



NVIDIA DGX SuperPOD

Administration Guide

Featuring NVIDIA DGX A100 Systems

Document History

DU-10263-001

Version	Date	Authors	Description of Change
01	2020-12-07	Adam DeConinck, Adam Tetelman, Craig Tierney, Jeff Weiss, Michael Balint, and Robert Sohigian	Initial release
02	2020-12-30	Adam DeConinck and Robert Sohigian	Management node backup, recovery, and major daemons
03	2023-01-29	Craig Tierney, Gianvito Taneburgo, Pawan Jaitly, Rangam Addepalli, Scott Ellis,	Update for Base Command Manager (BCM) 3.22.11
04	2023-02-08	Rangam Addepalli and Robert Sohigian	Updates for BCM 3.23.01

Contents

1.	NVIDIA DGX SuperPOD Overview	1
1.1	System Design	1
1.2	Management Servers	3
2.	Cluster Management	4
2.1	Concepts	4
2.1.1	Devices	4
2.1.2	Software Images	5
2.1.3	Node Categories	6
2.1.4	Node Groups	7
2.1.5	Roles	7
2.2	Authentication	8
2.2.1	Changing Administrative Passwords	8
2.2.2	ssh Logins	9
2.2.3	Certificates	10
2.2.4	Profiles	11
2.3	Base View GUI	11
2.3.1	Cluster Management GUI Service	11
2.3.2	Navigating the Cluster with Base View	13
2.4	Cluster Management Shell	16
2.4.1	Invoking <code>cmsh</code>	16
2.4.2	Levels, Modes, Help, and Commands Syntax in <code>cmsh</code>	20
2.4.3	Working with Objects	24
2.4.4	Advanced <code>cmsh</code> Features	36
3.	Cluster Management Daemon	49
3.1	Controlling <code>CMDaemon</code>	49
3.2	Configuring <code>CMDaemon</code>	51
3.2.1	<code>CMDaemon</code> Versions	51
3.3	Configuring <code>CMDaemon</code> Logging	51
3.3.1	<code>CMDaemon</code> Logging Configuration Global Debug Mode	52
3.3.2	<code>CMDaemon</code> Subsystem Logging Configuration Debug Mode	52
3.4	Configuration File Modification and the <code>FrozenFile</code> Directive	53
3.5	Configuration File Precedence	54
4.	User Management	55
4.1	Managing Users and Groups with Base View	55
4.2	Managing Users and Groups with <code>cmsh</code>	57
4.2.1	Adding a User	57

4.2.2	Saving the Modified State.....	58
4.2.3	Editing Properties of Users and Groups.....	59
4.2.4	Reverting to the Unmodified State	62
4.2.5	Removing a User.....	63
4.3	LDAP	63
4.4	Tokens and Profiles.....	63
4.4.1	Modifying Profiles	65
4.4.2	Creation of Custom Certificates with Profiles.....	66
4.4.3	Logging the Actions Of <code>CMDaemon</code> Users.....	70
4.4.4	Compute Node LDAP PEM and Key Creation	70
5.	Managing Slurm.....	71
5.1	Introduction	71
5.2	Checking Node Status	72
5.3	Showing Detailed Node Information.....	74
5.4	Draining a Node	74
5.5	Updating Slurm Configuration	75
5.6	Slurm Prolog and Epilog	75
5.6.1	Details of Prolog and Epilog Scripts.....	75
5.6.2	Workload Manager Configuration For Prolog and Epilog Scripts	77
5.7	Listing Slurm Jobs in the Queue.....	77
5.8	Canceling a Slurm Job	78
5.9	Managing the Parameters on a Job.....	78
5.9.1	Additional Resources.....	79
6.	Monitoring Cluster Devices.....	80
6.1	Basic Monitoring Example and Action.....	81
6.1.1	Synopsis Of Basic Monitoring Example	81
6.1.2	Setting Up the Pieces.....	82
6.1.3	Using the Basic Monitoring Example	82
6.2	Monitoring Concepts and Definitions	86
6.2.1	Measurables.....	86
6.2.2	Health Check	90
6.2.3	Trigger	92
6.2.4	Action.....	92
6.2.5	Severity	93
6.2.6	<code>AlertLevel</code>	93
6.2.7	Flapping	94
6.2.8	Data Producer	94
6.2.9	Main Monitoring Interfaces of Base View.....	96
6.3	Monitoring Visualization with Base View.....	97
6.3.1	The Monitoring Window	97

7.	Managing High-Speed Fabrics	100
7.1	Verifying that UFM is Running	101
8.	System Health Checks and Debugging	102
8.1	Collecting Log Files	103
8.1.1	Log Subsystems	103
8.1.2	Increasing Log Verbosity	103
8.1.3	Global Debug Mode	104
8.1.4	LOGPREFIX	104
9.	Provisioning Nodes	105
9.1	Role Setup with <code>cmsh</code>	106
9.2	Role Setup with Base View	107
9.3	Housekeeping	109
9.3.1	Provisioning Node Selection	109
9.3.2	Limiting Provisioning Tasks	109
9.3.3	Provisioning Tasks Deferral and Failure	110
9.3.4	Role Change Notification	110
9.3.5	Role Draining and Undraining Nodes	111
9.3.6	Provisioning Node Update Safeguards	111
10.	Product Security	114
11.	Backups	115
11.1	Cluster Installation Backup	115
11.2	Local Database and Data Backups and Restoration	116
11.2.1	Database Corruption and Repairs	116
11.2.2	Restoring from Local Backup	117
11.2.3	Cloning Databases	117
11.2.4	Cloning Extra Databases	117

1. NVIDIA DGX SuperPOD Overview

The NVIDIA DGX SuperPOD™ is a multi-user system designed to run large AI and HPC applications efficiently. Although a DGX SuperPOD is composed of many different components, it should be thought of as an entity that can manage simultaneous use by many users, provide advanced access controls for queuing, and schedule resources fairly to ensure maximum performance. It also provides the tools for collaboration between users and security controls to protect data and limit interaction between users where necessary. The management tools are designed to treat the multiple components as a single system. For more details about the physical architecture, refer to the NVIDIA DGX SuperPOD Reference Architecture.

This document discusses the range of features and tasks that are supported on the DGX SuperPOD. The constituent elements that make up a DGX SuperPOD, both in hardware and software, support a superset of features compared to the DGX SuperPOD solution. Contact the NVIDIA Technical Account Manager (TAM) if clarification is needed on what functionality is supported by the DGX SuperPOD product.

1.1 System Design

A logical depiction of the DGX SuperPOD is shown in Figure 1.

Figure 1. DGX SuperPOD logical design

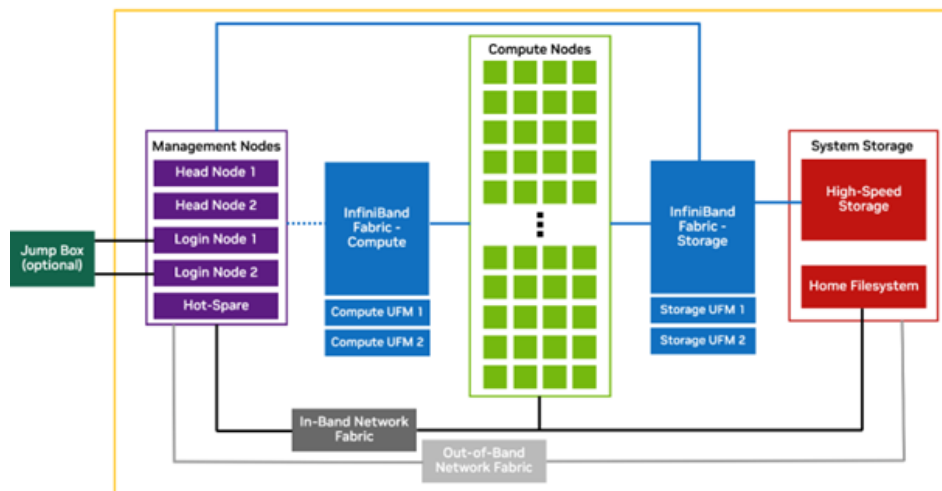


Table 1 describes the components shown in Figure 1.

Table 1. Component descriptions

DGX SuperPOD Component	Description
Jump box/entry point	The Jump Box/Entry Point is the gateway into the DGX SuperPOD intended to provide a single entry-point into the cluster and additional security when required. It is not actually a part of the DGX SuperPOD, but of the corporate IT environment. This function is defined and provided by local IT requirements.
Compute nodes	The compute nodes are where the user work gets done on the system. Each compute node is an individual DGX server
Management nodes	The management nodes provide the services necessary to support operation and monitoring of the DGX SuperPOD. Services, configured in high availability (HA) mode where needed, provide the highest system availability. See Table 2 for details of each node and its function.
High-speed storage	High-speed storage provides shared storage to all nodes in the DGX SuperPOD. This is where datasets, checkpoints, and other large files should be stored. High-speed storage typically holds large datasets that are being actively operated on by the DGX SuperPOD jobs. Data on the high-speed storage is a subset of all data housed in a data lake outside of the DGX SuperPOD.
Shared storage	Shared storage on a network file system (NFS) is allocated for user home directories as well for cluster services.
InfiniBand fabric—compute	The Compute InfiniBand Fabric is the high-speed network fabric connecting all compute nodes together to allow high-bandwidth and low-latency communication between the compute nodes.
InfiniBand fabric—storage	The Storage InfiniBand Fabric is the high-speed network fabric dedicated for storage traffic. Storage traffic is dedicated to its own fabric to remove interference with the node-to-node application traffic that can degrade overall performance.
In-band network fabric	The In-band Network Fabric provides fast Ethernet connectivity between all nodes in the DGX SuperPOD. The In-band fabric is used for TCP/IP-based communication and services.
Out-of-band network fabric	The out-of-band Ethernet network is used for system management using the BMC and provides connectivity to manage all networking equipment.

1.2 Management Servers

Table 2 details the function and services running on the management servers.

Table 2. DGX SuperPOD management servers

Server Function	Services
Head Node	<p>Head nodes serve various functions:</p> <ul style="list-style-type: none">> Provisioning: centrally store and deploy OS images of the compute and other various services. This ensures that there is a single authoritative source defining what should be on each node, and a way to re-provision if the node needs to be reimaged.> Workload management: resource management and orchestration services that organize the resources and coordinate the scheduling of user jobs across the cluster.> Metrics: system monitoring and reporting that gather all telemetry from each of the nodes. The data can be explored and analyzed through web services so better insight to the system can be studied and reported.
Login	<p>Entry point to the DGX SuperPOD for users. CPU only node that is a Slurm client and has filesystems mounted to support development, job submission, job monitoring, and file management. Multiple nodes are included for redundancy and supporting user workloads. These hosts can also be used for container caching.</p>
UFM Appliance	<p>NVIDIA Unified Fabric Manager (UFM) for both storage and compute.</p>

2. Cluster Management

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

2.1 Concepts

In this section, some concepts central to cluster management with the cluster manager are introduced.

2.1.1 Devices

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

- > Head node
- > Physical node
- > Chassis
- > Ethernet switch
- > InfiniBand switch
- > Power Distribution unit
- > Generic device

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

Every device that is managed by the cluster manager has a device state associated with it. Table 3 describes the most important states for devices. All have state tracking enabled.

Table 3. Cluster manager device states

Device status	Device is	Monitored by BCM?
[UP]	UP	monitored
[DOWN]	DOWN	monitored
[CLOSED] (UP)	UP	mostly ignored
[CLOSED] (DOWN)	DOWN	mostly ignored

These and other states are described in more detail in [Section 5.5](#) of the *Bright Cluster Manager Administrator Manual*.

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

2.1.2 Software Images

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

In the DGX SuperPOD, all nodes managed by (meaning all management and DGX nodes) share the same base operating system (OS), with the DGX nodes including the customizations of DGX Base OS.

When a non-compute node boots, the [node provisioning system](#) sets up the node with the software image associated with that node category (2.1.3). Often this is a copy of the default software image, called `default-image`. DGX nodes are provisioned with a copy of the DGX OS image, identified by the `dgx-` prefix in the image name.

After 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](#).

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.

Software images can be changed using regular Linux tools and commands (such as `apt` and `chroot`). More details on making changes to software images and doing image package management can be found [in Chapter 11 of the Bright Cluster Manager Administrator Manual](#).

2.1.3 Node Categories

The collection of settings in the cluster manager that can apply to a node is called the configuration of the node. The administrator usually configures nodes using the [Base View](#) or [cmsh](#) front-end tools, and the configurations are managed internally with a database.

A node category is a group of compute nodes that share the same configuration. Node categories bring efficiency, enabling an administrator to:

- > Configure a large group of nodes concurrently. For example, to set up a group of nodes with a particular disk layout.
- > Operate on a large group of nodes concurrently. For example, to conduct a reboot on an entire category.

The default node categories for BCM installed on a DGX SuperPOD are shown in Table 4.

Table 4. Default node categories

System Type	Node Category	Description
login	slogin	login-image
compute	dgxnodes	dgxos-image
others	default	default-image

The default category can be changed by accessing the base object of [partition mode](#) and setting the value of `defaultcategory` to another, existing, category. System types other than login or compute are automatically placed in the `default` node category.

Nodes are typically divided into categories based upon its hardware specifications or the task that it is to perform. Whether or not nodes should be placed in a separate category depends 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 that 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](#). However, there is no requirement for a one-to-one correspondence between node categories and software images. 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.

2.1.4 Node Groups

A node group consists of nodes that have been grouped together for convenience. The default node group assignments for BCM on a DGX SuperPOD are shown in Table 5.

Table 5. Default node group assignments

Node Group	Members
su1	dgx001..dgx020
su2	dgx021..dgx040
login	slogin1, slogin2

Node groups can consist of any mix of all kinds of nodes, irrespective of whether they are head nodes or compute nodes, and irrespective of what category they are in. A node may be in zero or more node groups at one time. That is, a node may belong to many node groups.

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

One important use for node groups is in the `nodegroups` property of the [provisioning role configuration](#) where a list of node groups that can configure node provisions 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. Once assigned, a role is implicitly assigned to all nodes inside the category.

A [configuration overlay](#) is a group of roles that can be assigned to designated groups of nodes within a cluster. This enables configuration of many 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 those parameters. This can be done by reassigning the role to the individual node with a different set of parameters. Roles that have been thus assigned 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. 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](#) of the *Bright Cluster Manager Administration Manual*.

2.2 Authentication

2.2.1 Changing Administrative Passwords



Note: How to setup or change regular user passwords is discussed in Chapter 4.

The `cm-change-passwd` command is used to administer these:

1. Head node: allows a root login to the head node.
2. Software images: allows a root login to a compute node running with that image and is stored in the image file.
3. Node installer: allows a root login to the node when the node-installer, a stripped-down operating system (OS), is running. The node-installer stage prepares the node for the final OS when the node is booting up. See [Section 5.4](#) of the *Bright Cluster Manager Administrator Manual* for more information about the node-installer.
4. MySQL: allows a root login to the MySQL server used by.

