NVIDIA AI Enterprise

User Guide

DU-10617-001 _v2.0 | October 2024

Table of Contents

Chapter 1. Introduction to NVIDIA AI Enterprise....1
1.1. NVIDIA AI Enterprise Software Architecture....2
1.2. Prerequisites for Using NVIDIA AI Enterprise....3
Chapter 2. Installing and Configuring NVIDIA Virtual GPU Manager....5
2.1. About NVIDIA Virtual GPUs....5
2.1.1. NVIDIA vGPU Architecture....5
2.1.1.1. Time-Sliced NVIDIA vGPU Internal Architecture....6
2.1.1.2. MIG-Backed NVIDIA vGPU Internal Architecture....7
2.1.2. About Virtual GPU Types....8
2.1.3. Valid Virtual GPU Configurations on a Single GPU....10
2.1.3.1. Valid Time-Sliced Virtual GPU Configurations on a Single GPU....10
2.1.3.2. Valid MIG-Backed Virtual GPU Configurations on a Single GPU....10
2.2. Switching the Mode of a GPU that Supports Multiple Display Modes....11
2.3. Installing and Configuring the NVIDIA Virtual GPU Manager for VMware vSphere....12
2.3.1. Installing the NVIDIA Virtual GPU Manager Package for vSphere....13
2.3.2. Verifying the Installation of the NVIDIA AI Enterprise Package for vSphere....14
2.3.3. Configuring VMware vMotion with vGPU for VMware vSphere....15
2.3.4. Changing the Default Graphics Type in VMware vSphere 6.5 and Later....16
2.3.5. Configuring a GPU for MIG-Backed vGPUs....23
2.3.5.1. Enabling MIG Mode for a GPU....23
2.3.5.2. Disabling MIG Mode for One or More GPUs....25
2.3.6. Configuring a vSphere VM with NVIDIA vGPU....26
2.3.7. Setting vGPU Plugin Parameters on VMware vSphere....29
2.4. Disabling and Enabling ECC Memory....30
2.4.1. Disabling ECC Memory....30
2.4.2. Enabling ECC Memory....32
Chapter 3. Installing and Licensing NVIDIA AI Enterprise Components Required in a Guest VM....34
3.1. Installing NVIDIA AI Enterprise Software Components by Using Kubernetes....34
3.1.1. Installing and Licensing the NVIDIA vGPU Software Graphics Driver by Using NVIDIA GPU Operator....34
3.1.2. Transforming Container Images for AI and Data Science Applications and Frameworks into Kubernetes Pods....35
3.2. Install NVIDIA AI Enterprise Software Components by Using Docker....35
3.2.1. Installing and Licensing the NVIDIA vGPU Software Graphics Driver Natively....35

3.2.2. Installing AI and Data Science Applications and Frameworks by Using Docker.....35
3.3. Installing and Licensing NVIDIA AI Enterprise Components Natively.....36
3.3.1. Installing the NVIDIA AI Enterprise Graphics Driver on Ubuntu from a Debian Package.....36
3.3.2. Prerequisites for Configuring a Licensed Client of NVIDIA License System with a Networked License.....37
3.3.2.1. Configuring a Licensed Client with a Networked License on Linux with Default Settings.....37
3.3.2.2. Verifying the NVIDIA AI Enterprise License Status of a Licensed Client.....39
3.3.3. Installing NVIDIA Container Toolkit.....39
3.3.4. Verifying the Installation of NVIDIA Container Toolkit.....41
3.3.5. Installing Software Distributed as Container Images.....41
3.3.6. Running ResNet-50 with TensorRT.....42
3.3.7. Running ResNet-50 with TensorFlow.....43
3.3.8. Optional: Updating NVIDIA Container Toolkit for a MIG-Enabled vGPU.....44
Chapter 4. Configuring Multinode Scaling.....46
4.1. Hardware and VM Configuration Requirements for Multinode Scaling.....46
4.1.1. Hardware Requirements for Multinode Scaling.....46
4.1.2. VM Requirements for Multinode Scaling.....47
4.2. Configuring NUMA Affinity for the VMs.....47
4.2.1. Configuring NUMA Affinity for a Whole-Server VM with Two GPUs and Two NICs Across Both NUMA Nodes.....48
4.2.2. Configuring NUMA Affinity for a Per-Socket VM with One GPU and One NIC on a Single NUMA Node.....50
4.3. Configuring RoCE on the NVIDIA Mellanox Spectrum Switch.....51
4.4. Enabling GPUDirect Technology for Peer-to-Peer Connections.....52
4.5. Installing the Mellanox OFED Driver.....53
4.6. Enabling ATS on the NVIDIA ConnectX-6 DX NICs in a VM.....53
4.7. Building and Installing the NVIDIA Peer Memory Driver.....55
Chapter 5. Modifying a VM's NVIDIA vGPU Configuration.....56
5.1. Removing a VM's NVIDIA vGPU Configuration.....56
5.1.1. Removing a vSphere VM's vGPU Configuration.....56
5.2. Modifying GPU Allocation Policy.....56
5.2.1. Modifying GPU Allocation Policy on VMware vSphere.....57
5.3. Migrating a VM Configured with vGPU.....60
5.3.1. Migrating a VM Configured with vGPU on VMware vSphere.....61
5.3.2. Suspending and Resuming a VM Configured with vGPU on VMware vSphere.....63
5.4. Modifying a MIG-Backed vGPU's Configuration.....63

5.5. Enabling Unified Memory for a vGPU....66
5.5.1. Enabling Unified Memory for a vGPU on VMware vSphere....66
5.6. Enabling NVIDIA CUDA Toolkit Development Tools for NVIDIA vGPU....67
5.6.1. Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU....67
5.6.2. Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU....68
5.6.2.1. Supported NVIDIA CUDA Toolkit Profiler Features....68
5.6.2.2. Clock Management for a vGPU VM for Which NVIDIA CUDA Toolkit Profilers Are Enabled....68
5.6.2.3. Limitations on the Use of NVIDIA CUDA Toolkit Profilers with NVIDIA vGPU....68
5.6.2.4. Enabling NVIDIA CUDA Toolkit Profilers for a vGPU VM....69
Chapter 6. Monitoring GPU Performance....70
6.1. NVIDIA System Management Interface nvidia-smi....70
6.2. Using nvidia-smi to Monitor GPU Performance from a Hypervisor....71
6.2.1. Getting a Summary of all Physical GPUs in the System....71
6.2.2. Getting a Summary of all vGPUs in the System....72
6.2.3. Getting vGPU Details....72
6.2.4. Monitoring vGPU engine usage....73
6.2.5. Monitoring vGPU engine usage by applications....74
6.2.6. Monitoring Encoder Sessions....75
6.2.7. Listing Supported vGPU Types....76
6.2.8. Listing the vGPU Types that Can Currently Be Created....77
6.3. Monitoring GPU Performance from a Guest VM....78
6.3.1. Using nvidia-smi to Monitor GPU Performance from a Guest VM....78
Chapter 7. Changing Scheduling Behavior for Time-Sliced vGPUs....81
7.1. Scheduling Policies for Time-Sliced vGPUs....81
7.2. Scheduler Time Slice for Time-Sliced vGPUs....82
7.3. RmPVMRL Registry Key....82
7.4. Getting the Current Time-Sliced vGPU Scheduling Behavior for All GPUs....84
7.5. Changing the Time-Sliced vGPU Scheduling Behavior for All GPUs....85
7.6. Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs....86
7.7. Restoring Default Time-Sliced vGPU Scheduler Settings....88
Chapter 8. Troubleshooting....89
8.1. Known issues....89
8.2. Troubleshooting steps....89
8.2.1. Verifying the NVIDIA Kernel Driver Is Loaded....89
8.2.2. Verifying that nvidia-smi works....90
8.2.3. Examining NVIDIA kernel driver output....90
8.2.4. Examining NVIDIA Virtual GPU Manager Messages....90

8.2.4.1. Examining VMware vSphere vGPU Manager Messages..... 91
8.3. Capturing configuration data by running nvidia-bug-report.sh.....91
Chapter 9. Additional Information.....93
Appendix A. Virtual GPU Types for Supported GPUs.....94
A.1. NVIDIA A100 PCIe 40GB Virtual GPU Types.....94
A.2. NVIDIA A100 HGX 40GB Virtual GPU Types.....95
A.3. NVIDIA A100 PCIe 80GB and NVIDIA A100X Virtual GPU Types..... 97
A.4. NVIDIA A100 HGX 80GB Virtual GPU Types.....98
A.5. NVIDIA A40 Virtual GPU Types.....99
A.6. NVIDIA A30 and NVIDIA A30X Virtual GPU Types.....103
A.7. NVIDIA A10 Virtual GPU Types.....104
A.8. NVIDIA RTX A6000 Virtual GPU Types..... 107
A.9. NVIDIA RTX A5000 Virtual GPU Types..... 111
A.10. Tesla T4 Virtual GPU Types.....114

List of Figures

Figure 1. NVIDIA vGPU System Architecture 6

Figure 2. Time-Sliced NVIDIA vGPU Internal Architecture 7

Figure 3. MIG-Backed NVIDIA vGPU Internal Architecture 8

Figure 4. Example MIG-Backed vGPU Configurations on NVIDIA A100 PCIe 40GB 11

Figure 5. Shared default graphics type 18

Figure 6. Host graphics settings for vGPU 20

Figure 7. Shared graphics type 21

Figure 8. Graphics device settings for a physical GPU 22

Figure 9. Shared direct graphics type 23

Figure 10. VM settings for vGPU 27

Figure 11. Breadth-first allocation scheme setting for vGPU-enabled VMs 58

Figure 12. Host graphics settings for vGPU 59

Figure 13. Depth-first allocation scheme setting for vGPU-enabled VMs 60

Figure 14. Using nvidia-smi from a Windows guest VM to get total resource usage by all applications 79

Figure 15. Using nvidia-smi from a Windows guest VM to get resource usage by individual applications 80

Chapter 1. Introduction to NVIDIA AI Enterprise

NVIDIA® AI Enterprise is an end-to-end, cloud-native suite of AI and data analytics software, optimized so every organization can succeed with AI. It's certified to deploy anywhere—from the enterprise data center to the public cloud—and includes global enterprise support and training.

NVIDIA AI Enterprise includes key enabling technologies and software from NVIDIA for rapid deployment, management, and scaling of AI workloads in the modern hybrid cloud.

NVIDIA AI Enterprise enables the following:

1. Leverage fully integrated, optimized, certified, and supported software from NVIDIA for AI workloads.
2. Run NVIDIA AI frameworks and tools optimized for GPU acceleration, reducing deployment time and ensuring reliable performance.
3. Deploy anywhere – including on popular data center platforms from VMware and Red Hat, mainstream NVIDIA-Certified Systems configured with or without GPUs, and on GPU-accelerated instances in the public cloud.
4. Leverage the jointly certified NVIDIA and Red Hat solution to deploy and manage AI workloads in containers or VMs with optimized software.
5. Scale out to multiple nodes, enabling even the largest deep learning training models to run on the VMware vSphere. Previously, scaling with bare metal performance in a fully virtualized environment was limited to a single node, limiting the complexity and size of AI workloads that could be supported.
6. Run AI workloads at near bare-metal performance with new optimizations for GPU acceleration on vSphere, including support for the latest Ampere architecture including the NVIDIA A100. Additionally, technologies like GPUDirect Communications can now be supported on vSphere. This provides communication between GPU memory and storage across a cluster for improved performance.

1.1. NVIDIA AI Enterprise Software Architecture

The software in the NVIDIA AI Enterprise suite is organized into separate layers for infrastructure optimization software, cloud native deployment software, and AI and data science frameworks.

The content of these layers is as follows:

- Infrastructure optimization software:
  - NVIDIA virtual GPU (vGPU) software
  - NVIDIA CUDA Toolkit
  - NVIDIA Magnum IO™ software stack for accelerated data centers
- Cloud native deployment software:
  - NVIDIA GPU Operator
  - NVIDIA Network Operator
- AI and data science frameworks:
  - TensorFlow
  - PyTorch
  - NVIDIA Triton Inference Server
  - NVIDIA TensorRT
  - RAPIDS

  The AI and data science frameworks are delivered as container images. Containerized software can be run directly with a tool such as Docker.

What Is Included?

The NVIDIA AI Enterprise suite includes frameworks that are broadly applicable and used across vertical industries such as manufacturing, logistics, financial services, retail, and healthcare.

NVIDIA AI Enterprise includes:

1. TensorFlow and PyTorch for maching learning.
2. NVIDIA TAO Toolkit for a faster, easier way to accelerate training and quickly create highly accurate and performant, domain-specific vision, and conversational AI models.

3. NVIDIA Tensor RT, for GPU optimized deep learning inference and Triton Inference Server to deploy trained AI models at scale.
4. Triton Inference Server supports all major frameworks, such as TensorFlow, TensorRT, PyTorch, MXNet, Python and more. Triton Inference Server also includes the RAPIDS FIL backend for the best inference performance for tree-based models on GPUs.
5. NVIDIA RAPIDS, for end-to-end data science, machine learning and analytics pipeline.
6. NVIDIA GPU and Network Operators, to deploy and manage NVIDIA GPU and Networking resources in Kubernetes.
7. NVIDIA vGPU Software, to deploy vGPU on common data center platforms, including VMware and Red Hat.

1.2. Prerequisites for Using NVIDIA AI Enterprise

Before proceeding, ensure that these prerequisites are met:

- You have a system that meets the requirements in *NVIDIA AI Enterprise Release Notes*.
- One or more supported NVIDIA GPUs are installed in your system.
- If you are using an NVIDIA A100 GPU, the following BIOS settings are enabled on your system:
  - Single Root I/O Virtualization (SR-IOV)
  - VT-d/IOMMU - Enabled
- The following software is installed according to the instructions in the VMware documentation:
  - VMware vSphere Hypervisor ESXi
  - VMware vCenter Server
- A VM to be enabled with NVIDIA vGPU is created with the virtual hardware resources in the following table.

Resource	Requirements
vCPUs	16
RAM	64 GB
Storage	500 GB thin provisioned virtual disk
NIC	VMXNet3 NIC connected to network

- A supported guest OS is installed in the VM.

For optimum performance, set options in your server configuration as follows:

- Enable the following options:
  - **Hyperthreading**

  - **Memory Mapped I/O above 4 GB** (if applicable)
- Set the **Power Setting** or **System Profile** option to **High Performance**.
- If applicable, set **CPU Performance** to **Enterprise** or **High Throughput**.

> **Note:** If NVIDIA card detection does not include all the installed GPUs, set this option to **Enabled**.

Chapter 2. Installing and Configuring NVIDIA Virtual GPU Manager

The process for installing and configuring NVIDIA Virtual GPU Manager depends on the hypervisor that you are using. After you complete this process, you can install the display drivers for your guest OS and license any NVIDIA AI Enterprise licensed products that you are using.

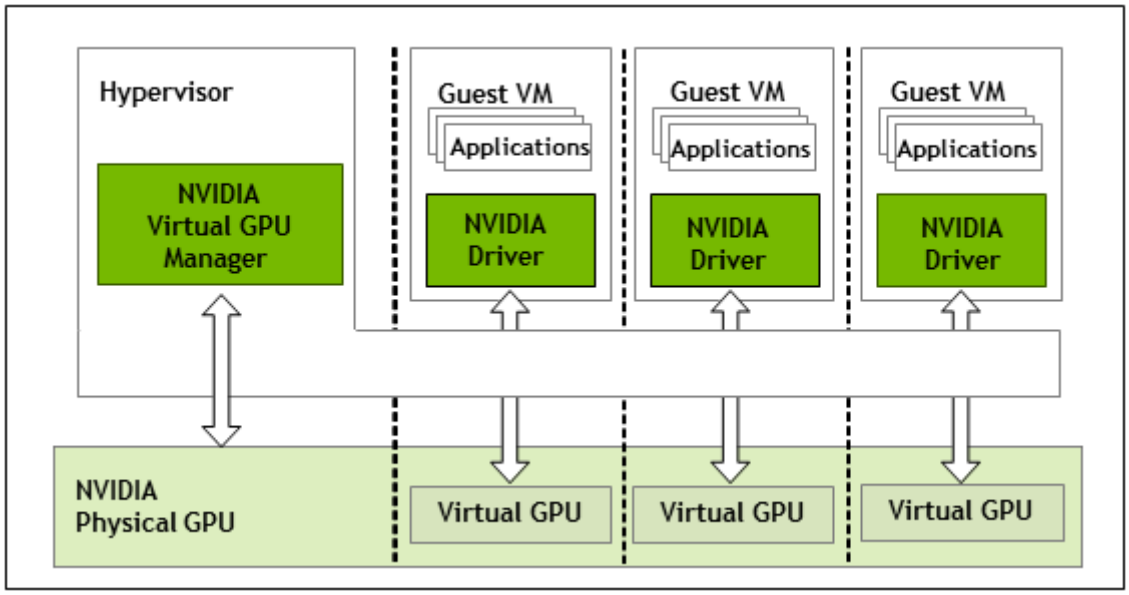
2.1. About NVIDIA Virtual GPUs

2.1.1. NVIDIA vGPU Architecture

The high-level architecture of NVIDIA vGPU is illustrated in Figure 1. Under the control of the NVIDIA Virtual GPU Manager running under the hypervisor, NVIDIA physical GPUs are capable of supporting multiple virtual GPU devices (vGPUs) that can be assigned directly to guest VMs.

Guest VMs use NVIDIA vGPUs in the same manner as a physical GPU that has been passed through by the hypervisor: an NVIDIA driver loaded in the guest VM provides direct access to the GPU for performance-critical fast paths, and a paravirtualized interface to the NVIDIA Virtual GPU Manager is used for non-performant management operations.

