



# **MNNVL User Guide**

*Release 1.1*

**NVIDIA Corporation**

**Oct 30, 2025**

# Contents

<b>1 Overview</b>	<b>1</b>
1.1 Software Overview	2
1.1.1 CUDA Software Platform	2
1.1.2 Internode Memory Exchange Service	2
1.1.3 DOCA Software Framework	2
1.1.4 NVIDIA Communication Libraries	3
1.1.5 Data Center GPU Manager	3
1.1.6 NVIDIA Switch OS	3
1.1.6.1 NMX-Controller	3
1.1.6.2 NMX-Telemetry	4
<b>2 Deploying</b>	<b>5</b>
2.1 Operating System Requirements	5
2.2 Installing NVIDIA Software	5
2.2.1 Remove Existing Packages	5
2.2.2 Clean the Local APT Cache	6
2.2.3 Unload the Kernel Modules	6
2.2.4 Install the NVIDIA Linux Kernel and Packages	6
2.2.5 Install the Kernel Build Packages	7
2.2.6 Install DOCA Host	7
2.2.7 Install IMEX and the NVIDIA GPU Driver	8
2.2.8 Configure the NVIDIA Software Packages	9
2.2.8.1 Enable Profiling	9
2.2.8.2 Enable the IMEX Control Channel	9
2.2.8.3 Configure IMEX Peers	10
2.2.8.4 Enable the NVIDIA Persistence Daemon	10
2.2.8.5 Enable the IMEX Daemon	11
2.2.8.6 Rebuild the initramfs	11
2.2.8.7 Reboot the Compute Tray	11
2.3 Switch Tray Software	11
2.3.1 Enable the NVLink Cluster	12
<b>3 Verifying</b>	<b>14</b>
3.1 Compute Tray Verification	14
3.1.1 IMEX	14
3.1.2 Linux Kernel	14
3.1.3 GPU Driver	14
3.1.4 NVLink Status	15
3.1.5 Topology Status	18
3.1.6 NVIDIA Persistenced Service Status	18
3.1.7 NVIDIA IMEX Service Status	19
3.1.8 IMEX Domain Status	19
3.2 Switch Tray Verification	20

3.2.1	System Health . . . . .	20
3.2.2	Control Plane Check . . . . .	20
3.2.2.1	NMX-T Health Check . . . . .	20
<b>4</b>	<b>Rack Reboot Sequence</b>	<b>22</b>
4.1	Cold Reboot Sequence . . . . .	22
4.2	Warm Reboot Sequence . . . . .	23
4.3	Post Reboot Verification . . . . .	24
<b>5</b>	<b>Multinode Testing</b>	<b>25</b>
5.1	Installing Required Packages . . . . .	25
5.2	Configure SSH . . . . .	25
5.2.1	Creating and Copy SSH Keys . . . . .	25
5.2.2	Configuring the SSH Server . . . . .	26
5.3	Build nvbandwidth . . . . .	26
5.4	Run nvbandwidth . . . . .	27
<b>6</b>	<b>Troubleshooting</b>	<b>29</b>
6.1	Troubleshooting the Compute Tray . . . . .	29
6.1.1	Driver Not Loading . . . . .	29
6.2	Troubleshooting the Switch Tray . . . . .	29
6.2.1	Switch Health . . . . .	29
6.2.2	BMC Firmware Issues . . . . .	29
6.3	Troubleshooting the NVLink Fabric . . . . .	30
6.3.1	NMX-Controller on Multiple Trays . . . . .	30
6.3.2	Viewing Subnet Manager Logs . . . . .	30
6.3.3	Viewing Global Fabric Manager Logs . . . . .	30
6.3.4	Viewing IMEX Logs . . . . .	30
6.3.5	IMEX TRANSIENT_FAILURE State . . . . .	31

---

# Chapter 1. Overview

The **NVIDIA Multi-Node NVLink** (MNNVL) system is a rack-scale solution combining NVIDIA ARM-based Grace CPUs, NVIDIA Blackwell GPUs and NVIDIA NVSwitches connected through a passive backplane assembly.

NVIDIA MNNVL systems enables higher GPU power and performance, improved serviceability and larger NVLink domains.

The MNNVL rack contains:

- ▶ 18x compute trays, each with 2 Grace CPUs and 4 Blackwell GPUs
- ▶ 9x NVLink switch trays providing 57.6 terabits per second (Tbps) of full duplex bandwidth over fifth generation NVLink connections.

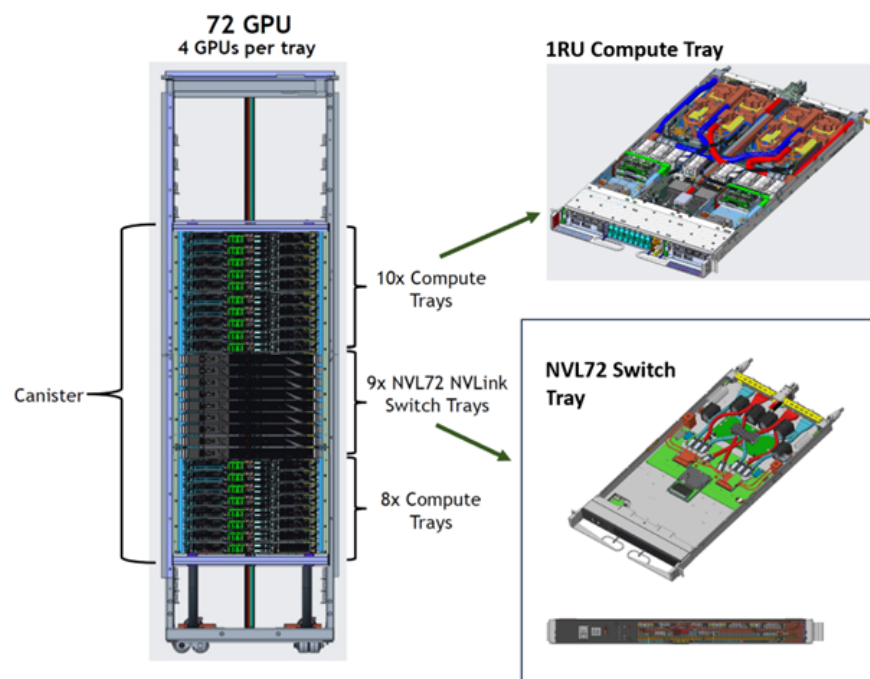


Figure 1.1: Illustration of an NVL72 rack.

## 1.1. Software Overview

A MNNVL system consists of software and firmware components across both the compute nodes and NVLink switches.

### 1.1.1. CUDA Software Platform

NVIDIA CUDA® enables applications to use GPUs for accelerated computing and provides more than 150 libraries to create high-performance applications across programming languages and use cases. CUDA provides the tools to optimize and debug GPU accelerated application.

CUDA-enabled applications automatically leverage the NVLink network to provide distributed processing across every GPU in the MNNVL system.

Refer to the [NVIDIA CUDA documentation](#) for more information about CUDA.

### 1.1.2. Internode Memory Exchange Service

NVIDIA Internode Memory Exchange (IMEX) is a secure service facilitating the mapping of GPU memory over NVLink between the GPUs in an NVLink domain. The service starts on all nodes in the NVLink domain during system startup or during the launch of the CUDA job.

