Fabric Manager for NVIDIA NVSwitch Systems

Virtualization/High Availability Modes

User Guide

DU-09883-001_v1.3 | October 2023

Document History

DU-09883-001_v1.3

Version	Date	Authors	Description of Change
0.1	October 25, 2019	SB	Initial Beta Release
0.2	March 23, 2020	SB	Updated error handling and bare metal mode
0.3	May 11, 2020	YL	Updated Shared NVSwitch APIs section with new API information
0.4	July 7, 2020	SB	Updated multi-instance GPU (MIG) interoperability and high availability details.
0.5	July 17, 2020	SB	Updated running as non-root instructions
0.6	August 03, 2020	SB	Updated installation instructions based on CUDA repo and updated SXid error details
0.7	January 26, 2021	GT, CC	Updated with NVIDIA Virtual GPU (vGPU) multitenancy virtualization mode
0.8	March 19, 2021	SB	Updated High Availability section to reflect recent GPU excluded option changes.
0.9	October 19, 2022	YL, SB, GT	Updated with NVIDIA® DGX™ H100 and NVIDIA HGX™ H100
1.0	Jan 20, 2023	YL	Updated GPU Module ID for DGX H100 and NVIDIA HGX H100
1.1	June 23, 2023	SB	• Updated with log rotation options. • Updated NVIDIA HGX H100 NVIDIA NVLink® topology information. • Added support language for NVIDIA HGX A800 and NVIDIA HGX H800.
1.2	July 7, 2023	EK, PKS	• Updated D.4 Non-Fatal NVSwitch SXid Errors • Updated D. Fatal NVSwitch SXid Errors • Added D.9 GPU/VM/System Reset Capabilities and Limitations
1.3	October 3, 2023	YL, SB	• Updated Shared NVSwitch 2 GPU partitions for DGX H100 and HGX H100 • Updated various FM package details, FM Service restart consideration for DGX H100 and HGX H100, Service VM memory requirements.

Table of Contents

Chapter 1. Overview

1.1 Introduction

As deep learning neural networks become more sophisticated, their size and complexity continue to expand. The result is 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.

1.2 NVSwitch-Based Systems

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

NVIDIA DGX-2™ and NVIDIA HGX-2 systems consists 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. Two GPU baseboards are connected to build a 16-GPU system. Between the two GPU baseboards, the only NVLink connections are between NVSwitches, where each NVSwitch from one GPU baseboard is connected to one NVSwitch on the second GPU baseboard for a total of eight NVLink connections.

The NVIDIA 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 has depreciated the support to connect two GPU baseboard using NVLink.

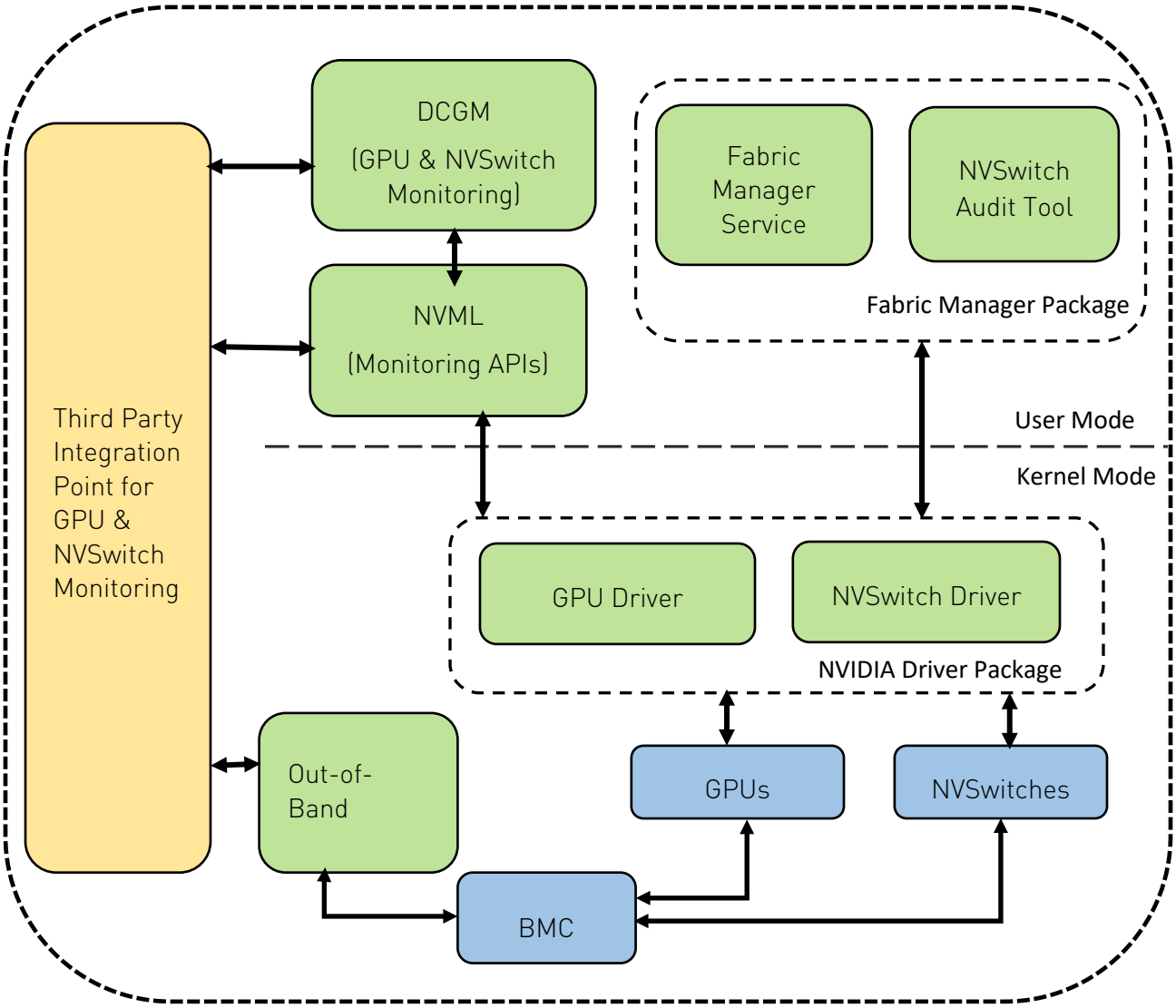
1.3 Terminology

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
VF	Virtual Function
GFID	GPU Function Identification
Partition	A collection of GPUs which are allowed to perform NVLink Peer-to-Peer Communication among themselves
ALI	Autonomous Link Initialization

1.4 NVSwitch Core Software Stack

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. NVSwitch core software stack



1.5 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 has the following responsibilities:

- Configures routing among NVSwitch ports.
- 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 (the first and second generation NVSwitch-based systems), FM also has the following additional responsibilities:
- Coordinate with the NVSwitch driver to train NVSwitch to NVSwitch NVLink interconnects.
- Coordinate with the GPU driver to initialize and train NVSwitch to GPU NVLink interconnects.

This document provides an overview of various FM features and is intended for system administrators and individual users of NVSwitch-based server systems.

Chapter 2. Getting Started with Fabric Manager

2.1 Basic Components

This section provides information about the basic components in FM.

2.1.1 Fabric Manager Service

The core component of FM is implemented as a standalone executable that runs as a Unix Daemon process. The FM installation package will install the required core components and registers the daemon as a system service called `nvidia-fabricmanager`.

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 Fabric Manager service to query/activate/deactivate GPU partitions when FM is running in Shared NVSwitch and vGPU multi-tenancy modes. All these SDK components are installed through a separate development package. For more information, refer to "Shared NVSwitch Virtualization Model" on page 44 and "vGPU Virtualization Model" on page 56.