Figure 1. NVIDIA vGPU System Architecture



Each NVIDIA vGPU is analogous to a conventional GPU, having a fixed amount of GPU framebuffer, and one or more virtual display outputs or "heads". The vGPU's framebuffer is allocated out of the physical GPU's framebuffer at the time the vGPU is created, and the vGPU retains exclusive use of that framebuffer until it is destroyed.

Depending on the physical GPU, different types of vGPU can be created on the vGPU:

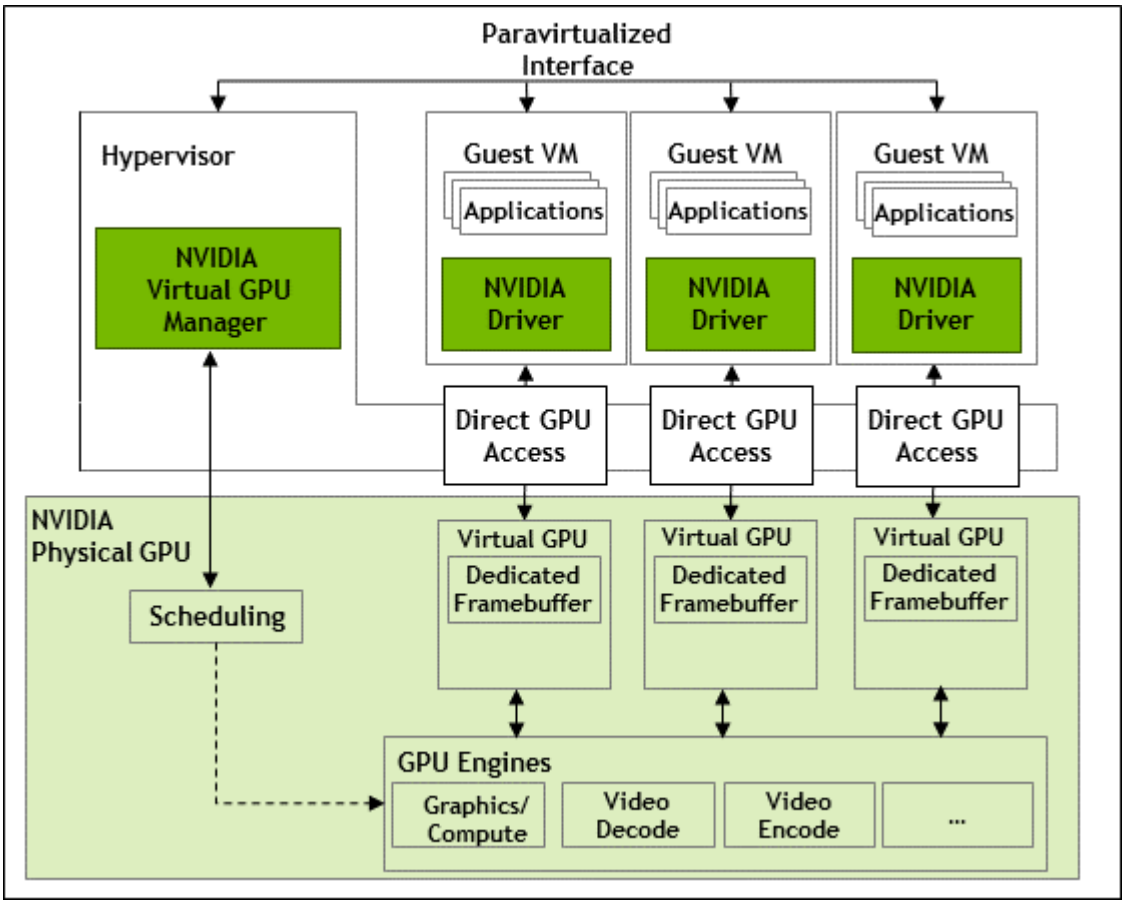
- On all GPUs that support NVIDIA AI Enterprise, time-sliced vGPUs can be created.
- Additionally, on GPUs that support the Multi-Instance GPU (MIG) feature, MIG-backed vGPUs can be created. The MIG feature is introduced on GPUs that are based on the NVIDIA Ampere GPU architecture.

2.1.1.1. Time-Sliced NVIDIA vGPU Internal Architecture

A time-sliced vGPU is a vGPU that resides on a physical GPU that is not partitioned into multiple GPU instances. All time-sliced vGPUs resident on a GPU share access to the GPU's engines including the graphics (3D), video decode, and video encode engines.

In a time-sliced vGPU, processes that run on the vGPU are scheduled to run in series. Each vGPU waits while other processes run on other vGPUs. While processes are running on a vGPU, the vGPU has exclusive use of the GPU's engines. You can change the default scheduling behavior as explained in Changing Scheduling Behavior for Time-Sliced vGPUs.

Figure 2. Time-Sliced NVIDIA vGPU Internal Architecture

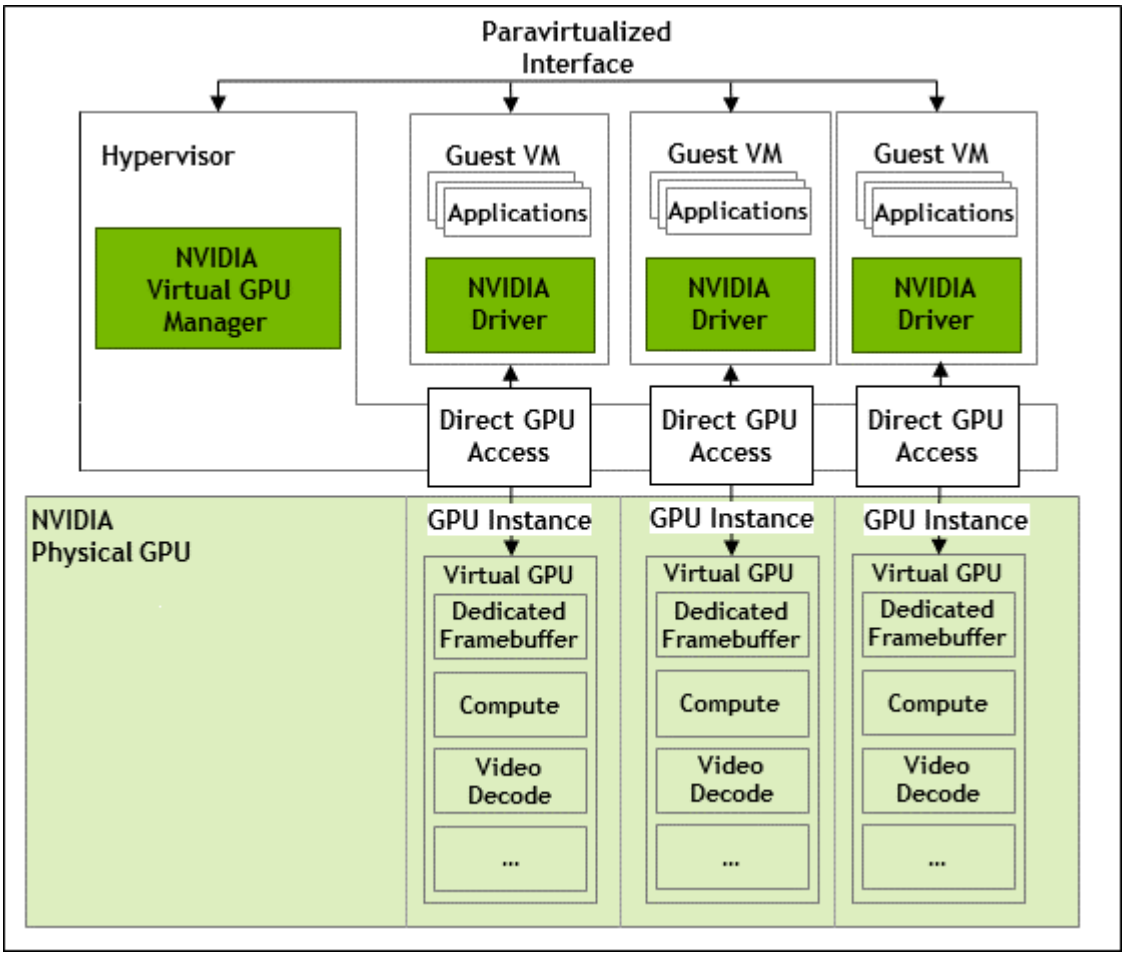


2.1.1.2. MIG-Backed NVIDIA vGPU Internal Architecture

A MIG-backed vGPU is a vGPU that resides on a GPU instance in a MIG-capable physical GPU. Each MIG-backed vGPU resident on a GPU has exclusive access to the *GPU instance's* engines, including the compute and video decode engines.

In a MIG-backed vGPU, processes that run on the vGPU run in parallel with processes running on other vGPUs on the GPU. Process run on all vGPUs resident on a physical GPU simultaneously.

Figure 3. MIG-Backed NVIDIA vGPU Internal Architecture



2.1.2. About Virtual GPU Types

The number of physical GPUs that a board has depends on the board. Each physical GPU can support several different types of virtual GPU (vGPU). vGPU types have a fixed amount of frame buffer, number of supported display heads, and maximum resolutions[1]. They are grouped into different series according to the different classes of workload for which they are optimized. Each series is identified by the last letter of the vGPU type name.

Series	Optimal Workload
Q-series	Virtual workstations for creative and technical professionals who require the performance and features of Quadro technology
C-series	Compute-intensive server workloads, such as artificial intelligence (AI), deep learning, or high-performance computing (HPC)[2, 3]

[1] NVIDIA vGPUs with less than 1 Gbyte of frame buffer support only 1 virtual display head on a Windows 10 guest OS.
[2] C-series vGPU types are NVIDIA Virtual Compute Server vGPU types, which are optimized for compute-intensive workloads. As a result, they support only a single display head and do not provide Quadro graphics acceleration.
[3] The maximum number of NVIDIA Virtual Compute Server vGPUs is limited to **eight** vGPUs per physical GPU, irrespective of the available hardware resources of the physical GPU.

Series	Optimal Workload
B-series	Virtual desktops for business professionals and knowledge workers
A-series	App streaming or session-based solutions for virtual applications users[6]

The number after the board type in the vGPU type name denotes the amount of frame buffer that is allocated to a vGPU of that type. For example, a vGPU of type A16-4C is allocated 4096 Mbytes of frame buffer on an NVIDIA A16 board.

Due to their differing resource requirements, the maximum number of vGPUs that can be created simultaneously on a physical GPU varies according to the vGPU type. For example, an NVDIA A16 board can support up to 4 A16-4C vGPUs on each of its two physical GPUs, for a total of 16 vGPUs, but only 2 A16-8C vGPUs, for a total of 8 vGPUs.

When enabled, the frame-rate limiter (FRL) limits the maximum frame rate in frames per second (FPS) for a vGPU as follows:

- For B-series vGPUs, the maximum frame rate is 45 FPS.
- For Q-series, C-series, and A-series vGPUs, the maximum frame rate is 60 FPS.

When enabled, the frame-rate limiter (FRL) limits the maximum frame rate in frames per second (FPS) for C-series vGPUs to 60 FPS.

By default, the FRL is enabled for all GPUs. The FRL is disabled when the vGPU scheduling behavior is changed from the default best-effort scheduler on GPUs that support alternative vGPU schedulers. For details, see Changing Scheduling Behavior for Time-Sliced vGPUs. On vGPUs that use the best-effort scheduler, the FRL can be disabled as explained in the release notes for your chosen hypervisor at NVIDIA AI Enterprise Documentation.

Note:

NVIDIA vGPU is a licensed product on all supported GPU boards. A software license is required to enable all vGPU features within the guest VM. The type of license required depends on the vGPU type.

- Q-series vGPU types require a vWS license.
- C-series vGPU types require an NVIDIA Virtual Compute Server (vCS) license but can also be used with a vWS license.
- B-series vGPU types require a vPC license but can also be used with a vWS license.

[4] The -1B4 and -2B4 vGPU types are deprecated in this release, and may be removed in a future release. In preparation for the possible removal of these vGPU types, use the following vGPU types, which provide equivalent functionality:

- Instead of -1B4 vGPU types, use -1B vGPU types.
- Instead of -2B4 vGPU types, use -2B vGPU types.

[5] With many workloads, -1B and -1B4 vGPUs perform adequately with only 2 2560×1600 virtual displays per vGPU. If you want to use more than 2 2560×1600 virtual displays per vGPU, use a vGPU with more frame buffer, such as a -2B or -2B4 vGPU. For more information, see *NVIDIA GRID vPC Sizing Guide* (PDF).

[6] A-series NVIDIA vGPUs support a single display at low resolution to be used as the console display in remote application environments such as RDSH and Citrix Virtual Apps and Desktops. The maximum resolution and number of virtual display heads for the A-series NVIDIA vGPUs applies only to the console display. The maximum resolution of each RDSH or Citrix Virtual Apps and Desktops session is determined by the remoting solution and is **not** restricted by the maximum resolution of the vGPU. Similarly, the number of virtual display heads supported by each session is determined by the remoting solution and is **not** restricted by the vGPU.

- A-series vGPU types require a vApps license.

Note: NVIDIA vGPU is a licensed product on all supported GPU boards. An NVIDIA AI Enterprise software license is required to enable all vGPU features within the guest VM.

For details of the virtual GPU types available from each supported GPU, see Virtual GPU Types for Supported GPUs.

2.1.3. Valid Virtual GPU Configurations on a Single GPU

Valid vGPU configurations on a single GPU depend on whether the vGPUs are time sliced or, on GPUs that support MIG, are MIG-backed.

2.1.3.1. Valid Time-Sliced Virtual GPU Configurations on a Single GPU

This release of NVIDIA vGPU supports time-sliced vGPUs with the same amount of frame buffer from different virtual GPU series on the same physical GPU. For example, A-series, B-series, C-series, and Q-series vGPUs with the same amount of frame buffer can reside on the same physical GPU simultaneously.

However, the requirement that all vGPUs have the same amount of frame buffer doesn't extend across physical GPUs on the same card. Different physical GPUs on the same card may host virtual GPUs with different amounts of frame buffer at the same time, provided that the vGPU types on any one physical GPU all have the same amount of frame buffer.

For example, an NVIDIA A16 card has four physical GPUs, and can support several types of virtual GPU.

- A configuration with a mixture of A16-4C vGPUs and A16-4Q vGPUs on GPU0 is valid.
- A configuration with A16-16C vGPUs on GPU 0 and GPU 1, A16-8C vGPUs on GPU 2, and A16-4C vGPUs on GPU3 is valid.
- A configuration with a mixture of A16-8C vGPUs and A16-4C vGPUs on GPU0 is invalid.

Not all hypervisors support time-sliced vGPUs with the same amount of frame buffer from different virtual GPU series on the same physical GPU. To determine if your chosen hypervisor supports this feature, consult the release notes for your hypervisor at NVIDIA AI Enterprise Documentation.

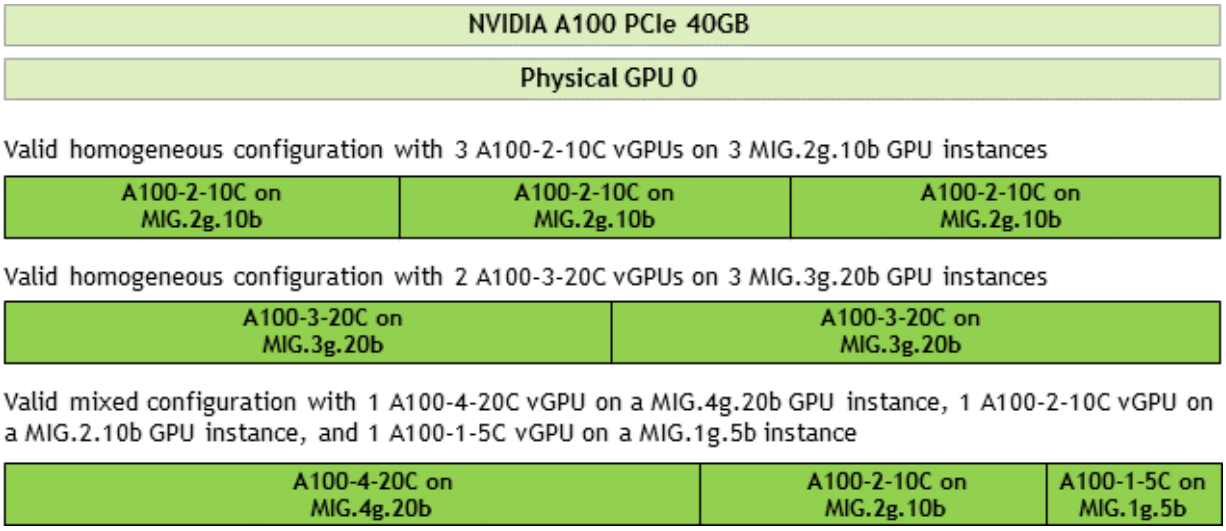
2.1.3.2. Valid MIG-Backed Virtual GPU Configurations on a Single GPU

This release of NVIDIA vGPU supports both homogeneous and mixed MIG-backed virtual GPUs based on the underlying GPU instance configuration.

For example, an NVIDIA A100 PCIe 40GB card has one physical GPU, and can support several types of virtual GPU. Figure 4 shows the following examples of valid homogeneous and mixed MIG-backed virtual GPU configurations on NVIDIA A100 PCIe 40GB.