IMEX manages the lifecycle of application shared memory and registers for memory import events with the GPU driver. IMEX doesn't rely on CUDA but communicates across compute nodes using TCP and gRPC.

#### Tip

IMEX provides inter-GPU memory mapping information over TCP/IP on the data or management Ethernet network. Inter-GPU memory read and writes are over the NVLink backplane.

IMEX calls the collection of nodes sharing memory an *IMEX domain*.

#### Tip

Single node systems or processes don't require IMEX.

Compute nodes communicate using an *IMEX channel*, a logical `/dev` character device. CUDA applications with permissions to access to the IMEX channel can securely share memory across nodes.

Refer to [IMEX Service for NVLink Networks](#) for more information.

### 1.1.3. DOCA Software Framework

The DOCA framework unlocks the potential of the NVIDIA® BlueField® networking platform by enabling the rapid creation of applications and services that offload, accelerate, and isolate datacenter workloads. Developers can create software-defined, cloud-native, DPU- and SuperNIC-accelerated services with zero-trust protection, which addresses the performance and security demands of modern data centers.

DOCA-Host includes the host drivers and tools for your BlueField and ConnectX® devices in the rack. Refer to the [NVIDIA DOCA Software Framework](#) for more information.

## 1.1.4. NVIDIA Communication Libraries

NVIDIA's communication libraries like NVIDIA Collective Communications Library (NCCL), NVIDIA NVSHMEM, and Unified Communication X (UCX) provide a framework for efficient and scalable GPU-to-GPU communication. These communication libraries support multiple interconnect technologies including PCIe, NVLink, InfiniBand, RoCE and more.

NCCL, provides GPU-to-GPU communication primitives that are topology-aware, removing the need to optimize applications for specific systems. NCCL implements collective communication and point-to-point send and receive primitives for intra- and inter-node data transfers. Refer to the [NCCL documentation](#) for more information.

NVSHMEM provides a partitioned global address space (PGAS) for data that spans the memory of multiple GPUs. NVSHMEM provides device APIs enabling low latency communication between GPUs and enables for fine grained or fused overlap of compute and communications from the CUDA kernel. Refer to the [NVIDIA NVSHMEM documentation](#) for more information.

UCX is a framework that provides a common set of CPU-initiated tag matching, active messaging, and remote memory access APIs over a variety of network protocols. Refer to [NVIDIA UCX documentation](#) for more information.

## 1.1.5. Data Center GPU Manager

NVIDIA Data Center GPU Manager (DCGM) is a suite of tools that manage and monitor NVIDIA data-center GPUs and NVSwitches in MNNVL systems. DCGM includes active health monitoring, comprehensive diagnostics, system alerts, and governance policies. DCGM can be a standalone monitoring platform or integrate into existing tools from NVIDIA partners. Refer to the [NVIDIA DCGM documentation](#) for more information.

## 1.1.6. NVIDIA Switch OS

The NVLink switch tray comes with the NVIDIA NVSwitch™ operating system (NVOS) enabling the management and configuration of NVIDIA's switch system platforms. NVOS provides a suite of management options, incorporating an industry-standard command line interface (CLI) and OpenAPI (Swagger) interface, enabling system configuration and management.

NVOS includes the Subnet Manager (SM), Fabric Manager (FM), NMX-Controller (NMX-C), NMX-Telemetry (NMX-T) and the NVSwitch firmware.

### 1.1.6.1 NMX-Controller

NMX-Controller (NMX-C) is a cluster application for fabric SDN services.

NMX-C manages SDN services configuration and operation through a gRPC interface with external manager or the NVOS CLI or REST APIs.

Refer to the [NMX-C documentation](#) for more information.

**Tip**

The NVOS software image includes the NMX-C application. There's no standalone software to install.

### 1.1.6.1.1 Fabric Manager

NVIDIA Fabric Manager (FM) configures the NVSwitch memory fabric to form a single memory fabric among the participating GPUs. FM coordinates with the GPU driver to initialize the GPUs and configure NVLink port mapping and routing between compute nodes and NVLink switches. FM provides fabric performance and error monitoring.

**Tip**

The NVOS software image includes the FM application. There's no standalone software to install.

### 1.1.6.1.2 NVLink Subnet Manager

NVLink Subnet Manager (NVLSM), based on the InfiniBand Subnet Manager, manages the NVSwitches and NVLink connections. NVLSM discovers the NVLink topology and assigns logical identifiers (LID) to the GPU and NVSwitch NVLink ports. NVLSM calculates and programs the NVSwitch forwarding table and monitors any changes in the NVLink fabric.

**Tip**

The NVOS software image includes the NVLSM application. There's no standalone software to install.

### 1.1.6.2 NMX-Telemetry

NMX-Telemetry (NMX-T) collects, aggregates, and transmits telemetry data from NVLink switches. Refer to the [NMX-T documentation](#) for more information.

---

# Chapter 2. Deploying

This section describes how to install the software components for your MNNVL system.

Follow these instructions to configure an MNNVL compute tray on each compute tray in the MNNVL rack.

## 2.1. Operating System Requirements

The MNNVL compute trays use NVIDIA Grace, ARM-based, CPUs.

NVIDIA provides guides for installing Linux distributions on Grace CPU platforms at the following links:

- ▶ [NVIDIA Grace Software with Ubuntu Installation Guide](#)
- ▶ [NVIDIA Grace Software with Red Hat Enterprise Linux 9 Installation Guide](#)
- ▶ [NVIDIA Grace Software with SUSE Linux Enterprise Server 15 Installation Guide](#)

### Important

Before installing NVIDIA software follow the directions from your manufacturer to update all system firmware.

## 2.2. Installing NVIDIA Software

The following steps describe how to install the NVIDIA software packages on Ubuntu 24.04.

### Important

Follow these directions for each compute tray in the MNNVL rack.

### 2.2.1. Remove Existing Packages

Remove any existing NVIDIA packages:

```
$ sudo apt purge nvidia-*  
$ sudo apt purge cuda*
```

(continues on next page)

(continued from previous page)

```
$ sudo apt purge libxnvctrl0
$ sudo apt purge glx-*
```

**Note**

These packages are not installed on a new Ubuntu 24.04 install. The apt command will not find the packages when running `apt purge`.

## 2.2.2. Clean the Local APT Cache

Clean and reset the existing APT cache, run the following commands.

```
$ sudo apt autoremove
$ sudo apt autoclean
```

## 2.2.3. Unload the Kernel Modules

Unload the kernel modules, run the following commands.

```
$ sudo modprobe -r mods
$ sudo modprobe -r nvidia_uvm
$ sudo modprobe -r nvidia_drm
$ sudo modprobe -r nvidia_modeset
$ sudo modprobe -r nvidia_vgpu_vfio
$ sudo modprobe -r nvidia_peermem
$ sudo modprobe -r nvidia
$ sudo modprobe -r nvidiafb
```

## 2.2.4. Install the NVIDIA Linux Kernel and Packages

For optimal performance, the Grace-Blackwell systems require the NVIDIA-specific Linux kernel.