It has a dialog prompting the administrator on which of them, if any, should be changed.

```
[root@headnode ~]# 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 an HA configuration, the passwords are copied over automatically to the other head node when a change is made to the root password of the software image. This allows a root login to a regular node running with that image.

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

Also, for software images passwords used by the compute nodes: the new password that is set for a compute node only works on the node after the image on the node itself has been updated, with, for example, the [imageupdate command](#). Alternatively, the new password can be made to work on the node by 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](#) of the *Bright Cluster Manager Administrator Manual*.

If the administrator has stored the password to the cluster in the Base View front-end, then the password should be modified there too (Figure 2).

2.2.2 ssh Logins

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 (2.2.1).

In contrast, `ssh` logins from the head node to the compute 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 compute node, so that by default a passwordless login works from the head node to the compute nodes, as well as from the compute nodes to the head node.

- > For the `root` user, an `ssh` passwordless login should always work from the head node to the compute nodes since the authorized keys are stored in `/root`. Logins from the compute 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 compute nodes using the [usernodelogin](#) or [User node login](#) settings.
- > On the head node by modifying the `sshd` configuration. For example, to allow only `root` logins, the value of `AllowUsers` can be set in `/etc/ssh/sshd_config` to `root`. See the [sshd config man page](#) for more information.

2.2.3 Certificates

2.2.3.1 PEM Certificates and `CMDaemon` Front-end Authentication

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

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

2.2.3.2 Using `cmsh` and Authenticating to `CMDaemon`

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 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 4.4.2.6.

2.2.3.3 Using Base View and Authenticating to the Cluster Manager

When an administrator uses the Base View front-end, a login to the cluster is conducted with username password authentication (Figure 2) unless the authentication has already been stored in the browser, or unless certificate-based authentication is used.

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

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

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

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

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

Certificate generation in general, including the generation and use of non-administrator certificates, is described in greater detail in 4.4.

2.2.4 Profiles

Certificates that authenticate to `CMDaemon` contain a profile.

A profile determines which cluster management operations that the certificate holder may perform. The administrator certificate is created with the admin profile, which is a built-in profile that enables all cluster management operations to be performed. In this sense, it is like the root account on unix systems. Other certificates may be created with different profiles giving certificate owners access to a predefined subset of the cluster management functionality (4.4).

2.3 Base View GUI

This section introduces the basics of the Base View, which is the web application front-end to the cluster manager.

Base View is supported to run on the latest two publicly available desktop versions of Firefox, Google Chrome, Edge, and Safari at the time of release of BCM.

Browsers that run on mobile devices are not supported.

2.3.1 Cluster Management GUI Service

In the DGX SuperPOD, the GUI interface is provided as a web service on port 8081 from the head node to the browser. Its direct URL takes the form:

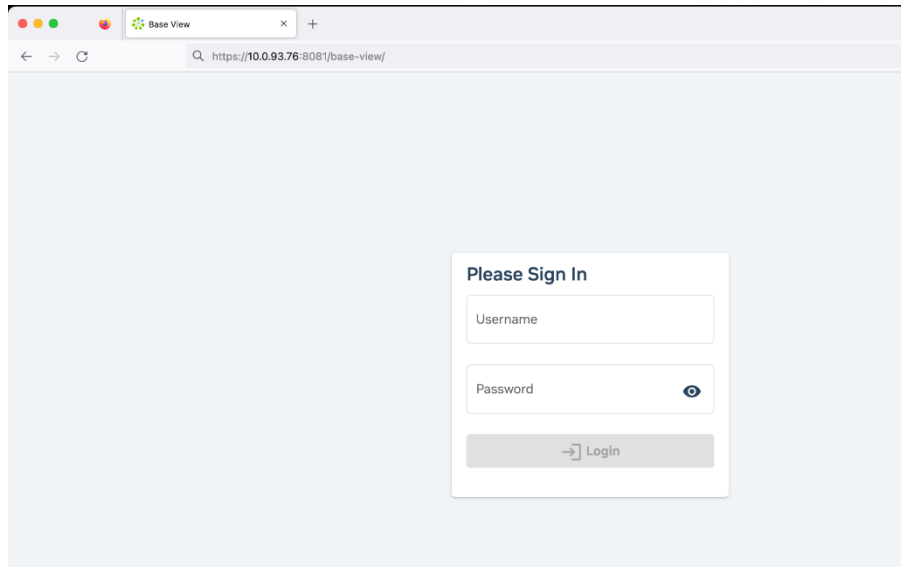
`https://<host name or IP address>:8081/base-view`

The cluster manager package that provides the service is `base-view`.

2.3.1.1 Base View Login Window

Figure 2 shows the login dialog window for Base View. Use this window to administer that Base View service on the cluster. At the time of DGX SuperPOD deployment, at least one login is available: the `root` user, with the password selected during DGX SuperPOD installation (often documented on the Site Survey before installation).

Figure 2. Base View login

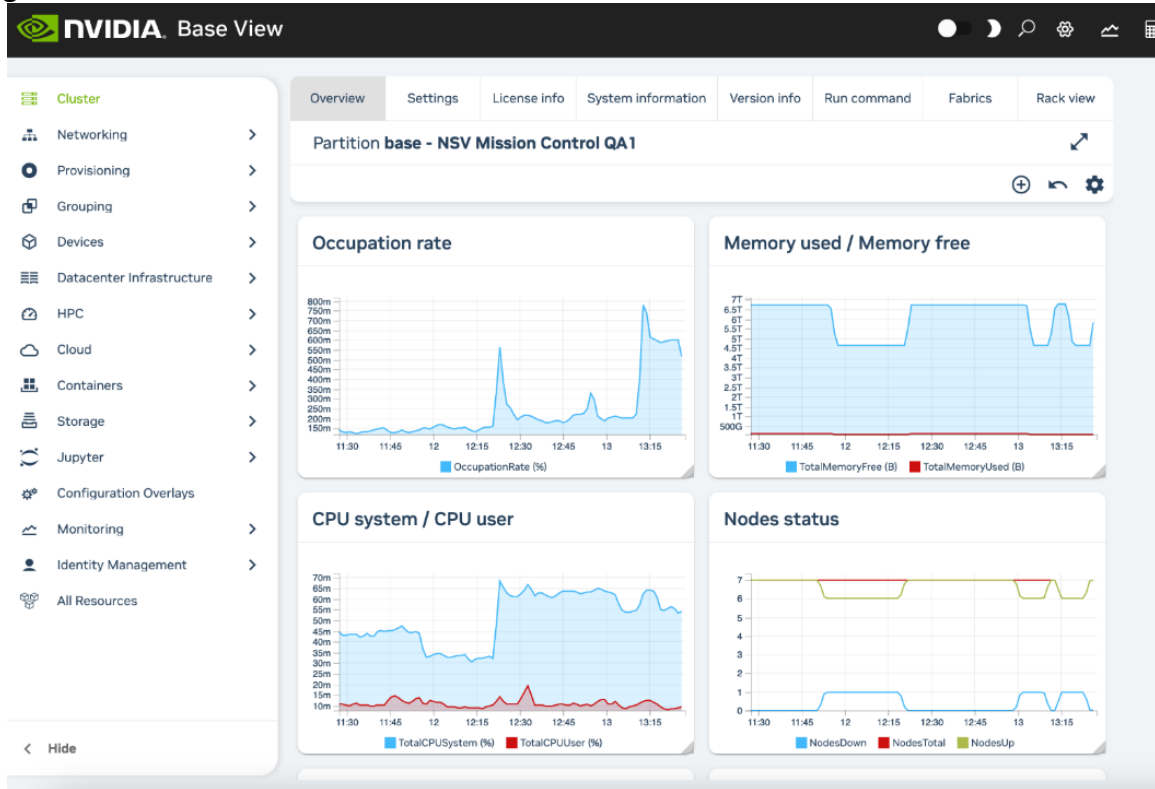


2.3.1.2 Base View Default Display on Connection

By default an overview window is displayed (Figure 3). It shows the [Occupation rate](#), memory used, CPU cycles used, node statuses, and other cluster details.

It corresponds to clickpath Cluster>Partition base.

Figure 3. Cluster overview



2.3.2 Navigating the Cluster with Base View

Aspects of the cluster can be managed by administrators using Base View (Figure 3).

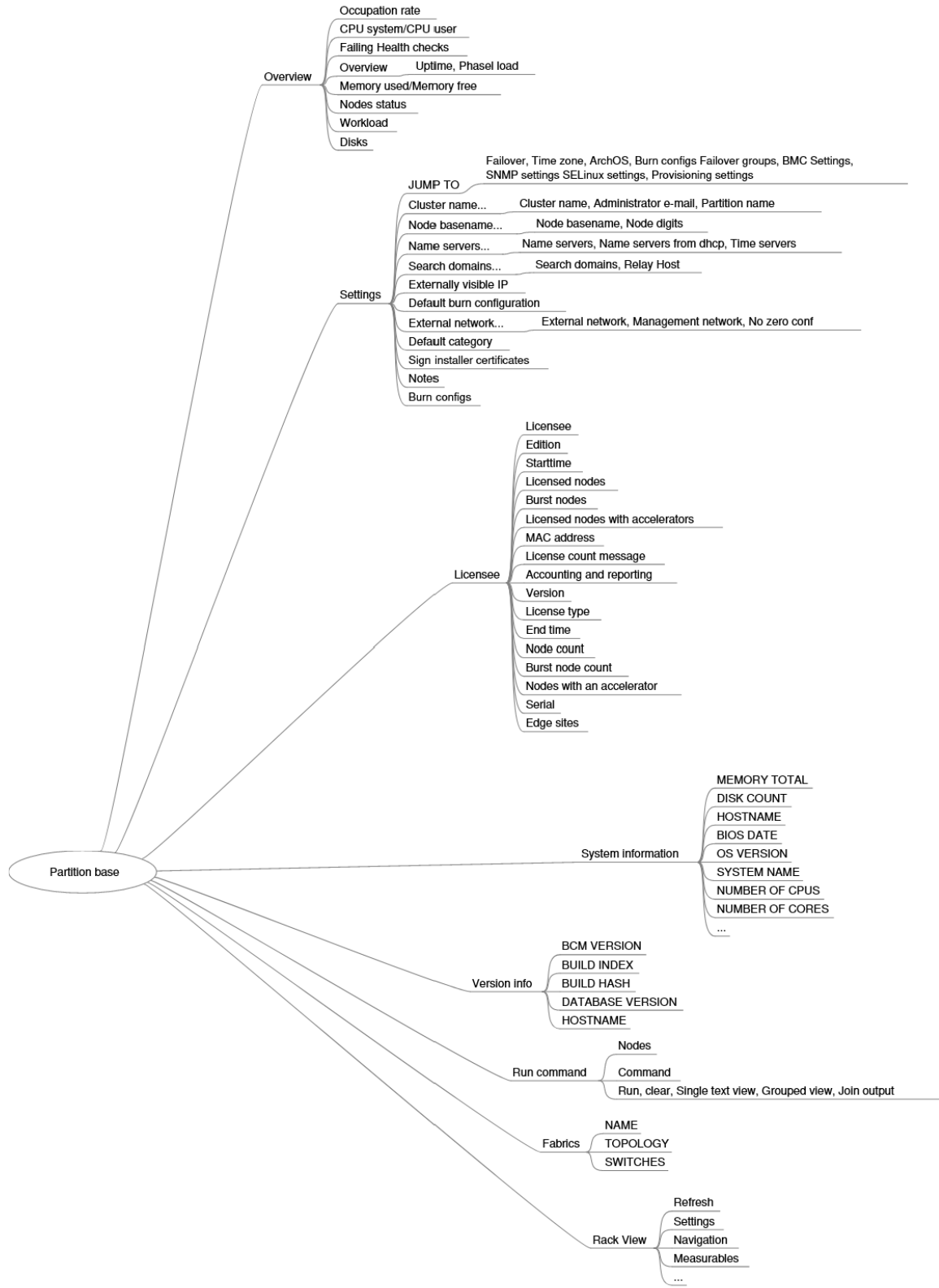
The resource tree, displayed on the left side of the window, consists of available cluster usage concepts such as Provisioning, Grouping, HPC, Cloud, and Containers. It also has a cluster-centric approach to miscellaneous system concepts such as hardware devices Devices, non-hardware resources such as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

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

As an example, the Cluster resource can be selected. This opens the Partition base window, which is a representation of the cluster instance.

The tabs within the Partition base window are mapped out in Figure 4 and described next.

Figure 4. Cluster navigation within the Partition base window



2.3.2.1 Settings

The `Settings` tab has several global cluster properties and property groups. These are loosely grouped as follows:

- > Buttons for jumping to various operational settings.
- > Cluster name, administrator e-mail, partition name.
- > Node basename, node digits.
- > Name servers, time servers.
- > Search domains, relay host.
- > Externally visible IP, `Provisioning node auto update timeout`.
- > Default burn configuration.
- > External network, management network.
- > Default category: sets the default category.
- > Sign installer certificates.
- > Notes.

2.3.2.2 Other Tab Information

Information about other tabs is shown in Table 6.

Table 6. Additional tab information

Tab	Description
System information	Shows the main hardware specifications of the node (CPU, memory, BIOS), along with the OS version that it runs.
Version Information	Shows version information for important cluster software components, such as the <code>CMDaemon</code> database version, and the cluster manager version and builds.
Run Command	Allows a specified command to be run on a selected node of the cluster.
Fabrics	Displays the topology and switches for the fabrics used.
Rack View	Displays a view of the rack as defined by node allocations made by the administrator to racks and chassis.

2.4 Cluster Management Shell

This section introduces the basics of the cluster management shell, `cmsh`. This is the command-line interface (CLI) to cluster management. Because `cmsh` and Base View give access to the same cluster management functionality.

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 for managing the cluster efficiently rather than for trying to be a good or complete programming language. For programming cluster management, [use Python bindings](#) instead of using `cmsh` in batch mode.

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

2.4.1 Invoking `cmsh`

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

```
[root@dgxsuperpod ~]# cmsh
\[dgxsuperpod]%
```

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

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

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

2.4.1.1 Batch Mode and Piping in `cmsh`

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

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

Alternatively, commands can be piped to `cmsh`:

```
[root@dgxsuperpod ~]# echo device status | cmsh
device status
apc01 ..... [  UP  ]
dgxsuperpod ..... [  UP  ]
dgx001 ..... [  UP  ]
dgx002 ..... [  UP  ]
switch01 ..... [  UP  ]
```

2.4.1.2 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:

```
> ~/.cm/cmsh/.cmshrc
> ~/.cm/.cmshrc
> ~/.cmshrc
```

If there is no dotfile for the user and the file `/etc/cmshrc` exists, it is sourced, and its settings used. If `/etc/cmshrc` exists, its settings are used, but the values can be overridden by user dotfiles, which is standard Unix behavior.

2.4.1.3 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`:

```
alias lv device list virtualnode
```

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

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

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

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

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

2.4.1.4 Built-in Aliases in `cmsh`

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

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

The meanings are:

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

2.4.1.5 Automatic Aliases in `cmsh`

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

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

In the following example:

- > The file `tablelist.cmsh` provides the alias `tablelist`, to list devices using the `|` symbol as a delimiter.
- > The file `dfh.cmsh` provides the alias `dfh` to conduct the Linux shell command `df -h`.

```
[root@dgxsuperpod ~]# cat /root/.cm/cmsh/tablelist.cmsh
list -d "|"
[root@dgxsuperpod ~]# cat /root/.cm/cmsh/dfh.cmsh
!df -h
[root@dgxsuperpod ~]# cmsh
[dgxsuperpod]% device
[dgxsuperpod->device]% alias | egrep '(tablelist|dfh)'
alias dfh run /root/.cm/cmsh/dfh.cmsh
alias tablelist run /root/.cm/cmsh/tablelist.cmsh
[dgxsuperpod->device]% list
Type                Hostname (key)    MAC                Category           Ip
-----
HeadNode            dgxsuperpod      FA:16:3E:B4:39:DB
PhysicalNode        dgx001           FA:16:3E:D5:87:71  default            10.141.0.1
PhysicalNode        dgx002           FA:16:3E:BE:05:FE  default            10.141.0.2
[dgxsuperpod->device]% tablelist
Type                |Hostname (key)|MAC                |Category           |Ip
-----
HeadNode            |dgxsuperpod   |FA:16:3E:B4:39:DB |                  |10.141.255.254
PhysicalNode        |dgx001        |FA:16:3E:D5:87:71 |default            |10.141.0.1
PhysicalNode        |dgx002        |FA:16:3E:BE:05:FE |default            |10.141.0.2
[dgxsuperpod->device]% dfh
Filesystem Size  Used Avail Use% Mounted on
devtmpfs   1.8G  0    1.8G   0%   /dev
tmpfs      1.9G  0    1.9G   0%   /dev/shm
tmpfs      1.9G 33M   1.8G   2%   /run
tmpfs      1.9G  0    1.9G   0%   /sys/fs/cgroup
/dev/vdb1  25G  17G   8.7G  66%   /
tmpfs      374M  0    374M   0%   /run/user/0
```

The `cmsh` session does not need restarting for the alias to become active.

2.4.1.6 Default Arguments in `cmsh` Scripts

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

```
[root@dgxsuperpod ~]# cat .cm/cmsh/encrypt-node-disk.cmsh home
device use ${1-dgx001}
set disksetup /root/my-encrypted-node-disk.xml set revision ${2-test}
commit
```

The script can be run without an argument (a blank value for the argument), in which case it takes on the default value of `dgx001` for the parameter:

```
[root@dgxsuperpod ~]# cmsh [dgxsuperpod]% encrypt-node-disk
[dgxsuperpod->device[dgx001]]%
```

The script can be run with an argument (`dgx002` here), in which case it takes on the passed value of `dgx002` for the parameter:

```
[root@dgxsuperpod ~]# cmsh
[dgxsuperpod]% encrypt-node-disk dgx002
[dgxsuperpod->device[dgx002]]%
```

2.4.1.7 `cmsh` Options

The options usage information is shown with `cmsh -h`:

```
Usage:
cmsh [options] [hostname[:port]] cmsh [options] -c <command>
cmsh [options] -f <filename>
Options:
--help|-h
    Display this help
--noconnect|-u
    Start unconnected
--controlflag|-z
    ETX in non-interactive mode
--color <yes/no>
    Define usage of colors
--spool <directory>
    Alternative /var/spool/cmd
--tty|-t
    Pretend a TTY is available
--noredirect|-r
    Do not follow redirects
--norc|-n
    Do not load cmshrc file on start-up
--noquitconfirmation|-Q
    Do not ask for quit confirmation
--echo|-x
    Echo all commands
--quit|-q
    Exit immediately after error
--disablemultiline|-m
```



```

    Disable multiline support
--hide-events
    Hide all events by default
--disable-events
    Disable all events by default
Arguments:
hostname
    The hostname or IP to connect to
command
    A list of cmsh commands to execute
filename
    A file which contains a list of cmsh commands to execute
Examples:
cmsh run in interactive mode
cmsh -c device status run the device status command and exit
cmsh --hide-events -c device status run the device status command and exit, without
showing any events that arrive during this time cmsh -f some.file -q -x    run and echo the
commands from some.file, exit

```

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.

2.4.2 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 using this hierarchy. To perform cluster management functions, the administrator descends through `cmsh` into the appropriate mode and object and conducts actions relevant to the mode or object.

For example, within user mode, an object representing a user instance, `userthree`, might be added or removed. Within the object `userthree`, the administrator can manage its properties. The properties can be data such as a password `password123`, or a home directory `/home/userthree`.

Typing help at the top level of `cmsh` shows the top-level commands.

```

alias ..... Set aliases
category ..... Enter category mode
cert ..... Enter cert mode
cloud ..... Enter cloud mode
cmjob ..... Enter cmjob mode
color ..... Manage console text color settings
configuration overlay ..... Enter configuration overlay mode
connect ..... Connect to cluster
delimiter ..... Display/set delimiter
device ..... Enter device mode

```

```

disconnect ..... Disconnect from cluster
edgesight..... Enter edgesight mode
etcd ..... Enter etcd mode
events ..... Manage events
exit ..... Exit from current object or mode
export ..... Display list of aliases current list formats
fspart ..... Enter fspart mode
group ..... Enter group mode
groupingsyntax ..... Manage the default grouping syntax
help ..... Display this help
hierarchy ..... Enter hierarchy mode
history ..... Display command history
keyvaluestore ..... Enter keyvaluestore mode
kubernetes ..... Enter kubernetes mode
list ..... List state for all modes
main ..... Enter main mode
modified ..... List modified objects
monitoring ..... Enter monitoring mode
network ..... Enter network mode
nodegroup ..... Enter nodegroup mode
partition ..... Enter partition mode
process ..... Enter process mode
profile ..... Enter profile mode
quit ..... Quit shell
quitconfirmation ..... Manage the status of quit confirmation
rack ..... Enter rack mode
refresh ..... Refresh all modes
run ..... Execute cmsh commands from specified file
session ..... Enter session mode
softwareimage ..... Enter softwareimage mode
task ..... Enter task mode
time ..... Measure time of executing command
unalias ..... Unset aliases
unmanagednodeconfiguration .... Enter unmanagednodeconfiguration mode
user ..... Enter user mode
watch ..... Execute a command periodically, showing output
wlm ..... Enter wlm mode

```

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

```

[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:

1. Under the title of `Top`: a list of top-level commands.

2. 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`:

```
[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]%
```

2.4.2.1 Navigation Through Modes and Objects in `cmsh`

The major modes tree is shown in [Appendix M.1](#) of the *Bright Cluster Manager Administrator Manual*.

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

- > To enter a mode, a user enters the mode name at the `cmsh` prompt. The prompt changes to indicate that `cmsh` is in the requested mode, and commands for that mode can then be run.
- > To use an object within a mode, the `use` command is used with the object name. In other words, a mode is entered, and an object within that mode is used. When an object is used, the prompt changes to display that the object within the mode is now being used, and that commands are applied for that object.
- > The `exit` command is used to leave a mode and go back up a level. Similarly, if an object is in use, the `exit` command exits the object. At the top level, `exit` has the same effect as the `quit` command, that is, the user leaves `cmsh` and returns to the unix shell. The string `..` is an alias for `exit`.
- > The `home` command, which is aliased to `/`, takes the user from any mode depth to the top level.
- > The `path` command at any mode depth displays a string that can be used as a path to the current mode and object, in a form that is convenient for copying and pasting into `cmsh`. The string can be used in many ways. For example, an alias can be defined in `.cmshrc` (2.4.1.2).

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

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

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

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

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

```
[headnode]% device
[headnode->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       headnode        00:0C:29:5D:55:46   10.141.255.254 internalnet [ UP ]
PhysicalNode    dgx001         00:0C:29:7A:41:78   default 10.141.0.1   internalnet [ UP ]
PhysicalNode    dgx002         00:0C:29:CC:4F:79   default 10.141.0.2   internalnet [ UP ]
[headnode->device]% exit
[headnode]% device
[headnode->device]% use dgx001
[headnode->device[dgx001]]% path
home;device;use dgx001;
[headnode->device[dgx001]]% home
[headnode]% home;device;use dgx001 #copy-pasted from path output earlier
[headnode->device[dgx001]]%
```

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 semicolons.

A `cmsh` input line has the following syntax:

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

Where `<mode>` and `<arg>` are optional.¹

¹ A more precise synopsis is:

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

```
[headnode->network]% device status headnode; list
headnode ..... [ UP ]
Name (key)      Type      Netmask bits Base address  Domain name      Ipv6
-----
externalnet     External  16           192.168.1.0   userdomain.com   no
globalnet       Global    0            0.0.0.0       cm.cluster
internalnet     Internal  16           10.141.0.0    eth.cluster
[headnode->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 semicolon still runs in network mode, as expected, and not in device mode, and so displays a list of networks.

Inserting a semicolon makes a difference, in that the mode is entered, so that the list displays a list of nodes:

```
[headnode->network]% device; status headnode; list
headnode ..... [ UP ]
Type      Hostname (key) MAC          Category  Ip          Network      Status
-----
HeadNode   headnode      FA:16:3E:C8:06:D1  10.141.255.254 internalnet [ UP ]
PhysicalNode dgx001 F      A:16:3E:A2:9C:87  default   10.141.0.1   internalnet [ UP ]
[headnode->device]%
```

2.4.3 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 the commands exist in every mode, and not all the commands function with an explicit object (Table 7).

Table 7. Command and objects

Command	Description
<code>use</code>	Use the specified object. That is: 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>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 list command
<code>format</code>	Set formatting preferences for list output
<code>foreach</code>	Execute a set of commands on several objects

show	Display all properties of the object
swap	Swap (exchange) the names of two objects
get	Display specified property of the object
set	Set a specified property of the object
clear	Set default value for a specified property of the object.
append	Append a value to a property of the object, for a multi-valued property
removefrom	Remove a value from a specific property of the object, for a multi-valued property
modified	List objects with uncommitted local changes
usedby	List objects that depend on the object
validate	Do a validation check on the properties of the object
exit	Exit from the current object or mode level

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

2.4.3.1 use and exit

```
[dgxsuperpod->device]% use dgx001
[dgxsuperpod->device[dgx001]]% status
dgx001 ..... [ UP      ]
[dgxsuperpod->device[dgx001]]% exit
[dgxsuperpod->device]%
```

In the preceding example, `use dgx001` issued from within `device` mode makes `dgx001` the *current object*. The prompt changes accordingly. The `status` command, without an argument, then returns status information just for `dgx001`, 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.

2.4.3.2 add, commit, and 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 to familiarize the reader with the commands and associated concepts.

```
[dgxsuperpod->device]% add physicalnode dgx100 10.141.0.100 [dgxsuperpod->device*[dgx100*]]%
commit
[dgxsuperpod->device[dgx100]]% category add test-category [dgxsuperpod->category*[test-
category*]]% commit
[dgxsuperpod->category[test-category]]% remove test-category
[dgxsuperpod->category*]% commit
Successfully removed 1 Categories
Successfully committed 0 Categories
[dgxsuperpod->category]% device remove dgx100
[dgxsuperpod->category]% device
[dgxsuperpod->device*]% commit
Successfully removed 1 Devices Successfully committed 0 Devices [dgxsuperpod->device]%
```

`add`: this 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 `dgx100`, is added. For this object, properties such as the type (`physicalnode`) and IP address (`10.141.0.100`) can be set. The node object level (`[dgx100*]`) is automatically dropped into from `device` mode when the `add` command is executed. After execution, the state achieved is that the object has been created with some properties. However, it is still in a temporary, modified state, and not yet persistent.

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

The `add` command—syntax notes:

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

```
[myheadnode->device]% help add
Name:
add - Create a new device of the given type with specified hostname. The IP address may also be
set.
Usage:
add <type> <hostname> [IP address]
Arguments:
type
chassis, genericdevice, gpuunit, litenode, cloudnode, physicalnode, headnode,
powerdistributionunit, racksensor, ethernetswitch, ibswitch, myrinetswitch
```

`commit`: this command is a further step that saves any changes made after executing a command. In this case, in the second line, it saves the `dgx100` object with its properties. The asterisk tag disappears for the prompt if settings for that mode level and below have been saved.

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.

`commit -w|--wait`: 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 causes problems if scripting with `cmsh` for cloning a software image (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`: this 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 conducted yet; it is only a proposed removal. This is indicated by the asterisk tag, which remains visible until the `commit` command is executed, and the test-category removal is saved. The `remove` command can also remove an object in a non-local mode if the non-local mode is associated with the command. This is illustrated in the example where, from within category mode, the device mode is declared before running the `remove` command for `dgx100`. 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 appears 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`.

`remove -d|--data`: 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.

`remove -a|--all`: 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.

2.4.3.3 clone, modified, and swap

The node object `dgx100` that was created in the previous example, can be cloned to `dgx101` as follows:

```
[dgxsuperpod->device]% clone dgx100 dgx101
Warning: The Ethernet switch settings were not cloned, and have to be set manually
[dgxsuperpod->device*[dgx101*]]% exit
[dgxsuperpod->device*]% modified
State  Type                Name
-----
+      Device              dgx101
[dgxsuperpod->device*]% commit
[dgxsuperpod->device]%
[dgxsuperpod->device]% remove dgx100
[dgxsuperpod->device*]% commit
[dgxsuperpod->device]%
```

The `modified` command is used to check what objects have uncommitted changes, and the new object `dgx101` that is seen to be modified, is saved with a `commit`. The device `dgx100` is then removed by using the `remove` command. A `commit` executes the removal.

The `modified` command corresponds to the functionality of the Unsaved entities icon Figure 11.

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

Cloning an object is a convenient method of duplicating a fully configured object. When duplicating a device object, `cmsh` will attempt to automatically assign a new IP address using several heuristics. In the preceding example, `dgx101` is assigned IP address `10.141.0.101`.

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

Sometimes an object may have been misnamed, or physically swapped. For example, `dgx001` exchanged physically with `dgx002` 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 swap their names using 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 must have their names exchanged, it can be done as follows:

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

2.4.3.4 get, set, and refresh

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

```
[dgxsuperpod->device]% use dgx101
[dgxsuperpod->device[dgx101]]% get category test-category
[dgxsuperpod->device[dgx101]]% set category default [dgxsuperpod->device*[dgx101*]]% get
category default
[dgxsuperpod->device*[dgx101*]]% modified
State  Type                               Name
-----
Device                               dgx101
[dgxsuperpod->device*[dgx101*]]% refresh
[dgxsuperpod->device[dgx101]]% modified
No modified objects of type device
[dgxsuperpod->device[dgx101]]% get category test-category
[dgxsuperpod->device[dgx101]]%
```

Here, the category property of the `dgx101` 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 `dgx101` has local uncommitted changes.

The `refresh` command undoes the changes made and corresponds to the Revert button in Base View when viewing Unsaved entities (Figure 11). The `modified` command then confirms that no local changes exist. Finally, the `get` command reconfirms that no local change took place.

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

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

```
[dgxsuperpod->device[dgx101]]% set revision "changed on 10th May"
[dgxsuperpod->device*[dgx101*]]% get revision
[dgxsuperpod->device*[dgx101*]]% 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 2.4.1.