- A valid homogeneous configuration with 3 A100-2-10C vGPUs on 3 MIG.2g.10b GPU instances
- A valid homogeneous configuration with 2 A100-3-20C vGPUs on 3 MIG.3g.20b GPU instances
- A valid mixed configuration with 1 A100-4-20C vGPU on a MIG.4g.20b GPU instance, 1 A100-2-10C vGPU on a MIG.2.10b GPU instance, and 1 A100-1-5C vGPU on a MIG.1g.5b instance

Figure 4. Example MIG-Backed vGPU Configurations on NVIDIA A100 PCIe 40GB



2.2. Switching the Mode of a GPU that Supports Multiple Display Modes

Some GPUs support displayless and display-enabled modes but must be used in NVIDIA AI Enterprise deployments in displayless mode.

The GPUs listed in the following table support multiple display modes. As shown in the table, some GPUs are supplied from the factory in displayless mode, but other GPUs are supplied in a display-enabled mode.

GPU	Mode as Supplied from the Factory
NVIDIA A40	Displayless
NVIDIA RTX A5000	Display enabled
NVIDIA RTX A6000	Display enabled

A GPU that is supplied from the factory in displayless mode, such as the NVIDIA A40 GPU, might be in a display-enabled mode if its mode has previously been changed.

To change the mode of a GPU that supports multiple display modes, use the `displaymodeselector` tool, which you can request from the NVIDIA Display Mode Selector Tool page on the NVIDIA Developer website.

Note:

Only the following GPUs support the `displaymodeselector` tool:

- NVIDIA A40
- NVIDIA RTX A5000
- NVIDIA RTX A6000

Other GPUs that support NVIDIA AI Enterprise do not support the `displaymodeselector` tool and, unless otherwise stated, do not require display mode switching.

2.3. Installing and Configuring the NVIDIA Virtual GPU Manager for VMware vSphere

You can use the NVIDIA Virtual GPU Manager for VMware vSphere to set up a VMware vSphere VM to use NVIDIA vGPU or VMware vSGA. The vGPU Manager vSphere Installation Bundles (VIBs) for VMware vSphere 6.5 and later provide vSGA and vGPU functionality in a single VIB. For VMware vSphere 6.0, vSGA and vGPU functionality are provided in separate vGPU Manager VIBs.

Note:

Some servers, for example, the Dell R740, do not configure SR-IOV capability if the SR-IOV SBIOS setting is disabled on the server. If you are using the Tesla T4 GPU with VMware vSphere on such a server, you must ensure that the SR-IOV SBIOS setting is enabled on the server.

However, with any server hardware, do not enable SR-IOV in VMware vCenter Server for the Tesla T4 GPU. If SR-IOV is enabled in VMware vCenter Server for T4, VMware vCenter Server lists the status of the GPU as needing a reboot. You can ignore this status message.

NVIDIA vGPU Instructions

Note: As of VMware vSphere 7.0 Update 1, the Xorg service is no longer required for graphics devices in NVIDIA vGPU mode. For more information, see #unique_18.

For NVIDIA vGPU, follow this sequence of instructions:

1. #unique_18
2. Configuring VMware vMotion with vGPU for VMware vSphere
3. Changing the Default Graphics Type in VMware vSphere 6.5 and Later

4. **MIG-backed vGPUs only:** Configuring a GPU for MIG-Backed vGPUs
5. Configuring a vSphere VM with NVIDIA vGPU
6. **Optional:** Setting vGPU Plugin Parameters on VMware vSphere

After configuring a vSphere VM to use NVIDIA vGPU, you can install the NVIDIA AI Enterprise graphics driver for your guest OS and license any NVIDIA AI Enterprise licensed products that you are using.

VMware vSGA Instructions

For VMware vSGA, follow this sequence of instructions:

1. #unique_18
2. #unique_24

Installation of the NVIDIA AI Enterprise graphics driver for the guest OS is not required for vSGA.

Requirements for Configuring NVIDIA vGPU in a DRS Cluster

You can configure a VM with NVIDIA vGPU on an ESXi host in a VMware Distributed Resource Scheduler (DRS) cluster. However, you must ensure that the automation level of the cluster supports VMs configured with NVIDIA vGPU:

- For any supported VMware vSphere release, set the automation level to **Manual**.
- For VMware vSphere 6.7 Update 1 or later, set the automation level to **Partially Automated** or **Manual**.

For more information about these settings, see Edit Cluster Settings in the VMware documentation.

2.3.1. Installing the NVIDIA Virtual GPU Manager Package for vSphere

To install the vGPU Manager package you need to access the ESXi host via the ESXi Shell or SSH. Refer to VMware's documentation on how to enable ESXi Shell or SSH for an ESXi host. Before you begin, ensure that the following prerequisites are met:

- The ZIP archive that contains NVIDIA AI Enterprise has been downloaded from the NVIDIA Licensing Portal.
- The NVIDIA Virtual GPU Manager package has been extracted from the downloaded ZIP archive.

1. Copy the NVIDIA Virtual GPU Manager package file to the ESXi host.
2. Put the ESXi host into maintenance mode.

```
$ esxcli system maintenanceMode set --enable true
```

3. Use the `esxcli` command to install the vGPU Manager package.

For more information about the `esxcli` command, see esxcli software Commands in the VMware vSphere documentation.

```
[root@esxi:~] esxcli software vib install -d /vmfs/volumes/datastore/software-component.zip
```

datastore
The name of the VMFS datastore to which you copied the software component.

software-component
The name of the file that contains the NVIDIA Virtual GPU Manager package in the form of a software component. Ensure that you specify the file that was extracted from the downloaded ZIP archive. For example, for VMware vSphere , *software-component* is ***-build-number***.

- For a software component, run the following command:

```
[root@esxi:~] esxcli software vib install -d /vmfs/volumes/datastore/software-component.zip
```

datastore
The name of the VMFS datastore to which you copied the software component.

software-component
The name of the file that contains the NVIDIA Virtual GPU Manager package in the form of a software component. Ensure that you specify the file that was extracted from the downloaded ZIP archive. For example, for VMware vSphere , *software-component* is ***-build-number***.

- For a VIB file, run the following command:

```
[root@esxi:~] esxcli software vib install -v directory/NVIDIA-AIE_ESXi_7.0.2_Driver_470.63-1OEM.702.0.0.17630552.vib
Installation Result
   Message: Operation finished successfully.
   Reboot Required: false
   VIBs Installed: NVIDIA-AIE_ESXi_7.0.2_Driver_470.63-1OEM.702.0.0.17630552
   VIBs Removed:
   VIBs Skipped:
```

directory
The absolute path to the directory to which you copied the VIB file. You must specify the absolute path even if the VIB file is in the current working directory.

4. Exit maintenance mode.

```
$ esxcli system maintenanceMode set --enable false
```

5. Reboot the ESXi host.

```
$ reboot
```

2.3.2. Verifying the Installation of the NVIDIA AI Enterprise Package for vSphere

After the ESXi host has rebooted, verify the installation of the NVIDIA AI Enterprise package for vSphere.

1. Verify that the NVIDIA AI Enterprise package installed and loaded correctly by checking for the NVIDIA kernel driver in the list of kernel loaded modules.

```
[root@esxi:~] vmkload_mod -l | grep nvidia
nvidia                    5    8420
```

2. If the NVIDIA driver is not listed in the output, check `dmesg` for any load-time errors reported by the driver.

3. Verify that the NVIDIA kernel driver can successfully communicate with the NVIDIA physical GPUs in your system by running the `nvidia-smi` command.

   The `nvidia-smi` command is described in more detail in NVIDIA System Management Interface nvidia-smi.

Running the `nvidia-smi` command should produce a listing of the GPUs in your platform.

```
[root@esxi:~] nvidia-smi
Fri Aug 27 17:56:22 2024
+-----------------------------------------------------------+
| NVIDIA-SMI 470.63      Driver Version: 470.63      |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla M60           On   | 00000000:05:00.0 Off |                  Off |
| N/A   25C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           On   | 00000000:06:00.0 Off |                  Off |
| N/A   24C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla M60           On   | 00000000:86:00.0 Off |                  Off |
| N/A   25C    P8    25W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla M60           On   | 00000000:87:00.0 Off |                  Off |
| N/A   28C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU       PID   Type   Process name                             GPU Memory |
|                                                                  Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

If `nvidia-smi` fails to report the expected output for all the NVIDIA GPUs in your system, see Troubleshooting for troubleshooting steps.

2.3.3. Configuring VMware vMotion with vGPU for VMware vSphere

NVIDIA AI Enterprise supports vGPU migration, which includes VMware vMotion and suspend-resume, for VMs that are configured with vGPU. To enable VMware vMotion with vGPU, an advanced **vCenter Server** setting must be enabled. However, suspend-resume for VMs that are configured with vGPU is enabled by default.

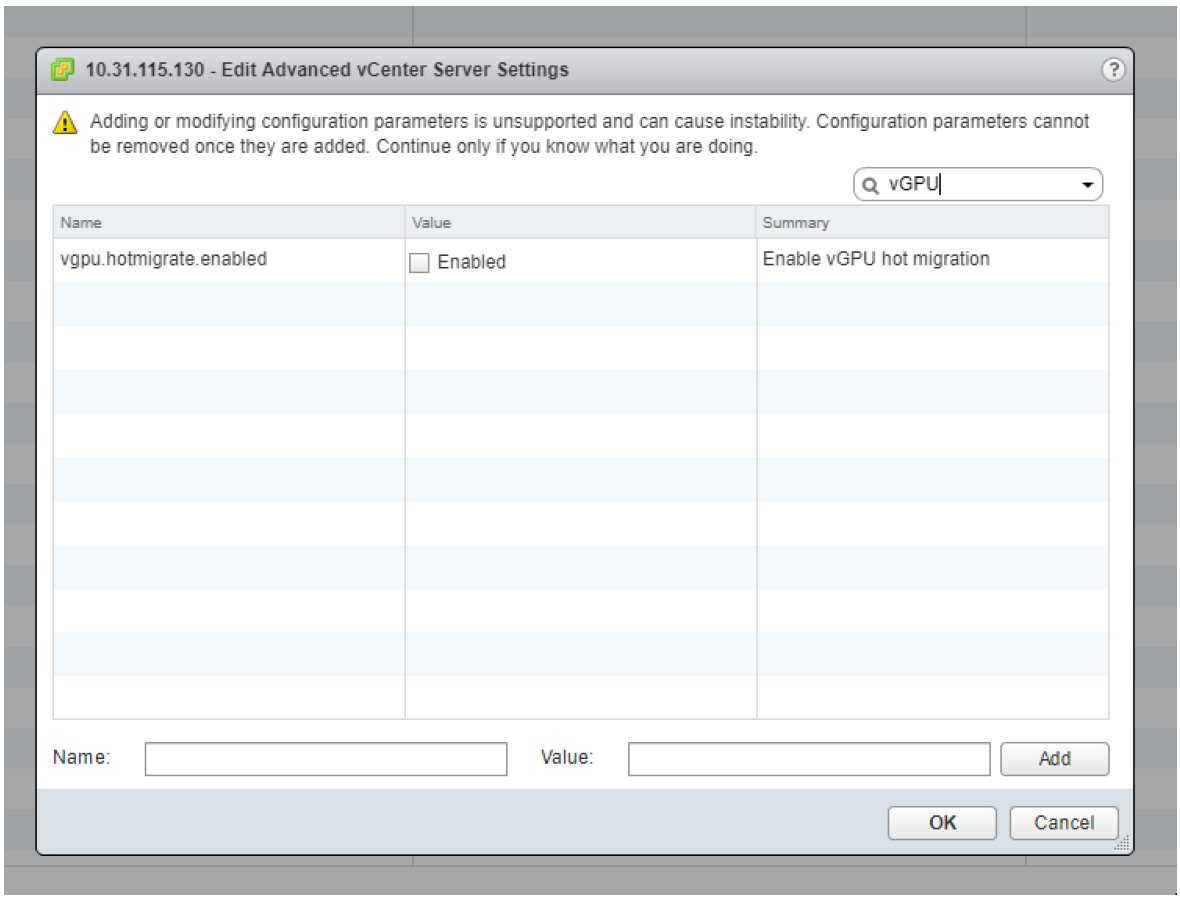
For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support vGPU migration, see *NVIDIA AI Enterprise Release Notes*.

Before configuring VMware vMotion with vGPU for an ESXi host, ensure that the current NVIDIA Virtual GPU Manager for VMware vSphere package is installed on the host.

1. Log in to **vCenter Server** by using the **vSphere Web Client**.
2. In the **Hosts and Clusters** view, select the **vCenter Server** instance.

> **Note:** Ensure that you select the **vCenter Server** instance, **not** the **vCenter Server** VM.

3. Click the **Configure** tab.
4. In the **Settings** section, select **Advanced Settings** and click **Edit**.
5. In the **Edit Advanced vCenter Server Settings** window that opens, type **vGPU** in the search field.
6. When the **vgpu.hotmigrate.enabled** setting appears, set the **Enabled** option and click **OK**.



2.3.4. Changing the Default Graphics Type in VMware vSphere 6.5 and Later

The vGPU Manager VIBs for VMware vSphere 6.5 and later provide vSGA and vGPU functionality in a single VIB. After this VIB is installed, the default graphics type is Shared, which provides vSGA functionality. To enable vGPU support for VMs in VMware vSphere 6.5, you must change the default graphics type to Shared Direct. If you do not change the default graphics type, VMs to which a vGPU is assigned fail to start and the following error message is displayed:

```
The amount of graphics resource available in the parent resource pool is
 insufficient for the operation.
```

Note:

If you are using a supported version of VMware vSphere earlier than 6.5, or are configuring a VM to use vSGA, omit this task.

Change the default graphics type **before** configuring vGPU. Output from the VM console in the VMware vSphere Web Client is not available for VMs that are running vGPU.

Before changing the default graphics type, ensure that the ESXi host is running and that all VMs on the host are powered off.

1. Log in to vCenter Server by using the vSphere Web Client.
2. In the navigation tree, select your ESXi host and click the **Configure** tab.
3. From the menu, choose **Graphics** and then click the **Host Graphics** tab.
4. On the **Host Graphics** tab, click **Edit**.

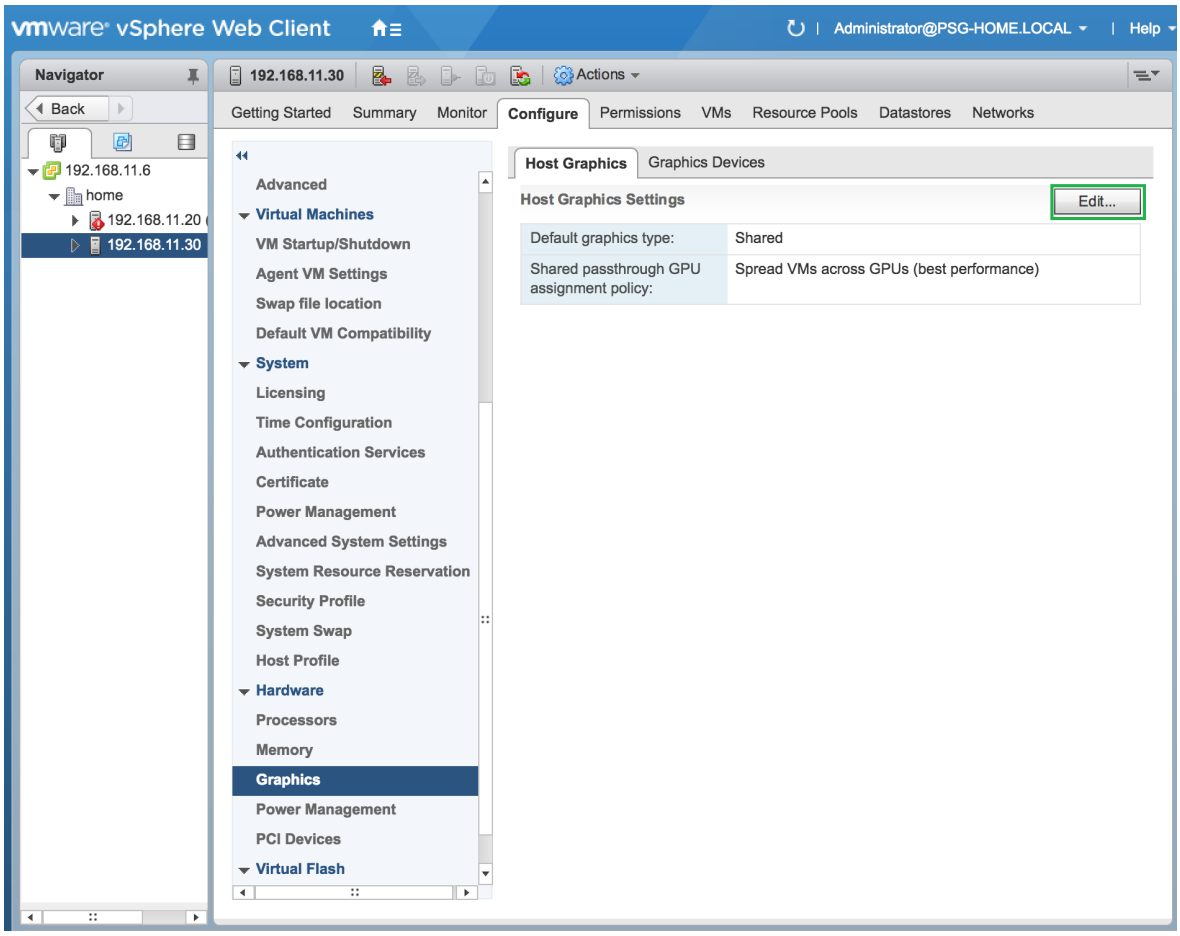
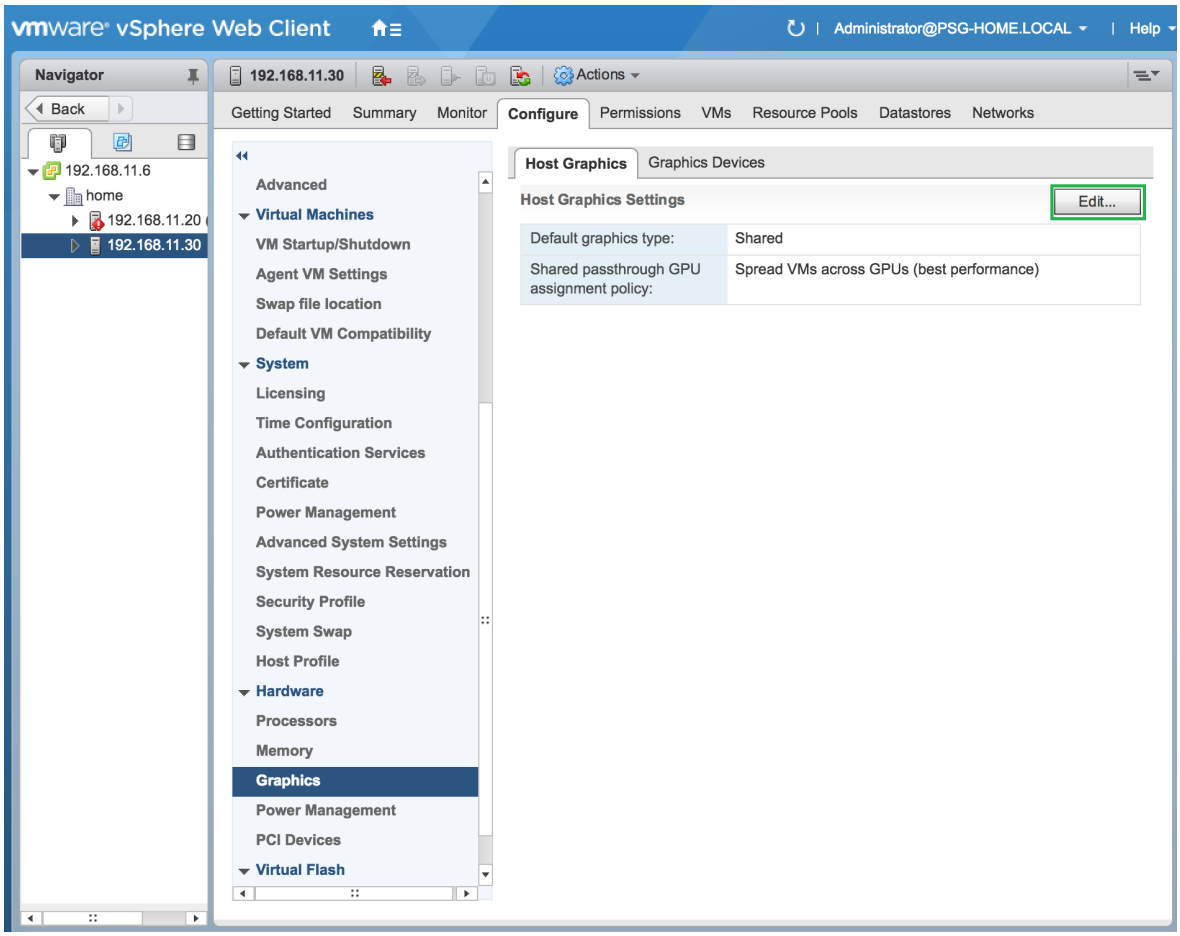


Figure 5. Shared default graphics type



5. In the **Edit Host Graphics Settings** dialog box that opens, select **Shared Direct** and click **OK**.

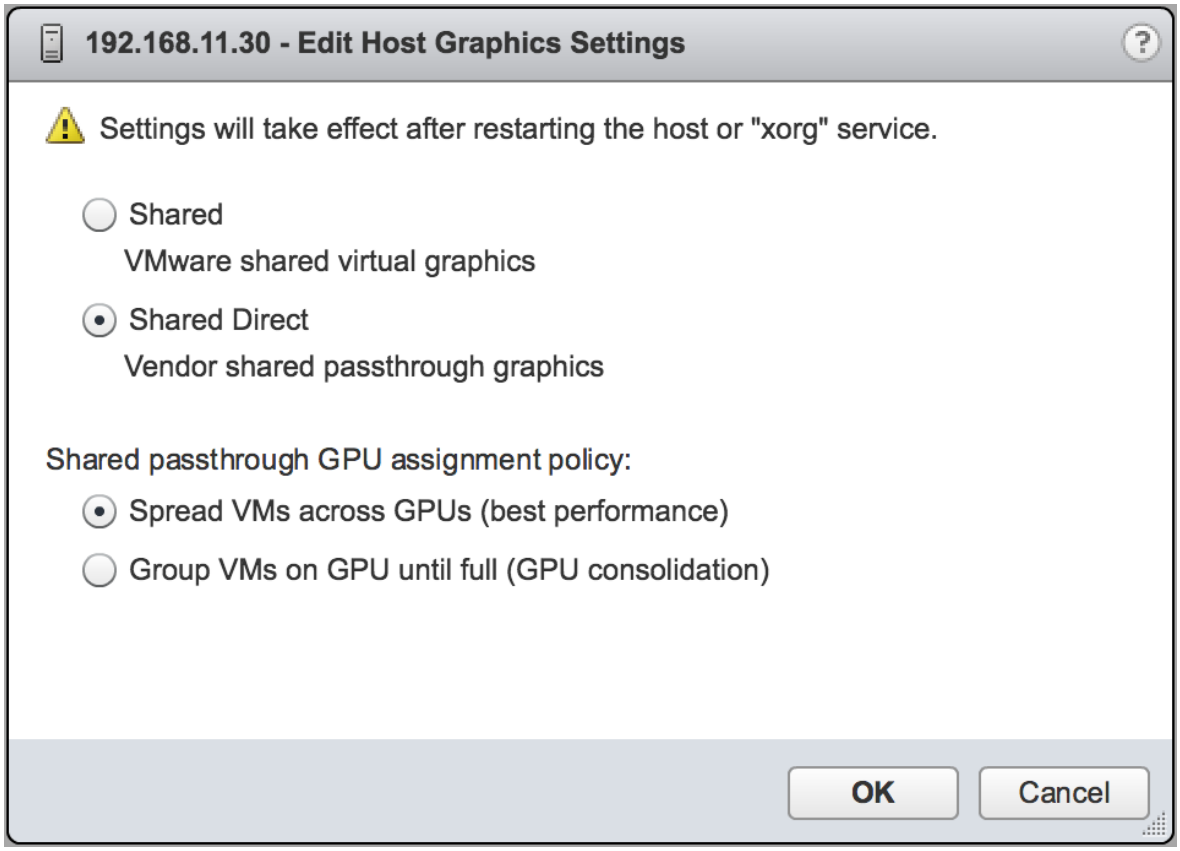
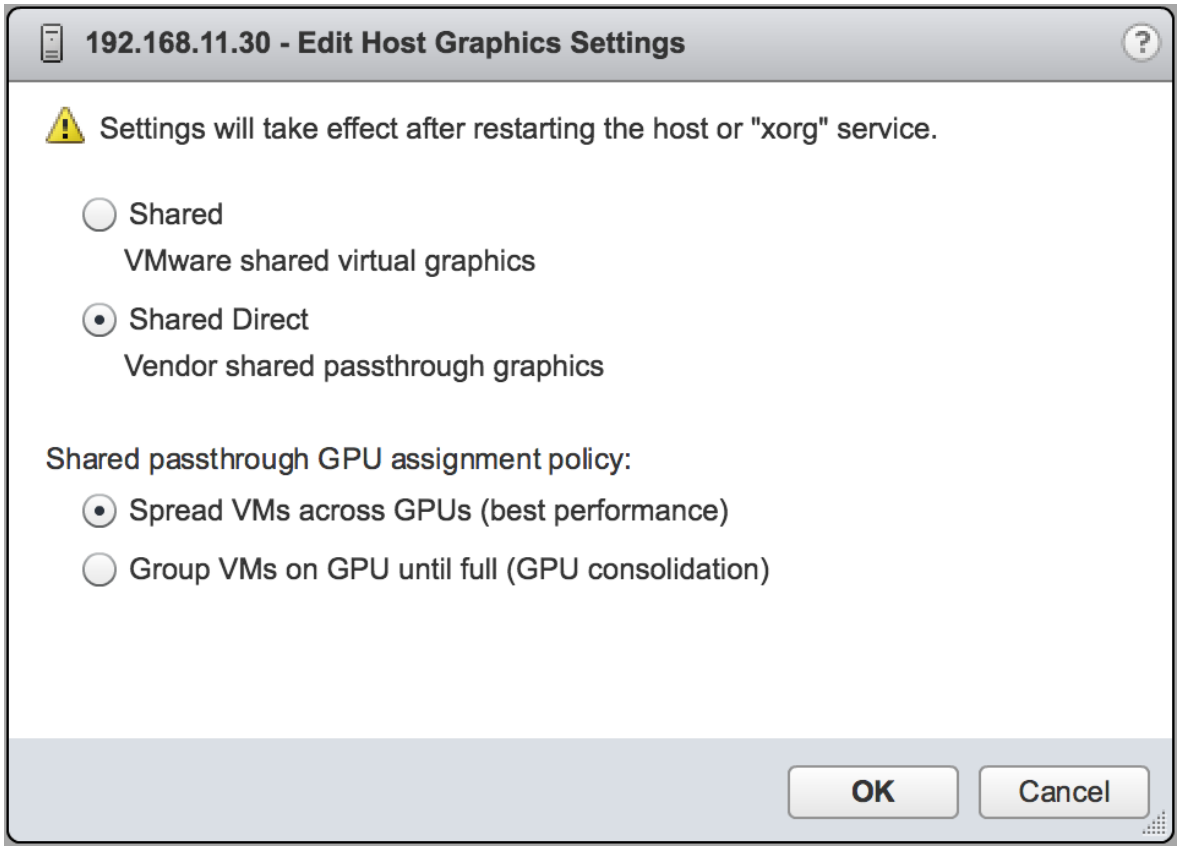

192.168.11.30 - Edit Host Graphics Settings
Settings will take effect after restarting the host or "xorg" service.
Shared
VMware shared virtual graphics
Shared Direct
Vendor shared passthrough graphics
Shared passthrough GPU assignment policy:
Spread VMs across GPUs (best performance)
Group VMs on GPU until full (GPU consolidation)
OK
Cancel

Figure 6. Host graphics settings for vGPU



Note: In this dialog box, you can also change the allocation scheme for vGPU-enabled VMs. For more information, see Modifying GPU Allocation Policy on VMware vSphere.

After you click OK, the default graphics type changes to Shared Direct.

6. Click the **Graphics Devices** tab to verify the configured type of each physical GPU on which you want to configure vGPU.

   The configured type of each physical GPU must be Shared Direct. For any physical GPU for which the configured type is Shared, change the configured type as follows:

   a). On the **Graphics Devices** tab, select the physical GPU and click the **Edit icon**.

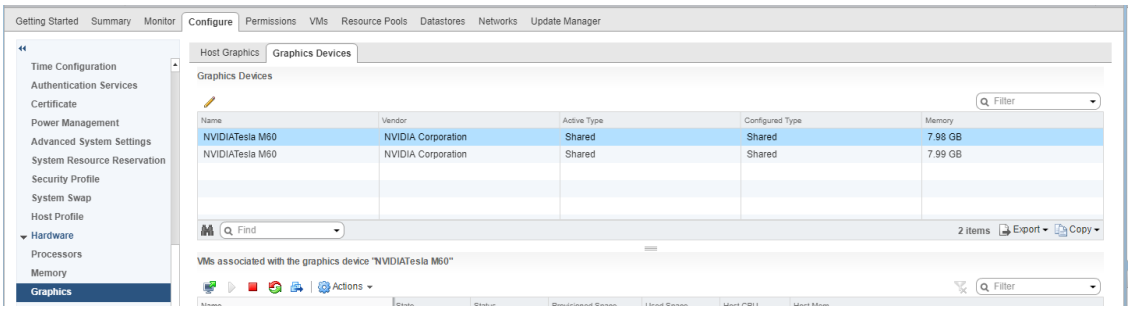
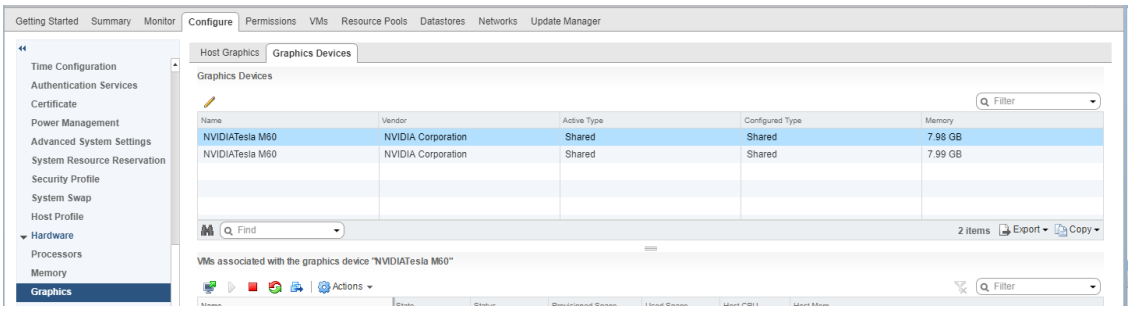


Figure 7. Shared graphics type



b). In the **Edit Graphics Device Settings** dialog box that opens, select **Shared Direct** and click **OK**.

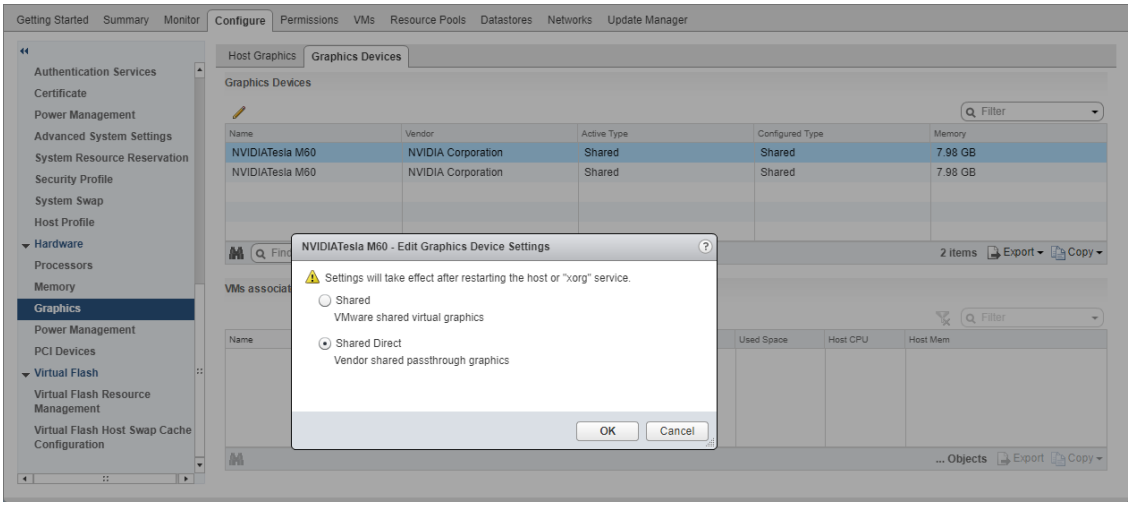
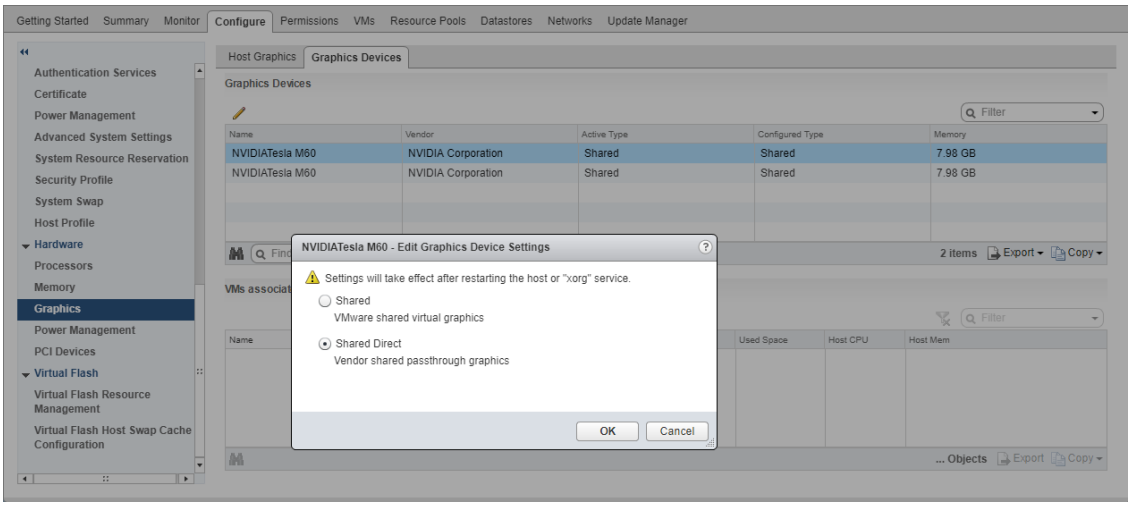


Figure 8. Graphics device settings for a physical GPU



7. Restart the ESXi host **or** stop and restart the Xorg service if necessary and `nv-hostengine` on the ESXi host.

   To stop and restart the Xorg service and `nv-hostengine`, perform these steps:

   a). **VMware vSphere releases before 7.0 Update 1 only:** Stop the Xorg service.

   As of VMware vSphere 7.0 Update 1, the Xorg service is no longer required for graphics devices in NVIDIA vGPU mode.

   b). Stop `nv-hostengine`.

   ```
   [root@esxi:~] nv-hostengine -t
   ```

   c). Wait for 1 second to allow `nv-hostengine` to stop.

   d). Start `nv-hostengine`.

   ```
   [root@esxi:~] nv-hostengine -d
   ```

   e). **VMware vSphere releases before 7.0 Update 1 only:** Start the Xorg service.

   As of VMware vSphere 7.0 Update 1, the Xorg service is no longer required for graphics devices in NVIDIA vGPU mode.

   ```
   [root@esxi:~] /etc/init.d/xorg start
   ```

8. In the **Graphics Devices** tab of the VMware vCenter Web UI, confirm that the active type and the configured type of each physical GPU are Shared Direct.

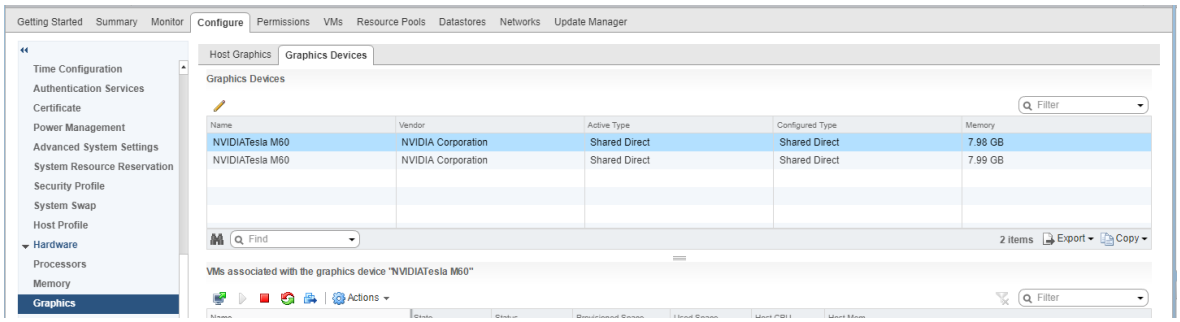
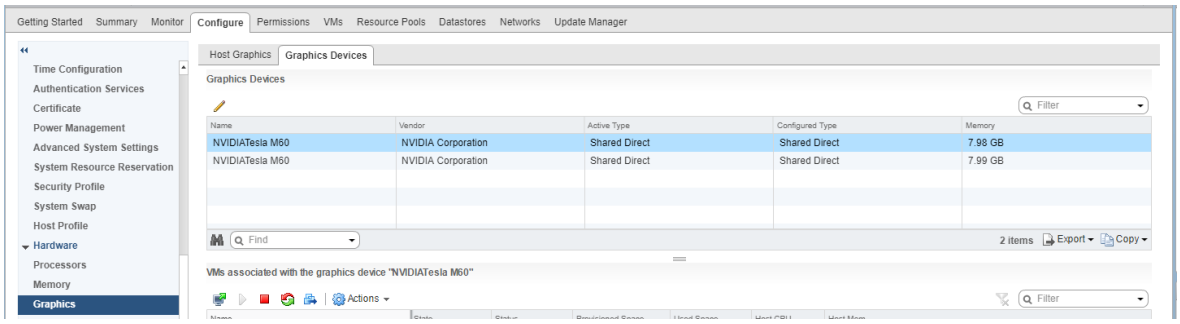


Figure 9. Shared direct graphics type



After changing the default graphics type, configure vGPU as explained in Configuring a vSphere VM with NVIDIA vGPU.

See also the following topics in the VMware vSphere documentation:

- Log in to vCenter Server by Using the vSphere Web Client
- Configuring Host Graphics

2.3.5. Configuring a GPU for MIG-Backed vGPUs

To support GPU instances with NVIDIA vGPU, a GPU must be configured with MIG mode enabled and GPU instances must be created and configured on the physical GPU. Optionally, you can create compute instances within the GPU instances. If you don't create compute instances within the GPU instances, they can be added later for individual vGPUs from within the guest VMs.

Ensure that the following prerequisites are met:

- The NVIDIA Virtual GPU Manager is installed on the hypervisor host.
- You have root user privileges on your hypervisor host machine.
- You have determined which GPU instances correspond to the vGPU types of the MIG-backed vGPUs that you will create.

  To get this information, consult the table of MIG-backed vGPUs for your GPU in Virtual GPU Types for Supported GPUs*NVIDIA AI Enterprise User Guide*.
- The GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

After configuring a GPU for MIG-backed vGPUs, create the vGPUs that you need and add them to their VMs.

2.3.5.1. Enabling MIG Mode for a GPU

Perform this task in your hypervisor command shell.

1. Open a command shell as the root user on your hypervisor host machine.

   On all supported hypervisors, you can use secure shell (SSH) for this purpose. Individual hypervisors may provide additional means for logging in. For details, refer to the documentation for your hypervisor.

   You can use secure shell (SSH) for this purpose.
2. Determine whether MIG mode is enabled.

   Use the `nvidia-smi` command for this purpose. By default, MIG mode is disabled.

   This example shows that MIG mode is disabled on GPU 0.

   > **Note:** In the output from `nvidia-smi`, the NVIDIA A100 HGX 40GB GPU is referred to as A100-SXM4-40GB.

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63     Driver Version: 470.63      CUDA Version:  11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      On   | 00000000:36:00.0 Off |                    0 |
| N/A   29C    P0    62W / 400W |      0MiB / 40537MiB |      6%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

3. If MIG mode is disabled, enable it.

