...age[default-image]->kernelmodules]% add mlx4_ib
[bright90...age*[default-image*]->kernelmodules*[mlx4_ib*]]% add ib_ipoib
[bright90...age*[default-image*]->kernelmodules*[ib_ipoib*]]% add ib_umad
[bright90...age*[default-image*]->kernelmodules*[ib_umad*]]% commit
[bright90->softwareimage[default-image]->kernelmodules[ib_umad]]%
Tue May 24 03:45:35 2011 bright90: Initial ramdisk for image default-im\
age was regenerated successfully.
```

If the modules are put in another image instead of default-image, then the default image that nodes boot from should be set to the new image (section 3.15.2).

InfiniBand Provisioning: Network Configuration

It is assumed that the networking configuration for the final system for InfiniBand is configured following the general guidelines of section 3.6. If it is not, that should be checked first to see if all is well with the InfiniBand network.

The provisioning aspect is set by defining the provisioning interface. An example of how it may be set up for 150 nodes with a working InfiniBand interface ib0 in cmsg is:

Example

```
[root@bright90~]# cmsg
[bright90]% device
[bright90->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[bright90->device*]% commit
```

5.3.4 Omni-Path Provisioning

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

Omni-Path Provisioning: Ramdisk Image Configuration

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

Example

```
[root@bright90 ~]# cmsg
[bright90]% softwareimage use default-image
[bright90->softwareimage[default-image]]% kernelmodules
[bright90...age[default-image]->kernelmodules]% add hfi1
[bright90...age*[default-image*]->kernelmodules*[hfi1*]]% add ib_ipoib
[bright90...age*[default-image*]->kernelmodules*[ib_ipoib*]]% commit
[bright90->softwareimage[default-image]->kernelmodules[ib_ipoib]]%
Thu Jul 4 15:03:35 2019 bright90: Initial ramdisk for image default-im\
age was regenerated successfully.
```

Omni-Path Provisioning: Network Configuration

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

Example

```
[root@bright90~]# cmsh
%[bright90]% device
[bright90->device]% foreach -n node001..node150 (set provisioninginterface ib0)
[bright90->device*]% commit
```

5.3.5 VLAN Provisioning

Nodes can be configured for provisioning over a VLAN interface, starting in Bright Cluster Manager version 8.2.

This requires:

- A VLAN network and node interface. The VLAN network is typically specified by the network switch. The interface that connects the node to the switch can be configured as a VLAN interface as outlined in section 3.4.
- The BIOS of the node must have the `VLANID` value set within the BIOS network options. If the BIOS does not support this setting, then PXE over VLAN cannot work. For a NIC that is missing this in the BIOS, the NIC hardware provider may sometimes have a BIOS update that supports this setting.
- The `VLANID` value should be set in the kernel parameters. Kernel parameters for a node can be specified in `cmsh` with the `kernelparameters` setting of `softwareimage` mode (section 11.3.4). An example where the `VLANID` is appended to some existing parameters could be:

Example

```
[root@bright90 ~]# cmsh
[bright90]% softwareimage
[bright90]->softwareimage% use default-image
[bright90->softwareimage[default-image]]% append kernelparameters " VLANID=89"
[bright90->softwareimage*[default-image*]]% commit
```

5.4 Node-Installer

After the kernel has started up, and the ramdisk kernel modules are in place on the node, the node launches the node-installer.

The node-installer is a software image (section 11.4.4) provided by the head node. It interacts with `CMDaemon` on the head node and takes care of the rest of the boot process.

As an aside, the node-installer modifies some files (Appendix A.3) on the node it is installing to, so that they differ from the otherwise-expected pre-init stage Linux system. Such modifications can be prevented by a `frozenFilesPerNode` or `frozenFilesPerCategory` directive, as documented within the `node-installer.conf` file, and explained in greater detail on page 715.

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:

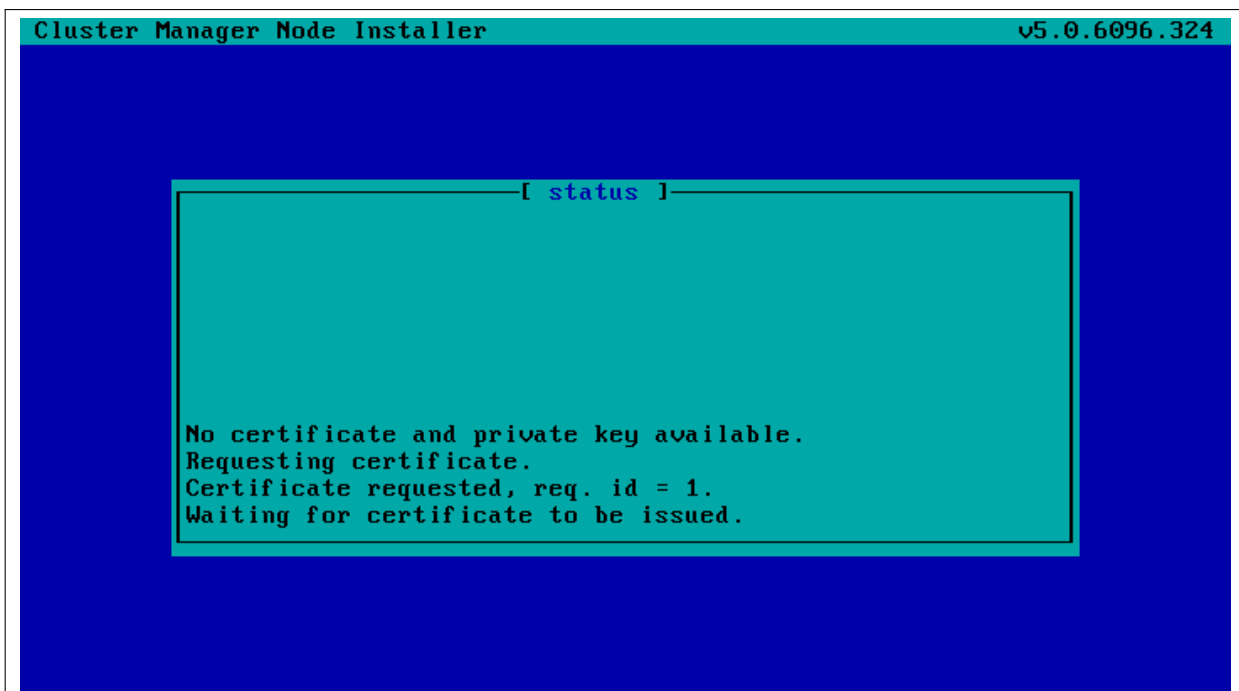
1. requesting a node certificate (section 5.4.1)
2. deciding or selecting node configuration (section 5.4.2)

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).



```
Cluster Manager Node Installer v5.0.6096.324
[ status ]
No certificate and private key available.
Requesting certificate.
Certificate requested, req. id = 1.
Waiting for certificate to be issued.
```

Figure 5.7: Certificate Request

The certificate is stored on the head node in `/cm/node-installer/certificates/` by MAC address.

Certificate Auto-signing

Certificate auto-signing means the cluster management daemon automatically signs a certificate signing request (CSR) that has been requested by a node. Certificate auto-signing can be configured from within partition mode of `cmsh`, with the `signinstallercertificates` parameter. It can take one of the following values:

- AUTO (the default)
- MANUAL

Example

```
[root@bright90 ~]# cmsh
[bright90]% partition
[bright90->partition[base]]% set signinstallercertificates auto
```

For untrusted networks, it may be wiser to approve certificate requests manually to prevent new nodes being added automatically without getting noticed.

Disabling certificate auto-signing for all networks can be done by setting `signinstallercertificates` to `MANUAL`.

Instead of disabling certificate autosigning for all networks, a finer tuning can be carried out for individual networks. This requires that `signinstallercertificates` be set to `AUTO` in partition mode. The `allowautosign` parameter in network mode can then be set for a particular network, and it can take one of the the following values:

- Always
- Automatic (the default)
- Never
- Secret

Example

```
[root@bright90 ~]# cmsh
[bright90]% network use internalnet
[bright90->network[internalnet]]% set allowautosign automatic[TAB] [TAB]
always      automatic never      secret
```

If `Always` is set, then incoming CSRs from all types of networks are automatically auto-signed.

If `Automatic` is set, then only networks that are of type `internal` are automatically auto-signed.

If `Never` is set, then all incoming CSRs that come in for that network have to be manually approved.

The value `Secret` is required for the `globalnet` network, for edge sites. A node on an edge site uses a shared secret that is passed along with the node request. The secret is set during edge site setup by Bright Cluster Manager (section 2.1.1 of the *Edge Manual*).

Manual Approval Of A CSR

Approval of the CSR from a regular node (not an edge node): Manual approval of a CSR is typically done from within `certs` mode. A list of requests can be found, and from the list, the appropriate unsigned request can be signed and issued. The following session illustrates the process:

```
[bright90->cert]% listrequests
Request ID  Client type  Session ID  Autosign Name
-----
6           installer   42949672986 No          fa-16-3e-22-cd-13
[bright90->cert]% issuecertificate 6
Issued 6
```

Approval of the CSR from an edge node: If the shared secret has not been set for the edge director—that is, if it has not been stored locally on the edge director, or if it has not been passed on via the installation medium— then the node-installer prompts for the secret the first time that it boots. If the secret that is typed in matches the site secret, then a CSR from the edge director is handled by the head node, and a signed certificate is issued.

The edge compute nodes pick up their secret from the director. If the director does not have the secret, then the compute node's node-installer prompts for the secret on first boot. Once the secret is set, then edge compute node sends its CSR to the head node (via the edge director) and gets a signed certificate automatically.

Section 2.3 has more information on certificate management in general.

Certificate Storage And Removal Implications

After receiving a valid certificate, the node-installer stores it in `/cm/node-installer/certificates/<node mac address>/` on the head node. This directory is NFS exported to the nodes, but can only be accessed by the root user. The node-installer does not request a new certificate if it finds a certificate in this directory, valid or invalid.

If an invalid certificate is received, the screen displays a communication error. Removing the node's corresponding certificate directory allows the node-installer to request a new certificate and proceed further.

5.4.2 Deciding Or Selecting Node Configuration

Once communication with the head node CMDaemon is established, the node-installer tries to identify the node it is running on so that it can select a configuration from CMDaemon's record for it, if any such record exists. It correlates any node configuration the node is expected to have according to network hardware detected. If there are issues during this correlation process then the administrator is prompted to select a node configuration until all nodes finally have a configuration.

Possible Node Configuration Scenarios

The correlations process and corresponding scenarios are now covered in more detail:

It starts with the node-installer sending a query to CMDaemon to check if the MAC address used for net booting the node is already associated with a node in the records of CMDaemon. In particular, it checks the MAC address for a match against the existing *node configuration* properties, and decides whether the node is *known* or *new*.

- the node is **known** if the query matches a node configuration. It means that node has been booted before.
- the node is **new** if no configuration is found.

In both cases the node-installer then asks CMDaemon to find out if the node is connected to an Ethernet switch, and if so, to which port. Setting up Ethernet switches for port detection is covered in section 3.8.

If a port is detected for the node, the node-installer queries CMDaemon for a node configuration associated with the detected Ethernet switch port. If a port is not detected for the node, then either the hardware involved with port detection needs checking, or a node configuration must be selected manually.

There are thus several scenarios:

1. The node is new, and an Ethernet switch port is detected. A node configuration associated with the port is found. The node-installer suggests to the administrator that the new node should use this configuration, and displays the configuration along with a confirmation dialog (figure 5.8).

This suggestion can be interrupted, and other node configurations can be selected manually instead through a sub-dialog (figure 5.9). By default (in the main dialog), the original suggestion is accepted after a timeout.

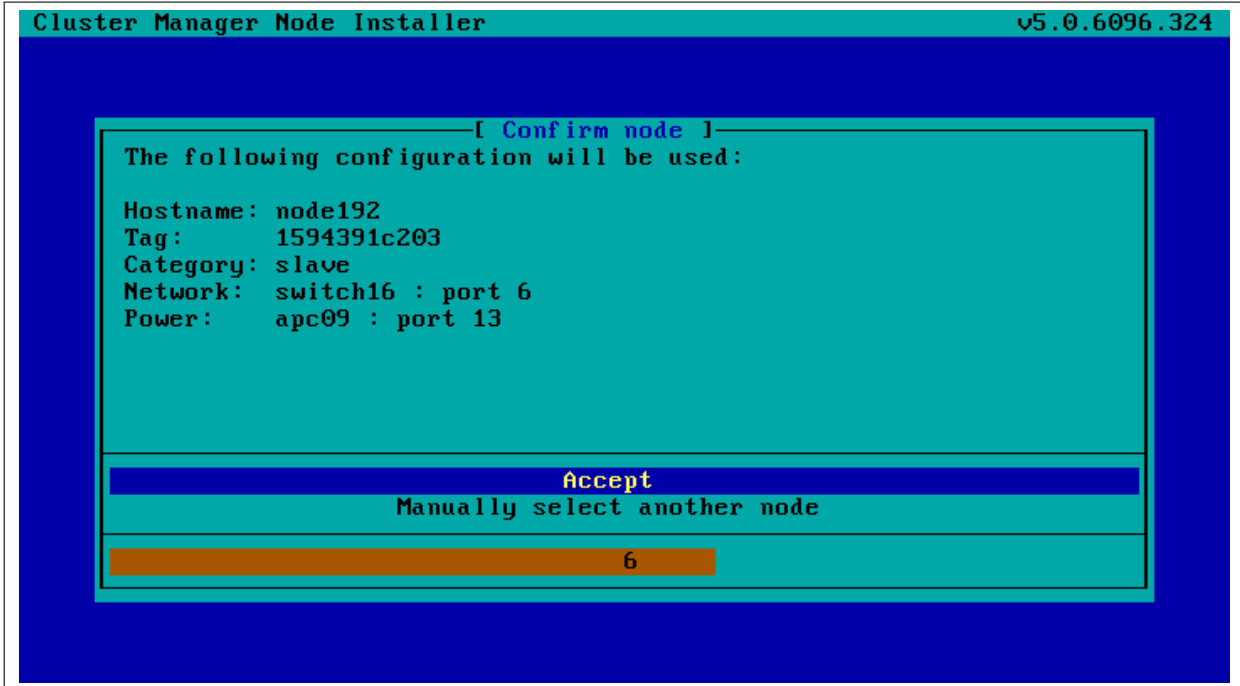


Figure 5.8: Scenarios: Configuration Found, Confirm Node Configuration

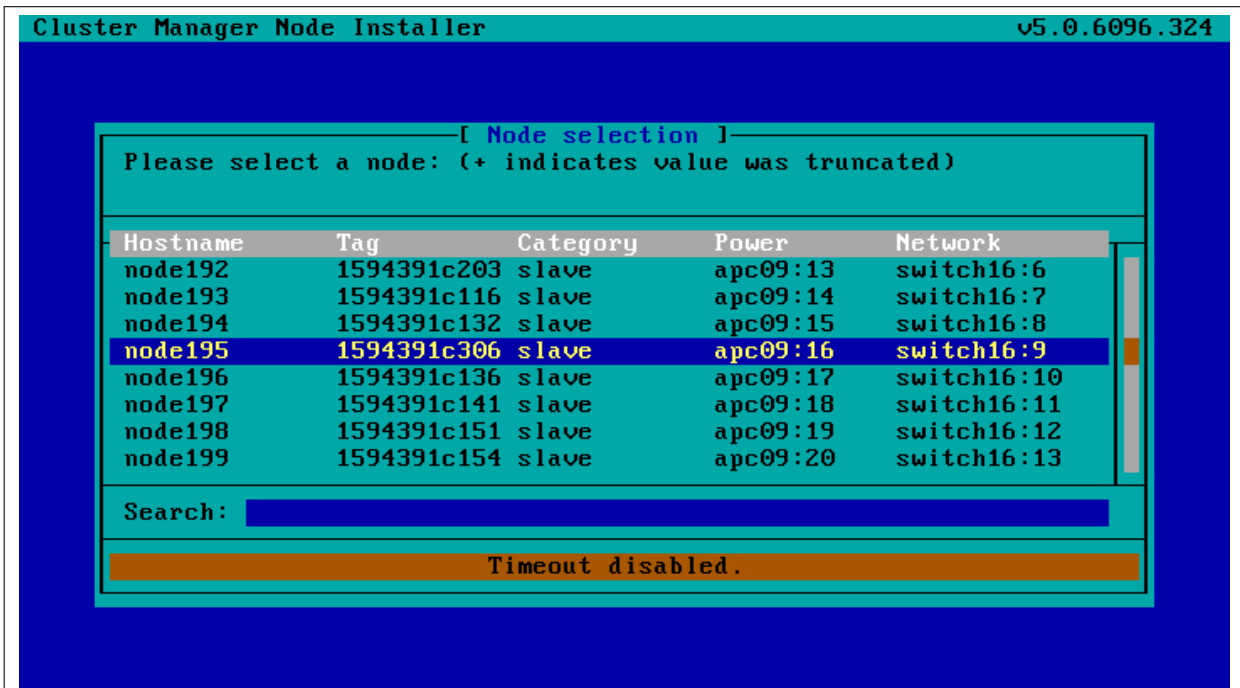


Figure 5.9: Scenarios: Node Selection Sub-Dialog

2. The node is new, and an Ethernet switch port is detected. A node configuration associated with

the port is not found. The node-installer then displays a dialog that allows the administrator to either retry Ethernet switch port detection (figure 5.10) or to drop into a sub-dialog to manually select a node configuration (figure 5.9). By default, port detection is retried after a timeout.

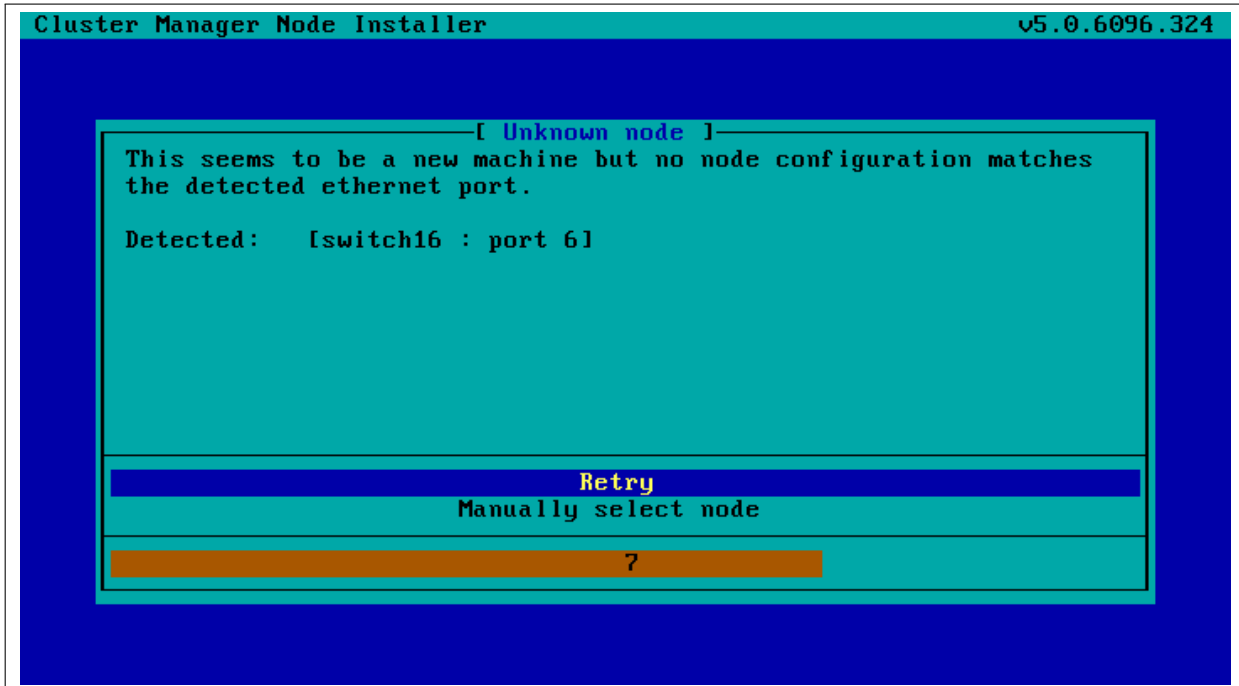


Figure 5.10: Scenarios: Unknown Node, Ethernet Port Detected

3. The node is new, and an Ethernet switch port is not detected. The node-installer then displays a dialog that allows the user to either retry Ethernet switch port detection (figure 5.11) or to drop into a sub-dialog to manually select a node configuration (figure 5.9). By default, port detection is retried after a timeout.

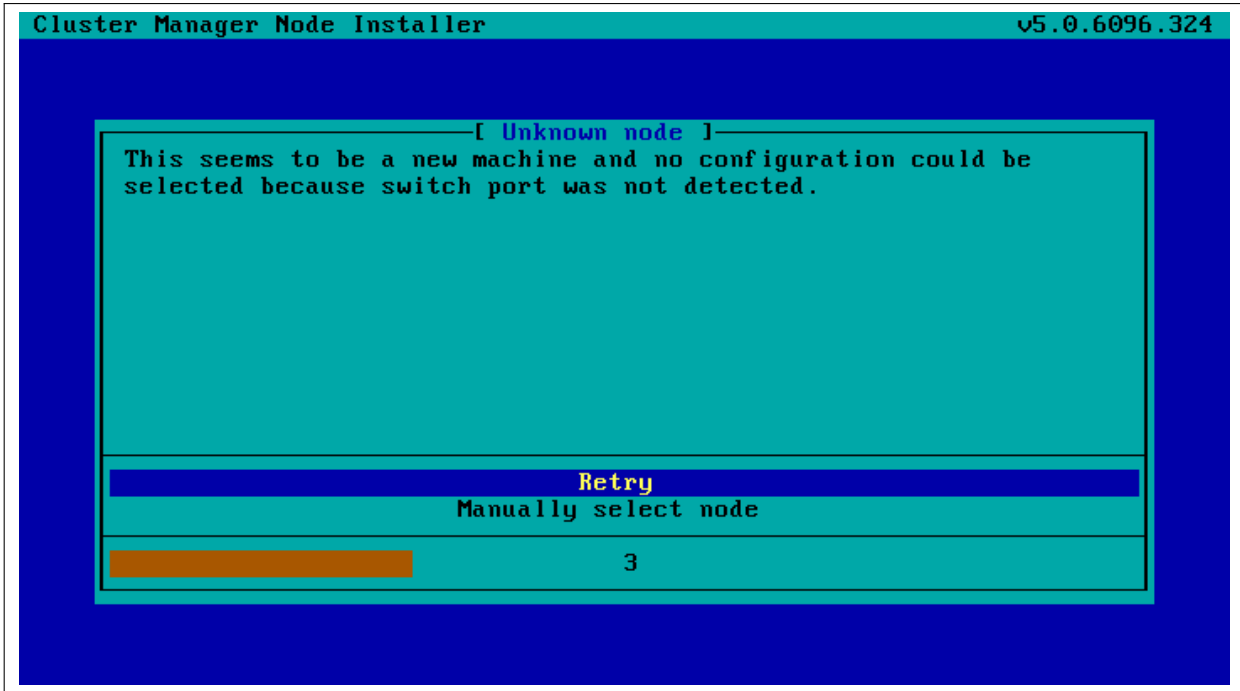


Figure 5.11: Scenarios: Unknown Node, No Ethernet Port Detected

4. The node is known, and an Ethernet switch port is detected. The configuration associated with the port is the same as the configuration associated with the node's MAC address. The node-installer then displays the configuration as a suggestion along with a confirmation dialog (figure 5.8). The suggestion can be interrupted, and other node configurations can be selected manually instead through a sub-dialog (figure 5.9). By default (in the main dialog), the original suggestion is accepted after a timeout.
5. The node is known, and an Ethernet switch port is detected. However, the configuration associated with the port is not the same as the configuration associated with the node's MAC address. This is called a *port mismatch*. This type of port mismatch situation occurs typically during a mistaken *node swap*, when two nodes are taken out of the cluster and returned, but their positions are swapped by mistake (or equivalently, they are returned to the correct place in the cluster, but the switch ports they connect to are swapped by mistake). To prevent configuration mistakes, the node-installer displays a port mismatch dialog (figure 5.12) allowing the user to retry, accept a node configuration that is associated with the detected Ethernet port, or to manually select another node configuration via a sub-dialog (figure 5.9). By default (in the main port mismatch dialog), port detection is retried after a timeout.

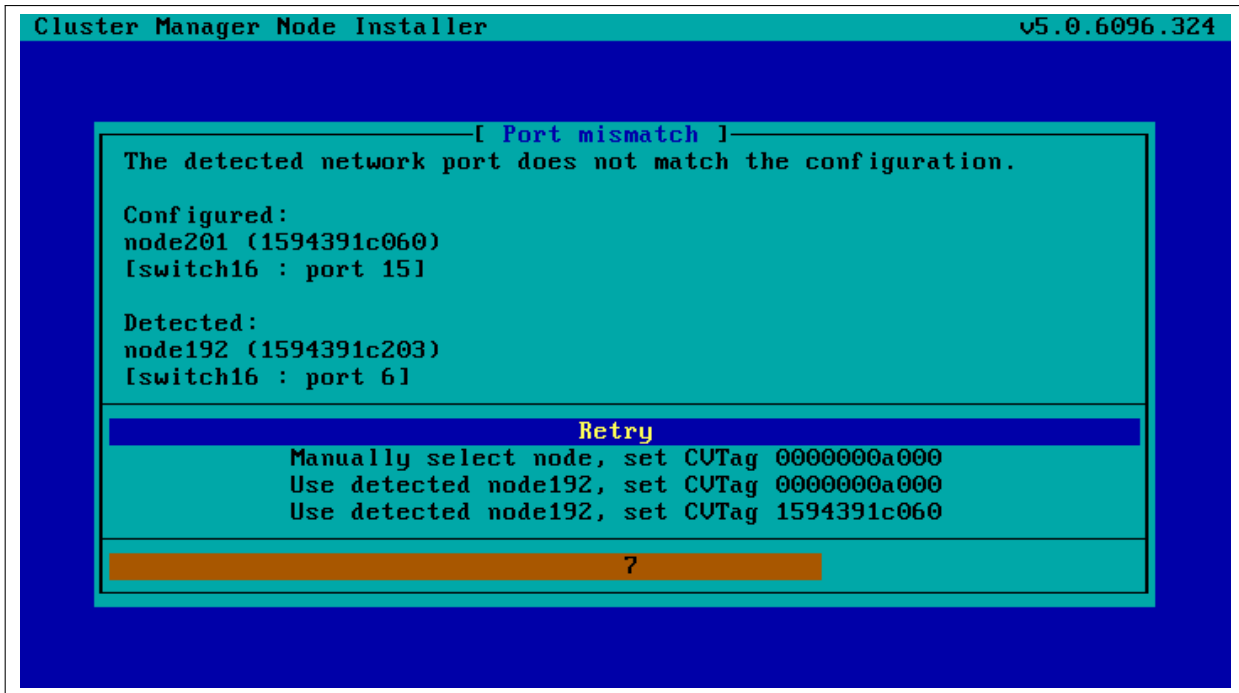


Figure 5.12: Scenarios: Port Mismatch Dialog

6. The node is known, and an Ethernet switch port is not detected. However, the configuration associated with the node's MAC address does have an Ethernet port associated with it. This is also considered a port mismatch. To prevent configuration mistakes, the node-installer displays a port mismatch dialog similar to figure 5.12, allowing the user to retry or to drop into a sub-dialog and manually select a node configuration that may work.

However, a more likely solution in most cases is to:

- either clear the switch port configuration in the cluster manager so that switch port detection is not attempted. For example, for node001, this can be done by running this `cmsh` command on the head node:
`cmsh -c "device clear node001 ethernetswitch; commit"`
- or enable switch port detection on the switch. This is usually quite straightforward, but may require going through the manuals or software application that the switch manufacturer has provided.

By default (in the port mismatch dialog), port detection is retried after a timeout. This means that if the administrator clears the switch port configuration or enables switch port detection, the node-installer is able to continue automatically with a consistent configuration.

7. The node is known, and an Ethernet switch port is detected. However, the configuration associated with the node's MAC address has no Ethernet switch port associated with it. This is not considered a port mismatch but an unset switch port configuration, and it typically occurs if switch port configuration has not been carried out, whether by mistake or deliberately. The node-installer displays the configuration as a suggestion along with a confirmation dialog (figure 5.13). The suggestion can be interrupted, and other node configurations can be selected manually instead using a sub-dialog. By default (in the main dialog) the configuration is accepted after a timeout.

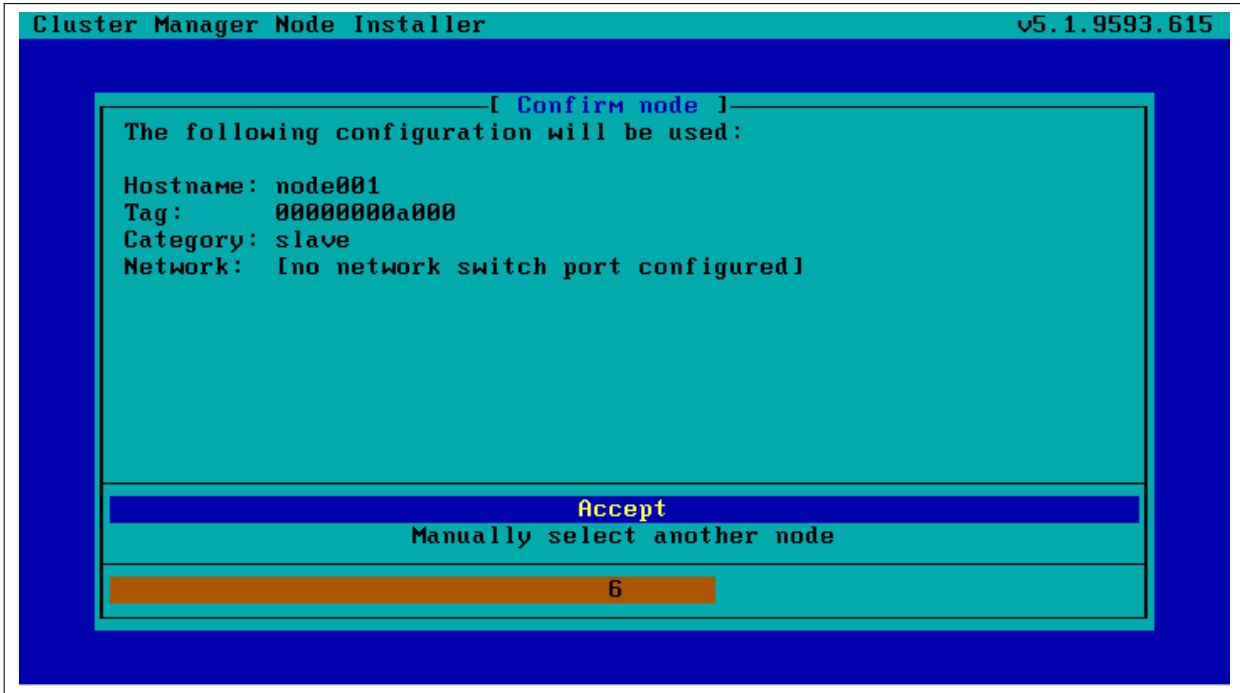


Figure 5.13: Scenarios: Port Unset Dialog

A truth table summarizing the scenarios is helpful:

Scenario	Node known?	Switch port detected?	Switch port configuration found?	Switch port configuration conflicts with node configuration?
1	No	Yes	Yes	No
2	No	Yes	No	No
3	No	No	No	No
4	Yes	Yes	Yes	No
5	Yes	Yes	Yes	Yes (configurations differ)
6	Yes	No	Yes	Yes (port expected by MAC configuration not found)
7	Yes	Yes	No	No (port not expected by MAC configuration)

In these scenarios, whenever the user manually selects a node configuration in the prompt dialog, an attempt to detect an Ethernet switch port is repeated. If a port mismatch still occurs, it is handled by the system as if the user has not made a selection.

Summary Of Behavior During Hardware Changes

The logic of the scenarios means that an unpreconfigured node always boots to a dialog loop requiring manual intervention during a first install (scenarios 2 and 3). For subsequent boots the behavior is:

- If the node MAC hardware has changed (scenarios 1, 2, 3):
 - if the node is new and the detected port has a configuration, the node automatically boots to that configuration (scenario 1).
 - else manual intervention is needed (scenarios 2, 3)

- If the node MAC hardware has not changed (scenarios 4, 5, 6, 7):
 - if there is no port mismatch, the node automatically boots to its last configuration (scenarios 4, 7).
 - else manual intervention is needed (scenarios 5, 6).

The `newnodes` Command

newnodes basic use: New nodes that have not been configured yet can be detected using the `newnodes` command from within the device mode of `cmsh`. A new node is detected when it reaches the node-installer stage after booting, and contacts the head node.

Example

```
[bright90->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 `ethernetwitch` property for the specified device can be set to the port and switch values.

Example

```
[bright90->device]% showport 00:0C:29:01:0F:F8
switch01:8
[bright90->device]% set node003 ethernetwitch switch01:8
[bright90->device*]% commit
```

When the node name (`node003` in the preceding example) is assigned, the node-installer stops looping and goes ahead with the installation to the node.

The preceding basic use of `newnodes` is useful for small numbers of nodes. For larger number of nodes, the advanced options of `newnodes` may help carry out node-to-MAC assignment with less effort.

newnodes advanced use—options: The list of MAC addresses discovered by a `newnodes` command can be assigned in various ways to nodes specified by the administrator. Node objects should be created in advance to allow the assignment to take place. The easiest way to set up node objects is to use the `--clone` option of the `foreach` command (section 2.5.5).

The advanced options of `newnodes` are particularly useful for quickly assigning node names to specific physical nodes. All that is needed is to power the nodes up in the right order. For nodes with the same hardware, the node that is powered up first reaches the stage where it tries to connect with the node-installer first. So its MAC address is detected first, and arrives on the list generated by `newnodes` first. If some time after the first node is powered up, the second node is powered up, then its MAC address becomes the second MAC address on the list, and so on for the third, fourth, and further nodes.

When assigning node names to a physical node, on a cluster that has no such assignment already, the first node that arrived on the list gets assigned the name `node001`, the second node that arrived on the list gets assigned the name `node002` and so on.

The advanced options are shown in device mode by running the `help newnodes` command. The options can be introduced as being of three kinds: straightforward, grouping, and miscellaneous:

- The straightforward options:

```
-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

```
[bright90->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

```
[bright90->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

```
[bright90->device]% newnodes -s -n node023..node025
MAC                First appeared          Hostname
-----
00:0C:29:EF:40:2A  Tue, 01 Nov 2011 11:42:31 CET  node023
00:0C:29:95:D3:5B  Tue, 01 Nov 2011 11:46:25 CET  node024
00:0C:29:65:9A:3C  Tue, 01 Nov 2011 11:47:13 CET  node025
```

Once this is done, the node-installer is able to stop looping, and to go ahead and install the new nodes with an image.

The physical layout in the rack may then look as indicated by this:

before	after	MAC
node020	node020	
node021	node021	
node022	node022	
	node023	...A
	node024	...B
	node025	...C

Here, node023 is the node with the MAC address ending in A.

If instead of the previous `newnodes` command, an offset of 1 is used to skip assigning the first new node:

Example

```
[bright90->device]% newnodes -s -o 1 node024..node025
```

then the rack layout looks like:

before	after	MAC
node020	node020	
node021	node021	
node022	node022	
	<i>unassigned</i>	...A
	node024	...B
	node025	...C

Here, *unassigned* is where node023 of the previous example is physically located, that is, the node with the MAC address . . . A. The lack of assignment means there is actually no association of the name

node023 with that MAC address, due to the `newnodes` command having skipped over it with the `-o` option.

If instead the assignment is done with:

Example

```
[bright90->device]% newnodes -s 1 node024..node026
```

then the `node023` name is unassigned, and the name `node024` is assigned instead to the node with the MAC address `...A`, so that the rack layout looks like:

before	after	MAC
node020	node020	
node021	node021	
node022	node022	
	node024	...A
	node025	...B
	node026	...C

newnodes advanced use—assignment grouping example: Node range assignments are one way of using `newnodes`. However assignments can also be made to a category, a rack, or a chassis. For example, with Bright View assigning node names to a rack can be done from the Racks option of the node. For example, to add a `node001` to a `rack1`, the clickpath would be:

```
Devices→Settings [node001] →Rack [rack1].
```

In `cmsh`, the assignment of multiple node names to a rack can conveniently be done with a `foreach` loop from within `device` mode:

Example

```
[bright90->device]% foreach -n node020..node029 (set rack rack02)
[bright90->device*]% commit
[bright90->device]% foreach -n node030..node039 (set rack rack03)
[bright90->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

```
[bright90->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

```
[bright90->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)i...]

[bright90->device]% newnodes

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

[bright90->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:

```
[bright90->device]% newnodes -s -f -n node022
```

Node Identification

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

Devices→Nodes Identification.

The node identification resource is roughly the Bright View equivalent to the `newnodes` command of `cmsh`. Like `newnodes`, the resource lists the MAC address of any unassigned node that the head node detects, and shows the associated detected switch port for the node. Also, like `newnodes`, it can help assign a node name to the node, assuming the node object exists. After assignment is done, the new status should be saved.

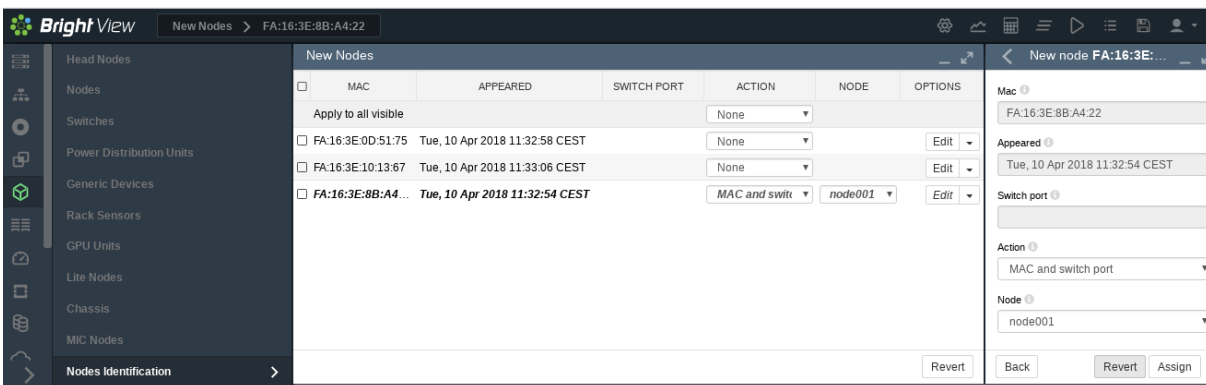


Figure 5.14: Node Identification Resource

The most useful way of using the node identification resource is for node assignment in large clusters.

¹with switch port assignment in place, scenario 1 means the new node simply boots up by default and becomes the new node022 without further intervention

To do this, it is assumed that the node objects have already been created for the new nodes. The creation of the node objects means that the node names exist, and so assignment to the node names is able to take place. An easy way to create nodes, set their provisioning interface, and set their IP addresses is described in the section on the *node creation wizard* (section 5.7.2). Node objects can also be created by running *cmsh's* *foreach* loop command on a node with a `--clone` option (section 2.5.5).

The nodes are also assumed to be set for net booting, typically set from a BIOS setting.

The physical nodes are then powered up in an arranged order. Because they are unknown new nodes, the node-installer keeps looping after a timeout. The head node in the meantime detects the new MAC addresses and switch ports in the sequence in which they first have come up and lists them in that order.

By default, all these newly detected nodes are set to *auto*, which means their numbering goes up sequentially from whatever number is assigned to the preceding node in the list. Thus, if there are 10 new unassigned nodes that are brought into the cluster, and the first node in the list is assigned to the first available number, say `node327`; then clicking on *assign* automatically assigns the remaining nodes to the next available numbers, say `node328-node337`.

After the assignment, the node-installer looping process on the new nodes notices that the nodes are now known. The node-installer then breaks out of the loop, and installation goes ahead without any intervention needed at the node console.

5.4.3 Starting Up All Network Interfaces

At the end of section 5.4.2, the node-installer knows which node it is running on, and has decided what its node configuration is.

Starting Up All Provisioning Network Interfaces

It now gets on with setting up the IP addresses on the provisioning interfaces required for the node-installer, while taking care of matters that come up on the way:

Avoiding duplicate IP addresses: The node-installer brings up all the network interfaces configured for the node. Before starting each interface, the node-installer first checks if the IP address that is about to be used is not already in use by another device. If it is, then a warning and retry dialog is displayed until the IP address conflict is resolved.

Using BOOTIF to specify the boot interface: `BOOTIF` is a special name for one of the possible interfaces. The node-installer automatically translates `BOOTIF` into the name of the device, such as `eth0` or `eth1`, used for network booting. This is useful for a machine with multiple network interfaces where it can be unclear whether to specify, for example, `eth0` or `eth1` for the interface that was used for booting. Using the name `BOOTIF` instead means that the underlying device, `eth0` or `eth1` in this example, does not need to be specified in the first place.

Halting on missing kernel modules for the interface: For some interface types like VLAN and channel bonding, the node-installer halts if the required kernel modules are not loaded or are loaded with the wrong module options. In this case the kernel modules configuration for the relevant software image should be reviewed. Recreating the ramdisk and rebooting the node to get the interfaces up again may be necessary, as described in section 5.8.5.

Bringing Up Non-Provisioning Network Interfaces

Provisioning interfaces are by default automatically brought up during the *init* stage, as the node is fully booted up. The BMC and non-provisioning interfaces on the other hand have a different behavior:

Bringing Up And Initializing BMC Interfaces: If a BMC interface is present and powered up, then it is expected to be running at least with layer 2 activity (ethernet). It can be initialized in the node config-

uration (section 3.7) with an IP address, netmask and user/password settings so that layer 3 (TCP/IP) networking works for it. BMC networking runs independently of node networking.

Bringing up non-BMC, non-provisioning network interfaces: Non-provisioning interfaces are inactive unless they are explicitly brought up. 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 183) 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 185) in case a full installation is to take place.

These values are merely determined at this stage; nothing is executed yet.

Install-mode Values

The *install-mode* can have one of five values: `AUTO`, `FULL`, `MAIN`, `NOSYNC`, and `SKIP`. It should be understood that the term “*install-mode*” implies that these values operate only during the node-installer phase.²

²For example, `imageupdate` (section 5.6.2), which is run by `CMDaemon`, ignores these settings, which is as expected. This means that, for example, if `imageupdate` is run with `NOSYNC` set, then the head node image is still synchronized over as usual to the regular node while the node is up. It is only during node boot, during the installer stage, that setting `NOSYNC` prevents synchronization.

- If the install-mode is set to `FULL`, then the node-installer re-partitions, creates new filesystems and synchronizes a full image onto the local drive according a *partition layout*. This process wipes out all pre-boot drive content.

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

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

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

- If the install-mode is set to `MAIN`, then the node-installer does not carry out a disk check, and goes on to maintenance mode, allowing manual investigation of specific problems. The local drive is untouched.
- If the install-mode is set to `NOSYNC`, and the partition layout check matches the stored XML configuration, then the node-installer skips synchronizing the image to the node, so that contents on the local drive persist from the previous boot. An exception to this is the node certificate and key, that is the files `/cm/local/apps/cmd/etc/cert.{pem|key}`. These are updated from the head node if missing.

If however the partition layout does not match the stored configuration, a `FULL` image sync is triggered. Thus, for example, a burn session (Chapter 8 of the *Installation Manual*), with the default burn configuration which destroys the existing partition layout on a node, will trigger a `FULL` image sync on reboot after the burn session.

The `NOSYNC` setting should therefore not be regarded as a way to protect data. Ways to preserve data across node reboots are discussed in the section that discusses the `FULL` install confirmation settings (page 185).

`NOSYNC` is useful during mass planned node reboots when set with the `nextinstallmode` option of `device mode`. This sets the nodes to use the OS on the hard drive, during the next boot only, without an image sync:

Example

```
[bright90]% device foreach -n node001..node999 (set nextinstallmode nosync)
[bright90]% device commit
```

- If the install-mode is set to `SKIP`, then the node-installer does not carry out a check of the partitions and filesystems, and it also does not carry out a software image synchronization. If a node runs into problems with its drive content during a normal start up attempt, then this mode can perhaps be used to attempt data recovery on the node.

Install-mode Logging

The decision that is made is normally logged to the node-installer file, `/var/log/node-installer` on the head node.

Example


```
08:40:58 node001 node-installer: Installmode is: AUTO
08:40:58 node001 node-installer: Fetching disks setup.
08:40:58 node001 node-installer: Setting up environment for initialize scripts.
08:40:58 node001 node-installer: Initialize script for category default is empty.
08:40:59 node001 node-installer: Checking partitions and filesystems.
08:40:59 node001 node-installer: Updating device status: checking disks
08:40:59 node001 node-installer: Detecting device '/dev/sda': found
08:41:00 node001 node-installer: Number of partitions on sda is ok.
08:41:00 node001 node-installer: Size for /dev/sda1 is ok.
08:41:00 node001 node-installer: Checking if /dev/sda1 contains ext3 filesystem.
08:41:01 node001 node-installer: fsck.ext3 -a /dev/sda1
08:41:01 node001 node-installer: /dev/sda1: recovering journal
08:41:02 node001 node-installer: /dev/sda1: clean, 129522/1250928 files, 886932/5000000 blocks
08:41:02 node001 node-installer: Filesystem check on /dev/sda1 is ok.
08:41:02 node001 node-installer: Size for /dev/sda2 is wrong.
08:41:02 node001 node-installer: Partitions and/or filesystems are missing/corrupt. (Exit code\
 18, signal 0)
08:41:03 node001 node-installer: Creating new disk layout.
```

In this case the node-installer detects that the size of `/dev/sda2` on the disk no longer matches the stored configuration, and triggers a full re-install. For further detail beyond that given by the node-installer log, the disks script at `/cm/node-installer/scripts/disks` on the head node can be examined. The node-installer checks the disk by calling the disks script. Exit codes, such as the 18 reported in the log example, are defined near the top of the disks script.

Install-mode's Execution Modes

Execution of an install-mode setting is possible in several ways, both permanently or just temporarily for the next boot. Execution can be set to apply to categories or individual nodes. The node-installer looks for install-mode execution settings in this order:

1. The "New node installmode" property of the node's category. This decides the install mode for a node that is detected to be new.

It can be set for the default category using a clickpath such as:

```
Grouping→Node categories[default]→Edit→Settings→Install mode
```

or using `cmsh` with a one-liner such as:

```
cmsh -c "category use default; set newnodeinstallmode FULL; commit"
```

By default, the "New node installmode" property is set to FULL.

2. The Install-mode setting as set by choosing a PXE menu option on the console of the node before it loads the kernel and ramdisk (figure 5.15). This only affects the current boot. By default the PXE menu install mode option is set to AUTO.



Figure 5.15: PXE Menu With Install-mode Set To AUTO

3. The “Next boot install-mode” property of the node configuration. This can be set for a node such as node001 using a clickpath such as:

Devices→Nodes[node001]→Edit→Settings→Install mode

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

```
cmsh -c "device use node001; set nextinstallmode FULL; commit"
```

The property is cleared when the node starts up again, after the node-installer finishes its installation tasks. So it is empty unless specifically set by the administrator during the current uptime for the node.

4. The install-mode property can be set in the node configuration using Bright View via Devices→Nodes[node001]→Edit→Settings→Install mode or using cmsh with a one-liner such as:

```
cmsh -c "device use node001; set installmode FULL; commit"
```

By default, the install-mode property is auto-linked to the property set for install-mode for that category of node. Since the property for that node’s category defaults to AUTO, the property for the install-mode of the node configuration defaults to “AUTO (Category)”.

5. The install-mode property of the node’s category. This can be set using Bright View with a clickpath such as:

Grouping→Node categories[default]→Edit→Settings→Install mode
or using cmsh with a one-liner such as:

```
cmsh -c "category use default; set installmode FULL; commit"
```

As already mentioned in a previous point, the install-mode is set by default to AUTO.

6. A dialog on the console of the node (figure 5.16) gives the user a last opportunity to overrule the install-mode value as determined by the node-installer. By default, it is set to AUTO:

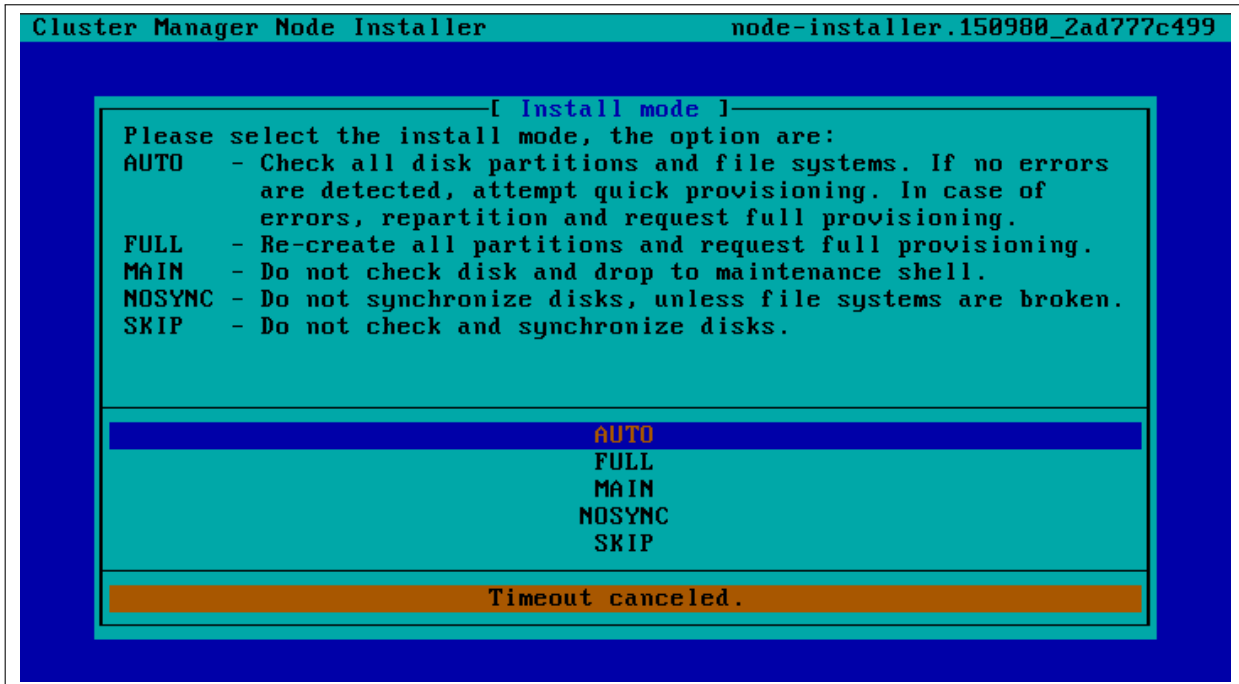


Figure 5.16: Install-mode Setting Option During Node-Installer Run

FULL Install Confirmation via datanode Setting

Related to execution mode values is the ability to carry out a FULL install only after explicit confirmation, via the `datanode` property. This must be set in order to prompt for a confirmation, when a FULL installation is about to take place. If it is set, then the node-installer only goes ahead with the FULL install after the administrator has explicitly confirmed it.

The `datanode` property can be set in the node configuration of, for example, `node001` with Bright View via the clickpath:

```
Devices→Nodes[node001]→Edit→Settings→Data node[Yes]
```

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

```
[root@bright90 ~]# cmsh -c "device use node001; set datanode yes; commit"
```

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

Why the FULL install confirmation is useful: The reason for such a setting is that a FULL installation can be triggered by disk or partition changes, or by a change in the MAC address. If that happens, then:

- considering a drive, say, `/dev/sda` that fails, this means that any drive `/dev/sdb` would then normally become `/dev/sda` upon reboot. In that case an unwanted FULL install would not only be triggered by an install-mode settings of FULL, but also by the install-mode settings of AUTO or NOSYNC. Having the new, “accidental” `/dev/sda` have a FULL install is unlikely to be the intention, since it would probably contain useful data that the node-installer earlier left untouched.

- considering a node with a new MAC address, but with local storage containing useful data from earlier. In this case, too, an unwanted FULL install would not only be triggered by an install-mode setting of FULL, but also by the install-mode settings AUTO or NOSYNC.

Thus, in cases where nodes are used to store data, an explicit confirmation before overwriting local storage contents is a good idea. However, by default, no confirmation is asked for when a FULL installation is about to take place.

Carrying out the confirmation: When the confirmation is required, then it can be carried out by the administrator as follows:

- From the node console. A remote console launched from Bright View or cmsh will also work if SOL connectivity has been configured.
- From cmsh, within device mode, using the `installerinteractions` command (some output elided):

Example

```
[bright90->device]% installerinteractions -w -n node001 --confirm
Hostname Action
-----
node001   Requesting FULL Install (partition mismatch)
[bright90->device]%
...07:57:36 [notice] bright90: node001 [ INSTALLER_CALLINGINIT ]...
[bright90->device]%
...07:58:20 [notice] bright90: node001 [  UP  ]
```

Besides confirmation, the `installerinteractions` command has options that include letting it:

- deny the installation, and put it into maintenance mode
- carry out a dry-run
- carry out its actions for node groupings such as: node lists, node categories, node groups, chassis, racks, as are possible in the grouping options (page 40).

Further details on the command can be viewed by running `help installerinteractions`.

An alternative way to avoid overwriting node storage: Besides the method of FULL install confirmation for `datanode`, there is a method based on XML assertions, that can also be used to prevent data loss on nodes.

It uses XML assertions to confirm that the physical drive is recognized (Appendix D.11).

A way to overwrite a specified block device: A related method is that sometimes, for reasons of performance or convenience, it may be desirable to clear data on particular block devices for a node, and carry it out during the next boot only. This can be 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:

```
[bright90->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, MAIN, or SKIP: If *install-mode* values of *FULL*, *MAIN*, or *SKIP* apply to the current node instead, then no partition checking or filesystem checking is done by the node-installer.

NOSYNC: If the *install-mode* value of *NOSYNC* applies, then if the partition and filesystem checks both show no errors, the node starts up without getting an image synced to it from the provisioning node. If the partition or the filesystem check show errors, then the node partition is rewritten, and a known good image is synced across.

Behavior As Decided By XML Configuration Settings

The node-installer is capable of creating advanced drive layouts, including LVM setups, and hardware and software RAID setups. Drive layout examples and relevant documentation are in Appendix D.

The XML description used to set the drive layouts can be deployed for a single device or to a category of devices.

Hardware RAID: Bright Cluster Manager supports hardware RAID levels 0, 1, 5, 10, and 50, and supports the following options:

- | | Option |
|-----------------------|--------|
| • stripe size: | 64kB |
| | 128kB |
| | 256kB |
| | 512kB |
| | 1024kB |

- | Option |
|-------------------------------|
| • cache policy: Cached |
| Direct |

- | Option | Description |
|----------------------------|---------------|
| • read policy: NORA | No Read Ahead |
| RA | Read Ahead |
| ADRA | Adaptive Read |

- | Option | Description |
|---------------------------|---------------|
| • write policy: WT | Write Through |
| WB | Write Back |

5.4.7 Synchronizing The Local Drive With The Software Image

After having mounted the local filesystems, these can be synchronized with the contents of the software image associated with the node (through its category). Synchronization is skipped if the install-mode values of `NOSYNC` or `SKIP` are set, and takes place if `FULL` or `AUTO` are set. Synchronization is delegated by the node-installer to the CMDaemon provisioning system. The node-installer just sends a provisioning request to CMDaemon on the head node.

For an install-mode of `FULL`, or for an install-mode of `AUTO` where the local filesystem is detected as being corrupted, full provisioning is done. For an install-mode of `AUTO` where the local filesystem is healthy and agrees with that of the software image, sync provisioning is done.

The `lock`, `unlock`, and `islocked` Commands For Software Images

The software image that is requested is available to nodes by default. Its availability can be altered and checked with the following commands:

- `lock`: this *locks* an image so that the image cannot be provisioned until the image is *unlocked*.
- `unlock`: this *unlocks* a locked image, so that request for provisioning the image is no longer prevented by a lock
- `islocked`: this lists the locked or unlocked states of images.

Locking an image is sometimes useful, for example, to make changes to an image when nodes are booting:

Example

```
[root@bright90 ~]# cmsh
[bright90]% softwareimage list
Name (key)          Path                                     Kernel version          Nodes
-----
default-image      /cm/images/default-image              3.10.0-1062.12.1.el7.x86_64  3
[bright90]% softwareimage
[bright90->softwareimage]% lock default-image
[bright90->softwareimage]% islocked
Name          Locked
-----
default-image yes
```

```
[bright90->softwareimage]% lock default-image
[bright90->softwareimage]% device
[bright90->device]% reboot node001
...the cluster administrator makes changes to the node image during a boot, as it waits for
the image to unlock...
[bright90->device]% softwareimage unlock default-image
```

For an unlocked image, on receiving the provisioning request, CMDaemon assigns the provisioning task to one of the provisioning nodes. The node-installer is notified when image synchronization starts, and also when the image synchronization task ends—whether it is completed successfully or not.

Exclude Lists: `excludelistsyncinstall` **And** `excludelistfullinstall`

What files are synchronized is decided by an *exclude list*. An exclude list is a property of the node category, and is a list of directories and files that are excluded from consideration during synchronization. The excluded list that is used is decided by the type of synchronization chosen: `full` or `sync`:

- A `full` type of synchronization rewrites the partition table of the node, then copies the filesystem from a software image to the node, using a list to specify files and directories to exclude from consideration when copying over the filesystem. The list of exclusions used is specified by the `excludelistfullinstall` property.

The intention of `full` synchronization is to allow a complete working filesystem to be copied over from a known good software image to the node. By default the `excludelistfullinstall` list contains `/proc/`, `/sys/`, and `lost+found/`, which have no content in 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-file systems which are generated during node boot.

The administrator should be aware that nothing on a node hard drive can be regarded as persistent because a `FULL sync` takes place if any error is noticed during a partition or filesystem check.

Anything already on the node that matches the content of these sub-lists is not overwritten by image content during an `excludelistsyncinstall sync`. However, image content that is not on the node is copied over to the node only for items matching the first sub-list. The remaining files and directories on the node, that is, the ones that are not in the sub-lists, lose their original contents, and are copied over from the software image.

A `cmsh` one-liner to get an exclude list for a category is:

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

Similarly, to set the list:

```
cmsh -c "category use default; set excludelistfullinstall; commit"
```

where a text-editor opens up to allow changes to be made to the list. In Bright View the clickpath is:

Grouping→Node Categories→Edit→Node Category→Settings→Exclude list full install

Image synchronization is done using `rsync`, and the syntax of the items in the exclude lists conforms to the “INCLUDE/EXCLUDE PATTERN RULES” section of the `rsync(1)` man page, which includes patterns such as “**”, “?”, and “[[:alpha:]]”.

The `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

```
[bright90->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 14), 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 14.1).

Transport Protocol Used For Image Data: `provisioningtransport`

The `provisioningtransport` property of the node sets whether the image data is sent encrypted or unencrypted to the node from the provisioner. The property value is set via the device mode for the receiving node to one of these values:

- `rsyncdaemon`, which sends the data unencrypted
- `rsyncssh`, which sends the data encrypted

The `provisioningtransport` value can be set for all nodes, including provisioning nodes, head nodes, and cloud-director (section 3.2 of the *Cloudbursting Manual*) nodes. Because encryption severely increases the load on the provisioning node, using `rsyncssh` is only suggested if the users on the network cannot be trusted. By default, `provisioningtransport` is set to `rsyncdaemon`. If high availability (Chapter 14) is set up with the head nodes exposed to the outside world on the external network, the administrator should consider setting up `rsyncssh` for the head nodes.

The `rsyncssh` transport requires passwordless root access via `ssh` from the provisioner to the node being provisioned. This is configured by default in the default Bright Cluster Manager nodes. However, if a new image is created with the `--exclude` options for `cm-create-image` as explained in (section 11.6.2), the keys must be copied over from `/root/.ssh/` on the existing nodes.

Tracking The Status Of Image Data Provisioning: provisioningstatus

The provisioningstatus command within the softwareimage mode of cmsh displays an updated state of the provisioning system. As a one-liner, it can be run as:

```
bright90:~ # 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 bright90:
  Max number of provisioning nodes: 10
  Nodes provisioning:              0
  Nodes currently being provisioned: <none>
```

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

The Bright View equivalent to provisioningstatus is accessed via the clickpath:

```
Provisioning→Provisioning nodes
```

By default, it displays basic status information on provisioning requests.

Tracking The Provisioning Log Changes: synclog

For a closer look into the image file changes carried out during provisioning requests, the synclog command from device mode can be used (lines elided in the following output):

Example

```
[bright90->device]% synclog node001
Tue, 11 Jan 2011 13:27:17 CET - Starting rsync daemon based provisioning. Mode is SYNC.

sending incremental file list
./
...
deleting var/lib/ntp/etc/localtime
var/lib/ntp/var/run/ntp/
...
sent 2258383 bytes  received 6989 bytes  156232.55 bytes/sec
total size is 1797091769  speedup is 793.29

Tue, 11 Jan 2011 13:27:31 CET - Rsync completed.
```

Statistical Analysis Of Provisioning Sessions: syncinfo

A provisioning session takes place between a provisioning image and a filesystem partition on a node. Statistics can be presented for the sessions using the syncinfo command. The statistical information presented is for number of files considered for transfer, the number of files that were actually transferred, how long the transfer took, which image and node were involved, and so on. The syncinfo command is run in device mode (output ellipsized and truncated):

Example

```
[head->device]% syncinfo
Node      Path                                     Provisioner  Age   Duration Total files Transferred files ...
-----  -
node001   /cm/images/default-image               head        34s  21s    171,504   328      ...
node002   /cm/images/default-image               head        34s  22s    171,504   328      ...
...
```

The `syncinfo` command has options to run it per node, category, rack, and so on. Details on the options can be seen by running the `help` command (`help syncinfo`).

Aborting Provisioning With `cancelprovisioningrequest`

The `cancelprovisioningrequest` command cancels provisioning.

Its usage is:

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

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

```
bright90:~ # cmsh -c "softwareimage cancelprovisioningrequest -a"
```

The `provisioningstatus` command of `cmsh`, can be used to find request IDs. Individual request IDs, for example 10 and 13, can then be specified in the `cancelprovisioningrequest` command, as:

```
bright90:~ # cmsh -c "softwareimage cancelprovisioningrequest 10 13"
```

The help page for `cancelprovisioningrequest` shows how to run the command on node ranges, groups, categories, racks, chassis, and so on.

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

5.4.8 Writing Network Configuration Files

In the previous section, the local drive of the node is synchronized according to install-mode settings with the software image from the provisioning node. The node-installer now sets up configuration files for each configured network interface. These are files like:

```
/etc/sysconfig/network-scripts/ifcfg-eth0
```

for Red Hat, Scientific Linux, 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 Bright View or `cmsh` (section 3.10.2).

5.4.10 Booting From The Local Hard Drive

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

The node-installer can, optionally, during image synchronization, install a local drive boot record on the local hard drive if the `installbootrecord` property of the node configuration or node category is set to `on`. Setting the local drive boot record means that the node tries to use a local hard drive boot installer during the next boot. This is a step toward having it become a standalone node that does not boot from the network. This step, and the other steps needed to allow booting from the local hard drive are covered next.

Setting The Boot Record To Allow The Node To Be Standalone

The local drive boot record is installed in the MBR of the local drive, overwriting the default iPXE boot record (section 5.1.2).

With a working custom software image, the boot record can be installed with `cmsh` commands for a node `node001` with:

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

or for a category default with:

```
cmsh -c "category use default; set installbootrecord yes; commit"
```

In Bright View, the equivalent is the `Install boot record` option. This can similarly be enabled and saved in the Bright View node configuration or node category.

Setting the local drive boot record allows the next boot to be from the local hard drive, if the node is set up right to boot from the local hard drive.

Booting from the local hard drive often requires some further changes, as explained next.

Managing Boot Sequence And Bootloader To Ensure The Node Can Be Standalone

For a local hard drive boot to work:

1. hard drive booting must be set to have a higher priority than network booting in the BIOS of the node. Otherwise regular PXE booting is attempted, despite whatever value `installbootrecord` has.
2. A working bootloader must be present.

By default, the node image for the cluster manager has nodes set to use a SYSLINUX bootloader.

If the administrator is not using the default software image, but is using a custom software image (section 11.6.1), 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 11.6.2) is impractical.

Providing a custom working image from a standalone node that has been customized after direct installation from the parent distribution, ensures the GRUB boot configuration layout of the custom image is as expected by the parent distribution. This then allows a standalone GRUB boot on the node to run properly.

Nodes can be set to use a GRUB bootloader from within device mode, or from within category mode, by changing the `bootloader` parameter within the mode. For example, for a node `node001`:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% get bootloader
syslinux (default)
[bright90->device[node001]]% set bootloader grub
[bright90->device*[node001*]]% commit
```

or, for the default category:

```
[root@bright90 ~]# cmsg
[bright90]% category use default
[bright90->category[default]]% get bootloader
syslinux
[bright90->category[default]]% set bootloader grub
[bright90->category*[default*]]% commit
```

Arranging for the two items in the preceding list ensures that the next boot is from GRUB on the hard drive. However, the `BOOTIF` also needs to be changed for booting to be successful. How it can be changed, and why it needs to be changed, is described next.

Changing `BOOTIF` To Ensure The Node Can Be Standalone

If the BIOS is set to boot from the hard drive, and if there is a working boot loader, and if the boot record has been installed, then the node boots via the boot record on the hard drive.

`BOOTIF` is the default value for the network interface for a node that is configured as a Bright software image. However, the `BOOTIF` interface is undefined during hard drive booting, because it depends on the network provisioning setup, which is not running. This means that the networking interface would fail during hard drive boot for a standard image. To remedy this, the interface should be set to a defined network device name, such as `eth0`, or the modern equivalents such as `en01` (section 5.8.1). The defined network device name, as the kernel sees it, can be found by logging into the node and taking a look at the output of `ip link`:

Example

```
[root@bright90 ~]# ssh node001 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc ...
```

In Bright Cluster Manager the provisioning interface is mandatory, even if it is not provisioning. So it is set to the value of kernel-defined network device name instead of `BOOTIF`:

Example

```
[bright90]% device interfaces node001
[bright90->device[node001]->interfaces]% list
Type          Network device name  IP          Network
-----
physical      BOOTIF [prov]        10.141.0.1  internalnet
[bright90->device[node001]->interfaces]% set bootif networkdevicename eth0
[bright90->device*[node001*]->interfaces*]% commit
[bright90->device[node001]->interfaces]% list
Type          Network device name  IP          Network
-----
physical      eth0 [prov]          10.141.0.1  internalnet
Tue Mar 31 13:46:44 2020 [notice] bright90: node001 [ UP ], restart required (eth0)
```

In the preceding example, the kernel-defined network device name is assumed to be `eth0`. It should be modified as required.

In addition, the new IP address is assumed to be on the same internal network. If the administrator intends the node to be standalone on another network, then the network and the IP address can be set to appropriate values.

When interface changes are carried out to make the node standalone, warnings show up saying that a reboot is required. A reboot of the node should be done when the interface configuration is complete.

During reboot, the node then boots from the hard drive as a standalone, with a non-`BOOTIF` network interface.

Bringing A Node That Boots From Its Hard Drive Back Into A Cluster

If the node is to be brought back into the cluster, then simply unsetting “Install boot record” and rebooting the node does not restore its iPXE boot record and hence its ability to iPXE boot. To restore the iPXE boot record, the node can be booted from the default image copy on the head node via a network boot again. Typically this is done by manual intervention during node boot to select network booting from the BIOS of the node.

Setting the value of `provisioninginterface` in `cmsh` for the node to `BOOTIF` is also recommended.

As suggested by the Bright Cluster Manager iPXE boot prompt, setting network booting to work from the BIOS (regular “PXE” booting) is preferred to the relatively roundabout way of iPXE booting from the disk.

SELinux Initialization For Hard Drive Boot And PXE Boot

If configured, SELinux (Chapter 9 of the *Installation Manual*) is initialized at this point. For a boot from the hard drive, the initialization occurs if an SELinux filesystem has been saved to disk previously. For a PXE boot, the initialization takes place if the `SELinuxInitialize` directive is set to `true` in the `node-installer.conf` file.

5.4.11 Running Finalize Scripts

A *finalize script* is similar to an *initialize script* (section 5.4.5), only it runs a few stages later in the node-provisioning process.

In the context of configuration (section 3.15.4) it is used when custom commands need to be executed after the preceding mounting, provisioning, and housekeeping steps, but before handing over control to the node’s local `init` process. For example, custom commands may be needed to:

- initialize some not explicitly supported hardware before `init` takes over
- supply a configuration file for the software image that cannot simply be added to the software image and used by `init` because it needs node-specific settings
- load a slightly altered standard software image on particular nodes, typically with the change depending on automatically detecting the hardware of the node it is being loaded onto. While this could also be done by creating a full new software image and loading it on to the nodes according to the hardware, it usually turns out to be better for simplicity’s sake (future maintainability) to minimize the number of software images for the cluster.

The custom commands used to implement such changes are then added to the `finalize` script. How to edit a `finalize` script is described in Appendix E.2.

A `finalize` script can be added to both a node’s category and the node configuration. The node-installer first runs a `finalize` script, if it exists, from the node’s category, and then a `finalize` script, if it exists, from the node’s configuration.

The node-installer sets several environment variables which can be used by the `finalize` script. Appendix E contains an example script which documents these variables.

5.4.12 Unloading Specific Drivers

Many kernel drivers are only required during the installation of the node. After installation they are not needed and can degrade node performance.

Baseboard Management Controllers (BMCs, section 3.7) that use IPMI drivers are an egregious example of this. The IPMI drivers are required to have the node-installer configure the IP address of any IPMI cards. Once the node is configured, these drivers are no longer needed, but they continue to consume significant CPU cycles and power if they stay loaded, which can affect job performance.

To solve this, the node-installer can be configured to unload a specified set of drivers just before it hands over control to the local `init` process. This is done by editing the `removeModulesBeforeInit`

setting in the node-installer configuration file

```
/cm/node-installer/scripts/node-installer.conf,
```

For the `node-installer.conf` file in `multidistro` and `multiarch` (section 11.7) configurations, the directory path `/cm/node-installer` takes the form:

```
/cm/node-installer-<distribution>-<architecture>
```

The values for `<distribution>` and `<architecture>` can take the values outlined on page 472.

By default, the IPMI drivers are placed in the `removeModulesBeforeInit` setting.

To pick up IPMI-related data values, IPMI access is then carried out over the network without the drivers.

5.4.13 Switching To The Local `init` Process

At this point the node-installer is done. The node's local drive now contains a complete Linux installation and is ready to be started. The node-installer hands over control to the local `/sbin/init` process, which continues the boot process and starts all runlevel services. From here on the boot process continues as if the machine was started from the drive just like any other regular Linux machine.

5.5 Node States

During the boot process, several state change messages are sent to the head node `CMDaemon` or detected by polling from the head node `CMDaemon`. The most important node states for a cluster after boot up are introduced in section 2.1.1. These states are described again, along with some less common ones to give a more complete picture of node states.

5.5.1 Node States Icons In Bright View

In the node icons used by Bright View:

- Nodes in the UP state are indicated by an up-arrow.
 - If all health checks (section 12.2.4) for the node are successful, the up-arrow is green.
 - If there is a health check that fails or if the node requires a reboot, the up-arrow is red.
- Nodes in the DOWN state are indicated by a blue down-arrow.
- There are some other states, including:
 - Nodes in a CLOSED state are indicated by an X
 - Nodes in a DOWN state that are installing are indicated by a underscored down-arrow icon: ↓

5.5.2 Node States Shown In `cmsh`

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

Example

```
[bright90->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 31) command:

Example

```
[bright90->device]% list -f "hostname:10, status:48"
hostname ( status
-----
apc01      [  UP  ]
```

```
bright90 [ UP ]
devhp [ UP ]
node001 [ UP ] restart-required, health check failed
node002 [ DOWN ] (hostname changed) restart-required
```

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

5.5.3 Node States Indicating Regular Start Up

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

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

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

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

Example

```
[bright90->device[node001]->interfaces[eth0]]% set revision bootif
```

- **INSTALLING.** This state is normally entered as soon as the node-installer has determined on which node the node-installer is running. Within this state, information messages display indicating what is being done while the node is in the INSTALLING state. Possible messages under the status column for the node within cmsh and Bright View are normally, in sequence:

1. node-installer started
2. Optionally, the following two messages:
 - (a) waiting for user input
 - (b) installation was resumed
3. checking disks
4. recreating partitions and filesystems
5. mounting disks
6. One of these following two messages:
 - (a) waiting for FULL provisioning to start
 - (b) waiting for SYNC provisioning to start
7. provisioning started, waiting for completion
8. provisioning complete
9. initializing SELinux

Between steps 1 and 3 in the preceding, these optional messages can also show up:

- If burn mode is entered or left:


```
running burn-in tests
burn-in test completed successfully
```
- If maintenance mode is entered:


```
entered maintenance mode
```

- `INSTALLER_CALLINGINIT`. This state is entered as soon as the node-installer has handed over control to the local `init` process. The associated message normally seen with it in `cmsh` and Bright View is:
 - switching to local root
- `UP`. This state is entered as soon as the `CMDaemon` of the node connects to the head node `CMDaemon`.

5.5.4 Node States That May Indicate Problems

Other node states are often associated with problems in the boot process:

- `DOWN`. This state is registered as soon as the `CMDaemon` on the regular node is no longer detected by `CMDaemon` on the head node. In this state, the state of the regular node is still tracked, so that `CMDaemon` is aware if the node state changes.
- `CLOSED`. This state is appended to the `UP` or `DOWN` state of the regular node by the administrator, and causes most `CMDaemon` monitoring actions for the node to cease. The state of the node is however still tracked by default, so that `CMDaemon` is aware if the node state changes.

The `CLOSED` state can be set from the device mode of `cmsh` using the `close` command. The help text for the command gives details on how it can be applied to categories, groups and so on. The `-m` option sets a message by the administrator for the closed node or nodes.

Example

```
root@headymcheadface ~]# cmsh
[headymcheadface]% device
[headymcheadface->device]% close -m "fan dead" -n node001,node009,node020
Mon May 2 16:32:01 [notice] headymcheadface: node001 ... [ DOWN/CLOSED ] (fan dead)
Mon May 2 16:32:01 [notice] headymcheadface: node009 ... [ DOWN/CLOSED ] (fan dead)
Mon May 2 16:32:01 [notice] headymcheadface: node020 ... [ DOWN/CLOSED ] (fan dead)
```

The `CLOSED` state can also be set from Bright View via the clickpath

Devices→Nodes→Edit↓Monitored state→Open/Close.

When the `CLOSED` state is set for a device, `CMDaemon` commands can still attempt to act upon it. For example, in the device mode of `cmsh`:

- `open`: This is the converse to the `close` command. It has the same options, including the `-m` option that logs a message. It also has the following extra options:
 - * `--reset`: Resets whatever the status is of the `devicestatus` check. However, this reset by itself does not solve any underlying issue. The issue may still require a fix, despite the status having been reset. For example, the `--reset` option can be used to reset the restart-required flag (section 5.5.2). However, the reason that set the restart-required flag is not solved by the reset. Restarts are required for regular nodes if there have been changes in the following: network settings, disk setup, software image, or category.
 - * `-f|--failbeforedown <count>`: Specifies the number of failed pings before a device is marked as down (default is 1).
- `drain` and `undrain` (Appendix G.4.1)
- For nodes that have power control³:

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


```
* power -f on
* power -f off
* power -f reset
```

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

- Devices→Nodes[node001]→Edit↓Monitored state→Open/Close
- Devices→Nodes[node001]→Edit↓Workload→Drain/Undrain
- Devices→Nodes[node001]→Edit↓Power→On/Off/Reset.

CMDaemon on the head node only maintains device monitoring logs for a device that is in the UP state. If the device is in a state other than UP, then CMDaemon only tracks its state, and can display the state if queried.

For example: if a node displays the state UP when queried about its state, and is given a ‘close’ command, it then goes into a CLOSED state. Querying the node state then displays the state UP/CLOSED. It remains in that CLOSED state when the node is powered down. Querying the node state after being powered down displays DOWN/CLOSED. Next, powering up the node from that state, and having it go through the boot process, has the node displaying various CLOSED states during queries. Typically the query responses show it transitioning from DOWN/CLOSED, to INSTALLING/CLOSED, to INSTALLER_CALLINGINIT/CLOSED, and ending up displaying the UP/CLOSED state.

Thus, a node set to a CLOSED state remains in a CLOSED state regardless of whether the node is in an UP or DOWN state. The only way out of a CLOSED state is for the administrator to tell the node to open via the `cmsh` “open” option discussed earlier. The node, as far as CMDaemon is concerned, then switches from the CLOSED state to the OPEN state. Whether the node listens or not does not matter—the head node records it as being in an OPENING state for a short time, and during this time the next OPEN state (UP/OPEN, DOWN/OPEN, etc) is agreed upon by the head node and the node.

When querying the state of a node, an OPEN tag is not displayed in the response, because it is the “standard” state. For example, UP is displayed rather than UP/OPEN. In contrast, a CLOSED tag is displayed when it is active, because it is a “special” state.

The CLOSED state is normally set to take a node that is unhealthy out of the cluster management system. The node can then still be in the UP state, displaying UP/CLOSED. It can even continue running workload jobs in this state, since workload managers run independent of CMDaemon. So, if the workload manager is still running, the jobs themselves are still handled by the workload manager, even if CMDaemon is no longer aware of the node state until the node is re-opened. For this reason, draining a node is often done before closing a node, although it is not obligatory.

- OPENING. This transitional state is entered as soon as the CMDaemon of the node rescinds the CLOSED state with an “open” command from `cmsh`. The state usually lasts no more than about 5 seconds, and never more than 30 seconds in the default configuration settings of 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 Bright View, and is queued and delegated to a provisioning node, just like a regular provisioning request. The properties that apply to the regular provisioning 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 Bright View, it is submitted, for a node `node001` for example, using the clickpath:

`Devices→Nodes[node001]→Edit↓Software image→Update node` (section 5.6.3). The `imageupdate` command and “Update node” menu option use a configuration file called `excludelistupdate`, which is, as its name suggests, a list of exclusions to the update.

The `imageupdate` command and “Update node” menu option 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 image` (Bright View), and involves further exclude lists `excludelistgrab` or `excludelistgrabnew`. The `grabimage` command and `Synchronize` option are covered in detail in section 11.5.2.

5.6.1 Updating Running Nodes: Configuration With `excludelistupdate`

The exclude list `excludelistupdate` used by the `imageupdate` command is defined as a property of the node’s category. It has the same structure and `rsync` patterns syntax as that used by the exclude lists for provisioning the nodes during installation (section 5.4.7).

Distinguishing Between The Intention Behind The Various Exclude Lists

The administrator should note that it is the `excludelistupdate` list that is being discussed here, in contrast with the `excludelistsyncinstall/excludelistfullinstall` lists which are discussed in section 5.4.7, and also in contrast with the `excludelistgrab/excludelistgrabnew` lists of section 11.5.2.

So, for the `imageupdate` command the `excludelistupdate` list concerns an *update* to a running system, while for installation `sync` or `full` provisioning, the corresponding exclude lists (`excludelistsyncinstall` and `excludelistfullinstall`) from section 5.4.7 are about an *install* during node start-up. Because the copying intention during updates is to be speedy, the `imageupdate` command synchronizes files rather than unnecessarily overwriting unchanged files. Thus, the `excludelistupdate` exclusion list it uses is actually analogous to the `excludelistsyncinstall` exclusion list used in the `sync` case of section 5.4.7, rather than being analogous to the `excludelistfullinstall` list.

Similarly, the `excludelistgrab/excludelistgrabnew` lists of section 11.5.2 are about a *grab* from the running node to the image.

- The `excludelistgrab` list here is intended for the case of synchronizing the existing image with the running node, and is thus analogous to the `excludelistsyncinstall` exclusion list.
- The `excludelistgrabnew` list here is intended for the case of copying a full image from the running node, and is thus analogous to the `excludelistfullinstall` list.

The following table summarizes this:

During:	Exclude list used is:	Copy intention:
update	excludelistupdate	sync, image to running node
install	excludelistfullinstall	full, image to starting node
	excludelistsyncinstall	sync, image to starting node
grab	excludelistgrabnew	full, running node to image
	excludelistgrab	sync, running node to image

The preceding table is rather terse. It may help to understand it if is expanded with some in-place footnotes, where the footnotes indicate what actions can cause the use of the exclude lists:

During:	Exclude list used is:	Copy intention:
update eg: imageupdate	excludelistupdate	sync, image to running node
install eg: node-provisioning process during pre- init stage depending on installmode decision	excludelistfullinstall	full, image to starting node
	eg: node provisioning with installmode FULL	
	excludelistsyncinstall	sync, image to starting node
	eg: node provisioning AUTO with healthy partition	
grab eg: grabimage (cmsh), Synchronize image (Bright View), and Grab to image (Bright View)	excludelistgrabnew	full, running node to image
	grabimage -i/Grab to image	
	excludelistgrab	sync, running node to image
	grabimage/Synchronize image	

The Exclude List Logic For `excludelistupdate`

During an `imageupdate` command, the synchronization process uses the `excludelistupdate` list, which is a list of files and directories. One of the cross checking actions that may run during the synchronization is that the items on the list are excluded when copying parts of the filesystem from a known good software image to the node. The detailed behavior is as follows:

The `excludelistupdate` list is in the form of two sublists. Both sublists are lists of paths, except that the second sublist is prefixed with the text `"no-new-files: "` (without the double quotes). For the node being updated, all of its files are looked at during an `imageupdate` synchronization run. During such a run, the logic that is followed is:

- if an excluded path from `excludelistupdate` exists on the node, then nothing from that path is copied over from the software image to the node
- if an excluded path from `excludelistupdate` does not exist on the node, then
 - if the path is on the first, non-prefixed list, then the path is copied over from the software image to the node.

- if the path is on the second, prefixed list, then the path is not copied over from the software image to the node. That is, no new files are copied over, like the prefix text implies.

This is illustrated by figure 5.17.

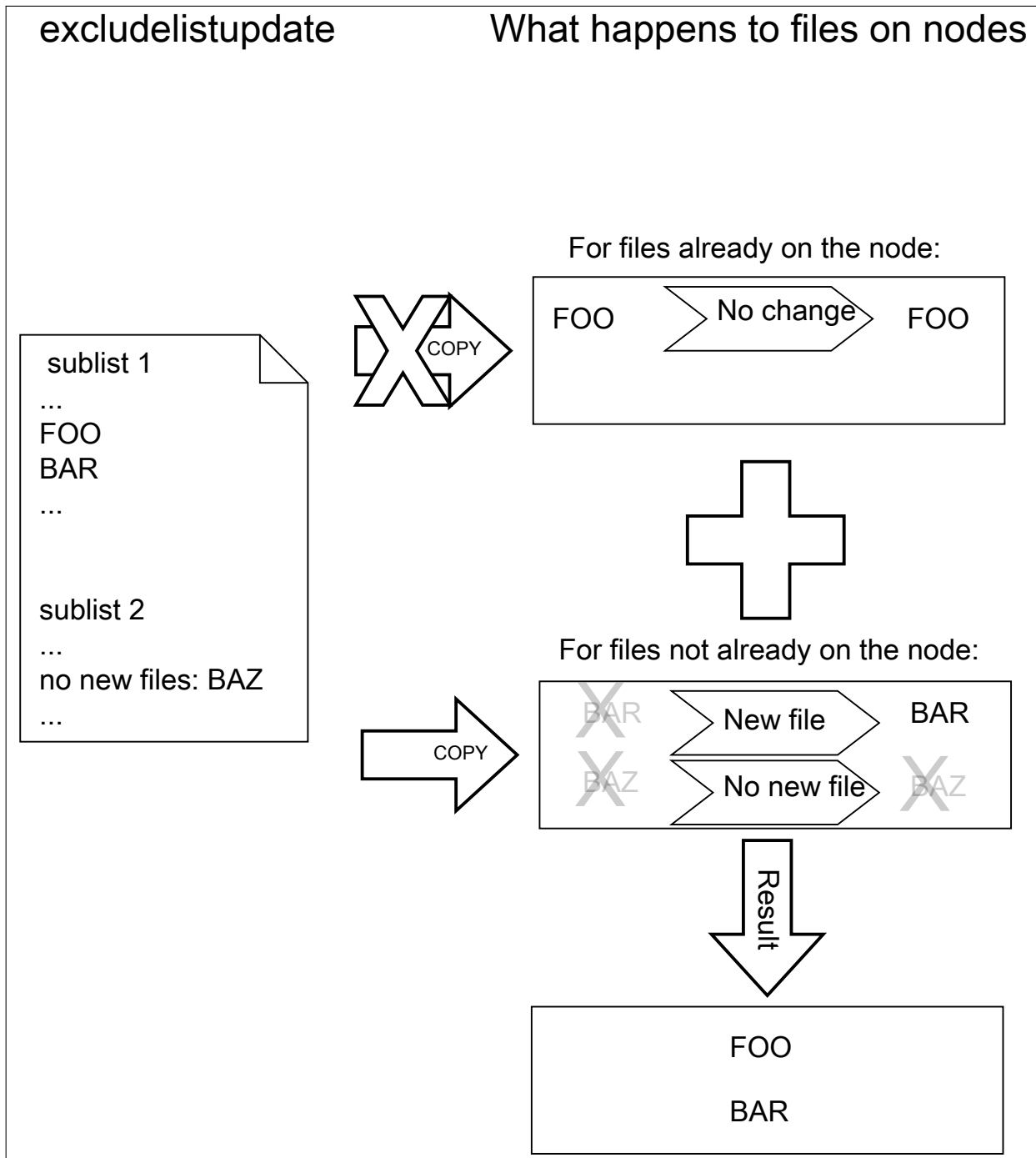


Figure 5.17: Exclude list logic

The files and directories on the node that are not in the sub-lists lose their original contents, and are copied over from the software image. So, content not covered by the sub-lists at that time is normally not protected from deletion.

Thus, the provisioning system excludes paths described according to the `excludelistupdate` prop-

erty.

The provisioning system also excludes a statically-imported filesystem on a node if the filesystem is a member of the following special list: NFS, Lustre, FUSE, CephFS, CIFS, PanFS, FhGFS, BeeGFS, GlusterFS, or GPFS. If this exclusion were not done, then all data on these imported filesystems would be wiped, since they are not part of the software image. The automatic exclusion for these imported filesystems does not rely on the `excludelist` values maintained by `CMDaemon`—instead, `CMDaemon` carries out the check on-the-fly when provisioning starts.

Statically-imported filesystems that have their mounts managed by 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 Bright View.

Filesystems mounted dynamically cannot have their appearance or disappearance detected reliably:

Any filesystem that may be imported via an auto-mount operation must therefore explicitly be excluded by the administrator manually adding the filesystem to the exclude list. This is to prevent an incorrect execution of `imageupdate`. Neglecting to do this may wipe out the filesystem, if it happens to be mounted in the middle of an `imageupdate` operation.

The `fstab` system is a statically mounting system, and not an auto-mounter: While `fstab` mounts filesystems automatically, system administrators should not confuse that with auto-mounting. Auto-mounting as provided by `autofs` is designed for the dynamic mounting of filesystems on demand by regular users. The `fstab` table is designed for mounting as carried out by the system, as occurs during boot, which is why it is regarded as a static, non-auto-mounting system.

Editing An Exclude List

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

```
cmsh -c "category use default; set excludelistupdate; commit"
```

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

```
Grouping→Node categories→Edit→Settings→Exclude list update
```

Provisioning Modifications Via `excludelistmanipulatescript`

Sometimes the administrator has a need to slightly modify the execution of exclude lists during provisioning. The `excludelistmanipulatescript` file takes as an input the exclude list inherited from a category, modifies it in some way, and then produces a new exclude list. Conceptually it is a bit like how an administrator might use `sed` if it worked without a pipe. As usual, setting it for node level overrides the category level.

A script that manipulates the exclude lists of a node can be specified as follows within `cmsh`:

```
[bright90]% device use node001
[bright90->device[node001]]% set excludelistmanipulatescript
(a vi session will start. A script is edited and saved)
[bright90->device[node001*]]% commit
```

The script can be as simple as:

Example

```
#!/bin/bash

echo "- *"
echo 'no-new-files: - *'
```

If provisioning a node from the head node, then the script modifies the node-provisioning exclude lists—`excludelistfullinstall`, `excludelistsyncinstall`, and `excludelistupdate`—so that they appear to contain these items only:

```
- *
no-new-files: - *
```

The provisioning node then excludes everything during provisioning.

Careful editing and testing of the script is advised. Saving a script with just a single whitespace, for example, usually has undesirable results.

A more useful script template is the following:

Example

```
#!/bin/bash

while read; do
  echo "$REPLY"
done

echo "# This and next line added by category excludelistmanipulatescript."
echo "# The command line arguments were: $@"
```

The provisioning exclude lists are simply read in, then sent out again without any change, except that the last two lines add comment lines to the exclude lists that are to be used.

Internally, the arguments taken by the `excludelistmanipulatescript` are the destination path and the sync mode (one of `install` | `update` | `full` | `grab` | `grabnew`). This can be seen in the output of `$@`, if running an `imageupdate` command to execute a dry run with the preceding example:

```
[bright90]% device use node001
[bright90->device[node001]]% get excludelistmanipulatescript
  (the script is put in)
[bright90->device[node001*]]% commit; imageupdate
Performing dry run (use synclog command to review result, then pass -w to perform real update)...
Wed Apr 15 04:55:46 2015 [notice] bright90: Provisioning started: sendi\
ng bright90:/cm/images/default-image to node001:/, mode UPDATE, dry run\
 = yes, no data changes!
[bright90->device[node001]]%
Wed Apr 15 04:55:51 2015 [notice] bright90: Provisioning completed: sen\
t bright90:/cm/images/default-image to node001:/, mode UPDATE, dry run \
 = yes, no data changes!
imageupdate [ COMPLETED ]
```

An excerpt from the sync log, after running the `synclog` command, then shows output similar to (some output elided):

```
...
- /cm/shared/*
- /cm/shared/
- /home/*
- /home/
- /cm/shared/apps/slurm/*
```

```
- /cm/shared/apps/slurm/
# This and next line added by category excludelistmanipulatescript.
# The command line arguments were: update /
```

```
Rsync output:
sending incremental file list
cm/local/apps/cmd/scripts/healthchecks/configfiles/
...
```

Here, the sync mode is `update` and the destination path is `"/"`. Which of the exclude lists is being modified can be determined by the `excludelistmanipulatescript` by parsing the sync mode.

The bash variable that accepts the exclude list text is set to a safely-marked form using curly braces. This is done to avoid expansion surprises, due to wild card characters in the exclude lists. For example, if `$REPLY` were used instead of `${REPLY}`, and the script were to accept an exclude list line containing `"/proc/*"`, then it would give quite confusing output.

Other Exclude List Handling Options

The `excludelistfailover` and `excludelistnormal` files: are two further exclude list files that modify standard provisioning behavior. These are discussed in section 14.4.8.

The `excludelistsnippets` tool: When synchronizing to a cloud director, or to an edge director, it is sometimes useful to exclude unneeded files and paths from the synchronization, in order to speed it up. The `excludelistsmanipulatescript` tool is powerful enough to do it, but it has some issues due to its power. For example, it is a script, which means that is called whenever it is used, and so uses up some extra resources. Also, it is a bit tricky to set up.

An easier way to manipulate exclude lists for the unneeded files and paths is via the `excludelistsnippets` tool, described in section 4.4.1 of the *Cloudbursting Manual*. This tool allows additional exclusion to be specified in a simpler way.

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

Exclude List State At Node Level

Exclude lists at category level and node level: An exclude list can be set at node level, as well as at category level. Roles and overlays can add implied exclude lists too.

At category level, an exclude list such as `excludelistfullinstall` can be set up explicitly with:

Example

```
[bright90->category[default]]% set excludelistfullinstall
...a text editor such as vi opens up and the list can be edited...
[bright90->category*[default*]]% commit
```

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

Example

```
[bright90->device[node001]]% set excludelistfullinstall
...a text editor such as vi opens up and the list can be edited...
[bright90->device*[node001*]]% commit
```

An exclude list that is not empty at node level overrules its corresponding category list. Exclude lists brought in via roles are however simply included in the exclude list.

The excludelist command: At node level it can be unclear what the resulting exclude list (“operational exclude list”) actually is. The exclude list state at node level can therefore be viewed using the `excludelist` command options. The `excludelist` command becomes active if a software image has been set at the node level.

- The `list` option to `excludelist` lists the source and destination paths:

Example

```
[bright90->device[node001]]% excludelist list
Source path (on the head node)  Destination path (on the node)
-----
/cm/images/default-image      /
```

- The `get` option to the `excludelist` command has synchronization mode and destination suboptions for a node.

Earlier on (page 200), the intention behind the various exclude lists, according to the type of update or synchronization, were distinguished.

The `excludelist get` command can have a destination path specified, and have the type of update or synchronization specified according to those distinguishing concepts.

The output to the `excludelist get` command then shows the operational exclude list as seen by a node for that path and for that update or synchronization.

Thus, for example:

- The full install operational exclude list for the path `/` on node `node001`, intended for a full installation to a node that is starting up, can be found as follows:

Example

```
[bright90->device[node001]]% excludelist get full /
# For details on the exclude patterns defined here please refer to
# the FILTER RULES section of the rsync man page.
#
# Files that match these patterns will not be installed onto the node.
- lost+found/
- /proc/*
- /sys/*
- /boot/efi

# extra defaults
- /proc/*
- /sys/*
```

- Similarly, the sync install operational exclude list for the path `/` on node `node001`, intended for a sync installation to a node that is starting up, can be found as follows:

```
[bright90->device[node001]]% excludelist get sync /

# For details on the exclude patterns defined here please refer to
# the FILTER RULES section of the rsync man page.
#
# Files that exist on a node and match one of these patterns will not be
# modified or deleted. Any files that match one of these patterns and that
```



```
# exist in the image but are absent on the node, will be copied to the node.
- /.autofsck
- /boot/grub*/grub.cfg
- /cm/local/apps/openldap/etc/certs/ldap.key
- /cm/local/apps/openldap/etc/certs/ldap.pem
- /data/*
- /home/*
...
```

- Other `excludelist` get options, besides `full` and `sync`, are:
 - * `grab` (a grab from a running node for a sync back to an existing image)
 - * `grabnew` (a grab from a running node for a full install to a new image)
 - * `update` (a sync update of a running node from an image).

All `excludelist` get options correspond to the intentions of the associated exclude list types as distinguished on page 200.

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

```
[bright90->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 bright90: Provisioning started on node node001
[bright90->device]% imageupdate -n node001: image update in progress ...
[bright90->device]%
Tue Jan 11 12:13:44 2011 bright90: 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

```
[bright90->device]% imageupdate -n node001 -w
Provisioning started on node node001
node001: image update in progress ...
[bright90->device]% Provisioning completed on node node001
```

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

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

```
Devices→Nodes[node001]→Edit↓Software image→Update node
```

5.6.4 Updating Running Nodes: Considerations

An attempt to update the image on a running node can run into some issues:

- Updating an image via `cmsh` or Bright View automatically updates the provisioners first via the `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 `provisioningnodeautoupdatetimetype` parameter (section 5.2.4).

So, with the default setting of 5 minutes, if there has been a new image created within the last 5 minutes, then provisioners do not get the updated image when doing the updates, which means that nodes in turn do not get those updates. Running the `updateprovisioners` command just before running the `imageupdate` command therefore usually makes sense.

- By default, Bright Cluster Manager does not allow provisioning if `automount` (page 725) is running.
- Also, when updating services, the services on the nodes may not restart since the `init` process may not notice the replacement.

For these reasons, especially for more extensive changes, it can be safer for the administrator to simply reboot the nodes instead of using `imageupdate` to provision the images to the nodes. A reboot by default ensures that a node places the latest image with an `AUTO` install (section 5.4.7), and restarts all services.

The `Reinstall node` option, which can be run, for example, on a node `node001`, using a clickpath of `Devices→Nodes[node001]→Edit↓Software image→Reinstall node` also does the same as a reboot with default settings, except for that it unconditionally places the latest image with a `FULL` install, and so may take longer to complete.

5.7 Adding New Nodes

How the administrator can add a single node to a cluster is described in section 1.3 of the *Installation Manual*. This section explains how nodes can be added in ways that are more convenient for larger numbers of nodes.

5.7.1 Adding New Nodes With `cmsh` And Bright View Add Functions

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

Example

```
[bright90->device]% add physicalnode node002 10.141.0.2
[bright90->device*[node002*]% commit
```

The Bright View equivalent of this is following the clickpath:

`Devices→Nodes→Add→Settings→Hostname`

then adding the value `node002` to `Hostname`, and saving it.

When adding the node objects in `cmsh` and Bright View, some values (IP addresses for example) may need to be filled in before the object validates. For regular nodes, there should be an interface and an IP address for the network that it boots from, as well as for the network that manages the nodes. A regular node typically has only one interface, which means that the same interface provides boot and management services. This interface is then the boot interface, `BOOTIF`, during the pre-init stage, but is also the management interface, typically `eth0` or whatever the device is called, after the pre-init stage. The IP address for `BOOTIF` is normally provided via DHCP, while the IP address for the management interface is set to a static IP address that is set via `cmsh` or Bright View by the administrator.

Adding new node objects as “placeholders” can also be done from `cmsh` or Bright View. By placeholders, here it is meant that an incomplete node object is set. For example, sometimes it is useful to create a node object with the MAC address setting unfilled because it is still unknown. Why this can be useful is covered shortly.

5.7.2 Adding New Nodes With The Node Creation Wizard

Besides adding nodes using the `add` command of `cmsh` or the `Add` button of Bright View as in the preceding text, there is also a Bright View wizard that guides the administrator through the process—the *node creation wizard*. This is useful when adding many nodes at a time. It is available via the clickpath:

`Devices→Nodes→Create nodes`

This wizard should not be confused with the closely-related node *identification* resource described in section 5.4.2, which identifies unassigned MAC addresses and switch ports, and helps assign them node names.

- The node *creation* wizard creates an object for nodes, assigns them node names, but it leaves the MAC address field for these nodes unfilled, keeping the node object as a “placeholder”.
- The node *identification* resource assigns MAC addresses so that node names are associated with a MAC address.

If a node is left with an unassigned MAC address—that is, in a “placeholder” state—then it means that when the node starts up, the provisioning system lets the administrator associate a MAC address and switch port number at the node console for the node. This occurs when the node-installer reaches the node configuration stage during node boot as described in section 5.4.2. This is sometimes preferable to associating the node name with a MAC address remotely with the node identification resource.

The node creation wizard sets IP addresses for the nodes. By default it starts IP addressing for the new nodes by guessing an appropriate value from the node names. The administrator can override this starting value in the main screen of the wizard by modifying it in the header section of the IP addresses column (figure 5.18).

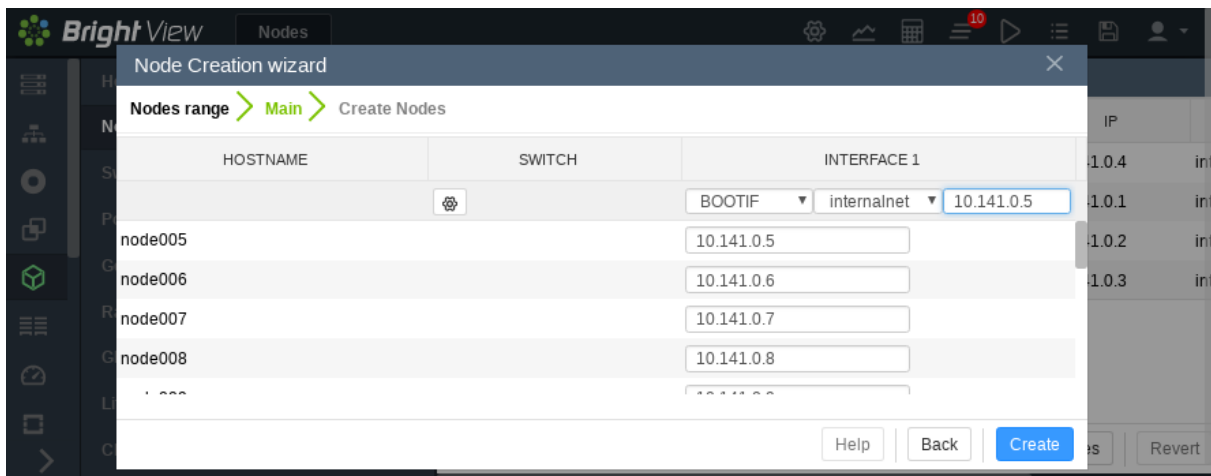


Figure 5.18: Node Creation Wizard: Setting Interfaces

The `cmsh` equivalent of the node creation wizard is the `foreach` loop with the `-clone` option acting on a node (section 2.5.5).

5.8 Troubleshooting The Node Boot Process

During the node boot process there are several common issues that can lead to an unsuccessful boot. This section describes these issues and their solutions. It also provides general hints on how to analyze boot problems.

Before looking at the various stages in detail, the administrator may find that simply updating software or firmware may fix the issue. In general, it is recommended that all available updates are deployed on a cluster.

- Updating software is covered in Chapter 11.
 - On the head node, the most relevant software can be updated with `yum`, `zypper`, or `apt`, as explained in section 11.2. For example, with `yum`:

Example

```
yum update cmdaemon node-installer
```

- Similarly for the software image, the most relevant software can be updated too. This is done via a procedure involving a chroot installation., as described in section 11.4. If using yum, then the update can be carried out within the image, *<software image>*, with:

Example

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

- UEFI or BIOS firmware should be updated as per the vendor recommendation

The various stages that may fail during node boot are now examined.

5.8.1 Node Fails To PXE Boot

Possible reasons to consider if a node is not even starting to network boot (PXE boot for x86 nodes) in the first place:

- DHCP may not be running. A check can be done to confirm that DHCP is running on the internal network interface (usually eth0):

```
[root@bright90 ~]# ps u -C dhcpd
USER PID %CPU %MEM VSZ  RSS TTY STAT START TIME COMMAND
root 2448 0.0 0.0 11208 436 ?   Ss   Jan22 0:05 /usr/sbin/dhcpd eth0
```

This may indicate that Node booting is disabled in Bright View (figure 3.5, page 64) and needs to be enabled. The equivalent in cmsh is to check if the response to:

```
cmsh -c "network use internalnet; get nodebooting"
```

needs to be set to yes.

- The DHCP daemon may be “locked down” (section 3.2.1, figure 3.5, table 3.2.1). New nodes are granted leases only after `lockdowndhcpd` is set to `no` in cmsh, or `Lock down dhcpd` is disabled in Bright View for the network.
- A rogue DHCP server may be running. If there are all sorts of other machines on the network the nodes are on, then it is possible that there is a rogue DHCP server active on it, perhaps on an IP address that the administrator has forgotten, and interfering with the expected PXE booting. Such stray DHCP servers should be eliminated.

In such a case, removing all the connections and switches and just connecting the head node directly to a problem node, NIC-to-NIC, should allow a normal network boot to happen. If a normal network boot then does happen, it indicates the problem is indeed due to a rogue DHCP server on the more-connected network.

- The boot sequence may be set wrongly in the BIOS. The boot interface should normally be set to be the first boot item in the BIOS.
- The node may be set to boot from UEFI mode. If UEFI mode has a buggy network boot implementation, then it may fail to network boot. For x86 nodes, setting the node to PXE boot using the legacy BIOS mode can be tried instead, or perhaps the UEFI firmware can be updated.
- There may 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. With STP on, nodes may randomly fail to network boot.

- The cable may be connected to the wrong interface. By default, on the head node, for a type 1 network, the first consistent network device name, for example `eno1`, is normally assigned the internal network interface, and the second one, for example `en02`, is assigned the external network interface. However, the following possibilities should be considered during troubleshooting:

- The two interfaces can be confused when physically viewing them and a connection to the wrong interface can therefore be made.
- It is also possible that the administrator has changed the default assignment.
- The interface may have been set by the administrator to follow the network device naming scheme that has been used prior to RHEL7. Interfaces with names such as `eth0` and `eth1` on the head node are suggestive of this. The problem with the pre-RHEL7 scheme is that it can sometimes lead to network interfaces swapping after reboot, which is why the scheme is no longer recommended. The workaround for this issue in pre-RHEL7 schemes was to define a persistent name in the `udev` ruleset for network interfaces.

In Bright Cluster Manager 9.0 onwards, the default scheme is the consistent network device naming scheme, and it is recommended.

Interface Naming Conventions Post-RHEL7 (Recommended)

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/ch-Consistent_Network_Device_Naming.html describes the consistent network device scheme for interfaces post-RHEL7. This scheme sets an interface assignment on iPXE boot for multiple interfaces that is also valid by default during the very first iPXE boot. This means that an administrator can know which interface is used for provisioning and can connect the provisioning cable accordingly.

Reverting To The Pre-RHEL7 Interface Naming Conventions (Not Recommended)

To revert to the pre-RHEL7 behavior, the text:

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

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

- * The `biosdevname` parameter only works if the `dev helper` is installed. The `dev helper` is available from the `biosdevname` RPM package. The parameter also requires that the system supports SMBIOS 2.6 or ACPI DSM.
- * The `net.ifnames` parameter is needed if `biosdevname` is not installed.

Example

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

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

```
[root@bright90 ~]# cp --preserve /boot/grub2/grub.cfg /boot/grub2/grub.cfg.orig
```

The GRUB configuration should be generated with:

```
[root@bright90 ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

If for some reason the administrator would like to carry out the pre-RHEL7 naming convention on a regular node, then the text `net.ifnames=0 biosdevname=0` can be appended to the `kernelparameters` property, for an image selected from `softwareimage` mode.

Example

```
[bright90->softwareimage]% list
Name (key)          Path
-----
default-image      /cm/images/default-image
openstack-image    /cm/images/openstack-image
[bright90->softwareimage]% use default-image
[bright90->softwareimage[default-image]]% append kernelparam\
ters " net.ifnames=0 biosdevname=0"
[bright90->softwareimage*[default-image*]]% commit
```

The `append` command requires a space at the start of the quote, in order to separate the kernel parameters from any pre-existing ones.

- The TFTP server that sends out the image may have hung. During a normal run, an output similar to this appears when an image is in the process of being served:

```
[root@bright90 ~]# ps ax | grep [t]ftp
7512 ?      Ss      0:03 in.tftpd --maxthread 500 /tftpboot
```

If the TFTP server is in a zombie state, the head node should be rebooted. If the TFTP service hangs regularly, there is likely a networking hardware issue that requires resolution.

Incidentally, grepping the process list for a TFTP service returns nothing when the head node is listening for TFTP requests, but not actively serving a TFTP image. This is because the TFTP service runs under `xinet.d` and is called on demand. Running

```
[root@bright90 ~]# chkconfig --list
```

should include in its output the line:

```
tftp:      on
```

if TFTP is running under `xinet.d`.

- The switchover process from TFTP to HTTP may have hung. During a normal provisioning run, assuming that `CMDaemon` uses the default `bootloaderprotocol` setting of HTTP, then TFTP is used to load the initial boot loader, but the kernel and ramdisk are loaded up via HTTP for speed. Some hardware has problems with switching over to using HTTP.

In that case, setting `bootloaderprotocol` to TFTP keeps the node using TFTP for loading the kernel and ramdisk, and should work. Another possible way to solve this is to upgrade the PXE boot BIOS to a version that does not have this problem.

ARMv8 hardware can only boot only via TFTP.

Setting `bootloaderprotocol` to HTTPS only works for some special hardware.

- Sometimes a manufacturer releases hardware with buggy drivers that have a variety of problems. For instance: Ethernet frames may be detected at the interface (for example, by `ethtool`), but TCP/IP packets may not be detected (for example, by `wireshark`). In that case, the manufacturer should be contacted to upgrade their driver.
- The interface may have a hardware failure. In that case, the interface should be replaced.

5.8.2 Node-installer Logging

If the node manages to get beyond the net booting stage to the node-installer stage, then the first place to look for hints on node boot failure is usually the node-installer log file. The node-installer runs on the node that is being provisioned, and sends logging output to the syslog daemon running on that node. This forwards all log data to the IP address from which the node received its DHCP lease, which is typically the IP address of the head node or failover node. In a default 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

```
[bright90->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@bright90 ~]# cmsn -c "softwareimage islocked"
Name          Locked
-----
default-image yes
[root@bright90 ~]# cmsn -c "softwareimage unlock default-image"
[root@bright90 ~]# cmsn -c "softwareimage islocked"
Name          Locked
-----
default-image no
```

The node automatically picks up the image after it is unlocked.

Optionally, extra log information can be written by enabling debug logging, which sets the syslog importance level at `LOG_DEBUG`. To enable debug logging, the `debug` field is changed in `/cm/node-installer/scripts/node-installer.conf`.

For the `node-installer.conf` file in `multidistro` and `multiarch` (section 11.7) configurations, the directory path `/cm/node-installer` takes the form:

```
/cm/node-installer-<distribution>-<architecture>
```

The values for `<distribution>` and `<architecture>` can take the values outlined on page 472.

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 13.7), to allow the viewing of console messages directly. A further depth in logging can be achieved by setting the kernel option `loglevel=N`, where `N` is a number from 0 (`KERN_EMERG`) to 7 (`KERN_DEBUG`).

One possible point at which the node-installer can fail on some hardware is if SOL (section 13.7) is enabled in the BIOS, but the hardware is unable to cope with the flow. The installation can freeze completely at that point. This should not be confused with the viewing quirk described in section 13.7.4, even though the freeze typically appears to take place at the same point, that point being when the console shows "freeing unused kernel memory" as the last text. One workaround to the freeze would be to disable SOL.

5.8.3 Provisioning Logging

The provisioning system sends log information to the CMDaemon log file. By default this is in `/var/log/cmdaemon` on the local host, that is, the provisioning host. The host this log runs on can be configured with the CMDaemon directive `SyslogHost` (Appendix C).

The image synchronization log file can be retrieved with the `synclog` command running from device mode in `cmsh`. Hints on provisioning problems are often found by looking at the tail end of the log.

If the tail end of the log shows an `rsync` exit code of 23, then it suggests a transfer error. Sometimes the cause of the error can be determined by examining the file or filesystem for which the error occurs. For the `rsync` transport, logs for node installation are kept under `/var/spool/cmd/`, with a log written for each node during provisioning. The name of the node is set as the prefix to the log name. For example `node002` generates the log:

```
/var/spool/cmd/node002-\.rsync
```

5.8.4 Ramdisk Fails During Loading Or Sometime Later

One issue that may come up after a software image update via `yum`, `zypper`, or `apt` (section 11.4), 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 the `Recreate Initrd` command via the clickpaths:

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

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

Another, somewhat related possible cause of a halt at this stage, is that the kernel modules that are to be loaded may have been specified at a wrongly by the administrator in the hierarchy of software image, category, or node (page 162). A check of the kernel modules specified in `softwareimage` mode, `category` mode, or `device` mode (for the particular node) may reveal a misconfiguration.

5.8.5 Ramdisk Cannot Start Network

The ramdisk must activate the node's network interface in order to fetch the node-installer. To activate the network device, the correct kernel module needs to be loaded. If this does not happen, booting fails, and the console of the node displays something similar to figure 5.19.


```
Creating initial device nodes
Setting up hotplug.
Creating block device nodes.
Loading ehci-hcd.ko module
Loading ohci-hcd.ko module
Loading uhci-hcd.ko module
Loading jbd.ko module
Loading ext3.ko module
Loading sunrpc.ko module
Loading nfs_acl.ko module
Loading fscache.ko module
Loading lockd.ko module
Loading nfs.ko module
Loading scsi_mod.ko module
Loading sd_mod.ko module
Loading libata.ko module
Loading ahci.ko module
Waiting for driver initialization.
Creating root device.
Finished original ramdisk.
Can't configure the ethernet device used for booting.
You should probably insert the correct kernel module into the ramdisk.
Boot failed.
/bin/sh: can't access tty: job control turned off
#
```

Figure 5.19: No Network Interface

To solve this issue the correct kernel module should be added to the software image's kernel module configuration (section 5.3.2). For example, to add the e1000 module to the default image using `cmsh`:

Example

```
[mc]% softwareimage use default-image
[mc->softwareimage[default-image]]% kernelmodules
[mc->softwareimage[default-image]->kernelmodules]% add e1000
[mc->softwareimage[default-image]->kernelmodules[e1000]]% commit
Initial ramdisk for image default-image was regenerated successfully
[mc->softwareimage[default-image]->kernelmodules[e1000]]%
```

After committing the change it typically takes about a minute before the initial ramdisk creation is completed via a `mkinitrd` run by `CMDaemon`.

5.8.6 Node-Installer Cannot Create Disk Layout

When the node-installer is not able to create a drive layout it displays a message similar to figure 5.20. The node-installer log file (section 5.8.2) contains something like:

```
Mar 24 13:55:31 10.141.0.1 node-installer: Installmode is: AUTO
Mar 24 13:55:31 10.141.0.1 node-installer: Fetching disks setup.
Mar 24 13:55:31 10.141.0.1 node-installer: Checking partitions and
filesystems.
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/sda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/hda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Can not find device(s) (/dev/sda /dev/hda).
Mar 24 13:55:32 10.141.0.1 node-installer: Partitions and/or filesystems
are missing/corrupt. (Exit code 4, signal 0)
Mar 24 13:55:32 10.141.0.1 node-installer: Creating new disk layout.
```

```

Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/sda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Detecting device '/dev/hda':
not found
Mar 24 13:55:32 10.141.0.1 node-installer: Can not find device(s) (/dev/sda /dev/hda).
Mar 24 13:55:32 10.141.0.1 node-installer: Failed to create disk layout.
(Exit code 4, signal 0)
Mar 24 13:55:32 10.141.0.1 node-installer: There was a fatal problem. This node can not be\
installed until the problem is corrected.

```

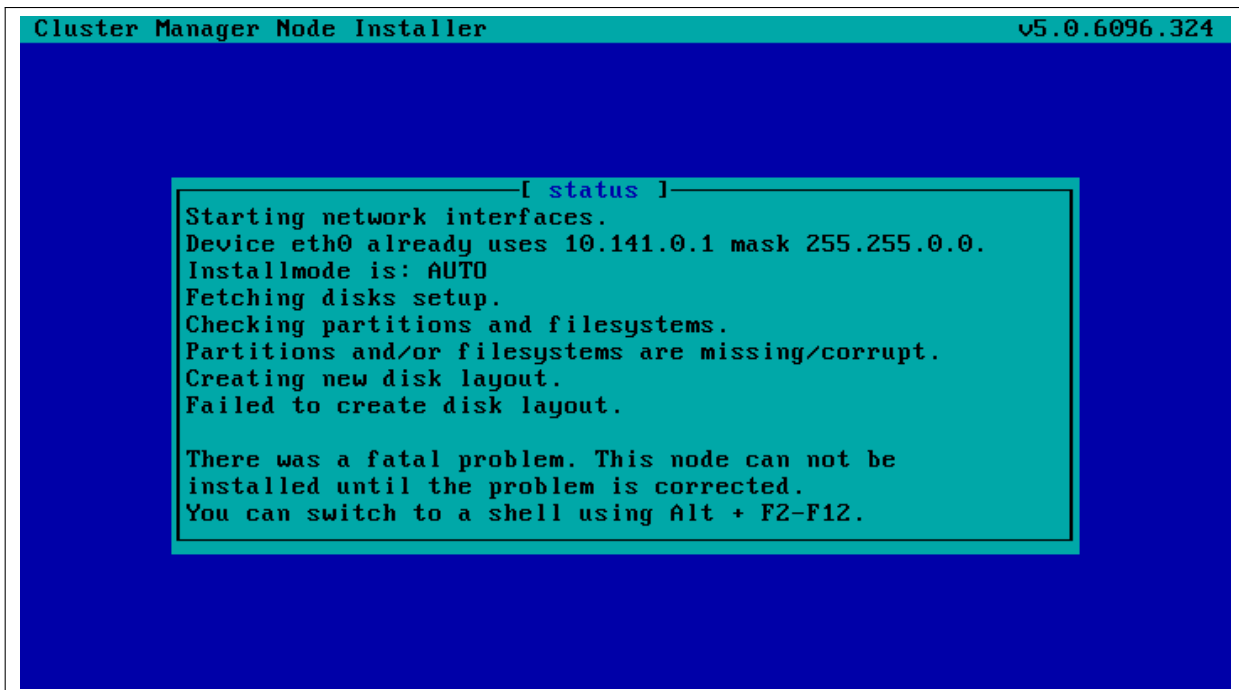


Figure 5.20: No Disk

Disk layout failures can have several reasons.

BIOS And Order Issues

One reason may be that the drive may be disabled in the BIOS. It should be enabled.

Another reason may be that the drive order was changed. This could happen if, for example, a defective motherboard has been replaced. The drive order should be kept the same as it was before a motherboard change.

Read-only Mode Issues

Another reason may be due to SSDs that have a hardware jumper or toggle switch that sets a drive to read-only mode. A read-only mode drive will typically fail at this point. The drive should be made writeable.

Hardware Issues

If the node-installer log for the node shows lines with the text `Input/output error`, then it generally indicates a hardware issue. Possible hardware issues include:

- a drive failure
- a faulty cable between storage and controller

- a faulty storage controller
- a faulty backplane in the server

If the node has enough RAM, then it is possible to boot up the node up as a diskless node, to carry out further diagnosis with disk tools such as `smartmontools`.

Software Driver Issues

One of the most common software issues is that the correct storage driver is not being loaded. To solve this issue, the correct kernel module should be added to the software image's kernel module configuration (section 5.3.2).

Experienced system administrators work out what drivers may be missing by checking the results of hardware probes. For example, going into the node-installer shell using Alt-F2, and then looking at the output of `lspci`, shows a list of hardware detected in the PCI slots and gives the chipset name of the storage controller hardware in this case:

Example

```
[<installer> root@node001 ~]# lspci | grep SCSI
00:10.0 Serial Attached SCSI controller: LSI Logic / Symbios Logic SAS2\
008 PCI-Express Fusion-MPT SAS-2 [Falcon] (rev 03)
```

The next step is to Google with likely search strings based on that output.

The Linux Kernel Driver DataBase (LKDDb) is a hardware database built from kernel sources that lists driver availability for Linux. It is available at <http://cateee.net/lkddb/>. Using the Google search engine's "site" operator to restrict results to the cateee.net web site only, a likely string to try might be:

Example

```
SAS2008 site:cateee.net
```

The search result indicates that the `mpt2sas` kernel module needs to be added to the node kernels. A look in the modules directory of the software image shows if it is available:

Example

```
find /cm/images/default-image/lib/modules/ -name "*mpt2sas*"
```

If it is not available, the driver module must then be obtained. If it is a source file, it will need to be compiled. By default, nodes run on standard distribution kernels, so that only standard procedures need to be followed to compile modules.

If the module is available, it can be added to the default image, by using `cmsh` in `softwareimage` mode to create the associated object. The object is given the same name as the module, i.e. `mp2sas` in this case:

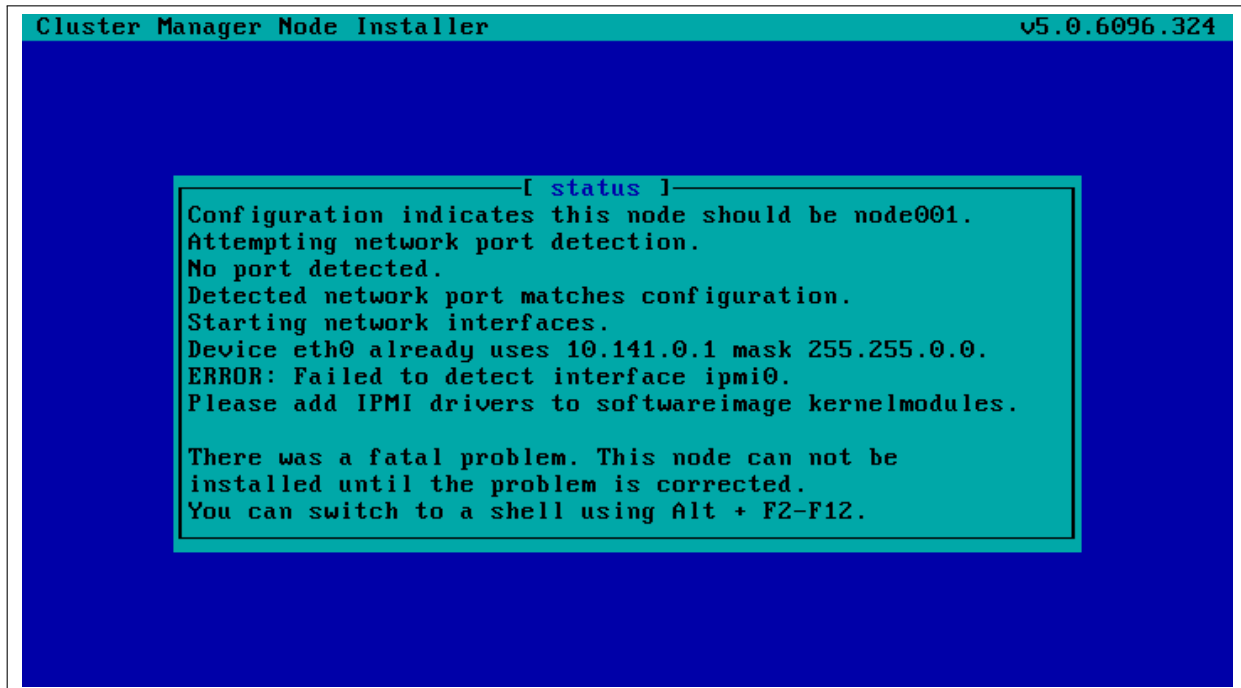
Example

```
[bright90]% softwareimage use default-image
[bright90->softwareimage[default-image]]% kernelmodules
[bright90->softwareimage[default-image]->kernelmodules]% add mpt2sas
[bright90->softwareimage[default-image]->kernelmodules*[mpt2sas*]]% commit
[bright90->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:54:52 2011 [notice] bright90: Initial ramdisk for image de\
fault-image is being generated
[bright90->softwareimage[default-image]->kernelmodules[mpt2sas]]%
Thu May 19 16:55:43 2011 [notice] bright90: Initial ramdisk for image de\
fault-image was regenerated successfully.
[bright90->softwareimage[default-image]->kernelmodules[mpt2sas]]%
```

After committing the change it can take some time before ramdisk creation is completed—typically about a minute, as the example shows. Once the ramdisk is created, the module can be seen in the list displayed from `kernelmodules` mode. On rebooting the node, it should now continue past the disk layout stage.

5.8.7 Node-Installer Cannot Start BMC (IPMI/iLO) Interface

In some cases the node-installer is not able to configure a node's BMC interface, and displays an error message similar to figure 5.21.



```

Cluster Manager Node Installer v5.0.6096.324

[ status ]
Configuration indicates this node should be node001.
Attempting network port detection.
No port detected.
Detected network port matches configuration.
Starting network interfaces.
Device eth0 already uses 10.141.0.1 mask 255.255.0.0.
ERROR: Failed to detect interface ipmi0.
Please add IPMI drivers to softwareimage kernelmodules.

There was a fatal problem. This node can not be
installed until the problem is corrected.
You can switch to a shell using Alt + F2-F12.

```

Figure 5.21: No BMC Interface

Usually the issue can be solved by adding the correct BMC (IPMI/iLO) kernel modules to the software image's kernel module configuration. However, in some cases the node-installer is still not able to configure the BMC interface. If this is the case the BMC probably does not support one of the commands the node-installer uses to set specific settings, or there may be a hardware glitch in the BMC.

The `setupBmc` Node-Installer Configuration Setting

To solve this issue, setting up BMC interfaces can be disabled globally by setting the `setupBmc` field to `false` in the node-installer configuration file `/cm/node-installer/scripts/node-installer.conf` (for multiarch/multidistro configurations the path takes the form: `/cm/node-installer-<distribution>-<architecture>/scripts/node-installer.conf`).

Doing this disables configuration of all BMC interfaces by the node-installer. A custom `finalize` script (Appendix E) can then be used to run the required commands instead.

The `setupBmc` field in the node-installer should not be confused with the `SetupBMC` directive in `cmd.conf` (Appendix C). The former is about enabling the BMC interface, while the latter is about enabling automated passwords to the BMC interface (an interface that must of course be enabled in the first place to work).

The `failOnMissingBmc` Node-Installer Configuration Setting

If the kernel modules for the BMC are loaded up correctly, and the BMC is configured, but it is not detected by the node-installer, then the node-installer halts by default. This corresponds to the set-

ting `failOnMissingBmc = true` in the node-installer configuration file `/cm/node-installer/scripts/node-installer.conf`. Toggling this to `false` skips BMC network device detection, and lets the node-installer continue past the BMC detection and configuration stage. This can be convenient, for example, if the BMC is not yet configured and the aim is to get on with setting up the rest of the cluster.

The `failOnFailedBmcCommand` Node-Installer Configuration Setting

If a BMC command fails, then the node-installer by default terminates node installation. The idea behind this is to allow the administrator to fix the problem. Sometimes, however, hardware can wrongly signal a failure. That is, it can signal a false failure, as opposed to a true failure.

A common case is the case of `ipmitool`. `ipmitool` is used by Bright Cluster Manager to configure the BMC. With most hardware vendors it works as expected, signaling success and failure correctly. As per the default behavior: with success, node installation proceeds, while with failure, it terminates.

With certain hardware vendors however `ipmitool` fails with an exit code 1, even though the BMC is properly configured. Again, as per the default behavior: success has node installation proceed, while failure has node installation terminate. Only this time, because the failure signal is incorrect, the termination on failure is also incorrect behavior.

To get around the default behavior for false failure cases, the administrator can force the node-installer to set the value of `failOnFailedBmcCommand` to `false` in the node-installer configuration file `/cm/node-installer/scripts/node-installer.conf` (for multi-arch/multidistro configurations the path takes the form: `/cm/node-installer-<distribution>-<architecture>/scripts/node-installer.conf`). The installation then skips past the false failure.

BMC Hardware Glitch And Cold Reset

Sometimes, typically due to a hardware glitch, a BMC can get into a state where it is not providing services, but the BMC is still up (responding to pings). Contrariwise, a BMC may not respond to pings, but still respond to IPMI commands. A fix for such glitchy states is usually to power cycle the BMC. This is typically done, either physically, or by using a BMC management tool such as `ipmitool`.

Physically resetting the power supply to the BMC is done typically by pulling the power cable out and then pushing it in again. For typical rack-based servers the server can just be pulled out and in again. Just doing a shutdown of the server with the power cable still in place normally does not power down the BMC.

BMC management does allow the BMC to power down and be reset from software, without having to physically handle the server. This software-based *cold reset* is a BIOS-manufacturer-dependent feature. A popular tool used for managing BMCs that can do such a cold reset is `ipmitool`. This can be run remotely, but also on the node console if the node cannot be reached remotely.

With `ipmitool`, a cold reset is typically carried out with a command such as:

```
[root@bright90 ~]# module load ipmitool
[root@bright90 ~]# ipmitool -U <bmcusername> -P <bmcpassword> -H <host IP> -I lanplus mc reset cold
```

The values for `<bmcusername>` and `<bmcpassword>` can be obtained as shown in section 3.7.2.

BMC Troubleshooting With The System Event Log

The System Event Log (SEL) can be read with:

```
[root@bright90 ~]# module load ipmitool
[root@bright90 ~]# ipmitool -U <bmcusername> -P <bmcpassword> -H <host IP> -I lanplus sel list
```

The timestamped output can be inspected for errors related to the CPU, ECC, or memory.

Other BMC Troubleshooting

Some more specific commands for handling IPMI might be via the `service ipmi <option>` commands, which can show the IPMI service has failed to start up:

Example

```
[root@bright90 ~]# service ipmi status
Redirecting to /bin/systemctl status ipmi.service
ipmi.service - IPMI Driver
Loaded: loaded (/usr/lib/systemd/system/ipmi.service; disabled; vendor preset: enabled)
Active: inactive (dead)
```

In the preceding session the driver has simply not been started up. It can be started up with the start option:

Example

```
[root@bright90 ~]# 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@bright90 ~]# 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 bright90 systemd[1]: Starting IPMI Driver...
Dec 19 14:34:27 bright90 openipmi-helper[8930]: Startup failed.
Dec 19 14:34:27 bright90 systemd[1]: ipmi.service: main process exited, code=exited, status=1/\
FAILURE
Dec 19 14:34:27 bright90 systemd[1]: Failed to start IPMI Driver.
Dec 19 14:34:27 bright90 systemd[1]: Unit ipmi.service entered failed state.
Dec 19 14:34:27 bright90 systemd[1]: ipmi.service failed.
```

Further details can be found in the journal:

Example

```
[root@bright90 ~]# journalctl -xe | grep -i ipmi
...
-- Unit ipmi.service has begun starting up.
Dec 19 14:34:27 bright90 kernel: ipmi message handler version 39.2
Dec 19 14:34:27 bright90 kernel: IPMI System Interface driver.
Dec 19 14:34:27 bright90 kernel: ipmi_si: Unable to find any System Interface(s)
Dec 19 14:34:27 bright90 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@bright90 ~]# 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 bright90 systemd[1]: Starting IPMI Driver...
Dec 19 14:37:10 bright90 systemd[1]: Started IPMI Driver.
```

Sometimes the issue may be an incorrect networking specification for the BMC interfaces. MAC and IP details that have been set for the BMC interface can be viewed with the `lan print` option to `ipmitool` if the service has been started:

Example

```
[root@bright90 ~]# module load ipmitool
[root@bright90 ~]# 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
Gratuitous ARP Intrvl : 0.0 seconds
Default Gateway IP   : 93.184.216.1
Default Gateway MAC  : 00:00:00:00:00:00
Backup Gateway IP    : 0.0.0.0
Backup Gateway MAC   : 00:00:00:00:00:00
802.1q VLAN ID       : Disabled
802.1q VLAN Priority : 0
RMCP+ Cipher Suites : 0,1,2,3,4,6,7,8,9,11,12,13,15,16,17,18
Cipher Suite Priv Max : caaaaaaaaaaaaaa
                    : X=Cipher Suite Unused
                    : c=CALLBACK
                    : u=USER
                    : o=OPERATOR
                    : a=ADMIN
                    : O=OEM
```

During normal operation the metrics (Appendix G) displayed by 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@bright90 ~]# module load ipmitool
[root@bright90 ~]# ipmitool sensor list all
```

```
# ipmitool sensor list
Ambient Temp | 22.000 | degrees C | ok | na | na | na | 38.000 | 41.000 | 45.000
AVG Power | 300.000 | Watts | ok | na | na | na | na | na | na
Fan 1 Tach | 4125.000 | RPM | ok | na | 750.000 | na | na | na | na
...
```


6

User Management

Users and groups for the cluster are presented to the administrator in a single system paradigm. That is, if the administrator manages them with the 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 Bright View or `cmsh`, using profiles with sets of tokens.

6.1 Managing Users And Groups With Bright View

Within Bright View:

- users can be managed via the clickpath `Identity Management`→`Users`
- groups can be managed via the clickpath `Identity Management`→`Groups`.

For users (figure 6.1) the LDAP entries for regular users are displayed. These entries are editable and each user can then be managed in further detail.

There is already one user on a newly installed 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, `Revert`, `Add`, `Delete`, `Save`, and `Edit` available in the window:

1. `Add`: allows users to be added via a dialog (figure 6.2). These additions can be committed via the `Save` button.

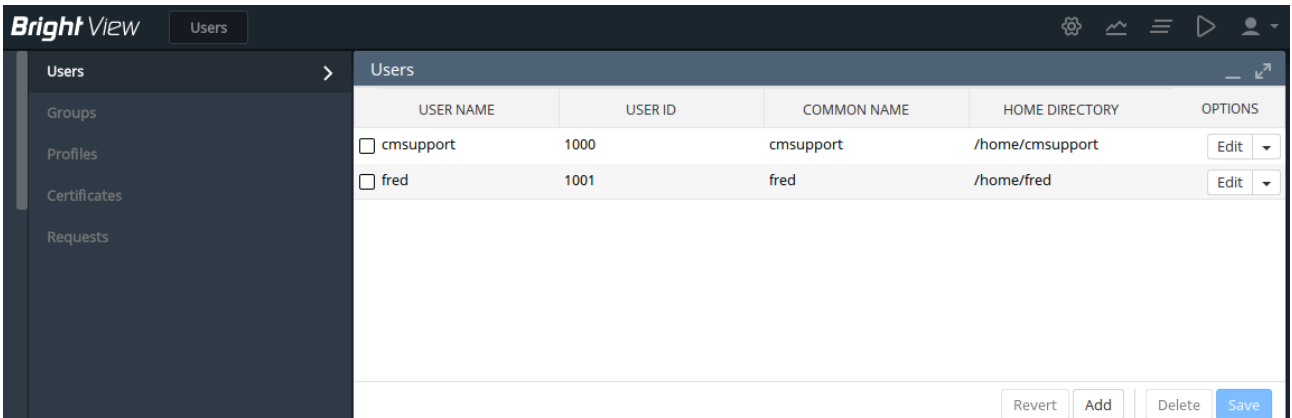


Figure 6.1: Bright View User Management

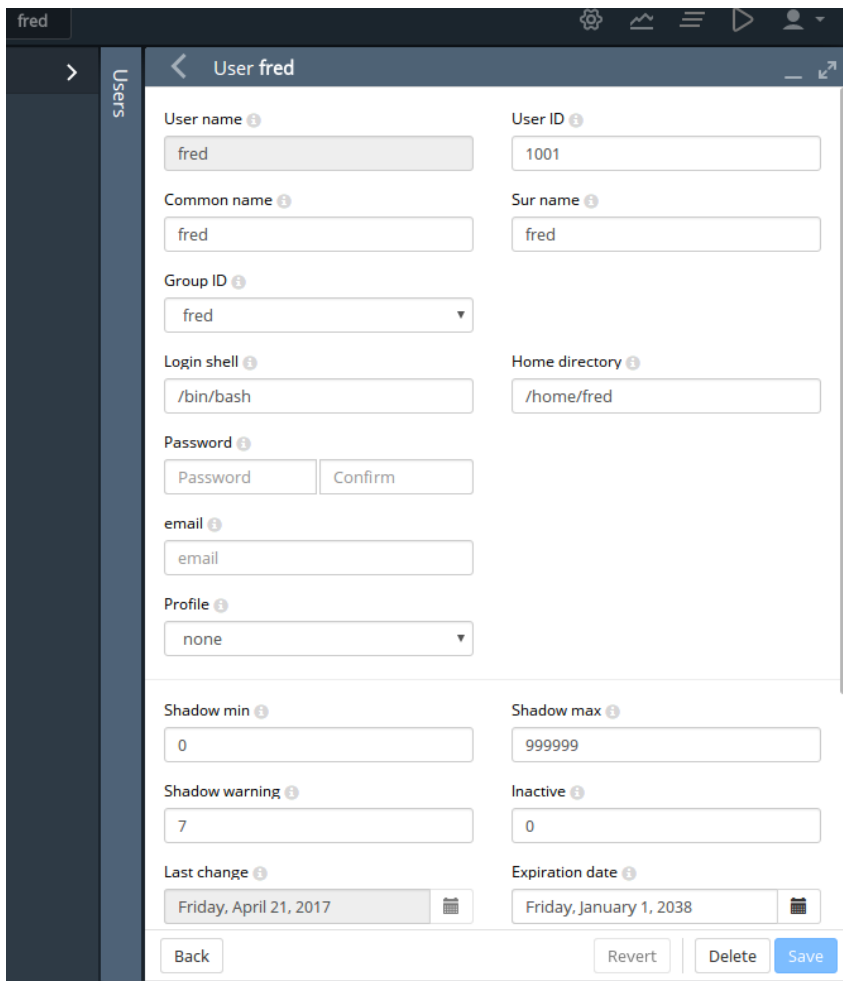


Figure 6.2: Bright View User Management: Add Dialog

An explanation of the less obvious items in the dialog follows:

- Shadow 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

- **Inactive:** 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
- **Profile:** The preconfigured capability that the user is assigned. Available settings are:
 - **admin:** Allows the user to run Bright View with the same privileges as user `admin`
 - **cloudjob:** Allows the user to run `cmjob`, the cloud job submission utility (section 4.7 of the *User Manual*, and section 4.3 of the *Cloudbursting Manual*)
 - **portal:** Allows the user to access the user portal
 - **readonly:** Allows the user to run Bright View without the ability to modify settings.
 - **none:** (default). Prevents the user from using Bright View

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 60 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.
- 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. **Delete:** deletes selected rows of users. By default, their home directories are not deleted.

Group management in Bright View is carried out via the clickpath `Identity Management`→`Groups`. Clickable LDAP object entries for regular groups then show up, similar to the user entries already covered. Management of these entries is done with the same button functions as for user management.

6.2 Managing Users And Groups With `cmsh`

User management tasks as carried out by Bright View in section 6.1, can be carried with the same end results in `cmsh` too.

A `cmsh` session is run here in order to cover the functions corresponding to the user management functions of Bright View of section 6.1. These functions are run from within the user mode of `cmsh`:

Example

```
[root@bright90 ~]# cmsh
[bright90]% user
[bright90->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

```
[bright90->user]% add maureen
[bright90->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

```
[bright90->user*[maureen*]]% modified
State  Type                Name
-----
+      User                maureen
```

This corresponds roughly to what is displayed by the `Unsaved` entities icon in the top right corner of the Bright View standard display (figure 12.5).

Running `show` at this point reveals a user name entry, but empty fields for the other properties of user `maureen`. So the account in preparation, while it is modified, is clearly not yet ready for use:

Example

```
[bright90->user*[maureen*]]% show
Parameter                Value
-----
Accounts
Managees
Name                      maureen
Primary group
Revision
Secondary groups
ID
Common name
Surname
Group ID
Login shell
Home directory
Password                  < not set >
email
Profile
Write ssh proxy config    no
Shadow min                0
Shadow max                999999
Shadow warning            7
Inactive                  0
Last change               1970/1/1
Expiration date           2038/1/1
Project manager           <submode>
Notes                     <OB>
```

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

```
[bright90->user*[maureen*]]% commit
[bright90->user[maureen]]% show
Parameter                                Value
-----
Accounts
Managees
Name                                       maureen
Primary group                             maureen
Revision
Secondary groups
ID                                         1001
Common name                               maureen
Surname                                   maureen
Group ID                                  1001
Login shell                               /bin/bash
Home directory                            /home/maureen
Password                                  *****
email
Profile
Write ssh proxy config                    no
Shadow min                                0
Shadow max                                999999
Shadow warning                             7
Inactive                                   0
Last change                               2021/3/12
Expiration date                            2038/1/1
Project manager                            <submode>
Notes                                      <OB>
```

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

```
[bright90->user[maureen]]% set
```

Name:

```
set - Set specific user property
```

Usage:

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

Arguments:

```
user
    name of the user, omit if current is set
```

Parameters: (type 1)

```
commonname ..... Full name (e.g. Donald Duck)
email ..... email
expirationdate ..... Date on which the user login will be disabled
groupid ..... Base group of this user
homedirectory ..... Home directory
id ..... User ID number
inactive ..... Number of days of inactivity allowed for the user
loginshell ..... Login shell
name ..... User login (e.g. donald)
password ..... Password
profile ..... Profile for Authorization
projectmanager ..... Project manager
revision ..... Entity revision
shadowmax ..... Maximum number of days for which the user password remains valid.
shadowmin ..... Minimum number of days required between password changes
shadowwarning ..... Number of days of advance warning given to the user before the
                    user password expires
surname ..... Surname (e.g. Duck)
writesshproxyconfig . Write ssh proxy config
```

Parameters: (type 2)

```
notes ..... Administrator notes
```

```
[bright90->user[maureen]]%
```

Continuing the session from the end of section 6.2.2, the password can be set at the user context prompt like this:

Example

```
[bright90->user[maureen]]% set password seteca5tr0n0my
[bright90->user*[maureen*]]% commit
[bright90->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

```
[bright90->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

```
[bright90->user[maureen]]% add tim; add fred
[bright90->user*[fred*]]% exit; group; add printer
[bright90->group*[printer*]]% commit
[bright90->group[printer]]% exit; exit; user
[bright90->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

```
[bright90->user*]% commit
Successfully committed 2 Users
[bright90->user]% group use printer
[bright90->group[printer]]% append members maureen tim fred; commit
[bright90->group[printer]]% show
Parameter          Value
-----
ID                  1002
Revision
Name                printer
Members             maureen,tim,fred
```

To remove users from a group, the `removefrom` command is used. A list of specific users, for example, `tim` and `fred`, can be removed from a group like this:

```
[bright90->group[printer]]% removefrom members tim fred; commit
[bright90->group[printer]]% show
Parameter          Value
-----
ID                  1002
Revision
Name                printer
Members             maureen
```

The `clear` command can also be used to clear members—but it also clears all of the extras from the group:

Example

```
[bright90->group[printer]]% clear members
[bright90->group*[printer*]]% show
Parameter          Value
```

```

-----
ID                1002
Revision
Name              printer
Members

```

The `commit` command is intentionally left out at this point in the session in order to illustrate how reversion is used in the next section.

6.2.4 Reverting To The Unmodified State

This corresponds roughly to the functionality of the Revert 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

```

[bright90->group*[printer*]]% refresh
[bright90->group[printer]]% show
Parameter          Value
-----
ID                1002
Revision
Name              printer
Members           maureen

```

Here, the user maureen reappears because she was stored in the last save. Also, because only the group object printer has been committed, the asterisk indicates the existence of other uncommitted, modified objects.

6.2.5 Removing A User

Removing a user using `cmsh` corresponds roughly to the functionality of the Delete 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

```

[bright90->group[printer]]% user use maureen
[bright90->user[maureen]]% remove -d; commit
Successfully removed 1 Users
Successfully committed 0 Users
[bright90->user]% !ls -d /home/*| grep maureen  #no maureen left behind
[bright90->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.

For RHEL7, the `/etc/nslcd.conf`, `/etc/openldap/ldap.conf`, and the certificate files under `/cm/local/apps/openldap/etc/certs/` should be copied over. Port 636 on ShoreWall running on the head node should be open for LDAP communication over the external network, if external nodes are using it on the external network. In addition, the external nodes and the head node must be able to resolve each other.

By default, 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. Health checks and health check objects are discussed in Chapter 12.

- 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@bright90 ~]# cmsh
[bright90]% user
[bright90->user]% add ldapcheck; commit
[bright90->user[ldapcheck]]% monitoring setup use ldap
[bright90->monitoring->setup[ldap]]% show
Parameter                               Value
-----
...
Arguments
...
[bright90->monitoring->setup[ldap]]% set arguments "ldapcheck"; commit
[bright90->monitoring->setup[ldap:ldapcheck]]%
```

Disabling the ldap health check: Instead of modifying the ldap health check to work when using an external LDAP server, it can be disabled entirely via Bright View or `cmsh`.

- Bright View: the ldap health check is disabled via the clickpath:

```
Monitoring->Data Producers->ldap->Edit
```

- `cmsh`: the disabled parameter of the ldap health check object is set to `yes`. The disabled parameter for the ldap health check can be set as follows:

```
[root@bright90 ~]# cmsh -c "monitoring setup use ldap; set disabled yes; commit"
```

Configuring The Cluster To Authenticate Against An External LDAP Server

The cluster can be configured in different ways to authenticate against an external LDAP server.

For smaller clusters, a configuration where LDAP clients on all nodes point directly to the external server is recommended. An easy way to set this up is as follows:

- On the head node:
 - In distributions that are derived from the RHEL 7.x series: the file `/etc/ldap.conf` does not exist. The files in which the changes then need to be made are `/etc/nslcd.conf` and `/etc/openldap/ldap.conf`. To implement the changes, the `nslcd` daemon must then be restarted, for example with `service nslcd restart`.
 - the `updateprovisioners` command (section 5.2.4) is run to update any other provisioners.
- Then, the configuration files are updated in the software images that the nodes use. If the nodes use the `default-image`, and if the nodes are based on RHEL7 and derivatives, then the files to update are `/cm/images/default-image/etc/nslcd.conf` and `/cm/images/default/etc/openldap/ldap.conf`. After the configuration change has been made, and the nodes have picked up the new configuration, the regular nodes can then carry out LDAP lookups.
 - Nodes can simply be rebooted to pick up the updated configuration, along with the new software image.
 - Alternatively, to avoid a reboot, the `imageupdate` command (section 5.6.2) can be run to pick up the new software image from a provisioner.
- The `CMDaemon` configuration file `cmd.conf` (Appendix C) has LDAP user management directives. These may need to be adjusted:
 - If another LDAP tool is to be used for external LDAP user management instead of Bright View or `cmsh`, then altering `cmd.conf` is not required, and Bright Cluster Manager's user management capabilities do nothing in any case.
 - If, however, system users and groups are to be managed via Bright View or `cmsh`, then `CM-Daemon`, 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
```

and:

```
syncprov-sessionlog 100
```

To allow the consumer to read the provider database, the consumer's access rights need to be configured. In particular, the `userPassword` attribute must be accessible. LDAP servers are often configured to prevent unauthorized users reading the `userPassword` attribute.

Read access to all attributes is available to users with replication privileges. So one way to allow the consumer to read the provider database is to bind it to replication requests.

Sometimes a user for replication requests already exists on the provider, or the root account is used for consumer access. If not, a user for replication access must be configured.

A replication user, `syncuser` with password `secret` can be added to the provider LDAP with adequate rights using the following `syncuser.ldif` file:

```
dn: cn=syncuser,<suffix>
objectClass: person
cn: syncuser
sn: syncuser
userPassword: secret
```

Here, `<suffix>` is the suffix set in `slapd.conf`, which is originally something like `dc=example,dc=com`. The `syncuser` is added using:

```
ldapadd -x -D "cn=root,<suffix>" -W -f syncuser.ldif
```

This prompts for the root password configured in `slapd.conf`.

To verify `syncuser` is in the LDAP database the output of `ldapsearch` can be checked:

```
ldapsearch -x "(sn=syncuser)"
```

To allow access to the `userPassword` attribute for `syncuser` the following lines in `slapd.conf` are changed, from:

```
access to attrs=userPassword
  by self write
  by anonymous auth
  by * none
```

to:

```

access to attrs=userPassword
  by self write
  by dn="cn=syncuser,<suffix>" read
  by anonymous auth
  by * none

```

Provider configuration is now complete. The server can be restarted using `service slapd restart` in RHEL 7.x.

External LDAP Server Replication: Configuring The Consumer(s)

The consumer is an LDAP server on a 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 `/etc/ldap.conf` file does not exist, then the note on page 232 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 `/etc/ldap.conf` files referred to here in the head and software images do not exist, then the note on page 232 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 232 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 Bright View or `cmsh`. Every cluster management operation requires that each user, including the administrator, has the relevant tokens in their *profile* for the operation.

The tokens for a user are grouped into a profile, and such a profile is typically given a name by the administrator according to the assigned capabilities. For example the profile might be called `readmonitoringonly` if it allows the user to read the monitoring data only, or it may be called `powerhandler` if the user is only allowed to carry out power operations. Each profile thus consists of a set of tokens, typically relevant to the name of the profile, and is typically assigned to several users.

The profile is stored as part of the authentication certificate (section 2.3) which is generated for running authentication operations to the cluster manager for the certificate owner.

Profiles are handled with the `profiles` mode of `cmsh`, or from the Profiles window, accessible via a clickpath of Identity Management→Profiles

The following preconfigured profiles are available from `cmsh`:

Profile name	Default Tasks Allowed
<code>admin</code>	all tasks
<code>cloudjob</code>	cloud job submission
<code>cmhealth</code>	health-related prejob tasks
<code>cmpam</code>	Bright Cluster Manager PAM tasks
<code>litenode</code>	CMDaemon Lite (section 2.6.7) tasks
<code>monitoringpush</code>	pushing raw monitoring data to CMDaemon via a JSON POST (page 519 of the <i>Developer Manual</i>)
<code>node</code>	node-related
<code>portal</code>	user portal viewing
<code>power</code>	device power
<code>readonly</code>	view-only

The available preconfigured profiles in `cmsh` can be seen by running the `list` command in profile mode. Another way to see them is by using tab-completion prompting, as follows:

Example

```
[root@bright90 ~]# cmsh
[bright90]% profile
[bright90->profile]% show <TAB> <TAB>
admin cloudjob cmhealth cmpam litenode monitoringpush node portal power readonly
```

The tokens, and other properties of a particular profile can be seen within profile mode as follows:

Example

```
[bright90->profile]% show readonly
Parameter      Value
-----
Name           readonly
Non user       no
Revision
Services       CMDevice CMNet CPart 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

```
[bright90->profile]% get readonly tokens
GET_DEVICE_TOKEN
GET_CATEGORY_TOKEN
GET_NODEGROUP_TOKEN
...
```

A profile can be set with `cmsh` for a user within user mode as follows:

Example

```
[root@bright90 ~]# cmsh
[bright90]% user use conner
[bright90->user[conner]]% get profile

[bright90->user[conner]]% set profile readonly; commit
```

6.4.1 Modifying Profiles

A profile can be modified by adding or removing appropriate tokens to it. For example, the `readonly` group by default has access to the burn status and burn log results. Removing the appropriate tokens stops users in that group from seeing these results.

In `cmsh` the removal can be done from within profile mode as follows:

```
[root@bright90 ~]# cmsh
[bright90]% profile use readonly
[...[readonly]]% removefrom tokens burn_status_token get_burn_log_token
[bright90%>profile*[readonly*]]% commit
```

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

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

Identity Management → Profiles → `readonly` → Edit → Tokens

In the resulting display it is convenient to maximize the window. Also convenient is running a search for `burn`, which will show the relevant tokens, `BURN_STATUS_TOKEN` and `GET_BURN_LOG_TOKEN`, as well as the subgroup they are in `device`. The ticks can be removed from the `BURN_STATUS_TOKEN` and `GET_BURN_LOG_TOKEN` checkboxes, and the changed settings can then be saved.

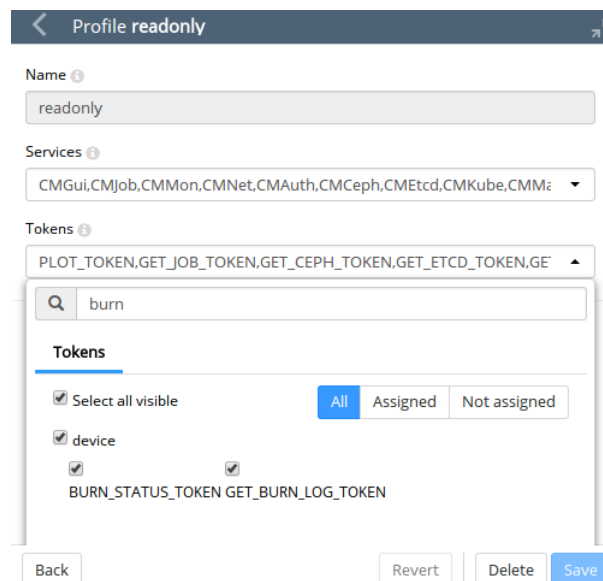


Figure 6.3: Bright View Profile Token Management

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 `cmsh` and Bright View. Cloning of profiles is also possible from `cmsh`.

A certificate file, with an associated expiry date, can be created based on a profile. The time of expiry for a certificate cannot be extended after creation. An entirely new certificate is required after expiry of the old one.

The creation of custom certificates using `cmsh` (page 239) or Bright View (page 240) is described later on. After creating such a certificate, the `openssl` utility can be used to examine its structure and properties. In the following example most of the output has been elided in order to highlight the expiry date (30 days from the time of generation), the common name (`democert`), the key size (2048), profile properties (`readonly`), and system login name (`peter`), for such a certificate:

```
[root@bright90]# openssl x509 -in peterfile.pem -text -noout
Data:
  ...
    Not After : Sep 21 13:18:27 2014 GMT
  Subject: ... CN=democert
    Public-Key: (2048 bit)
  ...
X509v3 extensions:
  1.3.6.1.4.4324.1:
    ..readonly
  1.3.6.1.4.4324.2:
    ..peter
[root@bright90]#
```

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@bright90 ~]# cmsh -c "cert; listcertificates"
Serial num Days left Profile Country Name Revoked
-----
1 36451 admin US Administrator Yes
10 9 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 Bright View: The Bright View equivalent for listing certificates is via the click-path `Identity Management`→`Certificates` (figure 6.4):

In the certificates list, node certificates that are generated by the node-installer (section 5.4.1) for each node for `CMDaemon` use are listed.

Custom certificates are also listed in the certificate lists.

Certificates						
NAME	REVOKED	SERIAL ...	REMAI...	PROFILE	COUNTRY	OPTIONS
<input type="checkbox"/> Administrator	false	2	36478 da...	admin	US	Edit ▾
<input type="checkbox"/> CMHealth	false	3	36478 da...	cmhealth	US	Edit ▾
<input type="checkbox"/> CMHealth	false	4	36478 da...	cmhealth	US	Edit ▾
<input type="checkbox"/> CMPam	false	5	36478 da...	cmpam	US	Edit ▾
<input type="checkbox"/> PJ-tr	false	1	36478 da...		US	Edit ▾
<input type="checkbox"/> WebPortal	false	6	36478 da...	portal	US	Edit ▾
<input type="checkbox"/> bright80	false	17	36488 da...		US	Edit ▾
<input type="checkbox"/> fa-16-3e-09-2c-79	false	10	36478 da...	node	US	Edit ▾

Figure 6.4: Bright View Certificates List Window

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:

- cmsht: with the createcertificate operation from within cert mode
- Bright View: via the clickpath Identity Management → Users → Edit → Profile to set the Profile.

Creating a new certificate for cmsht users: Creating a new certificate in cmsht is done from cert mode using the createcertificate command, which has the following help text:

```
[bright90->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 ef readonly peter 30 /home/peter\
/peterfile.key /home/peter/peterfile.pem
```

```
Fri Aug 22 06:18:27 2014 [notice] bright90: New certificate request with ID: 1
```

```
[bright90->cert]% createcertificate 2048 democert a b c d ef readonly pe\
```

```
ter 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 running `cmsh`. This means that user `peter` cannot use them until their ownership is changed to user `peter`:

Example

```
[root@bright90 ~]# cd /home/peter
[root@bright90 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@bright90 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@bright90 ~] cmsh -i peterfile.pem -k peterfile.key
[bright90]% quit
```

2. If the `-i` and `-k` options are not used, then `cmsh` searches for default keys. The default keys for `cmsh` are under these paths under `$HOME`, in the following order of priority:

- (a) `.cm/admin.{pem,key}`
- (b) `.cm/cert.{pem,key}`

Creating a custom certificate for Bright View users: As in the case of `cmsh`, a Bright View user having a sufficiently privileged tokens profile, such as the `admin` profile, can create a certificate and key file for themselves or another user. This is done by associating a value for the `Profile` from the `Add or Edit` dialog for the user (figure 6.2).

The certificate files, `cert.pem` and `cert.key`, are then automatically placed in the following paths and names, under `$HOME` for the user:

- `.cm/admin.{pem,key}`
- `.cm/cert.{pem,key}`

Users that authenticate with their user name and password when running Bright View use this certificate for their Bright View clients, and are then restricted to the set of tasks allowed by their associated profile.

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

The use of an external LDAP server instead of 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. This can be carried out with the `external-user-cert` script, which is provided with the `cluster-tools` package. The package is installed by default with Bright Cluster Manager.

Running the `external-user-cert` script embeds the user and profile in the certificate during certificate generation. The script has the following usage:

```
external-user-cert -h
Usage: for a single profile: external-user-cert <profile> <user> [<user> ... ]
        --home=<home-prefix> [-g <group>] [-o]
        for several profiles: external-user-cert --home=<home-prefix>
        --file=<inputfile> [-g <group>]
        where lines of <inputfile> have the syntax
        <profile> <user> [<user> ... ]
```

Options:

```
-h, --help          show this help message and exit
--file=FILE         input FILE
--home=HOME_PATH   path for home directories, default /home/
-g GROUP           name of primary group, e.g. wheel
-o                 overwrite existing certificates
```

Here,

- <profile> should be a valid profile
- <user> should be an existing user
- <home-prefix> is usually /home
- <group> is a group, such as wheel
- <inputfile> is a file with each line having the syntax

```
<profile> <user> [<user> ... ]
```

One or more external LDAP user certificates can be created by the script. The certificate files generated are `cert.pem` and `cert.key`. They are stored in the home directory of the user.

For example, a user `spongebob` that is managed on the external server, can have a read-only certificate generated with:

```
# external-user-cert readonly spongebob --home=/home
```

If the home directory of `spongebob` is `/home/spongebob`, then the key files that are generated are `/home/spongebob/.cm/cert.key` and `/home/spongebob/.cm/cert.pem`.

Assuming no other keys are used by `cmsh`, a `cmsh` session that runs as user `spongebob` with `readonly` privileges can now be launched:

```
$ module load cmsh
$ cmsh
```

If other keys do exist, then they may be used according to the logic explained in item 2 on page 240.

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.

7

Workload Management

For clusters that have many users and a significant load, a workload manager (WLM) system allows a more efficient use of resources to be enforced for all users than if there were no such system in place. This is because without resource management, there is a tendency for each individual user to over-exploit common resources.

When a WLM is used, the end user can submit a job to it. This can be done interactively, but it is typically done as a non-interactive batch job.

The WLM assigns resources to the job, and checks the current availability as well as checking its estimates of the future availability of the cluster resources that the job is asking for. The WLM then schedules and executes the job based on the assignment criteria that the administrator has set for the WLM system. After the job has finished executing, the job output is delivered back to the user.

Among the hardware resources that can be used for a job are GPUs. Installing CUDA software to enable the use of GPUs is described in section 7.4 of the *Installation Manual*. Configuring GPU settings for Bright Cluster Manager is described in section 3.13.2 of the *Administration Manual*. Configuring GPU settings for an individual WLM is described in the section on getting that particular WLM up and running.

The details of job submission from a user's perspective are covered in the *User Manual*.

Sections 7.1–7.5 cover the installation procedure to get a WLM up and running.

Sections 7.6–7.7 describe how Bright View and `cmsh` are used to view and handle jobs, queues and node drainage.

Section 7.8 shows examples of WLM assignments handled by Bright Cluster Manager.

Section 7.9 describes the power saving features of WLMs.

Section 7.10 describes cgroups, a resources limiter, mostly in the context of WLMs.

Section 7.11 describes WLM customizations for settings other than the common settings covered by Bright Cluster Manager.

7.1 Workload Managers Choices

Some WLM packages are installed by default, others require registration from the distributor before installation.

During cluster installation, a WLM can be chosen (figure 3.10 of the *Installation Manual*) for setting up. The choices are:

- **PBS**: An HPC job scheduler, originally developed at NASA, now developed by Altair. This comes in 3 variants:
 1. **PBS Pro v19**: A commercial variant, with commercial support from Altair.
 2. **PBS Pro CE**, renamed to **OpenPBS** (for version 20 and onward): A free (AGPL) variant. This is community-supported, and the default workload manager provided with Bright Cluster Manager.

3. **OpenPBS:** renamed from PBS Pro CE for version 20 and higher.
- **Slurm v19 and v20:** A free (GPL) job scheduler, with commercial support.
 - **Univa Grid Engine (UGE), v8.6.0:** This is a further development of the older Sun Grid Engine. UGE is proprietary commercial software, developed by Univa. Bright Cluster Manager 9.0 supports integration with UGE version 8.6.
 - **None:** For clusters that need no HPC job-scheduling.

The preceding WLMs can also be chosen and set up later using the `cm-wlm-setup` tool (section 7.3). After installation, if there are no major changes in the WLM for updated versions of the workload managers, then

- WLMs that are packaged with Bright Cluster Manager (Slurm, PBS) can have their packages updated using standard package update commands (`yum update` and similar). The installation and configuration of the WLM from the updated packages is carried out as described later on in this chapter.
- WLMs that are installed by picking up software from the vendor (UGE, LSF) can be updated by following vendor guidelines. For UGE the installation is carried out as outlined in section 7.5.2.

LSF installation: Besides the preceding WLMs, IBM Spectrum LSF (Load Sharing Facility v10) can also be installed. This is expected to be installed via `cm-wlm-setup` in a future version. For now, Bright Computing support can be contacted for Bright-related installation issues.

7.2 Forcing Jobs To Run In A Workload Management System

Another preliminary step is to consider forcing users to run jobs only within the WLM system. Having jobs run via a WLM is normally a best practice.

For convenience, the 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 without restriction, that is, outside the WLM system. For clusters with a significant load this policy results in a sub-optimal use of resources, since such unplanned-for jobs disturb any already-running jobs.

Disallowing user logins to nodes, so that users have to run their jobs through the WLM system, means that jobs are then distributed to the nodes only according to the planning of the WLM. If planning is based on sensible assignment criteria, then resources use is optimized—which is the entire aim of a WLM in the first place.

7.2.1 Disallowing User Logins To Regular Nodes Via `cmsh`

The `usernodelogin` setting of `cmsh` restricts direct user logins from outside the WLM, and is thus one way of preventing the user from using node resources in an unaccountable manner. The `usernodelogin` setting is applicable to node categories only, rather than to individual nodes.

In `cmsh` the attribute of `usernodelogin` is set from within `category` mode:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category use default
[bright90->category[default]]% set usernodelogin onlywhenjob
[bright90->category*[default*]]% commit
```

The attributes for `usernodelogin` are:

- **always** (the default): This allows all users to `ssh` directly into a node at any time.