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

2.4.3.5 clear

```
[dgxsuperpod->device]% set dgx101 mac 00:11:22:33:44:55
[dgxsuperpod->device*]% get dgx101 mac
00:11:22:33:44:55
[dgxsuperpod->device*]% clear dgx101 mac
[dgxsuperpod->device*]% get dgx101 mac
00:00:00:00:00:00
[dgxsuperpod->device*]%
```

The `get` and `set` commands are used to view and set the MAC address of `dgx101` without running the `use` command to make `dgx101` the *current object*. The `clear` command then

unsets the value of the property. The result of `clear` depends on the type of the property that 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.

2.4.3.6 list, format, and sort

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

```
[headnode->device]% list -f hostname:14,switchports:15,ip
hostname (key) switchports      ip
-----
apc01                10.142.254.1
headnode             switch01:46  10.142.255.254
dgx001               switch01:47  10.142.0.1
dgx002               switch01:45  10.142.0.2
switch01              10.142.253.1
[headnode->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 displays all available properties including a description of each property:

```
[headnode->device]% format
Current list printing format:
-----
type:22, hostname:[16-32], mac:18, category:[16-32], ip:15, network:[14-32], status:[16-32]
Valid fields:
-----
activation                : Date on which node was defined
additionalhostnames: List of additional hostnames that should resolve to the interfaces IP
address
allownetworkingrestart    : Allow node to update ifcfg files and restart networking
banks                     : Number of banks
...
```

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

```
[headnode->device]% format hostname:10
[headnode->device]% list
hostname (
-----
apc01
headnode
dgx001
dgx002
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:

```
[headnode->device]% format hostname:[10-14]
[headnode->device]% list
hostname (key)
-----
apc01
headnode
dgx001
dgx002
switch01
```

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

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

```
[headnode->device]% format hostname:[10-14],switchports:14,ip:20
[headnode->device]% list
hostname (key) switchports      ip
-----
apc01                10.142.254.1
headnode      switch01:46  10.142.255.254
dgx001      switch01:47  10.142.0.1
dgx002      switch01:45  10.142.0.2
switch01                10.142.253.1
```

The output of the `format` command without arguments shows the current list printing format string, with spaces.

In general, the string used in the `format` command can be set with enclosing quotes ("):

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

Or with the spaces removed:

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

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

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

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

```
[headnode->device]% sort type mac
[headnode->device]% list -f type:15,hostname:15,mac
type          hostname (key)  mac
-----
HeadNode      headnode      08:0A:27:BA:B9:43
PhysicalNode  dgx002      00:00:00:00:00:00
PhysicalNode  log001      52:54:00:DE:E3:6B
[headnode->device]% sort type hostname
[headnode->device]% list -f type:15,hostname:15,mac
type          hostname (key)  mac
-----
HeadNode      headnode      08:0A:27:BA:B9:43
PhysicalNode  log001      52:54:00:DE:E3:6B
PhysicalNode  dgx002      00:00:00:00:00:00
[headnode->device]% sort mac hostname
[headnode->device]% list -f type:15,hostname:15,mac
type          hostname (key)  mac
-----
PhysicalNode  dgx002      00:00:00:00:00:00
HeadNode      headnode      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:

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

2.4.3.7 append and 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 immediately, overwriting the existing list. In the following example, values are appended and removed from the `powerdistributionunits` properties of device `dgx001`.

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.

```
[dgxsuperpod->device]% use dgx001
[dgxsuperpod->device[dgx001]]% get powerdistributionunits
apc01:1
[...device[dgx001]]% append powerdistributionunits apc01:5
[...device*[dgx001*]]% get powerdistributionunits
apc01:1 apc01:5
[...device*[dgx001*]]% append powerdistributionunits apc01:6
[...device*[dgx001*]]% get powerdistributionunits
apc01:1 apc01:5 apc01:6
[...device*[dgx001*]]% removefrom powerdistributionunits apc01:5 [...device*[dgx001*]]% get
powerdistributionunits
apc01:1 apc01:6
[...device*[dgx001*]]% set powerdistributionunits apc01:1 apc 01:02 [...device*[dgx001*]]% get
powerdistributionunits
apc01:1 apc01:2
```

[Chapter 4](#) of the *Bright Cluster Manager Administrator Manual* has more information on power settings and operations.

2.4.3.8 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 `dgx001` and `dgx002` 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 two references to it first must be removed, and the device then must be brought to the [CLOSED state](#) by using the `close` command.

```
[dgxsuperpod->device]% usedby apc01
Device used by the following:
Type          Name      Parameter
-----
Device        apc01      Device is up
Device        dgx001     powerDistributionUnits
Device        dgx002     powerDistributionUnits
[dgxsuperpod->device]%
```

2.4.3.9 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.

```
[dgxsuperpod->device]% use dgx001
[dgxsuperpod->device[dgx001]]% clear category
[dgxsuperpod->device*[dgx001*]]% commit
Code  Field                      Message
-----
1      category                  The category should be set
[dgxsuperpod->device*[dgx001*]]% set category default
[dgxsuperpod->device*[dgx001*]]% validate
All good
[dgxsuperpod->device*[dgx001*]]% commit
[dgxsuperpod->device[dgx001]]%
```

2.4.3.10 show

The `show` command is used to show the parameters and values of a specific object. For example, for the object `dgx001`, the attributes displayed are:

```
[dgxsuperpod->device[dgx001]]% 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>
...
```

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

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

2.4.3.12 import

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.

```
[root@headnode ~]# cmsh
[headnode]% device roles dgx001
[headnode->device[dgx001]->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 `dgx001`, in `slurmclient`, then the role can be imported from a category or overlay. Importing the role duplicates the object and assigns the duplicate value to `dgx001`.

This differs from simply assigning a `slurmclient` role to `dgx001`, 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 `dgx001` level stay at that node level, and changes made to the category-level or overlay-level `slurmclient` role are not automatically inherited by the `dgx001` `slurmclient` role.

```
[headnode->device[dgx001]->roles]% import<TAB><TAB>
cephmds  cloudgateway  elasticsearch
...and other available roles including  slurmclient...
[headnode->device[dgx001]->roles]% import --overlay slurm-client slurmclient
[headnode->device*[dgx001*]->roles*]% list
Name (key)
-----
[category:default] cgroupsupervisor
slurmclient
[headnode->device*[dgx001*]->roles*]% set slurmclient queues dgxlq
[headnode->device*[dgx001*]->roles*]% commit
```


The preceding shows that a list of roles is prompted for using tab-completion after having typed `import`, and that the settings from the configuration overlay level are brought into `dgx001` 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 `dgx001` as the settings at the overlay level. The only change made is that a special queue, `dgx1q`, is configured just for `dgx001`.

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`), a node (`-n|--node`), or an overlay (`-o|--overlay`), as indicated by its help text (`help import`).

2.4.4 Advanced `cmsh` Features

This section describes some advanced features of `cmsh`.

2.4.4.1 CLI Editing

CLI 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 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.

2.4.4.2 `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.

```
[headnode->device[dgx001]]% history | tail -3
use dgx001
history
history | tail -3
[headnode->device[dgx001]]% history -t | tail -3
Thu Dec  3 15:15:18 2015 history
Thu Dec  3 15:15:43 2015 history | tail -3
Thu Dec  3 15:15:49 2015 history -t | tail -3
```

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

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

2.4.4.3 Mixing `cmsh` and Unix Shell Commands

It is often useful for an administrator to be able to execute unix shell commands while conducting 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:

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

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

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

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

2.4.4.4 Output Redirection

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

```
[dgxsuperpod]% device list > devices
[dgxsuperpod]% device status >> devices
[dgxsuperpod]% device list | grep dgx001
```

Type	Hostname (key)	MAC (key)	Category
PhysicalNode	dgx001	00:E0:81:2E:F7:96	default

2.4.4.5 Input Redirection

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

```
[root@dgxsuperpod ~]# cat runthis
device
get dgx001 ip
```

The commands can be run with the redirection operator as:

```
[root@dgxsuperpod ~]# cmsh < runthis
device
get dgx001 ip
10.141.0.1
```

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

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

2.4.4.6 ssh

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.

```
[headnode]% device ssh dgx001
<screen is cleared>
<some MOTD text and login information is displayed>
[root@dgx001 ~]# exit
Connection to dgx001 closed.
[headnode]% device use headnode
[headnode->device[headnode]]% #now let us connect to the head node from the head node object
[headnode->device[headnode]]% ssh
<screen is cleared>
<some MOTD text and login information is displayed>
[root@headnode ~]# exit
logout
Connection to headnode closed.
[headnode->device[headnode]]%
```

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

2.4.4.7 time

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.

```
[headnode->device]% time ds dgx001
dgx001 ..... [  UP  ]
time: 0.108s
```

2.4.4.8 watch

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 two 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`.

```
[headnode->device]% watch newnodes
screen clears
Every 2.0s: newnodes Thu Dec 3 13:01:45 2015
No new nodes currently available.
[headnode->device]% watch -n 3 status -n dgx001,dgx002
screen clears
Every 3.0s: status -n dgx001,dgx002 Thu Jun 30 17:53:21 2016
dgx001 .....[ UP ]
dgx002 .....[ UP ]
```

2.4.4.9 foreach

It is frequently convenient to be able to execute a `cmsh` command on several objects in parallel. The `foreach` command is available in several `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. 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:

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

[dgxsuperpod->device]% foreach dgx001 dgx002 (get hostname; status)
dgx001
dgx001 ..... [ UP ]
dgx002
dgx002 ..... [ UP ]
[dgxsuperpod->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 (2.4.4.12) 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@headnode ~]# cmsh -c "device help foreach"
```

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

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

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

In the device mode of `cmsh`, for example, the `foreach` command has many grouping options available. If objects are 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 (2.4.4.10) can also use the following, more elaborate, syntax:

```
<node>, . . . , <node>, <node>..<node>:
[demo->device]% foreach -c default (status)
dgx001 ..... [ DOWN ]
dgx002 ..... [ DOWN ]
[demo->device]% foreach -g rack8 (status)
...
[demo->device]% foreach -n dgx001,dgx008..dgx016,dgx032 (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.

`clone -o|--clone`: this option allows the cloning (2.4.3.3) of objects in a loop. In the following example, from device mode, `dgx001` is used as the base object from which other nodes from `dgx022` up to `dgx024` are cloned:

```
[headnode->device]% foreach --clone dgx001 -n dgx022..dgx024 () [headnode->device*]% list |
grep node
Type          Hostname (key) Ip
-----
PhysicalNode  dgx001          10.141.0.1
PhysicalNode  dgx022          10.141.0.22
PhysicalNode  dgx023          10.141.0.23
PhysicalNode  dgx024          10.141.0.24
[headnode->device*]% commit
```

The cloned objects are placeholder schematics and settings, with different values for some of the settings, such as IP addresses, decided by heuristics. It is not the software disk image of `dgx001` that is duplicated by object cloning to the other nodes by this action at this time.

`clone -a|--add`: this 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`, and `ibswitch`.

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

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

```
[headnode->device]% foreach -n dgx001..dgx004 --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 `--quitionunknown|-q` option allows the `foreach` loop to exit when an unknown command is detected.

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

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

```
[myheadnode->device]% foreach * (get ip; status)
10.141.253.1
switch01 ..... [ DOWN ]
10.141.255.254
myheadnode ..... [ UP ]
10.141.0.1
dgx001 ..... [ CLOSED ]
10.141.0.2
dgx002 ..... [ CLOSED ]
[myheadnode->device]%
```

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

```
[headnode->category]% foreach * (get name; listnodes)
default
Type           Hostname  MAC                Category  Ip           Network      Status
-----
PhysicalNode   dgx001    FA:16:3E:79:4B:77  default   10.141.0.1   internalnet  [ UP ]
PhysicalNode   dgx002    FA:16:3E:41:9E:A8  default   10.141.0.2   internalnet  [ UP ]
PhysicalNode   dgx003    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 timestamps, which can help in debugging.

2.4.4.10 Node List Syntax

Node list specifications, as used in the `foreach` specification and elsewhere, can be of several types. Here are some examples:

- > **adhoc** (with a comma, or a space):
example: `dgx001,dgx003,dgx005,dgx006`
- > **sequential** (with two dots or square brackets):
example: `dgx001..dgx004`
or equivalently: `dgx00[1-4]`
which is: `dgx001,dgx002,dgx003,dgx004`

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

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

which is:

`dgx001s001,dgx001s002,dgx001s003,dgx001s004,dgx001s005,\
dgx002s001,dgx002s002,dgx002s003,dgx002s004,dgx002s005`

- > 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 three nodes each.

example: `rack[1-2]dgx0[1-3]`

which is: `rack1dgx01,rack1dgx02,rack1dgx03,rack2dgx01,rack2dgx02,rack2dgx03`

Like the previous one, but for nodes that were named more verbosely.

- > sequential exclusion (negation):

example: `dgx001..dgx005,-dgx002..dgx003`

which is: `dgx001,dgx004,dgx005`

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

example: `dgx00[1..7:2]`

which is: `dgx001,dgx003,dgx005,dgx007`

- > mixed list:

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

example: `r1n001..r1n003,r2n003`

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

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

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

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

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

- > path to file that contains a list of nodes:

example: `~/some/filepath/<file with list of nodes>`

The caret sign is a special character in `cmsh` for node list specifications. It indicates the string that follows is a file path that is to be read.

2.4.4.11 groupingsyntax

`groupingsyntax` refers to usage of dots and square brackets. In other words, it is the 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` specifications 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` (2.4.1).

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

2.4.4.12 range

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

```
[headnode->device]% range -n dgx0[01-24]
[headnode->device{-n dgx001..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:

```
[headnode->device{-n dgx001..024}]% range -c default
[headnode->device{-c default}]% ds
dgx001  [  UP ] state flapping
dgx002  [  UP ]
dgx003  [  UP ]
```

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

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


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

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

```
[headnode->device]% range -n node[001-100]
[headnode->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):

```
[root@headnode ~]# 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
...
```

2.4.4.13 bookmark

A bookmark can be:

- > Set with the `bookmark` command.
- > Reached using the `goto` command.

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

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

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

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

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

```
[dgxsuperpod]% device use dgx001
[dgxsuperpod->device[dgx001]]% bookmark
[dgxsuperpod->device[dgx001]]% bookmark -l
Name                Bookmark
-----
-                    home;device;use dgx001;
-                    home;
[dgxsuperpod->device[dgx001]]% home
[dgxsuperpod]% goto
[dgxsuperpod->device[dgx001]]% goto -
[dgxsuperpod]% goto
[dgxsuperpod->device[dgx001]]% bookmark dn1
[dgxsuperpod->device[dgx001]]% goto -
[dgxsuperpod]% goto dn1
[dgxsuperpod->device[dgx001]]%
```

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

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

Bookmarks can be made persistent by setting `.cmshrc` files (2.4.1.2) 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.

```
[rootheadnode ~]# cat .cm/cmsh/.cmshrc
bookmark -x mysaved
```

2.4.4.14 rename

Nodes can be renamed globally from within `partition` mode, in the `Node` `basename` field associated with the prefix of the node in [Base View](#) or in [cmsh](#).

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 conducted as:

```
[headnode->device]% rename dgx001..dgx003 test
Renamed: dgx001 to test1
Renamed: dgx002 to test2
Renamed: dgx003 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 ten nodes are renamed:

```
[headnode->device]% rename node[001-010] test
Renamed: dgx001 to test01
Renamed: dgx002 to test02
...
Renamed: dgx009 to test09
Renamed: dgx010 to test10
```

Then two digits are used for each number suffix, to match the size of the last number.

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

```
[headnode->device]% rename node[001-003] test%04d
Renamed: dgx001 to test0001
Renamed: dgx002 to test0002
Renamed: dgx003 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:

```
[headnode->device]% rename node[001-005] test0[1,2,5-7]
Renamed: dgx001 to test01
Renamed: dgx002 to test02
Renamed: dgx003 to test05
Renamed: dgx004 to test06
Renamed: dgx005 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:

```
[root@headnode ~]# cmsn
[headnode]% device
[headnode->device]% rename node[001-004] userone, usertwo, userthree, userfour
Renamed: dgx001 to userone
Renamed: dgx002 to usertwo
Renamed: dgx003 to userthree
Renamed: dgx004 to userfour
[headnode->device]% commit
```

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

```
[root@headnode ~]# cmsh
[headnode]% device
[headnode->device]% rename node[001-100:2] left[001-050]
Renamed: dgx001 to left001
Renamed: dgx003 to left002
...
Renamed: dgx097 to left049
Renamed: dgx099 to left050
[headnode->device]% rename node[002-100:2] right[001-050]
Renamed: dgx002 to right001
Renamed: dgx004 to right002
...
Renamed: dgx098 to right049
Renamed: dgx100 to right050
[headnode->device]% commit
```

2.4.4.15 Using CMDaemon Environment Variables in Scripts

Within device mode, the `environment` command shows the [CMDaemon environment variables](#) that can be passed to scripts for a particular device.

```
[dgxsuperpod->device]% environment dgx001
```

Key	Value
CMD_ACTIVE_MASTER_IP	10.141.255.254
CMD_CATEGORY	default
CMD_CLUSTERNAME	dgxsuperpod
CMD_DEVICE_TYPE	ComputeNode
CMD_ENVIRONMENT_CACHE_EPOCH_MILLISECONDS	1615465821582
...	

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

```
[dgxsuperpod->device]% environment -e dgx001
export CMD_ENVIRONMENT_CACHE_UPDATES=4
export CMD_CATEGORY=default
export CMD_SOFTWAREIMAGE=default-image
export CMD_DEVICE_TYPE=ComputeNode
export CMD_ROLES=
export CMD_FSMOUNT__SLASH_home_FILESYSTEM=nfs
...
```

2.4.4.16 Converting Tables to JSON with `cms`

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

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

```
[headnode->device]% list -f hostname,ip,mac,status
hostname (key)      ip                mac                status
-----
dgx001              10.141.0.1        FA:16:3E:95:80:9F  [ UP ]
headnode            10.141.255.254    FA:16:3E:D3:56:E0  [ UP ]
[headnode->device]% color off; list -f hostname,ip,mac,status -d
[
  "hostname (key)": "headnode", "ip": "10.141.255.254",
  "mac": "FA:16:3E:D3:56:E0",
  "status": "[ UP ]"
  "hostname (key)": "dgx001",
  "ip": "10.141.0.1",
  "mac": "FA:16:3E:95:80:9F",
  "status": "[ UP ]"
]
```

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

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

3. Cluster Management Daemon

The cluster management daemon or `CMDaemon` is a server process that runs on all nodes of the DGX SuperPOD (including the head node). `CMDaemons` work together to make the cluster manageable. When applications such as `cmsh` and Base View communicate with the cluster, they are interacting with the `CMDaemon` running on the head node. Cluster management applications never communicate directly with `CMDaemons` 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 `CMDaemon` is through TCP port 8081. `CMDaemon` accepts only SSL connections, thereby ensuring all communications are encrypted. Authentication is also managed in the SSL layer using client-side X509v3 certificates (2.2).

On the head node, the `CMDaemon` uses a MySQL database server to store all its internal data. Raw monitoring data, on the other hand, is stored as binary data outside of the [MySQL database](#).

3.1 Controlling `CMDaemon`

It may be useful to shut down or restart `CMDaemon`. For instance, a restart may be necessary to activate changes when the `CMDaemon` configuration file is modified. `CMDaemon` operation can be controlled through the following init script arguments to `service cmd`.

cmdaemonctl command arguments are shown in Table 8.

Table 8. cmdaemonctl command arguments

Argument	Description
stop	Stop the CMDaemon
start	Start the CMDaemon
reload	Reload configuration of the CMDaemon
force-reload	Force reload configuration of the CMDaemon
restart	Restart the CMDaemon
try-restart	Try to restart the CMDaemon, but only if it is running
status report	Whether CMDaemon is running
full-status*	Report detailed statistics about CMDaemon
upgrade*	Update database schema after version upgrade (expert only)
debugon*	Enable debug logging (expert only)
debugoff*	Disable debug logging (expert only)
logconf*	Reload log configuration
* arguments that work with cmdeamonctl as well as with the service command	

Restarting the CMDaemon on the head node of a cluster:

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

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

```
[root@headnode etc]# service cmd status
CMDaemon version 2.1 is running (active) Running locally
Current Time: Fri, 29 Jan 2021 01:48:28 CET
Startup Time: Thu, 28 Jan 2021 15:45:17 CET Uptime: 10h 3m
CPU Usage: 66.8112u 50.5393s (0.3%)
Memory Usage: 172MB
Sessions Since Startup: 29 Active Sessions: 7
Number of occupied worker-threads: 7 Number of free worker-threads: 14
Connections handled: 2397
Requests processed: 6850 Total read: 1.98MB
Total written: 170MB
Average request rate: 11.4requests/m Average bandwidth usage: 4KB/s
```

Restarting the CMDaemon on a sequence of compute nodes dgx001 to dgx040:

```
[root@dgxsuperpod ~]# pdsh -w dgx00[1-9],dgx0[1-3][0-9],dgx040 service cmd restart
```

This uses [pdsh](#), the parallel shell command.

3.2 Configuring CMDaemon

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

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

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

[Appendix C](#) of the *Bright Cluster Manager Administrator Manual* 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 `CMDaemon` must be restarted to activate the changes.

3.2.1 CMDaemon Versions

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.

