NVIDIA Fabric Manager

Release 2.3

NVIDIA Corporation

May 06, 2025

Contents

Chapter 1. Overview

As deep learning neural networks become more sophisticated, their size and complexity continues to expand. The result is an exponential demand in the computing capacity that is required to train these networks during a reasonable period. To meet this challenge, applications have turned into multi-GPU implementations.

NVIDIA® NVLink™, which was introduced to connect multiple GPUs, is a direct GPU-to-GPU interconnect that scales multi-GPU input/output (IO) in the server. To additionally scale the performance and connect multiple GPUs, NVIDIA introduced NVIDIA NVSwitch™, which connects multiple NVLinks to provide all-to-all GPU communication at the total NVLink speed.

This document provides guidelines for setting up Fabric Manager, different virtualization models, high-availability modes and other details for NVSwitch-based single-node HGX and DGX systems.

1.1. NVSwitch-Based Systems

Over the years, NVIDIA introduced four generations of NVSwitches and the associated NVIDIA DGX™ and NVIDIA HGX™ server systems.

NVIDIA DGX-2™ and NVIDIA HGX-2 systems consist of two identical GPU baseboards with eight NVIDIA V100 GPUs and six first generation NVSwitches on each baseboard. Each V100 GPU has one NVLink connection to each NVSwitch on the same GPU baseboard, and the two GPU baseboards are connected to build a 16-GPU system. Between the two GPU baseboards, the only NVLink connections are between NVSwitches, and each NVSwitch from a GPU baseboard is connected to one NVSwitch on the second GPU baseboard for a total of eight NVLink connections.

The DGX A100 and NVIDIA HGX A100 8-GPU systems consist of a GPU baseboard, with eight NVIDIA A100 GPUs, and six second generation NVSwitches. The GPU baseboard NVLink topology is like the first-generation version, where each A100 GPU has two NVLink connections to each NVSwitch on the same GPU baseboard. This generation supports connecting two GPU baseboards for a total of sixteen NVLink connections between the baseboards.

Third-generation NVSwitches are used in DGX H100 and NVIDIA HGX H100 8-GPU server systems. This server variant consists of one GPU baseboard with eight NVIDIA H100 GPUs and four NVSwitches. The corresponding NVLink topology is different from the previous generation because every GPU has four NVLinks that connect to two of the NVSwitches, and five NVLinks that connect to the remaining two NVSwitches. This generation does not support the ability to connect two GPU baseboard using NVLink.

The DGX B200, NVIDIA HGX B200 8-GPU, and NVIDIA HGX B100 8-GPU systems use the fourth-generation NVSwitches and the B200 and B100 GPUs. The corresponding GPU baseboard NVLink topology has two NVSwitch ASICs and eight B200/B100 GPUs with nine NVLinks from each GPU is connected to an NVSwitch. Like the DGX H100 and HGX H100 generation, this baseboard does not support connecting two GPU baseboard using NVLink.

1.2. Terminology

Table 1.1 Terminology

Abbreviations	Definitions
FM	Fabric Manager.
MMIO	Memory Mapped IO.
VM	Virtual machine.
GPU register	A location in the GPU MMIO space.
SBR	Secondary Bus Reset.
DCGM	NVIDIA Data Center GPU manager.
NVML	NVIDIA Management Library.
Service VM	A privileged VM where NVIDIA NVSwitch software stack runs.
Access NVLink	NVLink between a GPU and an NVSwitch.
Trunk NVLink	NVLink between two GPU baseboards.
SMBPBI	NVIDIA SMBus Post-Box Interface.
vGPU	NVIDIA GRID Virtual GPU.
MIG	Multi-Instance GPU.
SR-IOV	Single-Root IO Virtualization.
PF	Physical Function.
LPF	Limited Physical Function
VF	Virtual Function.
GFID	GPU Function Identification.
Partition	A collection of GPUs that are allowed to perform NVLink Peer-to-Peer Communication.
ALI	Autonomous Link Initialization.
OFED	Open Fabrics Enterprise Distribution Driver.
MOFED	Mellanox/Nvidia version of OFED Driver package.
NVLSM	NVLink Subnet Manager
NVSDM	NVLink Switch Device Manager

1.3. NVSwitch Core Software Stack

This section provides information about the NVSwitch core software stack.

1.3.1. Systems Using NVSwitches that are Earlier than the Fourth Generation NVSwitches

The core software stack for NVSwitch management consists of an NVSwitch kernel driver and a privileged process called NVIDIA Fabric Manager (FM). The kernel driver performs low-level hardware management in response to FM requests. The software stack also provides in-band and out-of-band monitoring solutions to report NVSwitch and GPU errors and status information.

Figure 1.1 shows an NVSwitch core software stack.

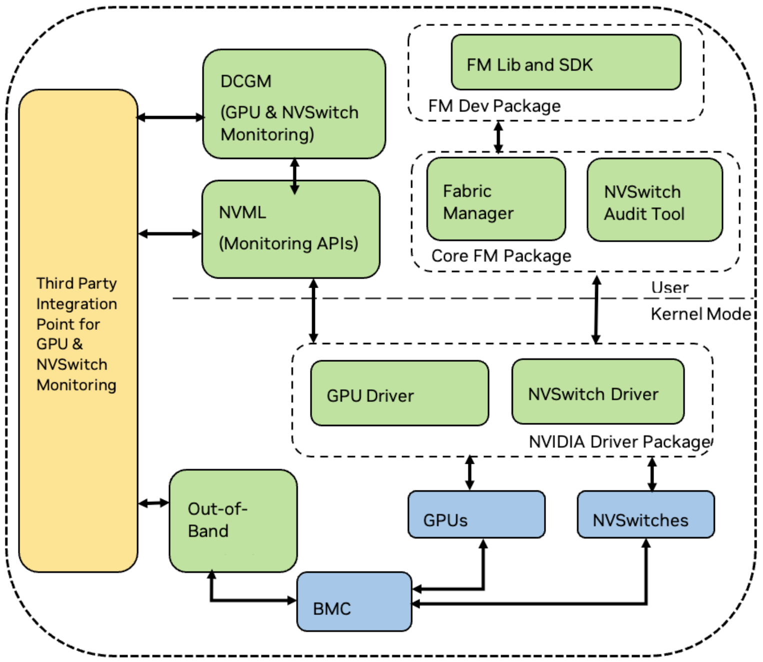

Figure 1.1: NVSwitch Core Software Stack

1.3.2. Systems Using Fourth Generation NVSwitches

With the fourth generation of NVSwitches, NVIDIA has implemented a unified architecture that spans across NVLink, InfiniBand, and Ethernet switches. This architectural coherence ensures that fourth generation NVSwitches share a common IP block with our InfiniBand (IB) switches, with the main focus on the link layer and control plane aspects. As a result of this integration, a new control plane entity called NVLink Subnet Manager (NVLSM) is introduced with FM. The SM service originates from NVIDIA IB Switches and has the necessary modifications to effectively manage NVSwitches.The NVLSM service originates from NVIDIA IB Switches and has the necessary modifications to effectively manage NVSwitches.

At a higher level, the NVLSM service is responsible for configuring NVSwitch routing tables, while FM handles GPU-side routing, NVLink configuration, and provides APIs for partition management. The interaction between FM and NVLSM is facilitated through an Inter-Process Communication (IPC) interface. This communication channel is essential to initialize and configure the fabric, which ensures seamless coordination between the FM and NVLSM.

Note

The DGX B200, NVIDIA HGX B200 8-GPU, and NVIDIA HGX B100 8-GPU systems use the fourth generation NVSwitches.

Figure 1.2 illustrates the systems that use the fourth generation NVSwitches.

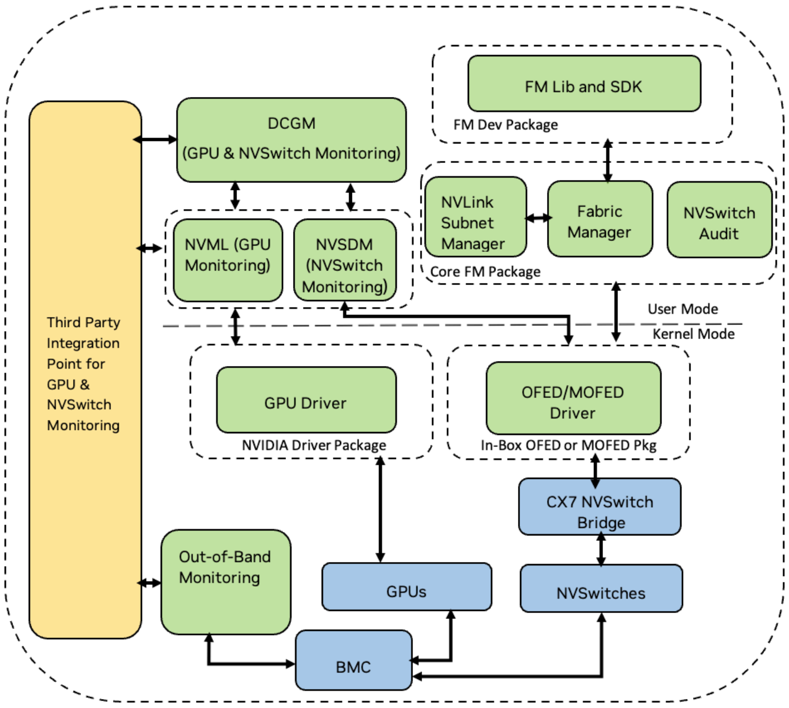

Figure 1.2: Fourth Generation NVSwitch Based Systems

1.4. What is Fabric Manager?

FM configures the NVSwitch memory fabrics to form one memory fabric among all participating GPUs and monitors the NVLinks that support the fabric. At a high level, FM completes the following tasks:

- Configures routing (earlier than the fourth generation NVSwitch) among NVSwitch ports.
- Sets up GPU routing and port map if applicable.
- Coordinates with the GPU driver to initialize GPUs.
- Monitors the fabric for NVLink and NVSwitch errors.
- On systems that are not capable of Autonomous Link Initialization (ALI)-based NVLink training (first and second generation NVSwitch-based systems), FM complets the following tasks:
- Coordinates with the NVSwitch driver to initialize and train NVSwitch-to-NVSwitch NVLink interconnects.

- Coordinates with the GPU driver to initialize and train NVSwitch-to-GPU NVLink interconnects.

This user's guide provides an overview of FM features and is intended for system administrators and NVSwitch-based server system users.

1.5. What is NVLink Subnet Manager?

NVLink Subnet manager (NVLSM) originated from the IB networking and contains additional logic to program NVSwitches and NVLinks. At a high level, the NVLSM provides the following functionality to NVSwitch-based systems:

- Discovers the NVLink network topology.
- Assigns a local identifier (LID) to all the GPU and NVSwitch NVLink ports.
- Calculates and programs switch forwarding tables.
- Programs the Partition Key (PKEY) for NVLink partitions.
- Monitors changes in the NVLink fabric.

1.6. GPU Baseboard Topologies

The following section provides information about different baseboard PCIe topologies, with a focus on GPU and NVSwitches, and how the topologies will appear on a host system.

1.6.1. The HGX-2 GPU Baseboard

Figure 1.3 shows a simplified HGX-2 GPU baseboard.

The HGX-2 baseboard contains eight V100 GPUs and six corresponding first generation NVSwitches. From a PCIe tree perspective, the eight GPUs and six NVSwitches will appear on the PCIe tree as PCIe devices on the host system.

Here is an example:

```
$ lspci | grep -i nvidia
34:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
36:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
39:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
3b:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
57:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
59:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
5c:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
5e:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 SXM3 32GB] (rev
↪a1)
```

(continues on next page)

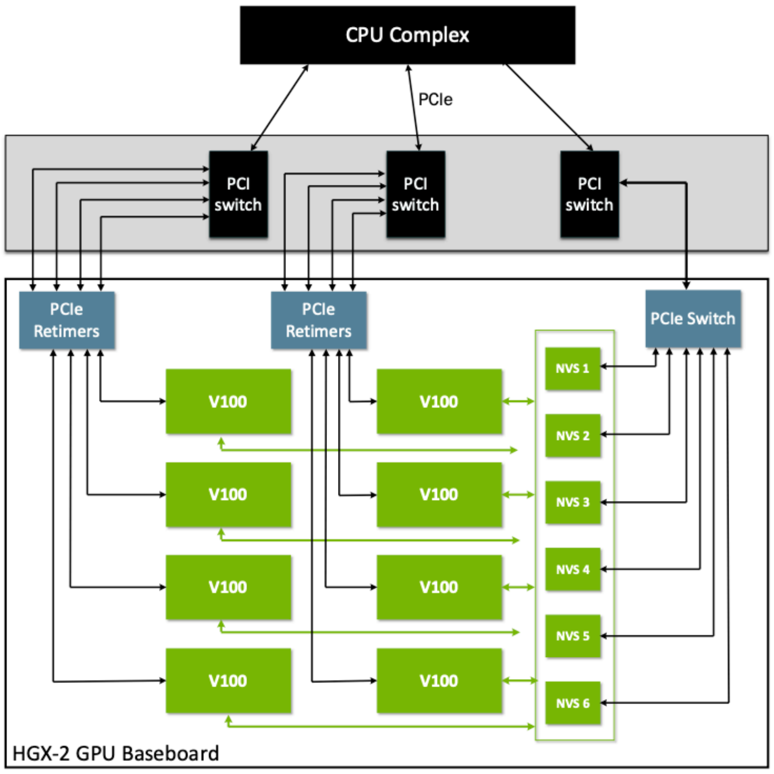

Figure 1.3: Simplified HGX-2 Baseboard

(continued from previous page)

```
61:00.0 Bridge: NVIDIA Corporation Device 1ac2 (rev a1)
62:00.0 Bridge: NVIDIA Corporation Device 1ac2 (rev a1)
63:00.0 Bridge: NVIDIA Corporation Device 1ac2 (rev a1)
65:00.0 Bridge: NVIDIA Corporation Device 1ac2 (rev a1)
66:00.0 Bridge: NVIDIA Corporation Device 1ac2 (rev a1)
67:00.0 Bridge: NVIDIA Corporation Device 1ac2 (rev a1)
```

1.6.2. The NVIDIA HGX A100 GPU Baseboard

Figure 1.4 shows a simplified NVIDIA HGX A100 baseboard diagram.

The NVIDIA HGX A100 baseboard PCIe topology is like an HGX-2 baseboard with eight A100 GPUs and six corresponding second-generation NVSwitches. The eight GPUs and six NVSwitches will appear on the PCIe tree as PCIe devices on the host system.

Here is an example:

```
$ lspci | grep -i nvidia
36:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
3b:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
41:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
45:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
59:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
5d:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
```

(continues on next page)

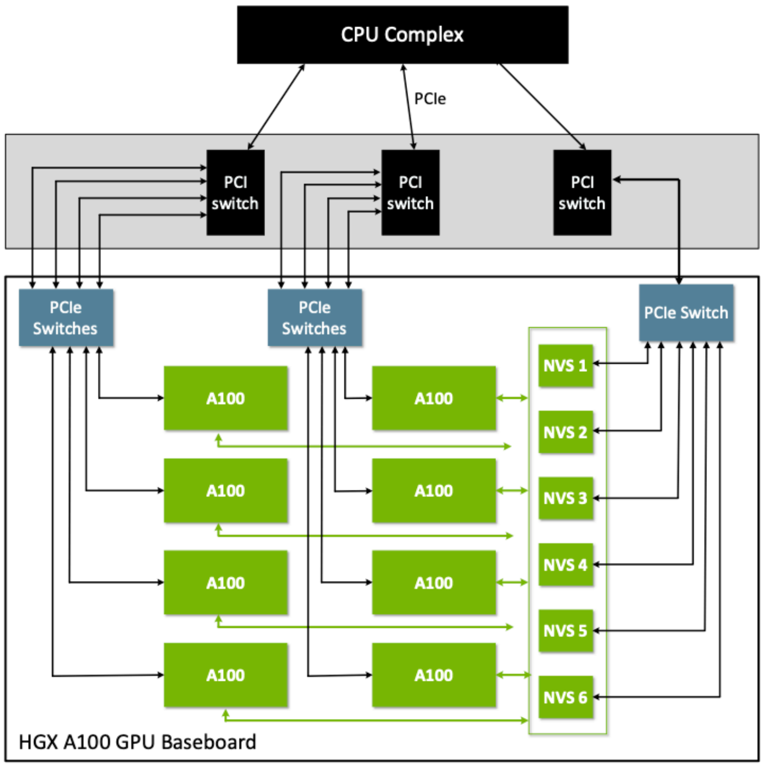

Figure 1.4: Simplified HGX A100 Baseboard

(continued from previous page)

```
63:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
67:00.0 3D controller: NVIDIA Corporation Device 20b0 (rev a1)
6d:00.0 Bridge: NVIDIA Corporation Device 1af1 (rev a1)
6e:00.0 Bridge: NVIDIA Corporation Device 1af1 (rev a1)
6f:00.0 Bridge: NVIDIA Corporation Device 1af1 (rev a1)
70:00.0 Bridge: NVIDIA Corporation Device 1af1 (rev a1)
71:00.0 Bridge: NVIDIA Corporation Device 1af1 (rev a1)
72:00.0 Bridge: NVIDIA Corporation Device 1af1 (rev a1)
```

1.6.3. The NVIDIA HGX H100 GPU Baseboard

Figure 1.5 shows an NVIDIA HGX H100 GPU baseboard.

The NVIDIA HGX H100 baseboard PCIe topology has eight GPUs and four NVSwitches on the PCIe tree as PCIe devices on the host system.

Here is an example:

```
$ lspci | grep -i nvidia
07:00.0 Bridge: NVIDIA Corporation Device 22a3 (rev a1)
08:00.0 Bridge: NVIDIA Corporation Device 22a3 (rev a1)
09:00.0 Bridge: NVIDIA Corporation Device 22a3 (rev a1)
0a:00.0 Bridge: NVIDIA Corporation Device 22a3 (rev a1)
1b:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
```

(continues on next page)

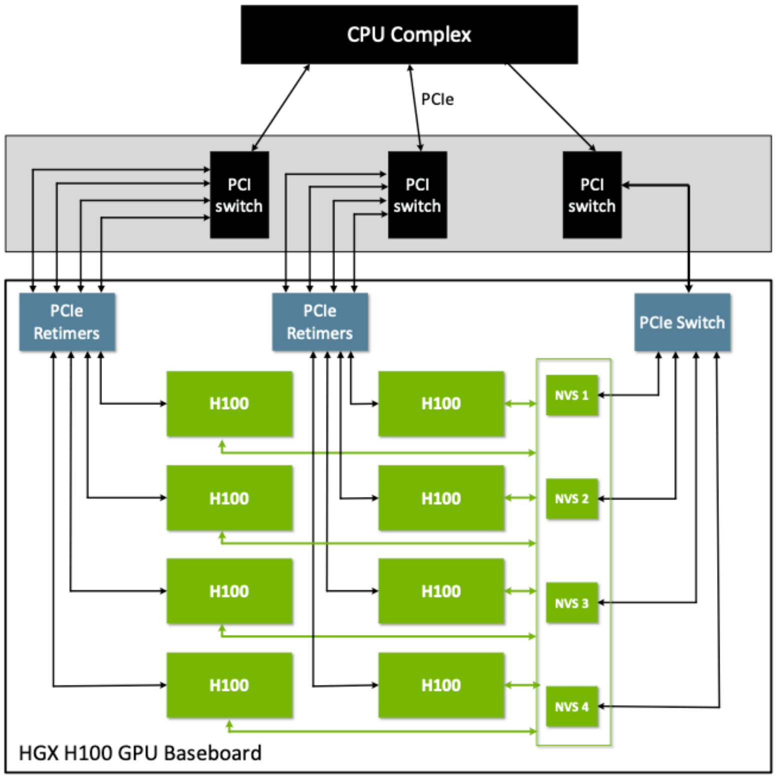

Figure 1.5: A Simplified Simple NVIDIA HGX H100 Baseboard Diagram

(continued from previous page)

```
43:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
52:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
61:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
9d:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
c3:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
d1:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
df:00.0 3D controller: NVIDIA Corporation Device 2330 (rev a1)
```

1.6.4. NVIDIA HGX B200/B100 GPU Baseboard

Figure 1.6 shows a simplified NVIDIA HGX B200/B100 baseboard diagram.

In the NVIDIA HGX B200/B100 baseboard PCIe topology, NVSwitches are not recognized as PCIe devices on the host system. To manage NVLinks, the NVSwitches are connected to a CX7 Bridge device. The host stack and control plane access are routed through this CX7 bridge device and use the corresponding in-box OFED or the MOFED driver driver.

The CX7 bridge device is integrated into the GPU baseboard, which includes two physical ports. Each port exposes one physical function (FC PF) and one Limited physical function (LPF) to the host system, which totals four PFs. The PFs are categorized into the following PFs:

- Limited PFs (LPF) are designated for specific tasks in the system.

They are used by the FM and the NVLSM to configure and set up NVSwitches, GPU, and NVLink routing

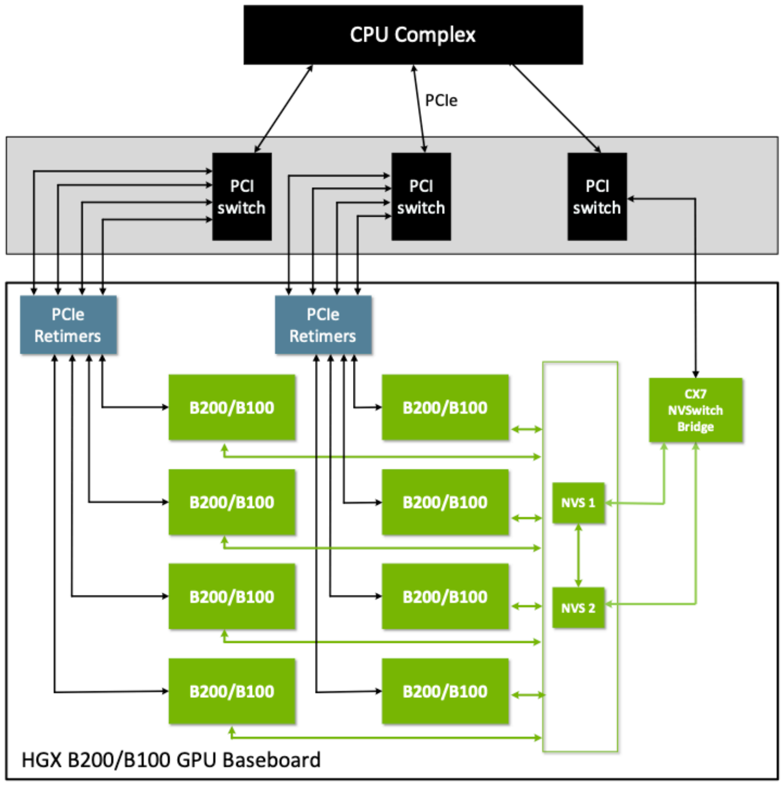

Figure 1.6: A Simplified NVIDIA HGX B200/B100 Baseboard Diagram

information. LPFs are also used by telemetry agents, such as NVIBDM and DCGM, to monitor and collect data. Resetting this PF with FLR also resets the corresponding NVSwitch device.

- Full Capabilities PF (FC PF) provides device administration level functionalities, such as issuing NVSwitch device resets and enabling or disabling links between NVSwitches.

FC PFs are valuable for partial pass-through virtualization scenarios, where subsets of GPUs and NVSwitches are allocated to Tenant VMs. However, this PF type does not support NVLink control plane entities, such as FM and NVLSM, and communication with telemetry agents.

From a hardware perspective, the CX7 Bridge device for NVLink management and traditional CX7 NICs share identical hardware. The PCIe Vital Product Data (VPD) information, which is programmed during production, differentiates the CX7 device for NVLink Management:

- On Linux-based systems, VPD information can be accessed using standard tools such as `vpddecode` or `lspci` commands.
- On Windows-based host systems, to query VPD information, run the `mstreg` command.

To differentiate between LPFs and FC PFs, the LPF VPD information includes a vendor-specific field called **SMDL**, with a non-zero value defined as `SW_MNG`. For bare-metal, full pass-through, and shared NVSwitch deployments, the prelaunch script in the FM service unit file will run and query the available CX7 devices for this VPD information. The file populates the required FM and NVLSM configuration values so that these communication entities can access the relevant devices.