- `never`: This allows no user other than root to directly ssh into the node.
- `onlywhenjob`: This allows the user to ssh directly into the node when a job is running on it. It typically also prevents other users from doing a direct ssh into the same node during the job run, since typically the WLM is set up so that only one job runs per node. However, an ssh session that is already running is not automatically terminated after the job is done.
 - Some cluster administrators may wish to allow some special user accounts to override the `onlywhenjob` setting, for example for diagnostic purposes. Before giving the details of how to override the setting, some background explanation is probably useful:
The `onlywhenjob` setting works with the PAM system, and adds the following line to `/etc/pam.d/ssh` on the regular nodes:

```
account      required      pam_bright.so
```

Nodes with the `onlywhenjob` restriction can be configured to allow a particular set of users to access them, despite the restriction, by whitelisting them in the PAM system, as follows: Within the software image `<node image>` used by the node, that is under `/cm/images/<node image>`, the administrator can add the set of user accounts to the file `etc/security/pam_bright.d/pam_whitelist.conf`. This file is installed in the software image with a chroot installation (section 11.4) of the `cm-libpam` package.

Other adjustments to PAM configuration, such as the number of attempted logins and the associated wait time per login, can be carried by editing the `/etc/security/pam_bright.d/cm-check-alloc.conf` file.

The image can then be updated to implement the whitelist, by running the `imageupdate` command in `cmsh` (section 5.6.2), or by clicking the `Update node` option in Bright View (section 5.6.3).

7.2.2 Disallowing User Logins To Regular Nodes Via Bright View

In Bright View, user node login access is set via a category setting, for example for the `default` category via the clickpath in figure 7.1:

Grouping→Node categories[default]→Edit→Settings→User node login

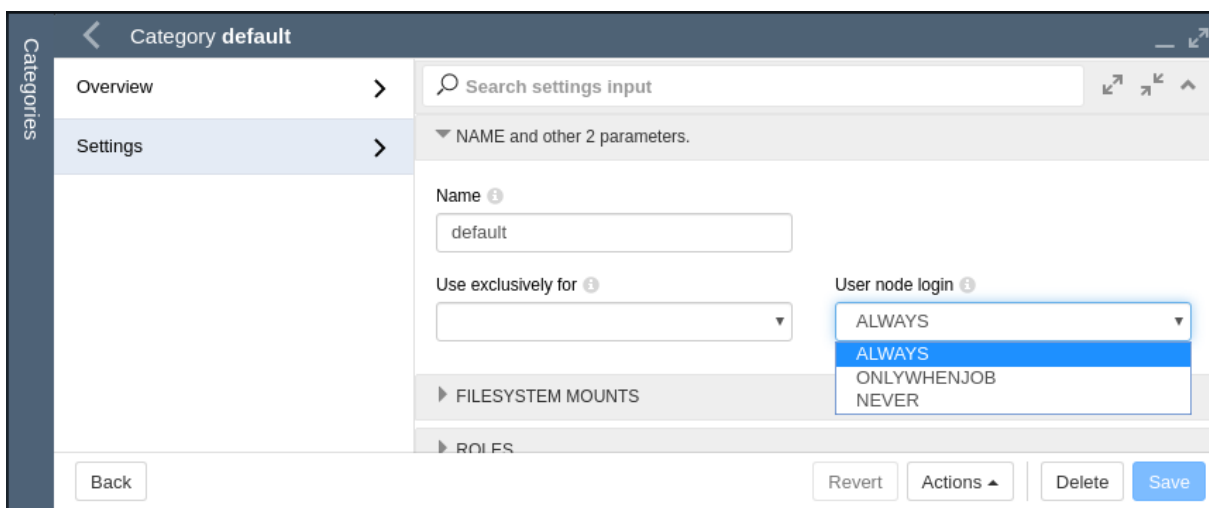


Figure 7.1: Disallowing User Logins To Nodes Via Bright View

7.2.3 Disallowing Other User Processes Outside Of Workload Manager User Processes

Besides disabling user logins, administrators may choose to disable interactive jobs in the WLM as an additional measure to prevent users from starting jobs on other nodes.

Administrators may also choose to set up scripts that run after job execution. Such scripts can terminate user processes outside the WLM, as part of a policy, or for general administrative hygiene. These are `Epilog` scripts and are part of the WLM.

The WLM documentation has more on configuring these options.

7.2.4 High Availability By Workload Managers

Bright Cluster Manager uses the existing built-in high availability (HA) functionalities of workload managers as much as possible. A double server HA configuration with the existing built-in HA functionality can be carried out using the `cm-wlm-setup` utility (section 7.3), or using the Bright View HA wizard (section 7.4.1). The built-in functionality makes use of the primary WLM server and secondary WLM server, which are placed on separate nodes.

The HA configuration aspect (section 14.1.3) in this case means that a WLM server role is assigned to both the WLM primary and the WLM secondary nodes. These primary WLM and secondary WLM servers can then:

- both be on head nodes,
- both be on regular (compute) nodes

It is not possible to configure a mixed setup—that is with one head node and one regular node—for the WLM servers on an HA setup.

If the head nodes are configured with the WLM server roles, then in the cluster entity configuration the `primary_server` value is empty. This is because Bright Cluster Manager always configures the primary WLM server on the primary cluster head node.

If two regular nodes are configured with the WLM server roles instead, then in the cluster entity configuration the `primary_server` parameter is set to one of the compute nodes.

The cluster primary head node and the WLM primary server should not be confused. In particular, the active cluster head node and active workload manager server are not necessarily the same for the case of the Slurm, PBS Pro, or LSF workload managers. For these, if the passive cluster head node becomes an active cluster head node without a crash, then this does not trigger a passive WLM server on the newly active head node to also become an active WLM server. Thus, the WLM server can be active on a passive head node, and vice versa, because the WLM primary server is independent of the cluster primary head node.

7.3 Installation Of Workload Managers

Normally the administrator selects a WLM to be used during Bright Cluster Manager installation (figure 3.10 of the *Installation Manual*). A WLM may however also be added and configured after Bright Cluster Manager has been installed, using `cm-wlm-setup`.

With most other objects, the Bright Cluster Manager front ends—`cmsh` and Bright View—can be used to create a new object from scratch, or can clone a new object from another existing object. However, the front ends cannot do this for a WLM object. An attempt to create or clone a new WLM object via `cmsh` or Bright View is prohibited by the front ends, because there are many pitfalls possible in configuration.

A new WLM object, and WLM instance, can therefore only be installed via `cm-wlm-setup`, or by selecting a WLM during the initial Bright Cluster Manager installation.

7.3.1 Running `cm-wlm-setup` In CLI Mode

The recommended way to run the `cm-wlm-setup` utility is without options or arguments, in which case an Ncurses dialog starts up. An Ncurses run with `cm-wlm-setup` is covered in section 7.3.2.

However, the `cm-wlm-setup` utility can alternatively be used in a non-Ncurses, command-line, mode, with options and arguments. The utility has the following usage:

```
[root@bright90 ~]# cm-wlm-setup -h
usage: Workload manager setup cm-wlm-setup [-v] [-h] [-c <config_file>]
                                           [--skip-modules <mod1,mod2,...>]
                                           [--only-these-modules <mod1,mod2,...>]
                                           [--dev]
                                           [--on-error-action {debug,remotedebug,undo,abort}]
                                           [--output-remote-execution-runner]
                                           [--json] [--no-distro-checks]
                                           [--min-reboot-timeout <reboot_timeout_seconds>]
                                           [--setup | --disable]
                                           [--wlm <name>]
                                           [--server-nodes SERVER_NODES]
                                           [--server-primary SERVER_PRIMARY]
                                           [--server-overlay-name SERVER_OVERLAY_NAME]
                                           [--server-overlay-priority SERVER_OVERLAY_PRIORITY]
                                           [--client-categories CLIENT_CATEGORIES]
                                           [--client-nodes CLIENT_NODES]
                                           [--client-overlay-name CLIENT_OVERLAY_NAME]
                                           [--client-overlay-priority CLIENT_OVERLAY_PRIORITY]
                                           [--client-slots <slots>]
                                           [--submit-categories SUBMIT_CATEGORIES]
                                           [--submit-nodes SUBMIT_NODES]
                                           [--submit-overlay-name SUBMIT_OVERLAY_NAME]
                                           [--submit-overlay-priority SUBMIT_OVERLAY_PRIORITY]
                                           [--wlm-cluster-name WLM_CLUSTER_NAME]
                                           [--reboot] [--reset-cgroups]
                                           [--yes-i-really-mean-it]
                                           [--archives-location <path>]
                                           [--license <license>] [--purge]
                                           [--accounting-overlay-name ACCOUNTING_OVERLAY_NAME]
                                           [--accounting-overlay-priority ACCOUNTING_OVERLAY_PRIORITY]
```

The help output from running `cm-wlm-setup -h` continues on beyond the preceding text output, and gives details on these options.

The usage options can be grouped as follows:

Common Options

- `-v`, `--verbose`: This displays a more verbose output. It can be helpful in troubleshooting.
- `-h`, `--help`: Displays the help screen.
- `-c <config_file>`: Loads a runtime configuration for plugins for the utility, using a specified YAML config file `<config_file>`

Setting Up Or Disabling The WLM

- `--setup`: Sets up the server, enables roles, creates the default queues/partitions, for a WLM specified with the `--wlm` option
- `--disable`: Disables WLM services for a WLM specified with the `--wlm` option

Specifying WLM Properties And Behavior

- `--wlm <WLM name>`: Specifies which WLM is to be set up. Choices for `<WLM name>` are:
 - `pbspro-ce`

- pbspro
 - openpbs
 - slurm
 - uge
 - lsf
- `--wlm-cluster-name <WLM cluster name>`: Specifies the name for the new WLM cluster that is to be set up.
 - `--reboot`: Reboot after install

Server Role Settings

- `--server-nodes <SERVER_NODES>`: The server roles of the WLM take the value set for `<SERVER_NODES>`. The default value is HEAD, which is a reserved name for the head node.
- `--server-primary <SERVER_PRIMARY>`: The hostname used for the primary server is set to `<SERVER_PRIMARY>`. It takes a default value of HEAD.
- `--server-overlay-name <SERVER_OVERLAY_NAME>`: Server role configuration overlay name. The default name for this is `<wlm_name>-server`, where `<wlm_name>` is the name specified in the `--wlm` option.
- `--server-overlay-priority <SERVER_OVERLAY_PRIORITY>`: Server role configuration overlay priority. This takes a default value of 500.

Client Role Settings

- `--client-categories <CLIENT_CATEGORIES>`: The client categories of the WLM take the value set for `<CLIENT_CATEGORIES>`, which is a comma-separated list of nodes categories. Its default value is default.
- `--client-nodes <CLIENT_NODES>`: The client roles of the WLM take the value set for `<CLIENT_NODES>`, which is a comma-separated list of nodes. It has no value set by default.
- `--client-overlay-name <CLIENT_OVERLAY_NAME>`: Client role configuration overlay name. The default name for this is `<wlm_name>-client`, where `<wlm_name>` is the name specified in the `--wlm` option.
- `--client-overlay-priority <CLIENT_OVERLAY_PRIORITY>`: Client role configuration overlay priority. This takes a default value of 500.
- `--client-slots <slots>`: Number of slots for a client.

Submit Role Settings

- `--submit-categories <SUBMIT_CATEGORIES>`: The submit roles of the WLM take the value set for `<SUBMIT_CATEGORIES>`. `<SUBMIT_CATEGORIES>` is a comma-separated list of node categories that are submit nodes. Its default value is default.
- `--submit-nodes <SUBMIT_NODES>`: The submit roles of the WLM take the value set for `<SUBMIT_NODES>`, which is a comma-separated list of submit nodes. It has no value set by default.
- `--submit-overlay-name <SUBMIT_OVERLAY_NAME>`: Submit role configuration overlay name. The default name for this is `<wlm_name>submit`, where `<wlm_name>` is the name specified in the `--wlm` option.
- `--submit-overlay-priority <SUBMIT_OVERLAY_PRIORITY>`: Submit role configuration overlay priority. This takes a default value of 500.

Disable Options

- `--reset-cgroups`: Disable joining cgroup controllers with systemd setting JoinControllers
- `--yes-i-really-mean-it`: Required for additional safety

Workload Manager Specific Options**For UGE:**

- `--purge`: used together with the `--yes-i-really-mean-it` option to wipe a UGE instance. This wipes the cell directory in UGE, which is like a spool directory, and contains its configuration.

For PBSPro:

- `--license`: used to specify path to PBSPro license.

For Slurm:

- Accounting node options: For Bright Cluster Manager, the Slurm accounting roles are assigned to the head node for on-premise nodes, and are assigned to the Edge directors for off-premise nodes. The roles can be reassigned to other nodes later on, although that typically reduces performance.
 - `--accounting-overlay-name <ACCOUNTING_OVERLAY_NAME>`: Accounting role configuration overlay name. The default name for this is `slurm-accounting`.
 - `--accounting-overlay-priority <ACCOUNTING_OVERLAY_PRIORITY>`: Accounting role configuration overlay priority. This takes a default value of 500.

For example, if UGE packages are downloaded to `/root/uged1`, then a UGE setup could be carried out with (much output elipsized):

Example

```
[bright90 ~]# cm-wlm-setup -v --setup --wlm uge --archives-location /root/uged1 --client-categories default
I 19-12-05 15:29:46 |
D 19-12-05 15:29:46 | Please wait...
D 19-12-05 15:29:46 | #####
D 19-12-05 15:29:46 | ##### START #####
D 19-12-05 15:29:46 | #####
D 19-12-05 15:29:46 |
D 19-12-05 15:29:46 | ### Logging Current Runtime environment
...
...
I 19-12-05 15:30:20 | Took:      00:33 min.
I 19-12-05 15:30:20 | Progress: 100/100
I 19-12-05 15:30:20 | ##### Finished execution for 'Workload manager setup', status: completed
I 19-12-05 15:30:20 |
I 19-12-05 15:30:20 | Workload manager setup finished!
I 19-12-05 15:30:20 |
D 19-12-05 15:30:20 |
D 19-12-05 15:30:20 | ##### END #####
D 19-12-05 15:30:20 |
I 19-12-05 15:30:20 |
```

A check to see if the UGE instance (WLM cluster) has been created could be:

Example

```
[bright90 ~]# cmsh -c "wlm; list"
Type   Name (key)           Server nodes Submit nodes           Client nodes
-----
UGE    uge                   bright90    bright90,node001,node002  node001,node002
```

The UGE instance can then be removed with:

Example

```
[bright90 ~]#
cm-wlm-setup -v --disable --wlm uge --purge --yes-i-really-mean-it --wlm-cluster-name=uge
```

A log file of the setup can be seen at `/var/log/cm-wlm-setup.log`.

7.3.2 Running `cm-wlm-setup` In Ncurses Mode

Running `cm-wlm-setup` with no options and with no arguments brings up an Ncurses screen (figure 7.2).

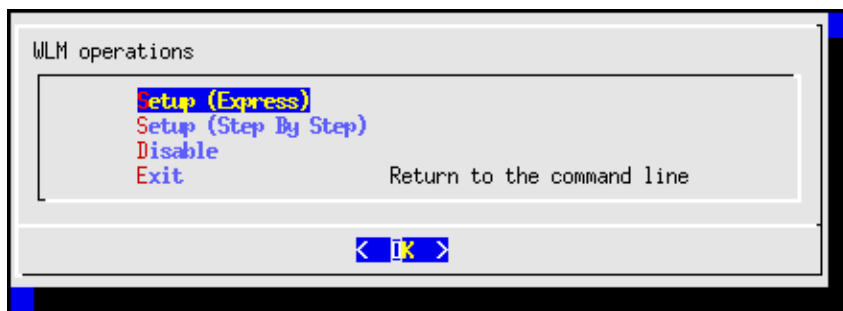


Figure 7.2: `cm-wlm-setup` Ncurses initial screen

Express Installation

The Setup (Express) menu option allows the administrator to select the workload manager in the next screen (figure 7.3), and to install it with a minimal number of configuration steps. If it has already been installed, but disabled via `cm-wlm-setup`, then it can also be re-enabled, instead of installed from scratch.



Figure 7.3: `cm-wlm-setup` Ncurses WLM selection screen

Step By Step Installation

If the Setup (Step By Step) menu option is chosen instead of the express option, then this also allows the administrator to select the workload manager in figure 7.3. But after selection, there are a number of extra configuration steps that can be carried out which are not available in the express configuration. Guidance is given for these extra steps, and sensible default values are already filled in for many options.

Disabling An Installation

The Disable option in figure 7.2 allows the administrator to disable an existing instance.

Summary Screen

The screen that appears after the configuration steps are completed, is the Summary screen (figure 7.4). This screen allows the configuration to be viewed, saved, or saved and deployed.

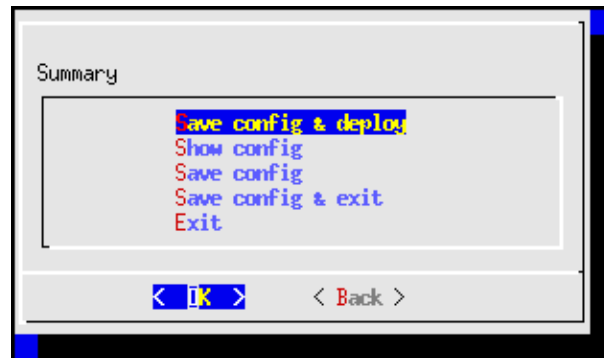


Figure 7.4: cm-wlm-setup Ncurses summary screen

If deployment is carried out, then several screens of output are displayed. After the deployment is completed, the log file can be viewed at `/var/log/cm-wlm-setup.log`.

7.3.3 Prolog And Epilog Scripts

What Prolog And Epilog Scripts Do

The workload manager runs prolog scripts before job execution, and epilog scripts after job execution. The purpose of these scripts can include:

- checking if a node is ready before submitting a job execution that may use it
- preparing a node in some way to handle the job execution
- cleaning up resources after job execution has ended.

The administrator can run custom prolog or epilog scripts for the queues from CMDaemon for UGE, or LSF, by setting such scripts in the Bright View or `cmsh` front ends.

Example

```
[bright90->wlm[uge]->jobqueue]% use all.q
[bright90->wlm[uge]->jobqueue[all.q]]% show | grep . | grep -i epilog
Epilog                               NONE
```

For PBS and Slurm, there are global prolog and epilog scripts, but editing them is not recommended. Indeed, in order to discourage editing them, the scripts cannot be set via the cluster manager front ends. Instead the scripts must be placed by the administrator in the software image, and the relevant nodes updated from the image.

Detailed Workings Of Prolog And Epilog Scripts

Even though it is not recommended, some administrators may nonetheless wish to link and edit the scripts directly for their own needs, outside of the Bright View or `cmsh` front ends. A more detailed explanation of how the prolog scripts work therefore follows:

The prolog scripts have names and function according to their locations. The scripts are placed in two directories:

1. In the **main scripts directory**, at:

```
/cm/local/apps/cmd/scripts/
```

In this directory, a main prolog script, `prolog`, can call a sequence of rc.d-style prolog scripts for a particular workload manager in an 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
 - `-cmjob`: script runs prior to job run in a cloud
- **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-cmjob`

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 PBS workload manager uses the symlink `10-prolog-prejob` within the rc.d-style directory `/cm/local/apps/pbspro/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 scripts, which run after a job run, can be used for UGE and Slurm.

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 at `/cm/local/apps/cmd/scripts/prolog-prejob`, is assigned to `PrologSlurmctld` in `/cm/shared/apps/slurm/var/etc/<Slurm instance name>/slurm.conf`. It runs by default during job execution, and is executed with the `slurm` user permissions.

- `prolog`: in the main scripts directory at `/cm/local/apps/cmd/scripts/prolog`, is assigned to the variable `Prolog` in `/cm/shared/apps/slurm/var/etc/<Slurm instance name>/slurm.conf`. The script executes the `<number>-prolog{-cmjob|-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.
- **UGE, LSF**
 - `prolog`: in the main scripts directory, executes any `<number>-prolog{-cmjob|-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**
 - `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 `<number>-prolog{-cmjob|-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

Rebooting The Compute Nodes For A Newly Installed Workload Manager

After a workload manager package is installed and initialized with `cm-wlm-setup` (section 7.3), the compute nodes that are being managed by the workload manager must be rebooted before running workload manager jobs on them for the first time. A list of which nodes must be rebooted can be seen by checking their status.

Example

```
[bright90->device]% ds
node001 ..... [ UP ], restart required (cm-wlm-setup: compute role assigned)
node002 ..... [ UP ], restart required (cm-wlm-setup: compute role assigned)
node003 ..... [ UP ], restart required (cm-wlm-setup: compute role assigned)
[bright90 ..... [ UP ]
```

Enabling And Disabling A WLM

A WLM can be disabled for all nodes with `cm-wlm-setup`. Disabling the WLM means the workload management services are stopped by removing roles, and removing the WLM cluster object.

Alternatively, a WLM can be enabled or disabled by the administrator via role addition and role removal with Bright View or `cmsh`. This is described further on in this section.

Multiple WLM instances of the same type: Versions of Bright Cluster Manager prior to 9.0 already had the ability to have different workload managers run at the same time. However, Bright Cluster Manager version 9.0 introduced the additional ability to run many workload managers of the same kind at the same time.

Example

Two WLM instances, Slurm and OpenPBS are already running at the same time in the cluster, with each WLM assigned to one category. Then, Bright Cluster Manager can start up a third WLM instance, such as another Slurm WLM instance. These WLM instances are alternatively called WLM clusters,

because they effectively allow one cluster to function as many separate clusters as far as running WLMs is concerned.

From the Bright View or `cmsh` point of view a WLM consists of

- a WLM server, usually on the head node
- WLM clients, usually on the compute nodes

For the administrator, enabling or disabling the servers or clients is then simply a matter of assigning or unassigning a particular WLM server or client role on the head or compute nodes, as deemed appropriate.

The administrator typically also sets up an appropriate WLM environment module (`slurm`, `uge`, `pbspro`, `lsf`), so that it is loaded up for the end user (section 2.2.3).

7.4.1 Enabling And Disabling A WLM With Bright View

A particular WLM package may be installed, but the WLM may not be enabled. This can happen, for example, if disabling a WLM that was previously enabled.

If a WLM instance exists, then the WLM client, submission, and server roles can be enabled or disabled from Bright View by assigning or removing the appropriate roles to nodes, categories, or configuration overlays. Within the role, the properties of the WLM may be further configured by setting options.

Workload Manager Role Assignment To An Individual Node With Bright View

Workload Manager Server The following roles are WLM roles that can be assigned to a node:

- server
- submit
- accounting (for the Slurm WLM only, to configure and run the `slurmdbd` service)
- client

For example, a Slurm server role can be assigned to a head node, `bright90`, via the clickpath:

Devices→Head Nodes[`bright90`]→Edit→Settings→Roles→Slurm server role→Add

The large number of roles that exist in general means that it is usually convenient to use a filter to narrow down the search. An excerpt of the text from role can be used for the filter. In the example in figure 7.5 the text in the filter is `slu`, which results in a display of the available Slurm roles.

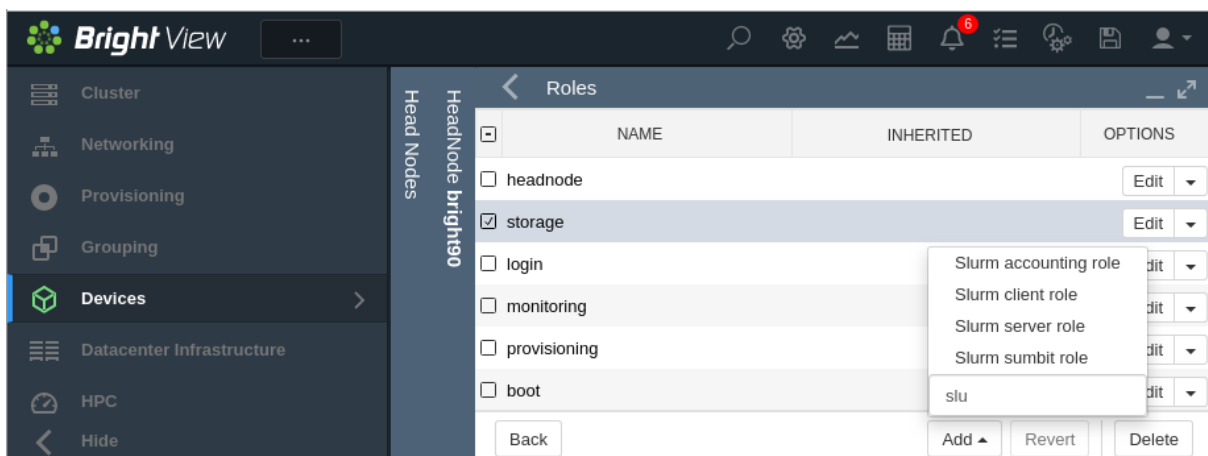


Figure 7.5: Workload Management Role Assignment On A Head Node

Options can be set within the workload manager, within the role. For example, for Slurm, a `builtin` or `backfill` option can be set for the Scheduler parameter. The workload manager server role is then saved with the selected options (figure 7.6). 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.

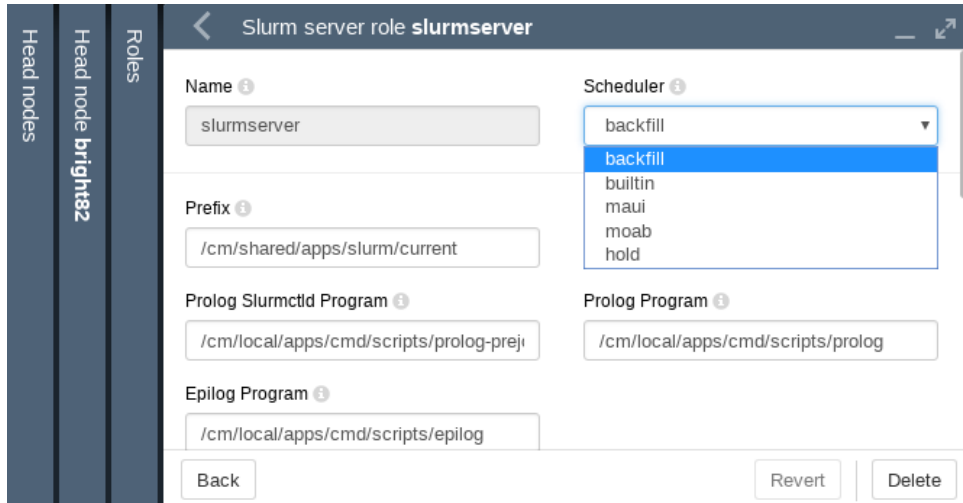


Figure 7.6: Workload Management Role Assignment Options On A Head Node

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 to it. Some basic options can be set for the client role right away.

Saving the role, and then running `imageupdate` (section 5.6.2), automatically starts up the client process with the options chosen, and managed by `CMDaemon`.

Workload Manager Role Assignment To A Category With Bright View

It is true that workload manager role assignment can be done as described in the preceding text for individual non-head nodes. However it is usually more efficient to assign roles using categories or configuration overlays, due to the large number of compute nodes in typical clusters.

Thus, as an example, the case can be considered of all physical on-premise non-head nodes. By default these are placed in the `default` category. This means that, by default, roles in the category are automatically assigned to all those non-head nodes, unless, as an exception, an individual node configuration overrides the category setting and uses a role setting instead at node level.

Viewing the possible workload manager roles for the category `default` is done by using the click-path:

Grouping → Node categories[default] → Edit → Settings → Roles → Add

Using the filter to narrow down the search is usually convenient. Once the role is selected its options can be edited and saved.

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 may also be specified, depending on the workload manager used.

The workload manager server role can also be assigned to a non-head node. For example, a Slurm server role can be taken on by a non-head node. This is the equivalent to the `--server-nodes` option of `cm-wlm-setup`.

Saving the roles with their options and then running `imageupdate` (section 5.6.2) automatically starts up the newly-configured workload manager.

Workload Manager Role Options With Bright View

Each compute node role (workload manager client role) has options that can be set for GPUs, Queues, and Slots.

- Slots, in a workload manager, corresponds in Bright Cluster Manager to:
 - the CPUs setting (a `NodeName` parameter) in Slurm's `slurm.conf`
 - the `np` setting in PBS,

and is normally set to the number of cores per node.

In LSF setting the number of slots for the client role to 0 means that the client node does not run jobs on itself, but becomes a submit host, which means it is able to forward jobs to other client nodes.

- Queues with a specified name are available in their associated role after they are created. The creation of queues is described in sections 7.6.2 (using Bright View) and 7.7.2 (using `cmsh`).

All server roles also provide the option to enable or disable the `External Server` setting. Enabling that means that the server is no longer managed by Bright Cluster Manager, but provided by an external device.

The `cmsh` equivalent of enabling an external server is described on page 260.

7.4.2 Enabling And Disabling A Workload Manager With `cmsh`

A particular workload manager package may be set up, but not enabled. This can happen, for example, if no WLM server or WLM client role has been assigned.

If a WLM instance exists, then the WLM client, server, or submit roles can be enabled from `cmsh` by assigning it from within the `roles` submodule. Within the assigned role, the properties of the WLM may be further configured by setting options.

Workload Manager Role Assignment To A Configuration Overlay With `cmsh`

In `cmsh`, workload manager role assignment to a configuration overlay (section 2.1.5) can be done using `configurationoverlay` mode. By default `cm-wlm-setup` run as an Ncurses session creates some configuration overlays with suggestive names, and assigns roles to the configuration overlays according to what the names suggest. Thus, for example, with the `cm-wlm-setup` Ncurses session used to carry out an express setup for Slurm, the configuration overlays that get created are the following:

Example

```
[bright90->configurationoverlay]% list
Name (key)      Priority  Nodes      Categories      Roles
-----
slurm-accounting 500
slurm-client    500              default          slurmclient
slurm-server    500
slurm-submit    500      bright90 default          slurmsubmit
```

Nodes in the default category can take on the `slurmclient` or `slurmsubmit` role by setting the nodes for the role using the configuration overlay `slurm-client`.

Example

```
[bright90->configurationoverlay]% use slurm-client
[bright90->configurationoverlay[slurm-client]]% show
Parameter      Value
-----
```

```

All head nodes          no
Categories              default
Customizations         <0 in submode>
Name                   slurm-client
Nodes
Priority                500
Revision
Roles                   slurmclient
[bright90->configurationoverlay[slurm-client]]% set nodes
node001 node002 node003 bright90
[bright90->configurationoverlay[slurm-client]]% set nodes node001..node002
[bright90->configurationoverlay*[slurm-client*]]% commit
[bright90->configurationoverlay[slurm-client]]% list
Name (key)      Priority  Nodes          Categories    Roles
-----
slurm-accounting 500
slurm-client    500      node001,node002 default      slurmclient
slurm-server    500
slurm-submit    500      bright90       default      slurmsubmit
[bright90->configurationoverlay[slurm-client]]%

```

Values for the parameters in a role, such as the `slurmclient` role, can be set within the configuration overlay:

Example

```

[bright90->configurationoverlay[slurm-client]]% roles
[bright90->configurationoverlay[slurm-client]->roles]% use slurmclient
[bright90->configurationoverlay[slurm-client]->roles[slurmclient]]% show
Parameter          Value
-----
Add services       yes
Name               slurmclient
Provisioning associations <0 internally used>
Revision
Type               SlurmClientRole
WLM cluster        slurm
Slots              0
Queues             defq
All Queues         no
Weight             0
Boards             0
SocketsPerBoard    0
Power Saving Allowed no
Reason
NodeAddr
Cores Per Socket   0
Features
Port               0
RealMemory         0
Sockets            0
ThreadsPerCore     0
TmpDisk            0
CPU Spec List
Core Spec Count    0
Mem Spec Limit     0

```

```
Node Customizations          <0 in submode>
Generic Resources            <0 in submode>
[bright90->configurationoverlay[slurm-client]->roles[slurmclient]]%
```

After the workload manager roles are assigned or unassigned, and after running `imageupdate` (section 5.6.2) for non-head nodes, the associated workload manager services automatically start up or stop as appropriate.

The configuration overlay role values are inherited by categories and nodes, unless the categories and nodes have their own values set. Thus, for role properties, a value set at node level overrides values set at category level, and a value set at configuration overlay level overrides a value set at category level. This is typical of how properties of objects are inherited in Bright Cluster Manager levels.

Workload Manager Role Assignment To A Category With `cmsh`

In `cmsh`, workload manager role assignment to a node category can be done using category mode, using the category name, assigning a role from the `roles` submodule, setting the WLM instance for that role, and committing the modified role:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category
[bright90->category]% use default
[bright90->category[default]]% roles
[bright90->category[default]->roles]% assign slurmclient
[bright90->category[default]->roles*[slurmclient*]]% wlm list
Type   Name (key)           Server nodes Submit nodes   Client nodes
-----
slurm  slurm1                bright90    bright90,node001 node001,node002
[bright90->category[default]->roles*[slurmclient*]]% set wlmcluster slurm1
[bright90->category[default]->roles*[slurmclient*]]% commit
```

Settings that are assigned in the `slurmclient` role of the category overrule the `slurmclient` role configuration overlay settings.

The role assignment at category level requires the value for a WLM instance to be specified for `wlmcluster` before the `commit` command is successful.

Workload Manager Role Assignment To An Individual Node With `cmsh`

In `cmsh`, assigning a workload manager role to a head node can be done in device mode. This can be done by using the head node name as the device, assigning the workload manager role to the device, setting the WLM instance value to the role within the `role` submodule, and committing the modified role.

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% use bright90
[bright90->device[bright90]]% roles
[bright90->device[bright90]->roles]% assign slurmserver
[bright90->category[default]->roles*[slurmserver*]]% wlm list
Type   Name (key)           Server nodes Submit nodes   Client nodes
-----
slurm  slurm1                bright90    bright90,node001 node001,node002
[bright90->category[default]->roles*[slurmserver*]]% set wlmcluster slurm1
[bright90->device*[bright90*]->roles*[slurmserver*]]% commit
[bright90->device[bright90]->roles[slurmserver]]%
```

For regular nodes, role assignment is done via device mode, using the node name. The node name is assigned the workload manager role, the WLM instance value is set for that role in the `role` submode, and the modified role is committed.

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% use node001
[bright90->device[node001]]% roles
[bright90->device[node001]->roles]% assign slurmclient
[bright90->device[node001]->roles*[slurmclient*]]% wlm list
Type      Name (key)          Server nodes Submit nodes   Client nodes
-----
slurm     slurm1              bright90    bright90,node001 node001,node002
[bright90->device[node001]->roles*[slurmclient*]]% set wlmcluster slurm1
[bright90->device[node001]->roles*[slurmclient*]]% commit
[bright90->device[node001]->roles[slurmclient]]%
```

The role assignment at node level requires the value for a WLM instance to be specified for `wlmcluster` before the `commit` command is successful.

Role assignment values set in device mode have precedence over any role assignment values set in category mode or for that node. This means, for example, that if a node is originally in a node category with a `slurmclient` role and queues set, then when the node is assigned a `slurmclient` role from device mode, its queue properties are empty by default.

Setting Options For Workload Manager Settings With `cmsh`

In the preceding text, it is explained how the workload manager client or server is assigned a role (such as `slurmclient` or `slurmserver`) within the `roles` submode. It is done from within a main mode of `cmsh`. The main modes from which role assignment can be done are: `configurationoverlay`, `category` or `device`.

Options for workload managers in general: Whatever main mode is used, the workload manager options for a role can then be set with the usual object commands introduced in section 2.5.3.

- **WLM client options:** For example, the configuration options of a WLM client, such as the PBS Pro client, can be seen by using the `show` command on the role. Here it can be seen at a category level, for the default category default:

Example

```
[bright90->category[default]->roles[pbsproclient]]% show
Parameter                                Value
-----
Add services                             yes
All Queues                               no
GPUs                                      0
Name                                       pbsproclient
Properties
Provisioning associations                 <0 internally used>
Queues
Revision
Slots                                    1
Type                                     PbsProClientRole
WLM cluster
```

```

Mom Settings                <submode>
Comm Settings              <submode>
Node Customizations        <0 in submode>

```

The Slots option can be set in the role

Example

```

[bright90->category[default]->roles[pbsproclient]]% set slots 2
[bright90->category*[default*]->roles*[pbsproclient*]]% commit
[bright90->category[default]->roles[pbsproclient]]%

```

- **WLM server options:** Similarly, WLM server options can be managed from an assigned server role. For PBS, the pbsproserver role for a device shows:

Example

```

[bright90->device[bright90]->roles[pbsproserver]]% show
[bright90->device[bright90]->roles]% use pbsproserver
Parameter                               Value
-----
Add services                             yes
Name                                     pbsproserver
Provisioning associations                 <0 internally used>
Revision
Type                                     PbsProServerRole
WLM cluster
External Server                          no
Comm Settings                            <submode>

```

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

```

[bright90->device[bright90]->roles[pbsproserver]]% set externalserver on
[bright90->device[bright90]->roles[pbsproserver*]]% commit

```

For convenience, setting it on the head node is recommended.

The Bright View equivalent of configuring externalserver is described on page 256.

7.4.3 Monitoring The Workload Manager Services

By default, the workload manager services are monitored. 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.

Workload manager roles and corresponding services can be disabled using `cm-wlm-setup` (section 7.3), Bright View role configuration (section 7.4.1), or `cmsh` role configuration (section 7.4.2).

The daemon service states can be viewed for each node via the shell, `cmsh`, or Bright View (section 3.11).

Queue submission and scheduling daemons normally run on the head node. From Bright View their states are viewable via the clickpath to the services running on the node. For example, on a head node (figure 7.7), via the clickpath:

```

Devices→Head Nodes→[bright90]→Settings→FILESYSTEM MOUNTS...→Services

```

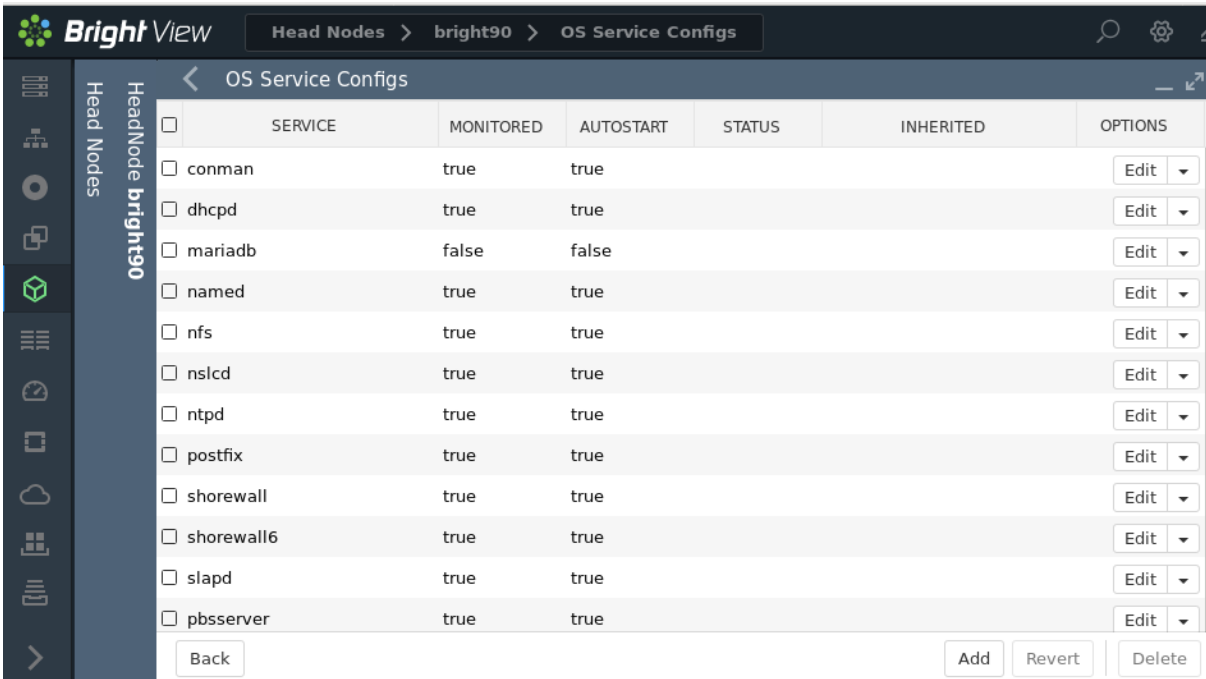


Figure 7.7: Services seen on head node in Bright View

For a regular node, a similar clickpath for node001, for example, is: Devices→Physical Nodes→node001→Settings→FILESYSTEM MOUNTS...→Services and leads to a view of services on the regular nodes (figure 7.8)

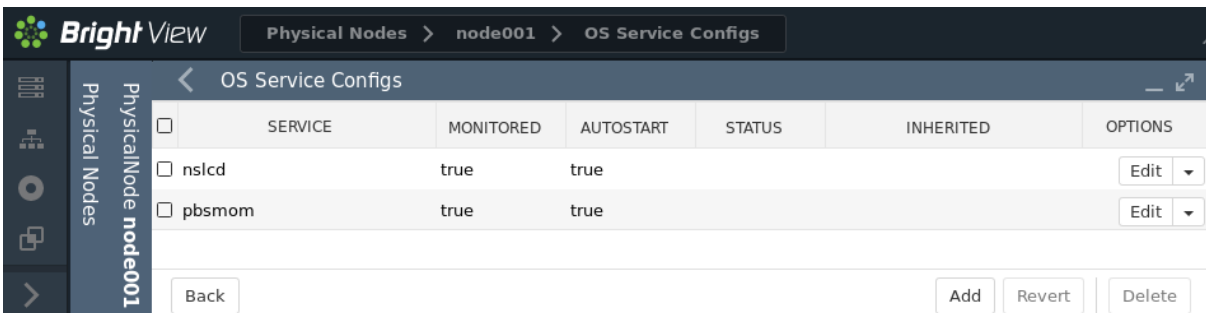


Figure 7.8: Services seen on regular node in Bright View

Considering only the WLMs: in figure 7.7 the pbsserver is seen running on the head node, while in figure 7.8 the pbsmom server is seen running on the compute node.

The roles that set up that result in those servers running on the head node and compute nodes are seen in figures 7.9 and 7.10:

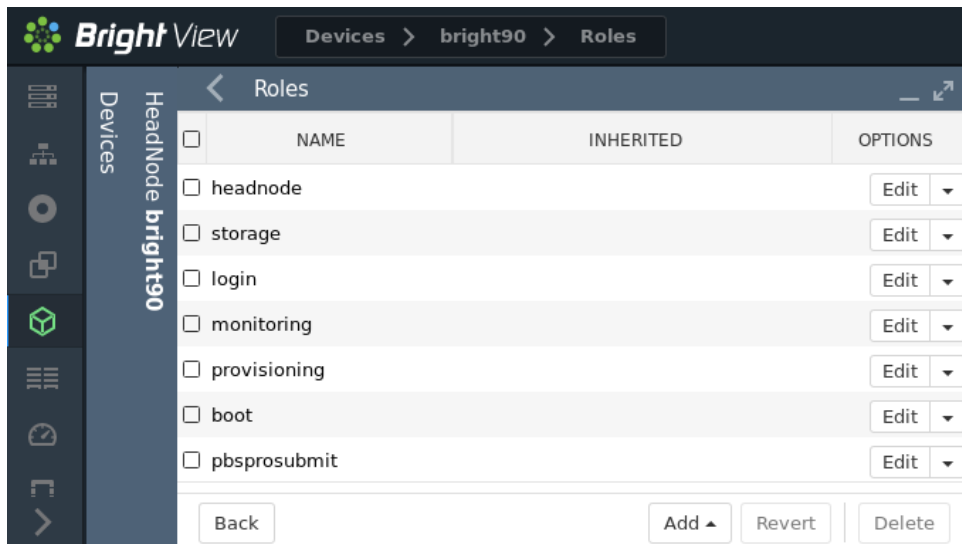


Figure 7.9: Roles seen on head node in Bright View

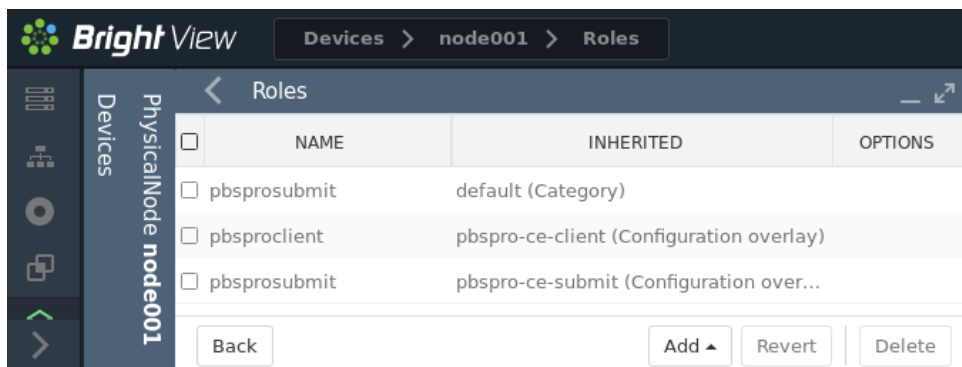


Figure 7.10: Roles seen on regular node in Bright View

The roles seen in these figures are from the defaults that `cm-wlm-setup` provides in an express setup.

For regular nodes, the inheritance of roles from category level or configuration overlay level is indicated by the values in the `INHERITED` column. Thus, in figure 7.10, the `pbsprosubmit` role is decided by the default setting from the category level, while the `pbsproclient` role is decided by the default setting in the `pbspro-ce-client` overlay.

The assignment of roles can be varied to taste for WLMs. This allows WLM services to run on the head node or on the regular nodes.

From `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@bright90 ~]# cmsh -c "device services node001; status"
Service      Status
-----
nslcd        [ UP ]
pbsmom       [ UP ]
[root@bright90 ~]# cmsh -c "device services bright90; status"

Service      Status
-----
```



```

    ...
pbsserver  [ UP ]

```

Roles can be viewed from within the main modes of `configurationoverlay`, `category`, or `device`. One-liners to view these are:

Example

```

[root@bright90 ~]# cmsht -c "configurationoverlay; list"
Name (key)      Priority  Nodes      Categories  Roles
-----
pbspro-ce-client 500      nodes      default     pbsproclient
pbspro-ce-server 500      nodes      default     pbsproserver
pbspro-ce-submit 500      bright90   default     pbsprosubmit

```

```

[root@bright90 ~]# cmsht -c "category; use default; roles; list"
Name (key)
-----
pbsprosubmit

```

```

[root@bright90 ~]# cmsht -c "device use node001; roles; list"
Name (key)
-----
[overlay:pbspro-ce-client] pbsproclient
[overlay:pbspro-ce-submit] pbsprosubmit

```

```

[root@bright90 ~]# cmsht -c "device use bright90; roles; list"
Name (key)
-----
[overlay:pbspro-ce-server] pbsproserver
boot
headnode
login
monitoring
pbsprosubmit
provisioning
storage

```

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:

```
FreezeChangesTo<workload manager>Config = <true|false>
```

in `cmd.conf` (Appendix C), where `<workload manager>` takes the value of `Slurm`, `UGE`, `LSF`, or `PBSPPro`, 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 9.0 system is given in Appendix F.

7.5.1 Configuring And Running Slurm

Slurm Version

At the time of writing (April 2022), Bright Cluster Manager is integrated with Slurm by using the following packages from the Bright Cluster Manager repositories:

- `slurm19`, `slurm19-*`: For Slurm version 19.05.8
- `slurm20`, `slurm20-*`: For Slurm version 20.02.7
- `slurm20.11`, `slurm20.11-*`: For Slurm version 20.11.8
- `slurm21.08`, `slurm21.08-*`: For Slurm version 21.08.5

The exact list of packages can be found by running, for example:

Example

```
[root@bright90 ~]# i="slurm19 slurm20 slurm20.11 slurm21.08"
for j in $i ; do echo "===="; yum info $j $j-* | grep ^Name; done
====
Name       : slurm19
Name       : slurm19-client
Name       : slurm19-contribs
Name       : slurm19-devel
Name       : slurm19-perlapi
Name       : slurm19-slurmdbd
Name       : slurm19-example-configs
Name       : slurm19-openlava
Name       : slurm19-pam
Name       : slurm19-torque
====
Name       : slurm20
Name       : slurm20-client
Name       : slurm20-contribs
...
```

Important updates from upstream are patched into the Bright Cluster Manager repositories.

Updating From `slurm19` To `slurm20`

Upgrading between major versions of Slurm is possible. It is a good idea to upgrade one version at a time, rather than jumping 2 or more versions ahead, which requires a full wipe of the Slurm configuration.

An upgrade from one major version of Slurm to another can be carried out according to the following example, which is for an update from major version 19 to version 20, and avoids total reconfiguration of the Slurm configuration:

- It is recommended that no jobs are running. Draining nodes (section 7.7.3) is one way to arrange this over time. No new jobs run on a drained node, but old ones are allowed to finish.
- When all running jobs are finished, then Slurm server services—`slurmctld` and `slurmdbd`—should be stopped using `cmsh` or Bright View (section 3.11.2):

Example

```
[bright90->device[bright90]->services]% stop slurmctld
[bright90->device[bright90]->services]% stop slurmdbd
```

- The old Slurm packages should then be removed. There can be only one version of Slurm at a time, so there will be a package installation conflict if a new version is installed while an old one is still there.

Removal can be carried out on RHEL-based systems with, for example:

```
[root@bright90 ~]# yum remove slurm19*
```

The old packages must also be removed from each software image that uses it:

```
[root@bright90 ~]# cm-chroot-sw-img /cm/images/<software image>
...
[root@<software image> /]# yum remove slurm19*
...removal takes place...
[root@<software image> /]# exit
```

- The new packages can then be installed. For installation onto the RHEL head node, the installation might be carried out as follows:

```
[root@bright90 ~]# yum install slurm20 slurm20-client slurm20-contribs slurm20-slurmdbd
```

The client package can be installed in each software image with, for example:

```
[root@bright90 ~]# cm-chroot-sw-img /cm/images/<software image>
...
[root@<software image> /]# yum install slurm20-client
...installation takes place
[root@<software image> /]# exit
```

Other Slurm packages from the repository may also be installed on the head node and within the software images, as needed.

- The new Slurm version is then set in cmsh or Bright View, in the Slurm WLM cluster configuration:

Example

```
[root@bright90 ~]# cmsh
[bright90]% wlm use slurm
[bright90->wlm[slurm]]% set version 20; commit
```

- Slurm server services `slurmctld` and `slurmdbd` should then be started again using `cmsh` or Bright View

Example

```
[bright90->device[bright90]->services]% start slurmdbd
[bright90->device[bright90]->services]% start slurmctld
```

- The nodes can then have their new image placed on them, and the new Slurm configuration can then be taken up. This can be done in the following two ways:

1. The regular nodes can then be restarted to supply the live nodes with the new image and get the new Slurm configuration running.
2. Alternatively, the `imageupdate` command (section 5.6.2) can be run on the live nodes to supply them with the image.

Running the `imageupdate` command in dry mode (the default) first is recommended. The `synclog` command can then be run to check there are no unexpected changes that will take place due to the update. If all is well, then `imageupdate`'s wet mode flag `-w` can be used in order to really carry out the task.

For example, the change can be checked, and then actually carried out, for the image on `node001` with:

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% imageupdate
Performing dry run (use synclog command to review result, then pass -w to perform real update)...
...some messages...
imageupdate [ COMPLETED ]
[bright90->device[node001]]% synclog
...rsync dry run output...
[bright90->device[node001]]% imageupdate -w
...same messages as before, but this time it really happens...
[bright90->device[node001]]% commit
```

The last `commit` command triggers the reconfiguration of the file `/etc/systemd/system/slurmd.service.d/99-cmd.conf` on the node. After a short time—around 30 seconds—the file is regenerated. The `slurmd` service on the node can then be restarted with:

```
[root@bright90 ~]# ssh node001 "systemctl daemon-reload"
cmsh -c "device services node001; use slurmd; restart"
```

Configuring Slurm

After package setup is done with `cm-wlm-setup` (section 7.3), Slurm software components are installed in a symlinked directory `/cm/shared/apps/slurm/current`.

Slurm clients and servers can be configured to some extent via role assignment (sections 7.4.1 and 7.4.2).

Using `cmsh`, advanced option parameters can be set under the `slurmclient` and `slurmserver` roles. The settings for the roles can be done at configuration overlay, category, or node level (sections 2.1.5, 2.1.6).

By default, the `cm-wlm-setup` utility configures Slurm using configuration overlays.

Example

```
[bright90->configurationoverlay]% list
```

Name (key)	Priority	All head nodes	Nodes	Categories	Roles
<code>slurm-accounting</code>	500	yes			<code>slurmaccounting</code>
<code>slurm-client</code>	500	no		default	<code>slurmclient</code>
<code>slurm-server</code>	500	yes			<code>slurmserver</code>
<code>slurm-submit</code>	500	yes		default	<code>slurmsubmit</code>

The settings within the roles can be viewed and modified. For example, the `slurmclient` role of the `slurm-client` configuration overlay can be viewed:

Example

```
[bright90->configurationoverlay]% roles slurm-client
[bright90->configurationoverlay[slurm-client]->roles]% show
Parameter                               Value
-----
Name                                     slurmclient
Revision
Type                                     SlurmClientRole
Add services                             yes
WLM cluster                              slurm
Slots                                    0
All Queues                              no
Queues                                   defq
Provisioning associations                <0 internally used>
Features
Sockets                                  0
Cores Per Socket                        0
ThreadsPerCore                          0
Boards                                   0
SocketsPerBoard                         0
RealMemory                              0B
NodeAddr
Weight                                   0
Port                                     0
TmpDisk                                  0
Reason
CPU Spec List
Core Spec Count                          2
Mem Spec Limit                           0B
Auto Detect                              None
Node Customizations                     <0 in submode>
Generic Resources                       <0 in submode>
Cpu Bindings                             None
```

The settings can be modified. For example Core Spec Count:

Example

```
[bright90->configurationoverlay[slurm-client]->roles]% set slurmclient corespeccount 2
[bright90->configurationoverlay*[slurm-client*]->roles*]% commit
```

As usual, values set at node level override the values set at categories and configuration overlays level.

For example, to set corespeccount to 4, only for node001 but not for other nodes, the session might run further as:

Example

```
[bright90->configurationoverlay[slurm-client]->roles]% device use node001
[bright90->device[node001]]% roles
[bright90->device[node001]->roles]% assign slurmclient
[bright90->device*[node001*]->roles*[slurmclient*]]% set corespeccount 4
[bright90->device*[node001*]->roles*[slurmclient*]]% commit
Field                                     Message
-----
wlmCluster                               Error: The WLM cluster should be set
[bright90->device*[node001*]->roles*[slurmclient*]]% wlm list
Type  Name (key)                               Server nodes Submit nodes  Client nodes
```

```
-----
Slurm slurm                bright90    bright90,node001 node001,node001
[bright90->device*[node001*]->roles*[slurmclient*]]% set wlmcluster slurm
[bright90->device*[node001*]->roles*[slurmclient*]]% commit
```

In the preceding session, the role needs to be assigned at node level with `assign slurmclient` because it does not initially exist at node level. If it already existed, then `use slurmclient` could have been used to descend into that role.

Also in the preceding session, one of the values that the Slurm client needs to know is `wlmcluster`, which decides which WLM it is to work with on the cluster. The value is selected from the list of WLM instance names in `wlm` mode.

The level of the active role can be seen with the `list` command. For example, the Slurm client role assignment at node level is seen here:

Example

```
[bright90->device[node001]->roles[slurmclient]]% list
Name (key)
-----
[overlay:slurm-submit] slurmsubmit
slurmclient
```

Removing the assignment has the `list` command display the configuration overlay Slurm client role assignment:

Example

```
[bright90->device[node001]->roles[slurmclient]]% unassign slurmclient; commit
[bright90->device[node001]->roles[slurmclient]]% list
Name (key)
-----
[overlay:slurm-submit] slurmsubmit
[overlay:slurm-client] slurmclient
```

Generic resources (gres) configuration in Slurm: In order to configure generic resources, the `genericresources` mode can be used to set a list of objects. Each object then represents one generic resource available on nodes.

Each value of `name` in `genericresources` must already be defined in the list of `GresTypes`. The list of `GresTypes` is defined in the `wlm` role for the instance.

Example

```
[bright90->wlm[slurm]]% get grestypes
gpu
```

Several generic resources entries can have the same value for `name` (for example `gpu`), but must have a unique alias. The alias is a string that is used to manage the resource entry in `cmsh` or in Bright View. The string is enclosed in square brackets in `cmsh`, and is used instead of the `name` for the object. The alias does not affect Slurm configuration.

For example, to add two GPUs for all the nodes in the default category which are of type `k20xm`, and to assign them to different CPU cores, the following `cmsh` commands can be run:

Example

```
[bright90]% configurationoverlay use slurm-client
[bright90->configurationoverlay[slurm-client]]% roles
[bright90->configurationoverlay[slurm-client]->roles*]% use slurmclient
[...[slurmclient]]% genericresources
[...[slurmclient]->genericresources]% add gpu0
[...[slurmclient*]->genericresources*[gpu0*]]% set name gpu
[...[slurmclient*]->genericresources*[gpu0*]]% set file /dev/nvidia0
[...[slurmclient*]->genericresources*[gpu0*]]% set cores 0-7
[...[slurmclient*]->genericresources*[gpu0*]]% set type k20xm
[...[slurmclient*]->genericresources*[gpu0*]]% add gpu1
[...[slurmclient*]->genericresources*[gpu1*]]% set name gpu
[...[slurmclient*]->genericresources*[gpu1*]]% set file /dev/nvidia1
[...[slurmclient*]->genericresources*[gpu1*]]% set cores 8-15
[...[slurmclient*]->genericresources*[gpu1*]]% set type k20xm
[...[slurmclient*]->genericresources*[gpu1*]]% commit
[...[slurmclient]->genericresources[gpu1]]% list
Alias (key) Name      Type      Count      File
-----
gpu0      gpu      k20xm      /dev/nvidia0
gpu1      gpu      k20xm      /dev/nvidia1
[...[slurmclient]->genericresources[gpu2]]%
```

After the generic resources are added, Bright Cluster Manager updates the `gres.conf` file.

Since Bright Cluster Manager version 8.2 and higher, a single `gres.conf` configuration file, located at `/cm/shared/apps/slurm/var/etc/slurm/gres.conf` is used.

If the category consists of `node001` and `node002`, then the entries to the `gres.conf` file in this case would look like:

Example

```
# This section of this file was automatically generated by cmd. Do not edit manually!
# BEGIN AUTOGENERATED SECTION -- DO NOT REMOVE
NodeName=node[001,002] Name=gpu Type=k20xm Count=1 File=/dev/nvidia0 Cores=0-7
NodeName=node[001,002] Name=gpu Type=k20xm Count=1 File=/dev/nvidia1 Cores=8-15
# END AUTOGENERATED SECTION -- DO NOT REMOVE
[root@bright90 ~]#
```

Slurm node settings: The parameter value for a Slurm option in `slurm.conf` is set by `CMDaemon`, if its value is not 0.

A parameter value of 0 means that the default values of Slurm are used. These often have the value 0.

The advanced options that `CMDaemon` manages for Slurm are:

Bright View Option	Slurm Option	Description
Features	Feature=<string> entry in the file slurm.conf	Arbitrary strings can be entered to indicate some characteristics of a node, one string per entry. For example: text1 text2 and so on. These become part of the: Feature=text1,text2... attribute to the NodeName=<node name> entry line in slurm.conf, as indicated in man(5) slurm.conf. The strings also become added attributes to the GresTypes entry of that file. Default: blank.
Slots	CPU	Number of logical processors on the node. Default: 0
Sockets	Sockets	Processor chips on node. If this is defined, then SocketsPerBoard must not be defined. Default: 0
Cores per socket	CoresPerSocket	Number of cores per socket. Default: 0
ThreadsPerCore	ThreadsPerCore	Number of logical threads for a single core. Default: 0
Boards	Boards	Number of baseboards in a node. Default: 0
SocketsPerBoard	SocketsPerBoard	Number of processor chips on baseboard. If this is defined, then Sockets must not be defined. Default: 0
RealMemory	RealMemory	Size of real memory on the node, MB. Default: 0
NodeHostname	NodeHostname	Default: as defined by Slurm's NodeName parameter.

...continues

...continued

Bright View Option	Slurm Option	Description
NodeAddr	NodeAddr	Default: as set by Slurm's NodeHostname parameter.
State	State	State of the node with user jobs. Possible Slurm values are: DOWN, DRAIN, FAIL, FAILING, and UNKNOWN. Default: UNKNOWN
Weight	Weight	The priority of the node for scheduling. Default: 0
Port	Port	Port that slurmd listens to on the compute node. Default: as defined by SlurmdPort parameter. If SlurmdPort is not specified during build: Default: 6818
TmpDisk	TmpDisk	Total size of Slurm's temporary filesystem, TmpFS, typically /tmp, in MB. TmpFS is the storage location available to user jobs for temporary storage. Default: 0
Options	extra options	Extra options that are added to slurm.conf

Further Slurm documentation is available:

- via man pages under /cm/shared/apps/slurm/current/man.
- as HTML documentation in the directory /cm/shared/apps/slurm/current/share/doc/slurm-20.11.5/html
- at the Slurm website at <http://slurm.schedmd.com/documentation.html>

Slurm is set up with reasonable defaults, but administrators familiar with Slurm can reconfigure the configuration file using a web browser with the following JavaScript-based configuration generators:

- /cm/shared/apps/slurm/current/share/doc/slurm-20.11.5/html/configurator.easy.html: this is a simplified configurator
- /cm/shared/apps/slurm/current/share/doc/slurm-20.11.5/html/configurator.html: this is the full version of the configurator.

If the configuration file becomes mangled beyond repair, the original default can be regenerated once again by re-installing the Slurm package, then running the script /cm/shared/apps/slurm/var/cm/cm-restore-db-password, and then running cm-wlm-setup. Care must be taken to avoid duplicate parameters being set in the configuration file—slurmd may not function correctly in such a configuration.

Running Slurm

Slurm can be disabled and re-initialized with the cm-wlm-setup tool (section 7.3) during package installation itself.

Alternatively, role assignment and role removal can be used to adjust what nodes, if any, run Slurm. The assignment and removal of roles can be carried out from Bright View (section 7.4.1) or cmsh (section 7.4.2).

The Slurm workload manager runs these daemons:

1. as servers:
 - (a) `slurmdbd`: The database that tracks job accounting. It is part of the `slurmdbd` service.
 - (b) `slurmctld`: The controller daemon. Monitors Slurm processes, accepts jobs, and assigns resources. It is part of the `slurm` service.
 - (c) `munged`: The authentication (client-and-server) daemon. It is part of the `munge` service.
2. as clients:
 - (a) `slurmd`: The compute node daemon that monitors and handles tasks allocated by `slurmctld` to the node. It is part of the `slurm` service.
 - (b) `slurmstepd`: A temporary process spawned by the `slurmd` compute node daemon to handle Slurm job steps. It is not initiated directly by users or administrators.
 - (c) `munged`: The authentication (client-and-server) daemon. It is part of the `munge` service.

Logs for the daemons are saved on the node that they run on. Accordingly, the locations are:

- `/var/log/slurmdbd`
- `/var/log/slurmd`
- `/var/log/slurmctld`
- `/var/log/munge/munged.log`

7.5.2 Installing, Configuring, And Running UGE

The workload manager, Univa Grid Engine (UGE), is a further development of the venerable SGE. Bright Cluster Manager 9.0 supports integration with UGE version 8.6.

UGE should be picked up directly from the Univa website at <http://www.univa.com>.

The installation and integration of UGE into Bright Cluster Manager 9.0 can be carried out as in the following steps:

1. These UGE `tar.gz` file collections should be downloaded from the Univa website to a directory on the head node:
 - Binary files:
 - The 64-bit Linux bundle
`ge-<uge_ver>-bin-lx-amd64.tar.gz`
 - or the 32-bit Linux 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: 8.6.0

To avoid installation issues, a check should be done to ensure that during the download, the `tar.gz` files have not been renamed, or that their names not been changed to upper case. Both packages must be located in the same directory before installation.

If a failover setup already exists, then the installation should be done on the active head node in the steps that follow.

2. The `cm-uge` package should be installed from the Bright Computing repository via the distribution's package manager. That is, via, `yum`, `zypper`, or `apt`, as appropriate.

The package installs, amongst others, the following template files under `/cm/shared/apps/uge/var/cm/`:

- (a) An environment module template, `uge.module.template`
- (b) An installation configuration template, `inst_template.conf.template`
- (c) Some other template files for a default GE configuration, under the directory `templates`

The templates decide the configuration of UGE (section “*Configuring UGE*”, page 274).

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@bright90 ~]# cd /cm/shared/apps/uge/var/cm
[root@bright90 cm]# cp inst_template.conf.template inst_template.conf
[root@bright90 cm]# vi inst_template.conf
```

The copied file, `inst_template.conf`, can be changed by the administrator to decide the location of UGE software and how UGE handles jobs, if needed. The changed file overrides the settings suggested by the `.template` file, when the Bright Cluster Manager utility `cm-wlm-setup` runs in the next step of the installation procedure.

Some of the values in the key-value pairs in the file are enclosed by percentage signs, `%`. For example: `CELL_NAME="%CELL_NAME%"`. Such flagged values should not be modified, since they are replaced by Bright Cluster Manager values by `cm-wlm-setup` during installation.

Values not enclosed by percentage signs are not replaced by `cm-wlm-setup`. Such unflagged values can, if needed, be tuned in the copied file by the administrator. These values are then kept by `cm-wlm-setup` during installation, and used when UGE is run.

3. `cm-wlm-setup` is run. The directory where the downloaded UGE files are located can be specified with the `--archives-location` option, if using the CLI mode with options.

Example

```
root@bright90:~# ls -al /root/uge/
total 67892
drwxr-xr-x  2 root root      82 Dec 18 16:03 .
drwx----- 10 root root    4096 Dec 18 16:03 ..
-rw-r--r--  1 root root 63426260 Dec 18 16:03 ge-8.6.7-demo-bin-lx-amd64.tar.gz
-rw-r--r--  1 root root  6088929 Dec 18 16:03 ge-8.6.7-demo-common.tar.gz
root@bright90:~# cm-wlm-setup --wlm uge --setup --archives-location /root/uge
```

4. The nodes are rebooted. The UGE command `qhost` then displays an output similar to:

Example

```
[root@bright90 ~]# module load uge
[root@bright90 ~]# qhost
HOSTNAME  ARCH      NCPU NSOC NCOR NTHR NLOAD  ...
-----
global   -         -    -    -    -    -    ...
node001  lx-amd64  8    1    1    8  0.01  ...
node002  lx-amd64  8    1    1    8  0.01  ...
node003  lx-amd64  8    1    1    8  0.01  ...
node004  lx-amd64  8    1    1    8  0.01  ...
```

The output in the preceding example has been truncated for this manual, for convenience.

Configuring UGE

After installation and initialization, UGE has reasonable defaults, with `$SGE_ROOT` set to `/cm/shared/apps/uge/current`.

By default, the UGE application is installed in `/cm/shared/apps/uge/current`, and job examples are kept in `/cm/shared/examples/workload/sge/jobscripts/`.

Running UGE

The UGE workload manager runs the following two daemons:

1. an `sge_qmaster` daemon running on the head node. This handles queue submissions and schedules them according to criteria set by the administrator.
2. an `sge_execd` execution daemon running on each compute node. This accepts, manages, and returns the results of the jobs on the compute nodes.

Messages from the qmaster daemon are logged in:

```
/cm/shared/apps/uge/current/<uge instance name>/spool/messages
```

On the associated compute nodes the execution log messages exists, alongside other job tracking files and directories, at:

```
/cm/local/apps/uge/var/spool/node<number>/messages
```

where `node<number>` is the node name, for example:

```
node001, node002 ...
```

UGE Submission, Administrative And Execution Hosts

UGE is still similar to SGE in many ways, but is evolving.

In UGE terminology:

- A *submission host*, or *submit host*, is a node that is allowed to submit jobs. Submission can be from the user, or from another job run by the user.
- An *execution host*, or *exec host*, is a node on which the computational part of the job is executed.
- An *administrative host*, or *admin host*, is a node that can carry out administrative Grid Engine commands.

Standard Bright Cluster Manager Use

In a standard Bright Cluster Manager configuration, a *submission*, *administrative* or *execution host* in UGE is configured with the following `ugeclient` role properties:

1. Admin Host – adds the node to a list of administrative hosts in UGE,
2. Exec Host – adds the node to a list of execution hosts in UGE,
3. Submit Host – adds the node to a list of submission hosts in UGE.

The nodes that are assigned a `ugeserver` role are always automatically added to the administrative and submission host lists.

Non-standard 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 additional execution hosts can be specified in the advanced configuration directive, `AdditionalExecHosts`, in the `cmd.conf` file:

Example

```
AdvancedConfig = {"AdditionalExecHosts=node002,login01"}
```

2. If the `ugeclient` role cannot be added, for instance due to the login service being managed outside of Bright Cluster Manager, then the execution hosts can be specified in the advanced configuration directive, `AdditionalSubmitHosts`, in the `cmd.conf` file:

Example

```
AdvancedConfig = {"AdditionalSubmitHosts=node002,login01"}
```

Parallel Environments

A parallel environment (PE) allows the configuration of applications that use shared or distributed memory. The main—but not sole—purpose of a PE is to provide a job environment within which MPI applications can run, using various MPI libraries. PEs can be provided by the cluster software or hardware vendor, as well as distributed with UGE itself. Default PEs are provided for various MPI libraries, such as Open MPI, MVAPICH, and so on. The default PEs are automatically added to UGE when `cm-wlm-setup` installs UGE.

The `CMDaemon` front ends, i.e. `cmsh` or Bright View, can be used to add, remove, or modify the existing PE. Each UGE cluster has a `Parallel Environments` submode. In the `Parallel Environments` submode, each PE is represented as a separate object with associated properties. For example, with `cmsh` the `wlm` mode is used to access the Open MPI properties within a UGE cluster (some text elided):

Example

```
[root@bright90 ~]# cmsh
[bright90]% wlm
[bright90->wlm]% list
Type      Name (key)  Server nodes  Submit nodes  Client nodes
-----
UGE       uge         bright90     node001..node003,bright90  node001..node003
[bright90->wlm]% use uge
[bright90->wlm[uge]]% parallelenvironments
[bright90->wlm[uge]->parallelenvironments]% list
Name (key)  Slots  Allocation Rule
-----
impi        999999  $round_robin
mpich       999999  $round_robin
mpich2      999999  $round_robin
openmpi     999999  $round_robin
openmpi_rankfile 999999  4
[bright90->wlm[uge]->parallelenvironments]% use openmpi
[bright90->wlm[uge]->parallelenvironments[openmpi]]% show
Parameter  Value
-----
Daemon Forks Slaves  yes
Master Forks Slaves  no
Revision
Type          UGEParallelEnvironment
```

```

Name                openmpi
Slots               999999
User Lists          NONE
X User Lists        NONE
Start Procedure Arguments  NONE
Stop Procedure Arguments  NONE
Allocation Rule     $round_robin
Control Slaves      yes
Job Is First Task   no
Urgency Slots       min
Accounting Summary  yes
Per PE Task Prolog  NONE
Per PE Task Epilog  NONE
Extra Parameters
[bright90->wlm[uge]->parallelenvironments[openmpi]]%

```

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 with the parent distribution Berkeley DB packages. The UGE versions include extra sanity tests for the UGE spool database, and it is best to use them instead of the standard versions. The utilities can be found in on the head node under:

```
/cm/shared/apps/uge/current/utilbin/lx-amd64/
```

This directory is not added to the `$PATH` when loading the UGE environment with `module load uge`.

The full Berkeley DB documentation for these tools is part of the UGE distribution. A local HTML format copy of the documentation is available at:

```
/cm/shared/apps/uge/current/doc/berkeleydb/db_*.html
```

GPU And MIC Management

When managing host resources such as GPU or Xeon Phi (MIC) PCI-e cards¹, 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.

¹Integration of PCI-e MIC cards is discontinued from Bright Cluster Manager 8.2 onwards, but UGE support for the cards via RSMAP attributes continues. This means that the administrator typically would download the MIC MPSS packages, compile the driver, install the images manually, and after getting the MICs to work, would integrate the MIC with the appropriate RSMAP attributes

- `Gpus`: the number of GPUs on a host.

The `GPU devices` parameter has a higher priority than `Gpus`, so that if names are set in `GPU devices`, then they are always used.

GPUs can also be bound to particular CPU sockets and CPU cores, using a *topology mask*.

The mask is defined with a set of characters. In principle, the mask is applied to the computing units available for a device. The computing units correspond to processing units, and are grouped as CPU sockets, cores, and hardware threads. These correspond to the letters S, C, T. In the mask. An uppercase character allows a device to use a computing unit, while a lowercase character forbids it, as indicated by the following table:

Table 7.5.2: Mask Composition

Unit To Mask	Enable Unit with	Disable Unit with
Socket	S	s
Core	C	c
Hardware Thread	T	t

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.
- `SCcCCScCC`: 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@bright90 ~]# cmsh
[bright90]% category roles default
[bright90->device[bright90]->roles]% assign ugeclient; commit
[bright90->device[bright90]->roles[ugeclient]] show | grep -i gpu
GPU devices          gpu0 gpu1
GPUs                  2
```

The RSMAP attribute value is then:

```
GPU=2(gpu0 gpu1)
```

- Allow GPUs to use only different sockets:

```
[bright90->device[bright90]->roles[ugeclient]]% show | grep -i gpu
GPU devices          gpu0:SCCCScCCC gpu1:ScCCcSCCC
GPUs                  0
```

The RSMAP attribute value is then:

```
GPU=2(gpu0:SCCCScCCC gpu1:ScCCcSCCC)
```

- 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:

```
[bright90->device[bright90]->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*.

7.5.3 Configuring And Running PBS

PBS Variants

Bright Cluster Manager provides three variants of PBS:

- **PBS Pro Commercial Edition.** This is the commercial variant. It requires a license or a license server in order to run jobs. This information can be provided during a run of the setup wizard, or it can be manually configured after setup.
 - The version 2020 packages for the commercial variant are available as:
 - * `pbspro2020` and
 - * `pbspro2020-client`.
 - The version 2021 packages for the commercial variant are available as:
 - * `pbspro2021` and
 - * `pbspro2021-client`.

Pre-2020 major versions of this PBS variant were denoted by 2 digits (such as 18 or 19) to signify the year. Since version 20 the numbering is denoted by 4 digits (2020 and 2021).
- **PBS Pro Community Edition (or CE).** This is the variant with community support (<http://pbspro.org>). The version 19 packages for the community edition are available as:
 - `pbspro19-ce` for the server
 - `pbspro19-ce-client` for the compute nodes.
- For version 20 Altair is renaming PBS Pro Community Edition to OpenPBS, and the packages are available as:
 - `openpbs20` for the server, and
 - `openpbs20-client` for the compute nodes.
- For version 22 the packages are available as:
 - `openpbs22-05` for the server, and
 - `openpbs22-05-client` for the compute nodes.

All three variants can be installed as a selection option during Bright Cluster Manager 9.0 installation, at the point when a workload manager must be selected (figure 3.10 of the *Installation Manual*). Alternatively they can be installed later on, when the cluster has already been set up.

When no particular edition is specified in the Bright Cluster Manager documentation, then the text is valid for all editions.

If Bright Cluster Manager has already been set up without PBS, then the `cm-wlm-setup` tool (section 7.3) should be used to install and initialize PBS.

PBS Pro Versions

Bright Cluster Manager is integrated with the version 20 (or 2020) PBS Pro Commercial and version 19 PBS Pro CE workload managers, and with the version 20 and 22.05 OpenPBS workload manager.

The PBS packages can be installed and removed with a package manager such as YUM.

Bright Cluster Manager provides a similar level of integration for the Commercial and CE packages. It is up to the cluster administrator to decide which version is set up.

For example, an existing `pbspro19-ce` package can be removed and a new `pbspro2020` package can be installed with the package manager as follows (assuming the administrator is using YUM, and that the PBS Pro workload manager is already disabled):

For example, an existing `pbspro-ce` package can be removed and a new `pbspro2020` package can be installed as follows (assuming the administrator is using YUM, and that the PBS Pro workload manager is already disabled):

Example

```
[root@bright90 ~]# yum remove pbspro19-ce pbspro19-ce-client
[root@bright90 ~]# yum remove --installroot=/cm/images/default-image pbspro19-ce-client
[root@bright90 ~]# yum install pbspro2020 pbspro2020-client
[root@bright90 ~]# yum install --installroot=/cm/images/default-image pbspro2020 pbspro2020-client
```

The PBS packages that Bright Computing provides from the Bright Computing repositories should be used instead of any other available versions, such as from the Linux distribution.

Installing PBS

After package installation via the package manager, as described in the preceding section, both PBS Pro commercial and PBS Pro CE can be installed and initialized to work with Bright Cluster Manager via `cm-wlm-setup`. With no options, an Ncurses session is started to guide the process. The alternative CLI process with options might take the following forms:

Example

```
[root@bright90 ~]# cm-wlm-setup --wlm pbspro --wlm-cluster-name ppro --license <license information>
```

or

```
[root@bright90 ~]# cm-wlm-setup --wlm pbspro-ce --wlm-cluster-name pce
```

The *license information* is either a path to a license file, or it is a Altair license server address list in the format:

```
<port1>@<host1>:<port2>@<host2>:<...>@<...>:<portN>@<hostN>
```

This license information can also be set manually for the `pbs_license_info` attribute. For example if there is just one license server, `pbspro-license-server`, serving on port 6200, it could be set with `qmgr` as follows:

Example

```
qmgr -c "set server pbs_license_info = 6200@pbspro-license-server"
```

The software components are installed and initialized by default under the Spool directory, which is defined by the PBS_HOME environment variable. The directory is named after the WLM cluster name, and follows a path of the form:

```
/cm/shared/apps/pbspro/var/spool/<WLM cluster name>
```

or

```
/cm/shared/apps/pbspro-ce/var/spool/<WLM cluster name>
```

as appropriate.

The paths /cm/shared/apps/pbspro, /cm/shared/apps/openpbs, or /cm/shared/apps/pbspro-ce are the Prefix settings of the WLM, and depend on whether the WLM is running PBS Pro, OpenPBS, or PBS Pro CE.

Users must load an environment module associated with the cluster name to set \$PBS_HOME and other environment variables, in order to use that cluster.

Example

```
[root@bright90 ~]# module load pbspro/ppro
```

PBS Configuration

PBS documentation is available at <http://www.pbsworks.com/SupportDocuments.aspx>.

By default, PBS examples are available under the directory /cm/shared/examples/workload/pbspro/jobscripts/

Some PBS configuration under Bright Cluster Manager can be done using roles. The roles are the same for all variants of PBS and denoted as PBS Pro roles:

- In Bright View the roles setting allows the configuration of PBS Pro client and server roles.
 - For the PBS Pro server role, the role is enabled, for example on a head node bright90, via a clickpath of:
Devices→Head Nodes→bright90→Settings→Roles→Add→PBS pro server role
 - * Within the role window for the server role, its installation path and server spool path can be specified.
 - For the PBS Pro client role, the role is enabled along a similar clickpath, just ending at PBS pro client role. The number of slots, GPUs and other properties can be specified, and queues can be selected.
- In cmsh the client and server roles can be managed for the individual nodes in device mode, or managed for a node category in category mode, or they can be managed for a configuration overlay in configurationoverlay mode.

For example, if there is a PBS Pro cluster instance called pbsfast, then the head node could be assigned a PBS Pro server role, with the following properties:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device roles bright90
[bright90->device[bright90]->roles]% assign pbsproserver
[bright90->device*[bright90*]->roles*[pbsproserver*]]% show
Parameter                               Value
-----
Add services                             yes
Name                                      pbsproserver
Provisioning associations                 <0 internally used>
```

```

Revision
Type                               PbsProServerRole
WLM cluster
External Server                     no
Comm Settings                       <submode>
[bright90->device*[bright90*]->roles*[pbsproserver*]]% set wlmcluster pbsfast; commit
[bright90->device[bright90]->roles[pbsproserver]]%

```

Similarly, a category of nodes that can be used by the instance pbsfast, and called fastnodes, may have a PBS Pro client role assigned and set with the following properties:

Example

```

[bright90->category[fastnodes]->roles[pbsproclient]]% show
Parameter                           Value
-----
Add services                          yes
All Queues                            no
GPUs                                  0
Name                                   pbsproclient
Properties
Provisioning associations              <0 internally used>
Queues
Revision
Slots                                  1
Type                                   PbsProClientRole
WLM cluster                            pbsfast
Mom Settings                           <submode>
Comm Settings                           <submode>
Node Customizations                   <0 in submode>

```

Cloudbursting with cluster extension may need special handling for PBS. If this feature is needed, then the administrator should contact Bright Computing support via the website <https://support.brightcomputing.com>.

Further configuration of PBS is done using its `qmgr` command and is covered in the PBS Pro documentation.

Running PBS

For the WLM cluster instances, PBS runs the following four daemons:

1. a `pbs_server` daemon running, typically on the head node. This handles submissions acceptance, and talks to the execution daemons on the compute nodes when sending and receiving jobs. It writes logs to the `var/spool/<WLM cluster instance>/server_logs/` directory, which is a directory that is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-ce`. Queues for this service are configured with the `qmgr` command.
2. a `pbs_sched` scheduler daemon, also typically running on the head node. It writes logs to the `var/spool/<WLM cluster instance>/sched_logs/` directory under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-ce`.
3. a `pbs_mom` execution daemon running on each compute node. This accepts, manages, and returns the results of jobs on the compute nodes. By default, it writes logs
 - to the relative directory `var/spool/<WLM cluster instance>/mom_logs/`, which is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-ce` and

- to `/cm/local/apps/pbspro/var/spool/mom_logs/` on nodes with the client role.
4. a `pbs_comm` communication daemon usually running on the head node. This handles communication between PBS daemons, except for server-to-scheduler and server-to-server daemons communications. It writes logs
 - to the relative directory `var/spool/<WLM cluster instance>/comm_logs/`, which is under `/cm/shared/apps/pbspro` or `/cm/shared/apps/pbspro-ce` and
 - to `/cm/local/apps/pbspro/var/spool/comm_logs/` on nodes with the client role.

Running PBS On Cluster Extension

When PBS is set up with `cm-wlm-setup` or during the installation of the head node, then the `pbsproclient` role is assigned by default via a configuration overlay to the default node category only.

```
[root@bright90 ~]# cmsh -c "category; roles default; list"
Name (key)
-----
[overlay:pbspro-ce-client] pbsproclient
...
```

In order to add cloud nodes to PBS, the administrator can assign the `pbsproclient` role manually.

There are two types of PBS configuration in this case. The configurations can be applied to both the commercial and to the community editions.

1. `pbs_mom` daemons on cloud compute nodes communicate to the `pbs_server` directly.

This scenario is suited to a non-VPN setup where cloud nodes have addresses on the same IP subnet for the cloud and for the on-premise parts of the cluster. Usually this kind of setup is used with Amazon DirectConnection or Azure ExpressRoute. In order to add new cloud nodes, the administrator just needs to assign the `pbsproclient` role to the `cloud` node category or the cloud nodes directly.
2. `pbs_mom` daemons on cloud compute nodes communicate to the `pbs_server` via a separate `pbs_comm` server. Bright Computing recommends the use of the `cloud-director` for this purpose.

This can be useful if cluster extension is configured with a VPN tunnel setup. In this case the `pbs_mom` running on a cloud node communicates with the `pbs_server` by using the VPN connection, and the communication traffic goes via an OpenVPN server. The OpenVPN connection adds overhead on the cloud-director where the OpenVPN daemon runs. If the traffic is routed via `pbs_comm` running on cloud-director, then the OpenVPN server is not used. This is because `pbs_comm` daemon on the cloud director resolves the cloud `pbs_mom` addresses with cloud IP addresses, while `pbs_server` resolves `pbs_comm` on the cloud director by using the VPN tunnel IP.

In order to configure PBS to pass the communication traffic via `pbs_comm`, the administrator should assign `pbsproclient` roles to not only the compute cloud nodes, but also to the cloud-director. On the cloud director, the administrator should enable `pbs_comm` daemon to start, and `pbs_mom` daemon to not start, automatically. These actions are done in the `commsettings` and `momsettings` submodes of `pbsproclient` role.

For the `pbsproclient` role assigned in the configuration overlay, the settings can be accessed in `cmsh` as in the following:

Example

```

[root@bright90 ~]# cmsh
[bright90]% configurationoverlay
[bright90->configurationoverlay]% use pbspro-ce-client
[bright90->configurationoverlay[pbspro-ce-client]]% roles
[bright90->configurationoverlay[pbspro-ce-client]->roles]% use pbsproclient
[bright90->configurationoverlay[pbspro-ce-client]->roles[pbsproclient]]% show
Parameter                               Value
-----
Add services                             yes
All Queues                               no
GPUs                                      0
Name                                      pbsproclient
Properties
Provisioning associations                 <0 internally used>
Queues                                    workq
Revision
Slots                                     1
Type                                      PbsProClientRole
WLM cluster                              pbspro-ce
Mom Settings                             <submode>
Comm Settings                            <submode>
Node Customizations                      <0 in submode>
[bright90->configurationoverlay[pbspro-ce-client]->roles[pbsproclient]]% commsettings
[bright90->configurationoverlay[pbspro-ce-client]->roles[pbsproclient]->commsettings]% show
Parameter                               Value
-----
Comm Routers
Comm Threads                             4
Revision
Start Comm                               no
[bright90->configurationoverlay[pbspro-ce-client]->roles[pbsproclient]->commsettings]% ..
[bright90->configurationoverlay[pbspro-ce-client]->roles[pbsproclient]]% momsettings
[bright90->configurationoverlay[pbspro-ce-client]->roles[pbsproclient]->momsettings]% show
Parameter                               Value
-----
Revision
Output Hostname
Leaf Routers
Leaf Name
Leaf Management FQDN                     no
Start Mom                                yes
Spool                                     /cm/local/apps/pbspro-ce/var/spool

```

Further configuration that should be carried out is to set the `commrouters` parameter on the cloud director to master, and set the `leafrouters` parameter on the compute cloud nodes to the host-name of the cloud director.

For example (some text elided):

Example

```

[root@bright90 ~]# cmsh
[bright90]% category roles cloud-nodes
[bright90->category[cloud-nodes]->roles]% assign pbsproclient
[bright90->category*[cloud-nodes*]->roles*[pbsproclient*]]% set queues cloudq
[bright90->...*]->roles*[pbsproclient*]->momsettings*]% momsettings

```

```
[bright90->...*]->roles*[pbsproclient*]->momsettings*]% set leafrouters director
[bright90->...*]->roles*[pbsproclient*]->momsettings*]% commit
[bright90->...*]->roles[pbsproclient]->momsettings]% device roles director
[bright90->...*]->roles]% assign pbsproclient
[bright90->...*]->roles*[pbsproclient*]]% momsettings
[bright90->...*]->roles*[pbsproclient*]->momsettings*]% set startmom no
[bright90->...*]->roles*[pbsproclient*]->momsettings*]% ..
[bright90->...*]->roles*[pbsproclient*]]% commsettings
[bright90->...*]->roles*[pbsproclient*]->commsettings*]% set startcomm yes
[bright90->...*]->roles*[pbsproclient*]->commsettings*]% set commrouters master
[bright90->...*]->roles*[pbsproclient*]->commsettings*]% commit
[bright90->...]->roles[pbsproclient]->commsettings]%
```

7.5.4 Installing, Configuring, And Running LSF

IBM prefers to make LSF available directly from their Passport Advantage Online website, which is why it is not available by direct selection in figure 3.10 of the *Installation Manual*.

Installing LSF

The workload manager LSF version 10.1 is installed and integrated into Bright Cluster Manager 9.0 with the following steps:

1. The following LSF files should be downloaded from the IBM web site into a directory on the head node:
 - Installation package: `lsf<lsf_ver>_lsfinstall_linux_<cpu_arch>.tar.Z`
 - Distribution package: `lsf<lsf_ver>_linux<kern_ver>-glibc<glibc_ver>-<cpu_arch>.tar.Z`
 - Documentation package (optional): `lsf<lsf_ver>_documentation.tar.Z`

Here:

- `<lsf_ver>` is the LSF version, for example: 10.1
- `<kern_ver>` is the Linux kernel version, for example: 2.6
- `<glibc_ver>` is the glibc library version, for example: 2.3
- `<cpu_arch>` is the CPU architecture, for example: x86_64

A check should be done to ensure that the `tar.Z` files have not been renamed or had their names changed to lower case during the download, in order to avoid installation issues. All the files must be in the same directory before installation.

In case of an existing failover setup, the installation is done on the active head node.

A license file for LSF may also be needed for the installation, but it does not have to be in the same directory as the other `tar.Z` files.

2. The `cm-lsf` package must be installed on the head node. The `cm-lsf-client` package must be installed on both the head node and within the software images. By default they should have been already installed. If not then they can be installed manually from the Bright Computing repository. For RHEL-based distributions the procedure looks like:

```
[root@bright90 ~]# yum install cm-lsf cm-lsf-client
[root@bright90 ~]# chroot <IMAGE> yum install cm-lsf-client
```

For SLES distributions the procedure looks like:

```
[root@bright90 ~]# zypper install cm-lsf cm-lsf-client
[root@bright90 ~]# chroot <IMAGE> zypper install cm-lsf-client
```

For Ubuntu distributions the procedure looks like:

```
[root@bright90 ~]# apt-get install cm-lsf cm-lsf-client
[root@bright90 ~]# cm-chroot-sw-img <IMAGE>
[root@bright90 ~]# apt-get install cm-lsf-client
...
[root@bright90 ~]# exit
```

The `cm-lsf` and `cm-lsf-client` packages contain a template for an environment module file, an installation configuration file, and systemd unit files. The installation configuration file may be tuned by the administrator if required. It is passed to the `lsfinstall` script distributed with LSF, which is executed by `cm-wlm-setup` during setup. To change the default values in the installation configuration file, the administrator should change the template file:

```
[root@bright90 ~]# cd /cm/shared/apps/lsf/var/cm/
[root@bright90 cm]# vi install.config.template
...
```

Values enclosed by a percentage sign, `'%'`, are replaced by `cm-wlm-setup` during installation. If such values are replaced by custom values, then `cm-wlm-setup` does not change them, and the custom values are used during installation.

If `install.config` is changed instead of the template file, then `cm-wlm-setup` replaces it with the configuration file generated from the template. So, changing `install.config` directly should almost certainly not ever be done.

3. `cm-wlm-setup` is run. The directory where the LSF files were downloaded is specified on one of the setup screens, or with the `--archives-location` option.

Example

```
[root@bright90 ~]# cm-wlm-setup --wlm lsf --setup --archives-location /root/lsf
```

The same can be achieved by executing `cm-wlm-setup` without any command line arguments. The required information can be specified within the Ncurses configuration screens in this case. Also the same can be achieved with the WLM Wizard in BrightView.

4. The nodes are then rebooted, and the LSF command `bhosts` then displays an output similar to:

Example

```
[root@bright90 ~]# module load lsf
[root@bright90 ~]# bhosts
HOST_NAME  STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  ...
bright90   ok      -    2    0      0      0  ...
head2      ok      -    0    0      0      0  ...
node001    ok      -    1    0      0      0  ...
node002    ok      -    1    0      0      0  ...
node003    unavail -    1    0      0      0  ...
node004    closed  -    1    0      0      0  ...
node005    ok      -    1    0      0      0  ...
```

The output in the preceding example has been truncated for this manual, for convenience.

The installation status can be checked with the service `lsfd` (some output elided):

```
[root@bright90 ~]# systemctl status lsfd
lsfd.service - IBM Spectrum LSF
Loaded: loaded (/usr/lib/systemd/system/lsfd.service; enabled; vendor preset: disabled)
Active: active (running) since Mon 2020-03-30 13:59:29 CEST; 27min ago
Tasks: 13
Memory: 107.6M
CGroup: /system.slice/lsfd.service
├─ 2411 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/lim
├─ 3470 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/pim
├─ 3472 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/res
├─ 3478 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/sbatchd
├─ 3483 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/mbatchd -d
    /cm/shared/apps/1...
├─ 3499 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/mbschd
├─ 3527 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/melim
└─ 11580 /cm/shared/apps/lsf/lsf...3-x86_64/10.1/linux2.6-glibc2.3-x86_64/etc/mbatchd -d
    /cm/shared/apps/1...

Mar 30 13:59:21 bright90 systemd[1]: Starting IBM Spectrum LSF...
Mar 30 13:59:22 bright90 lsf_daemons[2274]: Starting the LSF subsystem
Mar 30 13:59:29 bright90 systemd[1]: Started IBM Spectrum LSF.
[root@bright90 ~]#
```

while default queues can be seen by running:

```
[root@bright90 ~]# module load lsf
[root@bright90 ~]# bqueues
QUEUE_NAME      PRIO STATUS      MAX JL/U JL/P ...
owners          43  Open:Active  -   -   - ...
priority        43  Open:Active  -   -   - ...
night           40  Open:Active  -   -   - ...
chkpnt_rerun_qu 40  Open:Active  -   -   - ...
short           35  Open:Active  -   -   - ...
license         33  Open:Active  -   -   - ...
normal          30  Open:Active  -   -   - ...
interactive     30  Open:Active  -   -   - ...
idle            20  Open:Active  -   -   - ...
```

The output in the preceding example has been truncated for this manual, for convenience.

If more than one instance of an LSF cluster is set up, then the full modulefile name should be specified. For example:

```
[root@bright90 ~]# module load lsf/lsf1/10.1
```

Configuring LSF

LSF server configuration: After LSF is set up with `cm-wlm-setup`, the following CMDaemon settings can be modified for the LSF server role:

- **External Server:** a value of `yes` means that LSF server daemons are running on an external server that is not managed by Bright Cluster Manager.

The following settings are available in LSF cluster settings, and are applied to all the LSF roles of the LSF cluster:

- **Prefix:** this sets the path to the root location of the LSF installation. The default value is `/cm/shared/apps/lsf/current`.
- **Var:** this sets the path to the `var` directory of LSF. The default value is `/cm/shared/apps/lsf/var`.
- **Cgroups:** this is a submodule that contains LSF-related cgroups settings.

The cgroups settings that affect LSF behavior are available via the LSF instance. For example, for an instance `lsf`, the settings can be accessed within `cmsh` via `cmsh->wlm[lsf]->cgroups`. The settings available are:

- **Automount:** if yes, then the workload manager tries to mount a subsystem if it is not mounted yet. The default value is `No`.
- **Job Cgroup Template:** this is the template relative job cgroup path. The token `$CLUSTER` specified in this template path is replaced with the actual LSF cluster name, and `$JOBID` is replaced by the job ID. The path is used by the Bright Cluster Manager monitoring system in order collect job metrics from the cgroups. By default, the path is set to `"lsf/$CLUSTER/job.$JOBID.*"`.
- **Process Tracking:** if yes, then processes are tracked based on job control functions such as: termination, suspension, resume, and other signaling. These are used on Linux systems that support the freezer subsystem under cgroups. The parameter sets `LSF_PROCESS_TRACKING` in `lsf.conf`.
- **Linux Cgroup Accounting:** if yes, then LSF tracks processes based on CPU and memory accounting. This is for Linux systems that support the memory and `cpuacct` subsystems under cgroups. Once enabled, this parameter takes effect for new jobs. The parameter sets `LSF_LINUX_CGROUP_ACCT` in `lsf.conf`.

If this parameter and **Process Tracking** are both enabled, then they take precedence over the parameters `LSF_PIM_LINUX_ENHANCE` and `EGO_PIM_SWAP_REPORT` in `lsf.conf`.

- **Mount Point:** specifies a path where cgroups is mounted. It only makes sense to set this when the location is not standard for the operating system.
- **Resource Enforce:** If yes, then resource enforcement is carried out through the Linux memory and `cpuset` subsystems under cgroups. This is for Linux systems with cgroup support. The parameter sets `LSB_RESOURCE_ENFORCE` in `lsf.conf`.

The server role settings can be modified as follows:

- **Within Bright View:** For example for a head node `bright90`, via a clickpath of `Devices→Head Nodes→bright90→Settings→Roles→LSF server role`
- **Within cmsh:** For a particular category in the category mode, or a particular device in the device mode, the `roles` submodule is chosen. Within the `roles` submodule, the `lsfserver` object can be assigned or used. The following example shows how to set the LSF `prefix` parameter for the default category.

Example

```
[root@bright90~]# cmsh
[bright90]% configurationoverlay use lsf-server; roles
[bright90->configurationoverlay[lsf-server]->roles]% use lsfserver
[bright90->...[lsfserver]]% set externalserver yes
[bright90->...[lsfserver*]]% commit
```

The global LSF cluster settings can be modified as follows:

- **Within Bright View:**
For example, for a head node `bright90`, via a clickpath of:
`HPC`→`Wlm Clusters`→`CLUSTER_NAME`→`Settings`
- **Within `cmsh`:**
A particular LSF cluster instance can be chosen within `wlm` mode, to set its global parameters. The following example shows how to set the LSF prefix parameter for the `lsf1` cluster.

Example

```
[root@bright90~]# cmsh
[bright90]% wlm use lsf1
[bright90->wlm[lsf1]]% set prefix /cm/shared/apps/lsf2
[bright90->wlm*[lsf1*]]% commit
[bright90->wlm[lsf1]]% commit
```

LSF client configuration: After installation, the following CMDaemon settings can be specified for the LSF client role:

- **Queues** A restricted list of queues named “`qname1`”, “`qname2`” and so on can be set using a command syntax like this instead:

```
set queues <qname1> [<qname2> ...]
```

Alternatively, these, and other queues can be added or deleted using Bright View (section 7.6.2) or `cmsh` (section 7.7.2).

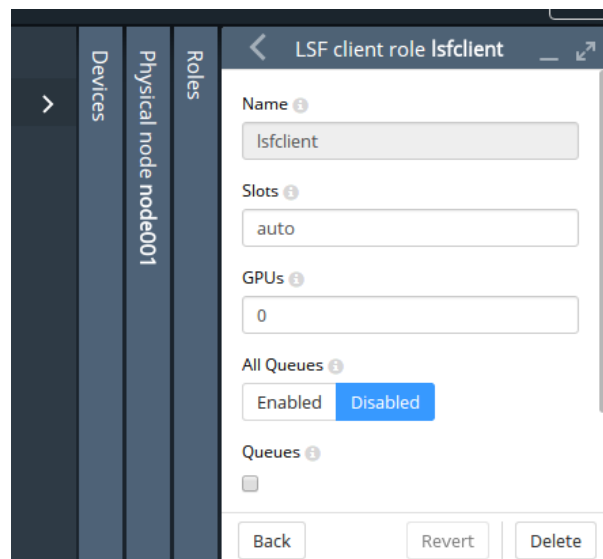


Figure 7.11: Bright View access to LSF configuration options via roles window options

- **Options** The options that can be set for a regular node `node001`, via figure 7.11, reached via a clickpath of
`Devices`→`Nodes [node001]`→`Edit`→`Settings`→`Roles`→`Add`→`LSF client role` are:
 - **Slots:** The number of CPUs per node. By default LSF tries to determine the value automatically. If the number of slots is set to 0, then the node becomes an LSF submit host, so that no jobs can run on the host, but users can submit their jobs from this host using the LSF utilities.

- **GPUs:** The number of GPUs per node
- **Queues:** All queues.
- **Advanced button:** This brings up a dialogue that allows GPU devices to be added.
- **Node Customizations:** LSF node custom properties (section 7.11).

From `cmsh` these properties are accessible from within the appropriate node or category roles submode (section 7.4.2).

Submission hosts: By default, all compute nodes to which the `lsfclient` role is assigned, also become submission hosts. This means that LSF users can submit their jobs from those compute nodes. If the administrator wants to allow users to submit their jobs from a non-compute node—for example from a login node—then the `lsfsubmit` role should be assigned, for example:

Example

```
[root@bright90~]# cmsh
[...->configurationoverlay[lsf1-submit]]% roles
[...->roles]% assign lsfsubmit
[...->roles*[lsfsubmit*]]% append lsfclusters lsf1
[...->roles*[lsfsubmit*]]% commit
[...->roles[lsfsubmit]]%
```

Configuring a node according to the preceding steps then allows users to submit their jobs from that node, without the submitted jobs getting scheduled to run on that node.

If more than one LSF cluster is set up, and they share the same submit host, then a single `lsfsubmit` role can be used. In this case, all the LSF cluster names should be appended to the `lsfcluster` parameter of the role.

Further configuration: For further configuration the *Administering Platform LSF* manual provided with the LSF software should be consulted.

Running LSF

Role assignment and role removal enables and disables LSF from Bright View (sections 7.4.1) or `cmsh` (section 7.4.2).

An active LSF master service (typically, but not necessarily on a head node) has the following LSF-related processes running on it:

Process/Service	Description
<code>res</code>	Remote Execution Server*
<code>sbatchd</code>	client batch job execution daemon*
<code>mbatchd</code>	master batch job execution daemon
<code>eauth</code>	External Authentication method
<code>lim</code>	Load Information Manager*
<code>pim</code>	Process Information Manager*
<code>pem</code>	Process Execution Manager*
<code>vemkd</code>	Platform LSF Kernel Daemon
<code>egosc</code>	Enterprise Grid Orchestrator service controller
<code>mbschd</code>	master batch scheduler daemon

*These services/processes run on compute nodes.

Non-active LSF-masters running as compute nodes run the processes marked with an asterisk only.

LSF daemon logs are kept under `/cm/local/apps/lsf/var/log/`, on each of the nodes where LSF services run.

7.6 Using Bright View With Workload Management

Viewing the workload manager services from Bright View is described in section 7.4.3. The HPC (High Performance Computing) icon, which looks like a speedometer, is the Bright View resource that allows the following items to be accessed:

- WLM clusters: for WLM cluster settings for a WLM cluster instance to be viewed and managed
- jobs: for jobs to be viewed and managed
- queues: for queues to be viewed and managed
- job queue stats: for job queue statistics to be viewed

These items are described next.

7.6.1 Jobs Display And Handling In Bright View


The clickpath `HPC→Jobs` opens a window that displays a list of recent job IDs, along with the scheduler, user, queue, and status of the job (figure 7.12).

The screenshot shows the Bright View interface with the 'Jobs' window open. The window displays a table with columns: JOB ID, USER, STATUS, SCHEDULER, QUEUE, NODES, and OPTIONS. The table lists several jobs with IDs ranging from 1308 to 1316. A context menu is open over the 'OPTIONS' column, showing options: Hold, Release, Suspend, Resume, and Cancel. A 'Manage' button is also visible over the table.

JOB ID	USER	STATUS	SCHEDULER	QUEUE	NODES	OPTIONS
1308.pj-rc7	pbuser	R	pbspro	workq	node001	Show
1309.pj-rc7	pbuser	R	pbspro	workq	node002	Show
1310.pj-rc7	pbuser	Q	pbspro	workq		Show
1311.pj-rc7	pbuser	Q	pbspro	workq		Show
1312.pj-rc7	pbuser	R	pbspro	workq	node001	Show
1313.pj-rc7	pbuser	R	pbspro	workq	node002	Show
1314.pj-rc7	pbuser	Q	pbspro	workq		Show
1315.pj-rc7	pbuser	Q	pbspro	workq		Show
1316.pj-rc7	pbuser	Q	pbspro	workq		Show

Figure 7.12: Workload Manager Jobs

If there are jobs running, then buttons in the `OPTIONS` column allow the jobs to be examined and managed.

- the Show button opens up a window with further details of the selected job.
- A selection button, , allows the selected job to be managed, via a Manage hint, using menu options as follows:
 - The Hold option stops selected queued jobs from being considered for running by putting them in a Hold state.
 - The Release option releases selected queued jobs in the Hold state so that they are considered for running again.
 - The Suspend option suspends selected running jobs.

- The Resume option allows selected suspended jobs to run again.
- The Cancel option removes selected jobs from the queue.

7.6.2 Queues Display And Handling In Bright View

The clickpath HPC→Wlm Clusters→Edit→Job Queues displays a list of queues available (figure 7.13).

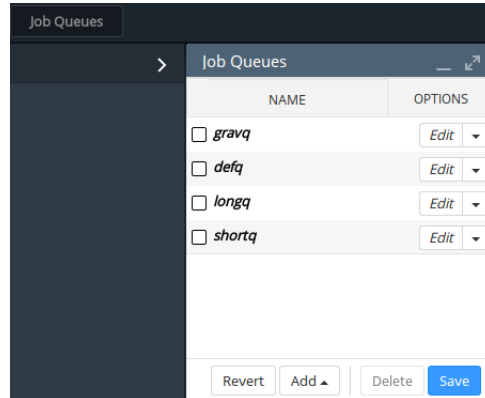


Figure 7.13: Workload Manager Queues

Queues can be added or deleted. An Edit button allows existing queue parameters to be set, for example as in figure 7.14.

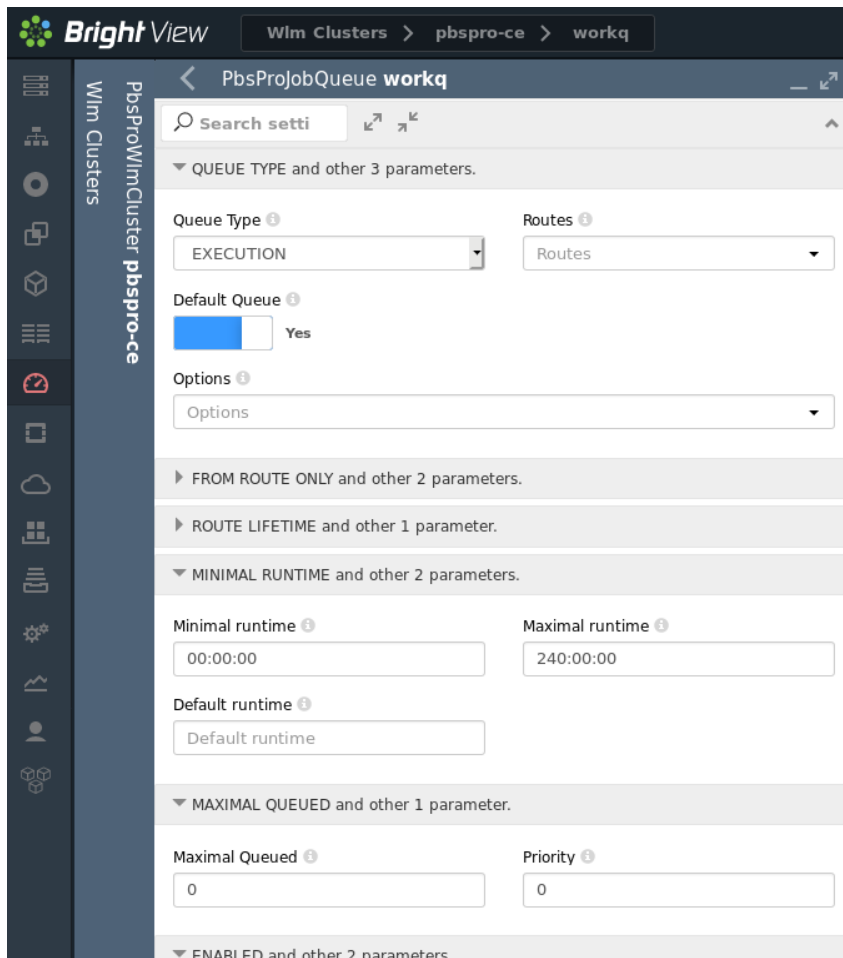


Figure 7.14: Workload Manager Queue Parameters Configuration

7.7 Using `cmsh` With Workload Management

The `wlm` mode in `cmsh` gives the administrator access to the WLM instances running on the cluster.

The `cmsh` tree (Appendix M) has the following submodes structure under the `wlm` mode:

```

`-- wlm
   |-- cgroups
   |-- jobqueue
   |-- jobs
   |-- parallelenvironments
   `-- placeholders
  
```

- `cgroups`: provides ways to configure WLMs via the `cgroup` mechanism of the Linux kernel (section 7.10)
- `jobqueue`: allows WLM job queues to be managed (section 7.7.2)
- `jobs`: allows WLM jobs to be viewed and managed (section 7.7.1)
- `parallelenvironments`: allows MPI job environment parameters to be configured (page 275) for WLMs
- `placeholders`: allows nodes to be configured in advance, for planning resource use in WLMs (section 8.4.7)

For basic WLM management via `cmsh`, the administrator is expected to use `wlm` mode with the `jobs` and `jobqueue` submodes.

If there is just one WLM instance running on the cluster, then an administrator that accesses `wlm` mode drops straight into the object representing that WLM instance, and can then access the submodes.

If there is more than one WLM running, then the administrator must select a WLM instance from the top level `wlm` mode, with the `use` command, in order to access the submodes.

While at the top level `wlm` mode, the instances can be listed. The list shows the nodes used by the instance, and the node roles assigned to the nodes.

Suppose Bright Cluster Manager is being used to manage two WLM instances:

- one Slurm WLM, named `slr`
- one UGE WLM, named `uc`

with their nodes having been allocated roles as follows:

Example

```
[bright90->wlm]% list
```

Type	Name (key)	Server nodes	Submit nodes	Client nodes
Slurm	<code>slr</code>	<code>node001</code>	<code>bright90,node001..node003</code>	<code>node002,node003</code>
UGE	<code>uc</code>	<code>node004</code>	<code>bright90,node004..node006</code>	<code>node005,node006</code>

Then one way to list all jobs running per WLM instance, queue, and user, is with the `filter` command:

```
[bright90->wlm]% filter --running
```

WLM	Job ID	Job name	User	Queue	Submit time	Start time	End time	Nodes	Exit code
<code>slr</code>	103	<code>sjob</code>	<code>maud</code>	<code>defq</code>	15:43:02	15:43:03	N/A	<code>node002,node003</code>	0
<code>slr</code>	104	<code>sjob</code>	<code>maud</code>	<code>defq</code>	15:43:02	15:43:25	N/A	<code>node002,node003</code>	0
<code>slr</code>	105	<code>sjob</code>	<code>maud</code>	<code>defq</code>	15:43:02	15:43:33	N/A	<code>node002,node003</code>	0
<code>slr</code>	106	<code>sjob</code>	<code>maud</code>	<code>defq</code>	15:43:02	15:43:36	N/A	<code>node002,node003</code>	0
<code>slr</code>	171	<code>sjob</code>	<code>maud</code>	<code>defq</code>	14:20:09	14:20:23	N/A	<code>node002,node003</code>	0
<code>uc</code>	200	<code>ugjob</code>	<code>fred</code>	<code>all.q</code>	14:22:18	14:23:20	N/A	<code>node005,node006</code>	0

The `filter` command can also be run within the `jobs` mode, for a particular WLM instance, and is described in more detail on page 295.

An alternate way to list all jobs from `wlm` mode is with a `foreach` command to descend into `jobs` submode to list the jobs for each WLM instance:

Example

```
[bright90->wlm]% foreach * (jobs ; list)
```

Type	Job ID	User	Queue	Running time	Status	Nodes
Slurm	55	<code>maud</code>	<code>defq</code>	18m 10s	COMPLETED	<code>node002,node003</code>
Slurm	56	<code>maud</code>	<code>defq</code>	18m 5s	COMPLETED	<code>node002,node003</code>
Slurm	57	<code>maud</code>	<code>defq</code>	2m 11s	RUNNING	<code>node002,node003</code>
Slurm	58	<code>maud</code>	<code>defq</code>	0s	PENDING	

Type	Job ID	User	Queue	Running time	Status	Nodes
UGE	96	<code>maud</code>	<code>all.q</code>	2m 30s	<code>r</code>	<code>node005,node006</code>
UGE	97	<code>maud</code>		0s	<code>qw</code>	
UGE	98	<code>maud</code>		0s	<code>qw</code>	

The `jobs` submode is now discussed further.

7.7.1 The jobs Submode In cmsh

Within the jobs submode of a WLM instance, the administrator can list jobs that are running or queued up for that particular instance. For example, the running and queued jobs listed for a PBS instance may be displayed as:

Example

```
[bright90->wlm[pbspro-ce]->jobs]% list
Type      Job ID      User      Queue      Running time Status  Nodes
-----
PBSPro    3117.bright90 pbuser1   hydroq     1s        R      node002
PBSPro    3118.bright90 pbuser2   workq      1s        R      node001
PBSPro    3119.bright90 pbuser1   hydroq     0s        Q
PBSPro    3120.bright90 pbuser3   hydroq     0s        Q
PBSPro    3121.bright90 pbuser1   hydroq     0s        Q
PBSPro    3122.bright90 pbuser1   hydroq     0s        Q
```

In the foreach jobs listings for the Slurm and UGE WLM instances shown earlier, the jobs are in a running, queued up, or completed state. They are also all being run by a user maud, who happens to be making use of both workload managers.

For Slurm, completed jobs are shown for a short time with the status COMPLETED. For UGE, completed jobs are shown for an even shorter time with a blank for their status.

Commands That Change The Status Of A Queued Or Running Job

Within a jobs submode, the following commands, used with a job ID, can change the status of a particular queued or running job:

- **hold**: puts a queued job into a hold state. This prevents the job from being considered for running.
- **release**: releases a job from a hold state, putting it back in the queue, so that it can be considered for running.
- **suspend**: pauses a running job
- **resume**: resumes a suspended job
- **remove**: removes a job

For example, continuing the case earlier on of the cmsh session with the foreach listing (page 293):

The administrator can suspend the running job with job ID 57 on the Slurm instance as follows:

Example

```
[bright90->wlm]% jobs slr; suspend 57
[bright90->wlm[slr]->jobs]% list | head -2; list | grep 57
Type Job ID User Queue Running time Status  Nodes
-----
Slurm 57    maud defq  2m 51s      SUSPENDED  node002,node003
```

and can hold the queued waiting job with job ID 98 on the UGE instance as follows:

Example

```
[bright90->wlm]% jobs uc; hold 98
[bright90->wlm[uc]->jobs]% list | head -2; list | grep 98
Type Job ID User Queue Running time Status  Nodes
-----
UGE 98    maud 0s        hqw
```


The administrator can then resume the suspended job ID 57, and can release the held job ID 98 as follows:

Example

```
[bright90->wlm[uc]->jobs]% wlm; jobs uc; release 98
[bright90->wlm[uc]->jobs]% wlm; jobs slr; resume 57
```

Commands To Inspect Jobs:

Within the jobs submode, the following commands can be used to inspect a job:

- `show <job ID>`: properties are shown for a running or pending job.
- `info <job ID>`: information is shown about a historic job.
- `statistics`: statistics are shown for jobs. More detail on the `statistics` command is given on page 296.
- `list`: lists pending and running jobs. Recently completed jobs are also displayed.
- `dumpmonitoringdata [OPTIONS] [<start-time> <end-time>] <metric> <job id>`: Displays the monitoring data for a specific metric or healthcheck for a running or completed job. Example: `dumpmonitoringdata CtxtSwitches 170`

More detail is given on the `dumpmonitoringdata` command in section 12.6.4.

- `latestmonitoringdata [OPTIONS] <job ID>`: Displays the last measured job monitoring data for the job.
 - `latesthealthdata <job ID>`: The subset of health checks within the last measured monitoring data.
 - `latestmetricdata <job ID>`: The subset of metrics within the last measured monitoring data.

More detail is given on the `latest*data` commands in section 12.6.3.

- `measurables <job ID>`: The measurables used for job monitoring data.
 - `healthchecks <job ID>`: The subset of health checks within the measurables used for job monitoring data.
 - `metrics <job ID>`: The subset of metrics within the measurables used for job monitoring data.
 - `enumetrics <job ID>`: The subset of enumetrics within the measurables used for job monitoring data.

The `filter` Command Within jobs Submode

If run at the top-level `wlm` mode, then the `filter` command lists jobs for all WLM instances.

It can also be run at the submode-level jobs mode, for a particular WLM instance, in which case it lists only the jobs for that instance (section 12.8.4).

The `filter` command without any options is a bit like an extension of the `list` command in that it lists currently pending and running jobs, although in a different format. However, in addition, it also lists past tasks, with their start and end times.

An example to illustrate the output format for a Slurm instance, somewhat simplified for clarity, is:

Example

```
[bright90->wlm[myslurminstance]]% filter
Job ID Job name      User  Queue Submit time Start time End time Nodes      Exit code
-----
 2     hello         fred  defq  15:00:51  15:00:51  15:00:52 node001      0
 3     mpirun         fred  defq  15:03:17  15:03:17  15:03:18 node001      0
...
25     slurmhello.sh    fred  defq  16:17:30  16:17:31  16:17:33 node001..node004 0
26     slurmhello.sh    fred  defq  16:18:39  16:18:40  16:19:02 node001..node004 0
27     slurmhello.sh    fred  defq  16:18:41  16:19:03  16:19:25 node001..node004 0
28     slurmhello.sh    fred  defq  16:18:41  N/A       N/A       node001..node004 0
29     slurmhello.sh    fred  defq  16:18:42  N/A       N/A       node001..node004 0
```

In the preceding example, the start and end times for jobs 28 and 29 are not yet available because the jobs are still pending.

Because all jobs for that WLM instance—historic and present—are displayed by the `filter` command, it means that finding a particular job in the output displayed can be hard. The displayed output can therefore be filtered further.

The following options to the `filter` command are therefore available in order to find particular groupings of jobs:

- `-w|--wlm`: by WLM instance
- `-u|--user`: by user
- `-g|--group`: by group
- `--ended`: by jobs that ended successfully, with a zero exit status
- `--running`: by jobs that are running
- `--pending`: by jobs that are pending
- `--failed`: by jobs that finished unsuccessfully, with a non-zero exit status

It may take about a minute for `CMDaemon` to become aware of job data. This means that, for example, a job submitted 10s ago may not show up in the output.

Further options to `filter` command can be seen by running the command `help filter` within the `wlm` or `jobs` modes.

The statistics Command

The `statistics` command without any options displays an overview of states for all past and present jobs:

Example

```
[bright90->wlm[uc]]% statistics
Pending  Running  Finished  Error    Nodes
-----
 4         0       154       5       309
```

The options available include the following:

- `-w|--wlm <WLM instance>`: by WLM instance
- `-u|--user`: by user
- `-g|--group`: by group
- `-a|--account`: by account

- `-p|--parentid`: by job parent ID
- `--hour` : by hour
- `--day`: by day
- `--week`: by week
- `--interval`: by interval

The jobs statistics can be split across users. An example of the output format in this case, somewhat simplified for clarity, is:

Example

```
[bright90->wlm[slurm]->jobs]% statistics --user
User      Pending  Running  Finished  Error    Nodes
-----
alice     0         0         2         0        1
bob       0         0         5         0        4
charline  0         1         2         0        2
dennis   0         0         6         0        4
eve       0         0         4         0        1
frank    0         0         1         0        1
```

The job statistics can be displayed over various time periods, if there are jobs within the period. An example of the output format for an interval of 60s is:

Example

```
[bright90->wlm[slurm]->jobs]% statistics --interval 60
Date      Pending  Running  Finished  Error    Nodes
-----
2020/05/13 15:00:00 0         0         1         0        1
2020/05/13 15:03:00 0         0         7         0        7
2020/05/13 15:04:00 0         0         1         1        2
2020/05/13 15:06:00 0         0         3         0        7
2020/05/13 15:07:00 0         0         1         0        0
2020/05/13 15:08:00 0         0         6         0        8
2020/05/13 16:17:00 0         0         1         0        4
2020/05/13 16:19:00 0         0         9         0       36
2020/05/13 16:20:00 0         0         6         0       24
2020/05/13 16:21:00 0         0         3         0       12
```

Job Directives

Job directives configure some of the ways in which CMDaemon manages job information processing.

The administrator can configure the following directives:

- `JobInformationKeepCount`: The maximal number of jobs that are kept in the cache, default 8192, maximal value 100 million (page 728).
- `JobInformationKeepDuration`: How long to keep jobs in the CMDaemon database, default 28 days (page 728).
- `JobInformationMinimalJobDuration`: Minimal duration for jobs to place them in the cache, default 0s (page 728).
- `JobInformationFlushInterval`: Over what time period to flush the cache to storage (page 728).
- `JobInformationDisabled`: Disables job information processing (page 727).

7.7.2 Job Queue Display And Handling In `cmsh: jobqueue Mode`

The `jobqueue` submode under the top level `wlm` mode can be used to manage queues for a WLM instance. Queue properties can be viewed and set.

From the top level `wlm` mode, a list of queues can be seen for each WLM instance by descending into the WLM instance with a `foreach`:

Example

```
[bright90->wlm]% foreach * (get name; jobqueue; list)
slr
Name (key)  Nodes
-----
defq       node002,node003
uc
Name (key)  Nodes
-----
all.q      node005,node006
[bright90->wlm]%
```

The usual object manipulation commands (`show`, `get`, `set` and others) of section 2.5.3 work in the `jobqueue` submode.

Job Queue Parameters For UGE

For example, for a UGE instance, the `show` command might display the following properties:

Example

```
[bright90->wlm[uc]->jobqueue]% show all.q
Parameter                               Value
-----
Epilog                                   NONE
Maximal runtime                          INFINITY
Minimal runtime                          INFINITY
Name                                       all.q
Nodes                                     node005,node006
Options
Parallel environments                    openmpi,mpich2
Prolog                                    NONE
Revision
Slots                                     0
Temp directory                           /tmp
Type                                       UGE
WlmCluster                               uc
Resume method                             NONE
Starter method                           NONE
Suspend method                           NONE
Terminate method                         NONE
```

Job Queue Parameters For Slurm

Similarly, for a Slurm instance, the `show` command might display the following properties:

```
[bright90->wlm[slr]->jobqueue]% show defq
Parameter                               Value
-----
Allow QOS                                 ALL
Allow accounts                            ALL
```

```

Allow groups          ALL
Alternate
Default memory per CPU      UNLIMITED
Default memory per Node    UNLIMITED
Default time              NONE
Deny Accounts
Deny Groups
Deny QOS
Disable root            no
Grace time              0
Hidden                 no
LLN                   no
Max CPUs per node       UNLIMITED
Max memory per CPU      UNLIMITED
Max memory per Node    UNLIMITED
Max nodes              UNLIMITED
Max time              UNLIMITED
Min nodes              1
Name                  defq
Nodes                node002,node003
Options
OverSubscribe         NO
Preemption mode      OFF
Priority Job Factor   1
Priority Tier        1
Require reservation  NO
Revision
Root only            no
Select Type Parameters
Status
TRES Billing Weights
Total cpus          0
Total nodes        0
Type               Slurm
WlmCluster         slr
default           yes

```

Job Queue Parameters For PBS

Likewise, for a PBS instance, the show command might display the following properties:

```

[bright90->wlm[pb]->jobqueue]% show workq
Parameter          Value
-----
ACL host enable    no
Default Queue     yes
Default runtime
Enabled           yes
From Route Only   no
Maximal Queued    0
Maximal runtime   240:00:00
Minimal runtime   00:00:00
Name              workq
Nodes            node008,node009
Options
Priority          0
Queue Type       EXECUTION

```

```

Revision
Route Held Jobs          no
Route Lifetime           0
Route Retry Time         0
Route Waiting Jobs       no
Routes
Started                  yes
Type                     PBSPro
WlmCluster                pb

```

7.7.3 Nodes Drainage Status And Handling In `cmsh`

Running the device mode command `drainstatus` displays if a specified node is in a Drained state or not. In a Drained state jobs are not allowed to start running on that node.

Running the device mode command `drain` puts a specified node in a Drained state:

Example

```

[root@bright90 ~]# cmsh
[bright90]% device
[bright90->device]% drainstatus
Node                Queue                Status
-----
node001             PbsPro/pb:workq
node002             UGE/uc/all.q
[bright90->device]% drain node001
Node                Queue                Status
-----
node001             PbsPro/pb:workq     Drained

```

The `undrain` command unsets the Drained state so that jobs may start running on the node again.

The `drain`, `undrain`, and `drainstatus` commands have the same grouping options. The grouping options can make the command apply to not just one node, but to a list of nodes, a group of nodes, a category of nodes, a rack, a chassis, an overlay, a role, or a status. Continuing the example:

```

[bright90->device]% drain -c default; !# for a category of nodes
Node                Queue                Status
-----
node001             PbsPro/pb:workq     Drained
node002             UGE/uc/all.q        Drained

```

The help text for each command indicates the syntax:

Example

```

[root@bright90 ~]# cmsh -c "device help drain"
Name:  drain - Drain jobs (not data) on a set of nodes

Usage:  drain [OPTIONS/node]

Options:  -n, --nodes <node>
          List of nodes, e.g.
          node001..node015,node020..node028,node030 or
          ~/some/file/containing/hostnames

          -g, --group <group>
          Include all nodes that belong to the node group, e.g.

```

```
testnodes or test01,test03

-c, --category <category>
    Include all nodes that belong to the category, e.g. default
    or default,gpu

-r, --rack <rack>
    Include all nodes that are located in the given rack, e.g
    rack01 or rack01..rack04

-h, --chassis <chassis>
    Include all nodes that are located in the given chassis, e.g
    chassis01 or chassis03..chassis05

-e, --overlay <overlay>
    Include all nodes that are part of the given overlay, e.g
    overlay1 or overlayA,overlayC

-m, --image <image>
    Include all nodes that have the given image, e.g default-image
    or default-image,gpu-image

-t, --type <type>
    Type of devices, e.g node or virtualnode,cloudnode

-i, --intersection
    Calculate the intersection of the above selections

-u, --union
    Calculate the union of the above selections

-l, --role <role>
    Filter all nodes that have the given
    role

-s, --status <status>
    Only run command on nodes with specified status, e.g. UP,
    "CLOSED|DOWN", "INST.*"

--setactions <actions>
    set drain actions, actions already set will be removed
    (comma-separated)

--appendactions <actions>
    append drain actions to already existing drain actions
    (comma-separated)

--removeactions <actions>
    remove drain actions

--listactions
    list all drain actions

--clearactions
    remove all drain actions
```

Examples: drain	Drain the current node
drain node001	Drain node001
drain -r rack01	Drain all nodes in rack01
drain --setactions reboot	Drain the current node, and reboot when all jobs are completed

A useful one-liner to reboot and reprovision a sequence of nodes could be in the following format:

Example

```
cmsh -c 'device; drain -n <nodes> --setactions powerreset ; drain -n <nodes> --appendactions undrain'
```

This drains the nodes and does a power reset action, which provisions the nodes. After the nodes are up again, they are undrained so that they are ready to accept jobs again. The command allows the sequence of nodes to be rebooted, for example for the purpose of upgrading a kernel, without needing to schedule a downtime for the cluster.

The cluster administrator should be aware that that using the preceding one-liner is not a universal solution. For example, it is not a good solution if the drain takes a long time, because it means that nodes that drain early could be idle for a long time.

7.8 Examples Of Workload Management Assignment

7.8.1 Setting Up A New Category And A New Queue For It

Suppose a new node with processor optimized to handle Shor's algorithm is added to a cluster that originally has no such nodes. This merits a new category, *shornodes*, so that administrators can configure more such new nodes efficiently. It also merits a new queue, *shorq*, so that users are aware that they can submit suitably-optimized jobs to this category.

Ways to configure the cluster manager for this are described next.

A New Category And A New Queue With Bright View For An Existing Workload Manager Cluster

Queues can be added to a WLM cluster by running the WLM wizard (clickpath HPC→Wlm Wizard) if the WLM cluster does not yet exist. The wizard provides an option during WLM cluster creation for adding queues.

To create a new queue in an existing WLM cluster, the clickpath HPC→Wlm Clusters<WLM cluster name>Job queues→<queue name>→Add is followed. The properties of the new queue can then be set—at least the queue name should be set—in the dialog window of the queue (figure 7.15).

The screenshot displays the 'Job Queues' management interface. On the left, a table lists existing queues, with 'shorq' selected. The main area shows the configuration form for 'shorq' with the following fields and controls:

- Name:** shorq
- Queue type:** Execution
- Routes:** Routes
- Minimal runtime:** 00:00:00
- Maximal runtime:** 240:00:00
- Default runtime:** Default runtime
- Maximal queueable:** 0
- Priority:** 0
- Started:** Enabled (selected), Disabled
- Enabled:** Enabled (selected), Disabled
- ACL host enable:** Enabled (selected), Disabled

At the bottom, there are navigation buttons: 'Revert', 'Add', 'Delete', 'Save', 'Back', 'Revert', 'Delete', and 'Save'.

Figure 7.15: Adding A New Queue Via Bright View

The configuration for the queue properties is saved, and then the job queue scheduler configuration is saved.

Next, a new category can be added via the clickpath `Grouping→Node categories→Add`. Parameters in the new category can be set—at least the category name should be set—to suit the new machines (figure 7.16).

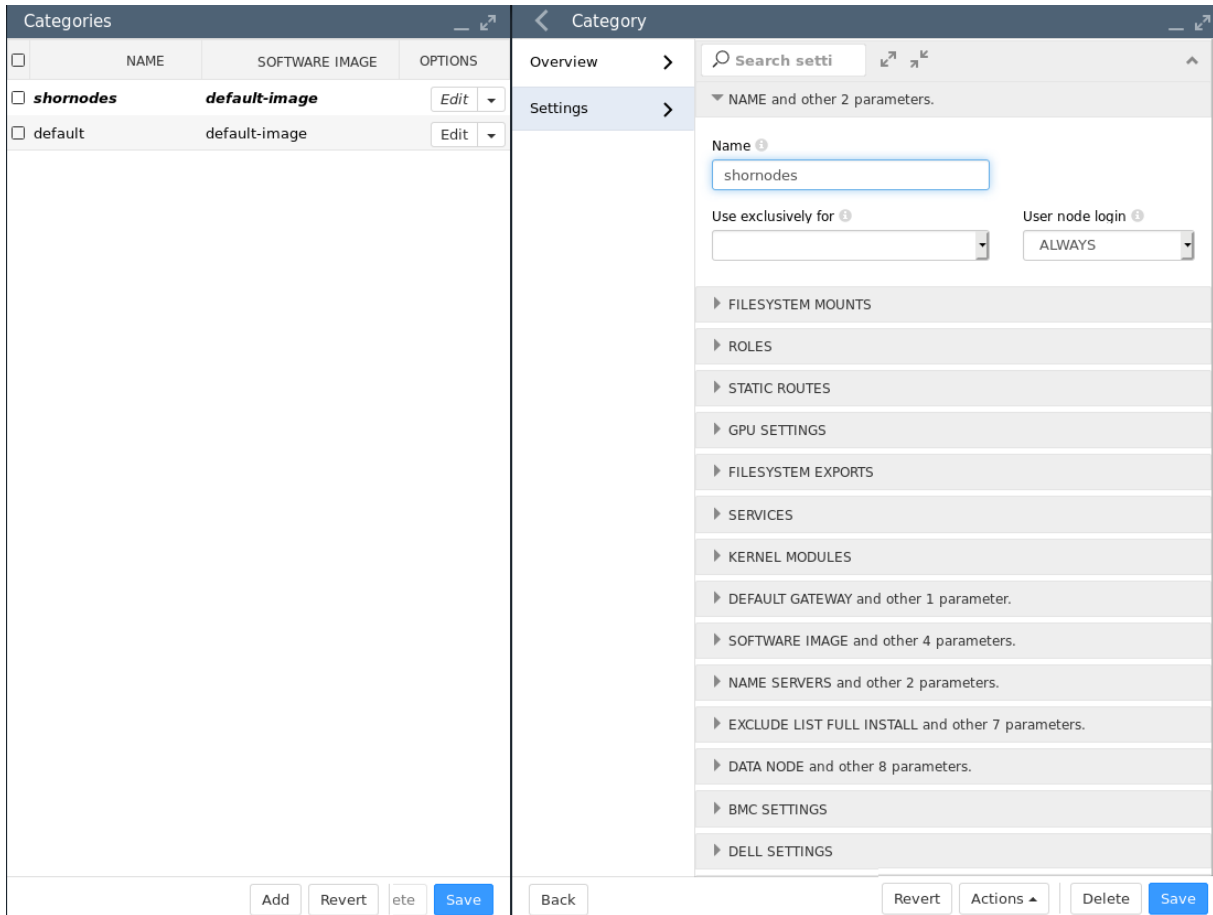


Figure 7.16: Adding A New Category Via Bright View

The name shornodes can therefore be set here.

Another option within the category is to set the queue. The queue, shorq, is therefore set here for this new category. Setting a queue for the category means configuring the options of the queue scheduler role, for example `PBS Pro client role`, for the category. Continuing from within the `Node categories options` window of figure 7.16, the relative clickpath to set a queue is `ROLES`→`Roles`→`Add`→`PBS pro client role`→`Edit`→`Queues`. The appropriate queue can be selected from the queue scheduler menu option (figure 7.17).

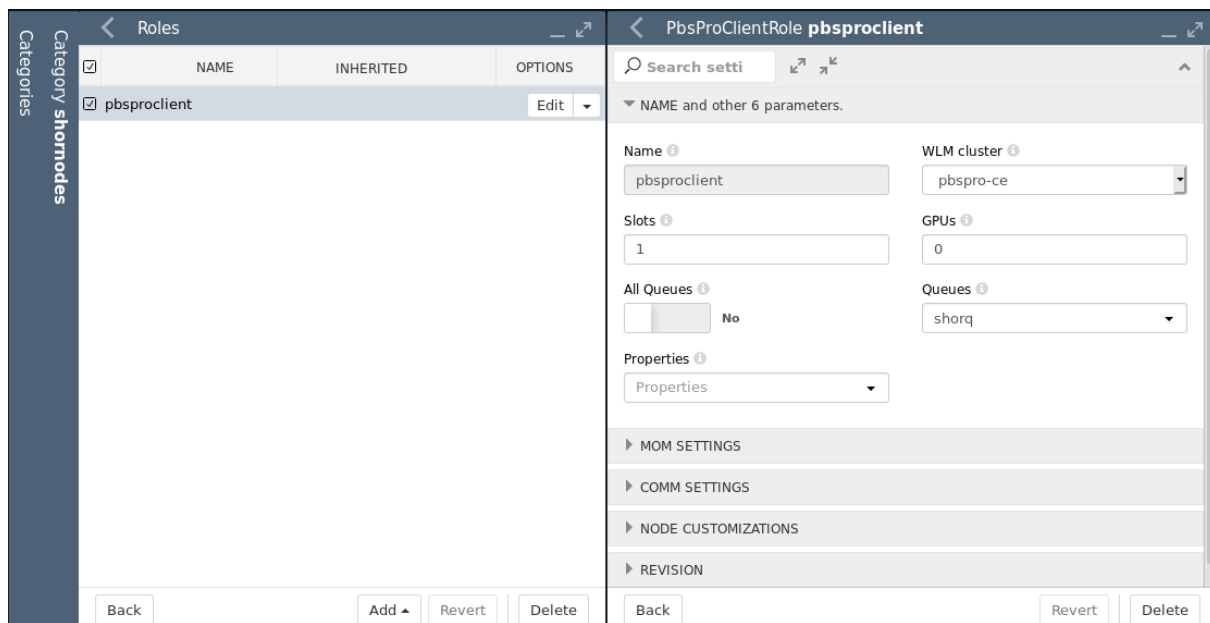


Figure 7.17: Setting A Queue For A New Category Via Bright View

In this case, the `shorq` created earlier on in figure 7.15 is presented for selection. After selection, the configuration settings can then be saved—for this example it means going back up through the levels and clicking on the Save button at the node category settings level (the right hand window shown in figure 7.16), and also going up one more level to click the Save button on the node category list window (the left hand window shown in figure 7.16).

Nodes that are to use the queue should be members of the `shornodes` category. The final step is then to allocate such nodes to the category. This can be done, for example for a `node001`, by going into the settings of the node, via the clickpath `Devices→Nodes[node001]→Edit→Settings→Category` and setting the category to `shornodes`.

A New Category And A New Queue With `cmsh`

The preceding example can also be configured in `cmsh` as follows:

The new queue can be added from within `jobqueue` mode, for the workload manager. For example, if Slurm is the WLM cluster that is enabled:

```
[bright90]% wlm; use slurm; jobqueue; add shorq
[bright90->wlm[slurm]->jobqueue*[shorq*]]% commit
```

The new category, `shornodes`, can be created by cloning an old category, such as `default`:

```
[bright90->wlm[slurm]->jobqueue[shorq]]% category
[bright90->category]% clone default shornodes
[bright90->category*[shornodes*]]% commit
```

Then, going into the roles submodule, appropriate workload manager roles can be assigned, and appropriate queues can be appended and committed for that category:

```
[bright90->category[shornodes]]% roles
[bright90->category[shornodes]->roles]% assign slurmclient; commit
[bright90->category[shornodes]->roles[slurmclient]]% append queues shorq
[bright90->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:

```
[bright90->category[shornodes]->roles]% device use node002
[bright90->device[node002]]% set category shornodes
[bright90->device*[node002*]]% commit
```

7.8.2 Setting Up A Prejob Check

How It Works

Measurables such as health checks (section 12.2.4) by default run as scheduled tasks over regular intervals. They can optionally be configured to run on demand instead, or as prejob checks, that is, before a job is run.

If a health check is configured with the prejob setting, then its response means the same as that of a health check, that is:

- If the response to a prejob health check is PASS, then it shows that the node is displaying healthy behavior for that particular health check.
- If the response to a prejob health check is FAIL, then it implies that the node is unhealthy, at least for that particular health check.

The reason a prejob health check is treated more carefully is that, if the node it has just run on is unhealthy, then it means that a job submitted to the node may fail, may not be able to start, or may even vanish outright. The way it can vanish in some cases, without any information beyond the job submission “event horizon” leads to this behavior sometimes being called the *Black Hole Node Syndrome*.

It can be troublesome for a system administrator to pinpoint the reason for such job failures, since a node may only fail under certain conditions that are hard to reproduce later on. It is therefore a good policy to disallow passing a job to a node which has just been flagged as unhealthy by a health check. Thus, a sensible action (section 12.2.6) taken by a prejob health check on receiving a FAIL response would be to put the node in a Drained state (section 7.7.3). The drain state means that Bright Cluster Manager arranges a rescheduling of the job so that the job runs only on nodes that are believed to be healthy.

A node that has been put in a Drained state with a health check is not automatically undrained. The administrator must clear such a state manually.

The failedprejob health check (page 787) is enabled by default, and logs any prejob health check passes and failures. By default there are no prejob health checks configured, so by default this health check should always pass.

Configuration Using Bright View

To configure the monitoring of nodes with a prejob check in Bright View, the clickpath:

Monitoring Configuration → Data Producers → Monitoring data producers → Edit → When ↓ Pre job is used to set the When value for when it runs. In this case, When is set to Pre job.

Configuration Using cmsh

Within cmsh, the equivalent can be carried out as follows:

Example

For example, for the ib health check:

```
[bright90->monitoring->setup]% get ib when
Timed
[bright90->monitoring->setup]% set ib when <TAB><TAB>
on\ demand pre\ job timed
[bright90->monitoring->setup]% set ib when pre\ job
[bright90->monitoring->setup*]% commit
```

Configuration Outside Of CMDaemon

For more unusual prejob checking requirements, further details on how prologs are configured are given in section 7.3.3.

7.9 Power Saving With `cm-scale`

The `cm-scale` service can be used by an administrator to reduce the energy and storage costs of compute nodes by changing their power state, or their existence state, according to workload demands. That is, `cm-scale` automatically scales a cluster up or down, on-demand, by powering up physical nodes, cloud nodes, or virtual nodes. The scaling is carried out according to settings in the `ScaleServer` role which set the nodes that are to use the `cm-scale` service.

The `cm-scale` service is covered extensively in Chapter 8.

7.10 Cgroups

Linux system processes and all their future children can be aggregated into sets. These sets can be made into hierarchical groups with specialized behavior using the Control Groups (cgroups) mechanism. The 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 reduce swap usage by processes that belong to the group, and so on.

Details about Linux cgroups and their subsystems can be found at <https://www.kernel.org/doc/Documentation/cgroups-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 cgroup support:

1. `/cm/shared/apps/slurm/var/etc/<cluster name>/cgroup.conf`: defines parameters used by Slurm's Linux cgroup-related plugins. The file contains a section that is autogenerated by CMDaemon, with cgroups-related parameters defined in the `SlurmServer` role.

2. `/etc/slurm/cgroup_allowed_devices_file.conf`: declares devices that need to be allowed by default for all jobs. The syntax of this file accepts one device per line. It permits wildcard definitions such as `/dev/sda*` or `/dev/cpu/*/*`. The path to this file can be changed in `cmsh` by setting the value of the `cgroup` parameter `Allowed Devices File` within the `cgroups` submode of the WLM mode for the Slurm instance.

Example

```
[root@bright90 ~]# cmsh
[bright90]% wlm use slurm
[bright90->wlm[slurm]]% cgroups; get alloweddevicesfile
/etc/slurm/cgroup_allowed_devices_file.conf
```

For Slurm, the administrator can set `cgroups` parameters using `cmsh` by going into the `cgroups` submode of the WLM instance.

Parameters that can be managed include:

Parameter	Description	Configuration Parameter In <code>cgroup.conf</code>
Auto Mount*	Force Slurm to mount <code>cgroup</code> subsystems if they are not mounted yet	<code>CgroupAutomount</code>
Mount Point	Where <code>cgroup</code> root is mounted	<code>CgroupMountpoint</code>
Task Affinity*	Set a default task affinity to bind each step task to a subset of the allocated cores using <code>sched_setaffinity</code>	<code>TaskAffinity</code>
Release Agent Dir	Directory containing Slurm <code>cgroup</code> release_agent files	<code>CgroupReleaseAgentDir</code>
Allowed Devices File	File used to declare the default devices for all the jobs, if <code>ConstrainDevices</code> is true	<code>AllowedDevicesFile</code>
Constrain Cores*	Constrain allowed cores to the subset of allocated resources	<code>ConstrainCores</code>

...continues

...continued

Parameter	Description	Configuration Parameter In <code>cgroup.conf</code>
Constrain RAM Space*	Constrain the job's RAM usage	ConstrainRAMSpace
Constrain Swap Space*	Constrain the job's swap space usage	ConstrainSwapSpace
Constrain Devices*	Constrain the job's allowed devices based on GRES allocated resources	ConstrainDevices
Allowed RAM Space	Percentage memory (default is 100%) out of the allocated RAM allowed for the job cgroup RAM. If this percentage is exceeded, then the job steps will be killed and a warning message will be written to standard error.	AllowedRAMSpace
Allowed Swap Space	Percent allocated memory allowed for the job cgroup swap space	AllowedSwapSpace
Max RAM Percent	Maximum percent of total RAM for a job	MaxRAMPercent
Max Swap Percent	Maximum percent of total RAM for the amount of RAM+Swap that may be used for a job	MaxSwapPercent
Min RAM Space	Minimum MB for the memory limits defined by Allowed RAM Space and Allowed Swap Space	MinRAMSpace

* Boolean (takes yes or no as a value)

The options are always written in the `cgroup.conf` file. More details on these options can be found in the man page `man(5) cgroup.conf`.

UGE

Univa Grid Engine allows cgroups settings to be defined at two levels.

- Globally: The parameters can be set globally in the global configuration. 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:

Parameter	Description	Configuration Parameter In qconf
Auto Mount*	Force UGE to mount a cgroup subsystems if they are not mounted yet	mount
Cpuset	If true, then core binding is done by the cgroup cpuset subsystem	cpuset
Freezer	If true, then it enables the cgroup freezer subsystem for job suspension and resumption	freezer
Freeze PE Tasks	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	freeze_pe_tasks
Killing	If true then UGE signals all processes forked/started by the job until all of them are killed	killing
Forced NUMA	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 <code>-mbind cores:strict</code>	forced_numa

...continues

...continued

Parameter	Description	Configuration Parameter In qconf
Virtual Memory Limit	Specifies if virtual memory can be limited with cgroups	h_vmem_limit
Memory Free Hard	If true then kernel ensures that the job does not use more main memory than required	h_mem_free_hard
Memory Free Soft	If true, and the hard memory limit is turned off, then the requested memory with m_mem_free is a soft limit	h_mem_free_soft
Min Memory Limit	A host based minimum memory limit, in bytes or values like 10M, 1G	min_memory_limit
Devices	Controls which devices the job can access. It permits wildcard definitions such as /dev/sda* or /dev/cpu/*/*	devices
Options	Additional cgroups parameters for UGE	

* Boolean (takes yes or no as a value)

The options are updated in UGE only when cgroups are enabled in the UGEServer or UGEClient roles. To enable them, enabled must be set to true within the cgroups submode of the role. By default only global cgroups settings are enabled in UGE.

PBS

PBS Professional supports cgroups through a special Python hook that is installed by `cm-wlm-setup` by default. The hook name is `pbs_cgroups`.

LSF

LSF allows resource enforcement to be controlled with the Linux cgroup memory and cpuset subsystems. By default, when LSF is set up with `cm-wlm-setup`, then both subsystems are enabled for LSF jobs. If job processes on a host use more memory than the defined limit, then the job is immediately killed by the Linux cgroup memory subsystem. The cgroups-related configuration options are available in `cmsh` or Bright View, and can be found in the cgroups submode of the LSF cluster settings:

Parameter	Description	Configuration Parameter In <code>lsf.conf</code>
Resource Enforce	Controls resource enforcement through the Linux cgroups memory and cpuset subsystem, on Linux systems with cgroups support. The resource can be either memory or cpu, or both cpu and memory, in either order (default: memory cpu)	LSB_RESOURCE_ENFORCE
Process Tracking*	This parameter, when enabled, has LSF track processes based on job control functions such as termination, suspension, resume, and other signals, on Linux systems which support the cgroups freezer subsystem	LSF_PROCESS_TRACKING
Linux Cgroup Accounting*	When enabled, processes are tracked based on CPU and memory accounting for Linux systems that support cgroup's memory and cpuacct subsystems	LSF_LINUX_CGROUP_ACCT

* Boolean (takes yes or no as a value)

7.11 Custom Node Parameters

Bright Cluster Manager 9.0 allows the administrator to specify the most important node parameters when a node is configured in a workload manager. But sometimes the workload manager allows the configuration of other advanced or user-defined settings for the nodes. When an administrator clones a node in Bright Cluster Manager, or creates it from scratch, then those advanced settings may need to be configured manually per node in the workload manager. Sometimes the nodes are created or cloned automatically, for example `cm-scale` (Chapter 8), in which case node parameters customization is not possible without manual intervention in the workload manager configuration.

Since Bright Cluster Manager version 8.2, the administrator can configure the advanced node settings in the node customizations mode of `cmsh` or Bright View. The customizations are available in the workload manager client roles, so they can be applied at the node, node category or configuration overlay levels.

In order to configure a custom node parameter for the workload manager, the administrator adds the node customization entry and sets its value. The entry is enabled by default, using the parameter `enabled`. Setting the value to `disabled` automatically removes the parameters from the node settings of the workload manager.

For example, in order to add a new customization entry for PBS, the following `cmsh` commands can be used:

Example

```
[bright90]% category use default
[bright90->category[default]]% use pbsproclient
...->roles[pbsproclient]]% nodecustomizations
...->roles[pbsproclient]->nodecustomizations)% add resv_enable
...->roles*[pbsproclient*]->nodecustomizations*[resv_enable*]]% set value False
...->roles*[pbsproclient*]->nodecustomizations*[resv_enable*]]% show
Parameter                Value
-----
```

```

Key                recv_enable
Value              False
Enabled            yes
Notes              <0 bytes>
...->roles*[pbsproclient*]->nodecustomizations*[resv_enable*]]% commit
...->roles[pbsproclient]->nodecustomizations[resv_enable]]%

```

Different workload managers allow different kinds of node settings to be set. For example, for the Grid Engines, the settings are the so-called complex attributes, as defined by the administrator. But for PBS, the node settings are built-in, and so the administrator can only change their values but not their names. So when the administrator configures node customization entries, Bright Cluster Manager configures them differently depending on the workload manager that is used.

Thus if the administrator creates a new customization entry with key name <KEY> and value <VALUE>, then this is applied to the workload managers as follows:

- Slurm. The node customization entry is appended to the NodeName line of the particular nodes in `slurm.conf` in the form `<KEY>=<VALUE>`. If the entry is removed or disabled in Bright Cluster Manager configuration, then the entry `<KEY>=<VALUE>` pair is removed from the NodeName line.
- PBS Pro, OpenPBS, and PBS Pro CE. The entry is configured with the `qmgr` utility, using this command: `set node NODENAME <KEY>=<VALUE>`. If the entry is removed or disabled, then it is unset in the `qmgr`.
- UGE. The entry is applied to an `exechost` complex attribute that must already be defined in the configuration. The complex attribute name is set to `<KEY>` and its value to `<VALUE>`. If the entry is removed or disabled, then it is removed from the `exechost` settings of the node.
- LSF. The entry is set in the `lsb.hosts` file, in the `Hosts` section. If the `<KEY>` of the entry is defined as one of the section columns, then the parameter value is replaced by Bright Cluster Manager to `<VALUE>`. When the customization entry is removed or disabled, then its value is replaced by `()` in the section.

8

Bright Cluster Manager Auto Scaler

8.1 Introduction

Bright Cluster Manager Auto Scaler can be used by an administrator to reduce the energy and storage costs of compute nodes by changing their power state, or their existence state, according to workload demands. The idea behind Auto Scaler is that it automatically scales a cluster up or down, on-demand, by powering up physical nodes, cloud nodes, or virtual nodes.

On the cluster itself, Auto Scaler is implemented by the `cm-scale` service. The scaling is carried out according to settings in the `ScaleServer` role which set the nodes that are to use the `cm-scale` service.

The `cm-scale` service runs as a daemon. It collects information about workloads from different workload engines, and it uses knowledge of the nodes in the cluster. In the case of HPC jobs, the daemon also gathers knowledge of the queues that the jobs are to be assigned to¹, and also gathers knowledge on which of the HPC jobs are requesting exclusive node access.

Based on the workload engine information and queues knowledge, the `cm-scale` service can clone and start compute nodes when the workloads are ready to start. The service also stops or terminates compute nodes, when no queued or running workloads remain on the managed nodes.

8.1.1 Use Cases

The `cm-scale` service can be considered as a basis for the construction of different kinds of dynamic data centers. Within such centers, nodes can be automatically re-purposed from one workload engine setup to another, or they can be powered on and off based on the current workload demand.

A few use cases are discussed next, to show how this basis can be built upon:

1. An organization wants to run PBS Pro and Slurm on the same cluster, but how much one workload engine is used relative to the other varies over time. For this case, nodes can be placed in the same pool and shared. When a node finishes an existing job, the `cm-scale` service can then re-purpose that node from one node category to another if needed, pretty much immediately. The re-purposing decision for the node is based on the jobs situation in the PBS Pro and Slurm queues.
2. An organization would like to use their cluster for both Kubernetes and for Slurm jobs. For this case, the admin adds Kubernetes- and Slurm-related settings to the `ScaleServer` role. Using these settings, the `cm-scale` service then switches nodes from one configuration overlay to another. For example, if Slurm jobs are pending and there is a free Kubernetes node, then the node is turned into a Slurm node pretty much immediately.

¹HPC UGE jobs have no queue assignment by default, and are therefore ignored by default. Adding a line such as:
`-q all.q`
to the `var/<instance name>/common/sge_request` file, under `/cm/shared/apps/uge/`, assigns UGE jobs to the `all.q` queue by default, so that `cm-scale` considers them.

3. An organization has a small cluster with Slurm installed on it, and would like to be able to run more jobs on it. This can be carried out using Bright Cluster Manager's Cluster Extension cloud-bursting capability (Chapter 3 of the *Cloudbursting Manual*) to extend the cluster when there are too many jobs for the local cluster. The extension to the cluster is made to a public cloud provider, such as AWS or Azure. For this case, the users can use the `cmjob` utility (section 4.7 of the *User Manual*, and section 4.3 of the *Cloudbursting Manual*) to submit their cloud jobs, and Slurm `sbatch` for their regular jobs. The `cm-scale` service then tracks the Slurm queues, and decides whether or not new cloud nodes should be added from the cloud as extensions to the local network and queues. When the jobs are finished, then the cloud nodes are terminated automatically. Cluster extension typically takes several minutes from prepared images, and longer if starting up from scratch, but in any case this change takes longer than simple re-purposing does.
4. An organization runs PBS Pro only and would like to power down free, local, physical nodes automatically, and start up such nodes when new jobs come in, if needed. In this case, `cm-scale` follows the queues and decides to stop or to start the nodes automatically depending on workload demand. Nodes typically power up in several minutes, so this change takes longer than simple re-purposing does.
5. An organization uses OpenStack as a private cloud, and would like to extend Slurm partitions into that OpenStack private cloud. In this case, `cm-scale` can clone Bright-managed instances in OpenStack (Chapter 5.3 of the *OpenStack Deployment Manual*), and adds them to Slurm's network and partitions. Cloning an instance can take several minutes, so that this change takes longer than simple re-purposing does.

8.1.2 Restrictions

1. Two workload engines of the same kind cannot be served by `cm-scale` on the same Bright Cluster Manager cluster. For example, two Slurm instances cannot be managed, but Slurm and LSF can.
2. When the service considers whether or not the node is suited for the workload, it considers only the number of CPU cores requested by the workload. It does not consider other types of resources. For example, if a workload needs Xeon Phi, then `cm-scale` does not verify if the node actually includes Xeon Phi. The validation is however performed by the engine. If a resource requested by the workload cannot be provided by the node, then the engine (the workload manager) specifies a pending reason that is used by `cm-scale` to decide on node operations.

Number of CPU cores calculation: The `cm-scale` service uses the number of CPU cores on a node in order to map workloads to the node. The calculation of this number is optimized for many node scenarios.

For each node, at the beginning of each `cm-scale` iteration, the following procedure is followed step-by-step to determine the CPU cores number. The procedure for the current iteration stops when a step yields a non-zero value, otherwise the next step is tried.

- (a) If the engine is a workload manager, then the workload manager client role is considered and its `slots` parameter is used. In the very unusual case of several workload manager roles assigned to the node at the same time, then the minimum number of slots is calculated.
- (b) If the node is a cloud node, then its `flavor Type` parameter in the case of EC2, or `VMSize` in case of Azure, is used. These parameters are accessible via `cmsh`, within `device` mode for the cloud node, within the `cloudsettings` submodule. For EC2 nodes the flavor value (`Type`) can be set to a long statement such as: "62 EC2 Compute Units (16 virtual cores with 3.00 EC2 Compute Units each)", and for this statement 16 is used as the number of CPU cores. If the flavor is not defined on the node, then its cloud provider is considered.

- (c) If a template node is defined in the dynamic node provider, and it is a cloud node, then the template node flavor is used for the CPU core number in the same way as shown in the preceding step.
- (d) If the `Default Resources` parameter (in a resource provider) contains "`cpus=N`", where `N` is a number of CPU cores, then `N` is used.
- (e) If the node exists at this moment (for example, it is not just cloned in Bright Cluster Manager), then the CPU cores number is requested from `CMDaemon`, which collects nodes system information. This is used as one of the last methods because it is slow.
- (f) If the node is defined in the `ScaleServer` role via the dynamic resource provider, then the CPU cores number is retrieved from its template node. This method is as slow as the preceding method. It is therefore not recommended when many nodes are managed by `cm-scale`, otherwise the iteration can take several minutes. In such a case, setting the `slots` parameter manually is typically wiser, so that step 1 is the step that decides the CPU cores number.

If `cm-scale` does not manage to find a value for the node, then it prints a warning in the log file and does not make use of the node during the iteration. It is usually wise for the administrator to set the `slots` parameter manually for nodes that otherwise persistently stay unused, so that resources are not wasted.

8.1.3 Setup

The `cm-scale`-related files are placed in a local directory when the administrator installs the `cm-scale` package. If `cm-scale` is needed on a non-head node, then the package should be installed in the corresponding software image. The administrator also typically configures the default user environment for users of the service, so that the `cm-scale` modules environment automatically loads up (section 2.2.3).

The `cm-scale` service does not require any special setup scripts or wizards to run. In order to start using the service, it is enough to assign the `ScaleServer` role to the nodes, and to configure the role properly by setting the role attributes. Unassigning the role stops the `cm-scale` service.

The service writes logs to `/var/log/cm-scale.log`. Running the `cm-scale` daemon in the foreground for debugging or demonstration purposes is also possible, using the `-f` option, and the logs are then duplicated to `STDOUT`:

Example

```
root@bright90$ module load cm-scale
root@bright90$ cm-scale -f
[...]
```

8.1.4 Comparison With `cm-scale-cluster`

Versions of Bright Cluster Manager prior to 8.0 had the `cm-scale-cluster` utility, which provided similar functionality to the current `cm-scale` utility. However, the older utility did not allow workloads (jobs) to be considered for more than one workload manager at a time, and also supported only HPC workload managers as workload engines.

The current `cm-scale` service on the other hand, allows, for example, Slurm workloads to be considered on the same cluster, and also supports other types of workload engines besides HPC workload managers.

Furthermore, the `cm-scale` utility does not use a `crontab` entry. The old `cm-scale-cluster` utility required `cron` to run it regularly, because only one iteration of decision-making was performed by it. The new `cm-scale` utility, on the other hand, runs the decision-making loop internally within the `cm-scale` daemon. The decision-making interval is set in the `ScaleServer` role.

8.2 Configuration

8.2.1 The ScaleServer Role

To configure `cm-scale`, the cluster administrator configures the `ScaleServer` role. The role is typically assigned to head nodes:

Example

```
[bright90]% device use master
[bright90->device[bright90]]% roles
[bright90->device[bright90]->roles]% assign scaleserver
```

The role is configured by setting values to its settings. There are some advanced settings for less common options:

```
[bright90->device*[bright90*]->roles*[scaleserver*]]% show
Parameter                               Value
-----
Add services                             yes
Name                                     scaleserver
Provisioning associations                 <0 internally used>
Revision
Type                                     ScaleServerRole
Engines                                  <0 in submode>
Resource Providers                       <0 in submode>
Dry Run                                  no
Debug                                    no
Run Interval                             120
Advanced Settings                        <submode>
[bright82->device*[bright82*]->roles*[scaleserver*]]% advancedsettings
[bright82->device*[bright82*]->roles*[scaleserver*]->advancedsettings*]% show
Parameter                               Value
-----
Revision
Debug2                                   no
Max Threads                              16
Power Operation Timeout                  30
Connection Retry Interval                5
Log File                                 /var/log/cm-scale.log
Pin Queues                               no
Mix Locations                            yes
Failed Node Is Healthy                   no
Azure Disk Image Name                    images
Azure Disk Container Name                 vhds
Azure Disk Account Prefix
```

An overview of the parameters and submodes is given next. An example showing how they can be configured is given afterwards, in section 8.3.

ScaleServer Role Global Parameters

The `ScaleServer` role has the following global parameters for controlling the `cm-scale` service itself:

- **Dry Run:** If set, then the service runs in dry run mode. In this mode it may claim that actions have been carried out on the nodes that use the `cm-scale` service, however, no action is actually carried out on nodes. This mode is useful for demonstration or debug purposes
- **Log File:** path to the log file, with a default path of `/var/log/cm-scale.log`
- **Run Interval:** interval, in seconds, between `cm-scale` decision-making

ScaleServer Role Submodes

Within the `ScaleServer` role are the following three submodes:

- `advancedsettings`: allows some advanced properties to be set for `cm-scale`, using the parameters displayed on page 318.
- `resourceproviders`: defines the nodes used by `cm-scale`. More explicitly, this submode is used to define resource provider objects. The resource providers can be added as `static` or `dynamic` types, and can then have nodes and settings defined within them. The nodes allocated to these resource provider objects are what provide resources to `cm-scale` when that resource provider is requested.
- `engines`: define the engines used by `cm-scale`. This can be an instance of the type `hpc`, `generic`, or `kubernetes` (page 324).
 - `trackers` (within `engines` submode): define the trackers used by `cm-scale` (page 325)

The parameters are enforced only when the next decision-making iteration takes place.

8.2.2 Resource Providers

The `cm-scale` service allows nodes to change state according to the workload demand. These managed nodes are defined by the administrator within the `resourceproviders` submode of `ScaleServer`. Bright Cluster Manager 9.0 supports two types of resource providers: `static` and `dynamic` node providers.

Static Node Provider

When managed nodes are well-known and will not be extended or shrunk dynamically, then a `static` node provider can be used. Specifying settings for the `static` node provider allows `cm-scale` to power on, power off, or re-purpose nodes, based on `nodegroups` or a list of nodes specified with a `node list` syntax (page 42).

The `static` node provider supports the following properties:

- `Enabled`: The static node provider is currently enabled.
- `Nodes`: A list of nodes managed by `cm-scale`. These can be regular local compute nodes (`nodes`), cluster extension cloud compute nodes (`cnodes`), and Bright-managed OpenStack virtual compute nodes (`vnodes`). For the purposes of this section on `cm-scale`, 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` service 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.
- `Nodegroups`: List of `nodegroups` with nodes to be managed by `cm-scale`. Node groups are classed into *types*. The class of node group types is independent of the class of node types, and should not be confused with it. Node types can be `physicalnode`, `cloudnode`, `gpuunit`, `genericdevice`, `headnode`, `ethernetswitch` and others, and are shown in the first column of the output of the default `list` command in device mode. Node group types currently (May 2017) consist only of `Normal` and `Storage pair`. Only node groups of type `Normal` can currently be managed by `cm-scale`.
- `Priority`: The provider priority. Nodes in the pool of a provider with a higher priority are used first by workloads. By default a resource provider has a priority value 0. These priority values should not be confused with the fairsharing priorities of page 324.

Dynamic Node Provider

When managed nodes can be cloned or removed from the configuration, then a dynamic node provider should be used. A compute node that is managed by `cm-scale` as a dynamic node provider is configured as a template node within the dynamic submode of the `ScaleServer` role.

The dynamic node provider supports the following properties:

- `Template Node`: A node that will be used as a template for cloning other nodes in the pool. The following restrictions apply to the template node:
 - A workload manager client role must be assigned with a positive number of slots.
 - New node names should not conflict with the node names of nodes in a `nodegroup` defined for the queue.
 - A specific template node is restricted to a specific queue.

