NVIDIA Mission Control Software with NVIDIA GB200 NVL72 Systems

Administration Guide

Contents

Chapter 1. Overview

The NVIDIA DGX SuperPOD™ with GB200 systems 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.

Important: NVIDIA DGX SuperPOD only supports Slurm for scheduling workloads.

1.1 System Design

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

Figure 1. DGX SuperPOD logical design

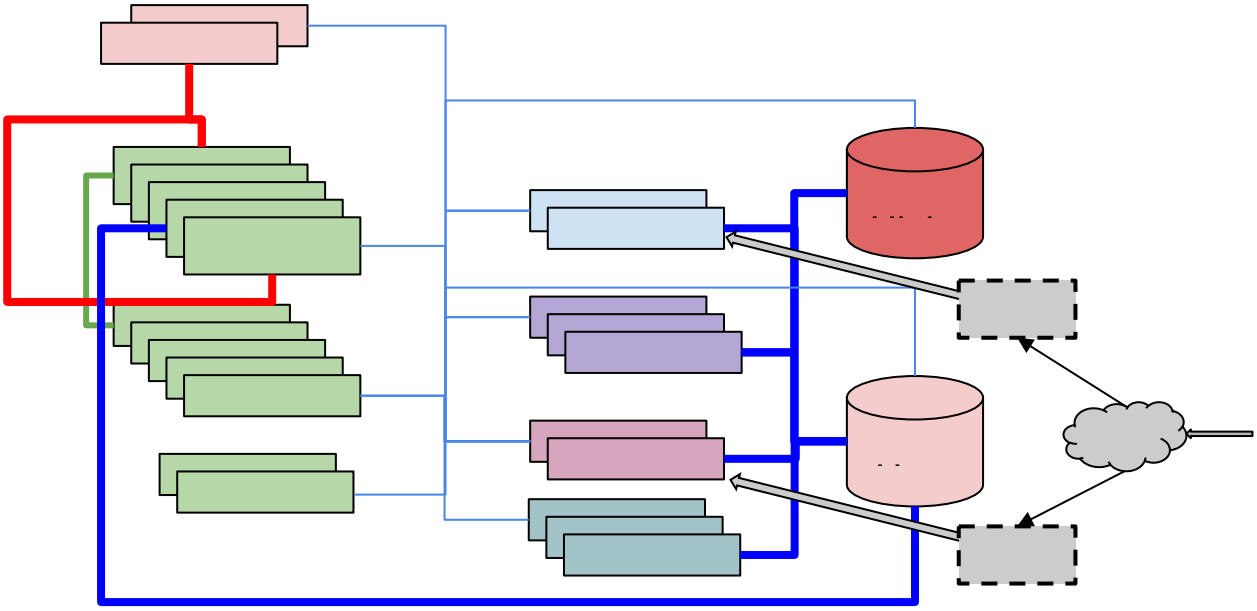


Table 1 describes the components shown in Figure 1.

Table 1. Component descriptions

DGX SuperPOD Component	Description
User Jumphost	The User Jumphost 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.
DGX Nodes / Compute Trays	The compute trays are where the user work gets done on the system. Each compute tray is considered an individual DGX node.
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 in the Management Servers section 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.
Home & High Speed 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 GB200 racks.
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 for provisioning and inband management.
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.
NVLink	NVIDIA NVLinK is a high-speed interconnect that allows multiple GPUs to communicate directly. Multi-Node NVLink is a capability enabled over an NVLink Switch network where multiple systems are interconnected to form a large GPU memory fabric also known as an NVLink Domain.

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	Head nodes serve various functions: Provisioning: centrally store and deploy OS images of the compute, management nodes, 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	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.
UFM Appliance	NVIDIA Unified Fabric Manager (UFM) for both storage and compute infiniBand fabric.
NVLink Management Software	NVLink Management Software (NMX) is an integrated platform for management and monitoring of NVLink connections

Chapter 2. Mission Control Software Stack

2.1 Cluster Management

NVIDIA Mission Control leverages NVIDIA Base Command Manager (BCM) for foundational cluster-management tasks such as provisioning compute nodes, configuring software images, assigning roles, and general cluster administration. This guide assumes that administrators have prior familiarity with BCM, including interacting via the `cmsh` command-line interface and the Base View graphical user interface (GUI).

For detailed instructions on provisioning nodes, managing software images, role assignments, and using `cmsh`, please refer directly to the relevant chapters in the BCM Administrator Guide, including:

- Base View Web GUI (Section 2.4)
- Using the Cluster Management Shell (`cmsh`) (Section 2.5)
- Cluster Management Daemon (Section 2.6)
- Provisioning and Node Management (Section 5)
- Using An External LDAP Server (Section 6.3)
- Software Image Management (Section 11)
- Day-to-day Administration (Section 14)

Administrators unfamiliar with these concepts or BCM workflows should first consult the BCM Administrator Guide before proceeding with NVIDIA Mission Control-specific configurations detailed later in this manual.

2.2 User Management

Through BCM, users and groups for the cluster are managed in a single system model. In other words, managing users and groups in BCM automatically handles changes across the cluster.

Out of the box, BCM runs its own LDAP service to help manage users and groups. This centralized LDAP service runs on the head nodes of the BCM managed cluster.

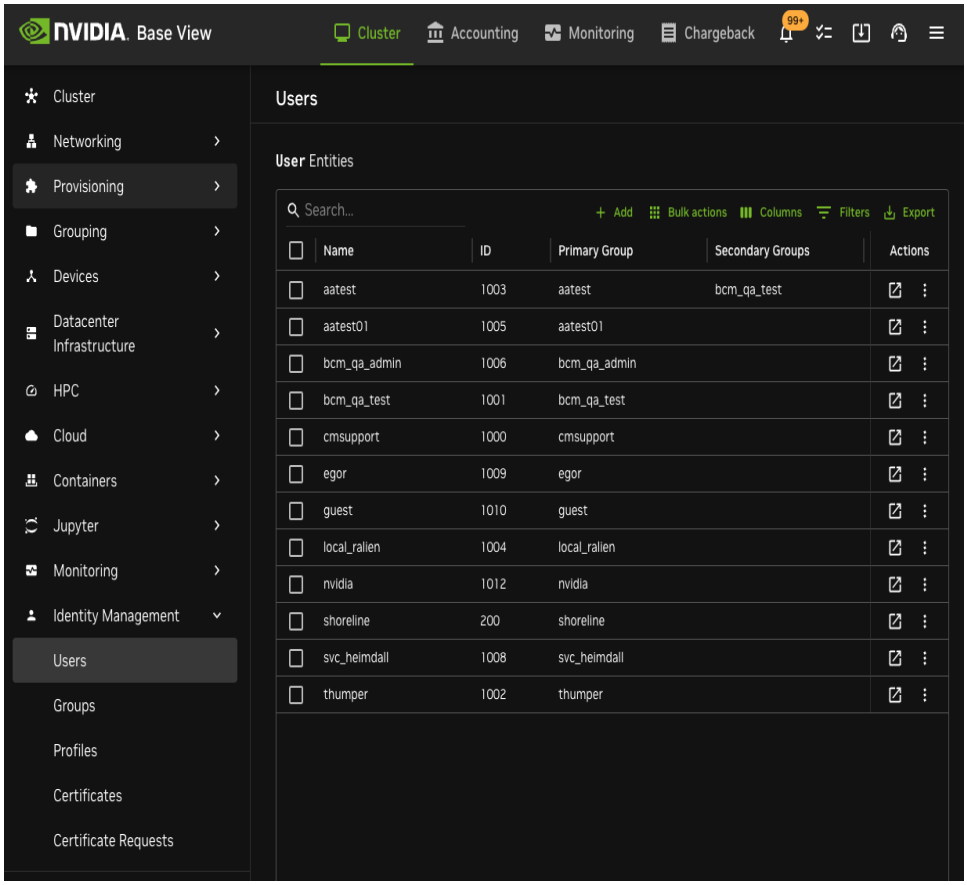
Additionally, It is possible to integrate an external LDAP server for authentication services instead of the one provided by BCM, but that is not covered here. Please refer to the BCM Admin Manual for those details.

2.2.1.1 Using the Base View GUI to Manage Users and Groups

BCM's Base View interface provides a comprehensive frontend for managing users and groups of the system.

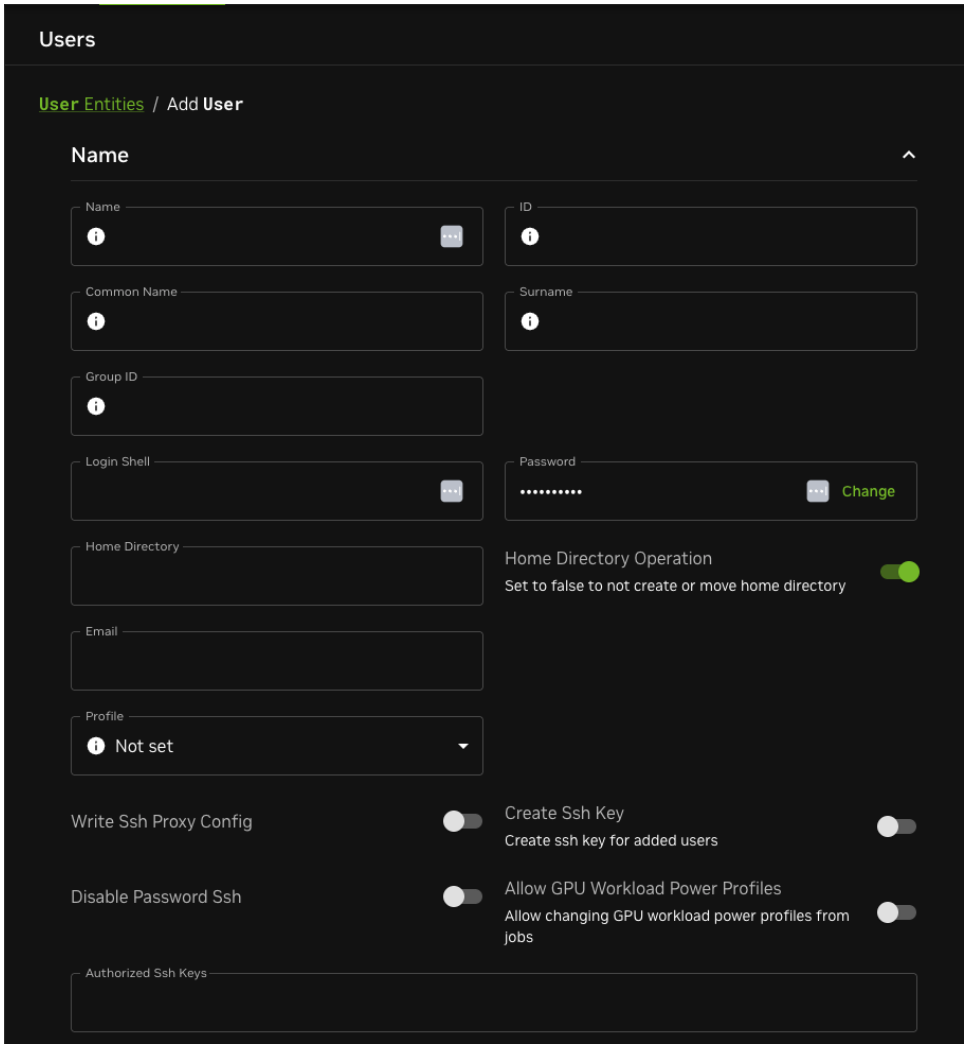
1. Within Base View, follow the navigation path *Identity-Management > Users* to manage users.

   Additionally, follow the navigation path *Identity-Management > Groups* to manage groups.



2. In the `Users` window, there are various options for managing users.

   The `ADD` button allows users to be added. When adding a user, many options are available to be set:



   These may also be edited at a later time by selecting specific users in the `Identity Management > Users` window.

3. It is important to note these points when creating users:
   - 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 is handled similarly to user management. There are clickable group objects that show up, similar to the user entries, and the management functions are the same.

2.2.1.2 Using the the BCM CLI (CMSH) to Manage Users and Groups

Using cmsh or Base View to manage users and groups will provide the same results. The only difference is that one path is a CLI and the other is a GUI.

In order to use `cmsh`, start a `cmsh` session on the BCM head node and then enter user management mode.

```
root@bcm-headnode-01:~# cmsh
[bcm-headnode-01]% user
[bcm-headnode-01->user]%
```

From here,type `help` and look at the specific `==user==` section of the output to see all the available options:

```
=============================== user ================================
add ......................... Create and use a user
append ...................... Append value(s) to user property
checkaccess ................. Check project manager access
clear ....................... Clear specific user property
clone ....................... Clone and use a user
commit ...................... Commit local changes
foreach ..................... Execute a set of commands on several users
format ...................... Modify or view current list format
get ......................... Get specific user property
list ........................ List overview
projectmanager .............. Enter project manager submode
projectmanageroverview ...... Project manager overview
range ....................... Set a range of several users to execute future commands on
refresh ..................... Revert local changes
remove ...................... Remove a user
removefrom .................. Remove value(s) from user property
set ......................... Set user properties
show ........................ Show user properties
sort ........................ Modify or view current list sort order
swap ........................ Swap uuid names of two user
undefine .................... Undefine specific user property
use ......................... Use the specified user
usedby ...................... List all entities which depend on this user
validate .................... Remote validate a user
```

Adding a user is as simple as using the `add` function:

```
[bcm-headnode-01->user]% add ophelia
[bcm-headnode-01->user*[ophelia*]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Accounts
Managees
Name                             ophelia
Primary group
Revision
```

```
Secondary groups
ID
Common name
Surname
Group ID
Login shell
Password                            < not set >
Home directory
Home directory operation            yes
Email
Profile
Write ssh proxy config              no
Create ssh key                      no
Disable password ssh                no
Allow GPU workload power profiles   no
Authorized ssh keys                 <0B>
Shadow min                          0
Shadow max                          999999
Shadow warning                      7
Shadow inactive                     0
Last change                         1969/12/31
Expiration date                     2037/12/31
Project manager                     <submode>
Notes                               <0B>
```

At this point we have not committed the user *ophelia* yet. We used the `add` function and the 'show' function. This is why you see empty fields for certain properties.

Whenever any changes are made via `cmsh`, it is important to remember to commit them or else they will not go into effect.

Now `commit` the user *ophelia* to the LDAP database and `show` the user again:

```
[bcm-headnode-01->user*[ophelia*]]% commit
[bcm-headnode-01->user[ophelia]]% show
Parameter                           Value
----------------------------------- ------------------------------------------------
Accounts
Managees
Name                                ophelia
Primary group                       1011
Revision
Secondary groups
ID                                  1007
Common name                         ophelia
Surname                             ophelia
Group ID                            1011
Login shell                         /bin/bash
Password                            ********
Home directory                      /home/ophelia
Home directory operation            yes
Email
```

```
Profile
Write ssh proxy config              no
Create ssh key                      no
Disable password ssh                no
Allow GPU workload power profiles   no
Authorized ssh keys                 <0B>
Shadow min                          0
Shadow max                          999999
Shadow warning                      7
Shadow inactive                     0
Last change                         2025/5/13
Expiration date                     2037/12/31
Project manager                     <submode>
Notes                               <0B>
```

We now see properties like we would expect. Removing a user is as simple as running `remove *Ophelia*` in the user mode prompt of `cmsh`:

```
[bcm-headnode-01->user[ophelia]]% remove ophelia
[bcm-headnode-01->user*]% commit
Successfully removed 1 Users
Successfully committed 0 Users
```

In the Slurm Workload Management section of this document we will go through how you can add users or groups to a Slurm Partition.

2.3 Integrating External LDAP

BCM installations deploy and use internal LDAP infrastructure by default. It's common however to have a requirement to use existing LDAP infrastructure to support an existing authentication model, allowing users to use existing credentials in a BCM managed environment. BCM can support this through the use of software packages like `sssd` and `krb5`.

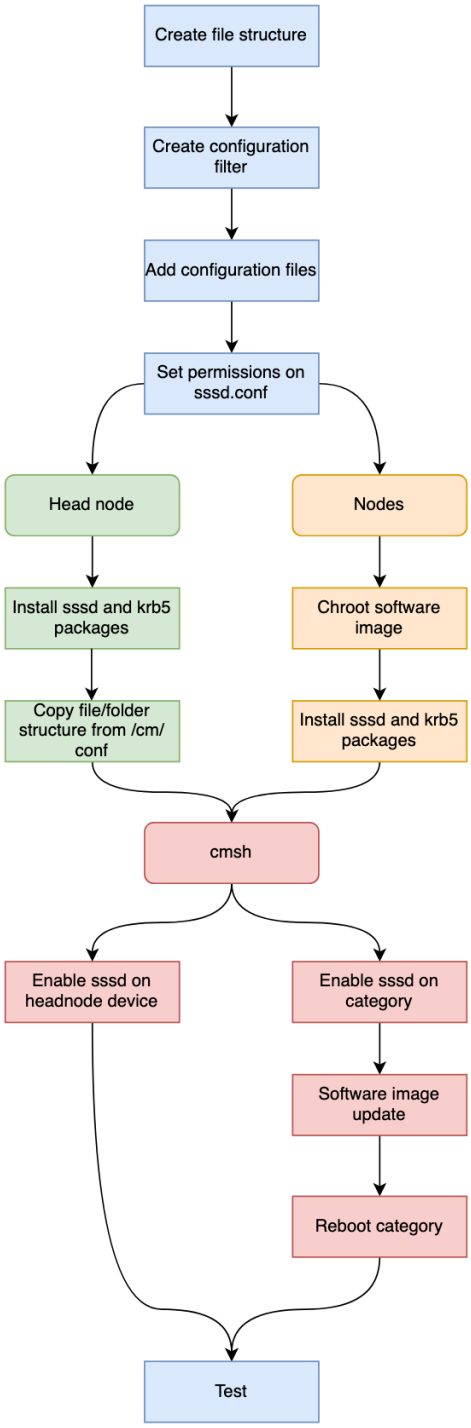
There are two options in which to support this:

4. Installation and configuration using a custom software image.
5. Installation and configuration using BCM's common configuration feature and setting of `authenticationserver` at the device or category level.

In the BCM admin manual, Section 6.3 covers the general configuration changes needed.

2.3.1 Common Configuration

2.3.1.1 Installation/configuration flow



Creating folder structures expected by BCM

```
mkdir -vp /cm/conf/all/001-cluster-ldap/etc/{pam.d,sssd}
```

```
mkdir -v /cm/conf/all/001-cluster-ldap/etc/sssd/conf.d
mkdir -vp /cm/conf/all/001-cluster-ldap/var/lib/sss/{db,pipes}
mkdir -vp /cm/conf/all/001-cluster-ldap/var/lib/sss/pipes/private
```

2.3.1.2 Configuration rule matcher

BCM uses this rule to match a category that is later defined in `cmsh`.

```
echo "category=sssd" >> /cm/conf/all/001-cluster-ldap/cm-config.match
```

2.3.1.3 File/Folder Structure

```
/cm/conf/
└── 001-cluster-ldap
    ├── cm-config.match
    ├── etc
    │   ├── krb5.conf
    │   ├── nsswitch.conf
    │   ├── pam.d
    │   │   ├── password-auth-ac
    │   │   └── system-auth-ac
    │   └── sssd
    │       └── conf.d
    │           └── sssd.conf
    └── var
        └── lib
            └── sss
                ├── db
                └── pipes
                    └── private
```

2.3.1.4 File permissions

With these all set, we need to fix permissions for sssd.conf, otherwise this will fail when starting the service.

```
chown root:root /cm/conf/all/001-cluster-ldap/etc/sssd/conf.d/sssd.conf
chmod 600 /cm/conf/all/001-cluster-ldap/etc/sssd/conf.d/sssd.conf
```

2.3.1.5 Package installation

2.3.1.5.1 Head node

```
apt-get update; DEBIAN_FRONTEND=noninteractive apt install -y sssd sssd-tools krb5-user
```

2.3.1.5.2 Nodes

```
cm-chroot-sw-img /cm/images/sssd-image
apt-get update; DEBIAN_FRONTEND=noninteractive apt install -y sssd sssd-tools krb5-user
systemctl enable sssd
```

2.3.1.6 Enable sssd via BCM

2.3.1.6.1 Head node

Copying the existing file structure and configuration files to their respective locations and finally enabling `sssd`.

```
rsync -av /cm/conf/all/001-cluster-ldap/etc /
cmsh -c "device; foreach -t headnode (set authenticationservice sssd; commit)"
```

2.3.1.6.2 Nodes

We'll assign devices in our cluster to a category where we'll enable the `sssd` service.