2.2 NVSwitch and NVLink Initialization

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 systems that have ALI support, 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
```

Fabric Manager plays a critical role in the functionality of NVSwitch-based systems that are typically initiated during a system boot or a workload activation. Restarting the service intermittently is unnecessary; but if such a restart is necessary because of workflow requirements, or as part of a GPU reset operation, complete the following procedure for DGX H100 and NVIDIA HGX H100 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 Fabric Manager Service by terminating the Fabric Manager service.
3. Perform GPU Reset by issuing the *nvidia-smi -r* command and executing a GPU reset by.
4. Start Fabric Manager Service Againby restaring the Fabric Manager service and restoring its functionality.
5. Resume Stopped Services by restarting any services that were halted in step 1, such as DCGM or other GPU-related services.
6. Launch 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.

2.3 Supported Platforms

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

2.3.1 Hardware Architectures

- x86_64
- AArch64

2.3.2 NVIDIA 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.

Note: Unless specified, the steps for NVIDIA HGX A800 and NVIDIA HGX H800 are the same as the steps NVIDIA HGX A100 and NVIDIA HGX H100. The only difference is that the number of GPU NVLinks will defer depending on the actual platform.

2.3.3 OS Environment

FM is supported on the following major Linux OS distributions:

- RHEL/CentOS 7.x and RHEL/CentOS 8.x
- Ubuntu 18.04.x, and Ubuntu 20.04.x, and Ubuntu 22.04.x

2.4 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 configuration” on page 20
- “Full Passthrough Virtualized Configurations” on page 72.

- "Shared NVSwitch Virtualization Configurations" on page 73
- "Bare Metal and vGPU Configurations" on page 72

2.5 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.

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.

2.6 Installation

2.6.1 On NVSwitch-Based DGX Server Systems

The FM service is preinstalled in all the NVSwitch-based DGX-2, DGX A100, and DGXH100 systems as part of the supported DGX OS package. The service is enabled and started on OS boot.

2.6.2 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. 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.

Each release version of Fabric Manager comprises the following packages:

- nvidia-fabricmanager-<version>

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

- nvidia-fabricmanager-devel-<version>

  The "devel" package encompasses 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.

By splitting the functionality into these packages, users can selectively install the components most relevant to their needs.

After the package manager network repositories are set up, use the following distro-specific FM installation commands:

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

- For Debian and Ubuntu based OS distributions:
```
sudo apt-get install cuda-drivers-fabricmanager-<driver-branch>
```
- For Red Hat Enterprise Linux 8 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: On NVSwitch-based NVIDIA HGX systems, **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.

2.7 Managing the Fabric Manager Service

2.7.1 Start Fabric Manager

- To start FM, run the following command:
```
# For Linux based OS distributions
sudo systemctl start nvidia-fabricmanager
```

2.7.2 Stop Fabric Manager

- To stop FM, run the following command:
```
# For Linux based OS distributions
sudo systemctl stop nvidia-fabricmanager
```

2.7.3 Check the Fabric Manager Status

- To check the FM status, run the following command:

```
# For Linux based OS distributions
sudo systemctl status nvidia-fabricmanager
```

2.7.4 Enable the Fabric Manager Service to Auto Start at Boot

- To enable the FM service to start automatically at boot, run the following command:

```
# For Linux based OS distributions
sudo systemctl enable nvidia-fabricmanager
```

2.7.5 Disable the Fabric Manager Service Auto Start at Boot

- To disable the FM service to start automatically at boot, run the following command:

```
# For Linux based OS distributions
sudo systemctl disable nvidia-fabricmanager
```

2.7.6 Check Fabric Manager System Log Messages

- To check the FM system log messages, run the following command:

```
# For Linux based OS distributions
sudo journalctl -u nvidia-fabricmanager
```

2.8 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]

    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.
```

Most of the FM configurable parameters and options are specified through a text config file. FM installation will copy a default config file to a predefined location, and the file will be used by default. To use a different config file location, specify the same using 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, subsequent FM package update will provide options such as merge/keep/overwrite.

2.9 Fabric Manager Service File

2.9.1 On Linux-Based Systems

On Linux-based systems, the installation package will register the FM service using the following `systemd` service unit file. To change the FM service start-up options, modify this service unit file in the `/lib/systemd/system/nvidia-fabricmanager.service` directory.

```
[Unit]
Description=FabricManager service
After=network-online.target
Requires=network-online.target

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

ExecStart=/usr/bin/nv-fabricmanager -c /usr/share/nvidia/nvswitch/fabricmanager.cfg

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

2.10 Running Fabric Manager as Non-Root

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

1. If the FM instance is running, stop it.
2. FM requires access to the following directory/file, so adjust the corresponding directory/file access to the desired user/user group.
   - `/var/run/nvidia-fabricmanager`

     This option provides a fixed location to save runtime information.
   - `/var/log/`

     This option provides a configurable location to save FM log file.
   - `/usr/share/nvidia/nvswitch`

     This option provides a configurable location for 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.
   - The NVIDIA driver will create the following proc entry with default permission to root.
3. Change its read/write access to the desired user/user group.
   `/proc/driver/nvidia-nvlink/capabilities/fabric-mgmt`
4. FM also requires access to the following device node files.
   - On all the NVSwitch-based NVIDIA HGX systems:
     - `/dev/nvidia-nvlink`
     - `/dev/nvidia-nvswitchctl`
     - `/dev/nvidia-nvswitchX (one for each NVSwitch device)`
   - Here are the additional device node files on DGX-2 and NVIDIA HGX A100 systems:
     - `/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, with access permission for all users. If these device node files are created manually or outside of `nvidia-modprobe`, assign read/write access to the user/user group.
5. After the required permissions are assigned, manually start the FM process from the user/user group account.
6. The NVIDIA driver will create/recreate the above /proc entry during driver load, so repeat steps 1-6 on every driver reload or system boot.

When FM is configured as a `systemd` service, the system administrator must edit the FM service unit file to instruct `systemd` to run FM service from a specific user/group. This specific 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 permission are changed to desired user/user group before FM service starts at system boot time.

When FM is configured to run from a specific user/user group as specified above, the `nvswitch-audit` command line utility should be started from the same user/user group account.

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

2.11 Fabric Manager Config Options

The configurable parameters and options used by FM are specified through a text config file. The following section lists all those currently supported configurable parameters and options.

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

2.11.1 Logging Related Config Items

2.11.1.1 Setting the Log File Location and Name

- Config Item

```
LOG_FILE_NAME=<value>
```

- Supported/Possible Values

  The complete path/filename string (max length of 256) for the log.

- Default Value

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

2.11.1.2 Setting Desired 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
```

2.11.1.3 Setting Log File Append Behavior

- Config Item

```
LOG_APPEND_TO_LOG=<value>
```

- Supported/Possible Values:
  - 0 – No, don't append to the existing log file, instead overwrite the existing log file.
  - 1 – Yes, append to the existing log file every time Fabric Manager service is started.
- Default Value

```
LOG_APPEND_TO_LOG=1
```

2.11.1.4 Setting Log File Size

- Config Item

```
LOG_FILE_MAX_SIZE=<value>
```

- Supported/Possible Values
  - The desired max log file size 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
```

2.11.1.5 Redirect Logs to Syslog

- Config Item

```
LOG_USE_SYSLOG=<value>
```

- Supported/Possible Values
  - `0 - Use the specified log file for storing all the Fabric Manager logs`
  - `1 - Redirect all the Fabric Manager logs to syslog instead file-based logging.`
- Default Value

```
LOG_USE_SYSLOG=0
```

2.11.1.6 Rotation Settings

- Config Item

```
LOG_MAX_ROTATE_COUNT=<value>
```

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

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

    The combined Fabric Manager 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 NVIDIA recommends that you run the FM service with logging enabled at the INFO level for troubleshooting field issues.

2.11.2 Operating Mode Related Config Items

Note: This section of config items is applicable only to Shared NVSwitch and vGPU Multitenancy deployment.

2.11.2.1 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 multitenancy mode.

    For more information, refer to "Shared NVSwitch Virtualization Configurations" on page 73.
  - 2: Start FM in vGPU multitenancy mode.

    For more information, refer to "vGPU Virtualization Model" on page 56.

- Default Value

```
FABRIC_MODE=0
```

Note: The older SHARED_FABRIC_MODE configuration item is still supported, but we recommend that you use the FABRIC_MODE configuration item.

2.11.2.2 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 to the FM process and is provided to enable the Shared NVSwitch or vGPU multitenancy mode resiliency without modifying command-line arguments to the FM process. Refer to "Appendix C" on page 85 for more information on the FM resiliency flow.

- Default Value

```
FABRIC_MODE_RESTART=0
```

Note: The older `SHARED_FABRIC_MODE_RESTART` configuration item is still supported but we recommend that you use `FABRIC_MODE_RESTART` configuration item.

2.11.2.3 Fabric Manager API Interface

- Config Item

```
FM_CMD_BIND_INTERFACE =<value>
```

- Supported/Possible Values

  The network interface for the FM SDK/API to listen and for the Hypervisor to communicate with the running FM instance for the shared NVSwitch and vGPU multitenancy operations.

- Default Value

```
FM_CMD_BIND_INTERFACE=127.0.0.1
```

2.11.2.4 Fabric Manager API TCP Port

- Config Item

```
FM_CMD_PORT_NUMBER=<value>
```

- Supported/Possible Values

  The TCP port number for the FM SDK/API for Hypervisor to communicate with the running FM instance for Shared NVSwitch and vGPU multitenancy operations.

- Default Value

```
FM_CMD_PORT_NUMBER=6666
```

2.11.2.5 Fabric Manager Domain Socket Interface

- Config Item

```
FM_CMD_UNIX_SOCKET_PATH=<value>
```

- Supported/Possible Values

  The Unix domain socket path instead of the TCP/IP socket for the FM SDK/API to listen and to communicate with the running FM instance for the shared NVSwitch and vGPU multitenancy operations.

- Default Value

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

2.11.2.6 Fabric Manager State

- Config Item

```
STATE_FILE_NAME=<value>
```

- Supported/Possible Values

  Specify the filename to be used to save the FM states to 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
```

2.11.3 Miscellaneous Config Items

2.11.3.1 Prevent 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
```

2.11.3.2 Fabric Manager Communication Socket Interface

- Config Item