A template node only has to exist as an object in 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 `Start Template Node` and `Stop Template Node` values apply:

- `Start Template Node`: The template node specified in the `Template Node` parameter is also started automatically on demand.
- `Stop Template Node`: The template node specified in the `Template Node` parameter is also stopped automatically on demand.
- `Never Terminate`: Number of cloud nodes that are never terminated even if no jobs need them. If there are this number or fewer cloud nodes, then `cm-scale` no longer terminates them. Cloud nodes that cannot be terminated can, however, still be powered off, allowing them to remain configured in Bright Cluster Manager. As an aside, local nodes that are under `cm-scale` control are powered off automatically when no jobs need them, regardless of the `Never Terminate` value.
- `Enabled`: Node provider is currently enabled.
- `Priority`: Node provider priority.
- `Node Range`: Range of nodes that can be created and managed by `cm-scale`.
- `Network Interface`: Which node network interface is changed on cloning (incremented).
- `Remove Nodes`: Should the new node be removed from Bright Cluster Manager when the node terminates? If the node is not going to be terminated, but just stopped, then it is never removed.
- `Leave Failed Nodes`: If nodes are discovered to be in a state of `INSTALLER_FAILED` or `INSTALLER_UNREACHABLE` (section 5.5.4) then this setting decides if they can be left alone, so that the administrator can decide what do with them later on.
- `Default Resources`: List of default resources, in format `[name=value]`.
 - `cpu`: value is the number of CPUs
 - `mem`: value is in bytes

These must be set when no real node instance is associated with a node defined in Bright Cluster Manager.

Extra Nodes Settings For Node Providers

Both the `dynamic` and `static` node providers support extra node settings. If configured, then `cm-scale` can start the extra nodes before the first workload is started, and can stop them after the last job from the managed queue is finished.

The most common use case scenario for extra nodes in the case of cloud nodes is a cloud director node. The cloud director node provisions cloud compute nodes and performs other management operations in a cloud.

In the case of non-cloud non-head nodes, extra nodes can be, for example, a license server, a provisioning node, or an additional storage node.

The configuration settings include:

- `Extra Nodes`: A list of extra nodes.
- `Extra Node Idle Time`: The maximum time, in seconds, that extra nodes can remain unused. The `cm-scale` service checks for the existence of queued and active workloads using the extra node, when the time elapsed since the last check reaches `Extra Node Idle Time`. If there are workloads using the extra node, then the time elapsed is reset to zero and a time stamp is written into the file `cm-scale.state` under the directory set by the `Spool` role parameter. The time stamp is used to decide when the next check is to take place. Setting `Extra Node Idle Time=0` means the extra node is stopped whenever it is found to be idle, and started again whenever workloads require it, which may result in a lot of stops and starts.
- `Extra Node Start`: Extra node is started by `cm-scale` before the first compute node is started.
- `Extra Node Stop`: Extra node is stopped by `cm-scale` after the last compute node stops.

The following table summarizes the default attributes in `cmsh` for the `static` and `dynamic` resource providers, along the path `cmsh->device[]->roles->scaleserver->resourceproviders[dynamic/static]`:

Parameter	static	dynamic
Revision		
Type	static	dynamic
Nodes		N/A
Template Node	N/A	
Start Template Node	N/A	no
Stop Template Node	N/A	no
Remove Nodes	N/A	no
Leave Failed Nodes	N/A	yes
Never Terminate	N/A	32
Node Range	N/A	
Network Interface	N/A	tun0
Name	static	dynamic
Enabled	yes	yes
Nodegroups		N/A
Priority	0	0
Keep Running		
Whole Time	0	0
Stopping Allowance Period	0	0
Extra Nodes		
Extra Node Idle Time	3600	3600
Extra Node Start	yes	yes
Extra Node Stop	yes	yes
Default Resources	cpus=1	cpus=1

In the preceding table, the entry N/A means that the parameter is not available for the corresponding resource provider.

8.2.3 Time Quanta Optimization

Time quanta optimization is an additional feature that `cm-scale` can use for further cost-saving with certain cloud providers.

For instance, Amazon charges per whole unit of time, or *time quantum*, used per cloud node, even if only a fraction of that unit of time was actually used. The aim of 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:

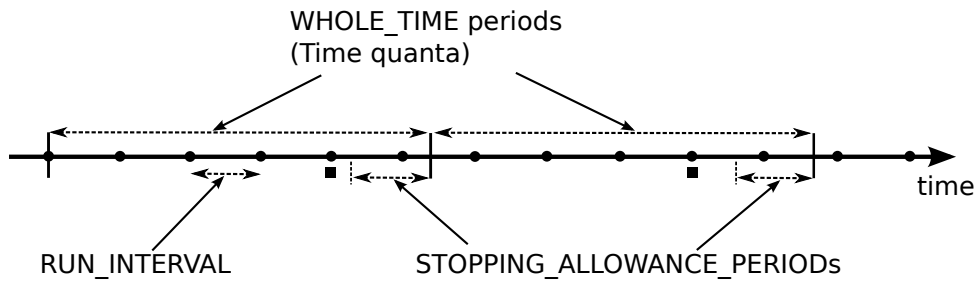
- keep a node up if it is running jobs in the cloud
- keep a node up if it is not running jobs in the cloud, if its cloud time has already been paid for, until that cloud time is about to run out
- take a node down if it is not running jobs in the cloud, if its cloud time is about to run out, in order to avoid being charged another unit of cloud time

Time quanta optimization is implemented with some guidance from the administrator for its associated parameters. The following parameters are common for both `static` and `dynamic` node resource providers:

- `Whole Time`. A compute node running time (in minutes) before it is stopped if no workload requires it. Currently (May 2017) Amazon's time quantum is 60 minutes. By default, Bright Cluster Manager uses a value of `Whole Time=0`, which is a special value that means `Whole Time` is ignored. Ignoring it 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.
- `Stopping Allowance Period`. A time (in minutes) just before the end of the `Whole Time` period, prior to which all power off (or terminate) operations must be started. The parameter associated with time quanta optimization is the `Stopping Allowance Period`. This parameter can also be set by the administrator. The `Stopping Allowance Period` can be understood by considering the *last call time period*. The last call time period is the period between the last call time, and the time that the next whole-time period starts. If the node is to be stopped before the next whole-time charge is applied, then the last call time period must be at least more than the maximum time period that the node takes to stop. The node stopping period in a cluster involves cleanly stopping many processes, rather than just terminating the node instance, and can therefore take some minutes. The maximum period in minutes allowed for stopping the node can be set by the administrator in the parameter `Stopping Allowance Period`. By default, `Stopping Allowance Period=0`. Thus, for nodes that are idling and have no jobs scheduled for them, only if the last call time period is more than `Stopping Allowance Period`, does `cm-scale` stop the node.

The preceding parameters are explained next.

Figure 8.1 illustrates a time line with the parameters used in time quanta optimization.



legend for instances on time line:

- `cm-scale` runs, `RUN_INTERVAL` starts
- | a time quantum ends and next one starts
- ⋮ `STOPPING_ALLOWANCE_PERIOD` starts
- last call

Figure 8.1: Time Quanta Optimization

The algorithm that `cm-scale` follows, with and without time quanta optimization, can now be described using the two parameters explained so far:

1. `cm-scale` as part of its normal working, checks every `Run Interval` seconds to see if it should start up nodes on demand or shut down idling nodes.
2. If it sees idling nodes, then:
 - (a) If `Whole Time` has not been set, or is 0, then there is no time quanta optimization that takes place. The `cm-scale` service then just goes ahead as part of its normal working, and shuts down nodes that have nothing running on them or nothing about to run on them.
 - (b) If a non-zero `Whole Time` has been set, then a time quanta optimization attempt is made. The `cm-scale` service calculates the time period until the next time quantum from public cloud starts. This time period is the current *closing time period*. Its value changes each time that `cm-scale` is run. If
 - the current closing time period is long enough to let the node stop cleanly before the next time quantum starts, and
 - the next closing time period—as calculated by the next `cm-scale` run but also running within the current time quantum—is not long enough for the node to stop cleanly before the next quantum starts

then the current closing time period starts at a time called the *last call*.

In drinking bars, the last call time by a bartender allows some time for a drinker to place the final orders. This allows a drinker to finish drinking in a civilized manner. The drinker is meant to stop drinking before closing time. If the drinker is still drinking beyond that time, then a vigilant law enforcement officer will fine the bartender.

Similarly, the last call time in a scaling cluster allows some time for a node to place its orders to stop running. It allows the node to finish running cleanly. The node is meant to stop running before the next time quantum starts. If the node is still running beyond that time, then a vigilant cloud provider will charge for the next whole time period.

The last call time is the last time that `cm-scale` can run during the current whole-time period and still have the node stop cleanly within that current whole-time period, and before the next whole-time period starts. Thus, when `Whole Time` has been set to a non-zero time:

- i. If the node is at the last call time, then the node begins with stopping
- ii. If the node is not at the last call time, then the node does not begin with stopping

The algorithm goes back again to step 1.

8.2.4 Fairsharing Priority Calculations And Node Management

At intervals of Run Interval, `cm-scale` collects workloads using trackers configured in the `ScaleServer` role, and puts all the workloads in a single internal queue. This queue is then sorted by priorities. The priorities are calculated for each workload using the following fairsharing formula:

$$p_{ij} = a_i \times k_1 + e_j \times k_2$$

where:

p_{ij} is the global priority for the i -th workload of the j -th engine. Its value is used to re-order the queue.

a_i is the age of the workload. That is, how long has passed since the i -th job submission, in seconds. This typically dominates the priority calculation, and makes older workloads become higher priority.

k_1 is the age factor. This is `agefactor` that can be set in the engine submode of the `ScaleServer` role. Usually it is 1.

e_j is the engine priority. This is `priority` in the engine submode of the `ScaleServer` role.

k_2 is the engine factor. This is `enginefactor` in the engine submode of the `ScaleServer` role.

When all the workload priorities are calculated and the queue is re-ordered, then `cm-scale` starts to find appropriate nodes for workloads, from the top of the queue where the higher priority workloads are, to the bottom. This way a higher priority engine has a greater chance of getting nodes for its workloads than a lower priority engine.

8.2.5 Engines

Each workload engine considered by `cm-scale` must be configured within the `engines` submode within the `ScaleServer` role. Bright Cluster Manager 9.0 supports the following workload engines:

- Slurm
- PBS Professional (community and commercial versions)
- LSF
- UGE

Engines can be of three types:

- `hpc`: for all HPC (High Performance Computing) workload managers
- `kubernetes`: for Kubernetes
- `generic`: for a generic type

Common Parameters For The `cm-scale` Engines

All three engine types have the following parameters and submode in common, although their values may differ:

- `Workloads Per Node`: The number of workloads that can be scheduled to run on the same node at the same time.

- For a Kubernetes engine this parameter restricts only the number of either the Kubernetes controllers, or the individual jobs (without any controller) per node. It does not restrict the total number of pods that can be mapped per node by the `cm-scale` scheduler.

For example, a Kubernetes job, or Job with a capital 'J' in Kubernetes terminology, may consist of many pods. Then, if `Workload Per Node` is, for example, 2, then only 2 Jobs can be mapped to the node.

The number of pods is also taken into account by `cm-scale`, but this number is taken from the `Kubernetes::Node` role, where the `Max Pods` option can be set. If the role is not assigned to a node, then `cm-scale` assumes that there is no possibility for any pods to run on the node.

- `Priority`: The engine priority
- `Age Factor`: Fairsharing coefficient for workload age
- `Engine Factor`: Fairsharing coefficient for engine priority
- `Max Nodes`: The maximum number of running nodes allowed
- `Trackers`: Enters the workload trackers submodule

Non-common parameters for the `cm-scale` engines:

- For the `hpc` engine:
 - `WLM Cluster`: A workload manager cluster name. The name is set during workload manager setup as the instance name. In `cmsh` the WLM cluster names are listed under `wlm` mode. In Bright View they can be seen along the clickpath `HPC`→`Wlm Clusters`.
- For the `kubernetes` engine, the following parameters can be set:
 - `Cluster`: These are the Kubernetes clusters for which pods are to be tracked. Bright Cluster Manager allows multiple Kubernetes clusters to run on a single compute cluster. Kubernetes must be already set up before this setting is configured.
 - `CPU Busy Threshold`: The CPU load is a value that can range from 0 to 1. The `CPU Busy Threshold` value defines if the node is too busy for new pods. Its default value is: 0.9.
 - `Memory Busy Threshold`: The Memory load is a value that can range from 0 to 1. The `Memory Busy Threshold` defines if the node is too busy for new pods. Its default value is: 0.9.

The CPU and Memory thresholds configured in the Kubernetes engine help `cm-scale` to decide when more nodes are needed. But `cm-scale` also retrieves the number of pods that are already running on the node and compares it with the `Max Pods` parameter that is configured in the `Kubernetes::Node` role assigned to the node. If the number of running pods is already equal or greater than the value of `Max Pods`, then (from the `cm-scale` point of view) the node cannot fit more pods, which means that a new node is needed.

8.2.6 Trackers

A workload tracker is a way to specify the workload and its node requirements to `cm-scale`. For HPC, the tracker may be associated with a specific queue, and `cm-scale` then tracks the jobs in that queue.

One or more trackers can be named and enabled within the `trackers` submodule, which is located within the `engines` submodule of the `ScaleServer` role. A queue (for workload managers) can be assigned to each tracker.

Example

There are three types of tracker objects supported in Bright Cluster Manager 9.0:

- **queue:** Used with an HPC type engine, where each workload (job) is associated with a particular queue. The attribute `Type` takes the value `ScaleHpcQueueTracker`, and the attribute `Queue` is set to the queue name for the job.
- **namespace:** Used with a `kubernetes` type engine.
- **generic:** Used with a `generic` type engine.

The following settings are common for both types of trackers:

- **Enabled:** Enabled means that workloads from this tracker are considered by `cm-scale`.
- **Assign Category:** A node category name that should be assigned to the managed nodes. When a node is supposed to be used by a workload, then `cm-scale` should assign the node category to that node. If the node is already running, and has no workloads running on it, but its category differs from the category specified for the jobs of the queue, then the node is drained, stopped and restarted on the next decision-making iteration of `cm-scale`, and takes on the assigned category. Further details on this are given in the section on dynamic nodes repurposing, page 340.
- **Primary Overlays:** A list of configuration overlays.

If a workload is associated with the tracker for which the overlays are specified by `Primary Overlays`, then the Bright Cluster Manager-managed nodes are appended to those configuration overlays by `cm-scale`. This takes place after the node is removed from the previous overlays that it is associated with.

If the node is already running, but has not yet been appended to the overlays specified for the workloads of the tracker, then the node is restarted when no other workloads run on the node, before going on to run the workloads with the new overlays.

- When a workload is associated with the tracker that has `Primary Overlays` set, then the pool of `cm-scale`-managed nodes is checked.

The check is to decide on if a node is to be made available for the workload.

If the node is appended to the `Primary Overlays` already and is not running workloads, then `cm-scale` simply hands over a workload from the tracker to run on the node.

If the node is not appended to the `Primary Overlays` already, and is not running workloads, then `cm-scale` prepares and boots the node as follows:

- * the node is drained and rebooted if it is up, or
- * the node is undrained and merely booted if it is not up

The node is removed from any previous overlays that it was with, before booting up, and it is appended to the new overlays of `Primary Overlays`.

- The threshold settings:
 - **Queue Length Threshold:** number of pending workloads that triggers cloudbursting.
 - **Age Threshold:** workload pending time threshold, in seconds, that triggers cloudbursting for this workload.

The queue length and age thresholds allow the administrator to set when `cm-scale` starts or creates cloudbursting nodes. Both thresholds can be used at the same time, or just one of them can be used and the other can be ignored by setting it to 0.

If the queue length threshold is set, then `cm-scale` ignores pending workloads that are located higher (added later) than the threshold in the managed queue.

Example

Assuming there are 5 jobs in the queue, with job IDs 1, 2, 3, 4, and 5, where the 1st one is the first in the queue. If the queue length threshold is 3, then only jobs 1, 2 and 3 are taken into account, while jobs 4 and 5 are ignored.

Example

If the age threshold is set to 100, then only workloads older than 100 seconds are taken into account, while younger jobs are ignored.

The queue type tracker has only one parameter specific to the tracker: `Queue`. This is set to the workload queue that is being tracked.

Namespace Tracker

The namespace tracker of `cm-scale` is used to track Kubernetes workloads. It tracks Kubernetes jobs (via its Job controllers) and tracks individual pods. It does not start new nodes for pending pods owned by other types of Kubernetes pod controllers, such as `ReplicaSet`, `DaemonSet`, and so on. If non-Job controllers are running, then `cm-scale` will not stop or terminate those nodes.

The tracker settings in `cmsh` or Bright View include additional parameters that are in common with other trackers:

1. **Controller Namespace:** Tracks the Kubernetes namespace name. Only Kubernetes workloads from this namespace are tracked. To track more than one namespace, one tracker must be created per namespace.
2. **Object:** Type of Kubernetes objects to track. Bright Cluster Manager supports the following object types:
 - (a) **Job:** A Kubernetes Job controller type represents one or several pods that are expected to eventually terminate. The controller nature makes this type of Kubernetes workload very suited to dynamic data centers.
 - (b) **Pod:** Individual pod, without any controller.

If the specified namespace does not exist in Kubernetes, then the tracked jobs or individual pods in this namespace are ignored by `cm-scale`.

Generic Engine And Tracker

The `cm-scale` service is able to deal with workloads that use various workload types. In order to add support of a new type of workload, the administrator

- adds an engine of type `generic`
- adds one or more trackers of type `generic` to the `ScaleServer` role
- implements `Tracker` and `Workload` classes in the Python programming language

When `cm-scale` starts a new iteration, it re-reads the engines and trackers settings from the `ScaleServer` role, and searches for the appropriate modules in its directories. In the case of a custom tracker, the module is always loaded according to the tracker handler path. When the tracker module is loaded, `cm-scale` requests a list of workloads from each of the tracker modules. So, the aim of the tracker module is to collect and provide the workloads to `cm-scale` in the correct format.

The path to the tracker module should be specified in the handler parameter of the generic tracker entity using `cmsh` or Bright View as follows:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device roles master
[bright90->device[bright90]->roles]% use scaleserver
[bright90->device[bright90]->roles[scaleserver]]% engines
...->roles[scaleserver]->engines]% add generic myengine
...*->roles*[scaleserver*]->engines*[myengine*]]% trackers
...*[myengine*]->trackers]% add generic mytracker
...*[mytracker*]]% set handler /cm/local/apps/cm-scale/lib/python3.7/site-packages/examples/tracker.py
...*->roles*[scaleserver*]->engines*[myengine*]->trackers*[mytracker*]]% commit
...->roles[scaleserver]->engines[myengine]->trackers[mytracker]]%
```

In the preceding example, the handler file `.../examples/tracker.py` is an example that is provided with the `cm-scale` package. Another example module file provided with the package is `.../examples/workload.py`, which implements the `ExampleWorkload` class. Together, the two examples can be used to generate any amount of simple workloads during each `cm-scale` iteration. It is recommended to use the tracker and workload classes as templates for custom tracker and workload modules created by the administrator.

The generic engine does not have any specific parameters associated with its type. It only has parameters common to all of the engine types.

If the generic engine is configured with at least one generic tracker in its role, then `cm-scale` loads the handler module and uses two functions that are implemented in the class. The class name can be chosen arbitrarily, but should contain the string `"Tracker"`, without the quotes. The two class functions used are:

1. `__init__`: initializes the tracker object. This can be omitted if there are no additional data values to initialize.
2. `get_workloads`: returns a list of objects belonging to a new class inherited from the `Workload` class. This new class should be created by the administrator.

The new workload class must provide the following functions and properties. The class name can be chosen arbitrarily:

1. `__init__`: initializes the workload object.
2. `to_string`: returns a string that identifies the workload. This is printed to the log file.
3. `begin_timestamp`: property that returns a unix timestamp that should be `>0` if the workload is not allowed to start before that time. If it is `0` then it is ignored by `cm-scale`.

For example, the following very simple tracker and workload classes can be implemented:

Example

```
class ExampleTracker(Tracker):
    def to_string(self):
        return "my workload"

    def get_workloads(self):
        return [ExampleWorkload(self)]

class ExampleWorkload(Workload):
    def __init__(self):
        Workload.__init__(self, tracker)
        self.set_id("1")
        self._update_state()
        self._update_age()
```

```
        self._update_resources()

def to_string(self):
    return "workload %s" % self._id

def _update_state():
    self._set_pending()

def _update_age(self):
    self._age = 0

def _update_resources(self):
    node_res = NodeResource("*")

    cpus_res = CpusResource(1)
    node_res.add_resource(cpu_res)

    engine_res = EngineResource("myengine")
    node_res.add_resource(engine_res)

    self.add_resource(node_res)
```

The classes should be located in different files, as Python module files. It is recommended, but not required, to keep the both files in the same directory. The `ExampleWorkload` class initializes the workload object with a state, age, and required resources. These values are described next.

State: The state can be pending, running or failed, which can be set with these appropriate functions:

- `self._set_pending()`
- `self._set_running()`
- `self._set_failed()`

If the state is running, then the workload is treated as one that occupies the nodes defined in the resources list. Each `NodeResource` object in the resources thus represents one occupied node.

If the state is pending, then the workload is treated as one that waits for free nodes. In this case `cm-scale` tries to find (start or clone) some more nodes in order to allow the workload engine to start this workload.

The failed workload state is considered by `cm-scale` as exceptional. Such a workload is logged in the log file, but is not considered when `cm-scale` decides what nodes to start or clone. Any other workload state is also ignored by `cm-scale`.

Age: The age defines how many seconds the workload is waiting for its resources since being added to the engine. Usually the engine can provide such information, so in the example age is set to 0, which means the workload has been added to the workload engine just now. The age is used in the fairsharing workload priority calculation (page 324). The value of age is not re-calculated by `cm-scale` after a while. This means that the number that the module sets in the class is used during the iteration, and then forgotten by `cm-scale` until the workload object is recreated from scratch on the next iteration.

Resources: The `Workload` class (a base class) includes the `resources` property with list types. This list includes resource objects that are used by `cm-scale` in order to find appropriate nodes for the workload. The top level resource type is always `NodeResource`. There can be one or several node resources requested by the workload.

If the node names are known, then one `NodeResource` object is created per compute node.

Otherwise a single `NodeResource` object is used as many times as the number of requested nodes, with the name set to `*`, which is treated by `cm-scale` as any suitable node. The number of nodes can be set in the `NodeResource` object with the `set_amount(number)` function of the resource.

In the preceding example one (any) node resource is added to the workload request, and the requirement for CPU (cores) number is set to 1. The engine resource is used in order to restrict the mapping of workloads from different engines to one node. Thus if a node has this resource assigned, then the node can take on the workload. If no engine resource is assigned to the node, then it can also take on the workload, but the engine resource of the workload is assigned to the node before other workloads are considered.

The resource types that can be added to the workload are defined in the Python module `/cm/local/apps/cm-scale/lib/core/resource.py`:

- `NodeResource`: top level resource, contains all other resources.
- `CpusResource`: defines the number of cpu cores required or already used by the workload.
- `CategoryResource`: node category required by the workload.
- `OverlayResource`: required configuration overlay.
- `QueueResource`: HPC queue that the workload (job) belongs to. Used only with engines that support queues.
- `EngineResource`: engine name that the workload belongs to.
- `FeatureResource`: required node feature (node property, in other terminology) that should be supported by the engine.

Custom resource types are not supported for now.

Enabling Node Shutdown

By default, `cm-scale` powers off nodes belonging to a resource pool once there is no more workload for them. Resource Providers can also enable shutdown, to give the node time to terminate gracefully.

Shutdown has two options that can be set by the cluster administrator in the configuration file at `/cm/local/apps/cm-scale/lib/python3.7/site-packages/cmscale/config.py`:

- `SHUTDOWN_ENABLED`: If set to `yes`, then the shutdown command is run to terminate the system services first, and after that a command is run to power off the system. A waiting time of `SHUTDOWN_TIMEOUT` seconds takes place between the two events.
- `SHUTDOWN_TIMEOUT`: The number of seconds to wait before powering off a node that is in a shutdown state.

It may take more than `Shutdown Timeout` seconds for a node to power off, depending on the `Run Interval` setting. For example, if `Shutdown Timeout` is 60 seconds, and `Run Interval` is 50, then effectively the `Shutdown Timeout` is 100 seconds, because the power off event only occurs during an iteration execution of `cm-scale`.

8.3 Examples Of `cm-scale` Use

8.3.1 Simple Static Node Provider Usage Example

The example session that follows explains how a static node provider (page 319) can be configured and used with `cm-scale`. The session considers a default cluster with a head node and 5 regular nodes which have been previously defined in Bright Cluster Manager configuration. 3 of the regular nodes are

powered down at the start of the run. The power control for the nodes must be functioning properly, or otherwise `cm-scale` cannot power nodes on and off.

The head node has the Slurm server role by default, and the regular nodes run with the Slurm client role by default. So, on a freshly-installed cluster, the `roleoverview` command should show something like:

Example

```
[bright90->device[bright90]]% roleoverview | head -2 ; roleoverview | grep slurm
Role           Nodes           Categories      Nodes up
-----
slurmclient    node001..node005  default        2 of 5
slurmserver    bright90          1 of 1
```

A test user, `fred` can be created by the administrator (section 6.2), and an MPI hello executable based on the `hello.c` code (from section 3.5.1 of the *User Manual*) can be built:

Example

```
[fred@bright90 ~]$ module add shared openmpi/gcc/64/1.10.3 slurm
[fred@bright90 ~]$ 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@bright90 ~]$ 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@bright90 ~]$ while (true); do sbatch slurmhello.sh; done
```

After putting enough jobs into the queue (a few thousand should be enough, and keeping it less than 5000 would be sensible) the flooding can be stopped with a `ctrl-c`.

The activity in the queue can be watched:

Example

```
[root@bright90 ~]# watch "squeue | head -3 ; squeue | tail -3"
```

```
Every 2.0s: squeue | head -3 ; squeue | tail -3 Thu Sep 15 10:33:17 2016
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
6423	defq	slurmhel	fred	CF	0:00	1	node001
6424	defq	slurmhel	fred	CF	0:00	1	node002
6572	defq	slurmhel	fred	PD	0:00	1	(Priority)
6422	defq	slurmhel	fred	R	0:00	1	node001
6423	defq	slurmhel	fred	R	0:00	1	node002

The preceding indicates that node001 and node002 are being kept busy running the batch jobs, while the remaining nodes are not in use.

The administrator can check on the job status via the job metrics of `cmsh` too, using the options to the `filter` command, such as `--pending` or `--running`:

Example

```
[root@bright90 ~]# cmsh -c "wlm use slurm; jobs; watch filter --running -u fred"
Every 2.0s: filter --running -u fred                      Wed May  7 12:50:05 2017
Job ID Job name      User Queue Submit time Start time End time Nodes          Exit code
-----
406    slurmhello.sh fred defq  16:16:56  16:27:21  N/A    node001,node002  0
```

and eventually, when jobs are no longer running, it should show something like:

```
[root@bright90 ~]# cmsh -c "wlm use slurm; jobs; watch filter --running -u fred"
Every 2.0s: filter --running                               Wed May  7 12:56:53 2017
No jobs found
```

So far, the cluster is queuing or running jobs without `cm-scale` being used.

The next steps are to modify the behavior by bringing in `cm-scale`. The administrator assigns the `ScaleServer` role to the head node. Within the role a new static node provider, Slurm engine, and queue tracker for the `defq` are set as follows:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device roles master
[bright90->device[bright90]->roles]% use scaleserver
[bright90->device[bright90]->roles[scaleserver]]% resourceproviders
...->roles[scaleserver]->resourceproviders]% add static pool1
...*->roles*[scaleserver*]->resourceproviders*[pool1*]]% set nodes node001..node005
...*->roles*[scaleserver*]->resourceproviders*[pool1*]]% commit
...->roles[scaleserver]->resourceproviders[pool1]]% ..;..
...->roles[scaleserver]]% engines
...->roles[scaleserver]->engines]% add hpc slurm1
...*->roles*[scaleserver*]->engines*[slurm1*]]% set wlmcluster slurm
...*->roles*[scaleserver*]->engines*[slurm1*]]% trackers
...*->roles*[scaleserver*]->engines*[slurm1*]->trackers]% add queue tr1
...*->roles*[scaleserver*]->engines*[slurm1*]->trackers*[tr1*]]% set queue defq
...*->roles*[scaleserver*]->engines*[slurm1*]->trackers*[tr1*]]% commit
...->roles[scaleserver]->engines[slurm1]->trackers[tr1]]%
```

The nodes `node001..node005` should already be in the queue `defq`, as assigned to them by default when they were assigned the `SlurmClient` role. With these settings, they can now be powered up or down on demand by `cm-scale` service, depending on the number of jobs that are pending. When the new `ScaleServer` role is committed in `cmsh` or Bright View, then the `cm-scale` service is started. If needed, the administrator can check the log file `/var/log/cm-scale` to see what the service is doing.

On each iteration `cm-scale` checks whether the node states should be changed. Thus after a while, the nodes `node003..node005` are started. Once up, they can start to process the jobs in the queue too.

Watching the running jobs should show something like:

Example

```
[root@bright90 ~]# cmsh -c "wlm use slurm; jobs ; watch filter --running"
```

```
Every 2.0s: filter --running                               Thu Sep  8 13:01:59 2016
Job ID Job name      User Queue Submit time      Start time      Nodes  Exit code
-----
16498 slurmhello.sh fred defq  07/05/2017 12:44:20 07/05/2017 13:20:58 node003 0
16499 slurmhello.sh fred defq  07/05/2017 12:44:20 07/05/2017 13:20:58 node002 0
```

Eventually, cm-scale finds that all jobs have been dealt with, and the nodes are then powered down.

High-availability And Using A Configuration Overlay For The ScaleServer Role

For high-availability clusters, where there are two head nodes, the scaleserver should run on the active head node. One labor-intensive way to set this up is to assign the service to both the head nodes, and match the scaleserver settings on both head nodes. A simpler way is to define a configuration overlay for the head nodes for the scaleserver. If the head nodes are bright90-1 and bright90-2, then a configuration overlay called brightheads can be created and assigned the service as follows:

Example

```
[bright90-1]% configurationoverlay add brightheads
[bright90-1->configurationoverlay*[brightheads*]]% append nodes bright90-1 bright90-2
[bright90-1->configurationoverlay*[brightheads*]]% roles
[bright90-1->configurationoverlay*[brightheads*]->roles]% assign scaleserver
[bright90-1->configurationoverlay*[brightheads*]->roles*[scaleserver*]]%
```

The scaleserver can then be configured within the configuration overlay instead of on a single head as was done previously in the example of page 332. After carrying out a commit, the scaleserver settings modifications are then mirrored automatically between the two head nodes.

Outside the scaleserver settings, one extra modification is to set the cm-scale service to run on a head node if the head node is active. This can be done with:

Example

```
[bright90-1->configurationoverlay[brightheads]->roles[scaleserver]]% device services bright90-1
[bright90-1->device[bright90-1]->services]% use cm-scale
[bright90-1->device[bright90-1]->services[cm-scale]]% set runif active
[bright90-1->device*[bright90-1*]->services*[cm-scale*]]% commit
[bright90-1->device[bright90-1]->services]% device use bright90-2
[bright90-1->device[bright90-2]->services]% use cm-scale
[bright90-1->device[bright90-2]->services[cm-scale]]% set runif active
[bright90-1->device*[bright90-2*]->services*[cm-scale*]]% commit
```

The result is a scaleserver that runs when the head node is active.

8.3.2 Simple Dynamic Node Provider Usage Example

The following example session explains how a dynamic node provider (page 320) can be configured and used with cm-scale. The session considers a default cluster with a head node and 2 regular nodes which have been previously defined in Bright Cluster Manager configuration, and also 1 cloud director node and 2 cloud compute nodes. The cloud nodes can be configured using cm-cluster-extension. Only the head node is running at the start of the session, while the regular nodes and cloud nodes are all powered down at the start of the run.

At the start, the device status shows something like:

Example

```
[bright90->device]% ds
eu-west-1-cnnode001 ..... [ DOWN ] (Unassigned)
eu-west-1-cnnode002 ..... [ DOWN ] (Unassigned)
eu-west-1-cnnode003 ..... [ DOWN ] (Unassigned)
eu-west-1-director ..... [ DOWN ]
node001 ..... [ DOWN ]
node002 ..... [ DOWN ]
bright90 ..... [ UP ]
```

The power control for the regular nodes must be functioning properly, or otherwise `cm-scale` cannot power them on and off.

If the head node has the `slurmserver` role, and the regular nodes have the `slurmclient` role, then the `roleoverview` command should show something like:

Example

```
[bright90->device[bright90]]% roleoverview
```

Role	Nodes	Categories	Nodes up
boot	bright90		1 of 1
cgroupsupervisor	eu-west-1-cnnode001..eu-west-1-cnnode002 ,eu-west-1-director,node001..node002 ,bright90	aws-cloud-director,default ,eu-west-1-cloud-node	1 of 6
clouddirector	eu-west-1-director		0 of 1
cloudgateway	bright90		1 of 1
login	bright90		1 of 1
master	bright90		1 of 1
monitoring	bright90		1 of 1
provisioning	eu-west-1-director,bright90		1 of 2
slurmclient	eu-west-1-cnnode001..eu-west-1-cnnode002 ,node001..node002	default,eu-west-1-cloud-node	0 of 3
slurmserver	bright90		1 of 1
storage	eu-west-1-director,bright90	aws-cloud-director	1 of 2

A test user, `fred` can be created by the administrator (section 6.2), and an MPI hello executable based on the `hello.c` code (from section 3.5.1 of the *User Manual*) can be built:

Example

```
[fred@bright90 ~]$ module add shared openmpi/gcc/64/1.10.3 slurm
[fred@bright90 ~]$ 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@bright90 ~]$ cat slurmhello.sh
#!/bin/sh
#SBATCH -o my.stdout
#SBATCH --time=30 #time limit to batch job
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
module add shared openmpi/gcc/64/1.10.1 slurm

mpirun /home/fred/hello
```


A default cluster can queue or run jobs without `cm-scale` being used. The default behavior is modified in the next steps, which bring in the `cm-scale` service:

The administrator assigns the `ScaleServer` role to the head node.

Example

```
[root@bright90 ~]# cmsh
[bright90]% device roles master
[bright90->device[bright90]->roles]% assign scaleserver
```

Within the assigned `scaleserver` role, a new dynamic node provider can be set, and properties for the dynamic pool of nodes can be set for the cloud compute nodes. Here the properties that are set are `priority` (page 319), `templatenode` (page 320), `noderange` (page 320), and `extranodes` (page 321).

Example

```
[bright90->device*[bright90*]->roles*[scaleserver*]]% resourceproviders
...->roles[scaleserver]->resourceproviders]% add dynamic pool2
...resourceproviders*[pool2*]]% set priority 2
...resourceproviders*[pool2*]]% set noderange eu-west-1-cnode001..eu-west-1-cnode002
...resourceproviders*[pool2*]]% set templatenode eu-west-1-cnode001
...resourceproviders*[pool2*]]% set extranodes eu-west-1-director
...resourceproviders*[pool2*]]% commit
...resourceproviders[pool2]]%
```

The regular compute nodes, `node001..node002` should be specified as nodes in the static pool.

The administrator may notice the similarity of dynamic and static pool configuration. The Bright Cluster Manager front end has deliberately been set up to present dynamic pool and static pool nodes to the cluster administrator as two different configuration methods. This is because separating the pool types as dynamic and static pools is simpler for the cluster administrator to deal with. This way, regular compute nodes are treated, not as a special case of a dynamic pool, but simply as static pool nodes. The fundamental reason behind this separate treatment is because physical nodes cannot “materialize” dynamically with properties in the way the cloud compute nodes—which are virtualized nodes—can, due to the need to associate a MAC address with a physical node.

Assigning regular compute nodes to a static pool can be done in a similar way to what was shown before in the example on page 332.

Continuing with the current session, the nodes `node001..node002` are added to the static pool of nodes, `on-premises-nodes`. For this example they are set to a lower priority than the cloud nodes:

Example

```
...->roles[scaleserver]->resourceproviders]% add static on-premises-nodes
...->roles*[scaleserver*]->resourceproviders*[on-premises-nodes*]]% set nodes node001..node002
...->roles*[scaleserver*]->resourceproviders*[on-premises-nodes*]]% set priority 1
...->roles*[scaleserver*]->resourceproviders*[on-premises-nodes*]]% commit
...->roles[scaleserver]->resourceproviders[on-premises-nodes]]%
```

What this lower priority means is that a node that is not up and is in the static pool of nodes, is only powered on after all the cloud nodes are powered on and busy running jobs. If there happen to be nodes from the static pool that are already up, but are not running jobs, then these nodes take a job, despite the lower priority of the static pool, and irrespective of whether the dynamic pool nodes are in use.

Job priorities can be overridden in `cm-scale` by:

- allowing locations by setting `Mix Locations` to `true` (page 342) or
- pinning queues by setting `Pin Queues` to `true` (page 344)

A Slurm engine, and queue tracker for the defq are set as follows:

Example

```
...]->roles[scaleserver]]% engines
...]->roles[scaleserver]->engines]% add hpc slurm2
...*]->roles*[scaleserver*]->engines*[slurm2]]% set wlmcluster slurm
...*]->roles*[scaleserver*]->engines*[slurm2]]% trackers
...*]->roles*[scaleserver*]->engines*[slurm2]->trackers]% add queue tr2
...*]->roles*[scaleserver*]->engines*[slurm2]->trackers*[tr2*]]% set queue defq
...*]->roles*[scaleserver*]->engines*[slurm2*]->trackers*[tr2*]]% commit
...->roles[scaleserver]->engines[slurm2]->trackers[tr2]]%
```

The nodes node001..node002 and eu-west-1-cnode001..eu-west-1-cnode002 should already be in the queue defq by default, ready to run the jobs:

Example

```
...->roles[scaleserver]->engines[slurm2]->trackers[tr2]]% wlm use slurm; jobqueue; get defq nodes
eu-west-1-cnode001
eu-west-1-cnode002
node001
node002
```

The `roleoverview` (page 334) command is also handy for an overview, and to confirm that the role assignment of these nodes are all set to the `SlurmClient` role:

With these settings, the nodes in the dynamic pool can now be powered up or down on demand by `cm-scale` service, depending on the number of jobs that are pending. When the new `ScaleServer` role is committed in `cmsh` or Bright View, then the `cm-scale` is run periodically. Each time it is run, `cm-scale` checks whether the node states should be changed. If needed, the administrator can check the log file `/var/log/cm-scale` to see what the service is doing.

Job submission can now be carried out, and the `scaleserver` assignment carried out earlier scales the cluster to cope with jobs according to the configuration that has been carried out in the session.

Before submitting the batch jobs, the administrator or user can check the jobs that are queued and running with the `squeue` command. If there are no jobs yet submitted, the output is simply the `squeue` headers, with no job IDs listed:

Example

```
[fred@bright90 ~]$ squeue
JOBID PARTITION    NAME      USER ST       TIME  NODES NODELIST(REASON)
```

As in the previous example for the static pool only case (page 330), a way for user fred to flood the default queue defq is to run the batch file in a loop:

Example

```
[fred@bright90 ~]$ while (true); do sbatch slurmhello.sh; done
Submitted batch job 1
Submitted batch job 2
Submitted batch job 3
...
```

After putting enough jobs into the queue (a few thousand should be enough, not more than five thousand would be sensible), the flooding can be stopped with a `ctrl-c`.

The changes in the queue can be watched by user fred:

Example

```
[fred@bright90 ~]$ watch "squeue | head -5 ; squeue | tail -4"
Every 2.0s: squeue | head -5 ; squeue | tail -4          Wed Nov 22 16:08:52 2017
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
1	defq	slurmhel	fred	PD	0:00	1	(Resources)
2	defq	slurmhel	fred	PD	0:00	1	(Resources)
3	defq	slurmhel	fred	PD	0:00	1	(Resources)
4	defq	slurmhel	fred	PD	0:00	1	(Resources)
3556	defq	slurmhel	fred	PD	0:00	1	(Resources)
3557	defq	slurmhel	fred	PD	0:00	1	(Resources)
3558	defq	slurmhel	fred	PD	0:00	1	(Resources)
3559	defq	slurmhel	fred	PD	0:00	1	(Resources)

The `head -4` and `tail -4` filters here are convenient for showing just the first 4 rows and last 4 rows of the very long `squeue` output, and skipping the bulk of the queue.

The preceding output illustrates how, with the jobs queued up, nothing is being processed yet from jobs number 1 to 3559 due to the resources not yet being available.

At this point `cm-scale` should have noticed that jobs are queued and that resources are needed to handle the jobs.

At the start of this example session the cloud director is not up. So, `cm-scale` powers it up. This can be seen by running the `ds` command, or from `CMDaemon` info messages:

```
[bright90->device]% ds | grep director
eu-west-1-director [ DOWN ]
  then some time later:
eu-west-1-director [ PENDING ] (External ip assigned: 34.249.166.63, setting up tunnel)
  then some time later:
eu-west-1-director [   INSTALLING   ] (node installer started)
  then some time later:
eu-west-1-director [ INSTALLER_CALLINGINIT ] (switching to local root)
  then some time later:
eu-west-1-director [  UP   ]
```

If the cloud director is yet to be provisioned to the cloud from the head node for the very first time (“from scratch”), then that can take a while. Then, because the cloud compute nodes are in turn provisioned from the cloud director, it takes a while for the cloud compute nodes to be ready to run the jobs. So, the jobs just have to wait around in the queue until the cloud compute nodes are ready, before they are handled. Fortunately, the startup of a cloud director is by default much faster after the very first time.

A quick aside about how provisioning is speeded up the next time around: The cloud compute nodes will be stopped if they are idle, and after there are no more jobs in the queue, because the jobs have all been dealt with. Then, when the `extranodeidletime` setting has been exceeded, the cloud director is also stopped. The next time that jobs are queued up, all the cloud nodes are provisioned from a stopped state, rather than from scratch, and so they are ready for job execution much faster. Therefore, unlike the first time, the jobs queued up the next time are processed with less waiting around.

Getting back to how things proceed in the example session after the cloud director is up: `cm-scale` then provisions the cloud compute nodes `eu-west-1-node001` and `eu-west-1-node002` from the cloud director.

Example

```
[bright90->device]% ds | grep cnode
eu-west-1-cnode001 ..... [ PENDING ] (Waiting for instance to start)
```

```
eu-west-1-cnode002 ..... [ PENDING ] (Waiting for instance to start)
  then some time later:
eu-west-1-cnode002 [   INSTALLING   ] (node installer started)
eu-west-1-cnode001 [   INSTALLING   ] (node installer started)
  and so on
```

Once these cloud compute nodes reach the state of UP, they can start to process the jobs in the queue. The queue activity then would show something like:

Example

when the dynamic pool nodes are being readied for job execution:

```
[fred@bright90 ~]$ squeue | head -5 ; squeue | tail -4
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
1	defq	slurmhel	fred	PD	0:00	1	(Resources)
2	defq	slurmhel	fred	PD	0:00	1	(Resources)
3	defq	slurmhel	fred	PD	0:00	1	(Resources)
4	defq	slurmhel	fred	PD	0:00	1	(Resources)
3556	defq	slurmhel	fred	PD	0:00	1	(Resources)
3557	defq	slurmhel	fred	PD	0:00	1	(Resources)
3558	defq	slurmhel	fred	PD	0:00	1	(Resources)
3559	defq	slurmhel	fred	PD	0:00	1	(Resources)

then later:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
11	defq	slurmhel	fred	CF	0:00	1	eu-west-1-cnode001
12	defq	slurmhel	fred	CF	0:00	1	eu-west-1-cnode002
13	defq	slurmhel	fred	CF	0:00	1	(priority)
14	defq	slurmhel	fred	CG	0:00	1	(priority)
3556	defq	slurmhel	fred	PD	0:00	1	(Priority)
3557	defq	slurmhel	fred	PD	0:00	1	(Priority)
3558	defq	slurmhel	fred	PD	0:00	1	(Priority)
3559	defq	slurmhel	fred	PD	0:00	1	(Priority)

then later, when cm-scale sees all of the dynamic pool is used up, the lower priority static pool gets started up:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
165	defq	slurmhel	fred	CF	0:00	1	eu-west-1-cnode001
166	defq	slurmhel	fred	CF	0:00	1	node001
168	defq	slurmhel	fred	CG	0:00	1	node002
3556	defq	slurmhel	fred	PD	0:00	1	(Priority)
3557	defq	slurmhel	fred	PD	0:00	1	(Priority)
3558	defq	slurmhel	fred	PD	0:00	1	(Priority)
3559	defq	slurmhel	fred	PD	0:00	1	(Priority)
167	defq	slurmhel	fred	R	0:00	1	eu-west-1-cnode002

In cmsg, the priority can be checked with:

Example

```
[bright90 ->device[bright90]->roles[scaleserver]->resourceproviders]% list
```

Name (key)	Priority	Enabled
on-premises-nodes	1	yes
pool2	2	yes

Also in cmsh, the jobs can be listed via the jobs submode:

Example

```
[bright90->wlm[slurm]->jobs]% list | head -5 ; list | tail -4
Type   Job ID User  Queue Running time Status      Nodes
-----
Slurm  334   fred  defq  1s          COMPLETED eu-west-1-cnode001
Slurm  336   fred  defq  1s          COMPLETED node001
Slurm  3556  fred  defq  0s          PENDING
Slurm  3557  fred  defq  0s          PENDING
Slurm  3558  fred  defq  0s          PENDING
Slurm  3559  fred  defq  0s          PENDING
Slurm  335   fred  defq  1s          RUNNING    eu-west-1-cnode002
[bright90->wlm[slurm]->jobs]%
```

Eventually, when the queue has been fully processed, the jobs are all gone:

Example

```
[fred@bright90 ~]$ squeue
JOBID PARTITION   NAME      USER ST       TIME  NODES NODELIST(REASON)
```

With the current configuration the cloud compute nodes in the dynamic pool pool2 are powered up before the regular compute nodes in the static pool on-premises-pool. That is because the cloud compute nodes have been set by the administrator in this example to have a higher priority. This is typically suboptimal, and is actually configured this way just for illustrative purposes. In a real production cluster, the priority of regular nodes is typically going to be set higher than that for cloud compute nodes, because using on-premises nodes is likely to be cheaper.

The administrator can also check on the job status via the job metrics of cmsh too, using the options to the filter command, such as `--pending` or `--running`:

Initially, before the jobs are being run, something like this will show up:

Example

```
[root@bright90 ~]# cmsh -c "wlm use slurm; jobs ; watch filter --running -u fred"
Every 2.0s: filter --running -u fred                               Wed Nov 22 16:03:18 2017
No jobs found
```

Then, eventually, when the jobs are being run, the cloud nodes, which have a higher priority, start job execution, so that the output looks like:

Example

```
Every 2.0s: filter --running -u fred                               Wed Nov 22 16:50:35 2017
Job ID Job name      User Queue Submit time Start time End time Nodes      Exit code
-----
406   slurmhello.sh fred defq  16:16:56  16:27:21  N/A    eu-west1-cnode001 0
407   slurmhello.sh fred defq  16:16:56  16:27:21  N/A    eu-west1-cnode002 0
```

and eventually the regular on-site nodes which are originally down are started up by the ScaleServer and are also listed.

8.4 Further `cm-scale` Configuration And Examples

8.4.1 Dynamic Nodes 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 in `cm-scale`. Different settings, or a different software image, then run on the re-assigned node after reprovisioning.

The feature is enabled by setting `Assign Category` parameter in the tracker settings.

For example, the following case uses Slurm as the workload engine, and sets up two queues `chem_q` and `phys_q`. Assuming in this example that jobs that are to go to `chem_q` require chemistry software on the node, but jobs for `phys_q` require physics software on the node, and that for some reason the softwares cannot run on the node at the same time. Then, the nodes can be 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 trackers configuration may then look as follows:

Example

```
[bright90->device [bright90] ->roles [scaleserver] ->engines [slurm] ->trackers [chem]]% show
Parameter                               Value
-----
Type                                     ScaleHpcQueueTracker
Name                                     chem
Queue                                    chem_q
Enabled                                  yes
Assign Category                          chem_cat
Primary Overlays
[bright90->device [bright90] ->roles [scaleserver] ->engines [slurm] ->trackers [chem]]% use phys
[bright90->device [bright90] ->roles [scaleserver] ->engines [slurm] ->trackers [phys]]% show
Parameter                               Value
-----
Type                                     ScaleHpcQueueTracker
Name                                     chem
Queue                                    chem_q
Enabled                                  yes
Assign Category                          phys_cat
Primary Overlays
[bright90->device [bright90] ->roles [scaleserver] ->engines [slurm] ->trackers [phys]]%
```

Assuming that initially there are two nodes, `node001` and `node002`, both in category `chem_cat`. Then, when `cm-scale` finds a pending job in queue `phys_q`, it may decide to assign category `phys_cat` to either `node001`, or to `node002`. In this way the number of nodes serving queue `phys_q` increases and number of nodes serving `chem_q` decreases, in order to handle the current workload. When the job is finished, the old node category is not assigned back to the node, until a new job appears in `chem_q` and requires this node to have the old category.

8.4.2 Pending Reasons

This section is related only to HPC engines (workload managers). In this section, the term job is used instead of workload.

If `cm-scale` makes a decision on how many nodes should be started for a job, then it checks the status of the job first. If the job status is pending, then it checks the list of *pending reasons* for that job. The checks are to find pending reasons that prevent the job from starting when more free nodes become available.

A pending reason can be one of the following 3 types:

Type 1: allows a job to start when new free nodes become available

Type 2: prevents a job from starting on particular nodes only

Type 3: prevents a job from starting anywhere

Each pending reason has a text associated with it. The text is usually printed by the workload manager job statistics utilities. The list of pending reasons texts of types 1 and 2 can be found in the pending reasons exclude file, `/cm/local/apps/cm-scale/lib/python3.7/site-packages/cmscale/trackers/hpc_queue/pending_reasons/WLM.exclude`, where WLM is a name of workload manager specified in the configuration of the engine in `ScaleServer` role.

In the pending reasons exclude file, the pending reason texts are listed as one reason per line. The reasons are grouped in two sublists, with headers:

- `[IGNORE_ALWAYS]`
- `[IGNORE_NO_NODE]`

The `[IGNORE_ALWAYS]` sublist lists the type 1 pending reason texts. If a job has only this group of reasons, then `cm-scale` considers the job as ready to start, and attempts to create or boot compute nodes for it.

The `[IGNORE_NO_NODE]` sublist lists the type 2 pending reason texts. If the reason does not specify the hostname of a new free node at the end of a pending reason after the colon (":"), then the job can start on the node. If the reason does specify the hostname of a new free node after the colon, and if the hostname is owned by one of the managed nodes—nodes that can be stopped/started/created by `cm-scale`—then the job is considered as one that is not to start, when nodes become available.

If a job has a pending reason text that is not in the pending reasons exclude file, then it is assumed to be a type 3 reason. New free nodes for such a job do not get the job started.

If there are several pending reason texts for a job, then `cm-scale` checks all the pending reasons one by one. If all reasons are from the `IGNORE_ALWAYS` or `IGNORE_NO_NODE` sublists, and if a pending reason text matched in the `IGNORE_NO_NODE` sublist does not include hostnames for the managed nodes, only then will the job be considered as one that can be started just with new nodes.

Custom Pending Reasons

If the workload manager supports them, then custom pending reason texts are also supported. The administrator can add a pending reason text to one of the sections in the pending reasons exclude file.

The `cm-scale` service checks only if the pending reason text for the job starts with a text from the pending reasons file. It is therefore enough to specify just a part of the text of the reason in order to make `cm-scale` take it into account. Regular expressions are also supported. For example, the next two pending reason expressions are equivalent when used to match the pending reason text `Not enough job slot(s)`:

Example

- `Not enough`
- `Not enough [a-z]* slot(s)`

The workload manager statistics utility can be used to find out what custom pending reason texts there are, and to add them to the pending reasons file. To do this, some test job can be forced to have such a pending reason, and the output of the job statistics utility can then be copy-pasted. For example, LSF shows custom pending reasons that look like this:

Example

```
Customized pending reason number <integer>
```

Here, `<integer>` is an identifier (an unsigned integer) for the pending reason, as defined by the administrator.

8.4.3 Locations

Sometimes it makes sense to restrict the workload manager to run jobs only on a defined subset of nodes. For example, if a user submits a multi-node job, then it is typically better to run all the job processes either on the on-premises nodes, or on the cloud nodes. That is, without mixing the node types used for the job. The *locations* feature of *cm-scale* allows this kind of restriction for HPC workload managers.

The *cm-scale* configuration allows one of these two modes to be selected:

1. *forced location*: when the workload is forced to use one of the locations chosen by *cm-scale*,
2. *unforced location*: when workloads are free to run on any of the compute nodes that are already managed (running, freed or started) by *cm-scale*.

In Bright Cluster Manager 9.0, for a forced location, *cm-scale* supports these three different locations:

1. *local*: on-premises nodes,
2. *cloud*: AWS instances (Chapter 3 of the *Cloudbursting Manual*) or Azure instances (Chapter 5 of the *Cloudbursting Manual*)
3. *virtual*: OpenStack Bright-managed virtual nodes (Chapter 5.3 of the *OpenStack Deployment Manual*)

The forced location mode is selected by setting the *Mix Locations* advanced setting in the *scaleserver* role for the node to *yes*. For example, for the head node:

Example

```
[bright90->device[bright90]->roles[scaleserver]->advancedsettings]% set mixlocations yes
[bright90->device*[bright90*]->roles*[scaleserver*]->advancedsettings*]% commit
```

For UGE, LSF, and PBS Pro, the compute nodes must already have their configuration defined. These workload managers are therefore not compatible in Bright Cluster Manager 9.0 with the *cm-scale* dynamic resource node provider when using *Mix Locations*, because the nodes could be removed and added automatically, and thus lose the location resource assignment. Bright Cluster Manager 9.0 does not support automatic assignment of the custom resources to the nodes of the mentioned workload managers. For now, only a static resource provider can be used with those workload managers when *Mix Locations* is set to *no*. For Slurm this restriction is not applied.

Slurm

Slurm does not allow the assignment of node properties (features, in Slurm terminology) to jobs if no node exists that is labelled by this property. Thus all properties used must be added to some node. This can be the template node in the case of a dynamic resource provider, or it can be an appropriate off-premises node in case of static resource provider use.

For example, in order to assign the *local* location feature to all nodes in the *default* category, the following *cmsh* commands can be executed:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category roles default
[bright90->category[default]->roles]% use slurmclient
[bright90->category[default]->roles[slurmclient]]% append features local
[bright90->category*[default*]->roles*[slurmclient*]]% commit
```


PBS Pro

A new generic resource, `resources_available.location`, lets the administrator use cm-scale to restrict PBS Pro or PBS Pro Community Edition jobs. The resource is added as a string, using the node customization list in the client roles (section 7.11):

Example

```
[root@bright90 ~]# cmsh
[bright90]% category roles default
...->roles]% use pbsproclient
...->roles[pbsproclient]]% nodecustomizations
...->roles[pbsproclient]->nodecustomizations]% add resources_available.location
...->roles*[pbsproclient*]->nodecustomizations*[location*]]% set value <location>
...->roles*[pbsproclient*]->nodecustomizations*[location*]]% commit
...->roles[pbsproclient]->nodecustomizations[location]]%
```

Here `<location>` is one of the three locations supported by cm-scale.

The current location value can be view with the `qmgr` command as follows:

Example

```
qmgr -c "print node node001" | grep location
```

UGE

A new generic resource, `location`, lets the administrator use cm-scale to restrict UGE jobs. The resource is added as a string type complex attribute, using the node customization list in the client roles (section 7.11):

Example

```
[root@bright90 ~]# cmsh
[bright90]% category roles default
...->roles]% use ugeclient
...->roles[ugeclient]]% nodecustomizations
...->roles[ugeclient]->nodecustomizations]% add location
...->roles*[ugeclient*]->nodecustomizations*[location*]]% set value <location>
...->roles*[ugeclient*]->nodecustomizations*[location*]]% commit
...->roles[ugeclient]->nodecustomizations[location]]%
```

Here `<location>` is one of the three locations supported by cm-scale.

LSF

In order to allow cm-scale to restrict LSFjobs, the administrator has to configure a generic resource called `location`. The resource is added as dynamic string resource using the role node customization (section 7.11), for instance:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category roles default
...->roles]% use ugeclient
...->roles[ugeclient]]% nodecustomizations
...->roles[ugeclient]->nodecustomizations]% add location
...->roles*[ugeclient*]->nodecustomizations*[location*]]% set value <location>
...->roles*[ugeclient*]->nodecustomizations*[location*]]% commit
...->roles[ugeclient]->nodecustomizations[location]]%
```

Here *<location>* is one of the three locations supported by `cm-scale`.

To verify that the resource is added, the following commands can be run:

Example

```
[root@bright90 ~]# module load lsf
[root@bright90 ~]# lsinfo | grep location
```

8.4.4 Azure Storage Accounts Assignment

If an Azure node is cloned manually from some node or node template, then the Azure node gets the same storage account as the node it has been cloned from. This may slow the nodes down if too many nodes use the same storage account. The `cm-scale` utility can therefore assign different storage accounts to nodes that are cloned like this.

The maximum number of nodes for such a storage account is defined by the `AZURE_DISK_ACCOUNT_NODES` parameter. This parameter has a value of 20 by default, and can be changed in the configuration file `/cm/local/apps/cm-scale/lib/python3.7/site-packages/cmscale/config.py`. The `cm-scale` utility must be restarted after the change.

The newly-cloned-by-`cm-scale` Azure node gets a randomly-generated storage account name if other storage accounts already have enough nodes associated with them. That is, if other storage accounts have `AZURE_DISK_ACCOUNT_NODES` or more nodes.

The storage account name is assigned in the node cloud settings in storage submode. For example, in `cmsh`, the assigned storage accounts can be viewed as follows:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use cnode001
[bright90->device[cnode001]]% cloudsettings
[bright90->device[cnode001]->cloudsettings]% storage
[bright90->...[cnode001]->cloudsettings->storage]% get root-disk storageaccountname
azurepaclogzjus1
[bright90->...[cnode001]->cloudsettings->storage]% get node-installer-disk storageaccountname
azurepaclogzjus1
[bright90->device[cnode001]->cloudsettings->storage]% ..
[bright90->device[cnode001]->cloudsettings]% get bootdiagnosticsstorageaccountname
azurepaclogzjus1
[bright90->device[cnode001]->cloudsettings]%
```

If a node is terminated and removed from the Bright configuration, then the storage account remains in Azure. It has to be explicitly manually removed by the administrator.

8.4.5 Mapping HPC Jobs To Particular Nodes

By default, `cm-scale` assumes that an HPC job submitted to a particular queue can take a node from outside the queue. This is because by assigning a category, or moving the node to a configuration overlay, the node will be moved to the appropriate queue eventually. From this point of view, the nodes form a single resource pool, and the nodes in the pool are repurposed on demand.

In some scenarios there is a need for certain types of HPC jobs run only on particular types of nodes, without the nodes being repurposed. A typical example: jobs with GPU code require cloud nodes that have access to GPU accelerators, while jobs that do not have GPU code can use the less expensive non-GPU cloud nodes. For this case then, the GPU cloud node is started when the GPU job requires a node, and otherwise a non-GPU node is started.

Job segregation is achieved in `cm-scale` as follows:

1. The `Pin Queues` setting, which is an advanced setting in the `scaleserver` role for the node, is enabled:

```
[bright90->device[bright90]->roles[scaleserver]->advancedsettings]% set pinqueues yes
[bright90->device*[bright90*]->roles*[scaleserver*]->advancedsettings*]% commit
```

2. A new queue is created, or an existing one is used. The queue is used for the jobs that require a particular node type.
3. The particular node type is added to this queue. If the node is already defined in Bright Cluster Manager, then the administrator can assign the queue to the node in the workload manager client role. For example, if the workload manager is Slurm, then the queue is assigned to the nodes in the `slurmclient` role. If the node has not been defined yet and will be cloned on demand (according to the dynamic resource provider settings, page 320), then its template node is assigned to the queue. When a new node is cloned from the template, the queue is then inherited from the template node.
4. The previous two steps are repeated for each job type.

After that, if a user submits a job to one of the queues, then `cm-scale` starts or clones a node that is linked with the job queue.

The following `cmsh` session snippet shows a configuration example:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device roles master
[bright90->device[bright90]->roles]% use scaleserver
[bright90->...roles[scaleserver]]% resourceproviders
[bright90->...roles[scaleserver]->resourceproviders]% add dynamic rp1
[bright90->...roles[scaleserver]->resourceproviders*[rp1*]]% set templatename tnode1
[bright90->...roles[scaleserver]->resourceproviders*[rp1*]]% set noderange cnode001..cnode100
[bright90->...roles[scaleserver]->resourceproviders*[rp1*]]% commit
[bright90->...roles[scaleserver]->resourceproviders[rp1]]% clone rp2
[bright90->...roles[scaleserver]->resourceproviders*[rp2*]]% set templatename tnode2
[bright90->...roles[scaleserver]->resourceproviders*[rp2*]]% set noderange cnode101..cnode200
[bright90->...roles[scaleserver]->resourceproviders*[rp2*]]% commit
[bright90->...roles[scaleserver]->resourceproviders[rp2]]% ...
[bright90->...roles[scaleserver]]% engines
[bright90->...roles[scaleserver]->engines]% add hpc s1
[bright90->...roles[scaleserver]->engines*[e1*]]% set workloadmanager slurm
[bright90->...roles[scaleserver]->engines*[e1*]]% trackers
[bright90->...roles[scaleserver]->engines*[e1*]->trackers]]% add queue tr1
[bright90->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% set queue q1
[bright90->...roles[scaleserver]->engines*[e1*]->trackers*[tr1*]]% commit
[bright90->...roles[scaleserver]->engines[e1]->trackers[tr1]]% clone tr2
[bright90->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% set queue q2
[bright90->...roles[scaleserver]->engines*[e1*]->trackers*[tr2*]]% commit
[bright90->...roles[scaleserver]->engines[e1]->trackers[tr2]]% category
[bright90->category]% clone default cat1
[bright90->category*[cat1*]]% roles
[bright90->category*[cat1*]->roles*]% assign slurmclient
[bright90->category*[cat1*]->roles*[slurmclient*]]% set queues q1
[bright90->category*[cat1*]->roles*[slurmclient*]]% commit
[bright90->category[cat1]->roles[slurmclient]]% category clone cat1 cat2
[bright90->category*[cat2*]->roles*[slurmclient*]]% set queues q2
[bright90->category*[cat2*]->roles*[slurmclient*]]% commit
[bright90->category[cat2]->roles[slurmclient]]% device use tnode1
[bright90->device[tnode1]]% set category cat1
[bright90->device*[tnode1*]]% commit
```

```
[bright90->device[tnode1]]% device use tnode2
[bright90->device[tnode2]]% set category cat2
[bright90->device*[tnode2*]]% commit
[bright90->device[tnode2]]%
```

Using the preceding configuration, the user may submit a job with a regular workload manager submission utility specifying the queue q1 or q2, depending on whether the job requires nodes that should be cloned from tnode1 or from tnode2.

8.4.6 How To Exclude Unused Nodes From Being Stopped

If a node is idle, then by default `cm-scale` automatically stops or terminates the node.

However, in some cases there may be a need to start a node on demand, and when it becomes idle, there may be a need to keep the node running. This can be useful if the administrator would like to investigate the performance of an application, or to debug some issues. After completing the investigation or debug session, the administrator can stop the node manually.

The parameter `KEEP_RUNNING_RANGES` keeps such nodes from being stopped or terminated. The parameter should be added to the configuration file `/cm/local/apps/cm-scale/lib/python3.7/site-packages/cmscale/config.py`. To have the changed setting take effect, the `cm-scale` service must be restarted.

`KEEP_RUNNING_RANGES` defines a map of resource provider names to node name ranges.

Extra nodes can be added to the range of the nodes. However, if the extra node must not be stopped or terminated by `cm-scale`, then for each resource provider that has such an extra node, the value of `extranodestop` must be set to `yes`.

In the following example, nodes `cnode002`, `cnode003`, `cnode004`, and `cnode010`, are associated with the `azurenodes1` resource provider. They are therefore never stopped or terminated by `cm-scale`. They are only started on demand by `cm-scale`.

The nodes `cnode012` and `cnode014` are associated with the `azurenodes2` resource provider. They are therefore also not stopped or terminated by `cm-scale`.

Example

```
opts = {
  [...]
  "KEEP_RUNNING_RANGES": {
    "azurenodes1": "cnode002..cnode004,cnode010",
    "azurenodes2": "cnode012,cnode014"
  }
}
```

8.4.7 Queue Node Placeholders

A queue node placeholder is a node that does not yet exist, but has a corresponding object that exists, and the object has queues defined, amongst other properties. It can be used to plan resource use.

Job Rejection For Exceeding Total Cluster Resources

At the time of job submission, the workload manager checks the total available number of slots (used and unused) in a queue. This is the sum of the available slots (used and unused) provided by each node in that queue.

- Jobs that require less than the total number of slots are normally made to wait until more slots become available.
- Jobs that require more than this total number of slots are normally rejected outright by the workload manager, without being put into a wait state. This is because workload managers normally follow a logic that relies on the assumption that if the job demands more slots than can exist on the

cluster as it is configured at present, then the cluster will never have enough slots to allow a job to run.

Assuming The Resources Can Never Be Provided

The latter assumption, that a cluster will never have enough slots to allow a job to run, is not true when the number of slots is dynamic, as is the case when `cm-scale` is used. When `cm-scale` starts up nodes, it adds them to a job queue, and the workload manager is automatically configured to allow users to submit jobs to the enlarged queue. That is, the newly available slots are configured as soon as possible so that waiting jobs are dealt with as soon as possible. For jobs that have already been rejected, and are not waiting, this is irrelevant, and users would have to submit the jobs once again.

Ideally, in this case, the workload manager should be configured to know about the number of nodes and slots that can be started up in the future, even if they do not exist yet. Based on that, jobs that would normally be rejected, could then also get told to wait until the resources are available, if it turns out that configured future resources will be enough to run the job.

Slurm Resources Planning With Placeholders:

Slurm allows nodes that do not exist yet to be defined. These are nodes with hostnames that do not resolve, and have the Slurm setting of `state=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
- `templatenode`: a template node that is used to provide user properties taken from its `slurmclient` role when new fake nodes are added.

For example, the following `cmsh` session uses the head node with an existing `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@bright90 ~]# scontrol show part defq | grep "Nodes="
Nodes=node001
[root@bright90 ~]# cmsh
[bright90]% device roles master
[bright90->device[bright90]->roles]% use slurmserver
[bright90->...]->roles[slurmserver]]% placeholders
[bright90->...mserver]->placeholders]% add defq
[bright90->...*]->placeholders*[defq*]]% set maxnodes 32
[bright90->...*]->placeholders*[defq*]]% set basenodename placeholder
[bright90->...*]->placeholders*[defq*]]% set templatenode node001
[bright90->...*]->placeholders*[defq*]]% commit
[bright90->...*]->placeholders[defq]]%
[root@bright90 ~]# scontrol show part defq | grep "Nodes="
Nodes=node001,placeholder[01-31]
```

If a new real node is added to the queue, then the number of placeholder nodes is decreased by one.

The placeholders can also be configured in Bright View via the Slurm server role options. For example, for a node node001, using the clickpath:

Devices→Nodes [node001]→Edit→Settings→Roles→Slurm server role
→Placeholders

9

Containerization

Containerization is a technology that allows processes to be isolated by combining *cgroups*, *Linux namespaces*, and (*container*) *images*.

- Cgroups are described in section 7.10.
- Linux namespaces represent independent spaces for different operating system facilities: process IDs, network interfaces, mount points, inter-process communication resources and others. Such cgroups and namespaces allow processes to be isolated from each other by separating the available resources as much as possible.
- A container image is a component of a container, and is a file that contains one or several layers. The layers cannot be altered as far the container is concerned, and a snapshot of the image can be used for other containers. A union file system is used to combine these layers into a single image. Union file systems allow files and directories of separate file systems to be transparently overlaid, forming a single coherent file system.

Cgroups, namespaces and image are the basis of a container. When the container is created, then a new process can be started within the container. Containerized processes running on a single machine all share the same operating system kernel, so they start instantly. No process is allowed to change the layers of the image. All changes are applied on a temporary layer created on top of the image, and these changes destroyed when the container is removed.

There are several ways to manage the containers, but the most powerful approaches use Docker, also known as Docker Engine (section 9.1), and Kubernetes (section 9.3). Docker manages containers on individual hosts, while Kubernetes manages containers across a cluster. 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.

9.1 Docker Engine

Docker integration with Bright Cluster Manager 9.0 for Docker version 19.03.5 is available at the time of writing of this paragraph (March 2020) on the x86_64 architecture for RHEL/SL/CentOS versions 7.x, for SLES versions 12 and above, and for Ubuntu 16.04 and above. For a more up-to-date status, the features matrix at <https://support.brightcomputing.com/feature-matrix/> can be checked.

Docker Engine (or just Docker) is a tool for container management. Docker allows containers and their images to be created, controlled, and monitored on a host using Docker command line tools or the Docker API.

Swarm mode, which allows containers to spawn on several hosts, is not formally supported by Bright Cluster Manager 9.0. This is because Bright Cluster Manager 9.0 provides Kubernetes for this purpose instead.

Docker provides a utility called `docker`, and two daemons called `dockerd` and `containerd`.

Additional functionality includes pulling the container image from a specific image registry (section 9.2), configuring the container network, setting `systemd` limits, and attaching volumes.

9.1.1 Docker Setup

Bright Cluster Manager provides the `cm-docker` package. The package includes the following components:

- `docker` itself, that provides API and delegates the container management to `containerd`;
- `containerd` runtime, that manages OCI images and OCI containers (via `runC`);
- `runC`, a CLI tool for spawning and running containers according to the OCI specification runtime;
- `docker-py`, a Python library for the Docker API.

Typically, however, the administrator is expected to simply run the `cm-docker-setup` utility, which is provided by Bright Cluster Manager's `cm-setup` package. Running `cm-docker-setup` takes care of the installation of the `cm-docker` package and also takes care of Docker setup. If run without options the utility starts up an Ncurses dialog (figure 9.1).

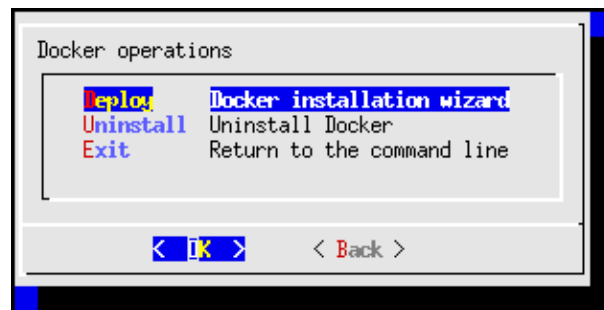


Figure 9.1: `cm-docker-setup` Ncurses startup

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 back end to configure, and so on. If `cm-docker-setup` is used with the `-c` option, and given a YAML configuration file `<YAMLfile>`, then a runtime configuration is loaded from that file. The YAML file is typically generated and saved from an earlier run.

When the questions in the Ncurses dialog have been answered and the deployment is carried out, the utility:

- installs the `cm-docker` package, if it has not been installed yet
- then assigns the `DockerHost` role to the node categories or head nodes that were specified
- adds health checks to the Bright Cluster Manager monitoring configuration
- performs the initial configuration of Docker.

The regular nodes on which Docker is to run, are restarted by the utility, if needed. The restart operation provisions the updated images from the image directory onto the nodes.

The `cm-docker` package also includes a module (section 2.2) `file, docker`, which must be loaded in order to use the `docker` command. By default only the administrator can run the `docker` commands after setup (some output ellipsized):

Example


```
[root@bright90 ~]# ssh node001
[root@node001 ~]# module load docker
[root@node001 ~]# docker info
Containers: 0
Images: 0
...
Docker Root Dir: /var/lib/docker
[root@node001 ~]#
```

and the hello-world image can be run as usual with:

Example

```
[root@node001 ~]# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Or, for example, importing and running Apache containers with Docker may result in the following output:

Example

```
[root@node001 ~]# module load docker
[root@node001 ~]# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS          NAMES
62071069436f   httpd    "httpd-foreground"     4 hours ago   Up 4 hours   80/tcp        elated_euler
3310d85b17f4   httpd    "httpd-foreground"     4 hours ago   Up 4 hours   80/tcp        sleepy_lichterman
35387f3cc606   httpd    "httpd-foreground"     4 hours ago   Up 4 hours   80/tcp        wonderful_wiles
...
[root@node001 ~]#
```

Using Docker directly means being root on the host. It is rarely sensible to carry out regular user actions as the root user at all times.

So, to make Docker available to regular users, a user management layer and restrictions are provided by Kubernetes.

After Docker has been installed, Kubernetes can be set up to allow regular user access to the Docker containers as covered in section 9.3. It is a best practice for regular users to use Kubernetes instead of Docker commands directly.

9.1.2 Integration With Workload Managers

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.
- PBS Pro – Altair provides a hook script that allows jobs to start in Docker containers. Altair should be contacted to obtain the script and instructions.

9.1.3 DockerHost Role

When `cm-docker-setup` is executed, the DockerHost role is assigned to nodes or categories. The DockerHost role is responsible for Docker service management and configuration.

From `cmsh`, the configuration parameters can be managed from the `Docker::Host` role:

Example

```
[root@bright90 ~]# cmsd
[bright90]% category use default
[bright90->category[default]]% roles
[bright90->category[default]->roles]% assign docker::host
[bright90->category*[default*]->roles*[Docker::Host*]]% show
Parameter                                Value
-----
Add services                              yes
Name                                       Docker::Host
Provisioning associations                  <0 internally used>
Revision
Type                                       DockerHostRole
Debug                                     no
Default Ulimits
Enable SELinux                            yes
Log Level                                 info
Spool                                     /var/lib/docker
Tmp dir                                   $spool/tmp
API Sockets                               unix:///var/run/docker.sock
Bridge
Bridge IP
Iptables                                  yes
MTU                                       0
User Namespace Remap
Insecure Registries
Certificates Path                          /etc/docker
Enable TLS                                 no
TLS CA
TLS Certificate
TLS Key
Verify TLS                                no
Storage Backends                          <0 in submode>
Containerd Socket
Runtime                                   runc
Options
[bright90->category*[default*]->roles*[Docker::Host*]]%
```

The parameters that CMDaemon can configure in the DockerHost role, along with a description, are shown in table 9.1:

Parameter	Description
Add services*	Add services to nodes belonging to this node. Care must be taken if setting this to no. (default: yes)
Debug*	Enable debug mode (default: no)

...continues

...continued

Parameter	Description
Default Ulimits	Set the default ulimit options for all containers
Enable SELinux*	Enable selinux support in Docker daemon (default: yes)
Log Level	Set the daemon logging level. In order of increasing verbosity: fatal, error, warn, info, debug. (default: info)
Spool	Root of the Docker runtime (default: /var/lib/docker)
Tmp dir	Location for temporary files. Default: \$<spool>/tmp, where \$<spool> is replaced by the path to the Docker runtime root directory
API Sockets	Daemon socket(s) to connect to (default: unix:///var/run/docker.sock)
Bridge	Attach containers to a network bridge (not defined by default)
Bridge IP	Network bridge IP (not defined by default)
Iptables*	Enable iptables rules (default: yes)
MTU	Set the containers network MTU, in bytes (default: 0, which does not set the MTU at all)
User Namespace Remap	User/Group setting for user namespaces (not defined by default). It can be set to any of <UID>, <UID:GID>, <username>, <username:groupname>. 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.
Insecure Registries	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)
Certificates Path	Path to Docker certificates (default: /etc/docker)
Enable TLS*	Use TLS (default: no)

...continues

...continued

Parameter	Description
TLS CA	Trust only certificates that are signed by this CA (not defined by default)
TLS Certificate	Path to TLS certificate file (not defined by default)
TLS Key	Path to TLS key file (not defined by default)
Verify TLS*	Use TLS and verify the remote (default: no)
Storage Backends	Docker storage back ends. Storage types can be created and managed, in a submode under this mode. The available types are described in table 9.2. Each of these storage types has options that can be set from within the submode.
Containerd Socket	Path to containerd socket (default: not used)
Runtime	Docker runtime
Options	Additional parameters for docker daemon

* Boolean (takes yes or no as a value)

Table 9.1: DockerHost role and docker options

9.1.4 Iptables

By default iptables rules have been added to nodes that function as a Docker host, to let network traffic go from the containers to outside the pods network. If this conflicts with other software that uses iptables, then this option can be disabled. For example, if the `docker::host` has already been assigned to the nodes via the `default` category, then the iptables rules that are set can be disabled by setting the `iptables` parameter in the `Docker::Host` role to `no`:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category use default
[bright90->category[default]]% roles
[bright90->category[default]->roles]% use docker::host
[bright90->category[default]->roles[Docker::Host]]% set iptables no
[bright90->category*[default*]->roles*[Docker::Host*]]% commit
```

9.1.5 Storage Backends

A core part of the Docker model is the efficient use of containers based on layered images. To implement this Docker provides different storage back ends, also called storage drivers. These storage back ends rely heavily on various filesystem features in the kernel or volume manager. Some storage back ends perform better than others, depending on the circumstances.

The default storage back end configured by `cm-docker-setup` is `overlay2`. Storage back ends supported by Docker are listed in table 9.2:

Technology	Description	Backend Name
OverlayFS	This is a modern union filesystem. It is the preferred storage driver for recent Docker versions. It has been in the mainline Linux kernel since version 3.18, with additional improvements for Docker in version 4.0. All of the distributions that Bright Cluster Manager 8.2 supports have backported the kernel changes needed for this to work.	overlay2
Device Mapper	<p>Deprecated in Docker Engine 18.09. It is a kernel-based framework that has been included in the mainline Linux kernel since version 2.6.9. It underpins many advanced volume management technologies on Linux. The driver stores every image and snapshot on its own virtual device, and works at the block level rather than the file level.</p> <ul style="list-style-type: none"> • A loopback mechanism can be implemented using <code>loop-lvm</code> mode. This allows files on a local disk to be managed as if they are on a physical disk or block device. This is simpler than the thin pool mode, but is strongly discouraged for production use. In Bright Cluster Manager this mode is implemented by selecting the option <code>loopback (testing only)</code>. This is selected in the storage back end selection screen of the <code>cm-docker-setup</code> installation. • A thin pool mode can be implemented using <code>direct-lvm</code> mode. This uses a logical volume as a thin pool to use as backing for the storage pool, and uses a spare block device. Configuring this is normally more involved. In Bright Cluster Manager this mode is implemented by selecting the option <code>block (production ready)</code>. This is selected in the storage back end selection screen of the <code>cm-docker-setup</code> installation session. <p>Device mapper options for these modes are described in Table 9.3</p>	devicemapper

...continues

...continued

Technology	Description	Backend Name
AUFS	This was the first storage back end that Docker used. AUFS is not included in the mainline Linux kernel. Out of the distributions that Bright Cluster Manager 8.2 supports, it is only Ubuntu that supports it.	aufs

Table 9.2: Docker storage back ends

The `docker info` command, amongst many other items, shows the storage driver and related settings that are being used in Docker:

Example

```
[root@bright90 ~]# module load docker
[root@bright90 ~]# docker info
Containers: 29
  Running: 18
  Paused: 0
  Stopped: 11
Images: 3
Server Version: 17.03.2-ce
Storage Driver: devicemapper
  Pool Name: docker-253:17-17199918-pool
  Pool Blocksize: 65.54 kB
  Base Device Size: 10.74 GB
  Backing Filesystem: xfs
  Data file: /dev/loop0
  Metadata file: /dev/loop1
  Data Space Used: 1.296 GB
  Data Space Total: 107.4 GB
  Data Space Available: 4.56 GB
  Metadata Space Used: 4.325 MB
  Metadata Space Total: 2.147 GB
  Metadata Space Available: 2.143 GB
  Thin Pool Minimum Free Space: 10.74 GB
  Udev Sync Supported: true
  Deferred Removal Enabled: false
  Deferred Deletion Enabled: false
  Deferred Deleted Device Count: 0
  Data loop file: /var/lib/docker/devicemapper/devicemapper/data
  Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
  Library Version: 1.02.146-RHEL7 (2018-01-22)
...
```

Docker data volumes are not controlled by the storage driver. Reads and writes to data volumes bypass the storage driver. It is possible to mount any number of data volumes into a container. Multiple containers can also share one or more data volumes.

More information about Docker storage back ends is available at <https://docs.docker.com/engine/userguide/storagedriver>.

Device Mapper Driver Settings Support

Bright Cluster Manager supports device mapper driver settings more explicitly than the other driver back end settings.

By default the device mapper storage back end is added automatically, and can be configured in the `storagebackends` submode of the `DockerHost` role:

Example

```
[bright90->device [bright90]->roles [dockerhost]]% storagebackends
[bright90->device [bright90]->roles [dockerhost]->storagebackends]% use devicemapper
[bright90->device [bright90]->roles [dockerhost]->storagebackends [devicemapper]]% show
Parameter                               Value
-----
Blk Discard                             yes
Block Size                               64K
Filesystem                               xfs
Loop Data Size                           100GB
Loop Device Size                          2GB
Loop Metadata Size                        2GB
Mkfs Arguments
Mount Options
Name                                      devicemapper
Pool Device
Type                                      DockerStorageDeviceMapperBackend
```

The parameters that are used in the Docker device mapper back end are described in table 9.3:

Parameter	Description	Option to docker
Blk Discard*	Enables or disables the use of blkdiscard when removing device mapper devices (default: yes)	dm.blkdiscard
Block Size	Custom blocksize to use for the thin pool (default: 64kB)	dm.blocksize
Filesystem	Filesystem type to use for the base device (default: xfs)	dm.fs
Loop Data Size	Size to use when creating the loopback file for the data virtual device which is used for the thin pool (default: 100GB)	dm.loopdatasize
Loop Device Size	Size to use when creating the base device, which limits the size of images and container (not set by default)	dm.basesize
Loop Metadata Size	Size to use when creating the loopback file for the metadata device which is used for the thin pool (default: 2GB)	dm.loopmetadatasize
Mkfs Arguments	Extra mkfs arguments to be used when creating the base device	dm.mkfsarg
Mount Options	Extra mount options used when mounting the thin devices	dm.mountopt

...continues

...continued

Parameter	Description	Option to docker
Pool Device	Custom block storage device to use for the thin pool (not set by default)	dm.thinpooldev

* Boolean (takes yes or no as a value)

Table 9.3: Device mapper back end Docker options

For back end driver storage settings other than device mapper, such as AUFS or OverlayFS, settings can be added as options if needed. In `cmsh` this can be done by setting the `options` parameter in the `storagebackend` submode under the `docker::host` role.

9.1.6 Docker Monitoring

When `cm-docker-setup` runs, it configures and runs the following Docker health checks:

1. makes a test API call to the endpoint of the Docker daemon
2. checks containers to see that none is in a dead state

The Docker daemon can be monitored outside of Bright Cluster Manager with the usual commands directly.

Bright Cluster Manager ways to manage or check on Docker include the following:

In `CMDaemon`, the `docker` service can be checked:

Example

```
[bright90->device[node001]->services]% list
Service (key)           Monitored  Autostart
-----
docker                  yes       yes
nslcd                   yes       yes
[bright90->device[node001]->services]% show docker
Parameter              Value
-----
Autostart               yes
Belongs to role        yes
Monitored               yes
Revision
Run if                  ALWAYS
Service                 docker
Sickness check interval 60
Sickness check script
Sickness check script timeout 10
Timeout                 -1
```

The `docker0` interface statistics can be checked within the `nodeoverview` output:

Example

```
[bright90->device[node001]]% nodeoverview
...

Interface  Received  Transmitted
-----
docker0    16.0 KiB  3.16 KiB
eth0       492 MiB  72.5 MiB
eth1       0 B      0 B
...
```


The measurable Docker checks if the docker service is healthy.

Example

```
[bright90->device[node001]]% dumpmonitoringdata -1h now Docker
Timestamp          Value      Info
-----
2019/03/19 17:25:17.249  PASS
2019/03/22 14:59:17.248  PASS
```

9.1.7 Docker Setup For NVIDIA

NVIDIA GPU Cloud (NGC) is a cloud platform that runs on NVIDIA GPUs. NGC containers are lightweight containers that run applications on NVIDIA GPUs. The applications are typically HPC, machine learning, or deep learning applications.

An NGC can be set up to run NGC containers from the registry <http://ngc.nvidia.com>.

Docker can be configured as an NGC running NGC containers by using the NVIDIA Container Toolkit.

The Bright Cluster Manager package provided for this is: `cm-nvidia-container-toolkit`. This package can be installed via the package manager, `yum` or `apt` after Docker has been set up (section 9.1.1). The toolkit has to be running on the compute nodes that have GPUs. The installation therefore must go to the appropriate node image (section 11.4). For example, if the appropriate image is `gpu-image`:

Example

```
# yum install --installroot=/cm/images/gpu-image cm-nvidia-container-toolkit
```

The nodes that use that GPU image can then be rebooted to pick up the new package.

The GPU status can then be printed with the NVIDIA system management interface command. For example, if the image has been picked up by `node001`:

Example

```
[root@bright90 ~]# ssh node001
Last login: Tue Apr  7 19:23:09 2020 from bright90.cm.cluster
[root@node001 ~]# module load docker/19.03.5
[root@node001 ~]# docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
Unable to find image 'nvidia/cuda:latest' locally
latest: Pulling from nvidia/cuda
7ddbc47eeb70: Pull complete
...
30b4effda4fd: Pull complete
Digest: sha256:31e2a1ca7b0e1f678fb1dd0c985b4223273f7c0f3dbde60053b371e2a1aee2cd
Status: Downloaded newer image for nvidia/cuda:latest
Tue Apr  7 17:30:48 2020
+-----+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2    |
|-----+-----+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+
|  0   Tesla K40c        On         | 00000000:00:06.0 Off  |                    Off |
| 23%   32C    P8      22W / 235W |  0MiB / 12206MiB |    0%      Default |
+-----+-----+-----+-----+
+-----+
| Processes:                                                       GPU Memory |
```

```

| GPU      PID   Type   Process name           Usage           |
|-----|-----|-----|-----|-----|-----|
| No running processes found |
+-----+-----+-----+-----+-----+-----+
[root@node001 ~]# logout
Connection to node001 closed.

```

How to run applications in an NGC with Bright Cluster Manager is illustrated for PyTorch, TensorFlow, and MXNet, in the white paper at <https://www.brightcomputing.com/hubfs/docs/Bright-NGC-WPP-v01.pdf>. The sessions illustrated in that paper require the installation of Kubernetes (section 9.3), or Singularity (section 9.4).

9.2 Docker Registries

When a user creates a new container, an image specified by the user should be used. The images are kept either locally on a host, or in a registry. The image registry can be provided by a cloud provider or locally.

9.2.1 Docker And Harbor Registries: Introduction

Docker Hub, A Remote Registry

By default, Docker searches for images in Docker Hub, which is a cloud-hosted public and private image registry. Docker Hub serves a huge collection of existing images that users can make use of. Every user is allowed to create a new account, and to upload and share images with other users. Using the Docker client, a user can search for already-published images, and then pull them down to a host in order to build containers from them.

When an image is found in the registry, 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 Options: Classic Docker Registry Vs Harbor

Besides using Docker Hub for the image registry, the administrator can also install a local image registry on the cluster. Bright Cluster Manager provides two ways to integrate such a local registry with the cluster, based on existing open source projects:

- The first one is the classic docker registry provided by Docker Inc, and can be useful if the registry is used by trusted users.
- The second registry, Harbor, developed by VMware and introduced in Bright Cluster Manager version 8.1-5, provides additional features such as security and identity management, and is aimed at the enterprise.

Both options can be installed with the `cm-docker-registry-setup` utility, which comes with Bright Cluster Manager's `cm-setup` package.

9.2.2 Docker And Harbor Registries: Setup And Configuration

Docker Registry and Harbor can be installed via the `cm-docker-registry-setup` command-line utility. They can also be installed via Bright View in Bright Cluster Manager for versions beyond 8.1-6 as follows:

- The Docker Registry Deployment Wizard is launched via the clickpath:
Containers→Docker→Docker Registry Wizard

- Either Docker Registry, or Harbor, should be chosen as a registry.
- A single node is ticked for the deployment. The address, port, and the root directory for storing the container images are configured. If the head node is selected for Harbor, then the setup later on asks to open the related port on the head node. This is to make Harbor and the Harbor UI externally accessible.
- In the summary page, if the Ready for deployment box is ticked, then the administrator can go ahead with deploying the registry.
- When the deployment is complete, the Docker Registry becomes ready for use. Harbor can take a few additional minutes to be ready for use.

Similar to the case of etcd nodes (section 9.3.2), nodes that run Harbor or Docker Registry have the `datanode` option (section 5.4.4) when installed by Bright Cluster Manager utilities. The option helps prevent the registry being wiped out by accident, and is added when the `cm-docker-registry-setup` utility is used to install Harbor or Docker Registry. This extra protection is put into place because if a registry is wiped out, then the state of images in the container becomes incoherent and cannot be restored.

Harbor UI If the head node is where Harbor is to be installed, and is to be made externally accessible, then the Harbor UI can be accessed at `https://<head node hostname>:9443`.

If a different node is used for Harbor to be installed, then the related port must be forwarded locally. Harbor can be logged into by default with the `admin` user and the default `Harbor12345` password. It is recommended to change that password after the first login.

Clair Harbor comes with Clair, a tool for vulnerability static analysis of container images.

More information on using Clair with Harbor can be found at `https://github.com/vmware/harbor/blob/master/docs/user_guide.md#vulnerability-scanning-via-clair`.

Dealing with a pre-existing Kubernetes or Harbor installation: Since Harbor uses Docker internally, and because Kubernetes customizes Docker networking, it means that nodes where Kubernetes is running cannot be reused for Harbor, and that nodes where Harbor is running cannot be reused for Kubernetes.

Docker Registry Daemon Configuration Using The Docker Registry Role

The Docker Registry role is used to configure and manage the `docker-registry` daemon, and its parameters are described in table 9.4:

Parameter	Description
Domain	Main domain name (default: hostname of the node)
Alt Domains	Alternative domain names (default: FQDN of the node)
Port	Port (default: 5000)

...continues

...continued

Parameter	Description
Spool Dir	Spool directory (default: /var/lib/docker-registry)

* Boolean (takes yes or no as a value)

Table 9.4: Docker Registry role parameters

The values stored in the Docker Registry role are not supposed to be changed, but they are useful for the uninstall procedure, and also as a record of the settings for the administrator.

```
[bright90->device[bright90]->roles[generic::docker_registry]]% environments
[bright90->device[bright90]->roles[generic::docker_registry]->environments]% list
```

Name (key)	Value	Node Environment
alt_domains	node001.cm.cluster	no
domain	node001	no
port	5000	no
spool_dir	/var/lib/docker-registry	no

Further details on the docker-registry daemon can be found at <https://github.com/docker/distribution>.

Harbor Daemon Configuration Using The Harbor Role

The Harbor role is used to configure and manage the harbor daemon. The parameters of the role are described in table 9.5:

Parameter	Description
Domain	Main domain name (default: hostname of the node)
Alt Domains	Alternative domain names (default: FQDN of the node)
Port	Port (default: 9443)
Spool Dir	Spool directory (default: /var/lib/harbor)
Default Password	Default password of the Harbor admin user (default: Harbor12345)
DB Password	Password of the Harbor database (default: randomly generated)
Clair DB Password	Password of the Clair database (default: randomly generated)

...continues

...continued

Parameter	Description
-----------	-------------

* Boolean (takes yes or no as a value)

Table 9.5: Harbor role parameters

The values stored in the Harbor role are not supposed to be changed, but they are useful for the uninstall procedure, and also as reminder of the settings for the administrator.

```
[bright90->device [bright90]->roles [generic::harbor]]% environments
[bright90->device [bright90]->roles [generic::harbor]->environments]% list
```

Name (key)	Value	Node Environment
alt_domains	harbor,node001.cm.cluster	no
clair_db_password	<generated password>	no
db_password	<generated password>	no
default_password	Harbor12345	no
domain	node001	no
port	9443	no
spool_dir	/var/lib/harbor	no

Further details on Harbor can be found at <https://vmware.github.io/harbor>.

9.3 Kubernetes

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts. With Kubernetes, it is possible to:

- scale applications on the fly
- seamlessly update running services
- optimize hardware usage by using only the resources that are needed

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 <https://kubernetes.io/docs/>.

Kubernetes integration with Bright Cluster Manager 9.0 for Kubernetes v1.16 is available at the time of writing of this paragraph (March 2020) on the x86_64 architecture for RHEL/SL/CentOS versions 7.x, for SLES versions 12 and above, and for Ubuntu 16.04 and above. For a more up-to-date status, the features matrix at <https://support.brightcomputing.com/feature-matrix/> can be checked.

9.3.1 Reference Architecture

A reference architecture for Kubernetes in Bright Cluster Manager comprises:

- *etcd nodes*: An etcd cluster—the Kubernetes distributed key-value storage—runs on regular nodes only, and uses an odd number (1, 3, 5 ...) of nodes.
- *master nodes*: Kubernetes master units run on head or compute nodes. 2 (or 3) are recommended for High Availability (HA). In a Bright HA configuration, both or none of the head nodes should be selected. That is, it must not run on only one head node of an HA configuration.
- *worker nodes*: Kubernetes worker units run on regular nodes only.

Since Bright Cluster Manager version 8.2, multiple clusters of Kubernetes can be deployed. In such a configuration the same node cannot be shared across different Kubernetes clusters.

A Kubernetes API server proxy based on NGINX runs on every node, except for on nodes that run `etcd`. The proxy also runs on the head node(s).

Because of the server proxy, a port is reserved on the head node(s) for every Kubernetes cluster. This is required for Kubernetes HA (section 9.3.1), and it also allows `kubect1` and other tools such as Helm to be used from the head node, to access each Kubernetes cluster.

Kubernetes HA

For a Kubernetes HA setup the minimum node requirements are:

- at least 3 nodes for Etcd
- at least 2 nodes for Kubernetes Master

In an average cluster, Bright Computing recommends 3 nodes for the `etcd` cluster, and 3 nodes for the Kubernetes master.

Even without a Kubernetes master in an HA configuration, there is no downtime for existing pods running on the worker nodes. The worker nodes will still continue to work.

However, Kubernetes HA is needed to be able to schedule tasks, spawn new pods, and in general keep the cluster in the desired state.

9.3.2 Kubernetes Setup

Bright Cluster Manager provides the following Kubernetes-related packages:

- The `cm-kubernetes-master`, `cm-kubernetes-node`, `contrack` and `nginx` packages: These are always installed on the head node(s) and on the master and worker node(s).
- The `cm-etcd` package is installed on the nodes selected for `etcd`. In a similar way to the case of Harbor or Docker Registry (section 9.2.2), the nodes that run `etcd` are protected by Bright Cluster Manager with the `datanode` option (section 5.4.4). For `etcd` nodes, the option is added during the `cm-kubernetes-setup` installation. As in the case for the registries, the `datanode` option is set in order to help prevent the administrator from wiping out the existing state of `etcd` nodes. Wiping out the state of `etcd` nodes means that the Kubernetes cluster becomes incoherent and that it cannot be restored to where it was just before the `etcd` nodes were wiped. The `etcd` version installed by the Bright Cluster Manager package is 3.4.3.

The packages are installed automatically from the repository when the administrator runs `cm-kubernetes-setup` from the command line.

The log file produced by the setup can be found in `/var/log/cm-kubernetes-setup.log`.

Kubernetes Core Addons

During setup, some critical addon components such as CoreDNS and Calico are automatically deployed in the `kube-system` namespace. In Bright Cluster Manager all addons are treated as Kubernetes applications (section 9.5), and belong to the default app group `system`.

A `cmsh` treeview illustrating the hierarchy to access these applications is:

```
[cmsh]
|-- ...
|
|-- kubernetes[default]
|     |-- appgroups[system]
|         |-- applications
|-- ...
```

CoreDNS: The DNS server add-on for internal service discovery. It reads the IP addresses of services and pods from Etcd, and resolves domain names for them. If a domain name is not found because the domain is external to the Kubernetes cluster, then CoreDNS forwards the request to the main DNS server. Bright Cluster Manager uses CoreDNS version 1.62.

Calico: A modern SDN (Software-Defined Networking) add-on to manage a cluster-wide network for pods. Calico uses an agent called Felix to run on each node as a pod. Bright Cluster Manager uses Calico version 3.10.

When the Kubernetes cluster is composed of more than 50 nodes, the component Typha is also deployed for better scalability. The number of Typha replicas is calculated by allocating one Typha replica per 150 nodes, with a minimum of 3 (above 50 nodes) and a maximum of 20.

Further details on Calico can be found at <https://docs.projectcalico.org/>.

Kubernetes Optional Addons

The following add-ons are installed by default unless otherwise noted. However, the user can choose to skip some or all of them during the setup.

Kubernetes Dashboard: The web user interface add-on for GUI cluster administration and metrics visualization. Bright Cluster Manager uses Kubernetes Dashboard version 2.0.

There are two ways to access the dashboard:

- Using `kubectl proxy` and accessing `http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/`. To use the proxy, `kubectl` must be set up locally (section 9.3.2 of the *User Manual*).
- Users on an external network can log into `kubectl` or Kubernetes Dashboard by following the procedures described in section 9.3.13.

Kubernetes Metrics Server: This add-on is a replacement for Heapster. It aggregates metrics, and provides container monitoring and performance analysis. It exposes metrics via an API. Bright Cluster Manager runs Metrics server version 1.0.0.

Helm: An add-on that manages *charts*, which are packages of pre-configured Kubernetes resources. The Helm component is installed and properly configured with Bright Cluster Manager's Kubernetes installation by default. It is initialized and ready for use for every Kubernetes user when the Kubernetes module is loaded. Bright Cluster Manager uses Helm version 3.

NGINX Ingress Controller: The official Kubernetes ingress controller add-on is built around the Kubernetes Ingress resource, using a ConfigMap to store the NGINX configuration. Ingress provides HTTP and HTTPS routes from outside a Kubernetes cluster to services within the cluster. Traffic routing is controlled by rules defined in the Ingress resource. Bright Cluster Manager uses NGINX Ingress Controller version 0.26.

By default, Bright Cluster Manager provides an ingress for Kubernetes Dashboard during the `cm-kubernetes-setup` run, so that the Dashboard works. Port 30080 is the default that is set for the HTTP, and port 30443 is the default that is set for HTTPS.

These 2 ports are exposed on every Kubernetes node, both masters and workers.

The Ingress Controller is deployed as a `NodePort` which means it comes with a default range of possible port values of 30000–32767.

NVIDIA device plugin for Kubernetes: An add-on option in the `cm-kubernetes-setup` run. By default it is not selected. To be used, in addition to being selected, it requires that the NVIDIA GPU drivers are first installed (section 7.4 of the *Installation Manual*) on the head node and regular node container hosts.

This means that, as described in that section, the regular nodes on which the GPUs are located must have the `cuda-driver` and `cuda-dcgm` packages installed, inside the software image.

After booting up the container host regular nodes, the functioning of the DCGM (Data Center GPU Manager) tools should be checked by running `module load cuda-dcgm; dcgmi config --get`.

The plugin add-on can then be installed as a selection option in the `cm-kubernetes-setup` session that is run by the system administrator after the NVIDIA GPU drivers have been installed. The plugin is somewhat beta at the time of writing (March 2019).

If `cuda-dcgm` and `cuda-driver` are installed for GPUs on the head node, then to ensure that the head node GPUs are detected by CMDaemon, it is recommended to restart CMDaemon on the head node with `service cmdaemon restart`.

The plugin then allows the GPUs to be consumed from the containers. Overcommitting GPUs (sharing) is not possible from containers or pods. Multiple GPUs can be requested by the container. Some documentation on its use can be found at <https://github.com/NVIDIA/k8s-device-plugin>.

Bright Cluster Manager provides NVIDIA device plugin version 1.11

Kubernetes Setup From The Command Line

The `cm-kubernetes-setup` command line utility has the following usage synopsis:

```
[root@bright90 ~]# cm-kubernetes-setup -h
```

```
usage: Kubernetes Setup cm-kubernetes-setup [-v] [-h] [-c <config_file>]
      [--skip-modules <mod1,mod2,...>]
      [--only-these-modules <mod1,mod2,...>]
      [--dev]
      [--on-error-action {OnErrorAction.debug,
                        OnErrorAction.remotedebug, OnErrorAction.undo,
                        OnErrorAction.abort}]
      [--output-remote-execution-runner]
      [--json] [--no-distro-checks]
      [--skip-packages]
      [--min-reboot-timeout <reboot_timeout_seconds>]
      [--cluster CLUSTER_NAME]
      [--skip-docker] [--skip-reboot]
      [--skip-image-update]
      [--add-user USERNAME]
      [--namespace NAMESPACE]
      [--role ROLE]
      [--no-secure-namespace]
      [--remove-user USERNAME_REMOVE]
      [--regenerate-certs] [--psp]
      [--apparmor] [--disable-psp]
      [--remove]
      [--yes-i-really-mean-it]
      [--remove-ceph-storage]
      [--allow-device-mapper]
      [--allow-disksetup-cleanup]
```

optional arguments:

```
--cluster CLUSTER_NAME
                        Name of the referred Kubernetes cluster
```


common:

Common arguments

```
-v, --verbose      Verbose output
-h, --help        Print this screen
-c <config_file>  Load runtime configuration for plugins from a YAML config file
```

advanced:

Various *advanced* configuration options flags.

```
--skip-modules <mod1,mod2,...>
    Load and use all the default plugins (kubernetes, docker), except from these
    ones. Comma-separated list.
--only-these-modules <mod1,mod2,...>
    Out of all default plugins, load and use only those. Comma-separated list.
--dev
    Enables additional command line arguments
--on-error-action {debug,remotedebug,undo,abort}
    Upon encountering a critical error, instead of asking the user for choice,
    setup will undo (revert) the deployment stages.
--output-remote-execution-runner
    Format output for CMDaemon
--json
    Use json formatting for log lines printed to stdout
--no-distro-checks
    Disable distribution checks based on ds.json
--skip-packages
    Skip the any stages which install packages. Requires packages to be already
    installed.
--min-reboot-timeout <reboot_timeout_seconds>
    How long to wait for nodes to finish reboot (default and minimum allowed:
    300 seconds).
```

installing Kubernetes clusters:

Flags for installing or managing Kubernetes clusters

```
--skip-docker      Skip the Docker installation steps.
--skip-reboot      Skip the reboot steps.
--skip-image-update Skip the image update steps.
```

adding user:

Flags for adding a new user in a Kubernetes cluster

```
--add-user USERNAME Create a new user in a Kubernetes cluster
--namespace NAMESPACE
    Specify namespace for the new (--add-user) role binding (Default: default)
--role ROLE
    Specify role for the new (--add-user) role binding (Default: admin)
--no-secure-namespace
    Specify secure namespace for the new (--add-user) role binding (Default:
    disabled)
```

removing user:

Flags for removing a user from a Kubernetes cluster

```
--remove-user USERNAME_REMOVE
    Remove existing user from a Kubernetes cluster
```

regenerate Kubernetes certificates:

```

Flag for regenerating the Kubernetes certificates

--regenerate-certs    Regenerate certificates for a Kubernetes cluster

pod security policies:
  Flags for setting up Pod Security Policies

--psp                Setup PSP
--apparmor           Use AppArmor (Default: false)

disable pod security policies:
  Flags for removing Pod Security Policies

--disable-psp       Remove PSP

removing Kubernetes clusters:
  Flags for removing a Kubernetes cluster

--remove            Remove a Kubernetes cluster
--yes-i-really-mean-it
                    Required for additional safety
--remove-ceph-storage
                    Remove Kubernetes osd pool from Ceph cluster

Docker storage backend specific:
--allow-device-mapper
                    Allow to select DeviceMapper (DEPRECATED) storage in wizard

Docker removal specific:
--allow-disksetup-cleanup
                    Allow to clean up disk setup during docker removal. (DeviceMapper thin pool
only).

```

The `cm-kubernetes-setup` utility should be executed on the console.

Dealing with a pre-existing Docker installation: Docker is required for Kubernetes configured by Bright Cluster Manager. The setup wizard checks if Docker has been installed (page 350), and automatically installs Docker, if needed. However, if Docker has already been configured on the same category of nodes on which Kubernetes is to be installed, then the installation stops, because overriding the existing Docker configuration may not be what is wanted. To override the existing Docker configuration, Docker for that category should first be removed with the `cm-docker-setup --remove` command.

Dealing with a pre-existing Etcd cluster: Etcd is required by Kubernetes to store all the key-value states of the Kubernetes cluster. If no Etcd cluster is found, then the setup wizard prompts to deploy an Etcd cluster. If Etcd is already installed, or present from a previous Kubernetes cluster, then the setup wizard prompts on whether to use the existing Etcd cluster.

Dealing with a pre-existing Harbor installation: Since Harbor uses Docker internally, and because Kubernetes customizes Docker networking, it means that nodes where Harbor is running cannot be reused for Kubernetes.

Kubernetes Setup From An Ncurses Session

When the Kubernetes installation is carried out using `cm-kubernetes-setup` without any options, an Ncurses wizard starts up. The administrator can answer several questions in the wizard. Questions

that are asked include questions about which of the node categories or which of the individual nodes should be configured to run the Kubernetes services. There are also questions about the service and pod networks parameters, the port numbers that will be configured for the daemons, whether to install specific add-ons, and so on. After the wizard has been completed, a configuration file can be saved that can be used to set up Kubernetes.

The configuration file can be deployed immediately from the wizard, or it can be deployed later by specifying it as an option to `cm-kubernetes-setup`, in the form `-c <file>`.

Example

```
[root@bright90 ~]# cm-kubernetes-setup
```

ncurses session starts up:

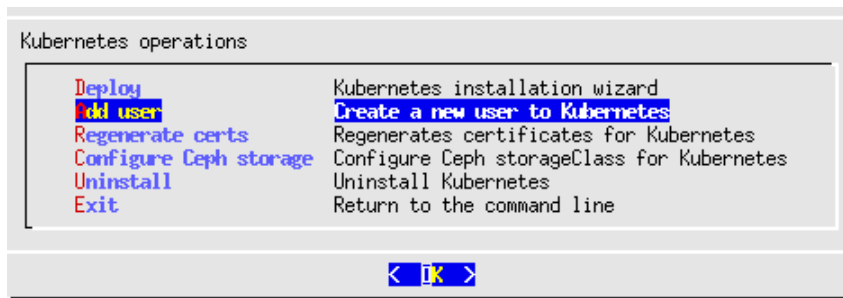


Figure 9.2: Kubernetes Ncurses Session: Main Operations Screen

The deployment via CLI or via Ncurses assigns the appropriate roles, it adds the new Kubernetes cluster, it adds health checks to the monitoring configuration, and it generates certificates for the Kubernetes daemons.

Docker and Etcd are also deployed when needed.

Testing Kubernetes

To test that Kubernetes works, the `cluster-info` command can be run. For example, on the head node, `bright90`:

Example

```
[root@bright90 ~]# module load kubernetes/default/1.16
[root@bright90 ~]# kubectl cluster-info
Kubernetes master is running at https://localhost:10443
CoreDNS is running at https://localhost:10444/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://localhost:10443/api/v1/namespaces/kube-system/services/\
https:metrics-server:/proxy
```

To further debug and diagnose cluster problems, use `'kubectl cluster-info dump'`.

After `cm-kubernetes-setup` finishes, and the regular nodes have been rebooted, the state of the nodes can be found by running the `get nodes` command:

Example

```
[root@bright90 ~]# kubectl get nodes
NAME      STATUS    ROLES    AGE   VERSION
node001   Ready    <none>   25m   v1.16
```

```
node002    Ready    <none>    25m    v1.16
node003    Ready    <none>    25m    v1.16
node004    Ready    <none>    25m    v1.16
node005    Ready    <none>    25m    v1.16
node006    Ready    <none>    25m    v1.16
```

A six node cluster should show the following Kubernetes installed addons, when using `kubectl` with the `get all -n kube-system` option (some lines truncated):

Example

```
[root@bright90 ~]# module load kubernetes/default/1.16
[root@bright90 ~]# kubectl get all -n kube-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/calico-typha	ClusterIP	10.150.160.210	<none>	5473/TCP	21m
service/kube-dns	ClusterIP	10.150.255.254	<none>	53/UDP,53/TCP,9153/TCP	31m
service/metrics-server	ClusterIP	10.150.246.100	<none>	443/TCP	33m

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR
daemonset.apps/calico-node	0	0	0	0	0	
beta.kubernetes.io/os=linux	21m					

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/calico-kube-controllers	0/1	0	0	21m
deployment.apps/calico-typha	0/0	0	0	21m
deployment.apps/coredns	0/2	0	0	31m
deployment.apps/metrics-server	0/2	0	0	33m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/calico-kube-controllers-759789cdfb	1	0	0	4m18s
replicaset.apps/coredns-77cc869d4c	2	0	0	4m18s

The administrator can now configure the cluster to suit the particular site requirements.

9.3.3 Using GPUs With Kubernetes: NVIDIA GPUs

Prerequisites

The GPUs have to be recognized by the nodes, and have the appropriate drivers (such as `cuda-driver`) installed. Details on how to do this are given in section 7.4 of the *Installation Manual*.

To verify the GPUs are recognized and have drivers in place, the invoking `nvidia-smi` command can be invoked. The response displayed for a GPU should look similar to the following:

Example

```
root@node001:~# nvidia-smi
Mon Sep 28 15:13:22 2020
+-----+
| NVIDIA-SMI 450.51.06      Driver Version: 450.51.06      CUDA Version: 11.0      |
+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf   Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           |                  |     MIG M. |
+=====+
| 0   Tesla K40c          On      | 00000000:00:06.0 Off  |                    |
| 23%   28C    P8      22W / 235W |      0MiB / 12206MiB |      0%      Default |
|                                           |                  |     N/A  |
+-----+-----+-----+
```

Existing Docker Deployment

If Docker has been deployed before Kubernetes is deployed, then the instructions from section 9.1.7 must be followed first. These are the instructions on making sure that GPUs can be used inside Docker. The NVIDIA container toolkit (cm-nvidia-container-toolkit) has to be present on the nodes.

To verify Docker is working with GPUs, `nvidia-smi` can be run from inside a container:

Example

```
root@node001:~# module load docker/19.03.5
root@node001:~# docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
Mon Sep 28 13:13:39 2020
+-----+
| NVIDIA-SMI 450.51.06      Driver Version: 450.51.06      CUDA Version: 11.0      |
+-----+-----+-----+-----+-----+-----+
| GPU  Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           |              | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+-----+
|   0   Tesla K40c          On          | 00000000:00:06.0 Off  |                    |
| 23%   28C    P8     22W / 235W |      0MiB / 12206MiB |      0%      Default |
|                                           |              | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+-----+
|                                           |              |                    N/A |
+-----+-----+-----+-----+-----+
+-----+
| Processes:                                |
| GPU  GI    CI          PID    Type    Process name                        GPU Memory |
|      ID    ID                                   |           Usage                      |
+-----+-----+-----+-----+-----+-----+
| No running processes found                |
+-----+-----+-----+-----+-----+-----+
+-----+
```

New Kubernetes Installation

If Docker is deployed as part of the Kubernetes setup, then the NVIDIA container toolkit is taken care of by the installer. During the installation of Kubernetes there is a step where custom addons can be selected (figure 9.3):

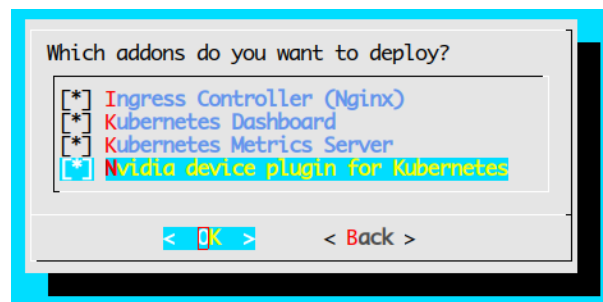


Figure 9.3: Kubernetes Ncurses Session: Deployment Of Addons

In that step, the add-on Nvidia device plugin for Kubernetes should be enabled. This ensures that `cm-nvidia-container-toolkit` is installed in the software image.

This is the default approach. It can be verified to work by deploying a Pod that requests a GPU, as requested using the `nvidia.com/gpu: 1` inside the yaml file. Alternatively there is a GPU operator that can be deployed via Helm (page 373)

If more GPUs are available on a single host, then only one GPU should be made visible, and recognized inside the Pod, when requesting a single GPU, as in the example:

Example

```

root@cluster:~> cat gpu.yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  restartPolicy: Never
  containers:
  - name: cuda-container
    image: nvidia/cuda:9.2-runtime
    command: ["nvidia-smi"]
    resources:
      limits:
        nvidia.com/gpu: 1
root@cluster:~> kubectl apply -f gpu.yaml
pod/gpu-pod configured
root@cluster:~> kubectl logs gpu-pod
Mon Sep 28 12:12:46 2020

```

```

+-----+
| NVIDIA-SMI 450.51.06      Driver Version: 450.51.06      CUDA Version: 11.0      |
+-----+-----+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           |              | MIG M. |
+=====+=====+=====+=====+=====+
|   0   Tesla K40c          On      | 00000000:00:06:0 Off |             Off      |
| 23%   29C    P8     22W / 235W |      0MiB / 12206MiB |      0%      Default  |
|                                           |              | N/A      |
+-----+-----+-----+-----+-----+

+-----+
| Processes: |
| GPU  GI  CI           PID   Type   Process name                      GPU Memory |
|          ID  ID                                   |             Usage  |
+=====+=====+=====+=====+=====+
| No running processes found |
+-----+

```

Existing Kubernetes Installation

The NVIDIA device plugin is deployed as an addon. If during the installation the addon was not selected (section 9.3.3 shows a screenshot), then it can be enabled in `cmsh` or Bright View afterwards.

```

[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% set enabled yes
[cluster->kubernetes*[default*]->appgroups*[system*]->applications*[nvidia*]]% commit

```

This addon deploys a DaemonSet that will run the NVIDIA device plugin for Kubernetes on nodes with the `brightcomputing.com/gpu-accelerator` label.

Bright Cluster Manager is responsible for setting this GPU label on the appropriate Kubernetes nodes, with parameter values specified for the labelset:

Example

```

[cluster->kubernetes[default]->labelsets[nvidia]]% show
Parameter          Value

```

```

-----
Categories
Name           nvidia
Nodes          node001
Overlays
Revision
Labels         brightcomputing.com/gpu-accelerator=

```

These labels should appear in Kubernetes Node resources as a result, and the device plugin running on these particular nodes (some columns of output have been removed for clarity):

Example

```

kubectl get nodes --show-labels
NAME          STATUS    ROLES    AGE    VERSION    LABELS
node001       Ready    <none>   3h13m  v1.16.1    brightcomputing.com/gpu-accelerator=,...
cluster       Ready    master   3h13m  v1.16.1    node-role.kubernetes.io/master=,...
root@rb-mygpufriend:~> kubectl get pod -l name=nvidia-device-plugin-ds -n kube-system -o wide
NAME          IP             NODE      NOMINATED NODE  READINESS GATES
nvidia-device-plugin-daemonset-pzjcl 172.29.152.142  node001   <none>           <none>

```

Using The NVIDIA GPU Operator

The NVIDIA GPU Operator (<https://github.com/NVIDIA/gpu-operator>) can be deployed using Helm. It leverages the Kubernetes Operator Framework, and has the benefit that it does not require any labels to be set by the user or by Bright Cluster Manager. GPUs should be detected by the operator, and it knows for which nodes the device plugin should be deployed automatically.

For this case, if it is in use, the NVIDIA device plugin addon configured by Bright should be disabled.

```

[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% set enabled no
[cluster->kubernetes*[default*]->appgroups*[system*]->applications*[nvidia*]]% commit

```

The nodes from the labelset can optionally be removed, since `brightcomputing.com/gpu-accelerator=` is no longer used for anything. The deployment can then proceed using Helm.

These instructions follow the guidance at <https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html> (and some columns of display output have been removed for clarity):

Example

```

root@cluster:~> helm repo add nvidia https://nvidia.github.io/gpu-operator
root@cluster:~> helm repo update

```

```

root@cluster:~> helm install nvidia/gpu-operator --wait --generate-name
NAME: gpu-operator-1601301333
LAST DEPLOYED: Mon Sep 28 15:55:36 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
root@cluster:~> helm list
NAME          NAMESPACE    STATUS    CHART          APP VERSION
gpu-operator-1601301333 default        deployed  gpu-operator-1.1.7 1.1.7

```

Uninstalling can be done through Helm as well:

Example

```

root@cluster:~> helm delete gpu-operator-1601301333
release "gpu-operator-1601301333" uninstalled

```

9.3.4 Using GPUs With Kubernetes: AMD GPUs

Prerequisites

The GPUs have to be recognized by the node. One way to check this from within Bright Cluster Manager is to run `sysinfo` for the node:

Example

```
[bright90->device[bright90]]% sysinfo | grep GPU
Number of GPUs                1
GPU Driver Version            4.18.0-193.el8.x86_64
GPU0 Name                     Radeon Instinct MI25
```

In order to make Kubernetes aware of nodes that have AMD GPUs, the AMD GPU device plugin has to be deployed as a DaemonSet inside Kubernetes. The official GitHub repository that hosts this plugin can be found at:

<https://github.com/RadeonOpenCompute/k8s-device-plugin>

The device plugin requires Kubernetes v1.16+, which has been around since Bright Cluster Manager version 9.0. With some extra instructions, the plugin can also be made a part of Bright Cluster Manager version 8.2.

The DaemonSet YAML file can be deployed with:

Example

```
kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/k8s-ds-amdgpu-dp.yaml
```

Managing The YAML File Through CMDaemon

The plugin can be added by the user via the Kubernetes appgroups as an application. In the session that follows, it is given the arbitrary name `device-plugin`:

Example

```
[root@bright90 ~]# wget https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/k8s-ds-amdgpu-dp.yaml -O /tmp/k8s-ds-amdgpu-dp.yaml
[root@bright90 ~]# cmsh
[bright90]% kubernetes
[bright90->kubernetes[default]]% appgroups
[bright90->kubernetes[default]->appgroups]% add amd
[bright90->kubernetes*[default*]->appgroups*[amd*]]% applications
[bright90->kubernetes*[default*]->appgroups*[amd*]->applications]% add device-plugin
```

The configuration of the plugin can be set to the YAML file, by setting the `config` parameter to take the value of the YAML file path.

Example

```
[bright90->...[amd*]->applications*[device-plugin*]]% set config /tmp/k8s-ds-amdgpu-dp.yaml
[bright90->kubernetes*[default*]->appgroups*[amd*]->applications*[device-plugin*]]% show
```

Parameter	Value
Name	device-plugin
Revision	
Format	Yaml
Enabled	yes
Config	<914B>


```
Environment <0 in submode>
Exclude list snippets <0 in submode>
[bright90->kubernetes*[default*]->appgroups*[amd*]->applications*[device-plugin*]]% commit
```

The YAML file can also be edited within `cmsh` after it has been set, by running `set config` without a value.

There are older releases available, starting from Kubernetes v1.10, if needed. Saving this device-plugin YAML should result in pods being scheduled on all the non-tainted nodes, as seen by listing the pods (some columns elided):

```
[root@bright90 ~]# module load kubernetes/default/1.18.8
[root@bright90 ~]# kubectl get pod -n kube-system -l name=amdgpu-dp-ds -o wide
NAME                                READY  STATUS   ... IP                                NODE   ...
amdgpu-device-plugin-daemonset-66jl7 1/1    Running  ... 172.29.112.135  gpu001 ...
amdgpu-device-plugin-daemonset-8mh9w 1/1    Running  ... 172.29.152.130  gpu002 ...
```

Including Head Nodes as part of the DaemonSet:

Bright Cluster Manager taints head nodes, so that they do not run non-critical pods. The taint can be removed with the “-” operator to allow non-critical pods to run:

Example

```
kubectl taint nodes bright90 node-role.kubernetes.io/master-
```

However, a more specific exception can be configured in the DaemonSet itself.

Within the YAML file, the following existing `tolerations` definition has to be modified, from:

```
tolerations:
- key: CriticalAddonsOnly
  operator: Exists

to:

tolerations:
- key: node-role.kubernetes.io/master
  effect: NoSchedule
  operator: Exists
```

The modified toleration tolerates this taint, and therefore has the device plugin run on such tainted nodes.

Verifying that AMD GPUs are recognized by Kubernetes: If Kubernetes is aware of the AMD GPUs for a node, then several mentions of `amd.com/gpu` are displayed when running the `kubectl describe node` command for the node. The following session shows output for a node `gpu01`, ellipsized for clarity:

Example

```
[root@bright90 ~]# kubectl describe node gpu01
Name:                gpu01
...
Capacity:
  amd.com/gpu:       3
  cpu:               64
  ephemeral-storage: 1813510Mi
  hugepages-1Gi:    0
  hugepages-2Mi:    0
  memory:            527954676Ki
  pods:              50
...
```

Running The DaemonSet Only On Specific Nodes

The AMD GPU device plugin, unlike the NVIDIA GPU device plugin Daemonset, is scheduled to run on each Kubernetes host. This means that it runs even if the host has no GPU.

This can be prevented with the following steps:

A LabelSet can be created via `cmsh`, and the nodes or categories that have GPUs are assigned within the `labelsets` mode:

Example

```
[root@bright90 ~]# cmsh
[bright90]% kubernetes
[bright90->kubernetes[default]]% labelsets
[bright90->kubernetes[default]->labelsets]% use nvidia
[bright90->kubernetes[default]->labelsets[nvidia]]% .. #but, we're using AMD GPUs, so:
[bright90->kubernetes[default]->labelsets]% add amd
[bright90->kubernetes*[default*]->labelsets*[amd*]]% set labels brightcomputing.com/amd-gpu-accelerator=
[bright90->kubernetes*[default*]->labelsets*[amd*]]% append categories gpu-nodes
[bright90->kubernetes*[default*]->labelsets*[amd*]]% commit
```

This assigns the labels to the nodes with GPUs. This can be verified with:

Example

```
kubectl get nodes -l brightcomputing.com/amd-gpu-accelerator=
NAME      STATUS    ROLES    AGE   VERSION
gpu001    Ready     master   66m   v1.18.8
gpu002    Ready     master   66m   v1.18.8
...
```

The DaemonSet YAML can now be adjusted to only run the device plugin on nodes with this new label. This can be done by adding an affinity block after the tolerations block:

Example

```
tolerations:
- key: CriticalAddonsOnly # toleration may be different, if changes were made to it
  operator: Exists
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: 'brightcomputing.com/amd-gpu-accelerator'
          operator: Exists
```

This results in the device plugin pods being removed immediately from all nodes that do not have the label.

Running An Example Workload

An example workload can be run as described in the official AMD GPU Kubernetes device plugin documentation at:

<https://github.com/RadeonOpenCompute/k8s-device-plugin/tree/v1.16#example-workload>

Thus it should now be possible to run:

```
[root@bright90 ~]# kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/example/pod/alexnet-gpu.yaml
```

The YAML requests only one GPU at the bottom of the YAML file:

```
apiVersion: v1
kind: Pod
metadata:
  name: alexnet-tf-gpu-pod
  labels:
    purpose: demo-tf-amdgpu
spec:
  containers:
    - name: alexnet-tf-gpu-container
      image: rocm/tensorflow:latest
      workingDir: /root
      env:
        - name: HIP_VISIBLE_DEVICES
          value: "0" # # 0,1,2,...,n for running on GPU and select the GPUs, -1 for running on CPU
      command: ["/bin/bash", "-c", "--"]
  args: ["python3 benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=alexnet;\
  trap : TERM INT; sleep infinity & wait"]
  resources:
    limits:
      amd.com/gpu: 1 # requesting a GPU
```

Container creation might take a while due to the image size. Once scheduled, it prints out that it found exactly one GPU, and proceeds to run a TensorFlow workload.

Example

```
[root@bright90 ~]# kubectl logs -f alexnet-tf-gpu-pod
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/compat/v2_compat.py:96:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed
in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
2021-01-08 21:03:29.222293: I tensorflow/core/platform/profile_utils/cpu_utils.cc:104]
CPU Frequency: 2495445000 Hz
2021-01-08 21:03:29.222398: I tensorflow/compiler/xla/service/service.cc:168]
XLA service 0x39f62f0 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2021-01-08 21:03:29.222420: I tensorflow/compiler/xla/service/service.cc:176]
   StreamExecutor device (0): Host, Default Version
2021-01-08 21:03:29.223754: I tensorflow/stream_executor/platform/default/dso_loader.cc:48]
Successfully opened dynamic library libamdhip64.so
2021-01-08 21:03:31.635339: I tensorflow/compiler/xla/service/service.cc:168]
XLA service 0x3a40bb0 initialized for platform ROCm (this does not guarantee that XLA will be used). Devices:
2021-01-08 21:03:31.635363: I tensorflow/compiler/xla/service/service.cc:176]
   StreamExecutor device (0): Device 738c, AMDGPU ISA version: gfx908
2021-01-08 21:03:31.931125: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1734]
Found device 0 with properties:
   pciBusID: 0000:27:00.0 name: Device 738c   ROCm AMD GPU ISA: gfx908
   coreClock: 1.502GHz coreCount: 120 deviceMemorySize: 31.98GiB deviceMemoryBandwidth: 1.12TiB/s
...
TensorFlow: 2.3
Model: alexnet
Dataset: imagenet (synthetic)
```

```

Mode:      training
SingleSess: False
Batch size: 512 global
           512 per device
Num batches: 100
Num epochs: 0.04
Devices:   ['/gpu:0']
...

```

Had more GPUs been requested, more would have been made available to the container.

For comparison, a CPU version of the container is also available. The official instructions can be referred to for these, too.

9.3.5 Kubernetes Configuration Overlays

A list of configuration overlays can be seen from within configurationoverlay mode:

Example

```

[bright90->configurationoverlay]% list
Name (key)      Priority  Nodes              Categories  Roles
-----
kube-default-etcd    500      node001..node003  Etcd::Host
kube-default-master  510      node001..node003  Docker::Host, Kube...
kube-default-worker  500      node004..node006  default    Docker::Host, Kube...

```

9.3.6 Removing A Kubernetes Cluster

A Kubernetes cluster can be removed using `cm-kubernetes-setup` with the `--remove` and `--yes-i-really-mean-it` options. Also, if there more than one cluster present, then the cluster name must be specified using the `--cluster` parameter.

A removal run looks as follows (some output ellipsized):

Example

```

[root@bright90 ~]# cm-kubernetes-setup --remove --cluster default --yes-i-really-mean-it

```

```

Connecting to CMDaemon
Executing 17 stages
##### Starting execution for 'Kubernetes Setup'
- kubernetes
- docker
## Progress: 0
#### stage: kubernetes: Get Kube Cluster
## Progress: 5
#### stage: kubernetes: Check Kube Cluster Exists
## Progress: 11
#### stage: kubernetes: Find Installed Components
## Progress: 17
#### stage: kubernetes: Find Files On Headnodes
## Progress: 23
#### stage: kubernetes: Close Api Server Proxy Port
Closing port 10443 in Shorewall
## Progress: 52
#### stage: kubernetes: Remove Installed Components
## Progress: 58
#### stage: kubernetes: Remove Files On Headnodes
## Progress: 64

```

```
#### stage: kubernetes: Remove Etcd Spool
## Progress: 70
#### stage: kubernetes: Set Reboot Required
You need to reboot 6 nodes to cleanup the network configuration
## Progress: 76
#### stage: kubernetes: Collection Update Provisioners
## Progress: 82
#### stage: docker: Get Objects To Process
## Progress: 100

Took:      00:01 min.
Progress: 100/100
##### Finished execution for 'Kubernetes Setup', status: completed
```

Kubernetes Setup finished!

Using the `--remove` option 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.

After the disabling procedure has finished, the cluster has no Kubernetes configured and running.

9.3.7 Kubernetes Cluster Configuration Options

Kubernetes allows many Kubernetes clusters to be configured. These are separated sets of hosts with different certificates, users and other global settings.

When carrying out the Kubernetes setup run, a Kubernetes cluster name will be asked, and a new object with the cluster settings is then added into the `CMDaemon` configuration. The administrator can change the settings of the cluster from within the `kubernetes` mode of `cmsh` or within the Kubernetes Clusters options window of Bright View, accessible via the clickpath `Containers→Kubernetes Clusters`.

The `cmsh` equivalent looks like:

Example

```
[root@bright90 ~]# cmsh
[bright90]% kubernetes list
Name (key)
-----
default
[bright90]% kubernetes use default
[bright90->kubernetes[default]]% show
Parameter                                     Value
-----
Kubernetes role bindings                       <0 in submode>
Notes
Revision
Name                                           default
Authorization Mode                             Node,RBAC
Kube Config                                   /cm/local/apps/kubernetes/var/etc/node.kubeconfig
Kube Client Config                             /cm/local/apps/kubernetes/var/etc/kubelet.kubeconfig
Kube Config Template                           <409 bytes>
CA                                              /cm/local/apps/kubernetes/var/etc/kubeca-default.pem
CA Key                                         /cm/local/apps/kubernetes/var/etc/kubeca-default.key
Kubernetes Certificate                         /cm/local/apps/kubernetes/var/etc/node.pem
Kubernetes Key                                 /cm/local/apps/kubernetes/var/etc/node.key
```

```

Kubernetes Client Certificate      /cm/local/apps/kubernetes/var/etc/kubelet.pem
Kubernetes Client Key             /cm/local/apps/kubernetes/var/etc/kubelet.key
Service Accounts Certificate      /cm/local/apps/kubernetes/var/etc/sa-default.pem
Service Accounts Certificate Key  /cm/local/apps/kubernetes/var/etc/sa-default.key
Kubernetes API Aggregator Certificate /cm/local/apps/kubernetes/var/etc/apiagggregator.pem
Kubernetes API Aggregator Certificate Key /cm/local/apps/kubernetes/var/etc/apiagggregator.key
Cluster Domain                    cluster.local
Etcd Cluster                      kube-default
Etcd Prefix                       /kube-apiserver
Etcd Servers
Service Network                   kube-default-service
Trusted domains                   bright90.openstacklocal,kubernetes,kubernetes.default,kubernetes.\
                                  default.svc,master,localhost

Pod Network                       kube-default-pod
Pod Network Node Mask
Internal Network                  internalnet
KubeDNS IP                       10.150.255.254
Kubernetes API server
Kubernetes API server proxy port  10444
App Groups                        <1 in submode>
Label Sets                        <2 in submode>
Module file template              <1130 bytes>
[bright90->kubernetes[default]]%

```

The preceding kubernetes mode parameters are described in table 9.6:

Parameter	Description
Authorization Mode	Selects how to authorize on the secure port (default: RBAC (Role-Based Access Control) and Node Authorization modes)
Kube Config	Path to a kubeconfig file, specifying how nodes authenticate to the API server
Kube Client Config	Path to a kubeconfig file, specifying how kubelets authenticate to the API server
Kube Config Template	Template Bright Cluster Manager uses to generate kubeconfig files for services and users.
CA	Path to PEM-encoded RSA or ECDSA certificate used for the CA
CA Key	Path to PEM-encoded RSA or ECDSA private key used for the CA
Kubernetes Certificate	File containing x509 certificate used by the kubelets
Kubernetes Key	File containing x509 private key used by the Kubernetes kubelets

...continues

...continued

Parameter	Description
Kubernetes Client Certificate	File containing x509 certificate used for the kubelets
Kubernetes Client Key	File containing x509 private key used for the Kubernetes kubelets
Service Accounts Certificate	File containing x509 certificate used for Kubernetes service accounts. This certificate value will be used as <code>--service-account-key-file</code> in the Kubernetes API service.
Service Accounts Key	File containing x509 private key used for Kubernetes Service Accounts. This key value will be used as <code>--service-account-private-key-file</code> in the controller manager service.
Cluster Domain	Domain for this cluster
Etcd Cluster	The Etcd cluster instance.
Etcd Prefix	The prefix for all resource paths in etcd
Etcd Servers	List of etcd servers to watch (format: <code>http://<IP address>:<port number></code>)
Service Network	Network from which the service cluster IP addresses will be assigned, in IPv4 CIDR format. Must not overlap with any IP address ranges assigned to nodes for pods. Default: <code>172.29.0.0/16</code>
Pod Network	Network where pod IP addresses will be assigned from
Internal Network	Network to back the internal communications.
Trusted Domains	Trusted domains to be included in Kubernetes-related certificates as Alternative Subject Names.
KubeDNS IP	CoreDNS IP Address.
Kubernetes API Server	Kubernetes API server address (format: <code>https://<host>:<port number></code>).
Kubernetes API Server Proxy Port	Kubernetes API server proxy port.
Addons	Kubernetes Add-ons managed by CMDaemon.

Table 9.6: kubernetes mode parameters

9.3.8 EtcdCluster

The EtcdCluster mode sets the global Etcd cluster settings. It can be accessed via the top level etcd mode of cmsh.

Parameter	Description	Option to etcd
Name	Etcd Cluster Name.	--initial-cluster-token
Election Timeout	Election Timeout, in milliseconds.	--election-timeout
Heart Beat Interval	Heart Beat Interval, in milliseconds.	--heartbeat-interval
CA	The Certificate Authority (CA) Certificate path for Etcd, used to generate certificates for Etcd.	--peer-trusted-ca-file
CA Key	The CA Key path for Etcd, used to generate certificates for Etcd.	
Member Certificate	The Certificate path to use for Etcd cluster members, signed with the Etcd CA. The EtcdHost Role can specify a Member CA as well, and in that case it overwrites any value set here.	--peer-cert-file
Member Certificate Key	The Key path to use for Etcd cluster members, signed with the Etcd CA. The EtcdHost Role can specify a Member CA as well, and in that case it overwrites any value set here.	--peer-key-file
Client CA	The CA used for client certificates. When set it is assumed client certificate and key will be generated and signed with this CA by another party. Etcd still expects the path to be correct for the Client Certificate and Key.	--trusted-ca-file
Client Certificate	The Client Certificate, used by Etdctl for example.	--cert-file
Client Certificate Key	The Client Certificate Key, used by Etdctl for example.	--key-file

* Boolean (takes yes or no as a value)

Table 9.7: EtcdCluster role parameters and etcd options

9.3.9 Kubernetes Roles

Kubernetes roles include the following roles:

- EtcdHost (page 383)
- KubernetesApiServerProxy (page 384)
- KubernetesApiServer (page 384)
- KubernetesController (page 386)
- KubernetesScheduler (page 388)

- `KubernetesProxy` (page 389)
- `KubernetesNode` (page 390)

When nodes are configured using Kubernetes roles, then settings in these roles may sometimes use the same pointer variables—for example the `Kubernetes` or `Etcd` cluster instance. Pointer variables such as these have definitions that are shared across the roles, as indicated by the parameter description tables for the roles, and which are described in the following pages.

In `cmsh`, the roles can be assigned:

- for individual nodes via the `roles` submode of `device` mode
- for a category via the `roles` submode of a category
- for a configuration overlay via the `roles` submode of `configurationoverlay` mode

EtcdHost Role

The `EtcdHost` role is used to configure and manage the `etcd` service for a node.

The `etcd` service manages the `etcd` database, which is a hierarchical distributed key-value database. The database is used by Kubernetes to store its configurations. The `EtcdHost` role parameters are described in table 9.8:

Parameter	Description	Option to <code>etcd</code>
Member Name	The human-readable name for this <code>etcd</code> member (<code>\$hostname</code> will be replaced by the node hostname)	<code>--name</code>
Spool	Path to the data directory (default: <code>/var/lib/etcd</code>)	<code>--data-dir</code>
Advertise Client URLs	List of client URLs for this member to advertise publicly (default: <code>http://\$hostname:5001</code>)	<code>--advertise-client-urls</code>
Advertise Peers URLs	List of peer URLs for this member to advertise to the rest of the cluster (default: <code>http://\$hostname:5002</code>)	<code>--initial-advertise-peer-urls</code>
Listen Client URLs	List of URLs to listen on for client traffic (default: <code>http://\$hostname:5001</code> , <code>http://127.0.0.1:2379</code>)	<code>--listen-client-urls</code>
Listen Peer URLs	List of URLs to listen on for peer traffic (default: <code>http://\$hostname:5002</code>)	<code>--listen-peer-urls</code>
Snapshot Count	Number of committed transactions that trigger a snapshot to disk (default: 5000)	<code>--snapshot-count</code>
Debug*	Drop the default log level to <code>DEBUG</code> for all subpackages? (default: <code>no</code>)	<code>--debug</code>
Member Certificate	<code>Etcd</code> member certificate, signed with CA specified in the <code>Etcd Cluster</code> . When set it will overrule the value from the <code>EtcdCluster</code> object. Default empty.	<code>--peer-cert-file</code>

...continues

...continued

Parameter	Description	Option to etcd
Member Certificate Key	Etcd member certificate key, signed with CA specified in the Etcd Cluster. When set it will overrule the value from the EtcdCluster object. Default empty.	--peer-keyt-file
Options	Additional parameters for the etcd daemon (empty by default)	

* Boolean (takes yes or no as a value)

Table 9.8: 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`.

The KubernetesAPIServerProxy Role

The KubernetesAPIServerProxy role sets up a proxy that provides the entry point for one or more instances of the Kubernetes API server. The proxy runs on every node of a Kubernetes cluster instance, including the head node.

If multiple Kubernetes master nodes are present, then it enables HA for the Kubernetes master components, as described in section 9.3.1.

The KubernetesApiServer Role

The KubernetesApiServer 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 replication controllers. The API Server processes REST operations, and provides a front end to the shared state of the cluster through which all the other components interact.

The KubernetesApiServer role parameters are described in table 9.9:

Parameter	Description	Option to kube-apiserver
Kubernetes Cluster	The Kubernetes cluster instance (pointer)	
Insecure API Port	The port on which to serve unsecured, unauthenticated access (disabled by default)	<code>--insecure-port</code>
Secure API Port	The port on which to serve HTTPS with authentication and authorization. If 0, then HTTPS will not be served at all. (default: 6443)	<code>--secure-port</code>
Advertise Address	The IP address on which to advertise the API server to members of the cluster with <code>--advertise-address</code> . If set to 0.0.0.0, then the IP address of the management network of the head node is used. (default: 0.0.0.0)	
Insecure Bind Address	IP address to serve on (default: 127.0.0.1)	<code>--insecure-bind-address</code>

...continues

...continued

Parameter	Description	Option to kube-apiserver
Secure Bind Address	The IP address on which to serve the read- and secure ports (default: 0.0.0.0)	--bind-address
Admission Control	Ordered list of plug-ins to control the admission of resources into the cluster (default: NamespaceLifecycle,LimitRanger,ServiceAccount,PersistentVolumeLabel,DefaultStorageClass,ValidatingAdmissionWebhook,ResourceQuota,DefaultTolerationSeconds,MutatingAdmissionWebhook)	--admission-control
Allowed Privileged*	If true, allow privileged containers (default: no)	--allow-privileged
Event TTL	Time period that events are retained. Empty by default. A format example: 1h0m0s	--event-ttl
Kubelet Timeout	Kubelet port timeout (default: 5s)	--kubelet-timeout
Log Level	Log level (default: 0)	--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 (empty by default)	

* Boolean (takes yes or no as a value)

Table 9.9: *KubernetesApiServer role parameters and kube-apiserver options*

Further details on the Kubernetes API Server can be found at <https://kubernetes.io/docs/admin/kube-apiserver/>.

KubernetesController Role

The Kubernetes Controller role is used to configure and manage the kube-controller-manager daemon that embeds the core control loops shipped with Kubernetes. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API server, and it makes changes in order to try to move the current state towards the desired state. Examples of controllers that ship with Kubernetes at the time of writing (January 2018) are:

- the replication controller
- the endpoints controller,
- the namespace controller,
- the serviceaccounts controller

The KubernetesController role parameters are described in table 9.10:

Parameter	Description	Option to kube-controller-manager
Kubernetes Cluster	The Kubernetes cluster instance (pointer)	
Address	IP address to serve on (default: 0.0.0.0)	--address
Port	Port to serve on (default: 10252)	--port
Concurrent Endpoint Syncs	Number of endpoint syncing operations that will be done concurrently. (default: 5)	--concurrent-endpoint-syncs
Concurrent Rc Syncs	The number of replication controllers that are allowed to sync concurrently. 5	--concurrent-rc-syncs
Namespace Sync Period	Period for syncing namespace life-cycle updates	--namespace-sync-period
Node Monitor Grace Period	Period for syncing NodeStatus in NodeController	--node-monitor-grace-period
Node Monitor Period	Period the running Node is allowed to be unresponsive before marking it unhealthy	--node-monitor-period
Node Startup Grace Period	Period the starting Node is allowed to be unresponsive before marking it unhealthy	--node-startup-grace-period
Node Sync Period	Period for syncing nodes from cloud-provider	--node-sync-period
Pod Eviction Timeout	Grace period for deleting pods on failed nodes	--pod-eviction-timeout
Pv Claim Binder Sync Period	Period for syncing persistent volumes and persistent volume claims	--pvclaimbinder-sync-period
Register Retry Count	Number of retries for initial node registration	--register-retry-count
Resource Quota Sync Period	Period for syncing quota usage status in the system	--resource-quota-sync-period
Log Level	Log level (default: 0)	--v
Log To StdErr*	Logging to stderr means getting it into the systemd journal (default: yes)	--logtostderr
Options	Additional parameters for the kube-controller-manager daemon	
Cluster signing Cert file	Filename containing a PEM-encoded X509 CA certificate used to issue cluster-scoped certificates. (leave empty to use the value of CA defined in the Kubernetes Cluster instance).	--cluster-signing-cert-file

...continued

Parameter	Description	Option to kube-controller-manager
Cluster signing Cert Key file	Filename containing a PEM-encoded RSA or ECDSA private key used to sign cluster-scoped certificates. (leave empty to use the value of CA Key defined in the Kubernetes Cluster instance).	--cluster-signing-key-file
Use Service Account Credentials	Flag to enable or disable use of Service Account Credentials.	--use-service-account-credentials
Allocate Node Cidrs	Allocate node CIDR in cluster using Pod Network and Node Mask size defined in Kubernetes Cluster Object.	--allocate-node-cidrs
Kube Config	Path to a kubeconfig file, specifying how to authenticate to API server.	--kubeconfig
Kubernetes Certificate	File containing x509 Certificate used by Kubernetes Controller Manager. This will be embedded in the Kube Config file.	(1)
Kubernetes Key	File containing x509 private key used by Kubernetes Controller Manager. This will be embedded in the Kube Config file.	(2)

* Boolean (takes yes or no as a value)

Table 9.10: *KubernetesController role parameters and kube-controller-manager options*

Further details on the Kubernetes controller manager can be found at <https://kubernetes.io/docs/admin/kube-controller-manager/>.

KubernetesScheduler Role

The KubernetesScheduler role is used to configure and manage the kube-scheduler daemon. The Kubernetes scheduler defines pod placement, taking into account the individual and collective resource requirements, quality of service requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, deadlines, and so on.

The KubernetesScheduler role parameters are described in table 9.11:

Parameter	Description	Option to kube-scheduler
Kubernetes Cluster	The Kubernetes cluster instance (pointer)	
Address	IP address to serve on (default: 0.0.0.0)	--address
Scheduler Port	Port to serve on (default: 10253)	--port
Algorithm Provider	The scheduling algorithm provider to use (default: DefaultProvider)	--algorithm-provider
Policy Config	File with scheduler policy configuration (default: /cm/local/apps/kubernetes/var/etc/scheduler-policy.json)	--policy-config-file

...continues

...continued

Parameter	Description	Option to kube-scheduler
Log Level	Log level (default: 0)	--v
Log To StdErr*	Logging to STDERR means getting it into the systemd journal (default: yes)	--logtostderr
Options	Additional parameters for the kube-scheduler daemon	
Kube Config	Path to a kubeconfig file, specifying how to authenticate to API server.	--kubeconfig
Kubernetes Certificate	File containing x509 Certificate used by Kubernetes Scheduler. This certificate will be embedded in the Kube Config file.	
Kubernetes Key	File containing x509 private key used by Kubernetes Scheduler. This certificate key will be embedded in the Kube Config file.	

* Boolean (takes yes or no as a value)

Table 9.11: *KubernetesScheduler* role parameters and kube-scheduler options

Further details on the Kubernetes scheduler can be found at <https://kubernetes.io/docs/admin/kube-scheduler/>.

KubernetesProxy Role

The KubernetesProxy role is used to configure and manage kube-proxy daemon. The kube-proxy daemon runs on each node, and reflects services as defined in the Kubernetes API. It can do simple TCP and UDP stream-forwarding or round-robin TCP and UDP forwarding across a set of back ends.

The KubernetesProxy role parameters are described in table 9.12:

Parameter	Description	Configuration Parameter Passed To kube-proxy
Kubernetes Cluster	The Kubernetes cluster instance (pointer)	
Address	IP address to serve on (default: 0.0.0.0)	--address
Proxy Port Range Start	Bottom of range of host ports that may be consumed in order to proxy service traffic (not set by default)	--proxy-port-range
Proxy Port Range End	Top of range of host ports that may be consumed in order to proxy service traffic (not set by default)	--proxy-port-range
Masquerade All	If using the pure iptables proxy, SNAT everything (default: yes)	--masquerade-all
Health Check Address	IP address for the health check server to serve on	--healthz-port
Health Check Port	Port to bind the health check server to serve on (default: 10251, use 0 to disable)	--healthz-port
Oom Score Adjust	The oom_score_adj value for the kube-proxy process, in range [-999, 999] (default: -999)	--oom-score-adj
Kube Config	Path to a kubeconfig file, specifying how to authenticate to the API server.	--kubeconfig
Kubernetes Certificate	File containing x509 Certificate used by kube-proxy. This certificate is embedded in the Kube Config.	
Kubernetes Key	File containing x509 private key used by Kubernetes API server. This certificate key is embedded in the Kube Config.	
Log Level	Log level (default: 0)	--v
Log To StdErr*	Logging to STDERR means it goes in the systemd journal (default: yes)	--logtostderr
Options	Additional parameters for the kube-scheduler daemon	

* Boolean (takes yes or no as a value)

Table 9.12: *KubernetesProxy role parameters and kube-proxy options*

Further details on the Kubernetes network proxy can be found at <https://kubernetes.io/docs/admin/kube-proxy/>.

KubernetesNode Role

The KubernetesNode role is used to configure and manage the kubelet daemon, which is the primary node agent that runs on each node. The kubelet daemon takes a set of pod specifications, called *PodSpecs*, and ensures that the containers described in the PodSpecs are running and healthy.

The KubernetesNode role parameters are described in table 9.13:

Parameter	Description	Option to kubelet
Kubernetes Cluster	The Kubernetes cluster instance (pointer)	
Address	IP address to serve on (default: 0.0.0.0)	--address
Kubelet Port	Port that the HTTP service of the node runs on (default: 10250)	--port
CNI plugin binaries path	The full path of the directory in which to search for CNI plugin binaries. (default: /cm/local/apps/kubernetes/current/bin/cni)	--cni-bin-dir
Enable Server*	Enable server mode of Kubelet	--enable-server
Host Network Sources	List of sources from which Kubelet allows pods use of the host network (default: file)	--host-network-sources
Hostname Override	If non-empty, use this string as identification instead of the actual hostname (not set by default)	--hostname-override
Manifests Path	Path to the config file or directory of files (default: /cm/local/apps/kubernetes/var/etc/manifests)	--pod-manifest-path
Network plugin	The name of the network plugin to be invoked for various events in kubelet/pod lifecycle. (default: cni)	--network-plugin
Spool	Directory path for managing Kubelet files (default: /cm/local/apps/kubernetes/var/kubelet)	--root-dir
Cgroup Root	Optional root cgroup to use for pods	--cgroup-root
Docker Endpoint	Docker endpoint address to connect to (default: unix:///var/run/docker.sock)	--docker-endpoint
Docker Spool	Absolute path to the Docker state root directory (default: /var/lib/docker)	--docker-root
Resource Container	Absolute name of the resource-only container to create and run the Kubelet in (default: /kubelet)	--resource-container
Allowed Privileged*	If true, allow privileged containers (default: no)	--allow-privileged
Labels	List of node labels	
Register On Start*	Register the node with the API server (default: yes)	--register-node
Eviction minimum reclaim	Minimum amount of resources reclaimed in an eviction (default: imagefs.available=1Gi)	--eviction-minimum-reclaim

...continues

...continued

Parameter	Description	Option to kubelet
Hard eviction	Hard eviction constraints (default: imagefs.available<1%)	--eviction-hard
Max Pods	Number of pods that can run on this node	--max-pods
Max pod eviction grace period	Maximum allowed grace period (in seconds) allowed to terminated pods (default: 60)	--eviction-max-pod-grace-period
Soft eviction	Soft eviction constraints (default: imagefs.available<5%)	--eviction-soft
Soft eviction grace period	Soft eviction grace period (default: imagefs.available=1m30s)	--eviction-soft-grace-period
File Check Frequency	Duration between checking configuration files for new data (default: 20s)	--file-check-frequency
HTTP Flush Frequency	Duration between checking HTTP for new data (default: 20s)	--http-check-frequency
Node Status Update Frequency	The absolute free disk space, in MB, to maintain (default: 10s)	--node-status-update-\frequency
Run Once*	If true, exit after spawning pods from local manifests or remote URLs (default: no)	--runonce
Streaming Connection Idle Timeout	Maximum time a streaming connection can be idle before the connection is automatically closed (default: 1h)	--streaming-connection-\idle-timeout
Sync Frequency	Maximum period between synchronizing running containers and config (default: 10s)	--sync-frequency
Image GC High Threshold	Percent of disk usage after which image garbage collection is always run (default: 90)	--image-gc-high-\threshold
Image GC Low Threshold	Percent of disk usage before which image garbage collection is never run (default: 80)	--image-gc-low-\threshold
Threshold	maintain (default: 256)	threshold-mb
Oom Score Adjust	The oom_score_adj value for the kube-proxy process, in range [-999, 999] (default: -999)	--oom-score-adj
Log Level	Log level (default: 0)	--v
Log To StdErr*	Logging to STDERR means it gets into the systemd journal (default: yes)	--logtostderr
Options	Additional parameters for the kube-scheduler daemon	

...continues

...continued

Parameter	Description	Option to kubelet
-----------	-------------	-------------------

* Boolean (takes yes or no as a value)

Table 9.13: *KubernetesNode* role parameters and kubelet options

Further details on the kubelet daemon can be found at <https://kubernetes.io/docs/admin/kubelet/>.

9.3.10 Security Model

The Kubernetes security model allows authentication using a certificate authority (CA), with the user and daemon certificates signed by a Kubernetes CA. The Kubernetes CA should not be confused with the Bright Cluster Manager CA.

Bright Cluster Manager will create a CA specifically for issuing all Kubernetes-related certificates. The certificates are put into `/cm/local/apps/kubernetes/var/etc/` by default, and `/etc/kubernetes/` is made a link to this directory.

In Kubernetes terminology a user is a unique identity accessing the Kubernetes API server. The user may be a human or an automated process. For example an admin or a developer are human users, but kubelet represents an infrastructure user. Both types of users are authorized and authenticated in the same way against the API server.

Kubernetes uses client certificates, tokens, or HTTP basic authentication methods to authenticate users for API calls. Bright Cluster Manager configures client certificate usage by default. The authentication is performed by the API server which validates the user certificate using the common name part of the certificate subject.

In Kubernetes, authorization happens as a separate step from authentication. Authorization applies to all HTTP accesses on the main (secure) API server port. Bright Cluster Manager by default enables RBAC (Role-Based Access Control) combined with Node Authorization. The authorization check for any request thus takes the common name and/or organization part of the certificate subject to determine which roles the user or service has associated. Roles carry a certain set of privileges for resources within Kubernetes.

PodSecurityPolicy

Bright Cluster Manager also has support for PodSecurityPolicy (PSP) in the Kubernetes API Server. PSP can be explicitly enabled or disabled (section 9.3.12).

For each user, a PSP is generated and assigned via their role binding YAML configuration. The following defaults are applied:

- Users can only mount their own home directory. They cannot mount other paths such as `/etc`
- Users cannot run privileged Pods.
- Users can only bind on ports higher than 1024.
- Users can only run with their own `uid` and `gid`.

There are also

- default role bindings that grant access to the resources covered by the `kubectl get all` command.
- some minimally-required privileges required to run, for example, JupyterHub notebooks.

9.3.11 Addition Of New Kubernetes Users And Kubernetes Role Bindings Configuration

Bright users can use Kubernetes by making them Kubernetes users. This means having Kubernetes configuration and access set up for them. This can be carried out via the `cm-kubernetes-setup` Ncurses utility, and choosing the `Add user` option (figure 9.2). The utility then prompts for

- a Kubernetes cluster
- a user name
- a namespace that the privileges are to be assigned to. This can be
 - left blank to use `<user>-restricted`
 - set to the value `default` if PSP is not enabled.)
- a role for the user, with choices provided from:
 - `cluster-admin`: cluster-wide administrator
 - `admin`: administrator
 - `edit`: regular user
 - `view`: read-only user

Later the RBAC that is generated under the hood for this role binding can be customized via `cmsh`, in the `rolebinding` submode of the main `kubernetes` mode. For example, if a Bright user `test` has been created, and then made a Kubernetes user, then its rolebindings can be accessed as follows:

Example

```
[bright90]% kubernetes
[bright90->kubernetes[default]]% rolebindings #default is Kubernetes Cluster auto-completed name
[bright90->kubernetes[default]->rolebinding]% use test
[bright90->kubernetes[default]->rolebinding[test]]% get config
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
  apiGroup: rbac.authorization.k8s.io
- kind: User
  name: test
- kind: ServiceAccount
  name: test
  namespace: default
[bright90->kubernetes[default]->rolebinding[test]]% set config # opens an $EDITOR
...
[bright90->kubernetes*[default*]->rolebinding*[test*]]% commit
```

For a user within the `rolebinding` submode, the command `set config` opens an editor, and an appropriate RBAC setting for the user can then be copy/pasted. Or modified, if it already exists, as is the case for the user `test`, who was made a Kubernetes user via `cm-kubernetes-setup`.

How to construct the RBAC from scratch is documented at <https://kubernetes.io/docs/admin/authorization/rbac/#kubect1-create-rolebinding>.

If there is a need to create custom roles, then it is recommended to add a YAML configuration file for the roles in the Kubernetes applications submode.

A `cmsh` treeview that displays the hierarchy to access the configuration of a Kubernetes application is:

```
[cmsh]
|-- ...
|
|-- kubernetes[default]
|       |-- appgroups[system]
|       |       |-- applications
|       |-- labelsets[label set, for example: ingress_controller]
|       |-- rolebinding[user name]
```

An application configuration file, such as for the Dashboard application, can be accessed as indicated by the following session:

Example

```
[root@bright90 ~]# cmsh
[bright90]% kubernetes use default
[bright90->kubernetes[default]]% appgroups
[bright90->kubernetes[default]->appgroups]% use system
[bright90->kubernetes[default]->appgroups[system]]% show
Parameter                               Value
-----
Name                                     system
Revision
Enabled                                  yes
applications                             <10 in submode>
[bright90->kubernetes[default]->appgroups[system]]% applications
[bright90->kubernetes[default]->appgroups[system]->applications]% list
Name (key)                               Format Enabled
-----
bootstrap                                Yaml    yes
calico                                    Yaml    yes
dashboard                                 Yaml    yes
dashboard_ingress                        Yaml    yes
dns                                        Yaml    yes
ingress_controller                       Yaml    yes
kubernetes_ingress                       Yaml    no
metrics_server                           Yaml    yes
nvidia                                    Yaml    no
root                                       Yaml    yes
[bright90->kubernetes[default]->appgroups[system]->applications]% use dashboard
[bright90->kubernetes[default]->appgroups[system]->applications[dashboard]]% show
Parameter                               Value
-----
Name                                     dashboard
Revision
Format                                    Yaml
Enabled                                  yes
Config                                   <6933 bytes>
Environment                              <0 in submode>
```

```

Exclude list
[bright90->kubernetes[default]->appgroups[system]->applications[dashboard]]% get config
# Copyright...
...
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard

---

apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard

---
...

```

Bright Cluster Manager then applies the custom YAML files using the Kubernetes API server. The user `test` should then be able to read pods, while for other resources the user should have no access:

```

[test@bright90 ~]$ module load kubernetes/default/1.16
[test@bright90 ~]$ kubectl get pods -n default
NAME                                READY    STATUS    RESTARTS   AGE
test-58bd44785c-7f4lm               1/1     Running   0           13m

```

Revoking these rights can be achieved via `cmsh` as well:

```

[bright90->kubernetes[default]->rolebinding[test]]% set enabled no
[bright90->kubernetes*[default*]->rolebinding*[test*]]% commit
...
[test@bright90 ~]$ kubectl get pods -n default
Error from server (Forbidden): pods is forbidden: User "test" cannot list resource\
"pods" in API group "" in the namespace "default"
[test@bright90 ~]$

```

If there is an error in the YAML configuration for some reason, then an error should show up in `cmsh` after committing:

```

[bright90->kubernetes[default]->rolebinding[test]]% commit
Commit user 'test' ...
Wed Jan 15 15:59:16 2020 [warning] bright90: Error applying role binding with\
  kubectl for user: test
For details type: events details 569
[bright90->kubernetes[default]->rolebinding[test]]%

```

If certain resources are removed from the role binding YAML, then Bright Cluster Manager detects the difference, and makes sure that resources that are no longer present in the new YAML are also removed from Kubernetes.

9.3.12 Enabling And Disabling Pod Security Policies for Kubernetes

Enabling Pod Security Policies for Kubernetes

By default Kubernetes has very few restrictions on users. For more fine-grained authorization it is possible to enable Pod Security Policies (PSP). This can be done via the `cm-kubernetes-setup` Ncurses wizard and choosing “Enable PSP”. Alternatively, it can be done non-interactively using `cm-kubernetes-setup -psp`.

Optionally AppArmor can be enabled as well, for which the non-interactive command `-apparmor` can be provided.

Enabling PSP creates two new Applications within the Kubernetes AppGroup “system”:

- `psp`: this defines the policy and roles for privileged services. The cluster administrator needs to bind these to services typically running in the `kube-system` namespace.
- `psp_system`: this is auto-generated by the wizard. It binds the previously-mentioned privileges to service accounts, for services defined in the system namespaces. This way Calico, CoreDNS, Ingress, and so on, can still function.

The wizard also removes access to the default namespace for existing users, and it restarts the Kubernetes API Server with the PodSecurityPolicy feature enabled to enforce all privileges.

Each user should already have their own `<user>-restricted` namespace and privileges to work within this namespace. After the Kubernetes API Server is enabled with the PodSecurityPolicy feature, these policies are enforced after the API server has restarted.

A useful command to check what exactly any given user, for example `test`, can do is the following:

```
kubectl --kubeconfig=/home/test/.kube/config auth can-i --list --namespace=test-restricted
```

Disabling Pod Security Policies For Kubernetes

Disabling PSP can be done via the `cm-kubernetes-setup` Ncurses wizard, and choosing “Disable PSP”. It can alternatively be carried out non-interactively using `cm-kubernetes-setup -disable-psp`.

When disabling, it should be noted that:

- Existing users are not be automatically re-added to the default namespace.
- Policies are still defined as resources, but are no longer enforced. This may result in more privileges for users than they had before (that is, they may be able to run as root in containers again)

9.3.13 Providing Access To External Users

To provide access to users on an external network, the requirements are:

- for `kubectl`, an entry in the company/internal DNS server should resolve the external FQDN to the head node or to one of the nodes where Kubernetes is running;
- for the Kubernetes Dashboard, `dashboard` is a subdomain that must be included as a DNS entry under the external FQDN.

The external FQDN, which is set during the Kubernetes cluster setup, is the first item in the list of trusted domains. This can be retrieved from the Kubernetes cluster entity with `cmsh` as follows:

Example

```
[bright90->kubernetes[default]]% get trusteddomains
bright90.example.com
kubernetes
kubernetes.default
kubernetes.default.svc
master
localhost
```

In the preceding example, the FQDN of the cluster is `bright90.example.com`. The cluster administrator managing their own cluster will have another FQDN, and not this FQDN.

For `kubect1`, the Kubernetes API server proxy port should be open to the external network. The proxy port can be retrieved from the Kubernetes cluster entity as follows:

```
[bright90->kubernetes[default]]% get kubernetesapiserverproxyport
10443
```

For the Kubernetes Dashboard, the Ingress Controller HTTPS port should be open to the external network. This port can be retrieved from the `ingress_controller` addon environment:

Example

```
[bright90->kubernetes[default]]% appgroups
[bright90->kubernetes[default]->appgroups]% applications system
[bright90->kubernetes[default]->appgroups[system]->applications]% environment ingress_controller
[bright90->...applications[ingress_controller]->environment]% list
```

Name (key)	Value	Nodes environment
CM_KUBE_EXTERNAL_FQDN	bright90.example.com	yes
CM_KUBE_INGRESS_HTTPS_PORT	30443	yes
CM_KUBE_INGRESS_HTTP_PORT	30080	yes
ingress_controller_label	brightcomputing.com/ingress-controller	no
replicas	1	no

If exposing the Kubernetes API server to the external network is selected during setup with `cm-kubernetes-setup`, then the HTTPS and HTTP ports in the preceding example are opened on the Shorewall service that runs on the head node. Exposure to the external network is enabled by default.

Convention of using a domain name as a prefix label: in the preceding example, the `brightcomputing.com` prefix that is part of the value for `ingress_controller_label` is just a label rather than a domain. The reason that prefix is used is that it simply follows the convention of using domain names as labels, such as is done by the Kubernetes community (domain: `kubernetes.io`) and RHEL OpenShift (domain: `openshift.io`). The prefix `brightcomputing.com` could equally well have been the prefix `brightaccess` instead. However it is probably less confusing now to follow the established convention. So that is what is done here for the label.

Users can access the Dashboard using `dashboard` along with the FQDN and the port value, in a URL that has the form:

```
https://dashboard.<CM_KUBE_EXTERNAL_FQDN>:<CM_KUBE_INGRESS_HTTPS_PORT>
```

So, for example, it could be something like:

Example

```
https://dashboard.bright90.example.com:30443
```

Ingress configuration for Dashboard in `cmsh`: The default Ingress rule described earlier can be found as an object within `cmsh`:

```
[bright90->kubernetes[default]->appgroups[system]->applications[dashboard_ingress]]% get config
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
```



```

name: kubernetes-dashboard
namespace: kubernetes-dashboard
annotations:
  kubernetes.io/ingress.class: "nginx"
  nginx.ingress.kubernetes.io/secure-backends: "true"
  nginx.ingress.kubernetes.io/ssl-passthrough: "true"
  nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  rules:
  - host: "dashboard.$CM_KUBE_EXTERNAL_FQDN"
    http:
      paths:
      - path: /
        backend:
          serviceName: kubernetes-dashboard
          servicePort: 443

```

Using `kubectl`, the Ingress resource can be found with:

```

bash$ kubectl get ingress -n kubernetes-dashboard
NAME                                HOSTS                                ADDRESS          PORTS   AGE
kubernetes-dashboard               dashboard.cluster1.local            10.150.153.251  80     45h

```

The official documentation for Ingress, at <https://v1-16.docs.kubernetes.io/docs/concepts/services-networking/ingress/>, explains it well. Path rewrites without domain names can also be used to set up Ingress with multiple backends (serviceName and servicePort pairs), without having to deal with setting up a DNS.

Ingress Controller running on compute nodes: For scenarios where the head node is not involved in a Kubernetes setup, Bright Cluster Manager does not currently set up any forwarding for the Ingress Controller. Bright Cluster Manager does set up an NGINX proxy to expose the Kubernetes API Server in such cases, and accessing the Dashboard can then be done with the `kubectl proxy` approach.

For now a workaround to forward Ingress to a compute node can be achieved with port-forwarding, for example by adding the following line to `/etc/shorewall/rules` in Shorewall:

Example

```

DNAT    net          nat:10.141.0.1:30443  tcp    30443

```

Using one Ingress controller for multiple Kubernetes clusters: Bright Cluster Manager does not offer an out-of-the-box solution for one Ingress Controller with multiple Kubernetes clusters. This configuration can be achieved by configuring software such as NGINX to proxy, based on the domain name to the appropriate backend(s).

9.3.14 Networking Model

Kubernetes expects all pods to have unique IP addresses, which are reachable from within the cluster. This can be implemented in several ways, including adding pod network interfaces to a network bridge created on each host, or by using 3rd party tools to manage pod virtual networks.

Since Bright Cluster Manager 9.0, the pod network provider is Calico (<https://www.projectcalico.org/>). Calico uses the Border Gateway Protocol (BGP) to distribute routes for every Kubernetes pod. This allows the Kubernetes cluster to be integrated without the need for overlays (IP-in-IP). Calico is particularly suitable for large Kubernetes deployments on bare metal, or in private clouds. This is because for larger deployments the performance and complexity costs of overlay networks can become significant.

9.3.15 Kubernetes Monitoring

When `cm-kubernetes-setup` is run, it configures the following Kubernetes-related health checks:

1. `KubernetesChildNode`: checks if all the expected agents and services are up and running for active nodes
2. `KubernetesComponentsStatus`: checks if all the daemons running on a node are healthy
3. `KubernetesNodesStatus`: checks if Kubernetes nodes have a status of `Ready`
4. `KubernetesPodsStatus`: checks if all the pods are in one of these states: `Running`, `Succeeded`, or `Pending`

9.3.16 Setup Of A Storage Class For Ceph

Pods running on Kubernetes can use Ceph as a distributed storage system to store data in a persistent way.

Instead of creating Kubernetes `PersistentVolumes` every time, a modern and practical way is using the `StorageClass` feature.

Further documentation on `StorageClass` is available at:

- <http://blog.kubernetes.io/2016/10/dynamic-provisioning-and-storage-in-kubernetes.html>
- <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#storageclasses>

This section assumes a working Ceph cluster. Ceph installation for Bright Cluster Manager is covered in Chapter 4 of the *OpenStack Deployment Manual*.

A new pool `kube` can be created with a replication factor of 3:

Example

```
[root@bright90 ~]# ceph osd pool create kube 100 100
pool 'kube' created
[root@bright90 ~]# ceph osd pool set kube size 3
set ppol 1 size to 3
[root@bright90 ~]# ceph osd pool set kube min_size 1
set pool 1 min_size to 1
```

The parameters settings in the preceding example are documented at the Ceph website, at

- <http://docs.ceph.com/docs/master/rados/operations/pools/> for documentation on Ceph operations
- <http://docs.ceph.com/docs/master/rados/configuration/pool-pg-config-ref/> for documentation on Ceph pool and PG (placement group) configuration

The pods of a given namespace have to have access to the Ceph RBD images created to back the volumes.

A kube client can be created with:

Example

```
[root@bright90 ~]# ceph auth get-or-create client.kube mon 'allow r' osd 'allow rwx pool=kube'
[client.kube]
    key = AQCn0vdZpYewBBAAWv1d7c7/XbEvj7Q07N0THg==
```

A list of the current users, and their access control can be viewed with (some output elided):

Example

```
[root@bright90 ~]# ceph auth list
installed auth entries:

osd.0
    key: AQC9M/dZw8HPNRAAT+X8mGSgRUKjLnQo38j4EA==
    caps: [mon] allow rwx
    caps: [osd] allow *

osd.1
...
client.admin
    key: AQCnM/dZONOPMxAAwqY9ADbJV+6i2Uq/ZNqh5A==
    auid: 0
    caps: [mds] allow *
    caps: [mgr] allow *
    caps: [mon] allow *
    caps: [osd] allow *
...
client.kube
    key: AQCnOvdZpYewBBAAwv1d7c7/XbEvj7Q07NOTHg==
    caps: [mon] allow r
    caps: [osd] allow rwx pool=kube
```

The admin user must be able to create images in the pool. The admin configuration must therefore look like the section for `client.admin` in the preceding example.

Similarly, the kube user must be able to map images. The kube configuration must therefore look similar to the section for `client.kube` in the preceding example.

A Kubernetes secret must be created in the `kube-system` namespace, using the Ceph admin key:

```
[root@bright90 ~]# kubectl create secret generic ceph-secret --type="kubernetes.io/rbd" \
--from-literal=key=$(ceph auth get-key client.admin) --namespace=kube-system
secret "ceph-secret" created
```

A Kubernetes secret must be created in the default namespace, and in every Kubernetes namespace that needs storage, using the Ceph user key:

```
[root@bright90 ~]# kubectl create secret generic ceph-secret-user --type="kubernetes.io/rbd" \
--from-literal=key=$(ceph auth get-key client.kube) --namespace=default
secret "ceph-secret-user" created
```

Ceph monitor `<IP address>:<port>` values can be found by running `ceph mon stat`:

Example

```
[root@bright90 ~]# ceph mon stat
e1: 3 mons at {node001=10.141.0.1:6789/0,node002=10.141.0.2:6789/0,node003=10.141.0.3:6789/0}, \
election epoch 38, quorum 0,1,2 node001,node002,node003
```

A `storage-class.yml` file can then be created, similar to:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.141.0.1:6789,10.141.0.2:6789,10.141.0.3:6789
  adminId: admin
```

```
adminSecretName: ceph-secret
adminSecretNamespace: kube-system
pool: kube
userId: kube
userSecretName: ceph-secret-user
```

Details about the StorageClass parameters can be found at: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#ceph-rbd>

The Kubernetes storage class for Ceph RBD can now be created:

```
[root@bright90 ~]# kubectl apply -f storage-class.yml
storageclass "fast" created
```

To verify it has been created, the new StorageClass can be listed with:

```
[root@bright90 ~]# kubectl get sc
NAME          PROVISIONER
fast          kubernetes.io/rbd
```

9.3.17 Integration With Harbor

In order to spawn pods that use images from the Harbor registry, a secret must first be created with the credentials:

```
[root@bright90 ~]# kubectl create secret docker-registry myregistrykey \
--docker-server=node001:9443 --docker-username=admin --docker-password=Harbor12345
```

The secret must then be referenced from the pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: foo
spec:
  containers:
  - name: foo
    image: node001:9443/library/nginx
    imagePullSecrets:
    - name: myregistrykey
```

Further information on this is available at <https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod>.

9.4 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. Bright Cluster Manager provides Singularity version 3.4.2.

9.4.1 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.

9.4.2 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 is able to run a Singularity image only if the Singularity package is installed on the node. In order to allow users to build an image, it makes sense to install the package on the head and login nodes as well. The tool does not provide services that run in the background, so a simple installation of the package is enough to start using it.

Singularity contexts are always run as the user running them. This means that there is no risk in allowing the containers to have access to, and interact with, the file system of the host.

This means that, if an image is created by the root user on a machine, then the files that require root access inside the image, still need to be allowed root permissions on any other machine. Thus, if a user creates an image on a laptop, and adds a file that can be read only by the root user, then when the container is started on another machine by a regular user, that regular user has no access to the root-only readable file inside the container.

While there is no daemon running as root, nor any persistent processes that an attacker may use to escalate privileges, there is a need to run some system calls as root so that the container is encapsulated. For this part of the run flow, there is a single SUID binary called `Sexec` (Singularity Exec). This is a simple binary that is as small as possible, and which the Singularity developers claim has been audited by multiple security experts.

9.4.3 MPI Integration

Because of the nature of Singularity, all MPI implementations should work fine inside a Singularity container. The developers of the tool have spent a lot of effort in making Singularity aware of Open MPI, as well as adding a Singularity module into Open MPI so that running at extreme scale is as efficient as possible. However, in some cases, starting an MPI process may not be as optimal as execution outside the container. So, specifically for Open MPI, Singularity provides a special mechanism to handle the execution of MPI processes. It adds all the MPI processes of the same MPI application to the same container on a host. This also reduces the application startup time. The Open MPI daemon `orted` in this case is not added to the running container, which means the overhead of starting up daemons is reduced.

When an Open MPI application that has been packaged to an image is started, the following steps take place:

1. `mpirun` is called;
2. `mpirun` forks and executes `orted`;
3. `orted` initializes the PMI (process management interface);
4. `orted` forks as many times as the number of processes per node requested;
5. the container image is started in each fork (because it is the original command specified in `mpirun` arguments);

6. each container process executes the command (that is, the MPI application) passed inside the given container;
7. each of the MPI process links to the dynamic Open MPI library, which loads shared libraries with `dlopen` system call;
8. Open MPI libraries connect back to the original orted process via PMI;
9. all non-shared memory communication then occurs through the PMI, and then passes on to local network interfaces.

Additional information about Singularity usage can be found in Chapter 11 of the *User Manual*. The official web site of the tool is <https://www.sylabs.io/singularity>.

9.5 Kubernetes Apps

Kubernetes addons were introduced in Bright Cluster Manager version 8.1, and could be managed in that version as part of the addons submode of the `kubernetes` mode in `cmsh`. In Bright Cluster Manager version 8.2 this feature was expanded into the *Kubernetes Applications & Groups* feature. Kubernetes Applications & Groups, less formally called app groups, can be accessed via the `appgroups` submode of `cmsh`:

Example

```
root@bright90 ~# cmsh
[bright90]% kubernetes
[bright90->kubernetes[default]]% appgroups
[bright90->kubernetes[default]->appgroups]% list
Name (key)   Applications
-----
system      <13 in submode>
[bright90->kubernetes[default]->appgroups]%
```

The version 8.1, addons mode parameters are now accessed from version 8.2 onwards via a default system app group instance. The system instance is accessed in the `appsgroup` submode.

Example

```
[bright90->kubernetes[default]->appgroups]% use system
[bright90->kubernetes[default]->appgroups[system]]% show
Parameter          Value
-----
Name                system
Revision
Enabled             yes
applications        <13 in submode>
[bright90->kubernetes[default]->appgroups[system]]% applications
[bright90->kubernetes[default]->appgroups[system]->applications]% list
Name (key)         Format Enabled
-----
bootstrap         Yaml   yes
calico             Yaml   yes
dashboard         Yaml   yes
dashboard_ingress Yaml   yes
dns               Yaml   yes
ingress_controller Yaml   yes
kubernetes_ingress Yaml   no
```

metrics_server	Yaml	yes
nvidia	Yaml	no
root	Yaml	yes
spark-auth	Yaml	yes

A Kubernetes application can span multiple namespaces. A name in appgroups therefore only exists to group logically-related applications. Each application contains a YAML configuration file, which Bright Cluster Manager synchronizes to the Kubernetes API.

The default system app group is pre-defined. Other app groups can be created as needed. For example, an app group called `monitoring` could be created to group applications for running Prometheus, node exporters, and anything else related to exposing or viewing Prometheus metrics.

Toggling the `Enable` parameter of an app group enables or disables all of its application components in Kubernetes. Finer-grained control is possible within the applications mode level, by toggling the `enabled` parameter per application component instance. For example, within the `calico` application component instance:

Example

```
[bright90->kubernetes[default]->appgroups[system]->applications]% use calico
[bright90->kubernetes[default]->appgroups[system]->applications[calico]]% show
Parameter                               Value
-----
Name                                     calico
Revision
Format                                    Yaml
Enabled                                   yes
Config                                    <25118 bytes>
Environment                               <3 in submenu>
Exclude list                              /etc/cni,/opt/cni,/run/calico,/var/lib/calico
```

A large YAML configuration file for each application component instance can be configured via the `Config` parameter property, using the `set` option of `cmsh`. This opens up a text editor and allows the environment variables in the YAML configuration file to be managed.

Exclude list entries can be set via the `Exclude list` parameter property to prevent Bright Cluster Manager software image updates from overwriting files or directories important to the given application.

Environment entries can be set via the `Environment` submenu. Environment entries are similar to environment variables, and are used to replace variables inside the YAML configuration file. Environment entries can be added to the environment as well, if the `Nodes environment` value inside the `Environment` submenu is set to `yes`.

Example

```
[bright90->kubernetes[default]->appgroups[system]->applications[calico]]% environment
[bright90->kubernetes[default]->appgroups[system]->applications[calico]->environment]% list
Name (key)                               Value                               Nodes environment
-----
calico_typha_replicas                     0                                   no
calico_typha_service                      none                                no
head_node_internal_ip                     10.141.255.254                     no
```

In Bright Cluster Manager 9.0, Spark can be installed as a Kubernetes application using `cm-spark-setup`. This differs from how `cm-spark-setup` worked in version 8.1, where Spark was only installable in a “traditional”, non-containerized way, rather than as a “Spark on Kubernetes” (section 9.6) implementation.

9.6 Spark On Kubernetes

The Apache Spark project is moving towards running Spark on Kubernetes. This adds a dependency on Kubernetes. On the other hand, there is no longer a need to have Spark Master and Worker services running continuously. Instead, Spark submits jobs directly to the Kubernetes API. Everything needed to execute the Spark workload then spins up on demand inside Kubernetes. Tearing down Kubernetes pods when they are no longer needed is also carried out automatically.

Official documentation for Spark on Kubernetes does a good job of explaining its architecture. For version 2.4.1, which is the Spark release version supported by Bright Cluster Manager 9.0, this documentation is at <https://spark.apache.org/docs/2.4.1/running-on-kubernetes.html>.

Bright Cluster Manager has been supporting Spark on Kubernetes since version 2.4.0 onwards, and has included support for the following features since that version:

- Support for containerized PySpark and SparkR applications on Kubernetes
- Client mode support that allows users to run interactive applications and notebooks
- Support for mounting certain types of Kubernetes volumes

By default, to run JAR- and/or Python-based jobs, two default containers are provided by Bright Cluster Manager via Docker Hub at:

- <https://hub.docker.com/r/brightcomputing/spark/>
- <https://hub.docker.com/r/brightcomputing/spark-py/>

The cluster administrator can also create custom containers by using a provided helper script (`/cm/shared/apps/spark/current/bin/docker-image-tool.sh`) that comes with Spark on Kubernetes. This is useful when making extra Scala or Python libraries available for the application. The helper script usage is explained in section 10.0.4 of the *User Manual*.

The containers should be pushed to a Docker registry before they are run inside Kubernetes. The registry can be Docker Hub, or it can be a local docker registry. To deploy a registry with Bright Cluster Manager 9.0, the `cm-docker-registry-setup` script (section 9.2) can be used.

Section 9.6.2 discusses granting users access to the docker CLI on the cluster, for creating Spark images.

9.6.1 Deploying Spark On Kubernetes

If Kubernetes has been deployed (section 9.3.2), then Spark on Kubernetes can be deployed by running the Bright Cluster Manager `cm-spark-setup` Ncurses wizard. The wizard is provided by the `cm-setup` package.

The `cm-spark-setup` setup wizard

If run with the `-h` option a short help text is displayed:

```
[root@bright90 ~]# cm-spark-setup -h

Please wait...
#####
##### START #####
#####

usage: Spark Setup cm-spark-setup [-v] [-h] [-c <config_file>] [--dev]
                                   [--tests] [--undo-on-error]
                                   [--on-error-action {debug,remotedebug,undo,abort}]
                                   [--output-remote-execution-runner] [--json]
                                   [--no-distro-checks] [--remove] [--yes]
```



```
[--skip-reboot]
```

common:

Common arguments

```
-v                Verbose output
-h, --help       Print this screen
-c <config_file> Load runtime configuration for plugins from a YAML config file
```

advanced:

Various *advanced* configuration options flags.