However, for partial pass-through deployments, additional steps are required at the hypervisor level to identify the pair of PFs that belong to a CX7 bridge device port. Refer to *Virtualization Models* for more information.

The NVIDIA HGX B200/B100 baseboard PCIe topology will display eight GPUs and four CX7 devices on the PCIe tree as PCIe devices on the host system.

Here is an example:

```
$ lspci | grep -i -E 'nvidia|mella'
05:00.0 Infiniband controller: Mellanox Technologies MT2910 Family [ConnectX-
↪7]
05:00.1 Infiniband controller: Mellanox Technologies MT2910 Family [ConnectX-
↪7]
05:00.2 Infiniband controller: Mellanox Technologies MT2910 Family [ConnectX-
↪7]
05:00.3 Infiniband controller: Mellanox Technologies MT2910 Family [ConnectX-
↪7]
1b:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
43:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
52:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
61:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
9d:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
c3:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
d1:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
df:00.0 3D controller: NVIDIA Corporation Device 29bc (rev a1)
```

The GPU and NVSwitch PCIe Device ID, Product ID information in the `lspci` command output above is only provided as a sample. Actual values might vary depending on the specific product and GPU variations in the system configuration.

Chapter 2. Getting Started with Fabric Manager

2.1. Basic Components

This section provides information about the basic components in FM.

2.1.1. The Fabric Manager Service

The core component of FM is implemented as a standalone executable file that runs as a UNIX daemon process. The FM installation package installs the required core components and registers the daemon as the `nvidia-fabricmanager` system service.

On DGX-B200, NVIDIA HGX-B200, NVIDIA HGX-B100 systems and later, the FM package needs an additional NVLSM dependency to get the SM package for proper operation. The FM service unit file is also updated to start the NVLSM process if applicable. In this case, the FM systemd service status indicates the process status for FM and NVLSM, and operations such as systemd start, stop, and so on will operate on both processes.

2.1.2. Software Development Kit

FM also provides a shared library, a set of C/C++ APIs (SDK), and the corresponding header files. These APIs are used to interface with the FM service to query/activate/deactivate GPU partitions when FM is running in shared NVSwitch and vGPU multi-tenancy modes. These SDK components are installed through a separate development package (refer to *Shared NVSwitch Virtualization Model* and *vGPU Virtualization Model*).

2.2. Supported Platforms

This section provides information about the products and environments that FM currently supports.

2.2.1. Hardware Architectures

Here is a list of the hardware architectures:

- x86_64

- aarch64

2.2.2. NVIDIA Server Architectures

Here is a list of the server architectures:

- DGX-2 and NVIDIA HGX-2 systems that use V100 GPUs and first-generation NVSwitches.
- DGX A100 and NVIDIA HGX A100 systems that use A100 GPUs and second-generation NVSwitches.
- NVIDIA HGX A800 systems that use A800 GPUs and second-generation NVSwitches.
- DGX H100 and NVIDIA HGX H100 systems that use H100 GPUs and third-generation NVSwitches.
- NVIDIA HGX H800 systems that use H800 GPUs and third-generation NVSwitches.
- DGX H200 and NVIDIA HGX H200 systems that use H200 GPUs and third-generation NVSwitches.
- NVIDIA HGX H20 systems that use H20 GPUs and third-generation NVSwitches.
- DGX B200 and NVIDIA HGX B200 systems that use B200 GPUs and fourth generation NVSwitches.
- NVIDIA HGX B100 systems that use B100 GPUs and fourth generation NVSwitches.

Note

Unless specified, the steps for NVIDIA HGX A800 is same as the steps for NVIDIA HGX A100. The only difference is that the number of GPU NVLinks will defer depending on the actual platform.

Note

Unless specified, the steps for NVIDIA HGX H800, DGX H200, NVIDIA HGX H200, NVIDIA HGX H20 are the same as the steps for NVIDIA HGX H100. The only difference is that the number of GPU NVLinks might be different depending on the actual platform.

Note

Unless specified, the steps for NVIDIA HGX B100 are the same as the steps for NVIDIA HGX B200. The only difference is that the platform uses B100 GPU variant.

2.2.3. OS Environment

FM is supported on the following major Linux OS distributions:

- RHEL/CentOS 7.x, RHEL/CentOS 8.x and RHEL/CentOS 9.x
- Ubuntu18.04.x, Ubuntu 20.04.x, Ubuntu 22.04.x and Ubuntu 24.0x

Note

DGX B200, NVIDIA HGX B200, and NVIDIA HGX B100 systems use B200/B100 GPUs, and the fourth generation NVSwitches requires the v5.17 or later Linux kernel. If your kernel version is older than the supported version, NVIDIA provides a list of kernel patches that need to be backported.

2.3. Supported Deployment Models

NVSwitch-based systems can be deployed as bare metal servers or in a virtualized (full passthrough, Shared NVSwitch, or vGPU) multi-tenant environment. FM supports these deployment models. Refer to the following sections for more information:

- *Bare Metal Mode*
- *Full Passthrough Virtualized Configurations*
- *Shared NVSwitch Virtualization Configurations*
- *Bare Metal and vGPU Configurations*

2.4. Other NVIDIA Software Packages

To run the FM service, the target system must include a compatible driver, starting with version R450, for the NVIDIA Data Center GPUs.

On DGX B200, NVIDIA HGX B200, and NVIDIA HGX B100 systems, an OFED or a MOFED driver is required. In addition, the system needs to be installed with `libibumad3` and `infiniband-diags` packages. For example, here are the packages on an Ubuntu system:

- `apt-get install libibumad3`
- `apt-get install infiniband-diag`

Note

During initialization, the FM service checks the currently loaded kernel driver stack version for compatibility, and if the loaded driver stack version is not compatible, aborts the process.

2.5. Installation

Refer to <project:#Bare Metal Mode> for more information about installing and configuring FM for DGX and NVIDIA HGX NVSwitch-based systems.

2.6. Managing the Fabric Manager Service

This section provides information about managing the FM service.

2.6.1. Starting Fabric Manager

To start FM, for Linux based OS distributions, run the following command:

```
sudo systemctl start nvidia-fabricmanager
```

2.6.2. Stopping Fabric Manager

To stop FM, for Linux based OS distributions, run the following command:

```
sudo systemctl stop nvidia-fabricmanager
```

2.6.3. Checking the Fabric Manager Status

To check FM, for Linux based OS distributions, run the following command:

```
sudo systemctl status nvidia-fabricmanager
```

2.6.4. Enabling the Fabric Manager Service to Auto Start at Boot

To enable FM, for Linux based OS distributions, run the following command:

```
sudo systemctl enable nvidia-fabricmanager
```

2.6.5. Disabling the Fabric Manager Service Auto Start at Boot

To prevent FM from starting at boot, for Linux based OS distributions, run the following command:

```
sudo systemctl disable nvidia-fabricmanager
```

2.6.6. Checking the Fabric Manager System Log Messages

To view FM log messages, for Linux based OS distributions, run the following command:

```
sudo journalctl -u nvidia-fabricmanager
```

2.7. Fabric Manager Startup Options

FM supports the following command-line options:

```
$ nv-fabricmanager -h
    NVIDIA Fabric Manager
    Runs as a background process to configure the NVSwitches to form
    a single memory fabric among all participating GPUs.
    Usage: nv-fabricmanager [options]
```

(continues on next page)

(continued from previous page)

```
    Options include:
    [-h | --help]:              Displays help information
    [-v | --version]:           Displays the Fabric Manager version and exit.
    [-c | --config]:            Provides Fabric Manager config file path/name
↪which controls all the config options.
    [-r | --restart]:           Restart Fabric Manager after exit. Applicable
↪to Shared NVSwitch and vGPU multitenancy modes.
    [-g | --fm-sm-mgmt-port-guid]: Fabric Manager and NVLink Subnet Manager
↪management port GUID for Control Traffic.
    [-d | --database]:                 Provides Fabric Manager database engine
```

Most of the FM configurable parameters and options are specified through a text config file. The FM installation copies a default config file to a predefined location, and the file will be used by default. To use a different config file location, use the `[-c | --config]` command-line argument.

Note

On Linux-based installations, the default FM config file will be in the `/usr/share/nvidia/nvswitch/fabricmanager.cfg` directory. If the default config file on the system is modified, to manage the existing config file, an FM package update will provide the merge/keep/overwrite options. The `[-d --database]` option is not applicable and should not be used for single-node HGX/DGX systems.

2.8. Fabric Manager Service File

This section provides information about the FM service file.

2.8.1. Linux-Based Systems

On Linux-based systems, depending on the underlying GPU baseboard variants, the FM service unit file comprises logic to start the FM and NVLSM daemon processes. The installation package registers the FM service using the `systemd` service unit file. To change the FM service start-up options, modify this file in the `/lib/systemd/system/nvidia-fabricmanager.service` directory.

Here is an example of the system file:

```
[Unit]
Description=NVIDIA fabric manager service
After=network-online.target
Requires=network-online.target
After=multi-user.target

[Service]
User=root
PrivateTmp=false
Type=forking
TimeoutStartSec=720

Environment="FM_CONFIG_FILE=/usr/share/nvidia/nvswitch/fabricmanager.cfg"
```

(continues on next page)

(continued from previous page)

```
Environment="FM_PID_FILE=/var/run/nvidia-fabricmanager/nv-fabricmanager.pid"
Environment="NVLSM_CONFIG_FILE=/usr/share/nvidia/nvlsm/nvlsm.conf"
Environment="NVLSM_PID_FILE=/var/run/nvidia-fabricmanager/nvlsm.pid"
Environment="NVLSM_LOG_FILE=/var/log/nvlsm.log"

PIDFile=/var/run/nvidia-fabricmanager/nv-fabricmanager.pid

ExecStart=/usr/bin/nvidia-fabricmanager-start.sh $FM_CONFIG_FILE $FM_PID_FILE
↪$NVLSM_CONFIG_FILE $NVLSM_PID_FILE $NVLSM_LOG_FILE
ExecStop=/bin/sh -c '\
  sed -i "/^FM_SM_MGMT_PORT_GUID=0x[a-fA-F0-9]\\+$/d" "$FM_CONFIG_FILE"; \
  if [ -f "$NVLSM_CONFIG_FILE" ]; then \
    sed -i "/^guid 0x[a-fA-F0-9]\\+$/d" "$NVLSM_CONFIG_FILE"; \
  fi; \
  if [ -f "$FM_PID_FILE" ] && [ -s "$FM_PID_FILE" ]; then \
    kill "$(cat "$FM_PID_FILE")"; \
  fi; \
  if [ -f "$NVLSM_PID_FILE" ] && [ -s "$NVLSM_PID_FILE" ]; then \
    kill "$(cat "$NVLSM_PID_FILE")"; \
  fi'
LimitCORE=infinity

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

The contents of the `nv-fabricmanager-start.sh` script that is used above to selectively start FM and NVLSM process depends on the underlaying platform:

```
#!/bin/sh

# flag to capture valid port_guids
capture_flag=false

# flag to indicate pre/post nvl5 generation
post_nvl5=false

# Temp input file
input_file="/tmp/nvl_fmsm_ibstat_input.txt"

# FM config file: /usr/share/nvidia/nvswitch/fabricmanager.cfg
fm_config_file=$1

# FM pid file: /var/run/nvidia-fabricmanager/nvidia-fabricmanager.pid
fm_pid_file=$2

# NVLSM config file: /usr/share/nvidia/nvlsm/nvlsm.cfg
nvlsm_config_file=$3

# NVLSM pid file: /var/run/nvidia-fabricmanager/nvlsm.pid
nvlsm_pid_file=$4

# NVLSM log file: /var/log/nvlsm.log
```

(continues on next page)

(continued from previous page)

```
nvlsm_log_file=$5

# ibstat output capture in input_file for further processing
launch_ibstat() {
    ibstat $1 > $input_file
}

launch_fm_proc() {
    /usr/bin/nv-fabricmanager -c $fm_config_file
}

remove_cruft_files() {
    rm $input_file
}

# Use -J pidfile and -B daemonize option in config file
launch_nvlsm_proc() {
    /opt/nvidia/nvlsm/sbin/nvlsm -F $nvlsm_config_file -B --pid_file $nvlsm_
↪pid_file -f $nvlsm_log_file
}

clear_guid_entries() {
    sed -i "/^FM_SM_MGMT_PORT_GUID=0x[a-fA-F0-9]\\+$/d" $fm_config_file && sed
↪-i "/^guid 0x[a-fA-F0-9]\\+$/d" $nvlsm_config_file
}

# process input_file to find port_guids that have has_smi bit set
capture_portguids() {
    # Clear any GUID entries if any present
    clear_guid_entries

    # Parse input file line by line
    while IFS= read -r line; do
        # For each line, check if it contains "Capability mask"
        if echo "$line" | grep -q "Capability mask:"; then
            # Isolate mask value
            hex_value=$(echo "$line" | sed -En 's/.*Capability mask: 0x([a-fA-
↪F0-9]+).*/\1/p')
            prefix_hex_value="0x$hex_value"

            # Check if isSMDisabled bit is unset. Bit position starts at 0
            bit_position=10
            mask=$((1<<bit_position))
            if [ $((prefix_hex_value & mask)) -eq 0 ]; then
                # Set flag to true to parse and grab next
                # occurence of port_guid
                capture_flag=true
            fi
        elif echo "$line" | grep -q "Port GUID:" && [ "$capture_flag" = true
↪]; then
            # Extract port guid from the line
            port_guid=$(echo "$line" | sed -En 's/.*Port GUID: 0x([a-fA-F0-
```

(continues on next page)

(continued from previous page)

```
↪9]+).*/\1/p')
            prefix_port_guid="0x$port_guid"
            echo "FM_SM_MGMT_PORT_GUID=$prefix_port_guid" >> "$fm_config_file"
            echo "guid $prefix_port_guid" >> "$nvlsm_config_file"
            return 0
        fi
    done < "$input_file"
}

# Loop through each Infiniband device directory
for dir in /sys/class/infiniband/*/device; do
    # Define the path to the VPD file
    vpd_file="$dir/vpd"

    # Check if the VPD file exists
    if [ -f "$vpd_file" ]; then
        # Search for 'SW_MNG' in the VPD file
        if grep -q "SW_MNG" "$vpd_file"; then
            # Extract the Infiniband device name using parameter expansion
            device_name="${dir%/device}"  # Removes '/device' from the end of
↪$dir
            device_name="${device_name##*/}"  # Extracts the part after the
↪last '/'
            launch_ibstat $device_name
            capture_portguids
            post_nvl5=true
            # if port guid was captured, return since we are only grabbing
↪first GUID
            if [ "$capture_flag" = "true" ]; then
                break
            fi
        fi
    fi
done

if [ "$post_nvl5" = "true" ]; then
    launch_nvlsm_proc
    # SM takes few mins to start the GRPC service, hence lets wait before
↪starting FM
    sleep 5
    launch_fm_proc
else
    launch_fm_proc
fi
if [ -f "$input_file" ]; then
    remove_cruft_files
fi
```

Note

The `systemd` and `nv-fabricmanager-start.sh` scripts assumes default installation path for the

PID file, the binaries, and the config file location. If these default paths/files are modified, this change must be made to these files and to the start up script.

2.9. Running Fabric Manager as a Non-Root User

On Linux-based systems, by default, the FM and NVLSM service requires administrative (root) privileges to configure the GPU NVLinks and NVSwitches and support a memory fabric. However, system administrators and advanced users can complete the following steps to run FM and NVLSM from a non-root account:

1. If the FM service is running, stop it.
2. Provide FM the required access to the following directories and files by adjusting the corresponding directory/file access to the desired user/user group.

- `/var/run/nvidia-fabricmanager`

  This option provides a fixed location to save the runtime information.

- `/var/log/`

  This option provides a configurable location to save the FM log file.

- `/usr/share/nvidia/nvswitch`

  This option provides a configurable location for the fabric topology files.

This configurable directory/file information is based on default FM config file options. If the default configuration values are changed, adjust the directory/file information accordingly.

3. Provide the following directory and file access for the following platforms:

- **NVIDIA HGX-2/NVIDIA HGX-A100/NVIDIA HGX-H100**

  The NVIDIA driver will create the following proc entry with default permission to root, and you need to change its read/write access to the desired user/user group.

  `/proc/driver/nvidia-nvlink/capabilities/fabric-mgmt`

- FM also requires access to the following device node files:
- `/dev/nvidia-nvlink`
- `/dev/nvidia-nvswitchctl`
- `/dev/nvidia-nvswitchX` (one for each NVSwitch device)
- `/dev/nvidiactl`
- `/dev/nvidiaX` (one for each GPU device)

  By default, these device node files are created by the `nvidia-modprobe` utility, which is installed as part of NVIDIA Driver package for Data Center GPUs, and includes access permission for all users. If these device node files are created manually or outside `nvidia-modprobe`, assign read/write access to the user/user group.

- **NVIDIA HGX-B200**
- `/dev/infiniband/umadX` (one for each CX7 bridge device port)
- `/dev/infiniband/issmX` (one for each CX7 bridge device port)

- `/sys/class/infiniband/mlx5_X/device/vpd`

  Read perm for a non-root user to read the vpd file that is required by the systemctl service.

4. (**For NVIDIA HGX-B200 systems only**) Provide access for the following NVLSM directories and files access.

- `/var/log/`

  The default location for for the NVLSM temporary file store, and the following files are created:

- `nvlsm-subnet.lst`
- `nvlsm.fdbs`
- `nvlsm.mcfdbs`
- `nvlsm-smdb.dump.tmp`
- `nvlsm-virtualization.dump.tmp`
- `nvlsm-routers.dump`
- `nvlsm.log`
- `nvlsm-activity.dump`
- `nvlsm-unhealthy-ports.dump`
- `nvlsm-perflog.json`
- `nvlsm-perflog.backup.json`
- `/var/cache/nvlsm`

  The is the default directory where NVLSM stores state information so that it can reload and ensure that subsequent runs are consistent. This directory includes the `guid2lid` file.

- `/usr/share/nvidia/nvlsm`
- `nvlsm.cfg`

  This option provides a location to save the NVLSM configuration information. To send and receive management datagrams, the systemd service edits the nvlsm.cfg file to point to the `cx port guid` that is used by the `nvidia-fabricmanager` service.

5. The NVIDIA driver creates/recreates the above `/proc` entry during driver load, so repeat steps 1-6 on every driver reload or system boot.

- When FM and NVLSM are configured as systemd services, the system administrator must edit the FM service unit file to instruct systemd to run FM, NVLSM, and the FM Startup script from a specific user/group.

  This user/group can be specified through the `User=` and `Group=` directive in the **`[Service]`** section of FM service unit file.

- The system administrator must ensure that the proc entry and associated file node permissions are changed to the user/user group **before** the FM service starts at system boot time.
- When FM and NVLSM are configured to run from a specific user/user group, the `nvswitch-audit` command-line utility should be started from the same user/user group account.

Note

System administrators can set up the necessary udev rules to automate the process of changing these proc entry permissions.

2.10. Fabric Manager Config Options

The configurable parameters and options used by FM are specified through a text config file. This section provides information about the currently supported configurable parameters and options.

Note

The FM config file is read as part of FM service startup. If you changed any options, for the new settings to take effect, restart the FM service.

2.10.1. Logging Related Config Items

This section provides information about logging-related configuration items.

2.10.1.1 Setting the Log File Location and Name

Here are the components to set the log file location and name:

- **Config Item**

```
LOG_FILE_NAME=<value>
```

- **Supported/Possible Values**

The complete path/filename string, with a maximum length of 255, for the log.

- **Default Value**

```
LOG_FILE_NAME=/var/log/fabricmanager.log
```

2.10.1.2 Setting the Log Level

Here are the components to set a log level:

- **Config Item**

```
LOG_LEVEL=<value>
```

- **Supported/Possible Values**
- **0**: All the logging is disabled.
- **1**: Set log level to CRITICAL and above.
- **2**: Set log level to ERROR and above.
- **3**: Set log level to WARNING and above.
- **4**: Set log level to INFO and above.
- **Default Value**

```
LOG_LEVEL=4
```

2.10.1.3 Setting the Log File Append Behavior

- **Config Item**

```
LOG_APPEND_TO_LOG=<value>
```

- **Supported/Possible Values**
- **0**: No, do not append to the existing log file. Overwrite the existing log file.
- **1**: Yes, append to the existing log file every time the FM service is started.
- **Default Value**

```
LOG_APPEND_TO_LOG=1
```

2.10.1.4 Setting the Log File Size

- **Config Item**

```
LOG_FILE_MAX_SIZE=<value>
```

- **Supported/Possible Values**
- The maximum log file size you want in MBs.
- After the specified size is reached, FM will skip additional logging to the specified log file.
- **Default Value**

```
LOG_FILE_MAX_SIZE=1024
```

2.10.1.5 Redirecting the Logs to Syslog

- **Config Item**

```
LOG_USE_SYSLOG=<value>
```

- **Supported/Possible Values**
- **0**: Use the specified log file to store the FM logs.
- **1**: Redirect the FM logs to `syslog` instead of file-based logging.
- **Default Value**

```
LOG_USE_SYSLOG=0
```

2.10.1.6 Rotation Settings

- **Config Item**

```
LOG_MAX_ROTATE_COUNT=<value>
```

- **Supported/Possible Values**
- **0**: The log is not rotated.

  Logging is stopped until the log file reaches the size specified in the `LOG_FILE_MAX_SIZE` option.
- **Non-zero**: Rotate the current log file after it reaches the individual log file size.

  The combined FM log size is `LOG_FILE_MAX_SIZE` multiplied by `LOG_MAX_ROTATE_COUNT` + 1. After this threshold is reached, the oldest log file will be purged.

- **Default Value**

```
LOG_MAX_ROTATE_COUNT=3
```

Note

The FM log is in a clear-text format and, to troubleshoot field issues, NVIDIA recommends that you run the FM service with logging enabled at the INFO level.

2.10.2. Operating Mode-Related Config Items

This section applies **only** to the Shared NVSwitch and vGPU Multitenancy deployment.

2.10.2.1 Fabric Manager Operating Mode

- **Config Item**

```
FABRIC_MODE=<value>
```

- **Supported/Possible Values**
- **`0`**: Start FM in bare metal or full passthrough virtualization mode.
- **1**: Start FM in Shared NVSwitch multi-tenancy mode.

  Refer to <project:#Shared NVSwitch Virtualization Configurations> for more information.
- **2**: Start FM in vGPU multitenancy mode.

  Refer to <project:#vgpu virtualization model> for more information.
- **Default Value**

```
FABRIC_MODE=0
```

Note

Although `SHARED_FABRIC_MODE` is still supported, we recommend you use `FABRIC_MODE` instead.

2.10.2.2 The Fabric Manager Restart Mode

- **Config Item**

```
FABRIC_MODE_RESTART=<value>
```

- **Supported/Possible Values**
- **`0`**: Start FM and complete the initialization sequence.
- **1**: Start FM and follow the Shared NVSwitch or vGPU multitenancy mode resiliency/restart sequence.

This option is equal to the `–restart` command-line argument and enables the Shared NVSwitch or vGPU multitenancy mode resiliency without modifying command-line arguments to the FM process. Refer to *Fabric Manager Resiliency* for more information on the FM resiliency flow.

- **Default Value**

```
FABRIC_MODE_RESTART=0
```

Note

Although the `SHARED_FABRIC_MODE_RESTART` configuration item is still supported, we recommend that you use `FABRIC_MODE_RESTART` instead.

2.10.2.3 The Fabric Manager API Interface

- **Config Item**

```
FM_CMD_BIND_INTERFACE =<value>
```

- **Supported/Possible Values**

For the shared NVSwitch and vGPU multitenancy operations, the network interface that allows the FM SDK/API that is used to listen and for the hypervisor to communicate with the running FM instance.

- **Default Value**

```
FM_CMD_BIND_INTERFACE=127.0.0.1
```

2.10.2.4 The Fabric Manager API TCP Port

- **Config Item**

```
FM_CMD_PORT_NUMBER=<value>
```

- **Supported/Possible Values**

For the shared NVSwitch and vGPU multi-tenancy operations, the TCP port number for the FM SDK/API for hypervisor to communicate with the running FM instance.

- **Default Value**