```
$ nvidia-smi -i [gpu-ids] -mig 1
```

   gpu-ids

   A comma-separated list of GPU indexes, PCI bus IDs or UUIDs that specifies the GPUs on which you want to enable MIG mode. If *gpu-ids* is omitted, MIG mode is enabled on all GPUs on the system.

   This example enables MIG mode on GPU 0.

```
$ nvidia-smi -i 0 -mig 1
Enabled MIG Mode for GPU 00000000:36:00.0
All done.
```

   > **Note:** If the GPU is being used by another process, this command fails and displays a warning message that MIG mode for the GPU is in the pending enable state. In this situation, stop all processes that are using the GPU and retry the command.

4. **VMware vSphere ESXi only:** Reboot the hypervisor host.
5. Query the GPUs on which you enabled MIG mode to confirm that MIG mode is enabled.

   This example queries GPU 0 for the PCI bus ID and MIG mode in comma-separated values (CSV) format.

```
$ nvidia-smi -i 0 --query-gpu=pci.bus_id,mig.mode.current --format=csv
pci.bus_id, mig.mode.current
00000000:36:00.0, Enabled
```

2.3.5.2. Disabling MIG Mode for One or More GPUs

If a GPU that you want to use for time-sliced vGPUs or GPU pass through has previously been configured for MIG-backed vGPUs, disable MIG mode on the GPU.

Ensure that the following prerequisites are met:

- The NVIDIA Virtual GPU Manager is installed on the hypervisor host.
- You have root user privileges on your hypervisor host machine.
- The GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

Perform this task in your hypervisor command shell.

1. Open a command shell as the root user on your hypervisor host machine.

   You can use secure shell (SSH) for this purpose.
2. Determine whether MIG mode is disabled.

   Use the `nvidia-smi` command for this purpose. By default, MIG mode is disabled, but might have previously been enabled.

   This example shows that MIG mode is enabled on GPU 0.

   Note: In the output from output from `nvidia-smi`, the NVIDIA A100 HGX 40GB GPU is referred to as A100-SXM4-40GB.

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63      Driver Version: 470.63     CUDA Version:  11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:36:00.0 Off |                    0 |
| N/A   29C    P0    62W / 400W |      0MiB / 40537MiB |      6%      Default |
|                               |                      |             Enabled |
+-------------------------------+----------------------+----------------------+
```

3. If MIG mode is enabled, disable it.

```
$ nvidia-smi -i [gpu-ids] -mig 0
```

   gpu-ids

   A comma-separated list of GPU indexes, PCI bus IDs or UUIDs that specifies the GPUs on which you want to disable MIG mode. If *gpu-ids* is omitted, MIG mode is disabled on all GPUs on the system.

   This example disables MIG mode on GPU 0.

```
$ sudo nvidia-smi -i 0 -mig 0
Disabled MIG Mode for GPU 00000000:36:00.0
All done.
```

4. Confirm that MIG mode was disabled.

   Use the `nvidia-smi` command for this purpose.

   This example shows that MIG mode is disabled on GPU 0.

```
$ nvidia-smi -i 0
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63      Driver Version: 470.63     CUDA Version:  11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:36:00.0 Off |                    0 |
| N/A   29C    P0    62W / 400W |      0MiB / 40537MiB |      6%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

2.3.6. Configuring a vSphere VM with NVIDIA vGPU

To support applications and workloads that are compute or graphics intensive, you can add multiple vGPUs to a single VM.

For details about which VMware vSphere versions and NVIDIA vGPUs support the assignment of multiple vGPUs to a VM, see *NVIDIA AI Enterprise Release Notes*.

If you upgraded to VMware vSphere 6.7 Update 3 from an earlier version and are using VMs that were created with that version, change the VM compatibility to **vSphere 6.7 Update 2 and later**. For details, see Virtual Machine Compatibility in the VMware documentation.

If you are adding multiple vGPUs to a single VM, perform this task for each vGPU that you want to add to the VM.

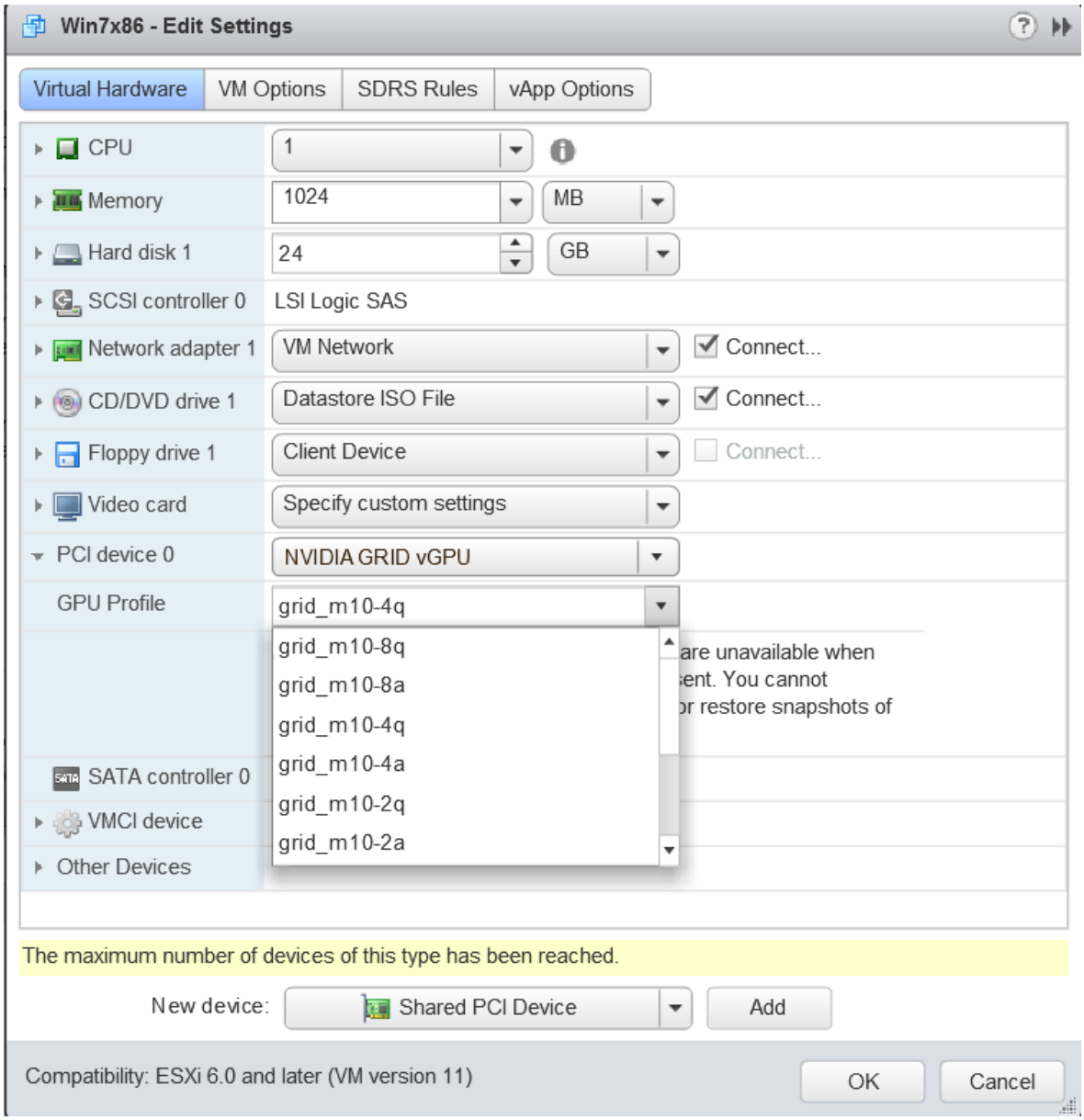
CAUTION: Output from the VM console in the VMware vSphere Web Client is not available for VMs that are running vGPU. Make sure that you have installed an alternate means of accessing the VM (such as VMware Horizon or a VNC server) before you configure vGPU.

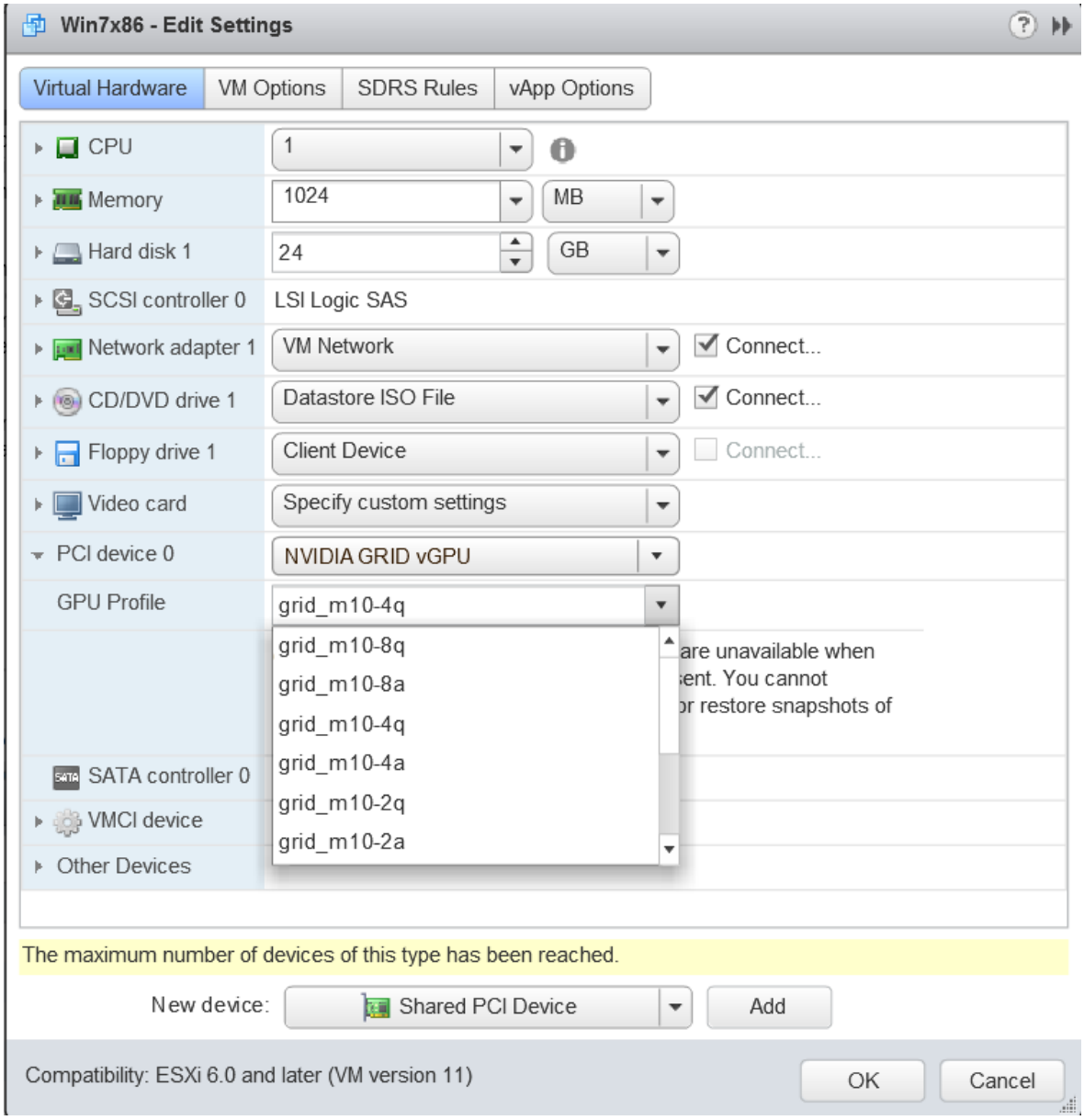
VM console in vSphere Web Client will become active again once the vGPU parameters are removed from the VM's configuration.

Note: If you are configuring a VM to use VMware vSGA, omit this task.

1. Open the vCenter Web UI.
2. In the vCenter Web UI, right-click the VM and choose **Edit Settings**.
3. Click the **Virtual Hardware** tab.
4. In the **New device** list, select **Shared PCI Device** and click **Add**.
   The **PCI device** field should be auto-populated with **`NVIDIA GRID vGPU`**.

Figure 10. VM settings for vGPU





5. From the **GPU Profile** drop-down menu, choose the type of vGPU you want to configure and click **OK**.

> **Note:** VMware vSphere does **not** support vCS. Therefore, C-series vGPU types are not available for selection from the **GPU Profile** drop-down menu.

6. Ensure that VMs running vGPU have all their memory reserved:
   a). Select **Edit virtual machine settings** from the vCenter Web UI.
   b). Expand the **Memory** section and click **Reserve all guest memory (All locked)**.

After you have configured a vSphere VM with a vGPU, start the VM. VM console in vSphere Web Client is not supported in this vGPU release. Therefore, use VMware Horizon or VNC to access the VM's desktop.

After the VM has booted, install the NVIDIA AI Enterprise graphics driver as explained in #unique_32Installing and Licensing NVIDIA AI Enterprise Components Required in a Guest VM.

2.3.7. Setting vGPU Plugin Parameters on VMware vSphere

Plugin parameters for a vGPU control the behavior of the vGPU, such as the frame rate limiter (FRL) configuration in frames per second or whether console virtual network computing (VNC) for the vGPU is enabled. The VM to which the vGPU is assigned is started with these parameters. If parameters are set for multiple vGPUs assigned to the same VM, the VM is started with the parameters assigned to each vGPU.

Ensure that the VM to which the vGPU is assigned is powered off.

For each vGPU for which you want to set plugin parameters, perform this task in the **vSphere Client**. vGPU plugin parameters are PCI pass through configuration parameters in advanced VM attributes.

1. In the **vSphere Client**, browse to the VM to which the vGPU is assigned.
2. Context-click the VM and choose **Edit Settings**.
3. In the **Edit Settings** window, click the **VM Options** tab.
4. From the **Advanced** drop-down list, select **Edit Configuration**.
5. In the **Configuration Parameters** dialog box, click **Add Row**.
6. In the **Name** field, type the parameter name **`pciPassthru`*`vgpu-id`*`.cfg.`*`parameter`***, in the **Value** field type the parameter value, and click **OK**.

   vgpu-id

   A positive integer that identifies the vGPU assigned to a VM. For the first vGPU assigned to a VM, *vgpu-id* is **`0`**. For example, if two vGPUs are assigned to a VM and you are setting a plugin parameter for both vGPUs, set the following parameters:

   - **`pciPassthru0.cfg.`*`parameter`***
   - **`pciPassthru1.cfg.`*`parameter`***

   parameter

   The name of the vGPU plugin parameter that you want to set. For example, the name of the vGPU plugin parameter for enabling unified memory is **`enable_uvm`**.

   To enable unified memory for two vGPUs that are assigned to a VM, set **`pciPassthru0.cfg.enable_uvm`** and **`pciPassthru1.cfg.enable_uvm`** to 1.

2.4. Disabling and Enabling ECC Memory

Some GPUs that support NVIDIA AI Enterprise support error correcting code (ECC) memory with NVIDIA vGPU. ECC memory improves data integrity by detecting and handling double-bit errors. However, not all GPUs, vGPU types, and hypervisor software versions support ECC memory with NVIDIA vGPU.

On GPUs that support ECC memory with NVIDIA vGPU, ECC memory is supported with C-series and Q-series vGPUs, but not with A-series and B-series vGPUs. Although A-series and B-series vGPUs start on physical GPUs on which ECC memory is enabled, enabling ECC with vGPUs that do not support it might incur some costs.

On physical GPUs that do not have HBM2 memory, the amount of frame buffer that is usable by vGPUs is reduced. All types of vGPU are affected, not just vGPUs that support ECC memory.

The effects of enabling ECC memory on a physical GPU are as follows:

- ECC memory is exposed as a feature on all supported vGPUs on the physical GPU.
- In VMs that support ECC memory, ECC memory is enabled, with the option to disable ECC in the VM.
- ECC memory can be enabled or disabled for individual VMs. Enabling or disabling ECC memory in a VM does not affect the amount of frame buffer that is usable by vGPUs.

GPUs based on the Pascal GPU architecture and later GPU architectures support ECC memory with NVIDIA vGPU. To determine whether ECC memory is enabled for a GPU, run **`nvidia-smi -q`** for the GPU.

Tesla M60 and M6 GPUs support ECC memory when used without GPU virtualization, but NVIDIA vGPU does not support ECC memory with these GPUs. In graphics mode, these GPUs are supplied with ECC memory disabled by default.

Some hypervisor software versions do not support ECC memory with NVIDIA vGPU.

If you are using a hypervisor software version or GPU that does not support ECC memory with NVIDIA vGPU and ECC memory is enabled, NVIDIA vGPU fails to start. In this situation, you must ensure that ECC memory is disabled on all GPUs if you are using NVIDIA vGPU.

2.4.1. Disabling ECC Memory

If ECC memory is unsuitable for your workloads but is enabled on your GPUs, disable it. You must also ensure that ECC memory is disabled on all GPUs if you are using NVIDIA vGPU with a hypervisor software version or a GPU that does not support ECC memory with NVIDIA vGPU. If your hypervisor software version or GPU does not support ECC memory and ECC memory is enabled, NVIDIA vGPU fails to start.

Where to perform this task depends on whether you are changing ECC memory settings for a physical GPU or a vGPU.

- For a physical GPU, perform this task from the hypervisor host.

- For a vGPU, perform this task from the VM to which the vGPU is assigned.

> **Note:** ECC memory must be enabled on the physical GPU on which the vGPUs reside.

Before you begin, ensure that NVIDIA Virtual GPU Manager is installed on your hypervisor. If you are changing ECC memory settings for a vGPU, also ensure that the NVIDIA AI Enterprise graphics driver is installed in the VM to which the vGPU is assigned.

1. Use `nvidia-smi` to list the status of all physical GPUs or vGPUs, and check for ECC noted as enabled.