1. Set categories on devices

Set categories on devices

```
cmsh -c "device; foreach -n node004..node006 (set category sssd; commit)"
```

2. Apply `authenticationservice` to category.

```
cmsh -c "category; use sssd; set authenticationservice sssd; commit"
```

3. Software image update.

```
cmsh -c "device; imageupdate -w -c sssd"
```

4. Reboot category.

```
cmsh -c "device; foreach -c sssd (reboot)"
```

2.3.1.7 Validate external LDAP is working

1. Using pdsh to run getent against an external LDAP user.

```
pdsh -g category=sssd 'getent passwd janedoe' | dshbak -c
----------------
node[004-006]
----------------
janedoe:*:86753:30:Jane Doe:/home/janedoe:/bin/bash
```

2. Using ldapsearch to query membership of an external LDAP user

```
ldapsearch -xLLL -H ldap://ldap.company.com -b "uid=janedoe,ou=people,dc=comapny,dc=com" -x
memberOf
dn: uid=janedoe,ou=people,dc=company,dc=com
memberOf: cn=colorado-engineer,ou=groups,dc=comapny,dc=com
memberOf: cn=colorado-admin,ou=groups,dc=comapny,dc=com
```

Chapter 3. Node and Category Management

When working with workload management systems in BCM, it is important to understand node categories. It is typically more efficient to assign BCM roles using categories or configuration overlays since that approach makes it easier to manage many nodes at once, in a common category.

3.1 Introduction to Categories

A node category is a group of regular nodes that share the same configuration. Ultimately, node categories provide an efficient way to manage a pool of compute nodes. Nodes are typically divided into categories based on hardware specifications and their specific purpose.

Here is a snapshot of what a list of categories could look like on a real system:

```
[bcm-headnode-01]% category

[bcm-headnode-01->category]% ls
Name (key)                  Software image                              Nodes
--------------------------- ------------------------------------------ --------
badler-gb200                baseos7.1rc2-image-arm64-2025.04.30-2      0
badler-gb200-lustre         baseos7.1rc2-image-arm64-2025.04.30-2      0
cvt-x86                     cvt-image-x86_64                           0
default-ubuntu2404-aarch64  default-image                              0
default-ubuntu2404-x86_64   default-image-ubuntu2404-x86_64            0
dgx-gb200                   baseos7-image-arm64-03-06-2025             72
gtc-gb200                   gtc-570.124.06-image-20250313              0
hwqa-gb200                  hwqa-570.133.20-image-20250422             18
k8s-ctrl-node               k8s-ctrl-image                             3
nmx-m                       NMX-M-image                                3
perf-team                   perf-team-baseos7-image-arm64-20250505     35
slogin                      slogin-image                               2
spare-aarch64               spare-image-aarch64                        0
spare-x86                   spare-image-x86_64                         0
swqa-570.124.06             swqa-570.124.06-image-20250324             0
swqa-570.133.20             swqa-570.133.20-image-20250414             0
swqa-575.51.03              swqa-575.51.03-image-20250418              0
```

```
test-baseos7.1rc2           baseos7.1rc2v3-image-arm64-05-07-2025.orig  1
```

As you can see, we have a list of categories in the left column, the software image they use in the middle, and how many nodes are using that category in the right column. One thing worth noting is that categories can share software images. It is not a one-to-one mapping between categories and software images.

The `help` command can be very useful in seeing what actions you can take when in a certain module of `cmsh`, here is what it looks like for the `category` module:

```
root@bcm-headnode-01:~# cmsh
[bcm-headnode-01]% category
[bcm-headnode-01]% help
...
============================== category ==============================
add ......................... Create and use a category
append ...................... Append value(s) to category property
biossettings ................ Enter BIOS settings mode
bmcsettings ................. Enter BMC settings setup mode
clear ....................... Clear specific category property
clone ....................... Clone and use a category
commit ...................... Commit local changes
createramdisk ............... Create a ramdisk for a category
dpusettings ................. Enter DPU settings setup mode
foreach ..................... Execute a set of commands on several categorys
format ...................... Modify or view current list format
fsexports ................... Enter fsexport setup mode
fsmounts .................... Enter fsmount setup mode
get ......................... Get specific category property
gpusettings ................. Enter GPU settings setup mode
kernelmodules ............... Enter kernel module mode
list ........................ List overview
listnodes ................... List nodes in a category
range ....................... Set a range of several categorys to execute future commands on
refresh ..................... Revert local changes
remove ...................... Remove a category
removefrom .................. Remove value(s) from category property
roles ....................... Enter role setup mode
selinuxsettings ............. Enter SELinux settings setup mode
services .................... Enter service config mode
set ......................... Set category properties
show ........................ Show category properties
sort ........................ Modify or view current list sort order
staticroutes ................ Enter staticroute setup mode
swap ........................ Swap uuid names of two category
undefine .................... Undefine specific category property
use ......................... Use the specified category
usedby ...................... List all entities which depend on this category
validate .................... Remote validate a category
ztpsettings ................. Enter ZTP settings setup mode
```

In the above output, we can see that the `category` module within `cmsh` provides us a range of actions. Notably you can add or clone categories. Additionally, you can control all sorts of things from the bios, kernel modules, disks, networks, and filesystems.

3.2 Overview of BaseOS for NMC DGX Systems

NMC provides an enhanced BaseOS image that builds upon the standard DGX OS. This BaseOS image comes ready to run on DGX nodes within the NMC SuperPOD environment. It is based on Ubuntu OS. These enhancements include:

- BCM CMDaemon process for command and control from BCM head nodes.
- Node Telemetry services.
- Autonomous Hardware Recovery agent.

In addition, BaseOS includes NVIDIA services you would expect in other DGX OS based environments:

- Datacenter GPU Manager (DCGM)
- NVIDIA GPU Driver
- Optimized Kernel
- NVIDIA System Management (NVSM)
- DOCA OFED

It is always good to clone the software images you have for backups. This can be done with these `cmsh` commands, within the `softwareimage` module:

```
[bcm-headnode-01->softwareimage]% clone <source-image-name> <backup-image-name>
```

Additionally, it is good to `show` a software image to see what properties it has. Here is an example that enters `cmsh -> softwareimage` module, lists software images, selects one, and shows its properties:

```
[bcm-headnode-01]% softwareimage

[bcm-headnode-01->softwareimage]% ls
Name (key)                                  Path (key)
Kernel version         Nodes
------------------------------------------- ------------------------------------------------------ --
------------------- --------
NMX-M-image                                 /cm/images/NMX-M-image
6.8.0-51-generic       3
baseos7-image-arm64                         /cm/images/baseos7-image-arm64
6.8.0-1021-nvidia-64k  0
baseos7-image-arm64-03-06-2025              /cm/images/baseos7-image-arm64-03-06-2025
6.8.0-1021-nvidia-64k  72
```

```
baseos7-image-arm64-03-06-2025-backup       /cm/images/baseos7-image-arm64-03-06-2025-backup
0
baseos7-image-arm64-backup-04-23-2025       /cm/images/baseos7-image-arm64-backup-04-23-2025
6.8.0-1021-nvidia-64k  0
baseos7.1rc2-image-arm64-2025.04.25         /cm/images/baseos7.1rc2-image-arm64-2025.04.25
6.8.0-1025-nvidia-64k  0
baseos7.1rc2-image-arm64-2025.04.25-backup  /cm/images/baseos7.1rc2-image-arm64-2025.04.25-backup
6.8.0-1025-nvidia-64k  0
baseos7.1rc2-image-arm64-2025.04.29-backup  /cm/images/baseos7.1rc2-image-arm64-2025.04.29-backup
6.8.0-1025-nvidia-64k  0
baseos7.1rc2-image-arm64-2025.04.30-2       /cm/images/baseos7.1rc2-image-arm64-2025.04.30-2
6.8.0-1025-nvidia-64k  0
baseos7.1rc2v2-image-arm64-2025.05.02-1     /cm/images/baseos7.1rc2v2-image-arm64-2025.05.02-1
6.8.0-1025-nvidia-64k  0
baseos7.1rc2v3-image-arm64-05-07-2025.orig  /cm/images/baseos7.1rc2v3-image-arm64-05-07-2025.orig
6.8.0-1025-nvidia-64k  1
cvt-image-x86_64                            /cm/images/cvt-image-x86_64
6.8.0-51-generic       0
default-image                               /cm/images/default-image
6.8.0-51-generic-64k   0
default-image-ubuntu2404-x86_64             /cm/images/default-image-ubuntu2404-x86_64
6.8.0-51-generic       0
gtc-570.124.06-image-20250313               /cm/images/gtc-570.124.06-image-20250313
6.8.0-1021-nvidia-64k  0
hwqa-570.133.20-image-20250422              /cm/images/hwqa-570.133.20-image-20250422
6.8.0-1021-nvidia-64k  18
k8s-ctrl-image                              /cm/images/k8s-ctrl-image
6.8.0-51-generic-64k   3
perf-team-baseos7-image-arm64-20250505      /cm/images/perf-team-baseos7-image-arm64-20250505
6.8.0-1021-nvidia-64k  35
slogin-image                                /cm/images/slogin-image
6.8.0-51-generic-64k   2
spare-image-aarch64                         /cm/images/spare-image-aarch64
6.8.0-51-generic-64k   0
spare-image-x86_64                          /cm/images/spare-image-x86_64
6.8.0-51-generic       0
swqa-570.124.06-image-20250324              /cm/images/swqa-570.124.06-image-20250324
6.8.0-1021-nvidia-64k  0
swqa-570.133.20-image-20250414              /cm/images/swqa-570.133.20-image-20250414
6.8.0-1021-nvidia-64k  0
swqa-575.51.03-image-20250418               /cm/images/swqa-575.51.03-image-20250418
6.8.0-1021-nvidia-64k  0

[bcm-headnode-01->softwareimage]% use baseos7-image-arm64

[bcm-headnode-01->softwareimage[baseos7-image-arm64]]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Name                           baseos7-image-arm64
Nodes                          0
Revision
```

```
Path                            /cm/images/baseos7-image-arm64
Creation time                   Fri, 28 Feb 2025 14:15:12 PST
Kernel version                  6.8.0-1021-nvidia-64k
Kernel parameters               nouveau.modeset=0
Kernel output console           tty0
Kernel modules                  <50 in submode>
Enable SOL                      no
SOL Port                        ttyS1
SOL Speed                       115200
SOL Flow Control                yes
FSPart                          /cm/images/baseos7-image-arm64
Boot FSPart                     /cm/images/baseos7-image-arm64/boot
Notes                           <0B>
```

3.3 Category for DGX Compute Nodes Using BaseOS

This section assumes you have access to the NMC BaseOS software image for DGX nodes.

Here is how you can create a category and set a software image to be associated with a particular category:

```
[bcm-headnode-01]% category
[bcm-headnode-01->category]% add dgx-gb200
[bcm-headnode-01->category*[dgx-gb200*]]% set softwareimage baseos7-image-arm64
[bcm-headnode-01->category*[dgx-gb200*]]% commit
```

This new `dgx-gb200` category now has the `base0s7-image-arm64` associated with it. All nodes using this category will boot into this software image.

The deployment process for nodes revolves around the PXE boot process that allows compute nodes to boot over the network instead of from local disks. It is important to configure this properly for the category.

Here is an example of adjusting properties associated with PXE at the category level:

```
[bcm-headnode-01->category*[dgx-gb200*]]% set kerneloutputconsole tty0
[bcm-headnode-01->category*[dgx-gb200*]]% set bootloaderprotocol http
[bcm-headnode-01->category*[dgx-gb200*]]% set bootloader grub
[bcm-headnode-01->category*[dgx-gb200*]]% commit
```

Here is what you might see when you `show` the `dgx-gb200` category:

```
[bcm-headnode-01->category[dgx-gb200]]% show
Parameter                       Value
------------------------------- ------------------------------------------------------------------------
------------------------------------------------------------------------
Name                            dgx-gb200
Nodes                           72
Revision
Use exclusively for
```

```
User node login                  ALWAYS
Data node                        no
Allow networking restart         no
Default gateway                  7.241.16.1 (network: internalnet)
Default gateway metric           0
Name servers
Time servers
Search domain
Management network               internalnet
Version config files             no
FIPS                             no
IO scheduler
Authentication service           sssd
ZTP Settings                     <submode>
Install mode                     AUTO
New node install mode            FULL
Node installer disk              no
Install boot record              no
Initialize script                <0B>
Finalize script                  <1.58KiB>
Exclude list full install        <234B>
Exclude list sync install        <1.38KiB>
Exclude list update              <4.3KiB>
Exclude list grab                <1.70KiB>
Exclude list grab new            <1.34KiB>
Exclude list manipulate script   <0B>
Disk setup                       <1.78KiB>
Hardware RAID configuration      <0B>
Software image                   baseos7-image-arm64-03-06-2025
Kernel version                   6.8.0-1021-nvidia-64k (software image:baseos7-image-arm64-03-06-
2025)
Kernel parameters                nouveau.modeset=0 systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller (software image:baseos7-image-arm64-03-06-2025)
Kernel output console            tty0 (software image:baseos7-image-arm64-03-06-2025)
Kernel modules                   51 (software image:baseos7-image-arm64-03-06-2025)
Boot loader                      grub
Boot loader protocol             HTTP
Boot loader file
Filesystem mounts                <6 in submode>
Static routes                    <0 in submode>
Roles                            <0 in submode>
GPU Settings                     <0 in submode>
Filesystem exports               <0 in submode>
Services                         <0 in submode>
BMC Settings                     <submode>
SELinux Settings                 <submode>
DPU Settings                     <submode>
Access Settings                  <submode>
Time zone                        America/Los_Angeles (base)
BIOS setup                       <0B>
```

```
Notes                                <0B>
```

3.4 Slurm Roles and Configuration Overlays

When working with workload management systems in BCM, it is important to understand node categories. It is typically more efficient to assign BCM roles using categories or configuration overlays since that approach makes it easier to manage many nodes at once, in a common category.

In this section, we will talk about categories and roles in the context of setting up compute nodes for Slurm.

By default, the `cm-wlm-setup` tool creates some configuration overlays and assigns roles to the configuration overlays accordingly. Here is what `cm-wlm-setup` provides for Slurm:

```
[basecm11->configurationoverlay]% list
Name (key)           Priority   All head nodes Nodes            Categories       Roles
-------------------- ---------- -------------- ---------------- ---------------- ----------------
slurm-accounting     500        yes                             slurmaccounting
slurm-client         500        no                              default          slurmclient
slurm-server         500        yes                             slurmserver
slurm-submit         500        no                              default          slurmsubmit
wlm-headnode-submit  600        yes                             slurmsubmit
```

In this example, you can see that the `default` category takes on the `slurmclient` and `slurmsubmit` role (far right column) by being associated with the `slurm-client` and `slurm-submit` configuration overlay.

The `wlm-headnode-submit` configuration overlay is a special overlay. It is applied only to the head node and enables the head node to have the submit role for a given workload manager.

3.5 Integrating categories with Slurm Roles and Configuration Overlays

Here we will go through some basics in order to associate the `dgx-gb200` category with the proper Slurm Configuration Overlays and Roles.

1. Enter `cmsh` and access the `configurationoverlay` module:

```
root@bcm-headnode-01:~# cmsh

[bcm-headnode-01]% configurationoverlay

[bcm-headnode-01->configurationoverlay]% help
```

```
..

======================== configurationoverlay ========================
add ............................ Create and use a configurationoverlay
append ......................... Append value(s) to configurationoverlay property
clear .......................... Clear specific configurationoverlay property
clone .......................... Clone and use a configurationoverlay
commit ......................... Commit local changes
customizations ................. Enter customizations mode
customizationsdisable ......... Disable all customizations for label
customizationsenable .......... Enable all customizations for label
customizationsoverview ........ Show table of all customizations applied to individual nodes
foreach ........................ Execute a set of commands on several configurationoverlays
format ......................... Modify or view current list format
get ............................ Get specific configurationoverlay property
list ........................... List overview
movenodes ...................... Move nodes from one overlay to another
range .......................... Set a range of several configurationoverlays to execute future
commands on
refresh ........................ Revert local changes
remove ......................... Remove a configurationoverlay
removefrom ..................... Remove value(s) from configurationoverlay property
roles .......................... Enter role setup mode
set ............................ Set configurationoverlay properties
show ........................... Show configurationoverlay properties
sort ........................... Modify or view current list sort order
swap ........................... Swap uuid names of two configurationoverlay
undefine ....................... Undefine specific configurationoverlay property
use ............................ Use the specified configurationoverlay
usedby ......................... List all entities which depend on this configurationoverlay
validate ....................... Remote validate a configurationoverlay

[bcm-headnode-01->configurationoverlay] use slurm-client

[bcm-headnode-01->configurationoverlay [slurm-client]] ls

#'ls' will show you how things currently look. Output is leftout here due to readability.

[bcm-headnode-01->configurationoverlay[slurm-client*]]% set categories dgx-gb200


[bcm-headnode-01->configurationoverlay[slurm-client*]] ls

# Do another 'ls' to see changes

[bcm-headnode-01->configurationoverlay[slurm-client*]] commit
```

2. Repeat the above to also add the `dgc-gb200` category to the `slurm-submit` Configuration Overlay and Role.

Chapter 4. Slurm Workload Management

NVIDIA Mission Control utilizes Slurm, an open-source workload manager developed by SchedMD, to orchestrate and schedule jobs across the DGX SuperPOD cluster. This guide assumes that administrators are already familiar with basic Slurm operations and configuration.

Detailed instructions on managing Slurm queues, submitting and monitoring jobs, managing node states (`drain`, `resume`), and configuring job `prolog`/`epilog` scripts can be found in the following resources:

- Base Command Manager (BCM) Administrator Guide:
  - Managing Slurm and Workload Integration (Section 7)
  - Slurm Prolog and Epilog Scripts (Section 7.3.4)
- SchedMD Slurm Documentation:
  - Slurm Quick Start User Guide
  - Slurm Quick Start Administrator Guide
  - Troubleshooting and FAQ

4.1 Slurm and IMEX

New to NVIDIA GB200 system, NVIDIA Internode Memory Exchange Service (IMEX) is a secure service that facilitates the mapping of GPU memory over NVLink between the GPUs in an NVLIink domain. BCM can enable the IMEX daemon either globally, or per job for Slurm.

It is recommended to implement IMEX daemon per-job for Slurm. Running the IMEX daemon per job has the advantage that one user running a job cannot read the memory of a job run by another user on another node.

Configuring per job means that IMEX runs just before the job starts, and that the service runs on only the nodes that need it.

To run IMEX per job, any global IMEX setting must first be cleared away. This can be done by first:

1. Removing the `nvidia-imex` service entirely from `CMDaemon`.
2. Stopping the service on the compute nodes, for example with `pdssh` or `pdexec`, or simply carrying out a reboot The value of `imex` in the `slurmclient` role can then be set:

```
[root@basecm11 ~]# cmsh
[basecm11]% configurationoverlay roles slurm-client-gpu
[basecm11->configurationoverlay[slurm-client-gpu]->roles]% use slurmclient
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]]% set imex yes
[basecm11->configurationoverlay*[slurm-client-gpu*]->roles*[slurmclient*]]% commit
[basecm11->configurationoverlay[slurm-client-gpu]->roles[slurmclient]]%
```

3. Committing the above IMEX setting configures Slurm `prolog` and `epilog` scripts in the backend as follows:
   a. The `prolog` script configures the IMEX daemon with the nodes allocated to the GPU job and starts the IMEX daemon on the node
   b. The `epilog` cleans up the configuration and stops the IMEX daemon on the node.

4.2 Slurm Topology Block Support

Slurm can be configured to support topology-aware resource allocation to optimize job performance. More information about topologies in Slurm is available here.

Previous version(s) of BCM only supported the topology/tree option for Slurm configuration. BCM 11 introduces support for topology/block. This is required for BCM to enable NVLINK-aware scheduling of Slurm jobs for GB200 systems.

The `topology.conf` file describes the cluster's network topology for optimized job resource allocation. BCM 11 is able to generate different types of `topology.conf` to customize the Slurm behavior of the topology plugins. More information about `topology.conf` is available here.

The location of topology.conf in BCM head-node(?) is:
`/cm/shared/apps/slurm/etc/<CLUSTER_NAME>/topology.conf`.

To support topology/block, CMD introduces new entities to customize Slurm in cmsh and Base View: `SlurmTreeTopologySettings` and `SlurmBlockTopologySettings` become both valid objects for the parent `SlurmTopologySettings` field.

The new commands to configure the `SlurmBlockTopologySettings` are available in `cmsh` and Base View in the `wlm` mode/section:

```
[basecm11->wlm[slurm]]% topologysettings
[basecm11->wlm[slurm]->topologysettings]% show
Parameter                      Value
------------------------------ ------------------------------------------
Topology plugin                None
Topology source                None
Parameters                     <submode>
```

```
Tree settings                    <submode>
Block settings                   <submode>
Topograph settings               <submode>
```

From the `topologysettings` menu a few parameters `submodes` are available:

1. Topology plugin: parameter that defines which Slurm topology plugin will be used (tree or block). By default none is set and `topology.conf` is not generated.
2. Topology source: parameter that defines where BCM takes information for the topology construction. Values: internal (taken from switch/racks information), `topograph`: taken from `topograph` service.
3. Parameters: `submode` that allows to tune Slurm parameters related to all topology plugins.
4. Tree settings: `submode` that allows configure settings of tree topology.
5. Block settings: `submode` that allows configure settings of block topology.

