CDW Documentation

nvidia_bcm-managing-nodes

Back to NVIDIA page

Back to BCM page

Topic: Managing Nodes & Software Images

What This Unit Covers

Creating, adding, and cloning nodes * Understanding node categories vs node groups * Creating, cloning, updating, and assigning software images * Synchronizing changes in both directions:

- image to node
- node to image

Practical considerations for safe rollout and rollback of changes

1. **Devices vs Nodes (Foundation Concept)**

In NVIDIA Base Command Manager, a **device** represents a cluster component. * Device types can include:

- Head Node
- Physical Node
- Virtual Node
- Cloud Node
- GPU Unit
- Chassis
- Switches:

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

Devices have properties such as rack position, hostname, and switch port, and the operations you can perform depend on the device type. * Practical distinction:

- A **node** is the device type BCM treats as an operating-system host for provisioning and cluster operations.
- A regular node reaches the **UP** state when that node’s CMDaemon connects back to the head node CMDaemon.

2. **Supported OS Platforms**

BCM 11 is available on the following Linux platforms:

- Red Hat Enterprise Linux and derivatives such as Rocky Linux

- 8.x
- 9.x
- SUSE Linux Enterprise Server 15

- Ubuntu
- 22.04
- 24.04
- DGX OS 7

By default, the node image OS matches the head node OS, but BCM also supports changing the image OS later. * Mixed-distribution clusters are supported, but NVIDIA notes they are harder to manage and more likely to introduce issues than a consistent distribution strategy.

3. **Node Types**

Head Node

The head node is the central control system for the cluster. It manages devices such as compute nodes, switches, and PDUs.

It typically provides core cluster services such as:

- central data storage
- workload management
- user management
- DNS
- DHCP

Minimum hardware requirements for a small cluster:

- x86-64 or ARMv8 CPU
- 4 GB RAM for x86
- 16 GB RAM for ARMv8
- 80 GB disk space
- 2 Gigabit Ethernet NICs for the common Type 1 topology

NVIDIA also notes that while 4 GB is the technical minimum for an x86 head node, a standard bare-metal installation generally runs best with at least 8 GB RAM.

Compute Node

Regular nodes are the non-head nodes managed by the head node. Most regular nodes in a cluster are compute nodes. hey typically install automatically through BCM’s node provisioning system. Minimum compute-node hardware requirements listed by NVIDIA:

- x86-64 or ARMv8 CPU
- 1 GB RAM
- at least 4 GB RAM recommended for diskless nodes
- 1 Gigabit Ethernet NIC

4. **Why Node Management Matters**

BCM is designed so that nodes can be returned to a **known state** through provisioning and synchronization from a software image. This reduces the need to log into compute nodes directly. It also makes node replacement easier after hardware failure, because the replacement node can be reprovisioned from the head node rather than rebuilt manually.

5. **Node Management Operations**

Common day-to-day node operations include:

- adding nodes
- cloning nodes
- removing nodes
- powering nodes on or off
- rebooting nodes
- updating running nodes from an image
- reinstalling nodes from the latest image
- reassigning nodes to different categories
- repurposing nodes by changing their configuration, category, or assigned roles

Practical install/update note:

a reboot with default settings usually applies the latest image with an **AUTO** install .**Reinstall node** applies the latest image with a **FULL** install and may take longer.

6. **Creating Nodes**

Using CMSH

In `device` mode, node objects can be added with the `add` command.

NVIDIA’s documented example is:

- `add physicalnode node002 10.141.0.2`
- After creating or modifying the object, changes are not permanent until `commit` is run.

Using Base View

Base View provides equivalent add functionality through the GUI.

Important configuration details when creating nodes

Some required values may still need to be filled in before the object validates, such as MAC address information.

For regular nodes, BCM expects network details for:

- the network the node boots from
- the network used to manage the node

In many clusters, a regular node has one interface that serves both functions:

- as **BOOTIF** during the pre-init stage
- as the management interface after boot

The BOOTIF address is typically provided by DHCP, while the management interface can be configured with a static IP by the administrator.

7. **Node Creation Wizard**

Base View also includes a **Node Creation Wizard** for adding many nodes more efficiently. This is especially useful at scale.

Important distinction:

- the wizard creates node objects and assigns names
- it leaves the MAC address unfilled, creating a **placeholder**
- later, BCM can associate the physical MAC address and switch port with that placeholder node

This is different from the device identification resource, which is used to match discovered MAC addresses and switch ports to node identities.