```
[headnode->device]% cmdaemonversions
Hostname          Version index Version hash
-----
headnode          146,965      e6f593b676
dgx001             146,965      e6f593b676
dgx002             146,965      e6f593b676
```

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

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

```
[headnode->device]% cmdaemonversions --join
Version index Version hash Count      Hostnames
-----
146,965      e6f593b676   3      headnode,dgx001..dgx002
```

3.3 Configuring CMDaemon Logging

`CMDaemon` generates log messages in `/var/log/cmdaemon` from specific internal subsystems, such as workload management, service management, monitoring, and certs. By default, none of those subsystems generate detailed (debug-level) messages, as that would make the log file grow rapidly.

3.3.1 CMDaemon Logging Configuration Global Debug Mode

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

```
[root@headnode ~]# 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@headnode ~]# 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.

3.3.2 CMDaemon Subsystem Logging Configuration Debug Mode

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

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

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

3.3.2.1 CMDaemon Subsystem Logging Configuration Severity Levels

In addition to the `debug` setting, 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:

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

3.3.2.2 CMDaemon Subsystem Logging Configuration Deployment

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

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

Or by reloading the logging configuration:

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

3.4 Configuration File Modification and the FrozenFile Directive

As part of its tasks, the CMDaemon modifies several system configuration files. Some configuration files are completely replaced, while other configuration files only have some sections modified. [Appendix A](#) of the *Bright Cluster Manager Administrator Manual* lists all system configuration files that are modified.

A file that has been generated entirely by the CMDaemon 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 is used to keep it frozen.

Sections of files that have been generated by the CMDaemon 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 in this 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](#), and one of [various possible configuration techniques](#).

3.5 Configuration File Precedence

While the cluster manager changes as little as possible of the standard distributions that it manages, there can sometimes be unavoidable issues. Sometimes a standard distribution utility or service generates a configuration file that conflicts with what the configuration file generated by the cluster manager conducts.

In such a case the configuration file generated by the cluster manager must be given precedence, and the generation of a configuration file from the standard distribution should be avoided. Sometimes using a fully or partially frozen configuration file (3.4) allows a workaround. Otherwise, the functionality of the cluster manager version usually allows the required configuration function to be implemented.

Details on the configuration files installed and updated by the package management system are further discussed in [Appendix A](#) of the *Bright Cluster Manager Administrator Manual*.

4. User Management

Users and groups for the DGX SuperPOD are presented to the administrator in a single system paradigm. That is, if the administrator manages them with the cluster manager, then the changes are automatically shared across the cluster (the single system).

The cluster manager runs its own LDAP service to manage users, rather than using unix user and group files. In other words, users and groups are managed by the centralizing LDAP database server running on the head node, and not through entries in `/etc/passwd` or `/etc/group` files.

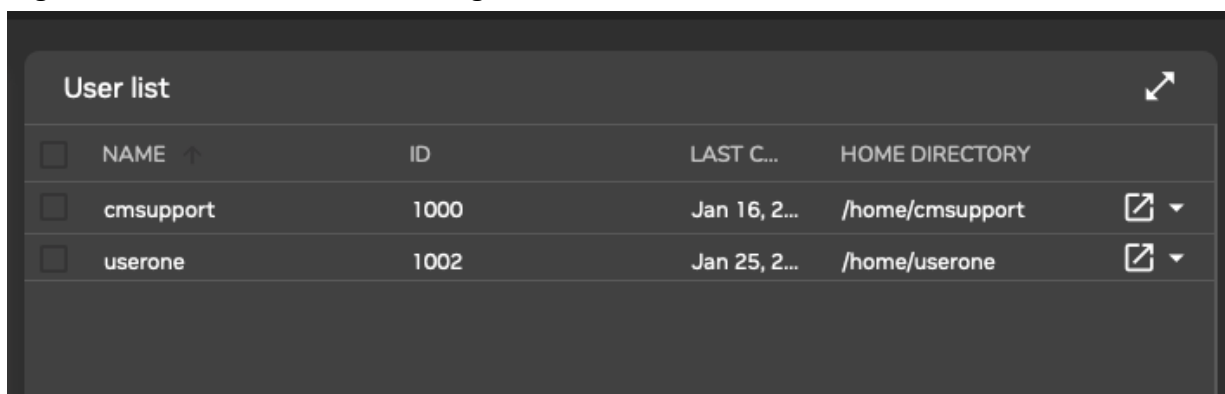
4.1 Managing Users and Groups with Base View

Within Base View:

- > Users can be managed through clickpath Identity Management>Users.
- > Groups can be managed using clickpath Identity Management>Groups.

For users (Figure 5) the LDAP entries for regular users are displayed. These entries are editable, and each user can then be managed in further detail.

Figure 5. Base View User Management



The screenshot shows a web interface titled "User list" with a table of users. The table has columns for NAME, ID, LAST C..., and HOME DIRECTORY. There are two users listed: cmsupport and userone. Each row has a checkbox on the left and an edit icon on the right.

<input type="checkbox"/>	NAME ↑	ID	LAST C...	HOME DIRECTORY	
<input type="checkbox"/>	cmsupport	1000	Jan 16, 2...	/home/cmsupport	▾
<input type="checkbox"/>	userone	1002	Jan 25, 2...	/home/userone	▾

There is one user on a newly installed cluster manager: `cmsupport`. This user has no password set by default, which means no logins to this account are allowed by default. The cluster manager uses `cmsupport` to run various diagnostics utilities, so it should not be removed, and the default contents of its home directory should also not be removed.

The + **ADD** button allows users to be added using a User parameters window (Figure 6). The changes in parameter values can be committed using the **SAVE** button in the User parameter window.

Figure 6. Base View User Management: Add

The screenshot shows a configuration window for a user named 'User 1002'. The window has a dark theme. At the top, there's a title bar 'User 1002' and a search bar 'Search settings inputs'. Below the search bar is a 'JUMP TO' section with a green button 'Project manager >'. The main content area is titled 'Name and 11 others' and contains several input fields: 'Name' (userone), 'ID' (1002), 'Common name' (userone), 'Surname' (userone), 'Group ID' (userone), 'Login shell' (/bin/bash), and 'Home directory' (/home/userone). Each field has a green information icon on the left.

When saving an addition or modification:

- > User and group ID numbers are automatically assigned from `UID` and `GID` 1000 onward.
- > A home directory is created, and a login shell is set. Users with unset passwords cannot log in. Group management in Base View is conducted using clickpath **Identity Management>Groups**.

Clickable LDAP object entries for regular groups then show up, like the user entries already covered. Management of these entries is done with the same functions as for user management.

4.2 Managing Users and Groups with `cmsh`

User management tasks as conducted by Base View can also be done with `cmsh`.

A `cmsh` session is run here to cover the functions corresponding to the user management functions of Base View.



Note: For the remainder of the document a command executed at `#` prompt would be run on the linux shell, and a command run executed on `%` prompt is run inside `cmsh`.

These functions are run from within the `user` mode of `cmsh`.

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

4.2.1 Adding a User

This corresponds to the functionality of the `+` Add button operation in Base View.

In user mode, the process of adding a user `userone` to the LDAP directory is started with the `add` command.

```
[headnode->user% add userone
[headnode->user*[userone*]]%
```

`cmsh` drops into the user object just added, and the prompt shows the username to reflect this. Going into the user object would otherwise be done manually by entering `user userone` at the user mode level.

Asterisks in the prompt are reminders 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.

```
[headnode->user*[userone*]]% modified
State  Type                      Name
-----
+      User                      userone
```

This corresponds to what is displayed by the `Unsaved entities` icon in the top-right corner of the Base View standard display.

Running `show` at this point reveals a username entry, but empty fields for the other properties of `userone`. So, the account in preparation, while it is modified, is not yet ready for use.

```
[headnode->user*[userone*]]% show
Parameter                                Value
-----
Accounts
Managees
Name                                    userone
Primary group
Revision
Secondary
```

4.2.2 Saving the Modified State

This corresponds to the functionality of the `SAVE` button operation in 4.1.

User `userone` that was added in 4.2.1 now exists as a proposed modification but has not yet been committed to the LDAP database.

Running the `commit` command now at the `userone` prompt stores the modified state at the user `userone` object level:

```
[bright92->user*[userone*]]% commit
[bright92->user[userone]]% show
Parameter                                Value
-----
Accounts
Managees
Name                                    userone
Primary group                          userone
Revision
Secondary groups
ID                                      1001
Common name                            userone
Surname                                userone
Group ID                               1001
Login shell                            /bin/bash
Home directory                          /home/userone
Password                               *****
Email
Profile
Create cmjob certificate                no
Write ssh proxy config                  no
Shadow min                              0
Shadow max                              999999
Shadow warning                           7
Inactive                                0
Last change                             2023/1/12
Expiration date                         2038/1/1
Project manager                         <submode>
Notes                                   <0B>
```

If, however, `commit` was to be run at the user mode level without dropping into the `userone` object level, 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. Also, as a security precaution, if an empty field (that is, a “not set”) password entry is committed, then a login to the account is not allowed. So, in the example, the account for user `userone` exists at this stage, but still cannot be logged into until the password is set. Editing passwords and other properties is covered in 4.2.3.

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 path is `/home/<username>` for a user `<username>`. If `homedirectory` is unset, then the default is determined by the `HomeRoot` directive.

4.2.3 Editing Properties of Users and Groups

This corresponds to the functionality of the Edit operation in 4.1.

In 4.2.2, a user account `userone` was made, with an unset password as one of its properties. Logins to accounts with an unset password are refused. The password therefore must be set if the account is to function.

4.2.3.1 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`.

```
[headnode->user[userone]]% set
Name:
set - Set specific user property
Usage:
set [user] <parameter> <value> [<value> ...] (type 1)
set [user] <parameter> [file] (type 2)
Arguments:
User
    name of the user, omit if current is set
Parameters: (type 1)
commonname ..... Full name (e.g. Donald Duck)
createcmjobcertificate Create a certificate with the cloudjob profile for cmjob
email..... email
expirationdate ..... Date on which the user login will be disabled
groupid ..... Base group of this user
homedirectory ..... Home directory
id ..... User ID number
inactive ..... Number of days of inactivity allowed for the user
loginshell ..... Login shell
name ..... User login (e.g. donald)
password ..... Password
```



```

profile ..... Profile for Authorization
projectmanager ..... Project manager
revision ..... Entity revision
shadowmax ..... Maximum number of days for which the user password remains valid.
shadowmin ..... Minimum number of days required between password changes
shadowwarning ..... Number of days of advance warning given to the user before the user
password expires surname .....
Surname (e.g. Duck) writesshproxyconfig . Write ssh proxy config
Parameters: (type 2)
notes ..... Administrator notes
[headnode->user[userone]]%

```

Continuing the session from the end of 4.2.2, the password can be set at the user context prompt like this:

```

[headnode->user[userone]]% set password seteca5tr0n0my
[headnode->user*[userone*]]% commit
[headnode->user[userone]]%

```

The account `userone` is now ready for use.

The converse of the `set` command is the `clear` command, which clears properties.

```

[headnode->user[userone]]% clear password; commit

```

Setting a password in `cmsh` is also possible by setting the LDAP hash (the encrypted storage format) that is generated from the password within `cmsh`. When setting passwords in `cmsh`, a string starting with `{MD5}`, `{CRYPT}`, or `{SSHA}` is the hash of the password.

```

[root@headnode ~]# #first create the LDAP salted SHA-1 hash of the password:
[root@headnode ~]# /cm/local/apps/openldap/sbin/slappasswd -h {SSHA} -s seteca5tr0n0my
[root@headnode ~]# {SSHA}sViD+lfSTtIy0MuGwPGfGd5XKHgEm5d
[root@headnode ~]# cmsh [headnode]% user use userone
[headnode->user[userone]]% set password
enter new password: #here and in the next line {SSHA}sViD+lfSTtIy0MuGwPGfGd5XKHgEm5d is typed
in
retype new password:
[headnode->user[userone]]% commit
[headnode->user[userone]]% !ssh userone@dgx001 #now will test the password that generated
the hash
userone@dgx001 s password: #here seteca5tr0n0my is typed in
Creating ECDSA key for ssh
[userone@node001 ~]$ #successfully logged in with the password associated with the hash

```

Managing passwords in `cmsh` by direct LDAP hash entry should not be done.

4.2.3.2 Editing Groups with `append` and `removefrom`

While the preceding commands `set` and `clear` also work with groups, there are two other commands available which suit the special nature of groups. These supplementary commands are `append` and `removefrom`. They are used to add extra users to and remove extra users from a group.

For example, it may be useful to have a printer group so that several users can share access to a printer. For the sake of this example (continuing the session from where it was left off in the preceding), `usertwo` and `userthree` are now added to the LDAP directory, along with a group `printer`:

```
[headnode->user[userone]]% add usertwo; add userthree
[headnode->user*[userthree*]]% exit; group; add printer
[headnode->group*[printer*]]% commit
[headnode->group[printer]]% exit; exit; user
[headnode->user*]%
```

In the previous example, semicolons are used to chain commands together on the same line.

The context switch that takes place in the preceding session should be noted. The context of user `userone` was eventually replaced by the context of group `printer`. As a result, the group `printer` is committed, but the users `usertwo` and `userthree` are not yet committed, which is shown by the asterisk at the user mode level.

To add users to a group, the `append` command is used. A list of users `userone`, `usertwo`, and `userthree` can be added to the group `printer` like this:

```
[headnode->user*]% commit
Successfully committed 2 Users
[headnode->user]% group use printer
[headnode->group[printer]]% append members userone usertwo userthree; commit [headnode-
>group[printer]]% show
Parameter                Value
-----
ID                        1002
Revision
Name                      printer
Members                   userone,usertwo,userthree
```

To remove users from a group, the `removefrom` command is used. A list of specific users, for example, `usertwo` and `userthree`, can be removed from a group like this:

```
[headnode->group[printer]]% removefrom members usertwo userthree; commit
[headnode->group[printer]]% show
Parameter                Value
-----
ID                        1002
Revision
Name                      printer
Members                   userone
```

The `clear` command can also be used to clear members—but it also clears all the extras from the group:

```
[headnode->group[printer]]% clear members [headnode->group*[printer*]]% show
```

Parameter	Value
ID	1002
Revision	
Name	printer
Members	

The `commit` command is intentionally left out at this point in the session to illustrate how reversion is used in the next section.

4.2.4 Reverting to the Unmodified State

This corresponds roughly to the functionality of the Revert operation in 4.1.

This section continues from the state of the session at the end of 4.2.3. 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 that 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, 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:

```
[headnode->group*[printer*]]% refresh [headnode->group[printer]]% show
```

Parameter	Value
ID	1002
Revision	
Name	printer
Members	userone

Here, the user `userone` reappears because they were 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.

4.2.5 Removing a User

Removing a user using `cmsh` corresponds roughly to the functionality of the Delete operation in 4.1.

The `remove` command removes a user or group. The `-d|--data` flag added to the end of the username removes the user's home directory as well. For example, within user mode, the command `remove user userone -d; commit` removes user `userone`, along with their home directory.

Continuing the session at the end of 4.2.4 from where it was left off, as follows, shows this result:

```
[headnode->group[printer]]% user use userone
[headnode->user[userone]]% remove -d; commit
Successfully removed 1 Users
Successfully committed 0 Users
[headnode->user]% !ls -d /home/* | grep userone      #no userone left behind
[headnode->user]%
```

4.3 LDAP

LDAP services are internal to DGX SuperPOD and provided by head node. If the cluster manager is set to high availability configuration, with LDAP running internally on head nodes, 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 ensures that the active head node propagates its LDAP database changes to the passive node using a `syncprov/syncrepl` configuration in `slapd.conf`.

4.4 Tokens and Profiles

Access to Base View and `cmsh` is based on user certificates.

Tokens can be assigned by the administrator to users so that users can conduct some of the operations that the administrator does with Base View or `cmsh`. Every cluster management operation requires that each user, including the administrator, has the relevant tokens in their profile for the operation. DGX SuperPOD configurations default to having the root user of the head node assigned the admin profile.

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](#) that 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 using clickpath `Identity Management>Profiles`.

Table 9 shows the preconfigured profiles that are available from `cmsh`.

Table 9. Preconfigured profiles in `cmsh`

Profile name	Default Tasks Allowed	nonuser?
admin	all tasks	no
cloudjob	cloud job submission	yes
cmhealth	health-related prejob tasks	yes
cmpam	the cluster manager PAM tasks	yes
litenode	CMDaemon Lite tasks	yes
monitoringpush	pushing raw monitoring data to <code>CMDaemon</code> through a JSON POST (page 404 of the <i>Bright Cluster Manager Developer Manual</i>)	yes
node	node-related tasks, for example by the node-installer	yes
portal	user portal viewing	no
power	device power	yes
readonly	view-only	no

The last column in the preceding table indicates whether the preconfigured profile is a nonuser profile or not. With `cmsh` this can be used to see profiles, with a command such as:

```
[root@headnode ~]# cmsh -c "profile; foreach * (get name; get nonuser)" | paste - -
```

- > Most of the preconfigured profiles are `nonuser` profiles. Such a profile is used by cluster manager clients and should not be modified by the cluster administrator.
- > The preconfigured profiles that are not `nonuser` profiles are `admin`, `readonly`, and `portal`. These can be modified by the cluster administrator and used for human users.

The cluster manager services that use the available preconfigured profiles can be viewed in `cmsh` with the `list` command in `profile` mode.