If `block` is chosen as the topology plugin, additional options are available for the administrator (e.g. block size, type of racks, etc) in block settings submode. The administrator can manually define Slurm blocks leveraging information on the cluster layout, such as full GB200 racks (1x72 configuration), half GB200 racks (2x36 configuration), or generic *nodegroups*.

4.3 Slurm Partitioning with NVLink

A Slurm partition is a distinct job queue that groups compute nodes together and enables the ability to set specific resource constraints and limits for those nodes.

Table 3 highlights example partitions (or job queues).

Table 3. Slurm partitions

Partition Name	Description	Node Types / Count	Priority	Preemptable	Default Runlimit	Max Runliimit	Resource Limits / QoS	Allowed Account
Batch (default)	Default partition, suitable for most jobs	All nodes	Medium	No	30 minutes	4 hours	Set max nodes per job	All
Batch_short	Same nodes as batch, but with higher priority and limited to 2 hours max runtime	All nodes	High	No	30 minutes	2 hours	No more than 4 nodes per user and no more than 20 nodes across all users	All
Batch_long	Same nodes as batch, but with lower priority and 8 hour max runtime	All nodes	Low	No	30 minutes	8 hours	No user should be able to take more than 10% of nodes in partition	Approval required
Batch_large	Same nodes as batch. Partition for very large jobs (> than half the cluster)	All nodes	High+++	No	30 minutes	4 hours	Prioritizes large jobs on the system	Approval required
Interactive	High priority queue for interactive jobs, useful for development	All nodes + 6 dedicated nodes	High+	No	30 minutes	4 hours	Max 2 nodes per job Max 2 nodes per user	ALL
Backfill	Least restrictions, low priority, preemtable	All nodes	Low	Yes	30 minutes	168 hours (7 days)	preempted jobs are set to requeue automatically	ALL
admin	Partition to debug only	All nodes, including dedicated interactive nodes	High++	No	30 minutes	8 hours	admin only	admin

Within BCM, you can create and manage Slurm partitions through `cmsh`.

```
[root@basecm11 ~]# cmsh
[basecm11]% wlm
[basecm11->wlm[slurm]]% jobqueue
[basecm11->wlm[slurm]->jobqueue]% help
...
============================== jobqueue ==============================
add ......................... Create and use a jobqueue
append ...................... Append value(s) to jobqueue property
clear ....................... Clear specific jobqueue property
clone ....................... Clone and use a jobqueue
commit ...................... Commit local changes
foreach ..................... Execute a set of commands on several jobqueues
format ...................... Modify or view current list format
get ......................... Get specific jobqueue property
jobqueue .................... Enter job queues mode
list ........................ List overview
range ....................... Set a range of several jobqueues to execute future commands on
refresh ..................... Revert local changes
remove ...................... Remove a jobqueue
removefrom .................. Remove value(s) from jobqueue property
set ......................... Set jobqueue properties
show ........................ Show jobqueue properties
sort ........................ Modify or view current list sort order
statistics .................. Get job queue statistics.
swap ........................ Swap uuid names of two jobqueue
undefine .................... Undefine specific jobqueue property
use ......................... Use the specified jobqueue
usedby ...................... List all entities which depend on this jobqueue
validate .................... Remote validate a jobqueue

[basecm11->wlm[slurm]->jobqueue]% add batch_long

[basecm11->wlm[slurm]->jobqueue[batch_long]% show

Parameter                      Value
------------------------------ ------------------------------------------------------------------------
------
Name                           batch_long
Revision
Type                           Slurm
WlmCluster                     slurm
Ordering                       0
Default                        no
Hidden                         no
Min nodes                      1
Max nodes
Default time                   UNLIMITED
Max time                       12:00:00
```

```
Priority Job Factor             1
Priority Tier                   1
OverSubscribe                   exclusive
Alternate
Grace time                      0
Preemption mode                 OFF
Require reservation             NO
Select Type Parameters
LLN                             no
TRES Billing Weights
Alloc nodes
CPU bindings                    None
QOS
Default memory per CPU          UNLIMITED
Max memory per CPU              UNLIMITED
Default memory per Node         UNLIMITED
Max memory per Node             UNLIMITED
Max CPUs per node               UNLIMITED
Default memory per GPU          UNLIMITED
Default CPU per GPU             UNLIMITED
Disable root                    no
Root only                       no
Allow groups
Allow accounts
Allow QOS                       ALL
Deny Accounts
Deny QOS
ExclusiveUser                   no
Queue State                     None
Nodesets
Overlays
Categories
Nodegroups
Compute nodes
All nodes
Options
```

At this point, you can configure the `batch_long` partition with various resource constraints and limits to fit the use case need.

- Example that sets compute nodes for `batch_long`:

```
[basecm11->wlm[slurm]->jobqueue[batch_long]% set computenodes <computenode names>
```

- Example that adds a user to a jobqueue (you can also append users/groups or add a list separated by comma):

```
[basecm11->wlm[slurm]->jobqueue[batch_long]% set allowaccounts <user_id>
[basecm11->wlm[slurm]->jobqueue[batch_long]% set allowgroups <group_id>
```

When designing partitions, it is important to consider the underlying NVLink network. For instance, each NVL72 rack is by default an NVLink partition that

includes 72 GPUs connected through NVLink. Ideally, compute nodes in a Slurm partition also share the same NVLink partition.

The exception is when you need to combine two NVL72 racks (or NVLink Domains) into a single Slurm partition. This is necessary for large training jobs that require more resources than a single NVL72 rack provides.

In this example scenario, where two NVL72 racks exists, a combination of Slurm partitions across both racks could be configured with this logic:

- *Flex-R1*: flexible partition, for small scale testing, that allows max of two nodes for max of two hours on `Rack 1`.
- *Flex-R2*: flexible partition, for small scale testing, that allows max of two nodes for max of two hours on `Rack 2`.
- *Benchmark-R1*: Allows full rack, 18 node, jobs with a max time of twelve hours on `Rack 1`.
- Benchmark-R2: Allows full rack, 18 node, jobs with a max time of twelve hours on `Rack 2`.
- *Benchmark-Combined*: Allows 36 node jobs across *both racks* with a max time of twelve hours

4.4 Common Slurm Management Tasks

Creating a Slurm partition:

- Within BCM, you can create and manage slurm partitions through `cmsh`:

```
[root@basecm11 ~]# cmsh


[basecm11]% wlm

[basecm11->wlm[slurm]]% jobqueue


[basecm11->wlm[slurm]->jobqueue]% help
...

============================== jobqueue ==============================
add ......................... Create and use a jobqueue
append ...................... Append value(s) to jobqueue property
clear ....................... Clear specific jobqueue property
clone ....................... Clone and use a jobqueue
commit ...................... Commit local changes
foreach ..................... Execute a set of commands on several jobqueues
format ...................... Modify or view current list format
get ......................... Get specific jobqueue property
jobqueue .................... Enter job queues mode
```

```
list ......................... List overview
range ........................ Set a range of several jobqueues to execute future commands on
refresh ...................... Revert local changes
remove ....................... Remove a jobqueue
removefrom ................... Remove value(s) from jobqueue property
set .......................... Set jobqueue properties
show ......................... Show jobqueue properties
sort ......................... Modify or view current list sort order
statistics ................... Get job queue statistics.
swap ......................... Swap uuid names of two jobqueue
undefine ..................... Undefine specific jobqueue property
use .......................... Use the specified jobqueue
usedby ....................... List all entities which depend on this jobqueue
validate ..................... Remote validate a jobqueue

[basecm11->wlm[slurm]->jobqueue]% add batch_long

[basecm11->wlm[slurm]->jobqueue[batch_long]% show

Parameter                      Value
------------------------------ ------------------------------------------------------------------
----------
Name                           batch_long
Revision
Type                           Slurm
WlmCluster                     slurm
Ordering                       0
Default                        no
Hidden                         no
Min nodes                      1
Max nodes
Default time                   UNLIMITED
Max time                       12:00:00
Priority Job Factor            1
Priority Tier                  1
OverSubscribe                  exclusive
Alternate
Grace time                     0
Preemption mode                OFF
Require reservation            NO
Select Type Parameters
LLN                            no
TRES Billing Weights
Alloc nodes
CPU bindings                   None
QOS
Default memory per CPU         UNLIMITED
Max memory per CPU             UNLIMITED
Default memory per Node        UNLIMITED
Max memory per Node            UNLIMITED
```

```
Max CPUs per node                UNLIMITED
Default memory per GPU           UNLIMITED
Default CPU per GPU              UNLIMITED
Disable root                     no
Root only                        no
Allow groups
Allow accounts
Allow QOS                        ALL
Deny Accounts
Deny QOS
ExclusiveUser                    no
Queue State                      None
Nodesets
Overlays
Categories
Nodegroups
Compute nodes
All nodes
Options
```

- Any of the properties above can be set through `cmsh` -> `wlm`. The below sets the compute nodes `b08-p1-dgx-08-c[01-18]` to be allocated from the `batch_long` partition.

```
[basecm11->wlm[slurm]->jobqueue[batch_long]% set computenodes b08-p1-dgx-08-c[01-18]
```

- Draining nodes can be done through the common Slurm CLI tool `scontrol`.

```
scontrol update nodename=a06-p1-dgx-02-c[01-04,06-11,13-14,16-18] state=drain
reason="maintenance"
```

- Resuming nodes can be done similarly:

```
scontrol update nodename=a06-p1-dgx-02-c[01-04,06-11,13-14,16-18] state=resume
```

4.4.1 Adding or Appending to a jobqueue

- Adding a user or group to a `jobqueue` (partition):

```
[basecm11->wlm[slurm]->jobqueue[batch_long]% set allowaccounts <user_id>
[basecm11->wlm[slurm]->jobqueue[batch_long]% set allowgroups <group_id>
```

- Append a user or group to a `jobqueue` (partition):

```
[basecm11->wlm[slurm]->jobqueue[batch_long]% append allowaccounts <user_id>
[basecm11->wlm[slurm]->jobqueue[batch_long]% append allowgroups <group_id>
```

4.5 Example: Running Interactive Slurm Job to confirm IMEX setup

NOTE: This example assumes IMEX is configured to run per-job and that you have access to the slurm-login node.

1. Check Slurm partitions and node availability with `sinfo`.

```
root@a03-p1-aps-arm-01:~# sinfo
```

```
PARTITION          AVAIL  TIMELIMIT  NODES  STATE NODELIST
bcm_qa_test           up   infinite     18  alloc b08-p1-dgx-08-c[01-18]
benchmark-a07         up   12:00:00      2  drain a07-p1-dgx-03-c[06,12]
benchmark-a07         up   12:00:00     15   idle a07-p1-dgx-03-c[01-05,07,09-11,13-18]
benchmark-b05         up   12:00:00     18   idle b05-p1-dgx-05-c[01-18]
benchmark-combined    up   12:00:00      2  drain a07-p1-dgx-03-c[06,12]
benchmark-combined    up   12:00:00     33   idle a07-p1-dgx-03-c[01-05,07,09-11,13-18],b05-p1-
dgx-05-c[01-18]
computex_demo         up   infinite      1  down* b06-p1-dgx-06-c01
computex_demo         up   infinite     16  alloc b06-p1-dgx-06-c[10-17],b07-p1-dgx-07-c[01-08]
computex_demo         up   infinite     13   idle b06-p1-dgx-06-c[02-03,18],b07-p1-dgx-07-c[09-
18]
defq*                 up   infinite      4 drain* a05-p1-dgx-01-c[01,07,16],b06-p1-dgx-06-c04
defq*                 up   infinite      4  down* a05-p1-dgx-01-c10,b06-p1-dgx-06-c[01,06-07]
defq*                 up   infinite      3  drain a07-p1-dgx-03-c[06,12],b06-p1-dgx-06-c05
defq*                 up   infinite     48  alloc a05-p1-dgx-01-c[02-06,08-09,11-15,17-18],b06-
p1-dgx-06-c[10-17],b07-p1-dgx-07-c[01-08],b08-p1-dgx-08-c[01-18]
defq*                 up   infinite     48   idle a07-p1-dgx-03-c[01-05,07,09-11,13-18],b05-p1-
dgx-05-c[01-18],b06-p1-dgx-06-c[02-03,08-09,18],b07-p1-dgx-07-c[09-18]
flex-a07              up    2:00:00      2  drain a07-p1-dgx-03-c[06,12]
flex-a07              up    2:00:00     15   idle a07-p1-dgx-03-c[01-05,07,09-11,13-18]
flex-b05              up    2:00:00     18   idle b05-p1-dgx-05-c[01-18]
```

2. Select idle nodes to test and create slurm allocation on (in this example a full NVL72 rack is selected).

```
root@a03-p1-aps-arm-01:~# salloc -N 18 -w b05-p1-dgx-05-c[01-18]
salloc: Granted job allocation 3912
salloc: Waiting for resource configuration
salloc: Nodes b05-p1-dgx-05-c[01-18] are ready for job
```

3. SSH to node and run `nvidia-imex-ctl -N`.

```
root@b05-p1-dgx-05-c01:~# nvidia-imex-ctl -N
Connectivity Table Legend:
I - Invalid - Node wasn't reachable, no connection status available
N - Never Connected
R - Recovering - Connection was lost, but clean up has not yet been triggered.
D - Disconnected - Connection was lost, and clean up has been triggreed.
A - Authenticating - If GSSAPI enabled, client has initiated mutual authentication.
!V! - Version mismatch, communication disabled.
!M! - Node map mismatch, communication disabled.
C - Connected - Ready for operation
5/13/2025 14:46:11.588
Nodes:
Node #0   - 7.241.18.139    - READY               - Version: 570.124.06
Node #1   - 7.241.18.140    - READY               - Version: 570.124.06
Node #2   - 7.241.18.141    - READY               - Version: 570.124.06
Node #3   - 7.241.18.142    - READY               - Version: 570.124.06
Node #4   - 7.241.18.143    - READY               - Version: 570.124.06
Node #5   - 7.241.18.144    - READY               - Version: 570.124.06
Node #6   - 7.241.18.145    - READY               - Version: 570.124.06
Node #7   - 7.241.18.146    - READY               - Version: 570.124.06
```

```
Node #8   - 7.241.18.147    - READY               - Version: 570.124.06
Node #9   - 7.241.18.148    - READY               - Version: 570.124.06
Node #10  - 7.241.18.149    - READY               - Version: 570.124.06
Node #11  - 7.241.18.150    - READY               - Version: 570.124.06
Node #12  - 7.241.18.151    - READY               - Version: 570.124.06
Node #13  - 7.241.18.152    - READY               - Version: 570.124.06
Node #14  - 7.241.18.153    - READY               - Version: 570.124.06
Node #15  - 7.241.18.154    - READY               - Version: 570.124.06
Node #16  - 7.241.18.155    - READY               - Version: 570.124.06
Node #17  - 7.241.18.156    - READY               - Version: 570.124.06
 Nodes From\To  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
       0        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       1        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       2        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       3        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       4        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       5        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       6        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       7        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       8        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
       9        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      10        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      11        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      12        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      13        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      14        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      15        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      16        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
      17        C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C   C
Domain State: UP
```

The `nvidia-imex-ctl -N` command prints the full status of the current IMEX Domain. It is a useful way to confirm the health of an NVL72 system.

As stated, BCM provides a way to create IMEX Domains per-job in Slurm. The IMEX service and configuration file can be observed too.

4. Checking `nvidia-imex.service` on a DGX node.

```
root@b05-p1-dgx-05-c01:~# systemctl status nvidia-imex.service
● nvidia-imex.service - NVIDIA IMEX service
     Loaded: loaded (/usr/lib/systemd/system/nvidia-imex.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-05-13 14:44:29 PDT; 5min ago
    Process: 99921 ExecStart=/usr/bin/nvidia-imex -c /etc/nvidia-imex/config.cfg (code=exited,
status=0/SUCCESS)
   Main PID: 99981 (nvidia-imex)
      Tasks: 32 (limit: 293736)
     Memory: 10.3M ()
     CGroup: /system.slice/nvidia-imex.service
             └─99981 /usr/bin/nvidia-imex -c /etc/nvidia-imex/config.cfg
May 13 14:44:29 b05-p1-dgx-05-c01 systemd[1]: Starting nvidia-imex.service - NVIDIA IMEX
service...
```

```
May 13 14:44:29 b05-p1-dgx-05-c01 systemd[1]: Started nvidia-imex.service - NVIDIA IMEX service.
```

5. Check `/etc/nvidia-imex/nodes_config.cfg` (this list is populated with the same nodes that are in `$SLURM_JOB_NODELIST`).

```
root@b05-p1-dgx-05-c01:~# cat /etc/nvidia-imex/nodes_config.cfg
# imex nodes_config for job 3912
# b05-p1-dgx-05-c01
7.241.18.139
# b05-p1-dgx-05-c02
7.241.18.140
# b05-p1-dgx-05-c03
7.241.18.141
# b05-p1-dgx-05-c04
7.241.18.142
# b05-p1-dgx-05-c05
7.241.18.143
# b05-p1-dgx-05-c06
7.241.18.144
# b05-p1-dgx-05-c07
7.241.18.145
# b05-p1-dgx-05-c08
7.241.18.146
# b05-p1-dgx-05-c09
7.241.18.147
# b05-p1-dgx-05-c10
7.241.18.148
# b05-p1-dgx-05-c11
7.241.18.149
# b05-p1-dgx-05-c12
7.241.18.150
# b05-p1-dgx-05-c13
7.241.18.151
# b05-p1-dgx-05-c14
7.241.18.152
# b05-p1-dgx-05-c15
7.241.18.153
# b05-p1-dgx-05-c16
7.241.18.154
# b05-p1-dgx-05-c17
7.241.18.155
# b05-p1-dgx-05-c18
7.241.18.156
```

6. Checking `/var/log/nvidia-imex.log` on DGX node.

```
IMEX Log initializing at: 5/13/2025 14:44:29.455
[May 13 2025 14:44:29] [INFO] [tid 99981] IMEX version 570.124.06 is running with the following
configuration options
[May 13 2025 14:44:29] [INFO] [tid 99981] Logging level = 4
[May 13 2025 14:44:29] [INFO] [tid 99981] Logging file name/path = /var/log/nvidia-imex.log
[May 13 2025 14:44:29] [INFO] [tid 99981] Append to log file = 1
[May 13 2025 14:44:29] [INFO] [tid 99981] Max Log file size = 1024 (MBs)
```

```
[May 13 2025 14:44:29] [INFO] [tid 99981] Use Syslog file = 0
[May 13 2025 14:44:29] [INFO] [tid 99981] IMEX Library communication bind interface =
[May 13 2025 14:44:29] [INFO] [tid 99981] IMEX library communication bind port = 29262
[May 13 2025 14:44:29] [INFO] [tid 99981] Identified this node as ID 0, using bind IP of
'7.241.18.139', and network interface of bond0
[May 13 2025 14:44:29] [INFO] [tid 99981] NvGpu Library version matched with GPU Driver version
[May 13 2025 14:44:29] [INFO] [tid 100014] Started processing of incoming messages.
[May 13 2025 14:44:29] [INFO] [tid 100015] Started processing of incoming messages.
[May 13 2025 14:44:29] [INFO] [tid 100016] Started processing of incoming messages.
[May 13 2025 14:44:29] [INFO] [tid 99981] Creating gRPC channels to all peers (nPeers = 18).
[May 13 2025 14:44:29] [INFO] [tid 100017] Started processing of incoming messages.
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 0 with ip address
7.241.18.139. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 3 with ip address
7.241.18.142. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 2 with ip address
7.241.18.141. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 4 with ip address
7.241.18.143. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 1 with ip address
7.241.18.140. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 5 with ip address
7.241.18.144. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 7 with ip address
7.241.18.146. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 6 with ip address
7.241.18.145. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 8 with ip address
7.241.18.147. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 99981] IMEX_WAIT_FOR_QUORUM != FULL, continuing initialization
without waiting for connections to all nodes.
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 9 with ip address
7.241.18.148. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 10 with ip address
7.241.18.149. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 11 with ip address
7.241.18.150. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 12 with ip address
7.241.18.151. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 13 with ip address
7.241.18.152. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 14 with ip address
7.241.18.153. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 15 with ip address
7.241.18.154. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 16 with ip address
7.241.18.155. Number of times connected: 1
[May 13 2025 14:44:29] [INFO] [tid 99981] GPU event successfully subscribed
[May 13 2025 14:44:29] [INFO] [tid 100018] Connection established to node 17 with ip address
7.241.18.156. Number of times connected: 1
```

Chapter 5. Observability Software

5.1 Base Command Manager Monitoring

NVIDIA Mission Control utilizes NVIDIA Base Command Manager's built-in monitoring infrastructure for comprehensive system observability. This guide assumes administrators are familiar with BCM monitoring concepts, including metrics collection, measurable definitions, trigger and alert configurations, and health checks.