8. **Cloning Nodes (Preferred for Reuse)**

Cloning is a convenient way to duplicate a fully configured object instead of rebuilding it by hand.

NVIDIA's documented cmsh example:

- `clone node100 node101`

Important caveats from the manual:

- Ethernet switch settings are **not** cloned and must be set manually
- BCM attempts to assign a new IP address automatically using heuristics
- that automatic IP result is only best-effort, so the administrator should inspect the cloned object carefully before relying on it

This is one of the most useful time-saving operations when building out larger clusters or creating patterned node definitions.

9. **Node Categories (Very Important)**

A **node category** is a group of regular nodes that share the same configuration.

Categories are one of the main scaling mechanisms in BCM because they let you:

- configure many nodes at once
- operate on many nodes at once

Every regular node is in **exactly one category** at all times. By default, nodes are placed in the **default** category.

Nodes are typically separated into categories based on:

- hardware profile
- intended function
- differences in configuration such as:
  - monitoring setup
  - disk layout
  - role assignment

This is why categories are the main administrative control layer in BCM.

10. **Category Inheritance**

BCM uses category-level values as the default configuration for nodes.

For **non-Boolean** values:

- a node inherits the value from its category
- that inherited value can still be overridden at the node level

For **Boolean** values:

- BCM does not use simple inheritance in the same way
- the category Boolean and node Boolean are combined with Boolean OR logic

This matters because it explains why some settings feel directly inherited while others behave more like enablement flags.

11. **Categories and Software Images Do Not Have to Match One-to-One**

A category has a software image as one of its configuration properties. But NVIDIA explicitly notes there is **no requirement** for a one-to-one relationship between categories and images. This means:

- multiple categories can use the same image
- a category can use an image while certain image-related behavior is overridden at category level

This flexibility is powerful, but it also means administrators must be deliberate about category design so image assignments stay understandable.

12. **Node Groups (Different from Categories)**

A **node group** is a convenience grouping of nodes. A node group can contain any mix of nodes, regardless of category. A node can belong to **zero or more** node groups at the same time. * Node groups are mainly for operating on sets of nodes together. They are **not** for shared configuration, because the nodes in a group do not necessarily have the same config. That is the core difference:

- **category** = configuration and standardized behavior
- **node group** = convenience and bulk operations

NVIDIA also notes node groups have use in provisioning-role configuration.

13. **Software Images (Critical Concept)**

A **software image** is the blueprint for the contents of the local filesystems on a regular node. In practice, a software image is a directory on the head node that contains a full Linux filesystem. BCM image directories are commonly managed under `/cm/images`. In a standard installation:

- the image is based on the same parent distribution as the head node
- the default image name is typically `default-image`

When a regular node boots, the provisioning system sets it up using a copy of that software image. Once the node is fully booted, BCM can also re-synchronize the node filesystem from the image without requiring a reboot. Software images can also be **locked** to prevent nodes from picking them up until the image is unlocked.

14. **Why Cloning Software Images Is Best Practice**

NVIDIA explicitly recommends cloning images before making larger changes. Reasons:

- safer testing
- controlled rollout
- rolling-update strategy
- easier rollback if problems appear

A careful administrator typically clones a known-good image before modifying it. This is especially important in production clusters where changing the active image directly increases risk. ([NVIDIA Docs][1])

15. **Updating Software Images**

General principle

Software images can be changed with regular Linux tools and commands. NVIDIA documents image modification through chroot-based workflows.

Examples from the BCM manual

For RHEL and derivatives:

- `chroot /cm/images/default-image`
- `yum update`

For SLES:

- `chroot /cm/images/default-image`
- `zypper up` *

For Ubuntu:

- `cm-chroot-sw-img /cm/images/default-image`
- `apt update; apt upgrade`

For non-packaged software, NVIDIA also documents direct placement into the image and chroot-based installation. When appropriate, NVIDIA recommends organizing shared software content under:

- `/cm/shared/apps`
- `/cm/shared/docs`
- `/cm/shared/examples`
- `/cm/shared/licenses`
- `/cm/shared/modulefiles`

16. **Updating Running Nodes from the Software Image**

CMSH

BCM uses `imageupdate` to synchronize a running node from its image. NVIDIA’s documented example:

- `imageupdate -n node001`

By default, this is a **dry run**. BCM tells you to review the result with `synclog`, then rerun with:

- `imageupdate -n node001 -w`

The `-w` switch performs the actual write.

Base View

Base View provides an **Update Node** action for the same operation.