```
BIND_INTERFACE_IP=<value>
```

- Supported/Possible Values

  Network interface to listen for the FM internal communication/IPC, and this value should be a valid IPv4 address.

- Default Value

```
BIND_INTERFACE_IP=127.0.0.1
```

2.11.3.3 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
```

2.11.3.4 Unix Domain Socket for Fabric Manager Communication

- Config Item

```
UNIX_SOCKET_PATH=<value>
```

- Supported/Possible Values

  Use Unix Domain socket instead of 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>
```

2.11.3.5 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
```

2.11.4 High Availability Mode-Related Config Items

2.11.4.1 Control Fabric Manager Behavior on Initialization Failure

- Config Item

```
FM_STAY_RESIDENT_ON_FAILURES=<value>
```

- Supported/Possible Values

  - 0: The FM service will terminate on errors such as, NVSwitch and GPU config failure, typical software errors, and so on.
  - 1: The FM service will stay running on errors such as, NVSwitch and GPU config failure, 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
```

2.11.4.2 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” on page 62 for more information about supported values and behavior.

- Default Value

```
ACCESS_LINK_FAILURE_MODE=0
```

2.11.4.3 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 Refer to "Supported High Availability Modes" on page 62 for more information about supported values and behavior.

- Default Value

```
TRUNK_LINK_FAILURE_MODE=0
```

2.11.4.4 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 Refer to "Supported High Availability Modes" on page 62 for more information about supported values and behavior.

- Default Value

```
NVSWITCH_FAILURE_MODE=0
```

2.11.4.5 CUDA Jobs Behavior When the Fabric Manager Service is Stopped or 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. However, a new CUDA job launch will fail with `cudaErrorSystemNotReady` error.
  - `1`: Abort all running CUDA jobs when the FM service is stopped or exits. Also, a new CUDA job launch will fail with `cudaErrorSystemNotReady` error.

Note: This is not effective on DGX H100 and NVIDIA HGX H100 NVSwitch based systems. Also, This config option is applicable to only bare metal and full passthrough virtualization models.

- Default Value

```
ABORT_CUDA_JOBS_ON_FM_EXIT=1
```

Chapter 3. Bare Metal Mode

3.1 Introduction

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.

3.2 Fabric Manager Installation

3.2.1 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.

3.2.2 On NVSwitch-Based NVIDIA HGX Server Systems

To configure NVSwitch-based NVIDIA HGX systems for bare metal mode, system administrators must install the NVIDIA FM Package, which is the same version as the driver package.

The driver package is for NVIDIA Data Center GPUs (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.

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

3.3 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, 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 vs fon-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.

3.3.1 NVSwitch SXid Errors

3.3.1.1 NVSwitch Non-Fatal SXid Errors

NVSwitch 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 exact error, CUDA jobs might experience issues such as a performance drop, no forward progress for brief time, and so on.

3.3.1.2 NVSwitch Fatal SXid Errors

When a fatal SXid error is reported on a NVSwitch port, which is connected between a GPU and a 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) via the NVIDIA System Management Interface (`nvidia-smi`) command line utility.

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

   When a fatal SXid error is reported on a NVSwitch port, which connects two GPU baseboards, FM will abort all the running CUDA jobs and prevent any 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. The system administrator must use the following recovery procedure to clear the error state and subsequent successful CUDA job launch:
   a. Reset all the GPUs and NVSwitches.
   b. Stop the FM service.
   c. Stop all the applications that are using the GPU.
   d. Reset all the GPU and NVSwitches using the `nvidia-smi` command line utility with the `-r` or —the `--gpu-reset` option.
   e. **Do not** use the `-i` or the `-id` options.
   f. After the reset operation is complete, start the FM service again.

If the problem persists, reboot or power cycle the system.

3.3.2 GPU Xid Errors

GPU Xid messages indicate that a general GPU error occurred. The messages can indicate a hardware problem, an NVIDIA software problem, or 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 the previous section for additional troubleshooting.

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.

3.4 Interoperability With MIG

Multi-Instance GPUs (MIGs) partition an NVIDIA A100 or H100 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, 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 can cooperate with GPU MIG instances. Also, on these systems, to successfully restore GPU NVLink peer-to-peer capability after the MIG mode is disabled, 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 retrained when MIG mode is disabled. However, on DGX H100 and NVIDIA HGX H100 systems, GPU NVLinks will stay active during MIG mode.

Chapter 4. Virtualization Models

4.1 Introduction

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 instead be controlled by the trusted software.

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

4.2 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.
- 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.
- 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 5. 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 runs in the shared NVSwitch and vGPU multi-tenant modes to query/activate/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 provided in this user guide, this package must be installed.

5.1 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;
    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
```

```
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
fmUnsupportedFabricPartitionList_version1
```

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

5.2 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
```

5.3 Shutting Down the Fabric Manager API interface

The following method is used to shut down the FM API interface library, and the remote connections that were established through `fmConnect()` will also be shut down.

```
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.
```

5.4 Connect to Running the Fabric Manager Instance

To connect to a running instance of FM. the FM 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
```

5.5 Disconnect from Running 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
```

5.6 Getting 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
       Pointer to fmFabricPartitionList_t structure. On success, the list of
supported (static) partition information will be populated in this structure.

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
       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
```

5.7 Activate 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
        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.
```

5.8 Activate a GPU Partition with Virtual Functions

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

5.9 Deactivate 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
```

5.10 Set Activated Partition List after a Fabric Manager Restart

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
```

```
      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
```

5.11 Get the NVLink Failed Devices

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 and NVIDIA HGX H100 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.

5.12 Get 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
```

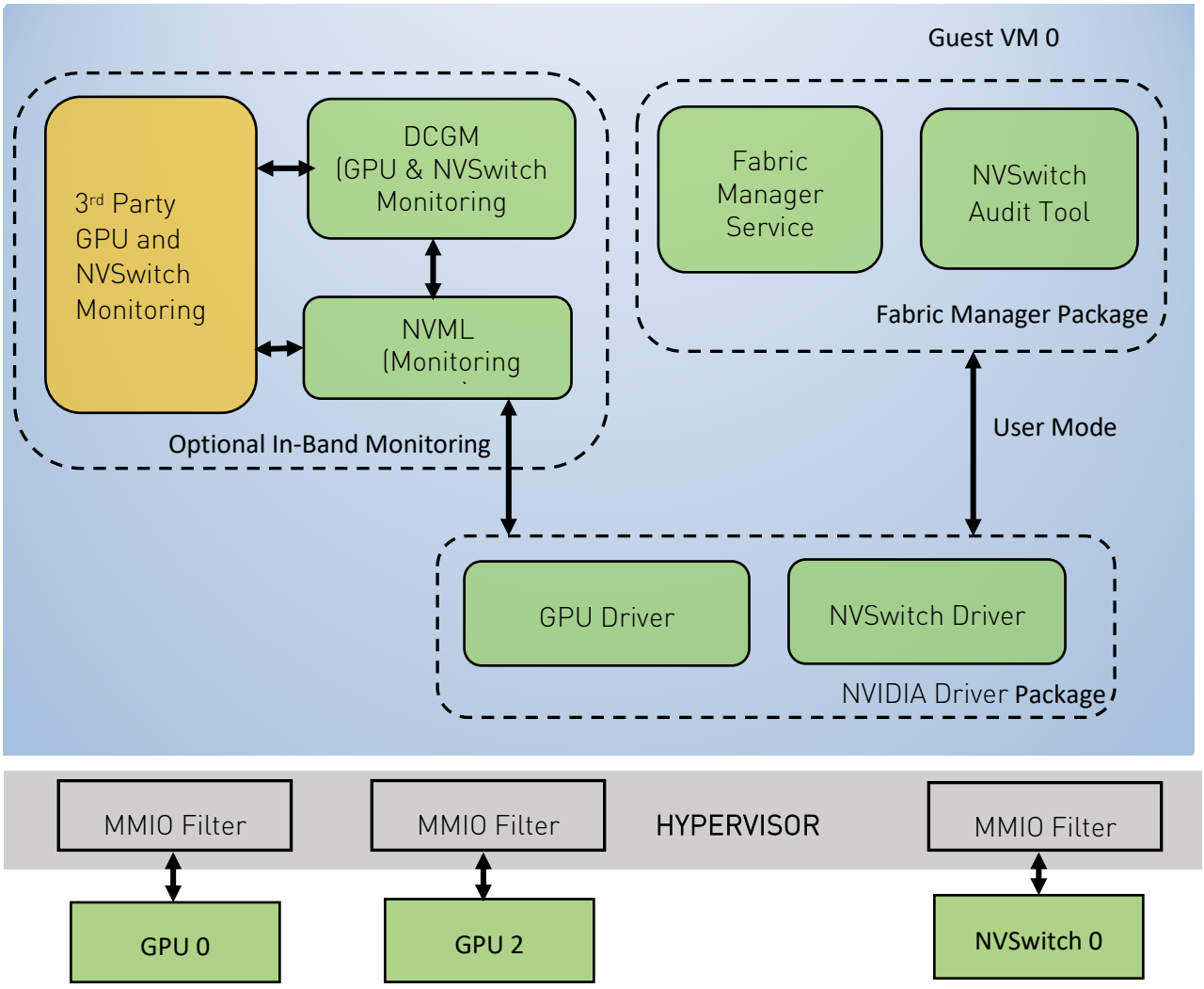
Note: On DGX H100 and NVIDIA HGX H100 systems, this API will always return `FM_ST_SUCCESS` with an empty list of unsupported partition.