For detailed coverage of observability and monitoring in Base Command Manager, administrators should refer directly to:

- Base Command Manager (BCM) Administrator Guide:
  - Monitoring Cluster Devices (Section 10)
  - Metrics, Health Checks, Enummetrics, and Actions (Appendix G)

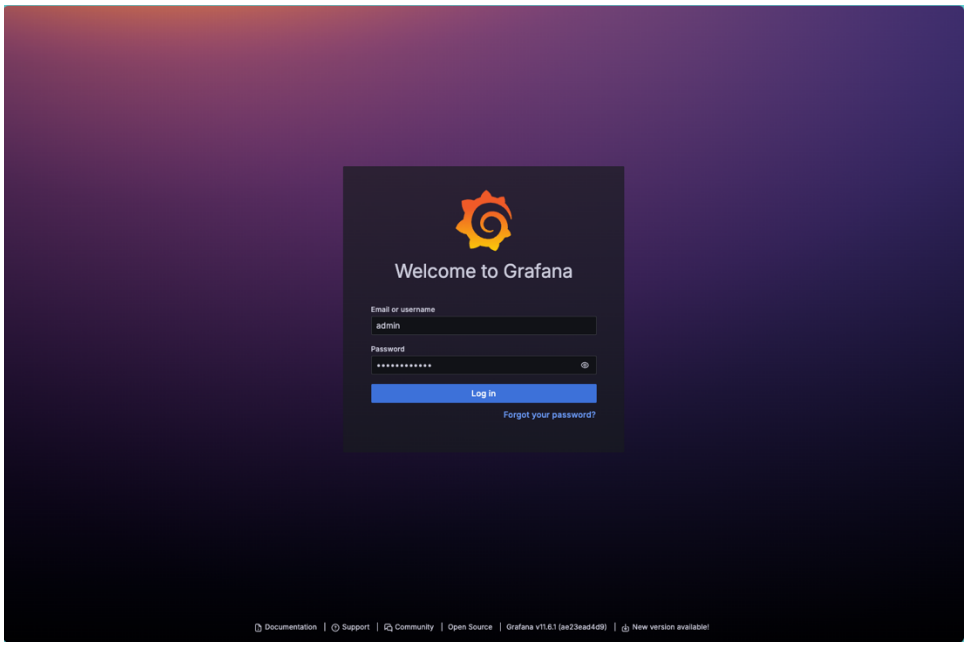
For customizing monitoring alerts, health checks, and related configurations, consult these sections before proceeding.

5.1.1 Prometheus and Grafana

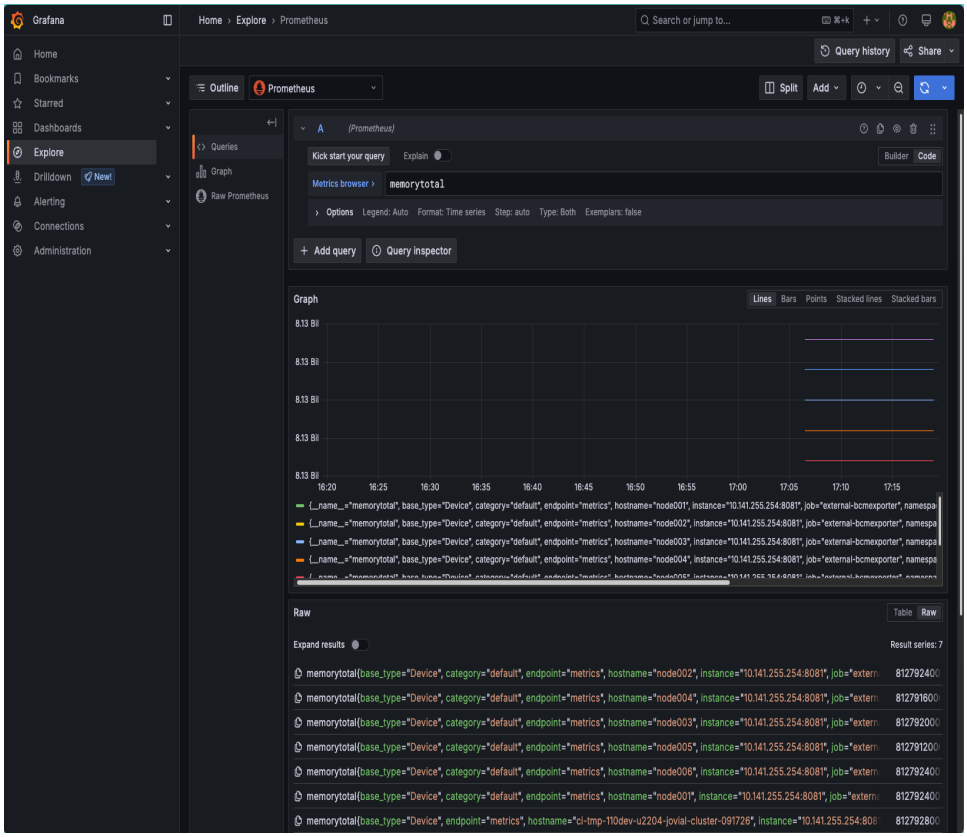
5.1.1.1 Validate metric collection using Grafana

Grafana should be available on your head node listening on HTTPS on the `/grafana` page.

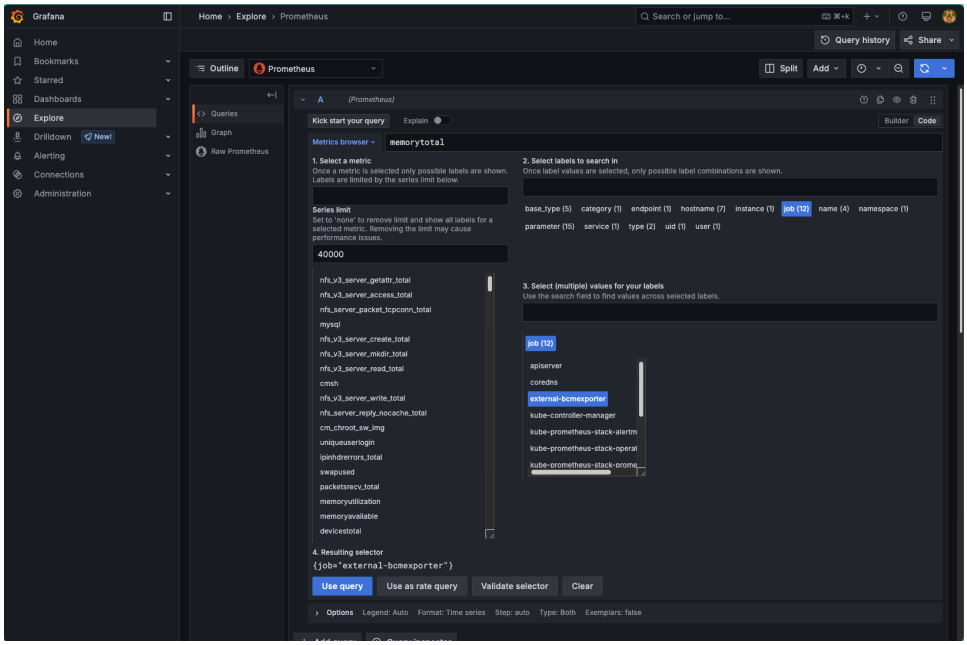
By default, the administrative account is `admin` with password `prom-operator`



Once authenticated, click on `Explore` on the left. Once in the Explore interface, set the query editor to `code` and query the metric `memorytotal`.



BCM provides many metrics out of the box, you can find a list of these in Grafana by using the `Metric browser` selecting all metrics which include the label `job` with the value `external-bcmexporter`.



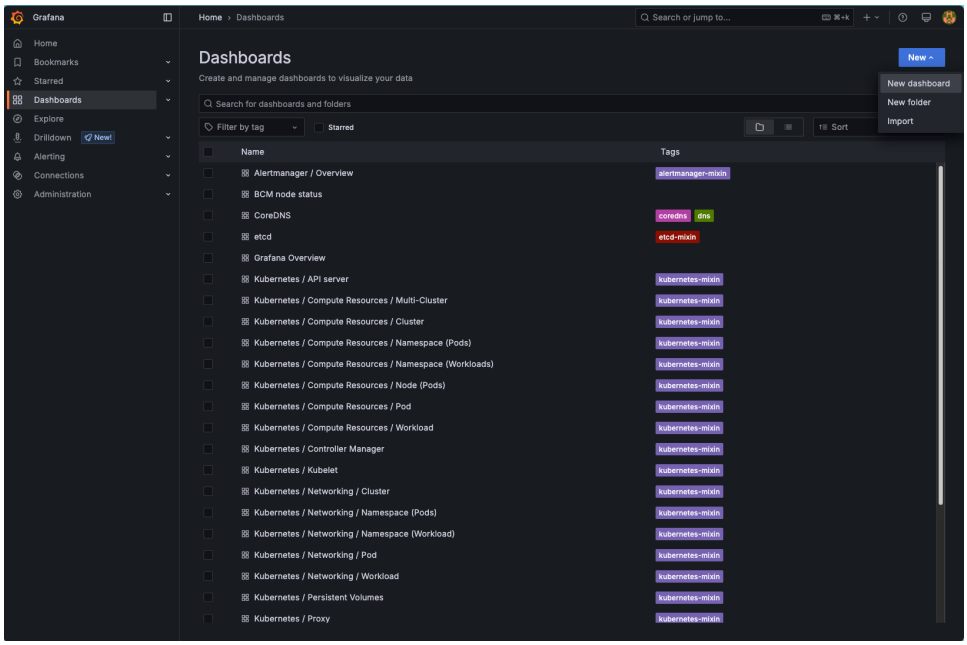
This can also be found by inspecting the output of the BCM Prometheus exporter endpoint. Here using curl with some light processing.

```
curl -sk https://localhost:8081/exporter | grep -Ev '# HELP|# TYPE' | cut -d '{' -f1 | sort -u

alertlevel
...
writetime_total
```

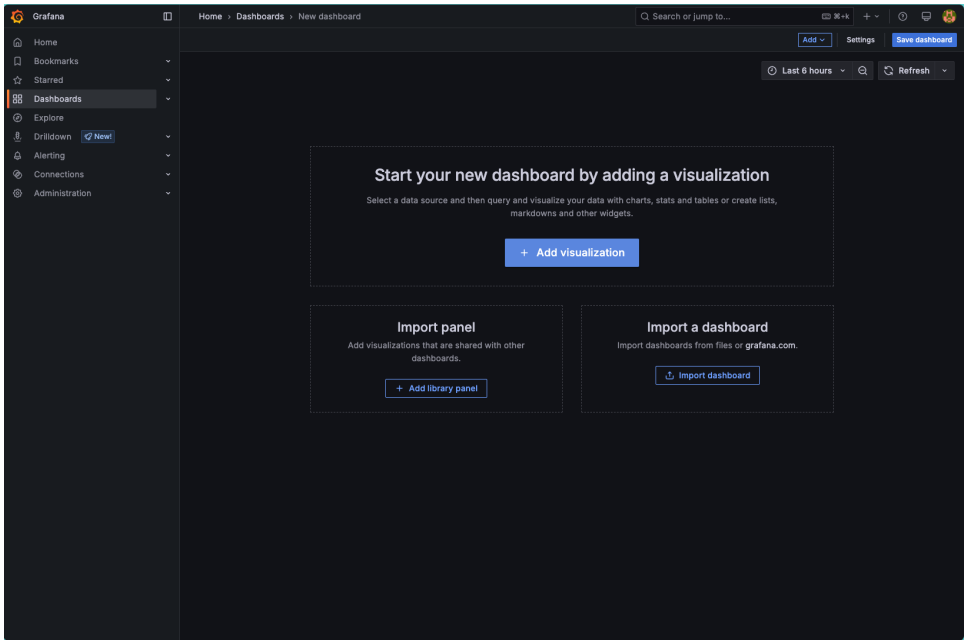
5.1.1.2 Build a Grafana dashboard

With BCM exporting metrics, we can create dashboards with these. In this example, we'll build a device state history dashboard.

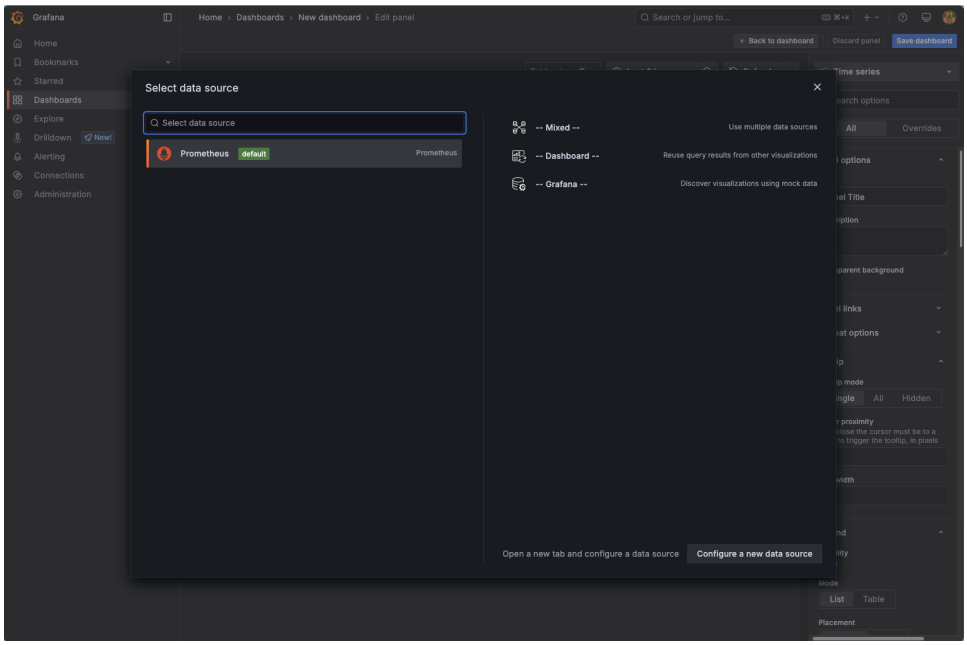
1. Select `Dashboards` on the left panel, select `New` and select `New dashboard`.



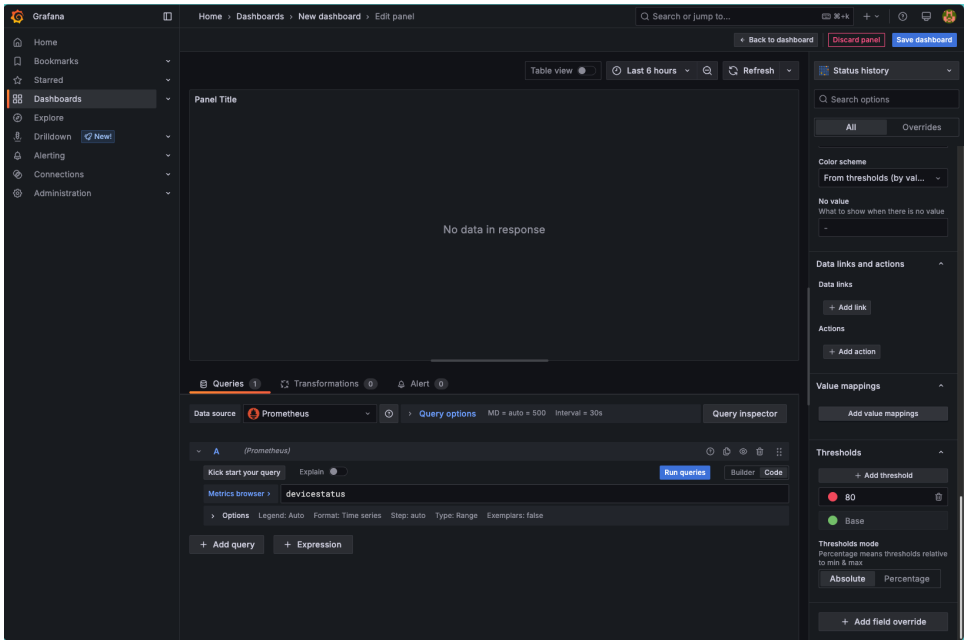
2. Click `Add visualization`.



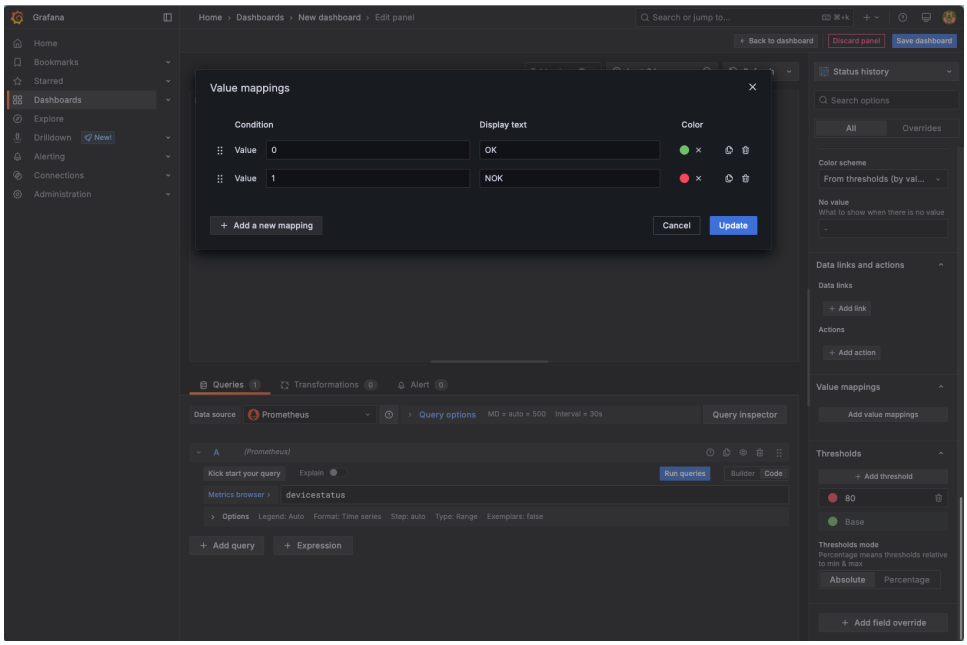
3. Select `Prometheus` as your datasource.



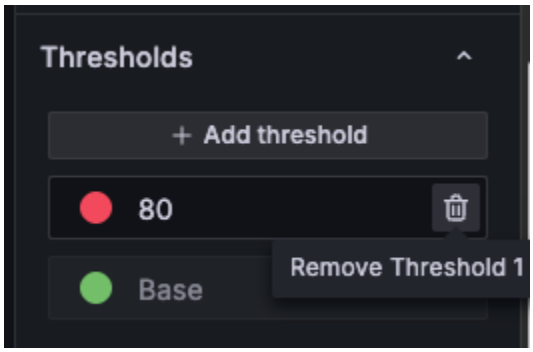
4. Start by setting the visualization to `Status history`. Query the metric `devicestatus`.



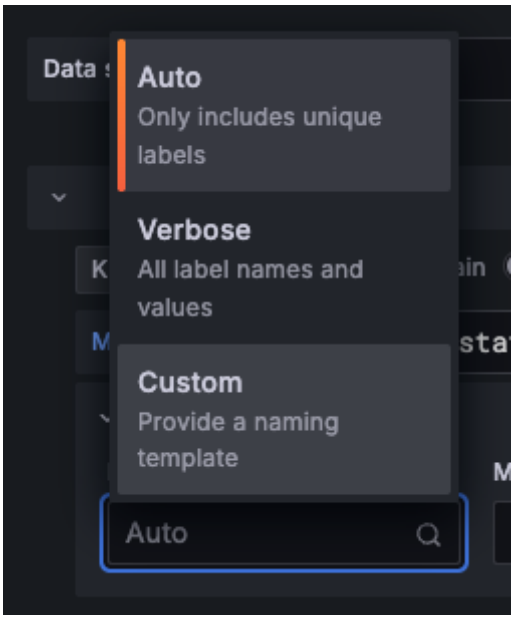
5. On the right pane, scroll down to `Value mappings` and click on `Add value mappings`. This metric returns either `0` which maps to `OK` or `1` which maps to `NOK`. Add value conditions for these with a desired color. We'll use `0` as `green` and `1` as `red`.