**Note**

If you use a custom kernel refer to the [NVIDIA Grace Platform Support Software Patches and Congurations](#) for a list of kernel customizations required for the Grace CPU.

1. Remove the existing Linux image, headers and modules.

```
sudo apt purge linux-image-$(uname -r) linux-headers-$(uname -r) linux-
->modules-$(uname -r)
```

2. Install the NVIDIA Linux kernel.

```
sudo apt update
sudo apt install linux-nvidia-64k-hwe-24.04
```

**Important**

Change the 24.04 value to match your Ubuntu version.

3. Hold the kernel version

```
apt-mark hold linux-nvidia-64k-hwe-24.04
```

**Important**

Change the 24.04 value to match your Ubuntu version.

4. Reboot the system to load the new NVIDIA kernel.

```
sudo reboot
```

## 2.2.5. Install the Kernel Build Packages

Install packages required to install the NVIDIA drivers:

```
sudo apt update -y && sudo apt install -y gcc dkms make
```

## 2.2.6. Install DOCA Host

The DOCA Host package provides the OS drivers for the BlueField-3 and ConnectX adapters.

1. Remove older versions of DOCA and adapter drivers.

**Note**

If there were no drivers previously installed the `ofed_uninstall.sh` script won't be found.

```
$ for f in $( dpkg --get-selections | grep -E 'dca|flexio|dca-gdbserver|dca-
→stats|dpaeumgmt' | awk '{print $2}' ); do echo $f ; apt remove --purge $f -y
→; done
$ /usr/sbin/ofed_uninstall.sh --force
$ sudo apt-get autoremove -y
```

2. Install the DOCA host Debian package and update APT.

**Tip**

DOCA packages can be downloaded from the NVIDIA website at

[https://developer.nvidia.com/doca-downloads?deployment\\_platform=Host-Server&deployment\\_package=DOCA-Host&target\\_os=Linux&Architecture=arm64-sbsa&Profile=doca-all&Distribution=Ubuntu](https://developer.nvidia.com/doca-downloads?deployment_platform=Host-Server&deployment_package=DOCA-Host&target_os=Linux&Architecture=arm64-sbsa&Profile=doca-all&Distribution=Ubuntu)

```
$ sudo dpkg -i doca-host_3.0.0-058000-25.04-ubuntu2404_arm64.deb
$ sudo apt-get update
```

3. Install the `doca-extra` package to enable NVIDIA kernel support.

```
sudo apt install -y doca-extra
```

4. Rebuild the kernel headers with `doca-kernel-support`.

```
rm -rf /tmp/DOCA*
sudo /opt/mellanox/doca/tools/doca-kernel-support
```

The `doca-kernel-support` command generates a new Debian package file, install this file using `dpkg -i <package_file>`.

For example:

```
$ sudo /opt/mellanox/doca/tools/doca-kernel-support
doca-kernel-support: Built single package: /tmp/DOCA.EuUfkWfV7Z/doca-kernel-
→repo-2.9.0-1.kver.5.14.0.356.e19.arm.deb
doca-kernel-support: Done
$ dpkg -i /tmp/DOCA.EuUfkWfV7Z/doca-kernel-repo-2.9.0-1.kver.5.14.0.356.e19.
→arm.deb
```

5. Set the DOCA-HOST APT package preference.

Set the DOCA-HOST APT package preference to 600, to match the CUDA repo preference.

This ensures APT installs the correct DOCA software packages from CUDA and DOCA repos.

```
cat <<'EOF' | sudo tee /etc/apt/preferences.d/doca-host-repository-pin-600
Package: *
Pin: release l=DOCA-HOST
Pin-Priority: 600
EOF
```

6. Update APT and install the `doca-all` host profile.

```
sudo apt update
sudo apt -y install doca-all
```

#### Note

`doca-kernel-support` does not support customized or unofficial kernels. Refer to [Installing Software on Host](#) for more information about the installation.

## 2.2.7. Install IMEX and the NVIDIA GPU Driver

#### Note

Contact NVIDIA for access to the Blackwell GPU driver and IMEX packages.

Installing the driver and IMEX packages with APT requires installing a local APT repository and configuring APT to prefer the local repository for those packages.

### Important

The following are example filenames. Check your specific release for the correct filenames and versions.

Install the NVIDIA GPU driver and IMEX package from the `.run` files provided by NVIDIA.

### Tip

Replace the version in the example with the version in your software release bundle.

```
sudo ./NVIDIA-Linux-aarch64-570.26.run --dkms -q -s -m=kernel-open
sudo ./nvidia-imex-aarch64-570.26.run
```

## 2.2.8. Configure the NVIDIA Software Packages

This section details the configuration options for the GPU driver and IMEX application.

### 2.2.8.1 Enable Profiling

Enable profiling for all users by applying the settings to `/etc/modprobe.d/nvprofiling.conf`.

```
echo 'options nvidia NVreg_RestrictProfilingToAdminUsers=0' | sudo tee /etc/
↪modprobe.d/nvprofiling.conf
```

### 2.2.8.2 Enable the IMEX Control Channel

Configure IMEX to automatically enable the IMEX control channel by configuring `/etc/modprobe.d/nvidia.conf`.

```
$ cat /etc/modprobe.d/nvidia.conf

options nvidia NVreg_CreateImexChannel=1
```

### Note

Multitenant MNNVL systems require more complex IMEX configuration beyond the scope of this document.

Refer to the [IMEX documentation](#) for more information.

After changing the `nvidia.conf` file, rebuild the `initramfs`.

On Ubuntu:

```
sudo update-initramfs -u -k all
```

On RHEL:

```
dracut --regenerate-all -f
```

### 2.2.8.3 Configure IMEX Peers

Create the file `/etc/nvidia-imex/nodes_config.cfg` with the management Ethernet interface IP address of all nodes in the cluster.

Here is an example `/etc/nvidia-imex/nodes_config.cfg` file:

#### Note

Place one IP address per line, with no other information.

```
10.114.228.6
10.114.228.7
10.114.228.8
10.114.228.9
10.114.228.10
10.114.228.11
10.114.228.12
10.114.228.13
10.114.228.14
```

#### Note

IMEX can use any compute tray IP address, as long as there is connectivity between all IP addresses in the configuration.

### 2.2.8.4 Enable the NVIDIA Persistence Daemon

1. Configure the NVIDIA persistence daemon by editing the `/etc/systemd/system/nvidia-persistenced.service` file.

```
$ cat /etc/systemd/system/nvidia-persistenced.service
[Unit]
Description=NVIDIA Persistence Daemon
Wants=syslog.target

[Service]
Type=forking
PIDFile=/var/run/nvidia-persistenced/nvidia-persistenced.pid
Restart=always
ExecStart=/usr/bin/nvidia-persistenced --verbose
ExecStopPost=/bin/rm -rf /var/run/nvidia-persistenced

[Install]
WantedBy=multi-user.target
```

2. Enable and start the daemon by running the following command:

```
$ sudo systemctl enable nvidia-persistenced.service
```

### 2.2.8.5 Enable the IMEX Daemon

Enable the IMEX service to start at boot:

```
sudo systemctl enable nvidia-imex.service
```

### 2.2.8.6 Rebuild the initramfs

After changing the IMEX configuration rebuild the initramfs.

```
sudo update-initramfs -u -k all
```

### 2.2.8.7 Reboot the Compute Tray

Reboot the compute tray to complete the driver and application configurations.

```
sudo reboot
```

## 2.3. Switch Tray Software

Each NVLink switch tray runs a local instance of the NVOS software. The NVLink switches ship from the factory with NVOS preinstalled.