```
FM_CMD_PORT_NUMBER=6666
```

2.10.2.5 The Fabric Manager Domain Socket Interface

- **Config Item**

```
FM_CMD_UNIX_SOCKET_PATH=<value>
```

- **Supported/Possible Values**

For the shared NVSwitch and vGPU multi-tenancy operations, instead of the TCP/IP socket, this is the UNIX domain socket path for the FM SDK/API to listen and to communicate with the running FM instance.

- **Default Value**

```
FM_CMD_UNIX_SOCKET_PATH=<empty value>
```

2.10.2.6 The Fabric Manager State

- **Config Item**

```
STATE_FILE_NAME=<value>
```

- **Supported/Possible Values**

Specifies the filename that will be used to save the FM states and restart FM after a crash or a successful exit. This is only valid when the shared NVSwitch or vGPU multitenancy mode is enabled.

- **Default Value**

```
STATE_FILE_NAME =/tmp/fabricmanager.state
```

Note

This value is only effective on DGX A100, HGX A100, DGX H100, HGX H100 NVSwitch-based systems.

2.10.3. Miscellaneous Config Items

This section provides information about miscellaneous config items.

2.10.3.1 Preventing Fabric Manager from Daemonizing

- **Config Item**

```
DAEMONIZE=<value>
```

- **Supported/Possible Values**
- **0**: Do not daemonize and run FM as a normal process.
- **1**: Run the FM process as a UNIX daemon.
- **Default Value**

```
DAEMONIZE=1
```

2.10.3.2 Fabric Manager Communication Socket Interface

- **Config Item**

```
BIND_INTERFACE_IP=<value>
```

- **Supported/Possible Values**

The network interface to listen for the FM internal communication/IPC. This value should be a valid IPv4 address.

- **Default Value**

```
BIND_INTERFACE_IP=127.0.0.1
```

Note

This is only effective on DGX A100, HGX A100, DGX H100, and HGX H100 NVSwitch-based systems.

2.10.3.3 Fabric Manager Communication TCP Port

- **Config Item**

```
STARTING_TCP_PORT=<value>
```

- **Supported/Possible Values**

Starting TCP port number for the FM internal communication/IPC, and this value should be between 0 and 65535.

- **Default Value**

```
STARTING_TCP_PORT=16000
```

Note

This is only effective on DGX A100, HGX A100, DGX H100, HGX H100 NVSwitch based systems.

2.10.3.4 Unix Domain Socket for Fabric Manager Communication

- **Config Item**

```
UNIX_SOCKET_PATH=<value>
```

- **Supported/Possible Values**

Use the Unix Domain socket instead of the TCP/IP socket for FM internal communication/IPC. An empty value means that the Unix domain socket is not used.

- **Default Value**

```
UNIX_SOCKET_PATH=<empty value>
```

Note

This is only effective on DGX A100, HGX A100, DGX H100, HGX H100 NVSwitch based systems.

2.10.3.5 Socket for Fabric Manager and Subnet Manager Communication

- **Config Item**

```
FM_SM_IPC_INTERFACE=<value>
```

- **Supported/Possible Values**
- Ipv4: `address:port`
- IPv6: `address:port`
- Unix: `//absolute_path to socket file`
- **Default Value**

```
FM_SM_IPC_INTERFACE=/var/run/nvidia-fabricmanager/fm_sm_ipc.socket
```

2.10.3.6 Management Port GUID for Control Traffic

- **Config Item**

```
FM_SM_MGMT_PORT_GUID=<value>
```

- **Supported/Possible Values**

A U64 bit number queried from the CX device to allow FM to communicate with underlying NVSwitches. If the underlying system is HGX B200 and a CX bridge device for NVLink fabric management is found, this information will be populated by the FM service startup script. If the command line to the FM binary and config option using the fabricmanager.cfg file are provided, the command line will take precedence.

- **Default Value**

```
FM_SM_MGMT_PORT_GUID=0x0
```

2.10.3.7 Fabric Manager System Topology File Location

- **Config Item**

```
TOPOLOGY_FILE_PATH =<value>
```

- **Supported/Possible Values**

Configuration option to specify the FM topology files directory path information.

- **Default Value**

```
TOPOLOGY_FILE_PATH=/usr/share/nvidia/nvswitch
```

Note

This topology file config option is not applicable to DGX B200 and NVIDIA HGX B200 and later NVSwitch-based systems.

2.10.4. High Availability Mode-Related Config Items

This section provides information about high availability mode-related config items.

2.10.4.1 Control Fabric Manager Behavior with An Initialization Failure

- **Config Item**

```
FM_STAY_RESIDENT_ON_FAILURES=<value>
```

- **Supported/Possible Values**
- `0`: The FM service will terminate on NVSwitch and a GPU config failures, typical software errors, and so on.
- **1**: The FM service will stay running on NVSwitch and GPU config failures, typical software errors, and so on.

However, the system will be uninitialized, and the CUDA application launch will fail.

- **Default Value**

```
FM_STAY_RESIDENT_ON_FAILURES=0
```

2.10.4.2 GPU Access NVLink Failure Mode

- **Config Item**

```
ACCESS_LINK_FAILURE_MODE=<value>
```

- **Supported/Possible Values**

The available high-availability options when there is an Access NVLink Failure (GPU to NVSwitch NVLink). Refer to *Supported High Availability Modes* for more information about supported values and behavior.

- **Default Value**

```
ACCESS_LINK_FAILURE_MODE=0
```

2.10.4.3 NVSwitch Trunk NVLink Failure Mode

- **Config Item**

```
TRUNK_LINK_FAILURE_MODE=<value>
```

- **Supported/Possible Values**

The available high-availability options when there is a Trunk Link failure (NVSwitch to NVSwitch connection between GPU baseboards). Refer to *Supported High Availability Modes* for more information about supported values and behavior.

- **Default Value**

```
TRUNK_LINK_FAILURE_MODE=0
```

2.10.4.4 NVSwitch Failure Mode

- **Config Item**

```
NVSWITCH_FAILURE_MODE=<value>
```

- **Supported/Possible Values**

The available high-availability options when there is an NVSwitch failure. Refer to *Supported High Availability Modes* for more information about supported values and behavior.

- **Default Value**

```
NVSWITCH_FAILURE_MODE=0
```

2.10.4.5 CUDA Jobs When the Fabric Manager Service is Stopped or is Terminated

- **Config Item**

```
ABORT_CUDA_JOBS_ON_FM_EXIT=<value>
```

- **Supported/Possible Values**
- **`0`**: Do not abort running CUDA jobs when the FM service is stopped or exits.

  A new CUDA job launch will fail with a `cudaErrorSystemNotReady` error.
- **1**: Abort all running CUDA jobs when the FM service is stopped or exits.

  A new CUDA job launch will fail with a `cudaErrorSystemNotReady` error.

Note

Here is some important information about this config method: It is not effective on DGX H100 and NVIDIA HGX H100 and later NVSwitch-based systems. It applies to only bare metal and full passthrough virtualization models.

- **Default Value**

```
ABORT_CUDA_JOBS_ON_FM_EXIT=1
```

Chapter 3. Getting Started with NVLink Subnet Manager

This chapter provides information about NVLSM.

3.1. NVLink Subnet Manager Configuration

The NVLink Subnet Manager (NVLSM) configuration options and parameters are specified in the `nvlsm.conf` file, and by default, this file will be loaded from the `/usr/share/nvidia/nvswitch/` directory.

Here is some additional information:

- You can override the contents of nvlsm.conf, but an incorrect override will result a fabric management failure.
- If a configuration option is not specified in SM configuration file, SM will use the default value.
- The options discussed below are additions to the configuration file that allow users to control certain operational aspects of nvlink subnet manager.

Ensure that you retain the original configuration file content when you configure the file with the options below.

To use a different configuration file, run NVLSM with `[-F <path> | --config <path>]` command-line argument, and the configuration file format is `<key> <value>`.

For example, to bind NVLSM to the IB port with port GUID 0x0001, the NVLSM configuration file should contain the following line:

```
guid 0x0001
```

Note

The NVLSM config file is read as part of NVLSM service startup. If you changed any configuration options, for the new settings to take effect, restart the FM service. This process restarts FM and NVLSM.

3.1.1. Configuring the NVLink Subnet Manager Port

This option controls the port to which NVLSM binds by setting the value to the port's GUID. This information will be populated by the FM service launch script that runs as part of the FM systemd service.

- **Config Item**

```
guid <value>
```

- **Supported/Possible Values**

IB port GUID.

3.1.2. Configuring the NVLink Subnet Manager Daemon Mode

This option controls nvlsm to run in daemon mode.

- **Config Item**

```
daemon <value>
```

- **Supported/Possible Values**
- `True`

  If set to `True`, NVLSM will be started in the daemon process mode.
- `False`

  If set to `False`, NVLSM will start in the foreground process mode
- **Default Value**

```
daemon false
```

3.1.3. Configuring NVLink Subnet Manager to Load the Fabric Manager GRPC Plugin

To configure NVLSM to load the plugin for NVLSM->FM communication, use the following plug-in configuration.

- **Config item**

```
plugin_name grpc_mgr
```

3.1.4. Configuring GRPC Plugin Properties

The NVLSM GRPC plugin configuration settings are completed by passing parameters using the NVLSM's `plugin_options` parameter by specifying the plugin name, the plugin parameters, and their values.

- **Config item**

```
Plugin_optoin [<plugin name> <plugin parameter> <value>]
```

- To configure the GRPC plugin listening address, set the GRPC plugin's `grpc_server_address` parameter.

- The following example configures the GRPC plugin to listen to the `/var/run/nvidia-fabricmanager/fm_sm_ipc.socket` UNIX domain socket file path.

```
plugin_options grpc_mgr --grpc_server_address unix:/var/run/nvidia-fabricmanager/fm_sm_ipc.socket
```

3.1.4.1 Setting the Log File Location and Name

This option controls the NVLSM log file location.

- **Config Item**

```
log_file <value>
```

- **Supported/Possible Values**

The complete path/filename string, with a maximum length of 256, for the log.

- **Default Value**

```
log_file /var/log/opensm.log
```

3.1.4.2 Setting a Log Level

This option is a flags field that control log verbosity level.

- **Config Item**

```
log_flags <value>
```

- **Supported/Possible bit flags**
- 0x01: ERROR (error messages)
- 0x02: INFO (basic messages, low volume)
- 0x04: VERBOSE (additional informative messages, moderate volume)
- 0x08: DEBUG (diagnostic, high volume)
- 0x10: FUNCS (function entry/exit, very high volume)
- 0x20: FRAMES (dumps all SMP and GMP frames)
- 0x40: ROUTING (dump FDB routing information)
- 0x80: SYS (syslog at LOG_INFO level in addition to NVLSM `logging`)
- **Default Value**

```
log_flags 0x3
```

3.1.4.3 Redirecting the Logs to the Syslog

This option is a flag that controls the verbosity level of messages to write to Syslog.

- **Config Item**

```
syslog_log_flags <value>
```

- **Supported/Possible bit flags**
- 0x01: ERROR (error messages)
- 0x02: INFO (basic messages, low volume)
- 0x04: VERBOSE (additional informative messages, moderate volume)

- 0x08: DEBUG (diagnostic, high volume)
- 0x10: FUNCS (function entry/exit, very high volume)
- **Default Value**

```
syslog_log_flags 0x0
```

3.1.4.4 Setting the Log File Append Behavior

This option determines whether to append the NVLSM log over multiple NVLSM sessions or to truncate the existing log file at startup.

- **Config Item**

```
accum_log_file <value>
```

- **Supported/Possible Values**
- TRUE: Accumulated log files.
- FALSE: Truncate the log file when NVLSM starts up.
- **Default Value**

```
accum_log_file TRUE
```

Chapter 4. Bare Metal Mode

The NVSwitch-based DGX and NVIDIA HGX server systems' default software configuration is to run the systems as bare-metal machines for workloads such as AI, machine learning, and so on. This chapter provides information about the FM installation requirements to support a bare-metal configuration.

4.1. Fabric Manager Packages

Each FM release comprises the following packages:

- **Core Fabric Manager** (`nvidia-fabricmanager-<version>`)

This package includes the essential components such as the core standalone FM service process, the service unit file, and the topology files. For bare metal, you can install just this package.

- **Fabric Manager SDK and Libaray** (`nvidia-fabricmanager-devel-<version>`)

The *devel* package includes the FM shared library and its associated header files. This package is important when you implement the Shared NVSwitch and vGPU multi-tenancy virtualization models.

4.2. Installing Fabric Manager

This section provides information about installing FM.

4.2.1. On NVSwitch-Based DGX Server Systems

As part of the supported DGX OS package installation, the FM service is preinstalled in all the NVSwitch-based DGX systems. The service is enabled and started when the OS boots, and the default installation configuration is to support bare metal mode.

4.2.2. On NVSwitch-Based NVIDIA HGX Server Systems

On NVSwitch-based NVIDIA HGX systems, to configure the NVLinks and NVSwitch memory fabrics to support one memory fabric, the FM service needs to be manually installed. The FM package is available through the NVIDIA CUDA network repository.

4.2.3. Systems Using NVSwitches that are Earlier than the Fourth-Generation NVSwitches

For systems that use NVSwitches earlier than the fourth generation, use the following installation instructions. These systems are defined as a generation before the the DGX B200, NVIDIA HGX B200 8-GPU, and NVIDIA HGX B100 8-GPU systems. Refer to NVIDIA Driver Installation Quickstart Guide for more information about setting up your system's package manager and download packages from the desired CUDA network repositories.

- For Debian and Ubuntu-based OS distributions:

```
sudo apt-get install cuda-drivers-fabricmanager-<driver-branch>
```

- For Red Hat Enterprise Linux 8 and 9-based OS distributions:

```
sudo dnf module install nvidia-driver:<driver-branch>/fm
```

- SUSE Linux-based OS distributions:

```
sudo zypper install cuda-drivers-fabricmanager-<driver-branch>
```

Note

In the commands above, *<driver-branch>* should be substituted with the required NVIDIA driver branch number for qualified datacenter drivers (for example, 450).

4.2.4. Systems Using Fourth Generation NVSwitches

- The DGX B200, NVIDIA HGX B200 8-GPU, and NVIDIA HGX B100 8-GPU systems use the fourth generation NVSwitches (based on NVIDIA NVLink5 protocol) and require an additional NVLSM service to configure the NVLinks and NVSwitches.

As a result, to get the required components, a different package installation is needed. Refer to NVIDIA Driver Installation Quickstart Guide for more information about setting up your system's package manager and download packages from the desired CUDA network repositories.

- For Debian and Ubuntu-based OS distributions:

```
sudo apt-get install -V nvidia-open-<driver-branch>
sudo apt-get install -V nvlink5-<driver-branch>
```

For example:

```
sudo apt-get install -V nvidia-open-570
sudo apt-get install -V nvlink5-570
```

- For Red Hat Enterprise Linux 8 and 9-based OS distributions:

```
sudo dnf module install nvidia-driver-<driver-branch>-open
sudo dnf install nvlink-<driver-branch>`
```

For example:

```
sudo dnf module install nvidia-driver-570-open
sudo dnf install nvlink-570
```

Note

On NVSwitch-based NVIDIA HGX systems, if you are using individual packages instead of a meta package-based installation, before you install FM, install the compatible Driver for NVIDIA Data Center GPUs. As part of the installation, the FM service unit file (`nvidia-fabricmanager.service`) will be copied to systemd. However, the system administrator must manually enable and start the FM service.

4.2.5. Minimum NVIDIA Driver/Fabric Manager Version

Here are the NVIDIA Data Center GPUs driver package minimum versions for the different platform:

- NVIDIA HGX-2 and NVIDIA HGX A100 systems: version 450.xx.
- NVIDIA HGX H100 systems: version 525.xx.
- NVIDIA HGX B200 and NVIDIA HGX B100 systems: version 570.xx.

The FM default installation mode and configuration file options support bare-metal mode.

4.3. Initializing NVSwitch and NVLink

NVIDIA GPUs and NVSwitch memory fabrics are PCIe endpoint devices that require an NVIDIA kernel driver to be used. On DGX-2, NVIDIA HGX-2, DGX A100, and NVIDIA HGX A100 systems that do not have ALI support, after the system boots, the NVLink connections are enabled after the NVIDIA kernel driver is loaded, and the FM configures these connections. CUDA initialization will fail with the `cudaErrorSystemNotReady` error if the application is launched before FM completely initializes the system or when FM fails to initialize the system.

On DGX H100 and NVIDIA HGX H100 and later systems that support ALI-based NVLink training, the NVLinks are trained at the GPU and NVSwitch hardware levels without FM. To enable NVLink peer-to-peer support, the GPUs must register with the NVLink fabric. If a GPU fails to register with the fabric, it will lose its NVLink peer-to-peer capability and be available for non-peer-to-peer use cases. The CUDA initialization process will start after the GPUs complete their registration process with the NVLink fabric.

GPU fabric registration status is exposed through the NVML APIs, and as part of `nvidia-smi -q` command. Refer the following `nvidia-smi` command output for more information.

Here is the Fabric state output when the GPU is being registered:

```
nvidia-smi -q -i 0 | grep -i -A 2 Fabric
            Fabric
                State                         : In Progress
                Status                        : N/A
```

Here is the Fabric state output after the GPU has been successfully registered:

```
nvidia-smi -q -i 0 | grep -i -A 2 Fabric
             Fabric
                State                         : Completed
                Status                        : Success
```

FM and NVLSM play critical roles in the NVSwitch-based system functionality that is typically initiated during a system boot or a workload activation. Restarting the service intermittently is unnecessary, but if a restart is necessary because of workflow requirements or as part of a GPU reset operation, complete the following steps:

For DGX H100 and NVIDIA HGX H100 systems and later systems, to ensure the system returns to a coherent state:

1. Stop all CUDA applications and GPU-related services.

   - Halt all running CUDA applications and services (for example, DCGM) that are actively using GPUs.
   - You can leave the `nvidia-persistenced` service running.

2. Stop the FM service.
3. To reset the GPU, run the `nvidia-smi -r` command.
4. Restart the FM service and restore its functionality.
5. Resume the stopped services that were halted in step 1.
6. Launch the CUDA applications.
7. After completing these steps, launch your CUDA applications as needed.

Note

System administrators can set their GPU application launcher services, such as SSHD, Docker, and so on to start after the FM service is started. Refer to your Linux distribution's manual for more information about setting up service dependencies and the service start order. Using infiniband tools over CX interface is not supported.

4.4. Runtime NVSwitch and GPU Errors

When an NVSwitch port or GPU generates a runtime error, the corresponding information will be logged into the operating system's kernel log or event log. An error report from NVSwitch will be logged with the `SXid` prefix, and a GPU error report will be logged with the `Xid` prefix by the NVIDIA driver.

The NVSwitch `SXids` errors use the following reporting convention:

```
<nvidia-nvswitchX: SXid (PCI:<switch_pci_bdf>):  <SXid_Value>, <Fatal or Non-
↪Fatal>, <Link No> < Error Description>
<raw error information for additional troubleshooting>


The following is an example of a SXid error log


[...] nvidia-nvswitch3: SXid (PCI:0000:c1:00.0): 28006, Non-fatal, Link 46 MC
↪TS crumbstore MCTO (First)
[...] nvidia-nvswitch3: SXid (PCI:0000:c1:00.0): 28006, Severity 0 Engine
↪instance 46 Sub-engine instance 00
[...] nvidia-nvswitch3: SXid (PCI:0000:c1:00.0): 28006, Data {0x00140004,
```

(continues on next page)

(continued from previous page)

```
↪0x00100000, 0x00140004, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
↪0x00000000}
```

The GPU `Xids` errors use the following reporting convention:

```
NVRM: GPU at PCI:<gpu_pci_bdf>: <gpu_uuid>
NVRM: GPU Board Serial Number: <gpu_serial_number>
NVRM: Xid (PCI:<gpu_pci_bdf>): <Xid_Value>, <raw error information>


The following is an example of a Xid error log
[...] NVRM: GPU at PCI:0000:34:00: GPU-c43f0536-e751-7211-d7a7-78c95249ee7d
[...] NVRM: GPU Board Serial Number: 0323618040756
[...] NVRM: Xid (PCI:0000:34:00): 45, Ch 00000010
```

Depending on the severity (fatal versus non-fatal) and the impacted port, the SXid and Xid errors can abort existing CUDA jobs and prevent new CUDA job launches. The next section provides information about the potential impact of SXid and Xid errors and the corresponding recovery procedure.

4.4.1. NVSwitch SXid Errors

This section provides information about the NVSwitch SXid errors.

4.4.1.1 Non-Fatal SXid Errors

Non-fatal SXids are for informational purposes only, and FM will not terminate CUDA jobs that are running or prevent new CUDA job launches. The existing CUDA jobs should resume, but depending on the error, CUDA jobs might experience issues such as a performance drop, no forward progress for brief time, and so on.

4.4.1.2 Fatal SXid Errors

When a fatal SXid error is reported on a NVSwitch port that connects a GPU and an NVSwitch, the corresponding error will be propagated to the GPU. The CUDA jobs that are running on that GPU will be aborted, and the GPU might report Xid 74 and Xid 45 errors. The FM service will log the corresponding GPU index and PCI bus information in its log file and syslog. The system administrator must use the following recovery procedure to clear the error state before using the GPU for an additional CUDA workload.

1. Reset the specified GPU and all the participating GPUs in the affected workload by using the NVIDIA System Management Interface (`nvidia-smi`) command-line utility.

Refer to the `-r` or the `--gpu-reset` options in `nvidia-smi` and the individual GPU reset operation for more information. If the problem persists, reboot or power cycle the system.

When a fatal SXid error is reported on a NVSwitch port that connects two GPU baseboards, FM will abort all the running CUDA jobs and prevent new CUDA job launches. The GPU will also report an Xid 45 error as part of aborting CUDA jobs. The FM service will log the corresponding error information in its log file and syslog.

2. To clear the error state and subsequent successful CUDA job launch, the system administrator must complete the following recovery procedure:

3. Reset all the GPUs and NVSwitches.

4. Stop the FM service.
5. Stop all the applications that are using the GPU.
6. Reset all the GPU and NVSwitches using the `nvidia-smi` command line utility with the `-r` or the `--gpu-reset` option.

   Do not use the `-i` or the `–id` options.
7. After the reset operation is complete, start the FM service again.
8. If the problem persists, reboot or power cycle the system.

Note

The NVSwitch Driver SXid fatal and non-fatal based error reporting does not apply on DGX B200 and NVIDIA HGX B200 systems.

4.4.2. NVSwitch Errors On DGX B200 and NVIDIA HGX B200 Systems

NVSwitch SXID errors are no longer applicable to DGX B200 and NVIDIA HGX B200 systems. DCGM now interfaces with a library called NVIDIA Switch Device Manager (NVSDM) to fetch errors related to NVSwitch. The following telemetry counters are retrieved from the NVSwitch:

- Port counters
- ASIC counters

Refer to NVOnline: **1115699** for more information.

4.4.3. GPU Xid Errors

GPU Xid messages indicate that a general GPU error occurred, and the messages can indicate one of the following issue types:

- A hardware problem
- An NVIDIA software problem
- A user application problem

When a GPU experiences an Xid error, the CUDA jobs that are running on that GPU will typically be aborted. Complete the GPU reset procedure in “NVSwitch SXid Errors” on page for more information.

On DGX H100 and NVIDIA HGX H100 systems, FM no longer monitors and logs GPU errors. The NVIDIA driver will continue to monitor and log GPU errors in the syslog.

4.5. Interoperability With Multi-Instance GPUs

Multi-Instance GPUs (MIGs) partition an NVIDIA A100, an H100, and a B200 GPU into many independent GPU instances. These instances run simultaneously, each with its own memory, cache and streaming, multiprocessors. However, when you enable the MIG mode, the GPU NVLinks will be disabled (or not used) and the GPU will lose its NVLink peer-to-peer (P2P) capability. After the MIG mode is successfully disabled, the GPU NVLinks will be enabled again, and the GPU NVLink P2P capability will be restored.

On NVSwitch-based DGX and NVIDIA HGX systems, the FM service interoperates with GPU MIG instances. To successfully restore GPU NVLink peer-to-peer capability after the MIG mode is disabled on these systems, the FM service must be running. On DGX A100 and NVIDIA HGX A100 systems, the corresponding GPU NVLinks and NVSwitch side NVLinks are trained off when MIG mode is enabled and are retrained when MIG mode is disabled. However, on DGX H100, NVIDIA HGX H100, and later systems, GPU NVLinks will stay active during MIG mode.

Chapter 5. Virtualization Models

NVSwitch-based systems support multiple models to isolate NVLink interconnects in a multi-tenant environment. In virtualized environments, VM workloads often cannot be trusted and must be isolated from each other and from the host or hypervisor. The switches used to maintain this isolation cannot be directly controlled by untrusted VMs and must be controlled by the trusted software.

This chapter provides a high-level overview of supported virtualization models.

5.1. Supported Virtualization Models

The NVSwitch-based systems support the following virtualization models:

- Full Passthrough
    - GPUs and NVSwitch memory fabrics are passed to the guest OS.
    - Easy to deploy and requires minimal changes to the hypervisor/host OS.
    - Reduced NVLink bandwidth for two and four GPU VMs.
    - For DGX H100, NVIDIA HGX H100, DGX H200, NVIDIA HGX H200, and NVIDIA HGX H20 systems, 4x GPU VMs will experience asymmetric GPU NVLink bandwidth due to the underlying physical NVLink topology.
    - For DGX B200 and NVIDIA HGX B200 systems, only two 2x GPU VMs are possible because the system has only two NVSwitches.
    - For 1x, 2x, or 4x GPU VMs, additional hypervisor configuration is required to disable GPU NVLinks that connect to other subsets of NVSwitches.
- Shared NVSwitch Multitenancy Mode
    - Only GPUs passed through to the guests.
    - NVSwitch memory fabrics are managed by a dedicated trusted VM called Service VM.
    - NVSwitch memory fabrics are shared by the guest VMs, but the fabrics are not visible to guests.
    - Requires the tightest integration with the hypervisor.
    - Complete bandwidth for two and four GPU VMs.
    - No need for direct communication between the guest VM and the Service VM.
    - The recommended option, as this model supports 1x, 2x, 4x, and 8x GPU VMs with consistent NVLink capabilities across all generations of HGX and DGX systems.
- vGPU Multitenancy Mode

- Only SR-IOV GPU VFs are passed through to the guests.
- GPU PFs and NVSwitch memory fabrics are managed by the vGPU host.
- NVSwitch memory fabrics are shared by all the guest VMs, but the fabrics are not visible to guests.
- Complete bandwidth for two and four GPU VMs.
- This mode is tightly coupled with the vGPU software stack.

Chapter 6. Fabric Manager SDK

FM provides a shared library, a set of C/C++ APIs (SDK), and the corresponding header files. The library and APIs are used to interface with FM when FM runs in the shared NVSwitch and vGPU multi-tenant modes to query, activate, and deactivate GPU partitions.

All FM interface API definitions, libraries, sample code, and associated data structure definitions are delivered as a separate development package (RPM/Debian). To compile the sample code in this user guide, install this package.

6.1. Data Structures

Here are the data structures:

```
// max number of GPU/fabric partitions supported by FM
#define FM_MAX_FABRIC_PARTITIONS 64


// max number of GPUs supported by FM
#define FM_MAX_NUM_GPUS      16


// Max number of ports per NVLink device supported by FM
#define FM_MAX_NUM_NVLINK_PORTS  64


// connection options for fmConnect()
typedef struct
{
    unsigned int version;
    char addressInfo[FM_MAX_STR_LENGTH];
    unsigned int timeoutMs;
    unsigned int addressIsUnixSocket;
} fmConnectParams_v1;


typedef fmConnectParams_v1 fmConnectParams_t;
// VF PCI Device Information
typedef struct
{
    unsigned int domain;
```

(continues on next page)

(continued from previous page)

```
    unsigned int bus;
    unsigned int device;
    unsigned int function;
} fmPciDevice_t;// structure to store information about a GPU belonging to
↪fabric partition
typedef struct
{
    unsigned int physicalId;
    char uuid[FM_UUID_BUFFER_SIZE];
    char pciBusId[FM_DEVICE_PCI_BUS_ID_BUFFER_SIZE];
    unsigned int numNvLinksAvailable;
    unsigned int maxNumNvLinks;
    unsigned int nvlinkLineRateMBps;
} fmFabricPartitionGpuInfo_t;


// structure to store information about a fabric partition
typedef struct
{
    fmFabricPartitionId_t partitionId;
    unsigned int isActive;
    unsigned int numGpus;
    fmFabricPartitionGpuInfo_t gpuInfo[FM_MAX_NUM_GPUS];
} fmFabricPartitionInfo_t;


// structure to store information about all the supported fabric partitions
typedef struct
{
    unsigned int version;
    unsigned int numPartitions;
    unsigned int maxNumPartitions;
    fmFabricPartitionInfo_t partitionInfo[FM_MAX_FABRIC_PARTITIONS];
} fmFabricPartitionList_v2;


typedef fmFabricPartitionList_v2 fmFabricPartitionList_t;


// structure to store information about all the activated fabric partitionIds
typedef struct
{
    unsigned int version;
    unsigned int numPartitions;
    fmFabricPartitionId_t partitionIds[FM_MAX_FABRIC_PARTITIONS];
} fmActivatedFabricPartitionList_v1;


typedef fmActivatedFabricPartitionList_v1 fmActivatedFabricPartitionList_t;


// Structure to store information about a NVSwitch or GPU with failed NVLinks
```

(continues on next page)

(continued from previous page)

```
typedef struct
{
    char         uuid[FM_UUID_BUFFER_SIZE];
    char         pciBusId[FM_DEVICE_PCI_BUS_ID_BUFFER_SIZE];
    unsigned int numPorts;
    unsigned int portNum[FM_MAX_NUM_NVLINK_PORTS];
} fmNvlinkFailedDeviceInfo_t;


// Structure to store a list of NVSwitches and GPUs with failed NVLinks
typedef struct
{
    unsigned int                 version;
    unsigned int                 numGpus;
    unsigned int                 numSwitches;
    fmNvlinkFailedDeviceInfo_t  gpuInfo[FM_MAX_NUM_GPUS];
    fmNvlinkFailedDeviceInfo_t  switchInfo[FM_MAX_NUM_NVSWITCHES];
} fmNvlinkFailedDevices_v1;


typedef fmNvlinkFailedDevices_v1 fmNvlinkFailedDevices_t;


/**
 * Structure to store information about a unsupported fabric partition
 */
typedef struct
{
    fmFabricPartitionId_t partitionId; //!< a unique id assigned to reference
↪this partition
    unsigned int numGpus;     //!< number of GPUs in this partition
    unsigned int gpuPhysicalIds[FM_MAX_NUM_GPUS];    //!< physicalId of each
↪GPU assigned to this partition.
} fmUnsupportedFabricPartitionInfo_t;
/**
 * Structure to store information about all the unsupported fabric partitions
 */
typedef struct
{
    unsigned int version;        //!< version number. Use
↪fmFabricPartitionList_version
    unsigned int numPartitions;   //!< total number of unsupported partitions
    fmUnsupportedFabricPartitionInfo_t partitionInfo[FM_MAX_FABRIC_
↪PARTITIONS];  /*!< detailed information of each

↪         unsupported partition*/
} fmUnsupportedFabricPartitionList_v1;
typedef fmUnsupportedFabricPartitionList_v1 fmUnsupportedFabricPartitionList_
↪t;
#define fmUnsupportedFabricPartitionList_version1 MAKE_FM_PARAM_
↪VERSION(fmUnsupportedFabricPartitionList_v1, 1)
#define fmUnsupportedFabricPartitionList_version
```

(continues on next page)

(continued from previous page)

```
↪fmUnsupportedFabricPartitionList_version1
```

Note

On DGX H100, NVIDIA HGX H100, and later systems, the GPU physical ID information has the same value as the GPU Module ID information that is returned by the `nvidia-smi-q` output. When reporting partition information, GPU information such as UUID, PCI Device (BDF) will be empty. To correlate between GPUs in the partition, the hypervisor stack should use GPU Physical ID information, and the GPUs needs to be assigned to corresponding partition's guest VM.

6.2. Initializing the Fabric Manager API interface

To initialize the FM API interface library, run the following command:

```
fmReturn_t fmLibInit(void)
Parameters
None
Return Values
        FM_ST_SUCCESS - if FM API interface library has been properly
↪initialized
FM_ST_IN_USE - FM API interface library is already in initialized state.
FM_ST_GENERIC_ERROR - A generic, unspecified error occurred
```

6.3. Shutting Down the Fabric Manager API interface

To shut down the FM API interface library and the remote connections that were established through `fmConnect()`, run the following command.

```
fmReturn_t fmLibShutdown(void)


Parameters
None
Return Values
        FM_ST_SUCCESS - if FM API interface library has been properly shut
↪down
FM_ST_UNINITIALIZED - interface library was not in initialized state.
```

6.4. Connecting to the Running Fabric Manager Instance

To connect to a running instance of FM, the instance is started as part of system service or manually by the system administrator. This connection will be used by the APIs to exchange information to the running FM instance.

```
fmReturn_t fmConnect(fmConnectParams_t *connectParams, fmHandle_t *pFmHandle)


Parameters
connectParams
Valid IP address for the remote host engine to connect to. If ipAddress
is specified as x.x.x.x it will attempt to connect to the default port
specified by FM_CMD_PORT_NUMBER.If ipAddress is specified as x.x.x.x:yyyy
it will attempt to connect to the port specified by yyyy. To connect to
an FM instance that was started with unix domain socket fill the socket
path in addressInfo member and set addressIsUnixSocket flag.
pfmHandle
        Fabric Manager API interface abstracted handle for subsequent API
↪calls
Return Values
         FM_ST_SUCCESS - successfully connected to the FM instance
FM_ST_CONNECTION_NOT_VALID - if the FM instance could not be reached
FM_ST_UNINITIALIZED - FM interface library has not been initialized
FM_ST_BADPARAM - pFmHandle is NULL or IP Address/format is invalid
FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

6.5. Disconnecting from the Fabric Manager Instance

To disconnect from an FM instance, run the following command.

```
fmReturn_t fmDisconnect(fmHandle_t pFmHandle)


Parameters
pfmHandle
        Handle that came from fmConnect
Return Values
         FM_ST_SUCCESS - successfully disconnected from the FM instance
FM_ST_UNINITIALIZED - FM interface library has not been initialized
FM_ST_BADPARAM - if pFmHandle is not a valid handle
FM_ST_GENERIC_ERROR - an unspecified internal error occurred
```

6.6. Getting a List of Supported Partitions

To query the list of supported (static) GPU fabric partitions in an NVSwitch-based system, run the following command.

```
fmReturn_t fmGetSupportedFabricPartitions(fmHandle_t pFmHandle,
↪fmFabricPartitionList_t *pFmFabricPartition)


Parameters
pFmHandle
        Handle returned by fmConnect()
pFmFabricPartition
```

Here is the pointer to the `fmFabricPartitionList_t` structure. When successful, the list of supported (static) partition information will be populated in this structure.

```
   FM_ST_SUCCESS – successfully queried the list of supported partitions
   FM_ST_UNINITIALIZED - FM interface library has not been initialized.
   FM_ST_BADPARAM – Invalid input parameters
   FM_ST_GENERIC_ERROR – an unspecified internal error occurred
   FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
   FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
   FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

6.7. Activating a GPU Partition

To activate a supported GPU fabric partition in an NVSwitch-based system, run the following command.

Note

This API is supported only in Shared NVSwitch multi-tenancy mode.

```
fmReturn_t fmActivateFabricPartition((fmHandle_t pFmHandle,
↪fmFabricPartitionId_t partitionId)


Parameters
pFmHandle
        Handle returned by fmConnect()
partitionId
        The partition id to be activated.


Return Values
   FM_ST_SUCCESS – successfully queried the list of supported partitions
   FM_ST_UNINITIALIZED - FM interface library has not been initialized.
   FM_ST_BADPARAM – Invalid input parameters or unsupported partition id
```

(continues on next page)

(continued from previous page)

```
   FM_ST_GENERIC_ERROR – an unspecified internal error occurred
   FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
   FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
   FM_ST_IN_USE - specified partition is already active or the GPUs are in use
↪by other partitions.
```

6.8. Activating a GPU Partition with Virtual Functions

In the vGPU Virtualization Mode, to activate an available GPU fabric partition with vGPU Virtual Functions (VFs), run the following command.

```
fmReturn_t fmActivateFabricPartitionWithVFs((fmHandle_t pFmHandle,
↪fmFabricPartitionId_t partitionId, fmPciDevice_t *vfList, unsigned int
↪numVfs)

Parameters:
pFmHandle
                Handle returned by fmConnect()
partitionId
                The partition id to be activated.
*vfList
                List of VFs associated with physical GPUs in the partition.
↪The ordering of VFs passed to this call is significant, especially for
↪migration/suspend/resume compatibility, the same ordering should be used
↪each time the partition is activated.
        numVfs
                Number of VFs

Return Values:
   FM_ST_SUCCESS – successfully queried the list of supported partitions
   FM_ST_UNINITIALIZED - FM interface library has not been initialized.
   FM_ST_BADPARAM – Invalid input parameters or unsupported partition id
   FM_ST_GENERIC_ERROR – an unspecified internal error occurred
   FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
   FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
   FM_ST_IN_USE - specified partition is already active or the GPUs are in use
↪by other partitions.
```

Note

Here is some important information: Before you start a vGPU VM, this API must be called, even if there is only one vGPU partition. A multi-vGPU partition activation will fail if MIG mode is enabled on the corresponding GPUs.

6.9. Deactivating a GPU Partition

To deactivate a previously activated GPU fabric partition in an NVSwitch-based system when FM is running in Shared NVSwitch or vGPU multi-tenancy mode, run the following command.

```
fmReturn_t fmDeactivateFabricPartition((fmHandle_t pFmHandle,
↪fmFabricPartitionId_t partitionId)


Parameters
pFmHandle
        Handle returned by fmConnect()
partitionId
        The partition id to be deactivated.


Return Values
   FM_ST_SUCCESS – successfully queried the list of supported partitions
   FM_ST_UNINITIALIZED - FM interface library has not been initialized.
   FM_ST_BADPARAM – Invalid input parameters or unsupported partition id
   FM_ST_GENERIC_ERROR – an unspecified internal error occurred
   FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
   FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
   FM_ST_UNINITIALIZED - specified partition is not activated
```

6.10. Setting an Activated Partition List After Restarting Fabric Manager

To send a list of currently activated fabric partitions to FM after it has been restarted, run the following command.

Note

If there are no active partitions when FM is restarted, this call must be made with the number of partitions as zero.

```
fmReturn_t fmSetActivatedFabricPartitions(fmHandle_t pFmHandle,
↪fmActivatedFabricPartitionList_t *pFmActivatedPartitionList)


Parameters
pFmHandle
        Handle returned by fmConnect()
pFmActivatedPartitionList
        List of currently activated fabric partitions.


Return Values
```

(continues on next page)

(continued from previous page)

```
        FM_ST_SUCCESS – FM state is updated with active partition information
        FM_ST_UNINITIALIZED - FM interface library has not been initialized.
        FM_ST_BADPARAM – Invalid input parameters
        FM_ST_GENERIC_ERROR – an unspecified internal error occurred
        FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
```

6.11. Getting a List of Devices with Failed NVLinks

To query all GPUs and NVSwitches with failed NVLinks as part of FM initialization, run the following command.

Note

This API is not supported when FM is running in Shared NVSwitch or vGPU multi-tenancy resiliency restart (`--restart`) modes.

```
fmReturn_t fmGetNvlinkFailedDevices(fmHandle_t pFmHandle,
↪fmNvlinkFailedDevices_t *pFmNvlinkFailedDevices)


Parameters
pFmHandle
        Handle returned by fmConnect()
pFmNvlinkFailedDevices
        List of GPU or NVSwitch devices that have failed NVLinks.


Return Values
        FM_ST_SUCCESS – successfully queried list of devices with failed
↪NVLinks
        FM_ST_UNINITIALIZED - FM interface library has not been initialized.
        FM_ST_BADPARAM – Invalid input parameters
        FM_ST_GENERIC_ERROR – an unspecified internal error occurred
        FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
        FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
        FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

Note

On DGX H100 ,NVIDIA HGX H100 and later systems, NVLinks are trained at GPU and NVSwitch hardware level using ALI feature and without FM coordination. On these systems, FM will always return `FM_ST_SUCCESS` with an empty list for this API.