6. Delete the threshold for the value `80`.



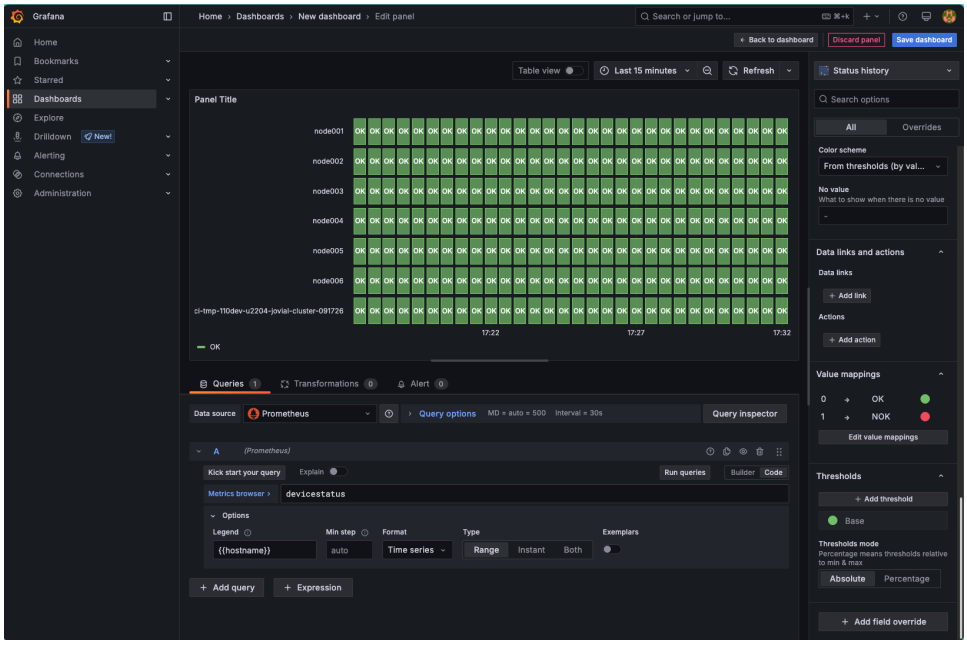
7. Expand options in the query window and set the Legend to `Custom` and set the value to `{{hostname}}`.



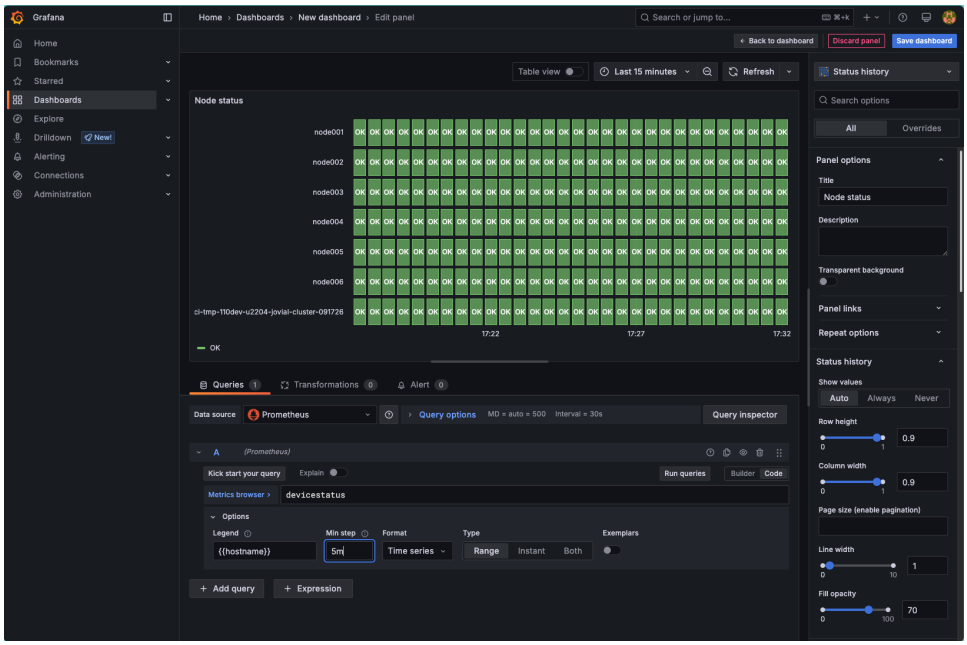
8. You may get this error:

```
Too many points to visualize properly.
```

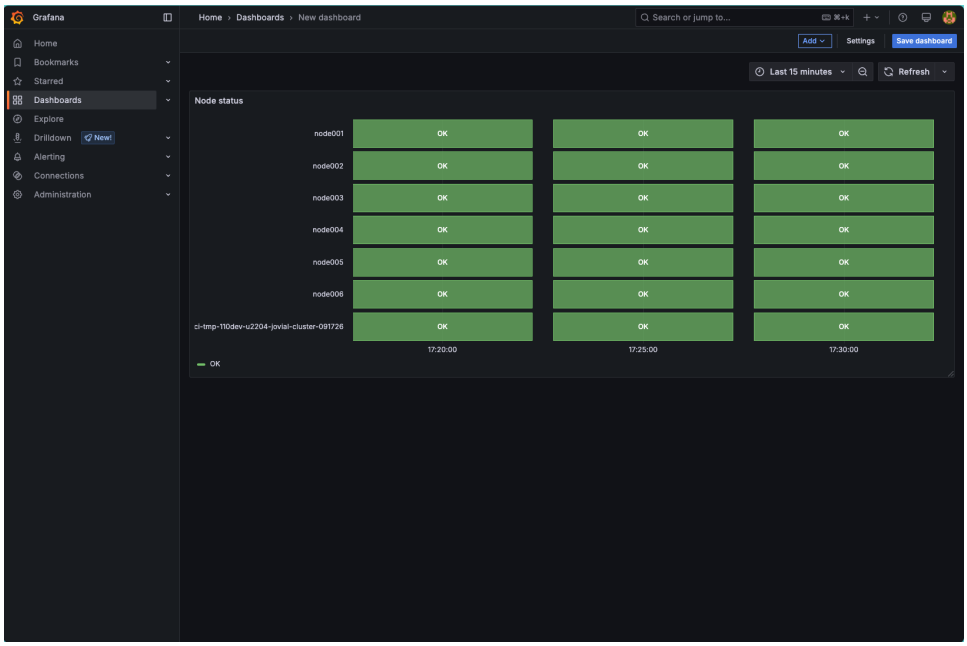
9. You can work around this by setting the time window of the dashboard from `Last 6 hours` to `Last 15 minutes`.



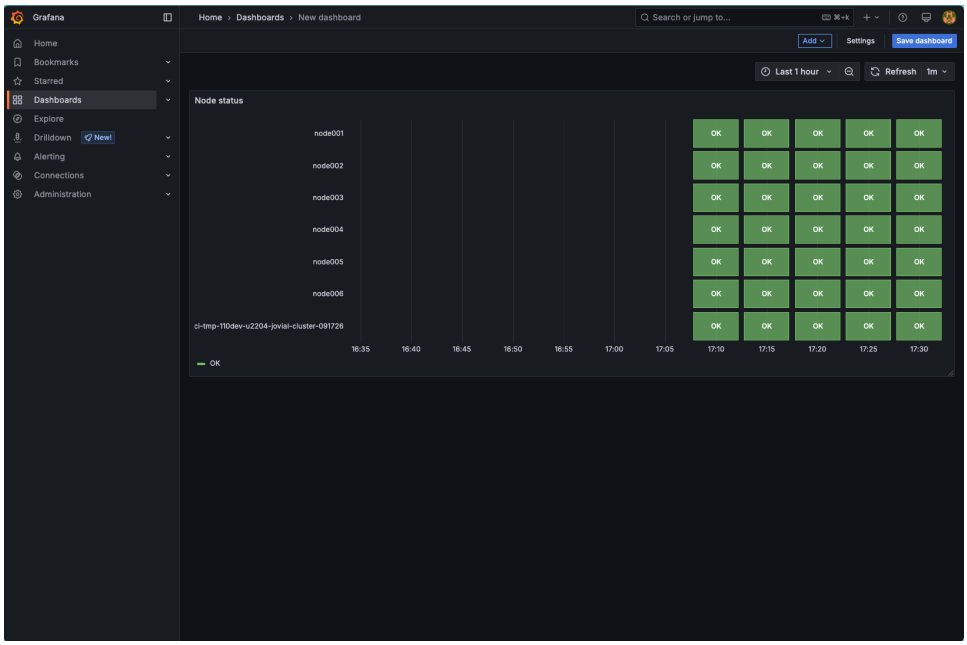
10. To allow for us to use this over a longer period of time, we'll set the `Min step` for our query to `5m`. Set the `Title` of the panel to `Node status`.



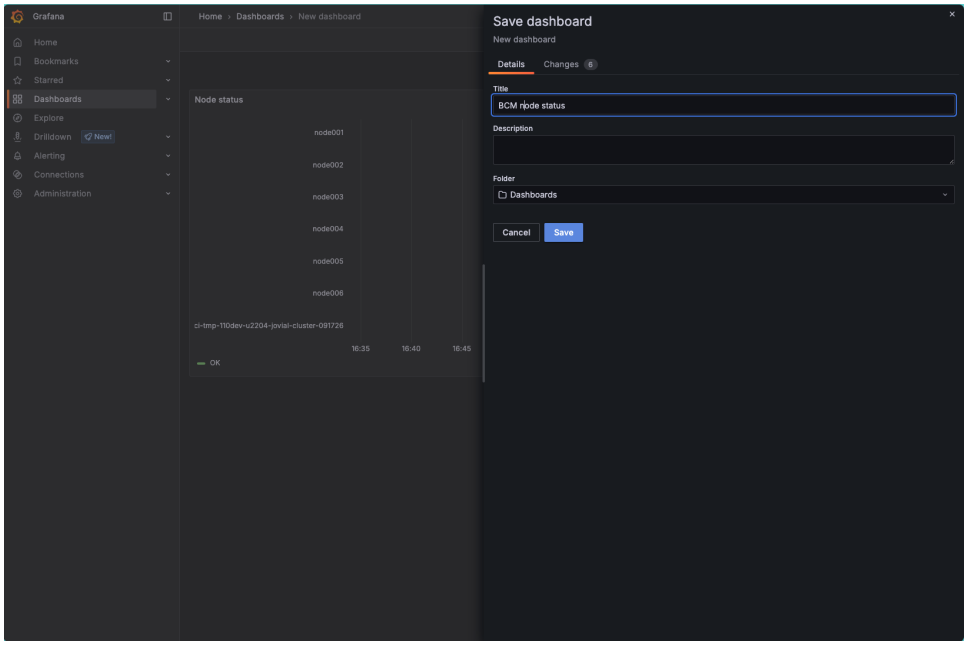
Click on `Back to dashboard`. Resize the panel to a desired size.



11. Set the window of time to the `Last 1 hour`, take note that the dashboard rendered more status boxes.



12. Select `Save dashboard`. Set the Title to `BCM node status` and click on `Save`.



5.1.2 Logs

With BCM, in-built logging aggregation comes out of the box. Each node that BCM deploys by default forwards logs to the head nodes for centralization of logs. From a head node, you can view these logs in `/var/log/syslog`.

5.1.2.1 Counting Xid errors

When a GPU fault has occurred, these are logged to syslog. These can indicate user driven exceptions, errors with the driver, or in some circumstances a hardware fault. It's good to understand the `Xids` encountered and the counts of these.

`Xid` values and descriptions can be found in this document **Xid errors**

```
grep "Xid (PCI" /var/log/syslog | awk -v cutoff="$(date -d '12 hours ago' '+%b %d %H:%M:%S')" '
  $1 " " $2 " " $3 >= cutoff {
    split($4, host, ".");
    if (match($0, /Xid \(PCI:[^)]+\): ([0-9]+)/, id)) {
      print host[1], id[1]
    }
  }
' | sort | uniq -c | awk '{print $2 ":", $1, "occurrences of error ID", $3}'

a07-p1-dgx-03-c08: 10158 occurrences of error ID 143
b06-p1-dgx-06-c01: 4 occurrences of error ID 149
b06-p1-dgx-06-c01: 4 occurrences of error ID 154
b06-p1-dgx-06-c02: 4 occurrences of error ID 149
b06-p1-dgx-06-c02: 3 occurrences of error ID 154
b06-p1-dgx-06-c04: 4 occurrences of error ID 149
b06-p1-dgx-06-c04: 4 occurrences of error ID 154
b06-p1-dgx-06-c05: 4 occurrences of error ID 149
b06-p1-dgx-06-c05: 4 occurrences of error ID 154
b06-p1-dgx-06-c06: 4 occurrences of error ID 149
b06-p1-dgx-06-c06: 4 occurrences of error ID 154
b06-p1-dgx-06-c07: 4 occurrences of error ID 149
b06-p1-dgx-06-c07: 2 occurrences of error ID 154
b06-p1-dgx-06-c08: 4 occurrences of error ID 149
b06-p1-dgx-06-c08: 4 occurrences of error ID 154
b06-p1-dgx-06-c09: 4 occurrences of error ID 149
b06-p1-dgx-06-c09: 4 occurrences of error ID 154
b06-p1-dgx-06-c11: 4 occurrences of error ID 149
b06-p1-dgx-06-c11: 4 occurrences of error ID 154
b06-p1-dgx-06-c12: 4 occurrences of error ID 149
b06-p1-dgx-06-c12: 3 occurrences of error ID 154
b06-p1-dgx-06-c13: 4 occurrences of error ID 149
b06-p1-dgx-06-c13: 4 occurrences of error ID 154
b06-p1-dgx-06-c13: 112 occurrences of error ID 45
b06-p1-dgx-06-c14: 4 occurrences of error ID 149
b06-p1-dgx-06-c14: 4 occurrences of error ID 154
b06-p1-dgx-06-c14: 112 occurrences of error ID 45
b06-p1-dgx-06-c15: 4 occurrences of error ID 149
b06-p1-dgx-06-c15: 4 occurrences of error ID 154
b06-p1-dgx-06-c15: 112 occurrences of error ID 45
b06-p1-dgx-06-c16: 4 occurrences of error ID 149
b06-p1-dgx-06-c16: 4 occurrences of error ID 154
b06-p1-dgx-06-c16: 112 occurrences of error ID 45
b06-p1-dgx-06-c17: 4 occurrences of error ID 149
```

```
b06-p1-dgx-06-c17: 4 occurrences of error ID 154
b06-p1-dgx-06-c17: 112 occurrences of error ID 45
b06-p1-dgx-06-c18: 4 occurrences of error ID 149
b06-p1-dgx-06-c18: 4 occurrences of error ID 154
```

Chapter 6. Autonomous Hardware Recovery

NVIDIA Mission Control includes support for Autonomous Hardware Recovery, a built-in service that enables automated detection and remediation of hardware issues across the DGX SuperPOD. This service continuously monitors the health of system components and applies predefined recovery actions when issues are detected—reducing downtime and minimizing the need for manual intervention.

In addition to real-time remediation, Autonomous Hardware Recovery can be used to run cluster validation, executing a suite of baseline tests to assess the current health and readiness of the environment. These tests help ensure that the cluster is operating within expected parameters and is suitable for running workloads.

For detailed setup, configuration, alert policy creation, and usage examples, refer to the Autonomous Hardware Recovery Administrator Guide.

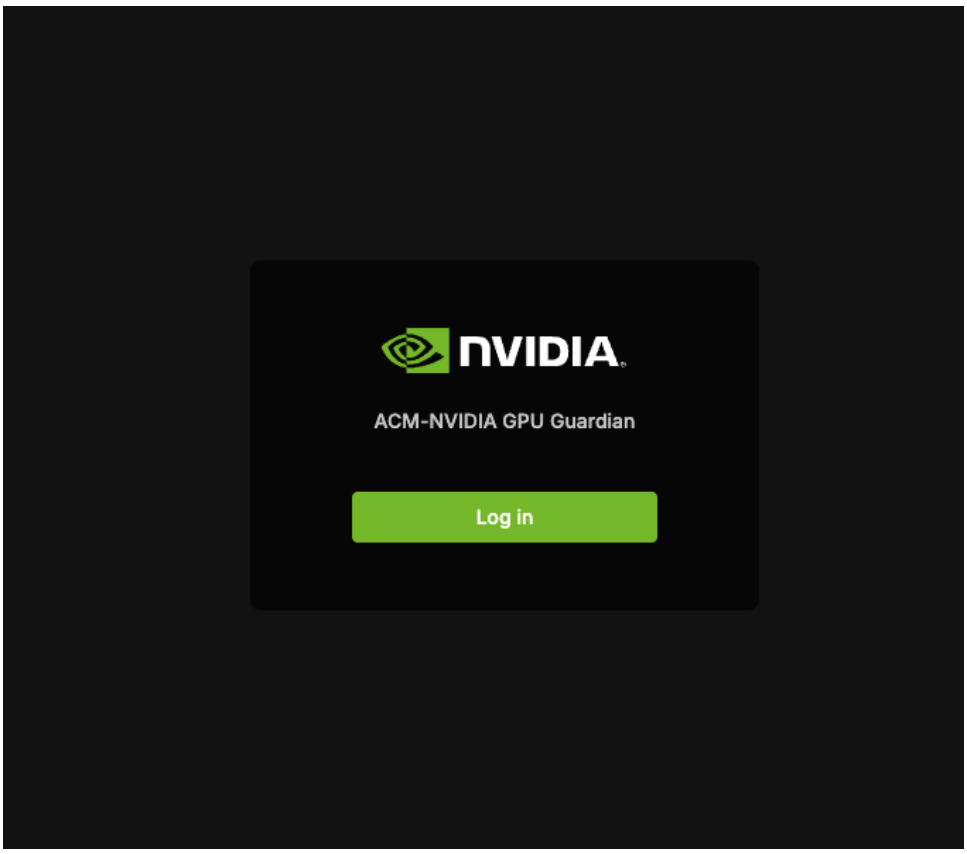
6.1 Accessing the Autonomous Hardware Recovery UI

NVIDIA Mission Control autonomous hardware recovery's authentication relies on BCM's LDAP authentication so that users will leverage their BCM credentials to login to NVIDIA Mission Control autonomous hardware recovery.

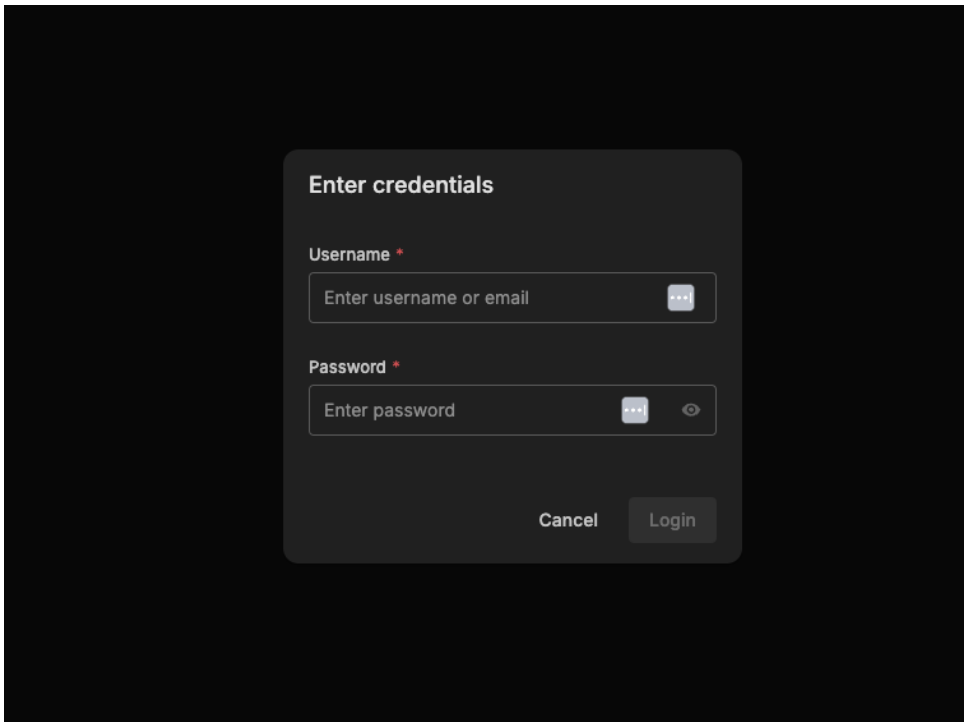
Upon successful authentication, the user session receives a short lived JWT and refresh token. The JWT is used for identity and refreshed by the UI at expiration time.

An administrator can opt to manage users and groups using BCM. Any changes will automatically be reflected within NVIDIA Mission Control autonomous hardware recovery. There are two options to access NVIDIA Mission Control autonomous hardware recovery. The first option is to authenticate using SSO with BCM identity. The other option is to go to the url for the NVIDIA Mission Control autonomous hardware recovery UI. You will be presented with a login screen.