```
--dev            Enables additional command line arguments
--tests         Test integrity of the utility by running Unit Tests
--undo-on-error  Upon encountering a critical error, instead of asking the user
                 for choice, setup will undo (revert) the deployment stages.
--on-error-action {debug,remotedebug,undo,abort}
                 Upon encountering a critical error, instead of asking the user
                 for choice, setup will undo (revert) the deployment stages.
--output-remote-execution-runner
                 Format output for CMDaemon
--json          Use json formatting for log lines printed to stdout
--no-distro-checks Disable distribution checks based on ds.json
```

removing spark deployment:

Flags which can be used for removing spark

```
--remove        Remove every instance of spark
--yes          Force removal without asking for confirmation
--skip-reboot   Don't reboot the nodes
```

```
##### END #####
```

The wizard can be run with a configuration file, using the `-c` option. Such a configuration file can be generated from an Ncurses-based run.

To run the wizard in Ncurses mode it is run without options. The user is then guided through various installation screens until Spark configuration is completed, and deployment can then begin.



Figure 9.4: `cm-spark-setup` main operations screen

Example Installation Run

A `cm-spark-setup` deployment session displays something similar to:

Example

```
[root@bright90 ~]# cm-spark-setup -c /root/cm-spark-setup.conf
##### Starting execution for 'Spark Setup'
- spark
## Progress: 0
#### stage: spark: Search Kube Clusters
Connecting to CMDaemon
## Progress: 16
#### stage: spark: Collection Packages Installer
## Progress: 33
#### stage: spark: spark-auth
## Progress: 50
#### stage: spark: Show Example Usage

-----
The following commands can be used to test drive Spark on k8s:
-----

module load kubernetes/default
module load spark

spark-submit \
  --master k8s://https://localhost:10443 \
  --deploy-mode cluster \
  --name spark-pi \
  --class org.apache.spark.examples.SparkPi \
  --conf spark.executor.instances=5 \
  --conf spark.kubernetes.container.image=docker.io/brightcomputing/spark:2.4.1 \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
  local:///opt/spark/examples/jars/spark-examples_2.11-2.4.1.jar

-----

## Progress: 66
#### stage: spark: Search Jupyter
## Progress: 100

Took:    00:37 min.
Progress: 100/100
##### Finished execution for 'Spark Setup', status: completed

Spark Setup finished!
```

In the preceding output, Jupyter/JupyterHub is not found and therefore no PySpark kernel is deployed.

Test Driving Spark On K8s

Towards the end of the preceding installation output display, an example command is displayed that can be used to test the deployment. A test driver session using that command displays something similar to the following (some output elided):

Example

```
[root@bright90 ~]# module load kubernetes/default
[root@bright90 ~]# module load spark
[root@bright90 ~]# spark-submit \
>   --master k8s://https://localhost:10443 \
>   --deploy-mode cluster \
```

```

> --name spark-pi \
> --class org.apache.spark.examples.SparkPi \
> --conf spark.executor.instances=5 \
> --conf spark.kubernetes.container.image=docker.io/brightcomputing/spark:2.4.1 \
> --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
> local:///opt/spark/examples/jars/spark-examples_2.11-2.4.1.jar
2019-05-06 12:28:48 INFO LoggingPodStatusWatcherImpl: - State changed, new state:
pod name: spark-pi-1557138166085-driver
namespace: default
labels: spark-app-selector -> spark-152fe4dea93747908c990c4c97da133f, spark-role -> driver
pod uid: e58702ff-6fe8-11e9-ab66-fa163e53895f
creation time: 2019-05-06T10:22:48Z
service account name: spark
volumes: spark-local-dir-1, spark-conf-volume, spark-token-xrvvq
node name: N/A
start time: N/A
container images: N/A
phase: Pending
status: []
....
19/05/06 12:24:44 INFO Client: Application spark-pi finished.
19/05/06 12:24:44 INFO ShutdownHookManager: Shutdown hook called
19/05/06 12:24:45 INFO ShutdownHookManager: Deleting directory /tmp/spark-f2676f82-d783-4d4a-abbe-27c00d2350d9

```

Kubernetes activity can be viewed during the deployment test drive:

Example

```

[root@bright90 ~]# kubectl get pod
NAME                                READY   STATUS             RESTARTS   AGE
spark-pi-1557144583970-driver       1/1     Running            0           13s
spark-pi-1557144583970-exec-1       0/1     ContainerCreating  0           3s
spark-pi-1557144583970-exec-2       0/1     Pending            0           3s
...

```

Depending on the state of the submitted job, the output can be quite different.

In the test drive, a driver pod is first created. This in turn spawns five executors for the submitted job, as specified by the `spark-submit` command option: `-conf spark.executor.instances=5`.

The first-time container images still need to be downloaded. The `ContainerCreating` status may therefore take a while to change status. After the Spark job runs and finishes, the output might look similar to:

Example

```

[root@bright90 ~]# kubectl get pod
NAME                                READY   STATUS             RESTARTS   AGE
spark-pi-1557144583970-exec-5       0/1     Terminating      0           1m1s
spark-pi-1557144583970-driver       0/1     Completed          0           1m33sm

```

The driver pod remains available for a while, so that the job output can be inspected. Logs can be inspected with a command such as, for example: `kubectl logs spark-pi-1557144583970-driver`.

For longer-running jobs, the Spark UI can be accessed using `kubectl port-forward`. This is explained in more detail on page 86 of the *User Manual*.

9.6.2 Providing Custom Docker Images

Allowing users to work with custom Docker images on the cluster requires adding the user to the docker group. This can be carried out with `usermod -aG docker <user>`.

How to use the scripts provided by Apache Spark to build a custom image is described in section 10.0.4 of the *User Manual*.

9.7 Kubernetes On Edge

How edge sites can be configured is described in Chapter 2 of the *Edge Manual*.

If there are Bright Edge sites configured in the cluster, then the Kubernetes setup prompts the user with edge sites that Kubernetes can be deployed on.

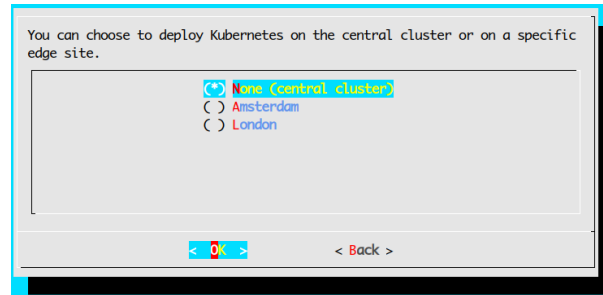


Figure 9.5: `cm-kubernetes-setup` prompting for edge sites.

If an edge site is selected, then the rest of the wizard prompts only for nodes available within that edge site; prompts only for the associated network interfaces; and so on.

9.7.1 Flags For Edge Installation

Edge directors often lack high-bandwidth connectivity to the central head node, or they often may benefit from coming up as quickly as possible. It can therefore sometimes be useful to skip stages of the setup.

Running `cm-kubernetes-setup --help` displays some additional flags that allow some setup stages, that bring up a cloud director, to be skipped explicitly:

```
cm-kubernetes-setup --help
...
installing Kubernetes clusters:
  Flags for installing or managing Kubernetes clusters

  --skip-package-install
                        Skip the package installation steps. Ignores skip_packages
                        flags in the config.
  --skip-reboot         Skip the reboot steps.
  --skip-image-update   Skip the image update steps.
  --skip-disksetup-changes
                        Never change the disk-setup. Use this flag if you manually
                        configure a partition or device for docker thin pool devices
                        for example.
...
```

Speeding Up Kubernetes Installation To Edge Nodes With The `--skip-*` Flags: Use Cases

Explanations and use cases for these flags are given in the following table:

Flag	Use case
<code>--skip-package-install</code>	all edge directors share the same software image, and the image is already up to date. So the installer does not need to install packages from that image to the edge director.
<code>--skip-image-update</code> and <code>--skip-reboot</code>	all edge directors are already provisioned with the up-to-date software image. So the installer does not need to carry out an update from the ISO or head node, and then reboot the edge director.
<code>--skip-disksetup-changes</code>	all edge directors already have the correct disk layout. This flag can be set if the disk layout was already configured up-front, in order to avoid full provisioning.

These flags can also be configured in the YAML configuration file of the `cm-kubernetes-setup-wizard`.

The flags can be used for scripted installations for quick Kubernetes setups. For a scripted installation of an edge director, preparations can be done beforehand so that all the requirements in the software images that the edge directors use are already installed, the right disk layouts are already configured, and packages are already updated.

All the stages in the flag options can then be skipped for installing onto edge sites. This can make the setup take just a few seconds per Kubernetes deployment.

10

BeeGFS

10.1 BeeGFS Introduction

10.1.1 BeeGFS Concepts

BeeGFS is a high-performance parallel file system, developed by the Fraunhofer Competence Center for High Performance Computing, and optimized for intensive I/O. It uses a distributed metadata architecture designed for scalability and flexibility. More information about BeeGFS can be found at the official online documentation at <https://www.beegfs.io/content/documentation/>.

Bright Cluster Manager provides packages to allow BeeGFS to be deployed, managed, and monitored on a Bright cluster running RHEL or Centos. The deployment tool provided by Bright Cluster Manager is `cm-beegfs-setup` (section 10.2).

A BeeGFS cluster consists of a management server, one or more metadata servers and one or more storage servers. Clients should have client servers running on them.

If high-availability is required, then the administrator needs to arrange for it separately, for the BeeGFS management servers and Admon node.

Since BeeGFS is a parallel filesystem, it means that adding more nodes increases not only the storage capacity, but also system performance. BeeGFS can work with any local file system for data storage as long as it is POSIX compliant.

Versions of BeeGFS before 7.2 have an optional graphical user interface called Admon (section 10.3.4), which is integrated with Bright Cluster Manager. Support for version 7.2 of BeeGFS was introduced in Bright Cluster Manager 9.1, and Admon is no longer supported since that version.

10.1.2 BeeGFS Installation Notes And Options

By default all logging is done to the systemd journal. The installation logs can be viewed using:

```
journalctl -u 'beegfs-*'.
```

If Admon is used, then it cannot use journalled logs by default. Logging into a file is needed instead.

Log locations can be changed using `cmsh` or Bright View at any moment after installation, by editing BeeGFS roles for the corresponding configuration overlays (section 10.2.1). After the changes are made, the Admon service must be manually restarted on the Admon node (section 10.2.1):

Example

```
systemctl restart beegfs-admon.service
```

Authentication is possible for the BeeGFS cluster via a shared secret file.

BeeGFS can be installed with these options:

- Admon (for versions of BeeGFS before 7.2)
- Shared secret file

10.2 Deployment And Uninstallation Of BeeGFS With `cm-beegfs-setup`

Deployment and uninstallation for a Bright Cluster Manager can be carried out with the Ncurses-based `cm-beegfs-setup` utility. The utility is a part of a `cluster-tools` package that comes with Bright Cluster Manager. If the BeeGFS packages are not installed, then the utility installs them.

The `cm-beegfs-setup` utility can be run as root from the head node. The first Ncurses screen that shows up is the main menu for operations (figure 10.1)

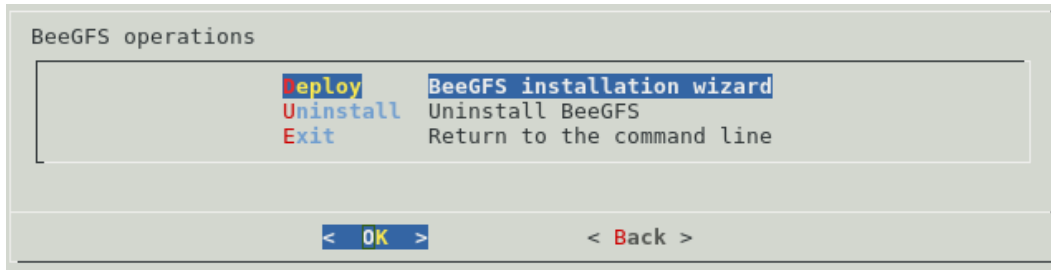


Figure 10.1: BeeGFS Script Main Menu

In the main menu, the administrator can choose to:

- Deploy BeeGFS (section 10.2.1)
- Uninstall BeeGFS (section 10.2.2)
- Simply exit the Ncurses session

10.2.1 Deployment Of BeeGFS

If the `Deploy` option is chosen, then the deployment wizard starts.

Configuration Overlays

The wizard firsts asks for names for the configuration overlays (figure 10.2):

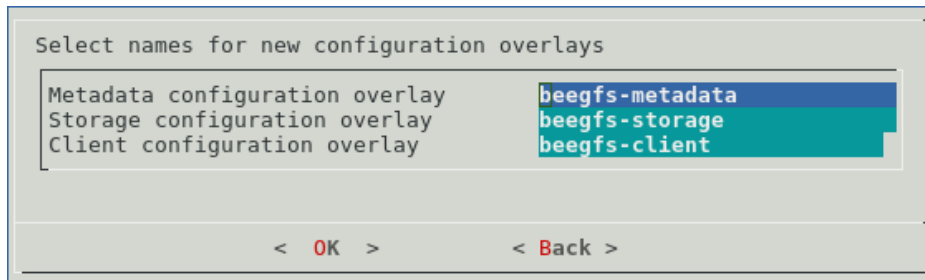


Figure 10.2: Configuration Overlays Creation

These overlay names are for :

- the BeeGFS metadata configuration overlay
- the BeeGFS storage configuration overlay
- the BeeGFS client configuration overlay

Select Management Node

The next screen (figure 10.3) then asks for a node to be selected to be the BeeGFS management server:

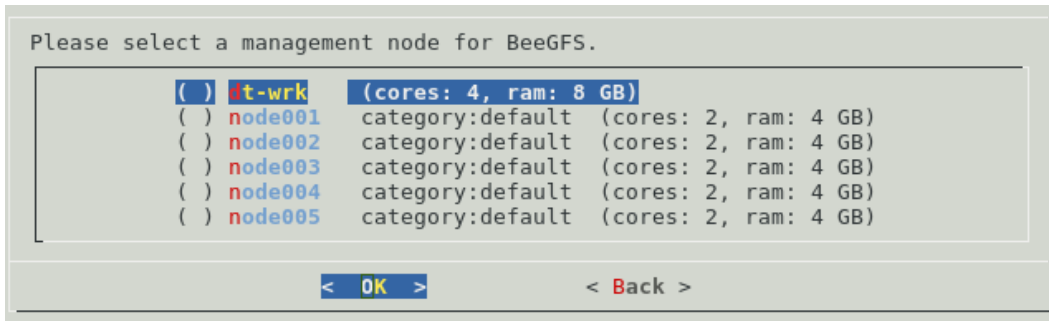


Figure 10.3: Management Node

Select Metadata Nodes

The next screen (figure 10.4) then asks for nodes to be selected to be the BeeGFS metadata servers:

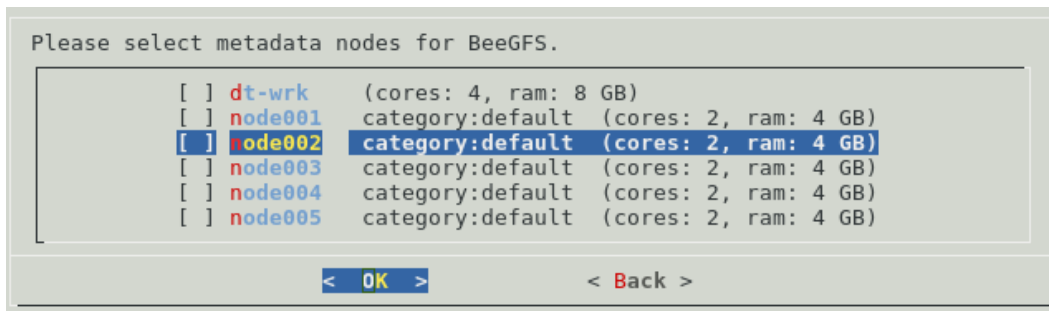


Figure 10.4: Select Metadata Nodes

Select Storage Nodes

The next screen (figure 10.5) asks for nodes to be selected to be the BeeGFS storage servers:

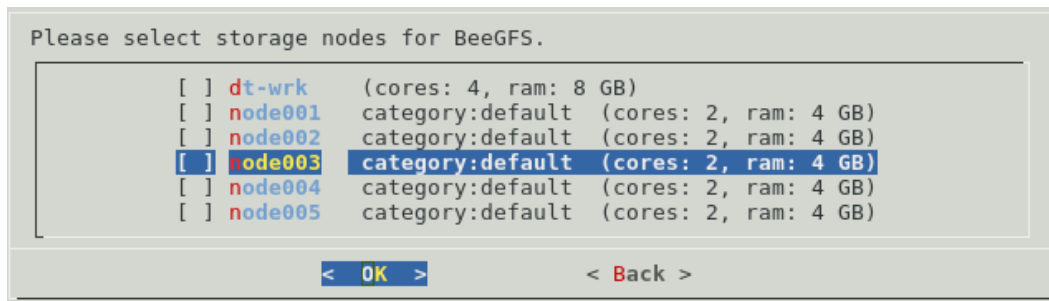


Figure 10.5: Select Storage Nodes

Select Client Nodes

The next screen (figure 10.6) asks for nodes to be selected to be the client nodes:

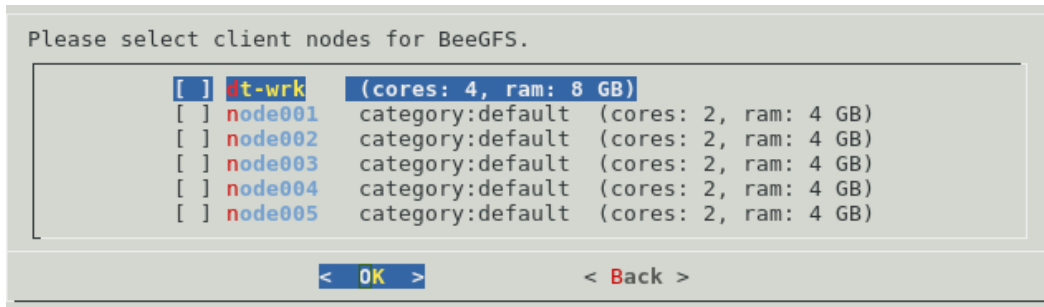


Figure 10.6: Select Client Nodes

Admon Deployment

The next screen (figure 10.7) asks if the Admon GUI for BeeGFS is to be deployed:



Figure 10.7: Admon Deployment

Select Admon Node

If Admon installation is chosen, then the wizard asks (figure 10.8) for a node to be selected for Admon use:

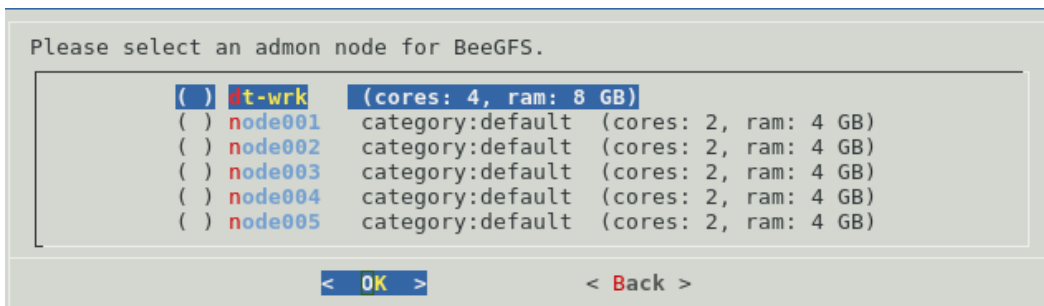


Figure 10.8: Select Admon Node

Management Server Configuration

The next screen (figure 10.9) asks for some configuration parameters for the BeeGFS management server to be set:

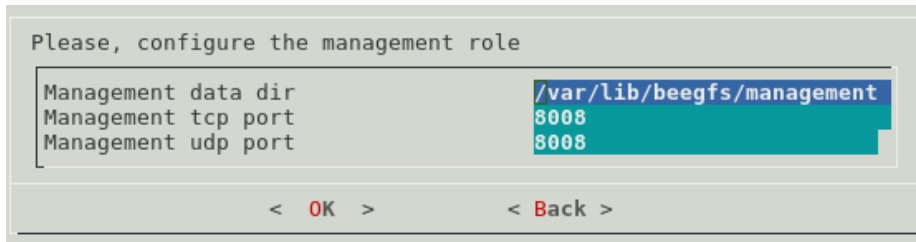


Figure 10.9: Management Server Configuration

The values required are for the:

- Path to a data directory
- Management server TCP port
- Management server UDP port

Metadata Server Configuration

The next screen (figure 10.10) asks for input on configuring the BeeGFS metadata servers:

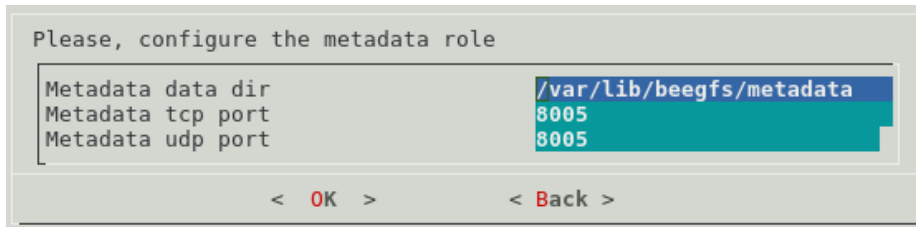


Figure 10.10: Metadata Server Configuration

The values required are for the:

- Path to a data directory
- Metadata server TCP port
- Metadata server UDP port

Storage Server Configuration

The next screen (figure 10.11) asks for input on setting some configuration parameters for the BeeGFS storage servers:

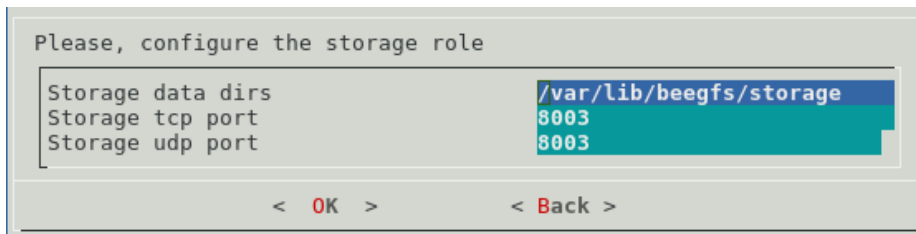


Figure 10.11: Storage Server Configuration

The items required are for the:

- Path to a data directory
- Storage server TCP port

- Storage server UDP port

Client Configuration

The next screen (figure 10.12) asks for input on configuring the BeeGFS clients:

```

Please, configure the client role
Client udp port      8004
Helper tcp port     8006
< OK >             < Back >
  
```

Figure 10.12: Client Configuration

The inputs required are for the:

- Client UDP port
- Helper TCP port

Admon Configuration

If Admon has been chosen for installation, then the wizard asks (figure 10.13) for some configuration settings for Admon:

```

Please, configure the admon role
Admon database file /var/lib/beegfs/beegfs-admon
Admon http port    8000
Admon udp port     8007
< OK >             < Back >
  
```

Figure 10.13: Admon Configuration

The input values required are:

- Path to the database file
- Admon HTTP port
- Admon UDP port

After installation is complete, the Admon jar file can be downloaded (section 10.3.4).

Authentication Configuration

In the next screen, (figure 10.14) a path must be specified for the shared secret file:

```

Please, configure the authentication
Shared secret file (optional)
< OK >             < Back >
  
```

Figure 10.14: Authentication Configuration

This is a file with arbitrary content. If it is set, then it must exist on all BeeGFS nodes.

Deployment Summary

Finally, the wizard displays a deployment summary (figure 10.15):

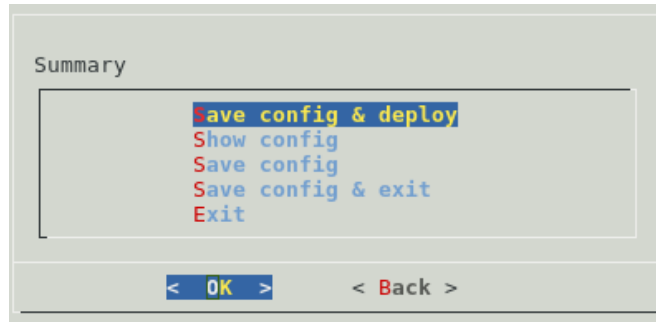


Figure 10.15: Deployment Summary

From this screen the deployment process can be started, the deployment configuration can be viewed and saved, or the wizard can simply be exited.

Command Line Installation

If a configuration file is saved from a `cm-beegfs-setup` run, then it can be used to install a BeeGFS cluster automatically:

Example

```
cm-beegfs-setup -c cm-beegfs-setup.conf
```

10.2.2 Uninstalling BeeGFS

If the Uninstall option is chosen from the main menu (figure 10.1), then the uninstallation process starts.

An extra Are you sure? prompt is thrown in the way first, to help avoid a mistaken removal. After confirmation, a BeeGFS instance must then be selected for the removal, by selecting its management node (figure 10.16):

Uninstall Management Node Selection



Figure 10.16: Uninstall Management Node Selection

The uninstallation process then starts. It can take several minutes in order to ensure that services are stopped properly, so that errors and junk files are avoided.

10.3 Managing The Deployed BeeGFS Instance

A BeeGFS cluster consists of:

- One Management node
- Some Metadata nodes
- Some Storage nodes

Information about the right number of nodes to use can be found at https://www.beegfs.io/docs/whitepapers/Picking_the_right_Number_of_Targets_per_Server_for_BeeGFS_by_ThinkParQ.pdf

BeeGFS clients should have client servers running on them.

10.3.1 Setup

Associated with BeeGFS are the following BeeGFS-related packages, most of which provide services:

1. beegfs-mgmt
2. beegfs-meta
3. beegfs-storage
4. beegfs-client
5. beegfs-helperd
6. beegfs-utils
7. beegfs-admon

The repository from which these packages are picked up is added to the repositories list when BeeGFS is installed by Bright Cluster Manager (section 10.2), and the packages are then installed on all of the cluster nodes. The services are assignable by the cluster administrator to cluster devices as roles. In `cmsh` these BeeGFS roles are treated as objects, and the configuration of these roles is determined by role parameters (object attributes). The roles are described in detail section 10.3.2.

10.3.2 BeeGFS Objects

BeeGFSManagement Role

The BeeGFSManagement role is used to configure BeeGFS management services. The role parameters are described in table 10.1:

Parameter	Description	Option to beegfs-mgmt
Data directory	Path to data directory	storeMgmtDirectory
Allow new servers	Allow new servers registration	sysAllowNewServers
Allow new storage targets	Allow new storage targets registration	sysAllowNewTargets
Authentication file	Path to shared secret authentication file	connAuthFile
Backlog TCP	TCP listen backlog size	connBacklogTCP

...continues

...continued

Parameter	Description	Option to beegfs-mgmt
Interfaces file	Path to file with interfaces list	connInterfacesFile
Management port TCP	TCP port for management service	connMgmtPortTCP
Management port UDP	UDP port for management service	connMgmtPortUDP
Net filter file	Path to file with list of allowed IP subnets	connNetFilterFile
Log level	Log level	logLevel
No log date	Do not show date along with time in log	logNoDate
Number of log lines	Number of lines in log file, after which it will be rotated	logNumLines
Number of log rotated files	Number of old log files to keep	logNumRotatedFiles
Log file	Path to log file, empty means logs go to the journal	logStdFile
Enable Quota	Enable quota	quotaEnableEnforcement
Quota query GID file	Path to file with GIDs to be checked by quota	quotaQueryGIDFile
Quota query GID range	GID range to be checked by quota	quotaQueryGIDRange
Quota query UID file	Path to file with UIDs to be checked by quota	quotaQueryUIDFile
Quota query UID range	UID range to be checked by quota	quotaQueryUIDRange
Quota query type	Query type for quota	quotaQueryType

...continues

...continued

Parameter	Description	Option to beegfs-mgmd
Quota query with system users groups	Allow also system users/groups to be checked by quota	quotaQueryWithSystemUsersGroups
Quota update interval	Quota update interval	quotaUpdateIntervalMin
Target offline timeout	Timeout for targets on storage server to be considered offline when no target status received	sysTargetOfflineTimeoutSecs
Client auto remove time	Time after which an unreachable node is considered dead	tuneClientAutoRemoveMins
Number of workers	Number of worker threads	tuneNumWorkers
Meta dynamic pools	Raise lower limits if difference in capacity becomes too large between targets	tuneMetaDynamicPools
Meta inodes low limit	Metadata inode free space pool threshold	tuneMetaInodesLowLimit
Meta inodes emergency limit	Metadata inode free space pool threshold	tuneMetaInodesEmergencyLimit
Meta space low limit	Metadata space free space pool threshold	tuneMetaSpaceLowLimit
Meta space emergency limit	Metadata space free space pool threshold	tuneMetaSpaceEmergencyLimit
Storage dynamic pools	Raise lower limits if difference in capacity becomes too large between targets	tuneStorageDynamicPools
Storage inodes low limit	Storage inode free space pool threshold	tuneStorageInodesLowLimit
Storage inodes emergency limit	Storage inode free space pool threshold	tuneStorageInodesEmergencyLimit

...continues

...continued

Parameter	Description	Option to beegfs-mgmt
Storage space low limit	Storage space free space pool threshold	tuneStorageSpaceLowLimit
Storage space emergency limit	Storage space free space pool threshold	tuneStorageSpaceEmergencyLimit

Table 10.1: BeeGFSManagement role parameters and beegfs-mgmt options

The beegfs-mgmt settings are updated by Bright Cluster Manager in `/etc/beegfs/beegfs-mgmt.conf`.

The settings can be managed in `cmsh` via the `beegfs::management` role. Roles are, as usual, assigned via `device`, `category` or `configurationoverlay` mode. The parameters of the role that correspond to the parameters in the preceding table can then be set.

Example

```
[bright90 ~]# cmsh
[bright90]% device roles node001
[bright90->device[node001]->roles]% assign bee<TAB><TAB><TAB>
beegfs::admon      beegfs::client      beegfs::management  beegfs::metadata    beegfs::storage
[bright90->device[node001]->roles]% assign beegfs::management
[bright90->device*[node001*]->roles*[BeeGFS::Management*]]% show
Parameter
-----
Value
-----
Add services          yes
Name                  BeeGFS::Management
Provisioning associations <0 internally used>
Revision
Type                  BeeGFSManagementRole
Allow new servers     yes
Data directory        /var/lib/beegfs/management
Allow new storage targets yes
Authentication file
Backlog TCP           128
Interfaces file
Management port TCP   8008
Management port UDP   8008
Net filter file
Log file
Log level             2
No log date           no
Number of log lines    50000
Number of log rotated files 5
Enable Quota          no
Quota query GID file
Quota query GID range
Quota query UID file
Quota query UID range
Quota query type      system
Quota query with system users groups no
Quota update interval 10m
```

Target offline timeout	3m
Client auto remove time	30m
Number of workers	4
Meta dynamic pools	yes
Meta inodes emergency limit	1M
Meta inodes low limit	10M
Meta space emergency limit	3G
Meta space low limit	10G
Storage dynamic pools	yes
Storage inodes emergency limit	1M
Storage inodes low limit	10M
Storage space emergency limit	20G
Storage space low limit	1T

BeeGFSMetadata Role

The BeeGFSMetadata role is used for configuring BeeGFS metadata services. The role parameters are described in Table 10.2:

Parameter	Description	Option to beegfs-meta
Data directory	Path to the data directory	storeMetaDirectory
Management node	Node with a management role	sysMgmtdHost
Authentication file	Path to the shared secret authentication file	connAuthFile
Backlog TCP	TCP listen backlog	connBacklogTCP
Fallback expiration	Time after which a connection to a fallback interface expires	connFallbackExpirationSecs
Interfaces file	Path to the file with a list of interfaces for communication	connInterfacesFile
Max internode number	Max number of simultaneous connections to the same node	connMaxInternodeNum
Meta port TCP	TCP port for metadata service	connMetaPortTCP
Meta port UDP	UDP port for metadata service	connMetaPortUDP

...continues

...continued

Parameter	Description	Option to beegfs-meta
Net filter file	Path to a file with a list of allowed IP subnets	connNetFilterFile
Use RDMA	Use RDMA	connUseRDMA
RDMA type of service	RDMA type of service	connRDMATypeOfService
Tcp only filter file	Path to a file with a list of no DRMA IP ranges	connTcpOnlyFilterFile
Log level	Log level	logLevel
No log date	Do not show date along with time in log	logNoDate
Number of log lines	Number of lines in log file, after which it will be rotated	logNumLines
Number of log rotated files	Number of old log files to keep	logNumRotatedFiles
Log file	Path to the log file, empty means logs go to the journal	logStdFile
Client xattrs	Enable client-side extended attributes	storeClientXAttrs
Client ACLs	Enable handling and storage of client-side ACLs client-side ACLs	storeClientACLs
Use extended attributes	Store metadata as extended attributes or not	storeUseExtendedAttribs
Target attachment file	File with a list of targets to be grouped within the same domain for randominternode	sysTargetAttachmentFile
Target offline timeout	Timeout for targets on a storage server to be considered offline when no target status is received	sysTargetOfflineTimeoutSecs

...continues

...continued

Parameter	Description	Option to beegfs-meta
Allow user set pattern	Allow non-privileged users to modify stripe pattern settings for directories they own	sysAllowUserSetPattern
Bind to numa zone	Zero-based NUMA zone number to which all threads of metadata process should be bound	tuneBindToNumaZone
Number of stream listeners	The number of threads waiting for incoming data events	tuneNumStreamListeners
Number of workers	Number of worker threads	tuneNumWorkers
Target chooser	The algorithm to choose storage targets for file creation	tuneTargetChooser
Use aggressive stream poll	Actively poll for events instead of sleeping until an event occurs	tuneUseAggressiveStreamPoll
Use per user msg queues	Use per-user queues for pending requests	tuneUsePerUserMsgQueues

Table 10.2: BeeGFSMetadata role parameters and beegfs-meta options

The beegfs-meta settings are updated by Bright Cluster Manager in `/etc/beegfs/beegfs-meta.conf`.

The settings can be managed in cmsh via the `beegfs::metadata` role. Roles are, as usual, assigned via device, category or configurationoverlay mode. The parameters of the role that correspond to the parameters in the preceding table can then be set.

Example

```
[bright90 ~]# cmsh
[bright90]% category roles default
[bright90->category[default]->roles]% assign bee<TAB><TAB><TAB>
beegfs::admon      beegfs::client      beegfs::management  beegfs::metadata    beegfs::storage
[bright90->category[default]->roles]% assign beegfs::metadata
[bright90->category*[default*]->roles*[BeeGFS::Metadata*]]% show
Parameter          Value
-----
Add services        yes
Name                BeeGFS::Metadata
Provisioning associations <0 internally used>
Revision
Type                BeeGFSMetadataRole
Data directory      /var/lib/beegfs/metadata
Management node
```

Authentication file	
Backlog TCP	128
Fallback expiration	15m
Interfaces file	
Max internode number	32
Meta port TCP	8005
Meta port UDP	8005
Net filter file	
RDMA type of service	0
Tcp only filter file	
Use RDMA	yes
Log file	
Log level	3
No log date	no
Number of log lines	50000
Number of log rotated files	5
Client ACLs	no
Client xattrs	no
Use extended attributes	yes
Allow user set pattern	no
Target attachment file	
Target offline timeout	3m
Bind to numa zone	
Number of stream listeners	1
Number of workers	0
Target chooser	randomized
Use aggressive stream poll	no
Use per user msg queues	no

BeeGFSStorage Role

The BeeGFSStorage role is used for configuring BeeGFS storage services. The role parameters are described in Table 10.3:

Parameter	Description	Option to beegfs-storage
Data directories	Path to the data directories	storeStorageDirectory
Management node	Node with a management role	sysMgmtHost
Authentication file	Path to the shared secret authentication file	connAuthFile
Backlog TCP	TCP listen backlog	connBacklogTCP
Interfaces file	Path to the file with a list of interfaces for communication	connInterfacesFile

...continues

...continued

Parameter	Description	Option to beegfs-storage
Max internode number	Max number of simultaneous connections to the same node	connMaxInternodeNum
Storage port TCP	TCP port for storage service	connStoragePortTCP
Storage port UDP	UDP port for storage service	connStoragePortUDP
Net filter file	Path to a file with a list of allowed IP subnets	connNetFilterFile
Use RDMA	Use RDMA	connUseRDMA
RDMA type of service	RDMA type of service	connRDMATypeOfService
Tcp only filter file	Path to a file with a list of subnets with no RDMA	connTcpOnlyFilterFile
Log level	Log level	logLevel
No log date	Do not show date along with time in log	logNoDate
Number of log lines	Number of lines in log file, after which it will be rotated	logNumLines
Number of log rotated files	Number of old log files to keep	logNumRotatedFiles
Log file	Path to the log file, empty means logs go to the journal	logStdFile
Resync safety threshold	Add an extra amount of time to the last successful communication timestamp in case of a potential cache loss	sysResyncSafetyThresholdMins
Target offline timeout	Timeout until targets on a storage server are considered offline when no target status is received	sysTargetOfflineTimeoutSecs

...continues

...continued

Parameter	Description	Option to beegfs-storage
Bind to numa zone	Zero-based NUMA zone number to which all threads of metadata process should be bound	tuneBindToNumaZone
File read ahead size	Byte range submitted to the kernel for read-ahead after number of bytes was already read from target	tuneFileReadAheadSize
File read ahead trigger size	Number of bytes read after which the read-ahead is triggered	tuneFileReadAheadTriggerSize
File read size	Maximum amount of data server should read in a single operation	tuneFileReadSize
File write size	Maximum amount of data server should write in a single operation	tuneFileWriteSize
File write sync size	Number of bytes after which kernel advised to commit data	tuneFileWriteSyncSize
Number of resync gather slaves	Number of threads to gather filesystem information for a buddy mirror resync	tuneNumResyncGatherSlaves
Number of resync slaves	Number of threads to sync filesystem information for a buddy mirror resync	tuneNumResyncSlaves
Number of stream listeners	Number of threads waiting for incoming data events	tuneNumStreamListeners
Number of workers	Number of worker threads	tuneNumWorkers
Use aggressive stream poll	Actively poll for events instead of sleeping until an event occur	tuneUseAggressiveStreamPoll
Use per target workers	Create a separate set of workers and attach it for each storage target	tuneUsePerTargetWorkers
Use per user msg queues	Use per-user queues for pending requests	tuneUsePerUserMsgQueues

...continues

...continued

Parameter	Description	Option to beegfs-storage
Worker buffer size	Size of network and io buffers, allocated for each worker	tuneWorkerBufSize

Table 10.3: BeeGFSStorage role parameters and beegfs-storage options

The beegfs-storage settings are updated by Bright Cluster Manager in `/etc/beegfs/beegfs-storage.conf`.

The settings can be managed in `cmsh` via the `beegfs::storage` role. Roles are, as usual, assigned via device, category or configurationoverlay mode. The parameters of the role that correspond to the parameters in the preceding table can then be set.

Example

```
[bright90 ~]# cmsh
[bright90]% category roles default
[bright90->category[default]->roles]% assign bee<TAB><TAB><TAB>
beegfs::admon      beegfs::client      beegfs::management  beegfs::metadata    beegfs::storage
[bright90->category[default]->roles]% assign beegfs::storage
[bright90->category*[default*]->roles*[BeeGFS::Storage*]]% show
Parameter          Value
-----
Add services        yes
Name                BeeGFS::Storage
Provisioning associations <0 internally used>
Revision
Type                BeeGFSStorageRole
Data directories    /var/lib/beegfs/storage
Management node
Authentication file
Backlog TCP         128
Interfaces file
Max internode number 12
Net filter file
RDMA type of service 0
Storage port TCP    8003
Storage port UDP    8003
Tcp only filter file
Use RDMA            yes
Log file
Log level           3
No log date         no
Number of log lines 50000
Number of log rotated files 5
Resync safety threshold 10m
Target offline timeout 3m
Bind to numa zone
File read ahead size 0m
File read ahead trigger size 4m
File read size      128k
File write size     128k
File write sync size 0m
Number of resync gather slaves 6
Number of resync slaves 12
```


Number of stream listeners	1
Number of workers	12
Use aggressive stream poll	no
Use per target workers	yes
Use per user msg queues	no
Worker buffer size	4m

BeeGFSAdmon Role

The BeeGFSAdmon role is used for configuring BeeGFS admon services. The role parameters are described in Table 10.4:

Parameter	Description	Option to beegfs-admon
Database file	Path to the database	databaseFile
Management node	Node with a management role	sysMgmtdHost
Clear database	Delete database on startup	clearDatabase
Query interval	Interval for querying servers for new statistics data	queryInterval
Authentication file	Path to the shared secret authentication file	connAuthFile
Fallback expiration	Time after which a connection to a fallback interface expires	connFallbackExpirationSecs
Interfaces file	Path to the file with a list of interfaces for communication	connInterfacesFile
Max internode number	Max number of simultaneous connections to the same node	connMaxInternodeNum
HTTP port	TCP port for built-in web server	httpPort
Admon port UDP	UDP port for admon service	connAdmonPortUDP
Net filter file	Path to a file with a list of allowed IP subnets	connNetFilterFile

...continues

...continued

Parameter	Description	Option to beegfs-admon
Tcp only filter file	Path to a file with a list of no RDMA IP ranges	connTcpOnlyFilterFile
Log level	Log level	logLevel
No log date	Do not show date along with time in log	logNoDate
Number of log lines	Number of lines in log file, after which it will be rotated	logNumLines
Number of log rotated files	Number of old log files to keep	logNumRotatedFiles
Log file	Path to the log file, empty means logs go to the journal	logStdFile
Enable mail	Enable email notification	mailEnabled
Smtplib send type	How to send the email to an SMTP server	mailSmtplibSendType
Sendmail path	Path to the sendmail binary	mailSendmailPath
Check interval time	Interval for checking for new events that must be mailed	mailCheckIntervalTimeSec
Minimal down time	Interval of node downtime after which it should be reported	mailMinDownTimeSec
Mail recipients	Email addresses to send notifications to	mailRecipient
Resend mail time	Time after which to send email again as a reminder	mailResendMailTimeMin
Mail sender	Email notification from address	mailSender
SMTP server	SMTP server to use for sending emails	mailSmtplibServer

...continues

...continued

Parameter	Description	Option to beegfs-admon
Number of workers	Number of worker threads	tuneNumWorkers

Table 10.4: BeeGFSAdmon role parameters and beegfs-admon options

The beegfs-admon settings are updated by Bright Cluster Manager in `/etc/beegfs/beegfs-admon.conf`.

The settings can be managed in `cmsh` via the `beegfs::admon` role:

Example

```
[bright90 ~]# cmsh
[bright90]% category roles default
[bright90->category[default]->roles]% assign bee<TAB><TAB><TAB>
beegfs::admon      beegfs::client      beegfs::management  beegfs::metadata    beegfs::storage
[bright90->category[default]->roles]% assign beegfs::admon
[bright90->category*[default*]->roles*[BeeGFS::Admon*]]% show
Parameter          Value
-----
Add services       yes
Name               BeeGFS::Admon
Provisioning associations <0 internally used>
Revision
Type               BeeGFSAdmonRole
Database file      /var/lib/beegfs/beegfs-admon.db
Management node
Clear database     no
Query interval    5s
Admon port UDP    8007
Authentication file
Fallback expiration 15m
HTTP port         8000
Interfaces file
Max internode number 3
Net filter file
Tcp only filter file
Log file
Log level         3
No log date       no
Number of log lines 50000
Number of log rotated files 5
Check interval time 30s
Enable mail       no
Mail recipients
Mail sender
Minimal down time 10s
Resend mail time  1h
SMTP server
Sendmail path     sendmail
Smtplib send type socket
Number of workers 4
```

BeeGFSClient Role

The BeeGFSClient role is used for configuring BeeGFS client and helper services.

The role parameters are described in table 10.5 and in table 10.6:

Parameter	Description	Option to beegfs-client
Management node	Node with a management role	sysMgmtdHost
Authentication file	Path to the shared secret authentication file	connAuthFile
Client port UDP	UDP port for client service	connClientPortUDP
Helper port TCP	TCP port for helper service	connHelperdPortTCP
Communication retry time	Time for retries in case of a network failure	connCommRetrySecs
Fallback expiration time	Time after which a connection to a fallback interface expires	connFallbackExpirationSecs
Interfaces file	Path to file with list of communication interfaces	connInterfacesFile
Max internode number	Maximum simultaneous connections to the same node	connMaxInternodeNum
Net filter file	Path to a file with a list of allowed IP subnets	connNetFilterFile
Use RDMA	Use RDMA	connUseRDMA
RDMA buffers number	Number of RDMA buffers	connRDMABufNum
RDMA buffer size	Maximum size of a buffer that will be sent over the network	connRDMABufSize
RDMA type of service	RDMA type of service	connRDMATypeOfService
Tcp only filter file	Path to a file with a list of no RDMA IP ranges	connTcpOnlyFilterFile

...continues

...continued

Parameter	Description	Option to beegfs-client
Log level	Log level	logLevel
Enable Quota	Enable quota	quotaEnabled
Create hardlinks as symlinks	Create a symlink when an application tries to create a hardlink	sysCreateHardlinksAsSymlinks
Mount sanity check ms	Time in ms that server has to respond after mount sanity check	sysMountSanityCheckMS
Session check on close	Check for valid sessions on storage server when a file is closed	sysSessionCheckOnClose
Sync on close	Sync file content on close	sysSyncOnClose
Target offline timeout	Timeout until all storage targets are considered offline when no target state updates can be fetched from management server	sysTargetOfflineTimeoutSecs
Update target states time	Interval for storage targets states check	sysUpdateTargetStatesSecs
Enable xattrs	Enable xattrs	sysXAttrsEnabled
Enable ACLs	Enable ACLs	sysACLsEnabled
File cache type	File read/write cache type	tuneFileCacheType
Preferred meta file	Path to a file with preferred metadata servers	tunePreferredMetaFile
Preferred storage file	Path to a file with preferred storage targets	tunePreferredStorageFile
Remote fsync	Should fsync be executed on server to flush cached file	tuneRemoteFSync

...continues

...continued

Parameter	Description	Option to beegfs-client
Use global append locks	Should files, opened in append mode, be protected by locks on local machine (false) or on servers (true)	tuneUseGlobalAppendLocks
Use global file locks	Should advisory locks be checked on local machine (false) or on servers (true)	tuneUseGlobalFileLocks

Table 10.5: BeeGFSClient role parameters and beegfs-client options

The beegfs-client settings are updated by Bright Cluster Manager in /etc/beegfs/beegfs-client.conf.

Parameter	Description	Option to beegfs-client
Authentication file	Path to the shared secret authentication file	connAuthFile
Helper port TCP	TCP port for helper service	connHelperdPortTCP
No log date	Do not show date along with time in log	logNoDate
Number of log lines	Number of lines in log file, after which it is rotated	logNumLines
Number of log rotated files	Number of old log files to keep	logNumRotatedFiles
Log file	Path to the log file, empty means logs go to the journal	logStdFile
Helper workers number	Number of worker threads for helper service	tuneNumWorkers

Table 10.6: BeeGFSClient role parameters and beegfs-helperd options

The beegfs-helperd settings are updated by Bright Cluster Manager within /etc/beegfs/beegfs-helperd.conf.

The settings can be managed in cmsh via the beegfs::client role:

Example

```
[bright90 ~]# cmsh
[bright90]% category roles default
[bright90->category[default]->roles]% assign bee<TAB><TAB><TAB>
beegfs::admon      beegfs::client      beegfs::management  beegfs::metadata    beegfs::storage
```

```
[bright90->category[default]->roles]% assign beegfs::client
[bright90->category*[default*]->roles*[BeeGFS::Client*]]% show
```

Parameter	Value
Add services	yes
Name	BeeGFS::Client
Provisioning associations	<0 internally used>
Revision	
Type	BeeGFSClientRole
Management node	
Authentication file	
Client port UDP	8004
Communication retry time	10m
Fallback expiration time	15m
Helper port TCP	8006
Interfaces file	
Max internode number	12
Mountpoint	/mnt/beegfs
Net filter file	
RDMA buffer size	8192
RDMA buffers number	70
RDMA type of service	0
Tcp only filter file	
Use RDMA	yes
Log file	
Log level	3
No log date	no
Number of log lines	50000
Number of log rotated files	5
Enable Quota	no
Create hardlinks as symlinks	no
Enable ACLs	no
Enable xattrs	no
Mount sanity check ms	11s
Session check on close	no
Sync on close	no
Target offline timeout	15m
Update target states time	1m
File cache type	buffered
Helper workers number	2
Preferred meta file	
Preferred storage file	
Remote fsync	yes
Use global append locks	no
Use global file locks	no

10.3.3 Usage

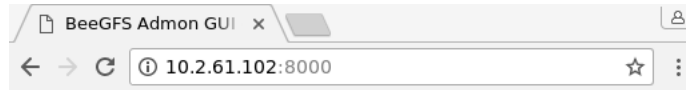
Accessing The Filesystem

After the installation process finishes, the BeeGFS filesystem can be accessed on the client nodes or on the head node. The default mount point is `/mnt/beegfs`, which can be changed within `/etc/beegfs/beegfs-mounts.conf`.

10.3.4 Admon Interface

After Admon is installed, then the Admon GUI jar file can be downloaded from the Admon node `<admon-node>`, via the Admon port `<admon-http-port>`, using a URL of the form (figure 10.17):

`http://<admon-node>:<admon-http-port>`



[Download BeeGFS Admon GUI \(Java .jar\)](#)

Figure 10.17: BeeGFS Admon client download

When the administrator runs the client, configuration options are requested (figure 10.18). The host-name of the Admon node should be specified:

Figure 10.18: BeeGFS Admon client configuration

Authentication is then requested by Admon (figure 10.19):

Figure 10.19: BeeGFS Admon client login

There are two users by default:

- **Information**, with the default password information. This user can only view BeeGFS statistics.
- **Administrator**, with the default password admin. This user can carry out BeeGFS administration.

It is highly recommended to change the default passwords after the first login.

After a successful login, the Admon interface becomes available for use (figure 10.20):

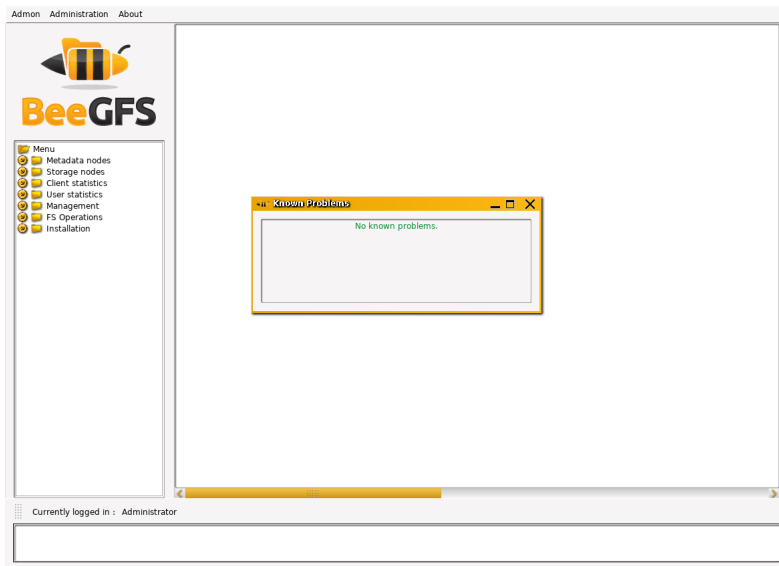


Figure 10.20: BeeGFS Admon client login

More information about Admon is at <http://www.beegfs.com/wiki/AdmonLogin>.

11

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 RHEL distributions require the purchase of their license in order to access their repositories. The other distributions do not.

Packages managed by 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*). Available packages for a particular Bright Cluster Manager version and distribution can be viewed via the *package dashboard* at <https://support.brightcomputing.com/packages-dashboard/>.

There may also be software that the administrator would like to install that is outside the default packages collection. These could be source files that need compilation, or packages in other repositories.

A software image (section 2.1.2) is a filesystem that a node picks up from a provisioner (a head node or a provisioning node) during provisioning so that the node can run as a linux system after provisioning. A subtopic of software management on a cluster is software image management—the management of software on a software image. By default, a node uses the same distribution as the head node for its base image along with necessary minimal, cluster-mandated changes. A node may however deviate from the default, and be customized by having software added to it in several ways.

This chapter covers the techniques of software management for the cluster.

Section 11.1 describes the naming convention for a Bright Cluster Manager RPM or .deb package.

Section 11.2 describes how an RPM or .deb package is managed for the head node.

Section 11.3 describes how an RPM or .deb kernel package can be managed on a head node or image.

Section 11.4 describes how an RPM or .deb package can be managed on a software image.

Section 11.5 describes how a software other than an RPM or .deb package can be managed on a software image.

Section 11.6 describes how custom software images are created that are completely independent of the existing software image distribution and version.

11.1 Bright Cluster Manager Packages And Their Naming Convention

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

```
cmdaemon-9.0-137541_cm9.0_075522ab7d.x86_64.rpm
cmdaemon_9.0-137541-cm9.0-f315823e4a_amd64.deb
```

The file name has the following structure:

```
package-version-revision_cm.x.y_hash.architecture.rpm
```

and

```
package_version-revision-cm.x.y-hash_architecture.deb
```

where:

- *package* (cmdaemon) is the name of the package
- *version* (9.0) is the version number of the package
- *revision* (137541) is the revision number of the package
- *cm* is used to indicate it is a package built by Bright Computing for the cluster manager
- *x.y* (9.0) is the version of Bright Cluster Manager for which the RPM was built
- *hash* (075522ab7d or f315823e4a) is a hash, and is only present for Bright Cluster Manager packages. It is used for reference by the developers of Bright Cluster Manager.
- *architecture* (x86_64 for RPMs or amd64 for APT) is the architecture for which the package was built. The architecture name of x86_64 or amd64 refers the same 64-bit x86 physical hardware in either case.

The differences in .rpm versus .deb package names are just some underbar/hyphen (_/-) changes, the hash (only for Bright Cluster Manager packages), and the architecture naming convention.

Among the distributions supported by Bright Cluster Manager, only Ubuntu uses .deb packages. The rest of the distributions use .rpm packages.

Querying The Packages

To check whether Bright Computing or the distribution has provided a file that is already installed on the system, the package it has come from can be found.

For RPM-based systems: `rpm -qf` can be used with the full path of the file:

Example

```
[root@bright90 ~]# rpm -qf /usr/bin/zless
gzip-1.5.-8.e17.x86_64
[root@bright90 ~]# rpm -qf /cm/local/apps/cmd/sbin/cmd
cmdaemon-9.0-127642_cm9.0d2f835c895.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.

For APT-based systems: A similar check can be done using `dpkg -S` to find the .deb package that provided the file, and then `dpkg -s` on the package name to reveal further information:

Example

```
[root@bright90:~# dpkg -S /cm/local/apps/cmd/etc/cmd.env
cmdaemon: /cm/local/apps/cmd/etc/cmd.env
[root@bright90:~# dpkg -s cmdaemon
Package: cmdaemon
Status: install ok installed
Priority: optional
Section: devel
Installed-Size: 123130
Maintainer: Cluster Manager Development <dev@brightcomputing.com>
Architecture: amd64
Version: 8.0-127523-cm8.0-8eabf447b8
Provides: cmdaemon
...
```

As an aside, system administrators should be aware that the Bright Computing version of a package is provided and used instead of a distribution-provided version for various technical reasons. The most important one is that it is tested and supported by Bright Cluster Manager. Replacing the Bright Computing version with a distribution-provided version can result in subtle and hard-to-trace problems in the cluster, and Bright Computing cannot offer support for a cluster that is in such a state, although some guidance may be given in special cases.

More information about the RPM Package Manager is available at <http://www.rpm.org>, while APT is documented for Ubuntu at <http://manpages.ubuntu.com/manpages/>.

11.1.1 The packages Command

Bright Cluster Manager also provides the `packages` command in the device mode of `cmsh`. This should not be confused with the `packages` command used by `zypper`. The `packages` command used by `cmsh` displays an overview of the installed packages, independent of `rpm` or `deb` package management.

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

Example

```
[bright90]% device use node001
[bright90->device[node001]]% packages -a
Name   Version           Arch  Size  Install date
-----
2to3   3.6.7-1~18.04     all   42B   2018/12/11 10:32:03
acl    2.2.52-3build1    amd64 200B  2018/04/26 19:08:39
acpid  1:2.0.28-1ubuntu1 amd64 139B  2018/04/26 19:08:39
...
```

The `-c|--category` option can be used to list all the packages installed in a node category:

Example

```
[bright90]% device
[bright90->device]% packages -a -c default
Node   Name   Version           Arch  Size  Install date
-----
node001 2to3   3.6.7-1~18.04     all   42B   2018/12/11 10:32:03
node001 acl    2.2.52-3build1    amd64 200B  2018/04/26 19:08:39
...
node002 2to3   3.6.7-1~18.04     all   42B   2018/12/11 10:32:03
node002 acl    2.2.52-3build1    amd64 200B  2018/04/26 19:08:39
...
```

Running the `-a` option for many nodes can be user-unfriendly. That is because per node this command typically returns about 100KB of data. So, for a 1000 nodes this would output about 100MB and a table with nearly a million lines.

When checking packages for many nodes, it is best to request the package by name. Multiple `-f` or `--find` options can be used in the command line to display several packages.

Example

```
[bright90]% device
[bright90->device]% packages -c default -f 2to3
Node      Name    Version          Arch   Size   Install date
-----
node001   2to3   3.6.7-1~18.04   all    42B   2018/12/11 10:32:03
node002   2to3   3.6.7-1~18.04   all    42B   2018/12/11 10:32:03
node003   2to3   3.6.7-1~18.04   all    42B   2018/12/11 10:32:03
node004   2to3   3.6.7-1~18.04   all    42B   2018/12/11 10:32:03
...
```

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

11.2 Managing Packages On The Head Node

11.2.1 Managing RPM Or .deb Packages On The Head Node

Once Bright Cluster Manager has been installed, distribution packages and Bright Cluster Manager software packages are conveniently managed using the `yum`, `zypper` or `apt` repository and package managers. The `zypper` tool is recommended for use with the SUSE distribution, the `apt` utility is recommended for use with Ubuntu, and `yum` is recommended for use with the other distributions that Bright Cluster Manager supports. YUM is not set up by default in SUSE, and it is better not to install and use it with SUSE unless the administrator is familiar with configuring YUM.

Listing Packages On The Head Node With YUM and Zypper

For YUM and `zypper`, the following commands list all available packages:

```
yum list
or
zypper refresh; zypper packages
```

For `zypper`, the short command option `pa` can also be used instead of `packages`.

Listing Packages On The Head Node With APT

For Ubuntu, the `apt-cache` command is used to view available packages. To generate the cache used by the command, the command:

```
apt-cache gencaches
```

can be run.

A verbose list of available packages can then be seen by running:

```
apt-cache dumpavail
```

It is usually more useful to use the `search` option to `apt-cache` to search for the package with a regex:

```
apt-cache search <regex>
```

A similar, but slightly more verbose option is the search option for apt:

```
apt search <regex>
```

Updating/Installing Packages On The Head Node

To install a new package called *<package name>* into a distribution, the corresponding package managers are used as follows:

```
yum install <package name>
zypper in <package name>      #for SLES
apt install <package name>    #for Ubuntu
```

Installed packages can be updated to the latest by the corresponding package manager as follows:

```
yum update
zypper refresh; zypper up      #refresh recommended to update package metadata
apt update; apt upgrade       #update recommended to update package metadata
```

An aside on the differences between the update, refresh/up, and update/upgrade options of the package managers: The update option in YUM by default installs any new packages. On the other hand, the refresh option in zypper, and the update option in APT only update the meta-data (the repository indices). Only if the meta-data is up-to-date will an update via zypper, or an upgrade via apt install any newly-known packages. For convenience, in the Bright Cluster Manager manuals, the term update is used in the YUM sense in general—that is, to mean including the installation of new packages—unless otherwise stated.

Bright Computing maintains YUM and zypper repositories of its packages at:

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

and updates are fetched by YUM and zypper for Bright Cluster Manager packages from there by default, to overwrite older package versions by default.

For Ubuntu, the Bright Computing .deb package repositories are at:

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

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

Cleaning Package Caches On The Head Node

The repository managers use caches to speed up their operations. Occasionally these caches may need flushing to clean up the index files associated with the repository. This can be done by the appropriate package manager with:

```
yum clean all
zypper clean -a  #for SUSE
apt-get clean    #for Ubuntu
```

Signed Package Verification

As an extra protection to prevent 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 repository packages database. Before answering with a “Y”, yum users may choose to compare the contents of `/etc/pki/rpm-gpg/RPM-GPG-KEY-cm` with the key listed in Appendix B to verify its integrity. Alternatively, the key may be imported into the local RPM database directly, using the following command:

```
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-cm
```

With APT the Bright Cluster Manager keyring is already imported into `/etc/apt/trusted.gpg.d/brightcomputing-archive-cm.gpg` if the `cm-config-apt` package, provided by the Bright Computing repository, has been installed. The `cm-config-apt` package is installed by default for the Ubuntu edition of Bright Cluster Manager.

Third Party Packages

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

Most of the third party packages in the following list are repackaged by Bright Computing for 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)
 - FLEXlm License Daemon (section 7.3.5)
- CUDA (section 7.4)

Exclusion of packages on the head node can be carried out as explained in section 11.3.2, 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 thus 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 in YUM or APT. For example in RHEL the installation can be carried out with:

```
yum groupinstall "GNOME Desktop"
```

The Network Manager package is a dependency of the Gnome and KDE desktops, and is therefore automatically installed by the package manager 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 RHEL 7 and derivatives, and Ubuntu:

Example

```
[root@bright90 ~]# systemctl status NetworkManager.service
...
[root@bright90 ~]# 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.

11.2.2 Installation Of Packages On The Head Node That Are Not .deb And Not .rpm Packages

Sometimes a package is not packaged as an RPM or .deb package by 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
```

11.3 Kernel Management On A Head Node Or Image

Care should be taken when updating a head node or a software image. This is particularly true when custom kernel modules compiled against a particular kernel version are being used.

A package can be managed in a software image and the image deployed to nodes. A careful administrator typically clones a copy of a working image that is known to work, before modifying the image.

11.3.1 Installing A Standard Distribution Kernel Into An Image Or On A Head Node

A standard distribution kernel is treated almost like any other package in a distribution.

This means that:

- For head nodes, installing a standard kernel is done according to the normal procedures of managing a package on a head node (section 11.2).
- For regular nodes, installing a standard distribution kernel is done according to the normal procedures of managing an package inside an image, via a changed root (chroot) directory (section 11.4), but with some special aspects that are discussed in this section.

When a kernel is updated or reinstalled (section 11.3.3), kernel-specific drivers, such as OFED drivers may need to be updated or reinstalled. OFED driver installation details are given in section 7.6 of the *Installation Manual*.

Kernel Package Name Formats

For RHEL, individual kernel package names take a form such as:

```
kernel-3.10.0-327.3.1.el7.x86_64.rpm
```

The actual one suited to a cluster varies according to the distribution used. RPM Packages with names that begin with “kernel-devel-” are development packages that can be used to compile custom kernels, and are not required when installing standard distribution kernels.

For Ubuntu, individual Linux kernel image package names take a form such as:

```
linux-image-*.deb
```

or

```
linux-signed-image-*.deb
```

Running `apt-cache search linux | grep 'kernel image'` shows the various packaged kernel images in the distribution.

Other Extra Considerations

When installing a kernel, besides the chroot steps of section 11.4, extra considerations for kernel packages are:

- The kernel must also be explicitly set in CMDaemon (section 11.3.3) before it may be used by the regular nodes.
- If using the chroot method to install the kernel rather than the `cm-chroot-sw-img` method, some other warnings to do with missing `/proc` paths may appear. For RHEL and derivatives, these warnings can be ignored.
- The ramdisk of a regular node must be regenerated using the `createramdisk` command (section 11.4.3).
- If the cluster is in a high availability configuration, then installing a new kernel on to the active head node may in some edge cases stop its network interface, and trigger a failover. It is therefore usually wiser to make the change on the passive head node first, or to disable automatic failover, before carrying out a change that could initiate a failover.

As is standard for Linux, both head or regular nodes must be rebooted to use the new kernel.

11.3.2 Excluding Kernels And Other Packages From Updates

Specifying A Kernel Or Other Package For Update Exclusion

Sometimes it may be desirable to exclude the kernel from updates on the head node.

- When using yum, to prevent an automatic update of a package, the package is listed after using the `--exclude` flag. So, to exclude the kernel from the list of packages that should be updated, the following command can be used:

```
yum --exclude kernel update
```

To exclude a package such as `kernel` permanently from all YUM updates, without having to specify it on the command line each time, the package can instead be excluded inside the repository configuration file. YUM repository configuration files are located in the `/etc/yum.repos.d` directory, and the packages to be excluded are specified with a space-separated format like this:

```
exclude = <package 1> <package 2> ...
```

- The `zypper` command can also carry out the task of excluding the kernel package from getting updated when updating. To do this, the kernel package is first locked (prevented from change) using the `addlock` command, and the update command is run. Optionally, the kernel package is unlocked again using the `removelock` command:

```
zypper addlock kernel
zypper update
zypper removelock kernel          #optional
```

- One APT way to upgrade the software while excluding the kernel image package is to first update the system, then to mark the kernel as a package that is to be held, and then to upgrade the system. Optionally, after the upgrade, the `hold` mark can be removed:

```
apt update
apt-mark hold <linux-image-version>
apt upgrade
apt-mark unhold <linux-image-version>          #optional
```

The complementary way to carry out an upgrade in APT while holding the kernel back, is to use *pinning*. Pinning can be used to set dependency priorities during upgrades. Once set, it can hold a particular package back while the rest of the system upgrades.

Specifying A Repository For Update Exclusion

Sometimes it is useful to exclude an entire repository from an update on the head node. For example, the administrator may wish to exclude updates to the parent distribution, and only want updates for the cluster manager to be pulled in. In that case, in RHEL-derivatives a construction such as the following may be used to specify that only the repository IDs matching the glob `cm*` are used, from the repositories in `/etc/yum.repos.d/`:

```
[root@bright90 ~]# yum repolist
...
122 packages excluded due to repository priority protections
repo id                repo name                status
base/7/x86_64          CentOS-7 - Base          10,067+30
cm-rhel7-9.0/x86_64    CM 9.0 for Red Hat Enterprise Linux 7 10,949+56
epel/x86_64            Extra Packages for Enterprise Linux 7 - x86_64 13,324+92
extras/7/x86_64        CentOS-7 - Extras        301+3
updates/7/x86_64       CentOS-7 - Updates       332
repolist: 34,973
[root@bright90 ~]# yum --disablerepo=* --enablerepo=cm* update
```

In Ubuntu, repositories can be added or removed by editing the repository sources under `/etc/apt/sources.list.d/`. There is also the `apt edit-sources` command, which, unsurprisingly, also edits the repository sources. The `add-apt-repository` command (`man (1) add-apt-repository`) edits the repository sources by line. Running `add-apt-repository -h` shows options and examples.

11.3.3 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 11.4), or specifying a relative root directory setting.

Package dependencies can sometimes prevent the package manager from carrying out the update, for example in the case of OFED packages (section 7.6 of the *Installation Manual*). In such cases, the administrator can specify how the dependency should be resolved.

Parent distributions are by default configured, by the distribution itself, so that only up to 3 kernel images are kept when installing a new kernel with the package manager. However, in a Bright Cluster Manager cluster, this default distribution value is overridden by a default Bright Cluster Manager value, so that kernel images are never removed during YUM updates, or apt upgrade, by default.

For a software image, if the kernel is updated by the package manager, then the kernel is not used on reboot until it is explicitly enabled with either Bright View or cmsh.

- To enable it using Bright View, the Kernel version entry for the software image should be set. This can be accessed via the clickpath Provisioning→Software images→Edit→Settings→Kernel version (figure 11.1).

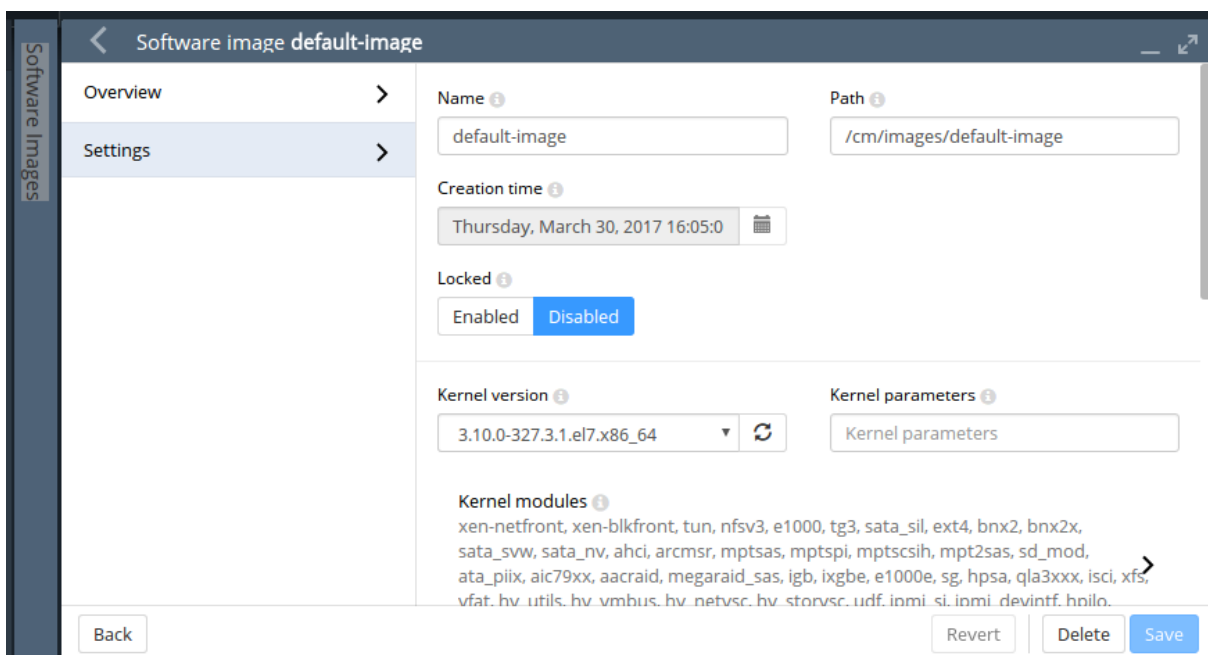


Figure 11.1: Updating A Software Image Kernel With Bright View

- To enable the updated kernel from cmsh, the softwareimage mode is used. The kernelversion property of a specified software image is then set and committed:

Example

```
[root@bright90 ~]# cmsh
[bright90]% softwareimage
[bright90->softwareimage% use default-image
[bright90->softwareimage[default-image]]% set kernelversion 3.10.0-327.3.1.el7.x86_64
[bright90->softwareimage*[default-image*]]% commit -w
```

Tab-completion suggestions for the set kernelversion command will display the available values for the kernel version.

11.3.4 Setting Kernel Options For Software Images

A standard kernel can be booted with special options that alter its functionality. For example, a kernel can boot with `apm=off`, to disable Advanced Power Management, which is sometimes useful as a workaround for nodes with a buggy BIOS that may crash occasionally when it remains enabled.

In Bright View, to enable booting with this kernel option setting, the clickpath Provisioning→Software images→Edit→Settings→Kernel parameters (figure 11.1) is used to set the kernel parameter to `apm=off` for that particular image.

In `cmsh`, the equivalent method is to modify the value of “kernel parameters” in `softwareimage` mode for the selected image:

Example

```
[root@bright90 ~]# cmsh
[bright90]% softwareimage
[bright90]->softwareimage% use default-image
[bright90->softwareimage[default-image]]% append kernelparameters " apm=off"
[bright90->softwareimage*[default-image*]]% commit
```

Often kernel options load up modules and their parameters. Making module loading persist after reboot and setting module loading order is covered in section 5.3.2

Some kernel options may require changes to be made in the BIOS settings in order to function.

11.3.5 Kernel Driver Modules

Bright Computing provides some packages which install new kernel drivers or update kernel drivers. In RPM-based distributions, such packages generally require the `kernel-devel` package. In this section, the `kernel-devel-check` utility is first described, followed by the various drivers that Bright Computing provides.

Kernel Driver Modules: `kernel-devel-check` **Compilation Check**

For RPM, the distribution’s `kernel-devel` package is required to compile kernel drivers for its kernel. It must be the same version and release as the kernel running on the node. For APT, the `linux-header` package corresponding to the kernel image is used.

In RPM-based distributions, to check the head node and software images for the installation status of the `kernel-devel` package, the Bright Cluster Manager utility `kernel-devel-check` is run from the head node:

Example

```
[root@mycluster ~]# kernel-devel-check
Head node: mycluster
No kernel development directories found, probably no kernel development package installed.
package kernel-devel-3.10.0-957.1.3.el7.x86_64 is not installed
Kernel development package kernel-devel-3.10.0-957.1.3.el7.x86_64 not found
If needed, try to install the kernel development package with:
# yum install kernel-devel-3.10.0-957.1.3.el7.x86_64

Software image: default-image
No kernel development directories found, probably no kernel development package installed.
package kernel-devel-3.10.0-957.1.3.el7.x86_64 is not installed
Kernel development package kernel-devel-3.10.0-957.1.3.el7.x86_64 not found
If needed, try to install the kernel development package with:
# chroot /cm/images/default-image yum install kernel-devel-3.10.0-957.1.3.el7.x86_64
```

As suggested by the output of `kernel-devel-check`, running a command on the head node such as:

```
[root@mycluster ~]# chroot /cm/images/default-image1 yum install kernel-devel-3.10.0-957.1.3.el7.x86_64
```

installs a `kernel-devel` package, to the software image called `default-image1` in this case. The package version suggested corresponds to the kernel version set for the image, rather than necessarily the latest one that the distribution provides.

Kernel Driver Modules: Improved Intel Wired Ethernet Drivers

Improved Intel wired Ethernet drivers—what they are: The standard distributions provide Intel wired Ethernet driver modules as part of the kernel they provide. 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 `ixgbev`. 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 11.5.2), this will write the improved driver module into the software image. Restoring to the standard version is then no longer possible with simply removing the improved drivers packages. This makes the image less close-to-standard, and distribution support is then less easily obtained for the node.

Thus, after the installation of the package is done on a head or regular node, for every boot from the next boot onwards, the standard distribution Intel wired Ethernet drivers are replaced by the improved versions for fully-booted kernels. This replacement occurs before the network and network services start. The head node simply boots from its drive with the new drivers, while a regular node initially starts with the kernel using the driver on the software image, but then if the driver differs from the improved one, the driver is unloaded and the improved one is compiled and loaded.

Improved Intel wired Ethernet drivers—installation: The drivers are compiled on the fly on the regular nodes, so a check should first be done that the `kernel-devel` package is installed on the regular nodes (section 11.3.5).

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 7.4 of the *Installation Manual*).

Kernel Driver Modules: OFED And OPA Stack Installation

By default, the distribution provides the OFED stack used by the kernel to manage the InfiniBand or RDMA interconnect. Installing a 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*.

11.4 Managing A Package In A Software Image And Running It On Nodes

A package can be managed in a software image and the image deployed to nodes. A careful administrator typically clones a copy of a working image that is known to work, before modifying the image.

11.4.1 Installing From Head Into The Image: Changing The Root Directory Into Which The Packages Are Deployed

Managing packages (including the kernel) inside a software image is most easily done while on the head node, using a “change root” (`chroot`) mechanism. The easiest way to carry out the `chroot` mechanism in Bright Cluster Manager is to use a wrapper provided by Bright Cluster Manager, `cm-chroot-sw-img`, which works with all distributions.

The same can be carried out more laboriously using the distribution package managers, such as `rpm`, `yum`, `zypper`, or `apt`, by using the associated `chroot` package manager option, or invoking `chroot` as a standalone command.

Change Root As An Option In The Package Manager Command

Using the `rpm` command: The `rpm` command supports the `--root` flag. To install an RPM package inside the default software image while in the head node environment, using the repositories of the head node, the command can be used as follows:

Example

```
rpm --root /cm/images/default-image -ivh /tmp/libxml2-2.6.16-6.x86_64.rpm
```

Using the `yum` command: The `yum` command allows more general updates with a change root option. For example, all packages in the default image can be updated using `yum` for RHEL and derivatives with:

Example

```
yum --installroot=/cm/images/default-image update #for RHEL variants
```

A useful option to restrict the version to which an image is updated, is to use the option `--releasever`. For example, to allow only updates up to RHEL7.4, the command in the preceding example would have `--releasever=7.4` appended to it.

Using the `zypper` command: For SLES, `zypper` can be used as follows to update the image:

Example

```
zypper --root /cm/images/default-image up #for SLES
```

Change Root With `chroot`, Then Running The Package Manager Commands

If the repositories used by the software image are the same as the repositories used by the head node, then the `chroot` command can be used instead of the `--installroot/--root` options to get the same result as the package manager options. That is, the same result is accomplished by first chrooting into an image, and subsequently executing the `rpm`, `yum`, or `zypper` commands without `--root` or `--installroot` arguments. Thus:

For RHEL and derivatives: For YUM-based update, running `yum update` is recommended to update the image, after using the `chroot` command to reach the root of the image:

Example

```
[root@bright90 ~]# chroot /cm/images/default-image
[root@bright90 /]# yum update      #for RHEL variants
...updates happen...
[root@bright90 /]# exit          #get out of chroot
```

For SLES: For SLES, running `zypper up` is recommended to update the image, after using the `chroot` command to reach the root of the image:

Example

```
bright90:~# chroot /cm/images/default-image
bright90.cm.cluster:/ # zypper up      #for SLES
...updates happen...
bright90.cm.cluster:/ # exit          #get out of chroot
```

For Ubuntu: For Ubuntu and APT, for package installation into the software image, there is often a need for the `/proc`, `/sys`, `/dev`, and perhaps other directories to be available within the `chroot` jail. Additionally, the `/proc` namespace used should not be that of the head node due to namespace issues that affect decision-making in some of the pre- and post-installation script bundled with the package.

Pre-configuring all this with `bind` mounting before going into the chrooted filesystem is a little tedious. Therefore a Bright Cluster Manager utility, `cm-chroot-sw-img`, can be used to take care of this.

Thus, for Ubuntu, if the cluster administrator would like to run `apt update`; `apt upgrade` to update the image, then the recommended way to do it is to start the process with the `cm-chroot-sw-img` command:

Example

```
root@bright90:~# cm-chroot-sw-img /cm/images/default-image
...messages indicate that the special directories have been mounted automatically, and a chroot jail has been entered...
root@bright90:/# apt update; apt upgrade #for Ubuntu
...An upgrade session runs in the image root. Some administrator inputs may be needed...
root@bright90:/# exit #get out of chroot
...messages indicate that the special directories have been unmounted automatically...
```

The `cm-chroot-sw-img` wrapper is less needed in other distributions, with `yum` and `zypper` instead of `apt`. This is because the namespace issues are not so serious in with those other distributions. However even in those other distributions, it is cleaner to use the wrapper.

Excluding Packages And Repositories From The Image

Sometimes it may be desirable to exclude a package or a repository from an image.

- If using `yum --installroot`, then to prevent an automatic update of a package, the package is listed after using the `--exclude` flag. For example, to exclude the kernel from the list of packages that should be updated, the following command can be used:

```
yum --installroot=/cm/images/default-image --exclude kernel update
```

To exclude a package such as `kernel` permanently from all YUM updates, without having to specify it on the command line each time, the package can instead be excluded inside the repository configuration file of the image. YUM repository configuration files are located in the `/cm/images/default-image/etc/yum.repos.d` directory, and the packages to be excluded are specified with a space-separated format like this:

```
exclude = <package 1> <package 2> ...
```

- The `zypper` command can also carry out the task of excluding a package from getting updated when during update. To do this, the package is first locked (prevented from change) using the `addlock` command, then the update command is run, and finally the package is unlocked again using the `removelock` command. For example, for the kernel package:

```
zypper --root /cm/images/default-image addlock kernel
zypper --root /cm/images/default-image update
zypper --root /cm/images/default-image removelock kernel
```

- For Ubuntu, the `apt-mark hold` command can be used to exclude a package. This is described in the particular case of excluding the kernel package earlier on, in section 11.3.2.
- Sometimes it is useful to exclude an entire repository from an update to the image. For example, the administrator may wish to exclude updates to the base distribution, and only want 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@bright90 ~]# cd /cm/images/default-image/etc/yum.repos.d/
[root@bright90 yum.repos.d]# yum --installroot=/cm/images/default-image --disablerepo=* --enablerepo=cm* update
```

- For Ubuntu, excluding a repository can be carried out by removing the repository under `/etc/apt/sources.list.d/`. Slightly handier may be to use the `add-apt-repository` command, or the `apt edit-sources` command.

11.4.2 Installing From Head Into The Image: Updating The Node

If the images are in place, then the node that use those images do not run those images until they have the changes placed on the nodes. Rebooting the nodes that use the software images is a straightforward way to have those nodes start up with the new images. Alternatively, the nodes can usually simply be updated without a reboot (section 5.6), if no reboot is required by the underlying Linux distribution.

11.4.3 Installing From Head Into The Image: Possible Issues When Using `rpm --root, yum --installroot` Or `chroot`

- The update process on an image, when using YUM, zypper, or APT, will fail to start if the image is being provisioned by a provisioner at the time. The administrator can either wait for provisioning requests to finish, or can ensure no provisioning happens by locking the image (section 5.4.7), before running the update process. The image can then be updated. The administrator normally unlocks the image after the update, to allow image maintenance by the provisioners again.

Example

```
[root@bright90 ~]# cmsg -c "softwareimage lock default-image"
[root@bright90 ~]# yum --installroot /cm/images/default-image update
[root@bright90 ~]# cmsg -c "softwareimage unlock default-image"
```

- The `rpm --root` or `yum --installroot` command can fail if the versions between the head node and the version in the software image differ significantly. For example, installation from a Scientific Linux 7 head node to a RHEL8 software image is not possible with those commands, and can only be carried out with `chroot`.
- While installing software into a software image with an `rpm --root`, `yum --installroot` or with a `chroot` method is convenient, there can be issues if daemons start up in the image, or if the distribution installation scripts exit with errors due to being in an image environment rather than a real instance.

For example, installation scripts that stop and re-start a system service during a package installation may successfully start that service within the image's `chroot` jail and thereby cause related, unexpected changes in the image. Pre- and post- (un)install scriptlets that are part of RPM or APT packages may cause similar problems.

Bright Computing's RPM and `.deb` packages are designed to install under `chroot` without issues. However packages from other repositories may cause the issues described. To deal with that, the cluster manager runs the `chrootprocess` health check, which alerts the administrator if there is a daemon process running in the image. The `chrootprocess` also checks and kills the process if it is a `crond` process.

- For some package updates, the distribution package management system attempts to modify the ramdisk image. This is true for kernel updates, many kernel module updates, and some other packages. Such a modification is designed to work on a normal machine. For a regular node on a cluster, which uses an extended ramdisk, the attempt does nothing.

In such cases, a new ramdisk image must nonetheless be generated for the regular nodes, or the nodes will fail during the ramdisk loading stage during start-up (section 5.8.4).

The ramdisk image for the regular nodes can be regenerated manually, using the `createramdisk` command (section 5.3.2).

- Trying to work out what is in the image from under `chroot` must be done with some care. For example, under `chroot`, running `"uname -a"` returns the kernel that is currently running—that is the kernel outside the `chroot`. This is typically not the same as the kernel that will load on the node from the filesystem under `chroot`. It is the kernel in the filesystem under `chroot` that an unwary administrator may wrongly expect to detect on running the `uname` command under `chroot`.

To find the kernel version that is to load from the image, the software image kernel version property (section 11.3.3) can be inspected using the cluster manager with:

Example

```
cmsg -c "softwareimage; use default-image; get kernelversion"
```

11.4.4 Managing A Package In The Node-Installer Image

A special software image is the node-installer image. The node-installer image was introduced in Bright Cluster Manager version 9.0, to make multiarch (section 11.7) possible.

The node-installer image is an image that, unsurprisingly, contains the node-installer (section 5.4). The default `/cm/node-installer` tree is a standalone image for that architecture. It requires updating just like the regular software image. So, for example, in YUM, the entire tree can be updated with:

```
chroot /cm/node-installer yum update
```

or

```
yum --installroot=/cm/node-installer update
```

while a particular package inside the image, such as `util-linux`, could be installed with:

```
yum --installroot=/cm/node-installer install util-linux
```

Updating the `node-installer` is recommended whenever there are updates available, in order to fix possible bugs that might affect the `node-installer` operations.

11.5 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 11.5.1)
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 11.5.2)

In both cases, before making changes, a careful administrator typically clones a copy of a working image that is known to work, before modifying the image.

As a somewhat related aside, completely overhauling the software image, including changing the base files that distinguish the distribution and version of the image is also possible. How to manage that kind of extreme change is covered separately in section 11.6.

However, this current section (11.5) 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 11.2.2, appropriate software components go under:

```
/cm/shared/  
|-- apps  
|-- docs  
|-- examples  
|-- licenses  
`-- modulefiles
```

11.5.1 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 11.4:

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.

11.5.2 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 11.5.1, the administrator manages it directly on a node itself, and then makes an updated image from the node after it is configured, to the provisioner.

For example, the administrator may wish to install and test an application from a node first before placing it in the image. Many files may be altered during installation in order to make the node work with the application. Eventually, when the node is in a satisfactory state, and possibly after removing any temporary installation-related files on the node, a new image can be created, or an existing image updated.

Administrators should be aware that until the new image is saved, the node loses its alterations and reverts back to the old image on reboot.

The node-to-image sync can be seen as the converse of the image-to-node sync that is done using `imageupdate` (section 5.6.2).

The node-to-image sync discussed in this section is done using the “Grab to image” or “Synchronize image” menu option from Bright View, or using the “grabimage” command with appropriate options in `cmsh`. The sync automatically excludes network mounts and parallel filesystems such as Lustre and GPFS, but includes any regular disk mounted on the node itself.

Some words of advice and a warning are in order here

- The cleanest, and recommended way, to change an image is to change it directly in the node image, typically via changes within a chroot environment (section 11.5.1).
- Changing the deployed image running on the node can lead to unwanted changes that are not obvious. While many unwanted changes are excluded because of the `excludelistgrab*` lists during a node-to-image sync, there is a chance that some unwanted changes do get captured. These changes can lead to unwanted or even buggy behavior. The changes from the original deployed image should therefore be scrutinized with care before using the new image.
- For scrutiny, the bash command:

```
vimdiff <(cd image1; find . | sort) <(cd image2; find . | sort)
```

run from `/cm/images/` shows the changed files for image directories `image1` and `image2`, with uninteresting parts folded away. The `<(commands)` construction is called *process substitution*, for administrators unfamiliar with this somewhat obscure technique.

Node-To-Image Sync Using Bright View

In Bright View, saving the node state from, for example, `node001` to a new image is done by selecting the appropriate menu option from the clickpath `Devices→Nodes[node001]→Edit→Settings→Actions→Software image→option` (figure 11.2).

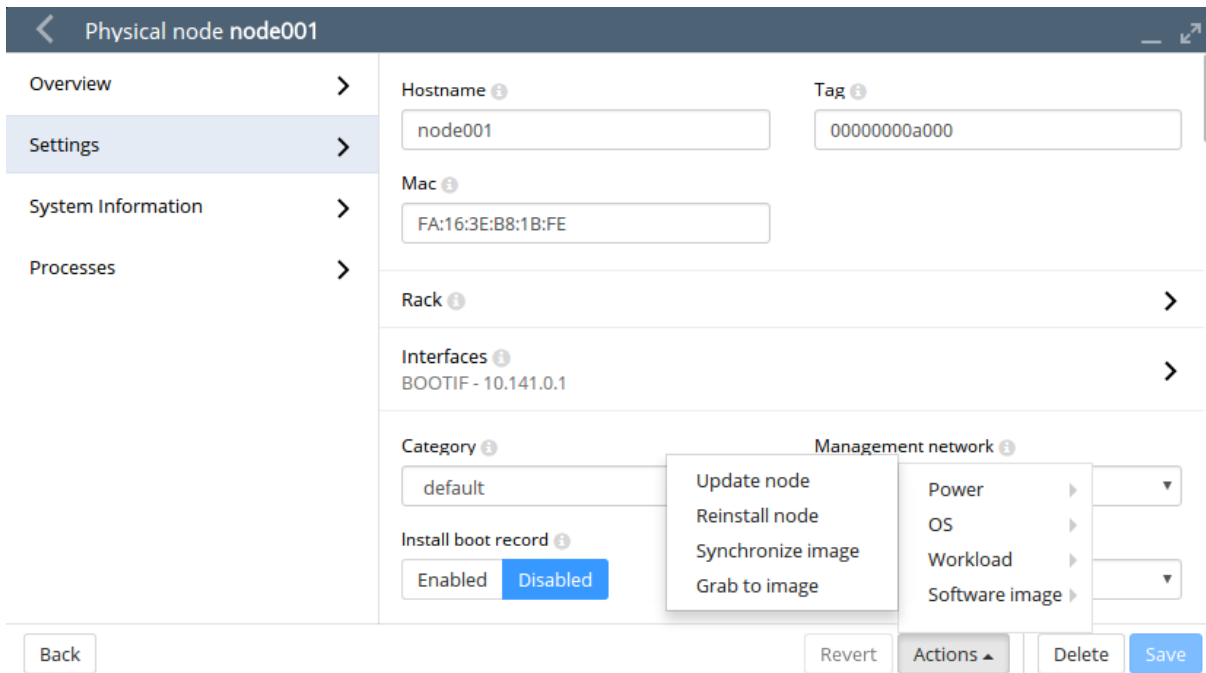


Figure 11.2: Synchronizing From A Node To A Software Image In Bright View

The possible options for saving a node state to an image are:

1. The `Grab to image` option: This opens up a dialog offering an image to sync to. Selecting the image creates a new image, grabbing what is to go to the image from the node. It wipes out whatever (if anything) is in the selected image, except for a list of excluded items. The excluded items are specified in the “Exclude list grabbing to a new image” list, available under Node Categories via the clickpath `Grouping→Node categories[default-image]→Settings→Exclude list grab new`. The exclude list is known as `excludelistgrabnew` (page 460) in `cmsh`.
2. The `Synchronize image` option: This does a sync from the node back to the software image that the node is provisioned with, using evaluation based on file change detection between the node and the image. It is thus a synchronization to the already existing software image that is currently in use by the node.

The items that it excludes from the synchronization are specified in the “Exclude list image grab” list, available under Node Categories via the clickpath `Grouping→Node categories[default-image]→Settings→Exclude list grab`. This exclude list is known as `excludelistgrab` (page 461) in `cmsh`.

The `synchronize` option can be viewed by the administrator as being a more “gentle” way to carry out a node-to-image sync, compared to the `grab` option—that is, a “gentle sync” that tries to avoid wiping out existing files, versus a “violent grab” that can wipe out existing files.

The exclude lists are there to ensure, among other things, that the configuration differences between nodes are left alone for different nodes with the same image. The exclude lists are simple by default, but they conform in structure and patterns syntax in the same way that the exclude lists detailed in section 5.4.7 do, and can therefore be quite powerful.

The images that are available for selection with the `Grab to image` option can be existing images, while the image that the `Synchronize image` option syncs to is the existing image. If such existing images are known to work well with nodes, then overwriting them with a new image on a production system may be reckless. A wise administrator who has prepared a node that is to write an image would therefore follow a process similar to the following instead of simply overwriting an existing image:

1. A new image can be cloned from the old image via the clickpath Provisioning→Software images↓Clone, and setting a name for the new image, for example: newimage. The node state with the software installed on it would then be saved using the Grab to image option, and choosing the image name newimage as the image to save it to.
2. A new category is then cloned from the old category via the clickpath Grouping→Categories↓Clone, and setting a name for the new category, for example newcategory. The old image in newcategory is changed to the new image newimage via the clickpath Grouping→Categories→Edit→Settings→Software image↓newimage.
3. A newly-cloned category has no nodes initially. Some nodes are set to the new category so that their behavior with the new image can be tested. The chosen nodes can be made members of the new category from within the Settings option of each node, and saving the change. The clickpath for this is Devices→Nodes→Edit→Settings→Category↓newcategory
4. The nodes that have been placed in the new category are now made to pick up and run their new images. This can be done with a reboot of those nodes.
5. After sufficient testing, all the remaining nodes can be moved to using the new image. The old image is removed if no longer needed, or perhaps kept around just in case for reference.

Node-To-Image Sync Using `cmsh`

The preceding Bright View method can alternatively be carried out using `cmsh` commands. The `cmsh` equivalent to the Synchronize image option is the `grabimage` command, available from device mode. The `cmsh` equivalent to the Grab to image option is the `grabimage -i` command, where the `-i` option specifies the new image it will write to. As before, that image must be created or cloned beforehand.

Cloning an image for use, setting a category of nodes that will use it, and then synchronizing a node that has the new software setup over to the new image on the provisioner might be carried out as follows via `cmsh`:

```
[root@bright90 ~]# cmsh
[bright90]% softwareimage
[bright90->softwareimage]% clone default-image default-image1
[bright90->softwareimage*[default-image1]]% commit
[bright90->softwareimage[default-image1]]% category
[bright90->category]% clone default default1
[bright90->category*[default1*]]% commit
[bright90->category[default1]]% set softwareimage default-image1
[bright90->category*[default1*]]% commit
[bright90->category[default1]]% device
[bright90->device]% grabimage -w -i default-image1 node001
[bright90->device]%
Mon Jul 18 16:13:00 2011 [notice] bright90: Provisioning started on node node001
[bright90->device]%
Mon Jul 18 16:13:04 2011 [notice] bright90: 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 459) in Bright View.

- 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 459) in Bright View.

11.6 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 11.6.1) is to create a *base (distribution) archive* from an installed base host. The second step (section 11.6.2) is to create the image from the base archive using a special utility, `cm-create-image`.

11.6.1 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 11.6.2.

Examples Of How To Build A Base Distribution Archive

In the following example, a base distribution `tar.gz` archive `/tmp/BASEDIST.tar.gz` is created from the base host `basehost64`. The archive that is created should normally have access control lists and extended attributes preserved too:

Example

```
ssh root@basehost64 \
"tar -cz --acls --xattrs \
--exclude /etc/HOSTNAME --exclude /etc/localtime \
--exclude /proc --exclude /lost+found --exclude /sys \
--exclude /root/.ssh --exclude /var/lib/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

```
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/* \
--exclude=/etc/udev/rules.d/*persistent*.rules \
--exclude=/rhn --exclude=/etc/sysconfig/rhn/systemid \
--exclude=/etc/sysconfig/rhn/systemid.save --exclude=/tmp/* \
--exclude=/var/spool/up2date/* --exclude=/var/log/* \
--exclude=/root/mbox --exclude=/var/cache/yum/* \
--exclude=/etc/cron.daily/rhn-updates \
root@basehost64:/ /cm/images/new-image/
```

The `--acls` and `--xattrs` options are not explicitly required for rsync with SLES12 and beyond. They are also not required for RHEL7 and beyond, and derivatives.

SELinux and file attributes: To use SELinux on compute nodes for RHEL7 and its derivatives, extended attributes must not be used.

The defaults can be modified, if needed, by adjusting attributes for partitions via `cmsh`, in `fspart` mode:

Example

```
[bright90->fspart]% foreach * (set rsyncxattr no)
[bright90->fspart*]% list -f path:0,rsyncxattr
path (key)                rsyncxattr
-----
/cm/images/default-image   no
/cm/images/default-image/boot no
/cm/node-installer         no
/cm/shared                 no
/tftpboot                 no
/var/spool/cmd/monitoring  no
```

Having built the archive by following the examples suggested, the first step in creating the software image is now complete.

11.6.2 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 466).

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 11.2) can be used to exclude updates for the packages that are not to be updated.

Repository access can be directly to the online repositories provided by the distribution, or it can be to a local copy. For RHEL, online repository access can be activated by registering with the Red Hat Network (section 5.1 of the *Installation Manual*). Similarly, for SUSE, online repository access can be activated by registering with Novell (section 5.2 of the *Installation Manual*). An offline repository can be constructed as described in section 11.6.3 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 --excludectmrepo	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
-l --resolvconf	resolv.conf to be used inside image dir, during image creation (by default /etc/resolv.conf from head node is used)
-x --excludectm <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 --excludelist <list>	List of distribution packages to exclude (comma-separated)
-q --excludehwvendor <list>	List of hardware vendor packages to exclude (comma-separated)
-g --enableextrarepo <value>	Argument must be the special string 'public' or path to a directory that has the Bright DVD/ISO mounted.

EXAMPLES:

1. `cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image`
2. `cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image -i /cm/images/test-image`
3. `cm-create-image -d /cm/images/SLES11-image -n sles11-image`
4. `cm-create-image -h node001 -n node001-image`
5. `cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image -i /cm/images/new-image -r`
6. `cm-create-image -a /tmp/RHEL7.tar.gz -n rhel7-image -i /cm/images/new-image -u`
7. `cm-create-image -d /cm/images/new-image -n bio-image -s -e`
8. `cm-create-image -d /cm/images/new-image -n bio-image -s -b /tmp/rhel7-updates.repo`
9. `cm-create-image -d /cm/images/new-image -n bio-image -s -c /tmp/cm-rhel7.repo`
10. `cm-create-image -d /cm/images/new-image -n bio-image -s -m`

Explanations Of The Examples In Usage Of `cm-create-image`

Explanations of the 10 examples in the usage text follow:

1. In the following, a base distribution archive file, `/tmp/RHEL7.tar.gz`, is written out to a software

image named `rhel7-image`:

```
cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image
```

The image with the name `rhel7-image` is created in the CMDaemon database, making it available for use by `cmsh` and Bright View. If an image with the above name already exists, then `/cm/create-image` will exit and advise the administrator to provide an alternate name.

By default, the image name specified sets the directory into which the software image is installed. Thus here the directory is `/cm/images/rhel7-image/`.

2. Instead of the image getting written into the default directory as in the previous item, an alternative directory can be specified with the `--imagedir` option. Thus, in the following, the base distribution archive file, `/tmp/RHEL7.tar.gz` is written out to the `/cm/images/test-image` directory. The software image is given the name `rhel7-image`:

```
cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image --imagedir \
/cm/images/test-image
```

3. If the contents of the base distribution file tree have been transferred to a directory, then no extraction is needed. The `--fromdir` option can then be used with that directory. Thus, in the following, the archive has already been transferred to the directory `/cm/images/SLES11-image`, and it is that directory which is then used to place the image under a directory named `/cm/images/sles11-image/`. Also, the software image is given the name `sles11-image`:

```
cm-create-image --fromdir /cm/images/SLES11-image --imagename sles11-image
```

4. A software image can be created from a running node using the `--fromhost` option. This option makes `cm-create-image` behave in a similar manner to `grabimage` (section 11.5.2) in `cmsh`. It requires passwordless access to the node in order to work. Generic nodes, that is nodes that are not managed by the Bright Cluster Manager, can also be used. An image named `node001-image` can then be created from a running node named `node001` as follows:

```
cm-create-image --fromhost node001 --imagename node001-image
```

By default the image goes under the `/cm/images/node001-image/` directory.

5. If the destination directory already exists, the `--recreate` option can be used to recreate the existing directory. The administrator should be aware that this means removal of the content of any existing directory of the same name. Thus, in the following, the content under `/cm/images/new-image/` is deleted, and new image content is taken from the base distribution archive file, `/tmp/RHEL7.tar.gz` and then placed under `/cm/images/new-image/`. Also, the software image is given the name `rhel7-image`:

```
cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image --imagedir \
/cm/images/new-image --recreate
```

If the `--recreate` option is not used, then the contents are simply overwritten, 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/RHEL7.tar.gz`, is used to update the contents under the directory `/cm/images/new-image/`. The name of the image is also set to `rhel7-image`.

```
cm-create-image --fromarchive /tmp/RHEL7.tar.gz --imagename rhel7-image --imagedir \
/cm/images/new-image --updateimage
```

7. With the default 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 `--basedistrepo` flag is used together with a `.repo` file. The file defines the base distribution repository for the image. The file is copied over into the repository directory of the image, (`/etc/yum.repos.d/` for Red Hat and similar, or `/etc/zypp/repos.d/` for SLES).
9. The `--cmrepo` flag is used together with a `.repo` file. The file defines the cluster manager repository for the image. The file is copied over into the repository directory of the image, (`/etc/yum.repos.d/` for Red Hat and similar, or `/etc/zypp/repos.d/` for SLES).
10. The `--minimal` flag tells `cm-create-image` to install only a limited number of Bright packages and its dependencies, which are required for the cluster management daemon to run the node. This also means that no Bright configuration RPMs will be installed, and all existing system configuration files will remain untouched. The minimal package selection list is read from `/cm/local/apps/cluster-tools/config/minimal`. The word “minimal” primarily refers to 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.
- 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 of RHEL7, there is an extra, supplementary, and an optional packages channel. Packages from these channels can be added to the base-distribution to enhance the capabilities of the image or resolve dependencies of Bright Cluster Manager. Section 7.4.1 of the *Installation Manual* considers an example of such a dependency for the CUDA package.

Unless the required distribution packages and dependencies are installed and configured, particular features of 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 `CENTOS7`, `CENTOS7u5`, `OEL7`, `RHEL7`, `RHEL7u5`, `RHEL8`, `SL7`, `SL7u5`, `SLES12`, `SLES12sp3`, `SLES12sp4`, `SLES15`, `SLES15sp1`, `UBUNTU1604`, `UBUNTU1804`.

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:

- Using Bright View via the clickpath Provisioning→Software images→Software Images→Edit→Settings→Path
- In `cmsh` within `softwareimage` mode, for example:

```
[bright90->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

11.6.3 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 11.6.2, 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 468)
- Setting up a local repository for RHEL (page 469)
- Setting up a local repository for CentOS and Bright from the Bright Cluster Manager ISO for CentOS (page 469)

Configuring Local Repositories For SLES For A Software Image

For SLES11 SP0, SLES11 SP1, and SLES11 SP2, the required packages are spread across two DVDs, and hence two repositories must be created. Assuming the image directory is `/cm/images/sles11sp1-image`, while the names of the DVDs are `SLES-11-SP1-SDK-DVD-x86_64-GM-DVD1.iso` and `SLES-11-SP1-DVD-x86_64-GM-DVD1.iso`, then the contents of the DVDs can be copied as follows:

```
mkdir /mnt1 /mnt2
mkdir /cm/images/sles11sp1-image/root/repo1
mkdir /cm/images/sles11sp1-image/root/repo2
```

```
mount -o loop,ro SLES-11-SP1-SDK-DVD-x86_64-GM-DVD1.iso /mnt1
cp -ar /mnt1/* /cm/images/sles11sp1-image/root/repo1/
mount -o loop,ro SLES-11-SP1-DVD-x86_64-GM-DVD1.iso /mnt2
cp -ar /mnt2/* /cm/images/sles11sp1-image/root/repo2/
```

The two repositories can be added for use by zypper in the image, as follows:

```
chroot /cm/images/sles11sp1-image
zypper addrepo /root/repo1 "SLES11SP1-SDK"
zypper addrepo /root/repo2 "SLES11SP1"
exit (chroot)
```

Configuring Local Repositories For RHEL For A Software Image

For RHEL distributions, the procedure is almost the same. The required packages are contained in one DVD.

```
mkdir /mnt1
mkdir /cm/images/rhel-image/root/repo1
mount -o loop,ro RHEL-DVD1.iso /mnt1
cp -ar /mnt1/* /cm/images/rhel-image/root/repo1/
```

The repository is added to YUM in the image, by creating the repository file `/cm/images/rhel-image/etc/yum.repos.d/rhel-base.repo` with the following contents:

```
[base]
name=Red Hat Enterprise Linux $releasever - $basearch - Base
baseurl=file:///root/repo1/Server
gpgcheck=0
enabled=1
```

Configuring Local Repositories For CentOS And Bright Computing For A Software Image

Mounting the ISOs The variable `$imagedir` is assigned as a shortcut for the software image that is to be configured to use a local repository:

```
imagedir=/cm/images/default-image
```

If the ISO is called `bright-centos.iso`, then its filesystem can be mounted by the root user on a new mount, `/mnt1`, as follows:

```
mkdir /mnt1
mount -o loop bright-centos.iso /mnt1
```

The head node can then access the ISO filesystem.

The same mounted filesystem can also be mounted with the `bind` option into the software image. This can be done inside the software image by the root user, in the same relative position as for the head node, as follows:

```
mkdir $imagedir/mnt1
mount -o bind /mnt1 $imagedir/mnt1
```

This allows an operation run under the `$imagedir` in a `chroot` environment to access the ISO filesystem too.

Creating YUM repository configuration files: YUM repository configuration files can be created:

- **for the head node:** A repository configuration file

```
/etc/yum.repos.d/cm9.0-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/packages/9.0-<subminor>
enabled=1
gpgcheck=1
exclude = slurm* pbspro* uge* cm-hwloc
```

- **for the regular node image:** A repository configuration file

```
$imagedir/etc/yum.repos.d/cm9.0-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@bright90 ~]# yum clean all
[root@bright90 ~]# 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.

11.6.4 Creating A Custom Image From The Local Repository

After having created the local repositories for SLES, RHEL or CentOS (section 11.6.3), 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.

11.7 Creating Images For Other Distributions And Architectures (Multidistro And Multiarch)

Bright Cluster Manager version 9.0 makes it easier to mix distributions in the cluster. This ability is called *multidistro*. However, it is often also loosely called *multiOS*.

Bright Cluster Manager version 9.0 also introduced the ability to run on certain mixed architecture combinations. The ability to run on multiple architectures is called *multiarch*. For Bright Cluster Manager, multiarch means that the node hardware can be based on either the x86-64 CPU architecture, or the ARMv8 CPU architecture, or on a mixture of both.

The Linux distributions and the hardware architectures supported by Bright Cluster Manager are shown in the following table:

		Head			
		R7 (x86-64)	R8 (x86-64, aarch64)	U16, 18 (x86-64, aarch64)	S12, 15
Image	R7	x86-64	x86-64	x86-64	-
	R8	x86-64	x86-64, aarch64	x86-64, aarch64	-
	U16	x86-64	x86-64	x86-64	-
	U18	x86-64	x86-64, aarch64	x86-64, aarch64	-

Legend

R7:	RHEL7
R8:	RHEL8
U16:	Ubuntu 16.04
U18:	Ubuntu 18.04
S12,15:	SLES12, SLES15

For example, a head node running Ubuntu 16.04, on x86-64 or ARMv8 hardware, can support an Ubuntu 16.04, Ubuntu 18.04, RHEL7, RHEL8 distribution running on x86-64 hardware for the compute nodes. In addition, the compute nodes can be running on ARMv8 hardware, in which case only the RHEL8 and Ubuntu 18.04 distributions are supported.

To configure multiarch and multidistro, the `cm-image` tool is used.

11.7.1 The `cm-image` Tool

The `cm-image` tool is essentially a wrapper for the `cm-create-image` (section 11.6) tool. The `cm-image` tool however has some extra features, including allowing the cluster administrator

- to bootstrap the creation of an image from an online or offline repository
- to create a separate node-installer image as well as a separate software image
- to create a directory under `/cm/shared` for each image
- to select the architecture
- to manage packages in an image more easily

When used to enable a distribution for the first time, multiple changes are made to critical files and paths that may put the regular nodes into an unstable state. This means that all regular nodes should be rebooted after its first use.

The command options of `cm-image` are illustrated by the following modes and options tree:

```
cm-image
|----- shell
|         -i|--image <image>
|
|----- create
|         |----- all
|         |         -a|--arch <architecture>
|         |         -d|--distro <distribution>
|         |         -f|--force
|         |         --archive <archive>
```

```

|         |         --dir <directory>
|         |         --host <host>
|         |         --bootstrap
|         |         -b|--baserepo <base repository>
|         |         -c|--cmrepo <cluster manager repository>
|         |
|         |----- node-installer
|         |         -a|--arch <architecture>
|         |         -d|--distro <distribution>
|         |         -f|--force
|         |         --archive <archive>
|         |         --dir <directory>
|         |         --host <host>
|         |         --bootstrap
|         |         -b|--baserepo <base repository>
|         |         -c|--cmrepo <cluster manager repository>
|         |
|         |----- swimage
|         |         -a|--arch <architecture>
|         |         -d|--distro <distribution>
|         |         -f|--force
|         |         --archive <archive>
|         |         --dir <directory>
|         |         --host <host>
|         |         --bootstrap
|         |         -b|--baserepo <base repository>
|         |         -c|--cmrepo <cluster manager repository>
|         |
|         |----- cmshared
|         |         -a|--arch <archive>
|         |         -d|--distro <distribution>
|         |         -f|--force
|         |         -i|--image <image>
|         |
|----- remove
|         -a|--arch <architecture>
|         -d|--distro <distribution>
|         -f|--force
|
|----- package
|         -i|--image <image>
|         --install <package to install>
|         --remove <package to remove>
|         --list
|         --update <package to update>
|         --update-all

```

Values that can be set are:

- *<architecture>*: aarch64, x86_64
- *<distribution>*: centos7, centos8, sles12, sles15, ubuntu1804, ubuntu1604
- *<archive>*: path to a base tar file (section 11.6.1), eg: /root/bastar/data/UBUNTU1804.tar.gz
- *<directory>*: path to a filesystem, eg: /root/bastar/data/untarred/
- *<host>*: URL to a host, eg: http://10.141.255.254/x86-iso/data/packages/dist

- *<base repository>*: repository file for base tar, eg: /root/bright9.0-centos8u0-iso.repo
- *<cluster manager repository>*: repository file for cluster manager, eg: /root/cm-bright9.0-centos8u0-iso.repo
- *<image>*: image to operate on when managing packages. Eg: /cm/images/default-image-rhel8-aarch64 or /cm/node-installer-centos7-x86
- *<package to install>*: eg: cluster-tools
- *<package to remove>*: eg: cluster-tools
- *<package to update>*: eg: cluster-tools

11.7.2 Multidistro Examples: Provisioning From CentOS 7 Head Node To Ubuntu 18.04 Regular Nodes

Using Bootstrap For Cluster With Network Access To Repositories

```
[root@bright90 ~]# module load cm-image
[root@bright90 ~]# cm-image create all -a x86_64 -d ubuntu1804 --bootstrap
Creating software image default-image-ubuntu1804-x86_64
...
Creating cmshared directory /cm/shared-ubuntu1804-x86_64
running command: 'mount --bind /cm/shared-ubuntu1804-x86_64
/cm/images/default-image-ubuntu1804-x86_64/cm/shared '
running command: 'cm-create-image --image-type cmshared -n default-image-ubuntu1804-x86_64 -d
/cm/images/default-image-ubuntu1804-x86_64 -u -s '
...
Installing CM packages..... [ OK ]
running command: 'umount /cm/images/default-image-ubuntu1804-x86_64/cm/shared '
Adding cmshared entry /cm/shared-ubuntu1804-x86_64
Cloning fsexports entry
Completed
Added new category: default-ubuntu1804-x86_64
Use this category for adding nodes
Completed
```

As suggested by the output, a new category, `default-ubuntu1804-x86_64`, appears.
A node can be placed in the new category and restarted:

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% set category default-ubuntu1804-x86_64
[bright90->device*[node001*]]% commit
[bright90->device[node001]]%
Thu Jan 16 14:35:08 2020 [notice] bright90: node001 [ UP ], restart required (category)
[bright90->device[node001]]% reboot
node001: Reboot in progress ...
```

Using ISO For Cluster Without Network Access

A base tar (section 11.6.1) can be used

```
[root@bright90 ~]# module load cm-image
[root@bright90 ~]# cm-image --verbose create all -a x86_64 -d ubuntu1804 --archive \
/root/bastar/data/UBUNTU1804.tar.gz
Creating software image default-image-ubuntu1804-x86_64
...
output follows similar to the preceding case where repository access was possible
```

11.7.3 Multiarch Example: Creating An Image From A Centos 7 Head Node For ARMv8 Architecture Regular Nodes

This section explains how to configure regular node ARMv8 hardware, assuming Bright Cluster Manager is installed on a head node (Chapter 3 of the *Installation Manual*).

Assuming an ARMv8 ISO `bright90-rhel8u0.aarch64.iso` has been picked up, it can be mounted for web access with:

```
[root@bright90 ~]# mkdir /var/www/html/aarch64-iso
[root@bright90 ~]# mount -o loop /root/bright90-rhel8u0.aarch64.iso /var/www/html/aarch64-iso
```

A repository file can be created with the following content:

```
[root@bright90 ~]# cat /root/rhel8-aarch64-cm-iso.repo
[dist-packages-rhel8-aarch64]
name=Dist packages rhel8 aarch64
baseurl=http://10.141.255.254/aarch64-iso/data/packages/dist
enabled=1
gpgcheck=0

[cm-packages-rhel8-aarch64]
name=CM packages rhel8 aarch64
baseurl=http://10.141.255.254/aarch64-iso/data/packages/9.0
enabled=1
gpgcheck=0

[cm-packages-rhel8-aarch64-hpc]
name=CM packages rhel8 aarch64 HPC
baseurl=http://10.141.255.254/aarch64-iso/data/packages/packagegroups/hpc
enabled=1
gpgcheck=0
```

This assumes that the head node has the IP address 10.141.255.254. It should be changed if needed. It also assumes the 9.0 packages are in the 9.0 directory of the ISO. This should also be changed if needed, for example to a more exact subversion number of 9.0 if the ISO uses that for where its packages are.

The images can then be created with:

```
[root@bright90 ~]# module load cm-image
[root@bright90 ~]# cm-image --verbose create all -a aarch64 -d rhel8 --archive \
/var/www/html/aarch64-iso/data/RHEL8u0.tar.gz -b /root/rhel8-aarch64-cm-iso.repo -c \
/root/rhel8-aarch64-cm-iso.repo
```

This takes a while to complete. At the end of the process the following ARMv8 images and entities are created by default:

- The node-installer image: `/cm/node-installer-rhel8-aarch64`
- The `/cm/shared-...` directory: `/cm/shared-rhel8-aarch64`
- The node image: `/cm/images/default-image-rhel8-aarch64`
- The node category: `default-rhel8-aarch64`

The preceding can be verified via `cmsh`:

```
[root@bright90 ~]# cms
[bright90]% category list
Name (key)                Software image                Nodes
-----
default-centos7-x86_64    default-image                  1
default-rhel8-aarch64     default-image-rhel8-aarch64   1
[bright90]% softwareimage list
Name (key)                Path                            ...
-----
default-image              /cm/images/default-image       ...
default-image-rhel8-aarch64 /cm/images/default-image-rhel8-aarch64 ...
[bright90]% partition archos base; list
Arch  OS    Primary image                Shared                    Installer
-----
x86_64  rhel7 default-image                /cm/shared-centos7-x86_64 /cm/node-installer-centos7-x86_64
aarch64 rhel8 default-image-rhel8-aarch64 /cm/shared-rhel8-aarch64 /cm/node-installer-rhel8-aarch64
```

The node settings should be updated. The new category can be assigned to any ARMv8 nodes:

```
[root@bright90 ~]# cms
[bright90]% device use arm-node001
[bright90->device[node001]]% set category default-rhel8-aarch64
[bright90->device*[node001*]]% commit
```

The `bootloaderprotocol` should be set to `tftp` to work with ARMv8 hardware:

Example

```
[root@bright90 ~]# cms
[bright90]% category use default-rhel8-aarch64
[bright90->category[default-rhel8-aarch64]]% set bootloaderprotocol tftp
[bright90->category*[default-rhel8-aarch64*]]% commit
```

The `kernelconsoleoutput` should be changed to `ttyAMA0` to work with the image running on the ARMv8 hardware:

```
[root@bright90 ~]# cms
[bright90]% softwareimage use default-rhel8-aarch64
[bright90->softwareimage[default-rhel8-aarch64]]% set kerneloutputconsole ttyAMA0
[bright90->softwareimage*[default-rhel8-aarch64*]]% commit
```

The settings configured so far are for generic ARMv8 hardware.

Fujitsu ARMv8 Hardware Configuration

Nodes using Fujitsu ARMv8 hardware can have their configuration options modified further.

The BMC settings of the nodes should be updated with extra arguments:

Example

```
[root@bright90 ~]# cms
[bright90]% device use arm-node001
[bright90->device[node001]]% bmcsettings; set extraarguments "-L USER -t 0x30"
[bright90->device*[node001*]]% commit
```

The configuration for fetching environmental metrics should also be updated. The `ipmitool` monitoring resource available for Fujitsu ARMv8 hardware is run via the `a64fx` resource.

The existence of the `a64fx` monitoring resource can be checked for on the ARMv8 node:

```
[root@bright90 ~]# cmsg  
[bright90]% device monitoringresources arm-node001 | grep a64fx  
a64fx
```

The monitoring settings for IPMI via the a64fx object can be enabled as follows:

```
[root@bright90 ~]# cmsg  
[bright90]% monitoring setup; use ipmi  
[bright90->monitoring->setup[ipmi]]% set script /cm/local/apps/cmd/scripts/metrics/sample_ipmitool.py  
[bright90->monitoring->setup*[ipmi*]]% executionmultiplexers  
[bright90->monitoring->setup*[ipmi*]->executionmultiplexers]% remove ipmi  
[bright90->monitoring->setup*[ipmi*]->executionmultiplexers*]% add resource a64fx  
[bright90->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% set resources a64fx  
[bright90->monitoring->setup*[ipmi*]->executionmultiplexers*[a64fx*]]% commit
```

12

Cluster Monitoring

Bright Cluster Manager monitoring allows a cluster administrator to monitor anything that can be monitored in the cluster. Much of the monitoring consists of pre-defined sampling configurations. If there is anything that is not configured, but the data on which it is based can be sampled, then monitoring can be configured for it too, by the administrator.

The monitoring data can be viewed historically, as well as on demand. The historical monitoring data can be stored raw, and optionally also as consolidated data—a way of summarizing data.

The data can be handled raw and processed externally, or it can be visualized within Bright View in the form of customizable charts. Visualization helps the administrator spot trends and abnormal behavior, and is helpful in providing summary reports for managers.

Monitoring can be configured to set off alerts based on triggers, and pre-defined or custom actions can be carried out automatically, depending on triggers. The triggers can be customized according to user-defined conditional expressions.

Carrying out such actions automatically after having set up triggers for them means that the monitoring system can free the administrator from having to carry out these chores.

In this chapter, the monitoring system is explained with the following approach:

1. A basic example is first presented in which processes are run on a node. These processes are monitored, and trigger an action when a threshold is exceeded.
2. With this easy-to-understand example as a basic model, the various features and associated functionality of the Bright Cluster Manager monitoring system are then described and discussed in further depth. These include visualization of data, concepts, configuration, monitoring customization and `cmsh` use.

12.1 A Basic Monitoring Example And Action

12.1.1 Synopsis Of Basic Monitoring Example

In section 12.1, after an overview (section 12.1.1), a minimal basic example of monitoring a process is set up (section 12.1.2) and used (section 12.1.3). The example is contrived, with the aim being to present a basic example that covers a part of what the monitoring system is capable of handling. The basic example gives the reader a structure to keep in mind, around which further details are fitted and filled in during the coverage in the rest of this chapter.

For the basic example, a user runs a large number of pointless CPU-intensive processes on a head node which is normally very lightly loaded. An administrator can monitor user mode CPU load usage throughout the cluster, and notices this usage spike. After getting the user to stop wasting CPU cycles, the administrator may decide that putting a stop to such processes automatically is a good idea. The administrator can set that up with an action that is triggered when a high load is detected. The action that is taken after triggering, is to stop the processes (figure 12.1).

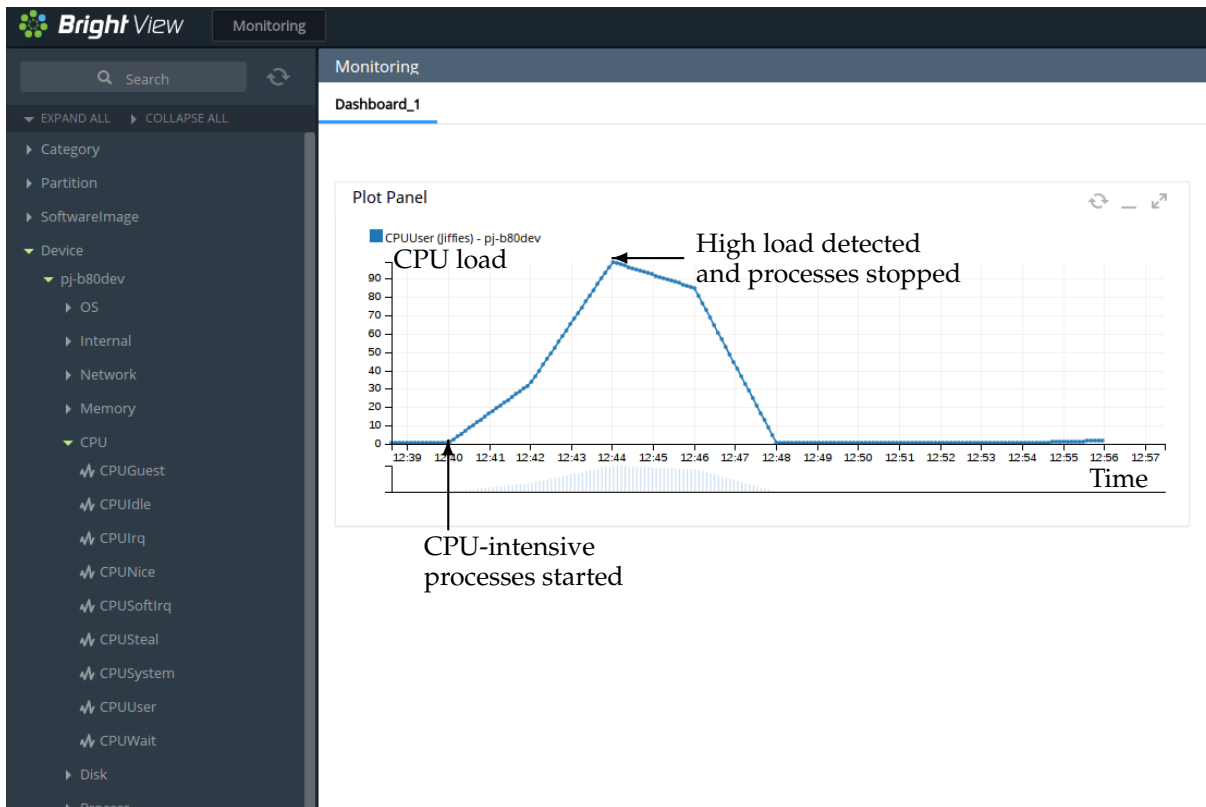


Figure 12.1: Monitoring Basic Example: CPU-intensive Processes Started, Detected And Stopped

The basic example thus illustrates how Bright Cluster Manager monitoring can be used to detect something on the cluster and how an action can be set up and triggered based on that detection.

12.1.2 Before Using The Basic Monitoring Example—Setting Up The Pieces

Running A Large Number Of Pointless CPU-Intensive Processes

One way to simulate a user running pointless CPU-intensive processes is to run several instances of the standard unix utility, `yes`. The `yes` command sends out an endless number of lines of “y” texts. It is typically used in scripts to answer dialog prompts for confirmation.

The administrator can run 8 subshell processes in the background from the command line on the head node, with `yes` output sent to `/dev/null`, as follows:

```
for i in {1..8}; do ( yes > /dev/null & ); done
```

Running “`mpstat 2`” shows usage statistics for each processor, updating every 2 seconds. It shows that `%user`, which is user mode CPU usage percentage, is close to 90% on an 8-core or less head node when the 8 subshell processes are running.

Setting Up The Kill Action

To stop the pointless CPU-intensive `yes` processes, the command `killall yes` can be used. The administrator can make it a part of a script `killalleyes`:

```
#!/bin/bash
killall yes
```

and make the script executable with a `chmod 700 killalleyes`. For convenience, it may be placed in the `/cm/local/apps/cmd/scripts/actions` directory where some other action scripts also reside.

12.1.3 Using The Monitoring Basic Example

Now that the pieces are in place, the administrator can use Bright View to add the `killallyesaction` action to its action list, and then set up a trigger for the action:

Adding The Action To The Actions List

In Bright View:

- The clickpath

Monitoring Configuration→Actions→Monitoring Actions→killprocess→Clone

is used to clone the structure of an existing action. The `killprocess` action is convenient because it is expected to function in a similar way, so its options should not have to be modified much. However, any action could be cloned and the clone modified in appropriate places.

- The name of the cloned action is changed. That is, the administrator sets Name to `killallyesaction`. This is just a sensible label—the name can be arbitrary.
- Script is set to the path `/cm/local/apps/cmd/scripts/actions/killallyes`, which is where the script was placed earlier (page 478).

After saving, the `killallyesaction` action becomes part of the list of monitoring actions (figure 12.2).

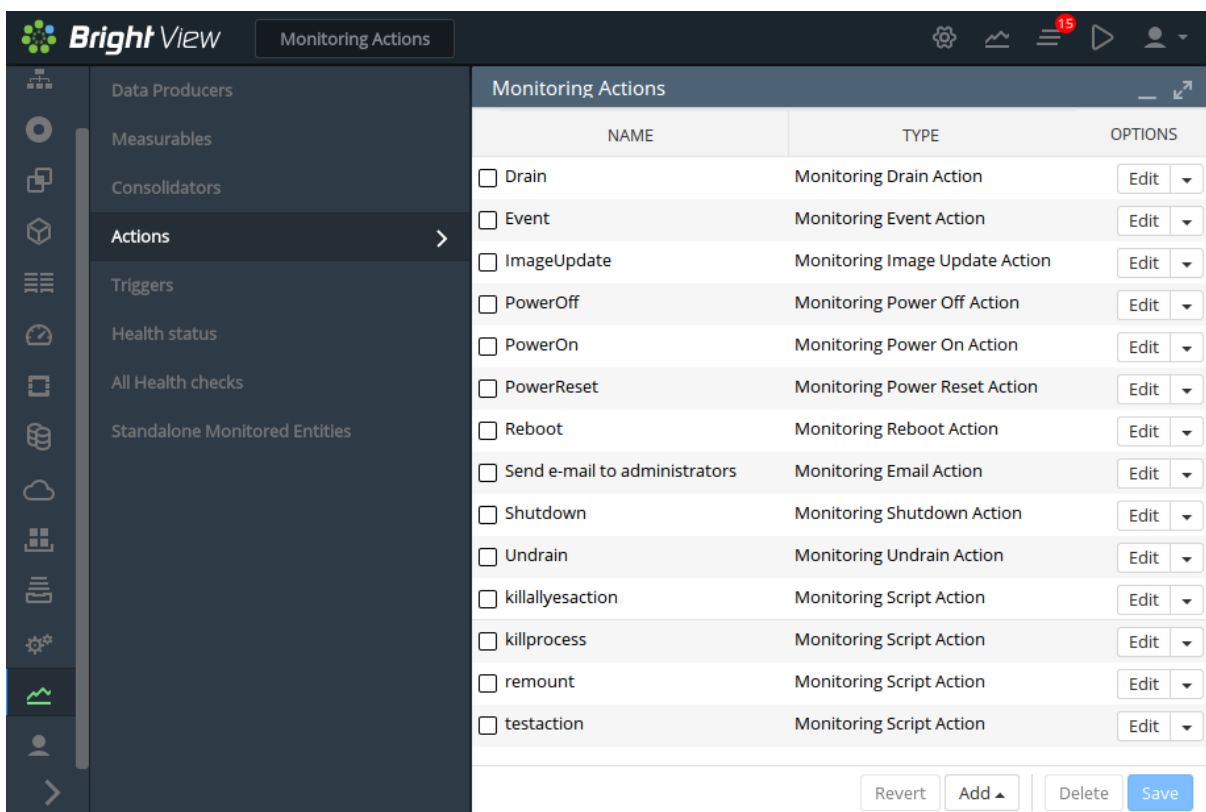


Figure 12.2: Bright View Monitoring Configuration: Adding An Action

Setting Up A Trigger Using CPUUser On The Head Node(s)

The clickpath

Monitoring Configuration→Triggers→Failing health checks→Clone

can be used to configure a monitoring trigger, by cloning an existing trigger. A trigger is a sample state condition that runs an action. In this case, the sample state condition may be that the metric (section 12.2.3) CPUUser must not exceed 50. If it does, then an action (killallyesaction) is run, which should kill the yes processes.

- CPUUser is a measure of the time spent in user mode CPU usage per second, and is measured in jiffy intervals per second.
- A jiffy interval is a somewhat arbitrary time interval that is predefined for kernel developers per platform. It is the minimum amount of time that a process has access to the CPU before the kernel can switch to another task.

Unlike %user from the top command, a jiffy interval is not a percentage.

To configure this, the trigger attributes can be modified as follows (figure 12.3):

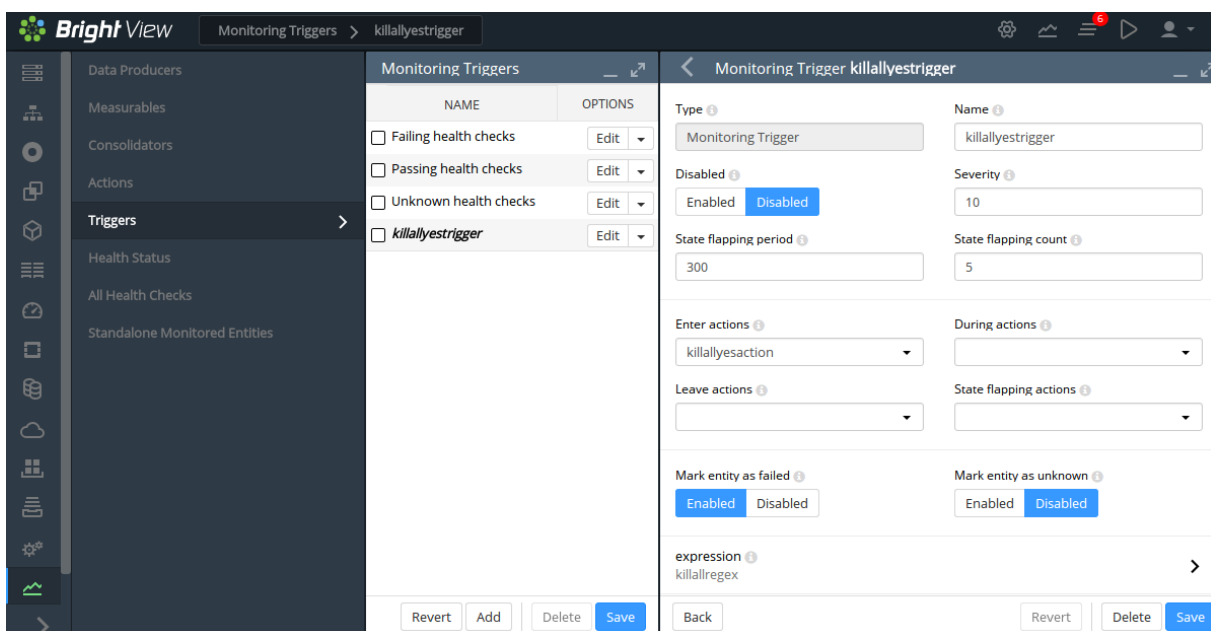


Figure 12.3: Bright View Monitoring Configuration: Setting A Trigger

- A name is set for the trigger. The name can be arbitrary, and killallyestrigger is used in this example.
- the trigger is made active by enabling it
- the trigger can be set to run an action script if the sample state crosses over into a state that meets the trigger condition. That is, Enter actions is configured for a particular condition.

The condition under which the Enter actions action script is run in the example, can simply be when CPUUser on the head node is above 50. Such a condition can be set by setting an expression in the expression subwindow. In this subwindow (figure 12.4):

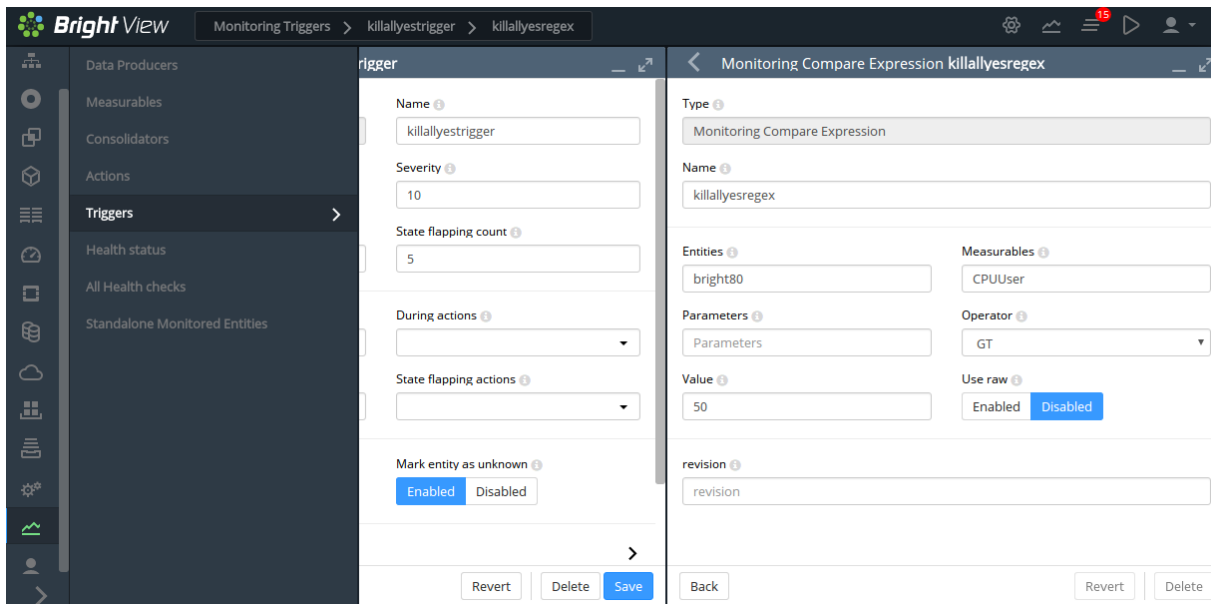


Figure 12.4: Bright View Monitoring Configuration: Setting An Expression

- A name is set for the expression. The name can be arbitrary, and `killallyesregex` is used for Name in this example.
- An entity is set. In this case, the entity being monitored is the head node. If the head node is called `bright90` in this example, then `bright90` is the value set for entities. An entity is often simply a device, but it can be any object that `CMDaemon` stores.
- A measurable is set. In this case, `Measurables` is set to `CPUUser`.
- An operator and threshold value are set. In this case `GT`, which is the greater than operator, and `50` which is a significant amount of `CPUUser` time in jiffies/s, are set for `Operator` and `Value`.

After saving the configuration, the `killallyesregex` regular expression evaluates the data being sampled for the `killallystrigger` trigger. If the expression is `TRUE`, then the trigger launches the `killallyesaction` action.

The Result

In the preceding section, an action was added, and a trigger was set up with a monitoring expression.

With a default installation on a newly installed cluster, the measurement of `CPUUser` is done every 120s (the period can be modified in the Data Producer window of Bright View, as seen in figure 12.9). The basic example configured with the defaults thus monitors if `CPUUser` on the head node has crossed the bound of 50 jiffies/s every 120s.

If `CPUUser` is found to have entered—that is crossed over from below the value and gone into the zone beyond 50 jiffies—then the `killallyesregex` regular expression notices that. Then, the trigger it is configured for, `killallystrigger` trigger, runs the `killallyesaction` action, which runs the `killallyes` script. The `killallyes` script kills all the running `yes` processes. Assuming the system is trivially loaded apart from these `yes` processes, the `CPUUser` metric value then drops to below 50 jiffies.

To clarify what “found to have entered” means in the previous paragraph:

After an `Enter` trigger condition has been met for a sample, the first sample immediately after that does not ever meet the `Enter` trigger condition, because an `Enter` threshold crossing condition requires the previous sample to be below the threshold.

The second sample can only launch an action if the `Enter` trigger condition is met and if the preceding sample is below the threshold.

Other non-yes CPU-intensive processes running on the head node can also trigger the `killalloyes` script. Since the script only kills `yes` processes, leaving any non-yes processes alone, it would in such a case run unnecessarily. This is a deficiency due to the contrived and simple nature of the basic example which is being illustrated here. In a production case the action script is expected to have a more sophisticated design.

At this point, having gone through section 12.1, the reader is expected to have a rough idea of how monitoring, triggers, trigger conditional expressions, and actions work. The following sections in this chapter cover the concepts and features for Bright Cluster Manager monitoring in greater detail.

12.2 Monitoring Concepts And Definitions

A discussion of the concepts of monitoring, along with definitions of terms used, is appropriate at this point. The features of the monitoring system in Bright Cluster Manager covered later on in this chapter will then be understood more clearly.

12.2.1 Measurables

Measurables are measurements (sample values) that are obtained via data producers (section 12.2.10) in `CMDaemon`'s monitoring system. The measurements can be made for nodes, head nodes, other devices, or other *entities*.

Types Of Measurables

Measurables can be:

- *enummetrics*: measurements with a small number of states. The states can be pre-defined, or user-defined. Further details on *enummetrics* are given in section 12.2.2.
- *metrics*: measurements with number values, and no data, as possible values. For example, values such as: `-13113143234.5`, `24`, `9234131299`. Further details on *metrics* are given in section 12.2.3.
- *health checks*: measurements with the states `PASS`, `FAIL`, and `UNKNOWN` as possible states, and no data as another possible state, when none of the other states are set. Further details on *health checks* are given in section 12.2.4.

no data And Measurables

If no measurements are carried out, but a sample value needs to be saved, then the sample value is set to `no data` for a measurable. This is a defined value, not a null data value. *metrics* and *enummetrics* can therefore also take the `no data` value.

Entities And Measurables

An entity is a concept introduced in Bright Cluster Manager version 8.0.

Normally, a device, or a category or some similar grouping is a convenient idea to keep in mind as an entity, for concreteness.

The default entities in a new installation of Bright Cluster Manager are the following:

`device category partition[base] softwareimages`

However, more generally, an entity can be an object from the following top level modes of `cmsh`:

`category ceph cloud cmjob configurationoverlay device edgesite etcd fspart group jobqueue jobs kubernetes network nodegroup openstack partition profile rack softwareimage user`

For example, a software image object that is to be provisioned to a node is an entity, with some of the possible attributes of the entity being the name, kernelversion, creationtime, or locked attributes of the image:

```
[root@bright90 ~]# cmsh -c "softwareimage use default-image; show"
Parameter          Value
-----
```

```

Creation time          Thu, 08 Jun 2017 18:15:13 CEST
Enable SOL            no
Kernel modules        <44 in submode>
Kernel parameters
Kernel version        3.10.0-327.3.1.el7.x86_64
Locked               no
Name                 default-image
...

```

Because measurements can be carried out on such a variety of entities, it means that the monitoring and conditional actions that can be carried out on a Bright Cluster Manager cluster can be very diverse. This makes entities a powerful and versatile concept in Bright Cluster Manager's monitoring system for managing clusters.

Listing Measurables Used By An Entity

In `cmsh`, for an entity, such as a device within device mode, a list of the measurables used by that device can be viewed with the `measurables` command.

Example

```

[bright90->device]% measurables node001
Type          Name          Parameter  Class    Producer
-----
Enum          DeviceStatus          Internal  DeviceState
HealthCheck   ManagedServicesOk     Internal  CMDaemonState
HealthCheck   defaultgateway        Network  defaultgateway
HealthCheck   diskspace              Disk     diskspace
HealthCheck   dmesg                  OS       dmesg
...

```

The subsets of these measurables—enummetrics, metrics, and health checks—can be listed with the `enummetrics` (section 12.2.2), `metrics` (section 12.2.3), or `healthchecks` (section 12.2.4) command.

In Bright View, all the entities that are using health checks can be viewed via the clickpath:

Monitoring→All Health Checks (figure 12.22 section 12.4.7)

Listing Entities That Use A Measurable

The entities using a specific measurable can be listed with the `usage` command:

Example

```

[bright90->monitoring->measurable]% usage nfs_v3_server_total
Measurable      Count  Entities
-----
nfs_v3_server_total  1      bright90

```

If the number of measurables is too large to view on the screen:

Example

```

[bright90->monitoring->measurable]% usage devicestatus
Measurable      Count  Entities
-----
DeviceStatus    21     node001,node002,node003,node004,node005,node006,node007,node008,node009,node010+

```

then the `-v` option can be used to list the entities over multiple lines:

Example

```
bright90->monitoring->measurable]% usage -v devicestatus
Measurable    Count  Entities
-----
DeviceStatus  21     node001,node002,node003,node004,node005,node006,node007,node008,node009,node010,
node011,node012,node013,node014,node015,node016,node017,node018,node019,node020,
bright90
```

Listing Measurables From monitoring Mode

Similarly, under monitoring mode, within the measurable submode, the list of measurable objects that can be used can be viewed with a `list` command:

Example

```
[bright90->monitoring]% measurable list
Type          Name (key)          Parameter  Class          Producer
-----
Enum          DeviceStatus        Internal   Internal       DeviceState
HealthCheck   ManagedServicesOk   Internal   Internal       CMDaemonState
HealthCheck   Mon::Storage        Internal/Monitoring/Storage  MonitoringSystem
HealthCheck   chrootprocess       OS         OS             chrootprocess
HealthCheck   cmsh                 Internal   Internal       cmsh
...
...
```

The subsets of these measurables—enummetrics, metrics, and health checks—can be listed with: `list enum` (section 12.2.2), `list metric` (section 12.2.3), or `list healthcheck` (section 12.2.4).

In Bright View, the equivalent to listing the measurables can be carried out via the clickpath:

Monitoring configuration → Measurables (figure 12.10, section 12.4.2)

and listing the subsets of the measurables is possible using column filtering (figure 12.11, section 12.4.2).

Viewing Parameters For A Particular Measurable From monitoring Mode

Within the measurable submode, parameters for a particular measurable can be viewed with the `show` command for that particular measurable:

Example

```
[bright90->monitoring->measurable]% use devicestatus
[bright90->monitoring->measurable[DeviceStatus]]% show
Parameter          Value
-----
Class              Internal
Consolidator       none
Description        The device status
Disabled           no (DeviceState)
Maximal age        0s (DeviceState)
Maximal samples    4,096 (DeviceState)
Name               DeviceStatus
Parameter
Producer           DeviceState
Revision
Type              Enum
```

12.2.2 Enummetrics

An *enummetric* is a measurable for an entity that can only take a limited set of values. At the time of writing (June 2017), `DeviceStatus` is the only enummetric. This may change in future versions of Bright Cluster Manager.

The full list of possible values for the enummetric `DeviceStatus` is:

up, down, closed, installing, installer_failed, installer_rebooting, installer_callinginit, installer_unreachable, installer_burning, burning, unknown, opening, going_down, pending, and no data.

The enummetrics available for use can be listed from within the measurable submode of the monitoring mode:

Example

```
[bright90->monitoring->measurable]% list enum
Type   Name (key)           Parameter           Class   Producer
-----
Enum   DeviceStatus          Internal           DeviceState
[bright90->monitoring->measurable]%
```

The list of enummetrics that is configured to be used by an entity, such as a device, can be viewed with the `enummetrics` command for that entity:

Example

```
[bright90->device]% enummetrics node001
Type   Name           Parameter           Class   Producer
-----
Enum   DeviceStatus   Internal           DeviceState
[bright90->device]%
```

The states that the entity has been through can be viewed with a `dumpmonitoringdata` command (section 12.6.4):

Example

```
[bright90->device]% dumpmonitoringdata -99d now devicestatus node001
Timestamp           Value           Info
-----
2017/07/03 16:07:00.001  down
2017/07/03 16:09:00.001  installing
2017/07/03 16:09:29.655   no data
2017/07/03 16:11:00     up
2017/07/12 16:05:00     up
```

The parameters of an enummetric such as `devicestatus` can be viewed and set from monitoring mode, from within the measurable submode (page 484).

12.2.3 Metrics

A *metric* for an entity is typically a numeric value for an entity. The value can have units associated with it.

In the basic example of section 12.1, the metric value considered was `CPUUser`, measured at the default regular time intervals of 120s.

The value can also be defined as `no data`. `no data` is substituted for a null value when there is no response for a sample. `no data` is not a null value once it has been set. This means that there are no null values stored for monitored data.

Other examples for metrics are:

- LoadOne (value is a number, for example: 1.23)
- WriteTime (value in ms/s, for example: 5 ms/s)
- MemoryFree (value in readable units, for example: 930 MiB, or 10.0 GiB)

A metric can be a built-in, which means it comes with Bright Cluster Manager as integrated code within CMDaemon. This is based on c++ and is therefore much faster than the alternative. The alternative is that a metric can be a standalone script, which means that it typically can be modified more easily by an administrator with scripting skills.