The tokens, and other properties of a particular profile can be seen within `profile` mode as follows:

```
[headnode->profile]% show readonly
Parameter      Value
-----
Name           readonly
Non user       no
Revision
Services       CMDevice CMNet CMPart CMMon CMJob CMAuth CMServ CMUser CMSession CMMain CMGui CMP+
Tokens         GET_DEVICE_TOKEN GET_CATEGORY_TOKEN GET_NODEGROUP_TOKEN POWER_STATUS_TOKEN GET_DE+
```

For screens that are not wide enough to view the parameter values, the values can also be listed.

```
[headnode->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:

```
[root@headnode ~]# cmsh
[headnode]% user use conner
[headnode->user[conner]]% get profile
[headnode->user[conner]]% set profile readonly; commit
```

Only a subset of the predefined profiles are available to users. The ones that are made available to users are `readonly`, `admin`, and `portal`.

4.4.1 Modifying Profiles

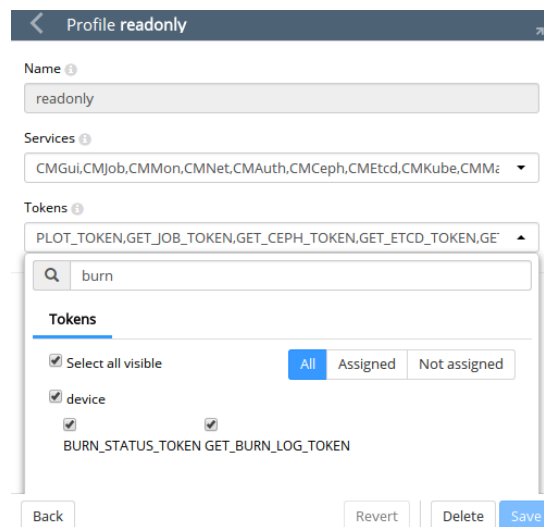
A profile can be modified by adding or removing appropriate tokens to it. For example, the `readonly` group by default has access to the burn status and burn log results. Removing the appropriate tokens stops users in that group from seeing these results.

In `cmsh` the removal can be done from within `profile` mode as follows:

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

Tab-completion after typing in `removefrom` tokens helps in filling in the tokens that can be removed. In Base View (Figure 7), the same removal action can be conducted using clickpath Identity Management>Profiles>readonly>Edit>Tokens.

Figure 7. Base View profile token management



Maximize the window in the resulting display. Run a search for burn to show the relevant tokens, `BURN_STATUS_TOKEN` and `GET_BURN_LOG_TOKEN`, as well as the `device` subgroup they are in. The ticks can be removed from the `BURN_STATUS_TOKEN` and `GET_BURN_LOG_TOKEN` checkboxes, and the changed settings can then be saved.

4.4.2 Creation of Custom Certificates with Profiles

Custom profiles can be created to include a custom collection of capabilities in `cmsh` and Base View. Cloning of profiles is also possible from `cmsh`.

A certificate file, with an associated expiry date, can be created based on a profile. Access to Base View and `cmsh` is based on user certificates. The time of expiry for a certificate cannot be extended after creation. An entirely new certificate is required after expiry of the previous one.

All certificates that have been generated by the cluster are noted by `CMDaemon`.

The creation of custom certificates is described starting in 4.4.2.4. After creating such a certificate, the `openssl` utility can be used to examine its structure and properties. Key values in the following example are 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 (`userfour`), for such a certificate:

```
[root@headnode]# openssl x509 -in userfourfile.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:
        ..userfour
[root@headnode]#
```

However, using the `openssl` utility for managing certificates is inconvenient. The cluster manager provides more convenient ways to do so, as described next.

4.4.2.1 Listing Certificates with `cmsh`

Within the `cert` mode of `cmsh`, the `listcertificates` command lists all cluster certificates and their properties:

```
[root@headnode ~]# cmsh
[headnode]% cert
[headnode-> cert]% listcertificates
```

Serial	Revoked	Time left	Profile	System log in	Name
1	No	5214w 1d	admin	root	Administrator
2	No	5214w 1d	cmhealth		CMHealth
3	No	5214w 1d	cmhealth		CMHealth
4	No	5214w 1d	power		Slurm
5	No	5214w 1d	bootstrap		CertificateRequest
6	No	5214w 1d	cmpam		CMPam
7	No	5214w 1d	portal		WebPortal
...					

4.4.2.2 Listing Certificates with Base View

The Base View equivalent for listing certificates is through clickpath Identity Management>Certificates (Figure 8).

Figure 8. Base View Certificates list window

Certificate list							
<input type="checkbox"/>	NAME	REVOKED	SERIAL NUMBER	REMAINING	PROFILE	COUNTRY	
<input type="checkbox"/>	Administrator	false	1	36d 11h 56m...	admin	US	
<input type="checkbox"/>	CMHealth	false	2	36d 11h 56m...	cmhealth	US	
<input type="checkbox"/>	CMHealth	false	3	36d 11h 56m...	cmhealth	US	
<input type="checkbox"/>	Slurm	false	4	36d 11h 56m...	power	US	
<input type="checkbox"/>	CertificateRequest	false	5	36d 11h 56m...	bootstrap	US	
<input type="checkbox"/>	CMPam	false	6	36d 11h 56m...	cmpam	US	
<input type="checkbox"/>	WebPortal	false	7	36d 11h 56m...	portal	US	
<input type="checkbox"/>	pj-92k.cm.cluster	false	8	36d 11h 56m...	admin	US	
<input type="checkbox"/>	pj-92k	false	9	36d 11h 56m...		US	
<input type="checkbox"/>	fa-16-3e-25-4e-95	false	10	36d 11h 56m...	node	US	
<input type="checkbox"/>	fa-16-3e-b6-69-6e	false	11	36d 11h 56m...	node	US	
<input type="checkbox"/>	fa-16-3e-e3-66-70	false	12	36d 11h 56m...	node	US	
<input type="checkbox"/>	fa-16-3e-0e-89-d5	false	13	36d 11h 56m...	node	US	

4.4.2.3 Node Certificates

In the certificates list, node certificates that are generated by the [node-installer](#) for each node for CMDaemon use are listed. These are entries that look like:

```
[headnode-> cert]% listcertificates
```

Serial	Revoked	Time left	Profile	System log in	Name
-----	-----	-----	-----	-----	-----
...					
10	No	5214w 1d	node		fa-16-3e-74-24-dc
11	No	5214w 1d	node		fa-16-3e-57-2c-8e
12	No	5214w 1d	node		fa-16-3e-b6-c7-4a
13	No	5214w 1d	node		fa-16-3e-bd-cd-05
14	No	5214w 1d	node		fa-16-3e-0d-ab-ea
...					

4.4.2.4 Creating a Custom Certificate

Custom certificates are also listed in the certificates list.

Unlike node certificates, which are normally system-generated, custom certificates are typically generated by a user with the appropriate tokens in their profile, such as root with the admin profile. Such a user can create a certificate containing a specified profile, as discussed in the next section, by using:

- > cmsh: with the `createcertificate` operation from within `cert` mode.
- > Base View: using clickpath Identity Management>Users>Edit>Profile to set the Profile.

4.4.2.5 Creating a New Certificate for cmsh Users

Creating a new certificate in `cmsh` is done from `cert` mode using the `createcertificate` command, which has the following `help` text:

```
[headnode->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 readonly profile set to expire in 30 days, to be run with the privileges of user `userfour`, can be created with:

```
[headnode->cert]% createcertificate 2048 democert a b c d ef readonly userfour 30
/home/userfour /userfourfile.key /home/userfour/userfourfile.pem
Thu Jan  5 15:13:01 2023 [notice] headnode: New certificate request with ID: 16
[headnode->cert]% createcertificate 2048 democert a b c d ef readonly pe er 30 /home/userfour
/userfourfile.key /home/userfour/userfourfile.pem
Certificate key written to file: /home/userfour/userfourfile.key
Certificate pem written to file: /home/userfour/userfourfile.pem
```

The certificate list would show it as something like:

```
[headnode-> cert]% listcertificates
Serial  Revoked  Time left  Profile  System log in  Name
-----
...
23      No      4w 1d     readonly  userfour      democert
```

Setting the ownership of the new custom certificate: The certificates are owned by the owner generating them, so they are root-owned if root was running `cmsh`. This means that user `userfour` cannot use them until ownership is changed to that user.

```
[root@headnode ~]# cd /home/userfour [root@headnode surefour]# ls -l userfourfile.*
-rw----- 1 root root 1704 Aug 22 06:18 userfourfile.key
-rw----- 1 root root 1107 Aug 22 06:18 userfourfile.pem [root@headnode userfour]# chown
userfour:userfour userfourfile.*
```

Other users must have the certificate ownership changed to their own usernames.

4.4.2.6 Associating Users with Paths to a New Custom Certificate

Users associated with such a certificate can then conduct `cmdaemon` tasks that have a `readonly` profile, and `CMDaemon` sees such users as being user `userfour`.

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 `userfour`, for the files generated in the preceding session, `cmsh` can be launched with these keys with:

```
[surefour@head node ~] cmsh -i userfourfile.pem -k userfourfile.key
[headnode]% 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}`

4.4.2.7 Creating a Custom Certificate for Base View Users

As in the case of `cmsh`, a Base View user having a sufficiently privileged tokens profile, such as the `admin` profile, can create a certificate and key file for themselves or another user. This is done by associating a value for the `Profile` from the `Add` or `Edit` dialog for the user (Figure 5).

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 username and password when running Base View use this certificate for their Base View clients and are then restricted to the set of tasks allowed by their associated profile.

4.4.3 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 profiles.

Details on these directives are given in [Appendix C](#) of the *Bright Cluster Manager Administrator Manual*.

4.4.3.1 Creation of Certificates for Nodes with `cm-component-certificate`

The `cm-component-certificate` utility can be used to generate or update SSL certificates for components of services. The cluster administrator is not expected to use this utility because the cluster manager manages the certificates without bothering the administrator about it during normal operations. If the utility is to be used, then it should be used with caution, to avoid failure in the components that use these certificates.

One of the SSL client components for which this utility works is LDAP.

Options include setting a new CA and creating a new certificate or key for nodes.

4.4.4 Compute Node LDAP PEM and Key Creation

If a compute node that was provisioned has a lost or corrupted LDAP key or certificate, then replacements for these can be made with:

```
[root@headnode ~]# cm-component-certificate -n dgx001
Sending request to recreate certificates for 1 node to cmd on headnode
[(38654705666, 1)] 1 0 0
1 certificates were successfully recreated Done.
```

The `ldap.{pem,key}` files are automatically placed on `dgx001`, by default at the location specified by the [`CMDaemon LDAPCertificate` and `LDAPPrivateKey` directives](#).

The files `/cm/node-installer/certificates/<dgx001-mac>/ldap.{pem,key}` should be removed on the head node.

The `nsld`, `sssd`, and `ldap` daemons should be restarted on `dgx001`, or more simply it can be rebooted if it is not in use. The reboot replaces the `ldap.{pem,key}` files on the head node with the newly generated ones.

5. Managing Slurm

5.1 Introduction

Workload management is the submission and control of work on the system. Slurm is the workload management system used on the DGX SuperPOD. It is an open-source job scheduling system for Linux clusters, most frequently used for high-performance computing (HPC) applications. This section will cover some of the basics to get started using Slurm as a user on the DGX SuperPOD. More advanced information about Slurm usage can be found in the [Slurm documentation](#).

The basic flow of a workload management system is the user submits a job to the queue. A job is a collection of work to be executed. What gets submitted is a command, either defined by a shell script or a binary. Shell scripts are the most common because a job often consists of many different commands.

The system will take all the jobs submitted that are not yet running, look at the state of the system, and then map those jobs to the available resources. This workflow allows users to manage their work within large groups and the system will find the optimal way to order the jobs to maximize system utilization or other metrics that can be configured by the system administrators.

Key Slurm terms are detailed in Table 10.

Table 10. Slurm key terms

Term	Definition
job	A unit of work that can be scheduled on the cluster. Each job requests a particular number of compute nodes and may be started by Slurm after the requested number of nodes is reached.
batch job	Submitted to the cluster with a job script, which is an executable such as a bash script. Slurm will wait for the requested number of nodes to be available and will then allocate those nodes to the job and run the script on the first node in the group. The commands in the script are then responsible for running the user workload across the nodes in the allocation. A batch job can be submitted using the <code>sbatch</code> command. When a job is submitted with <code>sbatch</code> , the command will return immediately and place the job in the queue.
interactive job	Submitted to the cluster and requests a pseudo-terminal, so that a user can work on the cluster interactively without having to write a script. Interactive jobs can be submitted by using the <code>srun</code> command with the <code>--pty</code> flag. When you submit a job interactively, the <code>srun</code> command will block until the requested nodes are available, and then provide an interactive terminal to the user.

Slurm controller	The server that is responsible for keeping track of all the servers in the cluster, accepting job submissions, and scheduling work on the cluster. The controller runs a <code>slurmd</code> daemon for managing work on the cluster, and a <code>slurmdbd</code> daemon for keeping track of the job accounting database.
compute node	A compute node, or just node, is an individual server that runs jobs in the cluster. For example, a single DGX A100 system is a Slurm node. Each node runs a <code>slurmd</code> daemon that manages jobs running on that node.
login node	A server that regular users SSH to submit work to the cluster. The login node does not run either a <code>slurmd</code> or <code>slurmd</code> daemon but has the Slurm tools installed so that users can query job information and submit work.
partition	A logical group of compute nodes in Slurm. Each compute node may belong to more than one partition. Jobs are submitted to run in a particular partition and will use nodes from that group.
queue	The list of jobs that are either currently running, or which are waiting to be allocated nodes to run on. If resources are available when a job is submitted, it will run immediately. If there are not sufficient resources to run a job, it will be placed in the queue and wait until resources are available.

5.2 Checking Node Status

Use the `sinfo` command to check the status of all the nodes on the cluster.

```
dgxa100@pg-login-mgmt001:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug      up 1-00:00:00      1 drain dgx049
debug      up 1-00:00:00      1 drain dgx017
debug      up 1-00:00:00      4 alloc dgx[070,081,090,097]
batch*     up 1-00:00:00     96 alloc dgx[001-012,018-048,050-069,091,098-129]
batch*     up 1-00:00:00     38 idle dgx[013-016,071-080,082-089,092-096,130-140]
su01       up 1-00:00:00     15 alloc dgx[001-012,018-020]
su01       up 1-00:00:00      4 idle dgx[013-016]
su02       up 1-00:00:00     20 alloc dgx[021-040]
su03       up 1-00:00:00     19 alloc dgx[041-048,050-060]
su04       up 1-00:00:00      9 alloc dgx[061-069]
su04       up 1-00:00:00     10 idle dgx[071-080]
su05       up 1-00:00:00      4 alloc dgx[091,098-100]
su05       up 1-00:00:00     13 idle dgx[082-089,092-096]
su06       up 1-00:00:00     20 alloc dgx[101-120]
su07       up 1-00:00:00      9 alloc dgx[121-129]
su07       up 1-00:00:00     11 idle dgx[130-140]
```

`sinfo` lists all the partitions in the cluster, and then groups each list of nodes by its status. Status descriptions are in Table 11.

Table 11. `sinfo` status descriptions

Status	Description
idle	Nodes are online, not currently running a job, and available to run a job
alloc	Nodes are online and allocated to a running job
drain	Nodes are online, but they have been marked as “drain” to prevent jobs from running on them. This might be because they failed a health check, or because an administrator manually marked them to drain so the administrator could do maintenance.
drng	Nodes are “draining”: they have been marked to drain, but still have a job running on them. When that job completes, no further jobs will run on them.
down	Nodes are not online and Slurm cannot contact them
boot	Nodes are being rebooted by Slurm

`sinfo` can be restricted to showing information about a particular partition using the `-p` option:

```
dgxa100@pg-login-mgmt001:~$ sinfo -p batch
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*      up 1-00:00:00    32  alloc dgx[098-129]
batch*      up 1-00:00:00   102  idle  dgx[001-016,018-048,050-069,071-080,082-089,091-096,130-140]
```

It can also display only the nodes that are down or drained and show the reason that they are in that state by using the `-R` option:

```
dgxa100@pg-login-mgmt001:~$ sinfo -R
REASON          USER      TIMESTAMP          NODELIST
crashing on boot root      2020-12-04T10:33:15 dgx049
NHC: check_hw_ib: N root      2020-12-04T10:32:42 dgx017
```

The `sinfo -R` command only shows a few characters of the reason for the failure. To see longer details about the reason, run the following command:

```
dgxa100@pg-login-mgmt001:~$ scontrol show node dgx017 | grep Reason
Reason=NHC: check_hw_ib: No IB port mlx5_4:1 is ACTIVE (LinkUp 40 Gb/sec). [root@2020-12-04T10:32:42]
```

5.3 Showing Detailed Node Information

Detailed information about a Slurm node is shown by using the `scontrol` command:

```
dgxa100@pg-login-mgmt001:~$ scontrol show node dgx017
NodeName=dgx017 Arch=x86_64 CoresPerSocket=64
  CPUAlloc=0 CPUTot=256 CPULoad=1.78
  AvailableFeatures=(null)
  ActiveFeatures=(null)
  Gres=gpu:8(S:0-1)
NodeAddr=dgx017 NodeHostName=dgx017 Version=20.02.4
OS=Linux 5.4.0-54-generic #60-Ubuntu SMP Fri Nov 6 10:37:59 UTC 2020
RealMemory=1031000 AllocMem=0 FreeMem=1017487 Sockets=2 Boards=1
State=DOWN+DRAIN ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=debug
BootTime=2020-12-05T19:26:50 SlurmdStartTime=2020-12-05T19:30:32
CfgTRES=cpu=256,mem=1031000M,billing=256
AllocTRES=
CapWatts=n/a
CurrentWatts=0 AveWatts=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
Reason=NHC: check_hw_ib: No IB port mlx5_4:1 is ACTIVE (LinkUp 40 Gb/sec). [root@2020-12-04T10:32:42]
```

5.4 Draining a Node

Draining a node will prevent any further jobs from running on a node. A node should be drained if it is unhealthy or for maintenance work that requires jobs not to be running.

Nodes can be drained by Slurm; by NHC; or manually by an administrator.

To manually drain a node:

```
[headnode01->device]% use dgx-001
[headnode01->device[dgx-001]]% drain
```

Engine	Node	Status	Reason
slurm-dgxsuperpod	dgx-001	Drained	Drained by CMDaemon

To un-drain a node that you want to run jobs on again:

```
[headnode01->device[dgx-001]]% undrain
```

Engine	Node	Status	Reason
slurm-dgxsuperpod	dgx-001		

5.5 Updating Slurm Configuration

Slurm is configured using `cm-wlm-setup`. This will set up `slurm.conf` file in `/cm/shared/apps/slurm/var/etc/slurm`. The file is shared to every node in the cluster and will have the same contents on every node.

Most of the options of `slurm.conf` are outside the scope of this document. To read about the available options for configuring Slurm, see the [slurm.conf documentation](#).



Note: Since Slurm is being managed by BCM, any changes manually made to this file will be overwritten. Changes should be made using Base View or `cmsh`.

5.6 Slurm Prolog and Epilog

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 manage the job execution.
- > Cleaning up resources after job execution has ended.

Although there are global prolog and epilog scripts, editing them should be avoided. The scripts cannot be set using Base View or `cmsh`. Instead, the scripts must be placed by the administrator in the software image, and the relevant nodes updated from the image.

5.6.1 Details of Prolog and Epilog Scripts

Even though it is not recommended, some administrators may nonetheless want to link and edit the scripts directly for their own needs, outside of the Base View or `cmsh` front-ends. A more detailed explanation of how the prolog scripts work follows.

When a Slurm is configured using `cm-wlm-setup` or the Base View setup wizard, it is configured to run the generic prolog located in `/cm/local/apps/cmd/scripts/prolog`, and the generic epilog located in `/cm/local/apps/cmd/scripts/epilog`. The generic prolog and epilog scripts call a sequence of scripts for a particular workload manager in special directories.

The directories have paths in the format:

- > `/cm/local/apps/slurm/var/prologs/`
- > `/cm/local/apps/slurm/var/epilogs/`

In these directories, scripts are stored with names that have suffixes and prefixes associated with them that makes them run in special ways, as follows:

- > Suffixes used in the prolog/epilog directory:
 - `-prejob` script runs before all jobs.
 - `-cmjob` script runs before job run in a cloud.
- > Prefixes used in the prolog/epilog directory: `00-` to `99-`.

Number prefixes determine the order of script execution, with scripts with a lower number running earlier.

The script names can therefore look like:

- > 01-prolog-prejob
- > 10-prolog-cmjob

Return values for the prolog/epilog scripts have these meanings:

- > 0: the next script in the directory is run.
- > A non-zero return value: no further scripts are executed from the prolog/epilog directory.

Often, the script in a prolog/epilog directory is not a real script but a symlink, with the symlink going to a real file located in a different directory. The general script is then able to take care of what is expected of the symlink. The name of the symlink, and destination file, usually hints at what the script is expected to do.

For example, if any health checks are marked to run as prejob checks during `cm-wlm-setup` configuration, then each of the PBS workload manager variants use the symlink `01-prolog-prejob` within the prolog directory `/cm/local/apps/<workload manager>/var/prologs/`. The symlink links to the script is `/cm/local/apps/cmd/scripts/prolog-prejob`.

In this case, the script is expected to run before the job.

```
[root@headnode apps]# pwd
/cm/local/apps
[root@headnode apps]# ls -l *pbs*/var/prologs/ openpbs/var/prologs/:
total 0
lrwxrwxrwx 1 root root ... 01-prolog-prejob -> /cm/local/apps/cmd/scripts/prolog-prejob
pbspro/var/prologs/:
total 0
lrwxrwxrwx 1 root root ... 01-prolog-prejob -> /cm/local/apps/cmd/scripts/prolog-prejob
```

Epilog scripts (which run after a job run) have the location `/cm/local/apps/<workload manager>/var/epilogs/`. Epilog script names follow the same execution sequence pattern as prolog script names.

The `01-prolog-prejob` symlink is created and removed by the cluster manager on each compute node where prejob is enabled in the workload manager entity. Each such entity provides an `Enable Prejob` parameter that affects the symlink existence:

```
[head->wlm[openpbs]]% get enableprejob yes
[head->wlm[openpbs]]%
```

This parameter is set to `yes` by `cm-wlm-setup` when at least one health check is selected as a prejob one. If any healthcheck was configured as a prejob check before `cm-wlm-setup` execution, and the administrator had a checkmark for that health check, then the prejob is considered enabled.

5.6.2 Workload Manager Configuration For Prolog and Epilog Scripts

The cluster manager configures generic prologs and epilogs during workload manager setup with `cm-wlm-setup`. The administrator can configure prologs and epilogs using appropriate parameters in the configuration of the workload managers, by creating the symlinks in the local prologs and epilogs directories.

Generic prologs and epilogs are configured by default to run on job compute nodes (one run per each node per job) for Slurm.

The following parameters for prologs and epilogs can be configured with `cmsh` or Base View:

- > **Prolog Slurmctld:** the fully qualified path of a program to execute before granting a new job allocation. The program is executed on the same node where the `slurm server` role is assigned. The path corresponds to the `PrologSlurmctld` parameter in `slurm.conf`.
- > **Epilog Slurmctld:** the fully qualified path of a program to execute upon termination of a job allocation. The program is executed on the same node where the `slurm server` role is assigned. Corresponds with the `EpilogSlurmctld` parameter in `slurm.conf`.
- > **Prolog:** the fully qualified path of a program to execute on job compute nodes before granting a new job or step allocation. The program corresponds to the `Prolog` parameter, and by default points to the generic prolog. This prolog runs on every node of the job if the `Prolog flags` parameter contains the flag `Alloc` (the default value), otherwise it is executed only on the first node of the job.
- > **Epilog:** the fully qualified path of a program to execute on job compute nodes when the job allocation is released.

5.7 Listing Slurm Jobs in the Queue

The jobs currently running on the cluster, or waiting in the queue to run, can be shown using the `squeue` command:

```
dgxa100@pg-login-mgmt001:~$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
25668	debug	submit_h	dgxa100	PD	0:00	1	(Priority)
25669	debug	submit_h	dgxa100	PD	0:00	1	(Priority)
25670	debug	submit_h	dgxa100	PD	0:00	1	(Priority)
25592	debug	submit_h	dgxa100	R	1:13	1	dgx081
25593	debug	submit_h	dgxa100	R	1:13	1	dgx090
25594	debug	submit_h	dgxa100	R	1:13	1	dgx097

The fifth column of the `squeue` output is the job state (`ST` in the header):

- > Jobs that are running will show in `R` state. The last column will list the nodes that the job is running on.
- > Jobs that are waiting to run will show in `PD` state, for pending. The last column will show the reason that the job is not running yet. This may be some reason such as `Resources` if there are not enough nodes available or `Priority` if the job is waiting in line behind another job with higher priority.