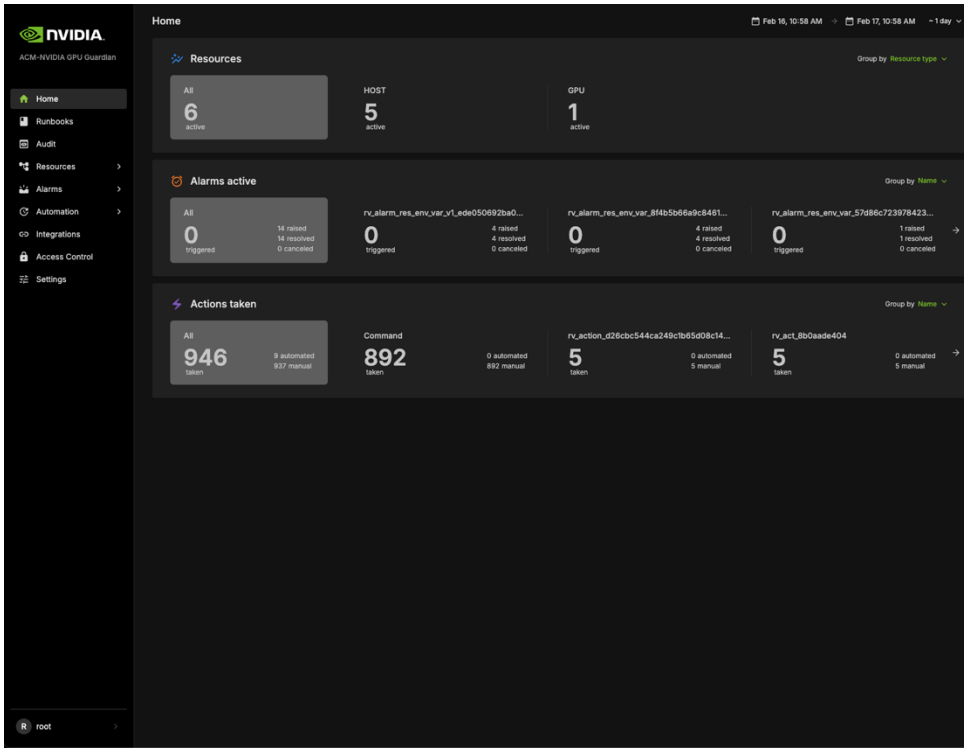
6.1.1 Login screen



Once you click the login button, the authentication page appears where you can enter credentials.



6.1.2 Main Landing Page



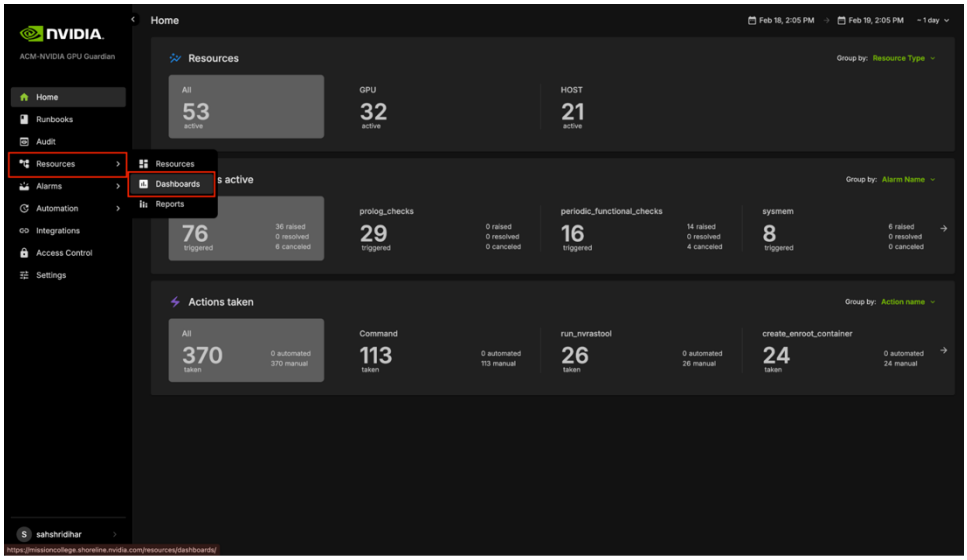
Users are authorized to perform different activities within NVIDIA Mission Control autonomous hardware recovery by configuring permission policies. Policies determine if the user can view resources and execute actions (named and/or anonymous). Action execution can be limited to a maximum number of impacted resources and/or specific resources. Permissions can also be attached to runbooks to allow/disallow certain users or groups.

New users are first created in BCM before they are able to access NVIDIA Mission Control autonomous hardware recovery. Upon logging in, there is a default permission policy that every user is assigned. The permissions of this policy are determined by the administrator. The administrator role has permission to perform any action in NVIDIA Mission Control autonomous hardware recovery. The configurator role has the permission to create, edit, delete any artifacts in NVIDIA Mission Control autonomous hardware recovery. By default, new users are granted administer and configure roles until the privileges are overridden. Defaults can be modified in the Access Control section of the NVIDIA Mission Control autonomous hardware recovery UI.

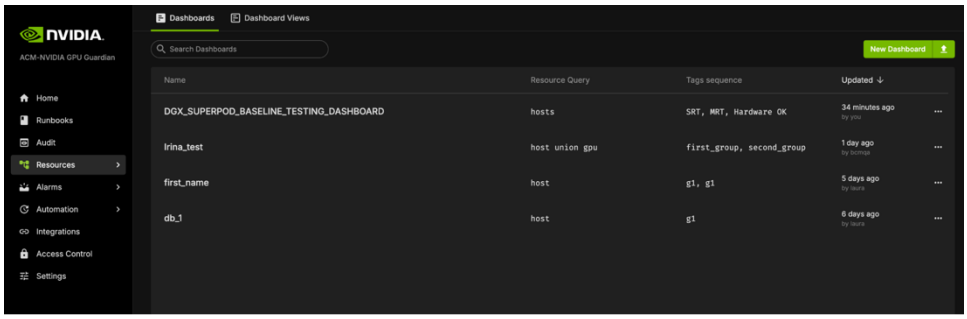
NVIDIA Mission Control autonomous hardware recovery dashboard provides a comprehensive view of the status of all resources in the cluster, displaying the progress of testing at various stages for each node (control, compute and switch nodes). It shows which tests are completed, which ones have passed, and which have failed. This allows users to easily track the overall health and status of the cluster, identify any issues, and assess the readiness of each resource.

6.1.3 Navigating to Dashboards

1. Visit the NVIDIA Mission Control autonomous hardware recovery homepage, navigate to the `Resources` menu, and click on `Dashboards` to access the dashboard page.



2. The Landing page contains two tabs: `Dashboards` and `Dashboard Views`.



3. The `Dashboards` lists all the `Dashboards` available. All these dashboards reflect the current status of the cluster and the test status.
4. A `Dashboard View` is a snapshot of the `Dashboard` that captures the data at a specific point in time. This snapshot remains static, preserving the exact state and data of the `Dashboard` as it appeared at that moment, regardless of any future changes or updates to the live data.

6.1.4 Cluster Validation

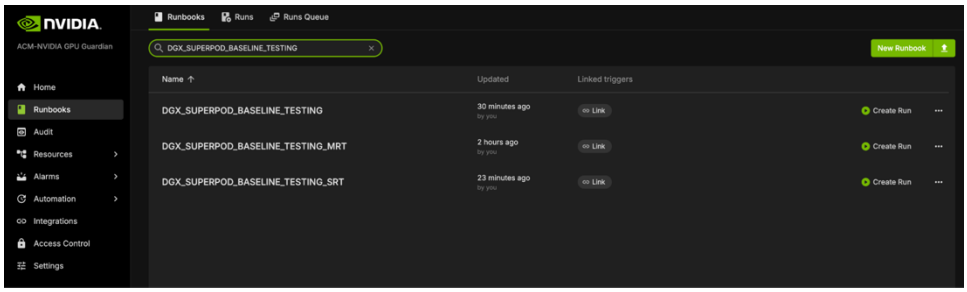
The NVIDIA Mission Control autonomous hardware recovery portal enables efficient automation of baseline testing procedures. This comprehensive testing framework can be flexibly executed across various scales of infrastructure, from individual compute nodes to complete racks or multiple rack configurations. The system performs extensive validation of critical compute node components, including CPU/GPU/Memory/storage functionality, network connectivity, and firmware versioning. Additionally, it incorporates industry-standard performance benchmarking tools such as HPL, NCCL, and Nemotron(Large Language Model) to assess system capabilities. This streamlined approach significantly enhances both testing efficiency and thoroughness while reducing execution time.

6.1.4.1 Entrypoint of Automated Testing

A centralized automated baseline testing interface has been established to facilitate streamlined test execution and management. This unified entry point provides comprehensive access to the testing framework, enabling efficient navigation and one click implementation of all testing procedures.

To access the baseline testing interface:

1. Access the NVIDIA Mission Control autonomous hardware recovery portal using your credentials and navigate to the Runbooks section.
2. Locate the `DGX_SUPERPOD_BASELINE_TESTING` runbook through the search functionality (reference the interface depicted below).



3. Upon selecting the `DGX_SUPERPOD_BASELINE_TESTING` runbook, you will be presented with the interface shown below.

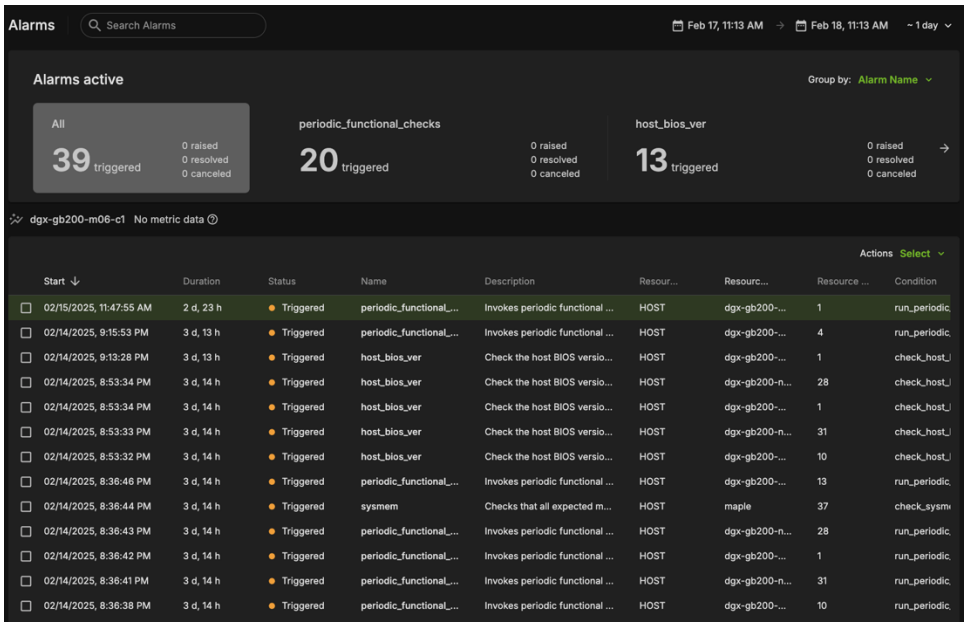
4. More about initializing and customizing these runbooks can be found in the extended Automated Hardware Recovery guide.

6.1.5 Health Checks & Alerts

NVIDIA Mission Control autonomous hardware recovery provides a full suite of automated health checks to detect failures at the tray, rack, and system levels for GB200. In addition, system wide health checks are performed by integrating with the UFM and NMX-M network control planes. Health check data is reported back to BCM BaseView and/or the in-cluster LGTM stack. These health checks are performed at two layers: BCM job invocation, and as periodic health checks via NVIDIA Mission Control autonomous hardware recovery.

6.1.5.1 Alarms Dashboard

The alarms dashboard is an overview of all alarms and the state of your system. In this view, alarms are summarized by counts of alarms firing, alarms firing most frequently, and a configurable list of most frequently firing, canceled, or resolved alarms. This is meant to be a starting point for any investigations of possible issues with your systems, and you may click any alarm for further details.



6.1.5.2 BCM Prolog, Epilog, and Periodic checks

When Slurm jobs are invoked, a series of checks will be automatically performed to ensure the state and quality of nodes. These checks are performed at the system level, and while the script files are pushed to nodes by NVIDIA Mission Control autonomous hardware recovery, there is no viewable configuration in NVIDIA Mission Control autonomous hardware recovery. They can instead be viewed in the NVIDIA Mission Control autonomous hardware recovery package in the `Shoreline_files/scripts/slurm` folder, along with their default values.

The Prolog checks run prior to the Slurm job, and on failure will set the node state to DRAIN and re-queue the job. On success, the job will proceed and the `Epilog` checks will run thereafter. On `Epilog` check failure, the node's state will also be set to `DRAIN`.

Unlike the `Prolog` and `Epilog` checks, Periodic Checks are defined within the NVIDIA Mission Control autonomous hardware recovery interface as Alarms, and are detailed in the next section. They will be automatically enabled for racks that pass Single Rack Testing but may also be manually enabled or disabled for specific racks by running the `ALARMS_RACK_ENABLE` and `ALARMS_RACK_DISABLE` runbooks.

`Prolog` and `Epilog` checks are disabled by default and should be enabled only after nodes are initially deemed healthy. To enable or disable them, please run the `SLURM_CHECKS_ENABLE` or `SLURM_CHECKS_DISABLE` runbooks respectively.

6.1.5.3 Alarms

Below is a list of the configured Alarms:

bmc_sensors

Checks the sensors from the Baseboard Management Controller (BMC) to ensure the proper data is returned.

cpu_stepping

Checks that the CPU stepping parameter is correct for each CPU.

dns_host

Checks the DNS configuration and resolution for the host.

gpu_alloc_temp

Checks if the GPU temperatures are above a threshold.

gpu_temp_history

Checks System Event Log (SEL) history looking for GPU temperature issues.

numa_node_count

Checks that the correct count of Non-uniform memory access (NUMA) nodes are configured with the CPU cores.

raid_count

Checks that the raid configuration matches the expected mdstat configuration.

sysmem

Checks that all expected memory DIMMs are present.

nfs_mounts

Verifies required mount points.

sysctl_rp_filter

Verifies RP Filters are strict.

daily_informational

Checks for which the severity and remediation may not be critical. This alarm will only be triggered once per day, and results may be viewed in the resulting runbook run.

check_sel_event - Read the SEL events from the BMC and ensure none are asserted

check_dgx_os_version - Verifies the DGX OS version matches the expected value

check_gpu_vbios_ver - Checks the VBIOS version of the GPUs and compares against an expected value

check_nvme_fw_ver - Checks that the FW version for each NVMe matches an expected value

check_kernel_commandline_opt - Verifies the specified kernel option(s) is present in the current kernel's boot parameters

check_host_bios_ver - Verifies the system's BIOS version

check_kernel_ver - Verifies the current version of the Linux kernel

check_host_package_versions - Queries the installed packages on the host

nv_container_cli_info - Retrieves information about the NVIDIA container CLI (driver and devices)

periodic_bmc_host_checks

The following groups of periodic functional checks are a subset of the BCM Prolog checks that run at predefined intervals as NVIDIA Mission Control autonomous hardware recovery Alarms.

check_bmc_ipmi_version - Checks BMC IPMI version against an expected value

check_nvidia_module_loaded - Verifies the NVIDIA module is loaded in the host OS

check_nvidiasmi_version - Verifies the NVIDIA module version matches the expected value

check_host_os_version - Verifies the DGX OS version matches the expected value

check_nvsm_status - Verify the NVSM service is currently active

periodic_cpu_mem_checks

check_cpu_health - Verifies CPU sockets and cores are present and online

check_dimm_count - Checks that all expected memory DIMMs are present

check_dimm_size - Checks that the size of each memory DIMM matches the expected values

check_memory_swap_size - Checks that the memory swap size matches the expected value

periodic_gpu_nvlink_checks

check_gpu_pci - Checks that all GPUs are present on the lspci interface and with the correct link width and speed

check_gpu_error - Checks GPUs for ECC errors, retired pages, and throttles present

check_gpu_powerstate - Checks the powerstate for each GPU and compares against an expected value

check_gpu_param - Checks that specified GPU parameters are present and correct for the host

check_nvlink_health - Checks that links are active for each GPU, the speed is correct, fabric registration has been completed, are running at full bandwidth, and belong to the same NVLink domain and partition.

check_gpu_topology - Checks that there are no issues with the p2p topology within the node

check_gpu_telemetry - Checks that various sensors can be successfully read from the GPU via nvidia-smi

check_gpu_power_limit - Checks that the power limit is correct for each GPU

check_nvidia_inforom_ver - Checks that the inforom version is correct for each GPU

check_gpu_clock_info - Checks that the maximum clock speed is correct for each GPU

check_remapped_row - Checks if any remapped row events have occurred

periodic_network_checks

check_ib_health - Checks that the CX7 devices are present, active, and in the physical LinkUp state via ibstat, and also matching the expected transfer rate

check_ib_ber_and_ro - Checks if the PCI_WR_ORDERING field is set to relaxed and also the bit error rate of the CX7 using mlxlink

check_ib_port_rcv_errors - Check Infiniband devices port RCV errors

check_ib_cables - Checks the cable info using mlxcables

check_bf3_speed - Validates that the BF3 devices are operating at the correct speed and that the proper number of devices are in the “Up” state

periodic_storage_checks

check_pex_switch_health - Checks that the PEX switches are present, have the correct PCIe link speed and width, and the downstream devices have enumerated to lspci

check_pcie_config - Checks that the CX7 devices have the correct PCIe link speed and width via lspci and ACS config via setpci

check_nvme_health - Checks that the PCIe link speed and width of each NMVe device matches the expected value

check_storage_dir - Checks that the host has functional access to the home storage

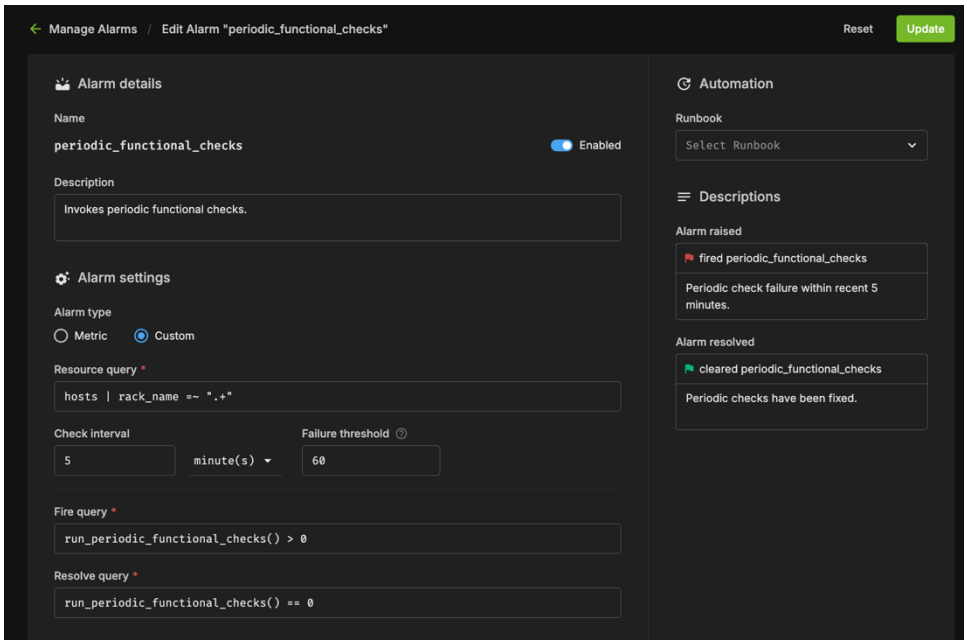
check_storage_util - Checks that the used local storage on the host is below a given threshold

periodic_error_checks

Checks journald for bmc issues, machine check errors, Xid, AER, CPER, I/O, GPU fell off the bus, and other generic errors.

6.1.5.4 NVIDIA Mission Control autonomous hardware recovery Alarm Configuration

There are several components to an alarm, with the key pieces being the Resource Query, Fire Query, Resolve Query, Check Interval and Automation. Below you can see an example configuration.



Resource Query

The resource query allows you to customize the resources (hosts, pods, gpus) on which the checks will be performed. In the example above, the `hosts | rack_name =~ ".*"` will only check alarms on hosts which have a value set for the `rack_name` tag.

Fire Query

The fire query is a condition that, when true, will cause the alarm to begin firing. It will be run at each interval.

Resolve Query

Similar to the fire query, the Resolve query is a condition that will resolve the alarm when true. Resolving an alarm will cause firing to cease and the state to change to `Resolved`.

Check Interval

The interval at which the fire and resolve queries are checked.