6.12. Getting a List of Unsupported Partitions

To query all the unsupported fabric partitions when FM is running in Shared NVSwitch or vGPU multi-tenancy modes, run the following command.

```
fmReturn_tfmGetUnsupportedFabricPartitions(fmHandle_t pFmHandle,
fmUnsupportedFabricPartitionList_t *pFmUnupportedFabricPartition)
Parameters
pFmHandle
         Handle returned by fmConnect()
pFmUnupportedFabricPartition
        List of unsupported fabric partitions on the system.
Return Values
        FM_ST_SUCCESS – successfully queried list of devices with failed
↪NVLinks
        FM_ST_UNINITIALIZED - FM interface library has not been initialized.
        FM_ST_BADPARAM – Invalid input parameters
        FM_ST_GENERIC_ERROR – an unspecified internal error occurred
        FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
        FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
        FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

Note

On DGX H100, NVIDIA HGX H100 and later systems, this API will always return FM_ST_SUCCESS with an empty list of unsupported partition.

Chapter 7. Full Passthrough Virtualization Model

The first supported virtualization model for NVSwitch-based systems is passthrough device assignment for GPUs and NVSwitch memory fabrics (switches). VMs with 16, eight, four, two, and one GPUs are supported with predefined subsets of GPUs and NVSwitches for each VM size.

A subset of GPUs and NVSwitches is referred to as a system partition. Non-overlapping partitions can be mixed and matched, which allows you to simultaneously support, for example, an eight-GPU VM, a four-GPU VM, and two-GPU VMs on an NVSwitch-based system with two GPU baseboards. VMs with 16 and eight GPUs have no loss in bandwidth while in smaller VMs, there is some bandwidth tradeoff for isolation by using dedicated switches.

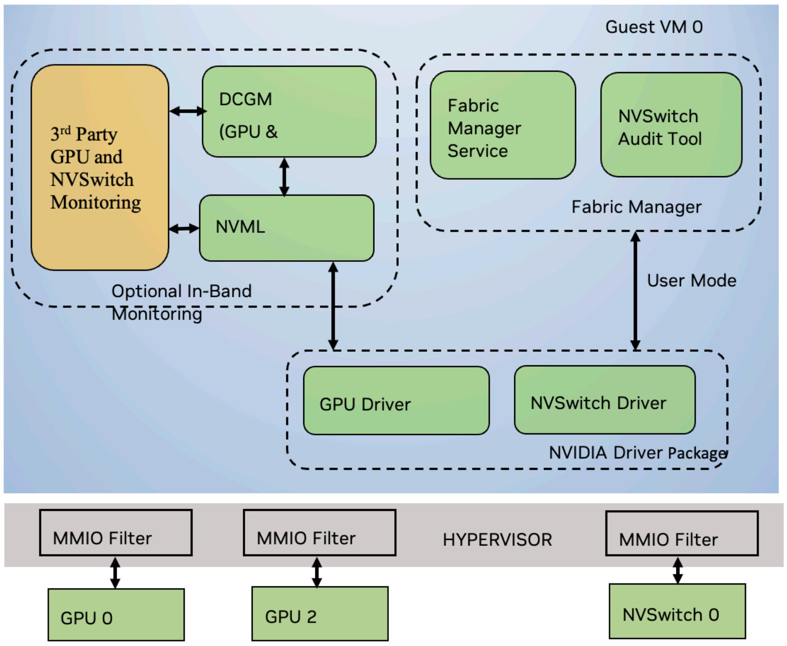

Figure 7.1: Software Stack in a Two-GPU Virtual Machine (A Full Passthrough Model)

7.1. Supported Virtual Machine Configurations

The tables in this section provide information about virtual machine configurations.

Table 7.1 DGX-2 and NVIDIA HGX-2 Systems Device Assignment

VM GPUs	NVSwitches	NVLinks Per GPU	NVLinks Per NVSwitch	Constraints
16	12	6 of 6	16 of 18	None
8	6	6 of 6	8 of 18	One set of eight GPUs from each GPU Baseboard
4	3	3 of 6	4 of 18	Two sets of four GPUs from each GPU Baseboard
2	1	1 of 6	2 of 18	Four sets of two GPUs from each GPU Baseboard
1	0	0 of 6	0 of 18	None

Table 7.2 DGX A100 and NVIDIA HGX A100 Systems Device Assignment

VM GPUs	NVSwitches	NVLinks Per GPU	NVLinks Per NVSwitch	Constraints
16	12	12 of 12	32 of 36	None
8	6	12 of 12	16 of 36	One set of eight GPUs from each GPU Baseboard.
4	3	6 of 12	6 of 36	Two sets of four GPUs from each GPU Baseboard.
2	1	2 of 12	4 of 36	Four sets of two GPUs from each GPU Baseboard.
1	0	0 of 12	0 of 36	None

Table 7.3 DGX H100 and NVIDIA HGX H100 Systems Device Assignment

VM GPUs	NVSwitches	NVLinks Per GPU	NVLinks Per NVSwitch	Constraints
8	4	18 of 18	32 of 64 for two NVSwitches. 40 of 64 for other two NVSwitches.	None
1	0	0 of 18	0 of 64	Need to disable GPU NVLinks.

Note

For DGX H200, NVIDIA HGX H200, and NVIDIA HGX H20 systems, the same H100 VM configuration, NVLink topology, and NVSwitch assignment apply. For NVIDIA HGX H800 systems, the same assignment applies, with a total of eight GPU NVLinks for 8x GPU VMs.

Table 7.4 DGX B200 and NVIDIA HGX B200 Systems Device Assignment

VM GPUs	NVSwitches	NVLinks Per GPU	NVLinks Per NVSwitch	Constraints
8	2 (Indirectly) Actual devices passed through is CX7 bridge devices	18 of 18	72 of 72	For DGX B200 and HGX B200, instead of the NVSwitches being directly passed to the guest VM, the CX7 bridge devices are passed. Refer to the following sections for more information.
1	0	0 of 18	0 of 64	Need to disable GPU NVLinks.

7.2. Virtual Machines with 16 GPUs

Here are the requirements for VMs with 16 GPUs:

- The available GPUs and NVSwitches are assigned to the guest VM.
- There are no disabled NVLink interconnects on the NVSwitches or on the GPUs.
- To support 16 GPU partitions, the system must be populated with two GPU baseboards.

7.3. Virtual Machines with Eight GPUS

Here are the requirements for VMs with eight GPUs:

- Each VM has eight GPUs, and the NVSwitches on the same base board (six for DGX A100 and NVIDIA HGX A100 and four for DGX H100 and NVIDIA HGX H100) must be assigned to the guest VM.
- Each GPU has all the NVLink interconnects enabled.
- If the system has two GPU baseboards, two system partitions will be available where eight GPU VMs can be created.

Otherwise only one partition will be available.

- All NVLink connections between the GPU baseboards are disabled.

7.4. Virtual Machines with Four GPUS

Here are the requirements for VMs with four GPUs:

- If this configuration is supported, each VM gets four GPUs and three switches.
- As specified in Table 7.3, only a subset of NVLink interconnects per GPU are enabled.
- If the system is populated with two GPU baseboards, four partitions are available on the system.
- For single baseboard systems, two partitions are available.
- All NVLink connections between GPU baseboards are disabled.

7.5. Virtual Machines with Two GPUs

Here are the requirements for VMs with two GPUs:

- If this configuration is supported, each VM gets two GPUs and one NVSwitch.
- Also, a subset of GPU NVLink interconnects per GPU are enabled.
- If the system is populated with two GPU baseboards, eight partitions are available on the system.
- For single baseboard systems, four partitions are available.
- All NVLink connections between GPU baseboards are disabled.

7.6. Virtual Machine with One GPU

Here are the requirements for VMs with one GPU:

- Each VM has one GPU and no switches.
- If the system is populated with two GPU baseboards, 16 partitions are available on the system.
- For single baseboard systems, eight partitions are available.
- All NVLink connections between GPU baseboards are disabled.

7.7. Other Requirements

Here are some other requirements:

- The hypervisor needs to maintain the partition configuration, including which NVLink connections to block on each GPU and switch for each partition.
- The hypervisor needs to implement MMIO filtering for NVSwitch.
- The hypervisor needs to finely control IOMMU mappings that were configured for GPUs and switches.
- Guest VMs with more than one GPU need to run the core NVSwitch software stack, which includes the NVIDIA Driver and FM to configure switches and NVLink connections.

7.8. Hypervisor Sequences

The hypervisor completes the following steps to launch, shutdown, and reboot guest VMs.

1. Start the guest VM.
    1. Select an unused partition of GPUs and switches.
    2. Reset the GPUs and switches in the partition.
    3. Block the disabled NVLink connections on each GPU by performing the specified MMIO configuration.
    4. Block the disabled NVLink connections on each switch by configuring the MMIO intercept.
    5. Avoid configuring IOMMU mappings between GPUs and switches.
    6. Ensure that switches cannot be accessed by any other PCIe device that the guest VM controls.
        - This way, the switches cannot bypass the MMIO restrictions that are implemented for the CPU.
        - GPUs do not need to be accessible by any other GPUs or switches.
        - GPUs need to be accessible by third-party devices to support NVIDIA GPUDirect™ RDMA.
    7. Avoid additional GPU resets, start the guest VM.
2. Shut down the guest VM and reset the GPUs and switches that belong to the partition.
3. Reboot the guest VM.
4. Repeat steps 1a to 1f, but this time, the partition has already been selected.

7.9. Additional Steps for NVIDIA HGX B200 Systems

In NVIDIA HGX B200 systems, NVSwitches do not appear as PCIe devces. They are behind a CX7 bridge device and the corresponding four CX7 device physical functions must be passed to the guest VM. Through this CX7 Bridge device, the FM and NVLSM service will configure the underlying NVSwitches, NVLinks, and GPUs.

To identify the desired CX7 bridge device with other traditional CX7 NIC devices, use one of the following mechanisms:

- **Using a fixed PCIe BDF assignment**: The CX7 bridge device is part of the GPU baseboard.

If the GPU baseboard PCIe resources are statically assigned, the four PF functions will be same all the time.

- **Query using VPD information**: The CX7 bridge device will have different VPD information as compared to other traditional CX7 device.

Query the VPD information using the `lspci -vvs` or `vpddecode` command and identify the four PF functions you want.

7.10. Monitoring Errors

The NVSwitch, GPU and NVLink errors are visible to guest VMs. Use one of the following methods to allow the hypervisor to monitor the same items,:

- In-band monitoring

Run NVIDIA Data Center GPU Manager (DCGM) on the guest VM or use the NVIDIA Management Library (NVML) APIs for GPU-specific monitoring.

- Out-of-Band monitoring

Use the GPU and NVSwitch SMBus Post Box Interface (SMBPBI)-based OOB commands.

7.11. Limitations

Here are the limitations:

- NVSwitch errors are visible to the guest VMs.
- Windows is only supported for single GPU VMs.

Chapter 8. Shared NVSwitch Virtualization Model

The shared NVSwitch virtualization model additionally extends the GPU Passthrough model by managing the switches from a Service VM that permanently runs. The GPUs are made accessible to the Service VM for link training and are reassigned to the guest VMs.

Sharing switches among the guest VMs allows FM to enable more NVLink connections for two- and four-GPU VMs that observe reduced bandwidth in GPU Passthrough model.

8.1. Software Stack

Figure 8.1 shows the software stack that is required for NVSwitch management and runs in a service VM.

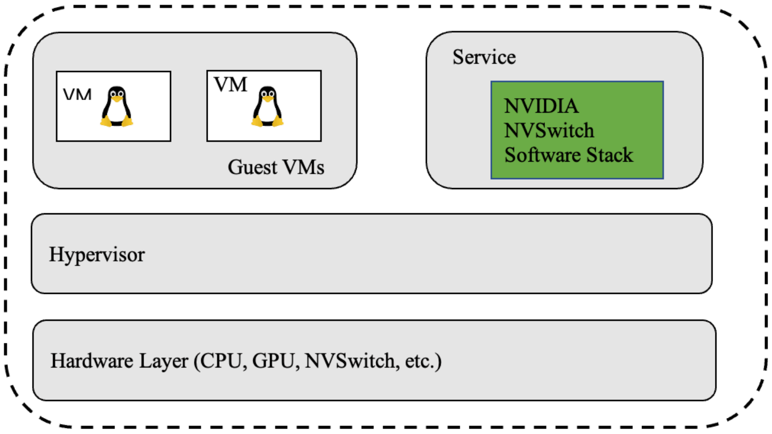
VM
VM
Guest VMs
Service
NVIDIA NVSwitch Software Stack
Hypervisor
Hardware Layer (CPU, GPU, NVSwitch, etc.)

Figure 8.1: Shared NVSwitch Software Stack

NVSwitch units are always assigned as a PCIe passthrough device to the service VM. GPUs are hot-plugged and hot-unplugged on-demand (as PCI passthrough) to the service VM.

At a high level, the service VM provides the following features:

- An interface to query the available GPU VM partitions (groupings) and corresponding GPU information.
- An interface to activate GPU VM partitions, which involves the following:
- Training NVSwitch to NVSwitch NVLink interconnects (if required).

- Training the corresponding GPU NVLink interfaces (if applicable).
- Programming NVSwitch to deny access to GPUs not assigned to the partition.
- An interface to deactivate GPU VM partitions, which involves the following:
- As applicable, untrain (power down) the NVSwitch to NVSwitch NVLink interconnects.
- As applicable, untrain (power down) the corresponding GPU NVLink interfaces.
- Disable the corresponding NVSwitch routing and GPU access.
- Report NVSwitch errors through in-band and out-of-band mechanisms.

8.2. Guest VM to Service VM Interaction

For NVIDIA HGX-2, NVIDIA HGX A100, and NVIDIA HGX A800 server systems, the GPU configurations that are required to enable NVLink communication are established as part of the initial partition activation process, which occurs before transferring GPU control to the guest VM. Consequently, there is no need for the guest VM to initiate communication with the service VM while workloads are running.

However, on NVIDIA HGX H100, NVIDIA HGX H800, and later systems, a different approach is required. In these systems, the GPUs are not assigned to the service VM during partition activation. As a result, the configurations for GPU NVLink communication must be passed to the guest VM. Additionally, the newly introduced NVLink Sharp feature in the H100 and H800 generations necessitates dynamic adjustments to the NVSwitch configuration based on the workload requirements of the guest VM.

To facilitate these functionalities on NVIDIA HGX H100, NVIDIA HGX H800, and later systems, GPUs in the guest VM communicate over NVLink by transmitting specialized packets to the FM that runs on the service VM. To simplify integration efforts, communicating these requests over NVLink is the optimal solution because it can be completely managed in NVIDIA's software and firmware, without requiring custom integrations for the customer. This communication protocol also is version agnostic, which allows compatibility between different versions of NVIDIA Drivers on the guest and service VMs.

8.3. Preparing the Service Virtual Machine

This section provides information about preparing the service VM.

8.3.1. The OS Image

Internally, NVIDIA uses an Ubuntu distro as the Service VM OS image. However, there are no known limitations with other major Linux OS distributions. Refer to *OS Environment* for more information.

8.3.2. Resource Requirements

Refer to the corresponding OS distributions minimum resource guidelines for more information about the exact service VM resource requirements. In addition to the specified minimum guidelines, internally, NVIDIA uses the following hardware resources for the service VM:

Note

The resource requirements for the service VM might vary if it is used for additional functionalities, such as conducting a GPU health check. The specific memory and vCPU demands might also fluctuate depending on the Linux distribution you selected and the OS features you enabled. We recommend that you make necessary adjustments to the allocated memory and vCPU resources accordingly. DGX B200 and NVIDIA HGX B200/B100 systems that use B200 GPUs and the fourth-generation NVSwitches requires a minimum v5.17 Linux kernel.

Table 8.1 Service VM Resources

Resource	Quantity/Size
vCPU	2
System Memory	4 GB

8.3.3. NVIDIA Software Packages

The service VM image must have the following NVIDIA software packages installed:

- NVIDIA Data Center GPU Driver (version 450.xx and later for NVIDIA HGX-2 and NVIDIA HGX A100 systems).
- For NVIDIA HGX H100 systems, version 525.xx and later is required.
- For NVIDIA HGX B200 systems, OFED or MOFED package is required.
- NVIDIA Fabric Manager Package (same version as the Driver package).

8.3.4. Fabric Manager Config File Modifications

To support the Shared NVSwitch mode, start the FM service in Shared NVSwitch mode by setting the FM config item `FABRIC_MODE=1`.

Note

NVSwitches and GPUs on NVIDIA HGX-2 and NVIDIA HGX A100 systems must bind to nvidia.ko before FM service starts. If the GPUs and NVSwitches are not plugged into the service VM as part of OS boot, start the FM service manually or the process directly by running the appropriate command line options after the NVSwitches and GPUs are bound to `nvidia.ko`.

In Shared NVSwitch mode, FM supports a resiliency feature, which allows the non-stop forwarding of NVLink traffic between GPUs on active guest VMs after FM gracefully or non-gracefully exits in the service VM. To support this feature, FM uses `/tmp/fabricmanager.state` to save certain metadata information. To use a different location/file to store this metadata information, modify the `STATE_FILE_NAME` FM config file item with the path and file name.

FM uses TCP I/P loopback (127.0.0.1)-based socket interface for communication. To use UNIX domain sockets instead, modify the FM `FM_CMD_UNIX_SOCKET_PATH` and `UNIX_SOCKET_PATH` config file options with the UNIX domain socket names.

In addition, to deploy NVIDIA HGX B200, change the FM and NVLSM communication default UNIX domain socket interface config item `FM_SM_IPC_INTERFACE` config to the path you want. Also, the same path information should be used in the `plugin_options grpc_mgr --grpc_server_address` NVLSM GRPC plug-in configuration item.

8.3.5. Fabric Manager Multicast (NVLink Sharp) Resource allocation

The H100 and later generations of DGX and HGX NVSwitch systems support NVLink multicast traffic and reduction operations. Refer to the CUDA *Multicast Object Management* APIs for more information about allocating multicast objects, assigning GPUs to multicast objects, and related operations.

NVSwitches have a finite number of multicast resources (also called as Multicast Slots). In H100-based DGX and HGX systems, these slots are independent of NVLink partitions, with each partition receiving full dedicated set of supported multicast slots. However, in B200-generation systems, multicast slots are shared across all NVLink partitions. To manage this, each partition is assigned a fixed upper limit of multicast slots during partition creation. The allocation policy is static and determined by the number of GPUs in the partition as follows:

- Eight-GPU partition: Receives all supported multicast slots.
- Four-GPU partition: Receives up to 50% of the supported multicast slots.
- Two-GPU partition: Receives up to 25% of the supported multicast slots.
- One-GPU partition: Receives zero slots, as a single-GPU partition cannot use multicast or reduction operations.

8.3.6. Other NVIDIA Software Packages

In the Shared NVSwitch mode, only the FM service should open and interact with GPUs while activating or deactivating the partition. The GPU health check applications must also be started after you activate the partition and must be closed before you unbind the GPUs from `nvidia.ko`.

8.4. Fabric Manager Shared Library and APIs

Refer to *Fabric Manager SDK* for the list of the APIs that manage a shared NVSwitch partition life cycle.

8.4.1. Sample Code

The following code snippet shows you how to query supported partitions, activate, or deactivate partitions, and so on by using the FM APIs mentioned in “Fabric Manager SDK” on page .

```
#include <iostream>
#include <string.h>


#include "nv_fm_agent.h"


```

(continues on next page)

(continued from previous page)

```
int main(int argc, char **argv)
{
    fmReturn_t fmReturn;
    fmHandle_t fmHandle = NULL;
    char hostIpAddress[16] = {0};
    unsigned int operation = 0;
    fmFabricPartitionId_t partitionId = 0;
    fmFabricPartitionList_t partitionList = {0};


    std::cout << "Select Shared Fabric Partition Operation:\n";
    std::cout << "0 - List Supported Partition\n";
    std::cout << "1 – Activate a Partition\n";
    std::cout << "2 – Deactivate a Partition\n";
    std::cin >> operation;
    if ( operation > 2 ) {
        std::cout << "Invalid input.\n" << std::endl;
        return FM_ST_BADPARAM;
    }
    std::cout << std::endl;


    if ( operation > 0 ) {
        std::cout << "Input Shared Fabric Partition ID: \n";
        std::cin >> partitionId;


        if ( partitionId >= FM_MAX_FABRIC_PARTITIONS ) {
            std::cout << "Invalid partition ID." << std::endl;
            return FM_ST_BADPARAM;
        }
    }
    std::cout << std::endl;


    std::cout << "Please input an IP address to connect to. (Localhost = 127.
↪0.0.1) \n";
    std::string buffer;
    std::cin >> buffer;
    if (buffer.length() > sizeof(hostIpAddress) – 1){
        std::cout << "Invalid IP address.\n" << std::endl;
        return FM_ST_BADPARAM;
    } else {
        buffer += '\0';
        strncpy(hostIpAddress, buffer.c_str(), 15);
    }


    /* Initialize Fabric Manager API interface library */
    fmReturn = fmLibInit();
    if (FM_ST_SUCCESS != fmReturn) {
        std::cout << "Failed to initialize Fabric Manager API interface
```

(continues on next page)

(continued from previous page)

```
↪library." << std::endl;
        return fmReturn;
    }


    /* Connect to Fabric Manager instance */
    fmConnectParams_t connectParams;
    strncpy(connectParams.addressInfo, hostIpAddress, sizeof(hostIpAddress));
    connectParams.timeoutMs = 1000; // in milliseconds
    connectParams.version = fmConnectParams_version;
    connectParams.addressIsUnixSocket = 0;
    fmReturn = fmConnect(&connectParams, &fmHandle);
    if (fmReturn != FM_ST_SUCCESS){
        std::cout << "Failed to connect to Fabric Manager instance." <<
↪std::endl;
        return fmReturn;
    }


    if ( operation == 0 ) {
        /* List supported partitions */
        partitionList.version = fmFabricPartitionList_version;
        fmReturn = fmGetSupportedFabricPartitions(fmHandle, &partitionList);
        if (fmReturn != FM_ST_SUCCESS) {
            std::cout << "Failed to get partition list. fmReturn: " <<
↪fmReturn << std::endl;
        } else {
            /* Only printing number of partitions for brevity */
            std::cout << "Total number of partitions supported: " <<
↪partitionList.numPartitions  << std::endl;
        }


    } else if ( operation == 1 ) {
        /* Activate a partition */
        fmReturn = fmActivateFabricPartition(fmHandle, partitionId);
        if (fmReturn != FM_ST_SUCCESS) {
            std::cout << "Failed to activate partition. fmReturn: " <<
↪fmReturn << std::endl;
        }


    } else if ( operation == 2 ) {
        /* Deactivate a partition */
        fmReturn = fmDeactivateFabricPartition(fmHandle, partitionId);
        if (fmReturn != FM_ST_SUCCESS) {
            std::cout << "Failed to deactivate partition. fmReturn: " <<
↪fmReturn << std::endl;
        }


    } else {
```

(continues on next page)

(continued from previous page)

```
        std::cout << "Unknown operation." << std::endl;
    }


    /* Clean up */
    fmDisconnect(fmHandle);
    fmLibShutdown();
    return fmReturn;
}


# Make file for the above sample assuming the source is saved into sampleCode.
↪cpp
# Note: Change the default include paths (/usr/include & /usr/lib) based on FM
↪API header files location.


IDIR := /usr/include
CXXFLAGS = -I $(IDIR)


LDIR := /usr/lib
LDFLAGS= -L$(LDIR) -lnvfm


sampleCode: sampleCode.o
        $(CXX) -o $@ $< $(CXXFLAGS) $(LDFLAGS)


clean:
        -@rm -f sameplCode.o
        -@rm -f sampleCode
```

8.5. Fabric Manager Resiliency

Refer to *Resiliency* for more information about FM resiliency in the Shared Virtualization mode.

8.6. Service Virtual Machine Life Cycle Management

This section provides information about managing the service VM's life cycle.

8.6.1. GPU Partitions

Refer to *GPU Partitions* for the default and supported partitions in the shared NVSwitch virtualization mode.

8.6.2. Building GPUs to Partition Mapping

The FM instance that runs on the service VM, and the hypervisor must use a common numbering scheme (GPU Physical ID) to uniquely identify each GPU. In this release, the Physical ID numbering is the same as in the *Baseboard Pinout* design collateral.

The hypervisor should maintain a list of GPU Physical IDs and corresponding PCI BDF mapping information to identify each GPUs in the hypervisor. This information is required to identify GPUs that belong to a partition and hot attach the GPUs to a service VM as part of guest VM activation.

8.6.3. Booting the Service Virtual Machine

As part of service VM boot, the hypervisor must do the following:

1. Assign/plug the available NVSwitches as PCI passthrough devices to the service VM without MMIO filtering.
2. On NVIDIA HGX B200 systems, assign the four PF functions of the CX7 bridge devices as PCI passthrough instead of NVSwitches.

To locate the CX7 bridge devices you want, refer to *Additional Steps for NVIDIA HGX B200 Systems*.

3. On NVIDIA HGX-2 and NVIDIA HGX A100 systems, assign/plug the available GPUs as PCI passthrough devices to the service VM without MMIO filtering.
4. Start FM and wait for it to fully initialize the GPUs and switches.

The FM APIs will return `FM_ST_NOT_CONFIGURED` until the fabric is initialized and ready.

5. Query the list of currently supported VM partitions and build the available guest VM combinations.
6. Deassign/unplug the GPUs from the service VM for the NVIDIA HGX-2 and NVIDIA HGX A100 systems.

8.6.4. Restarting the Service Virtual Machine

The NVSwitch kernel software stack is loaded and initializes the NVSwitches and GPUs as part of the service VM booting, so restarting the service VM will affect currently activated GPU partitions. The hypervisor must follow the same procedure and steps as described in “Booting the Service Virtual Machine” on page .

8.6.5. Shutting Down the Service Virtual Machine

Currently activated VM partitions will not be affected as part of service VM shutdown because the NVSwitch configuration is preserved. However, if the hypervisor or PCIe pass through driver issues a Secondary Bus Reset (SBR) to the NVSwitch devices as part of service VM shutdown, the activated partitions will be affected. Since FM is not running, and the driver is unloaded, there will be no active error monitoring and corresponding remediation.

Note

Do not leave the guest VMs in this state for a longer period.

8.7. Guest Virtual Machine Life Cycle Management

This section provides information about managing the life cycle of the guest VM.

8.7.1. Guest Virtual Machine NVIDIA Driver Package

To use GPU NVLink interconnects, ensure that one of the following the driver packages for NVIDIA Data Center GPUs is installed on the guest VM:

- Version 450.xx and later for NVIDIA HGX-2 and NVIDIA HGX A100 systems.
- Version 525.xx and later for NVIDIA HGX H100 systems.

8.7.2. Starting a Guest Virtual Machine

To start a guest VM, the hypervisor completes one of the following tasks:

Note

The sequences will be different depending on the NVSwitch generation used in the system. The key difference is whether the GPU needs to be attached to service VM and bound to `nvidia.ko`.

- On NVIDIA HGX-2 and NVIDIA HGX A100 systems:

1. Select one of the supported GPU partitions based on the guest VM GPU demand.
2. Identify the corresponding GPUs by using the GPU Physical-ID-to-PCI-BDF mapping.
3. Reset (SBR) the selected GPUs.
4. Hot plug the selected GPUs to the service VM.
5. Ensure that the GPUs are bound to `nvidia.ko`.
6. Request FM to activate the requested GPU partition using the `fmActivateFabricPartition()` API.
7. Unbind the GPUs from `nvidia.ko`.
8. Hot unplug the GPUs from service VM (if needed).
9. Start the guest VM without resetting the GPUs.

Note

If the GPUs get a PCIe reset as part of guest VM launch, the GPU NVLinks will be in an `InActive` state on the guest VM. Also, starting the guest VM without a GPU reset might require a modification in your hypervisor VM launch sequence path.

- On NVIDIA HGX H100 and later systems:

1. Select one of the supported GPU partitions based on the guest VM GPU demand.
2. Identify the corresponding GPUs by using the GPU Physical ID-to-PCI-BDF mapping.

3. Request FM to activate the requested GPU partition using the `fmActivateFabricPartition()` API.
4. Start the guest VM.

8.7.3. Shutting Down a Guest Virtual Machine

To shut down a guest VM, the hypervisor completes one of the following tasks:

Note

The sequences will be different depending on the NVSwitch generation used in the system.

- On NVIDIA HGX-2 and NVIDIA HGX A100 systems:

1. To avoid NVSwitch-related NVLink errors and GPU resets, shut down the guest VM.
2. Use the `fmDeactivateFabricPartition()` API and request FM to deactivate the GPU partition.
3. Reset the GPUs after the partition has been deactivated.

- On NVIDIA HGX H100 and later systems:

1. Shut down the guest VM.
2. Use the `fmDeactivateFabricPartition()` API and request FM to deactivate the GPU partition.
3. If the guest VM shutdown process does not complete an explicit GPU reset, reset the GPUs after the partition has been deactivated.

8.7.4. Rebooting a Guest Virtual Machine

When rebooting a guest VM, if the GPUs get an SBR as part of the VM reboot, the hypervisor must complete the steps in “Starting a Guest Virtual Machine” on page and “Shutting Down a Guest Virtual Machine” on page .

8.7.5. Verifying GPU Routing

The `nvswitch-audit` command-line utility, which was installed as part of the FM package, can output the number of NVLinks that the NVSwitches are programmed to handle for each GPU. The tool reconstructs this information by reading and decoding the internal NVSwitch hardware routing table information. We recommend that you periodically verify the GPU reachability matrix on each VM partition activation and deactivation cycle by running this tool in the service VM.

The following options are supported by `nvswitch-audit` command-line utility.

```
$ ./nvswitch-audit -h
NVIDIA NVSwitch audit tool
Reads NVSwitch hardware tables and outputs the current number of
NVlink connections between each pair of GPUs
```

(continues on next page)

(continued from previous page)

```
Usage: nvswitch-audit [options]


Options include:
[-h | --help]: Displays help information
[-v | --verbose]: Verbose output including all Request and Response table
↪entries
[-f | --full-matrix]: Display All possible GPUs including those with no
↪connecting paths
[-c | --csv]: Output the GPU Reachability Matrix as Comma Separated Values
[-s | --src]: Source GPU for displaying number of unidirectional connections
[-d | --dst]: Destination GPU for displaying number of unidirectional
↪connections
```

The following example output shows the maximum GPU NVLink connectivity when an eight-GPU VM partition on an NVIDIA HGX A100 is activated.