NVOS contains all the required software and firmware components.

#### Tip

Refer to your manufacturer for the latest version of NVOS and upgrade instructions.

To show the firmware information for the platform firmware components, run the following command.

```
$ nv show platform firmware
```

Here is an example output:

The actual firmware version on your system might be different.

```
admin@gb-nvl-042-switch01:~$ nv show platform firmware
Name          Actual FW          Part Number        FW Source
-----
ASIC          35.2014.1506      920-9K36F-00MV-JS0_IPN_Ax  default
BIOS          0ACTV_00.00.018d  N/A                N/A
BMC           88.0002.0588      692-13809-1404-000  N/A
CPLD1         CPLD000370_REV0109  0x0172             N/A
CPLD2         CPLD000377_REV0202  0x0179             N/A
CPLD3         CPLD000373_REV0201  0x0175             N/A
CPLD4         CPLD000390_REV0102  0x0186             N/A
EROT          01.03.0216.0000_n04  N/A                N/A
FPGA          0.19              N/A                N/A
SSD           CE00A400          Virtium VTPM24CEXI080-BM110006  N/A
transceiver  N/A               N/A                N/A
```

Figure 2.1: nv show platform firmware output

## 2.3.1. Enable the NVLink Cluster

The NVIDIA Management Controller (NMX-C) service runs on a single NVLink switch. NMX-C manages the network configuration for all NVLink switches.

1. Enable the NMX-C service on one NVLink switch.

```
nv set cluster state enabled
nv config apply
nv config save
```

2. On a compute or management node download and install [GRPCurl](#)
3. Using `grpcurl` create a gRPC connection, replacing `127.0.0.1` with the NVLink switch management IP address.

```
grpcurl -plaintext -d \
"{ \"gatewayId\": \"nvidia-bringup\", \
 \"major_version\": \"PROTO_MSG_MAJOR_VERSION\", \
 \"minor_version\": \"PROTO_MSG_MINOR_VERSION\" }" \
127.0.0.1:9371 nmx_c.NMX_Controller.Hello
```

4. After creating the gRPC connection, configure the FM topology, again replacing `127.0.0.1` with the NVLink switch management IP address.

```
grpcurl -plaintext -d \
'{ \"gatewayId\": \"nvidia-bringup\", \
 \"staticConfig\": { \"configKeyVals\": \
 { \"configKeyVal\": [ { \"configFileName\": \
 \"fm_config\", \"key\": \"MNNVL_TOPOLOGY\", \
 \"value\": \"gb200_nv172r1_c2g4_topology\"}]} }' \
127.0.0.1:9371 nmx_c.NMX_Controller.SetStaticConfig
```

### Note

If the gRPC option isn't possible, use the NVOS CLI to generate an FM configuration file.

```
nv action generate sdn config app nmx-controller type fm_config
```

Download the generated file and edit the `MNNVL_TOPOLOGY` to `MN-NVL_TOPOLOGY=gb200_nv172r1_c2g4_topology`

Upload the configuration file back to the switch and install it with the command

```
nv action install sdn config app nmx-controller type fm_config files <FILENAME>
```

5. After configuring FM, enable the cluster and save the configuration.

```
nv set cluster state enabled
nv config apply
nv config save
```

6. Start the NMX-C application.

```
nv action start cluster app nmx-controlle
```

7. Verify the cluster and NMX-C application are running.

```
nv show cluster apps running
```

---

# Chapter 3. Verifying

This section describes the tools to verify system health.

## 3.1. Compute Tray Verification

This section describes how to verify the health of each compute tray.

### 3.1.1. IMEX

Verify that the system created the default IMEX Channel 0 on each compute node.

IMEX requires channel 0 for multinode workloads.

```
$ ll /dev/nvidia-caps-imex-channels/
total 0
drwxr-xr-x  2 root root    80 Feb 13 19:46 ./
drwxr-xr-x 21 root root  4640 Feb 14 11:43 ../
crw-rw-rw-  1 root root  234, 0 Feb 13 19:46 channel0
```

Read the [IMEX Channels documentation](#) for more information.

### 3.1.2. Linux Kernel

Ensure the Linux kernel is at least [6.5.0-1024-nvidia-64k](#).

Use `uname -r` to view the current kernel version.

#### Tip

If you run your own kernel, ensure that the kernel version complies with the Grace patches published in the [NVIDIA Grace Platform Support Software Patches and Configurations guide](#).

### 3.1.3. GPU Driver

Verify that the system loaded the GPU driver correctly and all GPUs appear on each compute node.

Use the command `nvidia-smi` to view the status of the GPUs on a compute tray.

```

NVIDIA-SMI 570.14              Driver Version: 570.14          CUDA Version: 12.8
-----
GPU   Name                               Persistence-M  Bus-Id          Disp.A    Volatile Uncorr. ECC
Fan  Temp  Perf                               Pwr:Usage/Cap  Memory-Usage  GPU-Util  Compute M.
                                           MIG M.
-----
  0   NVIDIA Graphics Device             On            00000008:01:00.0 Off          0%          Default
     N/A  31C  P0                               237W / 1200W  1MiB / 192527MiB                               Disabled
-----
  1   NVIDIA Graphics Device             On            00000009:01:00.0 Off          0%          Default
     N/A  31C  P0                               215W / 1200W  3MiB / 192527MiB                               Disabled
-----
  2   NVIDIA Graphics Device             On            00000018:01:00.0 Off          0%          Default
     N/A  32C  P0                               215W / 1200W  1MiB / 192527MiB                               Disabled
-----
  3   NVIDIA Graphics Device             On            00000019:01:00.0 Off          0%          Default
     N/A  31C  P0                               232W / 1200W  1MiB / 192527MiB                               Disabled
-----

Processes:
GPU   GI   CI          PID  Type  Process name          GPU Memory
  ID  ID  ID                                     Usage
-----
No running processes found

```

Figure 3.1: Example nvidia-smi output with four GPUs.

Verify that each compute tray displays the platform information correctly.

View the GPU platform details with the command:

```
nvidia-smi -q | grep Platform -A 6
```

Figure 3.2 shows four GPUs connected to the NVLink network.

Verify the GPUs successfully registered to the NVLink fabric with the command

```
nvidia-smi -q | grep 'Fabric' -A 4
```

Figure 3.3 shows an example output with the expected state of Completed and status of Success.

### 3.1.4. NVLink Status

To ensure that the NVLink interfaces are operational and have the correct bandwidth, run the following command on each compute node:

```
nvidia-smi nvlink --status
```

Figure 3.4 is sample output 18 links active to the nine switch trays. Each link shows the expected 50 GB/s bandwidth.

If any of the links are <inactive>, the GPU driver can't interact with other GPUs.