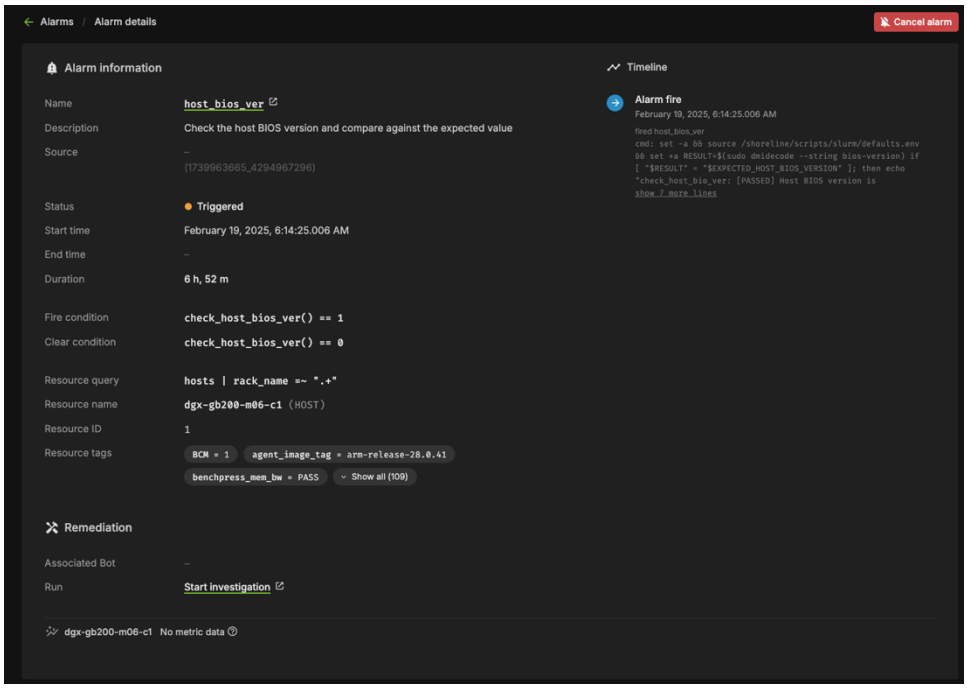
6.1.5.5 Alarm States

If an Alarm has triggered, it will be in one of the following three states:

`Triggered`

This state means the alarm is currently firing. Any automation (break/fix) will subsequently be invoked to remediate any issues, potentially resolving the alarm. Alternatively, the user could cancel the alarm by clicking the `Cancel alarm`.

Clicking into the triggering alarm will give you more details on what caused the alarm, metadata and resources relating to the alarm, and will also allow you to view log output from the check itself.



`Resolved`

When the clear query of an alarm evaluates to true for a firing alarm, the status will be changed to Resolved. Automation triggered runbooks will invoke break/fix operations that should be configured to result in a resolved alarm.

`Canceled`

When a user cancels an alarm from the dashboard, or from the triggered alarm itself, its state will become Canceled. Also, if an alarm configuration is changed for an alarm in the Triggered state, it will be canceled since it was triggered against a defunct configuration.

Chapter 7. Out-of-Band Management

7.1.1 Using cmsh to get BMC events

BCM integrates with BMC interfaces via the Redfish API. You can use `cmsh` to get events.

1. Get a list of physical nodes for a specific rack, filtering for a status of `DOWN`

```
root@a03-p1-head-01:~# cmsh

[a03-p1-head-01]% device list -t physicalnode -r a06 | grep DOWN
```

2. Show events from the BMC interface.

```
[a03-p1-head-01]% device use a06-p1-dgx-02-c18
[a03-p1-head-01->device[a06-p1-dgx-02-c18]]% bmceventlog
Device             id               message
severity         timestamp                 type             Result   Error
------------------ ---------------- ------------------------------------------------------------
--- ---------------- ------------------------- ---------------- -------- ----------------------
---------
a06-p1-dgx-02-c18  5424             The state of resource `Chassis_0_LeakDetector_0_Manifold` has
Warning          2025-05-12T10:28:41+00:00                   good
                                     changed to Degraded.
```

7.1.2 Using cmsh to access BMC SoL

The BMC on DGX nodes allows for two types of remote access during `POST` and `BOOT` via Serial over LAN and a graphical console, KVM. To access SoL, you can start this via `cmsh`.

This is covered in Section 14.7 in the BMC 11 administration manual.

```
root@a03-p1-head-01:~# cmsh

[a03-p1-head-01]% device
[a03-p1-head-01->device]% use a07-p1-dgx-03-c08
[a03-p1-head-01->device[a07-p1-dgx-03-c08]]% rconsole

================================================================================
ipmiconsole
To exit IPMI SOL, type <ENTER> "&" "."
================================================================================
```

7.1.3 Using cmsh to get BMC IPs

You can get the IP for a given device's out-of-band management interface using `cmsh`. In this example, we'll investigate why a node is in a down status using the node's BMC web interface.

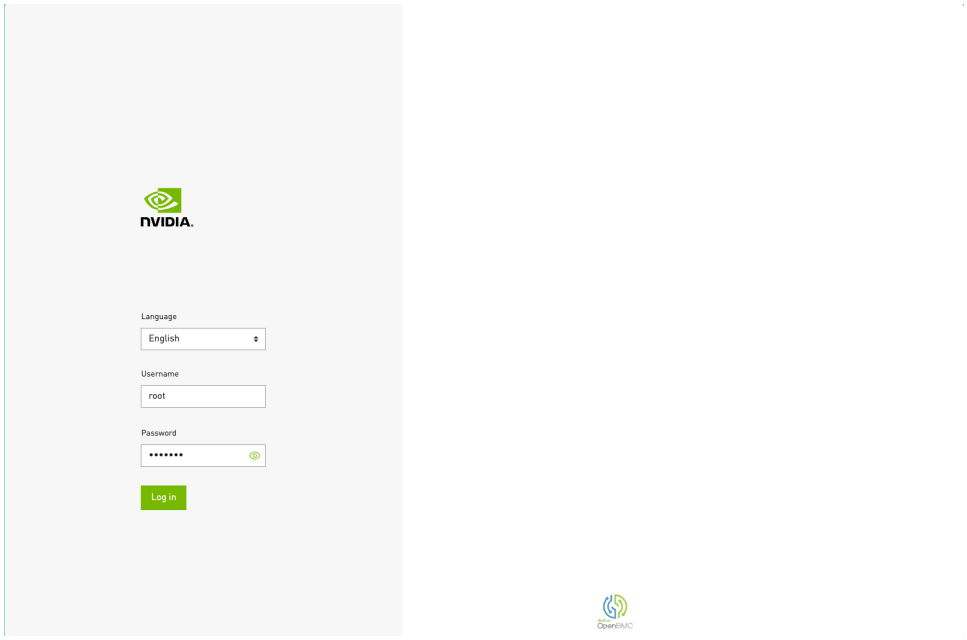
1. Get a list of physical nodes for a specific rack, filtering for a status of `DOWN`.

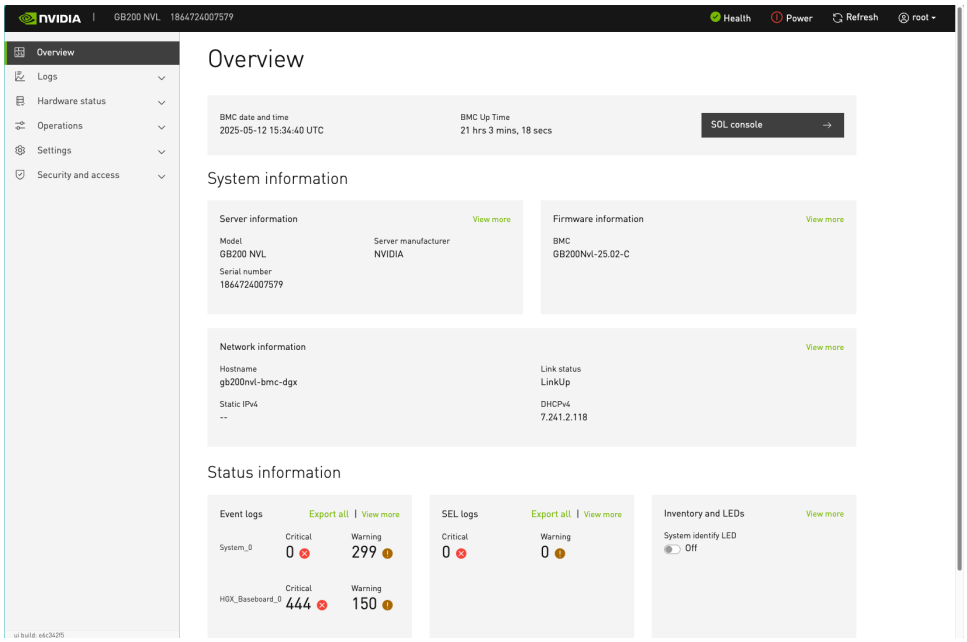
```
root@a03-p1-head-01:~# cmsh

[a03-p1-head-01]% device list -t physicalnode -r a06 | grep DOWN
PhysicalNode     a06-p1-dgx-02-c18  0E:63:D1:83:48:82  dgx-gb200        7.241.18.46      dgxnet1
[  DOWN  ], health check failed
```

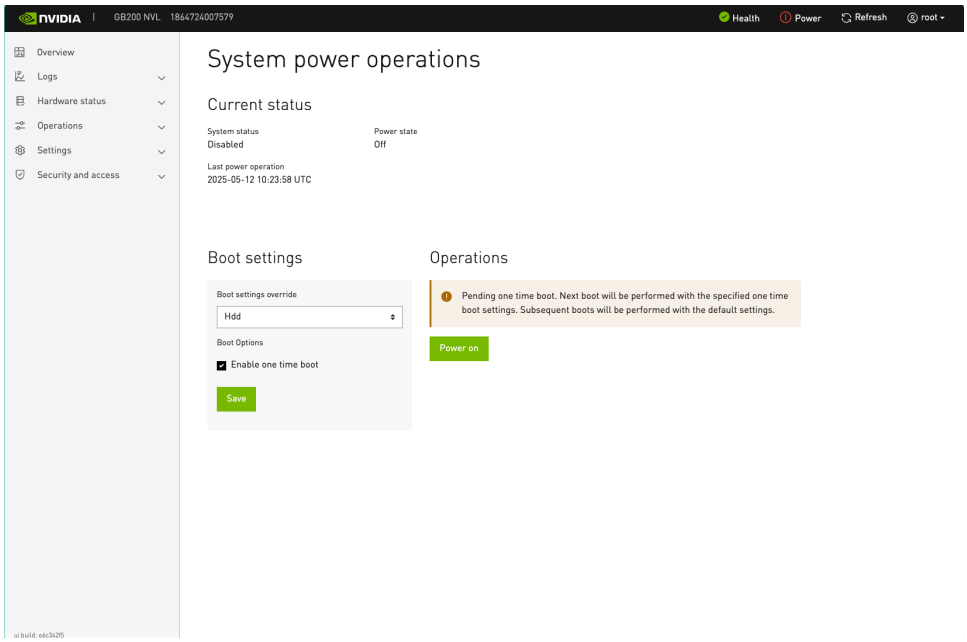
2. List the BMC interface.

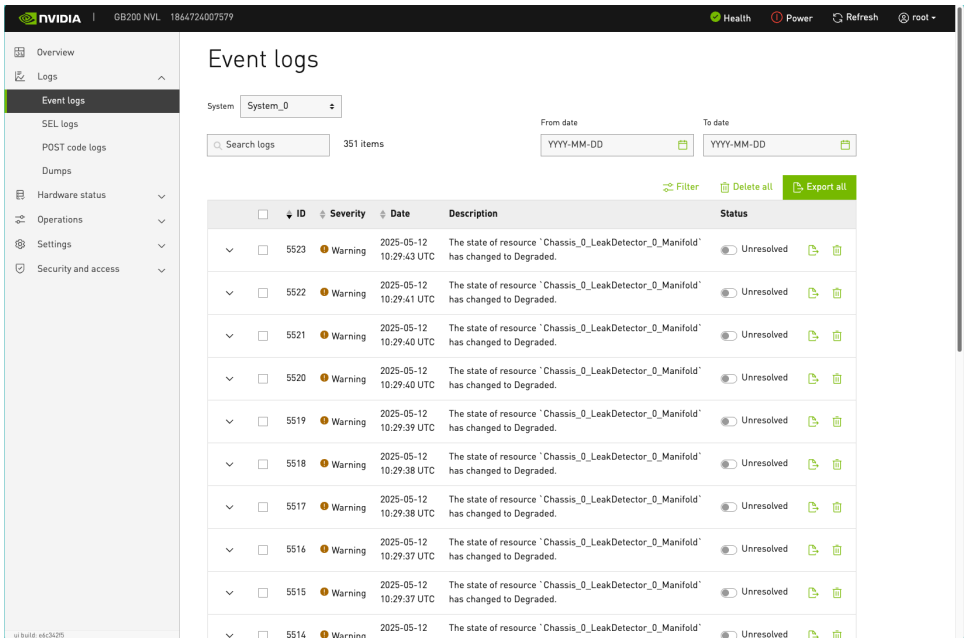
```
[a03-p1-head-01]% device use a06-p1-dgx-02-c18
[a03-p1-head-01->device[a06-p1-dgx-02-c18]]% interfaces
[a03-p1-head-01->device[a06-p1-dgx-02-c18]->interfaces]% list bmc
Type         Network device name  IP              Network          Start if
------------ -------------------- --------------- ---------------- --------
bmc          rf0                  7.241.2.118     ipminet2         always
```

3. With the IP, we can now use a browser to access the BMC interface.




NVIDIA | GB200 NVL 1864724007579
Health
Power
Refresh
root
Overview
Logs
Hardware status
Operations
Settings
Security and access
Overview
BMC date and time
2025-05-12 15:34:40 UTC
BMC Up Time
21 hrs 3 mins, 18 secs
SOL console
System information
Server information
View more
Model
GB200 NVL
Server manufacturer
NVIDIA
Serial number
1864724007579
Firmware information
View more
BMC
GB200Nvl-25.02-C
Network information
View more
Hostname
gb200nvl-bmc-dgx
Link status
LinkUp
Static IPv4
--
DHCPv4
7.241.2.118
Status information
Event logs
Export all | View more
Critical
Warning
System_0
0
299
Critical
Warning
HGX_Baseboard_0
444
150
SEL logs
Export all | View more
Critical
Warning
0
0
Inventory and LEDs
View more
System identify LED
Off


NVIDIA | GB200 NVL 1864724007579
Health
Power
Refresh
root
Overview
Logs
Hardware status
Operations
Settings
Security and access
System power operations
Current status
System status
Disabled
Power state
Off
Last power operation
2025-05-12 10:23:58 UTC
Boot settings
Boot settings override
Hdd
Boot Options
Enable one time boot
Save
Operations
Pending one time boot. Next boot will be performed with the specified one time boot settings. Subsequent boots will be performed with the default settings.
Power on


NVIDIA | GB200 NVL 1864724007579
Health
Power
Refresh
root
Overview
Logs
Event logs
SEL logs
POST code logs
Dumps
Hardware status
Operations
Settings
Security and access
Event logs
System
System_0
Search logs
351 items
From date
YYYY-MM-DD
To date
YYYY-MM-DD
Filter
Delete all
Export all
ID
Severity
Date
Description
Status
5523 Warning 2025-05-12 10:29:43 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5522 Warning 2025-05-12 10:29:41 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5521 Warning 2025-05-12 10:29:40 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5520 Warning 2025-05-12 10:29:40 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5519 Warning 2025-05-12 10:29:39 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5518 Warning 2025-05-12 10:29:38 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5517 Warning 2025-05-12 10:29:38 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5516 Warning 2025-05-12 10:29:37 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5515 Warning 2025-05-12 10:29:37 UTC The state of resource `Chassis_0_LeakDetector_0_Manifold` has changed to Degraded. Unresolved
5514 Warning 2025-05-12 The state of resource `Chassis_0_LeakDetector_0_Manifold` Unresolved

Chapter 8. High-Speed Fabric Management

8.1.1 NVLink and NVLink Switch

NVIDIA NVLink is a high-speed interconnect that allows multiple GPUs to communicate directly. In the past, the NVLink connections were limited to GPUs within a single node. With the addition of the NVLink Switch technology multi-node NVLink is now possible. This now allows multiple systems to interconnect over NVLink, using NVLink Switches, to form a large GPU and memory fabric within an NVLink Domain.

The NVIDIA GB200 NVL72 rack enables a 72 GPU NVLink Domain by default.This NVL72 NVLink Domain is set up as a single NVLink Partition. In other words, all GPUs in the NVL72 Domain have the potential to communicate with each other by default. In certain scenarios, a cluster administrator may need to create multiple NVLink Partitions within the NVL72 system. These are called User Partitions and they enable the ability to have multiple NVLink Partitions instead of one large Default Partition. This can provide a higher degree of multi-tenancy and isolation between workloads.

In most cases the Default Partition can be utilized, unless there are strong security requirements for certain workloads. IMEX and Slurm provide ways to dynamically split the NVLink Default Partition at job runtime and those will be discussed in more detail at the Slurm <provide link> section of this document.

It is worth noting that the connection between racks utilizes Infiniband but the connection within a rack will utilize NVLink (Table 4).

Table 4. NVLink definitions

Terms	Description
NVLink	High speed link to enable GPU to GPU communication
NVLink Partition	A subset of GPUs within a hardware isolation boundary
NVLink Domain	A set of nodes that can communicate over NVLink
Default Partition	Allows all GPUs in an NVLink Domain to communicate

User Partition	An NVLink partition that is created and managed by an admin

More details on NVLink Partitioning can be found in this GB200 NVL Partition Guide.

8.1.2 NVLink Management Software (NMX)

NVIDIA NMX is an integrated platform for managing and monitoring NVLink Switches, Domains, and Partitions.

It includes three main components:

- NMX-Telemetry (NMX-T)
- NMX-Manager (NMX-M)
- NMX-Controller (NNMX-C)

To manage NVLink Partitions gRCP API calls are sent to the NMX Controller or by utilizing the NVOS CLI on the NVSwitch.

The base case is highlighted below, where we utilize the NVOS CLI on `NVSWITCH-1`. In this case, `NVSWITCH-1` is our leader switch that runs NMX-C and NMX-T.

You can see the processes of the leader switch with the NVOS CLI and running `nv show cluster app`:

```
admin@a07-p1-nvsw-01-eth1.ipminet2.cluster:~$ nv show cluster app
Name            ID             Version                 Capabilities
Components Version                                              Status  Reason  Additional
Information          Summary
--------------  -------------  ----------------------  ------------------------------------------
-----  ----------------------------------------------------------------  ------  ------  ------------
-----------------  -------
nmx-controller  nmx-c-nvos     1.0.0_2025-04-11_15-23  sm, gfm, fib, gw-api
sm:2025.03.6, gfm:R570.133.15, fib-fe:1.0.1                     ok
CONTROL_PLANE_STATE_CONFIGURED
nmx-telemetry   nmx-telemetry  1.0.4                   nvl telemetry, gnmi aggregation, syslog
aggregation  nvl-telemetry:1.20.4, gnmi-aggregator:1.4.1, nmx-connector:1.4.1  ok
```

You can further check NVLink partition configuration:

```
admin@NVSWITCH-1:~$ nv show sdn partition
ID     Name               Num of GPUs  Health   Resiliency mode     Multicast groups limit  Partition
type  Summary
-----  -----------------  -----------  -------  ------------------  ----------------------  ---------
-----  -------
32766  Default Partition  72           healthy  adaptive_bandwidth  1024
gpuuid_based
```

Additionally, here is a way to see available commands for creating or modifying partitions:

```
admin@NVSWITCH-1:~$ nv list-commands | grep partition
nv show sdn partition
nv show sdn partition <partition-id>
nv show sdn partition <partition-id> location
nv show sdn partition <partition-id> uuid
```

```
nv action boot-next system image (partition1|partition2)
nv action update sdn partition <partition-id> [reroute]
nv action update sdn partition <partition-id> location <location-id> [no-reroute]
nv action update sdn partition <partition-id> uuid <uuid> [no-reroute]
nv action create sdn partition <partition-id> [name <value>] [resiliency-mode
(full_bandwidth|adaptive_bandwidth|user_action)] [mcast-limit (0-1024)] [location <location-id>]
[uuid (0-18446744073709551615)]
nv action delete sdn partition <partition-id>
nv action restore sdn partition <partition-id> location <location-id> [no-reroute]
nv action restore sdn partition <partition-id> uuid <uuid> [no-reroute]
```

More details on managing NVLink Switches can be found within the NVOS and NMX docs on the NVLink Networking docs page.