Chapter 6. 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 used 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 8-GPU VM, a 4-GPU VM, and two 2-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 2. Software Stack in a Two-GPU Virtual Machine (A Full Passthrough Model)



6.1 Supported Virtual Machine Configurations

Table 1. DGX-2 and NVIDIA HGX-2 Systems Device Assignment

Number of GPUs assigned to a VM	Number of NVSwitches assigned to a VM	Enabled NVLink Interconnects Per GPU	Enabled NVLink Interconnects Per NVSwitch	Constraints
16	12	6 out of 6	16 out of 18	None
8	6	6 out of 6	8 out of 18	One set of eight GPUs from each GPU Baseboard
4	3	3 out of 6	4 out of 18	Two sets of four GPUs from each GPU Baseboard
2	1	1 out of 6	2 out of 18	Four sets of two GPUs from each GPU Baseboard
1	0	0 out of 6	0 out of 18	None

Table 2. Two DGX A100 and NVIDIA HGX A100 Systems Device Assignment

Number of GPUs assigned to a VM	Number of NVSwitches assigned to a VM	Enabled NVLink Interconnects Per GPU	Enabled NVLink Interconnects Per NVSwitch	Constraints
16	12	12 out of 12	32 out of 36	None
8	6	12 out of 12	16 out of 36	One set of eight GPUs from each GPU Baseboard.
4	3	6 out of 12	6 out of 36	Two sets of four GPUs from each GPU Baseboard.
2	1	2 out of 12	4 out of 36	Four sets of two GPUs from each GPU Baseboard.
1	0	0 out of 12	0 out of 36	None

Table 3. DGX H100 and NVIDIA HGX H100 Systems Device Assignment

Number of GPUs assigned to a VM	Number of NVSwitches assigned to a VM	Enabled NVLink Interconnects Per GPU	Enabled NVLink Interconnects Per NVSwitch	Constraints
8	4	18 out of 18	32 out of 64 for two NVSwitches. 40 out of 64 for other two NVSwitches.	None
1	0	0 out of 18	0 out of 64	Need to disable GPU NVLinks.

6.2 Virtual Machines 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.

6.3 Virtual Machines 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 GPU baseboards are disabled.

6.4 Virtual Machines with Four GPUS

If this configuration is supported, each VM gets four GPUs and three switches. As specified in Table 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.

6.5 Virtual Machines 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.

6.6 Virtual Machine 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.

6.7 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, for example, NVIDIA Driver and FM to configure switches and NVLink connections.

6.8 Hypervisor Sequences

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

1. Start the guest VM.
   a. Select an unused partition of GPUs and switches.
   b. Reset the GPUs and switches in the partition.
   c. Block the disabled NVLink connections on each GPU by performing the specified MMIO configuration.
   d. Block the disabled NVLink connections on each switch by configuring the MMIO intercept.
   e. Avoid configuring any IOMMU mappings between GPUs and switches.
      > Switches cannot be accessible by any other PCIe device that the guest VM controls.

        This way, the switches cannot bypass the MMIO restrictions 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 NVIDIAGPUDirect™ RDMA.
   f. To avoid additional GPU resets, start the guest VM.

2. Shut down the guest VM.
   a. Shut down the Guest VM as usual.
   b. Reset the GPUs and switches that belong to the partition.
3. Reboot the Guest VM.
   a. Repeat steps 1a to 1f, but this time, the partition has already been selected.

6.9 Monitoring Errors

The NVSwitch, GPU and NVLink errors are visible to guest VMs. If you want the hypervisor to monitor the same items, use one of the following methods:

- 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.

6.10 Limitations

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

Chapter 7. Shared NVSwitch Virtualization Model

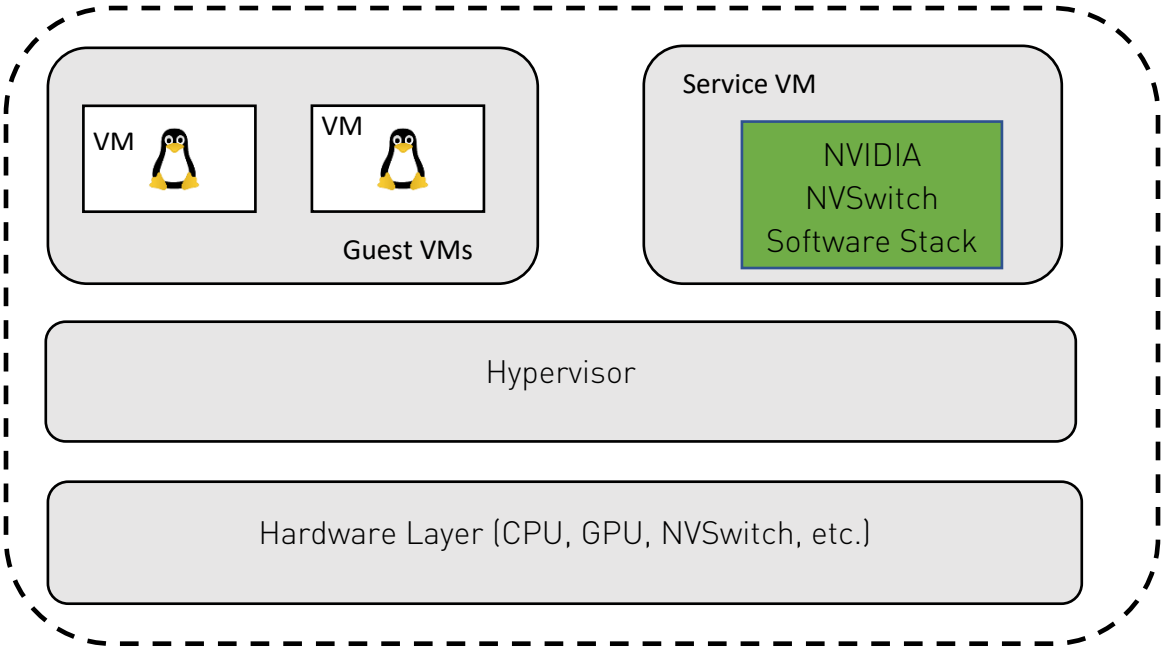
The shared NVSwitch virtualization model additionally extends the GPU Passthrough model by managing the switches from one Service VM that runs permanently. The GPUs are made accessible to the Service VM for link training and reassigned to the guest VMs.

Sharing switches among the guest VMs allows FM to enable more NVLink connections for 2 and 4 GPU VMs that observe reduced bandwidth in GPU Passthrough model.

7.1 Software Stack

The software stack required for NVSwitch management runs in a Service VM.

Figure 3. 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 has the following features:

- Provides an interface to query the available GPU VM partitions (groupings) and corresponding GPU information.
- Provides 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.
- Provides an interface to deactivate GPU VM partitions, which involves the following:
  - Untrain (power down) the NVSwitch to NVSwitch NVLink interconnects.
  - 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.

7.2 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 and NVIDIA HGX H800 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 and NVIDIA HGX H800 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 solutionbecause 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.

7.3 Preparing the Service Virtual Machine

7.3.1 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" on page 7 for more information.

7.3.2 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, NVIDIA internally uses the following hardware resources for 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.

Table 4. Service VM Resources

Resource	Quantity/Size
vCPU	2
System Memory	4 GB

7.3.3 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.
- NVIDIA Fabric Manager Package (same version as the Driver package).

7.3.4 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.

7.3.5 Other NVIDIA Software Packages

In Shared NVSwitch mode, no process or entity, other than FM, should open and interact with GPUs while activating or deactivating the partition. Also, all the GPU health check applications must be started after activating the partition and must be closed before unbinding the GPUs from `nvidia.ko`.

7.4 FM Shared Library and APIs

Refer to "Fabric Manager SDK" on page 28 for the list of the APIs that manage a shared NVSwitch partition life cycle.

7.4.1 Sample Code

The following code snippet shows how to query supported partitions, activate, or deactivate partitions, and so on by using the FM APIs mentioned in "Fabric Manager SDK" on page 28.