```
# nvidia-smi -q

==============NVSMI LOG==============

Timestamp                           : Mon Aug 30 18:36:45 2021
Driver Version                      : 470.63

Attached GPUs                       : 1
GPU 0000:02:00.0

[...]

    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled

[...]
```

2. Change the ECC status to off for each GPU for which ECC is enabled.
   - If you want to change the ECC status to off for all GPUs on your host machine or vGPUs assigned to the VM, run this command:

     ```
     # nvidia-smi -e 0
     ```

   - If you want to change the ECC status to off for a specific GPU or vGPU, run this command:

     ```
     # nvidia-smi -i id -e 0
     ```

     id is the index of the GPU or vGPU as reported by `nvidia-smi`.

     This example disables ECC for the GPU with index `0000:02:00.0`.

     ```
     # nvidia-smi -i 0000:02:00.0 -e 0
     ```

3. Reboot the host or restart the VM.
4. Confirm that ECC is now disabled for the GPU or vGPU.

```
# nvidia–smi –q

==============NVSMI LOG==============

Timestamp                           : Mon Aug 30 18:37:53 2021
Driver Version                      : 470.63

Attached GPUs                       : 1
GPU 0000:02:00.0
[...]

    Ecc Mode
        Current                     : Disabled
        Pending                     : Disabled

[...]
```

If you later need to enable ECC on your GPUs or vGPUs, follow the instructions in Enabling ECC Memory.

2.4.2. Enabling ECC Memory

If ECC memory is suitable for your workloads and is supported by your hypervisor software and GPUs, but is disabled on your GPUs or vGPUs, enable it.

Where to perform this task depends on whether you are changing ECC memory settings for a physical GPU or a vGPU.

- For a physical GPU, perform this task from the hypervisor host.
- For a vGPU, perform this task from the VM to which the vGPU is assigned.

> **Note:** ECC memory must be enabled on the physical GPU on which the vGPUs reside.

Before you begin, ensure that NVIDIA Virtual GPU Manager is installed on your hypervisor. If you are changing ECC memory settings for a vGPU, also ensure that the NVIDIA AI Enterprise graphics driver is installed in the VM to which the vGPU is assigned.

1. Use `nvidia-smi` to list the status of all physical GPUs or vGPUs, and check for ECC noted as disabled.

```
# nvidia-smi -q

==============NVSMI LOG==============

Timestamp                           : Mon Aug 30 18:36:45 2021
Driver Version                      : 470.63

Attached GPUs                       : 1
GPU 0000:02:00.0

[...]

    Ecc Mode
        Current                     : Disabled
        Pending                     : Disabled

[...]
```

2. Change the ECC status to on for each GPU or vGPU for which ECC is enabled.
   - If you want to change the ECC status to on for all GPUs on your host machine or vGPUs assigned to the VM, run this command:

     ```
     # nvidia-smi -e 1
     ```

   - If you want to change the ECC status to on for a specific GPU or vGPU, run this command:

     ```
     # nvidia-smi -i id -e 1
     ```

     id is the index of the GPU or vGPU as reported by `nvidia-smi`.

     This example enables ECC for the GPU with index `0000:02:00.0`.

     ```
     # nvidia-smi -i 0000:02:00.0 -e 1
     ```

3. Reboot the host or restart the VM.
4. Confirm that ECC is now enabled for the GPU or vGPU.

```
# nvidia-smi -q

==============NVSMI LOG==============

Timestamp                           : Mon Aug 30 18:37:53 2021
Driver Version                      : 470.63

Attached GPUs                       : 1
GPU 0000:02:00.0
[...]

    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled

[...]
```

If you later need to disable ECC on your GPUs or vGPUs, follow the instructions in Disabling ECC Memory.

Chapter 3. Installing and Licensing NVIDIA AI Enterprise Components Required in a Guest VM

3.1. Installing NVIDIA AI Enterprise Software Components by Using Kubernetes

Perform this task if you are using one of the following combinations of guest operating system and container platform:

- Ubuntu with Kubernetes

Ensure that the following prerequisites are met:

1. If you are using Kubernetes, ensure that:
   a). Kubernetes is installed in the VM.
   b). NVIDIA vGPU Manager is installed.
   c). NVIDIA vGPU License Server with licenses is installed.
2. Helm is installed.
3. You have generated your NGC API key for accessing the NVIDIA Enterprise Collection at the URL provided to you by NVIDIA.

3.1.1. Installing and Licensing the NVIDIA vGPU Software Graphics Driver by Using NVIDIA GPU Operator

Installation of the NVIDIA AI Enterprise GPU Operator is documented at:

https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#nvidia-ai-enterprise

3.1.2. Transforming Container Images for AI and Data Science Applications and Frameworks into Kubernetes Pods

The AI and data science applications and frameworks are distributed as NGC container images through the NGC private registry. If you are using Kubernetes or Red Hat OpenShift, you must transform each image that you want to use into a Kubernetes pod. Each container image contains the entire user-space software stack that is required to run the application or framework, namely, the CUDA libraries, cuDNN, any required Magnum IO components, TensorRT, and the framework.

3.2. Install NVIDIA AI Enterprise Software Components by Using Docker

Perform this task if you are using Ubuntu with Docker.

3.2.1. Installing and Licensing the NVIDIA vGPU Software Graphics Driver Natively

Perform this task in the guest VM.

1. Install the vGPU software graphics driver for Linux on the VM that you downloaded from the NVIDIA Licensing Portal.
2. License the NVIDIA vGPU.
3. Install NVIDIA Container Toolkit.

3.2.2. Installing AI and Data Science Applications and Frameworks by Using Docker

The AI and data science applications and frameworks are distributed as NGC container images through the NGC private registry. Each container image contains the entire user-space software stack that is required to run the application or framework, namely, the CUDA libraries, cuDNN, any required Magnum IO components, TensorRT, and the framework.

Perform this task in the guest VM.

1. Access the NGC private registry.
2. Load the container for each AI or data science application that you are interested in.

The following table lists the `docker pull` command for downloading the container for each application or framework.

Application or Framework	Docker `pull` Command
NVIDIA TensorRT	`docker pull nvcr.io/nvaie/tensorrt-1-1:21.08-py3`
NVIDIA Triton Inference Server	`docker pull nvcr.io/nvaie/tritonserver-1-1:21.08-py3-sdk`
NVIDIA Triton Inference Server	`docker pull nvcr.io/nvaie/tritonserver-1-1:21.08-py3-min`
NVIDIA Triton Inference Server	`docker pull nvcr.io/nvaie/tritonserver-1-1:21.08-py3`
PyTorch	`docker pull nvcr.io/nvaie/pytorch-1-1:21.08-nvaie1.1-py3`
RAPIDS	`docker pull nvcr.io/nvaie/nvidia-rapids-1-1:21.08-cuda11.4-ubuntu20.04-py3.8`
TensorFlow 1	`docker pull nvcr.io/nvaie/tensorflow-1-1:21.08-nvaie1.1-tf1-py3`
TensorFlow 2	`docker pull nvcr.io/nvaie/tensorflow-1-1:21.08-nvaie1.1-tf2-py3`

Other Software	Docker `pull` Command
GPU Operator	`docker pull nvcr.io/nvaie/gpu-operator-1-1:v1.9.1`
Network Operator	`docker pull nvcr.io/nvaie/network-operator-1-1:v1.1.0`
vGPU Guest Driver, Ubuntu	`docker pull nvcr.io/nvaie/vgpu-guest-driver-1-1:470.82.01-ubuntu20.04`

3.3. Installing and Licensing NVIDIA AI Enterprise Components Natively

3.3.1. Installing the NVIDIA AI Enterprise Graphics Driver on Ubuntu from a Debian Package

The NVIDIA AI Enterprise graphics driver for Ubuntu is distributed as a Debian package file. This task requires `sudo` privileges.

1. Copy the NVIDIA AI Enterprise Linux driver package, for example `nvidia-linux-grid-470_470.63.01_amd64.deb`, to the guest VM where you are installing the driver.
2. Log in to the guest VM as a user with `sudo` privileges.
3. Open a command shell and change to the directory that contains the NVIDIA AI Enterprise Linux driver package.
4. From the command shell, run the command to install the package.

```
$ sudo apt-get install ./nvidia-linux-grid-470_470.63.01_amd64.deb
```

5. Verify that the NVIDIA driver is operational.
   a). Reboot the system and log in.
   b). After the system has rebooted, confirm that you can see your NVIDIA vGPU device in the output from the `nvidia-smi` command.

```
$ nvidia-smi
```

3.3.2. Prerequisites for Configuring a Licensed Client of NVIDIA License System with a Networked License

A client with a network connection obtains a license by leasing it from a NVIDIA License System service instance. The service instance serves the license to the client over the network from a pool of floating licenses obtained from the NVIDIA Licensing Portal. The license is returned to the service instance when the licensed client no longer requires the license.

Note: NVIDIA AI Enterprise releases earlier than 13.0 do **not** support NVIDIA License System. For full details of NVIDIA AI Enterprise releases that support NVIDIA License System, refer to .

Before configuring a licensed client, ensure that the following prerequisites are met:

- The NVIDIA AI Enterprise graphics driver is installed on the client.
- The client configuration token that you want to deploy on the client has been created from the NVIDIA Licensing Portal or the DLS as explained in *NVIDIA License System User Guide*.
- Ports 443 and 80 in your firewall or proxy must be open to allow HTTPS traffic between a service instance and its the licensed clients. These ports must be open for both CLS instances and DLS instances.

  Note: For DLS releases **before** DLS 1.1, ports 8081 and 8082 were also required to be open to allow HTTPS traffic between a DLS instance and its licensed clients. Although these ports are no longer required, they remain supported for backward compatibility.

The graphics driver creates a default location in which to store the client configuration token on the client. If you want to use this location for the client configuration token and, on Windows, are configuring the client with NVIDIA vGPU, you can configure the client with default settings. Otherwise, you must configure the client with custom settings as explained in #unique_46.

The process for configuring a licensed client is the same for CLS and DLS instances but depends on the OS that is running on the client.

3.3.2.1. Configuring a Licensed Client with a Networked License on Linux with Default Settings

Perform this task from the client.

1. As root, open the file `/etc/nvidia/gridd.conf` in a plain-text editor, such as `vi`.

```
$ sudo vi /etc/nvidia/gridd.conf
```

Note: You can create the `/etc/nvidia/gridd.conf` file by copying the supplied template file `/etc/nvidia/gridd.conf.template`.

2. Add the `FeatureType` configuration parameter to the file `/etc/nvidia/gridd.conf` on a new line as `FeatureType="value"`.

   value depends on the type of the GPU assigned to the licensed client that you are configuring.

GPU Type	Value
NVIDIA vGPU	1. NVIDIA AI Enterprise automatically selects the correct type of license based on the vGPU type.
Physical GPU	The feature type of a GPU in pass-through mode or a bare-metal deployment: ▶ 0: NVIDIA Virtual Applications ▶ 2: NVIDIA RTX Virtual Workstation ▶ 4: NVIDIA Virtual Compute Server

Note: You can also perform this step from **NVIDIA X Server Settings**. Before using **NVIDIA X Server Settings** to perform this step, ensure that this option has been enabled as explained in *#unique_48NVIDIA AI Enterprise Client Licensing User Guide*.

This example shows how to configure a licensed Linux client for .