NMX-M provides for centralized monitoring of NVSwitches providing a REST API endpoint and Prometheus scrape endpoint per rack. BCM also integrates into NMX-M and will collect a subset of metrics.

```
[a03-p1-head-01]% partition
[a03-p1-head-01->partition[base]]% nmxmsettings
[a03-p1-head-01->partition[base]->nmxmsettings]% show
Parameter                      Value
------------------------------ ------------------------------------------------
Revision
Server                         7.241.0.145
User name                      rw-user
Password                       ********
Port                           443
Verify SSL                     no
CA certificate
Certificate
Private key
Prometheus metric forwarders   <0 in submode>
```

An example of updating the `Server` and `Password` fields in `nmxmsettings`.

```
[a03-p1-head-01->partition[base]->nmxmsettings]% set Server 7.241.0.144
[a03-p1-head-01->partition*[base*]->nmxmsettings*]% set Password ********
[a03-p1-head-01->partition*[base*]->nmxmsettings*]% commit
```

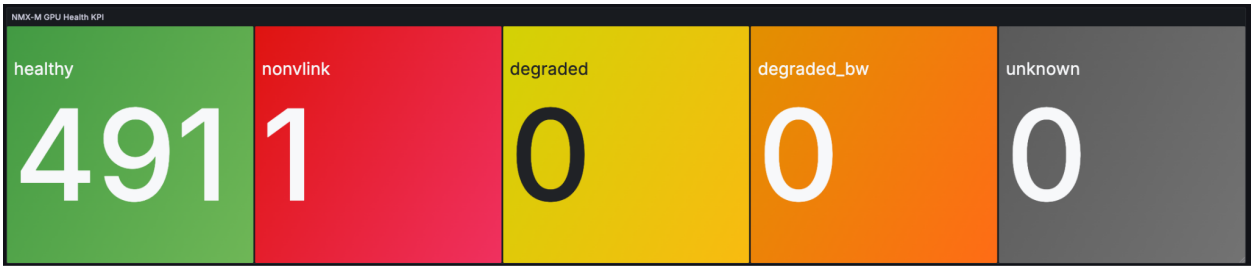
BCM exposes the counts from NMX-M's KPI endpoint as metrics

```
curl -sk https://localhost:8081/exporter | grep -Ev '# HELP|# TYPE' | grep -E '^nmxm' | cut -d '{' -
f1 | sort -u
nmxm_chassis_count
nmxm_compute_allocation_count
nmxm_compute_health_count
nmxm_compute_nodes_count
nmxm_domain_health_count
nmxm_gpu_health_count
nmxm_gpus_count
nmxm_ports_count
nmxm_switch_health_count
nmxm_switch_nodes_count
```

```
for i in $(curl -sk https://localhost:8081/exporter | grep -Ev '# HELP|# TYPE' | grep -E '^nmxm' |
cut -d '{' -f1 | sort -u); do curl -sk https://localhost:8081/exporter | grep -Ev '# HELP|# TYPE' |
grep $i; done

nmxm_chassis_count{base_type="Partition",name="base"} 7
nmxm_compute_allocation_count{base_type="Partition",name="base",parameter="full"} 126
nmxm_compute_allocation_count{base_type="Partition",name="base",parameter="all"} 126
nmxm_compute_allocation_count{base_type="Partition",name="base",parameter="partial"} 0
nmxm_compute_allocation_count{base_type="Partition",name="base",parameter="free"} 54
nmxm_compute_health_count{base_type="Partition",name="base",parameter="unhealthy"} 1
nmxm_compute_health_count{base_type="Partition",name="base",parameter="unknown"} 0
nmxm_compute_health_count{base_type="Partition",name="base",parameter="healthy"} 122
nmxm_compute_health_count{base_type="Partition",name="base",parameter="degraded"} 0
nmxm_compute_nodes_count{base_type="Partition",name="base"} 123
nmxm_domain_health_count{base_type="Partition",name="base",parameter="unknown"} 0
nmxm_domain_health_count{base_type="Partition",name="base",parameter="unhealthy"} 0
nmxm_domain_health_count{base_type="Partition",name="base",parameter="healthy"} 7
nmxm_domain_health_count{base_type="Partition",name="base",parameter="degraded"} 0
nmxm_gpu_health_count{base_type="Partition",name="base",parameter="degraded"} 0
nmxm_gpu_health_count{base_type="Partition",name="base",parameter="unknown"} 0
nmxm_gpu_health_count{base_type="Partition",name="base",parameter="healthy"} 491
nmxm_gpu_health_count{base_type="Partition",name="base",parameter="nonvlink"} 1
nmxm_gpu_health_count{base_type="Partition",name="base",parameter="degraded_bw"} 0
nmxm_gpus_count{base_type="Partition",name="base"} 492
nmxm_ports_count{base_type="Partition",name="base"} 15552
nmxm_switch_health_count{base_type="Partition",name="base",parameter="missing_nvlink"} 54
nmxm_switch_health_count{base_type="Partition",name="base",parameter="unknown"} 0
nmxm_switch_health_count{base_type="Partition",name="base",parameter="healthy"} 72
nmxm_switch_health_count{base_type="Partition",name="base",parameter="unhealthy"} 0
nmxm_switch_nodes_count{base_type="Partition",name="base"} 63
```

These count metrics are categorized by the label `parameter`.

An example dashboard, using the Single Stat panel consuming these KPI metrics:



8.1.3 InfiniBand 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

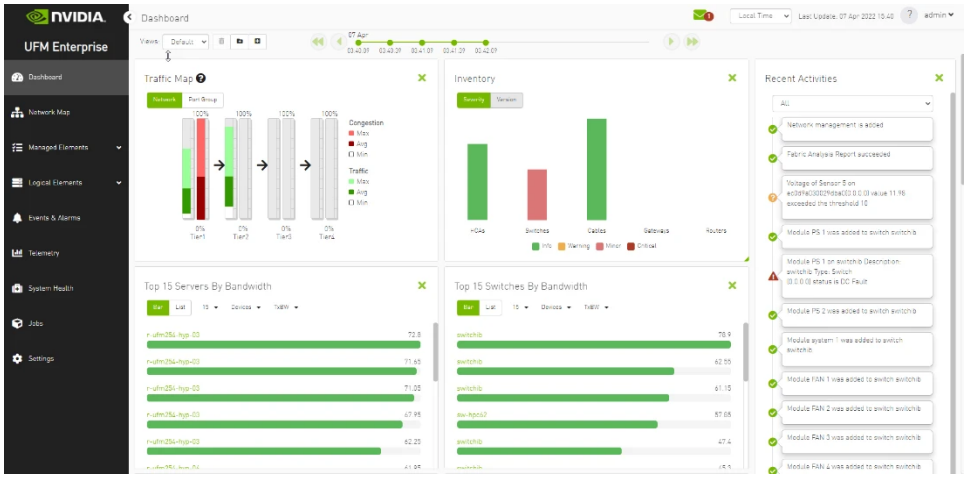
data center operators to efficiently monitor and operate the entire fabric, boost application performance, and maximize fabric resource utilization.

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.

8.1.3.1 UFM Dashboard

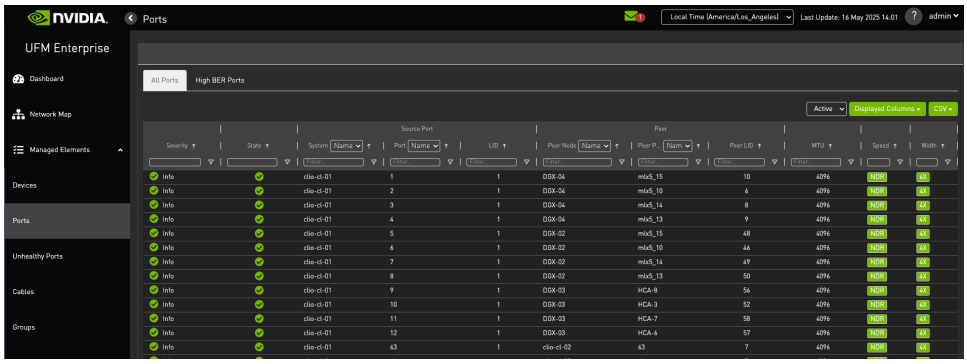
The dashboard window summarizes the fabric's status, including events, alarms, errors, traffic and statistics.

UFM Dashboard



8.1.3.2 UFM InfiniBand fabric ports view

Ports view provides a list of all ports in InfiniBand fabric and their speed/width and status.



8.1.3.3 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 http://nvidia.com/en-us/networking/infiniband/ufm/ for UFM documentation.

Chapter 9. Leak Detection

The GB200 system uses liquid cooling to achieve its scale-out density and efficiency. With a liquid, there is the chance for leaks and the GB200 system implements several fail-safes to avoid the worst effects of these. From an administrative point of view, these events will be collated within BCM.

9.1.1 Leak Example in cmsh

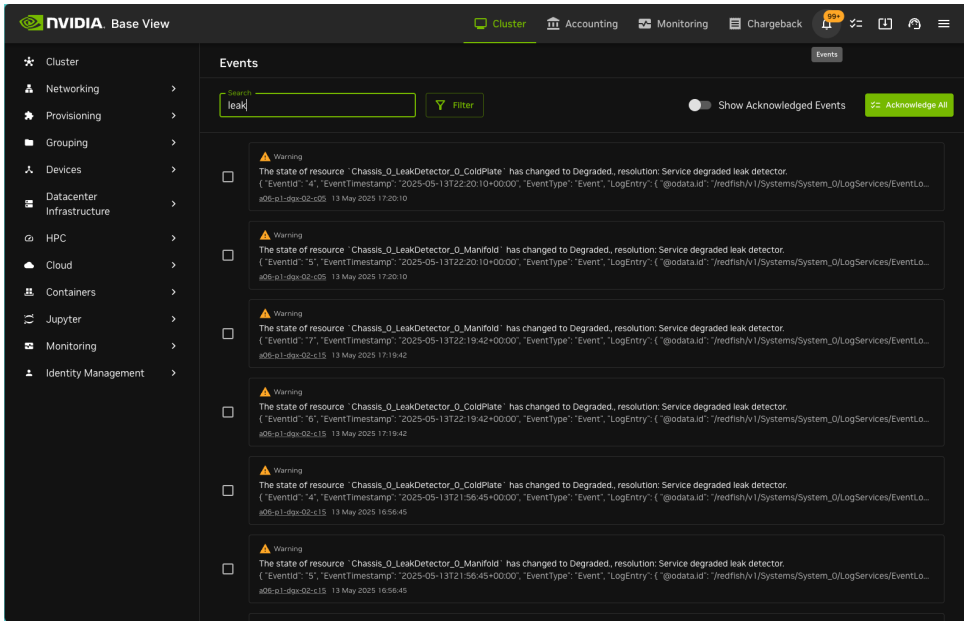
In this example, two nodes that are in a leak detection status.

```
[a03-p1-head-01]% events off broadcast
Broadcast events: off
[a03-p1-head-01]% device
[a03-p1-head-01->device]% list -r a06 -t physicalnode --status DOWN -v
Type             Hostname (key)     MAC                Category         IP               Network
Status
---------------- ------------------ ------------------ ---------------- ---------------- ------------
---- --------------------------------
PhysicalNode     a06-p1-dgx-02-c05  72:14:AF:B4:FC:7E  hwqa-gb200       7.241.18.33      dgxnet1
[  DOWN  ], leak:large (going

down timeout reached: 10m),

health check failed, health

check unknown, restart

required (interface:enP22p3s0f0np0)
PhysicalNode     a06-p1-dgx-02-c15  1A:A7:F7:32:98:AD  hwqa-gb200       7.241.18.43      dgxnet1
[  DOWN  ], leak:large (going

down timeout reached: 10m),

health check failed, health check unknown
```

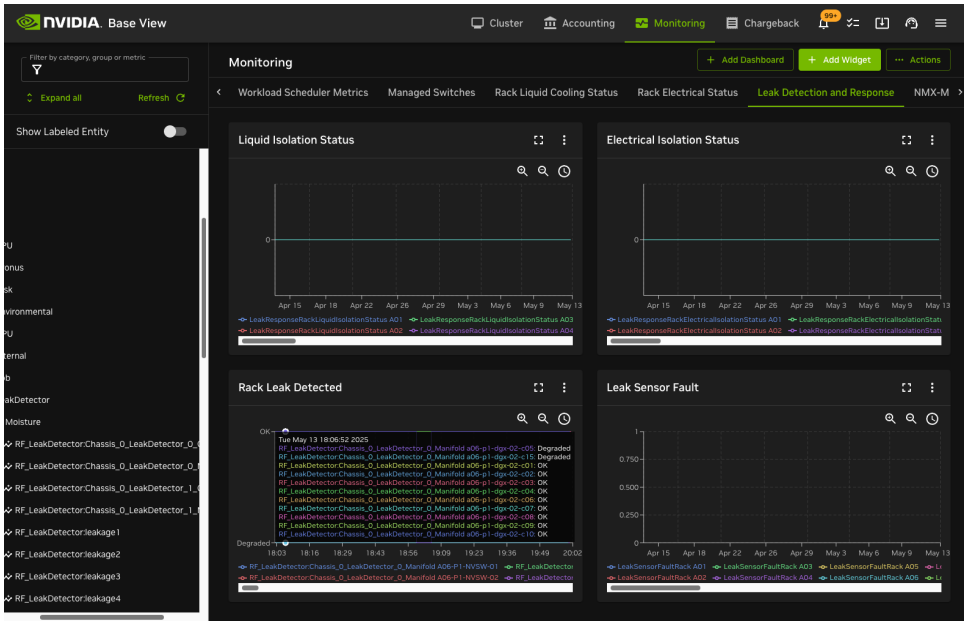
BCM also shows this within the UI in several locations.

9.1.2 Leak Example in BCM UI

Events

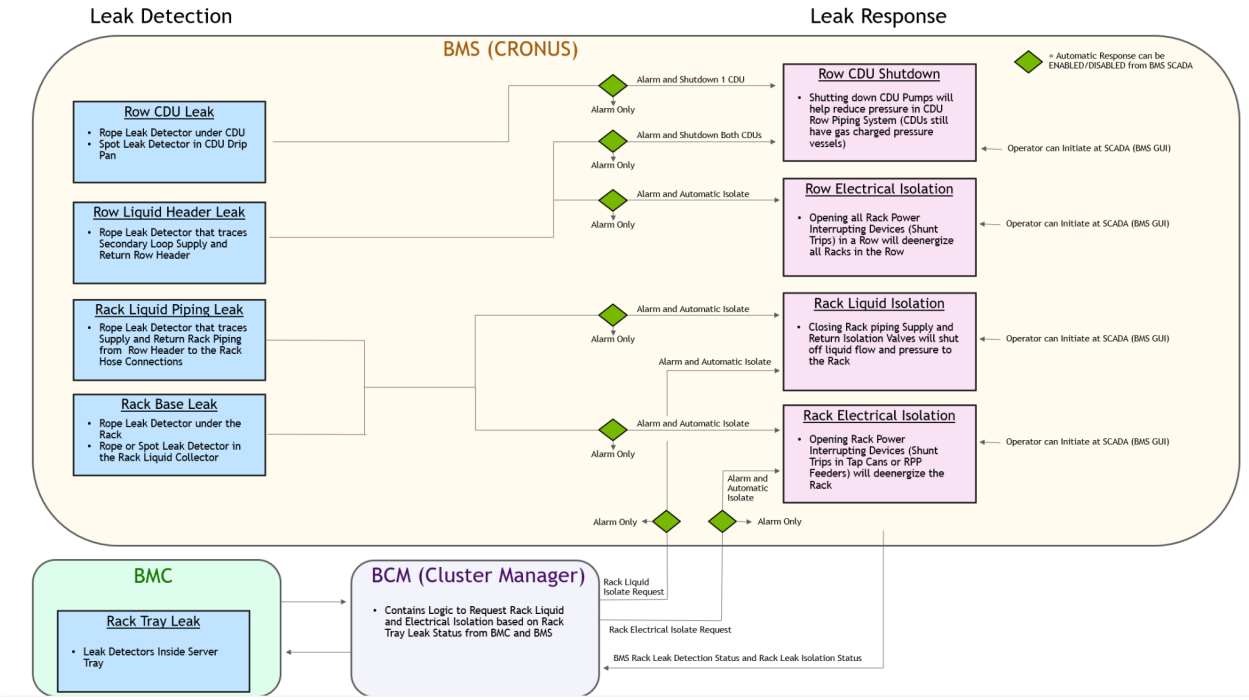


Monitoring -> Leak Detection and Response



BCM integrates with several data sources for leak detection:

- Redfish API at the device level via BMC.
- Building management System (BMS) at the rack and data center level.



These metrics are available through the UI, using the in-built reporting or as metrics via the Prometheus endpoint.

Examples of these metrics through the exporter endpoint:

```
for i in $(curl -sk https://localhost:8081/exporter | grep -i 'leak' | grep -Ev '# HELP|# TYPE' | cut -d '{' -f1 | sort -u); do echo Example metric $i; curl -sk https://localhost:8081/exporter | grep $i | head -n 3; done

Example metric deviceswithleaks
# HELP deviceswithleaks Number of devices that have detected a leak
# TYPE deviceswithleaks gauge
deviceswithleaks{base_type="Rack",name="B04"} 0
Example metric leakdetectrack
# HELP leakdetectrack undefined
# TYPE leakdetectrack gauge
leakdetectrack{base_type="Rack",name="A05"} 0
Example metric leakdetectracktray
# HELP leakdetectracktray undefined
# TYPE leakdetectracktray gauge
leakdetectracktray{base_type="Rack",name="A06"} 0
Example metric leakresponserackelectricalisolationstatus
# HELP leakresponserackelectricalisolationstatus undefined
# TYPE leakresponserackelectricalisolationstatus gauge
leakresponserackelectricalisolationstatus{base_type="Rack",name="A06"} 0
Example metric leakresponserackliquidisolationstatus
# HELP leakresponserackliquidisolationstatus undefined
# TYPE leakresponserackliquidisolationstatus gauge
leakresponserackliquidisolationstatus{base_type="Rack",name="A06"} 0
Example metric rf_chassis_0_leakdetector_0_coldplate
```

```
# HELP rf_chassis_0_leakdetector_0_coldplate Chassis 0 LeakDetector 0 ColdPlate
# TYPE rf_chassis_0_leakdetector_0_coldplate gauge
rf_chassis_0_leakdetector_0_coldplate{base_type="Device",category="perf-team",hostname="b05-p1-dgx-
05-c12",type="PhysicalNode"} 1.7047
Example metric rf_chassis_0_leakdetector_0_manifold
# HELP rf_chassis_0_leakdetector_0_manifold Chassis 0 LeakDetector 0 Manifold
# TYPE rf_chassis_0_leakdetector_0_manifold gauge
rf_chassis_0_leakdetector_0_manifold{base_type="Device",category="perf-team",hostname="b05-p1-dgx-05-
c12",type="PhysicalNode"} 1.7055
Example metric rf_chassis_0_leakdetector_1_coldplate
# HELP rf_chassis_0_leakdetector_1_coldplate Chassis 0 LeakDetector 1 ColdPlate
# TYPE rf_chassis_0_leakdetector_1_coldplate gauge
rf_chassis_0_leakdetector_1_coldplate{base_type="Device",category="perf-team",hostname="b05-p1-dgx-
05-c12",type="PhysicalNode",parameter="state"} 0
Example metric rf_chassis_0_leakdetector_1_manifold
# HELP rf_chassis_0_leakdetector_1_manifold Chassis 0 LeakDetector 1 Manifold
# TYPE rf_chassis_0_leakdetector_1_manifold gauge
rf_chassis_0_leakdetector_1_manifold{base_type="Device",category="perf-team",hostname="b05-p1-dgx-05-
c12",type="PhysicalNode",parameter="state"} 0
Example metric rf_ld_leakdetection
# HELP rf_ld_leakdetection Leak Detection Systems
# TYPE rf_ld_leakdetection gauge
rf_ld_leakdetection{base_type="Device",category="perf-team",hostname="b05-p1-dgx-05-
c12",type="PhysicalNode"} 0
Example metric rf_leakdetector
# HELP rf_leakdetector Chassis 0 LeakDetector 1 ColdPlate
# TYPE rf_leakdetector gauge
rf_leakdetector{base_type="Device",category="perf-team",hostname="b05-p1-dgx-05-
c12",type="PhysicalNode",parameter="Chassis_0_LeakDetector_1_ColdPlate"} 0
```

Chapter 10. Backups

10.1 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 15.4.8 of the BCM 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.

10.2 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.

10.2.1 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.
- Cannot find file when referring to an entire missing table.
- L:ocked 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.

10.2.2 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.

10.2.3 Cloning Databases

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

10.2.4 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 ungzipped and piped into the MySQL database for the user cmdaemon. The password, <password>, can be retrieved from /cm/local/apps/cmd/etc/cmd.conf, where it is configured in the DBPass directive (Appendix C 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 acknowledgement, 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 regards 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, and 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