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

#include "nv_fm_agent.h"

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
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 {
        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
```

7.5 Fabric Manager Resiliency

Refer to "Resiliency" on page 85 for more information about FM resiliency in Shared Virtualization mode.

7.6 Service Virtual Machine Life Cycle Management

7.6.1 GPU Partitions

Refer to "GPU Partitions" on page 81 for the default, and all supported partitions, for the shared NVSwitch virtualization mode.

7.6.2 Building GPUs to Partition Mapping

The FM instance that runs on the Service VM and 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.

7.6.3 Booting the Service Virtual Machine

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

- Assign/plug the available NVSwitches as PCI passthrough devices to the Service VM without MMIO filtering.
- 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.
- Start and wait for the FM to fully initialize the GPUs and switches.

  The FM APIs will return `FM_ST_NOT_CONFIGURED` until the fabric is initialized and ready.
- Query the list of currently supported VM partitions and build the available guest VM combinations accordingly.
- Deassign/unplug the GPUs from the Service VM for NVIDIA HGX-2 and NVIDIA HGX A100 systems.

7.6.4 Restarting the Service Virtual Machine

The NVSwitch kernel software stack gets loaded and initializes the NVSwitches and GPUs as part of the Service VM booting, so restarting Service VM will affect currently activated GPU partitions. The hypervisor must follow the same procedure and steps as described in “Booting the Service ” on page 51.

7.6.5 Shutdown the Service

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.

7.7 Guest Virtual Machine Life Cycle Management

7.7.1 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.

7.7.2 Starting a Guest Virtual Machine

To start a guest VM, the hypervisor must complete one of the following procedures:

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:
  a. Select one of the supported GPU partitions based on guest VM GPU demand.
  b. Identify the corresponding GPUs using the GPU Physical ID to PCI BDF mapping.
  c. Reset (SBR) the selected GPUs.
  d. Hot plug the selected GPUs to the Service VM.
  e. Ensure that the GPUs are bound to `nvidia.ko`.
  f. Request FM to activate the requested GPU partition using the `fmActivateFabricPartition()` API.
  g. Unbind the GPUs from `nvidia.ko`.
  h. Hot unplug the GPUs from Service VM (if needed).
  i. Start the guest VM without resetting the GPUs.

Note: If the GPUs get 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 systems:

a. Select one of the supported GPU partitions based on guest VM GPU demand.
b. Identify the corresponding GPUs by using the GPU Physical ID to PCI BDF mapping.
c. Request FM to activate the requested GPU partition using the `fmActivateFabricPartition() API`.
d. Start the guest VM.

7.7.3 Shutting Down a Guest Virtual Machine

To shut down a guest VM, the hypervisor must do the following.

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

- On NVIDIA HGX-2 and NVIDIA HGX A100 Systems:
  a. Shut down the guest VM but, to avoid any NVSwitch side NVLink errors, avoid GPU resets (
  b. Use the `fmDeactivateFabricPartition ()` API and request FM to deactivate the specific GPU partition.
  c. Reset the GPUs after the deactivation partition request has completed.
- On NVIDIA HGX H100 Systems:
  a. Shut down the guest VM.
  b. Use the `fmDeactivateFabricPartition ()` API and request FM to deactivate the specific GPU partition.
  c. If the guest VM shutdown process is not completing an explicit GPU reset, reset the GPUs after the deactivate partition request has completed.

7.7.4 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 52 and “Shutting Down a Guest Virtual Machine” on page 53.

7.7.5 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.

```
root@host1-servicevm:~# ./nvswitch-audit -h
NVIDIA NVSwitch audit tool
Reads NVSwitch hardware tables and outputs the current number of
NVlink connections between each pair of GPUs

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 8-GPU VM partition on an NVIDIA HGX A100 is activated.

```
root@host1-servicevm:~# ./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
```

7.8 Error Handling

Refer to “Error Handling” on page 88 for information about FM initialization, partition, and hardware specific errors and their handling.

7.8.1 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 52 and “Shutting Down a Guest Virtual Machine” on page 53.

7.8.2 Handling a Service Virtual Machine Crash

When a Service VM experiences a kernel crash, the remaining activated guest VMs will continue as expected. However, the VM partition activation and deactivation life cycle will be affected. To recover from this state, a Service VM restart, or a reboot is required.

7.9 Interoperability With a Multi-Instance GPU

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

7.9.1 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.

7.9.2 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 8. vGPU Virtualization Model

The vGPU virtualization model supports VF passthrough by enabling SR-IOV functionality in all the supported GPUs and assigning a specific 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 vGPU Software User Guide for more information about the supported vGPU functionality, features, and configurations.

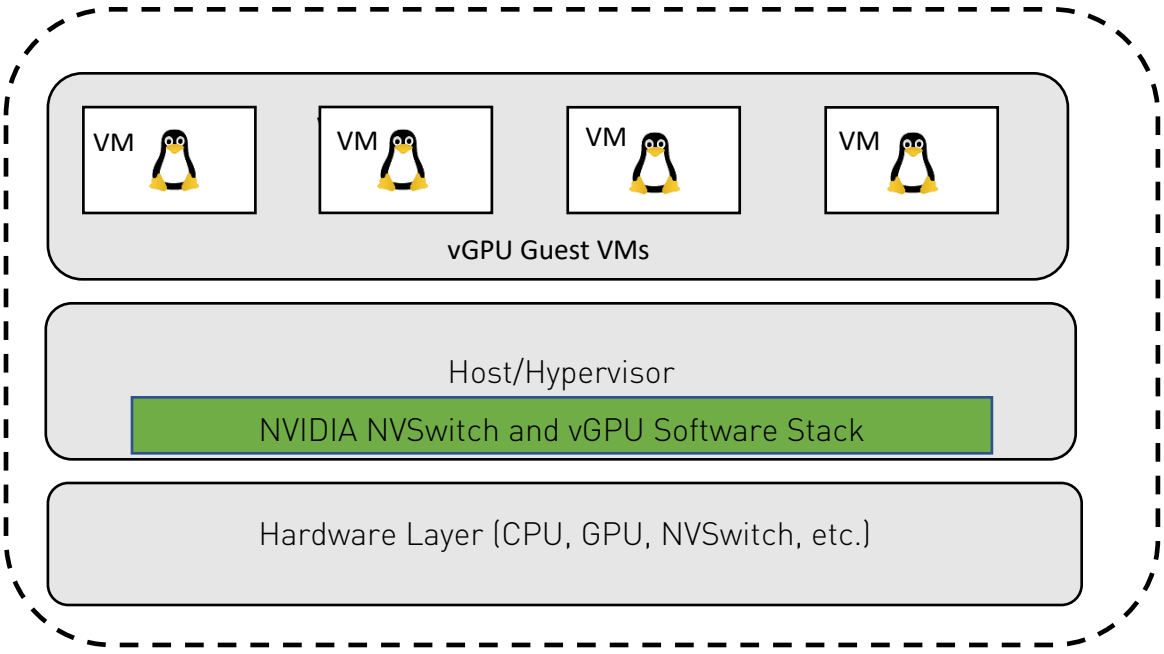
8.1 Software Stack

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 any 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.

Note: The vGPU-based deployment model is not supported on the current release of DGX H100 and NVIDIA HGX H100 systems. NVIDIA plans to add this support in a future software release.

Figure 4. The vGPU Software Stack



8.2 Preparing the vGPU Host

8.2.1 OS Image

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

8.2.2 NVIDIA Software Packages

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

- NVIDIA FM package
- NVIDIA Fabric Manager SDK Package

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

8.2.3 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 the 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 `/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 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.

8.3 Fabric Manager-Shared Library and APIs

Refer to "Fabric Manager SDK" on page 28 for a list of the APIs to manage the vGPU partition life cycle,.

8.4 Fabric Manager Resiliency

Refer to "Resiliency" on page 85 for more information about FM resiliency in vGPU Virtualization mode.

8.5 vGPU Partitions

Refer to "GPU Partitions" on page 81 for the default supported partitions for the vGPU virtualization model.

8.6 Guest Virtual Machine Life Cycle Management

Here is an overview of the guest VM life cycle:

1. System powers on and initializes.
   a). The vGPU host driver loads.
   b). SR-IOV is enabled.
   c). FM initializes in the vGPU Virtualization Mode.
   d). NVlinks are trained.
2. The partition is activated with the selected SR-IOV VFs.
3. The vGPU-enabled VM completes its life cycle with 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.

8.6.1 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.

8.6.2 Deactivating the Partition

Deactivate partitions only when no VM is 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.

8.6.3 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.

8.6.4 Verifying GPU Routing

The `nvswitch-audit` command line utility referenced in "Verifying GPU Routing" on page 53 can also be used to verify NVSwitch routing information in the vGPU mode. We recommend that you run this tool to periodically verify the GPU reachability matrix on each VM partition activation and deactivation cycle.

8.7 Error Handling

Refer to "Error Handling" on page 88 for information about FM initialization, partition, hardware specific errors, and their handling.

8.7.1 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.

8.8 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.

8.9 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.

8.9.1 Enabling MIG before Starting the Fabric Manager Service

- When MIG was enabled on a GPU before FM was 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 for deploying VMs.
- To enable partitions after disabling MIG mode on a GPU, reboot the system.

8.9.2 Enabling MIG After Starting the Fabric Manager Service

- MIG functionality might be enabled on any GPU after starting the FM Service, but before a partition that contained the GPU, is activated.
- Activating a GPU partition will return success even if the GPU is in MIG mode.
- Activating a multi-GPU partition will fail if any GPU in the partition is in MIG mode on DGX A100 and NVIDIA HGX A100 systems.