```
# /etc/nvidia/gridd.conf.template - Configuration file for NVIDIA Grid Daemon
…
# Description: Set Feature to be enabled
# Data type: integer
# Possible values:
# 0 => for unlicensed state
# 1 => for NVIDIA vGPU
# 2 => for NVIDIA RTX Virtual Workstation
# 4 => for NVIDIA Virtual Compute Server
FeatureType=
...
```

3. Copy the client configuration token to the `/etc/nvidia/ClientConfigToken` directory.
4. Ensure that the file access modes of the client configuration token allow the owner to read, write, and execute the token, and the group and others only to read the token.
   a). Determine the current file access modes of the client configuration token.
   ```
   # ls -l client-configuration-token-directory
   ```
   b). If necessary, change the mode of the client configuration token to 744.
   ```
   # chmod 744 client-configuration-token-directory/client_configuration_token_*.tok
   ```

   client-configuration-token-directory
   The directory to which you copied the client configuration token in the previous step.
5. Save your changes to the `/etc/nvidia/gridd.conf` file and close the file.
6. Restart the `nvidia-gridd` service.

The NVIDIA service on the client should now automatically obtain a license from the CLS or DLS instance.

3.3.2.2. Verifying the NVIDIA AI Enterprise License Status of a Licensed Client

After configuring a client with an NVIDIA AI Enterprise license, verify the license status by displaying the licensed product name and status.

To verify the license status of a licensed client, run `nvidia-smi` with the `-q` or `--query` option. If the product is licensed, the expiration date is shown in the license status.

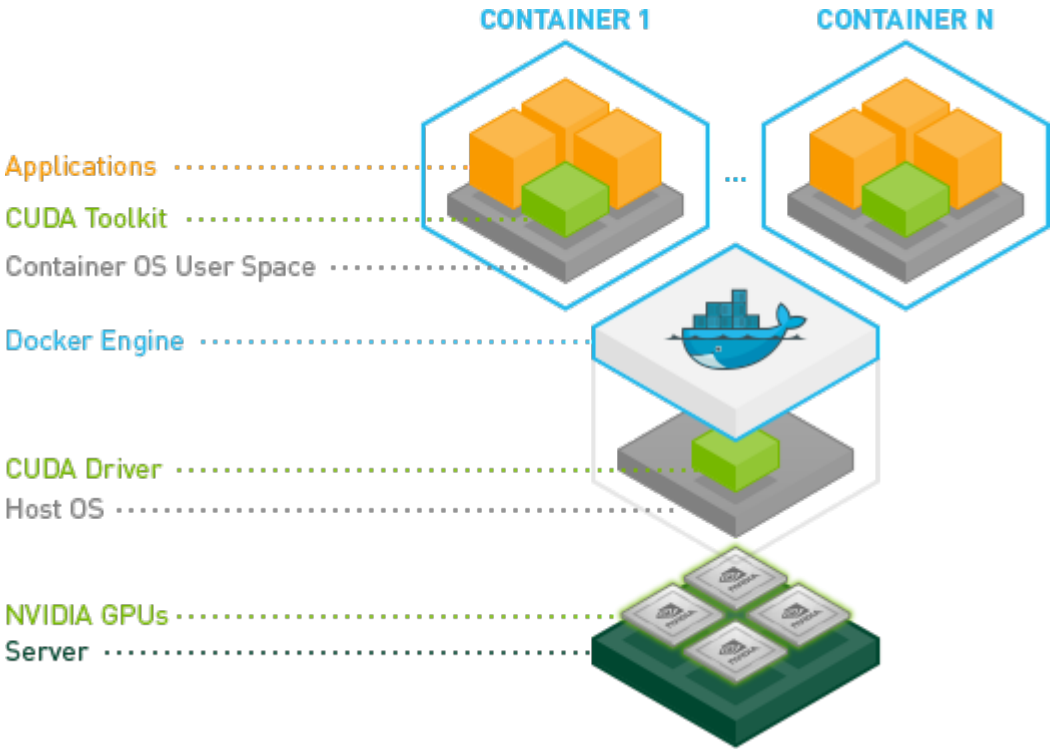
```
nvidia-smi -q
==============NVSMI LOG==============

Timestamp                           : Wed Mar 31 01:49:28 2020
Driver Version                      : 440.88
CUDA Version                        : 10.0

Attached GPUs                       : 1
GPU 00000000:00:08.0
    Product Name                    : Tesla T4
    Product Brand                   : Grid
    Display Mode                    : Enabled
    Display Active                  : Disabled
    Persistence Mode                : N/A
    Accounting Mode                 : Disabled
    Accounting Mode Buffer Size     : 4000
    Driver Model
        Current                     : WDDM
        Pending                     : WDDM
    Serial Number                   : 0334018000638
    GPU UUID                        : GPU-ba2310b6-95d1-802b-f96f-5865410fe517
    Minor Number                    : N/A
    VBIOS Version                   : 90.04.21.00.01
    MultiGPU Board                  : No
    Board ID                        : 0x8
    GPU Part Number                 : 699-2G183-0200-100
    Inforom Version
        Image Version               : G183.0200.00.02
        OEM Object                  : 1.1
        ECC Object                  : 5.0
        Power Management Object     : N/A
    GPU Operation Mode
        Current                     : N/A
        Pending                     : N/A
    GPU Virtualization Mode
        Virtualization mode         : Pass-Through
    vGPU Software Licensed Product
        Product Name                : NVIDIA Virtual Compute ServerGRID vGaming
        License Status              : Licensed (Expiry: 2021-11-13 18:29:59 GMT)
    …
    …
```

3.3.3. Installing NVIDIA Container Toolkit

Use NVIDIA Container Toolkit to build and run GPU accelerated Docker containers. The toolkit includes a container runtime library and utilities to configure containers to use NVIDIA GPUs automatically.



Ensure that the following software is installed in the guest VM:

- Docker 20.10 for your Linux distribution. For instructions, refer to Install Docker Engine on Ubuntu in the Docker product manuals.
- The NVIDIA AI Enterprise graphics driver. For instructions, refer to Installing the NVIDIA AI Enterprise Graphics Driver on Ubuntu from a Debian Package.

Note: You do **not** need to install NVIDIA CUDA Toolkit on the hypervisor host.

1. Set up the GPG key and configure `apt` to use NVIDIA Container Toolkit packages in the file `/etc/apt/sources.list.d/nvidia-docker.list`.

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
$ curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

2. Download information from all configured sources about the latest versions of the packages and install the `nvidia-container-toolkit` package.

```
$ sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit
```

3. Restart the Docker service.

```
$ sudo systemctl restart docker
```

3.3.4. Verifying the Installation of NVIDIA Container Toolkit

1. Run the `nvidia-smi` command contained in the latest official NVIDIA CUDA Toolkit image.
```
$ docker run --gpus all nvidia/cuda:11.0-base nvidia-smi
```
2. Start a GPU-enabled container on any two available GPUs.
```
$ docker run --gpus 2 nvidia/cuda:11.0-base nvidia-smi
```
3. Start a GPU-enabled container on two specific GPUs identified by their index numbers.
```
$ docker run --gpus '"device=1,2"' nvidia/cuda:10.0-base nvidia-smi
```
4. Start a GPU-enabled container on two specific GPUs with one GPU identified by its UUID and the other GPU identified by its index number.
```
$ docker run --gpus '"device=UUID-ABCDEF,1"' nvidia/cuda:11.0-base nvidia-smi
```
5. Specify a GPU capability for the container.
```
$ docker run --gpus all,capabilities=utility nvidia/cuda:11.0-base nvidia-smi
```

3.3.5. Installing Software Distributed as Container Images

The NGC container images accessed through the NVIDIA Enterprise Catalog includes the AI and data science applications, frameworks, and software in the infrastructure optimization and cloud native deployment layers. Each container image for an AI and data science application or framework contains the entire user-space software stack that is required to run the application or framework; namely, the CUDA libraries, cuDNN, any required Magnum IO components, TensorRT, and the framework.

Ensure that you have completed the following tasks in *NGC Private Registry User Guide*:

- Generating Your NGC API Key
- Accessing the NGC Container Registry

Perform this task from the VM.

For each AI or data science application that you are interested in, load the container as explained in Uploading an NVIDIA Container Image onto Your System in *NGC Private Registry User Guide*.

The following table lists the Docker pull command for downloading the container for each application or framework.

Application or Framework	Docker `pull` Command
NVIDIA TensorRT	`docker pull nvcr.io/nvaie/tensorflow-1-1:21.08-nvaie1.1-tf1-py3`
NVIDIA Triton Inference Server	`docker pull nvcr.io/nvaie/tritonserver:21.08-py3-sdk`
NVIDIA Triton Inference Server	`docker pull nvcr.io/nvaie/tritonserver-1-1:21.08-py3-min`
NVIDIA Triton Inference Server	`docker pull nvcr.io/nvaie/tritonserver-1-1:21.08-py3`

Application or Framework	Docker `pull` Command
PyTorch	`docker pull nvcr.io/nvaie/pytorch-1-1:21.08-py3`
RAPIDS	`docker pull nvcr.io/nvaie/nvidia-rapids-1-1:21.08-cuda11.4-ubuntu20.04-py3.8`
TensorFlow 1	`docker pull nvcr.io/nvaie/tensorflow-1-1:21.08-tf1-py3`
TensorFlow 2	`docker pull nvcr.io/nvaie/tensorflow-1-1:21.08-tf2-py3`

Other Software	Docker `pull` Command
GPU Operator	`docker pull nvcr.io/nvaie/gpu-operator-1-1:v1.9.1`
Network Operator	`docker pull nvcr.io/nvaie/network-operator-1-1:v1.1.0`
vGPU Guest Driver, Ubuntu	`docker pull nvcr.io/nvaie/vgpu-guest-driver-1-1:470.82.01-ubuntu20.04`

3.3.6. Running ResNet-50 with TensorRT

This test verifies correct operation of NVIDIA Virtual Compute Server by running the ResNet-50 convolutional neural network with the TensorRT container from the NVIDIA GPU Cloud (NGC) container registry.

Note: This test does not require results to be reported for review. A `PASSED` result reported by the test is sufficient for the test to pass.

To complete this test, you need a Linux VM that is configured with a C-series vGPU and in which Docker CE 19.03 or later and the NVIDIA CUDA Toolkit are installed.

1. Pull the TensorRT NGC Container from the NGC container registry.
   a). Copy the **Pull Command** provided in the listing for this container image on the NGC website.
   b). Run the command that you copied with `sudo` privileges.

   For example, to pull version 20.03 of the container image, run the following command:

   ```
   $ sudo docker pull nvcr.io/nvidia/tensorrt:20.03-py3
   ```
2. Launch the container image that you pulled in the previous step on all vGPUs in interactive mode, specifying that the container will be deleted when it is stopped.

   For example, if you pulled version 20.03 of the container image, run the following command to launch it:

   ```
   $ sudo docker run --gpus all -it --rm nvcr.io/nvidia/tensorrt:20.03-py3
   ```
3. Launch the NVIDIA TensorRT container image on all vGPUs in interactive mode, specifying that the container will be deleted when it is stopped.

   ```
   $ sudo docker run --gpus all -it --rm nvcr.io/nvaie/tensorrt:21.07-py3
   ```

4. From within the container runtime, change to the directory that contains test data for the ResNet-50 convolutional neural network.

```
# cd /workspace/tensorrt/data/resnet50
```

5. Run the ResNet-50 convolutional neural network with FP32, FP16, and INT8 precision and confirm that each test is completed with the result `PASSED`.

   a). To run ResNet-50 with the default FP32 precision, run this command:

```
# trtexec --duration=90 --workspace=1024 --percentile=99 --avgRuns=100 \
--deploy=ResNet50_N2.prototxt --batch=1 --output=prob
```

   b). To run ResNet-50 with FP16 precision, add the `--fp16` option:

```
# trtexec --duration=90 --workspace=1024 --percentile=99 --avgRuns=100 \
--deploy=ResNet50_N2.prototxt --batch=1 --output=prob --fp16
```

   c). To run ResNet-50 with INT8 precision, add the `--int8` option:

```
# trtexec --duration=90 --workspace=1024 --percentile=99 --avgRuns=100 \
--deploy=ResNet50_N2.prototxt --batch=1 --output=prob --int8
```

6. Press **Ctrl**+**P**, **Ctrl**+**Q** to exit the container runtime and return to the Linux command shell.

3.3.7. Running ResNet-50 with TensorFlow

This test verifies correct operation of NVIDIA Virtual Compute Server by running the ResNet-50 convolutional neural network with the **TensorFlow 1** container from the NVIDIA GPU Cloud (NGC) container registry.

> **Note:** This test does not require results to be reported for review. Any set of results reported by the test is sufficient for the test to pass.

To complete this test, you need a Linux VM that is configured with a C-series vGPU and in which Docker CE 19.03 or later and the NVIDIA CUDA Toolkit are installed.

1. From the NGC container registry, pull a container image release of the TensorRT NGC Container tagged `tf1`.

   > **Note:** Ensure that you do **not** pull a container image release that is tagged `tf2`. This test runs **only** with container image releases that are tagged `tf1`.

   a). In the listing for this container image on the NGC website, click the **Tags** tab and locate the most recent container image release that is tagged `tf1`.

   b). Click the ellipsis (**...**) for the container image release and click **Pull Tag** to copy the command to pull this container image release.

   c). Run the command that you copied with `sudo` privileges.

   For example, to pull version 20.03 of the container image, run the following command:

```
$ sudo docker pull nvcr.io/nvidia/tensorflow:20.03-tf1-py3
```

2. Launch the container image that you pulled in the previous step on all vGPUs in interactive mode, specifying that the container will be deleted when it is stopped.

   For example, if you pulled version 20.03 of the container image, run the following command to launch it:

```
$ sudo docker run --gpus all -it --rm \
```

```
nvcr.io/nvidia/tensorflow:20.03-tf1-py3
```

3. Launch the **TensorFlow 1** container image on all vGPUs in interactive mode, specifying that the container will be deleted when it is stopped.

```
$ sudo docker run --gpus all -it --rm \
nvcr.io/nvaie/tensorflow:21.07-tf1-py3
```

4. From within the container runtime, change to the directory that contains test data for `cnn` example.

```
# cd /workspace/nvidia-examples/cnn
```

5. Run the ResNet-50 training test with FP16 precision.

```
# python resnet.py --layers 50 -b 64 -i 200 -u batch --precision fp16
```

6. Confirm that all operations on the application are performed correctly and that a set of results is reported when the test is completed.
7. Press **Ctrl**+**P**, **Ctrl**+**Q** to exit the container runtime and return to the Linux command shell.

3.3.8. Optional: Updating NVIDIA Container Toolkit for a MIG-Enabled vGPU

To run containers on a MIG-enabled vGPU, you must update the NVIDIA Container Toolkit. This task requires `sudo` privileges.

Perform this task from the guest VM in which you want to run containers on a MIG-enabled vGPU.

1. Set up the GPG key and configure `apt` to use NVIDIA Container Toolkit packages in the file `/etc/apt/sources.list.d/nvidia-docker.list`.

```
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \
    && distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
    && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list
 | sudo tee /etc/apt/sources.list.d/nvidia-docker.list \
    && sudo apt-get update
```

2. Install the NVIDIA Container Toolkit packages and the packages on which it depends, and restart Docker.

```
$ sudo apt-get install -y nvidia-docker2 \
    && sudo systemctl restart docker
```

3. Test the installation of the NVIDIA Container Toolkit on the VM.

```
sudo docker run -runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=mig-device nvidia/cuda
 nvidia-smi
```

mig-device

The ID of the MIG-enabled vGPU in one of the following formats:

- **MIG-*gpu-uuid/gpu-instance-id/compute-instance-id***

  gpu-uuid

  The UUID of the physical GPU, for example, `GPU-786035d5-1e85-11b2-9fec-ac9c9a792daf`.

  gpu-instance-id

  The index number the GPU instance on which the vGPU resides, for example, 0 for the first GPU instance.

  compute-instance-id

  The index number of the compute instance within the GPU instance, for example, 0 for the first compute instance.

This example sets `NVIDIA_VISIBLE_DEVICES` for compute instance 0 on a MIG-enabled vGPU on GPU instance 0 of the physical GPU with UUID **`GPU-786035d5-1e85-11b2-9fec-ac9c9a792daf`**.