See the [squeue documentation](#) for information about the available job states and ways to filter the output.

5.8 Canceling a Slurm Job

Use the `scancel` command to cancel a Slurm job.

```
$ scancel <job-id>
```

If a running job is canceled, Slurm will send a `SIGTERM` signal to all the processes in the job. If the job processes do not end within a certain number of seconds (30s by default, configured with `KillWait`), then Slurm will send a `SIGKILL` signal.

If a pending job is canceled, Slurm will simply remove it from the queue.

5.9 Managing the Parameters on a Job

Each job has several configuration parameters associated with it, such as the time limit or the partition it is running in. These parameters can be viewed with the following command:

```
dgxa100@pg-login-mgmt001:~$ scontrol show job 25632
JobId=25632 JobName=submit_hpl_cuda11.0.sh
  UserId=dgxa100(13338) GroupId=dgxa100(13338) MCS_label=N/A
  Priority=44385 Nice=0 Account=compute-account QOS=normal
  JobState=PENDING Reason=Priority Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:00:00 TimeLimit=02:00:00 TimeMin=N/A
  SubmitTime=2020-12-06T06:36:59 EligibleTime=2020-12-06T06:36:59
  AccrueTime=2020-12-06T06:36:59
  StartTime=2020-12-06T22:04:00 EndTime=2020-12-07T00:04:00 Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-12-06T08:04:40
  Partition=debug AllocNode:Sid=pg-login-mgmt001:1107071
  ReqNodeList=dgx081 ExcNodeList=(null)
  NodeList=(null) SchedNodeList=dgx081
  NumNodes=1-1 NumCPUs=8 NumTasks=8 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=8,node=1,billing=8
  Socks/Node=* NtasksPerN:B:S:C=8:0:*:1 CoreSpec=*
  MinCPUsNode=8 MinMemoryCPU=0 MinTmpDiskNode=0
  Features=(null) DelayBoot=00:00:00
  OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
  Command=/mnt/test/deepops/workloads/burn-in/submit_hpl_cuda11.0.sh
```

```
WorkDir=/mnt/test/deepops/workloads/burn-in
StdErr=/mnt/test/deepops/workloads/burn-
in/results/1node_dgxa100_20201206063703/1node_dgxa100_20201206063703-25632.out
StdIn=/dev/null
StdOut=/mnt/test/deepops/workloads/burn-
in/results/1node_dgxa100_20201206063703/1node_dgxa100_20201206063703-25632.out
Power=
TresPerNode=gpu:8
MailUser=(null) MailType=NONE
```

Some of these parameters can be updated dynamically. For example, to extend the time limit of a job that might not finish in the time allowed, use the `scontrol update` command:

```
dgxa100@pg-login-mgmt001:~$ scontrol update job id=25632 timelimit=02:10:00
```

See the [scontrol documentation](#) for more information about viewing or modifying Slurm configurations.

5.9.1 Additional Resources

Slurm provides many advanced features that can provide more fine-grained control over job scheduling, system use, user and group accounting, and fairness of system use.

See these links for more information:

- > [SchedMD Slurm Quickstart Guide](#)
- > [LLNL Slurm Quickstart Guide](#)

6. Monitoring Cluster Devices

The cluster manager monitoring allows a cluster administrator to monitor anything that can be monitored in the cluster. Much of the monitoring consists of predefined sampling configurations. If there is anything that is not configured, but the data on which it is based can be sampled, then monitoring can be configured for it too, by the administrator.

The monitoring data can be viewed historically, as well as on demand. The historical monitoring data can be stored raw, and optionally also as consolidated data—a way of summarizing data.

The data can be handled raw and processed externally, or it can be visualized within Base View in the form of customizable charts. Visualization helps the administrator spot trends and abnormal behavior and is helpful in providing summary reports for managers.

Monitoring can be configured to set off alerts based on triggers, and predefined or custom actions can be conducted automatically, depending on triggers. The triggers can be customized according to user-defined conditional expressions.

Conducting such actions automatically after having set up triggers for them means that the monitoring system can free the administrator from having to carry out these chores.

In this chapter, the monitoring system is explained with the following approach:

1. A basic example is first presented in which processes are run on a node. These processes are monitored and trigger an action when a threshold is exceeded.
2. With this easy-to-understand example as a basic model, the various features and associated functionality of the cluster manager monitoring system are then described and discussed in further depth. These include visualization of data, concepts, configuration, monitoring customization and `cmsh` use.

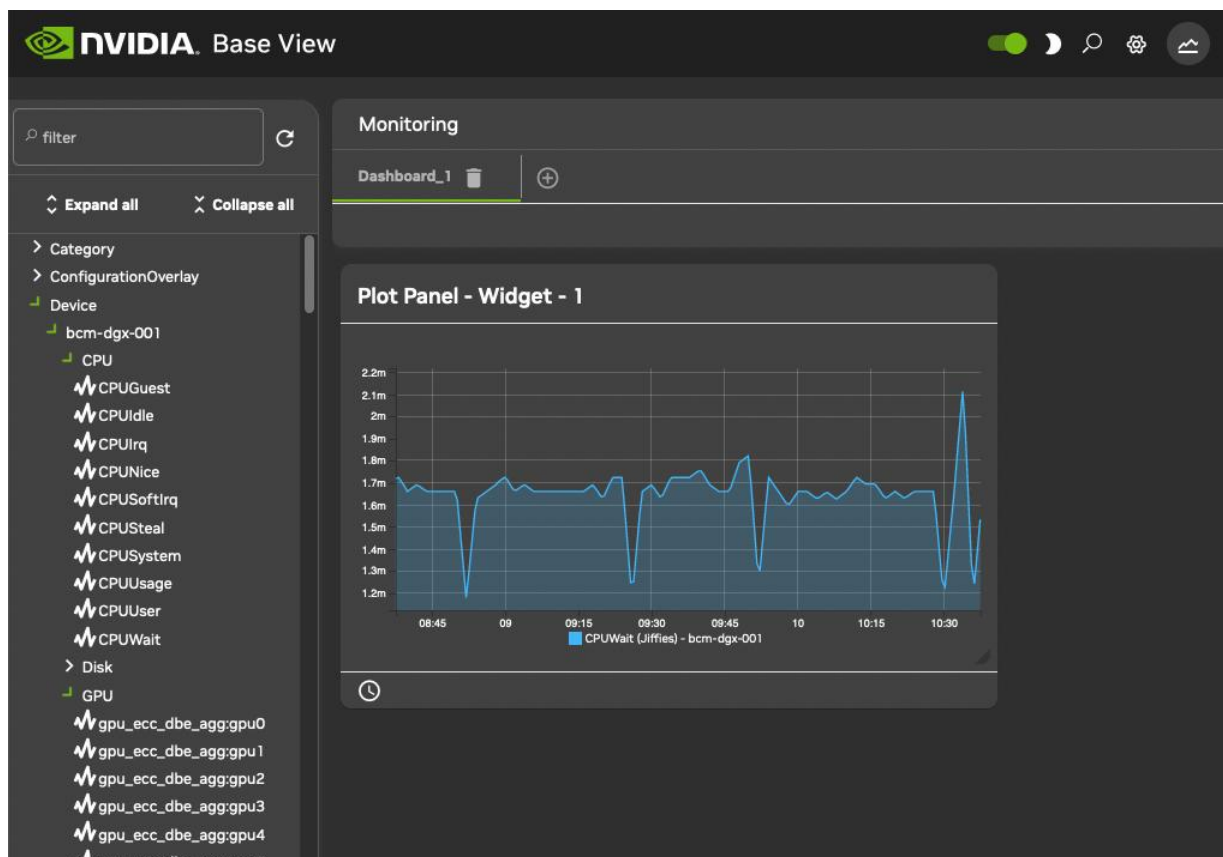
6.1 Basic Monitoring Example and Action

The example in this section is designed to present a basic illustration of what the monitoring system is capable of handling. This example describes a structure around which further details are fitted and filled in during the coverage in the rest of this chapter.

6.1.1 Synopsis Of Basic Monitoring Example

In this example, a user is running an artificial CPU-intensive process on a head node that 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 9).

Figure 9. CPU-intensive processes started, detected, and stopped



6.1.2 Setting Up the Pieces

6.1.2.1 Running Artificial Loads

One way to simulate a user running 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 eight subshell processes in the background from the CLI on the head node, with `yes` output sent to `/dev/null`:

```
for i in {1..8}; do ( yes > /dev/null & ); done
```

Running `mpstat 2` shows usage statistics for each processor, updating every two seconds. It shows that `%user`, which is user mode CPU usage percentage, is close to 90% on an eight-core or less head node when the eight subshell processes are running.

6.1.2.2 Setting Up the Kill Action

To stop the artificial CPU-intensive `yes` processes, the `killall yes` command can be used. The administrator can make it a part of a script `killalldoes`:

```
#!/bin/bash killall yes
```

and make the script executable with a `chmod 700 killalldoes`. For convenience, it may be placed in the `/cm/local/apps/cmd/scripts/actions` directory where some other action scripts also reside.

6.1.3 Using the Basic Monitoring Example

Now that the pieces are in place, the administrator can use Base View to add the `killalldoesaction` action to its action list, and then set up a trigger for the action.

6.1.3.1 Adding the Action to the Actions List

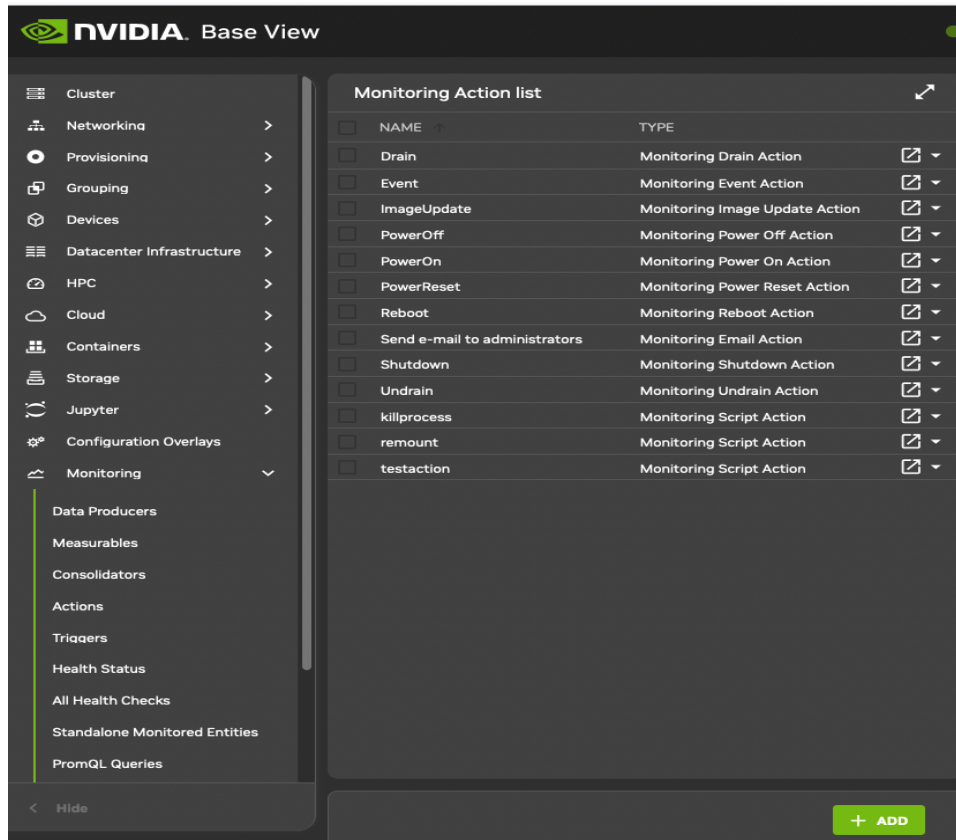
Clickpath Monitoring>Actions>Monitoring Actions>killprocess>Clone is used to clone the structure of an existing action. The `killprocess` action is convenient because it is expected to function in a similar way, so its options should not have to be modified much. However, any action could be cloned, and the clone modified in appropriate places.

The name of the cloned action is changed. That is, the administrator sets `Name` to `killalldoesaction`. The name of the file is arbitrary.

`Script` is set to the path `/cm/local/apps/cmd/scripts/actions/killalldoes`, which is where the script was placed earlier.

After saving, the `killallyesaction` action becomes part of the list of monitoring actions (Figure 10).

Figure 10. Base View monitoring configuration: adding an action



6.1.3.2 Setting Up a Trigger Using `CPUUser` on the Login Node

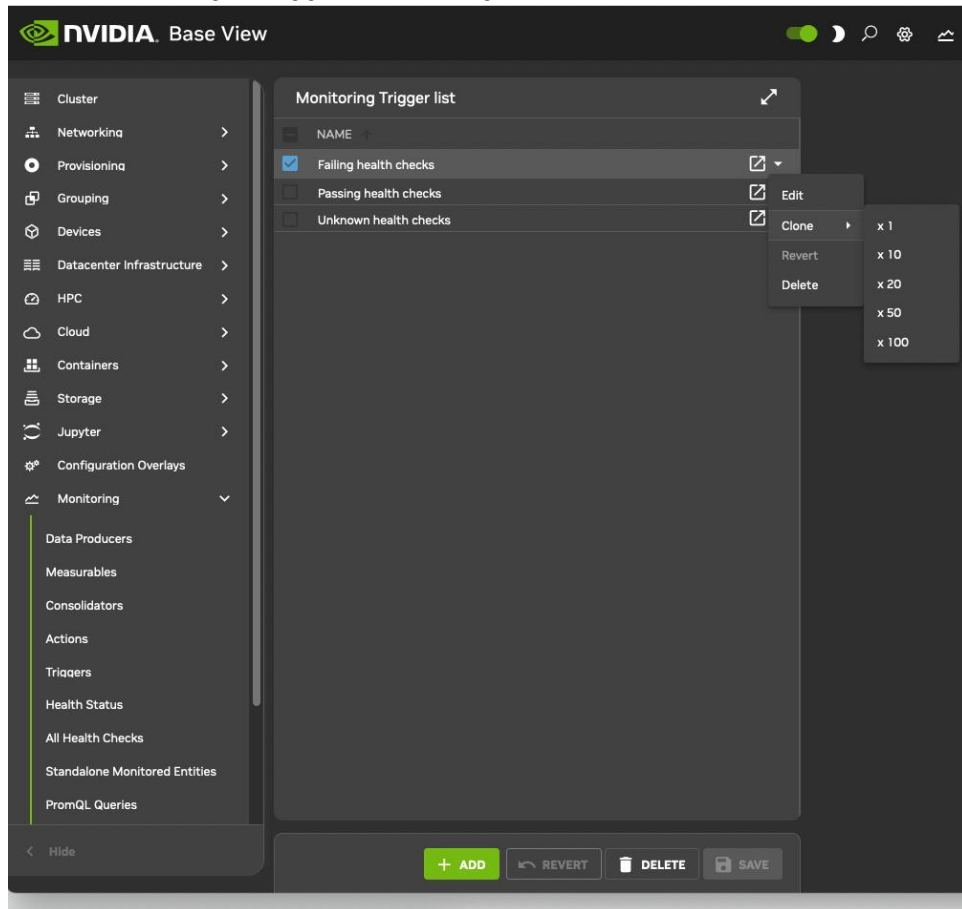
Clickpath `Monitoring>Triggers>Failing health checks>Clone` can be used to configure a monitoring trigger by cloning an existing trigger. A trigger is a sample state condition that runs an action. In this case, the sample state condition may be that the metric (6.2.1.8) `CPUUser` must not exceed 50. 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.

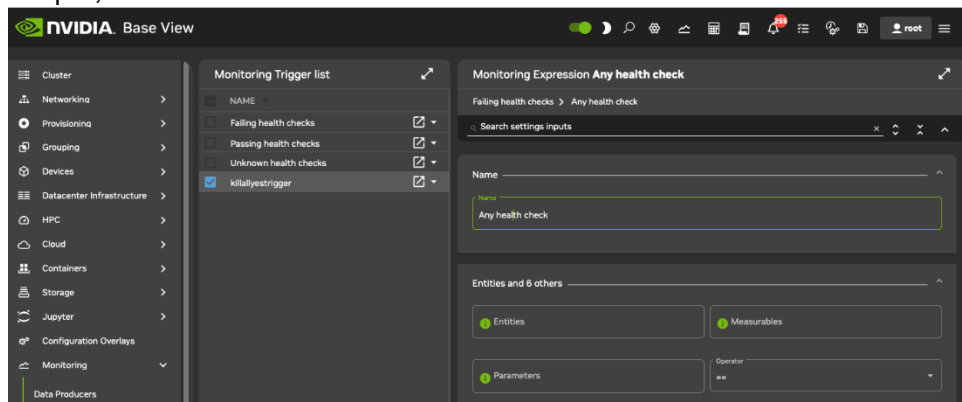
To configure the trigger attributes:

1. Goto the Monitoring Trigger list screen.

Clickpath Monitoring>Triggers>Failing health checks



2. Set a name for the trigger. The name is arbitrary. `killallyestrigger` is used in this example.
 3. Configure `Enter` actions so the trigger can run an action script if the sample state crosses over into a state that meets the trigger condition.
 4. Configure the condition under which the `Enter` actions action script is run. The condition can be set by setting an expression in the expression subwindow.
- For example, when `CPUUser` on the head node is above 50.



5. Set the name for the trigger. The name is arbitrary, and `killallyestrigger` is used in this example.
6. Set the `Name` for the expression. `Name` is arbitrary, and `killallyesregex` is in this example.
7. Set the entity. In this case, the entity being monitored is the head node. If the head node is called `headnode` in this example, then `headnode` is the value set for entities. An entity is often simply a device, but it can be any object that `CMDaemon` stores.
8. Set `Measurables`. In this case, `CPUUser` is used.
9. Set an operator and threshold. In this case `GT`, which is the greater than operator, and 50 that is a significant amount of `CPUUser` time in jiffies/s, are set for `Operator` and `Value`.
10. Enable the trigger to activate it.

After saving the configuration, the `killallyesregex` regular expression evaluates the data being sampled for the trigger. If the expression is `TRUE`, then the trigger launches the `killallyesaction` action.

6.1.3.3 The Result

In the preceding section, an action was added, and a trigger was set up with a monitoring expression. With a default installation on a newly installed cluster, the measurement of `CPUUser` is done every 120s (the period can be modified in the Data Producer window of Base View, as seen in [Figure 12.9](#) of the *Bright Cluster Manager Administrator Manual*). The basic example configured with the defaults thus monitors if `CPUUser` on the head node has crossed the bound of 50 jiffies/s every 120s.

If `CPUUser` is found to have entered—that is crossed over from below the value and gone into the zone beyond 50 jiffies—then the `killallyesregex` regular expression notices that. Then, the trigger it is configured for, `killallyestrigger` trigger, runs the `killallyesaction` action, which runs the `killallyes` script. The `killallyes` script kills all the running `yes` processes. Assuming the system is trivially loaded apart from these `yes` processes, the `CPUUser` metric value then drops to below 50 jiffies.

To clarify what “found to have entered” means in the previous paragraph:

After an `Enter` trigger condition has been met for a sample, the first sample immediately after that does not ever meet the `Enter` trigger condition, because an `Enter` threshold crossing condition requires the previous sample to be below the threshold.

The second sample can only launch an action if the `Enter` trigger condition is met and if the preceding sample is below the threshold.

Other non-`yes` CPU-intensive processes running on the head node can also trigger the `killallyes` script. Since the script only kills `yes` processes, leaving any non-`yes` processes alone, it would in such a case run unnecessarily. This is a deficiency due to the contrived and simple nature of the basic example that is being illustrated here. In a production case, the action script is expected to have a more sophisticated design.

The following sections in this chapter cover the concepts and features for the cluster manager monitoring in greater detail.

6.2 Monitoring Concepts and Definitions

A discussion of the concepts of monitoring, along with definitions of terms used, is appropriate at this point. The features of the monitoring system in the cluster manager covered later in this chapter will then be understood more clearly.

6.2.1 Measurables

Measurables are measurements (sample values) that are obtained using data producers (6.2.8) `CMDaemon` monitoring system. The measurements can be made for nodes, head nodes, other devices, or other entities.

6.2.1.1 Types of Measurables

Measurables can be:

- > **enummetrics**: measurements with a small number of states. The states can be predefined, or user-defined. These are covered in 6.2.1.7.
- > **metrics**: measurements with number values, and no data, as possible values. For example, values such as: `-13113143234.5`, `24`, `9234131299`. These are covered in 6.2.1.8
- > **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. These are covered in 6.2.2.

6.2.1.2 `no data` and Measurables

If no measurements are carried out, but a sample value must 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.

6.2.1.3 Entities and Measurables

Normally, a device, or a category or some similar grouping is a convenient idea to keep in mind as an entity, for concreteness.