#### Important

All NVLinks must be up and reporting 50 GB/s bandwidth.

Links can be down because a switch tray is offline, problems with the physical link connections to that switch tray, high bit error rates on that link or the GPU driver isn't running.

```
nvidia@localhost:~$ nvidia-smi -q | grep -i platf -A 6
Platform Info
  Chassis Serial Number      : 1784024160100
  Slot Number                : 1
  Tray Index                 : 0
  Host ID                    : 1
  Peer Type                  : Switch Connected
  Module Id                  : 2
--
Platform Info
  Chassis Serial Number      : 1784024160100
  Slot Number                : 1
  Tray Index                 : 0
  Host ID                    : 1
  Peer Type                  : Switch Connected
  Module Id                  : 1
--
Platform Info
  Chassis Serial Number      : 1784024160100
  Slot Number                : 1
  Tray Index                 : 0
  Host ID                    : 1
  Peer Type                  : Switch Connected
  Module Id                  : 4
--
Platform Info
  Chassis Serial Number      : 1784024160100
  Slot Number                : 1
  Tray Index                 : 0
  Host ID                    : 1
  Peer Type                  : Switch Connected
  Module Id                  : 3
```

Figure 3.2: Example nvidia-smi -q output.

```
nvidia@gb-nvl-027-compute2:~$ nvidia-smi -q | grep 'Fabric' -A 4
Fabric
  State      : Completed
  Status     : Success
  CliqueId   : 32766
  ClusterUUID : 53a24e94-52fa-f566-aa13-60ef67933749
--
Fabric
  State      : Completed
  Status     : Success
  CliqueId   : 32766
  ClusterUUID : 53a24e94-52fa-f566-aa13-60ef67933749
--
Fabric
  State      : Completed
  Status     : Success
  CliqueId   : 32766
  ClusterUUID : 53a24e94-52fa-f566-aa13-60ef67933749
--
Fabric
  State      : Completed
  Status     : Success
  CliqueId   : 32766
  ClusterUUID : 53a24e94-52fa-f566-aa13-60ef67933749
```

Figure 3.3: NVIDIA Fabric shows “Completed” and “Success”

```
root@gb-nvl-028-compute3:~# nvidia-smi nvlink -s
GPU 0: NVIDIA Graphics Device (UUID: GPU-96b43269-2745-2058-6be7-0fdae0ab8572)
  Link 0: 50 GB/s
  Link 1: 50 GB/s
  Link 2: 50 GB/s
  Link 3: 50 GB/s
  Link 4: 50 GB/s
  Link 5: 50 GB/s
  Link 6: 50 GB/s
  Link 7: 50 GB/s
  Link 8: 50 GB/s
  Link 9: 50 GB/s
  Link 10: 50 GB/s
  Link 11: 50 GB/s
  Link 12: 50 GB/s
  Link 13: 50 GB/s
  Link 14: 50 GB/s
  Link 15: 50 GB/s
  Link 16: 50 GB/s
  Link 17: 50 GB/s
GPU 1: NVIDIA Graphics Device (UUID: GPU-1a86c8d0-f35a-7e8b-7d97-7131501e4726)
  Link 0: 50 GB/s
  Link 1: 50 GB/s
  Link 2: 50 GB/s
  Link 3: 50 GB/s
  Link 4: 50 GB/s
  Link 5: 50 GB/s
  Link 6: 50 GB/s
  Link 7: 50 GB/s
  Link 8: 50 GB/s
  Link 9: 50 GB/s
  Link 10: 50 GB/s
  Link 11: 50 GB/s
  Link 12: 50 GB/s
  Link 13: 50 GB/s
  Link 14: 50 GB/s
  Link 15: 50 GB/s
  Link 16: 50 GB/s
  Link 17: 50 GB/s
GPU 2: NVIDIA Graphics Device (UUID: GPU-052c6ee7-e001-2fb1-523-a336e236c546)
  Link 0: 50 GB/s
  Link 1: 50 GB/s
  Link 2: 50 GB/s
  Link 3: 50 GB/s
  Link 4: 50 GB/s
  Link 5: 50 GB/s
  Link 6: 50 GB/s
  Link 7: 50 GB/s
  Link 8: 50 GB/s
  Link 9: 50 GB/s
  Link 10: 50 GB/s
  Link 11: 50 GB/s
  Link 12: 50 GB/s
  Link 13: 50 GB/s
  Link 14: 50 GB/s
  Link 15: 50 GB/s
  Link 16: 50 GB/s
  Link 17: 50 GB/s
GPU 3: NVIDIA Graphics Device (UUID: GPU-72eef6b7-3791-2a6d-f610-7fecac0aece)
  Link 0: 50 GB/s
  Link 1: 50 GB/s
  Link 2: 50 GB/s
  Link 3: 50 GB/s
  Link 4: 50 GB/s
  Link 5: 50 GB/s
  Link 6: 50 GB/s
  Link 7: 50 GB/s
  Link 8: 50 GB/s
  Link 9: 50 GB/s
  Link 10: 50 GB/s
  Link 11: 50 GB/s
  Link 12: 50 GB/s
  Link 13: 50 GB/s
  Link 14: 50 GB/s
  Link 15: 50 GB/s
  Link 16: 50 GB/s
  Link 17: 50 GB/s
root@gb-nvl-028-compute3:~#
```

Figure 3.4: NVLinks between each GPU and the NVLink switches each at 50 GB/s

### 3.1.5. Topology Status

Verify the local GPU topology with the command:

```
nvidia-smi topo -p2p n
```

Verify that all GPUs on the tray show OK.

```
nvidia@gb-nvl-030-compute3:/home/nvidia$ nvidia-smi topo -p2p n
  GPU0   GPU1   GPU2   GPU3
GPU0  X     OK     OK     OK
GPU1  OK    X     OK     OK
GPU2  OK    OK    X     OK
GPU3  OK    OK    OK    X

Legend:
  X     = Self
  OK    = Status Ok
  CNS   = Chipset not supported
  GNS   = GPU not supported
  TNS   = Topology not supported
  NS    = Not supported
  U     = Unknown
```

Figure 3.5: All GPU to GPU links are OK

If a GPU isn't OK, reset the specific GPU:

```
nvidia-smi -r -i <GPU_INDEX>
```

or reset all GPUs with:

```
nvidia-smi -r
```

### 3.1.6. NVIDIA Persistenced Service Status

Verify each compute node has the nvidia-persistenced service running.