```
$ ./nvswitch-audit
GPU Reachability Matrix
GPU  1  2  3  4  5  6  7  8
  1  X 12 12 12 12 12 12 12
  2 12  X 12 12 12 12 12 12
  3 12 12  X 12 12 12 12 12
  4 12 12 12  X 12 12 12 12
  5 12 12 12 12  X 12 12 12
  6 12 12 12 12 12  X 12 12
  7 12 12 12 12 12 12  X 12
  8 12 12 12 12 12 12  X 12
```

Note

The *nvswitch-audit* tool is not supported on DGX B200, NVIDIA HGX B200, and corresponding B100 system variants.

8.8. Error Handling

Refer to *Error Handling* for information about FM initialization, partition, and hardware specific errors and their handling.

8.8.1. Guest Virtual Machine GPU Errors

When the guest VM is active, all GPU runtime errors will be logged in the guest VM syslog as Xid errors. On NVIDIA HGX-2 and NVIDIA HGX A100 systems, the GPU NVLink errors that require retraining are not supported in this environment, and to recover, must complete the steps in “Starting a Guest Virtual Machine” on page and “Shutting Down a Guest Virtual Machine” on page .

8.8.2. Handling a Service Virtual Machine Crash

When a service VM experiences a kernel crash, the remaining activated guest VMs will continue as expected, but the VM partition activation and deactivation life cycle will be affected. To recover from this state, you must restart or boot the service VM.

8.9. Interoperability With a Multi-Instance GPU

The Shared NVSwitch virtualization model can interoperate with the MIG feature that is supported on A100, H100 and later GPUs. However, to expose a shared NVSwitch partition with MIG-enabled GPUs to guest VMs, maintain one of the options in this section. NVLinks are not trained on H100 and later GPUs when MIG is enabled, so these options do not apply to corresponding DGX and HGX systems.

8.9.1. Initializing Service Virtual Machine

When FM initializes on the service VM, without the `--restart` option for resiliency flow, the MIG mode must be disabled for the available GPUs. If any GPUs have MIG mode enabled, the FM service initialization will be aborted.

8.9.2. Activating the Guest Virtual Machine

The FM-shared NVSwitch partition activation and deactivation sequence can handle MIG-enabled GPUs. However, GPUs in which MIG was enabled **before** the partition was activated, for example by the VM before the VM reboot, will not have NVLinks trained as part of the partition activation. The activation/deactivation flow works as expected.

Chapter 9. vGPU Virtualization Model

The vGPU virtualization model supports VF passthrough by enabling SR-IOV in the supported GPUs and assigning a VF, or set of VFs, to the VM.

- GPU NVLinks are assigned to only one VF at a time.
- NVLink P2P between GPUs that belong to different VMs or partitions is not supported.

Refer to the Virtual GPU Software User Guide for more information about the supported vGPU functionality, features, and configurations.

9.1. Software Stack

In the vGPU virtualization model:

- The NVSwitch Software Stack (FM and Switch Driver) runs in the vGPU host.
- Like the bare-metal mode, the physical GPUs and NVSwitches are owned and managed by the vGPU host.
- The GPU and NVSwitch NVLinks are trained and configured as part of FM initialization.
- The switch routing table is initialized to prevent GPU-GPU communication.

Note

The vGPU-based deployment model is not supported on first generation-based NVSwitch systems such as DGX-2 and NVIDIA HGX-2.

9.2. Preparing the vGPU Host

This section provides information about how to prepare the vGPU host.

9.2.1. OS Image

Refer to the Virtual GPU Software User Guide for the list of supported OSs, hypervisors, and for information about installing and configuring the vGPU host driver software.

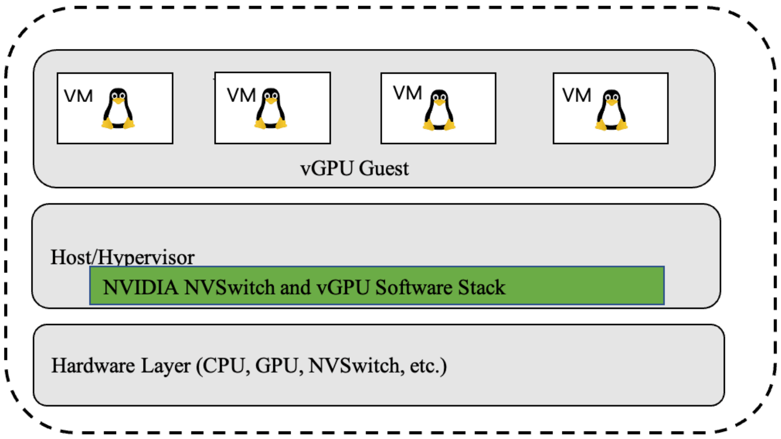
VM
VM
VM
VM
vGPU Guest
Host/Hypervisor
NVIDIA NVSwitch and vGPU Software Stack
Hardware Layer (CPU, GPU, NVSwitch, etc.)

Figure 9.1: The vGPU Software Stack

9.2.2. NVIDIA Software Packages

In addition to the NVIDIA vGPU host driver software, the vGPU host image must have the following NVIDIA software packages installed:

- The FM package
- The FM SDK Package

Note

Both packages must be the same version as the Driver package.

9.2.3. Fabric Manager Config File Modifications

To support vGPU virtualization, start the FM service in vGPU virtualization mode by setting the `FABRIC_MODE=2` FM config item.

Note

NVSwitches must bind to nvidia.ko before the FM service starts. On DGX A100 and NVIDIA HGX A100 systems, all GPUs must also be bound to nvidia.ko before the FM service starts.

In the vGPU virtualization mode, FM supports a resiliency feature that allows the continuous forwarding of NVLink traffic between GPUs on active guest VMs after FM exits gracefully or non-gracefully on the vGPU host. To support this feature, FM uses the `/tmp/fabricmanager.state` file to save certain metadata information. To use a different location/file to store this information, modify the `STATE_FILE_NAME FM` config file item with the new path and file name.

By default, FM uses TCP I/P loopback (127.0.0.1)-based socket interface for communication. To use Unix domain sockets instead, modify the `FM_CMD_UNIX_SOCKET_PATH and UNIX_SOCKET_PATH` FM config file options with the new Unix domain socket names.

9.3. Fabric Manager-Shared Library and APIs

Refer to *Fabric Manager SDK* for a list of the APIs that are used to manage the vGPU partition life cycle.

9.4. Fabric Manager Resiliency

Refer to *Resiliency* for more information about FM resiliency in the vGPU virtualization mode.

9.5. vGPU Partitions

Refer to *GPU Partitions* for the default supported partitions for the vGPU virtualization model.

9.6. Guest Virtual Machine Life Cycle Management

Here is an overview of the guest VM life cycle:

1. The system powers on and starts.
2. The vGPU host driver loads.
3. SR-IOV is enabled.
4. FM initializes in the vGPU virtualization mode.
5. NVlinks are trained.
6. The partition is activated with the selected SR-IOV VFs.
7. The vGPU-enabled VM completes its life cycle with the VFs selected in step 2.

This life cycle can involve boot, reboot, shutdown, suspend, resume, and migrate activities.

4. The partition deactivates.

These steps are explained in greater detail in the following sections.

9.6.1. Activating the Partition and Starting the Virtual Machine

SR-IOV VFs must be enabled on the physical GPUs before you activate partitions and power on the vGPU VMs.

When starting a guest VM, the hypervisor must do the following:

1. Select an available GPU partition that contains the required number of GPUs for the guest VM and select the VFs that will be used on those GPUs.
2. Use the `fmActivateFabricPartitionWithVFs()` API and request FM to activate the GPU partition, with the set of selected VFs.

3. Start the guest VM with the selected VFs.

Note

Partition activation is always required before starting a vGPU VM, even for VMs that use only one vGPU. The ordering of VFs used during partition activation and VM assignment must remain consistent to ensure the correct suspend, resume, and migration operations.

Refer to the Installing and Configuring the NVIDIA GPU Manager for Red Hat Linux KVM for more information about SR-IOV VF enablement and assigning VFs to VMs.

9.6.2. Deactivating the Partition

Do not deactivate partitions when VMs are executing on the GPUs in the partition.

To deactivate a partition:

1. Shut down the guest VM that is currently operating in the partition.
2. Use the `fmDeactivateFabricPartition()` API and request that FM deactivate the partition.

9.6.3. Migrating Virtual Machines

VM migration is supported only between partitions with an identical number, type of GPU, and NvLink topology.

Refer to Migrating a VM Configured with vGPU for more information.

9.6.4. Verifying GPU Routing

The `nvswitch-audit` command line utility referenced in "Verifying GPU Routing" on page can also be used to verify NVSwitch routing information in the vGPU mode. We recommend that you periodically run this tool to verify the GPU reachability matrix on each VM partition's activation and deactivation cycle.

9.7. Error Handling

Refer to *Error Handling* for information about FM initialization, partition, hardware specific errors, and their handling.

9.7.1. Guest Virtual Machine GPU Errors

When the guest VM is active, GPU runtime errors will be logged in the vGPU host syslog like the Xid errors. On DGX A100 and NVIDIA HGX A100 systems, GPU NVLink errors that require retraining are not supported in this environment and must complete the guest VM shutdown and start sequence to recover.

9.8. GPU Reset

If the GPU generates a runtime error, or gets an Xid NVLink error, the system administrator can clear the corresponding error state and recover the GPU using the GPU reset operation. The operation must be initiated from the vGPU host **after** a VM that is using the GPU is shut down and the corresponding partition is deactivated. Refer to the `nvidia-smi` command-line utility documentation for more information.

9.9. Interoperability with MIG

MIG-backed vGPUs on NVIDIA A100 and NVIDIA HGX A100 cannot use NVlink. The FM's vGPU virtualization mode still can interoperate with the MIG feature to support use cases where a subset of GPUs are being used in MIG mode.

9.9.1. Enabling MIG before Starting the Fabric Manager Service

When MIG is enabled on a GPU before FM is started, FM will remove the GPU partitions from its list of available partitions that contain GPUs in MIG mode. These GPU partitions will not be available to deploy VMs. To enable partitions after disabling MIG mode on a GPU, reboot the system.

9.9.2. Enabling MIG After Starting the Fabric Manager Service

MIG functionality can be enabled on any GPU after starting the FM Service, but before a partition that contained the GPU is activated.

Note

Activating a GPU partition will return success even if the GPU is in MIG mode.

On DGX A100 and NVIDIA HGX A100 systems, activating a multi-GPU partition will fail if a GPU in the partition is in MIG mode. This process will succeed on the DGX H100 and NVIDIA HGX H100 systems.

Chapter 10. Supported High Availability Modes

FM provides several High Availability Mode (Degraded Mode) configurations that allow system administrators to set appropriate policies when there are hardware failures, such as GPU failures, NVSwitch failures, NVLink connection failures, and so on, on NVSwitch-based systems. With this feature, system administrators can keep a subset of available GPUs that can be used while waiting to replace failed GPUs, baseboards, and so on.

DGX A100, NVIDIA HGX A100 and DGX H100, NVIDIA HGX H100 systems have different behaviors (Refer to *Error Handling* for more information).

10.1. Common Terms

- *GPU Access NVLink* is an NVLink connection between a GPU and a NVSwitch.
- *GPU Access NVLink failure* is a failure that occurs in the connection between a GPU and an NVSwitch.

Failures can be the result of a GPU/NVSwitch pin failure, a mechanical failure in the GPU baseboard, or a similar failure.

- *Trunk NVLink* are the links that connect two GPU baseboards.

Trunk NVLinks only occur between NVSwitches and travel over the NVLink bridge PCBs and connectors.

- *Trunk NVLink failure* is a trunk NVLink failure that traverses between the two GPU baseboard trays.

This failure can be the result of a bad backplane connector pin or a similar issue.

- *NVSwitch failure* is a NVSwitch failure that is categorized as an internal failure of the NVSwitch.

This failure can be the result of the NVSwitch not being displayed on the PCIe bus, a DBE error, or a similar issue.

- *GPU failure* is where the GPU has failed.

This failure can be the result of NVLink connectivity, a PCIe failure, or a similar issue.

Note

These high availability modes and their corresponding dynamic reconfiguration of the NVSwitch-based system are applied in response to errors that are detected during FM initialization. Runtime errors that occur after the system is initialized, or when a GPU job is running, will not trigger these high availability mode policies.

10.2. GPU Access NVLink Failure

This section provides information about the GPU access NVLink failure.

10.2.1. Fabric Manager Config Item

The GPU access NVLink failure mode is controlled the following item:

```
ACCESS_LINK_FAILURE_MODE=<value>
```

10.2.2. Bare Metal Behavior

- `ACCESS_LINK_FAILURE_MODE=0`

In this mode, FM removes the GPUs with an access NVLink failure from NVSwitch routing and configures the rest of the GPUs to form one memory fabric. The GPUs with the access NVLink failure will lose their NVLink P2P capability with other GPUs.

The failed GPUs are still visible to the NVIDIA software stack, such as CUDA, NVML, NVIDIA-SMI, and so on and can be used for non-NVLink workloads.

- `ACCESS_LINK_FAILURE_MODE=1`

In this mode, if there are two GPU baseboards where the GPU Access NVLink is connected, FM will disable NVSwitch and its pair of Trunk NVLinks. This reduces the NVLink P2P bandwidth to 5/6 throughout the fabric. If a GPU can access NVLink failures to more than one NVSwitch, this option will remove the GPU from the NVSwitch routing configuration and disable its NVLink P2P capability.

- This process will leave the other GPUs with complete NVLink P2P bandwidth.
- If multiple GPU access NVLink failures point to the same NVSwitch, that NVSwitch will be disabled.
- This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

10.2.3. Shared NVSwitch and vGPU Virtualization Behavior

- `ACCESS_LINK_FAILURE_MODE=0`

In this mode, FM removes the GPUs with access NVLink failures from the currently supported GPU partition list. Figure 10.1 shows the effect of one GPU having an access NVLink failure in a two-GPU baseboard system. The failed GPUs will be available for single GPU partitions.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

- `ACCESS_LINK_FAILURE_MODE=1`

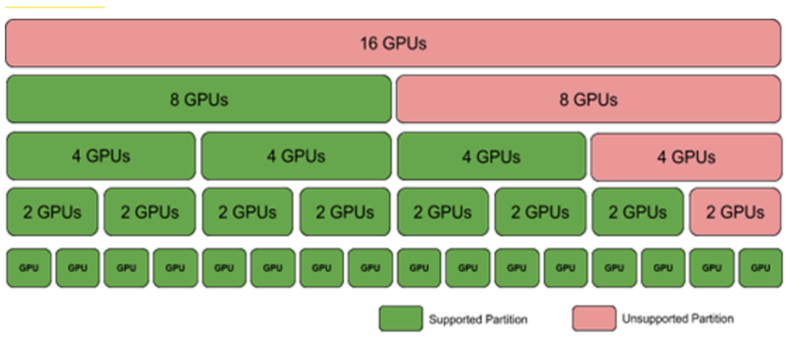

Figure 10.1: Shared NVSwitch and vGPU Partitions When a GPU Access NVLink Fails

In the Shared NVSwitch mode, all GPU partitions will be available, but the partitions will reduce the available bandwidth to 5/6 throughout the fabric. If multiple access NVLinks fail on one GPU, the GPU will be removed, and the available GPU partitions will be adjusted as mentioned earlier. The failed GPUs will be available for single GPU partitions.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

Note

Currently, the `ACCESS_LINK_FAILURE_MODE=1` configuration is not supported in the vGPU Multi-tenancy Mode.

10.3. Trunk NVLink Failure

This section provides information about the trunk NVLink failure.

10.3.1. Fabric Manager Config Item

The Trunk NVLink failure mode is controlled through the following FM config file item:

`TRUNK_LINK_FAILURE_MODE=<value>`

Note

This option applies only to systems with two GPU baseboards.

10.3.2. Bare Metal Behavior

- `TRUNK_LINK_FAILURE_MODE=0`

In this mode, FM aborts and leaves the system uninitialized when there is a trunk NVLink failure, and all CUDA application launches will fail with the `cudaErrorSystemNotReady` status. However, when `FM_STAY_RESIDENT_ON_FAILURES =1`, the `continue with error config` option is enabled, and the FM service continues to run, and the CUDA application launches will fail with `cudaErrorSystem-NotReady` status.

This mode is effective only on the DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

- `TRUNK_LINK_FAILURE_MODE=1`

In this mode, if an NVSwitch has one or more trunk NVLink failures, the NVSwitch will be disabled with its peer NVSwitch. This reduces the available bandwidth to 5/6 throughout the fabric. If multiple NVSwitches have trunk NVLink failures, FM will fall back to the `TRUNK_LINK_FAILURE_MODE=0` behavior.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

10.3.3. Shared NVSwitch and vGPU Virtualization Behavior

- `TRUNK_LINK_FAILURE_MODE=0`

In this mode, FM removes GPU partitions by using trunk NVLinks from the currently supported GPU partition list, so 16-GPU partitions and eight-GPU partitions across baseboards will be removed. The remaining partitions will run with complete NVLink bandwidth. This option will support an unlimited number of trunk NVLink failures on a connected pair of NVSwitches.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

- `TRUNK_LINK_FAILURE_MODE=1`

In the Shared NVSwitch mode, the GPU partitions will be available, but the partitions will reduce the available bandwidth to 5/6 throughout the fabric. This option will be supported when multiple trunk NVLink failures occur on the same NVSwitch pair. If multiple trunk NVLink failures affect different NVSwitch pairs, FM will fall back to the `TRUNK_LINK_FAILURE_MODE=0` behavior.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

Note

The `TRUNK_LINK_FAILURE_MODE=1` configuration is currently not supported in the vGPU Multitenancy Mode.

10.4. NVSwitch Failure

This section provides information about NVSwitch failures.

10.4.1. Fabric Manager Config Item

The NVSwitch failure mode is controlled through this FM config file item:

`NVSWITCH_FAILURE_MODE=<value>`

10.4.2. Bare Metal Behavior

- `NVSWITCH_FAILURE_MODE=0`

In this mode, when there is an NVSwitch failure, FM aborts and leaves the system uninitialized, and all CUDA application launches will fail with a `cudaErrorSystemNotReady` status. However, when `FM_STAY_RESIDENT_ON_FAILURES =1`, the `continue with error config` option is enabled, the FM service continues to run, and CUDA application launches will fail with a `cudaErrorSystemNotReady` status.

- `NVSWITCH_FAILURE_MODE =1`

In this mode, when there is an NVSwitch failure, the NVSwitch will be disabled with its peer NVSwitch. This will reduce the available bandwidth to 5/6 throughout the fabric. If multiple NVSwitch failures occur, FM will fall back to the `NVSWITCH_FAILURE_MODE=0`.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

10.4.3. Shared NVSwitch and vGPU Virtualization Behavior

- `NVSWITCH_FAILURE_MODE=0`

In this mode, FM will remove multi-GPU partitions from the baseboard with the failing NVSwitch and eight-GPU partitions across baseboards. In one baseboard system, only single GPU partitions will be supported. Figure 10.2 shows the supported partitions when an NVSwitch has failed.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

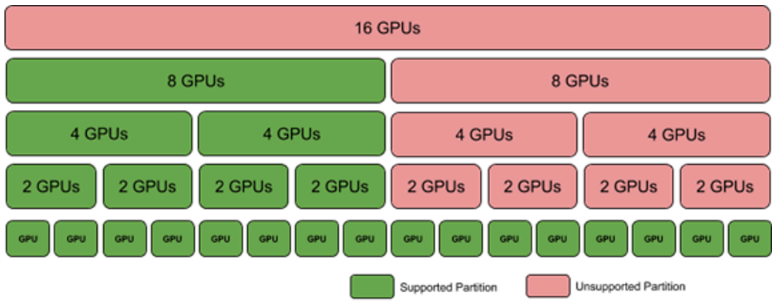

Figure 10.2: Shared NVSwitch and vGPU Partitions when an NVSwitch has Failed

- `NVSWITCH_FAILURE_MODE=1`

In the Shared NVSwitch mode, all GPU partitions will be available, but the partitions will reduce the available bandwidth to 5/6 throughout the fabric. If multiple NVSwitch failures happen, FM will fall back to `NVSWITCH_FAILURE_MODE =0` behavior.

This mode is effective only on DGX A100 and NVIDIA HGX A100 NVSwitch-based systems.

Note

Currently, the `NVSWITCH_FAILURE_MODE=1` configuration is not supported in the vGPU Multitenancy Mode.

10.5. GPU Failure

This section provides information about GPU failures.

10.5.1. Bare Metal Behavior

FM will ignore GPUs that have failed to initialize, are not displayed on the PCI bus, and so on. FM will set up routing and enable NVLink P2P among the available GPUs.

10.5.2. Shared NVSwitch and vGPU Virtualization Behavior

FM will continue initialization and adjust the currently supported partition list by excluding the failed GPU partitions. Figure 10.3 shows the supported partitions when a GPU is missing or has failed to initialize.

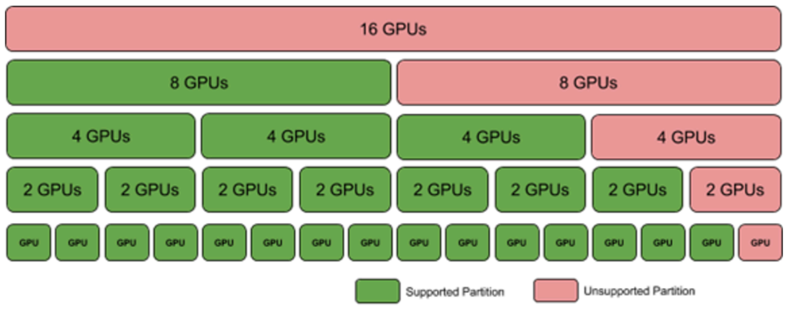

Figure 10.3: Shared NVSwitch and vGPU Partitions When a GPU is Missing or Has Failed

10.6. Manual Degradation

Manual degradation prevents a consistently failing GPU, NVSwitch, or baseboard from being enumerated by the NVSwitch system software stack. Depending on the failing component, the system administrator must configure appropriate action.

10.6.1. GPU Exclusion

Depending on the errors, certain GPUs might be candidates for exclusion from the system so that FM can successfully initialize and configure the rest of the GPU subsets. Based on the failure analysis data from previous generation GPUs, to exclude a GPU, here are the recommended error conditions:

- GPU double bit ECC errors.
- GPU falling off the PCIe bus.
- GPU failure to enumerate on the PCIe bus.
- GPU side NVLink training error.
- GPU side unexpected XID.
- This category can also be application induced.

For full passthrough virtualization, the administrator must identify the GPUs that should be excluded. The hypervisor must ensure that VMs are not created on the GPUs that have been identified as candidates for exclusion.

10.6.1.1 GPU Exclusion Flow

Here are the phases in the GPU exclusion flow:

1. Running application error handling.
2. Diagnosing GPU failures.
3. Remediating the error.

The steps for each of these phases can vary based on whether the system is running in bare metal or in virtualized mode. The following sections describe the flow for bare metal and virtualized platforms.

10.6.1.2 Running Application Error Handling

Errors faced by the GPU during active execution, such as GPU ECC errors, GPU falling off the bus, and so on, are reported through the following means:

- `/var/log/syslog` as an XID message
- DCGM
- NVIDIA Management Library (NVML)
- GPU SMBPBI-based OOB commands
- The FM log file.

Table 10.1 Error Conditions and Signatures

Error Condition	Error signature on Running Application
GPU Double Bit Error	XID 48 output by GPU driver
GPU falling off PCIe bus	XID 79 output by GPU driver
GPU failing to enumerate on bus	GPU does not appear to applications (CUDA applications or `nvidia-smi` query)
GPU side NVLink training error	Error output to `/var/log/syslog` by FM
GPU side errors	Other XIDs output by GPU driver. This can also be application induced.
GPU Double Bit Error	XID 48 output by GPU driver

10.6.1.3 Diagnosing GPU Failures

System administrators can create their own GPU monitoring/health check scripts to look for the error traces. This process requires looking for at least one of the above-mentioned sources (syslog, NVML APIs, and so on) to collect the necessary data.

DCGM includes an exclusion recommendation script that can be invoked by a system administrator to collect the GPU error information. This script queries information from the passive monitoring performed by DCGM to determine whether conditions that might require a GPU to be excluded have occurred since the previous time the DCGM daemon was started. As part of the execution, the script invokes a validation test that determines whether unexpected XIDs are being generated by the execution of a known good application. Users can prevent the validation test from being run and only monitor the passive information.

Note

The DCGM exclusion recommendation script code is provided as a reference for system administrators to extend as appropriate or build their own monitoring/health check scripts. Refer to the NVIDIA DCGM documentation for more information about the exclusion recommendation script such as its location and supported options.

10.6.1.4 In-Band GPU Exclude Mechanism

The GPU kernel driver on NVSwitch-based systems can be configured to ignore a set of GPUs, even if the GPUs were enumerated on the PCIe bus. The GPUs that will be excluded are identified by the GPU's unique identifier (GPU UUID) by using a kernel module parameter. After identifying whether the GPU exclude candidates are in the system, the GPU kernel module driver will exclude the GPU from being used by applications. If a GPU UUID is in the exclude candidate list, but the UUID was not detected at runtime because the UUID belonged to a GPU that is not on the system or because the PCIe enumeration of the GPU board failed, the GPU is not considered to have been excluded.

The list of exclude candidate GPUs can be persisted across reboots by specifying the module parameters by using a .conf file in the filesystem. The exclude mechanism is specific to a GPU rather than a physical location on the baseboard. As a result, if a GPU is on the exclude candidate list, and is later replaced by a new GPU, the new GPU will become visible to the system without updating the exclude candidates. Conversely, if a GPU has been excluded on a system, placing it in different PCIe slots will prevent the GPU from being visible to applications, unless the exclude candidate list is updated.

Updating the GPU excludes candidates requires manual intervention by the system administrator.

10.6.1.5 Kernel Module Parameters

The set of candidate GPU UUIDs that will be excluded are specified by using a kernel module parameter that consists of a set of comma-separated GPU UUIDs.

- The kernel parameter can be specified when the kernel module loads `nvidia.ko`.