The word metric is often used to mean the script or object associated with a metric as well as a metric value. The context makes it clear which is meant.

A list of metrics in use can be viewed in cmsh using the `list` command from monitoring mode:

Example

```
[bright90->monitoring]% measurable list metric
Type      Name (key)           Parameter      Class          Producer
-----
Metric    AlertLevel          count          Internal       AlertLevel
Metric    AlertLevel          maximum        Internal       AlertLevel
...
```

In Bright View, the metrics can be viewed with the clickpath:

Monitoring configuration→Measurables (figure 12.10, section 12.4.2)

and then clicking on the filter widget to select Metric (figure 12.11, section 12.4.2).

A list of metrics in use by an entity can be viewed in cmsh using the `metrics` command for that entity. For example, for the entity `node001` in mode `devices`:

Example

```
[bright90->devices]% metrics node001
Type      Name           Parameter      Class          Producer
-----
Metric    AlertLevel     count          Internal       AlertLevel
Metric    AlertLevel     maximum        Internal       AlertLevel
...
```

The parameters of a metric such as `AlertLevel:count` can be viewed and set from monitoring mode, from within the `measurable` submenu, just as for the other measurables:

Example

```
[bright90->monitoring->measurable]% use alertlevel:count
[bright90->monitoring->measurable[AlertLevel:count]]% show
Parameter          Value
-----
Class               Internal
Consolidator        default
Cumulative          no
Description         Number of active triggers
Disabled            no
Maximal age         0s
Maximal samples     0
```


Maximum	0
Minimum	0
Name	AlertLevel
Parameter	count
Producer	AlertLevel
Revision	
Type	Metric

The equivalent Bright View clickpath to edit the parameters is:

Monitoring configuration→Measurables→Edit

12.2.4 Health Check

A *health check* value is a response to a check carried out on an entity. The response indicates the health of the entity for the check that is being carried out.

For example, the `ssh2node` health check, which runs on the head node to check if the SSH port 22 passwordless access to regular nodes is reachable.

A health check is run at a regular time interval, and can have the following possible values:

- **PASS:** The health check succeeded. For example, if `ssh2node` is successful, which suggests that an ssh connection to the node is fine.
- **FAIL:** The health check failed. For example, if `ssh2node` was rejected. This suggests that the ssh connection to the node is failing.
- **UNKNOWN:** The health check did not succeed, did not fail, but had an unknown response. For example, if `ssh2node` has a timeout, for example due to routing or other issues. It means that it is unknown whether the connection is fine or failing, because the response that came in is unknown. Typically the administrator should investigate this further.
- **no data:** The health check did not run, so no data was obtained. For example, if `ssh2node` is disabled for some time, then no data values were obtained during this time. Since the health check is disabled, it means that no data cannot be recorded during this time by `ssh2node`. However, because having a `no data` value in the monitoring data for this situation is a good idea—explicitly knowing about having no data is helpful for various reasons—then `no data` values can be set, by `CMDaemon`, for samples that have no data.

Other examples of health checks are:

- `diskspace`: check if the hard drive still has enough space left on it
- `mounts`: check mounts are accessible
- `mysql`: check status and configuration of MySQL is correct
- `hpraid`: check RAID and health status for certain HP RAID hardware

These and others can be seen in the directory: `/cm/local/apps/cmd/scripts/healthchecks`.

Health Checks

In Bright View, the health checks that can be configured for all entities can be seen with the clickpath:

Monitoring configuration→Measurables (figure 12.10, section 12.4.2)

and then clicking on the filter widget to select `Health Check` (figure 12.11, section 12.4.2).

Options can be set for each health check by clicking through via the `Edit` button.

All Configured Health Checks

In Bright View, health checks that have been configured for all entities can be seen with the clickpath:

Monitoring configuration→All Health Checks (section 12.4.7)

The view can be filtered per column.

Configured Health Checks For An Entity

An overview can be seen for a particular entity *<entity>* via the clickpath:

Monitoring configuration→Health status→*<entity>*→Show

Severity Levels For Health Checks, And Overriding Them

A health check has a settable severity (section 12.2.7) associated with its response defined in the trigger options.

For standalone healthchecks, the severity level defined by the script overrides the value in the trigger. For example, FAIL 40 or UNKNOWN 10, as is set in the hpraid health check (/cm/local/apps/cmd/scripts/healthchecks/hpraid).

Severity values are processed for the AlertLevel metric (section 12.2.8) when the health check runs.

Default Templates For Health Checks And Triggers

A health check can also launch an action based on any of the response values.

Monitoring triggers have the following default templates:

- Failing health checks: With a default severity of 15
- Passing health checks: With a default severity of 0
- Unknown health checks: With a default severity of 10

The severity level is one of the default parameters for the corresponding health checks. These defaults can also be modified to allow an action to be launched when the trigger runs, for example, sending an e-mail notification whenever any health check fails.

With the default templates, the actions are by default set for all health checks. However, specific actions that are launched for a particular measurable instead of for all health checks can be configured. To do this, one of the templates can be cloned, the trigger can be renamed, and an action can be set to launch from a trigger. The reader should be able to recognize that in the basic example of section 12.1 this is how, when the metric measurable CPUUser crosses 50 jiffies, the killallyestrigger is activated, and the killallyes action script is run.

12.2.5 Trigger

A *trigger* is a threshold condition set for a sampled measurable. When a sample crosses the threshold condition, it enters or leaves a zone that is demarcated by the threshold.

A trigger zone also has a settable severity (section 12.2.7) associated with it. This value is processed for the AlertLevel metric (section 12.2.8) when an action is triggered by a threshold event.

Triggers are discussed further in section 12.4.5.

12.2.6 Action

In the basic example of section 12.1, the action script is the script added to the monitoring system to kill all yes processes. The script runs when the condition is met that CPUUser crosses 50 jiffies.

An *action* is a standalone script or a built-in command that is executed when a condition is met, and has exit code 0 on success. The condition that is met can be:

- A FAIL, PASS, UNKNOWN, or no data from a health check
- A trigger condition. This can be a FAIL or PASS for conditional expressions.
- State flapping (section 12.2.9).

The actions that can be run are listed from within the action submode of the monitoring mode.

Example

```
[bright90->monitoring->action]% list
Type          Name (key)      Run on  Action
-----
Drain         Drain          Active  Drain node from all WLM
Email        Send e-mail    Active  Send e-mail
Event        Event          Active  Send an event to users with connected client
ImageUpdate  ImageUpdate    Active  Update the image on the node
PowerOff     PowerOff       Active  Power off a device
PowerOn      PowerOn        Active  Power on a device
PowerReset   PowerReset     Active  Power reset a device
Reboot       Reboot         Node    Reboot a node
Script       killlallyesaction Node    /cm/local/apps/cmd/scripts/actions/killlallyes
Script       killprocess    Node    /cm/local/apps/cmd/scripts/actions/killprocess.pl
Script       remount        Node    /cm/local/apps/cmd/scripts/actions/remount
Script       testaction     Node    /cm/local/apps/cmd/scripts/actions/testaction
Shutdown     Shutdown       Node    Shutdown a node
Undrain      Undrain        Active  Undrain node from all WLM
```







The Bright View equivalent is accessible via the clickpath:

Monitoring configuration→Actions (figure 12.17, section 12.4.4)

Configuration of monitoring actions is discussed further in section 12.4.4.

12.2.7 Severity

Severity is a positive integer value that the administrator assigns for a trigger. It takes one of these 6 suggested values:

Value	Name	Icon	Description
0	debug		debug message
0	info		informational message
10	notice		normal, but significant, condition
20	warning		warning conditions
30	error		error conditions
40	alert		action must be taken immediately

Severity levels are used in the `AlertLevel` metric (section 12.2.8). They can also be set by the administrator in the return values of health check scripts (section 12.2.4).

By default the severity value is 15 for a health check FAIL response, 10 for a health check UNKNOWN response, and 0 for a health check PASS response (section 12.2.4).

12.2.8 AlertLevel

AlertLevel is a special metric. It is sampled and re-calculated when an event with an associated Severity (section 12.2.7) occurs. There are three types of *AlertLevel* metrics:

1. *AlertLevel (count)*: the *number* of events that are at notice level and higher . The aim of this metric is to alert the administrator to the *number* of issues.
2. *AlertLevel (max)*: simply the maximum severity of the latest value of all the events. The aim of this metric is to alert the administrator to the severity of the *most important* issue.
3. *AlertLevel (sum)*: the *sum* of the latest severity values of all the events. The aim of this metric is to alert the administrator to the *overall severity* of issues.

12.2.9 Flapping

Flapping, or *State Flapping*, is when a measurable trigger that is detecting changes (section 12.4.5) that are too frequent. That is, the measurable goes in and out of the zone too many times over a number of samples. In the basic example of section 12.1, if the *CPUUser* metric crossed the threshold zone 5 times within 5 minutes (the default values for flap detection), then it would by default be detected as flapping. A flapping alert would then be recorded in the event viewer, and a flapping action could also be launched if configured to do so.

12.2.10 Data Producer

A data producer produces measurables. Sometimes it can be a group of measurables, as in the measurables provided by a data producer that is being used:

Example

```
[bright90->monitoring->measurable]% list -f name:25,producer:15 | grep ProcStat
BlockedProcesses      ProcStat
CPUGuest              ProcStat
CPUIdle               ProcStat
CPUIrq                ProcStat
CPUNice               ProcStat
CPUSoftIrq            ProcStat
CPUSteal              ProcStat
CPUSystem             ProcStat
CPUUser               ProcStat
CPUWait               ProcStat
CtxtSwitches          ProcStat
Forks                  ProcStat
Interrupts            ProcStat
RunningProcesses      ProcStat
```

Sometimes it may just be one measurable, as provided by a used data producer:

Example

```
[bright90->monitoring->measurable]% list -f name:25,producer:15 | grep ssh2node
ssh2node               ssh2node
```

It can even have no measurables, and just be an empty container for measurables that are not in use yet.

In *cmsh* all possible data producers (used and unused) can be listed as follows:

Example

```
[bright90->monitoring->setup]% list
```

The equivalent in Bright View is via the clickpath:

Monitoring Configuration→Data Producers

The data producers configured for an entity, such as a head node bright90, can be listed with the `monitoringproducers` command:

Example

```
[bright90->device[bright90]]% monitoringproducers
```

Type	Name	Arguments	Measurables	Node execution filters
AlertLevel	AlertLevel		3 / 231	<0 in submode>
CMDDaemonState	CMDDaemonState		1 / 231	<0 in submode>
ClusterTotal	ClusterTotal		18 / 231	<1 in submode>
Collection	NFS		32 / 231	<0 in submode>
Collection	sdt		0 / 231	<0 in submode>
DeviceState	DeviceState		1 / 231	<1 in submode>
HealthCheckScript	chrootprocess		1 / 231	<1 in submode>
HealthCheckScript	cmsh		1 / 231	<1 in submode>
HealthCheckScript	defaultgateway		1 / 231	<0 in submode>
HealthCheckScript	diskspace		1 / 231	<0 in submode>
HealthCheckScript	dmesg		1 / 231	<0 in submode>
HealthCheckScript	exports		1 / 231	<0 in submode>
HealthCheckScript	failedprejob		1 / 231	<1 in submode>
HealthCheckScript	hardware-profile		0 / 231	<1 in submode>
HealthCheckScript	ib		1 / 231	<0 in submode>
HealthCheckScript	interfaces		1 / 231	<0 in submode>
HealthCheckScript	ldap		1 / 231	<0 in submode>
HealthCheckScript	lustre		1 / 231	<0 in submode>
HealthCheckScript	mounts		1 / 231	<0 in submode>
HealthCheckScript	mysql		1 / 231	<1 in submode>
HealthCheckScript	ntp		1 / 231	<0 in submode>
HealthCheckScript	oomkiller		1 / 231	<0 in submode>
HealthCheckScript	opalinkhealth		1 / 231	<0 in submode>
HealthCheckScript	rogueprocess		1 / 231	<1 in submode>
HealthCheckScript	schedulers		1 / 231	<0 in submode>
HealthCheckScript	smart		1 / 231	<0 in submode>
HealthCheckScript	ssh2node		1 / 231	<1 in submode>
Job	JobSampler		0 / 231	<1 in submode>
JobQueue	JobQueueSampler		7 / 231	<1 in submode>
MonitoringSystem	MonitoringSystem		36 / 231	<1 in submode>
ProcMemInfo	ProcMemInfo		10 / 231	<0 in submode>
ProcMount	ProcMounts		2 / 231	<0 in submode>
ProcNetDev	ProcNetDev		18 / 231	<0 in submode>
ProcNetSnmp	ProcNetSnmp		21 / 231	<0 in submode>
ProcPidStat	ProcPidStat		5 / 231	<0 in submode>
ProcStat	ProcStat		14 / 231	<0 in submode>
ProcVMStat	ProcVMStat		6 / 231	<0 in submode>
Smart	SmartDisk		0 / 231	<0 in submode>
SysBlockStat	SysBlockStat		20 / 231	<0 in submode>
SysInfo	SysInfo		5 / 231	<0 in submode>
UserCount	UserCount		3 / 231	<1 in submode>

The displayed data producers are the ones configured for the entity, even if there are no measurables used by the entity.

Data producer configuration in Bright View is discussed further in section 12.4.1.

Access Control For Monitoring Data

Access control to data producers: An access control setting for a data producer determines who can plot (via the measurables monitoring interface used by Bright View and the User Portal), or view data (using the text-based interface of `cmsh` or `pythoncm`) from the measurables generated by a data producer. Thus, for example, the charts in the user portal (section 12.7) can be restricted according to the data producer that generates them.

There are three possible settings for access control for data producers. If the data producer is set to:

1. **Public:** then it means any user can, by default, plot/view data derived from that data producer. This is because by default a user has the token `PLOT_TOKEN` in their profile.
2. **Private:** then it means that a non-root user cannot, by default, plot/view data derived from that data producer. This is because, by default, non-root users do not have the token `PRIVATE_MONITORING_TOKEN` in their profile. If that token is in the profile, then the user has an elevated privilege, and can plot/view data, just like root.
3. **Individual:** then it means that a non-root user, by default, can plot/view the data only if the job associated with that data was run by that same non-root user. More verbosely, with the default user settings: the user who ran a job for which job measurables are produced by a data producer, must be the same as the user that who wants to plot/view the data, or else the data cannot be plotted/viewed.¹ The exception to this is, as already suggested, if the user that wants to plot/view the data is root, or has a user profile with the token `PRIVATE_MONITORING_TOKEN`. In that case the data can be plotted/viewed.

On a regular Bright Cluster Manager cluster, only a few low level data producers are set to private. An administrator can decide to set a data producer access control value to one of the three possible values, by using the `setup` submode of `monitoring` mode:

Example

```
[root@bright90 ~]# cmsh
[bright90]% monitoring setup
[bright90->monitoring->setup]% use mounts
[bright90->...[mounts]]% set access private
[bright90->...*[mounts*]]% commit
```

If a data producer is newly added, then by default its access control value is set to `public`. Changing this to `private` at a later time means that access to past and future data values from that data producer are affected by the `private` setting. If access is changed once more back to `public`, then it means that access to past and future data values are once again viewable and plottable by all users.

The current settings for access control for the data producers can be seen with:

Example

¹Even more verbosely: Individual access control is meant for job-based measurables, and works like this:

All monitoring data is stored per (entity, measurable) pair.

If the measurable has an access control value of `individual`, then a user check is performed. If the login name is the same as the user that owns the entity, then data can be plotted/viewed. If the user check does not match, then no data is returned.

Jobs—which are entities—are owned by the user that ran the job. Similarly Prometheus (entities) can have a `'user="Alice"'` label set, to define ownership. No other entity managed by Bright Cluster Manager is owned by a user.

For all unowned entities, `individual` access is equivalent to `private` access.

```
[root@bright90 ~]# cmsh -c "monitoring setup; list -f name:40,access"
name (key)                                access
-----
AggregateNode                             Public
AlertLevel                                Public
BigDataTools                              Public
CMDaemonState                             Public
Cassandra                                  Public
ClusterTotal                              Public
...
```

Access control to measurables: Measurables can also have access controls.

Access control for measurables is by default inherited from the data producer that generates it. It can be overwritten:

Example

```
[root@bright90 ~]# cmsh
[bright90]% monitoring measurable
[bright90->monitoring->measurable]% use loadone
[bright90->...[LoadOne]]% get access
Public (SysInfo)
[bright90->...[LoadOne]]% set access private
[bright90->...*[LoadOne*]]% commit
```

A measurable can thus take an access control value of `Public`, `Private`, or `Individual`. It can also explicitly be set to a value of `inherit`, which sets it to the value of its data producer. The inheritance is indicated in `cmsh` by enclosing the parent data producer in parentheses, as shown in the preceding example.

The current settings for access control for the measurables can be seen with:

Example

```
[root@bright90 ~]# cmsh -c "monitoring measurable; list -f name:40,access"
name (key)                                access
-----
AlertLevel                                Public
AlertLevel                                Public
AlertLevel                                Public
AvgJobDuration                            Public
BlockedProcesses                          Public
...
```

12.2.11 Conceptual Overview: The Main Monitoring Interfaces Of Bright View

Bright View has its own display modes, event and action results logs, and account handling screen. These can be selected via some of the 8 icons in the top right corner of the Bright View standard display (figure 12.5):

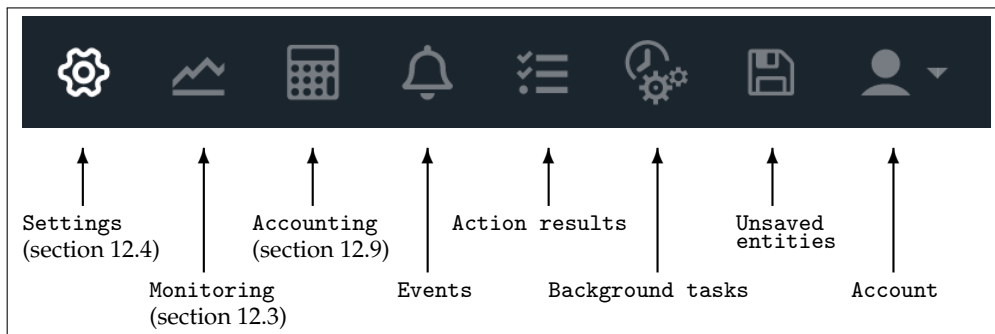


Figure 12.5: Bright View: Top Right Corner Icons

The 8 icons are described from left to right next:

1. **Settings mode** is the display that is shown when Bright View first starts up. Within **Settings mode** is the **Monitoring Configuration** resource. This resource should not be confused with the Bright View **Monitoring mode**, which is launched by the next icon in figure 12.5. **Monitoring configuration** is about configuring how items are monitored and how their data values are collected, and is discussed further in section 12.4.
2. The **Monitoring mode** allows visualization of the data values collected according to the specifications of the Bright View **Settings mode**. The visualization allows graphs to be configured, and is discussed further in section 12.3.
3. The **Accounting mode** typically allows visualization of job resources used by users, although it can be used to visualize job resources used by other aggregation entities. This is helpful tracking resources consumed by users. Job accounting is discussed further in section 12.9.
4. The **Events** icon allows logs of events (section 12.11.1) to be viewed.
5. The **Action results** icon allows the logs of the results of actions to be viewed.
6. The **Background tasks** icon allows background tasks to be viewed.
7. The **Unsaved entities** icon allows entities that have not yet been saved to be viewed.
8. The **Account handling** icon allows account settings to be managed for the Bright View user.

The monitoring aspects of the first two icons are discussed in greater detail in the sections indicated.

12.3 Monitoring Visualization With Bright View

The **Monitoring** icon in the menu bar of Bright View (item 2 in figure 12.5) launches an intuitive visualization tool that is the main GUI tool for getting a feel of the system's behavior over periods of time. With this tool the measurements and states of the system can be viewed as resizable and overlayable graphs. The graphs can be zoomed in and out on over a particular time period, the graphs can be laid out on top of each other or the graphs can be laid out as a giant grid. The graph scale settings can also be adjusted, stored and recalled for use the next time a session is started.

An alternative to Bright View's visualization tool is the command-line `cmsh`. This has the same functionality in the sense that data values can be selected and studied according to configurable parameters with it (section 12.6). The data values can even be plotted and displayed on graphs with `cmsh` with the help of unix pipes and graphing utilities. However, the strengths of monitoring with `cmsh` lie elsewhere: `cmsh` is more useful for scripting or for examining pre-decided metrics and health checks rather than a quick visual check over the system. This is because `cmsh` needs more familiarity with options, and is designed for text output instead of interactive graphs. Monitoring with `cmsh` is discussed in sections 12.5 and 12.6.

Visualization of monitoring graphs with Bright View is now described.

12.3.1 The Monitoring Window

If the Monitoring icon is clicked on from the menu bar of Bright View (figure 12.5), then a monitoring window for visualizing data opens up. By default, this displays blank plot panels—graph axes with a time scale going back some time on the x -axis, and with no y -axis measurable data values plotted (figure 12.6).

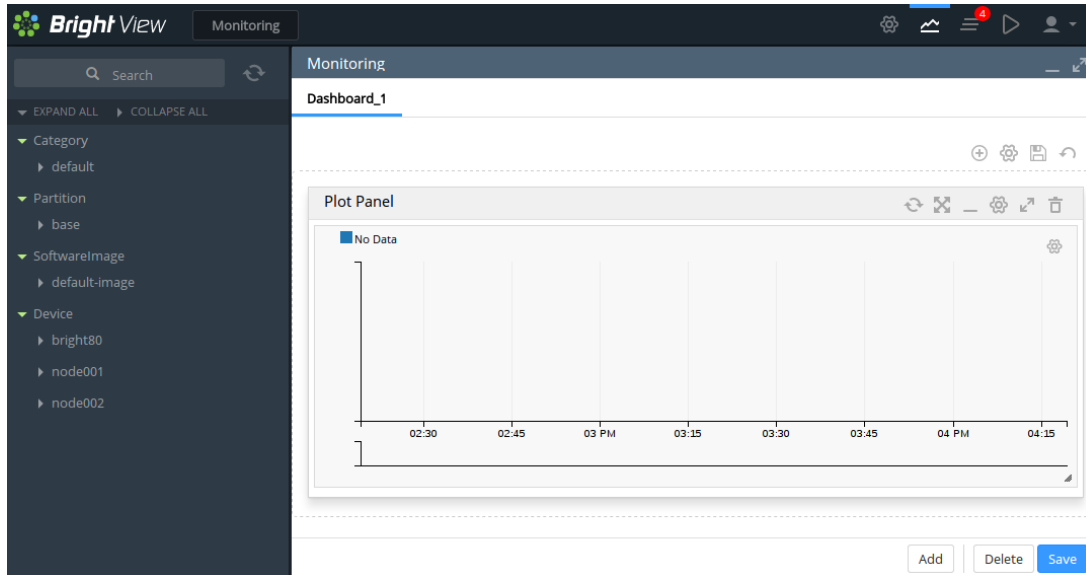


Figure 12.6: Bright View Monitoring Window: Blank Plot Panel

Finding And Selecting The Measurable To Be Plotted

To plot measurables, the entity which it belongs to should be selected from the navigation menu on the left-hand side. Once that has been selected, a class for that measurable can be chosen, and then the measurable itself can be selected. For example, to plot the measurable CPUUser for a head node bright90, it can be selected from the navigation clickpath `Device`→`bright90`→`CPU`→`CPUUser`.

Sometimes, finding a measurable is easier with the Search box. Typing in CPUUser there shows all the measurables with that text (figure 12.7). The search is case-insensitive.

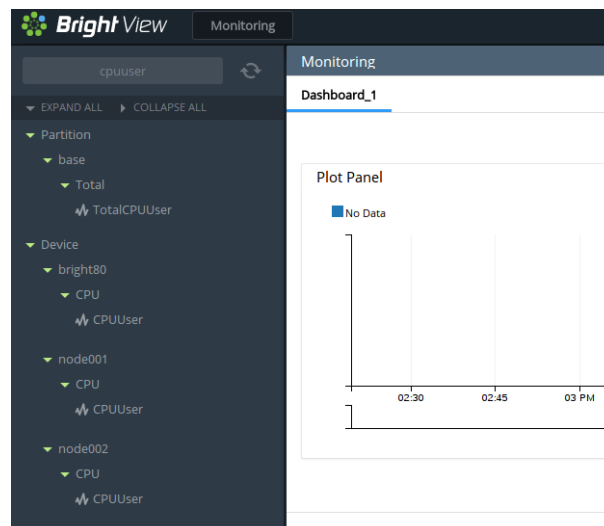


Figure 12.7: Bright View Monitoring Window: Search Box In Navigation

The search box can handle some simple regexes too, with `.` and `|` taking their usual meaning:

Example

- `node001.*cpuuser`: select a measurable with a data path that starts with `node001` and ends with `cpuuser`, with 0 or more characters of any kind in between.
- `(node001|node002).*cpuuser`: as for preceding example, but including `node002` as an alternative to `node001`.

The / (forward slash) allows filtering according to the data path. It corresponds to the navigation depth in the tree hierarchy:

Example

- `node001/cpu/cpuuser`: search for a measurable with a data path that matches `node001/cpu/cpuuser`

Plotting The Measurable

Once the measurable is selected, it can be drag-and-dropped into a plot panel. This causes the data values to be plotted.

When a measurable is plotted into a panel, two graph plots are displayed. The smaller, bottom plot, represents the polled value as a bar chart. The larger, upper plot, represents an interpolated line graph. Different kinds of interpolations can be set. To get a quick idea of the effect of different kinds of interpolations, <https://bl.ocks.org/mbostock/4342190> is an interactive overview that shows how they work on a small set of values.

The time axes can be expanded or shrunk using the mouse wheel in the graphing area of the plot panel. The resizing is carried out centered around the position of the mouse pointer.

12.4 Monitoring Configuration With Bright View

This section is about the configuration of monitoring for measurables, and about setting up trigger actions.

If Bright View is running in the standard Settings mode, which is the first icon in figure 12.5, page 494, then selecting Monitoring Configuration from the resources section makes the following menu items available for managing or viewing:

- Data Producers (section 12.4.1)
- Measurables (section 12.4.2)
- Consolidators (section 12.4.3)
- Actions (section 12.4.4)
- Triggers (section 12.4.5)
- Health status (section 12.4.6)
- All Health checks (section 12.4.7)
- Standalone Monitored Entities (section 12.4.8)



Figure 12.8: Bright View Monitoring Configuration Settings

These settings (figure 12.8) are now discussed in detail.

12.4.1 Monitoring Configuration: Data Producers

The Data Producers window lists all the data producers. Data producers are introduced in section 12.2.10.

Each data producer can have its settings edited within a subwindow. For example, the ProcStat data producer, which produces data for several measurables, including CPUUser, has the settings shown in figure 12.9:

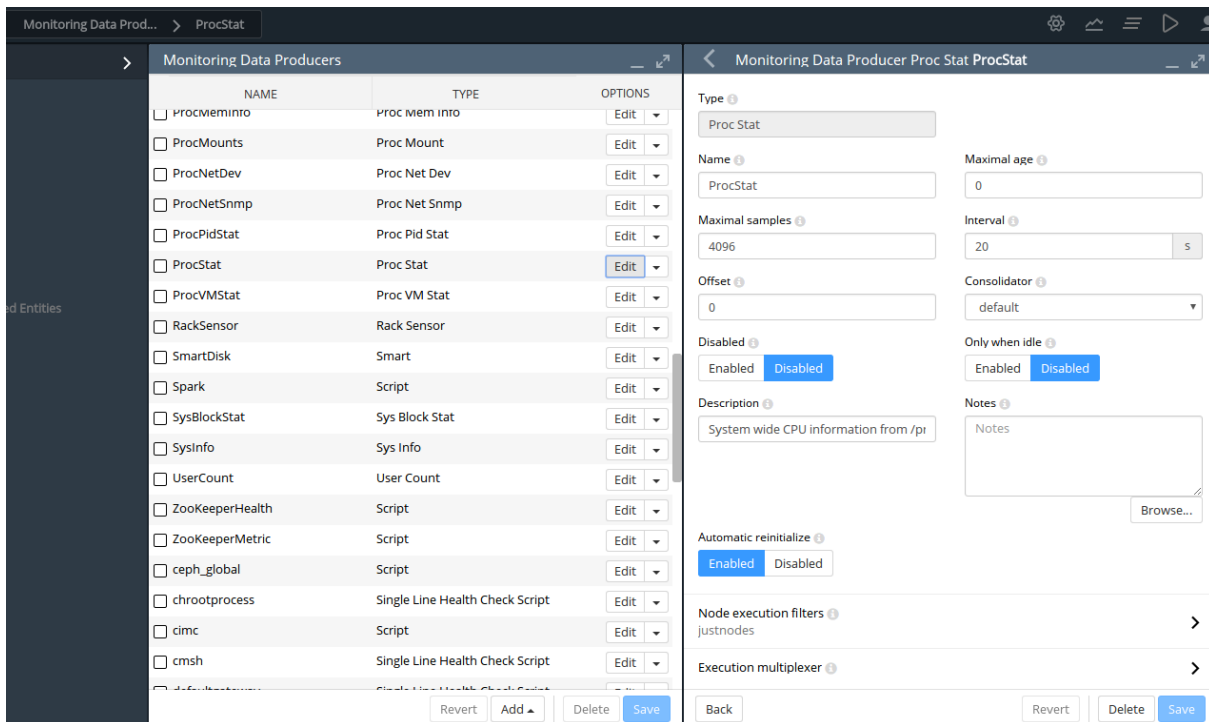


Figure 12.9: Bright View Monitoring Configuration Data Producer Settings

When the data producer takes samples to produce data, run length encoding (RLE) is used to compress the number of samples that are stored as data. Consolidation is carried out on the RLE samples. Consolidation in Bright Cluster Manager means gathering several data values, and making one value from them over time periods. Consolidation is done as data values are gathered. The point at which data values are discarded, if ever, is thus not dependent on consolidation.

Some remarks about some of the data producer settings that are seen in the subwindow of figure 12.9:

- **Maximal samples:** the maximum number of RLE samples that are kept. If set to 0, then the number of samples is not considered.
- **Maximal Age:** the maximum age of RLE samples that are kept. If **Maximal Age** is set to 0 then the sample age is not considered.

With **Maximal samples** and **Maximal Age**, the first of the rules that is reached is the one that causes the exceeding RLE samples to be dropped.

Samples are kept forever if **Maximal samples** and **Maximal Age** are both set to 0. This is discouraged due to the risk of exceeding the available data storage space.

- **Interval:** the interval between sampling, in seconds.
- **Offset:** A time offset from start of sampling. Some sampling depends on other sampling to be carried out first. This is used, for example, by data producers that rely on sampling from other data producers. For example, the **AggregateNode** data producer, which has measurables such as **TotalCPUIdle** and **TotalMemoryFree**. The samples for **AggregateNode** depend upon the **ProcStat** data producer, which produces the **CPUIdle** measurable; and the **ProcMemInfo** data producer, which produces the **MemoryFree** measurable.
- **Fuzzy offset:** a multiplier in the range from 0 to 1. It is multiplied against the sampling time interval to fix a maximum value for the time offset for when the sampling takes place. The actual offset used per node is spread out reasonably evenly within the range up to that maximum time offset.

For example, for a sampling time interval of 120s:

If the offset is 0, then there is no offset, and the sampling is attempted for all nodes at time instant when the interval restarts. This can lead to an overload at the time of sampling.

If, on the other hand, the offset is 0.25, then the sampling is done within a range offset from the time of sampling by a maximum of $0.25 \times 120\text{s} = 30\text{s}$. So, each node is sampled at a time that is offset by up to 30s from when the 120s interval restarts. From the time the change in value of the fuzzy offset starts working, the offset is set for each node. The instant at which sampling is carried out on a node then differs from the other nodes, even though each node still has an interval of 120s between sampling. An algorithm is used that tends to even out the spread of the instants at which sampling is carried out within the 30s range. The spreading of sampling has the effect of reducing the chance of overload at the time of sampling.

- **Consolidator:** By default, set to the default group. The default group consolidates (summarizes) the RLE samples over periods of an hour, a day, and a week. Consolidators are explained further in section 12.4.3.
- **Node execution filters:** A way to filter execution (restrict execution) of the data producer. It tells Bright Cluster Manager where the data producer runs. If not set, then the data producer runs on all nodes managed by CMDaemon. Filters can be for nodes, types, overlays, resources, and categories.
- **Execution multiplexer:** A way to multiplex execution (have execution work elsewhere) for a data producer. It tells Bright Cluster Manager about the entities that the data producer is sampling for. A data producer gathers data at the nodes defined by the node execution filter, and with multiplex execution the data producer gathers samples from other entities. These entities can be nodes, types, overlays, and resources. The entities from which it can sample are defined into groups called execution multiplexers. Execution multiplexers can thus be node multiplexers, type multiplexers, overlay multiplexers, or resource multiplexers.
- **When:** This has three possible values:
 - **Timed:** Data producer is run at a periodic Interval. This is the default.
 - **Pre job:** Data producer is only run before a job, in the prolog of the job. This is typically a check as part of the workload manager process (section 7.8.2).
 - **On demand:** Data producer is only run on demand, and not at a periodic Interval.
- **Only when idle:** By default a data producer runs regardless of how busy the nodes are. However, if the `Only when idle` setting is enabled, then the data producer runs only when the node is idle. Idle is a condition that is defined by the metric condition `LoadOne>1` (page 769).

12.4.2 Monitoring Configuration: Measurables

The Measurables window lists the available measurables (figure 12.10):

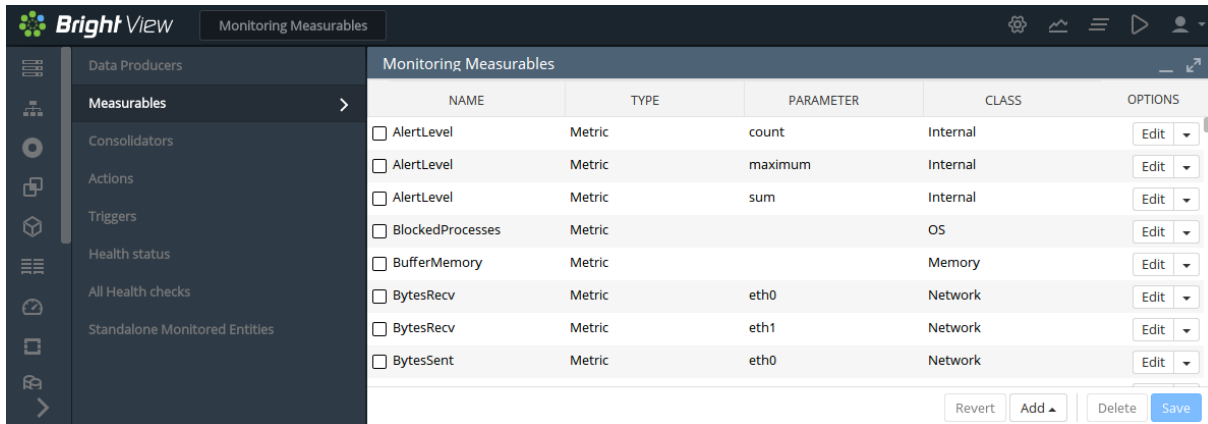


Figure 12.10: Bright View Monitoring Configuration Measurables

There are many measurables. It can therefore be useful to use the filtering-by-column option at the top of the columns. Each filtering option also has a search box associated with it (figure 12.11):

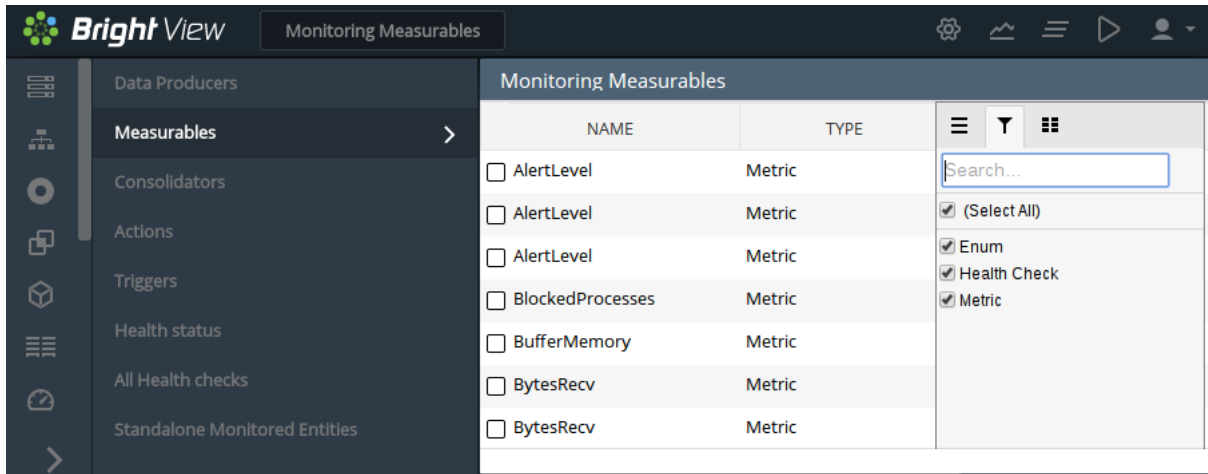


Figure 12.11: Bright View Monitoring Configuration Measurables Column Filter

From the measurables window, a subwindow can be opened with the Edit button for a measurable. This accesses the options for a particular measurable (figure 12.12):

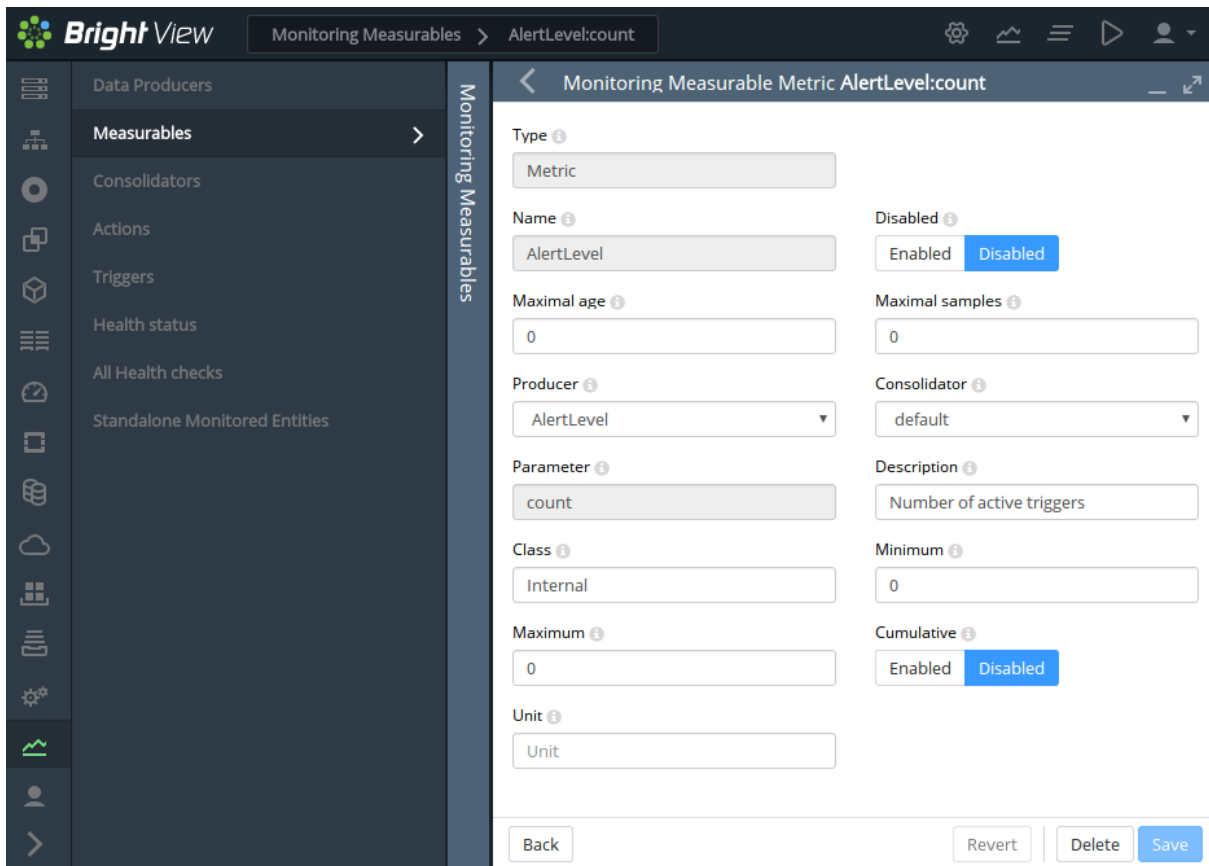


Figure 12.12: Bright View Monitoring Configuration Measurables Options Subwindow

The options shown include the sampling options: `Maximal age`, `Maximal samples`, `Consolidator`. The sampling options work as described for data producers (section 12.4.1).

Other options for a metric are setting the `Maximum` and `Minimum` values, the `Unit` used, and whether the metric is `Cumulative`.

If a metric is cumulative, then it is monotonic. Monotonic means that the metric only increments (is cumulative), as time passes. In other words, if the metric is plotted as a graph against time, with time on the x -axis, then the metric never descends. Normally the increments are from the time of boot onwards, and the metric resets at boot. For example, the number of bytes received at an interface is cumulative, and resets at boot time.

Usually the cluster administrator is only interested in the differential value of the metric per sample interval. That is, the change in the value of the current sample, from its value in the preceding sample. For example, bytes/second, rather than total number of bytes up to that time from boot.

12.4.3 Monitoring Configuration: Consolidators

Introduction To Consolidators

The concept of consolidators is explained using simple ascii graphics in Appendix K, while the `cmsh` interface to the `consolidators` submode is discussed in section 12.5.2.

In this current section, the Bright View interface to consolidators is discussed.

In Bright View, the `Consolidators` window lists all consolidator groups (figure 12.13).

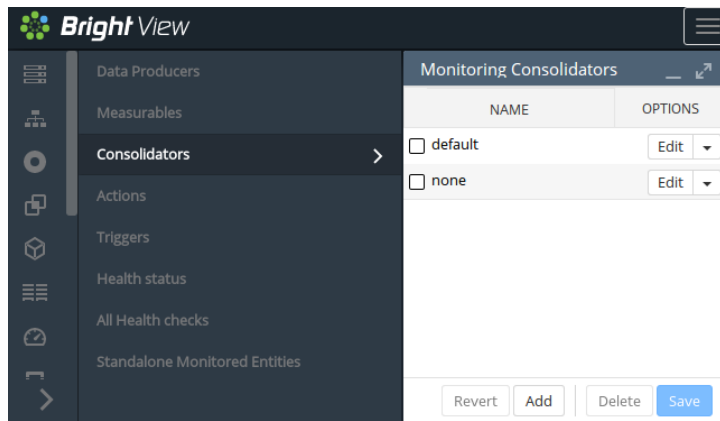


Figure 12.13: Bright View Monitoring Configuration Consolidator Groups

Subwindows allow the component consolidators to be created or modified (figure 12.14).

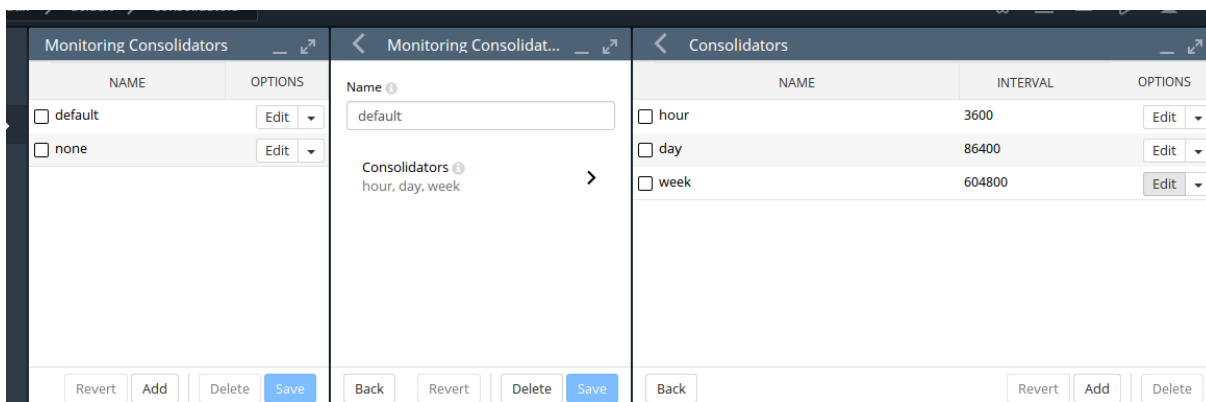


Figure 12.14: Bright View Monitoring Configuration Consolidator Groups Components

There are two pre-existing consolidator groups: `default` and `none`.

The `none` Consolidator Group

The `none` consolidator group, has no consolidators. Using a consolidator of `none` for a measurable or data producer means that samples are not consolidated. This can be dangerous if the cluster is more likely to run out of space due to unrestrained sampling, as can occur, for example, if `Maximal Age` and `Maximal samples` (section 12.4.1) for data producers are both set to 0.

The `default` Consolidator Group

The `default` consolidator group consists of the consolidators `hour`, `day`, and `week`. These are, unsurprisingly, defined to consolidate the samples in intervals of an hour, day, or week.

A consolidated value is generated on-the-fly. So, for example, during the hour that samples of a measurable come in, the hour consolidator uses the samples to readjust the consolidated value for that hour. When the hour is over, the consolidated value is stored for that hour as the data value for that hour, and a new consolidation for the next hour begins.

Consolidator values are kept, as for sample values, until the `Maximal Age` and `Maximal sample` settings prevent data values being kept.

Other Consolidator Group Possibilities

Other sets of custom intervals can also be defined. For example, instead of the `default` consolidator group, (hour, day, week), a similar group called `decimalminutes` consolidator group, (1min, 10min, 100min, 1000min, 10000min) could be created with the appropriate intervals (figure 12.15):

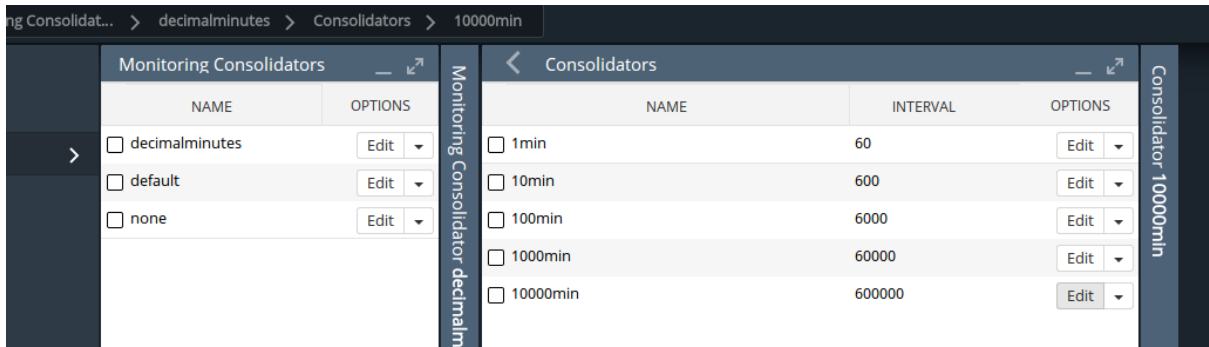


Figure 12.15: Bright View Monitoring Configuration Consolidators: decimalminutes Consolidator Group

Consolidator Item Settings

Consolidator items are members of the consolidator groups. The items have settings as properties, and can be managed (figure 12.16).

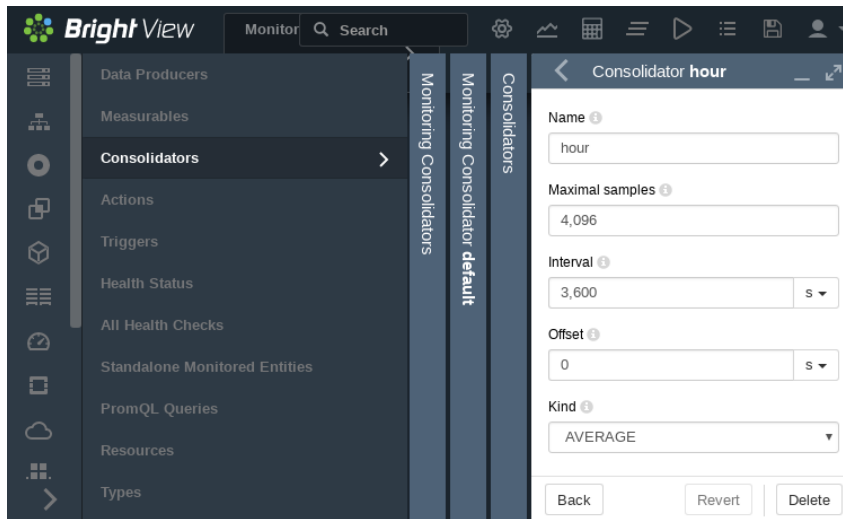


Figure 12.16: Bright View Monitoring Configuration Consolidators: Consolidator Item Settings

For the consolidator of hour, within the default consolidators group, the clickpath to edit its properties is:

Monitoring Configuration → Consolidators [default] → Edit → Consolidator [hour] → Edit

The properties that can be set for a consolidator item are:

- **Name:** The name of the consolidator item. By default, for the consolidator group default, the consolidator items with names of Day, Hour, and Month are already set up, with appropriate values for their corresponding fields.
- **Maximal samples:** The maximum number of samples that are stored for that consolidator item. This should not be confused with the Maximal samples of the measurable being consolidated.
- **Interval:** The time period (in seconds) covered by the consolidator sample. For example, the consolidator with the name Hour has a value of 3600. The property should not be confused with the time period between samples of the measurable being consolidated.
- **Offset:** The time offset from the default consolidation time, explained in more detail shortly.

- **Kind:** The kind of consolidation that is done on the raw data samples. The value of `kind` is set to `average` by default. The output result for a processed set of raw data—the consolidated data point—is an average, a maximum or a minimum of the input raw data values. `Kind` can thus have the value `Average`, `Maximum`, or `Minimum`. The value of `kind` is set to `average` by default.

For a given consolidator, when one `Kind` is changed to another, the historically processed data values become inconsistent with the newer data values being consolidated. Previous consolidated data values for that consolidator are therefore discarded during such a change.

To understand what `Offset` means, the `Maximal samples` of the measurable being consolidated can be considered. This is the maximum number of raw data points that the measurable stores. When this maximum is reached, the oldest data point is removed from the measurable data when a new data point is added. Each removed data point is gathered and used for data consolidation purposes.

For a measurable that adds a new data point every `Interval` seconds, the time $t_{\text{raw gone}}$, which is how many seconds into the past the raw data point is removed, is given by:

$$t_{\text{raw gone}} = (\text{Maximal samples})_{\text{measurable}} \times (\text{Interval})_{\text{measurable}}$$

This value is also the default consolidation time, because the consolidated data values are normally presented from $t_{\text{raw gone}}$ seconds ago, to further into the past. The default consolidation time occurs when the `Offset` has its default, zero value.

If however the `Offset` period is non-zero, then the consolidation time is offset, because the time into the past from which consolidation is presented to the user, $t_{\text{consolidation}}$, is then given by:

$$t_{\text{consolidation}} = t_{\text{raw gone}} + \text{Offset}$$

The monitoring visualization graphs then show consolidated data from $t_{\text{consolidation}}$ seconds into the past, to further into the past².

12.4.4 Monitoring Configuration: Actions

Actions are introduced in section 12.2.6. The `Actions` window (figure 12.17) displays actions that Bright Cluster Manager provides by default, and also displays any custom actions that have been created:

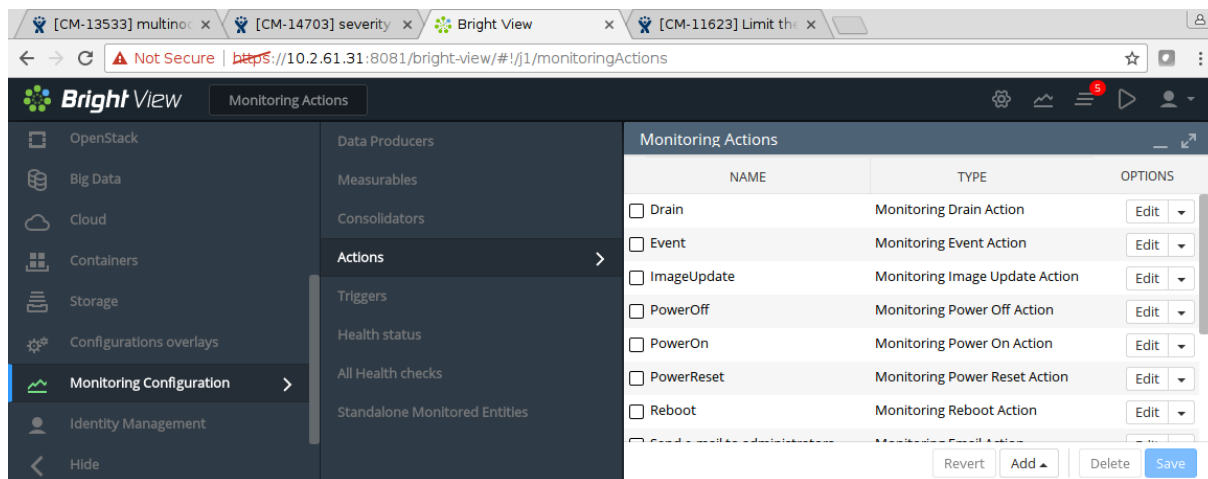


Figure 12.17: Bright View Monitoring Configuration: Actions

The `killalloyes` script from the basic example of section 12.1 would show up here if it has been implemented.

Actions are triggered, by triggers (section 12.4.5).

By default, the following actions exist:

² For completeness: the time $t_{\text{consolidation gone}}$, which is how many seconds into the past the consolidated data goes and is viewable, is given by an analogous equation to that of the equation defining $t_{\text{raw gone}}$:

$$t_{\text{consolidation gone}} = (\text{Maximalsamples})_{\text{consolidation}} \times (\text{Interval})_{\text{consolidation}}$$

- `Poweron`: Powers on the node
- `PowerOff`: Powers off the node
- `PowerReset`: Hard resets the node
- `Drain`: Drains the node (does not allow new jobs on that node)
- `Undrain`: Undrains the node (allows new jobs on that node)
- `Reboot`: Reboots node via the operating system
- `Shutdown`: Shuts the node down via the operating system
- `ImageUpdate`: Updates the node from the software image
- `Event`: Sends an event to users connected with `cmsh` or Bright View
- `killprocess`: A script to kill a process
- `remount`: A script to remount all devices
- `testaction`: A test script
- `Send e-mail to administrators`: Sends an e-mail out

The preceding actions show their options when the associated `Edit` button is clicked. A subwindow with options opens up. The following options are among those then displayed:

- `Run on`: What nodes the action should run on. Choices are:
 - `active`: the action runs on the active node only
 - `node`: the action runs on the triggering node
- `Allowed time`: The time interval in the 24 hour clock cycle that the action is allowed to start running. The interval can be restricted further to run within certain days of the week, months of the year, or dates of the month. Days and months must be specified in lower case.

Rather than defining a formal syntax, some examples are given, with explanations:

- `november-march`: November to March. April to October are forbidden.
- `november-march{monday-saturday}`: As in the preceding, but all Sundays are also forbidden.
- `november-march{monday-saturday{13:00-17:00}}`: Restricted to the period defined in the preceding example, and with the additional restriction that the action can start running only during the time 13:00-17:00.
- `09:00-17:00`: All year long, but during 09:00-17:00 only.
- `monday-friday{9:00-17:00}`: All year long, but during 9:00-17:00 only, and not on Saturdays or Sundays.
- `november-march{monday-saturday{13:00-17:00}}`: Not in April to October. In the other months, only on Mondays to Saturdays, from 13:00-17:00.
- `may-september{monday-friday{09:00-18:00};saturday-sunday{13:00-17:00}}`: May to September, with: Monday to Friday 09:00-18:00, and Saturday to Sunday 13:00-17:00.
- `may{1-31}`: All of May.
- `may,september{1-15}`: All of May, and only September 1-15.
- `may,september{1-15{monday-friday}}`: All of May. And only September 1-15 Monday to Friday.

A BNF grammar for allowed times is given in section 3.2.1 of the *Developer Manual*.

The following action scripts have some additional options:

- Send e-mail to administrators: Additional options here are:
 - info: body of text inserted into the default e-mail message text, before the line beginning “Please take action”. The default text can be managed in the file `/cm/local/apps/cmd/scripts/actions/sendemail.py`
 - recipients: a list of recipients
 - all admins: uses the list of users in the Administrator e-mail setting in partition[base] mode
- killprocess, and testaction: Additional options for these are:
 - arguments: text that can be used by the script.
 - script: The location of the script on the file system.

12.4.5 Monitoring Configuration: Triggers

Triggers are introduced in section 12.2.5. The Triggers window (figure 12.18) allows actions (section 12.2.6) to be triggered based on conditions defined by the cluster administrator.

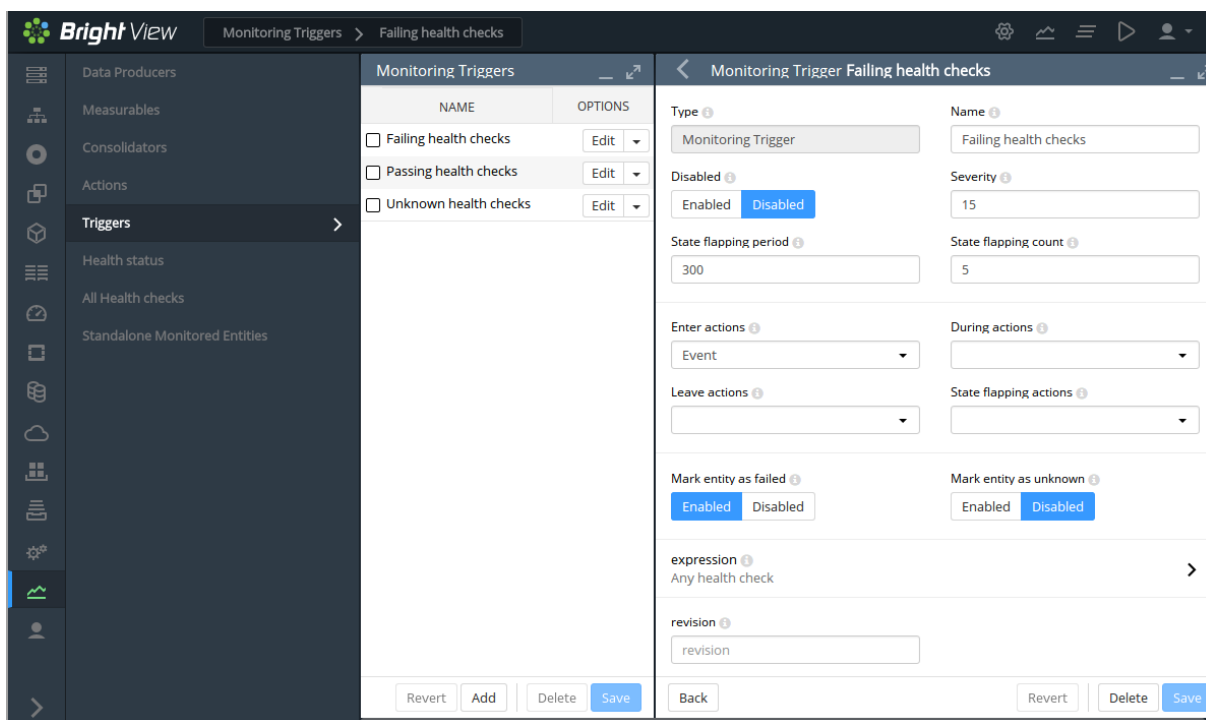


Figure 12.18: Bright View Monitoring Configuration: Triggers

Change Detection For Triggers

Triggers launch actions by detecting changes in the data of configured measurables. The detection of these changes can happen:

- When a threshold is crossed. That is: the latest sample value means that either the value has entered a zone when it was not in the zone in the preceding sample, or the latest sample means that the value has left a zone when it was in the zone in the preceding sample

- When the zone remains crossed. That is: the latest sample as well as the preceding sample are both within the zone of a crossed threshold.
- When state flapping is detected. This is when the threshold is crossed repeatedly (5 times by default) within a certain time period (5 minutes by default).

The monitoring configuration dialog triggers have four possible action launch configuration options to cover these cases:

1. **Enter actions:** if the sample has entered into the zone and the previous sample was not in the zone. This is a threshold-crossing change.
2. **Leave actions:** if the sample has left the zone and the previous sample was in the zone. This is also a threshold-crossing change.
3. **During actions:** if the sample is in the zone, and the previous sample was also in the zone.
4. **State flapping actions:** if the sample is entering and leaving the zone within a particular period (*State flapping period*, 5 minutes by default) a set number of times (*State flapping count*, 5 by default).

Pre-defined Triggers: Passing, Failing, And Unknown Health Checks

By default, the only triggers that are pre-defined are the following three health check triggers, which use the *Enter actions* launch configuration option, and which have the following default behavior:

- **Failing health checks:** If a health check fails, then on entering the state of the health check failing, an event is triggered as the action, and a severity of 15 is set for that health check.
- **Passing health checks:** If a health check passes, then on entering the state of the health check passing, an event is triggered as the action, and a severity of 0 is set for that health check.
- **Unknown health checks:** If a health check has an unknown response, then on entering the state of the health check returning an unknown response, an event is triggered as the action, and a severity of 10 is set for that health check.

Example: carry out a triggered action: `cmsh` or Bright View can be used for the configuration of carrying out an e-mail alert action (that is: sending out an e-mail) that is triggered by failing health checks.

- A `cmsh` way to configure it is:

The e-mail action (`send e-mail to administrators`) is first configured so that the right recipients get a useful e-mail.

```
[root@bright90 ~]# cmsh
[bright90]% monitoring action
[bright90->monitoring->action]% use send e-mail to administrators
[bright90->...[send e-mail to administrators]]% append recipients user1@example.com
[bright90->...*[send e-mail to administrators*]]% commit
```

Here, email alerts would go to `user1@example.com`, as well as to anyone already configured in `administratore-mail`. Additional text can be set in the body of the e-mail by setting a value for `info`.

The trigger can be configured to run the action when the health check enters a state where its value is `true`:

```
[bright90->monitoring->action[use send e-mail to administrators]]% monitoring trigger
[bright90->monitoring->trigger]% use failing health checks
[bright90->monitoring->trigger[Failing health checks]]% append enteractions send e-mail to administrators
[bright90->monitoring->trigger*[Failing health checks*]]% commit
```

The settings can be viewed with the show command. TAB-completion prompting can be used to suggest possible values for the settings.

- A Bright View way to carry out the configuration is using the clickpath:

Monitoring→Actions→Send e-mail to administrators→Edit

This can be used to set the recipients and other items, and the configuration saved.

The email action can then be configured in Bright View via the clickpath:

Monitoring→Triggers→Failing Health Checks→Edit→Enter Actions↓Send E-mail to Administrators

The checkbox for the "Send E-mail to Administrators" action should be ticked and the configuration saved.

Adding Custom Triggers: Any Measurable, Any Action

More triggers can be added. The killallyestrigger example from the basic example of section 12.1, seen in figures 12.3 and 12.4, is one such example.

The idea is that actions are launched from triggers, and the action for the trigger can be set to a predefined action, or to a custom action.

The Expression Subwindow For A Trigger

Clicking on the expression option in Trigger window of figure 12.18 opens up the expression subwindow. The expression for the trigger can then be configured by setting the entity, measurable, parameter, comparison operator, and measurable value, as shown in figure 12.19:

Figure 12.19: Bright View Monitoring Configuration: Triggers Expression

The trigger launch is carried out when, during sampling, CMDaemon evaluates the expression as being true.

An example `cmsh` session to set up an expression for a custom trigger might be as follows, where the administrator is setting up the configuration so that an e-mail is sent by the monitoring system when a node is detected as having gone down:

Example

```
[root@bright90 ~]# cmsh
[bright90]% monitoring trigger add nodedown
[bright90->monitoring->trigger*[nodedown*]]% expression
[bright90->monitoring->trigger*[nodedown*]->expression[compare]]% show
Parameter                               Value
-----
Name                                     compare
Revision
Type                                     MonitoringCompareExpression
Entities
Measurables
Parameters
Operator                                 ==
Value                                    FAIL
Use raw                                  no
[bright90->monitoring->trigger*[nodedown*]->expression*[compare*]]% set value down
[bright90->monitoring->trigger*[nodedown*]->expression*[compare*]]% set operator eq
[bright90->monitoring->trigger*[nodedown*]->expression*[compare*]]% set measurables devicestate
```

To add a touch of realism, a deliberate mistake is set here—the use of `devicestate` (the data producer) instead of `devicestatus` (the measurable). The `validate` command (page 35) gives a helpful warning here, so that the cluster administrator can fix the setting:

```
[bright90->monitoring->trigger*[nodedown*]->expression*[compare*]]% validate
Field      Message
-----
actions    Warning: No actions were set
measurables/ Warning: No known measurable matches the specified regexes ('devicestate', '')
parameters
[bright90->monitoring->trigger*[nodedown*]->expression*[compare*]]% set measurables devicestatus
[bright90->monitoring->trigger*[nodedown*]->expression*[compare*]]% ..;..
[bright90->monitoring->trigger*[nodedown*]]% set enteractions send e-mail to administrators
[bright90->monitoring->trigger*[nodedown*]]% commit
```

12.4.6 Monitoring Configuration: Health status

The `Health status` window (figure 12.20) displays all the nodes, and summarizes the results of all the health checks that have been run against them over time, by presenting a table of the associated severity levels (section 12.2.7):

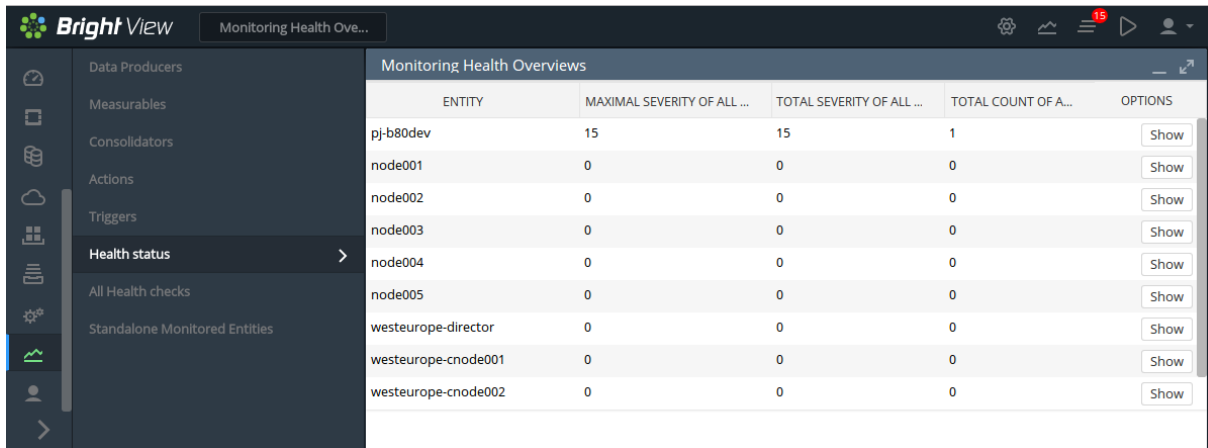


Figure 12.20: Bright View Monitoring Configuration: Health Status

In the example shown in figure 12.20 the first entity shows a severity issue, while the other devices are fine. Details of the individual health checks per node can be viewed in a subwindow using the Show button for a node.

Clicking on the Show button for the first entity in this example opens up a subwindow (figure 12.21). For this example the issue turns out to be due to an UNKNOWN status in the ssh2node measurable.

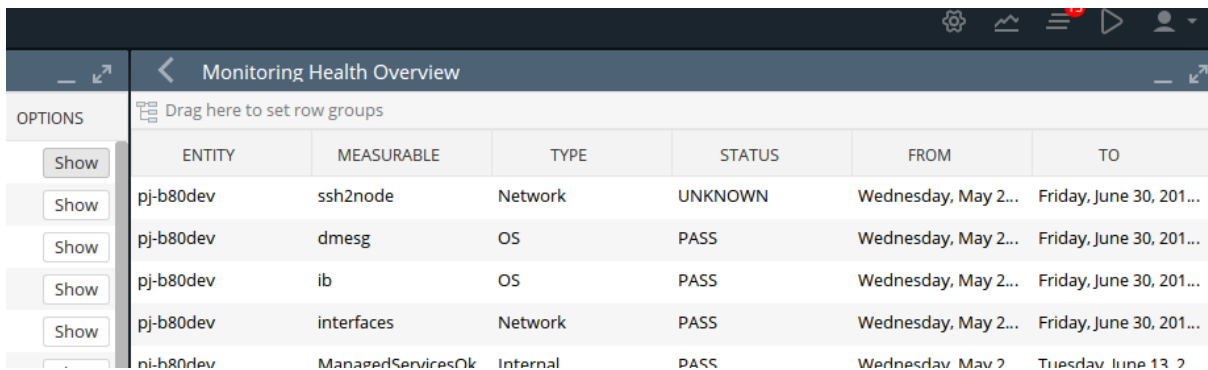


Figure 12.21: Bright View Monitoring Configuration: Health Status For An Entity

12.4.7 Monitoring Configuration: All Health Checks

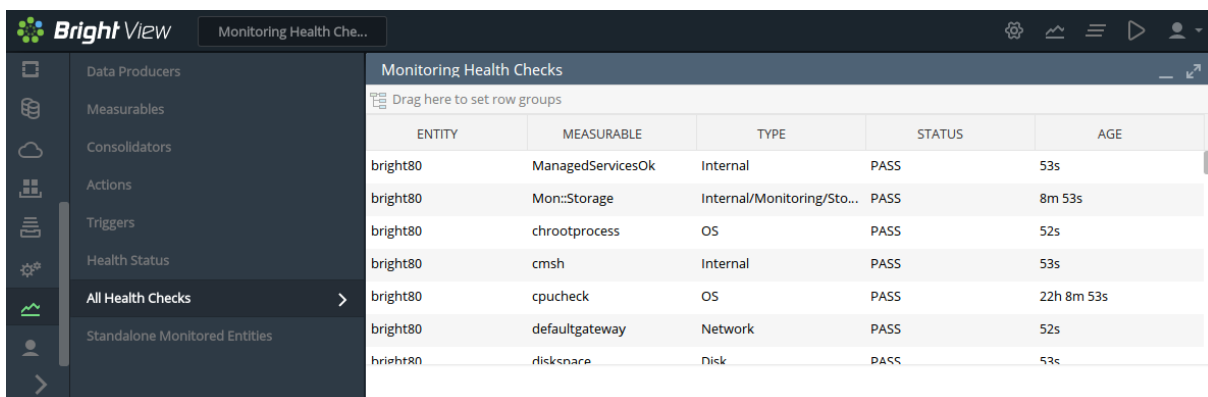


Figure 12.22: Bright View Monitoring Configuration: All Health Checks For All Entities

The All Health checks window shows all the running health checks for all entities. If the filtering-by-column option at the top of the column for the node is used to show the results for that node only, then the results are similar to what the Show button for a node produces in section 12.4.6, figure 12.21.

12.4.8 Monitoring Configuration: Standalone Monitored Entities

The Standalone Monitored Entities window allows the cluster administrator to define a standalone entity. A standalone entity is one that is not managed by Bright Cluster Manager—which means that no CMDaemon is running on it to gather data and for managing it—but the entity can still be monitored. For example, a workstation that is running the Bright View browser could be the standalone entity. This could have its connectivity monitored by pinging it from the head node with a custom script.

12.5 The monitoring Mode Of cmsh

This section covers how to use cmsh to configure monitoring. The monitoring mode in cmsh corresponds generally to the Monitoring Configuration resource of Bright View in section 12.4. Similarly to how monitoring subwindows are accessed in Bright View, the monitoring mode of cmsh is itself is not used directly, except as a way to access the monitoring configuration submodes of cmsh.

For this section some familiarity is assumed with handling of objects as described in the introduction to working with objects (section 2.5.3). When using cmsh's monitoring mode, the properties of objects in the submodes are how monitoring settings are carried out.

The monitoring mode of cmsh gives access to 8 modes under it:

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% monitoring help | tail -8
===== Monitoring =====
action ..... Enter action mode
consolidator ..... Enter consolidator mode
labeledentity ..... Enter labeled entity mode
measurable ..... Enter measurable mode
query..... Enter monitoring query mode
setup ..... Enter monitoring configuration setup mode
standalone ..... Enter standalone entity mode
trigger ..... Enter trigger mode
```

For convenience, a tree for monitoring submodes is shown in figure 12.23.

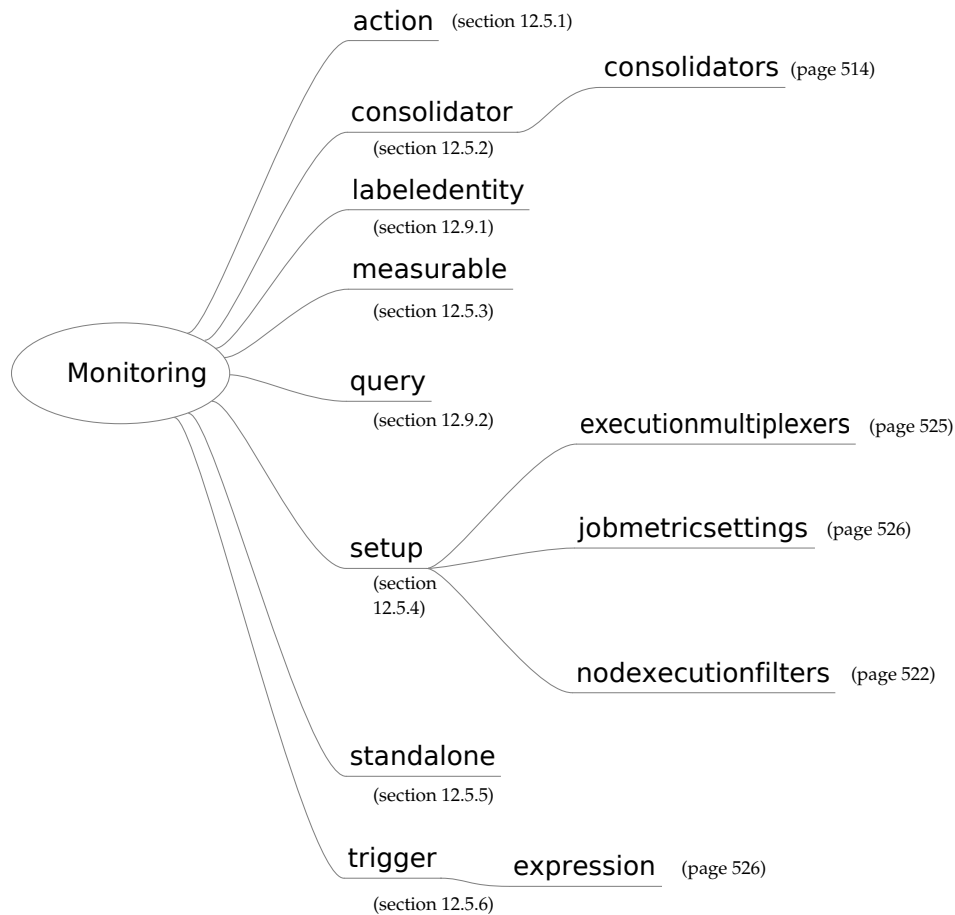


Figure 12.23: Submodes Under monitoring Mode

Sections 12.5.1–12.5.6 give examples of how objects are handled under these monitoring modes. To avoid repeating similar descriptions, section 12.5.1 is relatively detailed, and is often referred to by the other sections.

12.5.1 The `action` Submode

The `action` submode under the `monitoring` mode of `cmsh` allows monitoring actions to be configured. This mode in `cmsh` corresponds to the Bright View clickpath:

```
Monitoring Configuration→Actions
```

described earlier in section 12.4.4:

The `action` mode handles action objects in the way described in the introduction to working with objects (section 2.5.3). A typical reason to handle action objects—the properties associated with an action script or action built-in—might be to view the actions available, or to add a custom action for use by, for example, a metric or health check.

Some examples of how the `action` mode is used are now give.

The `action` Submode: `list`, `show`, **And** `get`

The `list` command by default lists the names and properties of actions available from `action` mode in a table:

Example

```
[myheadnode]% monitoring action
[myheadnode->monitoring->action]% list
Type          Name (key)      Run on  Action
-----
Drain         Drain          Active  Drain node from all WLM
Email        Send e-mail to Active  Send e-mail
             administrators
Event        Event          Active  Send an event to users with connected client
ImageUpdate  ImageUpdate    Active  Update the image on the node
PowerOff     PowerOff       Active  Power off a device
PowerOn      PowerOn        Active  Power on a device
PowerReset   PowerReset     Active  Power reset a device
Reboot       Reboot         Node    Reboot a node
Script       killprocess    Node    /cm/local/apps/cmd/scripts/actions/killprocess.pl
Script       remount        Node    /cm/local/apps/cmd/scripts/actions/remount
Script       testaction     Node    /cm/local/apps/cmd/scripts/actions/testaction
Shutdown     Shutdown       Node    Shutdown a node
Undrain      Undrain        Active  Undrain node from all WLM (node accepts new WLM jobs)
```

The preceding shows the actions available on a newly installed system.

The show command of cmsh displays the individual parameters and values of a specified action:

Example

```
[myheadnode->monitoring->action]% show poweroff
Parameter          Value
-----
Action              Power off a device
Allowed time
Disable             no
Name                PowerOff
Revision
Run on              Active
Type                PowerOff
```

Instead of using list, a convenient way to view the possible actions is to use the show command with tab-completion suggestions:

Example

```
[myheadnode->monitoring->action]% show<TAB><TAB>
drain      killprocess  powerreset  send\ e-mail\ to\ administrators  undrain
event      poweroff     reboot      shutdown
imageupdate  poweron     remount     testaction
```

The get command returns the value of an individual parameter of the action object:

Example

```
[myheadnode->monitoring->action]% get poweroff runon
active
```

The action Submode: add, use, remove, commit, refresh, modified, set, clear, **And** validate

In the basic example of section 12.1, in section 12.1.2, the killallyes action was cloned from a similar script using a clone option in Bright View.

The equivalent can be done with a clone command in cmsh. However, using the add command instead, while it requires more steps, makes it clearer what is going on. This section therefore covers adding the killallyes script of section 12.1.2 using the add command.

When `add` is used: an object is added, the object is made the current object, and the name of the object is set, all at the same time. After that, `set` can be used to set values for the parameters within the object, such as a path for the value of the parameter `command`.

Adding an action requires that the type of action be defined. Just as tab-completion with `show` comes up with action suggestions, in the same way, using tab-completion with `add` comes up with type suggestions.

Running the command `help add` in the action mode also lists the possible types. These types are `drain`, `e-mail`, `event`, `imageupdate`, `poweroff`, `poweron`, `powerreset`, `reboot`, `script`, `servicerestart`, `servicestart`, `servicestop`, `shutdown`, `undrain`.

The syntax for the `add` command takes the form:

```
add <type> <action>
```

If there is no `killalleges` action already, then the name is added in the action mode with the `add` command, and the `script` type, as follows:

Example

```
[myheadnode->monitoring->action]% add script killalleges
[myheadnode->monitoring->action*[killalleges*]]%
```

Using the `add` command drops the administrator into the `killalleges` object level, where its properties can be set. A successful commit means that the action is stored in `CMDaemon`.

The converse to the `add` command is the `remove` command, which removes an action that has had the `commit` command successfully run on it.

The `refresh` command can be run from outside the object level, and it removes the action if it has not yet been committed.

The `use` command is the usual way of "using" an object, where "using" means that the object being used is referred to by default by any command run. So if the `killalleges` object already exists, then `use killalleges` drops into the context of an already existing object (i.e. it "uses" the object).

The `set` command sets the value of each individual parameter displayed by a `show` command for that action. The individual parameter `script` can thus be set to the path of the `killalleges` script:

Example

```
[...oring->action*[killalleges*]]% set script /cm/local/apps/cmd/scripts/actions/killalleges
```

The `clear` command can be used to clear the value that has been set for `script`.

The `validate` command checks if the object has all required values set to sensible values. So, for example, `commit` only succeeds if the `killalleges` object passes validation.

Validation does not check if the script itself exists. It only does a sanity check on the values of the parameters of the object, which is another matter. If the `killalleges` script does not yet exist in the location given by the parameter, it can be created as suggested in the basic example of section 12.1, in section 12.1.2. In the basic example used in this chapter, the script is run only on the head node. If it were to run on regular nodes, then the script should be copied into the disk image.

The `modified` command lists changes that have not yet been committed.

12.5.2 The consolidator Submode

Consolidators are introduced in section 12.4.3. Consolidators can be managed in `cmsh` via the consolidator mode, which is the equivalent of the consolidators window (section 12.4.3) in Bright View.

The consolidator mode deals with groups of consolidators. One such pre-defined group is `default`, while the other is `none`, as discussed earlier in section 12.4.3:

```
[bright90->monitoring->consolidator]% list
Name (key)           Consolidators
-----
default             hour, day, week
none                <0 in submode>
```

Each consolidators entry can have its parameters accessed and adjusted.

For example, the parameters can be viewed with:

Example

```
[bright90->monitoring->consolidator]% use default
[bright90->monitoring->consolidator[default]]% show
Parameter           Value
-----
Consolidators       hour, day, week
Name                default
Revision
[bright90->monitoring->consolidator[default]]% consolidators
[bright90->monitoring->consolidator[default]->consolidators]% list
Name (key)          Interval
-----
day                 1d
hour                1h
week                1w
[bright90->monitoring->consolidator[default]->consolidators]% use day
[bright90->monitoring->consolidator[default]->consolidators[day]]% show
Parameter           Value
-----
Interval            1d
Kind                AVERAGE
Maximal age         0s
Maximal samples     4096
Name                day
Offset              0s
Revision
[bright90->monitoring->consolidator[default]->consolidators[day]]%
```

For the day consolidator shown in the preceding example, the number of samples saved per day can be doubled with:

Example

```
[bright90->monitoring->consolidator[default]->consolidators[day]]% set maximalsamples 8192
[bright90->monitoring->consolidator*[default*]->consolidators*[day*]]% commit
```

Previously consolidated data is discarded with this type of change, if the number of samples is reduced. Changing parameters should therefore be done with some care.

A new consolidators group can be created if needed.

A Bright View way, where a decimalminutes group is created, is discussed in the example in section 12.4.3, page 502.

A cmsh way, where a max-per-day group is created, is discussed in the following section:

Creation Of A Consolidator In cmsh

A new consolidator group, max-per-day, can be added to the default consolidator groups of default and none, with:

Example

```
[bright90]% monitoring consolidator
[bright90->monitoring->consolidator]% add max-per-day
[... [max-per-day*]]%
```

Within this new group, a new consolidator item, `max-per-day` can also be defined. The item can be defined so that it only calculates the maximum value per day, using the `kind` setting. Another setting is `interval`, which defines the interval with which the old data is compressed:

Example

```
[... [max-per-day*]]% consolidators
[... [max-per-day*]->consolidators]% add max-per-day
[... [max-per-day*]]% set interval 1d
[... [max-per-day*]]% set kind maximum
[... [max-per-day*]]% show
Parameter          Value
-----
Interval           1d
Kind               maximum
Maximal age        0s
Maximal samples    4096
Name               max-per-day
Offset             0s
Revision
[... [max-per-day*]]% commit
```

12.5.3 The measurable Submode

The measurable submode under the monitoring mode of `cmsh` handles measurable objects, that is: metrics, health checks, and enummetrics. This mode corresponds to the Bright View clickpath:

```
Monitoring Configuration->Measurables
```

covered earlier in section 12.4.2.

Measurable objects represent the configuration of scripts or built-ins. The properties of the objects are handled in `cmsh` in the way described in the introduction to working with objects (section 2.5.3).

A typical reason to handle measurable objects might be to view the measurables already available, or to remove a measurable that is in use by an entity.

Measurables cannot be added from this mode. To add a measurable, its associated data producer must be added from `monitoring setup` mode (section 12.5.4).

This section goes through a `cmsh` session giving some examples of how this mode is used.

The measurable Submode: `list`, `show`, **And** `get`

In measurable mode, the `list` command by default lists the names of all measurable objects along with parameters, their class, and data producer.

Example

```
[bright90->monitoring->measurable]% list
type          name (key)          parameter  class          producer
-----
Enum          DeviceStatus                Internal      DeviceState
HealthCheck   ManagedServicesOk        Internal     CMDaemonState
HealthCheck   Mon::Storage              Internal/Monitoring/Storage  MonitoringSystem
```

Metric	nfs_v3_server_total	Disk	NFS
Metric	nfs_v3_server_write	Disk	NFS
...			

The above example illustrates a list with some of the measurables that can be set for sampling on a newly installed system. A full list typically contains over two hundred items.

The list command in measurable submode can be run as:

- list metric: to display only metrics
- list healthcheck: to display only health checks
- list enum: to display only enummetrics

The show command of the measurable submode of monitoring mode displays the parameters and values of a specified measurable, such as, for example CPUUser, devicestatus, or diskspace:

Example

Example

```
[myheadnode->monitoring->measurable]% show cpuuser
Parameter      Value
-----
Class          CPU
Consolidator   default (ProcStat)
Cumulative     yes
Description    CPU time spent in user mode
Disabled       no (ProcStat)
Gap            0 (ProcStat)
Maximal age    0s (ProcStat)
Maximal samples 4,096 (ProcStat)
Maximum        0
Minimum        0
Name           CPUUser
Parameter
Producer       ProcStat
Revision
Type           Metric
Unit           Jiffies
[myheadnode->monitoring->measurable]% show devicestatus
Parameter      Value
-----
Class          Internal
Consolidator   none
Description    The device status
Disabled       no (DeviceState)
Gap            0 (DeviceState)
Maximal age    0s (DeviceState)
Maximal samples 4,096 (DeviceState)
Name           DeviceStatus
Parameter
Producer       DeviceState
Revision
Type           Enum
[myheadnode->monitoring->measurable]% show diskpace
Parameter      Value
```

```

-----
Class                Disk
Consolidator         - (diskspace)
Description          checks free disk space
Disabled             no (diskspace)
Gap                  0 (diskspace)
Maximal age          0s (diskspace)
Maximal samples      4,096 (diskspace)
Name                 diskspace
Parameter
Producer             diskspace
Revision
Type                 HealthCheck

```

The Gap setting here is a number. It sets how many samples are allowed to be missed before a value of NaN is set for the value of the metric.

As detailed in section 12.5.1, tab-completion suggestions for the show command suggest the names of objects that can be used, with the use command in this mode. For show in measurable mode, tab-completion suggestions suggests over 200 possible objects:

Example

```

[bright90->monitoring->measurable]% show
Display all 221 possibilities? (y or n)
alertlevel:count      iotime:vda          mon::storage::engine::elements  oomkiller
alertlevel:maximum    iotime:vdb          mon::storage::engine::size      opalinkhealth
alertlevel:sum        ipforwdatagrams     mon::storage::engine::usage     packetsrecv:eth0
blockedprocesses     ipfragcreates       mon::storage::message::elements  packetsrecv:eth1
buffermemory         ipfragfails         mon::storage::message::size      packetssent:eth0
bytesrecv:eth0       ipfragoks           mon::storage::message::usage     packetssent:eth1
...

```

The single colon (“:”) indicates an extra parameter for that measurable.

Because there are a large number of metrics, it means that grepping a metrics list is sometimes handy.

When listing and grepping, it is usually a good idea to allow for case, and be aware of the existence of the parameter column. For example, the AlertLevel metric shown in the first lines of the tab-completion suggestions of the show command of the previous example, shows up as alertlevel. However the list command displays it as AlertLevel. There are also several parameters associated with the AlertLevel command. So using the case-insensitive -i option of grep, and using the head command to display the headers is handy:

Example

```

[bright90->monitoring->measurable]% list | head -2 ; list metric | grep -i alertlevel
type          name (key)          parameter  class          producer
-----
Metric        AlertLevel          count      Internal        AlertLevel
Metric        AlertLevel          maximum    Internal        AlertLevel
Metric        AlertLevel          sum        Internal        AlertLevel

```

The get command returns the value of an individual parameter of a particular health check object:

Example

```

[myheadnode->monitoring->measurable]% get oomkiller description
Checks whether oomkiller has come into action (then this check returns FAIL)
[myheadnode->monitoring->measurable]%

```


The measurable Submode: The has Command

The has command is used with a measurable to list the entities that use the measurable. Typically these are nodes, but it can also be other entities, such as the base partition.

Example

```
[bright90->monitoring->measurable]% has alertlevel:sum
bright90
node001
node002
[bright90->monitoring->measurable]% use devicesup
[bright90->monitoring->measurable[DevicesUp]]% has
base
```

The remaining commands in measurable mode, such as use, remove, commit, refresh, modified, set, clear, and validate; all work as outlined in the introduction to working with objects (section 2.5.3). More detailed usage examples of these commands within a monitoring mode are given in the earlier section covering the action submode (section 12.5.1).

The measurable Submode: An Example Session On Viewing And Configuring A Measurable

A typical reason to look at metrics and health check objects—the properties associated with the script or built-in—might be, for example, to view the operating sampling configuration for an entity.

This section goes through a cmsh example session under monitoring mode, where the setup submode (page 520) is used to set up a health check. The healthcheck can then be viewed from the measurable submode.

In the basic example of section 12.1, a trigger was set up from Bright View to check if the CPUUser metric was above 50 jiffies, and if so, to launch an action.

A functionally equivalent task can be set up by creating and configuring a health check, because metrics and health checks are so similar in concept. This is done here to illustrate how cmsh can be used to do something similar to what was done with Bright View in the basic example. A start is made on the task by creating a health check data producer, and configuring its measurable properties. using the setup mode under the monitoring mode of cmsh. The task is completed in the section on the setup mode in section 12.5.4.

To start the task, cmsh's add command is used, and the type is specified, to create the new object:

Example

```
[root@myheadnode ~]# cmsh
[myheadnode]% monitoring setup
[myheadnode->monitoring->setup]% add healthcheck cpucheck
[myheadnode->monitoring->setup*[cpucheck*]]%
```

The show command shows the parameters.

The values for description, runinbash, script, and class should be set:

Example

```
[...->setup*[cpucheck*]]% set script /cm/local/apps/cmd/scripts/healthchecks/cpucheck
[...->setup*[cpucheck*]]% set description "CPUUser under 50%?"
[...->setup*[cpucheck*]]% set runinbash yes
[...->setup*[cpucheck*]]% set class OS
[...->setup*[cpucheck*]]% commit
[myheadnode->monitoring->setup[cpucheck]]%
```

On running commit, the data producer cpucheck is created:

Example

```
[myheadnode->monitoring->setup[cpucheck]]% exit; exit
[myheadnode->monitoring]% setup list | grep -i cpucheck
HealthCheckScript      cpucheck          1 / 222      <0 in submode>
```

The measurable submode shows that a measurable cpucheck is also created:

Example

```
[myheadnode->monitoring]% measurable list | grep -i cpucheck
HealthCheck  cpucheck          OS          cpucheck
```

Since the cpucheck script does not yet exist in the location given by the parameter script, it needs to be created. One ugly bash script that can do a health check is:

```
#!/bin/bash

## echo PASS if CPUUser < 50
## cpu is a %, ie: between 0 and 100

cpu=`mpstat 1 1 | tail -1 | awk '{print $3}'`
comparisonstring="$cpu" < 50

if (( $(bc <<< "$comparisonstring") )); then
    echo PASS
else
    echo FAIL
fi
```

The script should be placed in the location suggested by the object, /cm/local/apps/cmd/scripts/healthchecks/cpucheck, and made executable with a chmod 700.

The cpucheck object is handled further within the cmsh monitoring setup mode in section 12.5.4 to produce a fully configured health check.

12.5.4 The setup Submode

The setup Submode: Introduction

The setup submode under the monitoring mode of cmsh allows access to all the data producers. This mode in cmsh corresponds to the Bright View clickpath:

Monitoring Configuration→Data Producers

covered earlier in section 12.4.1.

The setup Submode: Data Producers And Their Associated Measurables

The list of data producers in setup mode should not be confused with the list of measurables in measurable mode. Data producers are not the same as measurables. Data producers produce measurables, although it is true that the measurables are often named the same as, or similar to, their data producer.

In cmsh, data producers are in the Name (key) column when the list command is run from the setup submode:

Example

```
[bright90->monitoring->setup]% list
Type          Name (key)      Arguments  Measurables  Node execution filters
```

```

-----
AggregateNode      AggregateNode      8 / 222      <1 in submode>
AlertLevel         AlertLevel         3 / 222      <1 in submode>
CMDaemonState     CMDaemonState     1 / 222      <0 in submode>
ClusterTotal      ClusterTotal      18 / 222     <1 in submode>
Collection        BigDataTools      0 / 222      <2 in submode>
Collection        Cassandra          0 / 222      <1 in submode>
...

```

In the preceding example, the `AlertLevel` data producer has 3 / 222 as the value for measurables. This means that this `AlertLevel` data producer provides 3 measurables out of the 222 configured measurables. They may be enabled or disabled, depending on whether the data producer is enabled or disabled, but they are provided in any case.

To clarify this point: if the `list` command is run from setup mode to list producers, then the producers that have configured measurables are the ones that have 1 or more as the numerator value in the `Measurables` column. Conversely, the data producers with 0 in the numerator of the `Measurables` column have no configured measurables, whether enabled or disabled, and are effectively just placeholders until the software for the data producers is installed.

So, comparing the list of producers in setup mode with the measurables in measurable mode:

Example

In measurable mode, the three `AlertLevel` measurables (the 3 out of 222) produced by the `AlertLevel` producer can be seen with:

```

[bright90->monitoring->measurable]% list | head -2; list | grep AlertLevel
Type      Name (key)      Parameter      Class      Producer
-----
Metric    AlertLevel      count          Internal   AlertLevel
Metric    AlertLevel      maximum        Internal   AlertLevel
Metric    AlertLevel      sum            Internal   AlertLevel

```

On the other hand, in measurable mode, there are no measurables seen for `BigDataTools` (the 0 out of 222) produced by the `BigDataTools` producer, when running, for example: `list | head -2; list | grep BigDataTools`.

The setup Submode: Listing Nodes That Use A Data Producer

The `nodes` command can be used to list the nodes on which a data producer *<data producer>* runs. It is run in the setup submode level of the monitoring mode as:

```
nodes <data producer>
```

Example

```

[bright90->monitoring->setup]% list | head -2; list | grep mount
Type      Name (key)      Arguments      Measurables  Node execution filters
-----
HealthCheckScript  mounts          1 / 229        <0 in submode>
[bright90->monitoring->setup]% nodes mounts
node001..node003,bright90

```

The setup Submode: Data Producers Properties

Any data producer from the full list in setup mode can, if suitable, be used to provide a measurable for any entity.

An example is the data producer `AlertLevel`. Its properties can be seen using the `show` command:

Example

```
[bright90->monitoring->setup]% show alertlevel
Parameter                               Value
-----
Automatic reinitialize                  yes
Consolidator                            default
Description                             Alert level as function of all trigger severities
Disabled                                 no
Execution multiplexer                   <1 in submode>
Fuzzy offset                            0
Gap                                      0
Interval                                2m
Maximal age                              0s
Maximal samples                          4096
Measurables                             3 / 222
Name                                     AlertLevel
Node execution filters                   <1 in submode>
Notes                                    <0 bytes>
Offset                                   1m
Only when idle                           no
Revision                                 
Type                                     AlertLevel
When                                     Timed
```

These properties are described in section 12.4.1. Most of these properties are inherited by the measurables associated with the data producer, which in the AlertLevel data producer case are `alertlevel:count`, `alertlevel:maximum`, and `alertlevel:sum`.

The setup Submode: Deeper Submodes

One level under the setup submode of monitoring mode are 3 further submodes (modes deeper than submodes are normally also just called submodes for convenience, rather than sub-submodes):

- `nodeexecutionfilters`
- `executionmultiplexers`
- `jobmetricsettings`

Node execution filters: A way to filter execution (restrict execution) of the data producer.

If no node execution filter is set for that data producer, then the data producer runs on all nodes of the cluster. Filters are of type `node`, `category`, `overlay`, `resource`, and `lua`. The type is set when the filter is created.

- **The nodes command for listing the execution nodes**

Running the `nodes` command for a data producer lists which nodes the execution of the data producer is run on.

Example

```
[myhost->monitoring->setup]% nodes procmeminfo
mon001..mon003,myhost,osd001,osd002,node001,node002
[myhost->monitoring->setup]% nodes ssh2node
myhost
[myhost->monitoring->setup]% nodes devicestate
```

```

myhost
[myhost->monitoring->setup]% foreach * ( get name; nodes) | paste - - | sort
AggregateNode    myhost
AggregatePDU     myhost
AlertLevel       myhost
Cassandra        Not used
ceph_global      Not used
chrootprocess    myhost
ClusterTotal     myhost
CMDaemonState    node001,node002,myhost
...

```

Most of the default data producers that are used by the cluster run on an active head node, and often on the regular nodes.

- **nodexecutionfilters to restrict data producer execution**

The `rogueprocess` (page 789) data producer is one of the few that by default runs on a regular node. Restricting a data producer to run on a particular list of nodes can be carried out as follows on a cluster that is originally in its default state:

Example

```

[bright90->monitoring->setup[rogueprocess]]% nodeexecutionfilters
[bright90->monitoring->setup[rogueprocess]->nodeexecutionfilters]% add<TAB><TAB>
category lua      node      overlay  resource type
[...tup[rogueprocess]->nodeexecutionfilters]% add node justthese
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% show
Parameter          Value
-----
Filter operation    Include
Name                justthese
Nodes
Revision
Type                Node
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% set nodes node001,node002
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% show
Parameter          Value
-----
Filter operation    Include
Name                justthese
Nodes                node001,node002
Revision
Type                Node
[...tup*[rogueprocess*]->nodeexecutionfilters*[justthese*]]% commit

```

This way, the `rogueprocess` health check runs on just those nodes (`node001`, `node002`), and none of the others.

- **Restricting a data producer execution to the head node—monitoring a process on the head node**

Another example of data producer restriction is as follows: an administrator may wish to monitor the `slapd` process on the head node. In `cmsh`, a session to achieve this could be:

```

[bright90->monitoring->setup]% add procpidstat slapd
[bright90->monitoring->setup*[slapd*]]% set process slapd

```

```
[bright90->monitoring->setup*[slapd*]]% set consolidator none
[bright90->monitoring->setup*[slapd*]]% nodeexecutionfilters
[bright90->monitoring->setup*[slapd*]->nodeexecutionfilters]% add type headnodes
[bright90->monitoring->setup*[slapd*]->nodeexecutionfilters*[headnodes*]]% set headnode yes
[bright90->monitoring->setup*[slapd*]->nodeexecutionfilters*[headnodes*]]% show
[bright90->monitoring->setup*[slapd*]]% commit
```

The newly-defined slapd metric can now have its output displayed or plotted just like any other metric:

```
[bright90->device[bright90]]% latestmetricdata | grep slapd
MemoryUsed          slapd  Process  739 MiB   28.6s
SystemTime          slapd  Process  2m 35s   28.6s
ThreadsUsed         slapd  Process  50        28.6s
UserTime             slapd  Process  1h 36m   28.6s
VirtualMemoryUsed   slapd  Process  4.77 GiB 28.6s
```

- **Filtering a data producer by resource**

A data producer can also be set up so that it is run on a particular list of nodes filtered by resource. The resources that are available to a node can be viewed using the command `monitoringresources` for that device:

Example

```
[bright90->device[bright90]]% monitoringresources
Active
CentOS7u5
Ethernet
OpenStack Compute
RDO
boot
...
```

An example of where running a node execution filter by resource is useful, is for data producers that are intended to run on the active head node. Most data producers that are used by the cluster run on an active head node (besides often running on the regular nodes too).

Thus, for example, the `cpucheck` health check from page 519 can be set to run on the active head node, by creating an arbitrary resource called `myactive`:

Example

```
[bright90->monitoring->setup[cpucheck]]% nodeexecutionfilters
[...tup[cpucheck]->nodeexecutionfilters]% add resource "myactive"
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% show
Parameter          Value
-----
Filter              Include
Name                myactive
Operator            OR
Resources
Revision
Type                Resource
```

and then setting the `Resources` parameter to `Active`:

```
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% set resources Active
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% show
Parameter                               Value
-----
Filter                                   Include
Name                                     myactive
Operator                                  OR
Resources                                 Active
Revision
Type                                       Resource
[...tup*[cpucheck*]->nodeexecutionfilters*[myactive*]]% commit
```

The cpucheck health check then runs on the active head node, whichever head node it is.

When node execution filtering is carried out, the filtered data is not dropped by default. Filtered data can be dropped for a measurable or an entity with the `monitoringdrop` command (section 12.6.7).

Execution multiplexer: A way to multiplex execution (have execution work elsewhere) for a data producer. It tells Bright Cluster Manager about the entities that the data producer is sampling for. A data producer runs and gathers data at the entity (node, category, lua, overlay, resource type) defined by the node execution filter, and with multiplex execution the data producer gathers samples from other entities. These entities can be nodes, categories, lua scripts, overlays, resources, and types. The entities from which it can sample are defined into groups called execution multiplexers. Execution multiplexers can thus be node multiplexers, category multiplexers, lua multiplexers, type multiplexers, overlay multiplexers, or resource multiplexers.

The `executionmultiplexers` mode can be entered for a data producer `dmesg` with:

Example

```
root@bright90 ~]# cmsh
[bright90]% monitoring setup executionmultiplexers dmesg
```

Running the commands: `help add`, or `help set`, can be used to show the valid syntax in this submode.

Most data producers run on a head node, but sample from the regular nodes. So, for example, the `dmesg` health check from Appendix G.2.1 can be set to sample from the regular nodes by setting it to carry out execution multiplexing to specified node entities using a node multiplexer with the arbitrary name of nodes as follows:

Example

```
[bright90->monitoring->setup[dmesg]->executionmultiplexers]% add<TAB><TAB>
category lua      node      overlay resource type
[bright90->monitoring->setup[dmesg]->executionmultiplexers]% add node nodes
[bright90->...*[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter                               Value
-----
Filter operation                         Include
Name                                     nodes
Nodes
Revision
Type                                       Node

[bright90->...*[dmesg*]->executionmultiplexers*[nodes*]]% set nodes node001,node002
[bright90->...*[dmesg*]->executionmultiplexers*[nodes*]]% show
Parameter                               Value
```

Filter operation	Include
Name	nodes
Nodes	node001,node002
Revision	
Type	Node

The concepts and expected behavior of node execution filters and execution multiplexers is covered in more explicit detail in Appendix L.

Job Metrics Settings: Job metrics settings are a submode for setting job metric collection options for the JobSampler data producer (section 12.8.3).

12.5.5 The standalone Submode

The standalone submode under the monitoring mode of cmsh allows entities that are not managed by Bright Cluster Manager to be configured for monitoring. This mode in cmsh corresponds to the Bright View clickpath:

Monitoring Configuration→Standalone Monitored Entities

covered earlier in section 12.4.8.

The monitoring for such entities has to avoid relying on a CMDaemon that is running on the entity. An example might be a chassis that is monitored via a ping script running on the Bright Cluster Manager head node.

12.5.6 The trigger Submode

The trigger submode under the monitoring mode of cmsh allows actions to be configured according to the result of a measurable.

This mode in cmsh corresponds to the Bright View clickpath:

Monitoring Configuration→Triggers

covered earlier in section 12.4.5.

By default, there are 3 triggers:

Example

```
[bright90->monitoring->trigger]% list
Name (key)           Expression           Enter actions During actions Leave actions
-----
Failing health checks  (*, *, *) == FAIL   Event
Passing health checks  (*, *, *) == PASS   Event
Unknown health checks  (*, *, *) == UNKNOWN Event
```

Thus, for a passing, failing, or unknown health check, an event action takes place if entering a state change. The default severity level of a passing health check does not affect the AlertLevel value. However, if the failing or unknown health checks are triggered on entering a state change, then these will affect the AlertLevel value.

The trigger Submode: Setting An Expression

In the basic example of section 12.1, a trigger to run the killalleges script was configured using Bright View.

The expression that was set for the killalleges script in the basic example using Bright View can also be set in cmsh. For example:

Example

```
[bright90->monitoring->trigger]% add killallyesttrigger
[bright90->monitoring->trigger*[killallyesttrigger*]]% show
Parameter                               Value
-----
Disabled                                no
During actions
Enter actions
Leave actions
Mark entity as failed                    yes
Mark entity as unknown                   no
Name                                     killallyesttrigger
Revision
Severity                                  10
State flapping actions
State flapping count                     5
State flapping period                    5m
expression                               (*, *, *) == FAIL
[bright90->monitoring->trigger*[killallyesttrigger*]]% expression
[bright90->monitoring->trigger*[killallyesttrigger*]->expression[]]% show
Parameter                               Value
-----
Entities
Measurables
Name
Operator                                 EQ
Parameters
Revision
Type                                     MonitoringCompareExpression
Use raw                                  no
Value                                     FAIL
[bright90->monitoring->trigger*[killallyesttrigger*]->expression[]]% set entities bright90
[bright90->monitoring->trigger*[killallyesttrigger*]->expression*[%]]% set measurables CPUUser
[bright90->monitoring->trigger*[killallyesttrigger*]->expression*[%]]% set operator GT
[bright90->monitoring->trigger*[killallyesttrigger*]->expression*[%]]% set value 50
[bright90->monitoring->trigger*[killallyesttrigger*]->expression*[%]]% commit
[bright90->monitoring->trigger*[killallyesttrigger*]->expression*[%]]% set name killallyesregex
Field                                    Message
-----
actions                                  Warning: No actions were set
===== killallyesttrigger =====
[bright90->monitoring->trigger[killallyesttrigger]->expression[killallyesregex]]% exit
[bright90->monitoring->trigger[killallyesttrigger]->expression]% exit
[bright90->monitoring->trigger[killallyesttrigger]]% set enteractions killallyesname
[bright90->monitoring->trigger*[killallyesttrigger*]]% commit
[bright90->monitoring->trigger[killallyesttrigger]]%
```

The expression format is shown in cmsh as:

(<entity>, <measurable>, <parameter>) <comparison operator> <value>

Here:

- an entity, as described in section 12.2.1, can be, for example, a node, category, device, or software image. To include more than one entity for the comparison, the alternation (pipe, |) symbol can be used, with double quotes to enclose the expression.

Example

```
... [killallystrigger*]->expression[]% set entities "bright90|node001|compute|gpuimage"
```

In the preceding example, the entity `compute` could be a category, and the entity `gpuimage` could be a software image.

- a measurable (section 12.2.1) can be a health check, a metric, or an enumetric. For example: `CPUUsage`. Alternation works for `<measurable>` in a similar way to that for `<entity>`.
- a parameter is a further option to a measurable. For example, the `FreeSpace` metric can take a mount point as a parameter. Alternation works for `<parameter>` in a similar way to that for `<entity>`.
- the comparison operator can be:

EQ: equivalent to, displayed as `==`

NE: not equivalent to, displayed as `!=`

GT: greater than, displayed as `>`

LT: less than, displayed as `<`

If the user uses an arithmetic symbol such as `>` in `cmsh` as an unescaped entry, then the entry may unintentionally be interpreted by the shell. That is why the two-letter entries are used instead for entry, even though when displayed they display like the arithmetic symbols for easier recognition.

- the value can be a string, or a number.

The regex evaluates to `TRUE` or `FALSE`. The trigger runs its associated action in the case of `TRUE`.

The wildcard `*` implies any entity, measurable, or parameter when used with the appropriate position according to the syntax of the expression format.

Using `.*` is also possible to match zero or more of any characters.

Some further expression matching examples:

Example

True for any failing health check:

```
(*, *, *) == FAIL
```

Example

True for any nearly full local disk (less than 10MB left):

```
(*, FreeSpace, sd[a-z]) < 10MB
```

Example

True for any cloud node that is too expensive (price more than more than 10\$):

```
(.*cnode.*, Price, *) > 10$
```

Example

Excluding node `agw001`:

```
(^(?!.*agw001).*$, *, *) == FAIL
```

Example

True for any node in the data, gpu, or hpc categories, that has a nearly full local disk (less than 10MB left):

```
(!resource=category:data|category:gpu|category:hpc, FreeSpace, sd[a-z]) < 10MB
```

The unusual syntax in the preceding example is liable to change in future versions.

At the end of section 12.5.3 a script called `cpucheck` was built. This script was part of a task to use health checks instead of metrics to set up the functional equivalent of the behavior of the basic example of section 12.1. In this section the task is continued and completed as follows:

```
[bright90->monitoring->trigger]% expression killallyesttrigger
[...trigger[killallyesttrigger]->expression[killallyesregex]]% get measurables
CPUUser
[...trigger[killallyesttrigger]->expression[killallyesregex]]% set measurables cpucheck
[...trigger*[killallyesttrigger*]->expression*[killallyesregex*]]% commit
```

12.6 Obtaining Monitoring Data Values

The monitoring data values that are logged by devices can be used to generate graphs using the methods in section 12.3. 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`.

12.6.1 Getting The List Of Measurables For An Entity: The `measurables`, `metrics`, `healthchecks` And `enummetrics` Commands

The measurables for a specified entity can be seen with the `measurables` command, and the measurable subtypes can be seen with the corresponding measurable subset commands: `metrics`, `healthchecks` and `enummetrics`. The results look quite similar to the results of the `measurable` submode of the monitoring mode (section 12.5.3). However, for entities, the measurables are a sublist of the full number of measurables listed in the `measurable` submode, which in turn are only the list of measurables for the data producers that have been enabled.

For example, within device mode where the entities are typically the head node and regular nodes, running `metrics` with a specified entity shows only the metrics that are configured for that entity. Thus if the entity is a head node, then only head node metrics are shown; and if the entity is a regular node, only regular node metrics are shown:

Example

```
[bright90->device]% enummetrics node001
Type      Name      Parameter  Class      Producer
-----
Enum      DeviceStatus      Internal    DeviceState
[bright90->device]% use bright90
[bright90->device[bright90]]% measurables
Type      Name      Parameter  Class      Producer
-----
Enum      DeviceStatus      Internal    DeviceState
HealthCheck ManagedServicesOk  Internal    CMDaemonState
HealthCheck Mon::Storage      Internal/Monitoring/Storage  MonitoringSystem
...
[bright90->device[bright90]]% exit
[bright90->device]% metrics node001
Type      Name      Parameter  Class      Producer
-----
```

```

Metric      AlertLevel      count      Internal      AlertLevel
Metric      AlertLevel      maximum    Internal      AlertLevel
Metric      AlertLevel      sum        Internal      AlertLevel
Metric      BlockedProcesses  OS         OS            ProcStat
...

```

Typically the number of metrics listed on the head node will differ from those listed on a regular node. Whatever each number is, it cannot be more than the number of metrics seen in the number of metrics listed in the `measurable` submode of section 12.5.3.

The preceding example shows the `measurables` listing commands being carried out on head nodes and regular nodes. These commands can be used on other entities too. For example, the base partition in partition mode, where the measurables can be listed with:

Example

```

[bright90->device]% partition use base
[bright90->partition[base]]% measurables
Type      Name      Parameter  Class      Producer
-----
Metric    CoresTotal      Total      ClusterTotal
Metric    CoresUp         Total      ClusterTotal
Metric    DevicesClosed   Total      ClusterTotal
Metric    DevicesDown     Total      ClusterTotal
...

```

The values for metric samples and health checks can be obtained from within device mode in various ways, and are explained next.

12.6.2 On-Demand Metric Sampling And Health Checks

The `samplenow` Command For On-Demand Measurable Samples

An administrator can do live sampling, or sampling on-demand, for specified entities by using the `samplenow` command. The command has the following syntax:

```
samplenow [OPTIONS] [<entity>] [<measurable> ...]
```

The command can be run without options when an entity object, such as a node is used (output truncated):

Example

```

[bright90->device]% use bright90
[bright90->device[bright90]]% samplenow
Measurable      Parameter Type      Value      Age      Info
-----
AlertLevel      count      Internal  0          2.01s
AlertLevel      maximum    Internal  0          2.01s
AlertLevel      sum        Internal  0          2.01s
BlockedProcesses  OS         OS         0 processes 2.01s
BufferMemory      Memory     Memory     847 KiB     2.01s
...

```

The entity used can also be in other modes that have measurables, such as the base partition (output truncated):

Example

```
[bright90->device]% partition use base
[bright90->partition[base]]% samplenow
Measurable      Parameter      Type      Value      Age      Info
-----
CoresTotal      Total          24        0.001s
CoresUp         Total          24        0.001s
DevicesClosed   Total          0         0.001s
DevicesDown     Total          0         0.001s
DevicesTotal    Total          0         0.001s
DevicesUp       Total          0         0.001s
...
```

The `-n|--nodes` Option

The `-n` option is used to sample specified nodes or node ranges:

Example

```
[bright90->partition[base]]% device
[bright90->device]% samplenow -n node001..node002 loadone
Entity      Measurable      Parameter Type      Value      Age      Info
-----
node001     LoadOne         OS        0.04      0.08s
node002     LoadOne         OS        0         0.077s
```

The `--metrics` And `--checks` Option

For a particular entity:

- All metrics can be sampled on demand with the `--metrics` option
- All health checks can be sampled on demand with the `--checks` option

Example

```
[bright90->device]% samplenow --metrics loadone loadfifteen --n node001,node002
Entity      Measurable      Parameter Type      Value      Age      Info
-----
node001     LoadOne         OS        0.04      0.08s
node002     LoadOne         OS        0         0.077s
[bright90->device]% samplenow --checks -n node001..node002
Entity      Measurable      Parameter Type      Value      Age      Info
-----
node001     ManagedServicesOk      Internal PASS      0.177s
node001     defaultgateway          Network PASS      0.145s
node001     diskspace              Disk   PASS      0.16s
node001     dmesg                  OS     PASS      0.177s
[bright90->device]% samplenow --checks diskspace -n node001..node002
Entity      Measurable      Parameter Type      Value      Age      Info
-----
node001     diskspace              Disk   PASS      0.095s
node002     diskspace              Disk   PASS      0.097s
[bright90->device]%
```

The `--debug` Option

The `--debug` option passes `CMD_DEBUG=1` to the script environment. This can be used to provide extra information on what is happening during sampling.

Example

```
[bright90->device[node001]]% samplnow ntp
Measurable  Type      Value      Age      Info
-----
ntp         Internal  PASS       0.51s
[bright90->device[node001]]% samplnow --debug ntp
Measurable  Type      Value      Age      Info
-----
ntp         Internal  PASS       0.524s   command: "ps -e"+
[bright90->device[node001]]% samplnow --debug -v ntp
Measurable  Type      Value      Age      Info
-----
ntp         Internal  PASS       0.543s   command: "ps -e"
                                         ntpd process found, pid: 11226
                                         command: "/sbin/ntpq -pn"
                                         found time syspeer: 10.141.255.254
                                         send time request to 10.141.255.254
                                         received a reply from 10.141.255.254
                                         time from 10.141.255.254 : 1586171027.783
                                         time on node                : 1586171027.771
                                         time difference              : 0.012
                                         execution time 0.06

[bright90->device[node001]]% !service ntpd stop
Redirecting to /bin/systemctl stop ntpd.service
```

```
[bright90->device[node001]] samplnow --debug -v ntp
measurable  Type      Value      Age      Info
-----
ntp         Internal  UNKNOWN    10s     timed out after: 10s
```

Many scripts under `/cm/local/apps/cmd/scripts/` can have their debug output inspected with `samplnow --debug`.

A recursive `grep` on the head node, similar to the following, should show which scripts have a settable debug environment:

```
grep -r CMD_DEBUG /cm/local/apps/cmd/scripts/
```

The `-s|--status` Option

Nodes in device mode which have a status of UP, as seen by the status command, can be sampled with the `-s|--status` option:

Example

```
[bright90->device]% samplnow -s UP
Entity      Measurable      Parameter Type      Value      Age      Info
-----
bright90    AlertLevel      count      Internal  0          4.67s
bright90    AlertLevel      maximum    Internal  0          4.67s
bright90    AlertLevel      sum        Internal  0          4.67s
bright90    BlockedProcesses OS         0 processes 4.67s
bright90    BufferMemory     Memory     847 KiB   4.67s
bright90    BytesRecv       eth0       Network   357 MiB   4.67s
bright90    BytesRecv       eth1       Network   78.7 MiB  4.67s
...
```

The preceding example is truncated because it is quite lengthy. However, on the screen, for the device mode, it shows all the sample values for the measurables for all the entities—head node and regular nodes—that are up.

To restrict the results to node001 only, it can be run as:

Example

```
[bright90->device]% samplenow -s UP -n node001
Measurable      Parameter Type      Value      Age      Info
-----
AlertLevel      count      Internal  0          0.081s
AlertLevel      maximum    Internal  0          0.081s
AlertLevel      sum        Internal  0          0.081s
...
```

Sampling according to a device status value other than UP is also possible.

The help text for the `samplenow` command gives further details on its possible options.

The `latestmetricdata` and `latesthealthdata` commands (section 12.6.3) display the results from the latest metric and health samples that have been gathered by the cluster, rather than sampling on demand.

The `dumpmonitoringdata` command (section 12.6.4) displays monitoring data gathered over a period of time in a variety of formats.

12.6.3 The Latest Data And Counter Values—The `latest*data` And `latestmetriccounters` Commands

Within device mode, the values obtained by the latest measurable sampling run can be displayed for a specified entity with the `latestmonitoringdata`, `latestmetricdata` and `latesthealthdata` commands:

- `latestmetricdata`: The `latestmetricdata` command for a specified entity displays the most recent metric value that has been obtained by the monitoring system for each metric used by the entity. For displaying metrics on-demand in `cmsh`, the `samplenow --metrics` command (page 531) can be used for a specified entity.
- `latesthealthdata`: The `latesthealthdata` command for a specified entity displays the most recent value that has been obtained by the monitoring system for each health check used by the entity. For displaying health check responses on demand in `cmsh`, the `samplenow --checks` command (page 531) can be used for a specified entity.
- `latestmonitoringdata`: The `latestmonitoringdata` command for a specified entity combines the output of the `latesthealthdata` and `latestmetricdata` commands, i.e. it displays the latest samples of the measurables for that entity. For displaying measurables on-demand in `cmsh`, the `samplenow` command (page 530) can be run without options, for a specified entity.

The `latestmetriccounters` command, on the other hand, displays the latest cumulative counter values of the cumulative metrics in use by the entity.

Using The `latest*data` Commands

When using the `latest*data` commands, the device must be specified (some output elided):

Example

```
[bright90->device]% use node001
[bright90->device[node001]]% latestmetricdata
Measurable      Parameter Type      Value      Age      Info
-----
AlertLevel      count      Internal  0          1m 12s  FAIL schedulers
AlertLevel      maximum    Internal  0          1m 12s  FAIL schedulers
AlertLevel      sum        Internal  0          1m 12s
```

```

BlockedProcesses      OS          0 processes  1m 12s
BufferMemory          Memory     847 KiB     1m 12s
BytesRecv             eth0       Network     311.611 B/s 1m 12s
BytesRecv             eth1       Network     0 B/s       1m 12s
BytesSent             eth0       Network     349.953 B/s 1m 12s
BytesSent             eth1       Network     0 B/s       1m 12s
CPUGuest              CPU        0 Jiffies   1m 12s
...

```

Valid device grouping options and other options can be seen in the help text for the `latestmetricdata` and `latesthealthdata` commands.

Example

```

[bright90->device]% help latestmetricdata
Name: Latestmetricdata - Display the latest metric data

```

```
Usage: latestmetricdata [OPTIONS] [<entity>]
```

Options:

```

-v, --verbose
    Be more verbose

-n, --nodes <node>
    List of nodes, e.g. node001..node015,node020..node028,node030
    or ~/some/file/containing/hostnames

-g, --group <group>
    Include all nodes that belong to the node group, e.g. testnodes
    or test01,test03

```

...

By default the data values are shown with human-friendly units. The `--raw` option displays the data values as raw units.

Using The `latestmetriccounter` Command

The `latestmetriccounter` is quite similar to the `latestmetricdata` command, except that it displays only cumulative metrics, and displays their accumulated counts since boot. The `latestmonitoringcounter` command is an alias for this command.

Example

```

[bright90->device]% latestmonitoringcounters node001
Measurable      Parameter  Type      Value      Age      Info
-----
BytesRecv       eth0       Network   286 MiB    11.7s
BytesRecv       eth1       Network   0 B        11.7s
BytesSent       eth0       Network   217 MiB    11.7s
BytesSent       eth1       Network   0 B        11.7s
CPUGuest        CPU        0 Jiffies  11.7s
CPUIdle         CPU        60.1 MJiffies  11.7s
CPUIrq          CPU        0 Jiffies  11.7s
CPUNice         CPU        66 Jiffies  11.7s
...

```

The reader can compare the preceding example output against the example output of the `latestmetricdata` command (page 533) to become familiar with the meaning of cumulative output.

12.6.4 Data Values Over A Period—The `dumpmonitoringdata` Command

The `dumpmonitoringdata` command displays monitoring data values over a specified period. This is for an entity, such as:

- a node in device mode
- the base partition in partition mode
- an image in softwareimage mode
- a job in the jobs submode. The jobs submode is under the path `cmsh→wlm[<workload manager>]→jobs`, and using `dumpmonitoringdata` with it is covered on page 556.

Using The `dumpmonitoringdata` Command

A concise overview of the `dumpmonitoringdata` command can be displayed by typing in “`help dumpmonitoringdata`” in a `cmsh` mode that has entities.

The usage of the `dumpmonitoringdata` command consists of the following options and mandatory arguments:

```
dumpmonitoringdata [OPTIONS] <start-time> <end-time> <measurable> [entity]
```

The mandatory arguments: The mandatory arguments for the times, the measurables being dumped, and the entities being sampled, have values that are specified as follows:

- The measurable *<measurable>* for which the data values are being gathered must always be given. Measurables currently in use can conveniently be listed by running the `measurables` command (section 12.6.1).
- If *[entity]* is not specified when running the `dumpmonitoringdata` command, then it must be set by specifying the entity object from its parent mode of `cmsh` (for example, with `use node001` in device mode). If the mode is device mode, then the entity can also be specified via the options as a list, a group, an overlay, or a category of nodes.
- The time pair *<start-time>* or *<end-time>* can be specified as follows:
 - *Fixed time format:* The format for the times that make up the time pair can be:
 - * `[YY/MM/DD] HH:MM[:SS]`
(If `YY/MM/DD` is used, then each time must be enclosed in double quotes)
 - * The unix epoch time (seconds since 00:00:00 1 January 1970)
 - `now`: For the *<end-time>*, a value of `now` can be set. The time at which the `dumpmonitoringdata` command is run is then used.
 - *Relative time format:* One item in the time pair can be set to a fixed time format. The other item in the time pair can then have its time set relative to the fixed time item. The format for the non-fixed time item (the relative time item) can then be specified as follows:
 - * For the *<start-time>*, a number prefixed with “-” is used. It indicates a time that much earlier to the fixed end time.
 - * For the *<end-time>*, a number prefixed with “+” is used. It indicates a time that much later to the fixed start time.
 - * The number values also have suffix values indicating the units of time, as seconds (s), minutes (m), hours (h), or days (d).

The relative time format is summarized in the following table:

Unit	<start-time>	<end-time>
seconds:	-<number>s	+<number>s
minutes:	-<number>m	+<number>m
hours:	-<number>h	+<number>h
days:	-<number>d	+<number>d

The options: The options applied to the samples are specified as follows:

Option	Argument(s)	Description
-v, --verbose		show the rest of the line on a new line instead of cutting it off
-i, --intervals	<number>	number of samples to show
-u, --unix-epoch		use a unix timestamp instead of using the default date format
-d, --delimiter	"<string>"	set the delimiter to a character
--sum		sum over specified entities
--max		maximum over specified entities
--min		minimum over specified entities
--avg		average over specified entities
--raw		show the metric value without units
--human		show the metric value with units (default)
--consolidationinterval		retrieve data from the consolidator with specified interval
--consolidationoffset		retrieve data from the consolidator with specified (interval, offset)
--timeaverage		calculate the average for the entire interval for specified devices
--delta		display change relative to previous value
--clip		clip data samples to the requested interval

The following options are valid only for device mode:

-n, --nodes	<list>	for list of nodes
-g, --groups	<list>	for list of groups
-c, --categories	<list>	for list of categories
-r, --racks	<list>	for list of racks
-h, --chassis	<list>	for list of chassis
-e, --overlay	<list>	Include all nodes in list of overlays
--union		calculate the union of specified devices
--intersection		calculate the intersection of the specified devices
-l, --role	<role>	Filter all nodes in role
-s, --status	<state>	for nodes in state UP, OPENING, DOWN, and so on

Notes And Examples Of `dumpmonitoringdata` Command Use

Notes and examples of how the `dumpmonitoringdata` command can be used now follow:

Fixed time formats: Time pairs can be specified for fixed times:

Example

```
[bright90->device[node001]]% dumpmonitoringdata 18:00:00 18:02:00 loadone
Timestamp          Value      Info
-----
2017/08/30 17:58:00    0.02
2017/08/30 18:00:00    0.01
2017/08/30 18:02:00    0.02
```

Double quotes are needed for times with a YY/MM/DD specification:

Example

```
[bright90->device[node001]]% dumpmonitoringdata "17/08/30 18:00" "17/08/30 18:02" loadone
Timestamp          Value      Info
-----
2017/08/30 17:58:00    0.02
2017/08/30 18:00:00    0.01
2017/08/30 18:02:00    0.02
```

Unix epoch time can also be set:

Example

```
[bright90->device[node001]]% !date -d "Aug 30 18:00:00 2017" +%s
1504108800
[bright90->device[node001]]% dumpmonitoringdata 1504108800 1504108920 loadone
Timestamp          Value      Info
-----
2017/08/30 17:58:00    0.02
2017/08/30 18:00:00    0.01
2017/08/30 18:02:00    0.02
```

Intervals and interpolation: The `-i|--intervals` option interpolates the data values that are to be displayed. The option needs `<number>` samples to be specified. This then becomes the number of interpolated samples across the given time range. Using `"-i 0"` outputs only the non-interpolated stored samples—the raw data—and is the default.

Example

```
[bright90->device]% dumpmonitoringdata -i 0 -10m now loadone node001
Timestamp          Value      Info
-----
2017/07/21 14:56:00    0.01
2017/07/21 14:58:00    0.14
2017/07/21 15:00:00    0.04
2017/07/21 15:02:00    0.04
2017/07/21 15:04:00    0.08
2017/07/21 15:06:00    0.08
```

If the number of intervals is set to a non-zero value, then the last value is always no data, since it cannot be interpolated.

Example

```
[bright90->device]% dumppmonitoringdata -i 3 -10m now loadone node001
Timestamp                Value      Info
-----
2017/07/21 21:49:36      0
2017/07/21 21:54:36      0.0419998
2017/07/21 21:59:36      no data
```

A set of nodes can be specified for the dump:

```
[bright90->device]% dumppmonitoringdata -n node001..node002 -5m now cpuidle
Entity      Timestamp                Value      Info
-----
node001     2017/07/20 20:14:00      99.8258 Jiffies
node001     2017/07/20 20:16:00      99.8233 Jiffies
node001     2017/07/20 20:18:00      99.8192 Jiffies
node001     2017/07/20 20:20:00      99.8475 Jiffies
node002     2017/07/20 20:14:00      99.7917 Jiffies
node002     2017/07/20 20:16:00      99.8083 Jiffies
node002     2017/07/20 20:18:00      99.7992 Jiffies
node002     2017/07/20 20:20:00      99.815 Jiffies
[bright90->device]%
```

Summing values: The `-m|--sum` option sums a specified metric for specified devices, 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

```
[bright90->device]% dumppmonitoringdata -2h now -i 3 loadone -n node00[1-2] --sum
Timestamp                Value      Info
-----
2017/07/20 18:30:27      0.0292462
2017/07/20 19:30:27      0
2017/07/20 20:30:27      no data
```

Each entry in the values column in the preceding table is the sum of `loadone` displayed by `node001`, and by `node002`, at that time, as can be seen from the following corresponding table:

Example

```
[bright90->device]% dumppmonitoringdata -2h now -i 3 loadone -n node00[1-2]
Entity      Timestamp                Value      Info
-----
node001     2017/07/20 18:30:27      0
node001     2017/07/20 19:30:27      0
node001     2017/07/20 20:30:27      no data
node002     2017/07/20 18:30:27      0.0292462
node002     2017/07/20 19:30:27      0
node002     2017/07/20 20:30:27      no data
```

Each `loadone` value shown by a node at a time shown in the preceding table, is in turn an average interpolated value, based on actual data values sampled for that node around that time.

Displaying values during a specified time period, with --clip: The --clip option is used with a specified time period. If there are raw values within the period, these are displayed.

A value is displayed for the start of the period, either by selection of a raw value if it exists at the exact starting time, or via interpolation if there is no raw value at the exact starting time. Similarly, at the end of the period a raw value is shown if it exists, or an interpolated value is shown if it does not.

Example

```
[bright90->device[node001]]% dumpmonitoringdata -5m now bytessent:eth0 --clip
# Start : 1552400295 / Tue Mar 12 14:18:15 2019
# End   : 1552400595 / Tue Mar 12 14:23:15 2019
Timestamp          Value          Info
-----
2019/03/12 14:18:15    201.942 B/s
2019/03/12 14:20:15    217.883 B/s
2019/03/12 14:22:15    233.058 B/s
2019/03/12 14:23:15    235.831 B/s
```

In the preceding example, the first 3 samples are raw samples, the last sample is an interpolated value, over a time period evaluated as being from 14:18:15 to 14:23:15. The epoch times for this period, and corresponding human-readable values are shown in the heading to the output table.

Displaying according to status: The -s|--status option selects only for nodes with the specified state. A state is one of the values output by the cmsh command ds or device status. It is also one of the values returned by the enummetric DeviceStatus (section 12.2.2).

Example

```
[bright90->device]% dumpmonitoringdata -2m now loadone -s up
Entity      Timestamp          Value      Info
-----
bright90    2017/07/21 15:00:00    0.35
bright90    2017/07/21 15:02:00    0.53
node001     2017/07/21 14:12:00    0.04
node001     2017/07/21 15:02:00    0.04
node002     2017/07/21 15:00:00    0.22
node002     2017/07/21 15:02:00    0.21
[bright90->device]%
```

The argument to -s|--status can be specified with simple regexes, which are case insensitive. For example, inst.* covers the states installing, installer_failed, installer_rebooting, installer_callinginit, installer_unreachable, installer_burning.

Displaying deltas: The --delta option lists the difference between successive monitoring data values. It subtracts the previous data value from the current data value, and divides the result by the time interval between the two values.

Example

```
[bright90->device[node001]]% dumpmonitoringdata --delta -6m now pageout
Timestamp          Value          Delta          Info
-----
2018/09/10 17:49:28    1015.46 B/s    nan
2018/09/10 17:51:28    1.35 KiB/s     2.7 B/s/s
2018/09/10 17:53:28    1.34 KiB/s     -0.083 B/s/s
```

Deltas are useful for seeing patterns in rates of change. For example, to check an experimental version of CMDaemon for a memory leak, an administrator may run:

Example

```
[bright90->device[bright90]]% dumpmonitoringdata -2h now memoryused:cmd -n node001 --delta
Timestamp                Value      Delta      Info
-----
2018/08/15 10:00:00.812  68.4 MiB  nan
2018/08/15 10:02:00.812  68.4 MiB  0.0341333 B/s
2018/08/15 12:10:00.812  68.4 MiB  0 B/s
```

The roughly 0B/s increase over 2 hours in the preceding output is a good sign.

Displaying union and intersection sets: The `--union` option displays the union of a set of specified devices. The devices can be specified by the device grouping options (the options that are used to group *<lists>*, such as `-c`, `-r` and so on).

For example:

if the overlay `galeranodes` has the node `mon001`

and

the overlay `openstackhypervisors` has the nodes `node001`, and `node002`

then an example of a union of the set of these two overlays is:

Example

```
[bright90->device]% dumpmonitoringdata --union -3m now pageout -e galeranodes,openstackhypervisors
Entity      Timestamp                Value      Info
-----
mon001      2018/09/11 11:31:56.198  192 KiB/s
mon001      2018/09/11 11:33:56.198  17.8 KiB/s
node001     2018/09/11 11:31:28.996  1.37 KiB/s
node001     2018/09/11 11:33:28.996  1.22 KiB/s
node002     2018/09/11 11:32:04.509  1.54 KiB/s
node002     2018/09/11 11:34:04.509  1.30 KiB/s
[bright90->device]%
```

A union of sets in the same grouping option can be carried out using comma-separation for the list of sets. In the preceding example, the same grouping option is `-e|--overlay`.

For a union of different grouping options however, the syntax is different. For example, for a union of the `galeranodes` overlay, and a `node001` node, a similar example is:

Example

```
[bright90->device]% dumpmonitoringdata --union -3m now pageout -u -e galeranodes -n node001
Entity      Timestamp                Value      Info
-----
mon001      1536659036.198          17.3 KiB/s
mon001      1536659156.198          116 KiB/s
node001     1536659008.997          1023.99 B/s
node001     1536659128.996          1.26 KiB/s
[bright90->device]%
```

For an intersection of sets, the only syntax allowed is one that uses different grouping options:

Example

```
[bright90->device]% dumpmonitoringdata --intersection -3m now pageout -e galeranodes -n node001
No remaining entities
```

For intersection, comma-separation within one grouping option is pointless, and is not supported.

Some non-interpolating RLE quirks: When a sample measurement is carried out, if the sample has the same value as the two preceding it in the records, then the “middle” sample is discarded from storage.

Thus, when viewing the sequence of output of non-interpolated samples, identical values do not exceed two entries one after the other. This is a common compression technique known as Run Length Encoding (RLE). It can have some implications in the output of the `dumpmonitoringdata` command.

Example

```
[bright90->device[node001]]% dumpmonitoringdata -10m now threadsused:cmd
Timestamp          Value      Info
-----
2017/07/21 11:16:00    42
2017/07/21 11:20:00    42
2017/07/21 11:22:00    41
2017/07/21 11:24:00    42
2017/07/21 11:26:00    42
```

In the preceding example, data values for the number of threads used by CMDaemon are dumped for the last 10 minutes.

Because of RLE, the value entry around 11:18:00 in the preceding example is skipped. It also means that at most only 2 of the same values are seen sequentially in the Value column. This means that 42 is not the answer to everything.

For a non-interpolated value, the nearest value in the past, relative to the time of sampling, is used as the sample value for the time of sampling. This means that for non-interpolated values, some care may need to be taken due to another aspect of the RLE behavior: The time over which the samples are presented may not be what a naive administrator may expect when specifying the time range. For example, if the administrator specifies a 10 minute time range as follows:

Example

```
[bright90->softwareimage]% dumpmonitoringdata -10m now nodesup default-image
Timestamp          Value      Info
-----
2017/07/13 16:43:00    2
2017/07/20 17:37:00    2
[bright90->softwareimage]%
```

then here, because the dump is for non-interpolated values, it means that the nearest value in the past, relative to the time of sampling, is used as the sample value. For values that are unlikely to change much, it means that rather than 10 minutes as the time period within which the samples are taken, the time period can be much longer. Here it turns out to be about 7 days because the nodes happened to be booted then.

12.6.5 Monitoring Data Health Overview—The `healthoverview` Command

In figure 12.20, section 12.4.6, the Bright View clickpath

```
Monitoring→Health Status
```

showed an overview of the health status of all nodes.

The `cmsh` equivalent is the `healthoverview` command, which is run from within device mode. If run without using a device, then it provides a summary of the alert levels for all nodes.

The help text in `cmsh` explains the options for the `healthoverview` command. The command can be run with options to restrict the display to specified nodes, and also to display according to the sort order of the alert level values.

Example

```
[bright90->device]% healthoverview -n node00[1-3]
```

Device	Sum	Maximum	Count	Age	Info
node001	30	15	2	50.7s	hot, fan high
node002	30	15	2	50.7s	hot, fan high
node003	15	15	1	50.7s	hot

12.6.6 Monitoring Data About The Monitoring System—The `monitoringinfo` Command

The `monitoringinfo` command provides information for specified head nodes or regular nodes about the monitoring subsystem. The help text shows the options for the command. Besides options to specify the nodes, there are options to specify what monitoring information aspect is shown, such as storage, cache, or services.

Example

```
[bright90->device]% monitoringinfo -n node001
```

Service	Queued	Handled	Cache miss	Stopped	Suspended	Last operation
Mon::CacheGather	0	0	0	yes	no	-
Mon::DataProcessor	0	0	0	yes	no	-
Mon::DataTranslator	0	932,257	0	no	no	Mon Jul 24 11:34:00
Mon::EntityMeasurableCache	0	0	0	no	no	Thu Jul 13 16:39:52
Mon::MeasurableBroker	0	0	0	no	no	-
Mon::Replicate::Collector	0	0	0	yes	yes	-
Mon::Replicate::Combiner	0	0	0	yes	yes	-
Mon::RepositoryAllocator	0	0	0	yes	no	-
Mon::RepositoryTrim	0	0	0	yes	no	-
Mon::TaskInitializer	0	30	0	no	no	Thu Jul 13 16:39:52
Mon::TaskSampler	30	233,039	0	no	no	Mon Jul 24 11:34:00
Mon::Trigger::Actuator	0	0	0	yes	no	-
Mon::Trigger::Dispatcher	0	0	0	yes	no	-

Cache	Size	Updates	Requests
ConsolidatorCache	0	17	0
EntityCache	10	17	935,280
GlobalLastRawDataCache	87	17	0
LastRawDataCache	142	17	427,301
MeasurableCache	231	17	935,230

Cache	Up	Down	Closed
DeviceStateCache	3	0	0

Replicator	First	Last	Requests	Samples	Sources
ReplicateRequestHandler	-	-	0	0	

Cache	Queued	Delivered	Handled	Pickup
Cache	0	120	932,257	7,766

Plotter	First	Last	Count	Samples	Sources	Requests


```
RequestDispatcher - - 0 0 0 -
RequestHandler - - 0 0 0 -
```

Storage	Elements	Disk size	Usage	Free disk
Mon::Storage::Engine	0	0 B	0.0%	-
Mon::Storage::Message	0	0 B	0.0%	-
Mon::Storage::RepositoryId	0	0 B	0.0%	-

12.6.7 Dropping Monitoring Data With The `monitoringdrop` Command

Monitoring data gathering can be restricted to certain nodes using node execution filtering and execution multiplexers. Entire data producers can also be disabled with the `disable` option in monitoring mode. However, restricting or disabling leaves historical samples in storage—the existing monitoring data values do not automatically get removed. So, in `cmsh` and Bright View the latest known monitoring data values then still show up, with a forever-increasing age.

If a data producer is removed, then the associated data values for its measurable or measurables are removed.

Alternatively, if adding execution filters to a monitoring data producer is intended to be a permanent change, then all previously collected data can be dropped for filtered nodes.

For example, if the `ssh` connectivity to only cloud nodes is to be checked:

Example

```
[root@bright90 ~]# cmsh
[bright90]% monitoring setup use ssh2node
[...->monitoring->setup[ssh2node]]% executionmultiplexers
[...->executionmultiplexers]% show
Type                None
[...->executionmultiplexers]% use all nodes
[...->executionmultiplexers[All nodes]]% get types
Node
[...->executionmultiplexers[All nodes]]% set types CloudNode
[...->executionmultiplexers*[All nodes*]]% commit
```

After this is set, the monitoring data values for a non-cloud node can be checked. The `ssh2node` health check data values are then seen to be getting older, without any more updates being added. These health check data values can then be dropped using the `monitoringdrop` command from within the device mode of `cmsh` command.

It is wise to run a dry-run operation first, in order to make sure that no data values are unintentionally removed:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[...->device[node001]]% latestmonitoringdata | grep ssh2node
ssh2node  Network    PASS    43m 38s  filtered
[...->device[node001]]% monitoringdrop --dry-run --filtered
Entity    Measurable
-----
node001   ssh2node
[...->device[node001]]% monitoringdrop --filtered
Removed 1 entity, measurable pairs
[...->device[node001]]% latestmonitoringdata | grep ssh2node
[...->device[node001]]%
```

The `--force` option can be used to remove non-filtered old data, such as data from a disabled measurable. This is also useful when correcting a bad metric script. After fixing the script, the old (incorrect) data can be dropped.

Example

```
[root@bright90 ~]# cmsh
[bright90]% device
[...->device]% monitoringdrop --category default my-metric --force
Removed 32 entity, measurable pairs
```

A reboot or `CMDaemon` restart is required for the node to start collecting data again on a non-filtered metric which has been dropped with the `--force` option.

12.6.8 Monitoring Suspension And Resumption—The `monitoringsuspend` And `monitoringresume` Commands

The `monitoringsuspend` command suspends monitoring. The `monitoringresume` command resumes monitoring.

When suspension is applied to a head node, the regular nodes simply continue sampling data up to a maximum of 1 million samples per node. The available backlog is fetched upon resumption.

Suspension can be used during benchmarking to measure the results of benchmarking runs without having monitoring get in the way.

Suspension can also be used as a quick sanity check during regular cluster operation, as a way for an administrator to see if it is monitoring that is consuming excessive resources, in comparison with the other processes on the system. For example, running it on a head node (some output omitted or elided):

Example

```
[root@head1 ~]# sar -b 1
Linux 3.10.0-957.1.3.el7.x86_64 (head1) 10/14/2019 _x86_64_ (28 CPU)

04:41:00 PM tps rtps wtps bread/s bwrtn/s
04:41:02 PM 1481.82 0.00 1481.82 0.00 24355.56
04:41:03 PM 849.49 0.00 849.49 0.00 10367.68
04:41:04 PM 509.00 0.00 509.00 0.00 4440.00
04:41:05 PM 709.90 0.00 709.90 0.00 5853.47
04:41:06 PM 1209.00 0.00 1209.00 0.00 18168.00
~C
[root@head1 ~]# cmsh
[head1]% device use master
[head1->device[head1]]% monitoringsuspend
suspend 14 on head1
[head1->device[head1]]% monitoringinfo
Service          Queued    Handled    Cache miss  Stopped    Suspended
-----
Mon::CacheGather      425      39,857      0           no         yes
Mon::DataConverter     0         0           0           no         yes
Mon::DataProcessor    0        6,609,369   0           no         yes
Mon::DataTranslator   0        311,658     0           no         yes
Mon::EntityMeasurableCache 0         0           0           no         yes
Mon::MeasurableBroker 0         0           0           no         yes
Mon::PerpetualTaskManager 0         0           0           no         yes
Mon::Replicate::Collector 0         0           0           yes        yes
Mon::Replicate::Combiner 0         0           0           yes        yes
...
```

```
[head1->device[head1]]% quit
[root@head1 ~]# sar -b 1
Linux 3.10.0-957.1.3.el7.x86_64 (head1) 10/14/2019 _x86_64_ (28 CPU)

04:41:58 PM tps rtps wtps bread/s bwrtn/s
04:41:59 PM 4.04 0.00 4.04 0.00 96.97
04:42:00 PM 3.00 0.00 3.00 0.00 96.00
04:42:01 PM 4.04 0.00 4.04 0.00 96.97
04:42:02 PM 0.00 0.00 0.00 0.00 0.00
04:42:03 PM 0.00 0.00 0.00 0.00 0.00
04:42:04 PM 43.00 0.00 43.00 0.00 528.00
04:42:05 PM 0.00 0.00 0.00 0.00 0.00
04:42:06 PM 3.06 0.00 3.06 0.00 130.61
04:42:07 PM 0.00 0.00 0.00 0.00 0.00
```

In the preceding example monitoring is seen to be consuming significant resources.

After running `monitoringsuspend`, resuming monitoring should not be forgotten, and it should be done soon enough after suspension. If that is not done, then backlogged samples that exceed the limit of 1 million samples per node on the regular nodes would be lost. Resumption is carried out with:

Example

```
[root@head1 ~]# cmsh
[head1]% device use master
[head1->device[head1]]% monitoringresume
resume 14 on head1
```

CMDaemon Directive Settings To Reduce Monitoring Resource Consumption

The following CMDaemon directive changes may reduce the resource consumption due to monitoring:

Increasing the job account collection interval: by increasing the value of the `JobsSamplingMetricsInterval` directive (page 724).

Disabling job information collection completely: by setting the value of the `JobInformationDisabled` directive to 0 (page 727).

For the Slurm workload manager only, disabling job accounting: by setting the value of the `SlurmDisableAccountingParsing` directive to 0 (page 723).

Reducing the duration for which job data is stored: by reducing the value of the `JobInformationKeepDuration` (page 728).

12.6.9 Monitoring Pickup Intervals

All nodes cache their monitoring data. This cached data gets picked up by the active head node at a regular pickup interval.

It is possible to alter the pickup interval using the `monitoringpickup` command covered in this section. The command is run from device mode.

The current pickup intervals can be listed with:

Example

```
[bright90]% device
[bright90->device]% monitoringpickup
Hostname      Interval    Times      Priority
```

```

-----
bright90      2m      -      0
node001      2m      -      0
node002      2m      -      0

```

An interval can be set for one or more nodes. For example, a 1-minute pickup interval can be set as follows:

Example

```

[bright90]% device use node001
[bright90->device[node001]]% monitoringpickup --interval 1m
Changed 1 pickup intervals
[bright90->device[node001]]% monitoringpickup
-----
Hostname      Interval    Times      Priority
-----
node001       1m         1          100

```

The pickup interval is carried out only once by default, unless otherwise specified.

The `--times` option allows the number of times to be specified:

Example

```

[bright90]% device use node001
[bright90->device[node001]]% monitoringpickup --interval 1m --times 10
Changed 1 pickup intervals
[bright90->device[node001]]% monitoringpickup
-----
Hostname      Interval    Times      Priority
-----
node001       1m         10         100

```

The `--forever` option lets the pickup be carried out “forever”³.

Example

```

[bright90->device[node001]]% monitoringpickup --interval 30s --forever
Changed 1 pickup intervals
[bright90->device[node001]]% monitoringpickup
-----
Hostname      Interval    Times      Priority
-----
node001       30s        -          100

```

The `--priority` option applies the priority to equal or lower priority settings:

Example

```

[bright90]% device
[bright90->device]% monitoringpickup -n node00[1-2]
-----
Hostname      Interval    Times      Priority
-----
node001       1m         12         80
node002       1m         17         20
[bright90->device]% monitoringpickup -n node00[1-2] --interval 5s --priority 50
Changed 1 pickup intervals
[bright90->device]% monitoringpickup -n node00[1-2]
-----
Hostname      Interval    Times      Priority
-----
node001       1m         12         80
node002       5s         1          50

```

³Strictly speaking, “forever” means $(2^{64} - 1)$ times on the 64-bit architecture that Bright Cluster Manager runs on. For comparison, $(2^{64} - 1)$ seconds is about 585 billion years.

In the preceding example, parameters for node002 only were changed, as the priority setting for node001 was higher than the applied priority option that was requested. Thus, the Interval value became 5s, as specified, the Times value defaulted to 1, and the specified Priority value of 50 was applied to node002 only.

The further behavior of the pickup from node002 is as follows:

After picking up data once from node002, five seconds from the change, the interval becomes the default of 2 minutes once again:

```
[bright90->device]% monitoringpickup -n node00[1-2]
Hostname      Interval    Times    Priority
-----
node001       1m         12       80
node002       2m         0        0
```

The yet further behavior of the pickup, during the next pickup event, is then as follows:

The Times value of 0 becomes unset. The unset value is represented by -, and is equivalent to --forever.

In other words, if a monitoring interval is changed, and the change is not specified as "forever", then after the Times value has decremented to zero, the monitoring interval reverts to the default value of 2 minutes. The Times value then becomes a value of -, which implies forever, when the next pickup occurs.

The job metric sampler can also automatically modify the pickup interval for nodes. Every time a new job is started, all the nodes that are used by the job are assigned a modified pickup interval. The new values for the pickup can be managed in the jobmetricsettings mode of cmsd.

```
[bright90->...->jobmetricsettings]% show
Parameter      Value
-----
...
Pickup interval 5s
Pickup priority 50
Pickup times    12
```

12.7 The User Portal

12.7.1 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 bright90, it is accessible to users for a login via a browser at the URL <http://bright90>, or more directly via <https://bright90:8081/userportal>. The state of the cluster can then be viewed by the users via an interactive interface.

The first time a browser is used to login to the portal, a warning about the site certificate being untrusted appears.

The certificate is a self-signed certificate (the X509v3 certificate of Chapter 4 of the *Installation Manual*), generated and signed by Bright Computing, and the attributes of the cluster owner are part of the certificate. However, Bright Computing is not a recognized Certificate Authority (CA) like the CAs that are recognized by a browser, which is why the warning appears.

The SSL configuration file is located at `/etc/httpd/conf.d/ssl.conf`. Within `ssl.conf`, by default the PEM-encoded server certificate is set to `/etc/pki/tls/certs/localhost.crt`. The entry should be changed if generating and installing the certificate elsewhere. A key-based certificate can be used instead, with a private key then expected by the `ssl.conf` file at `etc/pki/tls/private/localhost.key`.

For a portal that is not accessible from the outside world, such as the internet, the warning about Bright Computing not being a recognized Certificate Authority is not an issue, and the user can simply accept the “untrusted” certificate.

For a portal that is accessible via the internet, some administrators may regard it as more secure to ask users to trust the self-signed certificate rather than external certificate authorities. Alternatively the administrator can replace the self-signed certificate with one obtained by a trusted recognized CA, for example the one at <https://letsencrypt.org>, if that is preferred.

12.7.2 Setting A Common Username/Password For The User Portal

By default, each user has their own username/password login to the portal. Removing the login is not possible, because the portal is provided by CMDaemon, and users must connect to CMDaemon.

A shared (common) username/password for all users can be set in the configuration file, `common-credentials.json`. The default username/password settings are blank, which means that common access is not enabled:

Example

```
[root@bright90 ~]# cat /cm/local/apps/cmd/etc/htdocs/userportal/assets/config/common-credentials.json
{
  "username": "",
  "password": ""
}
```

To enable common access:

- the common username and password must be added via `cmsh` or Bright View

Example

```
[root@bright90 ~]# cmsh
[bright90]% user
[bright90->user]% add forrestgump
[bright90->user[forrestgump]]% set password
enter new password:
retype new password:
[bright90->user*[forrestgump*]]% commit
[bright90->user[forrestgump]]% quit
```

- the common username and password should be set in the appropriate place in the configuration file, `common-credentials.json`:

Example

```
[root@bright90 ~]# cat /cm/local/apps/cmd/etc/htdocs/userportal/assets/config/common-credentials.json
{
  "username": "forrestgump",
  "password": "1forrest1"
}
```

A minor stumbling block for the unwary administrator is:

If using Bright View, then if the password for the username has already been saved in the browser’s password manager before changing it in the configuration file, then the password saved in the browser’s password manager may need to be changed to the new one explicitly.

12.7.3 User Portal Access

By default, the user profile (section 6.4) is set to `readonly`, which allows viewing of the information presented in the user portal, without allowing it to be altered.

12.7.4 User Portal Home Page

User Portal Overview Page

The default user portal home page is the Overview page. This allows a quick glance to convey the most important cluster-related information for users (figure 12.24):

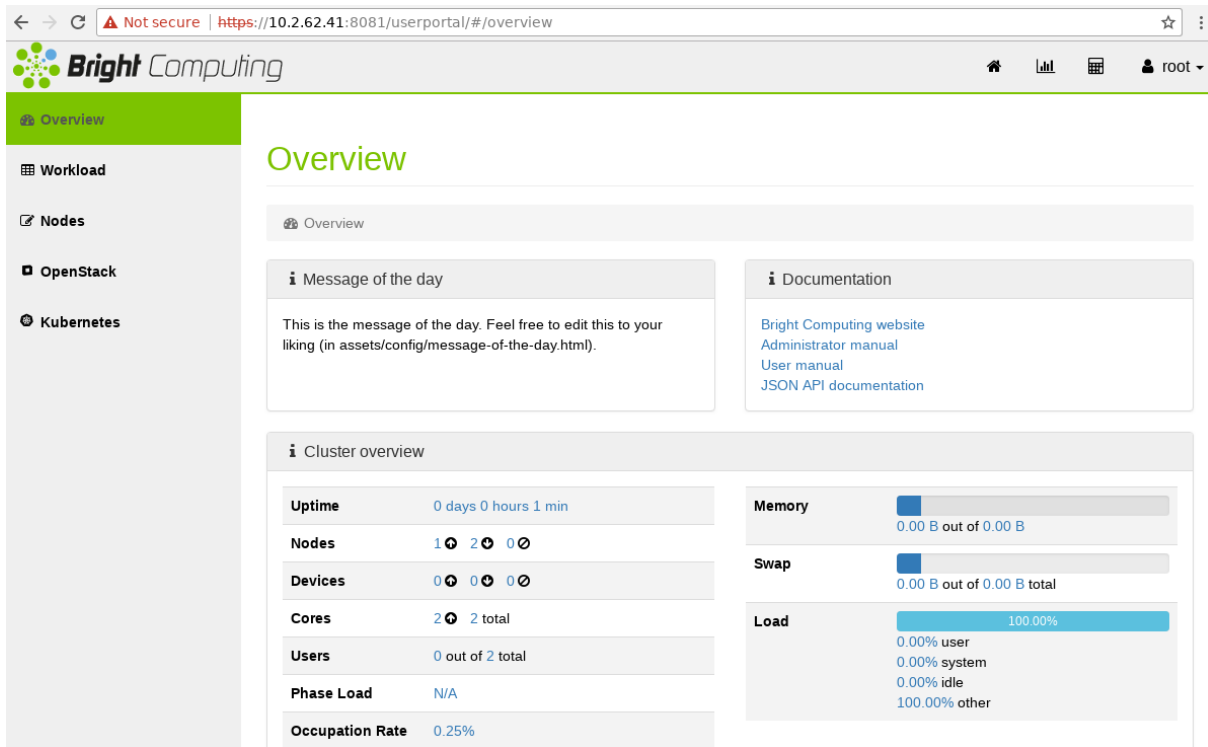




Figure 12.24: User Portal: Overview Page

The following items are displayed on the overview page:

- a Message Of The Day. This can be edited in `/cm/local/apps/cmd/etc/htdocs/userportal/assets/config/message-of-the-day.html`
- links to the documentation for the cluster
- an overview of the cluster state, displaying some cluster parameters. By default, it is refreshed every 10s.

The user portal is designed to serve files only, and will not run executables such as PHP or similar CGI scripts. Monitoring and job accounting charts can be viewed on clicking upon the associated icons,  and , at the top right corner of the user portal page.

User Portal Monitoring Page

The user portal's Monitoring page allows charts of measurables to be plotted in a monitoring window in a very similar manner to how it is done in section 12.3.1.

User Portal Job Accounting Page

The user portal's Accounting and reporting page allows job accounting to be viewed in an accounting panel in a very similar manner to how it is done in section 12.9.4.

12.8 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.

The administrator can also select a job that is currently running, or that has recently run, and get metrics for nodes, memory, CPU, storage, and other resource use for the job. This is known as *job monitoring*, which is, as the term suggests, about job-centric rather than device-centric monitoring. It is covered in this section (section 12.8), and uses *job metrics*.

So far, monitoring as discussed in this chapter has been based on using devices or jobs as the buckets for which resource use values are gathered. Administrators can also gather, for resources consumed by jobs, the resources used by users (or any other aggregation entity) as the buckets for the values. This is typically useful for watching over the resources used by a user (or other aggregation entity) when jobs are run on the cluster. User-centric monitoring—or more generally, aggregation-centric monitoring—for jobs is termed *job accounting* and is covered in section 12.9.

12.8.1 Job Metrics Introduction

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

Notes:

1. OGS for now does not support cgroups.
2. PBS Pro cgroups are enabled by default by `cm-wlm-setup`. Hooks are provided for cgroups with the PBS Pro packages.

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`
- `cpuacct`
- `memory`
- `freezer`

For example: if Slurm jobs are configured to run on the regular nodes via `cm-6/wlm-setup`, then the following cgroup directories structure becomes available for Slurm job metrics collection:

Example

```
[root@node001 ~]# cd /sys/fs/cgroup/
[root@node001 cgroup]# tree -d --charset=C -L 2 | grep blkio
|-- blkio -> blkio,cpuacct,memory,freezer
|-- blkio,cpuacct,memory,freezer
|-- cpuacct -> blkio,cpuacct,memory,freezer
|-- freezer -> blkio,cpuacct,memory,freezer
|-- memory -> blkio,cpuacct,memory,freezer
```


The directory

```
blkio,cpuacct,memory,freezer
```

is created, the appropriate cgroup controllers are mounted, the directory links are created, and this is all carried out automatically by systemd.

Example

```
[root@node001 ~]# grep JoinControllers /etc/systemd/system.conf
JoinControllers=blkio,cpuacct,memory,freezer
[root@node001 ~]#
```

The tables in Appendix G.1.5 list the job metrics that Bright Cluster Manager can monitor and visualize.

If job metrics are set up (section 12.8.3), then:

1. on virtual machines, block device metrics may be unavailable because of virtualization.
2. for now, the metrics are retrieved from cgroups created by the workload manager for each job. When the job is finished the cgroup is removed from the filesystem along with all the collected data. Retrieving the jobs metric data therefore means that CMDaemon must sample the cgroup metrics before the job is finished. If CMDaemon is not running during a time period for any reason, then the metrics for that time period cannot be collected, even if CMDaemon starts later.
3. block device metrics are collected for each block device by default. Thus, if there are N block devices, then there are N collected block device metrics. The monitored block devices can be excluded by configuration as indicated in section 12.8.3.

12.8.2 Job Information Retention

Each job adds a set of metric values to the monitoring data. The longer a job runs, the more data is added to the data. By default, old values are cleaned up from the database in order to limit its size. In Bright Cluster Manager 9.0 there are several advanced configuration directives to control the job data retention, with names and default values as follows:

Advanced Configuration Directive	Default value	Unit
JobInformationDisabled	0	
JobInformationKeepDuration	2419200	s
JobInformationKeepCount	8192	
JobInformationMinimalJobDuration	0	s
JobInformationFlushInterval	600	s

These directives are described in detail in Appendix C, page 727.

12.8.3 Job Metrics Sampling Configuration

Job metrics sampling can be configured to varying degrees. For clusters where hundreds of thousands of jobs are run in a day it often makes little sense to monitor jobs, and it is often helpful to disable the JobSampler and JobMetadataSampler data producers:

Example

```
[bright90]% monitoring setup
[bright90->monitoring->setup]% set jobsampler disabled yes
[bright90->monitoring->setup]% set jobmetadatasampler disabled yes
[bright90->monitoring->setup]% commit
```

An alternative is to use the equivalent CMDaemon directive `JobInformationDisabled`, as explained on page 545.

If however CMDaemon is to keep the monitoring data, then the collection of job metrics is carried out from the cgroups in which a job runs. The administrator can tune some low level metric collection options for the JobSampler data producer in the `jobmetricsettings` submode:

Example

```
[bright90]% monitoring setup
[bright90->monitoring->setup]% use jobsampler
[bright90->monitoring->setup[JobSampler]% jobmetricsettings
[bright90->monitoring->setup[JobSampler]->jobmetricsettings]% show
```

Parameter	Value
CGroup base directory	/sys/fs/cgroup
CGroup search	freezer
Enable Advanced Metrics	no
Exclude Devices	loop,sr
Exclude Metrics	
Include Devices	
Include Metrics	
Keep alive sleep	8w
No GPUs is all GPUs	no
Pickup interval	5s
Pickup priority	50
Pickup times	12
Sampling Type	Both

The configuration parameters are:

Parameter Name	Description
CGroup base directory	Cgroup base directory (default: /sys/fs/cgroup)
CGroup search	Search field for finding the workload manager cgroup (default: freezer)
Enable Advanced Metrics	Enables advanced metrics
Exclude Devices	Block devices for which job metrics will not collect metrics
Include Devices	If the list is not empty then only block device metrics for these devices will be collected, while for other devices the metrics will be skipped

...continues

...continued

Parameter Name	Description
Enable Advanced	Indicates whether advanced job metrics should be enabled (options: true, false)
Exclude Metrics	List of metric names that should not be collected
Include Metrics	List of metric names that should be added to metric collection
Keep alive sleep	Time the cgroup keepalive process sleeps (default: 8 weeks)
No GPUs is all GPUs	Include data from all GPUs even if the job has no GPUs defined (default: no)
Pickup interval	Initially higher pickup interval (default: 5s). By default this settles down to the normal pickup interval (with a default of 120s) after the value of Pickup times has been exceeded.
Pickup priority	Priority of the pickup interval change (default: 50)
Pickup times	Number of times to apply the initially higher pickup interval (default: 12)
Sampling Type	Type of metric sampling (default: Both)

The amount of monitoring data gathered can also be reduced by reducing the `Maximal age` and `Maximal samples` for data producers (section 12.4.1) to smaller, but still non-zero values. A way to do this is described in section 13.8.4.

The Job Metrics Collection Processing Mechanism

`cm-cgroup-job-keepalive`: From Bright Cluster Manager 9.0 onwards, when a WLM job starts, CMDaemon tracks the moment it starts and finishes, and is able to collect metrics for it. As part of this enhanced jobs metrics collection, a keepalive process, `cm-cgroup-job-keepalive`, is run for each job. Each keepalive process is a temporary process, and is added by CMDaemon to the same cgroup that the original job was placed in by the WLM.

The process `cm-cgroup-job-keepalive` itself does no work. It sleeps, and by existing it prevents its cgroup being deleted.

After a job is finished, the workload manager would normally remove the related cgroup. However the existence of the `cm-cgroup-job-keepalive` process prevents the deletion. This allows CMDaemon to collect the very last metrics data for the job from the cgroup when the job finishes. The CMDaemon then stops the `cm-cgroup-job-keepalive` process, and the cgroup is then removed because it is no longer needed.