On every compute node run:

```
systemctl status nvidia-persistenced
```

```
root@localhost:/home/nvidia# systemctl status nvidia-persistenced
● nvidia-persistenced.service - NVIDIA Persistence Daemon
   Loaded: loaded (/etc/systemd/system/nvidia-persistenced.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2024-05-11 13:28:42 UTC; 1 week 2 days ago
     Process: 2638 ExecStart=/usr/bin/nvidia-persistenced --verbose --persistence-mode (code=exited, status=0/SUCCESS)
    Main PID: 2639 (nvidia-persiste)
      Tasks: 1 (Limit: 294219)
     Memory: 38.0M
        CPU: 5.025s
    CGroup: /system.slice/nvidia-persistenced.service
            └─2639 /usr/bin/nvidia-persistenced --verbose --persistence-mode

May 11 13:28:37 localhost nvidia-persistenced[2639]: Started (2639)
May 11 13:28:37 localhost nvidia-persistenced[2639]: NUMA: Enabling NUMA memory Auto-Online due to GPU requirement
May 11 13:28:37 localhost nvidia-persistenced[2639]: device 0009:01:00.0 - registered
May 11 13:28:40 localhost nvidia-persistenced[2639]: device 0009:01:00.0 - persistence mode enabled.
May 11 13:28:40 localhost nvidia-persistenced[2639]: device 0009:01:00.0 - NUMA memory onlined.
May 11 13:28:40 localhost nvidia-persistenced[2639]: device 0019:01:00.0 - registered
May 11 13:28:42 localhost nvidia-persistenced[2639]: device 0019:01:00.0 - persistence mode enabled.
May 11 13:28:42 localhost nvidia-persistenced[2639]: device 0019:01:00.0 - NUMA memory onlined.
May 11 13:28:42 localhost nvidia-persistenced[2639]: Local RPC services initialized
May 11 13:28:42 localhost systemd[1]: Started NVIDIA Persistence Daemon.
root@localhost:/home/nvidia#
```

Figure 3.6: NVIDIA Persistenced service status

### 3.1.7. NVIDIA IMEX Service Status

Verify each compute node has the `nvidia-imex` service running.

On every compute node run:

```
systemctl status nvidia-imex
```

```
root@localhost:/home/nvidia# systemctl status nvidia-imex
● nvidia-imex.service - NVIDIA IMEX service
   Loaded: loaded (/lib/systemd/system/nvidia-imex.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2024-05-11 13:29:49 UTC; 1 week 2 days ago
     Process: 2635 ExecStart=/usr/bin/nvidia-imex -c /etc/nvidia-imex/config.cfg (code=exited, status=0/SUCCESS)
    Main PID: 2641 (nvidia-imex)
      Tasks: 23 (limit: 294219)
     Memory: 147.4M
        CPU: 15min 21.790s
    CGroup: /system.slice/nvidia-imex.service
            └─2641 /usr/bin/nvidia-imex -c /etc/nvidia-imex/config.cfg

May 11 13:28:37 localhost systemd[1]: Starting NVIDIA IMEX service...
May 11 13:29:49 localhost systemd[1]: Started NVIDIA IMEX service.
root@localhost:/home/nvidia#
```

Figure 3.7: NVIDIA IMEX service status

Also confirm every compute node runs the same version of the IMEX daemon.

```
$ /usr/bin/nvidia-imex --version
IMEX version is: 565.46
```

### 3.1.8. IMEX Domain Status

Use the IMEX control tool to get the full status of the IMEX domain.

```
nvidia-imex-ctl -N
```

Figure 3.8 is an example of a working IMEX domain.

```
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 triggered.
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

10/25/2024 18:22:11.657
Nodes:
Node #0 - 10.115.20.6 - READY - Version: 570.26
Node #1 - 10.115.20.7 - READY - Version: 570.26
Node #2 - 10.115.20.8 - READY - Version: 570.26
Node #3 - 10.115.20.9 - READY - Version: 570.26
Node #4 - 10.115.20.10 - READY - Version: 570.26
Node #5 - 10.115.20.11 - READY - Version: 570.26
Node #6 - 10.115.20.12 - READY - Version: 570.26
Node #7 - 10.115.20.13 - READY - Version: 570.26
Node #8 - 10.115.20.14 - READY - Version: 570.26

Nodes From\To 0 1 2 3 4 5 6 7 8
0 C C C C C C C C C
1 C C C C C C C C C
2 C C C C C C C C C
3 C C C C C C C C C
4 C C C C C C C C C
5 C C C C C C C C C
6 C C C C C C C C C
7 C C C C C C C C C
8 C C C C C C C C C
Domain State: UP
```

Figure 3.8: NVIDIA IMEX ctl command

**Tip**

Refer to the [NVIDIA IMEX User's Guide](#) for more information.

## 3.2. Switch Tray Verification

This section describes health checks for the NVLink switch trays.

### 3.2.1. System Health

Verify the health of each switch tray with the command `nv show system health`.

### 3.2.2. Control Plane Check

Verify the NMX-C and NMX-T applications are running on the NVLink switch.

**Important**

NMX-C and NMX-T can only run on a single switch in an NVLink domain.

```
$ nv show cluster apps
```

Name	ID	Version
nmx-controller	nmx-c-nvos	0.8.0_2024-11-27_11-25
nmx-telemetry	nmx-telemetry	0.8.3

```
$ nv show cluster apps running
```

Name	Status	Reason	Additional Information
nmx-controller	ok		CONTROL_PLANE_STATE_CONFIGURED
nmx-telemetry	ok		

#### 3.2.2.1 NMX-T Health Check

Run the NMX-T health check to verify the NMX-T application.

```
curl http://0.0.0.0:9350/healthcheck
```

The normal response is

```
{"status":0,"message":"OK"}
```

Display the NMX-T statistics with the command:

```
curl http://0.0.0.0:9352/management/statistics
```

Display the NMX-T management status with the command:

```
curl http://0.0.0.0:9352/management/check_status
```

---

# Chapter 4. Rack Reboot Sequence

This section describes how to “cold reboot” or “warm reboot” an MNNVL system.

## Note

These terms are defined as follows:

- ▶ A cold reboot powers the rack off and then back on.
- ▶ A warm reboot never loses power.

## Important

Always use a cold reboot unless directed by the release notes to use a warm reboot.

## 4.1. Cold Reboot Sequence

To cold reboot a rack:

## Note

If the rack will run L11 diagnostics, don't power on the compute nodes. Stop at step 8.

1. Run the `halt -p` command on all compute nodes.
2. Run the `nv action reboot system halt` on all NVLink switches.
3. AC power off all the power shelves in the rack.
4. Wait at least three minutes for the PSU capacitors to discharge.
5. Power on all power shelves in the rack.
6. Wait at least two minutes for the compute tray and switch tray BMCs to power on.
7. Wait at least two more minutes for the switch trays to boot NVOS.
8. Verify the health of each switch tray with the *switch tray verification* steps.
9. Access the BMC of each compute tray and power on the node.

**Tip**

Use the RedFish API to power on the compute trays.

```
curl -si -u <BMC_USERNAME>:<BMC_PASSWORD> -k -X POST --header 'Content-
↳Type: application/json' --header 'Accept: application/json' -d '{
↳"ResetType": "On"}' https://<BMC_IP>/redfish/v1/Systems/System_0/Actions/
↳ComputerSystem.Reset
```

10. Run the *compute tray verification steps*.

## 4.2. Warm Reboot Sequence

**Important**

Only use a warm reboot when directed by the release notes.