The default entities in a new installation of the cluster manager are:

```
device category partition[base] software images
```

However, more generally, an entity can be an object from the following top-level modes of `cmsh`:

```
category ceph cloud cmjob configuration overlay device edgesight etcd  
fspart group jobqueue jobs kubernetes network nodegroup partition  
profile rack softwareimage user
```

For example, a software image object that is to be provisioned to a node is an entity, with some of the possible attributes of the entity being the `name`, `kernel version`, `creationtime`, or `locked` attributes of the image:

```
[root@headnode ~]# cmsh -c "software image use default-image; show"
Parameter                               Value
-----
Creation time                           Thu, 08 Jun 2017 18:15:13 CEST
Enable SOL                              no
Kernel modules                          <44 in submode> Kernel parameters
Kernel version                          3.10.0-327.3.1.el7.x86_64
Locked                                  no
Name                                    default-image
...
```

Because measurements can be carried out on such a variety of entities, it means that the monitoring and conditional actions that can be carried out on a cluster manager cluster can be very diverse. This makes entities a powerful and versatile concept in the cluster manager's monitoring system for managing clusters.

6.2.1.4 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.

```
[headnode->device]% measurables dgx001
Type      Name                Parameter  Class    Producer
-----
Enum      DeviceStatus                Internal  DeviceState
HealthCheck ManagedServicesOk          Internal  CMDaemonState
HealthCheck default gateway        Network  default gateway
HealthCheck diskspace         Disk     diskspace
HealthCheck dmesg              OS       dmesg
...
```

The subsets of these measurables can be listed with `list enum` (6.2.1.7), `list metric` (6.2.1.8), and `list healthcheck` (6.2.2).

In Base View, the equivalent to listing the measurables can be conducted using `clickpath Monitoring>All Health Checks`.

6.2.1.5 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:

```
[headnode->monitoring]% measurable list
Type      Name (key)                Parameter  Class                    Producer
-----
Enum      DeviceStatus                Internal  DeviceState
HealthCheck ManagedServicesOk          Internal  CMDaemonState
HealthCheck Mon::Storage        Internal/Monitoring/Storage MonitoringSystem
HealthCheck chrootprocess        OS        chrootprocess
HealthCheck cmsh                 Internal  cmsh
...
```

The subsets of these measurables can be listed with `list enum` (6.2.1.7), `list metric` (6.2.1.8), and `list healthcheck` (6.2.2).

In Base View, the equivalent to listing the measurables can be conducted through clickpath `Monitoring>Measurables` and listing the subsets of the measurables is possible using column filtering.

6.2.1.6 Viewing Parameters for a measurable in monitoring Mode

Within the `measurable` submode, parameters for a particular measurable can be viewed with the `show` command for that measurable.

```
[headnode->monitoring->measurable]% use devicestatus [headnode->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

6.2.1.7 Enummetrics

An *enummetric* is a measurable for an entity that can only take a limited set of values. `DeviceStatus` is the only enummetric. This may change in future versions of the cluster manager.

The full list of possible values for the enummetric `DeviceStatus` is:

up, down, closed, installing, installer_failed, installer_rebooting, installer_callinginit, installer_unreachable, installer_burning, burning, unknown, opening, going_down, pending, and no data.

The enummetrics available for use can be listed from within the `measurable` submode of the monitoring mode:

```
[headnode->monitoring->measurable]% list enum
```

Type	Name (key)	Parameter	Class	Producer
Enum	DeviceStatus		Internal	DeviceState

```
[headnode->monitoring->measurable]%
```

The list of enummetrics that is configured by an entity, such as a device, can be viewed with the `enummetrics` command for that entity:

```
[headnode->device]% enummetrics dgx001
```

Type	Name	Parameter	Class	Producer
------	------	-----------	-------	----------

```
-----
Enum      DeviceStatus                                Internal  DeviceState
[headnode->device]%
```

The states that the entity has been through can be viewed with the [dumpmonitoringdata](#) command:

```
[headnode->device]% dumpmonitoringdata -99d now devicestatus dgx001
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 (6.2.1.6).

6.2.1.8 Metrics

A *metric* for an entity is typically a numeric value for an entity. The value can have units associated with it.

In the example of 6.1, the metric value considered was `CPUUser`, measured at the default regular time intervals of 120s.

The value can also be defined as `no data`. `no data` is substituted for a null value when there is no response for a sample. `no data` is not a null value once it has been set. This means that there are no null values stored for monitored data.

Other examples for metrics are:

- > `LoadOne` (value is a number, for example: 1.23).
- > `WriteTime` (value in ms/s, for example: 5 ms/s).
- > `MemoryFree` (value in readable units, for example: 930 MiB, or 10.0 GiB).

A metric can be a built-in, which means it comes with the cluster manager as integrated code within `CMDaemon`. This is based on `c++` and is therefore much faster than the alternative. The alternative is that a metric can be a standalone script, which means that it typically can be modified more easily by an administrator with scripting skills.

The word *metric* is often used to mean the script or object associated with a metric as well as a metric value. The context makes it clear that is meant.

Metrics in use can be viewed in `cmsh` using the `list` command from `monitoring` mode:

```
[headnode->monitoring]% measurable list metric
Type      Name (key)                Parameter      Class      Producer
-----
Metric    AlertLevel                count          Internal   AlertLevel
Metric    AlertLevel                maximum        Internal   AlertLevel
...
```

In Base View, the metrics can be viewed with clickpath `Monitoring>Measurables` and then clicking on the filter widget to select `Metric`.

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 `dgx001` in mode `devices`:

```
[headnode->devices]% metrics dgx001
```

Type	Name	Parameter	Class	Producer
Metric	AlertLevel	count	Internal	AlertLevel
Metric	AlertLevel	maximum	Internal	AlertLevel
...				

The parameters of a metric such as `AlertLevel:count` can be viewed and set from monitoring mode, from within the `measurable` submode, just as for the other **measurables**:

```
[headnode->monitoring->measurable]% use alert level:count
[headnode->monitoring->measurable[AlertLevel:count]]% show
```

Parameter	Value
Class	Internal
Consolidator	default
Cumulative	no
Description	Number of active triggers
Disabled	no
Maximal age	0s
Maximal samples	0
Maximum	0
Minimum	0
Name	AlertLevel
Parameter	count
Producer	AlertLevel
Revision	
Type	Metric

The equivalent Base View clickpath to edit the parameters is `Monitoring>Measurables>Edit`.

6.2.2 Health Check

A *health check* value is a response to a check conducted on an entity. The response indicates the health of the entity for that check.

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 had an unknown response. For example, if `ssh2node` has a timeout, due to routing or other issues, it means that it is unknown whether the connection is fine or failing. 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
```

6.2.2.1 Health Checks

In Base View, the health checks that can be configured for all entities can be seen with clickpath `Monitoring>Measurables` and then clicking on the filter widget to select `Health Check`. Options can be set for each health check by clicking through using the `Edit` button.

6.2.2.2 All Configured Health Checks

All configured healthchecks can be viewed using the clickpath `Monitoring>All Health Checks`. The view can be filtered per column.

6.2.2.3 Configured Health Checks for an Entity

An overview can be seen for a particular entity `<entity>` using clickpath `Monitoring>Health status>entity>Show`.

6.2.2.4 Severity Levels for Health Checks, and Overriding Them

A health check has a settable severity (6.2.5) associated with its response defined in the trigger options.

For standalone health checks, 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 (6.2.6) when the health check runs.

6.2.2.5 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:

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 email 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.

6.2.3 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` (6.2.5) associated with it. This value is processed for the `AlertLevel` metric (6.2.6) when an action is triggered by a threshold event.

Triggers are discussed in 6.1.3.2.

6.2.4 Action

In the basic example of 6.1, 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 (6.2.7)

The actions that can be run are listed from within the action submode of the monitoring mode.

```
[headnode->monitoring->action]% list
```

Type	Name (key)	Run on	Action
Drain	Drain	Active	Drain node from all WLM
Email	Send e-mail	Active	Send e-mail
Event	Event	Active	Send an event to users with connected client
ImageUpdate	ImageUpdate	Active	Update the image on the node
PowerOff	PowerOff	Active	Power off a device
PowerOn	PowerOn	Active	Power on a device
PowerReset	PowerReset	Active	Power reset a device
Reboot	Reboot	Node	Reboot a node

Script	killallyesaction	Node	/cm/local/apps/cmd/scripts/actions/killallyes
Script	killprocess	Node	/cm/local/apps/cmd/scripts/actions/killprocess.pl
Script	remount	Node	/cm/local/apps/cmd/scripts/actions/remount
Script	testaction	Node	/cm/local/apps/cmd/scripts/actions/testaction
Shutdown	Shutdown	Node	Shutdown a node
Undrain	Undrain	Active	Undrain node from all WLM







The Base View equivalent is accessible using clickpath **Monitoring>Actions**.

Configuration of monitoring actions is discussed further in 6.1.3.1.

6.2.5 Severity

Severity is a positive integer value that the administrator assigns for a trigger. It takes one of six suggested values (Table 12).

Table 12. Severity 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 (6.2.6). They can also be set by the administrator in the return values of health check scripts (6.2.2).

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 (6.2.2).

6.2.6 AlertLevel

AlertLevel is a special metric. It is sampled and re-calculated when an event with an associated *Severity* (6.2.5) occurs.

There are three types of `AlertLevel` metrics:

1. `AlertLevel (count)`: the number of events that are at notice level and higher. This metric alerts the administrator to the number of issues.
2. `AlertLevel (max)`: the maximum severity of the latest value of all the events. This metric alerts the administrator to the severity of the most important issue.
3. `AlertLevel (sum)`: the sum of the latest severity values of all the events. This metric alert the administrator to the overall severity of issues.

6.2.7 Flapping

Flapping, or *State Flapping*, is when a measurable trigger (6.2.3) that is detecting changes, changes that are too frequent. That is, the measurable goes in and out of the zone too many times over several samples.

In the basic example of 6.1, if the `CPUUser` metric crossed the threshold zone five times within five minutes (the default values for flap detection), 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.

6.2.8 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:

```
[headnode->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:

```
[headnode->monitoring->measurable]% list -f name:25,producer:15 | grep 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 data producers (used and unused) can be listed as follows:

```
[headnode->monitoring->setup]% list
```

The equivalent in Base View is using clickpath `Monitoring>Data Producers`.

The data producers configured for an entity, such as a head node `headnode`, can be listed with the `monitoringproducers` command:

```
[headnode->device[headnode]]% monitoringproducers
```

Type	Name	Arguments	Measurables	Node execution filters
AlertLevel	AlertLevel		3 / 231	<0 in submode>
CMDaemonState	CMDaemonState		1 / 231	<0 in submode>
ClusterTotal	ClusterTotal		18 / 231	<1 in submode>
Collection	NFS		32 / 231	<0 in submode>
Collection	sdt		0 / 231	<0 in submode>
DeviceState	DeviceState		1 / 231	<1 in submode>
HealthCheckScript	chrootprocess		1 / 231	<1 in submode>
HealthCheckScript	cmsh		1 / 231	<1 in submode>
HealthCheckScript	default gateway		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>

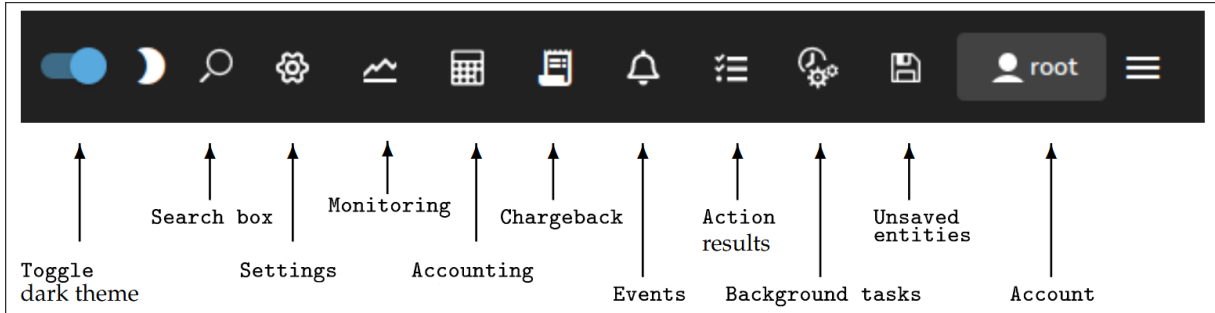
The displayed data producers are the ones configured for the entity, even if there are no measurables used by the entity.

Data producer configuration in Base View is discussed further in [Section 12.4.1](#) of the *Bright Cluster Manager Administrator Manual*.

6.2.9 Main Monitoring Interfaces of Base View

Base View, besides having the default settings mode, has some other display modes and logging view modes that can be selected using the 11 icons in the top-right corner of the Base View standard display (Figure 11).

Figure 11. Base View top-right corner icons



The 11 icons are described from left to right next:

1. `Toggle dark theme` option allows the display of Base View to use a darker theme.
2. `Search box` allows resource to be searched for, with predictive text suggestions.
3. `Settings` mode is active when Base View first starts up.
The `Settings` mode has a navigation panel to the left of it, showing the resources of the cluster as expandable items. One of the resources is `Monitoring`. This resource should not be confused with the Bright View `Monitoring` mode, which is launched by the next icon. The `Monitoring` resource is about configuring how items are monitored and how their data values are collected.
4. The `Monitoring` mode allows visualization of the data values collected according to the specifications of the Bright View `Monitoring` resource. The visualization allows graphs to be configured.
5. The `Accounting` mode typically allows visualization of job resources used by users, although it can be used to visualize job resources used by other aggregation entities. This is helpful tracking resources consumed by users.
6. The `Chargeback` mode allows the monitoring of resources requested over a period for jobs run by selected groups.
7. The `Events` icon allows logs of events to be viewed.
8. The `Action results` icon allows the logs of the results of actions to be viewed.
9. The `Background tasks` icon allows background tasks to be viewed.
10. The `Unsaved entities` icon allows unsaved entities to be viewed.
11. The `Account` handling icon allows account settings to be managed for the Base View user.

6.3 Monitoring Visualization with Base View

The Monitoring icon in the menu bar of Base View (Figure 11) 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 over a particular time, the graphs can be laid out on top of each other, or the graphs can be laid out as a giant grid. The graph scale settings can also be adjusted, stored, and recalled for use the next time a session is started.

An alternative to Base View's visualization tool is the CLI `cmsh`. This has the same functionality in the sense that data values can be selected and studied according to configurable parameters with it. 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.

See [Section 12.5](#) and [Section 12.6](#) of the *Base Command Manager Administrator Manual* for more information about monitoring with `cmsh`.

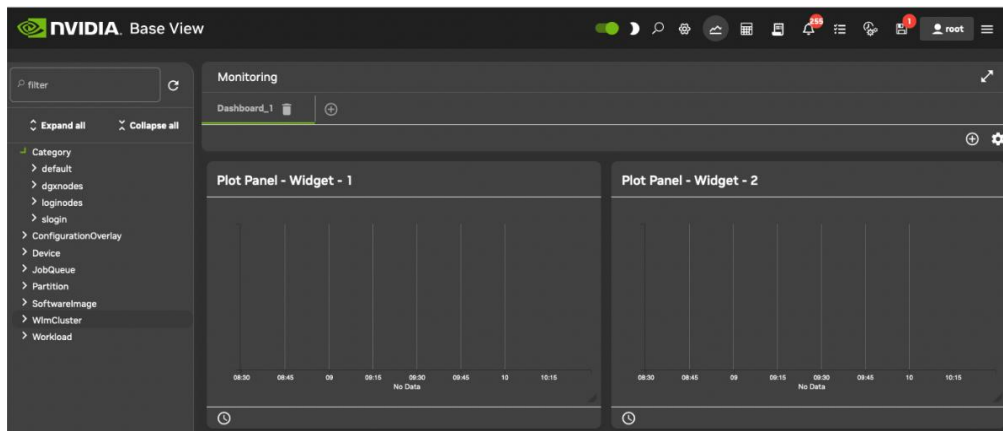
Visualization of monitoring graphs with Base View is now described.

6.3.1 The Monitoring Window

Selecting the `Monitoring` icon from the menu bar of Base View (Figure 11) launches a monitoring window for visualizing data opens. 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.

The monitoring window for Base View is shown in Figure 12.

Figure 12. Base View monitoring window

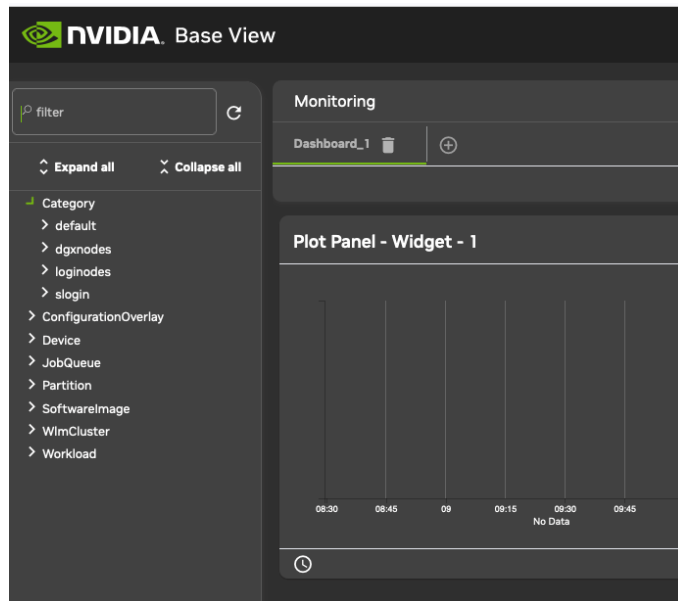


6.3.1.1 Finding and Selecting the Measurable to be Plotted

To plot measurables, the entity that 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 `headnode`, it can be selected from the navigation clickpath `Device>headnode>CPU>CPUUser`.

Sometimes, finding a measurable is easier with the filter search box. Entering `CPUUser` there shows all the measurables with that text (Figure 13). The search is case-insensitive.

Figure 13. Base View monitoring filter search box



The filter search box can handle some simple regexes too, with `.`, `*` and `|` taking their usual meaning:

- > `dgx001.*cpuuserdgx001cpuuser`
- > `(dgx001|dgx002).*cpuuserdgx002dgx001`

The `/` (forward slash) allows filtering according to the data path. It corresponds to the navigation depth in the tree hierarchy:

```
dgx001/cpu/cpu user
```

Searches for a measurable with a data path that matches `dgx001/cpu/cpu user`.

6.3.1.2 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://blocks.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 centered around the position of the mouse pointer.

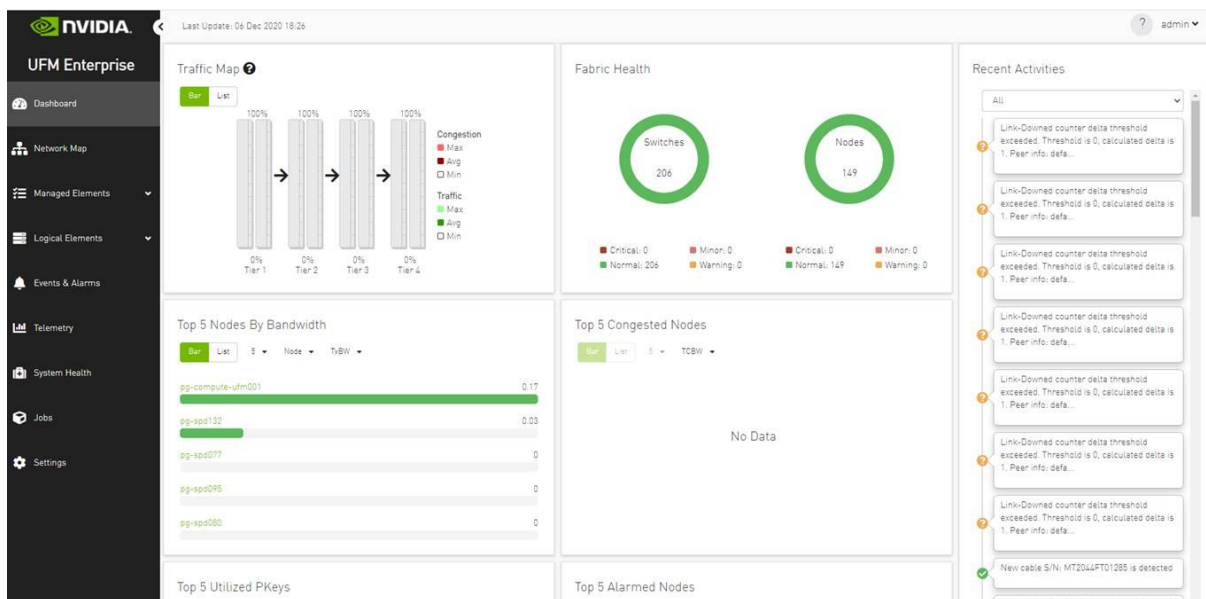
7. Managing High-Speed Fabrics

The high-speed InfiniBand fabrics are managed with NVIDIA Unified Fabric Manager (UFM). UFM is a powerful platform for managing scale-out computing environments. UFM enables data center operators to efficiently monitor and operate the entire fabric, boost application performance, and maximize fabric resource utilization.

While other tools are device-oriented and involve manual processes, UFM automated and application-centric approach bridges the gap between servers, applications, and fabric elements, thus enabling administrators to manage and optimize from the smallest to the largest and most performance-demanding clusters.

The dashboard for UFM is shown in Figure 14.

Figure 14. UFM Dashboard



7.1 Verifying that UFM is Running

Use the `service ufmha status` command to verify UFM is running:

```
ufm001# service ufmha status
ufmha status
=====
Local Host
Server          ufm001
Kernel          3.10.0-1127.19.1.el7.x86_64
IP Address      10.166.130.31
HA Interface    bond0
DRBD Partition  /dev/sda6
Heartbeat       Master
Mysql           Running
UFM Server      Running
DRBD State      Primary
DRBD Device State UpToDate
=====
Remote Host
Server          ufm002
Kernel          3.10.0-1127.19.1.el7.x86_64
IP Address      10.166.130.32
HA Interface    bond0
DRBD Partition  /dev/sda6
Heartbeat       Slave
Mysql           Stopped
UFM Server      Stopped
DRBD State      Secondary
DRBD Device State UpToDate
=====
Virtual IP      10.166.130.58/24
Broadcast IP    10.166.130.255
=====
```

Refer to <https://support.mellanox.com/s/productdetails/a2v50000000XcP4AAK/ufm> for the full documentation.

8. System Health Checks and Debugging

The key to successfully operating and managing a cluster is that all nodes are configured identically for their function, and they operate consistently. When issues arise, it becomes necessary to test systems to see if they are operating correctly.

If an issue is found, it should be removed from the batch partition for initial triage.

Unless the issue is obvious, follow the GPU System debugging guidelines process that is at <https://docs.nvidia.com/deploy/gpu-debug-guidelines/index.html>

In addition, run tools specific to the DGX A100 systems. For health checks, this is the NVIDIA System Management tool (`nvsml`).

It is also useful to develop a set of single-node and multi-node tests to help validate the operation and performance of the DGX SuperPOD (Table 13). Often it is best to use your own key applications for this purpose as those exercise the system in the way that is most important to the users.

Table 13. DGX SuperPOD validation tools

Software	Purpose	Link
NCCL	Fabric	https://github.com/NVIDIA/nccl-tests
HPL	Math intensive applications with network communications	https://github.com/NVIDIA/deepops/tree/master/workloads/burn-in

In addition, there are standard applications that can be used to validate both single- and multi-node performance. When running the following tests, you should expect that performance between runs of the same configuration on distinct parts of the system should run in a similar time or at a similar performance level. Performance can vary between run-to-run because of system configuration and existing job load. However, over multiple runs on the same sets of hardware a difference is found, it can indicate an issue with some component of that system.

8.1 Collecting Log Files

Important log files include:

1. `/var/log/cmdaemon` (the most important one—CMDaemon log file).
2. `/var/log/node-installer` (node-installer log file).
3. `/var/spool/cmd/<slave-node-name>.rsync` (provisioning logs).

8.1.1 Log Subsystems

Each logging message is emitted as part of a `subsystem` which is defined on a per-translation unit basis. Example subsystems include `CONFIG`, `MIC`, `GPU`, `CLOUD`, `PROV`, `SERVICE`, `WLM`, `DB`, `USER`, `JSON`, `HADOOP`, and `CMD` (can be found in `logger.h`).

Example log output `/var/log/cmdaemon`:

```
Mar 30 03:38:02 headnodeName cmd: [ CLOUD ] DevDbg:
Mar 30 03:38:02 headnodeName cmd: [ CMD ] Debug: [programrunner.cpp:797 ]
ProgramRunner: /cm/local/apps/cmd/scripts/cloudproviders/openstack/openstackcommands.py [DONE]
0 0
Mar 30 03:38:02 headnodeName cmd: [ CMD ] Warning: [magicmanager.cpp:1797 ] This
is a warning.
```

The subsystem enclosed in `[]`, followed by the log type (`debug`, `warning`, `info`, `error`), the location in the source code (present only in `-D _DEBUG` compiles), and followed by the log message.

8.1.2 Increasing Log Verbosity

Verbosity of individual subsystems can be changed using the `/cm/local/apps/cmd/etc/logging.cmd.conf` config file. After modifying `logging.cmd.conf` one must either restart `CMDaemon` or run `service cmd logconf` to reload the logging config file.

The default settings in this file are as follows:

```
Severity {
    info: *
    warning: *
    debug:
    error: *
}
```

Which means that messages from all subsystems in all verbosity levels (except "debug") are always logged. Modifying `logging.cmd.conf` is useful when focusing on developing futures for only specific subsystems, as it can be used to quiet down logs from the remaining subsystems and focus only on essentials.

That is, all log messages from the `CLOUD` subsystem can be enabled, while only allowing `WARNING` and `ERROR` messages from all other remaining subsystems.

```
Severity {  
    info: CLOUD  
    warning: *  
    debug: CLOUD  
    error: *  
}
```

One can also use `logging.cmd.conf` to optionally enable logging of ThreadIDs, subsystem names, and microsecond-resolutions in the timestamps.

8.1.3 Global Debug Mode

One can toggle the so called `global debug mode` by means of `service cmd debug{on|off}` or by means of starting `cmd` with `-d` flags. In this mode, the custom settings from `logging.cmd.conf` are ignored and instead all log messages from all subsystems are always logged in maximum verbosity. Global debug mode is equivalent to using the following `logging.cmd.conf`:

```
Severity {  
    info: *  
    warning: *  
    debug: *  
    error: *  
}
```

8.1.4 LOGPREFIX

Use the `LOGPREFIX("DeviceManager")` macro to prepend all subsequent `log{i,d,dd,e,w}()` calls with an additional text, for example:

```
Manager::someFun() {  
    LOGPREFIX("SomeFunction:");  
    logdd("entered function");  
    ...  
    logdd("left function");  
}
```

Will result in the following logs:

```
"SomeFunction: entered function"  
"SomeFunction: left function"
```

9. 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](#), the head node also always has a provisioning role.

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.

Table 14 shows provisioning role parameters.

Table 14. Provisioning role parameters

Parameter	Description
allImages	The following values decide what images that the provisioning node provides:
Onlocaldisk	all images on the local disk, regardless of any other parameters set
Onsharedstorage	all images on the shared storage, regardless of any other parameters set
no (the default)	only images listed in the localimages or sharedimages parameters
localimages	A list of software images on the local disk that the provisioning node accesses and provides. The list is used only if allImages is “no”
sharedimages	A list of software images on the shared storage that the provisioning node accesses and provides. The list is used only if allImages is “no”
Provisioning slots	The maximum number of nodes that can be provisioned in parallel by the provisioning node. The optimum number depends on the infrastructure. The default value is 10, which is safe for typical cluster setups. Setting it lower may sometimes be needed to prevent network and disk overload.
Nodegroups	A list of node groups (2.1.4). If set, the provisioning node only provisions nodes in the listed groups. Conversely, nodes in one of these groups can only be provisioned by provisioning nodes that have that group set. Nodes without a group, or nodes in a group not listed in nodegroups, can only be provisioned by provisioning nodes that have no nodegroups values set. By default, the nodegroups list is unset in the provisioning nodes. The nodegroups setting is typically used to set up a convenient hierarchy of provisioning, for example based on grouping by rack and by groups of racks.

9.1 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.

Once the shell is within the role level, the role properties can be edited.

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):

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

Assigning a provisioning role can also be done for an individual node instead, if using a category is deemed overkill:

```
[headnode]% device use dgx001 [headnode->device[dgx001]]% roles
[headnode->device[dgx001]->roles]% assign provisioning
[headnode->device*[dgx001*]->roles*[provisioning*]]%
...
```

A role change configures a provisioning node but does not directly update the provisioning node with images. After conducting a role change, the cluster manager runs the `updateprovisioners` command described in 9.3 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 compute node images propagate efficiently. Thus, for example, image update requests by provisioning nodes take priority over provisioning update requests from compute nodes.

Other assignable provisional roles include `monitoring`, `storage`, and `failover`.

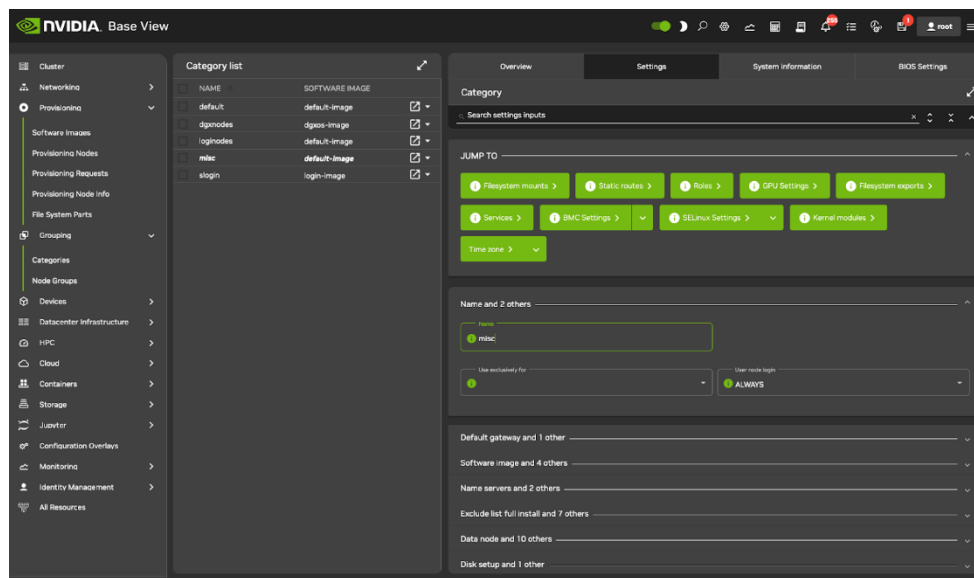
9.2 Role Setup with Base View

The provisioning configuration outlined in `cmsh` mode (9.1) can be done using Base View.

A `misc` category can be added using `clickpath`
`Grouping>Categories>Add>Settings<name>.`

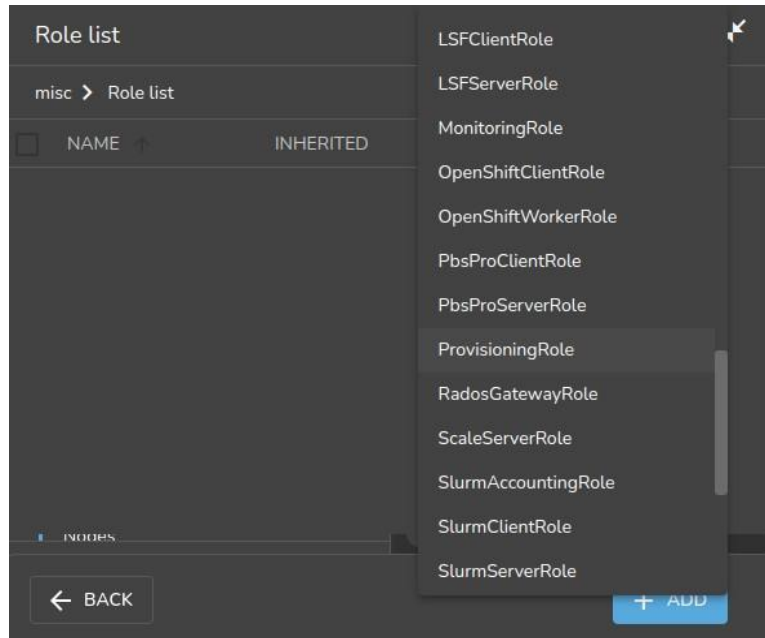
Within the `Settings` tab, the node category should be given a name `misc` (Figure 15) and saved.

Figure 15. Base View: Adding a `misc` category



The Roles window can then be opened from within the `JUMP TO` section of the settings pane. To add a role, select the `+ Add` button in the `Roles` window. A scrollable list of available roles is then displayed (Figure 16).

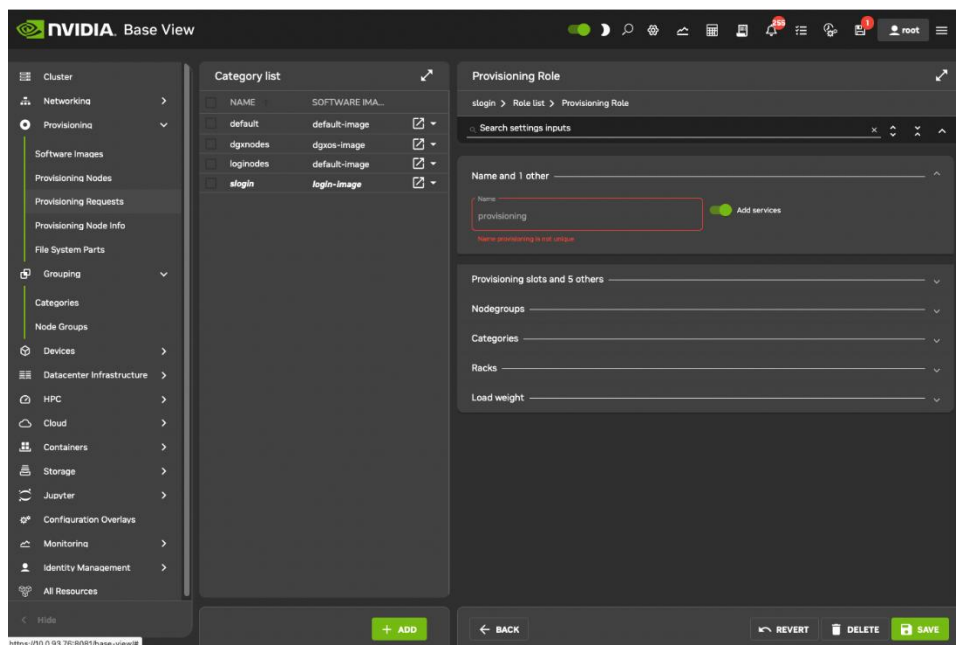
Figure 16. Base View: Setting a provisioning role



After selecting a role, navigating using the `Back` buttons to the `Settings` menu, and select the `Save` button.

The role has properties that can be edited (Figure 17).

Figure 17. Base View: Configuring a Provisioning Role



For example:

- > The `Provisioning slots` setting decides how many images can be supplied simultaneously from the provisioning node.
- > The `All images` setting decides if the role provides all images.
- > The `Local images` setting decides what images the provisioning node supplies from local storage.
- > The `Shared images` setting decides what images that the provisioning node supplies shared storage.

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.

9.3 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, must 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.

9.3.1 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.

9.3.2 Limiting Provisioning Tasks

Besides limiting how much simultaneous provisioning per provisioning node is allowed with `Provisioning slots` (9), 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](#) of the *Bright Cluster Manager Administrator Manual*.

9.3.3 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. Five retry attempts at provisioning the image are made in case a provisioning request fails.

A provisioning node that loses connectivity while carrying out requests, will have the provisioning requests fail after 180 seconds from the time that connectivity was lost.

9.3.4 Role Change Notification

The `updateprovisioners` command can be accessed from the `softwareimage` mode in `cmsh`. It can also be accessed from Base View, using clickpath `Provisioning>Provisioning requests>Update provisioning nodes`.

In the examples in 9.1, 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, 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 image. A provisioning node undergoing an image update does not provision other nodes until the update is completed.

```
[headnode]% software image updateprovisioners Provisioning nodes will be updated in the
background.
Sun Dec 12 13:45:09 2010 headnode: Starting update of software image(s)\ provisioning node(s).
(user initiated).
[headnode]% software image updateprovisioners
[headnode]%
Sun Dec 12 13:45:41 2010 headnode: Updating image default-image on provisioning node dgx001.
[headnode]%
Sun Dec 12 13:46:00 2010 headnode: Updating image default-image on provisioning node dgx001
completed.
Sun Dec 12 13:46:00 2010 headnode: Provisioning node dgx001 was updated Sun Dec 12 13:46:00
2010 headnode: Finished updating software image(s) \ on provisioning node(s).
```

9.3.5 Role Draining and Undraining Nodes

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, all 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.

```
[headnode->software image]% drain -n master Nodes drained
[headnode->software image]% provisioningstatus Provisioning subsystem status
Pending request:  dgx001, dgx002 Provisioning node status:
+ headnode
Slots:    1 / 10
State:    draining
Active nodes:  dgx003
Up to date images:  default-image [headnode->software image]% provisioningstatus Provisioning
subsystem status
Pending request:  dgx001, dgx002 Provisioning node status:
+ headnode
Slots:    0 / 10
State:    drained
Active nodes:  none
Up to date images:  default-image
```

Use the `--role provisioning` option to drain all nodes in parallel. All pending requests then remain in the queue until the nodes are undrained again.

```
[headnode->software image]% drain --role provisioning
...Time passes. Pending
requests stay in the queue. Then admin undrains it...
[headnode->software image]% undrain --role provisioning
```

9.3.6 Provisioning Node Update Safeguards

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 `provisioningnodeautoupdatetimeout`, which has a default value of 300s.

Exceeding the timeout does not by itself trigger an update to the provisioning node.

When the head node receives a provisioning request, it checks if the last update of the provisioning nodes is more than the timeout period. If true, then an update is triggered to the provisioning node. The update is disabled if the timeout is set to zero (`false`).

The parameter can be accessed and set within `cmsh` from `partition` mode:

```
[root@brght92 ]# cmsh [headnode]% partition use base
[headnode->partition[base]]% get provisioningnodeautoupdatetimeout
[headnode->partition[base]]% 300
[headnode->partition[base]]% set provisioningnodeautoupdatetimeout 0
[headnode->partition*[base*]]% commit
```

Within Base View, the parameter is accessible through `clickpath Cluster>Partition[base]>Provisioning Node Auto Update Timeout`.

To prevent provisioning an image to the nodes, [it can be locked](#). The provisioning request is then deferred until the image is again unlocked.

9.3.6.1 Synchronization of `fspart` Subdirectories to Provisioning Nodes

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

The `fsparts` can be listed with:

```
[root@headnode ]# cmsh [headnode]% fspart [headnode->fspartJ] 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 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 the non-image types can be updated with the `--all` option:

```
[headnode->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.

```
[headnode->fspart]% info
```

Path	Architecture	OS	Inotify watchers
/cm/images/default-image	x86_64	ubuntu2004	0
/cm/images/default-image/boot	-	-	0
/cm/node-installer	x86_64	ubuntu2004	0
/cm/shared	x86_64	ubuntu2004	0
/tftpboot	-	-	0
/var/spool/cmd/monitoring	-	-	0

```
[headnode->fspart]% info -s Path (!#with size, takes longer)
```

Path	Architecture	OS	Inotify watchers	Size
/cm/images/default-image	x86_64	ubuntu2004	0	4.2 GiB
/cm/images/default-image/boot	-	-	0	179 MiB
/cm/node-installer	x86_64	ubuntu2004	0	2.45 GiB
/cm/shared	x86_64	ubuntu2004	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.

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

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

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

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

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

Access to excludelistsnippets

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

```
[headnode->fspart]% excludelistsnippets /tftpboot
[headnode->fspart[/tftpboot]->exclude list snippets]% list
Name (key)    Lines    Disabled  Mode sync  Mode full  Mode update  Mode grab  Mode grab new
-----
Default      2        no        yes        yes        yes         no         no

[headnode->fspart[/tftpboot]->exclude list snippets]% 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

[headnode->fspart[/tftpboot]->exclude list snippets]% get default exclude list
# no need for rescue on nodes with a boot role
/rescue
/rescue/
```

10. Product Security

NVIDIA takes security concerns seriously and works to quickly evaluate and address them. Once a security concern is reported, NVIDIA commits the appropriate resources to analyze, validate, and provide corrective actions to address the issue.

For information on NVIDIA product security goto:

<https://www.nvidia.com/en-us/security/>

11. Backups

11.1 Cluster Installation Backup

The cluster manager does not include facilities to create backups of a cluster installation. The cluster administrator is responsible for deciding on the best way to back up the cluster, out of the many possible choices.

A backup method is strongly recommended and checking that restoration from backup 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 17.4.8](#) of the *Bright Cluster Manual Administrator Manual*.

When setting up a backup mechanism, include the full filesystem of the head node (i.e. including all software images). Unless the compute 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 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 by those ports from an IP address of 93.184.216.34 on the external network:

- `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>.

11.2 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 file system itself (“local rotating backup”) for the last seven 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 seven days too.

11.2.1 Database Corruption and Repairs

A corrupted MySQL database is often 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 Base 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 conducted on a database, `CMDaemon` should first be stopped.

```
[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.

If `CMDaemon` is unable to start up due to a corrupted database, messages in the `/var/log/cmdaemon` file might show something like:

```
Oct 11 15:48:19 headnode CMDaemon: Info: Initialize cmdaemon database
Oct 11 15:48:19 headnode CMDaemon: Info: Attempt to set provisioning Network (280374976710700)
not an element of networks
Oct 11 15:48:19 headnode CMDaemon: Fatal: Database corruption! Load Master Node with key:
280374976782569
Oct 11 15:48:20 headnode CMDaemon: Info: Sending reconnect command to all nodes which were up
before master went down ...
Oct 11 15:48:26 headnode CMDaemon: Info: Reconnect command processed.
```

Here it is the `CMDaemon 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 that a restoration from backup is needed. How to restore from backup is covered next.

11.2.2 Restoring from 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 should normally provide a `CMDaemon` and Slurm database that is current. The `dbreclone` option does not clone the monitoring database.

11.2.3 Cloning Databases

The `cm-clone-monitoring-db.sh` helper script that comes with `CMDaemon` can be used to clone the monitoring database.

11.2.4 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:

```
[root@headnode ~]# cd /var/spool/cmd/backup/
[root@headnode 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
...
```

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](#) of the *Bright Cluster Manager Administrator Manual*).

```
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](#).

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgment, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regard to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, NVIDIA Base Command, NVIDIA DGX, and NVIDIA DGX SuperPOD are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023 NVIDIA Corporation. All rights reserved.