```
insmod nvidia.ko NVreg_ExcludedGpus=uuid1,uuid2…
```

- To make the GPU UUID persistent, the set of exclude candidate GPU UUIDs can also be specified by using a `nvidia.conf` file in `/etc/modprobe.d`.

```
options nvidia NVreg_ExcludedGpus=uuid1, uuid2…
```

Adding GPUs into the exclude candidate list is a manual step that must be completed by a system administrator.

Note

The previously supported `NVreg_GpuBlacklist` module parameter option has been deprecated and will be removed in a future release.

10.6.1.6 Adding/Removing a GPU from the Exclude Candidate List

To add a GPU from the exclude candidate list or to remove it from the list, the system administrator must complete the following steps:

1. If a conf file does not exist, create a conf file for the NVIDIA kernel module parameters.
2. Complete one of the following tasks:

3. Add the UUID of the excluded GPU into the .conf file.
4. Remove the UUID of the GPU from the list.
5. Restart the system to load the kernel driver with updated module parameters.

10.6.1.7 Listing Excluded GPUs

An excluded GPU is not visible in CUDA applications or in basic queries by using `nvidia-smi -q` or through NVML. This section provides information about the options to identify when a GPU has been excluded, for example, the GPU's UUID was in the exclude candidate list, and the GPU was detected in the system.

10.6.1.8 nvidia-smi

The new command, `nvidia-smi -B or nvidia-smi --list-excluded-gpus` can be used to get a list of excluded GPUs.

10.6.1.9 Procfs

The procfs entry, `/proc/driver/nvidia/gpus/<PCI_ID>/information`, can specify whether the GPU has been excluded.

10.6.1.10 Out-of-Band

Refer to the *NVIDIA GPU SMBus Post-Box Interface (SMBPBI)* documentation for more information.

10.6.1.11 Running GPU Exclusion Scripts

The following section provides information about the recommended flow that a system administrator should follow to run GPU monitoring health checks or the DCGM exclusion recommendation script on various system configurations.

10.6.1.12 Bare Metal and vGPU Configurations

The system administrator will run the bare metal and vGPU virtualization configurations in the same OS instance as the application programs.

Here is the general flow that a system administrator will follow:

1. Periodically run the health check script or the DCGM exclusion recommendation script for all the GPUs and NVSwitches on the system.
2. (**Optional**) Monitor the system logs to trigger a run of the health check script or DCGM exclusion recommendation script.
3. Based on the output of the health check or exclusion recommendation script, add the GPU UUID to the exclude candidate list.
4. If you are using the DCGM exclusion recommendation script, update the periodic run of the exclude recommendation script with the newly expected GPU count.
5. Reboot the system to load the kernel driver with updated module parameters.

10.6.1.13 Full Passthrough Virtualized Configurations

The primary difference in virtualized configurations is that the GPU kernel driver is left to the guest VMs. As a result, the execution of the GPU diagnosis and remediation phases must be performed by the hypervisor with the VM provisioning mechanism.

Here is the general flow that a hypervisor will follow:

1. The guest VM finishes and returns controls of a set of GPUs and switches to the hypervisor.
2. The hypervisor invokes a special test VM, which is trusted by the hypervisor.
3. In test VM, there should be a complete instance of the NVIDIA NVSwitch core software stack, including GPU drivers and FM.
4. On this test VM, run the health check script or DCGM exclusion recommendation script.
5. Based on the output of the health check or exclusion recommendation script, add the GPU UUID to a hypervisor readable database.
6. The hypervisor shuts down the test VM.
7. To prevent that GPU from being assigned to future VM requests, the hypervisor reads the database, identifies the candidates to exclude, and updates its resource allocation mechanisms.

After the GPU board has been replaced, to make the GPU available again, the hypervisor updates the database.

10.6.1.14 Shared NVSwitch Virtualization Configurations

In a shared NVSwitch virtualization configuration, system administrators can run their GPU health check script or DCGM exclusion recommendation script in a dedicated test VM or on DGX A100 and NVIDIA HGX A100 systems, in the Service VM immediately after the GPU partition is activated.

To run GPU health on a special test VM:

1. The guest VM completes and returns control of the GPUs in the partition to the hypervisor.
2. After the shared NVSwitch guest VM shutdown procedure is complete, activate the same GPU partition again.
3. The hypervisor schedules a special test VM, which is trusted on those GPUs.
4. On this test VM, run the health check script or DCGM exclusion recommendation script.
5. Based on the output of the health check or exclusion recommendation script, add the GPU UUID into a hypervisor readable database.
6. If the partition activation/deactivation cycle is consistently failing, the hypervisor can consider adding all the GPU UUID s of a partition to the database.
7. After the health check is complete, shut down the test VM.
8. The hypervisor reads the database to identify the candidates for exclusion and removes the corresponding GPU partitions from its currently supported partitions.
9. The hypervisor resource allocation mechanisms ensure that the affected GPU partitions will not be activated.
10. When the service VM is rebooted, the hypervisor can choose not to bind the excluded GPUs to the service VM.

This way, FM will adjust its currently supported GPU partitions.

11. When the GPU board has been replaced, the hypervisor updates the database to make the GPU available and restarts the service VM with all the GPUs to enable previously disabled GPU partitions again.

To run GPU health on a service VM on DGX 100 and NVIDIA HGX 100 systems:

1. The `fmActivateFabricPartition()` call returned successfully in a Shared NVSwitch partition activation flow.
2. Before the hypervisor detaches/unbinds the GPUs in the partition, run the required health check script or DCGM exclusion recommendation script on those GPUs in the service VM.
3. Based on the output of the health check or exclusion recommendation script, add the GPU UUID into a hypervisor readable database.
4. The hypervisor executes the partition deactivation flow using `fmDeactivateFabricPartition()` when health check fails and corresponding guest VM launch is deferred.
5. If the partition activation/deactivation cycle is consistently failing, the hypervisor can consider adding the GPU UUID s of a partition to the database.
6. The hypervisor reads the database to identify the candidates for exclusion and removes the corresponding GPU partitions from its currently supported partitions.
7. The hypervisor resource allocation mechanisms ensure that the affected GPU partitions will not be activated.
8. After the service VM is rebooted, the hypervisor can choose not to bind the excluded GPUs to the service VM.

This way, FM will adjust its currently supported GPU partitions.

10.6.1.15 Supported High Availability Modes

After the GPU board has been replaced, the hypervisor updates the database to make the GPU available and restarts the service VM with the GPUs to enable previously disabled GPU partitions again.

10.6.2. NVSwitch Exclusion

In DGX A100 and NVIDIA HGX A100 systems, if an NVSwitch is consistently failing, the system administrator can explicitly exclude the NVSwitch.

10.6.2.1 In-Band NVSwitch Exclusion

The NVSwitch kernel driver on NVSwitch-based systems can be configured to ignore an NVSwitch even when the systems were enumerated on the PCIe bus like the GPU exclusion feature. If the NVSwitch exclusion candidates are in the system, the NVSwitch kernel module driver will exclude the NVSwitch from being used by applications. If an NVSwitch UUID is in the exclusion candidate list, but the UUID is not detected at runtime because the UUID belongs to a NVSwitch that is not on the system, or because the PCIe enumeration of the NVSwitch fails, the NVSwitch is not considered to have been excluded.

On NVIDIA HGX A100 systems with two GPU baseboards, if an NVSwitch is explicitly excluded, FM will manually exclude its peer NVSwitch across the Trunk NVLinks. This behavior can be configured using the `NVSWITCH_FAILURE_MODE` high availability configuration file item.

10.6.2.2 Kernel Module Parameters

- To specify a candidate NVSwitch UUID as a kernel module parameter, run the following command.

```
insmod nvidia.ko NvSwitchExcludelist=<NVSwitch_uuid>
```

- To make the NVSwitch UUID persistent, specify the UUID using an nvidia.conf file in `/etc/modprobe.d`.

```
options nvidia NvSwitchExcludelist=<NVSwitch_uuid>
```

The system administrator can get the NVSwitch UUID from the FM log file and add the UUID into the excluded candidate list.

Note

The previously supported `NvSwitchBlacklist` module parameter option has been deprecated and will be removed in a future release.

Note

On DGX B200 and NVIDIA HGX B200 systems, NVSwitches are not controlled by the kernel NVSwitch driver module, so NVSwitches cannot be excluded by using kernel module parameters.

10.6.2.3 Out-of-Band NVSwitch Exclusion

Refer to *SMBus Post Box Interface (SMBPBI)* for more information about NVSwitch.

Chapter 11. NVLink Topology

This chapter provides information about the link IDs used by each GPU to connect to each NVSwitch on different versions of NVIDIA HGX baseboards.

11.1. The NVIDIA HGX-2 GPU Baseboard

Every NVSwitch uses the 0/1, 2/3, 8/9 and 10/11 links for the inter-GPU baseboard connection, and the links are not listed. Other NVLink connections, two per NVSwitch, are unused.

Table 11.1 GPUs and NVSwitch Links

GPU	GPU link	NVSwitch	NVSwitch link
1	0	4	16
1	1	1	5
1	2	6	6
1	3	3	15
1	4	5	15
1	5	2	6
2	0	4	15
2	1	1	16
2	2	3	6
2	3	6	12
2	4	2	17
2	5	5	7
3	0	4	14
3	1	1	17
3	2	3	17
3	3	6	13
3	4	5	6

continues on next page

Table 11.1 – continued from previous page

GPU	GPU link	NVSwitch	NVSwitch link
3	5	2	4
4	0	4	17
4	1	1	4
4	2	3	7
4	3	6	7
4	4	2	7
4	5	5	17
5	0	4	13
5	1	1	13
5	2	5	13
5	3	3	13
5	4	6	14
5	5	2	16
6	0	4	5
6	1	1	14
6	2	6	5
6	3	3	4
6	4	2	12
6	5	5	14
7	0	4	12
7	1	1	15
7	2	5	5
7	3	3	5
7	4	2	13
7	5	6	17
8	0	4	4
8	1	1	12
8	2	6	15
8	3	5	12
8	4	3	14
8	5	2	5

11.2. The NVIDIA HGX A100 GPU Baseboard

Every NVSwitch uses links 0 to 7 and 16 to 23 for the inter-GPU baseboard connection, and the links are not listed. Other NVLink connections (four per NVSwitch) are unused.

The GPU numbering in Table 11.2 is the same numbering used in the *HGX A100 Baseboard Pinout* design document.

Table 11.2 NVLink Topology of the NVIDIA HGX A100 GPU Baseboard

GPU	GPU link	NVSwitch	NVSwitch link
1	0, 1	4	8, 9
1	2, 3	1	24, 25
1	4, 5	3	30, 31
1	6, 7	6	12, 13
1	8, 9	2	12, 13
1	10, 11	5	30, 31
2	0, 1	4	30, 31
2	2, 3	1	26, 27
2	4, 5	3	12, 13
2	6, 7	6	24, 25
2	8, 9	2	34, 35
2	10, 11	5	14, 15
3	0, 1	4	28, 29
3	2, 3	1	34, 35
3	4, 5	3	34, 35
3	6, 7	6	26, 27
3	8, 9	2	8, 9
3	10, 11	5	12, 13
4	0, 1	4	34, 35
4	2, 3	1	32, 33
4	4, 5	3	14, 15
4	6, 7	6	14, 15
4	8, 9	2	14, 15
4	10, 11	5	34, 35
5	0, 1	4	26, 27
5	2, 3	1	10, 11
5	4, 5	3	28, 29

continues on next page

Table 11.2 – continued from previous page

GPU	GPU link	NVSwitch	NVSwitch link
5	6, 7	6	28, 29
5	8, 9	2	10, 11
5	10, 11	5	26, 27
6	0, 1	4	10, 11
6	2, 3	1	28, 29
6	4, 5	3	8, 9
6	6, 7	6	10, 11
6	8, 9	2	24, 25
6	10, 11	5	28, 29
7	0, 1	4	24, 25
7	2, 3	1	30, 31
7	4, 5	3	26, 27
7	6, 7	6	30, 31
7	8, 9	2	26, 27
7	10, 11	5	10, 11
8	0, 1	4	32, 33
8	2, 3	1	8, 9
8	4, 5	3	10, 11
8	6, 7	6	34, 35
8	8, 9	2	32, 233
8	10, 11	5	24, 25

11.3. The NVIDIA HGX H100 GPU Baseboard

The GPU numbering in Table 11.3 is the same information that is returned through `nvidia-smi` as the module ID, which is derived based on the GPIO connections on the baseboard.

Table 11.3 NVLink Topology of the NVIDIA HGX H100 GPU Baseboard

GPU	GPU link	NVSwitch	NVSwitch link
1	2,3,12,13	1	40,41,44,45
1	0,1,11,16,17	2	36,37,40,46,47
1	15,14,10,6,7	3	42,43,45,62,63

continues on next page

Table 11.3 – continued from previous page

GPU	GPU link	NVSwitch	NVSwitch link
1	4,5,9,8	4	58,59,62,63
2	15,14,8,9	1	42,43,46,47
2	2,3,7,6,11	2	2,3,4,5,32
2	10,5,4,0,1	3	34,40,41,46,47
2	12,13,16,17	4	34,35,38,39
3	13,12,7,6	1	48,49,52,53
3	17,16,10,3,2	2	0,1,33,38,39
3	14,15,8,9,11	3	16,17,50,51,52
3	5,4,1,0	4	56,57,60,61
4	9,8,13,12	1	32,33,36,37
4	2,3,10,14,15	2	50,51,53,62,63
4	7,6,11,16,17	3	2,3,35,38,39
4	5,4,1,0	4	42,43,46,47
5	7,6,12,13	1	58,59,62,63
5	17,16,11,1,0	2	48,49,52,56,57
5	15,14,10,2,3	3	36,37,44,60,61
5	4,5,9,8	4	48,49,52,53
6	6,7,15,14	1	34,35,38,39
6	8,9,17,16,11	2	6,7,34,35,42
6	4,5,10,1,0	3	0,1,19,32,33
6	13,12,3,2	4	32,33,36,37
7	17,16,13,12	1	50,51,54,55
7	10,0,1,4,5	2	43,54,55,58,59
7	15,14,11,8,9	3	48,49,53,56,57
7	7,6,3,2	4	40,41,44,45
8	12,13,17,16	1	56,57,60,61
8	10,5,4,0,1	2	41,44,45,60,61
8	11,14,15,7,6	3	18,54,55,58,59
8	2,3,8,9	4	50,51,54,55

Note

The DGX H200, NVIDIA HGX H200 and NVIDIA HGX H20 NVLink topology is same as the H100 variant.

11.4. The NVIDIA HGX H800 GPU Baseboard

On HGX H800, every GPU has two NVLinks to each of the four NVSwitches.

Table 11.4 NVLink Topology of the NVIDIA HGX H800 GPU Baseboard

GPU	GPU link	NVSwitch	NVSwitch link
1	3,13	1	41, 45
1	1,17	2	37,47
1	15,6	3	42,62
1	5,8	4	59,63
2	15,8	1	42,46
2	3,6	2	3,5
2	5,1	3	40,47
2	13,17	4	35,39
3	13,6	1	48,53
3	17,3	2	0,38
3	15,8	3	17,50
3	5,1	4	56,60
4	8,13	1	33,36
4	3,15	2	51,63
4	6,17	3	3,39
4	5,1	4	42,46
5	6,13	1	59,63
5	17,1	2	48,56
5	15,3	3	36,61
5	5,8	4	49,53
6	6,15	1	34,38
6	8,17	2	6,34
6	5,1	3	1,32
6	13,3	4	32,36
7	17,13	1	50,54
7	1,5	2	55,59
7	15,8	3	48,56
7	6,3	4	41,44
8	13,17	1	57,60

continues on next page

Table 11.4 – continued from previous page

GPU	GPU link	NVSwitch	NVSwitch link
8	5,1	2	44,61
8	15,6	3	55,59
8	3,8	4	51,54

11.5. The NVIDIA HGX B200 GPU Baseboard

On HGX B200, every GPU has nine NVLinks to each of the two NVSwitches.

Table 11.5 NVLink Topology of the NVIDIA HGX B200 GPU Baseboard

GPU	GPU link	NVSwitch	NVSwitch link
1	0,1,3,4,6,9,11,14,16	2	2,10,12,6,4,14,16,8,1
1	2,5,7,8,10,12,13,15,17	1	3,2,4,1,7,5,6,8,10
2	0,1,3,4,6,9,11,14,16	2	3,20,18,7,5,24,22,9,11
2	2,5,7,8,10,12,13,15,17	1	9,12,14,11,13,15,16,18,20
3	0,1,3,4,6,9,11,14,16	2	13,26,28,17,15,30,32,19,21
3	2,5,7,8,10,12,13,15,17	1	19,22,24,17,23,21,26,28,30
4	0,1,3,4,6,9,11,14,16	2	23,36,34,27,25,40,38,29,31
4	2,5,7,8,10,12,13,15,17	1	25,32,34,27,29,31,36,38,40
5	0,1,3,4,6,9,11,14,16	2	33,42,44,37,35,46,48,39,41
5	2,5,7,8,10,12,13,15,17	1	35,42,44,33,39,37,46,48,50
6	0,1,3,4,6,9,11,14,16	2	43,52,50,47,45,56,54,49,51
6	2,5,7,8,10,12,13,15,17	1	41,52,54,43,45,47,56,58,60
7	0,1,3,4,6,9,11,14,16	2	53,58,60,57,55,62,64,59,61
7	2,5,7,8,10,12,13,15,17	1	51,62,64,49,55,53,66,68,70
8	0,1,3,4,6,9,11,14,16	2	63,68,66,67,65,72,70,69,71
8	2,5,7,8,10,12,13,15,17	1	57,72,65,59,61,63,67,69,71

Chapter 12. GPU Partitions

This chapter provides information about the default Shared NVSwitch and vGPU partitions for various GPU baseboards.

12.1. DGX-2 and NVIDIA HGX-2

Table 12.1 Default Shared NVSwitch Partitions for DGX-2 and NVIDIA HGX-2

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
0	16	1 to 16	6
1	8	1 to 8	6
2	8	9 to 16	6
3	8	1,4,6,7 from baseboard1 9, 12,14, 15 from baseboard2	5
4	8	2,3,5,8 from baseboard1 10, 11, 13,16 from baseboard2	5
5	4	1,4,6,7	5
6	4	2,3,5,8	5
7	4	9,12,14,15	5
8	4	10,11,13,16	5
9	2	1,4	5
10	2	2,3	5
11	2	5,8	5
12	2	6,7	5
13	2	9,12	5
14	2	10,11	5
15	2	13,16	5

continues on next page

Table 12.1 – continued from previous page

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
16	2	14,15	5
17 to 32	1	Physical ID 1 for Partition ID 17, Physical ID 2 for Partition ID 18, and so on.	0

In this generation of NVSwitch, the NVLink ports reset (even-odd pair of links) must be issued in pairs. As a result, NVIDIA HGX-2 and DGX-2 only support a fixed mapping of Shared NVSwitch partitions, and the four-GPU and two-GPU VMs can enable only five out of six NVLinks per GPU.

DGX A100 and NVIDIA HGX A100

This section provides information about DGX A100 and NVIDIA HGX A100.

12.1.1. Default GPU Partitions

Depending on the high availability mode configurations, when a GPU is unavailable because of failures, backlisting, and so on, the corresponding partitions will be removed from the supported partition list. However, the Partition ID and GPU Physical IDs will remain the same for the rest of the supported partitions.

Note

The GPU Physical IDs are based on how the GPU baseboard NVSwitch GPIOs are strapped. If there is only one baseboard, and the GPIOs are strapped for the bottom tray, the GPU Physical IDs range is 1-8. If the baseboard is strapped for the top tray, the GPU Physical IDs range is 9-16.

Table 12.2 Default Shared NVSwitch and vGPU Partitions for DGX A100 and NVIDIA HGX A100

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
0	16	1 to 16	12
1	8	1 to 8	12
2	8	9 to 16	12
3	8	1 to 4 & 9 to 12	12
4	8	5 to 8 & 13 to 16	12
5	8	1 to 4 & 13 to 16	12
6	8	5 to 8 & 9 to 12	12
7	4	1, 2, 3, 4	12

continues on next page

Table 12.2 – continued from previous page

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
8	4	5, 6, 7, 8	12
9	4	9, 10, 11, 12	12
10	4	13, 14, 15, 16	12
11	2	1, 2	12
12	2	3, 4	12
13	2	5, 6	12
14	2	7, 8	12
15	2	9, 10	12
16	2	11, 12	12
17	2	13, 14	12
18	2	15, 16	12
19	1	1	0
20 to 34	1	Physical ID 2 for Partition ID 20, Physical ID 3 for Partition ID 21, etc.	0

12.1.2. Supported GPU Partitions

In DGX A100 and NVIDIA HGX A100 systems, the earlier generation of the even-odd pair NVSwitch NVLink reset requirement is no longer applicable. So, if the default GPU partition mentioned above is not optimal based on the system's PCIe topology, the partition mapping can be changed. However, NVIDIA has the following restrictions for partition definitions:

- The two-GPU NVLink partitions must be in the same GPU baseboard.
- The four-GPU NVLink partitions must be in the same GPU baseboard.
- For eight-GPU NVLink partitions, which span across two GPU baseboards, four GPUs must be from each baseboard.

Note

NVIDIA will evaluate any custom partition definition requests and variations of the policy on a case-by-case basis and will provide necessary information to configure/override the default GPU partitions.

12.2. DGX H100 and NVIDA HGX H100

This section provides information about DGX H100 and NVIDIA H100.

12.2.1. Default GPU Partitions

Table 12.3 Default Shared NVSwitch Partitions for DGX H100 and NVIDIA HGX H100

Partition ID	Number of GPUs	GPU Physical ID Module ID	Number of NVLink Interconnects per GPU
0	8	1 to 8	18
1	4	1 to 4	18
2	4	5 to 8	18
3	2	1,3	18
4	2	2,4	18
5	2	5,7	18
6	2	6,8	18
7	1	1	0
8	1	2	0
9	1	3	0
10	1	4	0
11	1	5	0
12	1	6	0
13	1	7	0
14	1	8	0

Note

The DGX H200, NVIDIA HGX H200, NVIDIA HGX H800, and NVIDIA HGX H20 have the same default NVLink partition as the H100 variant.

12.2.2. Supported GPU Partitions

In DGX H100 and NVIDIA HGX H100 systems, regardless of the GPU degradation states, the GPU partitions above are returned in the get support partition API.

DGX B200 and NVIDIA HGX B200

This section provides information about DGX B200 and NVIDIA HGX B200.

12.2.3. Default GPU Partitions

Table 12.4 Default Shared NVSwitch Partitions for DGX B200 and NVIDIA HGX B200

Partition ID	Number of GPUs	GPU Physical ID Module ID	Number of NVLink Interconnects per GPU
1	8	1 to 8	18
2	4	1 to 4	18
3	4	5 to 8	18
4	2	1,2	18
5	2	3,4	18
6	2	5,6	18
7	2	7,8	18
8	1	1	0
9	1	2	0
10	1	3	0
11	1	4	0
12	1	5	0
13	1	6	0
14	1	7	0
15	1	8	0

12.2.4. Supported GPU Partitions

In DGX B200 and NVIDIA HGX B200 systems, regardless of the GPU degradation states, the GPU partitions in Table 12.4 are returned in the get support partition API.

12.2.5. Custom Shared NVSwitch Partitions

Customized shared NVSwitch partitions can be defined to replace the default ones by adding the `SHARED_PARTITION_DEFINITION_FILE` configuration in the fabricmanager.cfg configuration file. The path to the file is `SHARED_PARTITION_DEFINITION_FILE =/usr/share/nvidia/nvswitch/customPartition.json`.

The custom partitions are defined in JSON format, but you need to save the file with the .json extension.