For a warm reboot:

1. Run the `halt -p` command on all compute nodes.
2. Stop the NMX-C service on the switch node running NMX-C.

```
nv action stop cluster app nmx-controller
```

3. Verify the cluster is enabled from the switch node with the command `nv show cluster`.

```
nv show cluster
      operational  applied
-----
state      enabled      enabled
```

4. Start the NMX-C service on the switch node.

```
nv action start cluster app nmx-controller
```

5. Verify NMX-C is ok with the command `nv show cluster apps running`.

```
$ nv show cluster apps running
Name           Status  Reason  Additional Information
-----
nmx-controller  ok           CONTROL_PLANE_STATE_CONFIGURED
nmx-telemetry  ok
```

6. Power up all the compute trays through BMC.

**Tip**

Use the RedFish API to power on the compute trays.

```
curl -si -u <BMC_USERNAME>:<BMC_PASSWORD> -k -X POST --header 'Content-
↳Type: application/json' --header 'Accept: application/json' -d '{
```

```
↪ "ResetType": "On"}' https://<BMC_IP/redfish/v1/Systems/System_0/Actions/  
↪ ComputerSystem.Reset
```

7. Run all the verification checks for the compute tray (refer to *Post Reboot Verification*).

## 4.3. Post Reboot Verification

Complete the following verification checks when you reset a GPU, reboot the compute tray, or reboot the MNNVL rack.

1. Verify that the nvidia-persistenced service is running.

```
systemctl status nvidia-persistenced
```

2. Verify that the nvidia-imex service is running.

```
systemctl status nvidia-imex
```

3. Verify that all the NVLink connections are active.

```
nvidia-smi nvlink --status
```

4. Verify fabric state to make sure all GPUs have a Completed state and a Success status.

```
nvidia-smi -q | grep 'Fabric' -A 4`
```

5. Verify peer-to-peer topology to ensure all GPUs show OK.

```
nvidia-smi topo -p2p n
```

---

# Chapter 5. Multinode Testing

For multinode testing NVIDIA recommends using `nvbandwidth` application. The `nvbandwidth` application, when used with the Message Passing Interface (MPI), connects to each GPU instance and tracks test data copies across a full mesh of cluster GPUs.

## 5.1. Installing Required Packages

The `nvbandwidth` test requires `git`, `libboost-program-options-dev`, `sshpass`, `openmpi-bin`, `make`, and `libopenmpi-dev` and `net-tools`. Install them using `Apt`.

```
apt install git libboost-program-options-dev sshpass openmpi-bin make  
↳libopenmpi-dev net-tools
```

### Important

Install these packages on every MNNVL compute node.

## 5.2. Configure SSH

Each compute node must generate an SSH key, copy it to all other compute nodes and change the SSH Server configuration before running `nvbandwidth`.

### 5.2.1. Creating and Copy SSH Keys

The `nvbandwidth` application relies on `mpirun` to connect to each node through passwordless SSH to run tests and provide client-server connectivity.

Create SSH keys on each compute node with the `ssh-keygen` command. A passphrase isn't required.

```
$ ssh-keygen  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/nvidia/.ssh/id_rsa):  
/home/nvidia/.ssh/id_rsa already exists.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/nvidia/.ssh/id_rsa  
Your public key has been saved in /home/nvidia/.ssh/id_rsa.pub
```

Copy the `/home/nvidia/.ssh/id_rsa.pub` to each compute node using `ssh-copy-id`.

```
$ ssh-copy-id nvidia@10.28.238.236
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/nvidia/.
→ssh/id_rsa.pub"
```

```
Number of key(s) added: 1
```

**Tip**

Generate an SSH key on every compute node, then copy the `id_rsa.pub` key to every other compute node.

Every compute node copies their keys to every other compute node.

Verify that you can log in with SSH without a password from each compute node to all other compute nodes.

## 5.2.2. Configuring the SSH Server

To accommodate the number of SSH connections that `mpirun` requires, increase the number of allowed SSH connections.

On every compute node create the file `/etc/ssh/sshd_config.d/mpi.conf` with the following contents:

```
MaxSessions 550
MaxStartups 10:30:100
```

On every compute node, restart the SSH service and apply the new configuration.

```
systemctl restart sshd.service
```

## 5.3. Build `nvbandwidth`

Clone the `nvbandwidth` repository to a compute node.

```
git clone https://github.com/NVIDIA/nvbandwidth/tree/v0.7
```

Enter the cloned `nvbandwidth` directory and run the build script and copy the `nvbandwidth` binary to the `/usr/local/bin` directory.

```
cd nvbandwidth
sudo ./debian_install.sh
sudo cp nvbandwidth /usr/local/bin/
```

Copy the newly created `nvbandwidth` binary to every compute node.

**Tip**

Use `scp` and the IMEX configuration file to automate the file copy process.

```
while IFS= read -r dest; do
  scp nvbandwidth "$dest:/usr/local/bin/"
done </etc/nvidia-imex/nodes_config.cfg
```

## 5.4. Run nvbandwidth

The `mpirun` application requires an interface name to connect to the other compute trays and run the test application.

From a compute tray, get the name of the default interface with the `ip route show` command.

```
$ ip route show default
default via 10.114.160.1 dev enP5p9s0 proto dhcp src 10.114.160.219 metric 100
```

### Tip

This example assumes all compute trays can communicate over the default interface. Use the appropriate interface for your environment.

Use `mpirun` to run the `nvbandwidth` application between all GPUs on the compute nodes, providing the Ethernet interface name and full path to the `nvbandwidth` binary. The `-np` parameter specifies the number of GPUs.

### Note

Running `nvbandwidth` requires all compute nodes to have IMEX installed and running.

```
$ mpirun --allow-run-as-root --map-by ppr:4:node --bind-to core -np 72 --
  ↳report-bindings -mca btl_tcp_if_include enP5p9s0 --hostfile /etc/nvidia-
  ↳imex/nodes_config.cfg /usr/local/bin/nvbandwidth -t multinode_device_to_
  ↳device_memcpy_read_ce
```

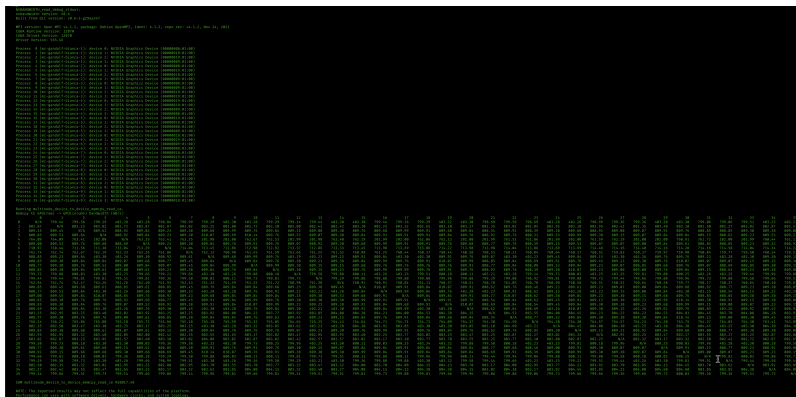


Figure 5.1: Example NVBandwidth Read Test

```
$ mpirun --allow-run-as-root --map-by ppr:4:node --bind-to core -np 72 --  
→report-bindings -mca btl_tcp_if_include enP5p9s0 --hostfile /etc/nvidia-  
→imex/nodes_config.cfg /usr/local/bin/nvbandwidth -t multinode_device_to_  
→device_memcpy_write_ce
```