When CMDaemon starts the `cm-cgroup-job-keepalive` process for a job, it passes the appropriate job ID, and how long it can run, in its command line options. Those values are not used by the `cm-cgroup-job-keepalive` process itself, but they are convenient for seeing what job the process is run-

ning for, and how long the job has run since it was started. For example, for a Slurm job with id 2 the running cgroup keeper process could look like:

Example

```
[root@node001 ~]# ps auxf | tail -n 10 | cut -b18-45 --complement
root      1954  0  Sl   18:12   0:03  _ /cm/local/apps/cmd/sbin/cmd -s -n -P /var/run/cmd.pid
root      2632  0  Ss   18:15   0:00  _ /cm/local/apps/cmd/sbin/cm-cgroup-job-keepalive --job 2 8w
root      2241  0  S    18:13   0:00 /cm/shared/apps/slurm/18.08.4/sbin/slurmd
root      2627  0  Sl   18:15   0:00 slurmstepd: [2.batch]
cmsuppo+ 2631  0  S    18:15   0:00  _ /bin/bash /cm/local/apps/slurm/var/spool/job00002/slurm_script
cmsuppo+ 2642  0  S    18:15   0:00  _ /cm/shared/apps/stresscpu/current/stresscpu2
cmsuppo+ 2643  0  S    18:15   0:00  _ /cm/shared/apps/stresscpu/current/stresscpu2
cmsuppo+ 2644 98  R    18:15   4:46  _ /cm/shared/apps/stresscpu/current/stresscpu2
[root@node001 ~]#
```

(The cut command is just used in the example to cut out the middle bits of the output so that it fits the page format well).

Keep Alive Sleep: By default the cgroup keeper process stops after 8 weeks. This value should be increased if the jobs that are expected to run will take longer than 8 weeks. The value can be set in the Keep Alive Sleep parameter of the job metrics settings. If a job runs for longer than the value of Keep Alive Sleep, then CMDaemon cannot collect the very last metrics (from around the time that the job has finished). However all other metrics will be collected for the job as expected, even if the job running time exceeds the Keep Alive Sleep time.

OOB intervals: When metric collection for a new job has just started, CMDaemon samples more frequently than later on. This more frequent sampling behaviour is defined by the parameter OOB intervals (out of band sampling interval) in the data producer configuration. In the case of job metrics collection this more frequent sampling behaviour is in JobSampler.

By default, the sampling interval returns to the standard Interval value (with a default value of 120s), as defined in the data producer settings, after the value of Pickup times (with a default value of 12) has been exceeded.

The parameter Exclude Metrics can be used to exclude metrics that are currently enabled. For example, if advanced metrics collection is enabled then Exclude Metrics allows either default or advanced metrics to be excluded by name.

12.8.4 Job Monitoring In cmsh

The following commands are associated with monitoring job measurables within jobs submode (cmsh->wlm<[workload manager]>->jobs, section 7.7):

measurables: A list of job-associated measurables can be seen in the jobs submode (cmsh->[<workload manager]>->jobs) using the measurables command with a job ID. For example (much output elided):

Example

```
[bright90->wlm[slurm]->jobs]% measurables 26
blkio.io_service_bytes_total
...
memory.usage
[bright90->wlm[slurm]->jobs]%
```

A list of node-associated measurables can also be seen if the -n option is used (much output elided):

Example

```
[bright90->wlm[slurm]->jobs]% measurables -n 26
...
gpu_power_usage:gpu0
...
memory.usage
```

filter: The filter command provides filtered historic job-related information. It does not provide measurables data. The command and its options can be used to:

- retrieve running, pending, failed or finished jobs information
- select job data using regular expressions to filter by job ID
- list jobs by user name, user group, or workload manager

The `--limit` option can be used to limit the number of results displayed

Example

```
[bright90->wlm[slurm]->jobs]% filter -n mgbench --ended --limit 2
Job ID Job name User Queue Submit time Start time End time Nodes Exit code
-----
26 mgbench alice defq May 14 11:21:41 May 14 11:36:36 May 14 11:50:24 node001 0
27 sleep bob defq May 14 11:23:00 May 14 11:50:24 May 14 12:00:25 node001 0
[bright90->wlm[slurm]->jobs]%
```

The data shown is retrieved from the running workload manager, as well as from the accounting file or database maintained by the workload manager.

Further details on the options to filter can be seen by running `help filter`.

info A handy command to obtain job information is `info`, followed by the job number:

Example

```
[bright90->wlm[slurm]->jobs]% info 26
Parameter Value
-----
Revision
WlmCluster slurm
Job ID 26
Job name mgbench
Persistent no
Queue defq
User alice
Group alice
Account
Parent ID
CGroup slurm/uid_1002/job_26
Nodes node001
Submit time 14/05/2020 11:21:41
Start time 14/05/2020 11:36:36
End time 14/05/2020 11:50:24
Exit code 0
Status COMPLETED
[bright90->wlm[slurm]->jobs]%
```

dumpmonitoringdata The `dumpmonitoringdata` command displays data for measurables in the jobs submode (`cmsh->[<workload manager>->jobs`). It is very similar to the `dumpmonitoringdata` command for measurables in device mode (section 12.6.4). The main difference in the behavior of `dumpmonitoringdata` for these modes is that:

- In the jobs submode it shows monitoring data over a period of time for a specified job ID
- In device mode (`cmsh->device`) it shows monitoring data over a period of time for a specified device.

A less obvious difference is that:

- In the jobs submode the start and end time for the monitoring data for the job does not need to be specified. By default the start and end time of the job is assumed.
- In device mode the start and end time for the monitoring data for the device must be specified

The usage of the `dumpmonitoringdata` command for job measurables is:

```
dumpmonitoringdata [OPTIONS] [<start-time> <end-time>] <measurable> <job ID>
```

Options allow measurables to be retrieved and presented in various ways, including by maximum value, raw or interpolated data, and human-friendly forms. The user can also specify custom periods for the options.

For example, the historical job with job ID 26 might display output for the job-associated measurable `memory.usage` as follows (output elided):

Example

```
[bright90->wlm[slurm]->jobs]% dumpmonitoringdata memory.usage 26
Start: Thu May 14 11:36:36 2020
End:   Thu May 14 11:50:24 2020
Nodes: node001
Timestamp          Value      Info
-----
2020/05/14 11:36:37    708 KiB
2020/05/14 11:36:38    117 MiB
2020/05/14 11:36:39    29.8 MiB
...
2020/05/14 11:50:24    86.6 MiB
[bright90->wlm[slurm]->jobs]%
```

The start and end times are optional, so specifying them is typically unnecessary. If they are not specified, then the data values over the entire period of the job run are displayed.

That job with ID 26, for the example used in this section 12.8.4, happens to be `mgbench`, a GPU benchmarking program. So displaying output for the device-associated measurable, `gpu_power_usage`, during the job run can also be useful:

Example

```
[bright90->wlm[slurm]->jobs]% dumpmonitoringdata gpu_power_usage:gpu0 26
Start: Thu May 14 11:36:36 2020
End:   Thu May 14 11:50:24 2020
Nodes: node001
Timestamp          Value      Info
-----
2020/05/14 11:36:36    no data
```

```
2020/05/14 11:37:03.883    195.58 W
...
2020/05/14 11:50:23.887    22.224 W
2020/05/14 11:50:24        22.2241 W
[bright90->wlm[slurm]->jobs]%
```

The data is shown per node if the job uses several nodes.

Further details of the options to `dumpmonitoringdata` for job metrics can be seen by running `help dumpmonitoringdata` within jobs mode.

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 statistics can be grouped by hour, day, week or a custom interval.

Example

```
[bright90->wlm[slurm]->jobs]% statistics
Pending    Running    Finished    Error        Nodes
-----
0           0           31          0            31
[bright90->wlm[slurm]->jobs]%
```

Further details of the options to `statistics` can be seen by running `help statistics`.

12.9 Job Accounting

In addition to the concept of device-based monitoring (described in most of this chapter), or the concept of job-based metrics (section 12.8 of this chapter), there is also the concept of aggregation-based metrics for resources used during jobs. This last one is typically user-based metrics for resources used during jobs. Aggregation-based metrics for jobs is more conveniently called *job accounting*, and its concept and implementation are described in this section (section 12.9).

Job accounting is thus about the accounting of resources used by jobs. Typically, for jobs, resource usage can be presented per user. For example, if there are jobs in a queue that are being processed, the jobs can be listed:

Example

```
[bright90->wlm[slurm]->jobs]% list | head
Type      Job ID    User   Queue  Running time Status      Nodes
-----
Slurm     1325     tim    defq   1m 2s     COMPLETED node001..node003
Slurm     1326     tim    defq   1m 1s     COMPLETED node001..node003
Slurm     1327     tim    defq   1m 2s     COMPLETED node001..node003
Slurm     1328     tim    defq   32s      RUNNING    node001..node003
Slurm     1329     tim    defq   0s       PENDING
```

The resource usage statistics gathered per user, for example a user `tim`, can then be analyzed and visualized using the job accounting interface of Bright View (section 12.9.4).

12.9.1 Labeled Entities

In job accounting, job metrics during a run are tagged with extra labels, such as the job ID, host name, and the user running the job. The modified job metrics object that is tagged in this way then becomes a job accounting-related object, called a labeled entity.

Administrators interested in using job accounting can simply skip ahead and start reading about the Bright View job accounting interface in 12.9.4, and just explore it directly. Those who would prefer some background on how job accounting is integrated with Bright Cluster Manager and PromQL, can continue reading this section (12.9.1) and the next one (12.9.2).

Dataproducers For Labeled Entities

To view labeled entities in `cmsh`, the path to the `labeledentity` submode is:

```
cmsh->monitoring->labeledentity
```

The `labeledentity` submode allows job accounting-related objects, called labeled entities, to be viewed. The labels are in the form `<key>=<value>`, for example: `hostname="node001"`, or `user="alice"`.

The default, existing labeled entities are created from the built-in `JobSampler` and `JobMetadataSampler` dataproducers when a job is run. Custom samplers, of type `prometheus`, can be used to create further custom labeled entities. A custom sampler dataproducer, for example `customsamplereextras`, can be created from the `monitoring` setup mode of `cmsh` as follows:

Example

```
[bright90]% monitoring setup
[bright90->monitoring->setup]% add prometheus customsamplereextras
[bright90->monitoring->setup*[customsamplereextras*]]%
```

The `customsamplereextras` dataproducer can now have its properties configured and committed as described in section 12.5.4.

PromQL And Labeled Entities

Labeled entities can be used by administrators to help create and debug job-related queries in the Prometheus query language, PromQL. PromQL is a part of the Prometheus monitoring and alerting toolkit (<https://prometheus.io>). Basic PromQL documentation is available at <https://prometheus.io/docs/prometheus/latest/querying/basics/>.

Job IDs And Labeled Entities

Each job ID has a number of labeled entities associated with it. Since the number of labeled entities scales with the number of nodes and jobs, the number of labeled entities can be very large. Therefore, if examining these entities using the CMDaemon front ends such as `cmsh` or Bright View, then filtering or sorting the output is useful. For example, labeled entities associated with `node001`, and with the `JobSampler` data producer, and with job 1329 from the preceding output, could be viewed by filtering the full list of labeled entities as follows (output truncated and ellipsized):

Example

```
[bright90->monitoring->labeledentity]% list|head -2; list|grep 'job_id="1329"' |grep node001 |grep JobSampler
Index  Name (key)
-----
45446  hostname="node001",job="JobSampler",job_id="1329",wlm="slurm"
45447  device="vda",hostname="node001",job="JobSampler",job_id="1329",wlm="slurm"
45448  device="vda",hostname="node001",job="JobSampler",job_id="1329",mode="read",wlm="slurm"...
...
```

Measurables And Labeled Entities

The measurables (metrics) for an entity can be listed with the `measurables (metrics)` command. For a particular entity with a `JobSampler` property and index value of 45447, the command can be run as follows:

```
[bright90->monitoring->labeledentity]% measurables 45447
Type      Name                                Parameter      Class      Producer
-----
Metric    job_blkio_sectors                   Prometheus     JobSampler
Metric    job_blkio_time_seconds              Prometheus     JobSampler
[bright90->monitoring->labeledentity]%
```

In the `labeledentity` mode of `cmsh`, the `measurables` listing command, which lists the measurables for labeled entities, should not be confused with the `measurable` navigation command, which brings the administrator to the `measurable` submode under the main `monitoring` mode.

12.9.2 PromQL Queries

The Default PromQL Queries...

By default there are several predefined PromQL queries already available. The queries can be listed from the query submode:

Example

```
[bright90->monitoring->query]% list
Name (key)                               Start time End time Interval Class
-----
account_job_effective_cpu_seconds_1w      now         0s         accounting
account_job_io_bytes_1w                  now         0s         accounting
account_job_memory_usage_bytes_1w        now         0s         accounting
account_job_running_count_1w             now         0s         accounting
account_job_waiting_seconds_1w           now         0s         accounting
account_job_wall_clock_seconds_1w        now         0s         accounting
account_job_wasted_cpu_seconds_1w        now         0s         accounting
cluster_cpu_usage_percent                 now-1d      now         15m       cluster
cpu_usage_by_cluster                      now-1d      now         1h        kubernetes
cpu_usage_by_deployment                   now-1d      now         1h        kubernetes
cpu_usage_by_namespace                    now-1d      now         1h        kubernetes
fs_usage_by_cluster                       now-1d      now         1h        kubernetes
fs_usage_by_deployment                    now-1d      now         1h        kubernetes
fs_usage_by_namespace                     now-1d      now         1h        kubernetes
job_information_by_account                 now         0s         drilldown/level/0
job_information_by_job_id_for_account_and_
  user_and_job_name                       now         0s         drilldown/level/3
job_information_by_job_id_for_user         now         0s         drilldown/level/1
job_information_by_job_id_for_user_and_job_name now         0s         drilldown/level/2
job_information_by_job_name_for_account    now         0s         drilldown/level/1
job_information_by_job_name_for_account_and_user now         0s         drilldown/level/2
job_information_by_job_name_for_user      now         0s         drilldown/level/1
job_information_by_user                    now         0s         drilldown/level/0
job_information_by_user_for_account        now         0s         drilldown/level/1
job_information_by_user_for_account_and_job_name now         0s         drilldown/level/2
memory_usage_by_cluster                   now-1d      now         1h        kubernetes
memory_usage_by_deployment                 now-1d      now         1h        kubernetes
memory_usage_by_namespace                  now-1d      now         1h        kubernetes
net_usage_by_cluster                       now-1d      now         1h        kubernetes
net_usage_by_deployment                    now-1d      now         1h        kubernetes
net_usage_by_namespace                     now-1d      now         1h        kubernetes
storage_ceph_host_apply_latency           now-1d      now         15m       ceph
storage_ceph_host_commit_latency          now-1d      now         15m       ceph
storage_ceph_host_cpu_usage               now-1d      now         15m       ceph
storage_ceph_osd_bytes_used                now-1d      now         15m       ceph
storage_ceph_osd_op_ratio                  now-1d      now         15m       ceph
users_job_allocated_nodes                  now-1d      now         15m       jobs
users_job_cpu_usage                        now-1d      now         15m       jobs
users_job_effective_cpu_seconds_1w        now         0s         accounting
users_job_io_bytes_1w                     now         0s         accounting
users_job_io_bytes_per_second              now-1d      now         15m       jobs
users_job_memory_bytes                     now-1d      now         15m       jobs
users_job_memory_usage_bytes_1w           now         0s         accounting
users_job_running_count_1w                now         0s         accounting
users_job_waiting                          now-1d      now         15m       jobs
users_job_waiting_seconds_1w              now         0s         accounting
```

users_job_wall_clock_seconds_1w	now	0s	accounting
users_job_wasted_cpu_seconds_1w	now	0s	accounting

The queries can be conceptually divided into their classes, which at the time of writing (November 2019) are: `accounting`, `ceph`, `cluster`, `jobs`, `kubernetes`, along with various drilldown levels which aggregates the queries according to various groups.

By default, queries of all classes are sampled over a period, except for the `accounting` and `drilldown` metrics.

A metric in the `accounting` class query is evaluated (interpolated) from existing values. These existing values are raw samples gathered from 1 week ago, up to the time when the query is evaluated. The `_1w` ending to the name indicates the 1 week period.

...And A Short Description Of Them

The description of each query can be listed with a little `cmsh` and `unix` text utility juggling:

Example

```
[bright90->monitoring->query]% foreach * (get name; get description) | paste - - | expand -t 60
```

This yields the following table:

Table 12.9.2: PromQL Query Descriptions

Name	Description
users_job_wall_clock_seconds_1w	Wall clock time used by users for the past 7 days
users_job_wasted_cpu_seconds_1w	CPU seconds allocated but not used by users for the past 7 days
users_job_effective_cpu_seconds_1w	CPU seconds effectively used by users for the past 7 days
users_job_running_count_1w	Number of jobs running by users during the past 7 days
users_job_waiting_seconds_1w	Total waiting time for users jobs in seconds during the past 7 days
users_job_memory_usage_bytes_1w	Total memory usage by users during the last week in Byte seconds
users_job_io_bytes_1w	Total I/O by users during the last week in Bytes
account_job_effective_cpu_seconds_1w	CPU seconds effectively used by account for the past 7 days
account_job_io_bytes_1w	Total I/O by account during the last week in Bytes
account_job_memory_usage_bytes_1w	Total memory usage by account during the last week in Byte seconds
account_job_running_count_1w	Number of jobs running by account during the past 7 days

...continues

Table 12.9.2: PromQL Query Descriptions...continued

Name	Description
account_job_waiting_seconds_1w	Total waiting time for account jobs in seconds during the past 7 days
account_job_wall_clock_seconds_1w	Wall clock time used by account for the past 7 days
account_job_wasted_cpu_seconds_1w	CPU seconds allocated but not used by account for the past 7 days
users_job_allocated_nodes	Number of nodes allocated by users
users_job_cpu_usage	Effective CPU usage by users
users_job_waiting	Number of jobs currently waiting for every user
users_job_memory_bytes	Current memory consumption for user jobs in Bytes
users_job_io_bytes_per_second	Current I/O for user jobs in B/s
cluster_cpu_usage_percent	CPU usage percentage over all nodes up
storage_ceph_osd_bytes_used	The amount of disk space used across all OSDs
storage_ceph_host_apply_latency	The average apply latency on each host running Ceph OSDs
storage_ceph_host_commit_latency	The average commit latency on each host running Ceph OSDs
storage_ceph_host_cpu_usage	CPU usage on each host running Ceph OSDs
storage_ceph_osd_op_ratio	The number of operations per second on each Ceph OSD
memory_usage_by_deployment	Total memory usage by deployment during the last week in Bytes per second
memory_usage_by_namespace	Total memory usage by namespace during the last week in Bytes per second
fs_usage_by_deployment	Current FS I/O by deployment in Bytes
fs_usage_by_cluster	Current FS I/O by cluster in Bytes
fs_usage_by_namespace	Current FS I/O by namespace in Bytes
net_usage_by_deployment	Network usage by deployment in Bytes per second
net_usage_by_cluster	Network usage by cluster in Bytes per second
net_usage_by_namespace	Network usage by namespace in Bytes per second
cpu_usage_by_deployment	CPU usage by deployment in nr. of cores
cpu_usage_by_cluster	CPU usage by cluster in nr. of cores

...continues

Table 12.9.2: PromQL Query Descriptions...continued

Name	Description
cpu_usage_by_namespace	CPU usage by namespace in nr. of cores
job_information_by_job_id_for_account_and_user_and_job_name	Generic job information drill down query grouped by job_id for a specific account, user and job_name
job_information_by_job_name_for_account_and_user	Generic job information drill down query grouped by job_name for a specific account and user
job_information_by_user_for_account_and_job_name	Generic job information drill down query grouped by user for a specific account and job_name
job_information_by_job_id_for_user_and_job_name	Generic job information drill down query grouped by job_id for a specific user and job_name
job_information_by_user_for_account	Generic job information drill down query grouped by user for a specific account
job_information_by_job_name_for_account	Generic job information drill down query grouped by job_name for a specific account
job_information_by_job_name_for_user	Generic job information drill down query grouped by job_name for a specific user
job_information_by_job_id_for_user	Generic job information drill down query grouped by job_id for a specific user
job_information_by_account	Generic job information drill down query grouped by account
job_information_by_user	Generic job information drill down query grouped by user
jobs_wasted_allocated_gpus	Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by job_id
users_wasted_allocated_gpus	Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by user
wasted_allocated_gpus_for_user	Average % of allocated GPUs wasted for jobs that ran in the specified period, averaged and grouped by job_id, for a particular user
used_gpu_job_name_for_user	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by job names that ran on them in the specified period, for a particular user

...continues

Table 12.9.2: PromQL Query Descriptions...continued

Name	Description
users_used_gpu	Used GPUs, for values of use greater than or equal to 0.1%, averaged and grouped by users using them in the specified period
unused_gpu_job_name_for_user	Unused GPUs, for values of use less than 0.1%, averaged and grouped by job names using them in the specified period, for a particular user
users_unused_gpu	Unused GPUs, for values of use less than 0.1%, averaged and grouped by users using them in the specified period

The listings give an idea of what the query does.

For example, for the `users_job_cpu_usage` utility, the idea is that it shows the CPU usage for jobs for each user.

Modifying The Default PromQL Query Properties

The properties of a particular query can be shown and modified:

Example

```
[bright90->monitoring->query]% use users_job_cpu_usage
[bright90->monitoring->query[users_job_cpu_usage]]% show
```

Parameter	Value
Access	Public
Class	jobs
Description	Effective CPU usage by users
End time	now
Interval	15m
Name	users_job_cpu_usage
Notes	<0 bytes>
PromQL Query	<131 bytes>
Revision	
Start time	now-1d
Unit	CPU

The PromQL query code itself is typically a few lines long, and can also be viewed and modified using `get`, and `set`.

An Example PromQL Query, Properties, And Disassembly

The `users_job_cpu_usage` query is a standard predefined query, and is used as an example here. The query shows the CPU usage by a user around the time the sample was taken. It is sometimes called an “instantaneous” value. However it is not that instantaneous, because its value is calculated by taking samples of the CPU usage over the last 10 minutes of the job run rather than at the query time. The code for the query can be viewed with:

Example

```
[bright90->monitoring->query]% get users_job_cpu_usage promqlquery
sum by(user) (
  irate(job_cpuacct_usage_seconds[10m])
  * on(job_id, hostname) group_right()
  (job_metadata_is_running)
)
```

For those unfamiliar with PromQL, some disassembly of the `users_job_cpu_usage` query is helpful.

Terminology used by PromQL and Bright Cluster Manager, for the pieces used to build the query, is listed in the following table:

PromQL Terminology	Example	Bright Cluster Manager Terminology
Query	<code>users_job_cpu_usage</code>	PromQL query
Instant query	<code>job_cpuacct_usage_seconds</code>	Metric (from JobSampler dataproducer, belonging to the Prometheus class)
Range vector	<code>job_cpuacct_usage_seconds[10m]</code>	Metric samples over a time span

As was mentioned before: job account metrics, unlike traditional metrics, are not directly associated with the device-related objects. For such metrics, the monitoring data command `dumpmonitoringdata` is therefore not accessed in `cmsh` from device mode or category mode.

Instead, the Prometheus metric `job_cpuacct_usage_seconds`, for example, is accessed via the `labeledentities` mode.

The properties and interpolated values at a particular instant of time for the metric can be accessed via an `instantquery` such as (some output excised for clarity):

Example

```
[bright90->monitoring->labeledentity]% instantquery job_cpuacct_usage_seconds
__name__          hostname job      job_id uid  user  wlm  Timestamp Value
-----
job_cpuacct_usage_seconds node001 JobSampler 623   1002 tony  slurm 15:52:15 129
job_cpuacct_usage_seconds node002 JobSampler 624   1002 tony  slurm 15:52:15 71
...
```

The properties and interpolated values of the metric over a range of time can be accessed via a `rangequery` such as (some output excised for clarity):

Example

```
[bright90->monitoring->labeledentity]% rangequery --start now-2h --end now job_cpuacct_usage_seconds
__name__          hostname job      job_id uid  user  wlm  Timestamp Value
-----
job_cpuacct_usage_seconds node001 JobSampler 623   1002 tony  slurm 13:52:15 82
job_cpuacct_usage_seconds node001 JobSampler 623   1002 tony  slurm 14:52:15 97
job_cpuacct_usage_seconds node001 JobSampler 623   1002 tony  slurm 15:52:15 129
job_cpuacct_usage_seconds node002 JobSampler 624   1002 tony  slurm 13:52:15 46
job_cpuacct_usage_seconds node002 JobSampler 624   1002 tony  slurm 14:52:15 23
...
```

The association of the `instantquery` and `rangequery` output with job accounts is because its dataproducer is JobSampler.

Further options for the `instantquery` and `rangequery` commands can be found in their help texts within `cmsh`.

Aside: Getting Raw Values For A Prometheus Class Metric

The PromQL language is aimed at providing an overall view of jobs and resource usage. The actual individual raw values that Prometheus metrics are built on—the entries in the Time Series Database (TSDB)—are not regarded as being important for the end user. The emphasis in PromQL is on seeing the values as seen by statistical reworking.

This section, which is about the raw TSDB values, is thus provided as background information for administrators who would anyway like to see what the raw values look like.

Raw values of the metric for a job ID can be accessed by using the index of the labeled identity that is associated with that job ID. For example, job ID 624 can have its index found with some grepping (some output elided or excised for clarity):

Example

```
[bright90->monitoring->labeledentity]% list|head -2; list|grep ' hostname='| grep 'job_id="624"'
Index Name (key) Introduction Last used
-----
4060 hostname="node001",job="JobSampler",job_id="624", ... 13:30:03 16:00:03
[bright90->monitoring->labeledentity]%
```

The index for the job ID 624 is 4060. The job ID can be used by the `dumpmonitoringdata` command to show the series raw values along with their time stamps:

Example

```
[bright90->monitoring->labeledentity]% dumpmonitoringdata -24h now job_cpuacct_usage_seconds 4060
Timestamp Value Info
-----
2019/08/01 16:08:03.255 10s
2019/08/01 16:10:03.255 2m 10s
2019/08/01 16:12:03.255 4m 9s
2019/08/01 16:14:03.255 6m 9s
2019/08/01 16:16:03.255 8m 8s
2019/08/01 16:18:03.255 10m 8s
2019/08/01 16:20:03.255 12m 7s
2019/08/01 16:22:03.255 14m 6s
2019/08/01 16:24:03.255 16m 6s
2019/08/01 16:26:03.255 18m 5s
2019/08/01 16:28:03.255 20m 5s
2019/08/01 16:30:03.255 no data
[bright90->monitoring->labeledentity]%
```

These raw values are the values that are used for interpolation during PromQL queries.

The label names for job samples can be seen using the index:

```
[bright90->monitoring->labeledentity]% show 4060
Parameter Value
-----
Index 4060
Introduction Thu, 01 Aug 2019 16:08:03 CEST
Last used Thu, 01 Aug 2019 16:30:03 CEST
Name hostname="node001",job="JobSampler",job_id="624",uid="1002",user="tony",wlm="slurm"
Permanent no
Revision
```

...An Example PromQL Query, Properties, And Disassembly (Continued)

Getting back from the aside about raw values, and continuing on with the example PromQL query from the start of this section (page 563), the query code for the query `users_job_cpu_usage` was:

```
sum by(user) (
  irate(job_cpuacct_usage_seconds[10m])
    * on(job_id, hostname) group_right()
    (job_metadata_is_running)
)
```

With the necessary background explanations having been carried out, the disassembly of this query can now be done:

The core of the query is built around the job sampler metric `job_cpuacct_usage_seconds`.

The `irate` measurement in this case calculates the rate of change based on the last two most recent values in the Prometheus range vector.

The Prometheus range vector is formed from the Prometheus instant query by using the square brackets with a time value enclosed (`job_cpuacct_usage_seconds[10m]`). The Prometheus instant query is, as the terminology table earlier pointed out, the Prometheus version of the job sampler metric.

Getting back to the range vector, a range vector in general is a series of values formed from the corresponding Prometheus instant query. With the instant query being `job_cpuacct_usage_seconds` here, the range vector is formed over a span of 10 minutes.

After the `irate` function has taken the average, the resultant is what in PromQL is called an instant vector value. This consists of data in the form {CPU seconds consumed during period, timestamp associated with time period sample}. The instant vector value is then joined against each vector element of the pair `job_id, hostname` to generate the labeled identifier for the job running on the node. The `group_right` of the result uses the `job_id` and `hostname` as the leading labels in the label identifiers. The `job_metadata_is_running` function means that values are generated only while `job_metadata` is running. The `sum by(user)` function means that sum of the result is aggregated per user.

Visualisation based on the result is most easily carried out by plotting job CPU usage for each user against time in the period specified, which can be done in a more user-friendly way with Bright View.

The `users_job_wall_clock_seconds_1w` query, is similar, and can be used to plot wall clock seconds consumed by a user over the last week:

Example

```
[bright90->monitoring->query]% get users_job_wall_clock_seconds_1w promqlquery
sum by(user) (
  max_over_time(job_metadata_running_seconds[1w])
    * on(job_id) group_right()
  max_over_time(job_metadata_num_cpus[1w])
)
[bright90->monitoring->query]%
```

Predefined queries can be executed in the `labeledentity` mode with the `-q` option:

```
[bright90->monitoring->labeledentity]% instantquery -q users_job_wall_clock_seconds_1w
user      Timestamp                Value
-----
alice     Mon Jun 24 10:19:34 2019  28.787
charline  Mon Jun 24 10:19:34 2019  102.83
eve       Mon Jun 24 10:19:34 2019  58.574
frank     Mon Jun 24 10:19:34 2019  59.575
```

If Bright View is used instead of `cmsh`, then Prometheus queries can be accessed via the clickpath:

Monitoring→PromQL Queries

12.9.3 Parameterized PromQL Queries

It is also possible to create parameterized queries using the `<key>=<value>` labels in the labeled entities.

This is handy for running the same query with different parameters.

For example, an existing unparameterized query

```
users_job_wall_clock_seconds_1w
```

can be used as a starting point:

```
[bright90->monitoring->query]% get users_job_wall_clock_seconds_1w promqlquery
sum by(user) (
  max_over_time(job_metadata_running_seconds[1w])
  * on(job_id) group_right()
  max_over_time(job_metadata_num_cpus[1w])
)
```

The original query can be cloned over to a new, soon-to-be-parameterized, query called

```
users_job_wall_clock_per_account_seconds_1w
```

for convenient development:

```
[bright90->monitoring->query]% clone users_job_wall_clock_seconds_1w promqlquery users_job_wal\
l_clock_per_account_seconds_1w promqlquery
```

One or more `#{parameter}` fields can then be added to the query. All fields in `users_job_wall_clock_per_account_seconds_1w` can be replaced verbatim. So any part of the query can be made into a parameter.

```
[bright90->monitoring->query]% get users_job_wall_clock_per_account_seconds_1w promqlquery
sum by(user) (
  max_over_time(job_metadata_running_seconds{account="#{account}"}[1w])
  * on(job_id) group_right()
  max_over_time(job_metadata_num_cpus{account="#{account}"}[1w])
)
```

The new query is then committed and can be run by the administrator on demand.

The original query sums over all accounts:


```
[bright90->monitoring->labeledentity]% instantquery -q users_job_wall_clock_seconds_1w
user      Timestamp                Value
-----
alice     Mon Jun 24 10:25:20 2019  28.787
bob       Mon Jun 24 10:25:20 2019  26.787
charline  Mon Jun 24 10:25:20 2019  102.83
eve       Mon Jun 24 10:25:20 2019  58.574
frank     Mon Jun 24 10:25:20 2019  85.362
```

The parameterized query lets the administrator run the same query for specific accounts. For example, if there are slurm accounts for physics `phys` and mathematics `math`, then here `account=phys` and `account=math`, are the `<key>=<value>` format options. The query can then be run with the `-p|--parameter` option as follows:

```
[bright90->monitoring->labeledentity]% instantquery -q users_job_wall_clock_per_account_second\
s_1w -p account=phys
user      Timestamp                Value
-----
alice     Mon Jun 24 10:25:22 2019  28.787
charline  Mon Jun 24 10:25:22 2019  29.787
frank     Mon Jun 24 10:25:22 2019  30.788
[bright90->monitoring->labeledentity]% instantquery -q users_job_wall_clock_per_account_second\
```

```
s_1w -p account=math
user      Timestamp                Value
-----
bob       Mon Jun 24 10:25:37 2019 26.787
charline  Mon Jun 24 10:25:37 2019 73.049
eve       Mon Jun 24 10:25:37 2019 30.787
```

12.9.4 Job Accounting In Bright View

Job accounting can be viewed within Bright View's accounting mode by clicking on the calculator icon  at the top right hand corner of the Bright View standard display (figure 12.5). An accounting panel is then displayed (figure 12.25), within a dashboard:

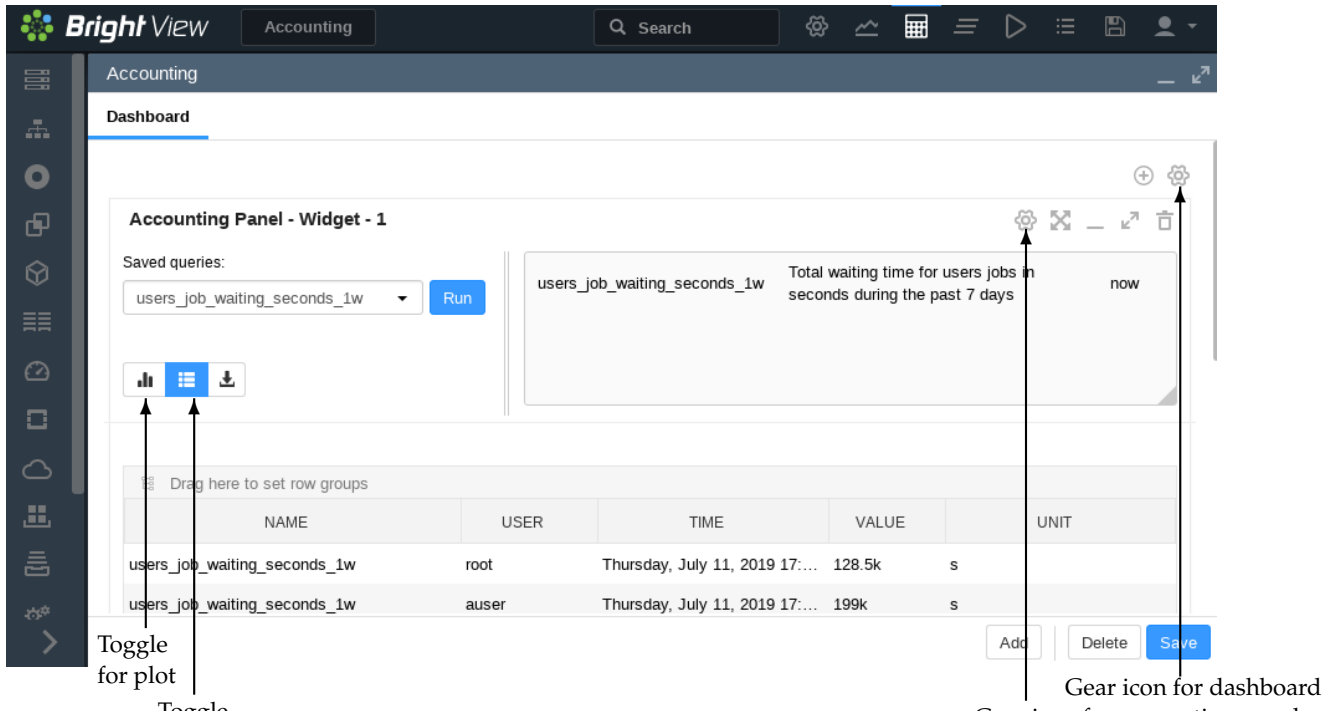


Figure 12.25: Job Accounting: Panel

Management And Use Of The Accounting Panel

A new panel can be added by clicking on the \oplus button in the dashboard panel.

By clicking on the gear icon in the dashboard panel, the accounting panel layout can be changed to allow other arrangements of the panels, or to allow more panels, with other queries.

An accounting panel itself allows a previously-created query to be selected and executed. A PromQL query can be executed with default parameters by just clicking on the Run button.

The box to the right of the Run button shows the query name, and its description.

How to run and view PromQL queries in the Bright View accounting panel is described in this section (12.9.4). The query specification in Bright View is described in more detail in section 12.9.5.

PromQL queries input in the accounting panel: Predefined PromQL queries (page 559) can be viewed via the dropdown list that appears under Saved queries. By default the panel has the query `account_job_effective_cpu_seconds_1w` selected.

- In the default Basic mode of the panel, multiple queries of the same class can be ticked per panel (figure 12.26). To tick a new class of query in the same panel, all the queries that do not match the

new class must first be unticked. In the panel of figure 12.26 the class is accounting.

Dashboard

Accounting Panel - Widget - 1

Saved queries:

accounting

- account_job_effective_cpu...
CPU seconds effectively used by account for the past 7 days
- account_job_io_bytes_1w
Total I/O by account during the last week in Bytes
- account_job_memory_usa...
Total memory usage by account during the last week in Byte...
- account_job_power_usag...
Power usage by account during the last week in Joule (Watt...
- account_job_running_cou...
Number of jobs running by account during the past 7 days
- account_job_waiting_seco...
Total waiting time for account jobs in seconds during the past...

Run

TIME	VALUE	UNIT
Tuesday, September 10, 2019 16...	102.8k	CPU*s

Accounting Panel - Widget - 1

- account_job_effective_cpu_seconds_1w: CPU seconds effectively used by account for the past 7 days now
- account_job_memory_usage_bytes_1w: Total memory usage by account during the last week in Byte seconds now
- account_job_running_count_1w: Number of jobs running by account during the past 7 days now

Add Delete Save

Figure 12.26: Job Accounting: Saved Queries

PromQL queries in Basic mode can only be run as instant queries. The results of such a run are then displayed in that single panel.

- In Advanced mode (section 12.9.5) only one query can be selected and run per panel if an instant query is run.

However, Advanced mode does allow PromQL range queries to be run, in addition to instant queries.

PromQL queries run results display in the accounting panel: Run results can be displayed using the row toggle or the plot toggle.

- The row toggle toggles displaying the result as rows of data per user, cluster, account, or other aggregate (figure 12.25)
- The plot toggle toggles displaying the result as a pie chart, or as a standard x - y plot:
 - The pie chart, or doughnut chart, is available for a PromQL Instant mode query (figure 12.27). The chart shows how much of the resource is used at that instant per user, or per cluster, or per account, or per other aggregate. How much of the resource is used can be displayed upon the chart in figures, either as the amount itself, or as a percentage of the total aggregate amount. The chart can display aggregates using a maximum of 10 slices, by default. If needed, the right amount of the smallest extra slices are grouped together as one slice, others, so that the maximum number is not exceeded.

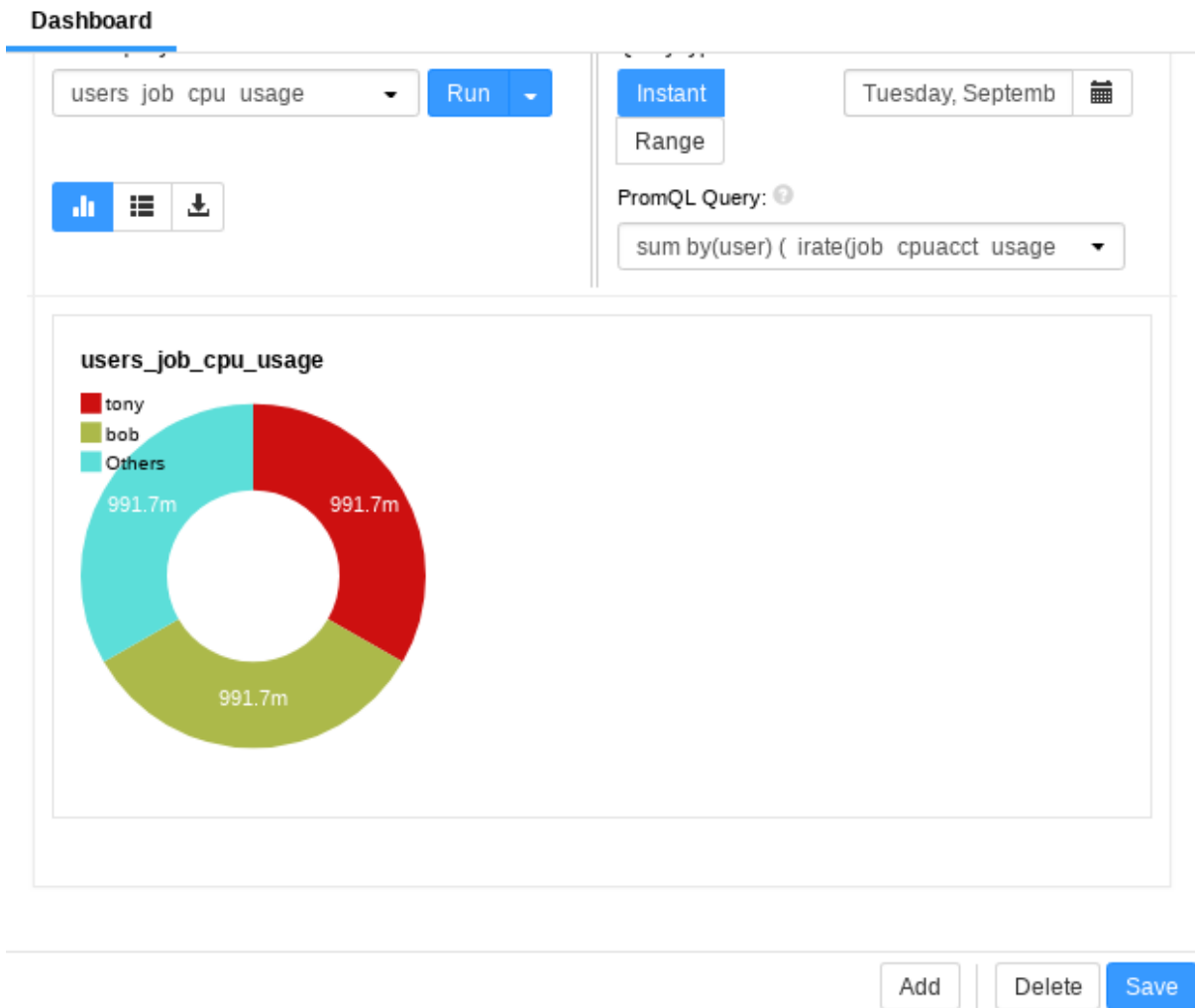


Figure 12.27: Job Accounting: Doughnut Display At An Instant

- The x - y plot is available for a PromQL Range mode query (figure 12.28). The plot shows resource use versus time. The plot can be per user, cluster, account, or other aggregate. A particular user or other aggregate can be highlighted on the plot by placing the mouse pointer over the appropriate legend color icon. The time scale can be zoomed in or out with the mouse scrollbar.

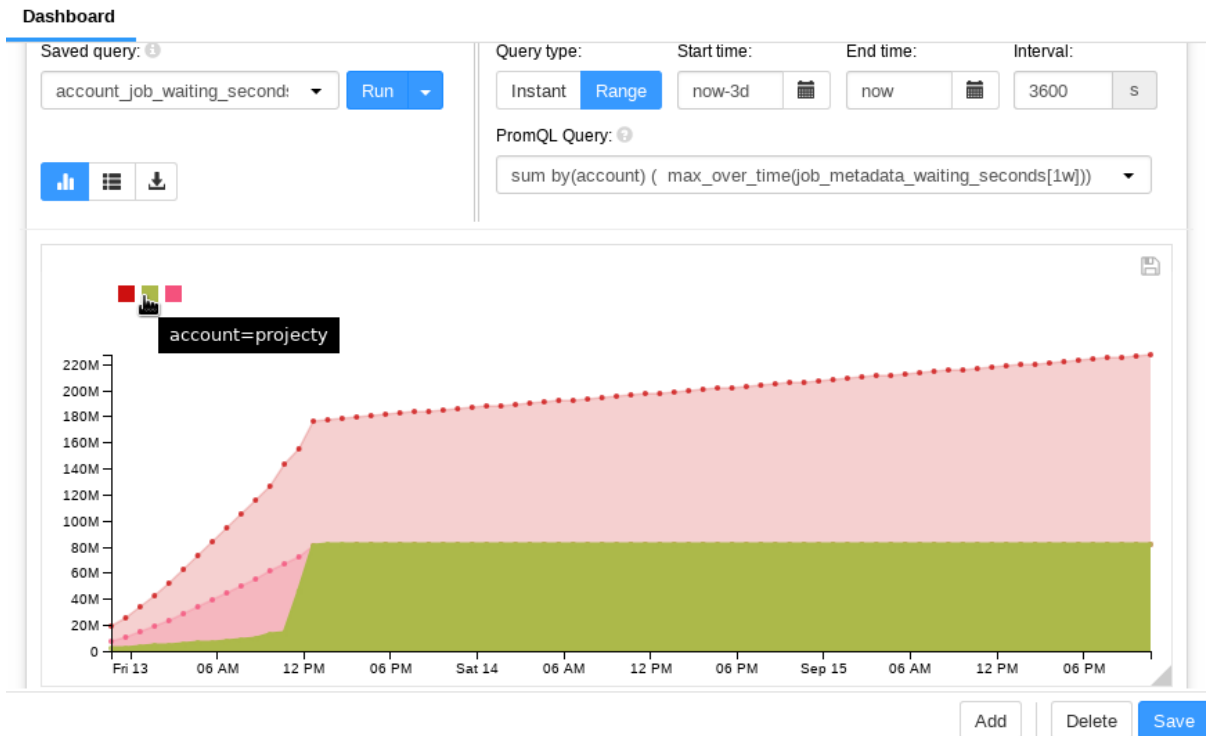


Figure 12.28: Job Accounting: Resource Versus Time Display

How Bright View is used to take PromQL query inputs, and present PromQL query results, has been covered in this section without going into much detail on PromQL specification. PromQL specification is covered in more detail in the next section.

12.9.5 PromQL Query Modes And Specification In Bright View

Clicking on the gear icon for the Accounting Panel (figure 12.25) allows the Accounting widget mode to be toggled to Advanced. Options that can then be modified include:

- the Query Type: This can be either Instant or Range.

An Instant query type: is executed at a particular instant. It outputs a single number for each entity (usually a user). The number is the interpolated data value for the PromQL query at that time. Each data value obtained by the PromQL query is typically an aggregate of the data values over a preceding time period, where the preceding time period is defined in the PromQL query.

The single number obtained by the query for each entity is typically used to summarize a value at that instant for that entity.

An example of the Instant query type is `users_job_waiting_seconds_1w`. This provides the total waiting time, in seconds, for each user, for the jobs the users ran during the past week. The waiting time is calculated at the instant specified for the query.

If the Time parameter for the query is kept as the default value of `now`, then the query provides the latest value for each user.

If the Time parameter is set to a time in the past, then the query picks up the value at that time in the past, or provides an interpolated value for that time in the past, for each user.

The display can be presented in rows of user, value pairs. As a plot, the display can be presented as a pie chart, which each slice being proportional to the value for the particular user compared

with the total value for all the users. Instead of a pie chart, the visualization can instead be a ring (or doughnut) chart, which is really just a pie chart with a hole in the centre.

PromQL allows a variety of time specifications. The interface validates whatever the interface user types in, and there is a calendar widget that allows an absolute time to be specified. Some useful time specifications are:

Example

Time specification	What time is meant
now	at the time it is run
now-30m	30 minutes ago
now-1h	one hour ago
now-1h/h	an hour ago, starting at the start of that hour
now-2d	48 hours ago
now-0d/d	today's midnight (the most recently-passed midnight)*
now-1d/d	yesterday's midnight**
now-2d/d	day before yesterday's midnight
Thursday, July 11, 2019 17:00:00	an absolute time (as set by the calendar widget)

*The meaning can be understood by looking at how /d operates. It rounds off the day by truncation, which in the case of now-0d/d means the midnight prior to now.

**Similarly, yesterday's midnight is the midnight immediately prior to 24 hours ago.

The time units for PromQL are (<https://prometheus.io/docs/prometheus/latest/querying/basics/#range-vector-selectors>):

- s - seconds
- m - minutes
- h - hour
- d - day
- w - week
- M - month (31 days). M is not a PromQL unit, so it cannot be used inside a query. But it is a handy alias in Bright Cluster Manager for an invariant time of 31 days.
- y - year

The - operator in Prometheus is an offset operator, used only after now in the time specification field. The /<time unit> syntax implies a start at that unit of time.

Thus if, for example, now-1d/d is set as the end time, then when the query runs, it picks up the values at "yesterday's midnight." For each user, the value is the number of seconds the job of the user was in a wait state for the week prior to yesterday's midnight. It rounds off the day by truncation, which in the case of now-1d/d means the midnight prior to 24 hours ago. Varying the user means that the number of seconds varies accordingly.

The results of a run can be

- displayed in rows as values in Bright View itself (figure 12.29)
- downloaded as a CSV file
- downloaded as an Excel spreadsheet.

A Range query type: This query is executed over a time period. It fetches a value for each interval within that time range, and then plots these values.

An example of the Range query type is `users_job_memory_bytes`. This fetches the instantaneous memory usage for jobs over intervals, grouped per user during that interval (figure 12.29). The query submitter can set the Start Time and End Time parameters. However, very long ranges can be computationally expensive, and the graph resolution may not be ideal when zoomed in.

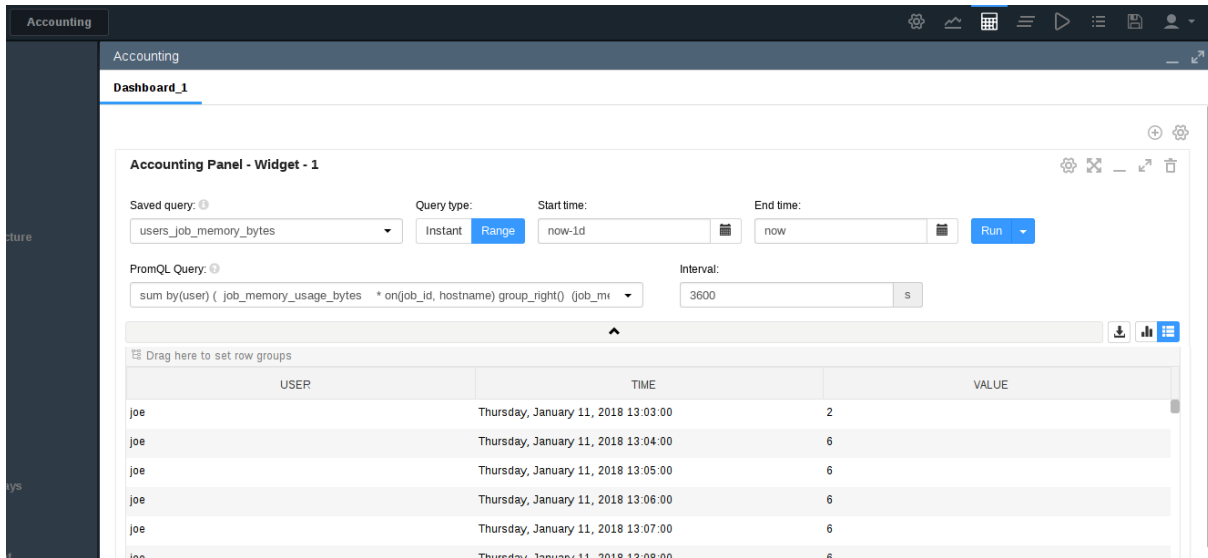


Figure 12.29: Job Accounting: User Values Over A Range

- the PromQL Query: Within advanced mode, the PromQL query itself can be edited.
- the Interval: When editing a Range query type via the Saved query option, the interval for a range query run can be set. Lower numbers lead to smaller intervals, hence higher resolution and higher execution times. A value of 0 means that a reasonable default interval is attempted. Choosing an interval that is less than the sampling time of the metric is a bit pointless, and tends to lead to data values that display non-smooth behaviour.

12.9.6 Access Control For Workload Accounting And Reporting

The ability to view jobs is controlled by four tokens defined in a user's profile:

1. `GET_JOB_TOKEN`: Allows all running jobs to be seen.
2. `GET_OWN_JOB_TOKEN`: Allows owned running jobs to be seen.
3. `GET_JOBINFO_TOKEN`: Allows all cached historic and running jobs to be seen.
4. `GET_OWN_JOBINFO_TOKEN`: Allows owned cached historic and running jobs to be seen.

To retrieve monitoring data for the job, the token `PLOT_TOKEN` must also be defined for the profile.

A job is always owned by the user that runs it. Ownership of a job can also be shared with other users by defining *project managers*. This establishes a 2-level hierarchy, with project managers above the *subordinates*, who are users that are assigned to the project manager. One or more *accounts*, can be assigned to project managers.

Defining Project Managers Using Internal User Management

Any user can be turned into a project manager. If the Bright Cluster Manager LDAP server is being used, then the project manager can be configured via `cmsh`.

Example

alice can be made the project manager of bob and charlie. This allows her access to the job data of her subordinates:

```
[bright90->user]% projectmanager alice
[bright90->user*[alice*]->projectmanager*]% set users bob charlie
[bright90->user*[alice*]->projectmanager*]% commit
```

Example

albert can be made the manager of the physics account. This gives him access to all jobs running under that account:

```
[bright90->user]% projectmanager albert
[bright90->user[albert*]->projectmanager*]% set accounts physics
[bright90->user[albert*]->projectmanager*]% commit
```

Both mechanisms, users and accounts, can be combined to provide access control.

Example

To limit access to jobs that are running under the physics account to a specific set of users:

```
[bright90->user]% projectmanager albert
[bright90->user*[albert*]->projectmanager*]% set accounts physics
[bright90->user*[albert*]->projectmanager*]% set users niels richard
[bright90->user*[albert*]->projectmanager*]% set operator and
[bright90->user*[albert*]->projectmanager*]% commit
```

Defining Project Managers Using External User Management

If an external user management server is used instead of the Bright Cluster Manager LDAP server, then project managers cannot be defined in `cmsh`. Instead, a script has to be written that provides the definitions for project managers in the form of a JSON object.

The full path of the script, for example `/path/to/the/script`, has to be set as a value to the `ProjectManagerScript` parameter in `cmd.conf`. This is done by adding it to the `AdvancedConfig` directive (page 720):

Example

```
AdvancedConfig = { "ProjectManagerScript:/path/to/the/script" }
```

The directive becomes active after restarting `CMDaemon`.

An example of a project manager script can be found at `/cm/local/apps/cmd/scripts/cm-project-managers.py`. It gives users access to each other's jobs if they share at least one group.

The easiest way to use the script is use a mapping to inform `CMDaemon` which of the other user's jobs each user has access to.

Example

So if frank can access data belonging to bob and dennis, while bob can access data belonging to dennis, while dennis can only access his own data, then the project manager configurations can be set up as:

```
{
  "frank": ["bob", "dennis"],
  "bob": ["dennis"],
  "dennis": []
}
```


Account access control can also be included in the output of the script, by setting values for the users, accounts, and the boolean operator (and, or) options:

Example

```
{
  "alice": {
    "users": ["charline", "eve"],
    "accounts": ["math", "chem"],
    "and": True
  },
  "eve": {
    "accounts": ["chem"]
  }
}
```

After restarting CMDaemon, it automatically runs this script when committing a change for a device or data producer. It is also possible to manually trigger the script to be run on the active head, by executing:

```
[root@bright90 ~]# echo "PROJECT.MANAGERS.UPDATE" > /var/spool/cmd/eventbucket
```

The script must not take more than a few seconds to process.

Workaround for project manager script that takes too long: If the script takes longer to run, then it must be run outside of CMDaemon, and its output should be saved as a file. If the output file is located at `/path/to/the/file`, then its path can be set as an input to the `ProjectManagerFile` parameter in `cmd.conf`. This is done by adding it to the `AdvancedConfig` directive (page 720):

Example

```
AdvancedConfig = { "ProjectManagerFile:/path/to/the/file" }
```

The project manager definitions become active after CMDaemon is restarted.

12.9.7 Drilldown Queries For Workload Accounting And Reporting

Metrics can be aggregated in various ways. Common ways are by:

- **device:** A typical hardware device is a node. Each node can then have its metrics, which are CPU usage, memory, storage, and other resource use, displayed over time. Device metrics are largely covered in this chapter in the sections up to and including section 12.7.
- **job:** With this aggregation, each job that is run by a workload manager can have its metrics, which are CPU usage, memory, storage and other resource use, displayed over time. Job metrics are covered in section 12.8 of this chapter.

Other ways of metrics aggregation are part of workload accounting. With workload accounting, a workload manager runs jobs, and a job metric can be aggregated and filtered by:

- user
- job (job ID)
- account
- job name

The aggregation types can be applied singly. However, they can also be applied at the same time, like filters. For example:

- each user could have a particular job metric aggregated
or
- a particular user could have a particular job metric
or
- a particular user could have a particular job metric aggregated for a particular job ID only
or
- a particular user could have a particular job metric aggregated for that particular job ID only for a particular account only

A cluster administrator that uses several filters to get to the “bottom” of how resources are being used, functions in a manner reminiscent of someone drilling to the bottom to find something. This type of filtering is therefore called *drilldown*. Each filter corresponds to a *level* of drilldown.

Drilldown is a bit like how in `cmsh` the use of the `filter` command within jobs mode can narrow down what is displayed:

Example

```
[bright90->wlm[slurm]->jobs]% filter -n iozone -u edgar -a projecty
Job ID  Job name  User   Queue  Submit time  Start time  End time  Nodes  Exit code
-----  -
15      iozone    edgar  defq   14:22:05     14:50:22   15:00:24  node001  0
19      iozone    edgar  defq   14:26:07     15:10:25   15:20:19  node001  0
25      iozone    edgar  defq   14:34:57     15:30:20   15:40:39  node001  0
36      iozone    edgar  defq   14:43:54     16:10:15   16:19:57  node001  0
41      iozone    edgar  defq   14:47:24     16:19:57   16:29:42  node001  0
```

except that in `cmsh` the value of the job metric is not specified or shown by the `filter` command.

Drilldown is also rather similar in concept to how pivot tables are used in Excel spreadsheets. Pivot tables are particular selections (like the filtered choices in drilldown). The selections are applied to a great deal of raw data. A function (like the metric in drilldown) is applied to the selection, to present the information more clearly to the end viewer.

In contrast with the filter output of `cmsh`, in Bright View the job metric value is made visible in a table or in graphs over the period.

12.10 Cloud Job Tagging

Cloud job tagging is about the ability for cloud job instances to have their associated cloud resources *tagged*. This is only possible for AWS at the time of writing (February 2020). Enabling cloud job tagging via Bright Cluster Manager was introduced in version 9.0.

Tags are key=value pairs for AWS resources, and can be *applied* to resources. Typically, tags that are applied are set by the user via the Tag Editor of the Amazon Management Console, and up to 50 tags can be applied per resource.

Cloud job tagging should not be confused with the tagging of job metrics for job accounting (section 12.9.1). AWS cloud resource tagging is only active and handled within AWS.

Cloud job tags allow the time span between tag creation and removal to be associated with a particular workload on the node.

In `cmsh`, for a cloud node, cloud job tagging can be enabled within `ccloud` mode by setting the `ccloudjobtagging` parameter for the `EC2Provider` entity to `yes`

Example

```
cmsh -c 'cloud; use amazon; set cloudjobtagging yes; commit'
```

If it is set to `yes`, then every job running on a cloud node using that specific provider is tagged according to the applied tags.

A subset of the tags for cloud jobs are *cost allocation tags*. Cloud job cost allocation tags allow AWS costs to be tracked for jobs. A cost allocation tag can be:

- an AWS generated tag: defined, created, and applied by AWS
- a user-defined tag: defined, created, and applied by the user

By default, Bright Cluster Manager provides the following tag names when the cloud job tagging feature is enabled:

- BCM_JOB_ID
- BCM_JOB_ACCOUNT
- BCM_JOB_USER
- BCM_JOB_NAME

When CMDaemon sees that a job has started, the resources of that job are then tagged with the job ID, the job account, the job user, and the job name. When CMDaemon detects that the job has stopped, it removes the tags.

The AWS Cost Explorer can be used to view the AWS costs for a billing period according to tags.

Further information on tagging can be found at:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html.

Further information on using the Cost Explorer with cost allocation tags can be found at:

<https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html>

12.11 Event Viewer

Monitoring in Bright Cluster Manager is normally taken by developers to mean how sampling with data producers is handled. However, cluster administrators using this manual typically consider watching and handling events in Bright Cluster Manager to also be a part of a more general concept of monitoring. This manual is aimed at cluster administrators, and therefore this section on event viewing and handling is placed in the current Cluster Monitoring chapter.

Bright Cluster Manager events can be handled and viewed in several ways.

12.11.1 Viewing Events In Bright View

In Bright View, events can be viewed by clicking on the Events icon of figure 12.5. This opens up a window with a sortable set of columns listing the events in the events log, and with by default with the most recent events showing up first.

12.11.2 Viewing Events In `cmsh`

The `events` command is a global `cmsh` command. It allows events to be viewed at several severity levels (section 12.2.7), and allows old events to be displayed. The usage and synopsis of the `events` command is:

```
Usage:    events
          events on [broadcast|private]
          events off [broadcast|private]
          events level <level>
          events clear
          events details <id> [<id>]
```

```
events <number> [level]
events follow
```

Arguments:

level info,notice,warning,error,alert

Running the command without any option shows event settings, and displays any event messages that have not been displayed yet in the session:

Example

```
[bright90->device]% events
Private events:  off
Broadcast events: on
Level:          notice
custom .....[  RESET  ] node001
```

Running the command with options allows the viewing and setting of events as follows:

- on [broadcast|private]: event messages are displayed as they happen in a session, with cmsh prompts showing in between messages:
 - If only on is set, then all event messages are displayed as they happen:
 - * either to all open cmsh sessions, and also in Bright View event viewer panes, if the event or its trigger has the “broadcast” property.
 - * or only in the cmsh session that is running the command, if the event or its trigger has the “private” property.
 - If the further option broadcast is set, then the event message is displayed as it happens in all open cmsh sessions, and also in all Bright View event viewer panes, if the event or its trigger has the “broadcast” property.
 - If the further option private is set, then the event message is displayed as it happens only in the cmsh session that ran the command, if the event or its trigger has the “private” property.
- off [broadcast|private]: disallows viewing of event messages as they happen in a session. Event messages that have not been displayed due to being forbidden with these options, are displayed when the events command is run without any options in the same session.
 - If only off is set, then no event message is displayed as it happens in the session. This is regardless of the “broadcast” or “private” property of the event or its trigger.
 - If the further option broadcast is set, then the event message is not displayed as it happens, if the event or its trigger has the “broadcast” property.
 - If the further option private is set, then the event message is not displayed as it happens, if the event or its trigger has the “private” property.
- level <info|notice|warning|error|alert>: sets a level. Messages are then displayed for this and higher levels.
- clear: clears the local cmsh event message cache. The cache indexes some of the events.
- details <id>: shows details for a specific event with the index value of <id>, which is a number that refers to an event.
- <number> [info|notice|warning|error|alert]: shows a specified <number> of past lines of events. If an optional level (info, notice,...) is also specified, then only that level and higher (more urgent) levels are displayed.

- `follow`: follows event messages in a `cmsh` session, similar to `tail -f /var/log/messages`. This is useful, for example, in tracking a series of events in a session without having the `cmsh` prompt showing. The output can also be filtered with the standard unix text utilities, for example: `events follow | grep node001`

A common example of events that send private messages as they happen are events triggered by the `updateprovisioners` command, which has the “private” property. The following example illustrates how setting the event viewing option to `private` controls what is sent to the `cmsh` session. Some of the output has been elided or truncated for clarity:

Example

```
[bright90->softwareimage]% events on private
Private events:  on
[bright90->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[bright90->softwareimage]%
Tue Apr 29 01:19:12 2014 [notice] bright90: Provisioning started: sendi...
[bright90->softwareimage]%
Tue Apr 29 01:19:52 2014 [notice] bright90: Provisioning completed: sen...
updateprovisioners [ COMPLETED ]
[bright90->softwareimage]% !#events were indeed seen in cmsh session
[bright90->softwareimage]% !#now block the events and rerun update:
[bright90->softwareimage]% events off private
Private events:  off
[bright90->softwareimage]% updateprovisioners
Provisioning nodes will be updated in the background.
[bright90->softwareimage]% !#let this 2nd update run for a while
[bright90->softwareimage]% !#(time passes)
[bright90->softwareimage]% !#nothing seen in cmsh session.
[bright90->softwareimage]% !#show a 2nd update did happen:
[bright90->softwareimage]% events 4 | grep -i provisioning
Tue Apr 29 01:19:12 2014 [notice] bright90: Provisioning started: sendi...
Tue Apr 29 01:19:52 2014 [notice] bright90: Provisioning completed: sen...
Tue Apr 29 01:25:37 2014 [notice] bright90: Provisioning started: sendi...
Tue Apr 29 01:26:01 2014 [notice] bright90: Provisioning completed: sen...
```

12.11.3 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@bright90 ~]# echo "Some text" > /var/spool/cmd/eventbucket
```

This adds an event with, by default, the `info` severity level, to the event pane, with the `InfoMessage` “Some text”.

12.11.4 InfoMessages

InfoMessages are optional messages that inform the administrator of the reason for the status change of a measurable, or an event in the cluster.

Measurable scripts can use file descriptor 3 within their scripts to write an InfoMessage:

Example

```
echo "Drive speed unknown: Reverse polarity" >&3
```

Event Bucket Severity Levels

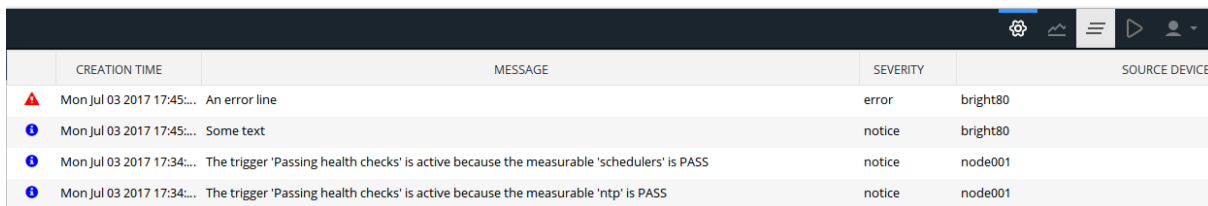
To write events at specific severity levels (section 12.2.7), and not just at the info level, the appropriate text can be prepended from the following to the text that is to be displayed:

```
EVENT_SEVERITY_DEBUG:
EVENT_SEVERITY_INFO:
EVENT_SEVERITY_NOTICE:
EVENT_SEVERITY_WARNING:
EVENT_SEVERITY_ERROR:
EVENT_SEVERITY_ALERT:
```

Example

```
echo "EVENT_SEVERITY_ERROR:An error line" > /var/spool/cmd/eventbucket
```

The preceding example displays an output notification in the Bright View event viewer as shown in figure 12.30:



CREATION TIME	MESSAGE	SEVERITY	SOURCE DEVICE
Mon Jul 03 2017 17:45:...	An error line	error	bright80
Mon Jul 03 2017 17:45:...	Some text	notice	bright80
Mon Jul 03 2017 17:34:...	The trigger 'Passing health checks' is active because the measurable 'schedulers' is PASS	notice	node001
Mon Jul 03 2017 17:34:...	The trigger 'Passing health checks' is active because the measurable 'ntp' is PASS	notice	node001

Figure 12.30: Bright View Monitoring: Event Bucket Message Example

Event Bucket Filter

Regex expressions can be used to conveniently filter out the user-defined messages that are about to go into the event bucket from the shell. The filters used are placed in the event bucket filter, located by default at `/cm/local/apps/cmd/etc/eventbucket.filter`.

Event Bucket CMDaemon Directives

The name and location of the event bucket file and the event bucket filter file can be set using the `EventBucket` and `EventBucketFilter` directives from the `CMDaemon` configuration file directives (Appendix C).

Adding A User-Defined Message To A Device State With The Event Bucket

While the event bucket is normally used to send a message to the event viewer, it can instead be used to add a message to the state of a device. The line passed to the `echo` command then has the message and device specified in the following format:

```
STATE.USERMESSAGE[. device] : [message].
```

The device can be anything with a status property, such as, for example, a node, a switch, or a chassis.

Example

```
echo "STATE.USERMESSAGE.node001:just right" > /var/spool/cmd/eventbucket
```

The state then shows as:

```
cmsh -c "device ; status node001"
node001 ..... (just right) [ UP ]
```

If the device is not specified, then the current host of the shell that is executing the `echo` command is used. For example, running these commands from the head node, `bright90`, as follows:

Example

```
echo "STATE.USERMESSAGE:too hot" > /var/spool/cmd/eventbucket
ssh node001 'echo "STATE.USERMESSAGE:too cold" > /var/spool/cmd/eventbucket'
```

yields these states:

```
cmsh -c "device ; status bright90"
bright90 ..... (too hot) [ UP ]
cmsh -c "device ; status node001"
node001 ..... (too cold) [ UP ]
```

The added text can be cleared with echoing a blank message to that device. For example, for `node001` that could be:

```
echo "STATE.USERMESSAGE.node001:" > /var/spool/cmd/eventbucket
```

Using An Event Bucket During The Node-installer Stage

The node-installer runs before `systemd` is up on the node that is being provisioned. This means that `CM-Daemon` is also not yet running on that node, so that the regular event bucket features are not available during that time. However, a simplified event bucket—the node-installer event bucket—is available during this stage.

The node-installer event bucket can be particularly useful if debugging larger initialize and finalize scripts (Appendix E).

To use it, text is echoed to `/tmp/eventbucket` within the node or category scripts. The text will show up (if permitted) within the sessions of `cmsh`, and within the events viewer of Bright View.

There are two different modes for the node-installer event bucket:

1. Device status info-message updater mode:

Example

```
echo "info-message: this text will be shown in the device status" > /tmp/eventbucket
```

2. Warning event mode:

Example

```
echo "Some text that will become an event" > /tmp/eventbucket
```


13

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.

13.1 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 13.1.1), runs from within the OS shell. That is, `pdsh` executes its commands from within `bash` by default.
- `pexec` (parallel execute, section 13.1.2), runs from within `CMDaemon`. That is, `pexec` executes its commands from within the `cmsh` front end.

A one-time execution of `pdsh` or `pexec` can run one or more shell commands on a group of nodes in parallel.

A Warning About Power Surge Risks With `pdsh` And `pexec`

Some care is needed when running `pdsh` or `pexec` with commands that affect power consumption. For example, running commands that power-cycle devices in parallel over a large number of nodes can be risky because it can put unacceptable surge demands on the power supplies.

Within `cmsh`, executing a `power reset` command from `device` mode to power cycle a large group of nodes is much safer than running a parallel command to do a reset using `pdsh` or `pexec`. This is because the `CMDaemon` power reset powers up nodes after a deliberate delay between nodes (section 4.2).

Which Command To Use Out Of `pdsh` And `pexec`

The choice of using `pdsh` or `pexec` commands is mostly up to the administrator. The only time that running `pdsh` from `bash` is currently required instead of running `pexec` from within `cmsh`, is when stopping and restarting `CMDaemon` on a large number of regular nodes (section 2.6.1). This is because a command to stop `CMDaemon` on a regular node, that itself relies on being run from a running `CMDaemon` on a regular node, can obviously give unexpected results.

13.1.1 `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`

- `pdsh-mod-cmupdown`
- `pdsh-mod-genders`
- `pdsh-rcmd-exec`
- `pdsh-ssh`

The `pdsh` utility is modular, that is, it gets extra functionality through the addition of modules.

The `genders` package is used to generate a default `/etc/genders` configuration file. The file is used to decide what and how nodes are to be used or excluded by default, when used with `pdsh`. Configuration details can be found in `man pdsh(1)`. The configuration can probably best be understood by viewing the file itself and noting that 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@bright90 ~]# csmsh -c "device status"
node001 ..... [ UP ]
node002 ..... [ DOWN ]
bright90 ..... [ 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@bright90 ~]# pdsh -A hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@bright90: node002: ssh exited with exit code 255
bright90: bright90
```

-v: With this option an attempt is made to run the command only on nodes nodes that are in the state UP:

Example

```
[root@bright90 ~]# pdsh -A -v hostname
node001: node001
bright90: bright90
```

-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@bright90 ~]# 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@bright90 ~]# pdsh -w node00[1-2] hostname
node001: node001
node002: ssh: connect to host node002 port 22: No route to host
pdsh@bright90: 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@bright90 ~]# 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@bright90 ~]# 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@bright90 ~]# pdsh -A ls /etc/services /etc/yp.conf
bright90: /etc/services
bright90: /etc/yp.conf
node001: /etc/services
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright90: node001: ssh exited with exit code 2
node002: /etc/services
node002: /etc/yp.conf
```

With **dshbak**, with no **dshbak** options:

Example

```
[root@bright90 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright90: node001: ssh exited with exit code 2
-----
bright90
-----
/etc/services
/etc/yp.conf
-----
node001
-----
/etc/services
-----
node002
```

```
-----
/etc/services
/etc/yp.conf
[root@bright90 ~]#
```

With dshbak, with the `-c` (coalesce) option:

Example

```
[root@bright90 ~]# pdsh -A ls /etc/services /etc/yp.conf | dshbak -c
node001: ls: cannot access /etc/yp.conf: No such file or directory
pdsh@bright90: node001: ssh exited with exit code 2
-----
node002,bright90
-----
/etc/services
/etc/yp.conf
-----
node001
-----
/etc/services
[root@bright90 ~]#
```

The dshbak utility is useful for creating human-friendly output in clusters with larger numbers of nodes.

13.1.2 pexec In cmsh

In cmsh, the pexec command is run from device mode:

Example

```
[bright90->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
```

13.1.3 pexec In Bright View

In Bright View, pexec is hidden, but executed in a GUI wrapper, using the clickpath `Cluster→Run command`.

For large numbers of nodes, rendering the output into the node subpanes (little boxes) can take a long time. To improve the Bright View experience, selecting the `Single text view` icon instead of the `Grouped view` icon speeds up the rendering significantly, but at the cost of removing the borders of the subpanes.

Ticking the `Join output` checkbox places output that is the same for particular nodes, into the same subpane.

Running parallel shell commands from cmsh instead of in Bright View is faster in most cases, due to less graphics rendering overhead.

13.1.4 Using The `-j|--join` Option Of `pexec` In `cmsh`

The output of the `pexec` command by default can come out in a sequence depending on node response time. To make it more useful for an administrator, order can be imposed on the output. Checking consistency across nodes is then trivial. For example, to see if all nodes have the same mounts on a cluster with 10 nodes, of which `node002` is down:

Example

```
[bright90->device]% pexec -j -c default "mount|sort"
Nodes down:          node002
[node002]
Node down

[node001,node003..node010]
/dev/hda1 on / type ext3 (rw,noatime,nodiratime)
/dev/hda2 on /var type ext3 (rw,noatime,nodiratime)
/dev/hda3 on /tmp type ext3 (rw,nosuid,nodev,noatime,nodiratime)
/dev/hda6 on /local type ext3 (rw,noatime,nodiratime)
master:/cm/shared on /cm/shared type nfs
(rw,rsize=32768,wsiz=32768,hard,intr,addr=10.141.255.254)
master:/home on /home type nfs
(rw,rsize=32768,wsiz=32768,hard,intr,addr=10.141.255.254)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
none on /proc type proc (rw,nosuid)
none on /sys type sysfs (rw)
```

Here, 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.

13.1.5 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:

```
[bright90->device]%help <pcommand>
```

13.2 Getting Support With Cluster Manager Issues, And Notifications For Release Updates

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, or a release update. This section describes how to get support for technical issues, and notifications for release updates.

13.2.1 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 e-mail. The e-mail exchanges begin via the Bright website. The website allows the administrator to submit a support request via a web form, and opens up a trouble ticket.

- A useful indirect starting point that can be used for all technical support is at:

<https://support.brightcomputing.com>

- Cluster administrators that have a Bright customer portal account can click on the Request Support link in the navigation menu on the left-hand column of the portal page. A direct URL for this is:

<https://customer.brightcomputing.com/Customer-Portal?p=requestsupport>

- Cluster administrators that do not have a Bright customer portal account can use the web form at:

<https://customer.brightcomputing.com/support?p=submitSupportRequest>

The correct, up-to-date, product key (section 4.3.2 of the *Installation Manual*) for the cluster should be entered in the web form to allow tickets to be dealt with smoothly. If the product key for the cluster has been registered (section 4.3 of the *Installation Manual*), then all tickets associated with the key can be viewed after logging in.

When opening up the ticket, it is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket goes via standard e-mail.

Putting obviously different problems into one ticket results in a slower resolution. Issues that are clearly independent of each other should be submitted as separate tickets. This ensures the quickest processing by the support team.

The document "How To Ask Questions The Smart Way" at <http://www.catb.org/esr/faqs/smart-questions.html> is quite readable, and explains how to submit a good technical query on on-line forums. While the document is not entirely relevant to the support provided by Bright Computing, it is true that many of the principles covered there are also useful when sending a query to Bright Computing when trying to resolve an issue. A read through the document may therefore be helpful before submitting a query to Bright Computing.

If the issue appears to be a bug, then a bug report should be submitted. It is helpful to include as many details as possible to ensure the development team is able to reproduce the apparent bug. The

policy at Bright Computing is to welcome such reports, to provide feedback to the reporter, and to resolve the problem.

If there is apparently no response from Bright Computing over e-mail, checking the spam folder is advised. Mail providers have been known to flag mails as spam, even in mid-thread. Users in China should also be aware that mails from Bright Computing may be deleted by accident by the Golden Shield Project (the “Great Firewall”) operated by the Chinese Ministry of Public Security.

As a supplement to e-mail support, Bright Computing also provides the `cm-diagnose` (section 13.2.2) and the `request-remote-assistance` (section 13.2.3) utilities to help resolve issues. The use of third-party screen-sharing utilities (section 13.2.4) is also possible.

13.2.2 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@bright90 ~]# cm-diagnose
To be able to contact you about the issue, please provide
your e-mail address (or support ticket number, if applicable):
frankfurter@example.com

To diagnose any issue, it would help if you could provide a description
about the issue at hand.
Do you want to enter such description? [Y/n]

End input with ctrl-d
I tried X, Y, and Z on the S2464 motherboard. When that didn't work, I
tried A, B, and C, but the florbish is grommicking.
Thank you.

If issues are suspected to be in the cmdaemon process, a gdb trace of
that process is useful.
In general such a trace is only needed if Bright Support asks for this.
Do you want to create a gdb trace of the running CMDaemon? [y/N]

Proceed to collect information? [Y/n]

Processing master
  Processing commands
  Processing file contents
  Processing large files and log files
  Collecting process information for CMDaemon
  Executing CMSH commands
  Finished executing CMSH commands

Processing default-image
  Processing commands
  Processing file contents

Creating log file: /root/bright90_609.tar.gz
```


Cleaning up

```
Automatically submit diagnostics to
http://support.brightcomputing.com/cm-diagnose/ ? [Y/n] y
```

```
Uploaded file: bright90_609.tar.gz
Remove log file (/root/bright90_609.tar.gz)? [y/N] y
[root@bright90 ~]
```

13.2.3 Requesting Remote Support With `request-remote-assistance`

The life-cycle of solving a ticket begins with opening a ticket, and then establishing that both the cluster administrator and the support engineer have a basic grasp of the issue at hand. This is best done via an e-mail exchange.

From this stage onward there are many possible paths. The support engineer may offer a solution, or ask for more details, or may ask for some tests to be run. Most of the time e-mail remains the most efficient way to troubleshoot an issue.

However at times it may be more appropriate for the cluster administrator to allow remote support from the Bright Computing support engineer in order to resolve the issue. The support engineer may in that case suggest that the `request-remote-assistance` utility be run.

The `request-remote-assistance` utility allows a Bright Computing engineer to securely tunnel into the cluster, often without a change in firewall or ssh settings of the cluster.

With `request-remote-assistance`:

- It must be allowed to access the `www` and `ssh` ports of Bright Computing's internet servers.
- For some problems, the engineer may wish to power cycle a node. In that case, indicating what node the engineer can power cycle should be added to the option for entering additional information.
- Administrators familiar with `screen` may wish to run it within a `screen` session and detach it so that they can resume the session from another machine. A very short reminder of the basics of how to run a `screen` session is: run the `screen` command, then run the `request-remote-assistance` command, then `ctrl-a d` to detach. To resume, run `screen -r`, and `exit` to exit.

The `request-remote-assistance` command itself is run as follows:

Example

```
[root@bright90 ~]# 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: bright90.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. Whether an engineer is logged in can be viewed with the `w` command, which shows a user running the ssh tunnel, and—if the engineer is logged in—another user session, other than whatever sessions the administrator may be running:

Example

```
[root@bright90 ~]# w
 13:35:00 up 97 days, 17 min, 2 users, load average: 0.28, 0.50, 0.52
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
root pts/0 10.2.37.101 12:24 1:10m 0.14s 0.05s ssh -q -R :7013:127.0.0.1:22\
remote@sandbox.brightcomputing.com bright90
root pts/1 localhost.locald 12:54 4.00s 0.03s 0.03s -bash
```

When the engineer has ended the session, the administrator may remove the secure tunnel with a `ctrl-c`, and the display then shows:

```
Tunnel to sandbox.brightcomputing.com terminated.
Removed temporary Bright public key.
[root@bright90 ~]#
```

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.

13.2.4 Requesting Remote Support With A Shared Screen Utility

If the request-remote-assistance utility (section 13.2.3) is restricted or otherwise disallowed by the site policy, then it may be permitted to allow a Bright support engineer to access the site using a third party shared screen utility that the administrator is comfortable with. In that case, possible options include:

- Zoom (<https://zoom.us>) (preferred)
- WebEx (<https://www.webex.com>)
- join.me (<https://www.join.me>)

Other screensharing options may also be possible. The support engineer and the administrator need to agree upon the option to be used, and decide upon a time (and time zone) for the shared screen session.

13.2.5 Getting Notified About Updates

Updates for the various Bright Cluster Manager releases continue for some time after the initial release is made public. The updates typically containing bugfixes and improvements, and an administrator may wish to install an update when it becomes publicly available.

An administrator can sign up to be notified about an update by logging into the customer portal at <https://customer.brightcomputing.com/>, and then clicking on **Edit Account** from the navigation menu in the left panel (figure 13.1). An e-mail address can be set in the account details, and the versions for which the administrator would like to receive notifications can be selected by ticking on the appropriate **Notify on releases for:** checkboxes.

After submitting the request, an e-mail is sent to the address when a new release is made public for the selected versions.

Bright Computing Contact Us | Partners | Support | Login

Products Solutions Resource Center Blog

Menu

- Overview
- Account
 - Account Overview
 - Edit Account
 - Edit Billing
 - Edit Password
- Product
 - My Clusters
 - My Product Keys
 - Unlock Product Key
- Show Usage
- Bright Links
 - Release Notes
 - Manuals
 - Download ISO
 - Download CMGUI

Edit Account

.....

You are editing your account details.

Organization:

Organization unit:

Sector:

First name:

Last name:

Phone number:

Timezone:

Address:

Country:

Contact e-mail:

Repeat e-mail:

Notify on releases for: v9.0 v8.2 v8.1 v8.0 v7.3 v7.2 v7.1 v7.0 v6.1 v6.0 v5.2 v5.1 v5.0

Figure 13.1: Signing up for release notification at the customer portal

13.3 Backups

13.3.1 Cluster Installation Backup

Bright Cluster Manager does not include facilities to create backups of a cluster installation. The cluster administrator is responsible for deciding on the best way to back up the cluster, out of the many possible choices.

A backup method is strongly recommended, and checking that restoration from backup actually works is also strongly recommended.

One option that may be appropriate for some cases is simply cloning the head node. A clone can be

created by PXE booting the new head node, and following the procedure in section 14.4.8.

When setting up a backup mechanism, it is recommended that the full filesystem of the head node (i.e. including all software images) is backed up. Unless the regular node hard drives are used to store important data, it is not necessary to back them up.

If no backup infrastructure is already in place at the cluster site, the following open source (GPL) software packages may be used to maintain regular backups:

- **Bacula:** Bacula is a mature network based backup program that can be used to backup to a remote storage location. If desired, it is also possible to use Bacula on nodes to back up relevant data that is stored on the local hard drives. More information is available at <http://www.bacula.org>

Bacula requires ports 9101-9103 to be accessible on the head node. Including the following lines in the Shorewall rules file for the head node allows access via those ports from an IP address of 93.184.216.34 on the external network:

Example

```
ACCEPT net:93.184.216.34 fw tcp 9101
ACCEPT net:93.184.216.34 fw tcp 9102
ACCEPT net:93.184.216.34 fw tcp 9103
```

The Shorewall service should then be restarted to enforce the added rules.

- **rsnapshot:** rsnapshot allows periodic incremental filesystem snapshots to be written to a local or remote filesystem. Despite its simplicity, it can be a very effective tool to maintain frequent backups of a system. More information is available at <http://www.rsnapshot.org>.

Rsnapshot requires access to port 22 on the head node.

13.3.2 Local Database And Data Backups And Restoration

The CMDaemon database is stored in the MySQL `cmdaemon` database, and contains most of the stored settings of the cluster.

Monitoring data values are stored as binaries in the filesystem, under `/var/spool/cmd/monitoring`.

The administrator is expected to run a regular backup mechanism for the cluster to allow restores of all files from a recent snapshot. As an additional, separate, convenience:

- For the CMDaemon database, the entire database is also backed up nightly on the cluster filesystem itself ("local rotating backup") for the last 7 days.
- For the monitoring data, the raw data records are not backed up locally, since these can get very large. However, the configuration of the monitoring data, which is stored in the CMDaemon database, is backed up for the last 7 days too.

Database Corruption Messages And Repairs

A corrupted MySQL database is commonly caused by an improper shutdown of the node. To deal with this, when starting up, MySQL checks itself for corrupted tables, and tries to repair any such by itself. Detected corruption causes an event notice to be sent to `cmsh` or Bright View.

When there is database corruption, InfoMessages in the `/var/log/cmdaemon` log may mention:

- "Unexpected eof found" in association with a table in the database,
- "can't find file" when referring to an entire missing table,
- locked tables,
- error numbers from table handlers,

- “Error while executing” a command.

If a basic repair is to be carried out on a database, CMDaemon should first be stopped.

Example

```
[root@headnode ~]# service cmd stop
[root@headnode ~]# myisamchk --recover /var/lib/mysql/mysql/user.MYI
[root@headnode ~]# service cmd start
```

If basic repair fails, more extreme repair options—`man myisamchk(1)` suggests what—can then be tried out.

Another example: If CMDaemon is unable to start up due to a corrupted database, then messages in the `/var/log/cmdaemon` file might show something like:

Example

```
Oct 11 15:48:19 solaris CMDaemon: Info: Initialize cmdaemon database
Oct 11 15:48:19 solaris CMDaemon: Info: Attempt to set provisioningNetwo\
rk (280374976710700) not an element of networks
Oct 11 15:48:19 solaris CMDaemon: Fatal: Database corruption! Load Maste\
rNode with key: 280374976782569
Oct 11 15:48:20 solaris CMDaemon: Info: Sending reconnect command to all\
nodes which were up before master went down ...
Oct 11 15:48:26 solaris CMDaemon: Info: Reconnect command processed.
```

Here it is CMDaemon’s “Database corruption” message that the administrator should be aware of, and which suggests database repairs are required for the CMDaemon database. The severity of the corruption, in this case not even allowing CMDaemon to start up, may mean a restoration from backup is needed. How to restore from backup is explained in the next section.

Restoring From The Local Backup

If the MySQL database repair tools of the previous section do not fix the problem, then, for a failover configuration, the `dbreclone` option (section 14.4.2) should normally provide a CMDaemon and Slurm database that is current. The `dbreclone` option does not clone the monitoring data filesystem.

Cloning extra databases: The file `/cm/local/apps/cluster-tools/ha/conf/extradbclone.xml`.template can be used as a template to create a file `extradbclone.xml` in the same directory. The `extradbclone.xml` file can then be used to define additional databases to be cloned. Running the `/cm/local/apps/cmd/scripts/cm-update-mycnf` script then updates `/etc/my.cnf`. The database can then be cloned with this new MySQL configuration by running

```
cmha dbreclone <passive>
```

where `<passive>` is the hostname of the passive head node.

If the head node is not part of a failover configuration, then a restoration from local backup can be done. The local backup directory is `/var/spool/cmd/backup`, with contents that look like (some text elided):

Example

```
[root@solaris ~]# cd /var/spool/cmd/backup/
[root@solaris backup]# ls -l
total 280
...
-rw----- 1 root root 33804 Oct 10 04:02 backup-Mon.sql.gz
-rw----- 1 root root 33805 Oct 9 04:02 backup-Sun.sql.gz
-rw----- 1 root root 33805 Oct 11 04:02 backup-Tue.sql.gz
...
```

The CMDaemon database snapshots are stored as `backup-<day of week>.sql.gz`. In the example, the latest backup available in the listing for CMDaemon turns out to be `backup-Tue.sql.gz`.

The latest backup can then be unzipped and piped into the MySQL database for the user `cmdaemon`. The password, `<password>`, can be retrieved from `/cm/local/apps/cmd/etc/cmd.conf`, where it is configured in the `DBPass` directive (Appendix C).

Example

```
gunzip backup-Tue.sql.gz
service cmd stop #(just to make sure)
mysql -ucmdaemon -p<password> cmdaemon < backup-Tue.sql
```

Running `“service cmd start”` should have CMDaemon running again, this time with a restored database from the time the snapshot was taken. That means, that any changes that were done to the cluster manager after the time the snapshot was taken are no longer implemented.

Monitoring data values are not kept in a database, but in files (section 13.8).

13.4 Revision Control For Images

Bright Cluster Manager 7 introduced support for the implementations of Btrfs provided by the distributions. Btrfs makes it possible to carry out revision control for images efficiently.

13.4.1 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.

13.4.2 Btrfs Availability And Distribution Support

Btrfs has been part of the Linux kernel since kernel 2.6.29-rc1. Depending on which Linux distribution is being used on a cluster, it may or may not be a good idea to use Btrfs in a production environment, as in the worst case it could lead to data loss.

Btrfs was made available from Red Hat as a technology preview release in RHEL7, but has been deprecated since RHEL7.4, and is not officially supported by Red Hat in any case. It has been officially removed from RHEL8.

Btrfs is also available from SUSE. SLES12 uses a 3.12 series kernel, which has a fully stable and supported implementation of Btrfs.

While Bright Computing has not experienced any problems with storing software images on Btrfs using RHEL7, it is highly advisable to keep backups of important software images on a non-Btrfs filesystems when Btrfs is used on these Linux distributions.

An issue with using `cm-clone-install` with Btrfs is described on page 643.

13.4.3 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@bright90 ~]# yum install btrfs-progs
...
Resolving Dependencies
--> Running transaction check
---> Package btrfs-progs.x86_64 0:4.9.1-1.el7 will be installed
...
Total download size: 678 k
Installed size: 4.0 M
...
Complete!
```

Creating A Btrfs Filesystem

The original images directory can be moved aside first, and a new images directory created to serve as a future mount point for Btrfs:

Example

```
[root@bright90 ~]# cd /cm/
[root@bright90 cm]# mv images images2
[root@bright90 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@bright90 cm]# mkfs.btrfs /dev/sdc1
[root@bright90 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@bright90 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@bright90 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@bright90 cm]# mount -t btrfs butter.img images -o loop
[root@bright90 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@bright90 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@bright90 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@bright90 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 Bright View.

13.4.4 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 13.4.3. 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

```
[bright90->softwareimage]% newrevision default-image "some changes"
[bright90->softwareimage]% newrevision default-image "more changes"
[bright90->softwareimage]% revisions default-image
ID    Date                               Description
-----
1     Fri, 04 Oct 2019 11:19:33 CEST some changes
2     Fri, 04 Oct 2019 11:19:52 CEST more changes
[bright90->softwareimage]% list -r
Name (key)      Path                               Kernel version
-----
default-image  /cm/images/default-image  3.10.0-957.1.3.el7.x86_64
default-image@1 /cm/images/default-image-1 3.10.0-957.1.3.el7.x86_64
default-image@2 /cm/images/default-image-2 3.10.0-957.1.3.el7.x86_64
[bright90->softwareimage]% category add storage; commit
```

With the cluster set up like that, the 3 kinds of revision control functionalities in category mode can be explained as follows:

1. Category revision control functionality is defined as unset

If the administrator sets the `softwareimage` property for the category to an image without any revision tags:

Example

```
[bright90->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

```
[bright90->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

```
[bright90->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 13.4.3.

First, a revision of the image is made to save its initial state:

```
[bright90->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:

```
[bright90->softwareimage]% newrevision default-image "Some modifications"
```

This image is then automatically called `default-image@2` and has the path `/cm/images/default-image-2`. If the administrator then wants to test the latest revision on nodes in the `default` category, then this new image can be set at category level, without having to specify it for every node in that category individually:

```
[bright90->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`:

```
[bright90->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:

```
[bright90->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.

13.5 BIOS Configuration And Updates

A number of tools can be used to configure and update the BIOS of nodes.

13.5.1 Updating BIOS

Upgrading the BIOS to a new version involves using the DOS tools that were supplied with the BIOS. The flash tool and the BIOS image must be copied to a DOS image. The file `autoexec.bat` should be altered to invoke the flash utility with the correct parameters. In case of doubt, it can be useful to

boot the DOS image and invoke the BIOS flash tool manually. Once the correct parameters have been determined, they can be added to the `autoexec.bat`.

After a BIOS upgrade, the contents of the NVRAM may no longer represent a valid BIOS configuration because different BIOS versions may store a configuration in different formats. It is therefore recommended to also write updated NVRAM settings immediately after flashing a BIOS image.

The next section describes how to boot the DOS image.

13.5.2 Booting DOS Image

To boot the DOS image over the network, it first needs to be copied to software image's `/boot` directory, and must be world-readable.

Example

```
cp flash.img /cm/images/default-image/boot/bios/flash.img
chmod 644 /cm/images/default-image/boot/bios/flash.img
```

An entry is added to the PXE boot menu to allow the DOS image to be selected. This can easily be achieved by modifying the contents of `/cm/images/default-image/boot/bios/menu.conf`, which is by default included automatically in the PXE menu. By default, one entry `Example` is included in the PXE menu, which is however invisible as a result of the `MENU HIDE` option. Removing the `MENU HIDE` line will make the BIOS flash option selectable. Optionally the `LABEL` and `MENU LABEL` may be set to an appropriate description.

The option `MENU DEFAULT` may be added to make the BIOS flash image the default boot option. This is convenient when flashing the BIOS of many nodes.

Example

```
LABEL FLASHBIOS
  KERNEL memdisk
  APPEND initrd=bios/flash.img
  MENU LABEL ^Flash BIOS
# MENU HIDE
  MENU DEFAULT
```

The `bios/menu.conf` file may contain multiple entries corresponding to several DOS images to allow for flashing of multiple BIOS versions or configurations.

13.6 Hardware Match Check With The `hardware-profile` Data Producer

Often a large number of identical nodes may be added to a cluster. In such a case it is a good practice to check that the hardware matches what is expected. This can be done easily as follows:

1. The new nodes, say `node129` to `node255`, are committed to a newly-created category `newbunch` as follows (output truncated):

```
[root@bright90 ~]# cmsh -c "category add newbunch; commit"
[root@bright90 ~]# 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:

```
[root@bright90 ~]# for ((i=129; i<=255; i++)) do echo "
> device
> set node00$i category newbunch
> commit"; done | cmsh
```

- The hardware profile of one of the new nodes, say node129, is saved into the category newbunch. This is done using the `node-hardware-profile` health check script:

Example

```
[root@bright90 ~]# /cm/local/apps/cmd/scripts/healthchecks/node-hardware-pr\
ofile -n node129 -s newbunch
```

The profile is intended to be the reference hardware against which all the other nodes should match, and is saved under the directory `/cm/shared/apps/cmd/hardware-profiles/`, and further under the directory name specified by the `-s` option, which in this case is `newbunch`.

- The `hardware-profile` data producer (section 12.2.10) can then be enabled, and the sampling frequency set as follows:

```
[root@bright90 ~]# cmsh
[bright90]% monitoring setup use hardware-profile
[bright90->monitoring->setup[hardware-profile]]% set interval 600; set disabled no; commit
```

The `hardware-profile` data producer should also be set to the category `newbunch` created in the earlier step. This can be done by creating a category group within the `nodeexecutionfilters` submode. Within that group, categories can be set for where the hardware check is to run. For the example, it is just run on one category, `newbunch`:

```
[bright90->monitoring->setup[hardware-profile]]% nodeexecutionfilters
[bright90->...-profile->nodeexecutionfilters]% add category filterhwp
[bright90->...-profile->nodeexecutionfilters*[filterhwp*]]% set categories newbunch
[bright90->...-profile->nodeexecutionfilters*[filterhwp*]]% commit
```

- 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.

13.7 Serial Over LAN Console Access

Direct console access to nodes is not always possible. Other possibilities to access the node are:

- SSH access via an ssh client.** This requires that an `ssh` server runs on the node and that it is accessible via the network. Access can be via one of the following options:
 - a regular SSH client, run from a bash shell
 - via an `ssh` command run from the device mode of `cmsh`
 - via an `ssh` terminal launched from Bright View via the clickpath:

```
Devices→Nodes→node→Connect→ssh.
```
- Remote shell via CMDaemon.** This is possible if CMDaemon is running on the node and accessible via Bright View or `cmsh`.

- In Bright View, An interactive root shell session can be started up on a node via the clickpath: Devices→Nodes→*node*→Connect→Root shell.
This session is connected to the node via CMDaemon, and runs bash by default.
- For cmsh, in device mode, running the command `rshell node001` launches an interactive bash session connected to node001 via CMDaemon.

3. **Connecting via a serial over LAN console.** If a serial console is configured, then a serial over LAN (SOL) console can be accessed from cmsh (`rconsole`).

Item 3 in the preceding list, SOL access, is a useful low-level access method that is covered next more thoroughly with:

- some background notes on serial over LAN console access (section 13.7.1)
- the configuration of SOL with Bright View (section 13.7.2)
- the configuration of SOL with cmsh (section 13.7.3)
- the conman SOL logger and viewer (section 13.7.4)

13.7.1 Background Notes On Serial Console And SOL

Serial ports are data ports that can usually be enabled or disabled for nodes in the BIOS.

If the serial port of a node is enabled, it can be configured in the node kernel to redirect a console to the port. The serial port can thus provide what is called serial console access. That is, the console can be viewed using a terminal software such as minicom (in Linux) or Hyperterminal (in Windows) on another machine to communicate with the node via the serial port, using a null-modem serial cable. This has traditionally been used by system administrators when remote access is otherwise disabled, for example if ssh access is not possible, or if the TCP/IP network parameters are not set up right.

While traditional serial port console access as just described can be useful, it is inconvenient, because of having to set arcane serial connection parameters, use the relatively slow serial port and use a special serial cable. Serial Over LAN (SOL) is a more recent development of serial port console access, which uses well-known TCP/IP networking over a faster Ethernet port, and uses a standard Ethernet cable. SOL is thus generally more convenient than traditional serial port console access. The serial port DB-9 or DB-25 connector and its associated 16550 UART chip rarely exist on modern servers that support SOL, but they are nonetheless usually implied to exist in the BIOS, and can be “enabled” or “disabled” there, thus enabling or disabling SOL.

SOL is a feature of the BMC (Baseboard Management Controller) for IPMI 2.0 and iLO. For DRAC, CIMC, and Redfish, SOL via IPMI is used. SOL is enabled by configuring the BMC BIOS. When enabled, data that is going to the BMC serial port is sent to the BMC LAN port. SOL clients can then process the LAN data to display the console. As far as the node kernel is concerned, the serial port is still just behaving like a serial port, so no change needs to be made in kernel configuration in doing whatever is traditionally done to configure serial connectivity. However, the console is now accessible to the administrator using the SOL client on the LAN.

SOL thus allows SOL clients on the LAN to access the Linux serial console if

1. SOL is enabled and configured in the BMC BIOS
2. the serial console is enabled and configured in the node kernel
3. the serial port is enabled and configured in the node BIOS

The BMC BIOS, node kernel, and node BIOS therefore all need to be configured to implement SOL console access.

Background Notes: BMC BIOS Configuration

The BMC BIOS SOL values are usually enabled and configured as a submenu or pop-up menu of the node BIOS. These settings must be manually made to match the values in 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 13.7.2 and 13.7.3 explain how SOL access configuration is set up for the node kernel using Bright View or `cmsh`. SOL access configuration on the node kernel is serial access configuration on the node kernel as far as the system administrator is concerned; the only difference is that the word “serial” is replaced by “SOL” in Bright Cluster Manager’s Bright View and `cmsh` front ends to give a cluster perspective on the configuration.

Background Notes: Node BIOS Configuration

Since BIOS implementations vary, and serial port access is linked with SOL access in various ways by the BIOS designers, it is not possible to give short and precise details on how to enable and configure them. The following rules-of-thumb, if followed carefully, should allow most BMCs to be configured for SOL access with 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 (section 13.7.3). 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.

13.7.2 SOL Console Configuration With Bright View

In Bright View, SOL configuration settings can be carried out per image via the clickpath Provisioning→Software Images→*image*→Edit→Settings

If the Enable SOL option is set to Yes then the kernel option to make the Linux serial console accessible is used after the node is rebooted.

This means that if the serial port and SOL are enabled for the node hardware, then after the node reboots the Linux serial console is accessible over the LAN via an SOL client.

If SOL is correctly configured in the BIOS and in the image, then access to the Linux serial console is possible via the minicom serial client running on the computer (from a bash shell for example), or via the rconsole serial client running in cmsh.

13.7.3 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@bright90 ~]# cmsh
[bright90]% softwareimage use default-image

[bright90->softwareimage[default-image]]% set enablesol yes
[bright90->softwareimage*[default-image*]]% commit
```

The SOL settings for a particular image can be seen with the show command:

```
[bright90->softwareimage[default-image]]% show | grep SOL
Parameter                      Value
-----
Enable SOL                      yes
SOL Flow Control                yes
SOL Port                        ttyS1
SOL Speed                       115200
```

Values can be adjusted if needed with the set command.

On rebooting the node, the new values are used.

To access a node via an SOL client, the node can be specified from within the device mode of cmsh, and the rconsole command run on cmsh on the head node:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node001
[bright90->device[node001]]% rconsole
  screen cleared and the following conman output is displayed:
=====
conman
To exit IPMI SOL, type <ENTER> "&" "."
=====
```

```
<ConMan> Connection to console [node001] opened.
```

If at this point, there is no further response in conman on the console after pressing the <ENTER> key, then there is a communication failure, probably due to a misconfigured communication parameter. This could happen, for example, if the serial port ttyS1 has been set, but the node is connected on ttyS0. Setting the value of SOL Port to ttyS0 and rebooting the node to pick up the new value, would solve that issue, so that pressing the <ENTER> key, would display the node console:

Example

```
Ubuntu 18.04.2 LTS node001 ttyS0
```

```
node001 login:
```

13.7.4 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 the “Enable SOL” option in Bright View, or if the `enablesol` in `cmsd` is enabled for the software image, then the `conman` configuration is written out and the `conman` service is started.

Logging The Serial Console

The data seen at the serial console is then logged via SOL to the head node after reboot. For each node that has logging enabled, a log file is kept on the head node. For example, for `node001` the log file would be at `/var/log/conman/node001.log`. To view the logged console output without destroying terminal settings, using `less` with the `-R` option is recommended, as in: `less -R /var/log/conman/node001.log`.

Using The Serial Console Interactively

Viewing quirk during boot: In contrast to the logs, the console viewer shows the initial booting stages of the node as it happens. There is however a quirk the system administrator should be aware of:

Normally the display on the physical console is a copy of the remote console. However, during boot, after the remote console has started up and been displaying the physical console for a while, the physical console display freezes. For the Linux 2.6 kernel series, the freeze occurs just before the ramdisk is run, and means that the display of the output of the launching `init.d` services is not seen on the physical console (figure 13.2).

```

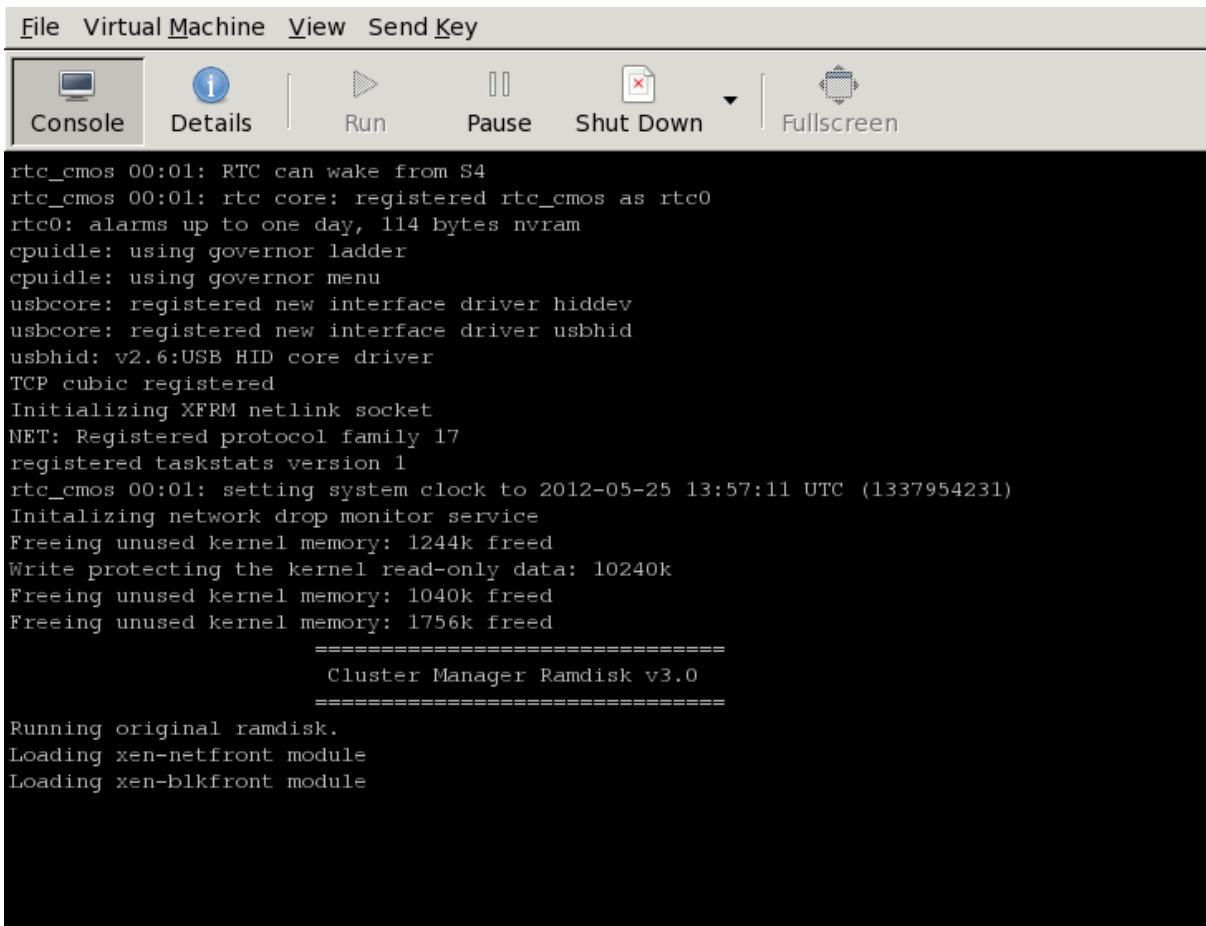
File Virtual Machine View Send Key
Console Details Run Pause Shut Down Fullscreen

PNP: PS/2 Controller [PNP0303:KBD,PNP0f13:MOU] at 0x60,0x64 irq 1,12
serio: i8042 KBD port at 0x60,0x64 irq 1
serio: i8042 AUX port at 0x60,0x64 irq 12
mice: PS/2 mouse device common for all mice
input: AT Translated Set 2 keyboard as /devices/platform/i8042/serio0/input/inpu
t2
rtc_cmos 00:01: RTC can wake from S4
rtc_cmos 00:01: rtc core: registered rtc_cmos as rtc0
rtc0: alarms up to one day, 114 bytes nvram
cpuidle: using governor ladder
cpuidle: using governor menu
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
TCP cubic registered
Initializing XFRM netlink socket
NET: Registered protocol family 17
registered taskstats version 1
rtc_cmos 00:01: setting system clock to 2012-05-25 13:57:11 UTC (1337954231)
Initalizing network drop monitor service
Freeing unused kernel memory: 1244k freed
Write protecting the kernel read-only data: 10240k
Freeing unused kernel memory: 1040k freed
Freeing unused kernel memory: 1756k freed
-

```

Figure 13.2: Physical Console Freeze During SOL Access

The freeze is only a freeze of the display, and should not be mistaken for a system freeze. It occurs because the kernel is configured during that stage to send to only one console, and that console is the remote console. The remote console continues to display its progress (figure 13.3) during the freeze of the physical console display.

The image shows a screenshot of a virtual machine's remote console interface. At the top, there is a menu bar with 'File', 'Virtual Machine', 'View', and 'Send Key'. Below the menu bar is a toolbar with icons for 'Console', 'Details', 'Run', 'Pause', 'Shut Down', and 'Fullscreen'. The main area of the window is a black terminal with white text displaying system boot logs. The logs include messages from the RTC, CPU idle governors, USB core drivers, TCP, XFRM, NET, and taskstats, as well as memory freeing statistics and the Cluster Manager Ramdisk v3.0 initialization process.

```
rtc_cmos 00:01: RTC can wake from S4
rtc_cmos 00:01: rtc core: registered rtc_cmos as rtc0
rtc0: alarms up to one day, 114 bytes nvram
cpuidle: using governor ladder
cpuidle: using governor menu
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
TCP cubic registered
Initializing XFRM netlink socket
NET: Registered protocol family 17
registered taskstats version 1
rtc_cmos 00:01: setting system clock to 2012-05-25 13:57:11 UTC (1337954231)
Initalizing network drop monitor service
Freeing unused kernel memory: 1244k freed
Write protecting the kernel read-only data: 10240k
Freeing unused kernel memory: 1040k freed
Freeing unused kernel memory: 1756k freed

=====
Cluster Manager Ramdisk v3.0
=====
Running original ramdisk.
Loading xen-netfront module
Loading xen-blkfront module
```

Figure 13.3: Remote Console Continues During SOL Access During Physical Console Freeze

Finally, just before login is displayed, the physical console once more (figure 13.4) starts to display what is on the remote console (figure 13.5).

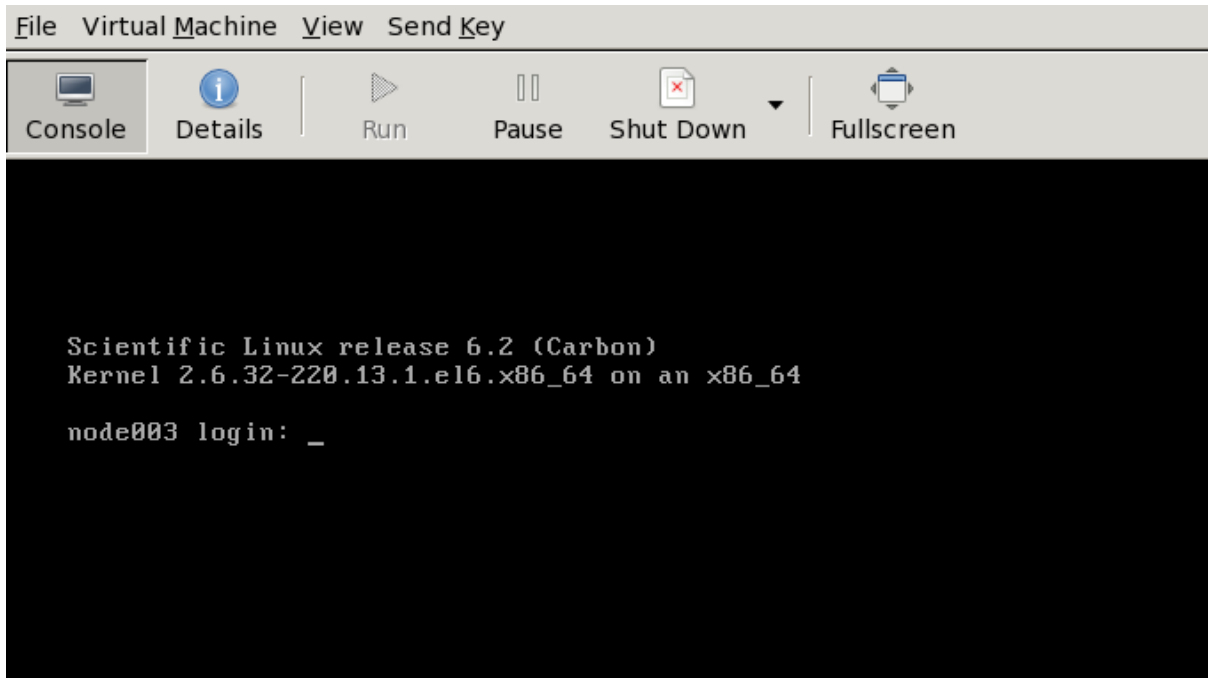


Figure 13.4: Physical Console Resumes After Freeze During SOL Access

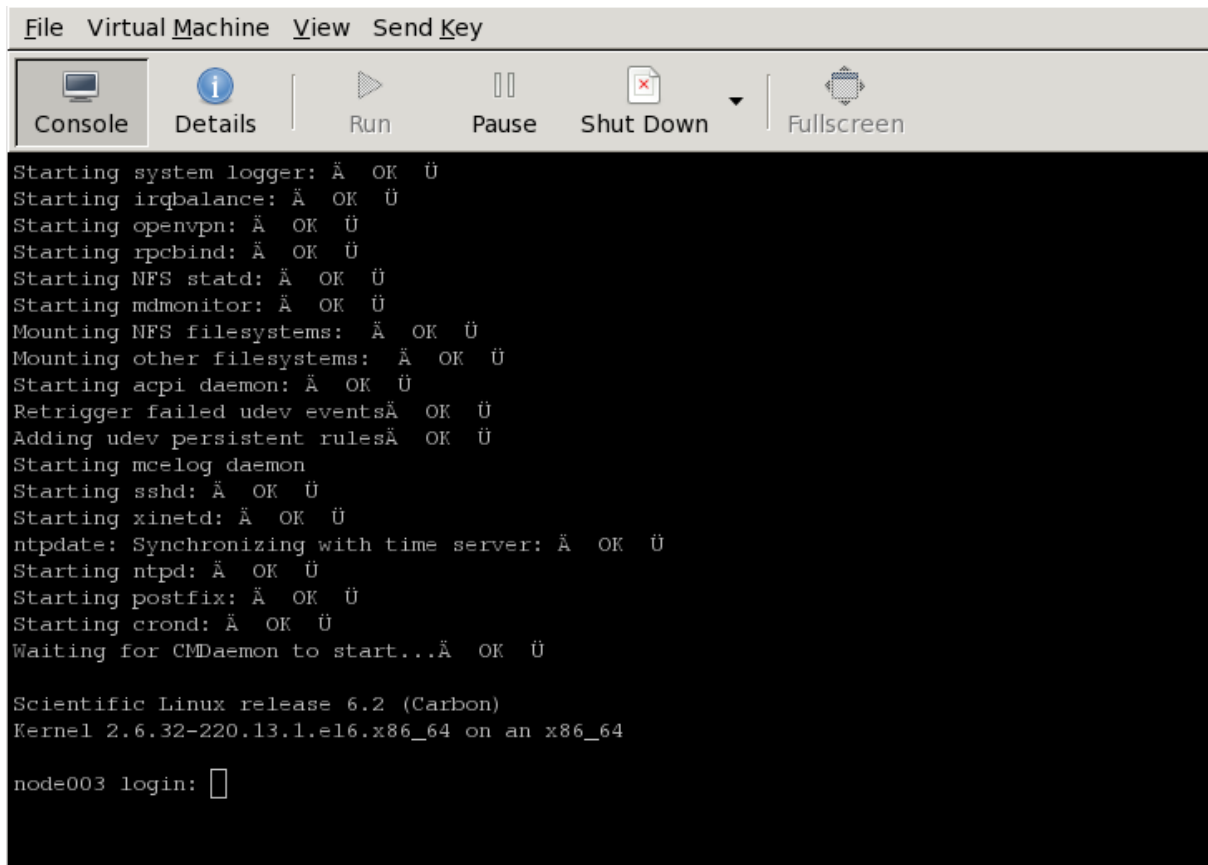


Figure 13.5: Remote Console End Display After Boot

The physical console thus misses displaying several parts of the boot progress.

Exit sequence: The `conman` console viewer session can be exited with the sequence `&.` (the last entry in the sequence being a period). Strictly speaking, the `&.` sequence must actually be preceded by an `<ENTER>`.

The console buffer issue when accessing the remote console: A feature of SOL console clients is that the administrator is not presented with any text prompt from the node that is being accessed. This is useful in some cases, and can be a problem in others.

An example of the issue is the case where the administrator has already logged into the console and typed in a command in the console shell, but has no intention of pressing the `<ENTER>` key until some other tasks are first carried out. If the connection breaks at this point, then the command typed in is held in the console shell command buffer, but is not displayed when a remote serial connection is re-established to the console—the previously entered text is invisible to the client making the connection. A subsequent `<ENTER>` would then attempt to execute the command. This is why an `<ENTER>` is not sent as the last key sequence during automated SOL access, and it is left to the administrator to enter the appropriate key strokes.

To avoid commands in the console shell buffer inadvertently being run when taking over the console remotely, the administrator can start the session with a `<CTRL>-u` to clear out text in the shell before pressing `<ENTER>`.

13.8 Managing Raw Monitoring Data

From Bright Cluster Manager version 8.0 onwards, the raw monitoring data values are stored as binary data under `/var/spool/cmd/monitoring` instead of as binary data within MySQL or MariaDB. The reason behind this change was to significantly increase performance. The monitoring subsystem in Bright Cluster Manager was thoroughly rewritten for this change.

13.8.1 Monitoring Subsystem Disk Usage With The `monitoringinfo --storage` Option

The disk usage by the monitoring subsystem can be viewed using the `monitoringinfo` command with the `--storage` option:

Example

```
[bright90->device]% monitoringinfo master --storage
Storage                Elements  Disk size  Usage   Free disk
-----
Mon::Storage::Engine   1,523    1.00 GiB   1.28%   14.1 GiB
Mon::Storage::Message   1        16.0 MiB   0.000%   -
Mon::Storage::RepositoryId 1,528    47.7 KiB   100.0%   -
```

The Engine component stores the raw monitoring data. It grows in 1GB increments each time its usage reaches 100%.

13.8.2 Estimating The Required Size Of The Storage Device

The final size of the monitoring directory can be estimated with the Bright Cluster Manager script, `cm-bright-monitoring-usage.py`.

The size estimate assumes that there are no changes in configuration, such as enabling advanced metrics for jobs, or increasing the maximum number of labeled entities, or large numbers of running jobs.

The size estimate value is the maximum value it will take if the cluster runs forever. It is therefore an over-estimate in practice.

Example

```
[root@bright90 ~]# /cm/local/apps/cmd/scripts/monitoring/cm-bright-monitoring-usage.py
Number of used entities:      6
Number of used measurables:  231
Number of measurables:      231
Number of data producers:    95
Number of consolidators:     2

Current monitoring directory: /var/spool/cmd/monitoring
Monitoring directory size:   1.024 GB
Maximal directory size:     1.125 GB
```

13.8.3 Moving Monitoring Data Elsewhere

A procedure to move monitoring data from the default `/var/spool/cmd/monitoring/` directory to a new directory is as follows:

1. A new directory in which monitoring should be saved is picked.

The block storage device for the directory should not be a shared DAS (Direct Attached Storage, such as a locally attached drive) or a NAS (Network Attached Storage, such as NFS or Lustre which work over a network connection). That is because if there is an outage, then:

- If such a DAS storage becomes unavailable at some time, then CMDaemon assumes that no monitoring data values exist, and creates an empty data file on the local storage. If the DAS storage comes back and is mounted again, then it hides the underlying files, which would lead to discontinuous values and related issues.
- If such a NAS storage is used, then an outage of the NAS can make CMDaemon unresponsive as it waits for input and output. In addition, when CMDaemon starts with a NAS storage, and if the NAS is unavailable for some reason, then an inappropriate mount may happen as in the DAS storage case, leading to discontinuous values and related issues.

2. The `MonitoringPath` directive (page 721) is given the new directory as its value.
3. CMDaemon is stopped (`service cmd stop`).
4. The `/var/spool/cmd/monitoring/` directory is moved to the new directory.
5. CMDaemon is restarted (`service cmd start`).

13.8.4 Reducing Monitoring Data By Reducing Samples

Options to reduce the amount of monitoring data gathered include reducing the `Maximal age` and `Maximal samples` for data producers (section 12.4.1) to smaller, but still non-zero values. After re-initializing the monitoring data, so that existing data is removed, the values reported by the `cm-bright-monitoring-usage.py` script (section 13.8.2) then show the new storage estimates for the monitoring data.

13.8.5 Deleting All Monitoring Data

A procedure to delete all monitoring data from the default `/var/spool/cmd/monitoring/` directory is as follows:

1. The CMDaemon service on all nodes can be stopped by running the following on the active head node:

```
pdsh -g all service cmd stop
```
2. On both head nodes, the monitoring data is removed with:

```
rm -f /var/spool/cmd/monitoring/*
rm -f /var/spool/cmd/backup*/var/spool/cmd/monitoring/*
```

3. On both head nodes, the associated database tables for the CMDaemon user are cleared with a MySQL session run on each head node.

The CMDaemon database user is `cmdaemon` by default, but the value can be checked with a `grep` on the `cmd.conf` file:

```
[root@bright90 ~]# grep ^DBUser /cm/local/apps/cmd/etc/cmd.conf
DBUser = "cmdaemon"
```

Similarly, the password for the `cmdaemon` user can be found with a `grep` as follows:

```
[root@bright90 ~]# grep ^DBPass /cm/local/apps/cmd/etc/cmd.conf
DBPass = "slarti8813bartfahrt"
```

The monitoring measurables can then be deleted by running a session on each head node as follows:

Example

```
[root@bright90 ~]# mysql -ucmdaemon -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2909
Server version: 5.5.56-MariaDB MariaDB Server
```

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
MariaDB [(none)]> use cmdaemon;
Database changed
MariaDB [cmdaemon]> truncate MonitoringMeasurables;
MariaDB [cmdaemon]> truncate MonitoringMeasurableMetrics;
MariaDB [cmdaemon]> truncate MonitoringMeasurableHealthChecks;
MariaDB [cmdaemon]> truncate MonitoringMeasurableEnums;
MariaDB [cmdaemon]> truncate EntityMeasurables;
MariaDB [cmdaemon]> truncate EnumMetricValues;
MariaDB [cmdaemon]> truncate LabeledEntities;
MariaDB [cmdaemon]> truncate JobInformation;
MariaDB [cmdaemon]> exit
repeat on other head node
```

4. On both head nodes, CMDaemon can then be restarted with:

```
service cmd start
```

5. On the active head node, after the command:

```
cmha status
```

shows all is OK, the CMDaemon service can be started on all regular nodes again. The OK state should be achieved in about 15 seconds.

The CMDaemon service is started with, for example:

```
pdsh -g computenode service cmd start
```

13.9 Node Replacement

To replace an existing node with a new node, the node information can be updated via `cmsh`.

If the new MAC address is known, then it can set that for the node. If the MAC address is not known, then the existing entry can be cleared.

If the MAC address is not known ahead of time, then the node name for the machine should be selected when it is provisioning for the first time. The steps for a new node `node031` would be as follows:

Example

```
[root@bright90 ~]# cmsh
[bright90]% device use node0031
if new mac address is known, then:
[bright90->device[node031]]% set mac <new mac address>
else if new mac address is not known:
[bright90->device[node031]]% clear mac
the changed setting in either case must be committed:
[bright90->device[node031]]% commit
```

If the disk is the same size as the one that is being replaced, and everything else matches up, then this should be all that needs to be done

There is more information on the node installing system in section 5.4. How to add a large number of nodes at a time efficiently is described in that section. The methods used can include the `newnodes` command of `cmsh` (page 175) and the `Nodes Identification` resource of Bright View (page 179).

14

High Availability

14.0 Introduction

14.0.1 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.

14.0.2 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. The purpose of this design is to increase availability to beyond that provided by a single head node.

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.

14.0.3 High Availability Usually Uses Shared Storage

HA is typically configured using shared storage (section 14.1.5), such as from an NFS service, which typically provides the /home directory on the active (section 14.1.1) head, and on the regular nodes.

14.0.4 Organization Of This Chapter

The remaining sections of this chapter are organized as follows:

- **HA On Head Nodes**
 - Section 14.1 describes the concepts behind HA, keeping the Bright Cluster Manager configuration in mind.
 - Section 14.2 describes the normal user-interactive way in which the Bright Cluster Manager implementation of a failover setup is configured.
 - Section 14.3 describes the implementation of the Bright Cluster Manager failover setup in a less user-interactive way, which avoids using the Ncurses dialogs of section 14.2
 - Section 14.4 describes how HA is managed with Bright Cluster Manager after it has been set up.
- **HA On Regular Nodes**

- Section 14.5 describes the concepts behind HA for regular nodes, and how to configure HA for them.
- **HA And Workload Manager Jobs**
 - Section 14.6 describes the support for workload manager job continuation during HA failover.

14.1 HA Concepts

14.1.1 Primary, Secondary, Active, Passive

Naming: In a cluster with an HA setup, one of the head nodes is named the *primary* head node and the other head node is named the *secondary* head node.

Mode: Under normal operation, one of the two head nodes is in *active* mode, whereas the other is in *passive* mode.

The difference between naming versus mode is illustrated by realizing that while a head node which is primary always remains primary, the mode that the node is in may change. Thus, the primary head node can be in passive mode when the secondary is in active mode. Similarly the primary head node may be in active mode while the secondary head node is in passive mode. As an aside: the definition for primary in HA for Bright Cluster Manager should not be confused with the definition for primary that is used by workload managers such as Slurm and PBS Pro when a failover mechanism is configured by the workload manager (section 7.2.4).

The difference between active and passive is that the active head takes the lead in cluster-related activity, while the passive follows it. Thus, for example, with MySQL transactions, CMDaemon carries them out with MySQL running on the active, while the passive trails the changes. This naturally means that the active corresponds to the master, and the passive to the slave, in the MySQL master-slave replication mode that MySQL is run as.

14.1.2 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.

14.1.3 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
- **Workload Management:** For each of the Slurm, PBS, LSF services, one server is active on one head node, while the other server is a passive standby on the other head node

When an HA setup is created from a single head node setup, the above services are automatically reconfigured to run in the HA environment over two head nodes.

Provisioning role runs on both head nodes In addition, both head nodes also take up the *provisioning role*, which means that nodes can be provisioned from both head nodes. As the passive head node is then also provisioned from the active, and the active can switch between primary and secondary, it means both heads are also given a value for `provisioninginterface` (section 5.4.7).

For a head node in a single-headed setup, there is no value set by default. For head nodes in an HA setup, the value of `provisioninginterface` for each head node is automatically set up by default to the interface device name over which the image can be received when the head node is passive.

The implications of running a cluster with multiple provisioning nodes are described in detail in section 5.2. One important aspect described in that section is how to make provisioning nodes aware of image changes.

From the administrator's point of view, achieving awareness of image changes for provisioning nodes in HA clusters is dealt with in the same way as for single-headed clusters. Thus, if using `cmsh`, the `updateprovisioners` command from within `softwareimage` mode is used, whereas if Bright View is used, then the clickpath `Provisioning`→`Provisioning requests`→`Update provisioning nodes` can be followed (section 5.2.4).

Services That Migrate To The Active Node

Although it is possible to configure any service to migrate from one head node to another in the event of a failover, in a typical HA setup only the following services migrate:

- NFS
- The User Portal
- Workload management: The SGE/UGE server (`sgemaster`)

14.1.4 Failover Network Topology

A two-head failover network layout is illustrated in figure 14.1.

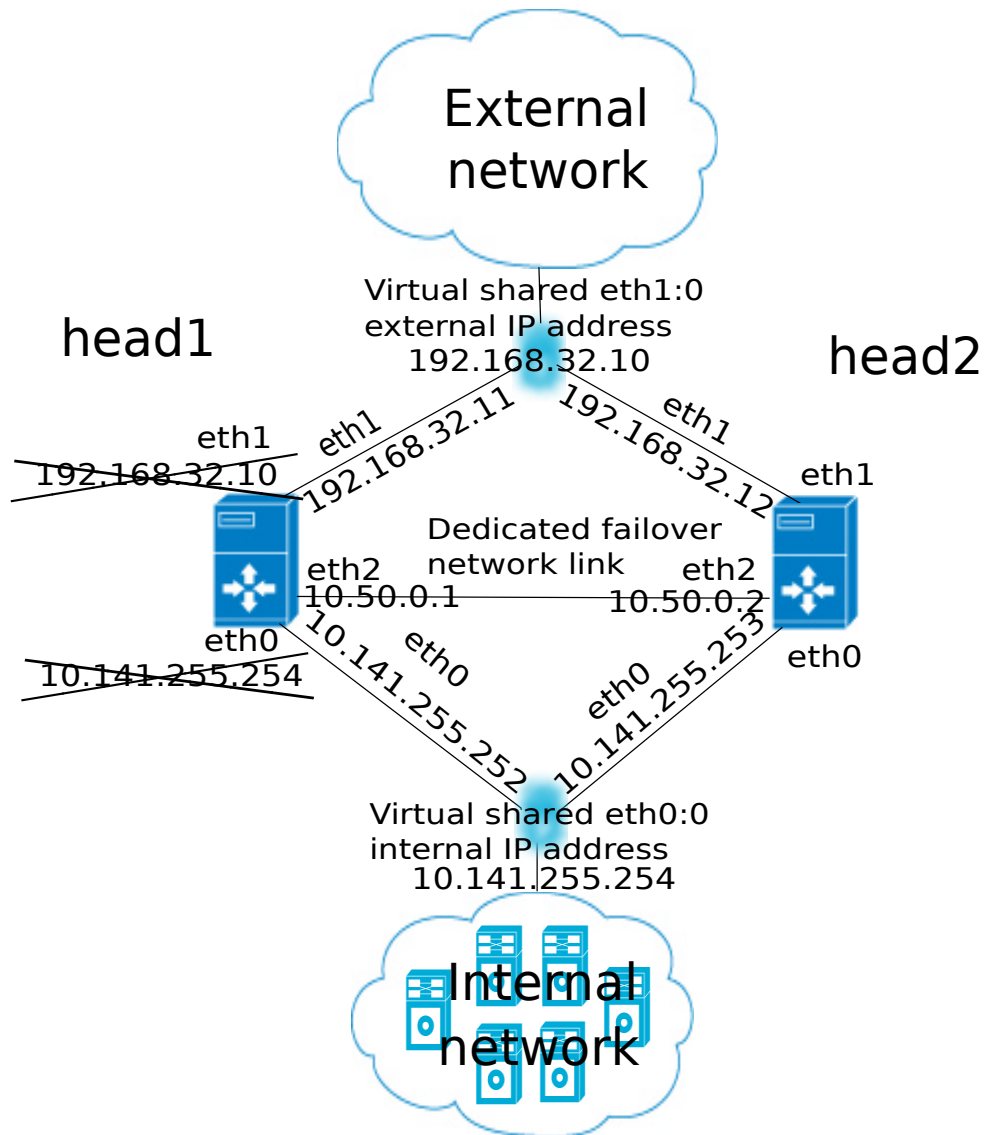


Figure 14.1: High Availability: Two-Head Failover Network Topology

In the illustration, the primary head1 is originally a head node before the failover design is implemented. It is originally set up as part of a Type 1 network (section 3.3.9 of the *Installation Manual*), with an internal interface eth0, and an external interface eth1.

When the secondary head is connected up to help form the failover system, several changes are made.

HA: Network Interfaces

Each head node in an HA setup typically has at least an external and an internal network interface, each configured with an IP address.

In addition, an HA setup uses two virtual IP interfaces, each of which has an associated virtual IP address: the external shared IP address and the internal shared IP address. These are shared between the head nodes, but only one head node can host the address and its interface at any time.

In a normal HA setup, a shared IP address has its interface hosted on the head node that is operating in active mode. On failover, the interface migrates and is hosted on the head node that then becomes active.

When head nodes are also being used as login nodes, users outside of the cluster are encouraged

to use the shared external IP address for connecting to the cluster. This ensures that they always reach whichever head node is active. Similarly, inside the cluster, nodes use the shared internal IP address wherever possible for referring to the head node. For example, nodes mount NFS filesystems on the shared internal IP interface so that the imported filesystems continue to be accessible in the event of a failover.

Shared interfaces are implemented as alias interfaces on the physical interfaces (e.g. `eth0:0`). They are activated when a head node becomes active, and deactivated when a head node becomes passive.

HA: Dedicated Failover Network

In addition to the normal internal and external network interfaces on both head nodes, the two head nodes are usually also connected using a direct dedicated network connection, `eth2` in figure 14.1. 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.

14.1.5 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. Lustre or GPFS storage are other popular choices.

Because imported mounts can typically not be re-exported (which is true at least for NFS), nodes typically mount filesystems directly from the NAS device.

DAS

In a Direct Attached Storage (DAS) setup, both head nodes share access to a block device, often accessed through a SCSI interface. This could be a disk-array that is connected to both head nodes, or it could be a block device that is exported by a corporate SAN infrastructure.

Although the block device is visible and can physically be accessed simultaneously on both head nodes, the filesystem that is used on the block device is typically not suited for simultaneous access. Simultaneous access to a filesystem from two head nodes must therefore be avoided because it generally leads to filesystem corruption. Only special purpose parallel filesystems such as GPFS and Lustre are capable of being accessed by two head nodes simultaneously.

DRBD

The use of DRBD is not recommended for Bright Cluster Manager, and RHEL7 and CentOS7 no longer provide packages for it.

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.

Custom Shared Storage With Mount And Unmount Scripts

The cluster management daemon on the two head nodes deals with shared storage through a *mount script* and an *unmount script*. When a head node is moving to active mode, it must acquire the shared filesystems. To accomplish this, the other head node first needs to relinquish any shared filesystems that may still be mounted. After this has been done, the head node that is moving to active mode invokes the *mount script* which has been configured during the HA setup procedure. When an active head node is requested to become *passive* (e.g. because the administrator wants to take it down for maintenance without disrupting jobs), the *unmount script* is invoked to release all shared filesystems.

By customizing the *mount* and *unmount* scripts, an administrator has full control over the form of shared storage that is used. Also an administrator can control which filesystems are shared.

Mount scripts paths can be set via `cmsh` or Bright View (section 14.4.6).

14.1.6 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 14.1.7. 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 14.2.4).

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.

14.1.7 Automatic Vs Manual Failover

Administrators have a choice between creating an HA setup with automatic or manual failover.

- In the case of an automatic failover, an active head node is powered off when it is no longer responding at all, and a failover sequence is initiated automatically.
- In the case of a manual failover, the administrator is responsible for initiating the failover when the active head node is no longer responding. No automatic power off is done, so the administrator is asked to confirm that the previously active node is powered off.

For automatic failover to be possible, power control must be defined for both head nodes. If power control is defined for the head nodes, then automatic failover is attempted by default.

The administrator may disable automatic failover. In `cmsh` this is done by setting the `disableautomaticfailover` property, which is a part of the HA-related parameters (section 14.4.6):

```
[root@bright90 ~]# cmsh
[bright90]% partition failover base
[bright90->partition[base]->failover]% set disableautomaticfailover yes
[bright90->partition*[base*]->failover*]% commit
```

With Bright View it is carried out via the clickpath Cluster→Partition[base]→Settings→Failover→Disable automatic failover

If no power control has been defined, or if automatic failover has been disabled, or if the power control mechanism is not working (for example due to inappropriate, broken or missing electronics or hardware), then a failover sequence must always be initiated manually by the administrator.

Sometimes, if automatic failover is enabled, but the active head is still slightly responsive (the so-called *mostly dead* state, described in section 14.4.2), then the failover sequence must also be initiated manually by the administrator.

14.1.8 HA And Cloud Nodes

As far as the administrator is concerned, HA setup remains the same whether a Cluster Extension (Chapter 3 of the *Cloudbursting Manual*) is configured or not, and whether a Cluster On Demand (Chapter 2 of the *Cloudbursting Manual*) is configured or not. Behind the scenes, on failover, any networks associated with the cloud requirements are taken care of by Bright Cluster Manager.

14.1.9 HA Using Virtual Head Nodes

Two physical servers are typically used for HA configurations. However, each head node can also be a virtual machine (VM). The use case for this might be to gain experience with an HA configuration.

Failover Network Considerations With HA VMs

With physical head nodes in an HA configuration, the failover network, used for HA heartbeats, is typically provided by running a network cable directly between the ethernet port on each machine. Not having even a switch in between is a best practice. Since the head nodes are typically in the same, or adjacent racks, setting this up is usually straightforward.

With VMs as head nodes in an HA configuration, however, setting up the failover network can be more complex:

- The cluster administrator may need to consider if HA is truly improved by, for example, connecting the failover network of the physical node to a switch, and then plumbing all that into all the hypervisors as a vnic (in the case of VMware).
- Often a virtual switch would be used between the virtual head nodes, just because it is often easier.
- Not using a failover network is also an option, just as in the physical case.
- If one head node is on one hypervisor, and another is on a second hypervisor, then a standard Bright Cluster Manager setup cannot have one head node carry out an automated failover STONITH because it cannot contact the other hypervisor. So powering off the VM in the other hypervisor would have to be done manually. An alternative to this, if automated failover is required, is to create custom power scripts.

Bright Cluster Manager has no specific guidelines for the network configuration of HA with VMs. The process of configuration is however essentially the same as for a physical node.

Size Considerations With HA VMs

A virtual head node in practice may be configured with fewer CPUs and memory than a physical head node, just because such configurations are more common options in a VM setup than for a physical setup, and cheaper to run. However, the storage requirement is the same as for a physical node. The important requirement is that the head nodes should have sufficient resources for the cluster.

14.2 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 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*).

Existing User Certificates Become Invalid

Installing the new license means that any existing user certificates will lose their validity (page 60 of the *Installation Manual*) on Bright View session logout. This means:

- If LDAP is managed by Bright Cluster Manager, then on logout, new user certificates are generated, and a new Bright View login session picks up the new certificates automatically.
- For LDAPs other than that of 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 Bright View or `cmsh` along with additional steps, this is not supported, and should not be attempted as it is error-prone.

A basic HA setup is created in three stages:

1. **Preparation** (section 14.2.1): the configuration parameters are set for the shared interface and for the secondary head node that is about to be installed.
2. **Cloning** (section 14.2.2): the secondary head node is installed by cloning it from the primary head node.
3. **Shared Storage Setup** (section 14.2.3): the method for shared storage is chosen and set up.

An optional extra stage is:

4. **Automated Failover Setup** (section 14.2.4): Power control to allow automated failover is set up.

14.2.1 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 14.2.1-14.2.3, and possibly section 14.2.4) are complete.
1. If bonding (section 3.5) is to be used on the head node used in an HA setup, then it is recommended to configure and test out bonding properly before carrying out the HA setup.

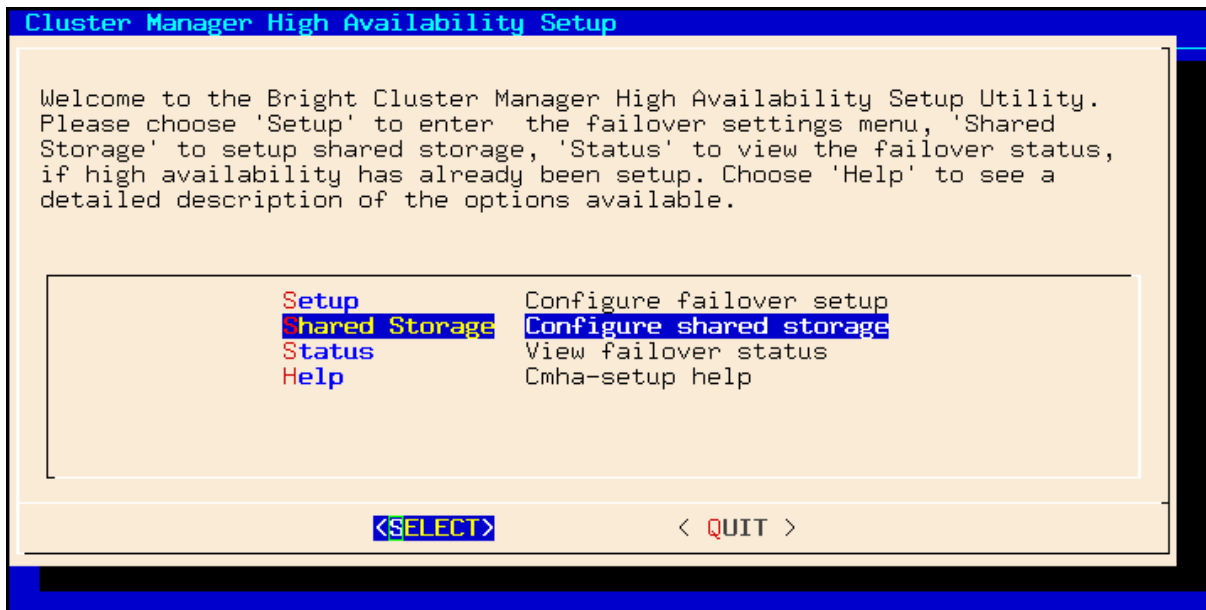


Figure 14.2: cmha-setup Main menu

2. To start the HA setup, the `cmha-setup` command is run from a root shell on the primary head node.
3. Setup is selected from the main menu (figure 14.2).
4. Configure is selected from the Setup menu.
5. A license check is done. Only if successful does the setup proceed further. If the cluster has no HA-enabled license, a new HA-enabled license must first be obtained from the Bright Cluster Manager reseller, and activated (section 4.3 of the *Installation Manual*).
6. The virtual shared internal alias interface name and virtual shared internal IP alias address are set.
7. The virtual shared external alias interface name and virtual shared external IP alias address are set. For the external shared virtual IP address as well as for the external regular IP addresses, each head node external interface address must be a static IP addresses for the HA configuration. Attempting to use DHCP for external addresses in HA is not going to work.
8. The host name of the passive is set.
9. Failover network parameters are set. The failover network physical interface should exist, but the interface need not be up. The network name, its base address, its netmask, and domain name are set. This is the network used for optional heartbeat monitoring.
10. Failover network interfaces have their name and IP address set for the active and passive nodes.
11. The primary head node may have other network interfaces (e.g. InfiniBand interfaces, a BMC interface, alias interface on the BMC network). These interfaces are also created on the secondary head node, but the IP address of the interfaces still need to be configured. For each such interface, when prompted, a unique IP address for the secondary head node is configured.
12. The network interfaces of the secondary head node are reviewed and can be adjusted as required. DHCP assignments on external interfaces can be set by using the word `DHCP`. If the primary head node has a DHCP-assigned IP address, then the input field for the secondary head node contain the word `DHCP` by default.

13. A summary screen displays the planned failover configuration. If alterations need to be made, they can be done via the next step.
14. The administrator is prompted to set the planned failover configuration. If it is not set, the main menu of cmha-setup is re-displayed.
15. If the option to set the planned failover configuration is chosen, then a password for the MySQL root user is requested. The procedure continues further after the password is entered.
16. Setup progress for the planned configuration is displayed (figure 14.3).

```

Setup progress
  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 ]

***
FAILOVER SETUP SUCCESS
***

[

< NEXT >      < SKIP >      < BACK >

```

Figure 14.3: cmha-setup Setup Progress For Planned Configuration

17. Instructions on what to run on the secondary to clone it from the primary are displayed (figure 14.4).

```

Cluster Manager High Availability Setup

The failover setup initialization on the primary master is done.
Now boot the secondary master into the rescue environment and run the
following command:

/cm/cm-clone-install --failover

and follow the instructions.

Once the installation has begun, select 'Install Progress' from the
Failover setup menu, to see
the installation progress of the clone machine. When the installation is
complete, and the
secondary master is up, select 'Finalize' from the Failover setup menu,
to complete the
failover setup process.

100%

< OK >

```

Figure 14.4: cmha-setup Instructions To Run On Secondary For Cloning

14.2.2 Failover Cloning (Replacing A Passive Head)

In the IT industry, if an image is made of a computer, then it means making a copy of the drive. In Bright Cluster Manager the word “image” is normally used for the software that can be placed on regular nodes. So, Bright Cluster Manager uses the word “cloning” to describe making a very similar, or even identical, copy of a head node, using the `/cm/cm-clone-install` command.

There are actually two kinds of cloning possible with the `/cm/cm-clone-install` command:

- **Failover cloning:** With this, a passive head node can be created from the active head node. This uses the `--failover` option to create a copy that is very similar to the active head node, but with changes to make it a passive head, ready for failover purposes, and replacing a head that has just failed.
- **Re-cloning:** An active head node can be created from the active head node. This uses the `--clone` option to create an exact copy (re-clone) of the head node. This might be useful if for some reason the administrator would like to take a snapshot of the head node at that moment. Using this snapshot to have a plug-in replacement head node—that is, a head node kept aside, and ready to replace a failed production head node later on—is not recommended, due to how impractical it is to update the snapshot.

The process described in this section PXE boots the passive from the active, thereby loading a special rescue image from the active that allows cloning from the active to the passive to take place. This section is therefore about failover cloning. How to carry out re-cloning is described in section 14.4.8.

After the preparation has been done by configuring parameters as outlined in section 14.2.1, 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 14.4.8). In that case the active head node can be either the primary or secondary.

1. The passive head node is PXE booted off the internal cluster network, from the active head node. It is highly recommended that the active and passive head nodes have identical hardware configurations. The BIOS clock of the head nodes should match and be set to the local time. Typically, the BIOS of both head nodes is also configured so that a hard disk boot is attempted first, and a PXE boot is attempted after a hard disk boot failure, leading to the Cluster Manager PXE Environment menu of options. This menu has a 5s time-out.
2. In the Cluster Manager PXE Environment menu of the node that is to become a clone, before the 5s time-out, “Start Rescue Environment” is selected to boot the node into a Linux ramdisk environment.
3. Once the rescue environment has finished booting, a login as root is done. No password is required (figure 14.5).
4. The following command is executed (figure 14.6) on the node that is to become a failover clone:

```
/cm/cm-clone-install --failover
```

When doing a re-clone as in section 14.4.8, instead of a failover clone, then it is the `--clone` option that is used instead of the `--failover` option.
5. When prompted to enter a network interface to use, the interface that was used to boot from the internal cluster network (e.g. `eth0`, `eth1`, ...) is entered. There is often uncertainty about what interface name corresponds to what physical port. This can be resolved by switching to another console and using “`ethtool -p <interface>`”, which makes the NIC corresponding to the interface blink.

```

-----
*Welcome to the Cluster Manager rescue environment*
-----
:
: 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:
:
: # pdmenu           "Menu frontend to programs!"
: # dhcpup dhcpcd   "Setup wired network connection!"
: # wificonfig       "Setup wireless network connection!"
: # mnsetup          "Setup mail and news!"
: # lynx (or) links  "WWW browsers!" # rtin (or) slrn "Newsreaders!"
:
: You can use 'backup-mbr' to backup/restore the MBR.
:
: login: root
-----

ClusterManager login: root

```

Figure 14.5: Login Screen After Booting Passive Into Rescue Mode From Active

6. If the provided network interface is correct, a root@master's password prompt appears. The administrator should enter the root password.
7. An opportunity to view or edit the master disk layout is offered.
8. A confirmation that the contents of the specified disk are to be erased is asked for.
9. The cloning takes place. The "syncing" stage usually takes the most time. Cloning progress can also be viewed on the active by selecting the "Install Progress" option from the Setup menu. When viewing progress using this option, the display is automatically updated as changes occur.
10. After the cloning process has finished, a prompt at the console of the passive asks if a reboot is to be carried out. A "y" is typed in response to this. The passive node should be set to reboot off its hard drive. This may require an interrupt during reboot, to enter a change in the BIOS setting, if for example, the passive node is set to network boot first.
11. Continuing on now on the active head node, Finalize is selected from the Setup menu of cmha-setup.
12. The MySQL root password is requested. After entering the MySQL password, the progress of the Finalize procedure is displayed, and the cloning procedure continues.
13. The cloning procedure of cmha-setup pauses to offer the option to reboot the passive. The administrator should accept the reboot option. After reboot, the cloning procedure is complete. The administrator can then go to the main menu and quit from there or go on to configure "Shared Storage" (section 14.2.3) 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

```

ClusterManager login: root
Welcome to Linux 2.6.32-220.23.1.el6.x86_64.
No mail.
# /cm/cm-clone-install --failover
Network interface to use [default: eth0]:
Please wait while authentication is being set up...
root@master's password:
Please wait while installation begins...
Verifying license ..... [ OK ]
Getting build config ..... [ OK ]
Getting disk layout ..... [ OK ]
The head node disk layout is saved in /cm/__masterdisksetup.xml
[lv - view, e - edit, c - continue ]: c
The contents of the following disks will be erased.
/dev/sda
Do you want to continue [yes/no]? yes
Getting mount points ..... [ OK ]
Partitioning hard drive ..... [ OK ]
Syncing hard drive ..... [ OK ]
Finalizing installation ..... [ OK ]
Do you want to reboot[y/n]:_

```

Figure 14.6: Cloning The Passive From The Active Via A Rescue Mode Session

```

[root@bright90 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright90* -> master2
  failoverping [ OK ]
  mysql        [ OK ]
  ping         [ OK ]
  status       [ OK ]
master2 -> bright90*
  failoverping [ OK ]
  mysql        [ OK ]
  ping         [ OK ]
  status       [ OK ]

```

Here, the `mysql`, `ping` and `status` states indicate that HA setup completed successfully. The `failoverping` state uses the dedicated failover network route for its checks, and starts working as soon as the passive head node has been rebooted.

14.2.3 Shared Storage Setup

After cloning the head node (section 14.2.2), 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 legacy form of shared storage at the time of writing (May 2018).

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 14.1.5. 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 620).

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. After 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.

14.2.4 Automated Failover And Relevant Testing

A power-off operation on the active head node server does not mean the same as just pulling out the power cable to the active head node. These actions typically have different effects, and should therefore not be confused with each other. During the power-off operation, the BMC remains up. However, in the case of pulling out the power cable, the BMC is typically turned off too. If the BMC is not reachable, then it means that verifying that the active head has been terminated is uncertain. This is because the data that CMDaemon can access implies a logical possibility that there is a network failure rather than a head node failure. CMDaemon therefore does not carry out an automatic failover if the power cable is pulled out.

For automatic failover to work, the two head nodes must be able to power off their counterpart. This is done by setting up power control (Chapter 4).

Testing If Power Control Is Working

The "device power status" command in cmsh can be used to verify that power control is functional:

Example

```
[master1]% device power status -n mycluster1,mycluster2
apc03:21 ..... [ ON ] mycluster1
apc04:18 ..... [ ON ] mycluster2
```

Testing The BMC Interface Is Working

If a BMC (Baseboard Management Controller, section 3.7) such as IPMI or iLO is used for power control, it is possible that a head node is not able to reach its own BMC interface over the network. This is especially true when no dedicated BMC network port is used. In this case, `cmsh -c "device power status"` reports a failure for the active head node. This does not necessarily mean that the head nodes cannot reach the BMC interface of their counterpart. Pinging a BMC interface can be used to verify that the BMC interface of a head node is reachable from its counterpart.

Example

Verifying that the BMC interface of mycluster2 is reachable from mycluster1:

```
[root@mycluster1 ~]# ping -c 1 mycluster2.bmc.cluster
PING mycluster2.bmc.cluster (10.148.255.253) 56(84) bytes of data.
64 bytes from mycluster2.bmc.cluster (10.148.255.253): icmp_seq=1
ttl=64 time=0.033 ms
```

Verifying that the BMC interface of mycluster1 is reachable from mycluster2:

```
[root@mycluster2 ~]# ping -c 1 mycluster1.bmc.cluster
PING mycluster1.bmc.cluster (10.148.255.254) 56(84) bytes of data.
64 bytes from mycluster1.bmc.cluster (10.148.255.254): icmp_seq=1
ttl=64 time=0.028 ms
```


Testing Automated Failover Against A Simulated Crash

A normal (graceful) shutdown of an active head node, does not cause the passive to become active, because HA assumes a graceful failover means there is no intention to trigger a failover. To carry out testing of an HA setup with automated failover, it is therefore useful to simulate a kernel crash on one of the head nodes. The following command crashes a head node instantly:

```
echo c > /proc/sysrq-trigger
```

After the active head node freezes as a result of the crash, the passive head node powers off the machine that has frozen and switches to active mode. A hard crash like this can cause a database replication inconsistency when the crashed head node is brought back up and running again, this time passively, alongside the node that took over. This is normally indicated by a FAILED status for the output of `cmha status` for MySQL (section 14.4). Database administration with the `dbreclone` command (section 14.4) 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 14.4.2). 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.

14.3 Running `cmha-setup` Without Ncurses, Using An XML Specification

14.3.1 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.

14.3.2 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 14.2.1. 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 14.2.2.
- `-f|--finalize`: After the passive node is cloned as described in steps 1 to 10 of section 14.2.2, 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 14.2.2.

- `-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.

14.3.3 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@bright90 ~]# cd /cm/local/apps/cluster-tools/ha/conf
[root@bright90 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 14.2.1.

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 14.2.2.

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@bright90 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 14.2.2.

Adding Storage:

Continuing with the session on the active, setting up a shared storage could be done with:

```
[root@bright90 conf]# cmha-setup -x -c failoverconf.xml -s nas
```

The preceding corresponds to carrying out the NAS procedure of section 14.2.3.

14.4 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.

14.4.1 Changing An Existing Failover Configuration

Changing an existing failover configuration is usually done most simply by running through the HA setup procedure of section 14.2 again, with one exception. The exception is that the existing failover configuration must be removed by using the “Undo Failover” menu option between steps 3 and 4 of the procedure described in section 14.2.1.

14.4.2 `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
                      [node] is specified, then make [node] the active
                      node in the failover group that [node] is part of.

dbreclone <host>     Clone MySQL database from this head node to
                      <host> (hostname of failover head node).
```

Some of the information and functions of `cmha` can also be carried out via `CMDaemon`:

- For `cmsh`, the following commands can be run from within the base object in partition mode:
 - For the head node, the `status` and `makeactive` commands are run from within the `failover` submode.
 - For regular nodes the `nodestatus` and `makeactive [node]` commands are run from within the `failovergroups` submode.

The `dbreclone` option cannot be carried out in Bright View or `cmsh` because it requires stopping `CMDaemon`.

The `cmha` options `status`, `makeactive`, and `dbreclone` are looked at in greater detail next:

`cmha status`: Querying HA Status

Information on the failover status is displayed thus:

Example

```
[root@mycluster1 ~]# cmha status
Node Status: running in active master mode
```

```
Failover status:
mycluster1* -> mycluster2
  failoverping [ OK ]
  mysql       [ OK ]
  ping        [ OK ]
  status      [ OK ]
mycluster2 -> mycluster1*
  failoverping [ OK ]
  mysql       [ OK ]
  ping        [ OK ]
  status      [ OK ]
```

The * in the output indicates the head node which is currently active. The status output shows 4 aspects of the HA subsystem from the perspective of each head node:

HA Status	Description
failoverping	the other head node is reachable via the dedicated failover network. This failover ping uses the failover route instead of the internal net route. It uses ICMP ping
mysql	MySQL replication status
ping	the other head node is reachable over the primary management network. It uses ICMP ping.
status	CMDaemon running on the other head node responds to REST calls

By default, Bright Cluster Manager prepares to carry out the failover sequence (the sequence that includes a STONITH) when all three of ping, failoverping and status are not OK on a head node. If these three are not OK, then the active node is *all dead* according to cmha. One way of initiating failover is thus by causing a system crash (section 14.2.4).

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 14.1.7), then the failover sequence attempts to complete automatically if power management is working properly, and the cmha status shows ping, failoverping and status as failed.

If automatic failover is disabled, then a manual failover operation must be executed to have a failover operation take place. A manual failover operation can be carried out with the “cmha makeactive” command:

Example

To initiate a failover manually:

```
[root@mycluster2 ~]# cmha makeactive
Proceeding will initiate a failover sequence which will make this node
(mycluster2) the active master.

Are you sure ? [Y/N]
y
Your session ended because: CMDaemon failover, no longer master
mycluster2 became active master, reconnecting your cmsh ...
```

On successful execution of the command, the former active head node simply continues to run as a passive head node.

The `cmha makeactive` command assumes both head nodes have no problems preventing the execution of the command.

One possible problem that can halt manual failover is if nodes are being provisioned by the provisioning subsystem at that time (section 5.2.4). In that case, provisioning should be cancelled by the cluster administrator before the `cmha makeactive` command can continue, for example, with `cmsh -c "softwareimage cancelprovisioningrequest -a"` (page 192).

For automatic failover no such intervention takes place—provisioning requests are killed when the active head node is powered off.

Another slightly similar problem that can occur for automatic failover, as well as manual failover, is the “mostly dead” edge case. This case requires careful consideration before the `Are you sure ?` prompt is answered by the cluster administrator.

`cmha makeactive` edge case—the mostly dead active:

- For a manual failover operation, if the execution of the `cmha makeactive` command has problems, then it can mean that there is a problem with the initially active head node being in a sluggish state. That is, neither fully functioning, nor all dead. The active head node is thus in a state that is still powered on, but what can be called *mostly dead*. Mostly dead means slightly alive (not all of `ping`, `failoverping`, and `status` are FAILED), while all dead means there is only one thing that can sensibly be done to make sure the cluster keeps running—that is, to make the old passive the new active.

Making an old passive the new active is only safe if the old active is guaranteed to not come back as an active head node. This guarantee is set by a STONITH (page 621) 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@bright90 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright90* -> head2
  failoverping [ OK ]
  mysql        [ OK ]
  ping         [ OK ]
  status       [ OK ]
head2 -> bright90*
  failoverping [ OK ]
  mysql        [FAILED] (11)
  ping         [ OK ]
  status       [ OK ]
[root@bright90 ~]# cmha dbreclone head2
Proceeding will cause the contents of the cmdaemon state database on he\
ad2 to be resynchronized from this node (i.e. bright90 -> 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@bright90 ~]# cmha status
Node Status: running in active master mode

Failover status:
bright90* -> head2
  failoverping [ OK ]
  mysql        [ OK ]
  ping         [ OK ]
  status       [ OK ]
head2 -> bright90*
  failoverping [ OK ]
  mysql        [ OK ]
  ping         [ OK ]
  status       [ OK ]
```

14.4.3 States

The state a head node is in can be determined in three different ways:

- 1 By looking at the message being displayed at login time.

Example

```
-----
Node Status: running in active master mode
-----
```

2 By executing `cmha status`.

Example

```
[root@mycluster ~]# cmha status
Node Status: running in active master mode
...
```

3 By examining `/var/spool/cmd/state`.

There are a number of possible states that a head node can be in:

State	Description
INIT	Head node is initializing
FENCING	Head node is trying to determine whether it should try to become active
ACTIVE	Head node is in active mode
PASSIVE	Head node is in passive mode
BECOMEACTIVE	Head node is in the process of becoming active
BECOMEPASSIVE	Head node is in the process of becoming passive
UNABLETOBECOMEACTIVE	Head node tried to become active but failed
ERROR	Head node is in error state due to unknown problem

Especially when developing custom mount and unmount scripts, it is quite possible for a head node to go into the `UNABLETOBECOMEACTIVE` state. This generally means that the mount and/or unmount script are not working properly or are returning incorrect exit codes. To debug these situations, it is helpful to examine the output in `/var/log/cmdaemon`. The `cmha makeactive` shell command can be used to instruct a head node to become active again.

14.4.4 Failover Action Decisions

A table summarizing the scenarios that decide when a passive head should take over is helpful:

Event on active	Reaction on passive	Reason
Reboot	Nothing	Event is usually an administrator action. To make the passive turn active, an administrator would run <code>cmha makeactive</code> on it.

...continues

...continued

Event on active	Reaction on passive	Reason
Shutdown	Nothing	As above.
Unusably sluggish or freezing system by state pingable with ICMP packets	Nothing	1. Active may still unfreeze. 2. Shared filesystems may still be in use by the active. Concurrent use by the passive taking over therefore risks corruption. 3. Mostly dead head can be powered off by administrator after examining situation (section 14.4.2).
Become passive in response to "cmha makeactive" run on passive	Become active when former active becomes passive	As ordered by administrator
Active dies	Quorum called, may lead to passive becoming new active	Confirms if active head is dead according to other nodes too. If so, then a "power off" command is sent to it. If the command is successful, the passive head becomes the new active head.

14.4.5 Keeping Head Nodes In Sync

What Should Be Kept In Sync?

- It is a best practice to carry out a manual `updateprovisioners` command on the active head node, immediately after a regular node software image change has been made.

A successful run of the `updateprovisioners` command means that in the event of a failover, the formerly passive head node already has up-to-date regular node software images, which makes further administration simpler.

The background behind why it is done can be skipped, but it is as follows:

An image on the passive head node, which is a node with a provisioning role, is treated as an image on any other provisioning node. This means that it eventually synchronizes to a changed image on the active head node. By default the synchronization happens at midnight, which means images may remain out-of-date for up to 24 hours. The images being in an out-of-date state should be viewed as normal, because the timeout period associated with being in an updated state is only 5 minutes by default.