```
CustomPartition.json
{
"version" : 0,
"name": "customer shared fabric partition name",
"time": "Fri Oct 18 11:11:11 2023",
"partitionInfo": [
{
"partitionId": 1,
```

(continues on next page)

(continued from previous page)

```
"gpuModuleIds": [1, 2, 3, 4, 5, 6, 7, 8],
},
{
"partitionId": 2,
"gpuModuleIds": [1, 2, 5, 6],
},
{
"partitionId": 3,
"gpuModuleIds": [3, 4, 7, 8],
},
{
"partitionId": 4,
"gpuModuleIds": [1, 3],
},
{
"partitionId": 5,
"gpuModuleIds": [2, 4],
}
{
"partitionId": 6,
"gpuModuleIds": [5, 7],
},
{
"partitionId": 7,
"gpuModuleIds": [6, 8],
},
{
"partitionId": 8,
"gpuModuleIds": [1],
},
{
"partitionId": 9,
"gpuModuleIds": [2],
},
{
"partitionId": 10,
"gpuModuleIds": [3],
},
{
"partitionId": 11,
"gpuModuleIds": [4],
},
{
"partitionId": 12,
"gpuModuleIds": [5],
},
{
"partitionId": 13,
"gpuModuleIds": [6],
},
{
"partitionId": 14,
```

(continues on next page)

(continued from previous page)

```
"gpuModuleIds": [7],
},
{
"partitionId": 15,
"gpuModuleIds": [8],
},
]
}
```

Chapter 13. Resiliency

The FM resiliency feature in the Shared NVSwitch and vGPU Model allows system administrators to resume normal operation after FM gracefully (or non-gracefully) exits in the service VM. With this feature, currently activated guest VMs will continue to forward NVLink traffic even when FM is not running. After FM is successfully restarted, FM will support the typical guest VM activation /deactivation workflow.

The NVSwitch and GPU NVLink errors that were detected when FM is not running will be cached into the NVSwitch Driver and be reported after FM has successfully restarted. Also, changing the FM version when FM is not running is not supported.

High-Level Flow

1. After an FM crash or a graceful exit, to start FM and resume the operation, the hypervisor will run the `–restart` option.
2. After restarting FM, in 60 seconds the hypervisor will use the `fmSetActivatedFabricPartitions()` API and provide a list of currently activated guest VM partitions.

This is because FM does not know about the guest VM changes when it is not running. If there are no activated guest VM partitions running when FM is restarted, the hypervisor will call the `fmSetActivatedFabricPartitions()` API with the number of partitions as zero.

3. To start FM with the typical process, or to reinitialize the software and hardware states, the hypervisor will follow the typical service VM starting sequence without the `--restart` option.

Detailed Resiliency Flow

When FM is started in normal mode, after initializing all the NVLink devices and discovering the NVLink connections, FM will save the required metadata information in the `/tmp/fabricmanger.state` file. However, this location can be changed by setting the new file location to the `STATE_FILE_NAME` FM config file item. The saved state is a snapshot of detected GPU information (UUID, physical Id) and the currently supported guest VM partition metadata.

Here is the workflow:

1. When FM is started with the `–restart` option, it will skip most of its NVLink and NVSwitch initialization steps and populate the required information from the stored file.
2. FM will wait for the hypervisor to provide a list of currently activated guest VM partitions.
3. During this time, typical partition operations such as querying the list of supported guest VM partitions, activating and deactivating guest VM partitions, and so on, will be rejected.
4. After the list of active guest VM partition information is received from hypervisor, FM will ensure that routing is enabled only for those partitions.
5. FM will enable the typical guest VM partition activation and the deactivation workflow.

6. If FM cannot resume from the current state or the hypervisor does not provide the list of currently activated guest VM partitions before the timeout period, the restart operation will be aborted, and FM will exit.

Figure 13.1 shows the high-level flow when FM is started with typical command-line options and the `−restart` option.

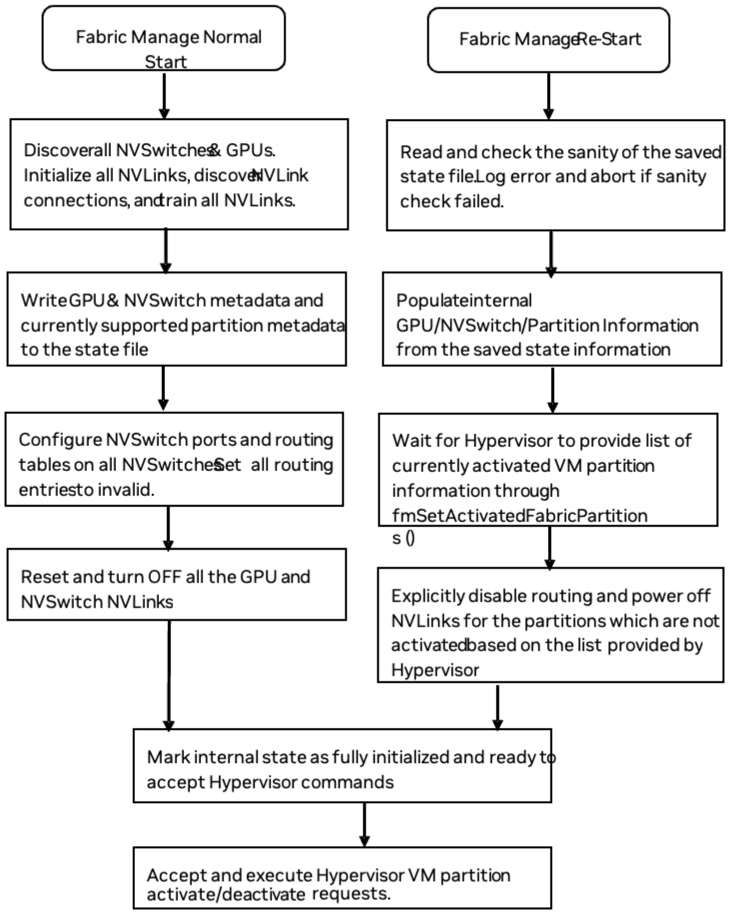
Fabric Manage Normal Start
Discoverall NVSwitches& GPUs. Initialize all NVLinks, discoverNVLink connections, andtrain all NVLinks.
WriteGPU& NVSwitch metadata and currently supported partition metadata to the state file
Configure NVSwitch ports and routing tables on all NVSwitchesSet all routing entriesto invalid.
Reset and turn OFF all the GPU and NVSwitch NVLinks
Fabric ManageRe-Start
Read and check the sanity of the saved state file.Log error and abort if sanity check failed.
Populateinternal GPU/NVSwitch/Partition Information from the saved state information
Wait for Hypervisor to provide list of currently activated VM partition information through fmSetActivatedFabricPartitions ()
Explicitly disable routing and power off NVLinks for the partitions which are not activatedbased on the list provided by Hypervisor
Mark internal state as fully initialized and ready to accept Hypervisor commands
Accept and execute Hypervisor VM partition activate/deactivate requests.

Figure 13.1: Fabric Manager Flow: Typical and Restart Options

Chapter 14. Error Handling

This chapter provides information about error handling.

14.1. Fabric Manager Initialization Errors

The errors in Table 14.1 might occur during FM initialization and topology discovery and happen only during Host boot time (vGPU mode) or service VM initialization (Shared NVSwitch mode).

Note

We assume no guest VMs are running.

Table 14.1 Errors During FM Initialization

Error Condition	Error Impact	Recovery
Access NVLink connection (GPU to NVSwitch) training failure.	Depending on the `ACCESS_LINK_FAILURE_M` configuration, FM will disable partitions that are using the Access NVLink failed GPU or disable the connected NVSwitch (and its peer NVSwitch) and support the partitions with reduced bandwidth.	Restart the FM service (vGPU mode) or the service VM (Shared NVSwitch mode). If the error persists, RMA the GPU.
Trunk NVLink connection (NVSwitch to NVSwitch) training failure.	Depending on the `TRUNK_LINK_FAILURE_` configuration, FM will remove partitions that are using Trunk NVLinks or disable the NVSwitch (and its peer NVSwitch) and support the partitions with reduced bandwidth.	Restart the FM service (vGPU mode) or the service VM (Shared NVSwitch mode). If the error persists, inspect/reseat the NVLink Trunk backplane connector.

continues on next page

Table 14.1 – continued from previous page

Error Condition	Error Impact	Recovery
Any NVSwitch or GPU programming/configuration failures and typical software errors.	Treated as fatal error and FM service will abort. However, if the `FM_STAY_RESIDENT_ON` configuration option is set, the FM service will stay running, but partition activation/deactivation flow will not be supported.	Restart the host and FM service (vGPU mode) or the service VM (NVSwitch mode) If the error persists, technical troubleshooting is required.

14.2. Partition Life Cycle Errors

Table 14.2 summarizes potential errors returned by FM SDK APIs when querying supported partitions or activating/deactivating VM partitions.

Table 14.2 Virtual Machine Partition Life Cycle Errors

Return Code	Error Condition/Impact	Recovery
FM_ST_BADPARAM	The provided partition ID or other parameters to the APIs are invalid.	Use only the partition IDs returned by `fmGetSupportedFabricPartitions` (). Ensure that pointer arguments are not NULL.
FM_ST_NOT_SUPPORTED	FM is not started with required config options.	Ensure that the shared fabric mode is enabled in the FM config file. If not, set the desired value and restart `FMservice`. Shared NVSwitch Mode: `SHARED_FABRIC_MODE = 1` vGPU Mode: `SHARED_FABRIC_MODE = 2`
FM_ST_NOT_CONFIGURED	The FM APIs were issued before FM is completely initialized.	Wait until the FM service is completely initialized.
FM_ST_UNINITIALIZED	The FM interface library has not been initialized.	Ensure that the FM interface library is initialized with a call to `fmLibInit()`.

continues on next page

Table 14.2 – continued from previous page

Return Code	Error Condition/Impact	Recovery
FM_ST_IN_USE	The provided partition ID is already activated or the GPUs required for the specified partition are being used on another activated partition.	Provide a non-activated partition ID. Ensure that the GPUs are not in use by other activated partitions.
FM_ST_UNINITIALIZED	The provided partition ID is already deactivated.	Provide an activated partition ID.
FM_ST_GENERIC_ERROR	A generic error occurred when activating/deactivating a VM partition.	Check the associated syslog for specific and detailed error information.
FM_ST_TIMEOUT	A GPU or NVSwitch configuration setting timed out.	Check the associated syslog for specific and detailed error information.
FM_ST_ VERSION_MISMATCH	The client application using the FM APIs might have compiled/linked with a different version of FM package that is running on the service VM.	Ensure that the client application is compiled and linked with the same, or a compatible, FM package that is installed on the Service VM.

14.3. Runtime NVSwitch Errors

NVSwitch runtime errors can be retrieved or monitored in one of the following ways:

- Through the Host or service VM syslog and Fabric Manager log file as SXid errors.
- Through the NVSwitch public API interface.
- Through the NVSwitch SMBPBI-based OOB commands.

When an NVSwitch port generates an SXid error, the corresponding error information and affected GPU partition information will be logged into the host or service VM syslog.

Depending on the type of SXid errors and the impacted port, the GPUs on the corresponding guest VM or all other guest VMs might be impacted. Generally, if the impact is local to a guest VM, the other running guest VMs will not be affected and should function normally.

14.4. Non-Fatal NVSwitch SXid Errors

Table 14.3 lists potential NVSwitch non-fatal SXid errors that might occur in the field and their impact.

Table 14.3 Potential Non-Fatal NVSwitch SXid Errors

SXid & Error String	Guest VM Impact	Guest VM Recovery	Other Guest VM Impact
11004 (Ingress invalid ACL) This SXid error can happen only because of an incorrect FM partition configuration and is expected not to occur in the field.	The corresponding GPU NVLink traffic will be stalled, and the subsequent GPU access will hang. The GPU driver on the guest VM will abort CUDA jobs with Xid 45.	Validate the GPU/NVSwitch fabric partition routing information using the NVSwitch-audit tool. Restart the guest VM.	If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports will be affected.
11012, 11021, 11022. 11023, 12021, 12023, 15008, 15011, 19049, 19055, 19057, 19059, 19062, 19065, 19068, 19071, 24001, 24002, 24003 (Single bit ECC errors)	No guest VM impact because the NVSwitch hardware will auto correct the ECC errors.	Not Applicable.	No Impact.
20001 (TX Replay Error)	The NVLink packet needs to be retransmitted. This error might impact the NVLink throughput of the specified port.	Not Applicable.	If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports might see a throughput impact.
12028 (egress non-posted PRIV error)	The corresponding GPU NVLink traffic will be stalled, and subsequent GPU access will hang. The GPU driver on the guest VM will abort CUDA jobs with Xid 45.	Restart the guest VM	If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports will be affected.

continues on next page

Table 14.3 – continued from previous page

SXid & Error String	Guest VM Impact	Guest VM Recovery	Other Guest VM Impact
19084(AN1 Heartbeat Timeout Error)	This error is usually accompanied by a fatal SXid error that will affect the corresponding GPU NVLink traffic.	Reset all GPUs and all NVSwitches (refer to *GPU VM System Reset Capabilities and Limitations*).	If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports will be affected.
22013(Minion Link DL-REQ interrupt	This SXid can be safely ignored.	Not Applicable.	No Impact.
20012	This error might occur as the result of a broken/inconsistent connection or uncoordinated shutdown.	If this issue was not due to an uncoordinated shutdown, check the link mechanical connections.	No impact if error is confined to one GPU.

14.5. Fatal NVSwitch SXid Errors

Table 14.4 lists potential NVSwitch fatal SXid errors that might occur in the field. The hypervisor must track these SXid source ports (NVLink) to determine whether the error occurred on an NVSwitch trunk port or NVSwitch access port. The fatal SXid will be propagated to the GPU as Xid 74 when applicable. The following recommended actions apply to all SXids in Table 14.4 unless otherwise noted.

- If the error occurred on an NVSwitch access port, the impact will be limited to the corresponding guest VM.

To recover, shut down the guest VM.

- If the errors occurred on an NVSwitch trunk port, to reset the trunk ports and recover, shut down the guest VM partitions that are crossing the trunk port.

The partitions can be recreated. Currently, the partitions that are using NVSwitch trunk ports are the 16x GPU partition and the 8x GPU partitions with four GPUs per baseboard.

Table 14.4 Potential Fatal NVSwitch SXid Errors

SXid	SXid Error String
11001	ingress invalid command
11009	ingress invalid VCSet
11013	ingress header DBE
11018	ingress RID DBE

continues on next page

Table 14.4 – continued from previous page

SXid	SXid Error String
11019	ingress RLAN DBE
11020	ingress control parity
12001	egress crossbar overflow
12002	egress packet route
12022	egress input ECC DBE error
12024	egress output ECC DBE error
12025	egress credit overflow
12026	egress destination request ID error
12027	egress destination response ID error
12030	egress control parity error
12031	egress credit parity error
12032	egress flit type mismatch
14017	TS ATO timeout
15001	route buffer over/underflow
15006	route transdone over/underflow
15009	route GLT DBE
15010	route parity
15012	route incoming DBE
15013	route credit parity
19047	NCISOC HDR ECC DBE Error
19048	NCISOC DAT ECC DBE Error
19054	HDR RAM ECC DBE Error
19056	DAT0 RAM ECC DBE Error
19058	DAT1 RAM ECC DBE Error
19060	CREQ RAM HDR ECC DBE Error
19061	CREQ RAM DAT ECC DBE Error
19063	Response RAM HDR ECC DBE Error
19064	Response RAM DAT ECC DBE Error
19066	COM RAM HDR ECC DBE Error
19067	COM RAM DAT ECC DBE Error
19069	RSP1 RAM HDR ECC DBE Error
19070	RSP1 RAM DAT ECC DBE Error

continues on next page

Table 14.4 – continued from previous page

SXid	SXid Error String
20034	LTSSM Fault Up Guest VM impact: This SXid is triggered when the associated link has gone down from active. This interrupt is usually associated with other NVLink errors. Guest VM recovery: In an A100 system, restart the VM. In an H100 system, reset the GPU (refer to *GPU VM System Reset Capabilities and Limitations*). If the issue persists, report the GPU issues. Other guest VM impact: No impact if error is confined to one GPU.
22012	Minion Link NA interrupt
24004	sourcetrack TCEN0 crubmstore DBE
24005	sourcetrack TCEN0 TD crubmstore DBE
24006	sourcetrack TCEN1 crubmstore DBE
24007	sourcetrack timeout error

14.6. Always Fatal NVSwitch SXid Errors

Table 14.5 lists the potential NVSwitch fatal SXid errors that are always fatal to the entire fabric/system. After an always fatal SXid error has occurred, the guest VM partitions need to be shut down and one of the following tasks must occur:

- The host needs to be restarted.
- After the NVSwitches and GPUs are SBRed, restart the service VM.

Table 14.5 Always Fatal NVSwitch SXid Errors

SXid	SXid Error String
12020	egress sequence ID error
22003	Minion Halt
22011	Minion exterror
23001	ingress SRC-VC buffer overflow
23002	ingress SRC-VC buffer underflow
23003	egress DST-VC credit overflow
23004	egress DST-VC credit underflow
23005	ingress packet burst error
23006	ingress packet sticky error
23007	possible bubbles at ingress
23008	ingress packet invalid dst error
23009	ingress packet parity error

continues on next page

Table 14.5 – continued from previous page

SXid	SXid Error String
23010	ingress SRC-VC buffer overflow
23011	ingress SRC-VC buffer underflow
23012	egress DST-VC credit overflow
23013	egress DST-VC credit underflow
23014	ingress packet burst error
23015	ingress packet sticky error
23016	possible bubbles at ingress
23017	ingress credit parity error

14.7. Other Notable NVSwitch SXid Errors

Table 14.6 provides additional SXid errors that might affect the overall fabric/system.

Table 14.6 Other Notable NVSwitch SXid Errors

SXid	SXid Error String	Comments/Description
10001	Host_priv_error	The errors are not fatal to the fabric/system, but they might be followed by other fatal events.
10002	Host_priv_timeout	The errors are not fatal to the fabric/system, but they might be followed by other fatal events.
10003	Host_unhandled_interru	This SXid error is never expected to occur. This error is fatal to the fabric/system. To recover, reset all GPUs and NVSwitches (refer to *GPU VM System Reset Capabilities and Limitations*). If the error is observed on a Trunk port, the partitions that use the NVSwitch trunk ports will be affected.
10004	Host_thermal_event_sta	Related to thermal events, which are not directly fatal to the fabric/system, but they indicate that system cooling might be insufficient. This error might force the specified NVSwitch Links to enter power saving mode (Single Lane Mode) and impact over the NVLink throughput.
10005	Host_thermal_event_en	Related to thermal events, which are not directly fatal to the fabric/system, but they do indicate that system cooling might be insufficient.

For the comprehensive list of other NVSwitch SXid errors, go to https://github.com/NVIDIA/open-gpu-kernel-modules/blob/4397463e738d2d90aa1164cc5948e723701f7b53/src/common/nvswitch/interface/ctrl_dev_nvswitch.h.

14.8. High Availability Mode Comparison

The following are high availability configuration options:

- `TRUNK_LINK_FAILURE_MODE`
  High Availability Mode options when there is a Trunk Link Failure (NVSwitch to NVSwitch NVLink failure).
- `NVSWITCH_FAILURE_MODE`
  High Availability Mode options when there is a NVSwitch failure or an NVSwitch is excluded.
- `ACCESS_LINK_FAILURE_MODE`
  High Availability Mode options when there is an Access Link (GPU to NVSwitch NVLink failure) Failure
- `ABORT_CUDA_JOBS_ON_FM_EXIT`
  Control running CUDA jobs behavior when FM service is stopped or terminated.

The behavior of each configuration option depends on the platform.

14.8.1. DGX A100/HGX A100

14.8.1.1 A100 Bare Metal Configuration or Full Virtualization Passthrough

`TRUNK_LINK_FAILURE_MODE`

- `0`: Exit FM and leave the system/NVLinks uninitialized.
- 1: Disable the NVSwitch and its peer NVSwitch, which reduces NVLink P2P bandwidth.

`NVSWITCH_FAILURE_MODE`

- `0`: Abort Fabric Manager.
- 1: Disable the NVSwitch and its peer NVSwitch, which reduces P2P bandwidth.

`ACCESS_LINK_FAILURE_MODE`

- `0`: Remove the GPU with the Access NVLink failure from NVLink P2P capability.
- 1: Disable the NVSwitch and its peer NVSwitch, which reduces NVLink P2P bandwidth.

`ABORT_CUDA_JOBS_ON_FM_EXIT`

- `0`: Do not abort running CUDA jobs when FM exits. However, new CUDA job launches will fail.
- 1: Abort all running CUDA jobs when Fabric Manager exits.

14.8.1.2 A100 Shared NVSwitch or vGPU-based Multitenancy

`TRUNK_LINK_FAILURE_MODE`

- `0`: Remove partitions that are using the Trunk NVLinks.
- 1: Disable the NVSwitch and its peer NVSwitch. All partitions will be available but with reduced NVLink P2P bandwidth.

`NVSWITCH_FAILURE_MODE`

- `0`: Abort Fabric Manager.
- 1: Disable the NVSwitch and its peer NVSwitch, which reduces P2P bandwidth.

`ACCESS_LINK_FAILURE_MODE`

- `0`: Disable partitions that are using the Access Link failed GPU
- 1: Disable the NVSwitch and its peer NVSwitch.

`ABORT_CUDA_JOBS_ON_FM_EXIT`

- `0`: Do not abort running CUDA jobs when FM exits. However, new CUDA job launches will fail.
- 1: Abort all running CUDA jobs when Fabric Manager exits.

14.8.2. DGX H100/HGX H100

`TRUNK_LINK_FAILURE_MODE`

- H100 systems ignore this setting. H100 systems don't have NVLink trunk links.

`NVSWITCH_FAILURE_MODE`

- H100 systems ignore this setting. If an NVLink Switch is unavailable GPUs will fail to complete registration with the NVLink fabirc and CUDA application launch fails.

`ACCESS_LINK_FAILURE_MODE`

- H100 systems ignore this setting. On access link failure `nvidia-smi` shows an inactive NVLink.

`ABORT_CUDA_JOBS_ON_FM_EXIT`

- H100 systems ignore this setting. Running CUDA jobs continue to run and new jobs fail to launch. If persistence mode is enabled, new CUDA jobs will launch. After a GPU reset, CUDA job launches will fail even if the GPUs have persistence mode enabled.

14.9. GPU VM System Reset Capabilities and Limitations

Here is some information from the nvidia-smi manpage about reset capabilities:

- Used to trigger a reset of one or more GPUs.
- Can be used to clear the GPU hardware and software states in situations that requires a machine reboot.
- Typically useful when a double-bit ECC error occurs.
- The `-i` option can be used to target one or more specific devices.
  - Without this option, all GPUs are reset, and root is required.
  - Applications, such as CUDA, graphics applications like X server, monitoring applications like another instance of nvidia-smi) cannot use these devices.

14.9.1. Direct NVLink Connect

In bare metal deployments, a GPU can be individually reset, and all GPUs can be reset without specifying a device.

GPU resets aren't supported with full passthrough virtualization, with all GPUs in the same VM. The VM must be restarted.

14.9.2. Ampere and NVSwitch

If the FM state is `Running`:

- GPUs can be individually reset. NVSwitch links are automatically reset by FM.
- GPU resets aren't supported with full passthrough virtualization, with all GPUs in the same VM. The VM must be restarted.
- If the GPUs and NVSwitches are in different VMs restart the GPU and the service VM resets the NVSwitch links.

If the FM state is `Not Running`:

- GPUs can't be individually reset. Reset all GPUs and NVSwitches.
- GPU resets aren't supported with full passthrough virtualization, with all GPUs in the same VM. The VM must be restarted.
- If the GPUs and NVSwitches are in different VMs restart the GPU and the service VM resets the NVSwitch links.

14.9.3. Hopper and NVSwitch

In bare metal deployments, a GPU can be individually reset. All GPUs and NVSwitches can be reset without specifying a device.

Both full fassthrough and dhared virtualization environments reset depends on the hypervisor permissions. A VM restart will restart the GPUs.

Copyright