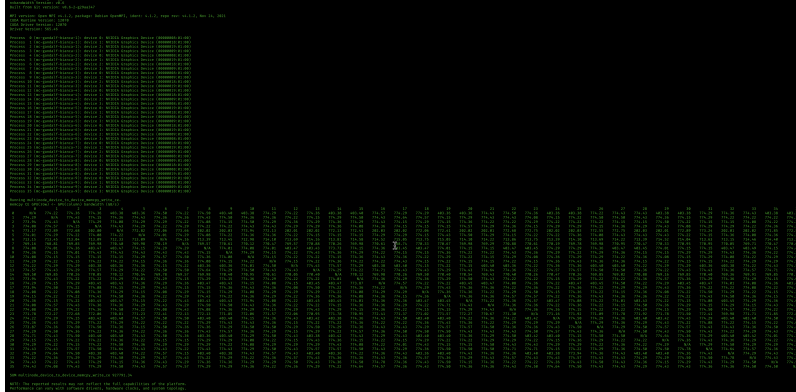


Figure 5.2: Example NVBandwidth Write Test

---

# Chapter 6. Troubleshooting

This section describes how to troubleshoot the MNNVL rack.

## 6.1. Troubleshooting the Compute Tray

This section provides information about troubleshooting the compute tray.

### 6.1.1. Driver Not Loading

If you run `nvidia-smi` and it shows `No devices were found`, the GPU driver isn't loading correctly. Collect the `dmesg` output and check for `XID` errors in the log.

## 6.2. Troubleshooting the Switch Tray

This section provides information about troubleshooting the switch tray.

#### Note

The [NVOS User Manual](#) contains additional troubleshooting information.

### 6.2.1. Switch Health

If the `nv show system health` status is `FATAL`, reboot the switch tray.

```
nv action reboot system
```

If the error persists, contact NVIDIA.

### 6.2.2. BMC Firmware Issues

If the BMC displays an error in `nv show platform firmware`, AC power-cycle the switch tray using the following command.

```
nv action power-cycle system
```

```
[System_Fatal_State]admin@nvos:~$ nv show system health
operational applied
-----
status      FATAL
status-led  amber

Health issues
-----
Component   Status information
-----
ASIC-HEALTH Switch ASIC in fatal state.
```

Figure 6.1: Fatal system health

**Note**

The reboot command only reboots the COMe and not the switch BMC, whereas the power-cycle command performs a full AC power-cycle without gracefully shutting down the system.

## 6.3. Troubleshooting the NVLink Fabric

This section provides information about troubleshooting the NVLink fabric.

### 6.3.1. NMX-Controller on Multiple Trays

The NVLink fabric requires the GPUs on each compute node to communicate with all the NVLink switches. Only one NVLink switch runs the NMX-C services.

Running NMX-C on more than one NVLink switch isn't supported.

### 6.3.2. Viewing Subnet Manager Logs

You can find the Subnet Manager (SM) logs in the switch tray that runs NMX-C:

```
/var/log/nmx/nmx-c/nv1sm.log
```

### 6.3.3. Viewing Global Fabric Manager Logs

You can find the Global Fabric Manager (GFM) logs in the switch tray that runs NMX-C:

```
/var/log/nmx/nmx-c/fabricmanager.log
```

### 6.3.4. Viewing IMEX Logs

IMEX provides GPU-to-GPU memory management, and the `mpi_memcpy` application requires IMEX.

To view the IMEX logs on each compute node in the `/var/log/nvidia-imex.log` file, run the following command.

```

cat /var/log/nvidia-imex.log
[Aug 05 2024 21:27:19] [INFO] [tid 2248] Node configuration validation from
↳NODE 8 successfully matched this node's configuration.
[Aug 05 2024 21:27:19] [INFO] [tid 2248] Node configuration validation from
↳NODE 1 successfully matched this node's configuration.
[Aug 05 2024 21:27:20] [INFO] [tid 2248] Node configuration validation from
↳NODE 15 successfully matched this node's configuration.
[Aug 05 2024 21:27:24] [INFO] [tid 2248] Node configuration validation from
↳NODE 13 successfully matched this node's configuration.
[Aug 05 2024 21:27:24] [WARNING] [tid 2242] Waiting for validation response
↳for 45 seconds from the following nodes:
[Aug 05 2024 21:27:24] [WARNING] [tid 2242]          0 - 10.28.238.87
[Aug 05 2024 21:27:24] [WARNING] [tid 2242]          7 - 10.28.238.161
[Aug 05 2024 21:27:24] [WARNING] [tid 2242]         10 - 10.28.238.192
[Aug 05 2024 21:27:24] [WARNING] [tid 2242]         12 - 10.28.238.211
[Aug 05 2024 21:27:24] [WARNING] [tid 2242]         14 - 10.28.238.239
[Aug 05 2024 21:27:25] [INFO] [tid 2248] Node configuration validation from
↳NODE 12 successfully matched this node's configuration.
[Aug 05 2024 21:27:29] [WARNING] [tid 2242] Waiting for validation response
↳for 50 seconds from the following nodes:
[Aug 05 2024 21:27:29] [WARNING] [tid 2242]          0 - 10.28.238.87
[Aug 05 2024 21:27:29] [WARNING] [tid 2242]          7 - 10.28.238.161
[Aug 05 2024 21:27:29] [WARNING] [tid 2242]         10 - 10.28.238.192
[Aug 05 2024 21:27:29] [WARNING] [tid 2242]         14 - 10.28.238.239
[Aug 05 2024 21:27:29] [INFO] [tid 2248] Node configuration validation from
↳NODE 14 successfully matched this node's configuration.
[Aug 05 2024 21:27:30] [INFO] [tid 2248] Node configuration validation from
↳NODE 7 successfully matched this node's configuration.
[Aug 05 2024 21:27:34] [WARNING] [tid 2242] Waiting for validation response
↳for 55 seconds from the following nodes:
[Aug 05 2024 21:27:34] [WARNING] [tid 2242]          0 - 10.28.238.87
[Aug 05 2024 21:27:34] [WARNING] [tid 2242]         10 - 10.28.238.192
[Aug 05 2024 21:27:34] [INFO] [tid 2248] Node configuration validation from
↳NODE 0 successfully matched this node's configuration.
[Aug 05 2024 21:27:37] [INFO] [tid 2248] Node configuration validation from
↳NODE 10 successfully matched this node's configuration.
[Aug 05 2024 21:27:37] [INFO] [tid 2242] Node map validation complete.
[Aug 05 2024 21:27:37] [INFO] [tid 2242] GPU event successfully subscribed

```

The message GPU event successfully subscribed indicates that IMEX has successfully registered with all other cluster GPUs.

### 6.3.5. IMEX TRANSIENT\_FAILURE State

IMEX operates as a distributed service that runs on each compute node. If a single compute tray isn't running the IMEX, the IMEX service on other compute trays never reaches the READY state.

Look at the IMEX log, /var/log/nvidia-imex.log for messages like State = TRANSIENT\_FAILURE (NOT OK) or Not all clients connected. Waiting and retrying..

Restart the nvidia-imex service on all compute nodes to try and rebuild the IMEX cluster.

## Copyright

©2025, NVIDIA Corporation