Since the passive head is a provisioning node, it also means that an attempt to provision regular nodes from it with the changed image will not succeed if it happens too soon after the image change event on the active head. "Too soon" means within the `autoupdate` period defined by the parameter `provisioningnodeautoupdatetimetype` (page 159).

If on the other hand the `autoupdate` timeout is exceeded, then by itself this does not lead to the image on the passive head node becoming synchronized with an image from the active head node. Such synchronization only takes place as part of regular housekeeping (at midnight by default). Or it takes place if a regular node sends a provisioning request to the passive head node, which can take place during the reboot of the regular node.

This synchronization logic is followed to reduce the load on the head node. The only pitfall in this is the case when an administrator changes an image on the active head node, and then soon

after that the passive head node becomes active as part of a failover, without the images having had enough time to synchronize. In that case the formerly passive node ends up with out-of-date software images. That is why it is a best practice to carry out a `manual updateprovisioners` command on the active head immediately after a regular node software image change has been made.

- Changes controlled by `CMDaemon` are synchronized automatically between the `CMDaemon` databases to the required extent during failover to the active node.

If the output of `cmha status` is not `OK`, then it typically means that the `CMDaemon` databases of the active head node and the passive head node are not synchronized. This situation may be resolved by waiting, typically for several minutes. If the status does not resolve on its own, then this indicates a more serious problem which should be investigated further.

- By default, software images are not stored on shared storage, and are synchronized between the head nodes by `CMDaemon`.

However, if images are kept on shared storage, then, within the provisioning role (section 5.2.1), the image-related parameters such as `allimages`, `localimages`, and `sharedimages`, must be adjusted according to the configuration used.

- If filesystem changes are made on an active head node without using `CMDaemon` (`cmsh` or Bright View), and if the changes are outside the shared filesystem, then these changes should normally also be made by the administrator on the passive head node. For example:
 - RPM installations/updates (section 11.2)
 - Applications installed locally
 - Files (such as drivers or values) placed in the `/cm/node-installer/` directory and referred to by `initialize` (section 5.4.5) and `finalize` scripts (section 5.4.11)
 - Any other configuration file changes outside of the shared filesystems

The reason behind not syncing everything automatically is to guarantee that a change that breaks a head node is not accidentally propagated to the passive. This way there is always a running head node. Otherwise, if automated syncing is used, there is a risk of ending up with two broken head nodes at the same time.

If the cluster is being built on bare metal, then a sensible way to minimize the amount of work to be done is to install a single head cluster first. All packages and applications should then be placed, updated and configured on that single head node until it is in a satisfactory state. Only then should HA be set up as described in section 14.2, 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 14.1.3 should also have their packages updated on both head nodes.

For the services that run simultaneously on the head nodes, such as CMDaemon, DHCP, LDAP, MySQL, NTP and DNS, their packages should be updated on both head nodes at about the same time. A suggested procedure is to stop the service on both nodes around the same time, update the service and ensure that it is restarted.

The provisioning node service is part of the CMDaemon package. The service updates images from the active head node to all provisioning nodes, including the passive head node, if the administrator runs the command to update provisioners. How to update provisioners is described in section 14.1.3.

For services that migrate across head nodes during failover, such as NFS, or the `sgemaster` it is recommended (but not mandated) to carry out this procedure: the package on the passive node (called the secondary for the sake of this example) is updated to check for any broken package behavior. The secondary is then made active with `cmha makeactive` (section 14.4.2), 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.

14.4.6 High Availability Parameters

There are several HA-related parameters that can be tuned. Accessing these via Bright View is described in section 14.4.6. In `cmsh` the settings can be accessed in the `failover` submode of the base partition.

Example

```
[mycluster1]% partition failover base
[mycluster1->partition[base]->failover]% show
Parameter                               Value
-----
Dead time                                10
Disable automatic failover               no
Failover network                          failovernet
Init dead                                 30
Keep alive                                1
Mount script
Postfailover script
Prefailover script
Quorum time                               60
Revision
Secondary headnode
Unmount script
Warn time                                  5
```

Dead time

When a passive head node determines that the active head node is not responding to any of the periodic checks for a period longer than the `Dead time` seconds, the active head node is considered dead and a quorum procedure starts. Depending on the outcome of the quorum, a failover sequence may be initiated.

Disable automatic failover

Setting this to yes disables automated failover. Section 14.1.7 covers this further.

Failover network

The `Failover network` setting determines which network is used as a dedicated network for the `failoverping` heartbeat check. The heartbeat connection is normally a direct cable from a NIC on one

head node to a NIC on the other head node. The network can be selected via tab-completion suggestions. By default, without a dedicated failover network, the possibilities are nothing, externalnet and internalnet.

Init dead

When head nodes are booted simultaneously, the standard `Dead time` might be too strict if one head node requires a bit more time for booting than the other. For this reason, when a head node boots (or more exactly, when the cluster management daemon is starting), a time of `Init dead` seconds is used rather than the `Dead time` to determine whether the other node is alive.

Keep alive

The `Keep alive` value is the time interval, in seconds, over which the passive head node carries out a check that the active head node is still up. If a dedicated failover network is used, 3 separate heartbeat checks are carried out to determine if a head node is reachable.

Mount script

The script pointed to by the `Mount script` setting is responsible for bringing up and mounting the shared filesystems.

Postfailover script

The script pointed to by the `Postfailover script` setting is run by `cmdaemon` on both head nodes. The script first runs on the head that is now passive, then on the head that is now active. It runs as soon as the former passive has become active. It is typically used by scripts mounting an NFS shared storage so that no more than one head node exports a filesystem to NFS clients at a time.

Prefailover script

The script pointed to by the `Prefailover script` setting is run by `cmdaemon` on both head nodes. The script first runs on the (still) active head, then on the (still) passive head. It runs as soon as the decision for the passive to become active has been made, but before the changes are implemented. It is typically used by scripts unmounting an NFS shared storage so that no more than one head node exports a filesystem to NFS clients at a time. When unmounting shared storage, it is very important to ensure that a non-zero exit code is returned if unmounting has problems, or the storage may become mounted twice during the `Postfailover script` stage, resulting in data corruption.

Quorum time

When a node is asked what head nodes it is able to reach over the network, the node has `Quorum time` seconds to respond. If a node does not respond to a call for quorum within that time, it is no longer considered for the results of the quorum check.

Secondary headnode

The `Secondary headnode` setting is used to define the secondary head node to the cluster.

Unmount script

The script pointed to by the `Unmount script` setting is responsible for bringing down and unmounting the shared filesystems.

Warn time

When a passive head node determines that the active head node is not responding to any of the periodic checks for a period longer than `Warn time` seconds, a warning is logged that the active head node might become unreachable soon.

14.4.7 Viewing Failover Via Bright View

Accessing `cmsh HA Parameters (partition failover base) Via Bright View`

The Bright View equivalents of the `cmsh HA parameters` in section 14.4.6 are accessed from the clickpath `Cluster→Partition[base]→Settings→Failover`

14.4.8 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 14.2.2. Instead of running the `cm-clone-install --failover` command as in that section, the following command can be run:

```
[root@bright90 ~]# /cm/cm-clone-install --clone --hostname=<new host name>
```

The new host name can be the same as the original, because the clone is not run at the same time as the original anyway. The clone should not be run after cloning on the same network segment as the original, in order to prevent IP address conflicts.

If the clone is merely intended as a backup, then the clone hardware does not have to match the head node. For a backup, typically the most important requirement is then that a clone drive should not run out of space—that is, its drive should be as large as, or larger than the matching drive of the head node.

If the clone is to be put to work as a head node, then, if the MAC address of one of the head nodes has changed, it is typically necessary to request that the product key is unlocked, so that a new license can be obtained (section 4.3 of the *Installation Manual*).

Also, for a clone that is to be put to work as a head node, the CMDaemon database should first be synchronized from the active head node to the clone. This can be done by running `cmha dbreclone` on the active head node (page 635) before carrying out tasks with `cmsh` or Bright View.

Exclude Lists And Cloning

Some files are normally excluded from being copied across from the head node to the clone, because syncing them is not appropriate.

The following exclude files are read from inside the directory `/cm/` on the clone node when the `cm-clone-install` command is run (step 4 in section 14.2.2).

- `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 Bright View or `cmsh` options, and are maintained by `CMDaemon`. On the other hand, the exclude lists introduced in this section (14.4.8):

- `excludelistfailover`
- `excludelistnormal`

are not Bright View or `cmsh` options, are not modified with `excludelistmanipulatescript`, are not maintained by `CMDaemon`, but are made use of when running the `cm-clone-install` command.

Btrfs And `cm-clone-install`

If a partition with Btrfs (section 13.4.1) is being cloned using `cm-clone-install`, then by default only mounted snapshots are cloned.

If all the snapshots are to be cloned, then the `--btrfs-full-clone` flag should be passed to the `cm-clone-install` command. This flag clones all the snapshots, but it is carried out with duplication (bypassing the COW method), which means the filesystem size can increase greatly.

14.5 HA For Regular Nodes

HA for regular nodes is available from Bright Cluster Manager version 7.0 onwards.

14.5.1 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.

14.5.2 Comparing Head And Regular Node HA

Many of the features of HA for regular nodes are as for head nodes. These include:

Head And Regular Node HA: Some Features In Common

Power control is needed for all HA nodes, in order to carry out automatic failover (section 14.1.7).

Warn time and dead time parameters can be set (section 14.4.6).

Mount and unmount scripts (page 620).

Pre- and post- failover scripts (section 14.4.6).

Disabling or enabling automatic failover (section 14.1.7).

A virtual shared IP address that is presented as the virtual node that is always up (section 14.1.4).

Some differences between head and regular node HA are:

Head And Regular Node HA: Some Features That Differ

Head Node HA

Only one passive node.

Can use the optional failover network and failoverping heartbeat.

Configured within failover submode of partition mode (section 14.4.6).

Installed with `cmha-setup` (section 14.2).

A quorum procedure (section 14.1.6). If more than half the nodes can only connect to the passive, then the passive powers off the active and becomes the new active.

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 14.5.3).

Installed by administrator using a procedure similar to section 14.5.3.

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.

14.5.3 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

```
[bright90->partition[base]->failovergroups]% status
No active failover groups
[bright90->partition[base]->failovergroups]% add cupsgroup
[bright90->partition*[base*]->failovergroups*[cupsgroup*]]% list
Name (key)          Nodes
-----
cupsgroup
```

By default, CUPS is provided in the standard image, in a stopped state. In Bright View a failover group can be added via the clickpath `Cluster`→`Partition[base]`→`Settings`→`Failover groups`→`Add`

Adding Nodes To The Failover Group

Regular nodes can then be added to a failover group. On adding, 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

```
[bright90->...[cupsgroup*]]% set nodes node001..node002
[bright90->...[cupsgroup*]]% commit
```

```
[bright90->...[cupsgroup]]%
...Failover group cupsgroup, make node001 become active
...Failover group cupsgroup, failover complete. node001 became active
[bright90->partition[base]->failovergroups[cupsgroup]]%
```

Setting Up A Server For The Failover Group

The CUPS server needs to be configured to run as a service on all the failover group nodes. The usual way to configure the service is to set it to run only if the node is active, and to be in a stopped state if the node is passive:

Example

```
[bright90->partition[base]->failovergroups[cupsgroup]]% device
[bright90->device]% foreach -n node001..node002 (services; add cups; \
set runif active; set autostart yes; set monitored yes)
[bright90->device]% commit
Successfully committed 2 Devices
[bright90->device]%
Mon Apr 7 08:45:54 2014 [notice] node001: Service cups was started
```

The runif options are described in section 3.11.1.

Setting And Viewing Parameters And Status In The Failover Group

Knowing which node is active: The status command shows a summary of the various failover groups in the failovergroups submodule, including which node in each group is currently the active one:

Example

```
[bright90->partition[base]->failovergroups]% status
Name      State      Active      Nodes
-----
cupsgroup ok         node001     node001,node002 [ UP ]
```

Making a node active: To set a particular node to be active, the makeactive command can be used from within the failover group:

Example

```
[bright90->partition[base]->failovergroups]% use cupsgroup
[bright90->...]->failovergroups[cupsgroup]]% makeactive node002
node002 becoming active ...
[bright90->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 14.4.2):

Example

```
[root@bright90 ~]# 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

```
[bright90->partition[base]->failovergroups]% show cupsgroup
Parameter                                         Value
-----
Automatic failover after graceful shutdown      no
Dead time                                       10
Disable automatic failover                     no
Mount script
Name                                           cupsgroup
Nodes                                          node001,node002
Postfailover script
Prefailover script
Revision
Unmount script
Warn time                                       5
```

Setting Up The Virtual Interface To Make The Server An HA Service

The administrator then assigns each node in the failover group the same alias interface name and IP address dotted quad on its physical interface. The alias interface for each node should be assigned to start up if the node becomes active.

Example

```
[bright90->device]% foreach -n node001..node002 (interfaces; add alias \
bootif:0 ; set ip 10.141.240.1; set startif active; set network internalnet)
[bright90->device*]% commit
Successfully committed 2 Devices
[bright90->device]% foreach -n node001..node002 (interfaces; list)
Type      Network device name  IP           Network
-----
alias     BOOTIF:0             10.141.240.1 internalnet
physical  BOOTIF [prov]       10.141.0.1   internalnet
Type      Network device name  IP           Network
-----
alias     BOOTIF:0             10.141.240.1 internalnet
physical  BOOTIF [prov]       10.141.0.2   internalnet
```

Optionally, each alias node interface can conveniently be assigned a common arbitrary additional host name, perhaps associated with the server, which is CUPS. This does not result in duplicate names here because only one alias interface is active at a time. Setting different additional hostnames for the alias interface to be associated with a unique virtual IP address is not recommended.

Example

```
[bright90->...interfaces*[BOOTIF:0*]]% set additionalhostnames cups
[bright90->...interfaces*[BOOTIF:0*]]% commit
```

The preceding can also simply be included as part of the set commands in the foreach statement earlier when the interface was created.

The nodes in the failover group should then be rebooted.

Only the virtual IP address should be used to access the service when using it as a service. Other IP addresses may be used to access the nodes that are in the failover group for other purposes, such as monitoring or direct access.

Service Configuration Adjustments

A service typically needs to have some modifications in its configuration done to serve the needs of the cluster.

CUPS uses port 631 for its service and by default it is only accessible to the local host. Its default configuration is modified by changing some directives within the `cupsd.conf` file. For example, some of the lines in the default file may be:

```
# Only listen for connections from the local machine.
Listen localhost:631
...
# Show shared printers on the local network.
...
BrowseLocalProtocols

...
<Location />
  # Restrict access to the server...
  Order allow,deny

</Location>
...
```

Corresponding lines in a modified `cupsd.conf` file that accepts printing from hosts on the internal network could be modified and end up looking like:

```
# Allow remote access
Port 631
...
# Enable printer sharing and shared printers.
...
BrowseAddress @LOCAL
BrowseLocalProtocols CUPS dnssd
...
<Location />
  # Allow shared printing...
  Order allow,deny
  Allow from 10.141.0.0/16
</Location>
...
```

The operating system that ends up on the failover group nodes should have the relevant service modifications running on those nodes after these nodes are up. In general, the required service modifications could be done:

- with an `initialize` or `finalize` script, as suggested for minor modifications in section 3.15.4
- by saving and using a new image with the modifications, as suggested for greater modifications in section 3.15.2, page 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 14.2.4.

Example

```
ssh node001
echo c > /proc/sysrq-trigger
~.
```

A passive node then takes over.

14.5.4 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 645):

Sequence Of Events In Making Another HA Regular Node Active	
All	Run pre-failover script
Active	Stop service Run umount script (stop and show <code>exit>0</code> 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.

14.6 HA And Workload Manager Jobs

Workload manager jobs continue to run through a failover handover if conditions allow it.

The 3 conditions that must be satisfied are:

1. The workload manager setup must have been carried out
 - (a) during initial installation
 - or
 - (b) during a run of `cm-wlm-setup`
2. The HA storage setup must support the possibility of job continuity for that workload manager. This support is possible for the workload manager and HA storage pairings indicated by the following table:

Table 14.6: HA Support For Jobs Vs Shared Filesystems

WLM	DRBD	DAS	NAS
Slurm	Y	Y	Y
OGS	Y	Y	Y
UGE	Y	Y	Y
PBSP _{Pro}	N	N	Y
LSF	N	N	Y

As the table shows, PBS Pro, and LSF 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.

- 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.

15

The Jupyter Notebook Environment Integration

15.1 Introduction

This chapter covers the installation and usage of the Jupyter Notebook environment in Bright Cluster Manager.

An updated list of the supported Linux distributions and Jupyter functionalities can be found in the feature matrix at <https://support.brightcomputing.com/feature-matrix/>, under the Feature column, in the section for Jupyter features.

An overview of the concepts and terminology follows.

What Is Jupyter Notebook?

Jupyter Notebook (<https://jupyter-notebook.readthedocs.io/>), or Jupyter, is a client-server open-source application that provides a convenient way for a cluster user to write and execute *notebook documents* in an interactive environment.

In Jupyter, a notebook document, or notebook, is content that can be managed by the application. Notebooks are organized in units called *cells* and can contain both executable code, as well as items that are not meant for execution.

Items not meant for execution can be, for example: explanatory text, figures, formulas, or tables. Notebooks can also store the inputs and outputs of an interactive session.

Notebooks can thus serve as a complete record of a user session, interleaving code with rich representations of resulting objects.

These documents are encoded as JSON files and saved with the `.ipynb` extension. Since JSON is a plain text format, notebooks can be version-controlled, shared with other users and exported to other formats, such as HTML, \LaTeX , PDF, and slide shows.

What Is A Notebook Kernel?

A *notebook kernel* (often shortened to *kernel*) is a computational engine that handles the various types of requests in a notebook (e.g. code execution, code completions, inspection) and provides replies to the user (<https://jupyter.readthedocs.io/en/latest/projects/kernels.html>). Usually kernels only allow execution of a single language. There are kernels available for many languages, of varying quality and features.

What Is JupyterHub?

Jupyter on its own provides a single user service. *JupyterHub* (<https://jupyterhub.readthedocs.io/>) allows Jupyter to provide a multi-user service, and is therefore commonly installed with it. JupyterHub is an open-source project that supports a number of authentication protocols, and can be configured in order to provide access to a subset of users.

What Is JupyterLab?

JupyterLab (<https://jupyterlab.readthedocs.io/>) is a modern and powerful interface for Jupyter. It enables users to work with notebooks and other applications, such as terminals or file browsers. It is open-source, flexible, integrated, and extensible.

JupyterLab works out of the box with JupyterHub. It can be used to arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning.

JupyterLab is extensible with plugins that can customize or enhance any part of the interface. Plugins exist for themes, file editors, keyboard shortcuts, as well as for other components.

What Is A Jupyter Extension?

Several components of the Jupyter environment can be customized in different ways with extensions. Some types of extensions are:

- IPython extensions (<https://ipython.readthedocs.io/en/stable/config/extensions/#ipython-extensions>)
- Jupyter Notebook server extensions (<https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Distributing%20Jupyter%20Extensions%20as%20Python%20Packages.html>)
- JupyterLab extensions (<https://jupyterlab.readthedocs.io/en/stable/user/extensions.html>)

Extensions are usually developed, bundled, released, installed, and enabled in different ways.

Each extension provides a new functionality for a specific component. For example, JupyterLab extensions can customize or enhance any part of the JupyterLab user interface. Extensions can provide new themes, file viewers, editors and renderers for rich output in notebooks. They can also add settings, add keyboard shortcuts, or add items to the menu or command palette.

What Is Jupyter Enterprise Gateway?

By default, Jupyter runs kernels locally, which can exhaust server resources. A resource manager can be used to deal with this issue. *Jupyter Enterprise Gateway* (<https://jupyter-enterprise-gateway.readthedocs.io/>) (JEG) is a pluggable open-source framework that leverages local underlying resource managers, such as Slurm or Kubernetes, to distribute kernels across the compute cluster. In addition to scalability, JEG also provides an improved multi-user support, and more granular security for Jupyter. This makes Jupyter more useful for enterprise, scientific, and academic implementations.

In Bright Cluster Manager, all the technologies mentioned in these sections are combined to provide a powerful, customizable and user-friendly Jupyter Notebook web interface running on a lightweight, multi-tenant, multi-language, scalable and secure environment, ready for a wide range of enterprise scenarios.

For convenience, in the following sections, *Jupyter* is generally used to collectively refer to Jupyter Notebook, JupyterHub, JupyterLab and Jupyter Enterprise Gateway.

15.2 Jupyter Environment Installation

Bright distributes Jupyter via two packages: `cm-jupyter` and `cm-jupyter-local`.

`cm-jupyter` is installed in the `/cm/shared` directory, which is by default exported over NFS. As a result, Jupyter kernels can run on all the compute nodes, without a separate installation to those nodes. `cm-jupyter` provides Jupyter Notebook, JupyterHub, JupyterLab, Jupyter Enterprise Gateway and some extensions.

`cm-jupyter-local` provides the JupyterHub system service (`cm-jupyterhub.service`), and is therefore designed to be installed only on the node exposing users to the web login page for Jupyter. For

convenience, this node is called the *login node*. A login node is typically the head node, but any cluster node can be used.

Since compute nodes are not reachable via a web interface by default, it is the responsibility of the cluster administrator to configure access to these nodes if they are configured to be login nodes while Jupyter runs. That is, login nodes that are compute nodes must have their access configured by assigning IP addresses, configuring the firewall, opening Jupyter ports, and so on. However, if the Jupyter login node is the head node, then Bright Cluster Manager takes care of configuring the firewall to open the required ports and of ensuring that the resulting environment is working out of the box.

Bright Jupyter Extensions

For a default deployment of Jupyter, Bright installs and enables the following extensions to the Jupyter environment:

- Jupyter Addons: A Jupyter Notebook server extension that performs API calls to CMDaemon and manages other server extensions;
- Jupyter Kernel Creator (section 15.5): A Jupyter Notebook server extension that provides a new interactive and user-friendly way to create kernels;
- Jupyter VNC (section 15.7): A Jupyter Notebook server extension that enables remote desktops with VNC from notebooks;
- JupyterLab Tools: A JupyterLab extension that exposes Bright server extensions functionalities to the users and shows the Cluster View section;
- Jupyter WLM Magic (section 15.8): An IPython extension that simplifies scheduling of workload manager jobs from the notebook.

15.2.1 Jupyter Setup

The `cm-jupyter-setup` script can be run on the head node of the cluster to deploy a working Jupyter environment with minimal effort. The script comes with Bright Cluster Manager's `cm-setup` package. It has no prerequisites, and can be run before or after configuring any resource manager, such as Kubernetes or Slurm.

The `cm-jupyter-setup` script installs the two packages `cm-jupyter` and `cm-jupyter-local`. By default, the Jupyter environment initially contains only Jupyter's default Python 3 kernel, which runs on the login node.

15.2.2 Jupyter Architecture

The default Jupyter architecture deployed by `cm-jupyter-setup` is shown in figure 15.1.

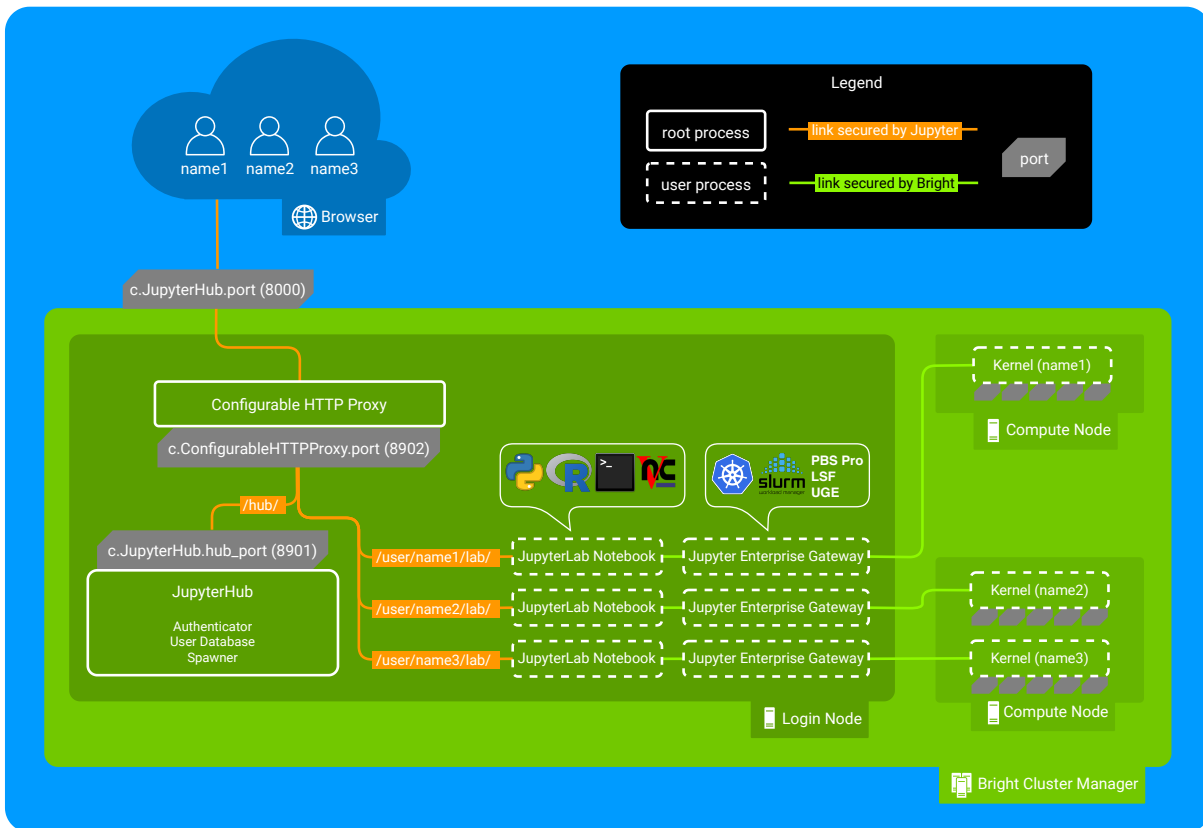


Figure 15.1: Jupyter architecture

In this architecture, the `cm-jupyterhub.service` provided by `cm-jupyter-local` starts JupyterHub on the port specified by `c.JupyterHub.hub_port` (default: 8901) on a login node (typically the head node).

JupyterHub then automatically spawns a dynamic proxy to route HTTP requests via Bright's `cm-npm-configurable-http-proxy` package.

The proxy is the only process that listens for clients' requests on Jupyter's public interface, as specified by `c.JupyterHub.port` (default: 8000).

JupyterHub instructs the proxy on how requests should be dispatched by using its REST API. This is exposed at the port specified by `c.ConfigurableHTTPProxy.port` (default: 8902), typically defined within `c.ConfigurableHTTPProxy.api_url`.

At this stage, users accessing Jupyter with their browsers are either redirected to the shared Hub (e.g. for authentication), or to their dedicated single-user servers to run notebooks.

By default, single-user servers are accessed at `/user/<username>`, while JupyterHub is accessed at `/hub` (<https://jupyterhub.readthedocs.io/en/stable/reference/technical-overview.html>).

The `cm-jupyter-setup` script automatically installs JupyterLab and sets `/lab` as the default URL (`c.Spawner.default_url`) to redirect users to the new interface.

Finally, Jupyter Enterprise Gateway is integrated to spawn kernels on different nodes by configuring `jupyterhub.spawner.LocalProcessSpawner` as the default spawning mechanism (`c.JupyterHub.spawner_class`), and `jupyterhub-singleuser-gw` as the default spawning command (`c.Spawner.cmd`). When a new notebook is started, JEG scans for an available port, and spawns a kernel chosen by the user on an appropriate cluster node. This process is then connected to JupyterLab.

JupyterHub and its HTTP proxy are run as two root processes, while JupyterLab and Jupyter Enterprise Gateway are run as user processes.

The privileges of the kernels spawned by Jupyter Enterprise Gateway can be configured by cluster administrators, and depend on the underlying computational engine (e.g. Kubernetes). By default, all

the kernels configurable by Bright are run as user processes, and no privilege escalation is possible.

Communication between users' browsers, JupyterHub, its HTTP proxy, and JupyterLab, is secured by default by JupyterHub. On the other hand, communication between JupyterLab, Jupyter Enterprise Gateway, and kernels, is secured by Bright.

Administrators can customize their Jupyter integration by setting some of the aforementioned options when the `cm-jupyter-setup` script runs. New values are automatically handled by Bright Cluster Manager and written to Jupyter configuration files.

Other configuration options can be found in `/cm/local/apps/jupyter/current/conf/jupyterhub_config.py`.

15.2.3 Verifying Jupyter Installation

The `cm-jupyter-setup` script automatically starts the `cm-jupyterhub` service.

Any user (not necessarily root) can then verify the installation is working as expected. Here an ordinary user, `jupyterhubuser` runs the checks.

It can take some time until the service is fully up and running, even if `systemctl status cm-jupyterhub -l` shows that the service is active:

Example

```
[jupyterhubuser@bright90 ~]$ systemctl status cm-jupyterhub -l

cm-jupyterhub.service - JupyterHub
  Loaded: loaded (/usr/lib/systemd/system/cm-jupyterhub.service; static; vendor preset: disabled)
  Active: active (running) since Fri 2021-07-02 11:58:49 CEST; 2h 42min ago
  Main PID: 6011 (run.sh)
  Tasks: 40 (limit: 23379)
  Memory: 267.6M
  CGroup: /system.slice/cm-jupyterhub.service
          |- 6011 /bin/bash /cm/shared/apps/jupyter/12.1.0/bin/run.sh
          |- 6025 /cm/local/apps/python37/bin/python3 /cm/shared/apps/jupyter/12.1.0/bin/jupyterhub --Jupyter>
          |- 6026 tee -a /var/log/jupyterhub.log
          |- 6035 node /cm/shared/apps/jupyter/12.1.0/bin/configurable-http-proxy --ip --port 8000 --api-ip 1>
          |-71482 /bin/bash /cm/shared/apps/jupyter/12.1.0/bin/jupyterhub-singleuser-gw --port=48913 --Single>
          |-71720 /cm/local/apps/python37/bin/python3 /cm/shared/apps/jupyter/12.1.0/bin/jupyter-enterprisega>
          '-71921 /cm/local/apps/python37/bin/python3 /cm/shared/apps/jupyter/12.1.0/bin/jupyter-labhub --con>
```

A check can then be done to see that the Jupyter extensions provided by Bright are installed and enabled:

Example

```
[jupyterhubuser@bright90 ~]$ module load jupyter
Loading jupyter/12.1.0
  Loading requirement: python37
[root@bright90 ~]# jupyter serverextension list
config dir: /cm/shared/apps/jupyter/12.1.0/etc/jupyter
  jupyterlab enabled
  - Validating...
    jupyterlab 3.0.16 OK
  cm_jupyter_kernel_creator enabled
  - Validating...
    cm_jupyter_kernel_creator 1.1.dev113+g0c6b26b OK
  cm_jupyter_addons enabled
  - Validating...
    cm_jupyter_addons 1.1.dev13+g2ba18eb OK
```

```

cm_jupyter_vnc enabled
- Validating...
  cm_jupyter_vnc 1.1.dev50+g11276cf OK
[jupyterhubuser@bright90 ~]$ jupyter labextension list
JupyterLab v3.0.16
/cm/shared/apps/jupyter/12.1.0/share/jupyter/labextensions
  @brightcomputing/jupyterlab-tools v0.2.7 enabled OK (python, brightcomputing_jupyterlab_tools)

```

15.2.4 Login Configuration

User Access To JupyterHub With The Default Configuration

Once functioning, the Jupyter web interface is accessible with a browser using the HTTPS protocol on the specified port (figure 15.2):

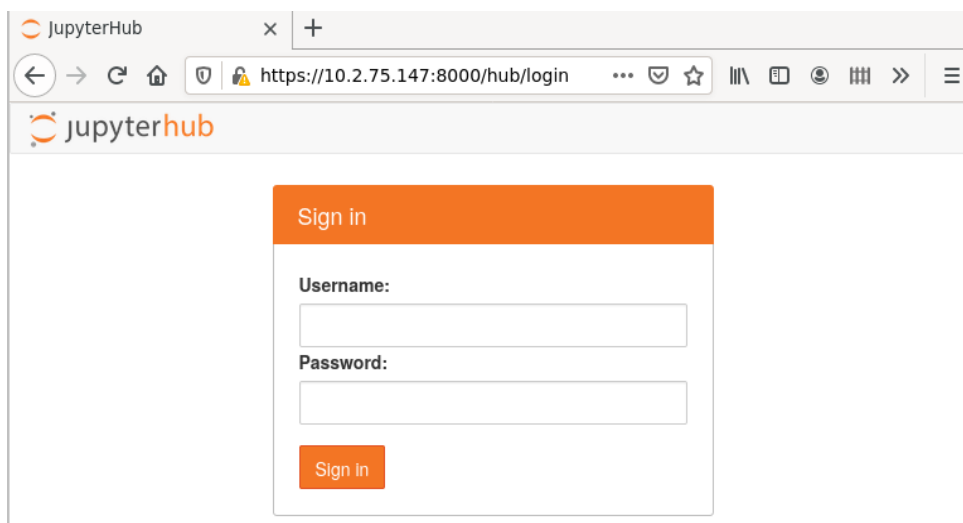


Figure 15.2: JupyterHub login screen

Running as root under JupyterHub is not recommended, so logging in to JupyterHub as root is not allowed by default.

Any other PAM user can log in to JupyterHub.

If needed, a test user `jupyterhubuser` with password `jupyterhubuser` can be created with, for example:

Example

```
[root@bright90 ~]# cmsh -c "user; add jupyterhubuser; set password jupyterhubuser; commit"
```

Restricting User Access To JupyterHub

JupyterHub logins can be limited to members of particular groups.

For example, `alice` could be made a member of the group `jupyterusers`:

Example

```

[root@bright90 ~]# cmsh
[bright90]% group
[bright90->group]% add jupyterusers
[bright90->group]*[jupyterusers*]% append members alice
[bright90->group]*[jupyterusers*]% commit

```

The group `jupyterusers` could then be set to be allowed to authenticate to JupyterHub with an authentication policy:

Example

```
[root@bright90 ~]# cmsht
[bright90]% configurationoverlay
[bright90->configurationoverlay]% use jupyterhub
[bright90->configurationoverlay[jupyterhub]]% roles
[bright90->configurationoverlay[jupyterhub]->roles]% use jupyterhub
[bright90->configurationoverlay[jupyterhub]->roles[jupyterhub]]% configs
[bright90->...roles[jupyterhub]->configs]% add c.BrightAuthenticator.groups_allow
[bright90->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.groups_allow*]]% set value ["jupyterusers\"]
[bright90->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.groups_allow*]]% commit
```

Similarly, restricting authentication to JupyterHub to a specific group of users can be done by configuring `c.BrightAuthenticator.groups_deny`.

Another way to restrict user access is based on the CMDaemon profile set for the user. In that case the JupyterHub configuration settings are `c.BrightAuthenticator.cmd_profiles_allow` and `c.BrightAuthenticator.cmd_profiles_deny`.

In the following example, users with a `readonly` profile are not allowed to authenticate for JupyterHub.

Example

```
[bright90->...roles[jupyterhub]->configs[c.BrightAuthenticator.groups_allow]]% ..
[bright90->...roles[jupyterhub]->configs]% add c.BrightAuthenticator.cmd_profiles_deny]]%
[bright90->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.cmd_profiles_deny*]]% set value ["readonly\"]
[bright90->...roles*[jupyterhub*]->configs*[c.BrightAuthenticator.cmd_profiles_deny*]]% commit
```

A table summarizing the authentication policies is:

Key	Value
<code>c.BrightAuthenticator.groups_allow</code>	users in the specified groups can authenticate
<code>c.BrightAuthenticator.groups_deny</code>	users in the specified groups cannot authenticate
<code>c.BrightAuthenticator.cmd_profiles_allow</code>	users with the specified profiles can authenticate
<code>c.BrightAuthenticator.cmd_profiles_deny</code>	users with the specified profiles cannot authenticate

15.2.5 JupyterHub Screen After Login

After the first login, a new single-user server is spawned (figure 15.3):

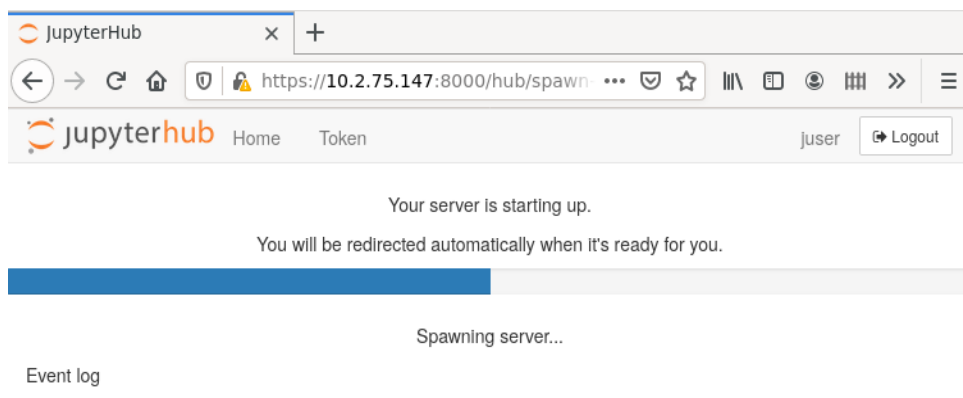


Figure 15.3: JupyterHub starting single-user server

Users are redirected to the JupyterLab interface and have access to Jupyter's default Python 3 kernel (figure 15.4):

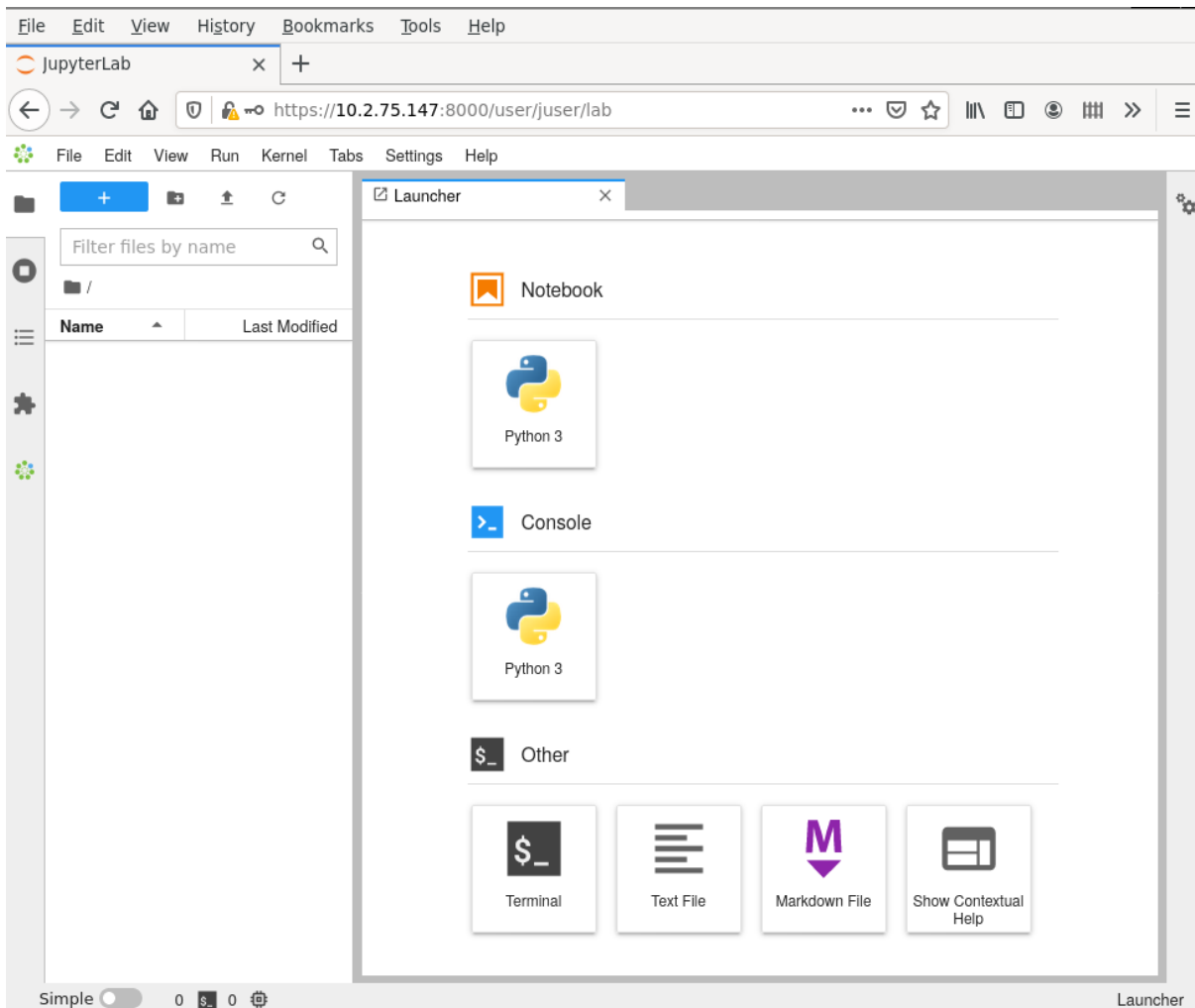


Figure 15.4: JupyterLab Launcher

If using Kubernetes under Jupyter, then a user registered under the Linux-PAM system must be added separately via Kubernetes with `cm-kubernetes-setup` (section 15.5.1).

15.3 Jupyter Notebook Examples

The `cm-jupyter` package (section 15.2.1) provides a number of machine learning notebook examples that can be executed with Jupyter.

The notebooks include some applications developed with TensorFlow, PyTorch, MXNet, and other frameworks. The applications can be found in the `/cm/shared/examples/jupyter/notebooks/` directory:

```
[jupyterhubuser@bright90 ~]$ ls /cm/shared/examples/jupyter/notebooks/
Keras+TensorFlow2-addition.ipynb  R-iris.ipynb          TensorFlow-minigo.ipynb
MXNet-superresolution.ipynb       Spark-pipeline.ipynb
Pytorch-cartpole.ipynb           Spark+XGBoost-mortgage.ipynb
```

The datasets needed to execute these notebooks can be found in the `/cm/shared/examples/jupyter/datasets/` directory:

```
[jupyterhubuser@bright90 ~]$ ls /cm/shared/examples/jupyter/datasets/
880f8b8a6fd-mortgage-small.tar.gz  kaggle-iris.csv
```

Users can copy these examples to their home directories, create or choose appropriate kernels to execute them, and interactively run them from Jupyter. In order to edit notebooks, the write permissions must be kept during the copy.

The distributed examples typically only require the packages provided by Bright with the Data Science Add-on, such as TensorFlow, PyTorch, and MXNet.

It is the responsibility of users to make sure that the required modules are loaded by their Jupyter kernels. The list of frameworks and libraries required to run an example is usually available at the beginning of each notebook.

15.4 Jupyter Kernels

In Jupyter, kernels are defined as JSON files.

Any user that the cluster administrator has registered in the Linux-PAM system can list the available Jupyter kernels via the command line. The following example is run in the initial Jupyter environment:

Example

```
[jupyterhubuser@bright90 ~]$ module load jupyter
Loading jupyter/12.0.0
  Loading requirement: python37
[jupyterhubuser@bright90 ~]$ jupyter kernelspec list
Available kernels:
  python3      /cm/shared/apps/jupyter/12.0.0/share/jupyter/kernels/python3
```

Each kernel directory contains a kernel .json file describing how Jupyter spawns that kernel:

Example

```
[jupyterhubuser@bright90 ~]$ ls /cm/shared/apps/jupyter/current/share/jupyter/kernels/*/kernel.json
/cm/shared/apps/jupyter/current/share/jupyter/kernels/python3/kernel.json
```

In addition to specifications for shared kernels, each user can define new personal ones in the home directory. By default, the Jupyter data directory for a user is located at `$HOME/.local/share/jupyter`.

This path can be verified with Jupyter by using the `--paths` option:

Example

```
[jupyterhubuser@bright90 ~]$ jupyter --paths
config:
  /home/jupyterhubuser/.jupyter
  /cm/local/apps/python37/etc/jupyter
  /cm/shared/apps/jupyter/12.0.0/etc/jupyter
  /etc/jupyter
data:
  /home/jupyterhubuser/.local/share/jupyter
  /cm/local/apps/python37/share/jupyter
  /cm/shared/apps/jupyter/12.0.0/share/jupyter
  /usr/share/jupyter
runtime:
  /home/jupyterhubuser/.local/share/jupyter/runtime
```

The simplest definition for a Python3 kernel designed to run on the login node is:

```
{
  "argv": ["python",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
  ],
  "display_name": "Python 3",
  "language": "python"
}
```

In the preceding kernel definition:

- `argv`: is the command to be executed to locally spawn the kernel
- `"display_name"`: is the name to be displayed in the JupyterLab interface
- `"language"`: is the supported programming language ("language")
- `"{connection_file}"` (<https://jupyter-client.readthedocs.io/en/stable/kernels.html#connection-files>) is a placeholder, and is replaced by Jupyter with the actual path to the connection file before starting the kernel.

The following kernel is Jupyter's default Python 3 kernel distributed by Bright in the initial environment:

Example

```
[jupyterhubuser@bright90 ~]$ cat /cm/shared/apps/jupyter/current/share/jupyter/kernels/python3/kernel.json
```

```
{
  "argv": [
    "/cm/local/apps/python37/bin/python3.7",
    "-m",
    "ipykernel_launcher",
    "--InteractiveShell.extra_extension=cm_jupyter_wlm_magic",
    "--TerminalIPythonApp.extra_extension=cm_jupyter_wlm_magic",
    "-f",
    "{connection_file}"
  ],
  "display_name": "Python 3",
  "language": "python",
  "env": {
    "PYTHONPATH": "/cm/shared/apps/jupyter/current/lib64/python3.7/site-packages:/cm/shared/apps/jupyter/current/lib/python3.7/site-packages"
  }
}
```

The two kernels are not very different. They differ from each other in the Python 3 binary path, the IPython extension (Jupyter WLM Magic), and the exported PYTHONPATH environment variable ("env").

15.4.1 Jupyter Enterprise Gateway Kernels

Jupyter is designed to run both the kernel processes, as well as the user interface (JupyterLab or Jupyter Notebook) on the same host. The kernel `{connection_file}` is therefore stored in the `~/.local/share/jupyter/runtime` directory, or in the `/run` directory.

JupyterLab can delegate the task of spawning kernels to another component. In the Bright Cluster Manager installation it is Jupyter Enterprise Gateway (JEG) that takes on this role, performing a complete lifecycle management of the kernel. That is: start, status monitoring, and termination.

JEG requires an extended `kernel.json` definition to describe a particular process-proxy module to handle the kernel.

A simple definition for a Python3 kernel designed to be scheduled via JEG is:

```
{
  "argv": ["python",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
  ],
  "display_name": "Python 3",
  "language": "python",
  "metadata": {
    "process_proxy": {
      "class_name": "enterprise_gateway.services.processproxies.processproxy.LocalProcessProxy",
      "config": {}
    }
  }
}
```

In this example, the "metadata" entry has been added. It includes "class_name", which defines a path to a Python class compatible with the JEG API; and "config", an initial arguments dictionary passed to the JEG process.

JEG can use several pluggable modules that are included in the Bright environment to interact with a wide range of resource managers, such as Kubernetes or Slurm. By doing so, kernels can be scheduled across compute nodes.

Bright recommends that JEG kernels are created and used with the Jupyter Kernel Creator (section 15.5) extension.

15.5 Jupyter Kernel Creator Extension

Creating or editing kernels can be cumbersome and error-prone for users, depending on the features of the execution context desired for their notebooks.

To provide a more user-friendly experience, Bright includes the *Jupyter Kernel Creator* extension in JupyterLab. This extension is accessed from the navigation pane in the JupyterLab interface, by clicking on the Bright icon.

Jupyter Kernel Creator allows users to create kernels using the JupyterLab interface, without the need to directly edit JSON files. With this interface users can create kernels by customizing an available *template* according to their needs.

A template can be considered to be the skeleton of a kernel, with several preconfigured options, and others options that are yet to be specified. Common customizations for templates include environment modules to be loaded, workload manager queues to be used, number and type of GPUs to acquire, and so on.

Templates are usually defined by administrators according to cluster capabilities, programming languages and user requirements. Each template can provide different options for customizations.

Administrators often create different templates to take advantage of different workload managers, programming languages and hardware resources. For example, an administrator may define a template for scheduling Python kernels via Kubernetes, another one for R kernels via Slurm, and yet another one for Bash kernels via Platform LSF.

To simplify Jupyter configuration for administrators, Bright distributes a number of pre-defined templates with Jupyter Kernel Creator. These templates can be used for default configurations of Bright workload managers, and can be customized and extended for more advanced use.

Kernel templates defined by Bright can be found in the Jupyter Kernel Creator installation directory:

```
[jupyterhubuser@bright90 ~]$ ls /cm/shared/apps/jupyter/current/lib/python3.7/site-packages/\
cm_jupyter_kernel_creator/kerneltemplates/
jupyter-eg-kernel-k8s-bash      jupyter-eg-kernel-lsf-bash      jupyter-eg-kernel-slurm-bash
jupyter-eg-kernel-k8s-py      jupyter-eg-kernel-lsf-py37      jupyter-eg-kernel-slurm-py37
jupyter-eg-kernel-k8s-py-spark jupyter-eg-kernel-pbspro-bash   jupyter-eg-kernel-uge-bash
jupyter-eg-kernel-k8s-r       jupyter-eg-kernel-pbspro-py37   jupyter-eg-kernel-uge-py37
```

Users can view available templates in the Jupyter web browser interface, within the `KERNEL TEMPLATES` section of the dedicated Bright extensions panel (figure 15.5):

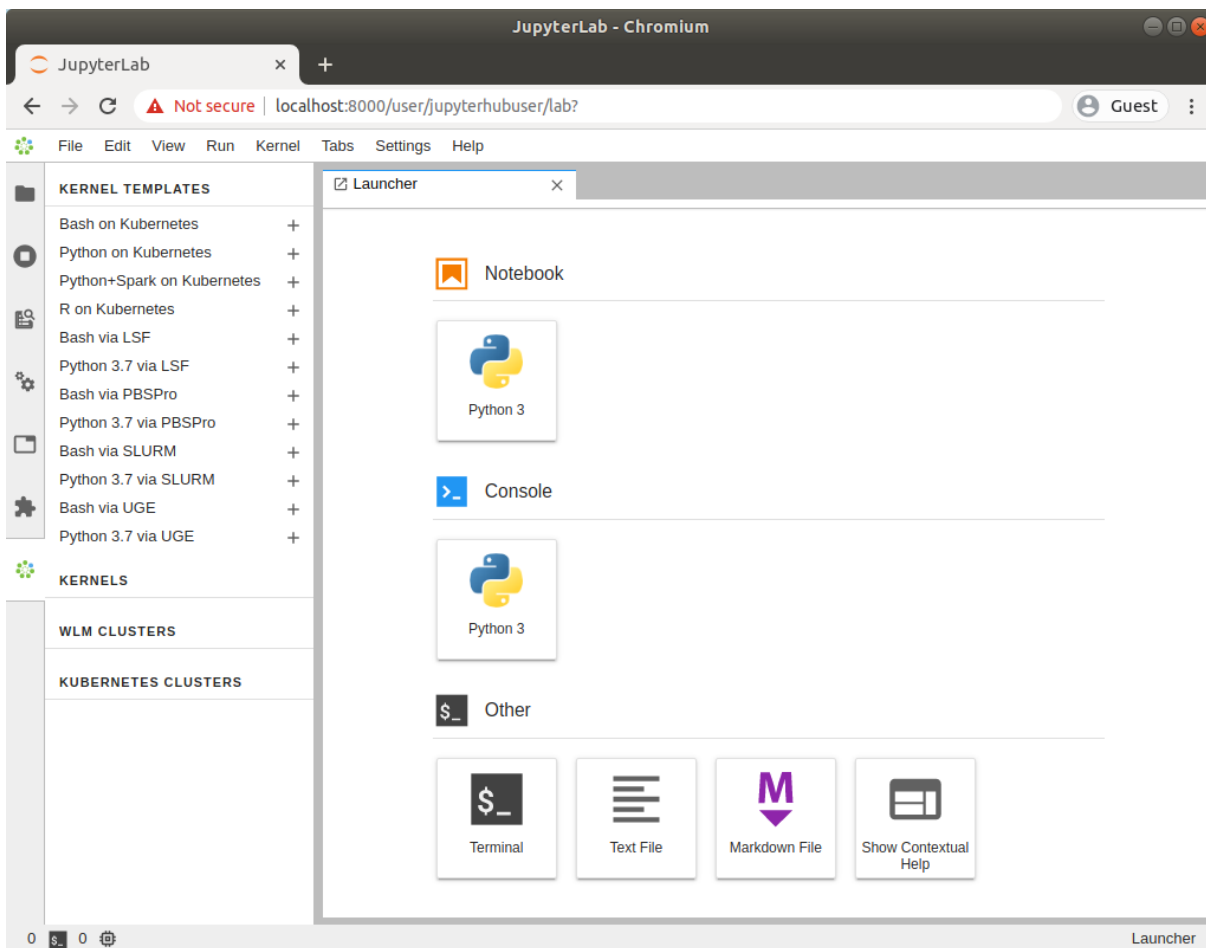


Figure 15.5: JupyterLab Bright extensions section with kernel templates

Templates provided by Bright are not listed in the panel, if no associated workload manager instance or if no Kubernetes instance has been deployed on the cluster.

Users can instantiate a template to create an actual kernel from the dedicated Bright extensions section (+ button).

A dialog is dynamically generated for the template being instantiated, and users are asked to fill a number of customization options defined by administrators (figure 15.6):

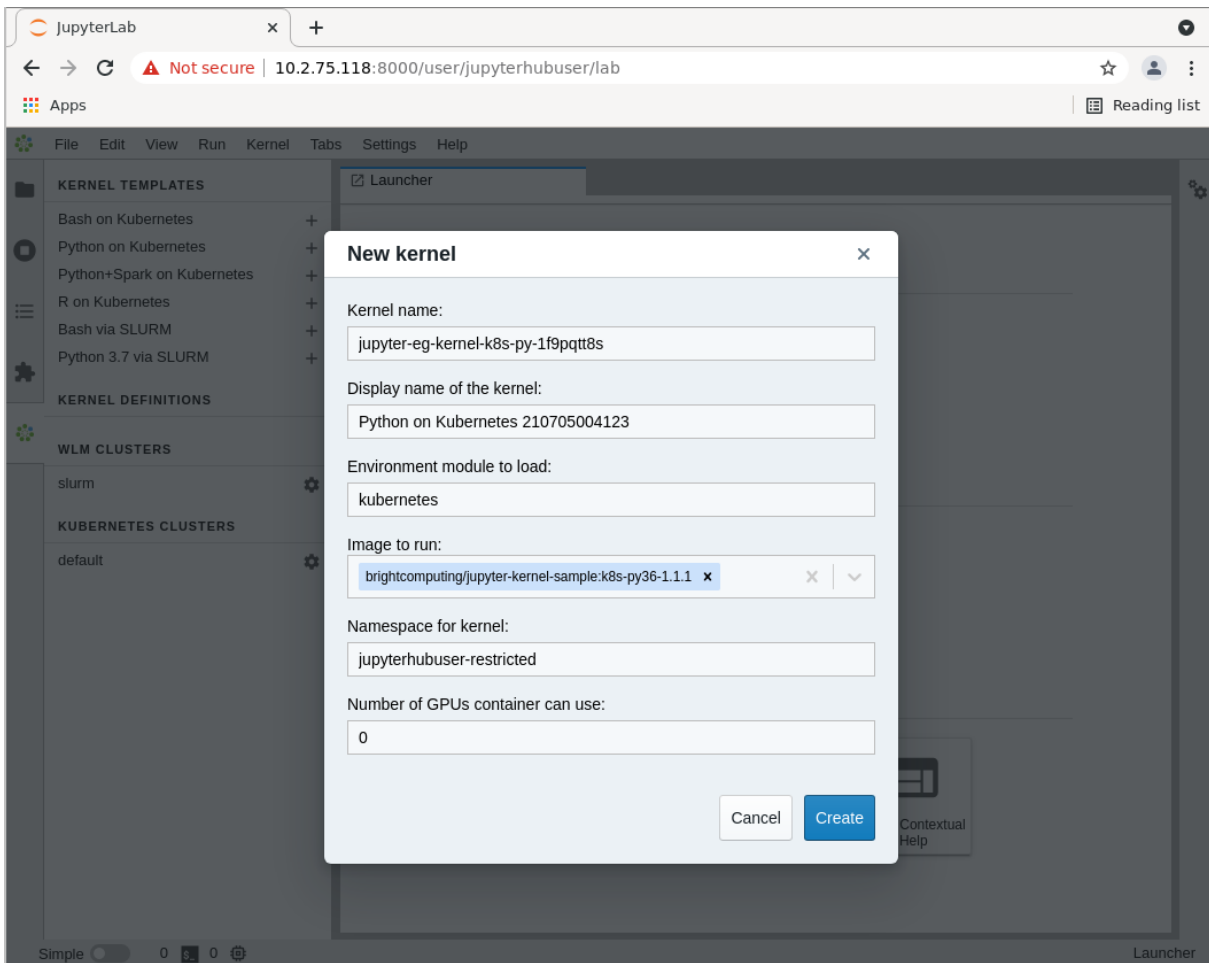


Figure 15.6: Jupyter kernel template customization screen

Once the template is completely customized, the kernel can be created. It automatically appears in the JupyterLab Launcher screen (figure 15.7) and can be used to run notebooks.

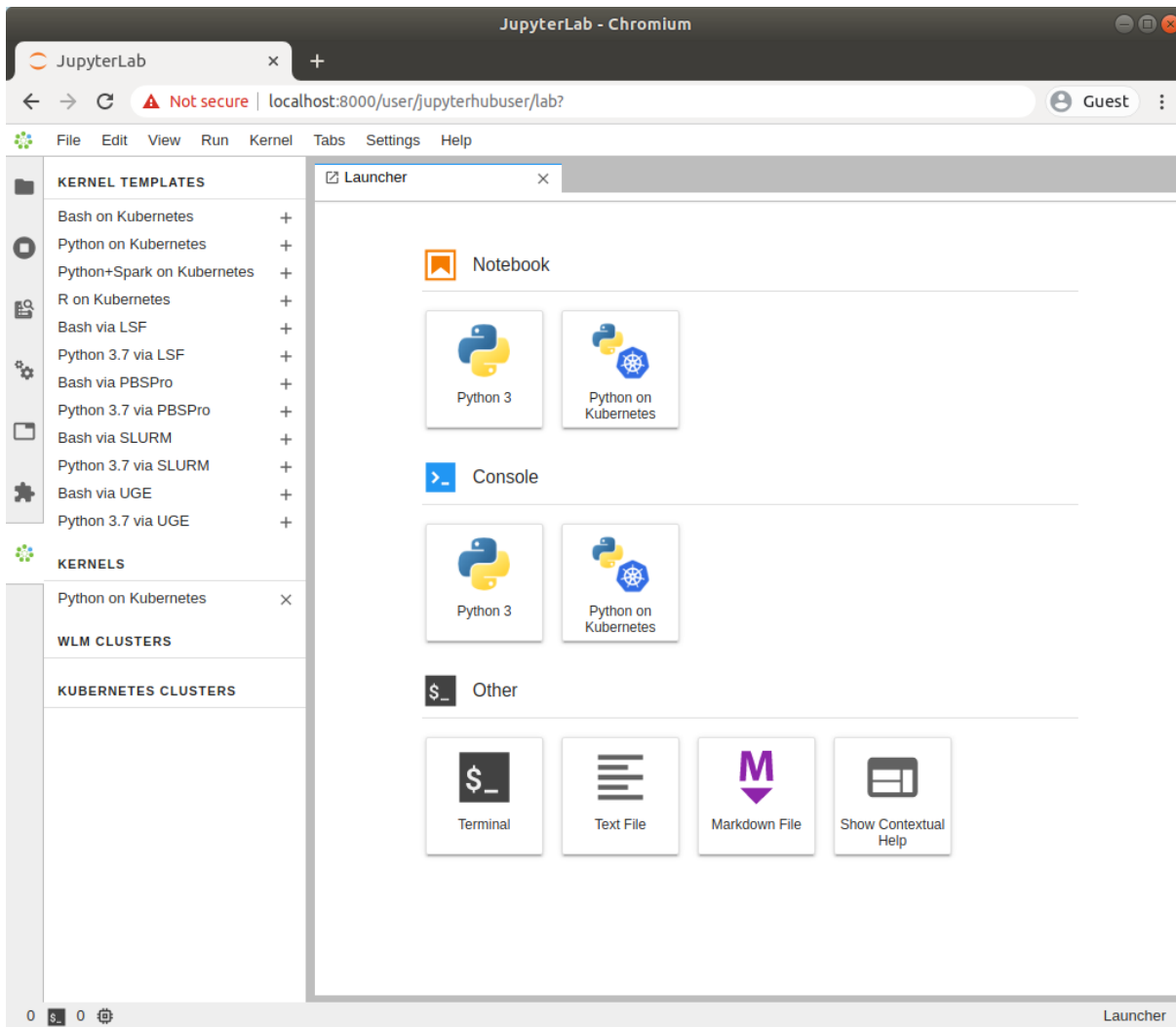


Figure 15.7: JupyterLab Launcher screen with new kernel

A user who lists available Jupyter kernels via the command line now sees the newly-created kernel:

```
[jupyterhubuser@bright90 ~]$ module load jupyter
Loading jupyter/12.0.0
  Loading requirement: python37
[jupyterhubuser@bright90 ~]$ jupyter kernelspec list
Available kernels:
  jupyter-eg-kernel-k8s-py-1f9pqqoiq4  /home/jupyterhubuser/.local/share/jupyter/kernels/jupyter-eg-kernel\
-k8s-py-1f9pqqoiq4
  python3                               /cm/shared/apps/jupyter/12.0.0/share/jupyter/kernels/python3
```

The new kernel directory will contain the JSON definition generated by the Jupyter Kernel Creator:

```
[jupyterhubuser@bright90 ~]$ cat .local/share/jupyter/kernels/jupyter-eg-kernel-k8s-py-1f8verf93/
kernel.json
{
  "language": "python",
  "display_name": "Python on Kubernetes 210624185005",
  "metadata": {
    "process_proxy": {
```

```

"class_name": "cm_jupyter_kernel_creator.eg_processproxies.extk8s.ExternalKubernetesProcessProxy",
"config": {
  "k8s_env_module": "kubernetes",
  "pod_template": "templates/pod.yaml.j2",
  "service_template": "templates/service.yaml.j2",
  "secret_template": "templates/secret.yaml.j2",
  "config_map_template": "templates/config_map.yaml.j2",
  "image": "brightcomputing/jupyter-kernel-sample:k8s-py36-1.1.1",
  "namespace": "default",
  "suffix": "jupyter",
  "gpu_limit": 0
}
},
"argv": [
  "/usr/bin/python3",
  "/ipykernel-k8s.py",
  "{kernel_id}"
]
}

```

Jupyter kernel names need not be unique. Users should therefore choose meaningful and distinguishable display names for their kernels. Doing so makes the JupyterLab Launcher screen easier to use.

For convenience, a summary of the available kernel templates and their requirements is shown in table 15.1:

Table 15.1: Available Jupyter kernel templates for Bright and their requirements

Template name	Requirement	Description
jupyter-eg-kernel-k8s-bash	Kubernetes ¹	Bash via Kubernetes
jupyter-eg-kernel-k8s-py	Kubernetes ¹	Python 3.6 via Kubernetes
jupyter-eg-kernel-k8s-py-spark	Kubernetes ²	Python 3.7 + Spark 3 via Kubernetes
jupyter-eg-kernel-k8s-r	Kubernetes ³	R 3.4 via Kubernetes
jupyter-eg-kernel-lsf-bash	Platform LSF	Bash via Platform LSF
jupyter-eg-kernel-lsf-py37	Platform LSF	Python 3.7 via Platform LSF
jupyter-eg-kernel-pbspro-bash	PBS Pro	Bash via PBS Pro
jupyter-eg-kernel-pbspro-py37	PBS Pro	Python 3.7 via PBS Pro
jupyter-eg-kernel-slurm-bash	Slurm	Bash via Slurm

...continues

Table 15.1: Available Jupyter kernel templates...continued

Template name	Requirement	Description
<code>jupyter-eg-kernel-slurm-py37</code>	Slurm	Python 3.7 via Slurm
<code>jupyter-eg-kernel-uge-bash</code>	UGE	Bash via UGE
<code>jupyter-eg-kernel-uge-py37</code>	UGE	Python 3.7 via UGE

¹ Docker image: `brightcomputing/jupyter-kernel-sample:k8s-py36-1.1.1`

² Docker image: `brightcomputing/jupyter-kernel-sample:k8s-spark-py37-1.2.1`

³ Docker image: `brightcomputing/jupyter-kernel-sample:k8s-r-1.0.0`

DockerHub kernels page: <https://hub.docker.com/r/brightcomputing/jupyter-kernel-sample/tags>

15.5.1 Running Jupyter Kernels With Kubernetes

The following Bright templates allow users to create and run Jupyter Enterprise Gateway kernels on compute nodes via Kubernetes:

Table 15.2: Bright templates for creating and running Jupyter Enterprise Gateway kernels on cluster nodes via Kubernetes

Template	Description
<code>jupyter-eg-kernel-k8s-bash</code>	Custom kernel with Bash
<code>jupyter-eg-kernel-k8s-py</code>	Custom kernel with Python 3.6
<code>jupyter-eg-kernel-k8s-py-spark</code>	Custom kernel with Python 3.7 and Spark 3
<code>jupyter-eg-kernel-k8s-r</code>	Custom kernel with R 3.4

It should be noted that the Spark environment provided by the `jupyter-eg-kernel-k8s-py-spark` template in the preceding table is unrelated to the one available via `cm-spark-setup` (section 9.6.1). The two Spark instances may also differ in their major versions.

An administrator has to make sure that Kubernetes is correctly configured on the cluster, and that Pod Security Policy (PSP) is enabled (section 9.3.10).

Details on Kubernetes installation are provided in section 9.3.

The default configuration proposed by `cm-kubernetes-setup` is usually sufficient to run Kubernetes kernels created from Bright's templates. However, it is the responsibility of the administrator to add users registered in the Linux-PAM system to Kubernetes.

For example, a test user `jupyterhubuser` can be added to Kubernetes with:

Example

```
[root@bright90 ~]# cm-kubernetes-setup --add-user jupyterhubuser
```

For every new user added, `cm-kubernetes-setup` automatically generates a dedicated namespace with a name in the form `<user>-restricted`. For instance, the command in the example above, creates the namespace: `jupyterhubuser-restricted`.

Cluster administrators are strongly recommended to review pods security policies for these dedicated namespaces.

In order to speed up kernel creation for the first user logging into JupyterLab, Bright recommends that all the relevant Kubernetes images are pre-loaded on the compute nodes that Jupyter Enterprise Gateway can contact.

Jupyter notebooks using Kubernetes kernels cannot natively use machine learning libraries and frameworks provided by Bright, even if installed on the software image used by Kubernetes. The machine learning libraries and frameworks have to be included in the chosen Kubernetes image.

15.5.2 Running Jupyter Kernels With Workload Managers

Bright's Jupyter Enterprise Gateway kernels can be created and run by users on compute nodes via workload managers (WLMs). For convenience, Slurm is used as an example in this section. However, the same instructions are valid for the other WLMs listed in table 15.1.

The templates used to create Slurm kernels are Bright's

- `jupyter-eg-kernel-slurm-bash` and
- `jupyter-eg-kernel-slurm-py37`,

which offer a Bash and a Python 3.7 environment respectively.

The administrator has to make sure that Slurm is correctly configured on the cluster. Slurm installation is described in section 7.3.

The default configuration that `cm-wlm-setup` suggests in Express mode is usually sufficient to run Slurm kernels created from Bright's templates. If Slurm is configured to automatically detect GPUs, they will be listed as available resources while instantiating Slurm templates.

The Jupyter login node must be authorized to submit Slurm jobs. This is typically the case, since the JupyterHub login node is by default the head node, and `cm-wlm-setup` by default assigns the `slurmsubmit` role to the head node.

Finally, the administrator must make sure that relevant dependency packages are installed on the software image used by Slurm clients. The image used is typically for the compute nodes. Missing packages may cause kernels to fail at startup or at run time.

In particular, administrators need to install `cm-python37` to use kernels based on the `jupyter-eg-kernel-slurm-py37` template.

Jupyter notebooks using Slurm kernels can natively use the machine learning libraries and frameworks provided by Bright on Slurm clients, if these are available. Further details on machine learning packages installation on both the head node and the compute nodes are given in section 1.4 of the *Machine Learning Manual*.

15.6 Jupyter Kernel Creator Extension Customization

The Jupyter kernel templates described in section 15.5 are stored under the directory `/cm/shared/apps/jupyter/current/lib/python3.7/site-packages/cm_jupyter_kernel_creator/kerneltemplates/`.

The template of a particular Jupyter kernel in Bright's Jupyter Kernel Creator extension is a directory containing at least two files: `meta.yaml` and `kernel.json.j2`. The kernel template can also require other files placed in the same directory, such as icons. These files are copied to the target user kernels directory upon kernel creation, after the template is instantiated.

The `meta.yaml` file includes all the parameters that can be substituted into `kernel.json.j2` and is defined with the YAML format (<https://yaml.org/>).

The `kernel.json.j2` file is a skeleton of the `kernel.json` file to be generated. It can contain some placeholders, and is defined with the Jinja2 format (<https://jinja.palletsprojects.com/>).

15.6.1 Kernel Template Parameters Definition

The kernel template `meta.yaml` file defines all the parameters that can be used in `kernel.json.j2`.

It should contain three entries:

- `display_name`: the name that will be displayed for the kernel in the JupyterLab Launcher;
- `features`: a list of features that must be available on the cluster to show this kernel template to JupyterLab users in the Bright extensions panel;
- `parameters`: the variables for `kernel.json.j2`.

The `display_name` entry is an arbitrary string:

```
[...]
display_name: "A simple kernel"
[...]
```

The `features` entry is a list of strings. The only possible values for these strings are:

- "kubernetes"
- "lsf"
- "pbspro"
- "slurm"
- "uge"

For example, a kernel template that only requires Kubernetes to work contains this line in its `meta.yaml`:

```
[...]
features: ["kubernetes"]
[...]
```

The same principle applies to the other workload managers. If multiple features are listed, then the kernel template is displayed if at least one of them is available.

Finally, the `parameters` entry contains a dictionary of kernel parameters. The keys of the dictionary are parameter names that are used in `kernel.json.j2`. The values of the dictionary are options that help users choose a correct value for the parameter. Parameter options are also dictionaries.

A kernel template `meta.yaml` is thus structured as:

```
[...]
parameters:
  <parameter_name1>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
  <parameter_name2>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
  ...
  <parameter_nameN>:
    <option_key>: <option_value>
    ...
    <option_key>: <option_value>
[...]
```

Parameter names are arbitrary strings.

Option keys are strings. The only possible values for these strings are:

- `type`
- `definition`
- `limits`

The option keys `type` and `definition` are mandatory. The option key `limits` is optional.

For example, a kernel template with two parameters `foo` and `bar` looks as follows:

Example

```
[...]  
parameters:  
  foo:  
    type: <option_value>  
    definition: <option_value>  
    limits: <option_value>  
  bar:  
    type: <option_value>  
    definition: <option_value>  
    limits: <option_value>  
[...]
```

The type Option

The type option key defines the type of the kernel parameter.

The type option key only accepts one of the following string values:

- `num`: for numeric values (both float and integer are supported);
- `str`: for arbitrary strings;
- `bool`: for boolean values (a checkbox is presented in the user interface);
- `list`: for lists of pre-defined or dynamically-generated settings;
- `uri`: for interactive RESTful endpoints (only `/kernelcreator/envmodules` is currently supported).

The definition Option

The definition option key defines how the parameter value is retrieved and displayed.

The definition option key accepts dictionary-like values. Allowed string keys for the dictionary are:

- `display_name`: the name that is displayed for the parameter in the kernel customization dialog. It accepts an arbitrary string as a value;
- `getter`: how parameter values are retrieved. It only accepts as value one of the following strings:
 - `static`: the values for the parameter are pre-defined;
 - `shell`: the values for the parameter are the output of the shell script;
 - `python`: the values for the parameter are the output of a Python script;
- `values`: possible values for the parameter that are displayed in the kernel customization dialog. It accepts a list of arbitrary values;
- `default`: default value for the parameter. It accepts a value from `values`;
- `exec`: the script to be executed to fill values when `getter` is `shell` or `python`. It accepts a shell/Python script.

If type option value is `list`, then every line of `getter` with a setting of `shell` or `python` is treated as an element of the list.

The `limits` Option

The `limits` option key offers a way to apply bounds on values provided by users. This usually reduces the chances of making mistakes and helps users defining correct kernels before actually running them.

The `limits` option key accepts dictionary-like values according to the value chosen for the type option key.

If `num` is the type, then `limits` can contain:

- `min`: the minimum numeric value;
- `max`: the maximum numeric value.

If `str` or `list` is the type, then `limits` can contain:

- `min_len`: the minimum length of the string or the list;
- `max_len`: the maximum length of the string or the list.

For example, if every node in the cluster has no more than 4 GPUs, then the upper limit on the requested GPU number can be set to 4:

```
[...]
parameters:
  gpus:
    type: num
    definition: <option_value>
    limits:
      min: 1
      max: 4
[...]
```

These limits are not a security measure. They should be considered as convenient sanity checks for values entered while instantiating a kernel template. This is because users are always able to later directly edit the generated `kernel.json`, thereby ignoring such limits.

15.6.2 Kernel Template Parameters Usage

In the `kernel.json.j2` file of a kernel template, two types of variables can be defined: Jinja2 variables (<https://jinja.palletsprojects.com/en/master/templates/#variables>) and Python formatted string literals (also known as f-strings) (https://docs.python.org/3/reference/lexical_analysis.html#f-strings).

They should not be mixed as they work on different steps of the template lifecycle:

- `{{ . }}` and `{% . %}`: These formats are used to specify Jinja2 expressions and statements. Jinja2 expressions and statements are substituted or computed once, when the user creates a kernel with Bright's Jupyter Kernel Creator extension. The actual values will be those specified in the kernel customization dialog;
- `{ . }`: This format is used to specify Python f-strings. F-strings are substituted every time a user starts the kernel for a Jupyter notebook. The actual values will be a path to the kernel connection file, a job ID in a WLM, a UID of the running kernel process, a unique name for a Kubernetes pod, etc.

Some process-proxies in Jupyter Enterprise Gateway may require another type of variables template. For example, the Kubernetes process-proxy requires multiple objects to be set up in its API. Several unique configuration files need to be created at runtime. Templates for such objects are located in the `templates/` directory, along with the `meta.yaml` and `kernel.json.j2` files. These templates are rendered every time a Kubernetes kernel is started for a Jupyter notebook. Variables for these templates are located inside the `config` section of the `kernel.json` file.

15.6.3 Kernel Template Creation Example

To illustrate the kernel variables syntax described in sections 15.6.1 and 15.6.2, a kernel template is now created from Jupyter's default Python 3 kernel.

The new kernel template has to be placed in a suitable directory for Bright's Jupyter Kernel Creator extension. This directory is the `kerneltemplates/` directory in the extension's Python module:

```
[root@bright90 ~]# ls /cm/shared/apps/jupyter/current/lib/python3.7/site-packages/\
cm_jupyter_kernel_creator/kerneltemplates/
jupyter-eg-kernel-k8s-bash      jupyter-eg-kernel-lsf-bash      jupyter-eg-kernel-slurm-bash
jupyter-eg-kernel-k8s-py       jupyter-eg-kernel-lsf-py37      jupyter-eg-kernel-slurm-py37
jupyter-eg-kernel-k8s-py-spark jupyter-eg-kernel-pbspro-bash   jupyter-eg-kernel-uge-bash
jupyter-eg-kernel-k8s-r        jupyter-eg-kernel-pbspro-py37   jupyter-eg-kernel-uge-py37
```

The content of the original Jupyter kernel directory, including its images, is copied. The `kernel.json` file is then renamed to `kernel.json.j2` to make it a Jinja2 template:

```
[root@bright90 ~]# cd /cm/shared/apps/jupyter/current/lib/python3.7/site-packages/\
cm_jupyter_kernel_creator/kerneltemplates/
[root@bright90 kerneltemplates]# cp -pr /cm/shared/apps/jupyter/current/share/jupyter/kernels/\
python3 python3-test
[root@bright90 kerneltemplates]# mv python3-test/kernel.json{,.j2}
```

The next step is to create the `meta.yaml` file with the kernel template parameters. In this example a request is made to have Slurm deployed on the cluster:

```
[root@bright90 kerneltemplates]# cat << EOF >python3-test/meta.yaml
---
display_name: "Python 3 demo kernel"
features: ["slurm"]
EOF
```

The new kernel template is now available in JupyterLab (figure 15.8):

KERNEL TEMPLATES	
Bash via SLURM	+
Python 3.7 via SLURM	+
Python 3 demo kernel	+

Figure 15.8: New kernel template in the JupyterLab interface

The next step is to define kernel template parameters. The first one being added is `kernel_name`, which is the name of the kernel in the JupyterLab Launcher. The `meta.yaml` file is updated:

```
---
display_name: "Python 3 demo kernel"
features: ["slurm"]
parameters:
  kernel_name:
    type: str
    definition:
      getter: static
      default: "Demo kernel"
      display_name: "Display name of the kernel"
```

The `display_name` definition in the `kernel.json.j2` is now changed to a Jinja2 placeholder matching this `meta.yaml` parameter with the appropriate Jinja2 syntax:

```
{
  "display_name": "{{ kernel_name }}",
  [...]
}
```

The JupyterLab interface changes and now prompts the user to enter a value, while providing a default option (figure 15.9):

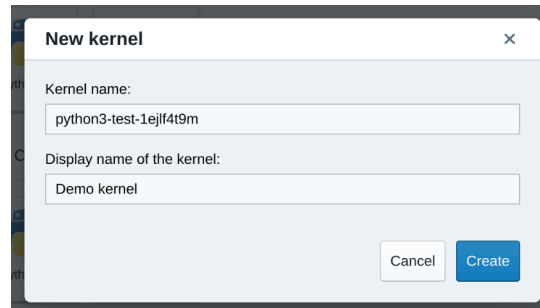


Figure 15.9: New kernel template dialog during template instantiation

If the user clicks on the **Create** button, then a new directory is created in `~/.local/share/jupyter/kernels`. The content of the kernel template directory, except for the `meta.yaml` and `kernel.json.j2` files, is copied into this new directory. The kernel name provided by the user is substituted into the `kernel.json.j2` template, and the resulting `kernel.json` file is written into the new user kernel directory.

As a result, the new kernel then appears in the JupyterLab Launcher.

15.6.4 Extending Kubernetes Kernel Templates

Bright's JEG kernel templates for Kubernetes can be extended with additional parameters.

In the following example, a Kubernetes kernel template is extended, to show a dropdown menu with additional environment variables in the kernel customization dialog. The selected variables are exported to the context of the running kernel and made available in the Jupyter notebook.

In order to preserve the original kernel template, the existing `jupyter-eg-kernel-k8s-py` template is copied to a user directory. The new customized kernel template directory could be later copied back to the original Bright Jupyter Kernel Creator extension templates directory, to make it available for all the users.

```
[jupyterhubuser@bright90 ~]# mkdir -p ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright90 ~]# cd ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright90 kerneltemplates]# cp -pr /cm/shared/apps/jupyter/current/lib/python3.7/\
site-packages/cm_jupyter_kernel_creator/kerneltemplates/jupyter-eg-kernel-k8s-py\
./jupyter-eg-kernel-k8s-py-demo
```

To distinguish the new kernel template from the original one, it is convenient to change the display name:

```
---
display_name: "Python on Kubernetes Demo"
features: ["kubernetes"]
[...]
```

The next step is to add a new parameter in the `meta.yaml` file:

```
[...]
parameters:
```

```
[...]
additional_vars:
  type: list
  definition:
    getter: shell
    default: []
    display_name: "Additional variables to export in the pod"
    exec:
      - echo -e "VAR1=value1\nCUR_DATE=$(date +%Y-%m-%d)"
[...]
```

Now, if the kernel template is instantiated, then a new parameter is displayed in the kernel customization dialog (figure 15.10):

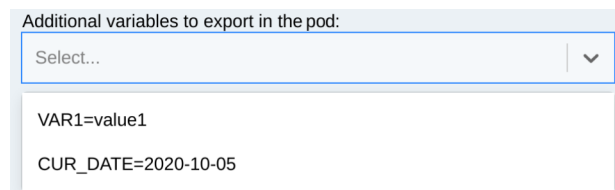


Figure 15.10: New environment variables dropdown menu

The kernel customization dialog also shows that every output line of the script was rendered as an item of the menu.

To pass the selected environment variables to the actual container in the running pod, the `kernel.json.j2` and `templates/pod.yaml.j2` files must both be modified. This is due to the two-step template rendering mechanism:

1. the first rendering takes place when the user instantiates the kernel template. The `kernel.json.j2` file becomes `kernel.json`, and kernel parameters are replaced with the values specified in the dialog;
2. the second rendering takes place when the user starts the kernel for a Jupyter notebook. The `pod.yaml.j2` file and other Jinja2 files in the `templates/` directory are rendered to the actual YAML representations of the Kubernetes objects.

In the example that follows, the `kernel.json.j2` file is extended with the "additional_vars" kernel parameter:

```
{
  [...]
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        [...]
        "additional_vars": {{ additional_vars | tojson }},
        [...]
      }
    }
  },
  [...]
}
```

The last step requires the use of the Jinja2 loop syntax in the `templates/pod.yaml.j2` file:

```
[...]
spec:
  [...]
  containers:
  - name: notebook
    [...]
    env:
      [...]
      {% for env_var in additional_vars %}
      {% set var, value = env_var.split("=") %}
      - name: {{ var }}
        value: "{{ value }}"
      {% endfor %}
[...]
```

Now, if a kernel is instantiated with both the environment variables (figure 15.11) and then started from a notebook, then the variables are available in the process (figure 15.12):

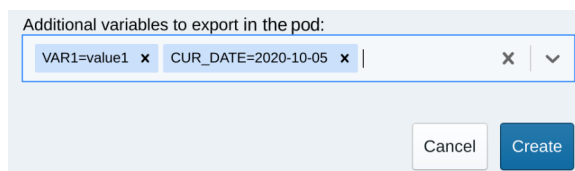


Figure 15.11: Environment variables are chosen in the kernel customization dialog

```
[1]: import os
[2]: os.environ["VAR1"]
[2]: 'value1'
[3]: os.environ["CUR_DATE"]
[3]: '2020-10-05'
```

Figure 15.12: Environment variables are available in the running kernel

15.6.5 Extending Workload Managers Kernel Templates

Bright's JEG kernel templates for workload managers can be extended with additional parameters. For convenience, Slurm is used as example in this section. However, the same instructions are valid for the other WLMs listed in table 15.1.

In the following example, a Slurm kernel template is extended, to show a dropdown menu with additional environment variables in the kernel customization dialog. The selected variables are exported to the context of the running kernel and made available in the Jupyter notebook.

In order to preserve the original kernel template, the existing `jupyter-eg-kernel-slurm-py37` template is copied to a user directory. The new customized kernel template directory could be later copied back to the original Bright Jupyter Kernel Creator extension templates directory, to make it available for all users.

```
[jupyterhubuser@bright90 ~]# mkdir -p ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright90 ~]# cd ~/.local/share/jupyter/kerneltemplates
[jupyterhubuser@bright90 kerneltemplates]# cp -pr /cm/shared/apps/jupyter/current/lib/python3.7/\
site-packages/cm_jupyter_kernel_creator/kerneltemplates/jupyter-eg-kernel-slurm-py37\
./jupyter-eg-kernel-slurm-py37-demo
```

To distinguish the new kernel template from the original one, it is convenient to change the display name:

```
---
display_name: "Python 3.7 via SLURM Demo"
features: ["slurm"]
[...]
```

The next step is to add a new parameter in the `meta.yaml` file:

```
[...]
parameters:
[...]
```

```
  additional_vars:
    type: list
    definition:
      getter: shell
      default: []
      display_name: "Additional variables to export in the job"
      exec:
        - echo -e "VAR1=value1\nCUR_DATE=$(date +%Y-%m-%d)"
[...]
```

Now, if the kernel template is instantiated, then a new parameter is displayed in the kernel customization dialog (figure 15.13):

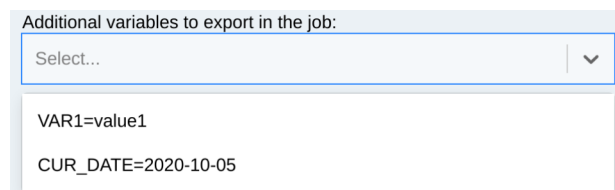


Figure 15.13: New environment variables dropdown menu

The kernel customization dialog also shows that every output line of the script was rendered as an item of the menu.

To pass the selected environment variables to the actual workload manager job, the `"submit_script"` section inside the `kernel.json.j2` file must be modified to take into account the new `"additional_vars"` kernel parameter:

```
{
[...]
```

```
  "metadata": {
    "process_proxy": {
      [...]
      "config": {
        [...]
        "submit_script": [
          [...]
          "set -xe",
          "{environ}",
          {% for env in additional_vars %}
          "export {{ env }}",
          {% endfor %}
          "{kernel_cmd}"
        ]
      }
    }
  }
}
```

```

    }
  }
},
[...]
```

Now, if a kernel is instantiated with both the environment variables (figure 15.14) and then started from a notebook, then the variables are available in the process (figure 15.15):

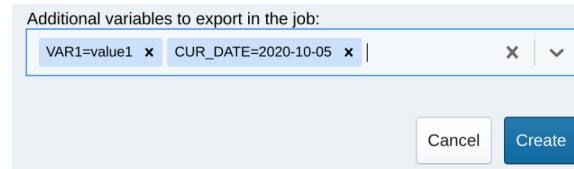


Figure 15.14: Environment variables are chosen in the kernel customization dialog

```

[1]: import os
[2]: os.environ["VAR1"]
[2]: 'value1'
[3]: os.environ["CUR_DATE"]
[3]: '2020-10-05'
```

Figure 15.15: Environment variables are available in the running kernel

15.7 Jupyter VNC Extension

If VNC is allowed by the cluster administrator, then the Jupyter environment configured by Bright can be used to start and control remote desktops via VNC with the *Jupyter VNC* extension.

Several kernels created from Bright's templates are capable of running VNC sessions so that users can run GUI applications. In order to do so, the cluster nodes where kernels are executed must support VNC. It is outside of the scope of this section to provide guidance on how to install and configure VNC on different Linux distributions. However, it is important to note that Jupyter kernels usually work with several types of VNC client/server.

Bright's default choice for VNC is TigerVNC, a high-performance, platform-neutral implementation of VNC.

For example, the administrator can install the VNC server on the head node as follows:

Example

```

[root@bright90 ]# yum install tigervnc-server #on RHEL 7 and 8

[root@bright90 ]# zypper install tigervnc #on SLES 15

[root@bright90 ]# apt-get install tightvncserver #on Ubuntu 18.04
```

The preceding command must be applied to all software images that are used to run Jupyter kernels for VNC.

Users can start a VNC session with the button added by Jupyter VNC (figure 15.16). Additional VNC parameters can be optionally specified.

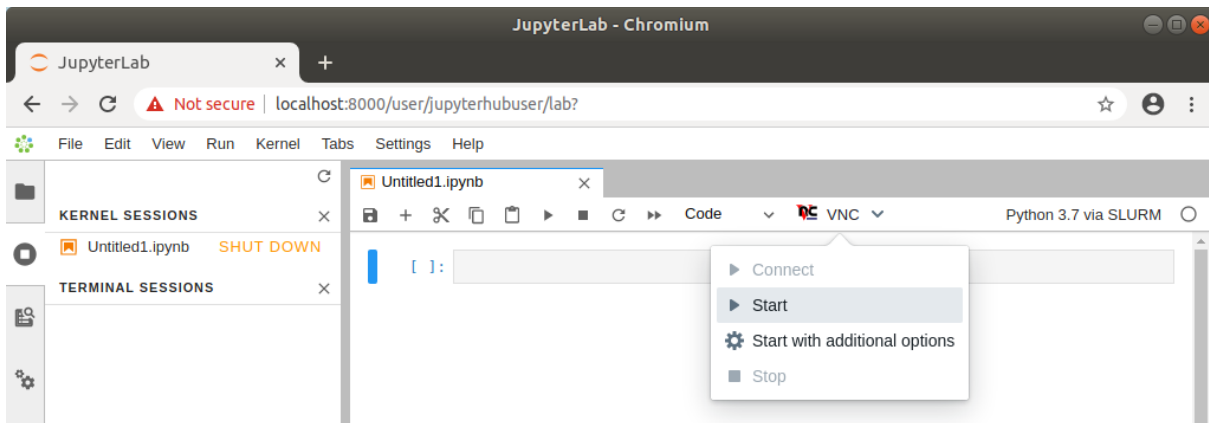


Figure 15.16: Starting Jupyter VNC session from kernel

If VNC is available and correctly configured on the node where the kernel is running, then a new tab is automatically created by Jupyter VNC containing the new session (figure 15.17). A user can now freely interact in JupyterLab both with the notebook and with the desktop environment.

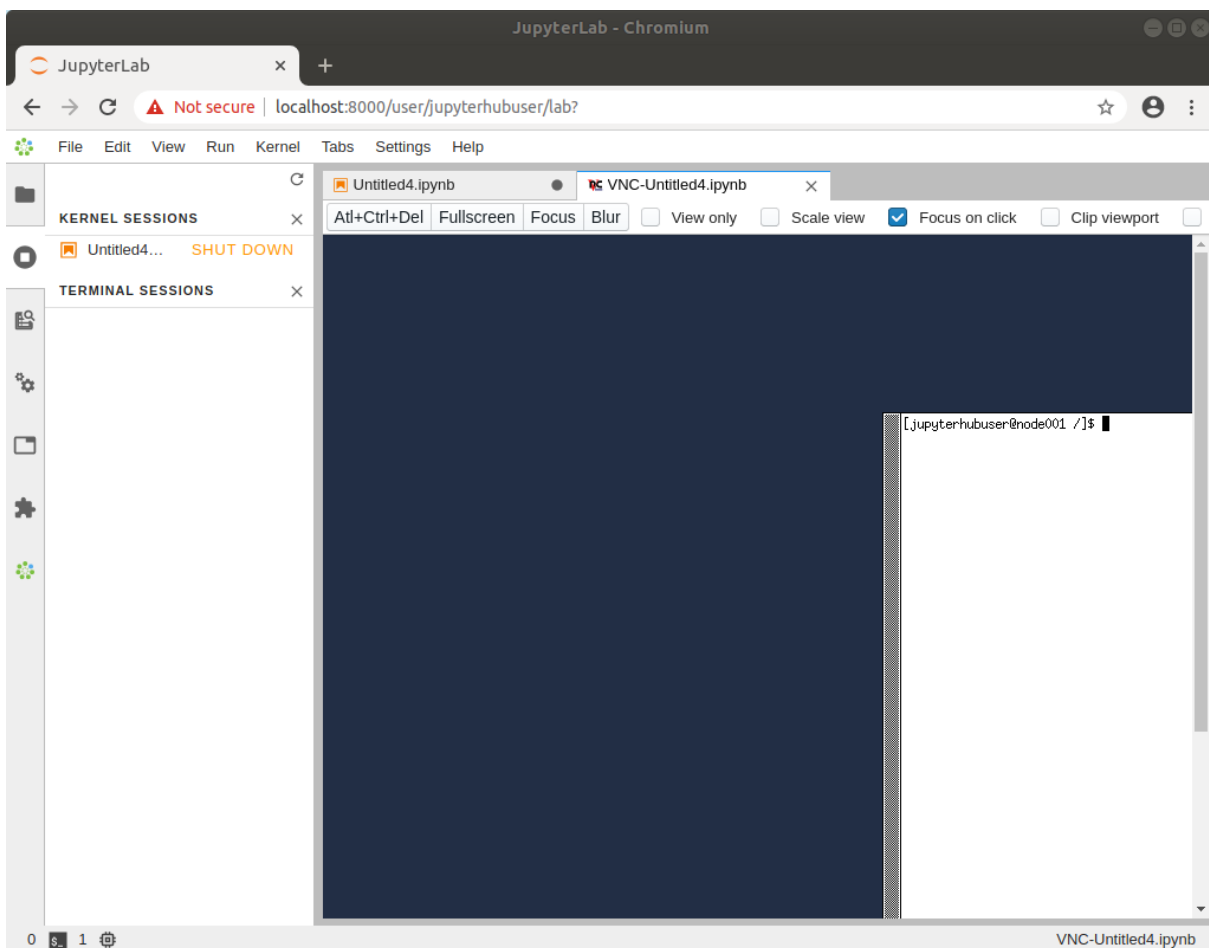


Figure 15.17: Running Jupyter VNC session from kernel

To provide a user-friendly experience, Jupyter VNC also allows the graphical viewport to be resized, so that the desktop application can run full-screen (figure 15.18).

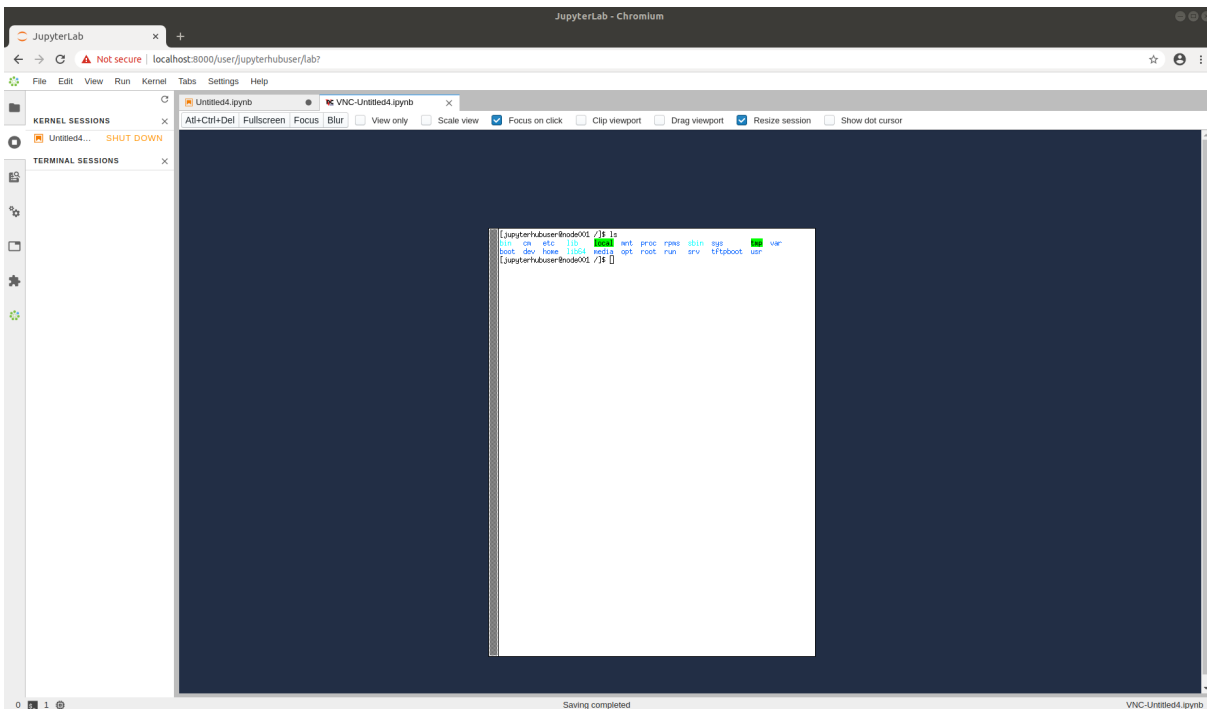


Figure 15.18: Running Jupyter VNC session from kernel (full-screen)

Once the VNC session is correctly started and the new JupyterLab tab has been created, Jupyter VNC automatically exports the `DISPLAY` environment variable to the running notebook (figure 15.19). Doing so means that any application or library running in the notebook can make use of the freshly created desktop environment. An example of such a library is OpenAI Gym, a toolkit for developing and comparing reinforcement learning algorithms, that is distributed by Bright.

Among the examples distributed by Bright (section 15.3), a notebook running PyTorch in the OpenAI Gym CartPole environment can be found. If executed after a VNC session has been started, a user can then observe the model being trained in real time in the graphical environment.

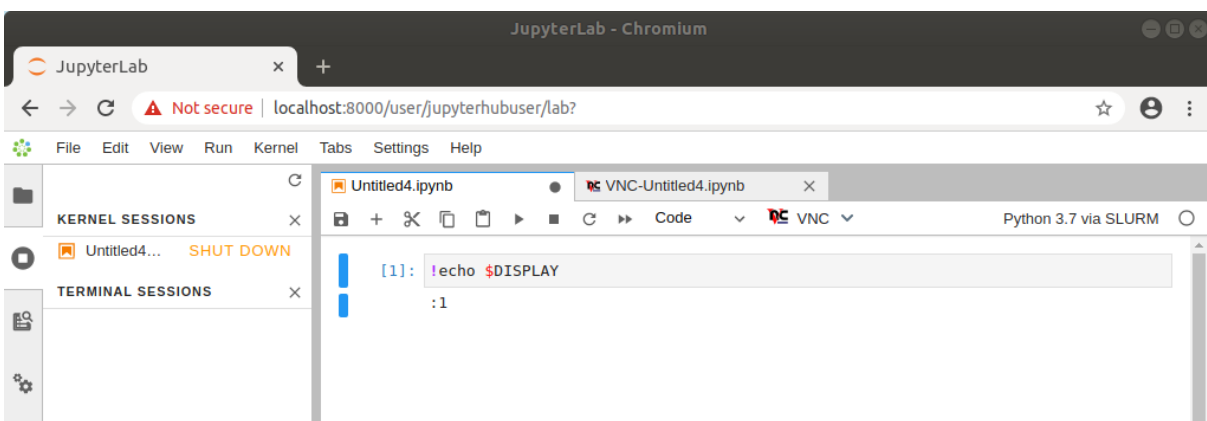


Figure 15.19: Automatic configuration of `DISPLAY` environment variable

15.8 Jupyter WLM Magic Extension

In the Jupyter environment configured by Bright, the *Jupyter WLM Magic* extension can be used to schedule workload manager jobs from notebooks.

The Jupyter WLM Magic extension is an IPython extension. It is designed to improve the capabilities

of Jupyter's default Python 3 kernel, which runs on the login node.

The Jupyter WLM Magic extension should therefore not be used from kernels running on compute nodes, such as those typically created with Bright's Jupyter Kernel Creator extension (section 15.5), and submitted via Jupyter Enterprise Gateway. Indeed, compute nodes running these kernels are often incapable of starting workload manager jobs in many default WLM configurations.

Jupyter WLM Magic extension makes it possible for users to programmatically submit WLM jobs, and then interact with their results. This can be done while using the Python programming language and its libraries, which are available in the notebook.

Users submit jobs and check their progress from the login node. The actual computation is distributed by the underlying workload manager across compute nodes, which means that server resources are spared.

Jupyter WLM Magic commands are available in the IPython kernel as *magic functions* (<https://ipython.readthedocs.io/en/stable/interactive/tutorial.html#magic-functions>). A new line magic (%) and a new cell magic (%%) are now added in the kernel, according to the workload manager:

- Platform LSF: %lsf_job and %%lsf_job
- PBS Pro: %pbspro_job and %%pbspro_job
- Slurm: %slurm_job and %%slurm_job
- UGE: %uge_job and %%uge_job

A user can list the magic functions in the kernel to see if they are available, with Jupyter's builtin command %lsmagic (<https://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-lsmagic>):

Example

```
In []: %lsmagic
Out []: root:
      line:
        automagic:"AutoMagics"
        autocall:"AutoMagics"
        [...]
        slurm_job:"SLURMMagic"
        pbspro_job:"PBSProMagic"
        lsf_job:"LSFMagic"
        uge_job:"UGEMagic"
      cell:
        js:"DisplayMagics"
        javascript:"DisplayMagics"
        [...]
        slurm_job:"SLURMMagic"
        pbspro_job:"PBSProMagic"
        lsf_job:"LSFMagic"
        uge_job:"UGEMagic"
```

The magic functions introduced by this Bright extension share a similar syntax. For convenience, Slurm is used as an example in this section. However, the same instructions are valid for the other WLMs.

Users can check which options are available for a WLM function with the line magic helper:

Example

```
In []: %slurm_job --help
Out []: usage: %slurm_job [-h] [--module MODULE] [--module-load-cmd MODULE_LOAD_CMD]
        [--shell SHELL] [--submit-command SUBMIT_COMMAND]
        [--cancel-command CANCEL_COMMAND]
        [--control-command CONTROL_COMMAND]
        [--stdout-file STDOUT_FILE] [--stderr-file STDERR_FILE]
        [--preamble PREAMBLE] [--timeout TIMEOUT]
        [--check-condition-var CHECK_CONDITION_VAR]
        [--job-id-var JOB_ID_VAR]
        [--stdout-file-var STDOUT_FILE_VAR]
        [--stderr-file-var STDERR_FILE_VAR] [--dont-wait]
        [--write-updates WRITE_UPDATES]
        [--check-status-every CHECK_STATUS_EVERY]
```

optional arguments:

```
  -h, --help            show this help message and exit
  [...]
```

Line magic functions are typically used to set options with a global scope in the notebook. By doing so, a user will not need to specify the same option every time a job will be submitted via cell magic. For example, if two Slurm instances are deployed on the cluster and their associated environment modules are `slurm-primary` and `slurm-secondary`, a user could run the following line magic once to configure the Jupyter WLM Magic extension to always use the second deployment:

Example

```
In []: %slurm_job --module slurm-secondary
Out []:
```

Now, jobs will always be submitted to `slurm-secondary`. This is more convenient than repeatedly defining the same module option for every cell magic upon scheduling a job:

Example

```
In []: %%slurm_job --module slurm-secondary
        <WLM JOB DEFINITION>
Out []: <WLM JOB OUTPUT>
In []: %%slurm_job --module slurm-secondary
        <WLM JOB DEFINITION>
Out []: <WLM JOB OUTPUT>
```

It should be noted that line magic functions cannot be used to submit WLM jobs. Cell magic functions have to be used instead.

A well-defined cell contains the WLM cell magic function provided by the extension, followed by the traditional job definition. For example, a simple MPI job running on two nodes can be submitted to Slurm by defining and running this cell:

Example

```
In []: %%slurm_job
        #SBATCH -J mpi-job-example
        #SBATCH -N 2
        module load openmpi
        mpirun hostname
Out []: COMPLETED
        STDOUT file content: /home/demo/.jupyter/wlm_magic/slurm-1.out
```

```
node001
node001
node002
node002
```

Users can take advantage of the Jupyter WLM Magic extension to store some information into Python variables about the job being submitted. The information could be the ID or the output file name, for example. Users can then later programmatically interact with them in Python. This feature is convenient when a user wants to, for example, programmatically carry out new actions depending on the job output:

Example

```
In []: %%slurm_job --job-id-var my_job_id --stdout-file-var my_job_out
      #SBATCH -J mpi-job-example
      #SBATCH -N 2
      module load openmpi
      mpirun hostname
Out []: COMPLETED
      STDOUT file content: /home/demo/.jupyter/wlm_magic/slurm-2.out
      node001
      node001
      node002
      node002

In []: print(f"Job id {my_job_id} was written to {my_job_out}")
      print(f"Output lines: {open(my_job_out).readlines()}")
Out []: Job id 2 was written to /home/demo/.jupyter/wlm_magic/slurm-2.out
      Output lines: ['node001\n', 'node001\n', 'node002\n', 'node002\n']
```

Users can also exploit Python variables to define the behavior of the Jupyter WLM Magic extension. For example, they can define a Python boolean variable to submit a WLM job only if a condition is true:

Example

```
In []: run_job = 1 == 2
Out []:

In []: %%slurm_job --check-condition-var run_job
      #SBATCH -J mpi-job-example
      #SBATCH -N 2
      module load openmpi
      mpirun hostname
Out []: Variable run_job is 'False'. Skipping submit.
```

15.9 Jupyter Environment Removal

Before removing Jupyter, the administrator should ensure that all kernels have been halted, and that no user is still logged onto the web interface. Stopping the `cm-jupyterhub` service with users that are still logged in, or with running kernels, has undefined behavior.

To remove Jupyter, the script `cm-jupyter-setup` must be run, either in interactive mode, or with the option `--remove`.

Removing Jupyter does not remove or affect Kubernetes or WLM deployments.

For a more complete cleanup, the following packages must be manually removed from the nodes involved in the Jupyter deployment: `cm-jupyter`, `cm-jupyter-local`, and `cm-npm-configurable-http-proxy`.

16

Dell BIOS Management

16.1 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, Bright View 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:

Models

14G

R430, R630, R730, R730XD, R930

FC430, FC630, FC830

M630, M830

C6320

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 16.1)
- The prerequisites for the integration to work (section 16.2)
- A description of settings management in Bright View and `cmsh` (section 16.3)
- A troubleshooting FAQ (section 16.4)

16.2 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 3.3.10 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 be performed manually. This privilege cannot be enabled during node provisioning by Bright during the BMC configuration step.

16.3 BIOS settings

16.3.1 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

```
[bright90->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

```
[bright90->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

```
[bright90->device]% foreach -c default (dellsettings; clearsettings; commit; )
```

16.3.2 Managing The BIOS Settings Via Bright View

The Dell BIOS settings can be accessed via Bright View, using settings within the node or within the category, using the clickpaths:

Devices→Nodes→Edit→Settings→Dell Settings

or:

Grouping→Node Categories→Settings→Dell Settings

If `CMDaemon` is able to communicate with the `racadm` utility the Dell Settings can then be configured (figure 16.1).

Some extra Dell-related fields can also be seen in the `SYS INFO COLLECTOR` subpane for a node, under the clickpath:

Devices→Nodes→Edit→Settings→System Overview

The motherboard manufacturer, name, and system manufacturer are among the values collected in this subpane.

The Dell Settings Subwindow

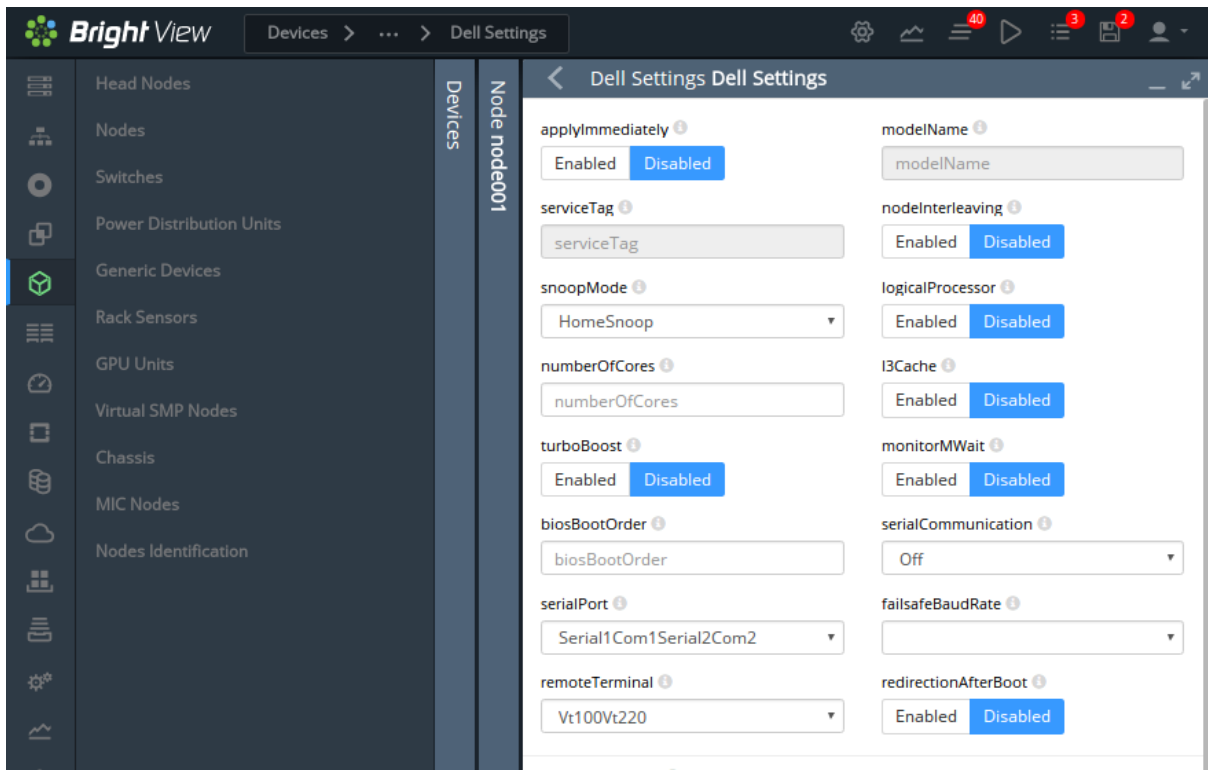


Figure 16.1: Selection of a node node001, then opening the Dell Settings subwindow

The BIOS Settings subwindow gives access to some basic BIOS properties, including:

- (figure 16.1) boot order, whether to apply the configuration changes immediately or on the next boot, and serial communication settings
- (figure 16.2) a devicefirmware subwindow, where the BIOS firmware can be set.
- (figure 16.2) a nics subwindow, where the NIC properties can be viewed and set.
- (figure 16.3) performance and power profiles, processor frequency and memory refresh rates.

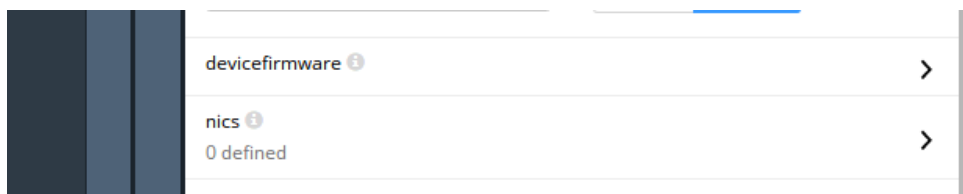


Figure 16.2: devicefirmware and nics subwindow access from the Dell Settings subwindow

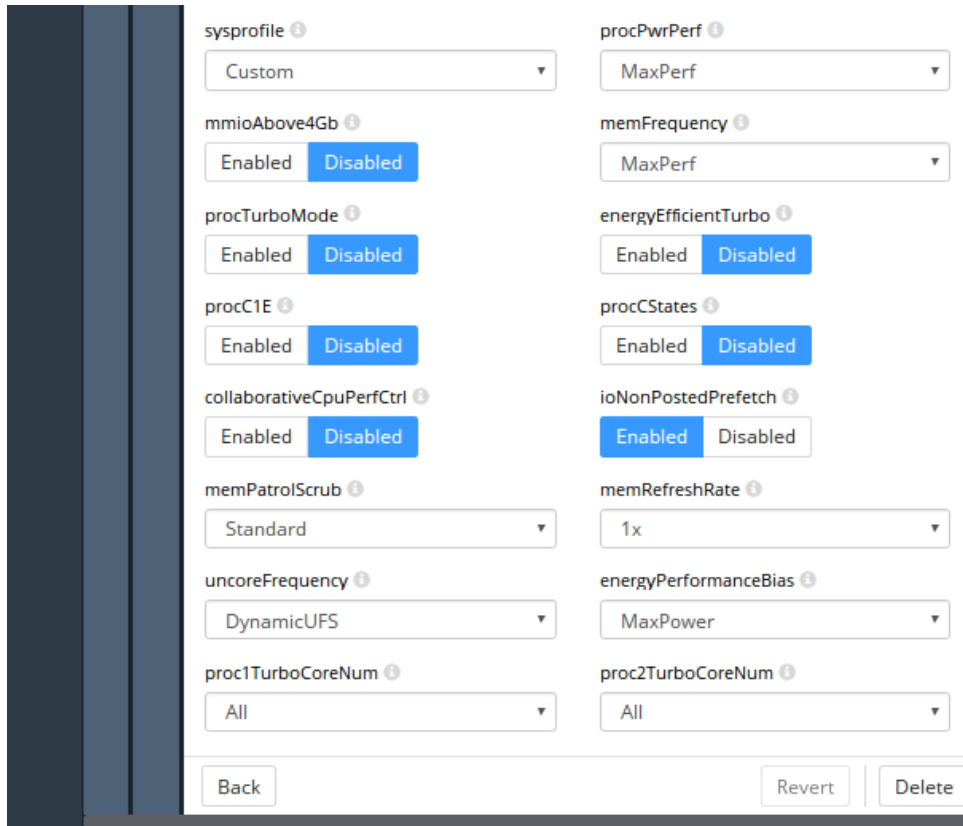


Figure 16.3: Performance profiles, processor/memory BIOS settings in the Dell Settings subwindow

The devicefirmware Subwindow

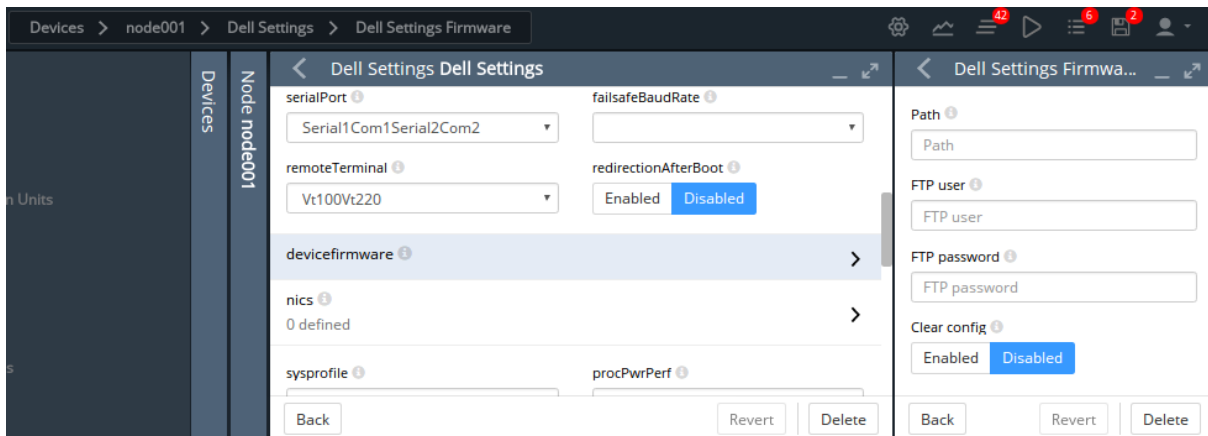


Figure 16.4: Dell settings firmware window

The devicefirmware subwindow (figure 16.4) 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 devicefirmware subwindow 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 racdm update command. The racdam 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.

16.3.3 Applying Dell Settings And Firmware Updates Via Bright View

To save modifications to the Dell settings, the usual way of saving the properties is followed. In this case it is the Save button in the window that opens up the Dell Settings subwindow, that is used to save the properties.

16.3.4 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

The Dell BIOS properties can be viewed with the `show` command run within the submode:

Example

```
[bright90->category[default]->dellsettings]% show
```

Parameter	Value
Revision	
applyImmediately	no
biosBootOrder	
collaborativeCpuPerfCtrl	no
devicefirmware	<submode>
energyEfficientTurbo	no
energyPerformanceBias	MaxPower
failsafeBaudRate	115200
ioNonPostedPrefetch	yes
l3Cache	no
logicalProcessor	no
memFrequency	MaxPerf
memPatrolScrub	Standard
memRefreshRate	1x
mmioAbove4Gb	no
modelName	
monitorMWait	no
nics	<0 in submode>
nodeInterleaving	no
numberOfCores	
proc1TurboCoreNum	All
proc2TurboCoreNum	All
procC1E	no
procCStates	no
procPwrPerf	MaxPerf
procTurboMode	no
redirectionAfterBoot	no

```

remoteTerminal          Vt100Vt220
serialCommunication     Off
serialPort             Serial1Com1Serial2Com2
serviceTag
snoopMode              HomeSnoop
sysprofile             Custom
turboBoost             no
uncoreFrequency        DynamicUFS

```

BIOS: Setting Properties

Tab-completion suggestions for the set show the possible properties that can be modified.

Example

```

[bright90->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

```

[bright90->category[default]->dellsettings]% set monitormwait <TAB><TAB>
no yes
[bright90->category[default]->dellsettings]% set monitormwait yes
[bright90->category*[default]*->dellsettings]% commit

```

BIOS: Importing Properties With `importlive`

The `importlive` command imports the Dell settings for a category from an existing node configuration.

Example

```

[bright90->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

```

```

[bright90->category*[default]*->dellsettings]% importlive node001
[bright90->category*[default]*->dellsettings]% commit
[bright90->category[default]->dellsettings]%

```

Firmware Updates

Firmware update configurations can be listed and set within the `firmware` submode:

Example

```
[bright90->category*[default*]->dellsettings]% firmware
[bright90->category*[default*]->dellsettings->firmware]% show
Parameter          Value
-----
Readonly           no
Revision
clearConfig       no
ftpPassword
ftpUser
path
```

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

```
[bright90->device[node001]]% applydellsettings
[bright90->device[node001]]% task list
```

Details for each task can be shown with the `show` command, run from within the task submode.

Example

```
[bright90->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

```
[bright90->task]% show 1
...
Log: ERROR: Unable to connect to RAC at specified IP address.
```

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

```
[bright90->device[node001]->dellsettings]% showlive
[bright90->device[node001]->dellsettings]% showlive -f
Parameter          Value
-----
Readonly           no
Revision
applyImmediately   no
biosBootOrder      NIC.Embedded.1-1-1, NIC.Mezzanine.1A-1,...
collaborativeCpuPerfCtrl no
```

```

devicefirmware          <submode>
energyEfficientTurbo    no
energyPerformanceBias   MaxPower
failsafeBaudRate        115200
l3Cache                 yes
logicalProcessor        no
memFrequency            MaxPerf
memPatrolScrub          Standard
memRefreshRate          1x
mmioAbove4Gb           no
modelName
monitorMWait           yes
nics                    <2 in submode>
nodeInterleaving        no
numberOfCores           All
proc1TurboCoreNum       All
proc2TurboCoreNum       All
procC1E                 no
procCStates             no
procPwrPerf             MaxPerf
procTurboMode           no
redirectionAfterBoot    yes
remoteTerminal          Vt100Vt220
serialCommunication     OnConRedirAuto
serialPort               Serial1Com2Serial2Com1
serviceTag
snoopMode               ClusterOnDie
sysprofile              PerfOptimized
turboBoost              no
uncoreFrequency         DynamicUFS
[bright90->device[node001]->dellsettings]%

```

16.4 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 the Bright View display of the BIOS setting display nothing?

This is likely because the required information is not fetched using `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 689). It is generally a good idea to run that command if unsure about the status of current and planned BIOS settings for the next boot.

```
[bright90->device[node001]->dellsettings]% showlive -f
```

3. Why do the Bright View Dell settings 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 using `cmsh` with the `importlive` command. If there is no live information for the node yet, then the command `importlive -f` can be used:

Example

```
[bright90->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 Settings window of the node or category in Bright View:

Devices→Nodes→Edit→Settings→BMC settings

Grouping→Node Categories→Settings→BMC settings
or from each node or base partition submode in `cmsh`.

A

Generated Files

This appendix contains lists of system configuration files that are managed by CMDaemon, and system configuration files that are managed by node-installer. These are files created or modified on the head nodes (section A.1), in the software images (section A.2), and on the regular nodes (section A.3). These files should not be confused with configuration files that are merely installed (section A.4).

Section 2.6.5 describes how system configuration files on all nodes are written out using the Cluster Management Daemon (CMDaemon). CMDaemon is introduced in section 2.6.5 and its configuration directives are listed in Appendix C.

All of these configuration files may be listed as `Frozen Files` in the CMDaemon configuration file to prevent them from being modified further by CMDaemon. The files can be frozen for the head node by setting the directive at `/cm/local/apps/cmd/etc/cmd.conf`. They can also be frozen on the regular nodes by setting the directive in the software image, by default at `/cm/images/default-image/cm/local/apps/cmd/etc/cmd.conf`.

A.1 System Configuration Files Created Or Modified By CMDaemon On Head Nodes

A list of CMDaemon-managed files on the head node can be seen by running the script `/cm/local/apps/cmd/scripts/cm-list-managed-files` on the head node. The head node files that are managed by CMDaemon are listed here for a plain installation on RHEL7 and derivatives. The list excludes the files managed by CMDaemon in the software images used for regular nodes (section A.2), which are by default located on the head node under `/cm/images/`:

Files generated or modified automatically on the head node by CMDaemon In RHEL7:

File	Part	Comment
<code>/cm/local/apps/openldap/etc/slapd.conf</code>	Section	
<code>/cm/node-installer/etc/sysconfig/clock</code>	Section	
<code>/etc/aliases</code>	Section	
<code>/etc/dhcpd.conf</code>	Entire file	

... continues

...continued

File	Part	Comment
/etc/dhclient.conf	Entire file	
/etc/dhcpd.internalnet.conf	Entire file	For internal networks other than internalnet, corresponding files are generated if node booting (table 3.2.1) is enabled
/etc/exports	Section	
/etc/fstab	Section	
/etc/genders	Section	
/etc/hosts	Section	
/etc/localtime	Symlink	
/etc/named.conf	Entire file	For zone additions use /etc/named.conf.include ¹ . For options additions, use /etc/named.conf.global.options.include
/etc/ntp.conf	Section	
/etc/ntp/step-tickers	Section	
/etc/postfix/canonical	Section	
/etc/postfix/generic	Section	
/etc/postfix/main.cf	Section	
/etc/resolv.conf	Section	
/etc/shorewall6/interfaces	Section	
/etc/shorewall6/policy	Section	
/etc/shorewall6/zones	Section	
/etc/shorewall/interfaces	Section	
/etc/shorewall/masq	Section	
/etc/shorewall/netmap	Section	
/etc/shorewall/policy	Section	
/etc/shorewall/zones	Section	
/etc/sysconfig/bmccfg	Entire file	BMC configuration
/etc/sysconfig/clock	Section	
/etc/sysconfig/dhcpd	Entire file	
/etc/sysconfig/network-scripts/ ifcfg-*	Section	

...continues

...continued

File	Part	Comment
/tftpboot/mtu.conf	Entire file	Bright Computing configuration
/tftpboot/pxelinux.cfg/category.default	Entire file	Bright Computing configuration
/var/named/*.zone ¹	Entire file	For custom additions use /var/named/*.zone.include

¹ User-added zone files ending in *.zone that are placed for a corresponding zone statement in the include file /etc/named.conf.include are wiped by CMDaemon activity. Another pattern, eg: *.myzone, must therefore be used instead

A.2 System Configuration Files Created Or Modified By CMDaemon In Software Images:

The list of CMDaemon-managed system configuration files in the software images, that is, under /cm/images/<image>, can be seen by running cm-list-managed-files and filtering the list appropriately. For example using `| grep /cm/images`.

For the default image, under /cm/images/default-image, for RHEL7, the list is:

System configuration files created or modified by CMDaemon in software images in RHEL7

File	Part	Comment
/etc/aliases	Section	
/etc/hosts	Section	
/etc/mkinitrd_cm.conf	Section	
/etc/modprobe.d/bright-cmdaemon.conf	Section	
/etc/postfix/main.cf	Section	
/etc/securetty	Section	
/etc/sysconfig/clock	Section	

A.3 Files Created On Regular Nodes By The Node-Installer

The list of files on the regular node that are generated or modified by the node-installer can be viewed in the logs at /var/log/modified-by-node-installer.log on the node. For a default installation the result in RHEL7 is shown in the following table:

System configuration files created or modified on regular nodes by the node-installer in RHEL7

File	Part	Comment
/etc/sysconfig/network-scripts/ifcfg-*	Entire file	not ifcfg-log
/etc/sysconfig/network	Section	
/etc/resolv.conf	Entire file	
/etc/hosts	Section	
/etc/postfix/main.cf	Section	
/etc/ntp.conf	Entire file	
/etc/ntp/step-tickers	Entire file	
/etc/hostname	Section	
/cm/local/apps/cmd/etc/cert.pem	Section	
/cm/local/apps/cmd/etc/cert.key	Section	
/cm/local/apps/openldap/etc/certs/ldap.pem	Section	
/cm/local/apps/openldap/etc/certs/ldap.key	Section	
/var/spool/cmd/disks.xml	Section	
/cm/local/apps/cmd/etc/cluster.pem	Section	
/var/log/rsyncd.log	Section	
/var/log/node-installer	Section	
/var/log/modified-by-node-installer.log	Section	

A.4 Files Not Generated, But Installed In RHEL

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.

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@bright90 ~]# repoquery -l $(repoquery -a | grep -F _cm9.0) | grep nsswitch.conf$
```

The inner `repoquery` displays a list of all the packages. By grepping for the cluster manager version string, for example `_cm9.0` for Bright Cluster Manager 9.0, 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@bright90 ~]# repoquery -l $(repoquery -a | grep -F -v _cm9.0) | grep nsswitch.conf$
/etc/nsswitch.conf
/etc/nsswitch.conf
/cm/node-installer/etc/nsswitch.conf
/usr/share/rear/skel/default/etc/nsswitch.conf
/usr/share/doc/yp-tools-2.14/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
...
cm-config-ldap-client<various types and versions are seen>
```

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 the less specific case:

```
~# yum whatprovides */etc/nsswitch.conf
...
cm-config-ldap-client*
glibc*
node-installer-nfsroot*
rear*
```

shows that `glibc` provides a distribution version of the `nsswitch.conf` file, that the `rear` package from the distribution provides a version of it, and that there is also a `node-installer` version of this file available from the Bright Cluster Manager packages. The glob `*` in the output in this manual represents a variety of types and versions. The actual display that is seen on the screen an expansion of the glob.

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:

```

~# mkdir yumextracted ; cd yumextracted
~# yumdownloader glibc-2.17
~# rpm2cpio glibc-2.17-292.el7.x86_64.rpm | cpio -idmv
~# yumdownloader cm-config-ldap-client-9.0
~# rpm2cpio cm-config-ldap-client-9.0-114_cm9.0.noarch.rpm
~# diff etc/nsswitch.conf cm/conf/etc/nsswitch.conf
...

```

What are the configuration files in an RPM package? An RPM package allows files within it to be marked as configuration files. Files marked as configuration files can be listed with `rpm -qc <package>`. Optionally, piping the list through “`sort -u`” filters out duplicates.

Example

```

~# rpm -qc glibc | sort -u
/etc/gai.conf
/etc/ld.so.cache
/etc/ld.so.conf
/etc/nsswitch.conf
/etc/rpc
/usr/lib64/gconv/gconv-modules
/usr/lib/gconv/gconv-modules
/var/cache/ldconfig/aux-cache

```

How does an RPM installation deal with local configuration changes? Are there configuration files or critical files that 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 frozen¹. 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:

```

rpm -q --queryformat '%{FILENAMES}\t%{FILEFLAGS:fflags}\n' glibc | egr\
ep '[:,space:]].*(c|n).*$' | sort -u
/etc/gai.conf      cmng
/etc/ld.so.cache   cmng
/etc/ld.so.conf    cn
/etc/nsswitch.conf  cn
/etc/rpc           cn
/usr/lib64/gconv/gconv-modules  cn
/usr/lib/gconv/gconv-modules     cn
/var/cache/ldconfig/aux-cache    cmng

```

Here, the second column of the output displayed shows which of the files in the package have a configuration (c) flag or a noreplace (n) flag. The c flag without the n flag indicates that an `.rpmsave` file will be created, while a c flag together with an n flag indicates that an `.rpmnew` file will be created.

In any case, files that are not marked as configuration files are overwritten during installation.

So:

¹This freezing should not be confused with the `FrozenFile` directive (Appendix C), where the file or section of a file is being maintained by `CMDaemon`, and where freezing the file prevents `CMDaemon` from maintaining it.

- 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 2.6.5), or checked against the list of generated files (Appendix A). This is just to make sure to avoid confusion about how changes are occurring in such a file.
 - If it is a `CMDaemon`-maintained file, then configuration changes put in by the administrator will also not persist in the maintained section of the file unless the `FrozenFile` directive (section C) is used to freeze the change.
 - If it is only a section that `CMDaemon` maintains, then configuration changes can be placed outside of the maintained section.

Wherever the changes are placed in such a file, these changes are in any case by default overwritten on RPM updates if the file has no `n` flag.

- Some regular node updates can effectively be maintained in a desired state with the help of a `finalize` script (Appendix E).
- Updates can be excluded from YUM/zypper (section 11.3.2), 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.

B

Bright Computing Public Key

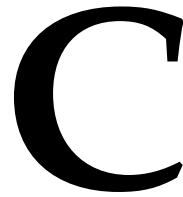
-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v1.4.0 (GNU/Linux)

```
mQGibEqtYegRbADStdQjn1XxbYorXbFGncF2IcMFiNA7hamArt4w7hjtWZoKGhbC
zSLsQTmgZO+FZs+tXcZa50LjGwhpxT6qhCe8Y7zIh2vwKrKlaAVKj2PUU28vKj1p
2W/OIiG/HKLTahLiCk0L3ahPOevJHh8B7e1ClrZ0TKTBb6qIUbC5vHtjiwCgydm3
THLJsKnwk4qZetluTupld0EEANCzJ1nZxZzN6ZAMkIBrct8GivWC1T1nBG4UwjHd
EDcG1REJxpg/OhpEP8TY1e0YUKRWvMqSVChPzkLUTIsd/04RGTw0PGCo6Q3TLXpM
RVoonyPR1tRymPNZyW8VJeTUEn0kd1CaqZykp1sRb3jFAiJIRcMBrc854i/jRXmo
foTPBACJQyoEH9Qfe3VcqR6+vR2tX91PvKxS7A5AnJIRs3Sv6yM4oV+7k/HrfYKt
fy16widtEbQ1870s4x3NYXmmne71z1nGxBfAxzPG9rtjRSXyVxc+KGVd6gKeCV6d
o7ks/LJHRiOLb5G4NZRFy5CGqg641iJwp/f2J4uyRbC8b+/LQbQ7QnJpZ2h0IENv
bXB1dG1uZyBEZXZ1bG9wbWVudCBUZWFtIDxkZXZAYnJpZ2h0Y29tcHV0aW5nLmNv
bT6IXgQTEQIAHGUCSqi1h6A1bAwYLCQgHAWIDFQIDAxYCAQIEAQIXgAAKCRDvaS9m
+k3m0J00AKCOGLTZiqoCQ6TRWW2ijjITEQ8CXACgg3o4oVbrG67VFzHUntcAOYTE
DXW5Ag0ESq1h6xAIAMJiaZI/OEqnrhSfiMsMT3sxx3mZkrQQL82Fob7s+S7nm18
A8btPzL1K8NzZytCglrIwPCYG6vfza/nkvyKEPh/f2it941bh7qiu4rBLqr+kGx3
zepSMRqIzW5FpIrUgDZOL9J+twSSUtPW0YQ5jBBJrgJ8LQy9dK2RhaOLuHfb0SVB
JLIwNKxafkhMRwDoUNs4BiZKWypFu47vd8fM67IPT1nM10iCOR/QBn29MYuWnBcw
61344pd/Ij0u3gM6YBqmRRU6yBeVi0TxxbYYnWcts6tEGAlTjHU0Q7gxVp4RDia2
jLVtbee8H464wxkkC3SSkng216RaBBAoaAykzhcAAwUH/iG4WsJHFw3+CRhUqy51
jnmb1FTF08KQXI8JLPXMOh6vv0PtP5rw5D5V2cyVe2i4ez9Y8Xmvfcbf601ptKyY
bRUjQq+9SNjt12ESU67YyLstSN68ach9Af03PoSZIKkiNwfAO+VBILv2Mhn7xd74
5L0M/eJ71HSpeJA2Rzs6szc2340b/VxGfGWjogaK3NELSY0zQo+/kOVMdMwsQm/8
Ras19IA9P5j1SbcZQ1H1PjndS4x4XQ8P41ATczsIDyWhsJC51rTuw9/Q07fqvvPn
xsRz1pFmiiN7I4JLjw0nAlXexn4EaeVa7Eb+uTjvxJZNdShs7Td740mlF7RKFccI
wLuISQQYEQIACQUCSqi1h6wIbDAAKCRDvaS9m+k3m0C/oAJshMmKrlPhjCdZyHbB1
e19+5JABUwCfU0PoawBNOHzDnfr3MLaTgCwjsEE=
```

=WJX7

-----END PGP PUBLIC KEY BLOCK-----



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 715.

Master directive

Syntax: `Master = hostname`

Default: `Master = master`

The cluster management daemon treats the host specified in the `Master` directive as the head node. A cluster management daemon running on a node specified as the head node starts in *head* mode. On a regular node, it starts in *node* mode.

Port directive

Syntax: `Port = number`

Default: `Port = 8080`

The *number* used in the syntax above is a number between 0 and 65535. The default value is 8080.

The `Port` directive sets the value of the port of the cluster management daemon to listen for non-SSL HTTP calls. By default, this happens only during `init`. All other communication with the cluster

management daemon is carried out over the SSL port. Pre-init port adjustment can be carried out in the `node-installer.conf` configuration. Shorewall may need to be modified to allow traffic through for a changed port.

SSLPort directive

Syntax: `SSLPort = number`

Default: `SSLPort = 8081`

The *number* used in the syntax above is a number between 0 and 65535. The default value is 8081.

The `SSLPort` directive sets the value of the SSL port of the cluster management daemon to listen for SSL HTTP calls. By default, it is used for all communication of CMDaemon with Bright View and `cmsh`, except for when CMDaemon is started up from `init`.

SSLPortOnly directive

Syntax: `SSLPortOnly = yes|no`

Default: `SSLPortOnly = no`

The `SSLPortOnly` directive allows the non-SSL port to be disabled. By default, during normal running, both SSL and non-SSL ports are listening, but only the SSL port is used. Also by default, the non-SSL port is only used during CMDaemon start up.

If `bootloaderprotocol` (section 5.1.6) is set to HTTP, then `SSLPortOnly` must be set to `no`. The HTTPS protocol is unsupported by most bootloaders.

CertificateFile directive

Syntax: `CertificateFile = filename`

Default: `CertificateFile = "/cm/local/apps/cmd/etc/cert.pem"`

The `CertificateFile` directive specifies the PEM-format certificate which is to be used for 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.

ClusterCertificateFile directive

Syntax: `ClusterCertificateFile = filename`

Default: `ClusterCertificateFile = "/cm/local/apps/cmd/etc/cluster.pem"`

The `ClusterCertificateFile` directive specifies the path to the Bright Cluster Manager PEM-format cluster certificate file used as a software license, and to sign all client certificates

ClusterPrivateKeyFile directive

Syntax: `ClusterPrivateKeyFile = filename`

Default: `ClusterPrivateKeyFile = "/cm/local/apps/cmd/etc/cluster.key"`

The `ClusterPrivateKeyFile` directive specifies the path to the Bright Cluster Manager PEM-format private key which corresponds to the cluster certificate file.

RandomSeedFile directive

Syntax: `RandomSeedFile = filename`

Default: `RandomSeedFile = "/dev/urandom"`

The `RandomSeedFile` directive specifies the path to a source of randomness for a random seed.

RandomSeedFileSize directive

Syntax: `RandomSeedFileSize = number`

Default: `RandomSeedFileSize = 8192`

The `RandomSeedFileSize` directive specifies the size of the random seed.

DHParamFile directive

Syntax: `DHParamFile = filename`

Default: `DHParamFile = "/cm/local/apps/cmd/etc/dh1024.pem"`

The `DHParamFile` directive specifies the path to the Diffie-Hellman parameters.

SSLHandshakeTimeout directive

Syntax: `SSLHandshakeTimeout = number`

Default: `SSLHandshakeTimeout = 10`

The `SSLHandshakeTimeout` directive controls the time-out period (in seconds) for SSL handshakes.

SSLSessionCacheExpirationTime directive

Syntax: `SSLSessionCacheExpirationTime = number`

Default: `SSLSessionCacheExpirationTime = 300`

The `SSLSessionCacheExpirationTime` directive controls the period (in seconds) for which SSL sessions are cached. Specifying the value 0 can be used to disable SSL session caching.

DBHost directive

Syntax: `DBHost = hostname`

Default: `DBHost = "localhost"`

The DBHost directive specifies the hostname of the MySQL database server.

DBPort directive

Syntax: DBPort = *number*

Default: DBHost = 3306

The DBPort directive specifies the TCP port of the MySQL database server.

DBUser directive

Syntax: DBUser = *username*

Default: DBUser = cmdaemon

The DBUser directive specifies the username used to connect to the MySQL database server.

DBPass directive

Syntax: DBPass = *password*

Default: DBPass = "<random string set during installation>"

The DBPass directive specifies the password used to connect to the MySQL database server.

DBName directive

Syntax: DBName = *database*

Default: DBName = "cmdaemon"

The DBName directive specifies the database used on the MySQL database server to store CMDaemon related configuration and status information.

DBUnixSocket directive

Syntax: DBUnixSocket = *filename*

Default: DBUnixSocket = "/var/lib/mysql/mysql.sock"

The DBUnixSocket directive specifies the named pipe used to connect to the MySQL database server if it is running on the same machine.

DBUpdateFile directive

Syntax: DBUpdateFile = *filename*

Default: DBUpdateFile = "/cm/local/apps/cmd/etc/cmdaemon_upgrade.sql"

The DBUpdateFile directive specifies the path to the file that contains information on how to upgrade the database from one revision to another.

EventBucket directive

Syntax: EventBucket = *filename*

Default: EventBucket = "/var/spool/cmd/eventbucket"

The `EventBucket` directive (section 12.11.3) 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 12.11.3) 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 "rogroun" group, whose members have a read-only access to the whole LDAP directory.

LDAPReadOnlyPass directive

Syntax: `LDAPReadOnlyPass = password`

Default: `LDAPReadOnlyPass = "<random string set during installation>"`

The `LDAPReadOnlyPass` directive specifies the password that will be used when connecting to the LDAP server during LDAP replication.

LDAPSearchDN directive

Syntax: `LDAPSearchDN = dn`

Default: LDAPSearchDN = "dc=cm,dc=cluster"

The LDAPSearchDN directive specifies the Distinguished Name (DN) used when querying the LDAP server.

LDAPProtocol directive

Syntax: LDAPProtocol = ldap|ldaps

Default: LDAPProtocol = "ldaps"

The LDAPProtocol directive specifies the LDAP protocol to be used when querying the LDAP server.

LDAPPort directive

Syntax: LDAPPort = *number*

Default: LDAPPort = 636

The LDAPPort directive specifies the port to be used when querying the LDAP server.

LDAPCACertificate directive

Syntax: LDAPCACertificate = *filename*

Default: LDAPCACertificate = "/cm/local/apps/openldap/etc/certs/ca.pem"

The LDAPCACertificate directive specifies the CA certificate to be used when querying the LDAP server.

LDAPCertificate directive

Syntax: LDAPCertificate = *filename*

Default: LDAPCertificate = "/cm/local/apps/openldap/etc/certs/ldap.pem"

The LDAPCertificate directive specifies the LDAP certificate to be used when querying the LDAP server.

LDAPPrivateKey directive

Syntax: LDAPPrivateKey= *filename*

Default: LDAPPrivateKey= "/cm/local/apps/openldap/etc/certs/ldap.key"

The LDAPPrivateKey directive specifies the LDAP key to be used when querying the LDAP server.

HomeRoot directive

Syntax: HomeRoot = *path*

Default: HomeRoot = "/home"

The HomeRoot directive specifies the default user home directory used by CMDaemon. It is used for automatic mounts, exports, and when creating new users.

DocumentRoot directive

Syntax: DocumentRoot = *path*

Default: DocumentRoot = "/cm/local/apps/cmd/etc/htdocs"

The DocumentRoot directive specifies the directory mapped to the web-root of the CMDaemon. The CMDaemon acts as a HTTP-server, and can therefore in principle also be accessed by web-browsers.

SpoolDir directive

Syntax: SpoolDir = *path*

Default: SpoolDir = "/var/spool/cmd"

The SpoolDir directive specifies the directory which is used by the CMDaemon to store temporary and semi-temporary files.

EnableShellService directive

Syntax: EnableShellService = true|false

Default: EnableShellService = true

The EnableShellService directive allows shells to be started from Bright View.

The connection runs over CMDaemon, which is running over SSL, which means that between Bright View and the device, the connection is encrypted.

The directive does not affect cmsh's rshell, rconsole telnet, and ssh commands.

EnableWebSocketService directive

Syntax: EnableWebSocketService = true|false

Default: EnableWebSocketService = true

The EnableWebSocketService directive allows the use of CMDaemon Lite (section 2.6.7).

EnablePrometheusMetricService directive

Syntax: EnablePrometheusMetricService = true|false

Default: EnablePrometheusMetricService = true

If true, the EnablePrometheusMetricService directive creates an HTTP endpoint for Prometheus-style exporters that do not have their own HTTP endpoint.

PrometheusMetricServicePath directive

Syntax: PrometheusMetricServicePath = *path*

Default: PrometheusMetricServicePath = SCRIPTS_DIR"metrics/prometheus"

The PrometheusMetricServicePath directive is the path from which CMDaemon can serve Prometheus metrics. SCRIPTS_DIR is the stem path /cm/local/apps/cmd/scripts by default.

CMDaemonAudit directive

Syntax: CMDaemonAudit = yes|no

Default: CMDaemonAudit = no

When the CMDaemonAudit directive is set to yes, and a value is set for the CMDaemon auditor file with

the `CMDaemonAuditorFile` directive, then `CMDaemon` actions are time-stamped and logged in the `CMDaemon` auditor file.

CMDaemonAuditorFile directive

Syntax: `CMDaemonAuditorFile = path to audit.log file`
or

Syntax: `CMDaemonAuditorFile = path to audit.json file`
or

Syntax: `CMDaemonAuditorFile = http(s) path to JSON server`

Default: `CMDaemonAuditorFile = "/var/spool/cmd/audit.log"`

The `CMDaemonAuditorFile` directive sets where the audit logs for `CMDaemon` actions are logged. The log format for a `.log` file in a standard directory path is:
(time stamp) profile [IP-address] action (unique key)

Example

```
(Mon Jan 31 12:41:37 2011) Administrator [127.0.0.1] added Profile: arbitprof(4294967301)
```

The directive can be set in the following kinds of formats:

- `CMDaemonAuditorFile = "/var/spool/cmd/audit.log"`
- `CMDaemonAuditorFile = "/var/spool/cmd/audit.json"`
- `CMDaemonAuditorFile = "http://<IP address>:<port>/some/path"`
- `CMDaemonAuditorFile = "https://<IP address>:<port>/some/path"`

A simple POST web service can be faked using `netcat`:

Example

```
nc -l 1234 -k
```

The JSON file always contains a valid array. An RPC call looks like this:

Example

```
...
"entity": "node001",
"rpc":
  "address": "127.0.0.1",
  "call": "updateDevice",
  "service": "cmdevice",
  "timestamp": 1579185696,
  "user": "Administrator.root"
,
"task_id": 0,
"updated": true
,
...
```


DisableAuditorForProfiles directive

Syntax: `DisableAuditorForProfiles = {profile [, profile] ...}`

Default: `DisableAuditorForProfiles = {NODE}`

The `DisableAuditorForProfiles` directive sets the profile for which an audit log for `CMDaemon` actions is disabled. A profile (section 2.3.4) defines the services that `CMDaemon` provides for that profile user. More than one profile can be set as a comma-separated list. Out of the profiles that are available on a newly-installed system: `node`, `admin`, `cmhealth`, and `readonly`; only the profile `node` is enabled by default. New profiles can also be created via the `profile` mode of `cmsh` or via the `clickpath Identity Management → Profiles →` of Bright View, thus making it possible to disable auditing for arbitrary groups of `CMDaemon` services.

EventLogger directive

Syntax: `EventLogger = true|false`

Default: `EventLogger = false`

The `EventLogger` directive sets whether to log events. If active, then by default it logs events to `/var/spool/cmd/events.log` on the active head. If a failover takes place, then the event logs on both heads should be checked and merged for a complete list of events.

The location of the event log on the filesystem can be changed using the `EventLoggerFile` directive (page 711).

Whether events are logged in files or not, events are cached and accessible using `cmsh` or Bright View. The number of events cached by `CMDaemon` is determined by the parameter `MaxEventHistory` (page 711).

EventLoggerFile directive

Syntax: `EventLoggerFile = filename`

Default: `EventLoggerFile = "/var/spool/cmd/events.log"`

The `EventLogger` directive sets where the events seen in the event viewer (section 12.11.1) are logged.

MaxEventHistory directive

Syntax: `AdvancedConfig = {"MaxEventHistory=number", ...}`

Default: `MaxEventHistory=8192`

`MaxEventHistory` is a parameter of the `AdvancedConfig` (page 720) 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 711) to `true`, to activate logging to a file, is advised instead.

TimingOverview directive

Syntax: `TimingOverview = filename`

Default: `TimingOverview = true|false`

If set to `true`, the `TimingOverview` directive records timing information for `CMDaemon`.

TimingOverviewFile directive

Syntax: TimingOverviewFile = *filename*

Default: TimingOverviewFile = "/var/spool/cmd/timing.overview.log"

The TimingOverviewFile directive sets the file where the timing data values for CMDaemon go.

PublicDNS directive

Syntax: PublicDNS = true|false

Default: PublicDNS = false

By default, internal hosts are resolved only if requests are from the internal network. Setting PublicDNS to true allows the head node name server to resolve internal network hosts for any network, including networks that are on other interfaces on the head node. Separate from this directive, port 53/UDP must also be opened up in Shorewall (section 7.2 of the *Installation Manual*) if DNS is to be provided for queries from an external network.

MaximalSearchDomains directive

Syntax: GlobalConfig = {"MaximalSearchDomains = *number*", ...}

Default: none

The MaximalSearchDomains directive is a parameter of the GlobalConfig (page 721) directive.

By default, the number of names that can be set as search domains used by the cluster has a maximum limit of 6. This is a hardcoded limit imposed by the Linux operating system in older versions.

More recent versions of glibc (glibc > 2.17-222.el7 in RHEL7) no longer set a limit. However using more than 6 search domains currently requires the use of the GlobalConfig directive, MaximalSearchDomains. For example, to set 30 domains, the directive setting would be: GlobalConfig = { "MaximalSearchDomains=30" }

LockDownDhcpd directive

Syntax: LockDownDhcpd = true|false

Default: LockDownDhcpd = false

LockDownDhcpd is a deprecated legacy directive. If set to true, a global DHCP "deny unknown-clients" option is set. This means no new DHCP leases are granted to unknown clients for all networks. Unknown clients are nodes for which 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 Bright View (section 3.2.1, figure 3.5, table 3.2.1). Setting the `cmd.conf` LockDownDhcpd directive overrides `lockdowndhcpd` values set by `cmsh` or Bright View.

MaxNumberOfProvisioningThreads directive

Syntax: MaxNumberOfProvisioningThreads = *number*

Default: MaxNumberOfProvisioningThreads = 10000

The MaxNumberOfProvisioningThreads directive specifies the cluster-wide total number of nodes that can be provisioned simultaneously. Individual provisioning servers typically define a much lower

bound on the number of nodes that may be provisioned simultaneously.

SetupBMC directive

Syntax: SetupBMC = true|false

Default: SetupBMC = true

Configure the username and password for the BMC interface of the head node and regular nodes automatically. (This should not be confused with the `setupBmc` field of the node-installer configuration file, described in section 5.8.7.)

BMCSessionTimeout directive

Syntax: BMCSessionTimeout = *number*

Default: BMCSessionTimeout = 2000

The BMCSessionTimeout specifies the time-out for BMC calls in milliseconds.

BMCIdentifyScript directive

Syntax: AdvancedConfig = {"BMCIdentify=*filename*", ...}

Default: *unset*

BMCIdentifyScript is a parameter of the AdvancedConfig (page 720) directive.

The parameter takes a full file path to a script that can be used for identification with a BMC (section 3.7.4).

BMCIdentifyScriptTimeout directive

Syntax: AdvancedConfig = {"BMCIdentifyScriptTimeout=*number from 1 to 360*", ...}

Default: 60

BMCIdentifyScriptTimeout is a parameter of the AdvancedConfig (page 720) directive.

CMDaemon waits at the most BMCIdentifyScriptTimeout seconds for the script used by the BMCIdentify directive to complete.

BMCIdentifyCache directive

Syntax: AdvancedConfig = {"BMCIdentifyCache=0|1", ...}

Default: 1

BMCIdentifyCache is a parameter of the AdvancedConfig (page 720) directive.

If set to 1, then CMDaemon remembers the last value of the output of the script used by the BMCIdentify directive.

SnmpSessionTimeout directive

Syntax: SnmpSessionTimeout = *number*

Default: SnmpSessionTimeout = 500000

The SnmpSessionTimeout specifies the time-out for SNMP calls in microseconds.

PowerOffPDUOutlet directive

Syntax: PowerOffPDUOutlet = true|false

Default: PowerOffPDUOutlet = false

Enabling the `PowerOffPDUOutlet` directive allows PDU ports to be powered off for clusters that have both PDU and IPMI power control. Section 4.1.3 has more on this.

PowerThreadPoolSize directive

Syntax: `AdvancedConfig = {"PowerThreadPoolSize=<integer>", ...}`

Default: `PowerThreadPoolSize=32`

`PowerThreadPoolSize` is a parameter of the `AdvancedConfig` (page 720) directive.

The parameter can take positive integer values. Increasing its value increases the number of threads that are used to power up the nodes in a cluster (section 4.2.3), so that the cluster is fully operational quicker. The administrator should however take into account the power surge due to increasing the number of threads (number of subprocesses) before increasing the value beyond its default.

DisableBootLogo directive

Syntax: `DisableBootLogo = true|false`

Default: `DisableBootLogo = false`

When `DisableBootLogo` is set to `true`, the 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:

FreezeChangesToPBSPPro directive

FreezeChangesToUGConfig directive

FreezeChangesToSlurmConfig directive

FreezeChangesToLSFConfig directive

Syntax: `FreezeChangesTo<wlm>Config= true|false`

Default: `FreezeChangesTo<wlm>Config = false`

When `FreezeChangesTo<wlm>Config` is set to `true`, the CMDaemon running on that node does not make any modifications to the workload manager configuration for that node. Workload managers for which this value can be set are:

- PBSPPro
- UGE
- Slurm
- LSF

Monitoring of jobs, and workload accounting and reporting continues for frozen workload managers.

Upgrades to newer workload manager versions may still require some manual adjustments of the configuration file, typically if a newer version of the workload manager configuration changes the syntax of one of the options in the file.

FrozenFile directive

Syntax: `FrozenFile = { filename[, filename]... }`

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

The `FrozenFile` directive is used to prevent the CMDaemon-maintained sections of configuration files from being automatically generated. This is useful when site-specific modifications to configuration files have to be made.

To avoid problems, the file that is frozen should not be a symlink, but should be the ultimate destination file. The `readlink -f <symlinkname>` command returns the ultimate destination file of a symlink called `<symlinkname>`. This is also the case for an ultimate destination file that is reached via several chained symlinks.

FrozenFile directive for regular nodes

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 `Bright View` (section 5.6.3). After the update, CMDaemon should be restarted within that category of nodes:

Example

```
[root@bright90 ~]# 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`

For the `node-installer.conf` file in multidistro and multiarch (section 11.7) configurations, the directory path `/cm/node-installer` takes the form:

```
/cm/node-installer-<distribution>-<architecture>
```

The values for `<distribution>` and `<architecture>` can take the values outlined on page 472.

Example

Per node:

```
frozenFilesPerNode = "*/localdisk/etc/ntp.conf", "node003:/localdisk/etc/hosts"
```

Here, the `*` wildcard means that no restriction is set. Setting `node003` means that only `node003` is frozen.

Example

Per category:

```
frozenFilesPerCategory = "mycategory:/localdisk/etc/sysconfig/network-scripts/ifcfg-eth1"
```

Here, the nodes in the category `mycategory` are prevented from being changed by the node-installer.

The Necessity Of A FrozenFile Directive

In a configuration file after a node is fully up, the effect of a statement earlier on can often be overridden by a statement later in the file. So, the following useful behavior is independent of whether `FrozenFile` is being used for a configuration file or not: A configuration file, for example `/etc/postfix/main.cf`, with a configuration statement in an earlier CMDaemon-maintained part of the file, for example:

```
mydomain = eth.cluster
```

can often be overridden by a statement later on outside the CMDaemon-maintained part of the file:

```
mydomain = eth.gig.cluster
```

Using `FrozenFile` in CMDaemon or the node-installer can thus sometimes be avoided by the use of such overriding statements later on.

Whether overriding later on is possible depends on the software being configured. It is true for Postfix configuration files, for example, but it may not be so for the configuration files of other applications.

EaseNetworkValidation directive

Syntax: `EaseNetworkValidation = 0|1|2`

Default: `EaseNetworkValidation = 0`

CMDaemon enforces certain requirements on network interfaces and management/node-booting networks by default. In heavily customized setups, such as is common in Type 3 networks (section 3.3.9 of the *Installation Manual*), the user may wish to disable these requirements.

- 0 enforces all requirements.
- 1 allows violation of the requirements, with validation warnings. This value should never be set except under instructions from 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
- /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_LOCALO..7

NameServerLocalhostLocation directive

Syntax: AdvancedConfig = {"NameServerLocalhostLocation=0|1", ...}

Default: NameServerLocalhostLocation=0

NameServerLocalhostLocation is a parameter of the AdvancedConfig (page 720) directive.

When set to 1, the location of the localhost as specified by the nameserver directive in /etc/resolv.conf is moved to the bottom of the list of nameserver entries. The default value of 0 places it at the top of those entries.

ResolveToExternalName directive

Syntax: ResolveToExternalName = true|false

Default: ResolveToExternalName = false

The value of the ResolveToExternalName directive determines under which domain name the primary and secondary head node hostnames are visible from within the head nodes, and to which IP addresses their hostnames are resolved. Enabling this directive resolves the head nodes' hostnames to the IP addresses of their external interfaces.

Thus, on head nodes and regular nodes in both single-head and failover clusters

- with ResolveToExternalName disabled, the master hostname and the actual hostname of the head node (e.g. head1, head2) by default always resolve to the internal IP address of the head node.
- with ResolveToExternalName enabled, the master hostname and the actual hostname of the head node (e.g. head1, head2) by default always resolve to the external IP address of the head node.

The resolution behavior can be summarized by the following table:

ResolveToExternalName Directive Effects

on simple head, resolving:		on failover head, resolving:			on regular node, resolving:		Using the DNS?
master	head	master	head1	head2	master	head(s)	
ResolveToExternalName = False							
I	I	I	I	I	I	I	No
I	I	I	I	I	I	I	Yes
ResolveToExternalName = True							
E	E	E	E	E	E	E	No
E	E	E	E	E	E	E	Yes

Key: I: resolves to internal IP address of head
E: resolves to external IP address of head

The system configuration files on the head nodes that get affected by this directive include /etc/hosts and, on SLES systems, also the /etc/HOSTNAME. Also, the DNS zone configuration files get affected.

Additionally, in both the single-head and failover clusters, using the "hostname -f" command on a head node while ResolveToExternalName is enabled results in the host's Fully Qualified Domain

Name (FQDN) being returned with the host's external domain name. That is, the domain name of the network that is specified as the "External network" in the base partition in `cmsh` (the output of "`cmsh -c "partition use base; get externalnetwork"`").

Modifying the value of the `ResolveToExternalName` directive and restarting the `CMDaemon` while important system services (e.g. `Slurm`) are running should not be done. Doing so is likely to cause problems with accessing such services due to them then running with a different domain name than the one with which they originally started.

On a tangential note that is closely, but not directly related to the `ResolveToExternalName` directive: the cluster can be configured so that the "`hostname -f`" command executed on a regular node returns the FQDN of that node, and so that the FQDN in turn resolves to an external IP for that regular node. The details on how to do this are in the Bright Cluster Manager Knowledge Base at <http://kb.brightcomputing.com/>. A search query for FQDN leads to the relevant entry.

ResolveMasterToExternalName directive

Syntax: `AdvancedConfig = {"ResolveMasterToExternalName=0|1", ...}`

Default: `ResolveMasterToExternalName = 1`

`ResolveMasterToExternalName` is a parameter of the `AdvancedConfig` (page 720) directive.

`ResolveMasterToExternalName` can only be used if `ResolveToExternalName` (page 718) is active.

If set to 1 (the default), then the master head node, as specified by the name `master`, resolves to the IP address as set by `ResolveToExternalName`.

If set to 0 then the master head node resolves to the internal shared IP address.

ResolveMasterToExternalDomainName directive

Syntax: `AdvancedConfig = {"ResolveMasterToExternalDomainName=0|1", ...}`

Default: `ResolveMasterToExternalDomainName = 1`

`ResolveMasterToExternalDomainName` is a parameter of the `AdvancedConfig` (page 720) directive.

`ResolveMasterToExternalDomainName` can only be used if `ResolveToExternalName` (page 718) is active.

The external domain name as defined for the external network, in the form `master.<external domain>`, can be used in `/etc/hosts` during name resolution.

If set to 1 (the default), then the external domain name is used in `/etc/hosts`.

If set to 0, then the external domain name is not used in `/etc/hosts`.

Resolution occurs as shown in the following table:

ResolveMasterToExternalDomainName Directive Effects

on simple head, resolving:		on failover head, resolving:			on regular node, resolving:		Using the DNS?
master	head	master	head1	head2	master	head(s)	
ResolveToExternalName = False							
I	I	I	I	I	I	I	No
I	I	I	I	I	I	I	Yes
ResolveToExternalName = True							
E	E	I	E	E	I	I	No
E	E	I	E	E	I	E	Yes

Key: I: resolves to internal IP address of head

E: resolves to external IP address of head

DisableLua directive

Syntax: `DisableLua = true|false`

Default: `DisableLua = false`

The value of the `DisableLua` directive determines if Lua code (section L.6) used in monitoring expressions can be executed.

AdvancedConfig directive

Syntax: `AdvancedConfig = { "<key1>=<value1>", "<key2>=<value2>", ... }`

Default: *Commented out in the default `cmd.conf` file*

The `AdvancedConfig` directive is not part of the standard directives. It takes a set of key/value pairs as parameters, with each key/value pair allowing a particular functionality, and is quite normal in that respect. However, the functionality of a parameter to this directive is often applicable only under restricted conditions, or for non-standard configurations. The `AdvancedConfig` parameters are therefore generally not recommended for use by the administrator, nor are they generally documented.

Like for the other directives, only one `AdvancedConfig` directive line is used. This means that whatever functionality is to be enabled by this directive, its corresponding parameters must be added to that one line. These key/value pairs are therefore added by appending them to any existing `AdvancedConfig` key/value pairs, which means that the directive line can be a long list of key/value pairs to do with a variety of configurations.

Managing Key/Value Pairs With The `cm-manipulate-advanced-config.py` Utility

The `cm-manipulate-advanced-config.py` utility can be used to make it easier to manage `AdvancedConfig` key/value pairs,

For example, to add a key/value pair `key8=value8`:

```
[root@bright90 ~]# cm-manipulate-advanced-config.py key8=value8
Updated: /cm/local/apps/cmd/etc/cmd.conf
```

To show the current state of the `AdvancedConfig`, the `-s|--show` option can be used:

```
[root@bright90 ~]# cm-manipulate-advanced-config.py -s
=== /cm/local/apps/cmd/etc/cmd.conf ===
VirtualCluster=1
key8=value8
```

A key/value pair can be removed by specifying its key with the `-r|--remove` option:

```
[root@bright90 ~]# cm-manipulate-advanced-config.py -r key8
Updated: /cm/local/apps/cmd/etc/cmd.conf
[root@bright90 ~]# cm-manipulate-advanced-config.py -s
=== /cm/local/apps/cmd/etc/cmd.conf ===
VirtualCluster=1
```

The utility can be used on `cmd.conf` in node images too, using the `-i|--image` option.

```
[root@bright90 ~]# cm-manipulate-advanced-config.py -i /cm/images/default-image
Updated: /cm/images/default-image//cm/local/apps/cmd/etc/cmd.conf
```

The `-q` option causes the utility to exit with code 1 if `cmd.conf` has changed.

Further options can be seen with the `-h|--help` option.

GlobalConfig directive

Syntax: `GlobalConfig = { "<key1>=<value1>", "<key2>=<value2>", ... }`

Default: *not in the default cmd.conf file*

The `GlobalConfig` directive is not part of the standard directives. It takes a set of key/value pairs as parameters, with each key/value pair allowing a particular functionality, and is quite normal in that respect. However, the parameter to this directive only needs to be specified on the head node. The non-head node `CMDaemons` take this value upon connection, which means that the `cmd.conf` file on the non-head nodes do not need to have this specified.

This allows nodes to set up, for example, their search domains using the `MaximalSearchDomains` `GlobalConfig` directive (page 712).

Like for the other directives, only one `GlobalConfig` directive line is used. This means that whatever functionality is to be enabled by this directive, its corresponding parameters must be added to that one line. These key/value pairs are therefore added by appending them to any existing `GlobalConfig` key/value pairs, which means that the directive line can be a long list of key/value pairs to do with a variety of configurations.

ScriptEnvironment directive

Syntax: `ScriptEnvironment = { "CMD_ENV1=<value1>", "CMD_ENV2=<value2>", ... }`

Default: *Commented out in the default cmd.conf file*

The `ScriptEnvironment` directive sets extra environment variables for `CMDaemon` and child processes.