  The process will succeed on the DGX H100 and NVIDIA HGX H100 systems.

Chapter 9. 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" on page 88 for more information.

9.1 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, DBE error, or a similar issue.
- GPU failure is a GPU failure 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.

9.2 GPU Access NVLink Failure

9.2.1 Fabric Manager Config Item

The GPU access NVLink failure mode is controlled through this FM config file item:

```
ACCESS_LINK_FAILURE_MODE=<value>
```

9.2.2 Bare Metal Behavior

- `ACCESS_LINK_FAILURE_MODE=0`

  In this mode, FM removes the GPUs with access NVLink failure from NVSwitch routing and configures the rest of the GPUs to form one memory fabric. This means 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, FM will disable NVSwitch and its pair of Trunk NVLinks if there are two GPU baseboards where the GPU Access NVLink is connected. 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.

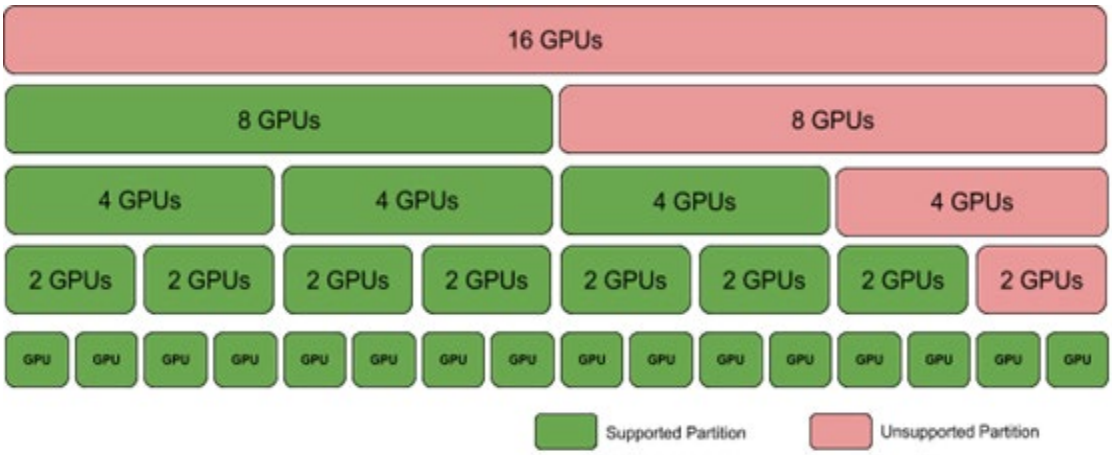
9.2.3 Shared NVSwitch and vGPU Virtualization Behavior

- `ACCESS_LINK_FAILURE_MODE=0`

  In this mode, FM removes the GPUs with access NVLink failures from the currently supported GPU partition list. Figure 4 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.

Figure 4. Shared NVSwitch and vGPU Partitions When a GPU Access NVLink Fails



- `ACCESS_LINK_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 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 Multitenancy Mode.

9.3 Trunk NVLink Failure

9.3.1 Fabric Manager Config Item

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

```
TRUNK_LINK_FAILURE_MODE=<value>
```

Note: This option applies only to systems with two GPU baseboards.

9.3.2 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 `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 `cudaErrorSystemNotReady` 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 as mentioned above.

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

9.3.3 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. This means 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 are present 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 as mentioned above.

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

Note: Currently, the `TRUNK_LINK_FAILURE_MODE=1` configuration is not supported in the vGPU Multitenancy Mode.

9.4 NVSwitch Failure

9.4.1 Fabric Manager Config Item

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

```
NVSWITCH_FAILURE_MODE=<value>
```

9.4.2 Bare Metal Behavior

- `NVSWITCH_FAILURE_MODE=0`

  In this mode, FM aborts and leaves the system uninitialized when there is an NVSwitch failure, 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 happen, FM will fall back to the `NVSWITCH_FAILURE_MODE=0` behavior as mentioned above.

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

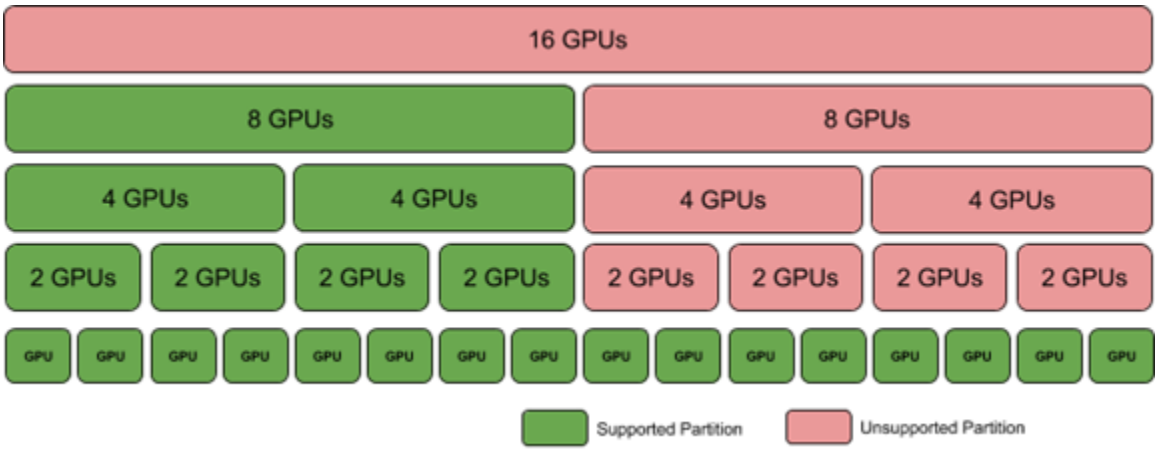
9.4.3 Shared NVSwitch and vGPU Virtualization Behavior

- `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 5 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 5. Shared NVSwitch and vGPU Partitions When an NVSwitch has Failed



- `NVSWITCH_FAILURE_MODE=1`

  In the Shared NVSwitch mode, all the 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 as mentioned above.

  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.

9.5 GPU Failure

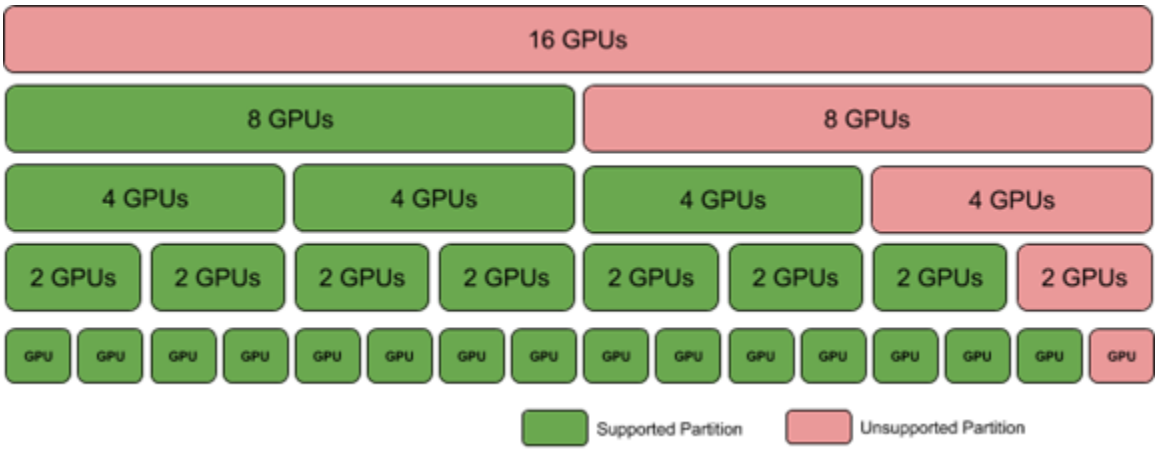
9.5.1 Bare Metal Behavior

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.

9.5.2 Shared NVSwitch and vGPU Virtualization Behavior

FM will continue initialization and adjust the currently supported partition list by excluding the failed GPU partitions. Figure 6 shows the supported partitions when a GPU is missing or has failed to initialize.

Figure 6. Shared NVSwitch and vGPU Partitions When a GPU is Missing/Has Failed



9.6 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.

9.6.1 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.

9.6.1.1 GPU Exclusion Flow

The GPU exclusion flow can be broken down into the following phases:

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.

9.6.1.2 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 5. 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

9.6.1.3 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 any 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 choose to only monitor the passive information.

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.

Note: Refer to the NVIDIA DCGM Documentation for more information about the exclusion recommendation script such as its location and supported options.

9.6.1.4 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 to be excluded are identified by the GPU's unique identifier (GPU UUID) via 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 still 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.

9.6.1.5 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.

9.6.1.6 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:
   a. Add the UUID of the excluded GPU into the .conf file.
   b. Remove the UUID of the GPU from the list.
3. Restart the system to load the kernel driver with updated module parameters.