Important operational notes

If provisioners have not been updated recently, BCM may first run `updateprovisioners`. NVIDIA notes that running `updateprovisioners` yourself just before `imageupdate` often makes sense, especially if a new image was recently created. For more extensive changes, NVIDIA says it can be safer to reboot nodes rather than rely only on `imageupdate`, because rebooting ensures the node boots into the latest image and restarts services cleanly.

17. **If You Change an Image Outside Base View or CMSH**

If the image is changed through CMDaemon-aware front ends such as Base View or cmsh, BCM handles provisioning-image propagation automatically. If the image is changed **outside** those front ends, such as by copying files directly into the image from a bash prompt, NVIDIA says `updateprovisioners` should be run manually. This is an easy thing to forget and is a common reason admins do not see changes propagate the way they expect.

18. **Node-to-Image Synchronization**

What it is

This is the reverse direction:

- instead of pushing image changes to a node
- you pull the state of a configured node back into an image

Why it is used

- NVIDIA describes this as helpful when the software is more complex and is easier to install and test directly on a node first.
- After validation, that node state can be saved back to an image for broader rollout.

Critical warning

- Until the new image is actually saved, the node loses those local alterations on reboot and returns to the old image.
- NVIDIA also warns that the cleanest and recommended method is still to modify the image directly when possible.
- Syncing node-to-image can capture unwanted changes, so results should be reviewed carefully before broad deployment.

19. **grabimage Command**

In cmsh, the command for node-to-image synchronization is `grabimage`.

`grabimage` without `-i`

- grabs back to the current image

`*grabimage -i <image>*`

- writes to another image
- that destination image must already exist or be cloned beforehand

`*grabimage*` *without* `*-w*`

- is a dry run

`*grabimage -w*`

- performs the actual write

NVIDIA's documented example:

`*grabimage -w -i default-image1 node001*`

BCM also distinguishes exclude lists for:

- grabbing back to the original image
- grabbing into a new image

20. **What grabimage Includes and Excludes**

NVIDIA notes that node-to-image sync:

- automatically excludes network mounts
- automatically excludes parallel filesystems such as Lustre and GPFS
- includes regular disks mounted on the node itself

Even with exclusion lists, NVIDIA warns that some unwanted changes may still get captured. That is why grabbing from a running node is powerful, but also riskier than editing the image directly.

21. **Advanced Note: Software Image Revisions**

BCM also supports revision control in `softwareimage` mode. A new revision can be created with `newrevision`. Revisions are named in the form:

`*<parent-image>@<revision-number>*`

NVIDIA documents that a new Btrfs subvolume is created for that revision. Categories can be pointed to:

- the parent image
- a specific revision such as `default-image@1`
- `@latest`, so the category tracks the most recent revision

This is an advanced but very useful feature for controlled lifecycle management of images.

22. **Key Takeaways**

- **Device** is the broad object type; **node** is the OS-host device type you provision and manage as

a cluster host.
- **Every regular node belongs to exactly one category** at all times. Categories are the primary mechanism for shared configuration and bulk administration.
- **Node groups are not categories.** Groups are for convenience and operations, not shared configuration.
- **Software image = node filesystem blueprint.** It lives on the head node and is used for provisioning and synchronization.
- **Clone before you change.** NVIDIA explicitly recommends cloning working images before making larger modifications. ([NVIDIA Docs][1])
- **imageupdate is dry-run first.** Review the sync result, then use `-w` to apply changes.
- **grabimage is the reverse flow.** It is useful, but riskier than editing the image directly, so validate carefully.
- **A recently changed image may still require `updateprovisioners` awareness.** This is a very real troubleshooting point in practice.

23. **What Actually Matters in Real Life**

Node not behaving as expected

- Check the node’s category first
- Then confirm which software image the node is actually using
- Then verify whether category-level settings are overriding what you thought the node inherited

Update did not apply

- You may have only run the dry run
- You may have forgotten the `-w` flag
- You may need to inspect `synclog`

You may need to ensure provisioners were updated before expecting nodes to receive recent image changes

Cluster inconsistency

- Someone may have changed a running node directly
- Those changes may not have been written back to the image
- Or they may have been written back with grabimage and unintentionally captured extra changes
- This is why disciplined image management matters so much in BCM

[1]: https://docs.nvidia.com/dgx/baseos-on-bcm-install-guide/managing-images-bcm.html “Managing Images in BCM — Creating a BaseOS image in Base Command Manager”

Back to NVIDIA page

Back to BCM page