For example, if `CMDaemon` is running behind a web proxy, then the environment variable `http_proxy` may need to be set for it. If, for example, the proxy is the host `brawndo`, and it is accessed via port 8080 using a username/password pair of `joe/electrolytes`, then the directive becomes:

```
ScriptEnvironment = { "http_proxy=joe:electrolytes@brawndo:8080" }
```

BurnSpoolDir directive

Syntax: `BurnSpoolDir = path`

Default: `BurnSpoolDir = "/var/spool/burn/"`

The `BurnSpoolDir` directive specifies the directory under which node burn log files are placed (Chapter 8 of the *Installation Manual*). The log files are logged under a directory named after the booting MAC address of the NIC of the node. For example, for a MAC address of `00:0c:29:92:55:5e` the directory is `/var/spool/burn/00-0c-29-92-55-5e`.

IdleThreshold directive

Syntax: `IdleThreshold = number`

Default: `IdleThreshold = 1.0`

The `IdleThreshold` directive sets a threshold value for `loadone`. If `loadone` exceeds this value, then data producers that have `Only when idle` (page 499) set to `true` (enabled), will not run. If the data producer is sampled on a regular node rather than on the head node, then `cmd.conf` on the regular node should be modified and its `CMDaemon` restarted.

MonitoringPath directive

Syntax: `AdvancedConfig = {"MonitoringPath=path", ...}`

Default: Implicit value: `"MonitoringPath=/var/spool/cmd/monitoring/"`

MonitoringPath is a parameter of the AdvancedConfig (page 720) directive.

Its value determines the path of the directory in which monitoring data is saved (section 13.8).

MaxServiceFailureCount directive

Syntax: AdvancedConfig = {"MaxServiceFailureCount=*number*", ...}

Default: Implicit value: "MaxServiceFailureCount=10"

MaxServiceFailureCount is a parameter of the AdvancedConfig (page 720) directive.

Its value determines the number of times a service failure event is logged (page 114). 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 720) 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 722), then the event is no longer logged, but the restart attempts continue.

Example

An fhgfs startup takes a bit longer than 30 seconds, and therefore times out with the default timeout value of 30s. This results in the following logs in /var/log/cmdaemon:

```
cmd: [SERVICE] Debug: ProgramRunner: /etc/init.d/fhgfs-client restart
[DONE] 0 9
cmd: [SERVICE] Debug: /etc/init.d/fhgfs-client restart, exitcode = 0,
signal = 9
```

Here, *service* is fhgfs-client, so setting the parameter can be done with:

```
AdvancedConfig = { ..., "initdScriptTimeout.fhgfs-client=60", ...}
```

This allows a more generous timeout of 60 seconds instead.

Restarting CMDaemon then should allow the fhgfs startup to complete

```
# service cmd restart
```

A more refined approach that avoids a complete CMDaemon restart would be to execute a `reset` (page 114) on the `fhgfs-client` from within CMDaemon, as follows:

```
[bright90->category[default]->services[fhgfs-client]]% reset fhgfs-client
Successfully reset service fhgfs-client on: node001,node002
[bright90->category[default]->services[fhgfs-client]]%
```

CMDaemonListenOnInterfaces directive

Syntax: `AdvancedConfig = {"CMDaemonListenOnInterfaces=<interfaces>", ...}`

Default: *all interfaces listening to port 8081*

`CMDaemonListenOnInterfaces` is a parameter of the `AdvancedConfig` (page 720) 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 720) directive.

It defines the number of seconds until the Bright View connection to CMDaemon times out, if there is no user activity at the Bright View client side.

OpenStackVXLANGroup directive

Syntax: `AdvancedConfig = {"OpenStackVXLANGroup=IP address", ...}`

Default: 224.0.0.1

`OpenStackVXLANGroup` is a parameter of the `AdvancedConfig` (page 720) directive.

It defines the multicast address used by OpenStack for broadcast traffic with VXLAN networking (page 30 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.

SlurmDisableAccountingParsing directive

Syntax: `AdvancedConfig = {"SlurmDisableAccountingParsing=0|1", ...}`

Default: 0

`SlurmDisableAccountingParsing` is a parameter of the `AdvancedConfig` (page 720) directive. If set to 1, it disables collection of accounting information for the Slurm workload manager.

JobsSamplingMetricsInterval directive

Syntax: AdvancedConfig = {"JobsSamplingMetricsInterval=<number>", ...}

Default: 60

JobsSamplingMetricsInterval is a parameter of the AdvancedConfig (page 720) directive. Its value is a time period, in seconds, and it applies only to metrics associated with job queues. Such metric sampling is carried out with this time period if job queues are added, or if job queues are re-created after disappearing.

MembershipQueryInterval directive

Syntax: AdvancedConfig = {"MembershipQueryInterval=<number>", ...}

Default: 4

MembershipQueryInterval is a parameter of the AdvancedConfig (page 720) directive. Its value is a time period, in seconds. This time period value elapses between the checks that CMDaemon makes to determine the node states (section 5.5) in a cluster. If the network is very congested, then a larger value can be used to reduce the network load caused by these checks.

AddUserScript directive

Syntax: AdvancedConfig = {"AddUserScript=<path>", ...}

Default: none

AddUserScript is a parameter of the AdvancedConfig (page 720) directive. If this parameter is set to a path leading to a script, and if a new user is added using cmsh or Bright View, then the script is automatically run by 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 720) 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 720) directive. If this parameter is set to a path leading to a script, and if an existing user is removed using cmsh or Bright View, then the script is automatically run by 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 720) 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 720) 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 720) 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 can be used with the SGE and UGE workload managers only, so that `CMDaemon` does not remove such hosts from the workload manager configuration file during a configuration file maintenance run. An example of their use is given on page 274.

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 720) directive. The values it takes decide how an auto-mounted filesystem should be dealt with during image updates (section 5.6.2) or grab image (section 11.5.2). It must be set per software image or per node.

Value	Description
0	If auto-mount is running, abort provisioning (default)
1	If auto-mount is running, warn but continue
2	Do not check auto-mount status. This saves a little time, but it risks data loss, unless the automounted filesystem has been added to <code>excludelistupdate</code> . If the automounted filesystem is not added to <code>excludelistupdate</code> , then if the auto-mounted filesystem happens to be unavailable at the time that an <code>imageupdate</code> is issued, then the <code>rsync</code> process can end up deleting the automounted filesystem contents during the <code>rsync</code> , because it assumes that content should not be there.
3	Pretend auto-mount is running. This prevents an image update

How an auto-mounted filesystem can be configured using the `autofs` service in Bright Cluster Manager is discussed in section 3.10. The need for adding the automounted filesystem to `excludelistupdate` is discussed on page 203.

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 720) 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 712) simply adds the entry `0.0.0.0/0` to the `allow-query` section and can be used if no specific subnet needs to be added.

CipherList directive

Syntax: `AdvancedConfig = {"CipherList=<ciphers>", ...}`

Default: `CipherList=ALL:!aNULL:!eNULL`

The `CipherList` directive is a parameter of the `AdvancedConfig` (page 720) 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@bright90 ~]# openssl ciphers -v
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH    Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH    Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH    Au=RSA  Enc=AES(256)  Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH    Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES256-SHA    SSLv3 Kx=ECDH    Au=RSA  Enc=AES(256)  Mac=SHA1
...
```

The columns are: the cipher name, SSL protocol, key exchange used, authentication mechanism, encryption used, and MAC digest algorithm.

Ciphers can be specified for the `CipherList` directive according to the specification described in man (1) `ciphers`. For example, as:

Example

```
AdvancedConfig = {"CipherList=ALL:!aNULL:!ADH:!eNULL:!LOW:!EXP:RC4+RSA:+HIGH:+MEDIUM"}
```


An analogous non-CMDaemon directive is found in LDAP client-server negotiation, where `TLSCipherSuite` can be set in the `slapd.conf` file, described in `man(5) slapd.conf`.

SSLServerMethod directive (with TLS masking)

Syntax: `AdvancedConfig = {"SSLServerMethod=TLS <versionnumber>", ...}`

Syntax: `GlobalConfig = {"SSLServerMethod=TLS <versionnumber>", ...}`

Masking syntax: `AdvancedConfig = {"TLS <versionnumber>=<boolean>", ...}`

Masking syntax: `GlobalConfig = {"TLS <versionnumber>=<boolean>", ...}`

Default: `GlobalConfig = { "TLS 1.2=1", "TLS 1.3=1", ... }`

The `SSLServerMethod` directive is a parameter of the `AdvancedConfig` (page 720) directive, as well as the `GlobalConfig` (page 721) directive. It sets the SSL server method for the OpenSSL suite that `CMDaemon` negotiates with clients. Possible values for `<versionnumber>` are:

- 1
- 1.1
- 1.2
- 1.3

By default, with no `SSLServerMethod` directive, TLS versions 1.2 and 1.3 are enabled.

This is because there are still some SSL clients that require a TLS 1.2 cipher and have no TLS 1.3 cipher negotiation ability.

If the `SSLServerMethod` directive is specified, then only that TLS version is negotiated. For example, to allow only TLS 1.2 negotiation:

Example

```
GlobalConfig = { "SSLServerMethod=TLS 1.2", ... }
```

The `SSLServerMethod` directive is implied by the TLS masking syntax. The TLS masking syntax allows TLS negotiation methods to be set concurrently. For example, TLS 1 and TLS1.1 could be disabled, and TLS 1.2 and TLS 1.3 could be enabled, with:

Example

```
AdvancedConfig = { "TLS 1=0", "TLS 1.1=0", "TLS 1.2=1", "TLS 1.3=1", ... }
```

JobInformationDisabled directive

Syntax: `AdvancedConfig = {"JobInformationDisabled=0|1", ...}`

or **Syntax:** `GlobalConfig = {"JobInformationDisabled=0|1", ...}`

Default: `JobInformationDisabled=0`

The `JobInformationDisabled` directive is a parameter of the `AdvancedConfig` (page 720) directive, as well as the `GlobalConfig` (page 721) directive. If set to 1 it disables job-based monitoring (section 12.8). The details of how this is done are discussed shortly.

For a cluster that is running millions of jobs at a time, job-based monitoring can typically consume significant resources. The monitoring of data for so many small jobs is typically not useful and not of interest. For such a case, setting this directive improves cluster performance by not having to deal with the information on the current jobs running on the cluster.

The `GlobalConfig` value is the directive that the cluster administrator should set in most cases. It only needs to be set on `cmd.conf` on the head node (or head nodes in a high availability configuration).

Setting it disables the collection of job information by the nodes running the jobs, and no collection of the job information from those nodes is done by the head (monitoring) node.

The `GlobalConfig` value is overridden by the `AdvancedConfig` value. The `AdvancedConfig` value needs to be set on `cmd.conf` on the head node(s), but also needs to be set on the non-head nodes that are to be disabled. Setting the `AdvancedConfig` directive still allows job information to be collected by the nodes running the jobs, but the information is not collected from the nodes by the head (monitoring) node. This means that nodes are running extra processes for probably no reason. Running the directive as an `AdvancedConfig` is therefore suboptimal for most use cases.

JobInformationKeepDuration directive

Syntax: `AdvancedConfig = {"JobInformationKeepDuration=<number>", ...}`

Default: `JobInformationKeepDuration=2419200`

The `JobInformationKeepDuration` directive is a parameter of the `AdvancedConfig` (page 720) directive. It takes on a value in seconds. If a job has finished more than that many seconds ago, then it will be removed along with all its monitoring data from the database. By default it is set to 28 days ($24 \times 3600 \times 28$ seconds).

JobInformationKeepCount directive

Syntax: `AdvancedConfig = {"JobInformationKeepCount=<number>", ...}`

Default: `JobInformationKeepCount=8192`

The `JobInformationKeepCount` directive is a parameter of the `AdvancedConfig` (page 720) directive. If the total number of jobs is greater than (`JobInformationKeepCount + 10%`), then the job record and its data content are discarded, starting from the oldest job first, until the total number of jobs remaining becomes `JobInformationKeepCount`. If it is set to 0, then none of the job records and content are removed.

The maximum value for this directive is 100 million.

JobInformationMinimalJobDuration directive

Syntax: `AdvancedConfig = {"JobInformationMinimalJobDuration=<number>", ...}`

Default: `JobInformationMinimalJobDuration=0`

The `JobInformationMinimalJobDuration` directive is a parameter of the `AdvancedConfig` (page 720) directive. If set, then jobs that run for less than this number of seconds are not stored in the cache. Its default value of 0 seconds means that all jobs are handled.

JobInformationFlushInterval directive

Syntax: `AdvancedConfig = {"JobInformationFlushInterval=<number>", ...}`

Default: `JobInformationFlushInterval=600`

The `JobInformationFlushInterval` directive is a parameter of the `AdvancedConfig` (page 720) directive. If this interval, in seconds, is set, then the cache is flushed to the database with that interval. Its default value is 10 minutes (10×60 seconds). Values of around 30 seconds or less will conflict with the default CMDaemon maintenance timeout value of 30 seconds, and will mostly simply add load.

ActionLoggerFile directive

Syntax: `AdvancedConfig = {"ActionLoggerFile=filename", ...}`

Default: /var/spool/cmd/actions.log

The directive is a parameter of the `AdvancedConfig` (page 720) directive. Its value overrides the default path.

The directive needs to be implemented per node or image.

ActionLoggerOnSuccess directive

Syntax: `AdvancedConfig = {"ActionLoggerOnSuccess=0|1", ...}`

Default: `ActionLoggerOnSuccess=0`

The directive is a parameter of the `AdvancedConfig` (page 720) directive.

By default, only failed actions are logged. Successful actions are also logged if setting `ActionLoggerOnSuccess=1`

The directive needs to be implemented per node or image.

The log file shows timestamped output with one line per run for an action script, with the script response.

Example

```
(time) /path/to/action/script [ timeout ]
(time) /path/to/action/script [ failed ] (exit code: 1)
(time) /path/to/action/script [ success ]
```

The success line only appears if `ActionLoggerOnSuccess=1`.

FailoverPowerRetries directive

Syntax: `AdvancedConfig = {"FailoverPowerRetries=<number>", ...}`

Default: `FailoverPowerRetries=5`

The `FailoverPowerRetries` directive is a parameter of the `AdvancedConfig` (page 720) directive.

After a decision to carry out the failover has been made, `CMDaemon` sends a power off command to the BMC of the head node that is meant to be powered off. If the power off command fails, then on getting the fail response, `CMDaemon` waits for 1 second. After that second, it sends out a power off command again. The value of `FailoverPowerRetries` is the number of times that `CMDaemon` retries sending the power off command to the BMC of the head node that is intended to be powered off during the failover, if the response from the power off command remains a fail response.

The power down attempts cease, either when the BMC reports that the head node is OFF, or when the number of attempts reaches the `FailoverPowerRetries` value.

`FailoverPowerRetries` takes a maximum value of 120.

A value of 0 means that 1 attempt to power off is carried out during failover, but no retry is attempted after the first attempt.

Because `CMDaemon` waits for a period of 1s before checking for an OFF response, it means that the number of retries is about the same as the number of seconds before `CMDaemon` decides that powering off has failed, unless the OFF response also takes some time to get to `CMDaemon`.

Increasing the value for this directive can be useful for some BMC cards that take longer than about 5s to report their power status, because the power off attempt may otherwise time out.

AddUserDefaultGroupID directive

Syntax: `AdvancedConfig = { "AddUserDefaultGroupID = <number>", ...}`

Default: *none*

The `AddUserDefaultGroupID` directive is a parameter of the `AdvancedConfig` (page 720) directive.

If the `AddUserDefaultGroupID` is unset, then the default group ID of a new user is the same as the User ID.

If the `AddUserDefaultGroupID` is set, then the set value becomes the default GUID of new users, when new users are created via the `cmsh` or Bright View front ends to CMDaemon.

The directive is not intended to set a non-default group occasionally during user creation. In that case, a non-default group ID can be set from the `cmsh` or Bright View front ends, by setting the `groupid` value for the user.

To set a default group for a directory, while retaining the default group for the user, it may be possible to use the `setgid` bit for a directory.

D

Disk Partitioning And RAID Configuration

Disk partitioning is initially configured on the head node and regular nodes during installation (section 3.3.16 of the *Installation Manual*).

For the head node it cannot be changed from within the 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`, `xfs`, and so on, is used. The changes are implemented during the node partitioning stage of the node-installer (section 5.4.6).

Diskless operation can also be implemented by using an appropriate XML file. This is introduced in section 3.9.1.

Software or hardware RAID configuration can also be set for the regular nodes. The layouts must be specified following the XML schema files stored on the head node in the directory `/cm/node-installer/scripts/`:

- Software RAID configuration is set in the global partitioning XML schema file `disks.xsd` (section D.1).
- Hardware RAID configuration is set in the separate hardware RAID XML schema file `raid.xsd` (section D.2).

D.1 Structure Of Partitioning Definition—The Global Partitioning XML Schema File

In 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/local/apps/cmd/etc/htdocs/xsd/disks.xsd`.

This schema does not include a hardware RAID description. The hardware RAID schema is defined separately in the file `raid.xsd` (section D.2).

Examples of schemas in use, with and without hardware RAID, are given in sections D.3 and beyond.

An XML file can be validated against a schema using the `xmllint` tool:

Example

```
[root@bright90 ~]# xmllint --noout --schema /cm/local/apps/cmd/etc/htdocs/xsd/disks.xsd \  
/cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml \  
/cm/local/apps/cmd/etc/htdocs/disk-setup/slave-diskless.xml validates  
[root@bright90 ~]#
```

XML schema for partitioning

```
<?xml version='1.0'?>

<!--
#
# Copyright (c) 2004-2017 Bright Computing, Inc. All Rights Reserved.
#
# This software is the confidential and proprietary information of
# Bright Computing, Inc.("Confidential Information"). You shall not
# disclose such Confidential Information and shall use it only in
# accordance with the terms of the license agreement you entered into
# with Bright Computing, Inc.

This is the XML schema description of the partition layout XML file.
It can be used by software to validate partitioning XML files.
There are however a few things the schema does not check:
- There should be exactly one root mountpoint (/), unless diskless.
- There can only be one partition with a 'max' size on a particular device.
- Something similar applies to logical volumes.
- The 'auto' size can only be used for a swap partition.
- Partitions of type 'linux swap' should not have a filesystem.
- Partitions of type 'linux raid' should not have a filesystem.
- Partitions of type 'linux lvm' should not have a filesystem.
- Partitions of type 'unspecified' should not have a filesystem.
- If a raid is a member of another raid then it can not have a filesystem.
- Partitions, which are listed as raid members, should be of type 'linux raid'.
- If diskless is not set, there should be at least one device.
- The priority tag is only valid for partitions which have type set to
  "linux swap".
-->

<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema' elementFormDefault='qualified'>

  <xs:element name='diskSetup'>

    <xs:complexType>
      <xs:sequence>
        <xs:element name='diskless' type='diskless' minOccurs='0' maxOccurs='1'/>
        <xs:element name='device' type='device' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='raid' type='raid' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='volumeGroup' type='volumeGroup' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='subVolumes' type='subVolumes' minOccurs='0' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>

    <xs:key name='partitionAndRaidIds'>
      <xs:selector xpath='./raid|./partition'/>
      <xs:field xpath='@id'/>
    </xs:key>

    <xs:keyref name='raidMemberIds' refer='partitionAndRaidIds'>
      <xs:selector xpath='./raid/member'/>
      <xs:field xpath='.'/>
    </xs:keyref>
  </xs:element>
</xs:schema>
```

```

<xs:keyref name='volumeGroupPhysicalVolumes' refer='partitionAndRaidIds'>
  <xs:selector xpath='./volumeGroup/physicalVolumes/member' />
  <xs:field xpath='.' />
</xs:keyref>

<xs:keyref name='subVolumeIds' refer='partitionAndRaidIds'>
  <xs:selector xpath='./subVolumes' />
  <xs:field xpath='@parent' />
</xs:keyref>

<xs:unique name='raidAndVolumeMembersUnique'>
  <xs:selector xpath='./member' />
  <xs:field xpath='.' />
</xs:unique>

<xs:unique name='deviceNodesUnique'>
  <xs:selector xpath='./device/blockdev' />
  <xs:field xpath='.' />
  <xs:field xpath='@mode' />
</xs:unique>

<xs:unique name='mountPointsUnique'>
  <xs:selector xpath='./mountPoint' />
  <xs:field xpath='.' />
</xs:unique>

<xs:unique name='assertNamesUnique'>
  <xs:selector xpath='./assert' />
  <xs:field xpath='@name' />
</xs:unique>

</xs:element>

<xs:complexType name='diskless'>
  <xs:attribute name='maxMemSize' type='memSize' use='required' />
  <xs:attribute name='mountOptions' type='xs:string' />
</xs:complexType>

<xs:simpleType name='memSize'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='([0-9]+[MG])|100%|[0-9][0-9%]|[0-9%]|0' />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='stripeSize'>
  <xs:restriction base='xs:string'>
    <xs:pattern value='4|8|16|32|64|128|256|512|1024|1K|2048|2K|4096|4K' />
    <xs:pattern value='8192|8K|16384|16K|32768|32K|65536|64K|131072|128K' />
    <xs:pattern value='262144|256K|524288|512K|1048576|1024K|1M' />
    <xs:pattern value='2097152|2048K|2M|4194304|4096K|4M' />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name='size'>

```

```

    <xs:restriction base='xs:string'>
      <xs:pattern value='max|auto|[0-9]+[MGT]' />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='relativeOrAbsoluteSize'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='max|auto|[0-9]+[MGT] | [0-9]+([.][0-9]+)?%|[0-9]+/[0-9]+' />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='extentSize'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='([0-9])+M' />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='blockdevName'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='/dev/.+' />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name='blockdev'>
    <xs:simpleContent>
      <xs:extension base="blockdevName">
        <xs:attribute name="mode" default='normal'>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="normal|cloud|both" />
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name='device'>
    <xs:sequence>
      <xs:element name='blockdev' type='blockdev' minOccurs='1' maxOccurs='unbounded' />
      <xs:element name='vendor' type='xs:string' minOccurs='0' maxOccurs='1' />
      <xs:element name='requiredSize' type='size' minOccurs='0' maxOccurs='1' />
      <xs:element name='assert' minOccurs='0' maxOccurs='unbounded'>
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base='xs:string'>
              <xs:attribute name='name' use='required'>
                <xs:simpleType>
                  <xs:restriction base='xs:string'>
                    <xs:pattern value='[a-zA-Z0-9-_.]+' />
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name='args' type='xs:string' />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

```



```

        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name='alignMiB' type='xs:boolean' minOccurs='0' maxOccurs='1' />
<xs:element name="partitionTable" minOccurs='0' maxOccurs='1'>
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="gpt|msdos"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name='partition' type='partition' minOccurs='1' maxOccurs='unbounded' />
</xs:sequence>
<xs:attribute name='origin' type='xs:string' />
</xs:complexType>

<xs:complexType name='partition'>
    <xs:sequence>
        <xs:element name='cephosdassociation' type='xs:string' minOccurs='0' maxOccurs='1' />
        <xs:element name='size' type='relativeOrAbsoluteSize' />
        <xs:element name='type'>
            <xs:simpleType>
                <xs:restriction base='xs:string'>
                    <xs:enumeration value='linux' />
                    <xs:enumeration value='linux swap' />
                    <xs:enumeration value='linux raid' />
                    <xs:enumeration value='linux lvm' />
                    <xs:enumeration value='unspecified' />
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs: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:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:group>

```

```

        <xs:enumeration value='zfs' />
        <xs:enumeration value='fat32' />
        <xs:enumeration value='fat' />
    </xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name='mkfsFlags' type='xs:string' minOccurs='0' maxOccurs='1' />
<xs:element name='mountPoint' type='xs:string' minOccurs='0' maxOccurs='1' />
<xs:element name='mountOptions' type='xs:string' default='defaults' minOccurs='0' />
</xs:sequence>
</xs:group>

<xs:complexType name='raid'>
    <xs:sequence>
        <xs:element name='member' type='xs:string' minOccurs='2' maxOccurs='unbounded' />
        <xs:element name='level' type='xs:int' />
        <xs:choice minOccurs='0' maxOccurs='1'>
            <xs:group ref='filesystem' />
            <xs:sequence>
                <xs:element name='swap'><xs:complexType /></xs:element>
                <xs:element name='priority' minOccurs='0' maxOccurs='1'>
                    <xs:simpleType>
                        <xs:restriction base="xs:integer">
                            <xs:minInclusive value="0" />
                            <xs:maxInclusive value="32767" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name='id' type='xs:string' use='required' />
</xs:complexType>

<xs:complexType name='volumeGroup'>
    <xs:sequence>
        <xs:element name='name' type='xs:string' />
        <xs:element name='extentSize' type='extentSize' />
        <xs:element name='physicalVolumes'>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name='member' type='xs:string' minOccurs='1' maxOccurs='unbounded' />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name='logicalVolumes'>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name='volume' type='logicalVolume' minOccurs='1' maxOccurs='unbounded' />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>

```

```
<xs:complexType name='logicalVolume'>
  <xs:sequence>
    <xs:element name='name' type='xs:string' />
    <xs:element name='size' type='size' />
    <xs:element name='pool' type='xs:string' minOccurs='0' maxOccurs='1' />
    <xs:element name='stripes' minOccurs='0' maxOccurs='1'>
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="1"/>
          <xs:maxInclusive value="32768"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name='stripeSize' type='stripeSize' minOccurs='0' maxOccurs='1' />
    <xs:group ref='filesystem' minOccurs='0' maxOccurs='1' />
  </xs:sequence>
  <xs:attribute name="thinpool">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="1"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="metadatasize">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="([1-9] [MGT])|([1-9] [0-9]+ [MGT])"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

<xs:complexType name='subVolumes'>
  <xs:sequence>
    <xs:element name='subVolume' type='subVolume' minOccurs='1' maxOccurs='unbounded' />
  </xs:sequence>
  <xs:attribute name='parent' type='xs:string' use='required' />
</xs:complexType>

<xs:complexType name='subVolume'>
  <xs:sequence>
    <xs:element name='mountPoint' type='xs:string' />
    <xs:element name='mountOptions' type='xs:string' />
  </xs:sequence>
</xs:complexType>

</xs:schema>
```

Examples Of Element Types In XML Schema

Name Of Type	Example Values
size	10G, 128M, 1T, 2.5T, 1/3, 33.333%, auto, max
device	/dev/sda, /dev/hda, /dev/cciss/c0d0
partition	linux, linux raid, linux swap, unspecified
filesystem	ext2, ext3, ext4, xfs

D.2 Structure Of Hardware RAID Definition—The Hardware RAID XML Schema File

If a hardware RAID has already been created outside of the cluster manager, then no XML definition is needed. This assumes that required kernel modules for the device load, so that the operating system ends up treating the RAID as a standard block device, which can therefore have its layout configured as described in section 3.9.

If, instead, hardware RAID is to be created and managed by the cluster manager, then it can be specified using an XML schema, stored on the head node in `/cm/local/apps/cmd/etc/htdocs/xsd/raid.xsd`. The full schema specification is listed next, while schema examples are listed in section D.4.1.

Configuration using the cluster manager is currently limited to MegaRAID hardware. It may not work for newer controllers that do not support MegaCLI, or its successor StorCLI. It may also not work for controllers by vendors that do not use MegaCLI or StorCLI.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!--
#
# 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 hardware RAID layout XML file.
It can be used by software to validate partitioning XML files.
There are however a few things the schema does not check:
- All of the spans (drive groups) in an raidArray must have the same
  number of drives.
- There can only be one volume with a 'max' size on a particular array
  and this must be the last volume in the array.
- If there is only one enclosure defined for a particular RAID
  controller the actual enclosureID can be omitted by using "auto"
  instead. Otherwise, the actual enclosureID must be specified.
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="raidLevel">
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:pattern value="0|1|5|10|50"/>
    </xs:restriction>
  </xs:simpleType>
```

```
<xs:simpleType name="volumeSize">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]1,5[MGT]|max"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="stripeSize">
  <xs:restriction base="xs:string">
    <xs:pattern value="8K|16K|32K|64K|128K|256K|512K|1024K"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="cachePolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="Cached|Direct"/>
  </xs:restriction>
</xs:simpleType>

<!--
  NORA : No Read Ahead
  RA    : Read Ahead
  ADRA  : Adaptive Read
-->
<xs:simpleType name="readPolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="NORA|RA|ADRA"/>
  </xs:restriction>
</xs:simpleType>

<!--
  WT : Write Through
  WB : Write Back
-->
<xs:simpleType name="writePolicy">
  <xs:restriction base="xs:string">
    <xs:pattern value="WT|WB"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="enclosureID">
  <xs:restriction base="xs:string">
    <xs:pattern value="auto|[0-9]{1,4}"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="slotNumber">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:pattern value="[0-9]{1,2}"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="raidSetup">
  <xs:complexType>
    <xs:sequence>
```

```

<xs:element name="raidArray" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>

      <xs:element name="level" type="raidLevel"/>

      <xs:element name="raidVolume" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>

            <xs:element name="stripeSize" type="stripeSize"/>

            <xs:element name="cachePolicy" type="cachePolicy"/>

            <xs:element name="readPolicy" type="readPolicy"/>

            <xs:element name="writePolicy" type="writePolicy"/>

            <xs:element name="size" type="volumeSize"/>

          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:choice>

        <xs:element name="device" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>

              <xs:element name="enclosureID" type="enclosureID"/>

              <xs:element name="slotNumber" type="slotNumber"/>

            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="span" minOccurs="2" maxOccurs="8">
          <xs:complexType>
            <xs:sequence>

              <xs:element name="device" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>

                    <xs:element name="enclosureID" type="enclosureID"/>

                    <xs:element name="slotNumber" type="slotNumber"/>

                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xs:choice>

  </xs:sequence>
</xs:complexType>
</xs:element>

</xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>

```

D.3 Example: Default Node Partitioning

The following example follows the schema specification of section D.1, and shows the default layout used for regular nodes:

Example

```

<?xml version="1.0" encoding="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>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>2G</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/tmp</mountPoint>

```

```

    <mountOptions>defaults,noatime,nodiratime,nosuid,nodev</mountOptions>
  </partition>
  <partition id="a4">
    <size>12G</size>
    <type>linux swap</type>
  </partition>
  <partition id="a5">
    <size>max</size>
    <type>linux</type>
    <filesystem>ext3</filesystem>
    <mountPoint>/local</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </partition>
</device>
</diskSetup>

```

The example assumes a single disk. Another disk can be added by adding another pair of `<device><device>` tags and filling in the partitioning specifications for the next disk. Because multiple `blockdev` tags are used, the node-installer first tries to use `/dev/sda`, then `/dev/hda`, then `/dev/vda` (virtio disks), then `/dev/xvda` (xen disks), and so on. Cloud devices can also be accessed using the `mode="cloud"` option. Removing block devices from the layout if they are not going to be used does no harm.

For each partition, a size tag is specified. Sizes can be specified using megabytes (e.g. 500M), gigabytes (e.g. 50G) or terabytes (e.g. 2T or 4.5T). Relative sizes, without units, can be used in the form of fractions (e.g. 2/3) or percentages (e.g. 70%), which can be useful for disk sizes that are not known in advance.

Small differences in size do not trigger a full install for existing relative partitions.

For swap partitions, a size of `auto` sets a swap partition to twice the node memory size. If there is more than one swap partition, then the priority tag can be set so that the partition with the higher priority is used first.

For a device, the attribute `max` for a size tag forces the last device in the partition to use all remaining space, and if needed, adjusts the implementation of the sequence of size tags in the remaining partitions accordingly. The use of `max` for a partition is convenient.

In the example, all filesystems are specified as `ext3`. Valid alternatives are `ext2`, `ext4` and `xf`s.

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@bright90 ~]# cms
[bright90]% category use default
[bright90->category[default]]% set hardwareraidconfiguration

```

This opens up an editor in which the XML file can be specified according to the schema in section D.2. XML validation is carried out.

D.4.1 RAID level 0 And RAID 10 Example

In the following configuration the node has two RAID arrays, one in a RAID 0 and the other in a RAID 10 configuration:

- The RAID 0 array contains three volumes and is made up of two hard disks, placed in slots 0 and 1. The volumes have different values for the options and policies.
- The RAID 10 array consists of just one volume and has two spans, in slots 2 and 3. Each span has two hard disks.

Example

```

<raidSetup>
  <raidArray>
    <level>0</level>

    <raidVolume>
      <stripeSize>64K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>NORA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>40G</size>
    </raidVolume>

    <raidVolume>
      <stripeSize>128K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>RA</readPolicy>
      <writePolicy>WB</writePolicy>
      <size>80G</size>
    </raidVolume>

    <raidVolume>
      <stripeSize>256K</stripeSize>
      <cachePolicy>Cached</cachePolicy>
      <readPolicy>ADRA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>100G</size>
    </raidVolume>

    <device>
      <enclosureID>auto</enclosureID>
      <slotNumber>0</slotNumber>
    </device>

    <device>
      <enclosureID>32</enclosureID>
      <slotNumber>1</slotNumber>
    </device>
  </raidArray>

  <raidArray>
    <level>10</level>

    <raidVolume>
      <stripeSize>64K</stripeSize>
      <cachePolicy>Direct</cachePolicy>
      <readPolicy>NORA</readPolicy>
      <writePolicy>WT</writePolicy>
      <size>40G</size>
  </raidArray>

```

```

</raidVolume>

<span>
  <device>
    <enclosureID>auto</enclosureID>
    <slotNumber>2</slotNumber>
  </device>

  <device>
    <enclosureID>auto</enclosureID>
    <slotNumber>3</slotNumber>
  </device>
</span>

<span>
  <device>
    <enclosureID>auto</enclosureID>
    <slotNumber>4</slotNumber>
  </device>

  <device>
    <enclosureID>auto</enclosureID>
    <slotNumber>5</slotNumber>
  </device>
</span>
</raidArray>
</raidSetup>

```

D.5 Example: Software RAID

The following example shows a simple software RAID setup:

Example

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema.xsd">

  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>25G</size>
      <type>linux raid</type>
    </partition>
  </device>

  <device>
    <blockdev>/dev/sdb</blockdev>
    <partition id="b1">
      <size>25G</size>
      <type>linux raid</type>
    </partition>
  </device>

  <raid id="r1">

```

```

    <member>a1</member>
    <member>b1</member>
    <level>1</level>
    <filesystem>ext3</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </raid>

```

```
</diskSetup>
```

The `level` tag specifies the RAID level used. The following are supported:

- 0 (striping without parity)
- 1 (mirroring)
- 4 (striping with dedicated parity drive)
- 5 (striping with distributed parity)
- 6 (striping with distributed double parity)

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></swap>` tag is used to indicate a swap partition in a RAID device specified in the XML schema of section D.1. For example, the following marks a 1GB RAID 1 partition as being used for swap, and the second partition for an ext3 filesystem:

```

<?xml version="1.0" encoding="UTF-8"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="a2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
  <device>
    <blockdev>/dev/sdb</blockdev>
    <partition id="b1">
      <size>1G</size>
      <type>linux raid</type>
    </partition>
    <partition id="b2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>
</diskSetup>

```

```

    </partition>
</device>
<raid id="r1">
  <member>a1</member>
  <member>b1</member>
  <level>1</level>
  <swap></swap>
</raid>
<raid id="r2">
  <member>a2</member>
  <member>b2</member>
  <level>1</level>
  <filesystem>ext3</filesystem>
  <mountPoint></mountPoint>
  <mountOptions>defaults,noatime,nodiratime</mountOptions>
</raid>
</diskSetup>

```

As in section D.5, the appropriate RAID modules must be loaded beforehand.

D.7 Example: Logical Volume Manager

This example shows a simple LVM setup:

Example

```

<?xml version="1.0" encoding="UTF-8"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/hda</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev mode="cloud">/dev/sdb</blockdev>
    <blockdev mode="cloud">/dev/hdb</blockdev>
    <blockdev mode="cloud">/dev/vdb</blockdev>
    <blockdev mode="cloud">/dev/xvdb</blockdev>
    <blockdev mode="cloud">/dev/xvdf</blockdev>
    <partition id="a1">
      <size>512M</size>
      <type>linux</type>
      <filesystem>ext2</filesystem>
      <mountPoint>/boot</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
    <partition id="a2">
      <size>16G</size>
      <type>linux swap</type>
    </partition>
    <partition id="a3">
      <size>max</size>
      <type>linux lvm</type>
    </partition>
  </device>
  <volumeGroup>

```

```

<name>vg01</name>
<extentSize>8M</extentSize>
<physicalVolumes>
  <member>a3</member>
</physicalVolumes>
<logicalVolumes>
  <volume>
    <name>lv00</name>
    <size>max</size>
    <filesystem>ext3</filesystem>
    <mountPoint>/</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </volume>
  <volume>
    <name>lv01</name>
    <size>8G</size>
    <filesystem>ext3</filesystem>
    <mountPoint>/tmp</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </volume>
</logicalVolumes>
</volumeGroup>
</diskSetup>

```

The member tags must refer to an id attribute of a partition tag, or an id attribute of a raid tag. The administrator must ensure that the dm-mod kernel module is loaded when LVM is used.

D.8 Example: Logical Volume Manager With RAID 1

This example shows an LVM setup, but with the LVM partitions mirrored using RAID 1:

Example

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:no\
NamespaceSchemaLocation="schema.xsd">

<device>
  <blockdev>/dev/sda</blockdev>
  <partition id="a1">
    <size>512M</size>
    <type>linux raid</type>
  </partition>
  <partition id="a2">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/sdb</blockdev>
  <partition id="b1">
    <size>512M</size>
    <type>linux raid</type>
  </partition>
  <partition id="b2">

```

```
        <size>max</size>
        <type>linux raid</type>
    </partition>
</device>

<raid id="r1">
    <member>a1</member>
    <member>b1</member>
    <level>1</level>
    <filesystem>ext3</filesystem>
    <mountPoint>/boot</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
</raid>

<raid id="r2">
    <member>a2</member>
    <member>b2</member>
    <level>1</level>
</raid>

<volumeGroup>
    <name>vg01</name>
    <extentSize>8M</extentSize>
    <physicalVolumes>
        <member>r2</member>
    </physicalVolumes>

    <logicalVolumes>
        <volume>
            <name>lv00</name>
            <size>50G</size>
            <filesystem>ext3</filesystem>
            <mountPoint>/</mountPoint>
            <mountOptions>defaults,noatime,nodiratime</mountOptions>
        </volume>
        <volume>
            <name>lv01</name>
            <size>25G</size>
            <filesystem>ext3</filesystem>
            <mountPoint>/tmp</mountPoint>
            <mountOptions>defaults,noatime,nodiratime</mountOptions>
        </volume>
        <volume>
            <name>lv02</name>
            <size>25G</size>
            <filesystem>ext3</filesystem>
            <mountPoint>/var</mountPoint>
            <mountOptions>defaults,noatime,nodiratime</mountOptions>
        </volume>
    </logicalVolumes>
</volumeGroup>

</diskSetup>
```

D.9 Example: Diskless

This example shows a node configured for diskless operation:

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <diskless maxMemSize="90%"></diskless>
</diskSetup>
```

An example of the implementation of a diskless configuration is given in section 3.9.3.

In diskless mode the software image is transferred by the node-installer to a RAM-based filesystem on the node called `tmpfs`.

The obvious advantage of running from RAM is the elimination of the physical disk, cutting power consumption and reducing the chance of hardware failure. On the other hand, some of the RAM on the node is then no longer available for user applications.

Special considerations with diskless mode:

- **Recommended minimum RAM size:** The available RAM per node should be sufficient to run the OS and the required tasks. At least 4GB is recommended for diskless nodes.
- **The `tmpfs` size limit:** The maximum amount of RAM that can be used for a filesystem is set with the `maxMemSize` attribute. A value of 100% allows all of the RAM to be used. The default value is 90%. A value of 0, without the % sign, removes all restrictions.
A limit does not however necessarily prevent the node from crashing, as some processes might not deal properly with a situation when there is no more space left on the filesystem.
- **Persistence issues:** While running as a diskless node, the node is unable to retain any non-shared data each time it reboots. For example the files in `/var/log/*`, which are normally preserved by the `exclude` list settings for disked nodes, are lost from RAM during diskless mode reboots. The `installmode NOSYNC` setting cannot be used with diskless nodes during a node reboot.
- **Leftover disk issues:** Administrators in charge of sensitive environments should be aware that the disk of a node that is now running in diskless mode still contains files from the last time the disk was used, unless the files are explicitly wiped.
- **Reducing the software image size in `tmpfs` on a diskless node:** To make more RAM available for tasks, the software image size held in RAM can be reduced:
 - by removing unnecessary software from the image.
 - by mounting parts of the filesystem in the image over NFS during normal use. This is especially worthwhile for less frequently accessed parts of the image (section 3.10.3).

D.10 Example: Semi-diskless

Diskless operation (section D.9) can also be mixed with certain parts of the filesystem on the local physical disk. This frees up RAM which the node can then put to other use. In this example all data in `/local` is on the physical disk, the rest in RAM.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">
  <diskless maxMemSize="0"></diskless>
  <device>
    <blockdev>/dev/sda</blockdev>
    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/local</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
</diskSetup>
```

When nodes operate in semi-diskless mode the node-installer always uses `excludelistfullinstall` (section 5.4.7) when synchronizing the software image to memory and disk.

An alternative to using a local disk for freeing up RAM is to use NFS storage, as is described in section 3.10.3.

D.11 Example: Preventing Accidental Data Loss

Optional tags, `vendor` and `requiredSize`, can be used to prevent accidentally repartitioning the wrong drive. Such a tag use is shown in the following example.

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="schema.xsd">
  <device>
    <blockdev>/dev/sda</blockdev>
    <vendor>Hitachi</vendor>
    <requiredSize>200G</requiredSize>

    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

  </device>
  <device>
    <blockdev>/dev/sdb</blockdev>
    <vendor>BigRaid</vendor>
    <requiredSize>2T</requiredSize>

    <partition id="b1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
```



```

    <mountPoint>/data</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </partition>

</device>
</diskSetup>

```

If a vendor or a `requiredSize` element is specified, it is treated as an assertion which is checked by the node-installer. The node-installer reads the drive vendor string from `/sys/block/<drive name>/device/vendor`. For the assertion to succeed, the ratio of actual disk size to the value specified by `requiredSize`, should be at least 0.85:1, and at most 1:0.85.

That is: to be able to get past the `requiredSize` assertion, the actual drive size as seen from `fdisk -l` should be 85% to about 118% of the asserted size.

If any assertion fails, no partitioning changes will be made to any of the specified devices.

For assertions with drives that are similar or identical in size, and are from the same vendor, the `requiredSize` and `vendor` elements are not enough to differentiate between the drives. In such cases, custom assertions (section D.12) can be set for particular drives.

Specifying device assertions is recommended for nodes that contain important data because it protects against a situation where a drive is assigned to an incorrect block device. This can happen, for example, when the first drive, for example `/dev/sda`, in a multi-drive system is not detected (e.g. due to a hardware failure, or a BIOS update) which could cause the second drive to become known as `/dev/sda`, potentially causing much woe.

As an aside, `CMDaemon` does offer another way, outside of assertions, to avoid wiping out drive data automatically. This is done in `cmsh` by setting the value of `datanode` to `yes` (section 5.4.4).

D.12 Example: Using Custom Assertions

The following example shows the use of the `assert` tag, which can be added to a device definition:

Example

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema.xsd">
  <device>
    <blockdev>/dev/sda</blockdev>
    <assert name="modelCheck" args="WD800AAJS">
      <![CDATA[
        #!/bin/bash
        if grep -q $1 /sys/block/$ASSERT_DEV/device/model; then
          exit 0
        else
          exit 1
        fi
      ]]>
    </assert>

    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>ext3</filesystem>
      <mountPoint></mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
</diskSetup>

```

```
</device>
<device>
  <blockdev>/dev/sdb</blockdev>
  <vendor>BigRaid</vendor>
  <requiredSize>2T</requiredSize>

  <partition id="b1">
    <size>max</size>
    <type>linux</type>
    <filesystem>ext3</filesystem>
    <mountPoint>/data</mountPoint>
    <mountOptions>defaults,noatime,nodiratime</mountOptions>
  </partition>

</device>
</diskSetup>
```

The `assert` tag is similar to the `vendor` and `size` tags described in section D.11.

It can be used to define custom assertions. The assertions can be implemented using any script language.

The script can access the environment variables `ASSERT_DEV` (eg: `sda`) and `ASSERT_NODE` (eg: `/dev/sda`) during the node-installer stage.

Each `assert` needs to be assigned an arbitrary name and can be passed custom parameters. A non-zero exit code in the assertion causes the node-installer to halt.

E

Example initialize And finalize Scripts

The node-installer executes any `initialize` and `finalize` scripts at particular stages of its 13-step run during node-provisioning (section 5.4). They are sometimes useful for troubleshooting or workarounds during those stages. The scripts are stored in the CMDaemon database, rather than in the filesystem as plain text files, because they run before the node's `init` process takes over and establishes the final filesystem.

Default `initialize` and `finalize` scripts are provided with the default category:

```
[bright90->category[default]]% show | grep ize
Initialize script      <1.46KiB>
Finalize script       <3.4KiB>
```

E.1 When Are They Used?

The `initialize` and `finalize` scripts are sometimes used as an alternative configuration option out of a choice of other possible options (section 3.15.1). 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 Bright View And `cmsh`

The `initialize` and `finalize` scripts are accessible for viewing and editing:

- In Bright View, via the `Node Categories` or `Nodes` window, under the `Settings` window. The clickpaths for these are:

```
- Grouping->Node categories[default]->Edit->Settings
```

– Devices→Nodes[node001]→Edit→Settings

- In `cmsh`, using the category or device modes. The `get` command is used for viewing the script, and the `set` command to start up the default text editor to edit the script. Output is truncated in the two following examples at the point where the editor starts up:

Example

```
[root@bright90 ~]# cmsh
[bright90]% category use default
[bright90->category[default]]% show | grep script
Parameter                               Value
-----
Finalize script                          <1367 bytes>
Initialize script                         <0 bytes>
[bright90->category[default]]% set initializescript
```

Example

```
[bright90]% device use node001
[bright90->device[node001]]%
[bright90->device[node001]]% set finalizescript
```

E.3 Environment Variables Available To initialize And finalize Scripts

For the initialize and finalize scripts, node-specific customizations can be made from a script using environment variables.

The environment variables can be listed using the following script as a `finalizescript`:

Example

```
[bright90->device[node001]]% get finalizescript
#!/bin/bash
#
# All cluster manager environment variables are prefixed with CMD_
# The root / of the running node is always mounted on /localdisk
#
set | grep CMD_ > /localdisk/var/log/node-installer-finalize.env
```

After the node comes up, the contents of the saved file on that node are the available variables:

Example

```
[root@node001 ~]# cat /var/log/node-installer-finalize.env
CMD_ACTIVE_MASTER_IP=10.141.255.254
CMD_CATEGORY=default
CMD_CHASSIS=
CMD_CHASSIS_IP=0.0.0.0
CMD_CHASSIS_PASSWORD=
CMD_CHASSIS_SLOT=
CMD_CHASSIS_USERNAME=
...
```

The following table shows the available variables with some example values:

Table E: Environment Variables For The initialize And Finalize Scripts

Variable	Example Value
CMD_ACTIVE_MASTER_IP	10.141.255.254
CMD_CATEGORY	default
CMD_CHASSIS	chassis01
CMD_CHASSIS_IP	10.141.1.1
CMD_CHASSIS_PASSWORD	ADMIN
CMD_CHASSIS_SLOT	1
CMD_CHASSIS_USERNAME	ADMIN
CMD_CLUSTERNAME	Bright 9.0 Cluster
CMD_DEVICE_HEIGHT	1
CMD_DEVICE_POSITION	10
CMD_DEVICE_TYPE	SlaveNode
CMD_ETHERNETSWITCH	switch01:1
CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_ALLOWWRITE	no
CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_HOSTS	10.141.0.0/16
CMD_FSEXPORT__SLASH_cm_SLASH_node-installer_PATH	/cm/node-installer
CMD_FSEXPORTS	_SLASH_cm_SLASH_node-installer
CMD_FSMOUNT__SLASH_cm_SLASH_shared_DEVICE	master:/cm/shared
CMD_FSMOUNT__SLASH_cm_SLASH_shared_FILESYSTEM	nfs
CMD_FSMOUNT__SLASH_cm_SLASH_shared_MOUNTPOINT	/cm/shared
CMD_FSMOUNT__SLASH_cm_SLASH_shared_OPTIONS	rsize=32768,wsiz=32768,\ hard,intr,async
CMD_FSMOUNT__SLASH_dev_SLASH_pts_DEVICE	none
CMD_FSMOUNT__SLASH_dev_SLASH_pts_FILESYSTEM	devpts
CMD_FSMOUNT__SLASH_dev_SLASH_pts_MOUNTPOINT	/dev/pts
CMD_FSMOUNT__SLASH_dev_SLASH_pts_OPTIONS	gid=5,mode=620
CMD_FSMOUNT__SLASH_dev_SLASH_shm_DEVICE	none
CMD_FSMOUNT__SLASH_dev_SLASH_shm_FILESYSTEM	tmpfs
CMD_FSMOUNT__SLASH_dev_SLASH_shm_MOUNTPOINT	/dev/shm
CMD_FSMOUNT__SLASH_dev_SLASH_shm_OPTIONS	defaults
CMD_FSMOUNT__SLASH_home_DEVICE	master:/home
CMD_FSMOUNT__SLASH_home_FILESYSTEM	nfs
CMD_FSMOUNT__SLASH_home_MOUNTPOINT	home
CMD_FSMOUNT__SLASH_home_OPTIONS	rsize=32768,wsiz=32768,\ hard,intr,async
CMD_FSMOUNT__SLASH_proc_DEVICE	none
CMD_FSMOUNT__SLASH_proc_FILESYSTEM	proc

...continues

Table E: Environment Variables For The initialize And Finalize Scripts...continued

Variable	Example Value
CMD_FSMOUNT__SLASH_proc_MOUNTPOINT	/proc
CMD_FSMOUNT__SLASH_proc_OPTIONS	defaults,nosuid
CMD_FSMOUNT__SLASH_sys_DEVICE	none
CMD_FSMOUNT__SLASH_sys_FILESYSTEM	sysfs
CMD_FSMOUNT__SLASH_sys_MOUNTPOINT	/sys
CMD_FSMOUNT__SLASH_sys_OPTIONS	defaults
CMD_FSMOUNTS *	_SLASH_dev_SLASH_pts _SLASH_proc _SLASH_sys _SLASH_dev_SLASH_shm _SLASH_cm_SLASH_shared _SLASH_home
CMD_GATEWAY	10.141.255.254
CMD_HOSTNAME	node001
CMD_INSTALLMODE	AUTO
CMD_INTERFACE_eth0_IP **	10.141.0.1
CMD_INTERFACE_eth0_MTU **	1500
CMD_INTERFACE_eth0_NETMASK **	255.255.0.0
CMD_INTERFACE_eth0_TYPE **	physical
CMD_INTERFACES *	eth0 eth1 eth2 ipmi0
CMD_IP	10.141.0.1
CMD_MAC	00:00:00:00:00:01
CMD_PARTITION	base
CMD_PASSIVE_MASTER_IP	10.141.255.253
CMD_PDUS	
CMD_POWER_CONTROL	custom
CMD_RACK	rack01
CMD_RACK_HEIGHT	42
CMD_RACK_ROOM	serverroom
CMD_ROLES	sgeclient storage
CMD_SHARED_MASTER_IP	10.141.255.252
CMD_SOFTWAREIMAGE_PATH	/cm/images/default-image
CMD_SOFTWAREIMAGE	default-image
CMD_TAG	00000000a000
CMD_USERDEFINED1	var1
CMD_USERDEFINED2	var2

* The value for this variable is a string with spaces, not an array. Eg:

CMD_FSMOUNTS="_SLASH_dev_SLASH_pts _SLASH_proc _SLASH_sys _SLASH_dev_SLASH_shm ..."

** The name of this variable varies according to the interfaces available. So,

eth0 can be replaced by eth1, eth2, ipmi0, and so on.

E.4 Using Environment Variables Stored In Multiple Variables

Some data values, such as those related to interfaces (CMD_INTERFACES_*), mount points (CMD_FSMOUNT__SLASH_*) and exports (CMD_FSEXPOR__SLASH_cm__SLASH_node-installer_*) are stored in multiple variables. The following finalize script set for node001 shows how they can be used:

Example

```
[head->device*[node001*]]% get finalizescript
#!/bin/bash
echo "These are the interfaces:" >> /localdisk/env
CMD_ENV=`env`
function parser {
  for s in TYPE IP NETMASK; do
    echo $((grep CMD_INTERFACE_{1/:-}_-${s} | grep -Po "[\w.]+${s}") <<< "${CMD_ENV[@]}")
  done
}
for interface in $CMD_INTERFACES
do
  read -r type ip mask <<< $(parser $interface)

  echo $interface type=$type >> /localdisk/env
  echo $interface ip=$ip >> /localdisk/env
  echo $interface netmask=$mask >> /localdisk/env
done
```

The technique of storage of values in a file under the path within the node of /localdisk/ is described later on in section E.5.2. When the node boots up and runs the finalize script, then files stored under /localdisk/, end up under the path of / after the node is fully up.

The detailed workings of the parser function in the preceding bash script are not easy, but the result is that the parser function returns output so that the interface type, IP address, and netmask are listed for each interface. The parser works for physical interfaces, VLAN interfaces, and alias interfaces. For example, if there are two interfaces, eth0 and eth0:1, then the file env might be seen to have the following data:

Example

```
[root@head ~]# ssh node001 cat /env
These are the interfaces:
eth0 type=physical
eth0 ip=10.141.0.1
eth0 netmask=255.255.0.0
eth0:1 type=alias
eth0:1 ip=10.141.0.2
eth0:1 netmask=255.255.0.0
```

For remotely mounted devices, the name of the environment variables for mount entries have the following naming convention:

Description	Naming Convention
volume	CMD_FSMOUNT_<x>_DEVICE
mount point	CMD_FSMOUNT_<x>_MOUNTPOINT
filesystem type	CMD_FSMOUNT_<x>_FILESYSTEM
mount point options	CMD_FSMOUNT_<x>_OPTIONS

For the names, the entries `<x>` are substituted with the local mount point path, such as `/cm/shared`, but with the `/` character replaced with the text `_SLASH_`. So, for a local mount point path `/cm/shared`, the name of the associated volume environment variable becomes `CMD_FSMOUNT__SLASH_cm_SLASH_shared_DEVICE`.

A similar naming convention is applicable to the names of the environment variables for the export entries:

Description	Naming Convention
exported system writable?	<code>CMD_FSEXPORT_<y>_ALLOWWRITE</code>
allowed hosts or networks	<code>CMD_FSEXPORT_<y>_HOSTS</code>
path on exporter	<code>CMD_FSMOUNT_<y>_PATH</code>

Here, the entry `<y>` is replaced by the file path to the exported filesystem on the exporting node. This is actually the same as the value of `"CMD_FSMOUNT_<y>_PATH"`, but with the `/` character replaced with the text `_SLASH_`.

The entries for the local mount values and the export values in the table in section E.3 are the default values for a newly installed cluster. If the administrator wishes to add more devices and mount entries, this is done by configuring `fsexports` on the head node, and `fsmounts` on the regular nodes, using Bright View or `cmsh` (section 3.10).

E.5 Storing A Configuration To A Filesystem

E.5.1 Storing With Initialize Scripts

The `initialize` script (section 5.4.5) runs after the install-mode type and execution have been determined (section 5.4.4), but before unloading specific drivers and before partitions are checked and filesystems mounted (section 5.4.6). Data output cannot therefore be written to a local drive. It can however be written by the script to the `tmpfs`, but data placed there is lost quite soon, namely during the `pivot_root` process that runs when the node-installer hands over control to the `init` process running from the local drive. However, if needed, the data can be placed on the local drive later by using the `finalize` script to copy it over from the `tmpfs`.

Due to this, and other reasons, a `finalize` script is easier to use for an administrator than an `initialize` script, and the use of the `finalize` script is therefore preferred.

E.5.2 Ways Of Writing A Finalize Script To Configure The Destination Nodes

Basic Example—Copying A File To The Image

For a `finalize` script (section 5.4.11), which runs just before switching from using the `ramdrive` to using the local hard drive, the local hard drive is mounted under `/localdisk`. Data can therefore be written to the local hard drive if needed, but is only persistent until a reboot, when it gets rewritten. For example, predetermined configuration files can be written from the NFS drive for a particular node, or they can be written from an image prepared earlier and now running on the node at this stage, overwriting a node-installer configuration:

Example

```
#!/bin/bash
cp /etc/myapp.conf.overwrite /localdisk/etc/myapp.conf
```

This technique is used in a `finalize` script example in section 3.15.4, except that an `append` operation is used instead of a `copy` operation, to overcome a network issue by modifying a network configuration file slightly.

There are three important considerations for most `finalize` scripts:

1. Running A Finalize Script Without `exit 0` Considered Harmful

Failed Finalize Script Logic Flow: For a default configuration without a finalize script, if PXE boot fails from the network during node provisioning, the node then goes on to attempt booting from the local drive via iPXE (section 5.1.2).

However, if the configuration has a finalize script, such as in the preceding example, and if the finalize script fails, then the failure is passed to the node installer.

Avoiding Remote Node Hang During A Finalize Script: `exit 0` Recommended: If the node-installer fails, then no attempt is made to continue booting, and the node remains hung at that stage. This is usually undesirable, and can also make remote debugging of a finalize script annoying.

Adding an `exit 0` to the end of the finalize script is therefore recommended, and means that an error in the script will still allow the node-installer to continue with an attempt to boot from the local drive.

Debugging Tips When A Node Hangs During A Finalize Script: If there is a need to understand the failure, then if the node-installer hangs, the administrator can ssh into the node into the node-installer environment, and run the finalize script manually to debug it. Once the bug has been understood, the script can be copied over to the appropriate location in the head node, for nodes or categories.

Additional aid in understanding a failure may be available by looking through the node-installer logs. The debug mode for the node-installer can be enabled by setting `debug=true` instead of `debug=false` in the file `/cm/node-installer/scripts/node-installer.conf` (for multiarch/multidistro configurations the path takes the form: `/cm/node-installer-<distribution>-<architecture>/scripts/node-installer.conf`).

Another way to help debug a failure could be by setting custom event messages in the script, as explained on page 581.

2. Protecting A Configuration File Change From Provisioning Erasure With `excludelistupdate`

In the preceding example, the finalize script saves a file `/etc/myapp.conf` to the destination nodes.

To protect such a configuration file from erasure, its file path must be covered in the second sublist in the `excludelistupdate` list (section 5.6.1).

3. Finalize scripts cannot modify `/proc`, `/sys`, and `/dev` filesystems of end result on node directly.

The `/proc`, `/sys`, and `/dev` filesystems are unmounted after the finalize script is run before pivoting into the root filesystem under the `/localdisk` directory, which means any changes made to them are simply discarded. To change values under these filesystems on the node, an `rc.local` file inside the software image can be used.

For example, if swappiness is to be set to 20 via the `/proc` filesystem, one way to do it is to set it in the `rc.local` file:

Example

```
# cat /cm/images/<image-name>/etc/rc.local | grep -v ^# | grep .
echo 20 > /proc/sys/vm/swappiness
exit 0
# chmod 755 /cm/images/<image-name>/etc/rc.d/rc.local      # must be made executable
```

The preceding way of using `rc.local` set to run a command to modify the image just for illustration. A better way to get the same result in this case would be to not involve `rc.local`, but to add

a line within the `/cm/images/<image-name>/etc/sysctl.conf` file:

```
vm.swappiness = 20
```

Copying A File To The Image—Decision Based On Detection

Detection within a basic `finalize` script is useful extra technique. The `finalize` script example of section 3.15.4 does detection too, to decide if a configuration change is to be done on the node or not.

A further variation on a `finalize` script with detection is a script selecting from a choice of possible configurations. A symlink is set to one of the possible configurations based on hardware detection or detection of an environment variable. The environment variable can be a node parameter or similar, from the table in section E.3. If it is necessary to overwrite different nodes with different configurations, then the previous `finalize` script example might become something like:

Example

```
#!/bin/bash
if [[ $CMD_HOSTNAME = node00[1-7] ]]
then ln -s /etc/myapp.conf.first /localdisk/etc/myapp.conf
fi
if [[ $CMD_HOSTNAME = node01[5-8] ]]
then ln -s /etc/myapp.conf.second /localdisk/etc/myapp.conf
fi
if [[ $CMD_HOSTNAME = node02[3-6] ]]
then ln -s /etc/myapp.conf.third /localdisk/etc/myapp.conf
fi
```

In the preceding example, the configuration file in the image has several versions: `/etc/myapp.conf.<first|second|third>`. Nodes `node001` to `node007` are configured with the first version, nodes `node015` to `node018` with the second version, and nodes `node023` to `node026` with the third version. It is convenient to add more versions to the structure of this decision mechanism.

Copying A File To The Image—With Environmental Variables Evaluated In The File

Sometimes there can be a need to use the `CMDaemon` environment variables within a `finalize` script to specify a configuration change that depends on the environment.

For example a special service may need a configuration file, `test`, that requires the hostname `myhost`, as a `parameter=value` pair:

Example

```
SPECIALSERVICEPARAMETER=myhost
```

Ideally the placeholder value `myhost` would be the hostname of the node rather than the fixed value `myhost`. Conveniently, the `CMDaemon` environment variable `CMD_HOSTNAME` has the name of the host as its value.

So, inside the configuration file, after the administrator changes the host name from its placeholder name to the environment variable:

```
SPECIALSERVICE=${CMD_HOSTNAME}
```

then when the node-installer runs the `finalize` script, the file could be modified in-place by the `finalize` script, and `${CMD_HOSTNAME}` be substituted by the actual hostname.

A suitable `finalize` Bash script, which runs an in-line Perl substitution, is the following:

```
#!/bin/bash
perl -p -i -e 's/\${([~}+)}\}/defined $ENV{${1}} ? $ENV{${1}} : $&/eg' /localdisk/some/directory/file
```

Here, `/some/directory/file` means that, if for example the final configuration file path for the node is to be `/var/spool/test` then the file name should be set to `/localdisk/var/spool/test` inside the `finalize` script.

The `finalize` script replaces all lines within the file that have environment variable names of the form:

```
PARAMETER=${<environment variable name>}
```

with the value of that environment variable. Thus, if `<environment variable name>` is `CMD_HOSTNAME`, then that variable is replaced by the name of the host.

E.5.3 Restricting The Script To Nodes Or Node Categories

As mentioned in section 2.1.3, node settings can be adjusted within a category. So the configuration changes to `ifcfg-eth0` is best implemented per node by accessing and adjusting the `finalize` script per node if only a few nodes in the category are to be set up like this. If all the nodes in a category are to be set up like this, then the changes are best implemented in a `finalize` script accessed and adjusted at the category level. Accessing the scripts at the node and category levels is covered in section E.2.

People used to normal object inheritance behavior should be aware of the following when considering category level and node level `finalize` scripts:

With objects, a node item value overrules a category level value. On the other hand, `finalize` scripts, while treated in an analogous way to objects, cannot always inherit properties from each other in the precise way that true objects can. Thus, it is possible that a `finalize` script run at the node level may not have anything to do with what is changed by running it at the category level. However, to allow it to resemble the inheritance behavior of object properties a bit, the node-level `finalize` script, if it exists, is always run after the category-level script. This gives it the ability to “overrule” the category level.

F

Workload Managers Quick Reference

F.1 Slurm

Slurm is a GPL-licensed workload management system and developed largely at Lawrence Livermore National Laboratory. The name was originally an acronym for Simple Linux Utility for Resource Management, but the acronym is deprecated because it no longer does justice to the advanced capabilities of Slurm.

The Slurm service and outputs are normally handled using the Bright View or `cmsh` front end tools for `CMDaemon` (section 7.4).

From the command line, direct Slurm commands that may sometimes come in useful include the following:

- `sacct`: used to report job or job step accounting information about active or completed jobs.

Example

```
# sacct -j 43 -o jobid,AllocCPUs,NCPUS,NNodes,NTasks,ReqCPUs
      JobID  AllocCPUS      NCPUS  NNodes  NTasks  ReqCPUS
-----
43                1          1        1         1         1
```

- `salloc`: used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute `srun` commands to launch parallel tasks.
- `sattach` used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.
- `sbatch`: used to submit a job script for later execution. The script typically contains one or more `srun` commands to launch parallel tasks.
- `sbcast`: used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared filesystem.
- `scancel`: used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.
- `scontrol`: the administrative tool used to view and/or modify Slurm state. Note that many `scontrol` commands can only be executed as user root.

Example

```
[fred@bright90 ~]$ scontrol show nodes
NodeName=bright90 Arch=x86_64 CoresPerSocket=1
  CPUAlloc=0 CPUErr=0 CPUTot=1 CPULoad=0.05 Features=(null)
...
```

If a node, for example node001, is stuck in a CG state (“completing”), and rebooting it is not feasible, then the following may clear it in some cases:

Example

```
[fred@bright90 ~]$ scontrol update nodename=node001 state=down reason=hung
[fred@bright90 ~]$ scontrol update nodename=node001 state=resume
```

- **sinfo**: reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

Example

```
bright90:~ # 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 bright90
```

- **smap**: reports state information for jobs, partitions, and nodes managed by Slurm, but graphically displays the information to reflect network topology.
- **squeue**: reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

Example

```
bright90:~ # 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 bright90
...
```

- **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>.

F.2 Univa Grid Engine

Univa Grid Engine (UGE) is a workload management system that was originally derived from Sun Grid Engine (SGE).

UGE services should be handled using CMDaemon, as explained in section 7.4.

Managing queues and jobs can be done via the `cmsh` or Bright View front ends, but can also be carried out via the UGE tools.

- The `sge_qmaster` daemon on the head node can be started, stopped, or viewed using `service sgemaster start | stop | status`, or alternatively handled at a lower, more fine-grained, level using the `-k` (kill), `-s` (show) and `-a` (activate) options of `qconf`. Further details on these options can be found in the `qconf` help text, or `man qconf(1)`. For example, help on the `-k` options can be found by running:

```
qconf -help | grep '\-k'
```

or running:

```
man qconf(1) and scrolling down to the interesting part by searching for shutdown).
```

- The `sge_execd` execution daemon running on each compute node accepts, manages, and returns the results of the jobs on the compute nodes. The daemon can be managed via `service sgeexecd start | stop | status`, or alternatively (de)registered from `qmaster` via `qconf` options.
- To see why a queue is in an error state, `qstat -explain E` shows the reason. Further details on these options can be found in the `qconf` help text, or `man qstat(1)`. Queues in an error state can be cleared with a `qmod -cq <queue name>`.

UGE can be configured and managed generally with the command line utility `qconf`, which is what most administrators become familiar with. UGE also provides a GUI alternative, `qmon`.

UGE commands are listed below. The details of these are in the `man` page of the command and the UGE documentation.

- `qalter`: modify existing batch jobs
- `qacct`: show usage information from accounting data
- `qconf`: configure UGE
- `qdel`: delete batch jobs
- `qhold`: place hold on batch jobs
- `qhost`: display compute node queues, states, jobs
- `qlogin`: start login-based interactive session with a node
- `qmake`: distributed, parallel make utility
- `qmod`: suspend/enable queues and jobs
- `qmon`: configure UGE with an X11 GUI interface
- `qping`: check `sge_qmaster` and `sge_execd` status
- `qqota`: 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`)

F.3 PBS Pro

The following commands can be used in PBS Pro to view queues, jobs, and server status:

```

qstat                query queue status
qstat -a            alternate display format, for all queued and running jobs
qstat -r            show only running or suspended jobs
qstat -q            show available queues
qstat -rn           only running or suspended jobs, with list of allocated nodes
qstat -i            information on queued, held, waiting jobs
qstat -B            display server status
qstat -u <username> show jobs for named user

```

Other useful commands are:

```

tracejob <job id>          show what happened today to <job id>
tracejob -n <number> <job id> search last <number> days for <job id>

qmgr                    administrator interface to batch system

qterm                    terminates PBS server (but Bright Cluster Manager starts pbs_server again)

pbsnodes <node>         query status of compute node
pbsnodes -a             query status of all compute nodes

```

The commands of PBS Pro are documented in the man pages, and also in the extensive documentation available at <http://www.altair.com/pbs-works-documentation/>

G

Metrics, Health Checks, Enummetrics, And Actions

This appendix describes the metrics (section G.1), health checks (section G.2), enummetrics (section 12.2.2), and actions (section G.4), along with their parameters, in a newly-installed cluster. Metrics, health checks, enummetrics, and actions can each be standalone scripts, or they can be built-ins. Standalone scripts can be those supplied with the system, or they can be custom scripts built by the administrator. Scripts often require environment variables (as described in section 3.3.1 of the *Developer Manual*). On success scripts must exit with a status of 0, as is the normal practice.

G.1 Metrics And Their Parameters

A list of metric names can be viewed, for example, for the head node, using `cmsh` as follows (section 12.5.3):

```
[bright90 ~]# cmsh -c "monitoring measurable; list metric"
```

Metrics are listed in this section as four types:

- regular metrics (section G.1.1)
- NFS metrics (section G.1.2)
- monitoring system metrics (section G.1.3)
- GPU metrics (section G.1.4)
- Job metrics (section G.1.5)
- Prometheus metrics (section G.1.6)

G.1.1 Regular Metrics

Table G.1.1: List Of Metrics

Metric	Description
--------	-------------

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
AlertLevel	Indicates the healthiness of a device based on severity of events (section 12.2.8). The lower it is, the better. There are 3 parameters it can take: <ul style="list-style-type: none"> count: the number of active triggers maximum: the maximum alert level of all active triggers sum: the summed alert level of all active triggers
AvgJobDuration	Average Job Duration of current jobs
BlockedProcesses	Blocked processes waiting for I/O
BufferMemory	System memory used for buffering
BytesRecv*	Bytes/s received [‡]
BytesSent*	Bytes/s sent [‡]
CPUGuest	CPU time spent in guest mode.
CPUIidle	CPU time spent in idle mode.
CPUIrq	CPU time spent in servicing IRQ.
CPUNice	CPU time spent in nice mode.
CPUSoftIrq	CPU time spent in servicing soft IRQ.
CPUSteal	CPU time spent in steal mode.
CPUSystem	CPU time spent in system mode.
CPUser	CPU time spent in user mode.
CPUWait	CPU time spent in I/O wait mode.
CacheMemory	System memory used for caching.
CompletedJobs	Successfully completed jobs
Cores	Number of cores for a node
CoresTotal	Total number of known cores for all nodes
CoresUp	Number of cores for all nodes marked as UP
CtxtSwitches*	Context switches/s
Current_1**	First current seen by BMC sensor, in amps (file: sample_ipmi)
Current_2**	Second current seen by BMC sensor, in amps (file: sample_ipmi)
DevicesClosed	Number of devices not marked as UP or DOWN
DevicesDown	Number of devices marked as DOWN
DevicesTotal	Total number of devices
DevicesUp	Number of devices in status UP. A node (head, regular, virtual, cloud) or GPU Unit is not classed as a device. A device can be an item such as a switch, PDU, chassis, or rack, if the item is enabled and configured for management.
DropRecv*	Packets/s received and dropped [‡]
DropSent*	Packets/s sent and dropped [‡]
EC2SpotPrice	Amazon EC2 price for spot instances
EccDBitGPU**	Total number of double bit ECC errors (file: sample_gpu)

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
EccSBitGPU**	Total number of single bit ECC errors (file: sample_gpu)
ErrorsRecv*	Packets/s received with error [‡]
ErrorsSent*	Packets/s sent with error [‡]
EstimatedDelay	Estimated delay time to execute jobs
FailedJobs	Failed completed jobs
Forks*	Forked processes/s
FrameErrors*	Packet framing errors/s [‡]
FreeSpace	Free space for non-root user. Takes mount point as a parameter
GPUUnitsClosed	Number of GPU units not marked as UP or DOWN
GPUUnitsDown	Number of GPU units marked as DOWN
GPUUnitsTotal	Total number of GPU Units
GPUUnitsUp	Number of GPU units marked as UP
HardwareCorruptedMemory	Hardware corrupted memory detected by ECC
IOInProgress	I/O operations in progress [†]
IOTime	I/O operations time in milliseconds/s [†]
Interrupts	System interrupts
IpForwDatagrams*	Input IP datagrams/s to be forwarded
IpFragCreates*	IP datagram fragments/s generated
IpFragFails*	IP datagrams/s which needed to be fragmented but could not
IpFragOKs*	IP datagrams/s successfully fragmented
IpInAddrErrors*	Input datagrams/s discarded because the IP address in their header was not a valid address
IpInDelivers*	Input IP datagrams/s successfully delivered
IpInDiscards*	Input IP datagrams/s discarded
IpInHdrErrors*	Input IP datagrams/s discarded due to errors in their IP headers
IpInReceives*	Input IP datagrams/s, including ones with errors, received from all interfaces
IpInUnknownProtos*	Input IP datagrams/s received but discarded due to an unknown or unsupported protocol
IpOutDiscards*	Output IP datagrams/s discarded
IpOutNoRoutes*	Output IP datagrams/s discarded because no route could be found
IpOutRequests*	Output IP datagrams/s supplied to IP in requests for transmission
IpReasmOKs*	IP datagrams/s successfully re-assembled
IpReasmReqds*	IP fragments/s received needing re-assembly
LoadFifteen	Load average on 15 minutes
LoadFive	Load average on 5 minutes
LoadOne	Load average on 1 minute
MajorPageFaults*	Page faults/s that require I/O

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
MemoryAvailable	Available system memory
MemoryFree	Free system memory
MemoryTotal	Total system memory
MemoryUsed	Used system memory
MemoryUsed:cmd	Used system memory for process cmd
MergedReads*	Merged reads/s [†]
MergedWrites*	Merged writes/s [†]
NodesClosed	Number of nodes not marked as UP or DOWN
NodesDown	Number of nodes marked as DOWN
NodesInQueue	Number of nodes in the queue
NodesTotal	Total number of nodes
NodesUp	Number of nodes in status UP
OccupationRate	<p>Cluster occupation rate—a normalized cluster load percentage. 100% means all cores on all nodes are fully loaded.</p> <p>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.</p> <p>A high value can indicate the cluster is being used optimally. However, a value that is 100% most of the time suggests the cluster may need to be expanded.</p>
PacketsRecv*	Packets/s received [‡]
PacketsSent*	Packets/s sent [‡]
PageFaults*	Page faults/s
PageIn	Number of bytes the system has paged in from disk
PageOut	Number of bytes the system has paged out to disk
PageSwapIn	Number of bytes the system has swapped in from disk
PageSwapOut	Number of bytes the system has swapped out to disk
PDUBankLoad	Total PDU bank load, in amps
PDUload	Total PDU phase load, in amps
PDUUptime*	PDU uptime per second. I.e. ideally=1, but in practice has jitter effects.
ProcessCount	Total number of all processes in the OS. These are running processes (RunningProcesses) and blocked processes(BlockedProcesses).
QueuedJobs	Queued jobs
ReadTime*	Read time in milliseconds/s [†]
Reads*	Reads/s completed successfully [†]
RunningJobs	Running jobs

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
RunningProcesses	Running processes
SMARTDATemp	Temperature of a Hard Disk Assembly [†]
SMARTReallocSecCnt	SMART reallocated sectors count [†]
SMARTSeekErrRate	SMART seek errors/s [†]
SMARTSeekTimePerf	SMART average seek time [†]
SMARTSoftReadErrRate	SMART software read errors/s [†]
SectorsRead*	Sectors/s read successfully/s [†]
SectorsWritten*	Sectors/s written successfully [†]
SwapFree	Free swap memory
SwapTotal	Total swap memory
SwapUsed	Used swap memory
SystemTime:cmd	System time used by CMDaemon
TcpCurrEstab	TCP connections that are either ESTABLISHED or CLOSE-WAIT
TcpInErrs*	Input IP segments/s received in error
TcpRetransSegs*	Total number of IP segments/s re-transmitted
ThreadsUsed:cmd	Threads used by CMDaemon
TotalCPUIdle	Cluster-wide core usage in idle tasks (sum of all CPUIdle metric percentages)
TotalCPUSystem	Cluster-wide core usage in system mode (sum of all CPUSystem metric percentages)
TotalCPUUser	Cluster-wide core usage in user mode (sum of all CPUUser metric percentages)
TotalGPUUtilization	Average of gpu_utilization:average (table G.1.4) over all nodes
TotalMemoryFree	Cluster-wide total of memory free
TotalMemoryUsed	Cluster-wide total of memory used
TotalSwapFree	Cluster-wide total swap free
TotalSwapUsed	Cluster-wide total swap used
TotalUser	Total number of known users
TotalUserLogin	Total number of logged in users
UdpInDatagrams*	Input UDP datagrams/s delivered to UDP users
UdpInErrors*	Input UDP datagrams/s received that could not be delivered/s for other reasons (no port excl.)
UdpNoPorts*	Received UDP datagrams/s for which there was no application at the destination port
UniqueUserLogin	Number of unique users logged in
Uptime*	System uptime per second. Ie, ideally=1, but in practice has jitter effects
UsedSpace	Total used space by a mount point. Takes mount point as a parameter.

...continues

Table G.1.1: List Of Metrics...continued

Metric	Description
UserTime:cmd	User time used by CMDaemon
VirtualMemoryUsed:cmd	Virtual memory used by CMDaemon
WriteTime*	Write time in milliseconds/s [†]
Writes*	Writes/s completed successfully [†]

* Cumulative metric. I.e. the metric is derived from cumulative raw measurements taken at two different times, according to:

$$metric_{time_2} = \frac{measurement_2 - measurement_1}{time_2 - time_1}$$

** Standalone scripts, not built-ins.

If sampling from a head node, the script is in directory: /cm/local/apps/cmd/scripts/metrics/

For regular nodes, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/metrics/

[†] Takes block device name (sda, sdc and so on) as parameter

[‡] Takes interface device name (eth0, eth1 and so on) as parameter

G.1.2 NFS Metrics

The NFS metrics correspond to `nfsstat` output, and are shown in table G.1.2.

Table G.1.2: NFS Metrics

NFS Metric	Description
nfs_client_packet_packets	NFS client packets statistics: packets
nfs_client_packet_tcp	NFS client package statistics: TCP/IP packets
nfs_client_packet_tcpconn	NFS client package statistics: TCP/IP connections
nfs_client_packet_udp	NFS client package statistics: UDP packets
nfs_client_rpc_authrefrsh	NFS Client RPC statistics: authenticated refreshes to RPC server
nfs_client_rpc_calls	NFS Client RPC statistics: calls
nfs_client_rpc_retrans	NFS Client RPC statistics: re-transmissions
nfs_server_file_anon	NFS Server file statistics: anonymous access
nfs_server_file_lookup	NFS Server file statistics: look-ups
nfs_server_file_ncachedir	NFS Server file statistics: ncachedir
nfs_server_file_stale	NFS Server file statistics: stale files
nfs_server_packet_packets	NFS Server packet statistics: packets
nfs_server_packet_tcp	NFS Server packet statistics: TCP/IP packets
nfs_server_packet_tcpconn	NFS Server packet statistics: TCP/IP connections
nfs_server_packet_udp	NFS Server packet statistics: UDP packets
nfs_server_reply_hits	NFS Server reply statistics: hits
nfs_server_reply_misses	NFS Server reply statistics: misses
nfs_server_reply_nocache	NFS Server reply statistics: no cache
nfs_server_rpc_badauth	NFS Server RPC statistics: bad authentication
nfs_server_rpc_badcalls	NFS Server RPC statistics:bad RPC requests
nfs_server_rpc_badclnt	NFS Server RPC statistics: badclnt
nfs_server_rpc_calls	NFS Server RPC statistics: all calls to NFS and NLM
nfs_server_rpc_xdrcll	NFS Server RPC statistics: malformed XDR calls

...continues

Table G.1.2: NFS Metrics

NFS Metric	Description
nfs_v3_client_access	NFSv3 client statistics: access
nfs_v3_client_fsinfo	NFSv3 client statistics: static file system information
nfs_v3_client_fsstat	NFSv3 client statistics: dynamic file system status
nfs_v3_client_getattr	NFSv3 client statistics: file system attributes
nfs_v3_client_lookup	NFSv3 client statistics: lookup
nfs_v3_client_pathconf	NFSv3 client statistics: configuration path
nfs_v3_client_read	NFSv3 client statistics: reads
nfs_v3_client_total	NFSv3 client statistics: total
nfs_v3_server_access	NFSv3 server statistics: access
nfs_v3_server_create	NFSv3 server statistics: create
nfs_v3_server_fsinfo	NFSv3 server statistics: static file system information
nfs_v3_server_fsstat	NFSv3 server statistics: dynamic file system information
nfs_v3_server_getattr	NFSv3 server statistics: file system attributes gets
nfs_v3_server_lookup	NFSv3 server statistics: file name look-ups
nfs_v3_server_mkdir	NFSv3 server statistics: directory creation
nfs_v3_server_null	NFSv3 server statistics: null operations
nfs_v3_server_pathconf	NFSv3 server statistics: retrieve POSIX information
nfs_v3_server_read	NFSv3 server statistics: reads
nfs_v3_server_readdirplus	NFSv3 server statistics: REaddirPLUS procedures
nfs_v3_server_readlink	NFSv3 server statistics: Symbolic link reads
nfs_v3_server_setattr	NFSv3 server statistics: file system attribute sets
nfs_v3_server_total	NFSv3 server statistics: total
nfs_v3_server_write	NFSv3 server statistics: writes

G.1.3 Monitoring System Metrics

The monitoring system metrics correspond to metrics for the monitoring data storage itself. These are shown in table G.1.3.

Table G.1.3: Monitoring System Metrics

Monitoring System Metric	Description
Mon::CacheGather::handled	Cache gathers handled/s
Mon::CacheGather::miss	Cache gathers missed/s
Mon::DataProcessor::handled	Data processors handled/s
Mon::DataProcessor::miss	Data processors missed /s
Mon::DataTranslator::handled	Data translators handled/s

...continues

Table G.1.3: Monitoring System Metrics...continued

Monitoring System Metric	Description
Mon::DataTranslator::miss	Data translators missed/s
Mon::EntityMeasurableCache::handled	Measurable cache handled/s
Mon::EntityMeasurableCache::miss	Measurable cache missed/s
Mon::MeasurableBroker::handled	Measurable broker handled/s
Mon::MeasurableBroker::miss	Measurable broker missed/s
Mon::OOB::TaskService::handled	Out-of-band task service handled/s
Mon::OOB::TaskService::miss	Out-of-band task service missed/s
Mon::Replicate::Collector::handled	Replication collection handled/s
Mon::Replicate::Collector::miss	Replication collection missed/s
Mon::Replicate::Combiner::handled	Replication combiner handled/s
Mon::Replicate::Combiner::miss	Replication combiner missed/s
Mon::RepositoryAllocator::handled	Repository allocator handled/s
Mon::RepositoryAllocator::miss	Repository allocator missed/s
Mon::RepositoryTrim::handled	Repository trim handled/s
Mon::RepositoryTrim::miss	Repository trim missed/s
Mon::Storage::Engine::elements	Storage engine data elements, in total
Mon::Storage::Engine::size	Storage engine size, in bytes
Mon::Storage::Engine::usage	Storage engine usage
Mon::Storage::Message::elements	Storage message data elements, in total
Mon::Storage::Message::size	Storage message size in bytes
Mon::Storage::Message::usage	Storage message usage
Mon::Storage::RepositoryId::elements	Storage repository ID data elements, in total
Mon::Storage::RepositoryId::size	Storage repository ID, size, in bytes
Mon::Storage::RepositoryId::usage	Repository ID usage
Mon::TaskInitializer::handled	Task initializer handled/s
Mon::TaskInitializer::miss	Task initializer missed/s
Mon::TaskSampler::handled	Task sampler handled/s
Mon::TaskSampler::miss	Task sampler missed/s
Mon::Trigger::Actuator::handled	Trigger actuators handled/s
Mon::Trigger::Actuator::miss	Trigger actuators missed/s
Mon::Trigger::Dispatcher::handled	Trigger dispatchers handled/s
Mon::Trigger::Dispatcher::miss	Trigger dispatchers missed/s
Prometheus::DataTranslator::handled	DataTranslator queries handled. The DataTranslator is a translation layer between PromQL format and Bright Cluster Manager format sampling.
Prometheus::DataTranslator::miss	DataTranslator misses

G.1.4 GPU Metrics

The GPU metrics correspond to metrics provided by the GPU devices. The device parameter for the GPU metric specifies the device slot number that the GPU uses. The parameter takes the form `gpu0`, `gpu1`, and so on. It is appended to the metric with a colon character. For example, the `gpu_ecc_dbe_agg`

metric, if used with `gpu1`, is specified as:

Example

```
gpu_ecc_dbe_agg:gpu1
```

The data producer (section 12.2.10) for the GPU metrics is `GPUSampler`.

The available GPU metrics for the K40 and V100 GPUs are listed in table G.1.4. The available GPU health checks for the K40 and V100 GPUs are listed in table G.2.1.

Table G.1.4: GPU Metrics

GPU Metric	Description
<code>gpu_ecc_dbe_agg</code>	Total double bit aggregate ECC errors
<code>gpu_ecc_dbe_vol</code>	Total double bit volatile ECC errors
<code>gpu_ecc_sbe_agg</code>	Total single bit aggregate ECC errors
<code>gpu_ecc_sbe_vol</code>	Total single bit volatile ECC errors
<code>gpu_enforced_power_limit</code>	GPU enforced power limit
<code>gpu_fb_total</code>	GPU framebuffer size
<code>gpu_fb_used</code>	Amount of GPU memory used
<code>gpu_memory_clock</code>	GPU memory clock
<code>gpu_power_management_limit</code>	GPU power management limit
<code>gpu_power_usage</code>	GPU power usage
<code>gpu_shutdown_temp</code>	GPU shutdown temperature
<code>gpu_slowdown_temp</code>	GPU slowdown temperature
<code>gpu_sm_clock</code>	GPU shader multiprocessor clock
<code>gpu_temperature</code>	GPU temperature
<code>gpu_utilization:average</code>	Average GPU utilization percentage. When the average parameter replaces the usual device parameter, it samples the average percentage over all the GPU devices that are in use on that node
<code>gpu_utilization:gpu0</code>	GPU utilization percentage for GPU using slot 0 on that node. This follows the usual format

G.1.5 Job Metrics

Job metrics are introduced in section 12.8.1.

Basic Job Metrics

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`.

Table G.1.5.1: Basic Job Metrics

Job Metric	Description	Cgroup Source File
------------	-------------	--------------------

...continues

...continued

Job Metric	Description	Cgroup Source File
blkio.time:<device>	Time job had I/O access to device	blkio.time_recursive
blkio.sectors:<device>	Sectors transferred to or from specific devices by a cgroup	blkio.sectors_recursive
blkio.io_service_read:<device>	Bytes read	blkio.io_service_bytes_recursive
blkio.io_service_write:<device>	Bytes written	blkio.io_service_bytes_recursive
blkio.io_service_sync:<device>	Bytes transferred synchronously	blkio.io_service_bytes_recursive
blkio.io_service_async:<device>	Bytes transferred asynchronously	blkio.io_service_bytes_recursive
blkio.io_wait_time_read:<device>	Total time spent waiting for service in the scheduler queues for I/O read operations	blkio.io_wait_time_recursive
blkio.io_wait_time_write:<device>	Total time spent waiting for service in the scheduler queues for I/O write operations	blkio.io_wait_time_recursive
blkio.io_wait_time_sync:<device>	Total time spent waiting for service in the scheduler queues for I/O synchronous operations	blkio.io_wait_time_recursive
blkio.io_wait_time_async:<device>	Total time spent waiting for service in the scheduler queues for I/O asynchronous operations	blkio.io_wait_time_recursive
cpuacct.usage	Total CPU time consumed by all job processes	cpuacct.usage
cpuacct.stat.user	User CPU time consumed by all job processes	cpuacct.stat
cpuacct.stat.system	System CPU time consumed by all job processes	cpuacct.stat

...continues

...continued

Job Metric	Description	Cgroup Source File
memory.usage	Total current memory usage	memory.usage_in_bytes
memory.memsw.usage	Sum of current memory plus swap space usage	memory.memsw.usage_in_bytes
memory.memsw.max_usage	Maximum amount of memory and swap space used	memory.memsw.max_usage_in_bytes
memory.failcnt	How often the memory limit has reached the value set in memory.limit_in_bytes	memory.failcnt
memory.memsw.failcnt	How often the memory plus swap space limit has reached the value set in memory.memsw.limit_in_bytes	memory.memsw.failcnt
memory.swap	Total swap usage	memory
memory.cache	Total page cache, including tmpfs (shmem)	memory
memory.mapped_file	Size of memory-mapped mapped files, including tmpfs (shmem)	memory
memory.unevictable	Memory that cannot be reclaimed	memory

The third column in the table shows the precise source file name that is used when the value is retrieved. These files are all virtual files, and are created as the cgroup controllers are mounted to the cgroup directory. In this case several controllers are mounted to the same directory, which means that all the virtual files will show up in that directory, and in its associated subdirectories—job cgroup directories—when the job runs.

Advanced Job Metrics

The metrics in the preceding table are enabled by default. There are also over 40 other advanced metrics that can be enabled via the `Enable Advanced Metrics` property of the `jobmetricsettings` object:

```
[bright90->monitoring->setup[JobSampler]->jobmetricsettings]% show
Parameter                               Value
-----
Enable Advanced Metrics                 no
Exclude Devices                         loop,sr
Exclude Metrics
Include Devices
Include Metrics
Revision
Sampling Type                           Both
[bright90->monitoring->setup[JobSampler]->jobmetricsettings]% set enableadvancedmetrics yes
[bright90->monitoring->setup*[JobSampler*]->jobmetricsettings*]% commit
```

The advanced job metrics are:

Table G.1.5.2: Advanced Job Metrics

Advanced Job Metric	Description	Cgroup Source File
<code>blkio.io_service_time_read:<device></code>	Total time between request dispatch and request completion according to CFQ scheduler for I/O read operations	<code>blkio.io_service_time_recursive</code>
<code>blkio.io_service_time_write:<device></code>	Total time between request dispatch and request completion according to CFQ scheduler for I/O write operations	<code>blkio.io_service_time_recursive</code>
<code>blkio.io_service_time_sync:<device></code>	Total time between request dispatch and request completion according to CFQ scheduler for I/O synchronous operations	<code>blkio.io_service_time_recursive</code>
<code>blkio.io_service_time_async:<device></code>	Total time between request dispatch and request completion according to CFQ scheduler for I/O asynchronous operations	<code>blkio.io_service_time_recursive</code>
<code>blkio.io_serviced_read:<device></code>	Read I/O operations	<code>blkio.io_serviced_recursive</code>
<code>blkio.io_serviced_write:<device></code>	Write I/O operations	<code>blkio.io_serviced_recursive</code>
<code>blkio.io_serviced_sync:<device></code>	Synchronous I/O operations	<code>blkio.io_serviced_recursive</code>
<code>blkio.io_serviced_async:<device></code>	Asynchronous I/O operations	<code>blkio.io_serviced_recursive</code>
<code>blkio.io_merged_read:<device></code>	Number of block I/Os (requests) merged into requests for I/O read operations	<code>blkio.io_merged_recursive</code>

...continues

...continued

Advanced Job Metric	Description	Cgroup Source File
<code>blkio.io_merged_write:<device></code>	Number of block I/Os (requests) merged into requests for I/O write operations	<code>blkio.io_merged_recursive</code>
<code>blkio.io_merged_sync:<device></code>	Number of block I/Os (requests) merged into requests for I/O synchronous operations	<code>blkio.io_merged_recursive</code>
<code>blkio.io_merged_async:<device></code>	Number of block I/Os (requests) merged into requests for I/O asynchronous operations	<code>blkio.io_merged_recursive</code>
<code>blkio.io_queued_read:<device></code>	Number of requests queued for I/O read operations	<code>blkio.io_queued_recursive</code>
<code>blkio.io_queued_write:<device></code>	Number of requests queued for I/O write operations	<code>blkio.io_queued_recursive</code>
<code>blkio.io_queued_sync:<device></code>	Number of requests queued for I/O synchronous operations	<code>blkio.io_queued_recursive</code>
<code>blkio.io_queued_async:<device></code>	Number of requests queued for I/O asynchronous operations	<code>blkio.io_queued_recursive</code>
<code>memory.rss</code>	Anonymous and swap cache, not including tmpfs (shmem)	<code>memory</code>
<code>memory.pgpgin</code>	Number of pages paged into memory	<code>memory</code>
<code>memory.pgpgout</code>	Number of pages paged out of memory	<code>memory</code>
<code>memory.active_anon</code>	Anonymous and swap cache on active least-recently-used (LRU) list, including tmpfs (shmem)	<code>memory</code>

...continues

...continued

Advanced Job Metric	Description	Cgroup Source File
memory.inactive_anon	Anonymous and swap cache on inactive LRU list, including tmpfs (shmem)	memory
memory.active_file	File-backed memory on active LRU list	memory
memory.inactive_file	File-backed memory on inactive LRU list	memory
memory.hierarchical_memory_limit	Memory limit for the Hierarchy that contains the memory cgroup of job	memory
memory.hierarchical_memsw_limit	Memory plus swap limit for Hierarchy that contains the memory cgroup of job	memory

G.1.6 Prometheus Metrics

Prometheus metrics are introduced in section 12.9.1.

The data producers for Prometheus metrics are JobSampler, JobMetadataSampler, and some others.

Table G.1.6: Prometheus Metrics

Prometheus Metric	Description (for a job, unless asterisked)
job_blkio_io_merged	Number of block I/Os (requests) merged into requests for I/O operations by a cgroup
job_blkio_io_queued	Number of requests queued for I/O operations by a cgroup
job_blkio_io_service_bytes	Reports the number of bytes transferred to or from specific devices by a cgroup as seen by the CFQ scheduler
job_blkio_io_service_bytes_total	Reports the number of bytes transferred to or from specific devices by a cgroup as seen by the CFQ scheduler (for all jobs)*
job_blkio_io_service_time_seconds	Reports the total time in seconds between request dispatch and request completion for I/O operations on specific devices by a cgroup as seen by the CFQ scheduler

...continues

Table G.1.6: Prometheus Metrics...continued

Prometheus Metric	Description (for a job, unless asterisked)
<code>job_blkio_io_serviced</code>	Reports the number of I/O operations performed on specific devices by a cgroup as seen by the CFQ scheduler
<code>job_blkio_io_wait_time_seconds</code>	Reports the total time I/O operations on specific devices by a cgroup spent waiting for service in the scheduler queues
<code>job_blkio_sectors</code>	Reports the number of sectors transferred to or from specific devices by a cgroup
<code>job_blkio_time_seconds</code>	Reports the time that a cgroup had I/O access to specific devices
<code>job_cpuacct_stat_system</code>	System CPU time consumed by processes
<code>job_cpuacct_stat_user</code>	User CPU time consumed by processes
<code>job_cpuacct_usage_seconds</code>	CPU usage time consumed
<code>job_memory_active_anon_bytes</code>	Anonymous and swap cache on active least-recently-used (LRU) list, including tmpfs (shmem), in bytes
<code>job_memory_active_file_bytes</code>	File-backed memory on active LRU list, in bytes
<code>job_memory_cache_bytes</code>	Page cache, including tmpfs (shmem), in bytes
<code>job_memory_failcnt</code>	Reports the number of times that the memory limit has reached the value set in <code>memory.limit_in_bytes</code>
<code>job_memory_hierarchical_memory_limit_bytes</code>	Memory limit for the hierarchy that contains the memory cgroup, in bytes
<code>job_memory_hierarchical_mems_limit_bytes</code>	Memory plus swap limit for the hierarchy that contains the memory cgroup, in bytes
<code>job_memory_inactive_anon_bytes</code>	Anonymous and swap cache on inactive LRU list, including tmpfs (shmem), in bytes
<code>job_memory_inactive_file_bytes</code>	File-backed memory on inactive LRU list, in bytes
<code>job_memory_mapped_file_bytes</code>	Size of memory-mapped mapped files, including tmpfs (shmem), in bytes
<code>job_memory_mems_failcnt</code>	Reports the number of times that the memory plus swap space limit has reached the value set in <code>memory.mems.limit_in_bytes</code>
<code>job_memory_mems_max_usage_bytes</code>	Reports the maximum amount of memory and swap space used by processes in the cgroup, in bytes

...continues

Table G.1.6: Prometheus Metrics...continued

Prometheus Metric	Description (for a job, unless asterisked)
job_memory_memsw_usage_bytes	Reports the sum of current memory usage plus swap space used by processes in the cgroup, in bytes
job_memory_pgpgin_bytes	Number of pages paged into memory
job_memory_pgpgout_bytes	Number of pages paged out of memory
job_memory_rss_bytes	Anonymous and swap cache, not including tmpfs (shmem), in bytes
job_memory_swap_bytes	Swap usage, in bytes
job_memory_unevictable_bytes	Memory that cannot be reclaimed, in bytes,
job_memory_usage_bytes	Reports the total current memory usage by processes in the cgroup, in bytes
job_memory_usage_bytes_total	Reports the total current memory usage by processes in the cgroup, in bytes (for all jobs)*
job_metadata_allocated_cpu_cores	CPU cores used by a job by the user
job_metadata_allocated_gpus	GPUs used by a job by the user
job_metadata_is_running	Returns 1 if the job metadata sampler is running a job
job_metadata_is_waiting	Returns 1 if the job metadata sampler is waiting
job_metadata_num_cpus	Number of CPUs that the job runs on
job_metadata_num_nodes	Number of nodes that the job runs on
job_metadata_pending_jobs	Number of pending jobs for the user
job_metadata_running_jobs	Number of running jobs for the user
job_metadata_running_seconds	Time the job has run
job_metadata_waiting_seconds	Time the job has been waiting to run
users_job_effective_cpu_seconds:1w	CPU seconds used by users over the past week*
users_job_running_count:1w	Number of jobs run by users over the past week*
users_job_waiting_seconds:1w	Time users have been waiting for jobs to run over the past week*
users_job_wall_clock_seconds:1w	Time the jobs runs for users according to wall time over the past week*
users_job_wasted_cpu_seconds:1w	CPU time wasted during users jobs over the past week*

* total in the past 7x24x60x60 seconds, as measured at the time of sampling

G.1.7 Parameters For Metrics

Metrics have the parameters indicated by the left column in the following example:

Example

```
[bright90->monitoring->measurable[CPUUser]]% show
Parameter          Value
-----
Class              CPU
Consolidator       default (ProcStat)
Cumulative         yes
Description        CPU time spent in user mode
Disabled           no (ProcStat)
Gap                0 (ProcStat)
Maximal age        0s (ProcStat)
Maximal samples    4,096 (ProcStat)
Maximum            0
Minimum            0
Name               CPUUser
Parameter
Producer           ProcStat
Revision
Type               Metric
Unit               Jiffies
```

If the value is inherited from the producer, then it is shown in parentheses next to the value. An inherited value can be overwritten by setting it directly for the parameter of a measurable.

The meanings of the parameters are:

Class: A choice assigned to a metric. It can be an internal type, or it can be a standalone class type. A slash (/) is used to separate class levels. A partial list of the class values is:

- CPU: CPU-related
- Disk: Disk-related
- Internal: An internal metric
- Internal/Monitoring/Service: An internal metric for monitoring a service
- Internal/Monitoring/Storage: An internal metric for monitoring storage
- Job: Job metric
- Memory: Memory-related
- Network: Network-related
- OS: Operating-system-related
- Process: Process-related
- Prometheus: Prometheus-related
- Total: Total cluster-wide-related
- Workload: Workload-related

Consolidator: This is described in detail in sections 12.4.3 and 12.5.2

Cumulative: If set to no, then the raw value is treated as not cumulative (for example, CoresUp), and the raw value is presented as the metric value.

If set to yes, then the metric is treated as being cumulative, which means that a rate (per second) value is presented.

More explicitly: When set to `yes`, it means that the raw sample used to calculate the metric is expected to be cumulative, like, for example, the bytes-received counter for an Ethernet interface. This in turn means that the metric is calculated from the raw value by taking the difference in raw sample measurement values, and dividing it by the time period over which the raw values are sampled. Thus, for example:

- The bytes-received raw measurements, which accumulate as the packets are received, and are in bytes, and have `Cumulative` set to `yes`, and then have a corresponding metric, `BytesRecv`, with a value in bytes/second.
- The system uptime raw measurements, which accumulate at the rate of 1 second per second, and are in seconds, have `Cumulative` set to `yes`, and have a corresponding metric, `Uptime`, with a value that uses no units. Ideally, the metric has a value of 1, but in practice the measured value varies a little due to jitter.

Description: Description of the raw measurement used by the metric. Empty by default.

Disabled: If set to `no` (default) then the metric runs.

Gap: The number of samples that are allowed to be missed before a value of NaN is set for the value of the metric.

Maximal age: the maximum age of RLE samples that are kept. If `Maximal age` is set to 0 then the sample age is not considered. Units can be `w`, `d`, `h`, `m`, `s` (weeks, days, hours, minutes, seconds), with `s` as the default.

Maximal samples: the maximum number of RLE samples that are kept. If `Maximal samples` is set to 0 then the number of sample age is not considered.

Maximum: the default minimum value the y-axis maximum will take in graphs plotted in Bright View.¹

Minimum: the default maximum value the y-axis minimum will take in graphs plotted in Bright View.¹

Name: The name given to the metric.

Parameter: Parameter used for this metric. For example, `eth0` with the metric `BytesRecv`

Producer: The data producer that produces the metric

Revision: User-definable revision number for the object

Type: This can be one of `metric`, `healthcheck`, or `enummetric`

Unit: A unit for the metric. For example: `B/s` (bytes/second) for `BytesRecv` metric, or unit-less for the `Uptime` metric. A percent is indicated with `%`

¹To clarify the concept, if `maximum=3`, `minimum=0`, then a data-point with a y-value of 2 is plotted on a graph with the y-axis spanning from 0 to 3. However, if the data-point has a y-value of 4 instead, then it means the default y-axis maximum of 3 is re-sized to 4, and the y-axis will now span from 0 to 4.

G.2 Health Checks And Their Parameters

A list of health checks can be viewed, for example, for the head node, using `cmsh` as follows (section 12.5.3):

```
[bright90 ~]# cmsh -c "monitoring measurable; list healthcheck"
```

Health checks (section 12.2.4) are listed and described in this section.

G.2.1 Health Checks

Table G.2.1: List Of Health Checks

Name	Query (script response is PASS/FAIL)
ManagedServicesOk*	Are CMDaemon-monitored services all OK? If the response is FAIL, then at least one of the services being monitored is failing. The <code>latesthealthdata -v</code> command (section 12.6.3) should show which one(s). After correcting the problem with the service, a reset of the service is normally carried out (section 3.11, page 114).
Mon::Storage	Is space available for the monitoring system metrics (section G.1.3)?
chrootprocess	Are there daemon processes running using <code>chroot</code> in software images? (here: yes = FAIL). On failure, kill cron daemon processes running in the software images.
beegfs	Is BeeGFS properly connected as expected to all the nodes? Unreachable nodes and failed routes are listed in <code>InfoMessages</code> .
cmsh	Is <code>cmsh</code> available?
defaultgateway	Is there a default gateway available?
dellnss	If running, is the Dell NFS Storage Solution healthy?
diskspace	Is there less local disk space available to non-root users than any of the space parameters specified? <i>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 <code>freespace</code> (page 769) instead.</i>

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)								
	<p>Examples:</p> <ul style="list-style-type: none"> • <code>diskspace 10%</code> less than 10% space = FAIL, severity 10 • <code>diskspace 10% 20% 30%</code> less than 30% space = FAIL, with severity levels as indicated: <table border="1" data-bbox="571 526 903 694"> <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> • <code>diskspace 10GB 20GB</code> less than 20GB space = FAIL, severity 10 less than 10GB space = FAIL, severity 20 • <code>diskspace 10% 20% /var</code> For the filesystem /var: less than 20% space = FAIL, severity 10 less than 10% space = FAIL, severity 20 	space left	severity	10%	30	20%	20	30%	10
space left	severity								
10%	30								
20%	20								
30%	10								

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
dmesg	Is dmesg output OK? <i>Regexes to parse the output can be constructed in the configuration file at /cm/local/apps/cmd/scripts/healthchecks/configfiles/dmesg.py</i>
docker	Is Docker running OK? Checks for Docker server availability and corruption, dead containers, proper endpoints
dockerregistry	Is the Docker registry running OK? Checks registry endpoint and registry availability
exports	Are all filesystems as defined by the cluster management system exported?
etcd	Are the core etcd processes of Kubernetes running OK? Checks endpoints and interfaces
failedprejob	Are there failed prejob health checks (section 7.8.2)? Here: yes = FAIL. By default, the job ID is saved under /cm/shared/apps/<scheduler>/var/cm/ <ul style="list-style-type: none"> • On FAIL, in failedprejobs. • On PASS, in allprejobs <p>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.</p>
failover	Is the failover status OK?
gpu_health_driver	Is the Driver-related subsystem OK?
gpu_health_inforom	Is the Inforom OK?
gpu_health_mcu	Is the microcontroller unit OK?
gpu_health_mem	Is the memory subsystem OK?
gpu_health_nvlink	Is the NVLINK system OK?
gpu_health_pcie	Is the PCIe system OK?
gpu_health_pmu	Is the power management unit OK?
gpu_health_power	Is the power OK?
gpu_health_sm	Is the streaming multiprocessor OK?
gpu_health_thermal	Is the temperature OK?
hpraid	Are the HP Smart Array controllers OK?
ib	Is the InfiniBand Host Channel Adapter working properly? <i>A configuration file for this health check is at /cm/local/apps/cmd/scripts/healthchecks/configfiles/ib.py</i>

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
interfaces	Are the interfaces all up and OK?
ipmihealth	Is the BMC (IPMI or iLO) health OK? Uses the script <code>sample_ipmi</code> .
kuberneteschildnode	Are all Kubernetes child nodes up?
kubernetescomponen\tsstatus	Are all expected agents and services up and running for active nodes?
kubernetesnodesstatus	Is the status for all Kubernetes nodes OK?
kubernetespodsstatus	Is the status for all pods OK?
ldap	Can the ID of the user be looked up with LDAP?
lustre	Is the Lustre filesystem running OK?
marathon	Is Marathon running OK?
messagequeue	Is the OpenStack Message Queue (RabbitMQ) running OK?
megaraid	Are the MegaRAID controllers OK? <i>Either the proprietary MegaCLI software, or its successor, the proprietary StorCLI software is needed for this health check. The MegaCLI software was originally provided by LSI Logic, but LSI is now part of Broadcom. Both the MegaCLI software and the StorCLI software are now available from the Broadcom website (http://www.broadcom.com). For Bright Cluster Manager versions up to 9.2, the healthcheck first checks for MegaCLI, and then StorCLI, and uses the first binary that is detected. For versions later than 9.2, the healthcheck first checks for StorCLI, and then for MegaCLI, and uses the first binary that is detected.</i>
mounts	Are all mounts defined in the fstab OK?
mysql	Is the status and configuration of MySQL correct?
node-hardware-profile	Is the specified node's hardware configuration during health check use unchanged? <i>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"</i>
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 <code>/cm/local/apps/cmd/scripts/healthchecks/configfiles/oomkiller.conf</code> 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 <code>/var/spool/cmd/save-oomkilleraction</code> . <i>A consideration of the causes and consequences of the killed processes is strongly recommended. A reset of the node is generally recommended.</i>

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
opalinkhealth	Are the quality and the integrity of the Intel OPA HFI link OK?
openstackaddressremove	Is the addition and removal of objects in Keystone, Neutron, Nova, Cinder, Glance, and Heat working OK?
openstackagents	Are the agents (OpenStack service components) OK?
openstackauth	Is the authentication for Keystone, Neutron, Nova, Cinder, Glance, and Heat working OK?
openstackconfiguration	Are some basic configuration options for OpenStack working OK?
openstackdashboard	Is the Horizon dashboard login working OK?
openstackportstatus	Within OpenStack, are DHCP, router ports, and router gateway ports of the Neutron service all up?
openstackresponse	Is Keystone is responsive and are the endpoints of each service pingable?
openstackvmsok	Are the virtual machines under OpenStack running OK?
portchecker	Is the specified port on the specified host open for TCP (default) or UDP connections?
rogueprocess	<p>Are the processes that are running legitimate (ie, not 'rogue')? Besides the FAIL/PASS/UNKNOWN response to CMDaemon, also returns a list of rogue process IDs to file descriptor 3 (InfoMessages), which the killprocess action (page 793) 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:</p> <ul style="list-style-type: none"> • not part of the workload manager service or its jobs • not a root- or system-owned process • in the state Z, T, W, or X. States are described in the ps man pages in the section on "PROCESS STATE CODES"

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
schedulers	<p>Rogue process criteria can be configured in the file <code>/cm/local/apps/cmd/scripts/healthchecks/configfiles/rogueprocess.py</code> within the software image. To implement a changed criteria configuration, the software image used by systems on which the health check is run should be updated (section 5.6). For example, using: <code>cmsh -c "device; imageupdate -c default -w"</code> for the default category of nodes.</p>
smart	<p>Are the queue instances of all schedulers on a node healthy ?</p> <p>Is the SMART response healthy? The severities can be configured in the file <code>/cm/local/apps/cmd/scripts/healthchecks/configfiles/smart.conf</code>.</p> <p>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.</p> <p>The info messages can be suppressed by setting a whitelist of the disks to be checked within <code>/cm/local/apps/cmd/scripts/healthchecks/smart</code>.</p>
ssh2node	<p>Is passwordless ssh root login, from head to a node that is up, working? Some details of its behavior are:</p> <ul style="list-style-type: none"> • If root ssh login to the head node has been disabled, then the health check fails on the head node. • The healthcheck <code>ssh2node</code> fails if ssh certificate-based access, to a regular head node that is in the UP state, fails. The UP state is determined by whether <code>CMDaemon</code> is running on that node. • If the regular node is in a DOWN state—which could be due to <code>CMDaemon</code> being down, or the node having been powered off gracefully, or the node suffering a sudden power failure—then the health check responds with a PASS. The idea here is to check key access, and decouple it from the node state.

...continued

Table G.2.1: List Of Health Checks...continued

Name	Query (response is PASS/FAIL)
swraid	Are the software RAID arrays healthy?
testhealthcheck	<i>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.</i>
zookeeper	Are Zookeeper components healthy?

* built-ins, not standalone scripts.

If sampling from a head node, a standalone script is in directory:

/cm/local/apps/cmd/scripts/healthchecks/

If sampling from a regular node, a standalone script is in directory:

/cm/images/default-image/cm/local/apps/cmd/scripts/healthchecks/

G.2.2 Parameters For Health Checks

Health checks have the parameters indicated by the left column in the example below:

Example

```
[myheadnode->monitoring->measurable]% show cmsh
Parameter                               Value
-----
Class                                    Internal
Consolidator                             - (cmsh)
Description                               Checks whether cmsh is available, i.e. can we
                                          use cmsh for the default cluster?
Disabled                                  no (cmsh)
Gap                                        0 (cmsh)
Maximal age                               0s (cmsh)
Maximal samples                           4,096 (cmsh)
Name                                       cmsh
Parameter
Producer                                  cmsh
Revision
Type                                       HealthCheck
```

If the value is inherited from the producer, then it is shown in parentheses next to the value. An inherited value can be overwritten by setting it directly for the parameter of a measurable.

The parameters are a subset of the parameters for metrics described in section G.1.7.

G.3 Enummetrics

Table G.3: List Of Enummetrics

Name	Query
DeviceStatus	<p data-bbox="496 365 1353 394">What is the status of the device? Possible values are:</p> <ul data-bbox="539 421 1353 1238" style="list-style-type: none"><li data-bbox="539 421 1353 450">• up<li data-bbox="539 477 1353 506">• down<li data-bbox="539 533 1353 562">• closed<li data-bbox="539 589 1353 618">• installing<li data-bbox="539 645 1353 674">• installer_failed<li data-bbox="539 701 1353 730">• installer_rebooting,<li data-bbox="539 757 1353 786">• installer_callinginit<li data-bbox="539 813 1353 842">• installer_unreachable<li data-bbox="539 869 1353 898">• installer_burning<li data-bbox="539 925 1353 954">• burning<li data-bbox="539 981 1353 1010">• unknown<li data-bbox="539 1037 1353 1066">• opening<li data-bbox="539 1093 1353 1122">• going_down<li data-bbox="539 1149 1353 1178">• pending<li data-bbox="539 1205 1353 1234">• no data

G.4 Actions And Their Parameters

G.4.1 Actions

Table G.4.1: List Of Actions

Name	Description
Drain	Allows no new processes on a compute node from the workload manager. This means that already running jobs are permitted to complete. Usage Tip: Plan for undrain from another node becoming active
Send e-mail to administrators	Sends mail using the mailserver that was set up during server configuration. Default destination is root@localhost. The e-mail address that it is otherwise sent to is specified by the recipient parameter for this action.
Event	Send an event to users with a connected client
ImageUpdate	Update the image on the node
PowerOff	Powers off, hard
PowerOn	Powers on, hard
PowerReset	Power reset, hard
Reboot	Reboot via the system, trying to shut everything down cleanly, and then start up again
killprocess*	Kills processes seen by CMDaemon with the KILL (-9) signal. The PIDs are passed via the CMD_INFO_MESSAGE environmental variable. Syntax: killprocess <PID1[,<PID2>,...]> This action is designed to work with rogueprocess (page 789)
remount*	remounts all defined mounts
testaction*	An action script example for users who would like to create their own scripts. The source has helpful remarks about the environment variables that can be used as well as tips on configuring it generally
Shutdown	Power off via system, trying to shut everything down cleanly
Undrain node	Allow processes to run on the node from the workload manager

* standalone scripts, not built-ins.

If running from a head node, the script is in directory: /cm/local/apps/cmd/scripts/actions/

If running from a regular node, the script is in directory: /cm/images/default-image/cm/local/apps/cmd/scripts/actions/

G.4.2 Parameters For A Monitoring Action

The default monitoring actions are listed in section 12.4.4.

All actions have in common the parameters shown by the left column, illustrated by the example below for the drain action:

Example

```
[myheadnode->monitoring->action]% show drain
Parameter          Value
-----
Action             Drain node from all WLM
Allowed time
Disable            no
Name               Drain
Revision
Run on             Active
Type               DrainAction
```

Out of the full list of default actions, the actions with only the common parameter settings are:

- `Poweron`: Powers off the node
- `PowerOff`: Powers off the node
- `PowerReset`: Hard resets the node
- `Drain`: Drains the node (does not allow new jobs on that node)
- `Undrain`: Undrains the node (allows new jobs on that node)
- `Reboot`: Reboots node via the operating system.
- `Shutdown`: Shuts the node down via the operating system.
- `ImageUpdate`: Updates the node from the software image
- `Event`: Sends an event to users connected with `cmsh` or Bright View

Extra Parameters For Some Actions

The following actions have extra parameters:

Action of the type `ScriptAction`:

- `killprocess`: A script that kills a specified process
- `testaction`: A test script
- `remount`: A script to remount all devices

The extra parameters for an action of type `ScriptAction` are:

- `Arguments`: List of arguments that are taken by the script
- `Node environment`: Does the script run in the node environment?
- `Script`: The script path
- `timeout`: Time within which the script must run before giving up

Action of the type `EmailAction`:

- `Send e-mail to administrators`: Sends an e-mail out, by default to the administrators

The extra parameters for an action of type `EmailAction` are:

- `All administrators`: sends the e-mail to the list of users in the `Administrator e-mail setting` in `partition[base]` mode
- `Info`: the body of the e-mail message
- `Recipients`: a list of recipients

H

Workload Manager Configuration Files Updated By CMDaemon

This appendix lists workload manager configuration files changed by CMDaemon, events causing such change, and the file or property changed.

H.1 Slurm

File/Property	Updates What?	Updated During
/cm/shared/apps/slurm/var/etc/ <Slurm instance name>/slurm.conf	head node	Add/Remove/Update nodes, hostname change
/cm/shared/apps/slurm/var/etc/ <Slurm instance name>/slurmdbd.conf	head node	Add/Remove/Update nodes, hostname change
/cm/shared/apps/slurm/var/etc/ <Slurm instance name>/gres.conf	all nodes	Add/Remove/Update nodes
/cm/shared/apps/slurm/var/etc/ <Slurm instance name>/topology.conf	head node	Add/Remove/Update nodes

H.2 Univa Grid Engine (UGE)

File/Property	Updates What?	Updated During
/cm/shared/apps/uge/var/ <UGE instance name>/common/host_aliases	head node	hostname/domain change, failover
/cm/shared/apps/uge/var/ <UGE instance name>/common/act_qmaster	head node	hostname/domain change, failover

H.3 PBS Pro

File/Property	Updates What?	Updated During
\$PBS_CONF_FILE	head node, software image	hostname/domain change, failover
/cm/local/apps/pbspro-ce/ /var/spool/mom_priv/config	head node	hostname change, failover

The default value of \$PBS_CONF_FILE in Bright Cluster Manager is /cm/local/apps/<pbspro-ce or pbspro>/var/etc/pbs.conf



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@bright90 ~]# module load openldap
[root@bright90 ~]# slappasswd
New password:
Re-enter new password:
SSHAJ/3wy0+IqyAwhh8Q4obL8489CWJlHpLg
```

The input is the plain text password, and the output is the encrypted password. The encrypted password is set as a value for the `rootpw` tag in the `slapd.conf` file on the head node:

```
[root@bright90 ~]# grep ^rootpw /cm/local/apps/openldap/etc/slapd.conf
rootpw SSHAJ/3wy0+IqyAwhh8Q4obL8489CWJlHpLg
```

The password can also be saved in plain text instead of as an SSHA hash generated with `slappasswd`, but this is considered insecure.

After setting the value for `rootpw`, the LDAP server is restarted:

```
[root@bright90 ~]# service ldap restart          #Centos 6
[root@bright90 ~]# 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@bright90 ~]# grep LDAPPass /cm/local/apps/cmd/etc/cmd.conf
LDAPPass = "MysecretldappasswOrd"
```

CMDaemon is then restarted:

```
[root@bright90 ~]# 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@bright90 ~]# ldapsearch -x
# extended LDIF
#
# LDAPv3
# base <dc=cm,dc=cluster> (default) with scope subtree
...
```

- root cn without a password (this should fail):

```
[root@bright90 ~]# 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@bright90 ~]#
```

- root cn with a password (this should work):

```
[root@bright90 ~]# ldapsearch -x -D 'cn=root,dc=cm,dc=cluster' -w Mysecret1dappassw0rd
# extended LDIF
#
# LDAPv3
# base <dc=cm,dc=cluster> (default) with scope subtree
...
```


J

Tokens

This appendix describes authorization tokens available for profiles. Profiles are introduced in Section 6.4:

Table J: List Of Tokens

Service and token name	User can...
Service: CMAuth	
GET_PROFILE_TOKEN	Retrieve list of profiles and profile properties
ADD_PROFILE_TOKEN	Add a new profile
MODIFY_PROFILE_TOKEN	Modify existing user profile
GET_CMSEVICES_TOKEN	Get a list of available CMDaemon services
Service: CMCeph	
GET_CEPH_TOKEN	Get Ceph properties
ADD_CEPH_TOKEN	Add new Ceph configuration
UPDATE_CEPH_TOKEN	Update Ceph configuration
CEPH_KEYS_TOKEN	Manipulate Ceph keys
CEPH_CLUSTER_SERVICES_TOKEN	List Ceph services
CEPH_SET_DECOMMISSION_TOKEN	Decommission Ceph nodes
CEPH_GET_DECOMMISSION_TOKEN	Get decommissioning information
UPDATE_CEPH_STATE_TOKEN	Update Ceph state
GENERIC_CEPH_COMMAND_TOKEN	Run generic Ceph command
Service: CMCert	
ISSUE_CERTIFICATE_TOKEN	Accept certificate request and issue signed certificate
GET_CERTIFICATE_REQUEST_TOKEN	List pending certificate requests
RECREATE_COMPONENT_CERTIFICATE_TOKEN	Recreate component certificate

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
REMOVE_CERTIFICATE_REQUEST_TOKEN	Cancel certificate request
GET_CERTIFICATE_TOKEN	Show certificate information
REVOKE_CERTIFICATE_TOKEN	Revoke a certificate
UNREVOKE_CERTIFICATE_TOKEN	Unrevoke a revoked certificate
GET_AUTOSIGN_TOKEN	Get information on certificate auto-signing CMDaemon's private key
SET_AUTOSIGN_TOKEN	Enable certificate auto-signing with CMDaemon's private key
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_CLOUDERRORS_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	Start a VPN tunnel to a cloud node

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
DETACH_CLOUD_VIRTUAL_NETWORK_INTERFACE	Stop a VPN tunnel to a cloud node
ADD_CLOUD_JOB_DESCRIPTION_TOKEN	Add cloud job description
GET_CLOUD_JOB_DESCRIPTION_TOKEN	Get cloud job description
GET_ALL_CLOUD_JOB_DESCRIPTION_TOKEN	Get all cloud job description
SUBMIT_CLOUD_JOB_DESCRIPTION_TOKEN	Submit cloud job description
UPDATE_CLOUD_JOB_DESCRIPTION_TOKEN	Update cloud job description
Service: CMDevice	
SNMP_STRING_TOKEN	Get ethernet switch SNMP public string
CHASSIS_USER_PASSWORD_TOKEN	Get/set chassis username and password
BMC_USERNAME_PASSWORD_TOKEN	View/set BMC (e.g. HP ilo4, IPMI) username and password
GET_DEVICE_TOKEN	View all device properties
GET_DEVICE_BY_PORT_TOKEN	View list of devices according to the ethernet switch port that they are connected to
ADD_DEVICE_TOKEN	Add a new device
UPDATE_DEVICE_TOKEN	Update device properties
GET_CATEGORY_TOKEN	Get list of categories
ADD_CATEGORY_TOKEN	Create new category
UPDATE_CATEGORY_TOKEN	Update a category property
GET_NODEGROUP_TOKEN	Get list of nodegroups
ADD_NODEGROUP_TOKEN	Add a new nodegroup
UPDATE_NODEGROUP_TOKEN	Update nodegroup properties (e.g. add a new member node)
GET_DEVICE_STATUS_TOKEN	Get device status (e.g. UP as well as status string e.g. restart required)
SET_DEVICE_STATUS_TOKEN	Set device status (only via RPC API calls)
POWER_ON_TOKEN	Power on a device using BMC or PDU power control
POWER_OFF_TOKEN	Power off a device
POWER_CYCLE_TOKEN	Power reset a device
POWER_STATUS_TOKEN	Get power status e.g on or off
POWER_RESULT_TOKEN	Get the result of the previous power command e.g. failed
SHUTDOWN_NODE_TOKEN	Shutdown a remote node managed by CMDaemon
REBOOT_NODE_TOKEN	Reboot a remote a node
FORCE_RECONNECT_TOKEN	Force remote client to reconnect (RPC API)
NODE_IDENTIFY_TOKEN	Identify a node (RPC API, used by node installer)

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
NODE_GET_MOUNTPOINTS_TOKEN	Get list of mountpoints defined for a node
PPING_TOKEN	Run parallel ping
BURN_STATUS_TOKEN	Get burn status
GET_BURN_LOG_TOKEN	Retrieve burn log
GET_SYNC_LOG_TOKEN	Get rsync provisioning log
GET_PORT_BY_MAC_TOKEN	Determine to which switch port a given MAC is connected to.
SYSINFO_COLLECTOR_TOKEN	Get information about a node (executes dmidecode)
UPDATE_CONFIG_TOKEN	Update node configuration
GET_LIVE_UCS_TOKEN	Get UCS live status
SET_LIVE_UCS_TOKEN	Set UCS live
VALIDATE_UCS_TOKEN	Validate UCS configuration
START_KVM_UCS_TOKEN	Start UCS KVM console
GET_UCS_LOG_TOKEN	Retrieve UCS log
CLEAR_UCS_LOG_TOKEN	Clear Cisco UCS log
CONVERT_XML_TO_UCS_TOKEN	Parse and serialize XML UCS configuration so that CMDaemon can use it
GET_EXCLUDE_LIST_TOKEN	Retrieve the various exclude lists
INSTALLER_REBOOT_REQUIRED_TOKEN	Set restart-required flag (typically set by CMDaemon)
ADD_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN	Add a node-installer interaction (Used by CMDaemon)
REMOVE_REMOTE_NODE_INSTALLER_INTERACTION_TOKEN	Remove a node installer interaction
GET_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN	Get list of pending installer interactions
UPDATE_REMOTE_NODE_INSTALLER_INTERACTIONS_TOKEN	Update installer interactions (e.g. confirm full provisioning)
GET_CLUSTAT	Get cluster status (RPC API, Internal)
SET_CLUSTAT	Set cluster status (RPC API, Internal)
Service: CMGui	
GET_CLUSTER_OVERVIEW_TOKEN	Get cluster overview
GET_NODE_OVERVIEW_TOKEN	Get node overview
GET_NETSWITCH_OVERVIEW_TOKEN	Get switch overview
GET_PDU_OVERVIEW_TOKEN	Get PDU overview
GET_NODE_STATUS_TOKEN	Get node status
TOKEN	
GET_CEPH_OVERVIEW_TOKEN	Get Ceph overview
GET_OPENSTACK_OVERVIEW_TOKEN	Get OpenStack overview
GET_OPENSTACK_TENANT_TOKEN	Get OpenStack VM

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
OVERVIEW_TOKEN	overview
Service: CMJob	
GET_JOB_TOKEN	Get list of jobs that are currently running
HOLD_JOB_TOKEN	Place a job on hold
SUSPEND_JOB_TOKEN	Suspend a job
RESUME_JOB_TOKEN	Resume suspended job
RELEASE_JOB_TOKEN	Release a held job
UPDATE_JOB_TOKEN	Update job run-timer parameters
SUBMIT_JOB_TOKEN	Submit a job using JSON
GET_JOBQUEUE_TOKEN	Retrieve list of job queues and properties
UPDATE_JOBQUEUE_TOKEN	Modify job queues
ADD_JOBQUEUE_TOKEN	Add a new job queue
GET_PE_TOKEN	Get list of SGE parallel environments
DRAIN_TOKEN	Drain a node
DRAIN_OVERVIEW_TOKEN	Obtain list of drained nodes
Service: CMMain	
GET_LICENSE_INFO_TOKEN	Retrieve information about BCM license
GET_VERSION_TOKEN	Get CMDaemon version and revision
GET_SERVER_STATUS_TOKEN	Head node status (e.g. ACTIVE, BECOMEACTIVE etc.)
GET_CLUSTER_SETUP_TOKEN	Get cluster configuration
PING_TOKEN	TCP SYN ping managed devices
PCOPY_TOKEN	Copy a specified file in parallel to a list of nodes
READ_FILE_TOKEN	Read a text file. The text file will be serialized as a JSON object
SAVE_FILE_TOKEN	Save a file on a remote node
UPDATE_SELF_TOKEN	Update a category property
CMDAEMON_FAILOVER_TOKEN	Set CMDaemon failover condition achieved
CMDAEMON_QUORUM_TOKEN	Set CMDaemon quorum achieved
GENERIC_CALL_TOKEN	Make a generic call
REPORT_CRITICAL_ERROR_TOKEN	View critical error report
SET_SERVICESTATE_TOKEN	Set the state of a service
GET_SERVICESTATE_TOKEN	Get the state of a service
GET_BACKGROUND_TASKS_TOKEN	Get background tasks
CANCEL_BACKGROUND_TASKS_TOKEN	Cancel background tasks
CALL_EXTENSION_TOKEN	
Service: CMMon	

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
GET_MONCONF_TOKEN	Get monitoring configuration settings
UPDATE_MONCONF_TOKEN	Update monitoring configuration settings
GET_METRIC_TOKEN	Get metric settings
UPDATE_METRIC_TOKEN	Update metric settings
ADD_METRIC_TOKEN	Add metric settings
GET_HEALTHCHECK_TOKEN	Get health check settings
UPDATE_HEALTHCHECK_TOKEN	Update health check settings
ADD_HEALTHCHECK_TOKEN	Add health check settings
GET_THRESHACTION_TOKEN	Get threshold action settings
UPDATE_THRESHACTION_TOKEN	Update threshold action settings
ADD_THRESHACTION_TOKEN	Add threshold action settings
GET_MONITORING_DATA_TOKEN	Get monitoring data settings
SIGNAL_THRESHOLD_EXCEEDED_TOKEN	See if signal threshold exceeded
MONITORING_INTERNAL_TOKEN	Set up internal monitoring
PREJOB_TOKEN	Set up prejob check
FAKE_MONITORING_TOKEN	Set up fake monitoring
ONDEMAND_TOKEN	Sample on demand
ONDEMAND_RESULT_TOKEN	Read sample results on demand.
GET_METRICCLASS_TOKEN	Get metric class settings
Service: CMNet	
GET_NETWORK_TOKEN	Get network settings
ADD_NETWORK_TOKEN	Add network settings
UPDATE_NETWORK_TOKEN	Update network settings
CMOpenStack	
GET_OPENSTACK_TOKEN	Get OpenStack settings
ADD_OPENSTACK_TOKEN	Add OpenStack settings
UPDATE_OPENSTACK_TOKEN	Update OpenStack settings
OPENSTACK_USERNAMES_PASSWORDS_TOKEN	Get/set username and password settings
OPENSTACK_MANAGE_USER_INSTANCES_TOKEN	Get/set user instances
OPENSTACK_VIEW_CONSOLE_LOG_TOKEN	View OpenStack console log
OPENSTACK_TERMINATE_USER_INSTANCES_TOKEN	Terminate OpenStack user instances
OPENSTACK_EXECUTE_SETUP_TOKEN	
OPENSTACK_GET_SETUP_EXECUTION_TOKEN	Execute OpenStack setup
OPENSTACK_REMOVE_SETUP_TOKEN	Not execute OpenStack setup

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
EXECUTION_TOKEN	
GET_OPENSTACK_VNC_URL_TOKEN	Get OpenStack VNC URL
Service: CMPart	
GET_PARTITION_TOKEN	Get partition settings
ADD_PARTITION_TOKEN	Add partition settings
UPDATE_PARTITION_TOKEN	Update partition settings
GET_RACK_TOKEN	Get rack settings
ADD_RACK_TOKEN	Add rack settings
UPDATE_RACK_TOKEN	Update rack settings
GET_SOFTWAREIMAGE_TOKEN	Get softwareimage settings
ADD_SOFTWAREIMAGE_TOKEN	Add softwareimage settings
UPDATE_SOFTWAREIMAGE_TOKEN	Update softwareimage settings
REMOVE_SOFTWAREIMAGE_TOKEN	Remove softwareimage settings
UPDATE_PROVISIONERS_TOKEN	Update provisioners settings
UPDATE_PROVISIONING_NODE_TOKEN	Update provisioning node
CMDAEMON_FAILOVER_STATUS_TOKEN	Obtain status of failover
Service: CMProc	
GET_PROCESS_TOKEN	Retrieve list of processes that are currently running on a device managed by CMDaemon
GET_SHARED_MEM_TOKEN	Get shared memory
GET_SEMAPHORE_TOKEN	Get semaphore
GET_MSGQUEUE_TOKEN	Get message queue status
CLEAN_IPC_TOKEN	Clear IPC state
SEND_SIGNAL_TOKEN	Send signal to a process
START_SHELL_TOKEN	Start SSH session
START_MINICOM_TOKEN	Start minicom serial session
EXEC_COMMAND_TOKEN	Execute a command on a head node
NODE_EXEC_COMMAND_TOKEN	Remotely execute a command on a compute node
NODE_EXEC_COMMAND_MAINTENANCE_TOKEN	Execute a maintenance command (used by CMDaemon)
EXEC_INTERNAL_COMMAND_TOKEN	Execute internal command (defined in the source code, RPC API, internal)
Service: CProv	
GET_FSPART_TOKEN	Get FSPart (internal)
ADD_FSPART_TOKEN	Set FSPart (internal)

...continues

Table J: List Of Tokens...continued

Service and token name	User can...
UPDATE_FSPART_TOKEN	Update FSPart (internal)
REMOVE_FSPART_TOKEN	Remove FSPart (internal)
RUN_PROVISIONINGPROCESSORJOB_TOKEN	Start and run a provisioning job (nodes with a provisioning role)
UPDATE_PROVISIONINGPROCESSORJOB_TOKEN	Update status of running provisioning jobs (CMDaemon)
REQUEST_PROVISIONING_TOKEN	Request provisioning (nodes with a provisioning role)
MANAGE_RSYNC_DAEMON_TOKEN	Manage the rsync process (CMDaemon)
IMAGEUPDATE_TOKEN	Send image changes to nodes
UPDATEPROVISIONERS_TOKEN	Synchronize software images across provisioning systems (requires at least two provisioners)
PROVISIONERS_STATUS_TOKEN	Check status of provisioners e.g. images are in sync
CANCEL_PROVISIONING_REQUEST_TOKEN	Cancel provisioning request
GRAB_IMAGEUPDATE_TOKEN	Grab changes from node to software image and vice versa
Service: CMServ	
GET_OSSERVICE_TOKEN	Get system service information
START_OSSERVICE_TOKEN	Start system services (service foo start)
STOP_OSSERVICE_TOKEN	Stop system services
CALLINIT_OSSERVICE_TOKEN	Call init (useful for the node-installer itself)
RESTART_OSSERVICE_TOKEN	Restart system services
RELOAD_OSSERVICE_TOKEN	Reload system services
RESET_OSSERVICE_TOKEN	Reset system services
Service: CMSession	
GET_SESSION_TOKEN	Retrieve session information
REGISTER_NODE_SESSION_TOKEN	Register new nodes in a special CMDaemon session (node-installer)
END_SESSION_TOKEN	Terminate sessions
HANDLE_EVENT_TOKEN	Handle events
GET_BROADCAST_EVENTS_TOKEN	Receive broadcast events
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

K

Understanding Consolidation

K.1 Introduction

Consolidation is discussed in the sections on using consolidation in the Monitoring chapter (sections 12.4.3 and 12.5.2).

However, it may be confusing to have the concept of consolidation discussed in the same place as the use of consolidation. Also, the algebra that appears in that discussion (page 504) may not appeal to people. There are many who would like an explanation that may be more intuitive, even if it is less rigorous.

Therefore, in this section a more informal and visual approach is taken to explain consolidation.

K.2 What Is Consolidation?

Consolidation is the compression of data, for data values that have been measured over a fixed interval.

The compression is nothing particularly sophisticated. It is carried out by using some simple mathematical functions to the data points: the average, the maximum, or the minimum.

K.3 Raw Data And Consolidation

Suppose raw data is sampled every 2 minutes.

And the raw data values are consolidated every 10 minutes.

A visual representation of the data values available to the system is:

```
          --- time --->
raw:      | | | | | | | | | | | | | | | |
consolidated: |           |           |           |           |
```

Here, every “|” indicates a data point, so that the visual shows 5 times as many raw data points as consolidated data values.

In the preceding visual it makes no sense to use consolidated data since the data values for raw data and consolidated data overlap. I.e., the more accurate raw data values exist for the entire period.

As time passes, the intention is to start dropping old raw data, to save space on the disk.

For example, for the first 20 minutes in the following visual, there are no longer raw data values available:

Example

```
          --- time --->
raw:      | | | | | | | | | | | | | | | |
consolidated: |           |           |           |           |
```

But the consolidated data points for this period are still available to the system.

When the data values are plotted in Bright View graphs, periods without raw data values automatically have consolidated data values used.

So a combination of both data sources is used, which can be visually represented with:

Example

```

          --- time --->
plot:      |          |          | | | | | | | | | | | | | |

```

That behavior holds true for `cmsh` too.

The behavior illustrated in the last visual assumes that the cluster has been UP for long enough that raw data is being dropped.

In this case, “long enough” means at least 7 days.

However, because RLE (Run Length Encoding) is used to compress the sampled monitoring data values on disk, this minimal “long enough” time can be (much) longer than 7 days. It depends on how much the measurable that is being sampled is changing as each sample is taken. If it is not changing, then RLE can compress over a longer time period.

For example, if a node has been up and reachable without issues for 1000 days, then the `ssh2node` health check raw data values would be PASS over that 1000 days. For the period from now to 7 days ago, the raw data values of PASS are kept as they are for now. However, for the period from 7 days ago to 1000 days ago, consolidation on the unchanging raw values means that only two values, namely the PASS value of 1000 days and the PASS value of 7 days ago, need to be retained, in order to have a totally accurate record of what the values were in that period.

On the other hand, the `forks` metric changes very quickly, and thus can do little RLE compression. That makes it a good choice for demonstrating the kind of output that the preceding visuals imply.

K.4 A Demonstration Of The Output

So, as a demonstration, the last 7 days for `forks` are now shown, with the data values in the middle elided:

Example

```

[bright90->device[bright90]]% dumpmonitoringdata -7d now forks
Timestamp                Value                Info
-----
2018/10/17 10:30:00      2.76243 processes/s
2018/10/17 11:30:00      2.52528 processes/s
2018/10/17 12:30:00      2.53972 processes/s
...
2018/10/24 10:42:00      2.66669 processes/s
2018/10/24 10:44:00      2.63333 processes/s
2018/10/24 10:46:00      2.64167 processes/s

```

The first part of the output shows samples listed every hour. These are the consolidated data values.

The last part of the output shows samples listed every 2 minutes. These are the raw data value values.

I.e.: consolidated data values are used beyond a certain time in the past.

If the administrator would like to explore this further, then displaying only consolidation values is possible in `cmsh` by using the `--consolidationinterval` option of the `dumpmonitoringdata` command:

Example

```
[bright90->device[bright90]]% dumpmonitoringdata --consolidationinterval 1h -7d now forks
Timestamp          Value          Info
-----
2019/01/07 11:39:06    2.65704 processes/s
2019/01/07 12:30:00    2.60111 processes/s
2019/01/07 13:30:00    2.58328 processes/s
...
[bright90->device[bright90]]% dumpmonitoringdata --consolidationinterval 1d -7d now forks
Timestamp          Value          Info
-----
2019/01/07 18:09:06    2.59586 processes/s
2019/01/08 13:00:00    2.58953 processes/s
2019/01/09 06:06:06    2.58854 processes/s
[bright90->device[bright90]]% dumpmonitoringdata --consolidationinterval 1w -7d now forks
Timestamp          Value          Info
-----
2019/01/08 11:15:12.194 2.59113 processes/s
```


L

Node Execution Filters And Execution Multiplexers

Node execution filters and execution multiplexers define where data producers are executed on the nodes of a cluster, and what nodes are targeted to obtain the data.

This appendix explains how node execution filters and execution multiplexers work with the help of some explicit basic examples. The aim is to have the cluster administrator understand how they work and how to use them.

The reference cluster in this section is a 5-node cluster, made up of a head node (`bright90`) and 4 regular nodes (`node001..node004`). The commands run in this appendix are carried out during a `cmsh` session that continues on from the point that it left off earlier.

The terms “node execution filters” and “execution multiplexers” are commonly abbreviated to filters and multiplexers in this appendix.

A simple custom data producer script is created and used to explain some of the more-involved concepts of filters and multiplexers more clearly. The custom script is:

```
[root@bright90 ~]# cat /cm/shared/fm.sh
#!/bin/bash

echo $((RANDOM%100))
echo "Sampled on $(hostname) for $CMD_HOSTNAME" >&3
```

The preceding script outputs a random number, along with the host it is being run on (`$(hostname)`), and the host the metric is targeting (`$CMD_HOSTNAME`). The hosts that it is run on can be defined by filters, while the hosts that are targeted by the metric can be defined by multiplexers.

The Term Multiplex: The word “multiplex” can be confusing to system administrators. In electronics, the term multiplex implies that signals are being gathered from various inputs, and going into a main input.

Here the idea is applicable to the signals (samples) from the execution multiplexers (nodes where the samples are). The samples are multiplexed (gathered) from those nodes, to the node (or nodes) where the data producer is executing.

- The execution of the data producer is on the node (or nodes) defined by `nodeexecutionfilter`. The data producer execution nodes are the ones listed using the `nodes` command of `cmsh`.
- The nodes where the samples are obtained from are defined by the `executionmultiplexer` setting. Those multiplexer nodes can have their samples displayed as output using the `samplenow` command (section 12.6.2, page 530) of `cmsh`.

L.1 Data Producers: Default Configuration For Running And Sampling

If there is no configuration defined for the data producer in the filters or multiplexers for that data producer, then each node runs a data producer on itself, and that data producer targets the node that it is running on.

For example, the existing `dmesg` data producer comprises the `dmesg` health check (section G.2.1) and by default has no filter or multiplexer defined for it. If an attempt is made to list any filter or multiplexer for `dmesg`, then by default there is no content under the table headings:

```
[root@bright90 ~]# cmsg
[bright90]% monitoring setup
[bright90->monitoring->setup]% nodeexecutionfilters dmesg; list; ..;..
Type          Name (key)          Filter          Filter operation
-----
[bright90->monitoring->setup]% executionmultiplexers dmesg; list; ..;..
Name (key)
-----
```

Another way of seeing that no such filters or multiplexers have been defined for `dmesg` could be by seeing that none are defined in its submodes:

```
[bright90->monitoring->setup]% show dmesg | grep submode
Execution multiplexer          <0 in submode>
Node execution filters        <0 in submode>
```

Most existing data producers have filters and multiplexers defined. The number of filters and multiplexers set per data producer can conveniently be viewed via list formatting:

Example

```
[bright90->monitoring->setup]% list -f name,nodeexecutionfilters,executionmultiplexer | more
name (key)          nodeexecutionfilters  executionmultiplexer
-----
AggregateNode      <1 in submode>       <1 in submode>
AggregatePDU       <1 in submode>       <1 in submode>
AlertLevel         <1 in submode>       <1 in submode>
CMDaemonState     <0 in submode>       <0 in submode>
Cassandra          <1 in submode>       <0 in submode>
ClusterTotal       <1 in submode>       <0 in submode>
DeviceState        <1 in submode>       <0 in submode>
...
```

L.1.1 Nodes That Data Producers Are Running On By Default—The `nodes` Command

The `nodes` command shows which nodes the data producer runs on. By default, the data producer runs on all nodes, when nothing has been set explicitly, because each node runs the data producer for itself:

```
[bright90->monitoring->setup]% nodes dmesg
node001..node004,bright90
```

L.1.2 Nodes That Data Producers Target By Default—The `sampLenow` Command

Nodes where samples are being obtained at can be seen using the `sampLenow` command for the specified nodes.

Again, by default, each node is a target, because the target is same node that the `dmesg` data producer runs on:

```
[bright90->monitoring->setup]% device samplenow -t node dmesg
Entity      Measurable  Type      Value      Age         Info
-----
node001    dmesg       OS        PASS       0.093s
node002    dmesg       OS        PASS       0.087s
node003    dmesg       OS        PASS       0.088s
node004    dmesg       OS        PASS       0.09s
bright90   dmesg       OS        PASS       0.179s
```

In the outputs to `samplenow` displayed in this appendix, some columns are omitted for the sake of clarity.

The `-t node` option to `samplenow` expands to `-n node001..node004,bright90` for this reference cluster.

L.2 Data Producers: Configuration For Running And Targeting

Filters and multiplexers define which nodes run the data producers, and which nodes are the targets for measurables.

The `fm.sh` script introduced on page 811 can be used to define several custom metrics according to what nodes run the script and what nodes are targeted by the script.

L.2.1 Custom Metrics From The `fm.sh` Custom Script

Custom data producers of type `metric` are created in this section. These data producers comprise custom metrics, which are now set up with varying filtering and multiplexing definitions, to illustrate how the definitions work.

The Metric `all_for_self`

The metric from the data producer `all_for_self` can be set up with no filtering or multiplexing defined, as follows:

```
[bright90->monitoring->setup]% add metric all_for_self
[bright90->monitoring->setup*[all_for_self*]]% set consolidator none
[bright90->monitoring->setup*[all_for_self*]]% set script /cm/shared/fm.sh
[bright90->monitoring->setup*[all_for_self*]]% set class Test
[bright90->monitoring->setup*[all_for_self*]]% commit
```

The value for `class` is mandatory but arbitrary. It is an arbitrary grouping mechanism, which can be useful in Bright View for grouping folders in trees.

Sampling results for the metric `all_for_self`: With no filtering or multiplexing, the metric just runs everywhere by default, with the target of the running metric being itself too. The `Info` field output from `samplenow` shows this behavior in the script output:

```
[bright90->monitoring->setup[all_for_self]]% exit
[bright90->monitoring->setup]% device samplenow -t node all_for_self
Entity      Measurable  Type  Value  Age      Info
-----
node001    all_for_self Test   2      0.08s   Sampled on node001 for node001
node002    all_for_self Test  13     0.084s  Sampled on node002 for node002
node003    all_for_self Test  97     0.08s   Sampled on node003 for node003
node004    all_for_self Test  18     0.084s  Sampled on node004 for node004
bright90   all_for_self Test  63     0.144s  Sampled on bright90 for bright90
```

The Metric some_for_self

The metric some_for_self can be set up with filtering set up for some nodes, and no multiplexing set up, as follows:

```
# on node001,node002 for itself
[bright90->monitoring->setup]% add metric some_for_self
[bright90->monitoring->setup*[some_for_self*]]% set consolidator none
[bright90->monitoring->setup*[some_for_self*]]% set class Test
[bright90->monitoring->setup*[some_for_self*]]% set script /cm/shared/fm.sh
[bright90->...*[some_for_self*]]% nodeexecutionfilters
[bright90->...*[some_for_self*]->nodeexecutionfilters]% add node some_nodes
[bright90->...->nodeexecutionfilters*[some_nodes*]]% set nodes node001 node002
[bright90->...->nodeexecutionfilters*[some_nodes*]]% show
Parameter                               Value
-----
Filter operation                         Include
Name                                     some_nodes
Nodes                                    node001,node002
Revision
Type                                     Node
[bright90->...->nodeexecutionfilters*[some_nodes*]]% commit
```

Sampling results for the metric some_for_self: Only some filtering defined, and no multiplexing defined at all, means that the metric just runs on the filtered nodes, and targets only the nodes defined in filter too:

```
[bright90->...->nodeexecutionfilters[some_nodes]]% ..; ..; ..
[bright90->monitoring->setup]% device samplenow -t node some_for_self
Entity      Measurable   Type  Value  Age    Info
-----
node001     some_for_self Test   88     0.094s Sampled on node001 for node001
node002     some_for_self Test   98     0.075s Sampled on node002 for node002
```

The Metric from_head_for_some_others

The metric from_head_for_some_others can be set up with filtering defined for the head node, and multiplexing defined for some other regular nodes (other than node001 and node002 here), as follows:

```
# on active head for node003,node004
[bright90->monitoring->setup]% add metric from_head_for_some_others
[bright90->monitoring->setup*[from_head_for_some_others*]]% set consolidator none
[bright90->monitoring->setup*[from_head_for_some_others*]]% set script /cm/shared/fm.sh
[bright90->monitoring->setup*[from_head_for_some_others*]]% set class Test
[bright90->...*[from_head_for_some_others*]]% nodeexecutionfilters
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters]% active
Added active resource filter
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters]% show active head node
Parameter                               Value
-----
Filter operation                         Include
Name                                     Active head node
Operator                                  OR
Resources                                 Active
Revision
Type                                     Resource
[bright90->monitoring->setup*[from_head_for_some_others*]->nodeexecutionfilters]% ..
[bright90->...*[from_head_for_some_others*]]% executionmultiplexers
```



```
[bright90->...*[from_head_for_some_others*]->executionmultiplexers]% add node other_nodes
[bright90->..._for_some_others*]->executionmultiplexers*[other_nodes*]]% set nodes node003 node004
[bright90->...*[from_head_for_some_others*]->executionmultiplexers*[other_nodes*]]% show
Parameter                               Value
-----
Filter operation                         Include
Name                                     other_nodes
Nodes                                    node003,node004
Revision
Type                                     Node
[bright90->..._for_some_others*]->executionmultiplexers*[other_nodes*]]% commit; ..; ..;..
[bright90->monitoring->setup]%
```

The filter is given a resource, `Active`, which is a way to set the filter for the active node only. Available resources for a node can be seen by running the command `monitoringresources` for a device:

Example

```
[bright90->monitoring->setup]% device monitoringresources bright90
Active
CentOS7u5
Ethernet
OpenStack Compute
RDO
boot
...
```

Sampling results for the metric `from_head_for_some_others`: With filtering defined for the active head node, and multiplexing defined for those other nodes, it means that the metric targets those other nodes, and the metric runs on the head node. That is, the other nodes are targeted by the head node that is running the metric:

```
[bright90->monitoring->setup]% device samplenow -t node from_head_for_some_others
Entity      Measurable                Type  Value Age      Info
-----
node003     from_head_for_some_others  Test  20    0.084s Sampled on bright90 for node003
node004     from_head_for_some_others  Test  44    0.067s Sampled on bright90 for node004
```

The `nodes` command confirms that the head node, `bright90` is the filtered node, that is, the only node(s) running the metric:

```
[bright90->monitoring->setup]% nodes from_head_for_some_others
bright90
```

L.3 Replacing A Resource With An Explicit Node Specification

Within the filter `Active` head node, associated with the metric `from_head_for_some_others`, the resource object `Active` can be replaced with a node object instead, if the node is defined as being `bright90`. Doing this on a high-availability cluster where there is one active and one passive head node, would be unwise. However, doing this on the reference cluster for teaching purposes is of course absolutely fine because it helps make things a bit more concrete for the reader. The replacement can be carried out for the session as follows:

```
[bright90->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters]% list
Type      Name (key)                Filter                Filter operation
```

```

-----
Resource      Active head node      Active      Include
[bright90->...[from_head_for_some_others]->nodeexecutionfilters]% remove active head node
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters*]% add node head
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% set nodes bright90
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% show
Parameter      Value
-----
Filter operation      Include
Name                  head
Nodes                 bright90
Revision
Type                  Node
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters*[head*]]% commit
[bright90->...[from_head_for_some_others]->nodeexecutionfilters*[head]]% .;.;..

```

The sample results are the same kind of output, and the filter node used is the same. The target sample outputs are the same kind of output as before the replacement. The filter node on which the metric runs is also seen to be the same:

```

[bright90->monitoring->setup]% device samplenow -t node from_head_for_some_others
Entity      Measurable      Type  Value Age      Info
-----
node003    from_head_for_some_others  Test  11    0.081s  Sampled on bright90 for node003
node004    from_head_for_some_others  Test  12    0.07s   Sampled on bright90 for node004
[bright90->monitoring->setup]% nodes
bright90

```

L.4 Excessive Sampling

If another node is appended to the node in the filter, then a warning comes up.

```

[bright90->monitoring->setup]% nodeexecutionfilters from_head_for_some_others
[bright90->...[from_head_for_some_others]->nodeexecutionfilters]% append head nodes node003
[bright90->...*[from_head_for_some_others*]->nodeexecutionfilters*]% commit
===== from_head_for_some_others =====
Field      Message
-----
executionMultiplexers  Warning: Execution filters/multiplexers are set to run
                        on multiple nodes for the same target. This likely
                        means they are badly configured

[bright90->...[from_head_for_some_others]->nodeexecutionfilters]% show head

```

```

Parameter      Value
-----
Filter operation      Include
Name                  head
Nodes                 node003,bright90
Revision
Type                  Node

```

The warning is there because the node execution filters are doing the same thing from different nodes for an execution multiplexer target, and duplicating monitoring execution is typically a waste of resources, and thus typically a mistake.

However, the warnings are merely warnings, and not errors. So Bright Cluster Manager just goes ahead with setting up the filter/multiplex system according to what the administrator has specified. The `nodes` and `samplingnow` commands now show:

```
[bright90->...[from_head_for_some_others]->nodeexecutionfilters]% ..;..
[bright90->monitoring->setup]% device samplenow -t node from_head_for_some_others
Entity      Measurable                Type  Value Age      Info
-----
node003     from_head_for_some_others  Test  3     0.151s  Sampled on bright90 for node003
node003     from_head_for_some_others  Test  76    0.08s   Sampled on node003 for node003
node004     from_head_for_some_others  Test  25    0.08s   Sampled on node003 for node004
node004     from_head_for_some_others  Test  88    0.151s  Sampled on bright90 for node004
[bright90->monitoring->setup]% nodes from_head_for_some_others
node003,bright90
```

The node `node003` is now doing what the head node is, sampling the same targets, which is typically a bad idea. However the behavior is indeed as expected for this particular configuration.

Whether running a particular configuration is actually wise, is up to the administrator—but in any case the filter/multiplex system allows plenty of abuse of this kind.

L.5 Not Just For Nodes

Nodes are what node execution filters and execution multiplexers run on. However, sometimes it is more convenient to execute based on other types.

The possible types can be listed with tab-completion suggestions when adding a node execution filter or an execution multiplier:

```
[bright90->monitoring->setup[dmesg]->nodeexecutionfilters]% add<TAB><TAB>
category lua      node      overlay  resource type
```

L.6 Lua Node Execution Filters

Lua (<https://www.lua.org/>) is a lightweight scripting language embedded into CMDaemon. It allows more advanced node execution filters to be written using a Lua filter file.

Example

```
[bright90->monitoring->setup[<data producer>->nodeexecutionfilters]% add lua lua-filter
[bright90->...nodeexecutionfilters*[lua-filter*]]% set code <Lua filter file name>
```

In the preceding example session, the name `lua-filter` is an arbitrary name, that is added to the object that is associated with the Lua filter file `<Lua filter file name>`.

The `self` Lua table is passed by CMDaemon, and contains the entity for which the filter is evaluated. Only devices have a `self` table that is not `nil`. All other entities have a `self` table that is `nil`.

For example, a filter can be created based on a regex match of the hostname:

Example

```
if self == nil then
  return false
else
  return self.hostname:match("^node[0-9]+") ~= nil
end
```

The Lua filter is evaluated for all nodes:

Example

```
[bright90->monitoring->setup[dmesg]]% nodes
node001..node004
```

Development of Lua filters is best done outside of CMDaemon. Doing so requires the cluster administrator to create node environments to be evaluated by hand:

Example

```
[root@bright90 ~]# cat node001.lua
self = {}
self.hostname = "node001"
self.category = "default"

[root@bright90 ~]# cat bright90.lua
self = {}
self.hostname = "bright90"
```

This is in addition to the original filter script:

Example

```
[root@bright90 ~]# cat filter.lua
if self == nil then
  return false
else
  return self.hostname:match("^node[0-9]+") ~= nil
end
```

Both nodes can then be run through the filter using the lua interpreter:

Example

```
[root@bright90 ~]# lua
Lua 5.1.4 Copyright (C) 1994-2015 Lua.org, PUC-Rio
> dofile('bright82.lua')
> print(dofile('filter.lua'))
false
> dofile('node001.lua')
> print(dofile('filter.lua'))
true
```

The Lua `self` exported by CMDaemon contains the following:

- `self`: The main table object, `nil` for non-devices.
- `self.hostname`: The hostname of the device.
- `self.partition`: The name of the partition to which the device belongs.
- `self.category`: The name of the category to which the compute node belongs.
- `self.nodegroups`: An array with the name of node groups the node belongs to.
- `self.mac`: The MAC address of the device.
- `self.ip`: The IP of the device, only available for non-node devices.
- `self.network`: The IP address of the device, only available for non-node devices.
- `self.interfaces`: The array of interface of a node.

- `self.interfaces[1].name`: The name of the first interface.
- `self.interfaces[1].ip`: The IP address of the first interface.
- `self.interfaces[1].network`: The name of the network of the first interface.
- `self.roles`: The array of roles of a node.
- `self.roles[1].name`: The name of the first role.
- `self.roles[1].type`: The type of the first role.
- `self.status.status`: The status of the device.
- `self.status.user_message`: The user message set for the device.
- `self.status.info_message`: The information message set for the device.
- `self.status.closed`: A boolean marking the node as closed.
- `self.status.restart_required`: A boolean marking the node needs to be rebooted.
- `self.status.healthcheck_failed`: A boolean marking at least one health check has returned FAIL.
- `self.status.healthcheck_unknown`: A boolean marking at least one health check has returned UNKNOWN.
- `self.status.state_flapping`: A boolean marking the node status transitioning often in a short time span
- `self.system.name`: The name of the system.
- `self.system.manufacturer`: The manufacturer of the system.
- `self.system.motherboard.name`: The name of the motherboard.
- `self.system.motherboard.manufacturer`: The manufacturer of the motherboard.
- `self.system.bios.version`: The BIOS version.
- `self.system.bios.vendor`: The BIOS vendor name.
- `self.system.bios.date`: The BIOS date.
- `self.system.os.name`: The OS name.
- `self.system.os.version`: The OS version.
- `self.system.os.flavor`: The OS flavor.

Using the `self` environment, complex filters can easily be created. For example, the following filter can be built to include only the nodes on the IB network, which also have a Slurm client role:

Example

```
[root@bright90 ~]# cat ib-slurm-filter.lua
if self == nil then
  return false
end

on_ib_network = false
```

```
for index, interface in ipairs(self.interface) do
    on_ib_network = on_ib_network or (interface.network == "ibnet")
done

slurm_client = false
for index, role in ipairs(self.roles) do
    slurm_client = slurm_client or (role.name == "SlurmClient")
done

return on_ib_network and slurm_client
```

It is also possible to use external sources, like the file system, to determine the filter for a node.

Example

```
[root@bright90 ~]# cat file-check-filter.lua
if self == nil then
    return false
end

function file_exists(name)
    local f = io.open(name, "r")
    if f ~= nil then
        io.close(f)
        return true
    else
        return false
    end
end

return file_exists(string.format('/opt/filter/%s', self.hostname))
```

It is important to understand that this Lua script is evaluated for all nodes, on the active head node. The Lua script should therefore be fast, and return within a few milliseconds.

M

A Tree View Of cmsh

M.1 Modes

A 3-level tree of the modes in cmsh is:

```
|-- category
|   |-- bmcsettings
|   |-- dellsettings
|   |   |-- firmware
|   |-- fsexports
|   |-- fsmounts
|   |-- gpusettings
|   |-- kernelmodules
|   |-- roles
|   |   |-- advancedsettings
|   |   |-- backupbackends
|   |   |-- commsettings
|   |   |-- configurations
|   |   |-- customizations
|   |   |-- engines
|   |   |-- environments
|   |   |-- genericresources
|   |   |-- imagebackends
|   |   |-- momsettings
|   |   |-- nodecustomizations
|   |   |-- osdconfigurations
|   |   |-- provisioningassociations
|   |   |-- resourceproviders
|   |   |-- settings
|   |   |-- slurmjobqueueaccesslist
|   |   |-- spawner
|   |   |-- storagebackends
|   |   |-- storagedrivers
|   |   |-- volumebackends
|   |-- services
|   |-- staticroutes
|   |-- ucs
|       |-- biossettings
|       |-- fcadaptersettings
|       |-- lsbootdef
|       |-- ntpsettings
|       |-- syslogsettings
|       |-- vicadaptersettings
```

```
|-- ceph
|  |-- pools
|-- cert
|-- cloud
|  |-- extensions
|  |-- images
|  |-- regions
|  |-- types
|  |-- vpcs
|-- cmjob
|  |-- cloudjobs
|  |-- storagenodepolicies
|-- configurationoverlay
|  |-- customizations
|  |-- roles
|     |-- advancedsettings
|     |-- backupbackends
|     |-- commsettings
|     |-- configurations
|     |-- customizations
|     |-- engines
|     |-- entries
|     |-- environments
|     |-- genericresources
|     |-- imagebackends
|     |-- momsettings
|     |-- nodecustomizations
|     |-- osdconfigurations
|     |-- provisioningassociations
|     |-- resourceproviders
|     |-- settings
|     |-- spawner
|     |-- storagebackends
|     |-- storagedrivers
|     |-- volumebackends
|-- device
|  |-- bmcsettings
|  |-- cloudsettings
|  |-- dellsettings
|  |-- fsexports
|  |-- fsmounts
|  |-- gpusettings
|  |-- interfaces
|  |-- kernelmodules
|  |-- roles
|     |-- advancedsettings
|     |-- backupbackends
|     |-- commsettings
|     |-- configurations
|     |-- customizations
|     |-- engines
|     |-- environments
|     |-- genericresources
|     |-- imagebackends
|     |-- momsettings
```



```
| | |-- nodecustomizations
| | |-- osdconfigurations
| | |-- provisioningassociations
| | |-- resourceproviders
| | |-- settings
| | |-- slurmjobqueueaccesslist
| | |-- spawner
| | |-- storagebackends
| | |-- storagedrivers
| | `-- volumebackends
|-- services
|-- snmpsettings
|-- staticroutes
|-- ucs
| `-- virtualnodesettings
|-- edgewise
|-- etcd
|-- fspart
|-- group
|-- hierarchy
| |-- sources
| `-- targets
|-- keyvaluestore
|-- kubernetes
| |-- appgroups
| |-- labelsets
| `-- rolebindings
|-- main
|-- monitoring
| |-- action
| |-- consolidator
| | `-- consolidators
| |-- labeledentity
| |-- measurable
| |-- query
| | `-- drilldown
| |-- setup
| | |-- executionmultiplexers
| | |-- jobmetricsettings
| | |-- nodeexecutionfilters
| | `-- rules
|-- standalone
| `-- trigger
|     `-- expression
|-- network
|-- nodegroup
|-- openstack
| |-- agents
| |-- domains
| |-- endpoints
| |-- flavors
| |-- floatingips
| |-- groups
| |-- hostaggregates
| |-- hypervisors
```

```
| |-- images
| |-- networks
| |-- ports
| |-- projects
| |-- roleassignments
| |-- roles
| |-- routers
| |-- securitygroups
| |-- servers
| |-- services
| |-- settings
| |-- subnets
|   |-- allocationpool
| |-- users
| |-- volumes
| |-- volumesnapshots
| |-- volumetypes
|-- partition
| |-- archos
| |-- bmcsettings
| |-- burnconfigs
| |-- failover
| |-- failovergroups
| |-- resourcepools
| |-- snmpsettings
|-- process
|-- profile
|-- rack
|-- session
|-- softwareimage
| |-- kernelmodules
| |-- selection
|-- task
|-- user
| |-- projectmanager
|-- wlm
| |-- cgroups
| |-- jobqueue
| |-- jobs
| |-- parallelenvironments
| |-- placeholders
```