9.6.1.7 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.

9.6.1.8 nvidia-smi

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

9.6.1.9 Procfs

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

9.6.1.10 Out-of-Band Query

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

9.6.1.11 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.

9.6.1.12 Bare Metal and vGPU Configurations

In, 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. Also, 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.

9.6.1.13 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.

   In test VM, there should be a complete instance of the NVIDIA NVSwitch core software stack, including GPU drivers and FM.
3. On this test VM, run the health check script or DCGM exclusion recommendation script.
4. Based on the output of the health check or exclusion recommendation script, add the GPU UUID to a hypervisor readable database.

   The hypervisor shuts down the test VM.

   a). The hypervisor reads the database to identify the candidates for excluding and updates its resource allocation mechanisms to prevent that GPU from being assigned to future VM requests.

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

9.6.1.14 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 ensures 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.
9. 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.

9.6.2 NVSwitch Exclusion

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

9.6.2.1 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.

Also, in 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.

9.6.2.2 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.

9.6.2.3 Out-of-Band NVSwitch Exclusion

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

Appendix A. NVLink Topology

The following section lists the link IDs used by each GPU to connect to each NVSwitch on different versions of NVIDIA HGX baseboards.

A.1 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 6. 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
3	5	2	4
4	0	4	17
4	1	1	4
4	2	3	7
4	3	6	7

GPU	GPU link	NVSwitch	NVSwitch link
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

A.2 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 7 is the same numbering used in the *HGX A100 Baseboard Pinout* design document.

Table 7. 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
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

GPU	GPU link	NVSwitch	NVSwitch link
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

A.3 NVIDIA HGX H100 GPU Baseboard

The GPU numbering in Table 8 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 8. 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
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

GPU	GPU link	NVSwitch	NVSwitch link
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

Appendix B. GPU Partitions

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

B.1 DGX-2 and NVIDIA HGX-2

Table 9. 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
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. Due to this limitation, the four-GPU and two-GPU VMs can enable only five out of six NVLinks per GPU.

B.2 DGX A100 and NVIDIA HGX A100

B.1.1 Default GPU Partitions

Depending on the high availability mode configurations, when a GPU is unavailable (due to failures, backlisting, and so on), the corresponding partitions will be removed from the currently 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 only one baseboard is present, and the GPIOs are strapped as for the bottom tray, the GPU Physical IDs range is between 1 and 8. If the baseboard is strapped as for the top tray, the GPU Physical IDs range between 9 and 16.

Table 10. 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
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

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
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

B.1.2 Supported GPU Partitions

In DGX A100 and NVIDIA HGX A100 systems, the earlier generation 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 above-mentioned policy on a case-by-case basis and will provide necessary information to configure/override the default GPU partitions.

B.3 DGX H100 and HGX H100

B.1.1 Default GPU Partitions

Table 11. 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

B.1.2 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.

Appendix C. Resiliency

The FM resiliency feature in the Shared NVSwitch and vGPU Model allows system administrators to resume the 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 while FM is not running will be cached into the NVSwitch Driver and will be reported after FM has successfully restarted. Also, changing the FM version when FM is not running is not supported.

C.1 High-Level Flow

1. After an FM crash or graceful exit, the hypervisor will run the `–restart` command line option to start FM and resume the operation.
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 has no knowledge 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 number of partitions as zero.
3. To start Fabric Manager 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.

C.2 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 `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.

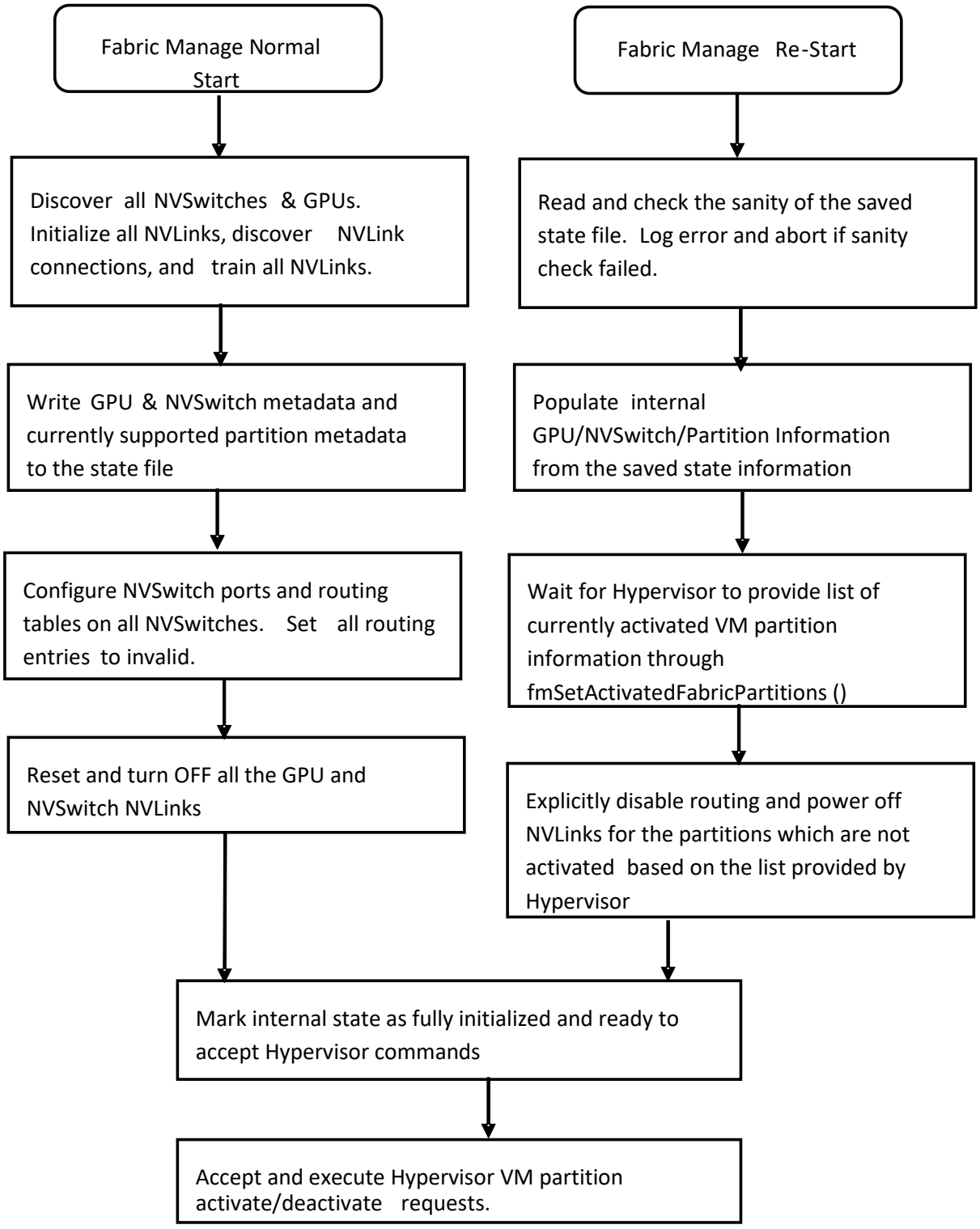
When FM is started with the `--restart` command line option, it will skip most of its NVLink and NVSwitch initialization steps and populate the required information from the stored file. FM will wait for the hypervisor to provide a list of currently activated guest VM partitions. During this time, typical partition operations such as querying the list of supported guest VM partitions, activating and deactivating guest VM partitions, and so on, will be rejected. After the

list of active guest VM partition information is received from hypervisor, FM will ensure that routing is enabled only for those partitions. After these steps have completed, FM will enable typical guest VM partition activation and the deactivation workflow.

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

Figure 7. Fabric Manager Flow: Typical and Restart Options



Appendix D. Error Handling

D.1 FM Initialization Errors

The following errors might occur during FM initialization and topology discovery. These errors happen only during Host boot time (vGPU mode) or Service VM initialization (Shared NVSwitch mode), and the assumption is that no guest VMs are running.

Table 12. Errors During FM Initialization

Error Condition	Error Impact	Recovery
Access NVLink connection (GPU to NVSwitch) training failure	Depending on the `ACCESS_LINK_FAILURE_MODE` 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 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_MODE` 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 FM service (vGPU mode) or Service VM (Shared NVSwitch mode) • If the error persists, inspect/reseat the NVLink Trunk backplane connector.
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_FAILURES` 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 Service VM (NVSwitch mode) • If the error persists, technical troubleshooting is required.

D.2 Partition Life Cycle Errors

Table 13 summarizes potential errors returned by FM SDK APIs while querying supported partitions or activating/deactivating VM partitions.

Table 13. VM Partition Life Cycle Errors

Return Code	Error Condition/Impact	Recovery
FM_ST_BADPARAM	Provided partition ID or other parameters to the APIs are invalid.	Use only the partition IDs returned by `fmGetSupportedFabricPartitions()`. Make sure pointer arguments are not NULL.
FM_ST_NOT_SUPPORTED	FM is not started with required config options.	Make sure that 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	FM APIs are issued before FM is completely initialized.	Wait until FM service is completely initialized.
FM_ST_UNINITIALIZED	FM interface library has not been initialized.	Make sure FM interface library is initialized with call to `fmLibInit()`.
FM_ST_IN_USE	Provided partition ID is already activated or the GPUs required for the specified partition are in use on other activated partitions.	• Provide a non-activated partition ID. • Ensure that the GPUs are not in use by other activated partitions.
FM_ST_UNINITIALIZED	Provided partition ID is already deactivated.	Provide an activated partition ID.
FM_ST_GENERIC_ERROR	A generic error has happened while 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.

D.3 Runtime NVSwitch Errors

NVSwitch runtime errors can be retrieved or monitored using the following options:

- 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.

D.4 Non-Fatal NVSwitch SXid Errors

Table 14 lists potential NVSwitch non-fatal SXid errors that might occur in the field and their impact.

Table 14. 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.	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 GPU/NVSwitch fabric partition routing information using the NVSwitch-audit tool. • Restart the guest VM.	If the error is observed on a Trunk port, 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)	NVLink packet needs to be retransmitted. This error	Not Applicable	If the error is observed on a Trunk port, the

SXid & Error String	Guest VM Impact	Guest VM Recovery	Other Guest VM Impact
	might impact the NVLink throughput of the specified port.		partitions that are using NVSwitch trunk ports might see throughput impact.
12028 (egress non-posted PRIV error)	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 Guest VM	If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports will be affected.
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 section D.9)	If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports will be affected.
22013(Minion Link DLREQ interrupt	This SXid can be safely ignored	Not Applicable	No Impact.
20012	This error could occur due to a broken/inconsistent connection or uncoordinated shutdown	If this issue was not due to an uncoordinated shutdown, check link mechanical connections.	No impact if error is confined to a single GPU.

D.5 Fatal NVSwitch SXid Errors

Table 15 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 15 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 15. 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
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

SXid	SXid Error String
19070	RSP1 RAM DAT ECC DBE Error
20034	LTSSM Fault Up Guest VM Impact: This SXid is triggered whenever the associated link has gone down from active. This interrupt is usually associated with other NVLink errors. Guest VM Recovery: In case of A100, restart the VM. In case of H100, reset the GPU (refer to section D.9). If issue persists, report GPU issues. Other Guest VM Impact: No impact if error is confined to a single 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

D.6 Always Fatal NVSwitch SXid Errors

Table 16 lists 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 restart.

Table 16. 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

SXid	SXid Error String
23008	ingress packet invalid dst error
23009	ingress packet parity error
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

D.7 Other Notable NVSwitch SXid Errors

Table 17 provides additional SXid errors that might affect the overall fabric/system.

Table 17. 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_interrupt	This SXid error is never expected to occur. If it occurs, it is fatal to the fabric/system, and to recover, it will require a reset to all GPUs and NVSwitches (refer to section D.9). If the error is observed on a Trunk port, the partitions that are using NVSwitch trunk ports will be affected.
10004	Host_thermal_event_start	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_end	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

D.8 High Availability Mode Comparison

Table 18 provides information about the expected behaviors for the existing FM high availability mode configuration options on DGX A100/ NVIDIA HGX A100 and DGX H100/ NVIDIA HGX H100 systems.

Table 18. High Availability Mode Comparison Between DGX A100/NVIDIA HGX A100 and DGX-HGX100/ NVIDIA HGX-H100

Configuration	DGX A100/ NVIDIA HGX A100	DGX A100/ NVIDIA HGX A100
TRUNK_LINK_FAILURE_MODE High Availability Mode options when there is a Trunk Link Failure (NVSwitch to NVSwitch NVLink failure).	In bare metal or full passthrough virtualization mode: • 0:Exit FM and leave the system/NVLinks uninitialized. • 1: Disable the NVSwitch and its peer NVSwitch, which reduces NVLink P2P bandwidth. In Shared NVSwitch or vGPU-based multitenancy 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.	The configuration is not applicable on DGX H100 and NVIDIA HGX H100 systems, because there are no trunk NVLinks on these systems. Also on these systems, this configuration value is ignored.
NVSWITCH_FAILURE_MODE High Availability Mode options when there is a NVSwitch failure or an NVSwitch is excluded.	In bare metal or full passthrough virtualization mode: • 0: Abort Fabric Manager.	This configuration is not applicable on DGX H100 and NVIDIA HGXH100 systems. Also on these systems, this configuration value is ignored.

Configuration	DGX A100/ NVIDIA HGX A100	DGX A100/ NVIDIA HGX A100
	• 1: Disable the NVSwitch and its peer NVSwitch, which reduces P2P bandwidth. In Shared NVSwitch or vGPU based multitenancy mode: • 0: Disable partitions that are using the NVSwitch • 1: Disable the NVSwitch and its peer NVSwitch. All partitions will be available but with reduced NVLink P2P bandwidth.	In bare metal or full passthrough virtualization mode: The four NVSwitches on the baseboard should be accessible to FM. If an NVSwitch is not available, FM will abort and leave the system as uninitialized. Depending on the nature of the failure, the GPU NVLinks might be active. However, the GPUs will fail to complete registration with NVLink fabric and CUDA application launch will fail. In Shared NVSwitch or vGPU based multitenancy mode The four NVSwitches on the baseboard should be accessible to FM. If an NVSwitch is not available, FM will abort and leave the system as uninitialized, and the Shared NVSwitch-related partition APIs will fail/return an error.
ACCESS_LINK_FAILURE_MODE High Availability Mode options when there is an Access Link Failure (GPU to NVSwitch NVLink failure)	In bare metal or full passthrough virtualization 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 In Shared NVSwitch or vGPU based multitenancy mode: • 0: Disable partitions that are using the Access Link failed GPU.	The configuration is not applicable on DGX H100 and NVIDIA HGX H100 systems, and the configured value is ignored. In bare metal or full passthrough virtualization mode When there is an Access Link Failure, the corresponding `nvidia-smi` output will show the inactive NVLinks. The GPU will lose its NVLink peer-to-peer capability. In Shared NVSwitch or vGPU based multitenancy mode: FM will report that all partitions are available, and partition activation will be allowed. However, the corresponding GPU on the

Configuration	DGX A100/ NVIDIA HGX A100	DGX A100/ NVIDIA HGX A100
	• 1: Disable the NVSwitch and its peer NVSwitch. All partitions will be available but with reduced NVLink P2P bandwidth.	guest VM will report some NVLinks as `InActive` and lose the NVLink peer-to-peer capability.
ABORT_CUDA_JOBS_ON_FM_EXIT Control running CUDA jobs behavior when FM service is stopped or terminated.	• 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.	The configuration is not applicable on DGX H100 and NVIDIA HGX H100 systems and the configured value is ignored. On these systems, after FM exits, the following occurs: • Running CUDA jobs will continue to run. • A new CUDA job launch will fail. However, if the GPUs have persistence mode enabled, new CUDA job launches will succeed. Also, after GPU reset operation, the CUDA job launch will fail even if the GPUs have persistence mode enabled.

D.9 GPU/VM/System Reset Capabilities and Limitations

Refer to the following excerpt from nvidia-smi man-page

- Used to trigger a reset of one or more GPUs. Can be used to clear GPU HW and SW state in situations that would otherwise require 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 asCUDA application, graphics application like X server, monitoring application like other instance of nvidia-smi) cannot use these devices.

Table 19. GPU/VM/System Reset Capabilities and Limitations

	FM State	Bare Metal	Full Passthrough Virtualization (All devices in the same VM)	Shared NVSwitch Virtualization (GPUs and NVSwitches in different VMs)
Ampere and later with direct NVLink connect	N/A	A GPU can be reset individually. In addition, all GPUs can be reset without specifying any device.	GPU reset not supported in VMs. Restart the VM.	N/A
Ampere + NVSwitch	Running	A GPU can be reset individually. As part of GPU reset operation, corresponding NVSwitch side links will be reset automatically as well by the FM.	GPU reset is not supported in VMs. Restart the VM.	Restart the VM. The service VM flow should ensure proper reset of NVSwitch links through communication with the FM.
	Not Running	Individual GPU reset is not supported. Reset all GPUs and NVSwitches connected together via NVLink.	GPU reset is not supported in VMs. Restart the VM.	
Hopper and later + NVSwitch	N/A	GPUs can be reset individually regardless of FM dependency. In addition, all GPUs and NVSwitches can	GPU reset ability depends on permissions allowed to VM by hypervisor. If not allowed, restart VM.	GPU reset ability depends on permissions allowed to VM by hypervisor. If not allowed, Restart VM.

	FM State	Bare Metal	Full Passthrough Virtualization (All devices in the same VM)	Shared NVSwitch Virtualization (GPUs and NVSwitches in different VMs)
		be reset without specifying any device.		

Notice

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

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

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

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

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

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

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

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

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

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

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051

http://www.nvidia.com