```
NVIDIA_VISIBLE_DEVICES=MIG-GPU-786035d5-1e85-11b2-9fec-ac9c9a792daf/0/0
```

- ***`gpu-device-index`:`mig-device-index`***

  gpu-device-index
  The index number the physical GPU.

  mig-device-index
  The index number the GPU instance.

Chapter 4. Configuring Multinode Scaling

Multinode scaling improves the performance of applications and frameworks, such as PyTorch and Tensorflow, that can run in a cluster of multiple hypervisor hosts.

> **Note:**
>
> Perform the tasks for configuring multinode scaling before performing the tasks in Getting Started with NVIDIA AI Enterprise.
>
> The procedures for configuring switches and NICs apply to NVIDIA Mellanox NICs and switches. If you are using other makes of NICs and switches, consult the vendor's documentation for the products that you are using.

You are free to choose how to run your training jobs in a cluster. For information about the cluster architecture that can be used to run BERT training jobs, see Multi-node BERT User Guide.

4.1. Hardware and VM Configuration Requirements for Multinode Scaling

If you are configuring multinode scaling, your hardware and VM configuration must meet some specific requirements in addition to the requirements for a single node.

4.1.1. Hardware Requirements for Multinode Scaling

In addition to the requirements for a single node, the hardware used for multinode scaling must meet the following requirements:

- An Ethernet NIC that supports RoCE is required in each VM used for multinode scaling. For best performance, NVIDIA recommends the NVIDIA® Mellanox® ConnectX®-6 Dx.
- The hypervisor hosts must be connected to a network switch that supports RoCE. For best performance, NVIDIA recommends the NVIDIA Mellanox Spectrum switch.

- One GPU is required for each VM.

  For best performance, NVIDIA recommends the NVIDIA A100 GPU.
- Each GPU on each hypervisor host must be paired with a NIC in the same NUMA node.

4.1.2. VM Requirements for Multinode Scaling

In addition to meeting the requirements for using C-Series vCS vGPUs, each VM used for multinode scaling must be assigned an RoCE NIC PCIe device.

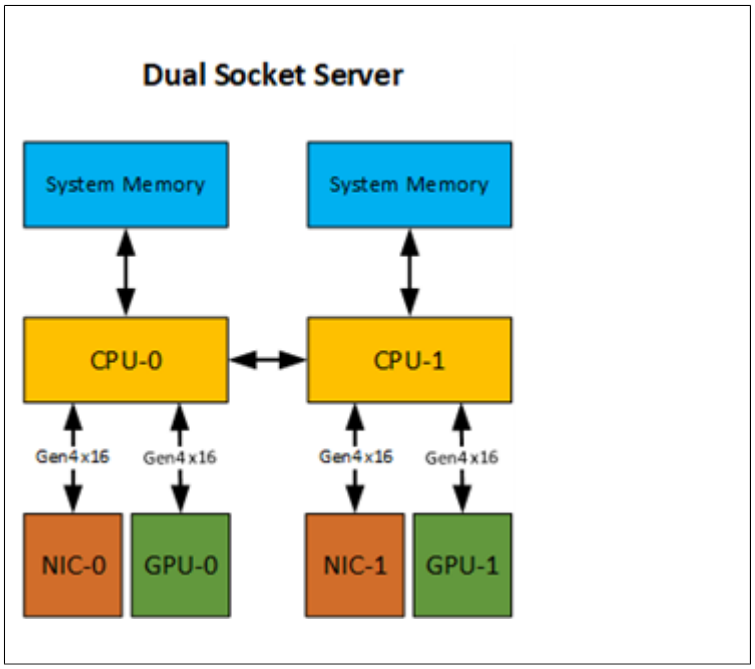
4.2. Configuring NUMA Affinity for the VMs

To ensure that your multinode setup performs adequately, each GPU on each hypervisor host must be paired with a NIC in the same NUMA node. If a GPU is not paired with a NIC in the same NUMA node, reconfigure your server hardware to ensure that this prerequisite is met.

Examples of how to configure NUMA affinity for the VMs in a two-socket server are provided for the following configurations:.

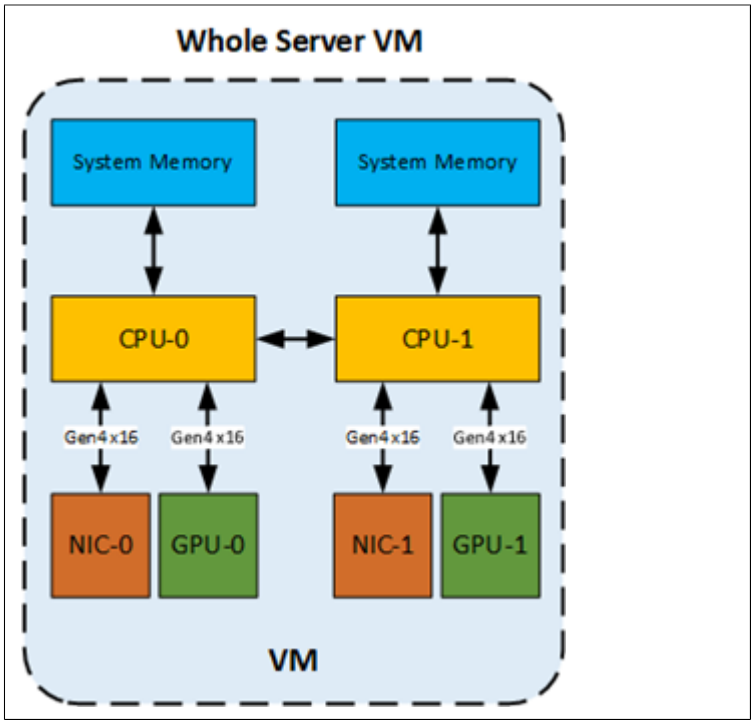
- Whole-server VM with two GPUs and two NICs across both NUMA nodes
- Per-socket VM with one GPU and one NIC paired on a single NUMA node

The hardware configuration of the server is as follows:



4.2.1. Configuring NUMA Affinity for a Whole-Server VM with Two GPUs and Two NICs Across Both NUMA Nodes

The allocation of hardware resources to a VM that is assigned the whole server is as follows:



Perform this task on each hypervisor host.

1. Determine the NUMA node to which the GPUs and NICs are attached.
   a). Determine the NUMA node to which the GPUs are attached.
   ```
   $ esxcli hardware pci list | grep -A 30 -B 10 NVIDIA
   ```
   b). Determine the NUMA node to which the NICs are attached.
   ```
   $ esxcli hardware pci list | grep -A 30 -B 10 Mellanox
   ```

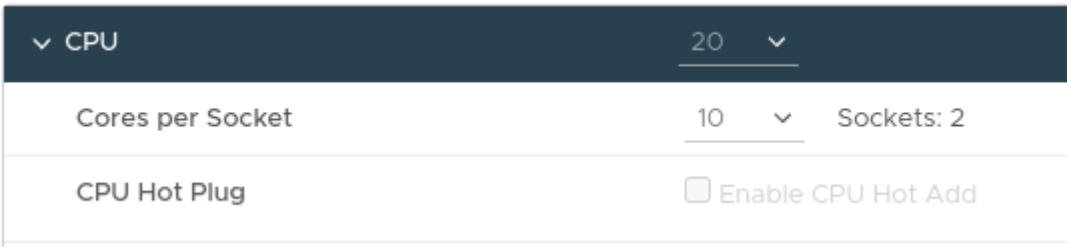
   The following output describes two GPUs. One GPU is attached to NUMA node 0 and the other GPU is attached to NUMA node 1.
   ```
   #GPU 1
   0000:37:00.0
      Address: 0000:37:00.0
      Segment: 0x0000
      Bus: 0x37
      Slot: 0x00
      Function: 0x0
      VMkernel Name: vmgfx0
      Vendor Name: NVIDIA Corporation
      Device Name: NVIDIAA100-PCIE-40GB
      Configured Owner: VMkernel
      Current Owner: VMkernel
      Vendor ID: 0x10de
   ```

```
   Device ID: 0x20f1
   SubVendor ID: 0x10de
   SubDevice ID: 0x145f
   Device Class: 0x0302
   Device Class Name: 3D controller
   Programming Interface: 0x00
   Revision ID: 0xa1
   Interrupt Line: 0xff
   IRQ: 255
   Interrupt Vector: 0x00
   PCI Pin: 0x00
   Spawned Bus: 0x00
   Flags: 0x3001
   Module ID: 50
   Module Name: nvidia
   Chassis: 0
   Physical Slot: 2
   Slot Description: PCI-E Slot 2
   Device Layer Bus Address: s00000002.00
   Passthru Capable: true
   Parent Device: PCI 0:54:0:0
   Dependent Device: PCI 0:55:0:0
   Reset Method: Bridge reset
   FPT Sharable: true
   NUMA Node: 0
   Extended Device ID: 0
   Extended Device Name:

#GPU 2
0000:86:00.0
   Address: 0000:86:00.0
   Segment: 0x0000
   Bus: 0x86
   Slot: 0x00
   Function: 0x0
   VMkernel Name: vmgfx1
   Vendor Name: NVIDIA Corporation
   Device Name: NVIDIAA100-PCIE-40GB
   Configured Owner: VMkernel
   Current Owner: VMkernel
   Vendor ID: 0x10de
   Device ID: 0x20f1
   SubVendor ID: 0x10de
   SubDevice ID: 0x145f
   Device Class: 0x0302
   Device Class Name: 3D controller
   Programming Interface: 0x00
   Revision ID: 0xa1
   Interrupt Line: 0xff
   IRQ: 255
   Interrupt Vector: 0x00
   PCI Pin: 0x00
   Spawned Bus: 0x00
   Flags: 0x3001
   Module ID: 50
   Module Name: nvidia
   Chassis: 0
   Physical Slot: 5
   Slot Description: PCI-E Slot 5
   Device Layer Bus Address: s00000005.00
   Passthru Capable: true
   Parent Device: PCI 0:133:0:0
   Dependent Device: PCI 0:134:0:0
   Reset Method: Bridge reset
   FPT Sharable: true
   NUMA Node: 1
   Extended Device ID: 0
```

```
    Extended Device Name:
```

2. Set up vCPUs for the VM so that the VM has two sockets with the vCPU cores evenly divided between the sockets.

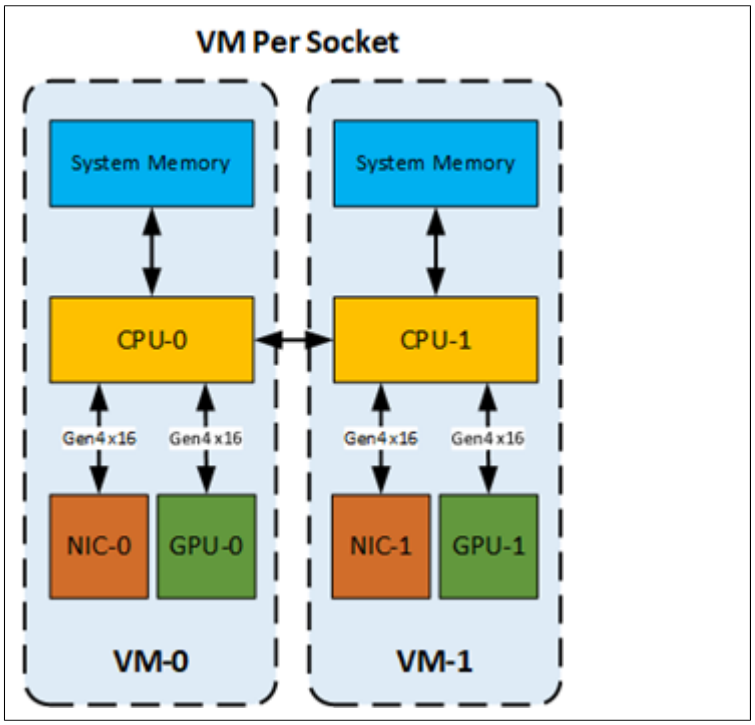


3. With two GPUs and NICs in the VM across NUMA nodes, set the NUMA affinity in the VM configuration to include both NUMA nodes 0 and 1.

```
numa.nodeAffinity = 0,1
```

4.2.2. Configuring NUMA Affinity for a Per-Socket VM with One GPU and One NIC on a Single NUMA Node

The allocation of hardware resources to the VMs that are each assigned one socket in a server is as follows:



Perform this task on each hypervisor host.

1. Determine the NUMA node to which the GPUs and NICs are attached.
   a). Determine the NUMA node to which the GPUs are attached.

```
$ esxcli hardware pci list | grep -A 30 -B 10 NVIDIA
```

b). Determine the NUMA node to which the NICs are attached.

```
$ esxcli hardware pci list | grep -A 30 -B 10 Mellanox
```

The following output describes a GPU that is attached to NUMA node 0.

```
0000:3b:02.3
   Address: 0000:3b:02.3
   Segment: 0x0000
   Bus: 0x3b
   Slot: 0x02
   Function: 0x3
   VMkernel Name: PF_0.59.0_VF_15
   Vendor Name: NVIDIA Corporation
   Device Name: NVIDIAA100-PCIE-40GB
   Configured Owner:
   Current Owner: VMkernel
   Vendor ID: 0x10de
   Device ID: 0x20f1
   SubVendor ID: 0x10de
   SubDevice ID: 0x0000
   Device Class: 0x0302
   Device Class Name: 3D controller
   Programming Interface: 0x00
   Revision ID: 0xa1
   Interrupt Line: 0xff
   IRQ: 255
   Interrupt Vector: 0x00
PCI Pin: 0xff
   Spawned Bus: 0x00
   Flags: 0x0001
   Module ID: 54
   Module Name: nvidia
   Chassis: 0
   Physical Slot: -1
   Slot Description:
   Device Layer Bus Address: s00000001.00.vf15
   Passthru Capable: true
   Parent Device: PCI 0:58:0:0
   Dependent Device: PCI 0:59:2:3
   Reset Method: Function reset
   FPT Sharable: true
   NUMA Node: 0
   Extended Device ID: 65535
   Extended Device Name:
```

2. For each GPU that you want to pair with a NIC, set the NUMA affinity in the VM configuration to the NUMA node to which the NIC and the GPU in the pair belong.

```
numa.nodeAffinity = numa-node-value
```

4.3. Configuring RoCE on the NVIDIA Mellanox Spectrum Switch

The NVIDIA Mellanox Spectrum switch must be able to run RDMA over Converged Ethernet (RoCE) over a lossless network in DSCP-based QoS mode.

Perform this task from a host computer that has an Ethernet LAN connection to the switch.

1. Use secure shell (SSH) to log in to the switch.

To obtain the username and password for logging in to the switch, consult the documentation for the switch.

2. Set the mode of the switch to RoCE.

```
switch (config) # roce
```

3. Create an isolated vLAN.

```
switch (config) # interface vlan vlan-id
```

The vLAN context is entered automatically after the vLAN is created.

The following example creates a vLAN with the identifier 111.

```
switch (config) # interface vlan 111
switch (config vlan 111) #
```

4. Exit the vLAN context.

```
switch (config vlan 111) # exit
```

5. Place the NVIDIA ConnectX NICs into the created vLAN as access ports.

```
switch (config) # interface ethernet port-range switchport access vlan-id
```

This example puts four NVIDIA ConnectX NICs into the vLAN with the identifier 111 as access ports 1/1 - 1/4.

```
switch (config) # interface ethernet 1/1-1/4 switchport access vlan 111
```

6. Set the maximum transmission unit (MTU) frame size to 9216.
   a). Disable all the ports related to the interface.

```
switch (config) # interface ethernet port-range shutdown
```

   b). Set the MTU frame size for the NVIDIA ConnectX NICs in the created vLAN to 9216.

```
switch (config) # interface ethernet port-range mtu 9216
```

   c). Enable all the ports related to the interface.

```
switch (config) # interface ethernet port-range no shutdown
```

7. If your switch is running Cumulus Linux, enable RoCE with Cumulus Linux.

4.4. Enabling GPUDirect Technology for Peer-to-Peer Connections

Enabling GPUDirect® Technology for peer-to-peer connections involves enabling Address Translation Services (ATS) in the VMware ESXi VMkernel and modifying Access Control Services (ACS) settings for the VM.

Perform this task from each hypervisor host in your multinode cluster.

1. As root, log in to the hypervisor host.
2. Update the VMkernel settings.
   a). Enable Address Translation Services (ATS) in the boot options.

```
[root@localhost:~] esxcli system settings kernel set -s atsSupport -v TRUE
```

   b). Reboot the hypervisor host.
   c). Confirm that ATS is enabled.

```
[root@localhost:~] esxcli system settings kernel list -o atsSupport
Name       Type Configured Runtime Default Description
```

```
---------- ---- ---------- ------- ------- -----------
atsSupport Bool TRUE       TRUE    FALSE   Enable Support for PCIe
                                           ATS.
```

3. Update the VM configuration.
   a). Set the option to enable peer-to-peer connections.
   ```
   pciPassthru.allowP2P=true
   ```
   b). Set the option to relax ACS settings for peer-to-peer connections.
   ```
   pciPassthru.RelaxACSforP2P=true
   ```
   When this option is set, VMware vSphere ESXi locates an ATS-capable pass-through device, finds its parent switch or root port, and enables the ACS Direct Translated bit.

4.5. Installing the Mellanox OFED Driver

Perform this task on each guest VM on each hypervisor host.

1. Install the default version of python.
   ```
   $ sudo apt install python
   ```
2. Download the compressed tar archive that contains the driver.
   ```
   $ wget \
   https://content.mellanox.com/ofed/MLNX_OFED-5.2-2.2.0.0/\
   MLNX_OFED_LINUX-5.2-2.2.0.0-ubuntu20.04-x86_64.tgz
   ```
3. Extract the contents of the compressed tar archive that contains the driver.
   ```
   $ tar xvf MLNX_OFED_LINUX-5.2-2.2.0.0-ubuntu20.04-x86_64.tgz
   ```
4. Change to the `MLNX_OFED_LINUX-5.2-2.2.0.0-ubuntu20.04-x86_64` directory.
   ```
   $ cd MLNX_OFED_LINUX-5.2-2.2.0.0-ubuntu20.04-x86_64
   ```
5. Run the script that installs the driver.
   ```
   $ sudo ./mlnxofedinstall
   ```
   During the installation process, OFED detects the ConnectX-6 NICs and updates the firmware.
6. When the installation is complete, confirm that the versions of OFED are correct.
   a). Determine the OFED version.
   ```
   $ dpkg -l | grep mlnx-ofed
   ```
   b). Determine the firmware version.
   ```
   $ cat /sys/class/infiniband/mlx5*/fw_ver
   ```
   If the firmware is not updated, download the latest firmware, update the firmware manually, and install the Mellanox OFED driver again.
7. Load the installed driver.
   ```
   $ sudo /etc/init.d/openibd restart
   ```

4.6. Enabling ATS on the NVIDIA ConnectX-6 DX NICs in a VM

Perform this task on each guest VM on each hypervisor host.

1. Change the ATS configuration to enabled on each guest VM on the hypervisor host.
   a). Start Mellanox software tools.
   ```
   $ sudo mst start
   ```
   b). Determine whether ATS is enabled.
   ```
   $ sudo mlxconfig -d /dev/mst/mt4123_pciconf0 query | grep -i ATS
   ```
   If the installed version of the firmware supports ATS, output similar to the following example is displayed.
   ```
   ATS_ENABLED                          False(0)
   ```
   If no output is displayed, the installed version of the firmware does not support ATS. In this situation, update to a version of the firmware that supports ATS.
   c). If ATS is disabled, enable it.
   ```
   $sudo mlxconfig -d /dev/mst/mt4123_pciconf0 set ATS_ENABLED=true
   Device #1:
   ----------
   Device type:    ConnectX6
   Name:           MCX653105A-HDA_Ax
   Description:    ConnectX-6 VPI adapter card; HDR IB (200Gb/s) and 200GbE;
    single-port QSFP56; PCIe4.0 x16; tall bracket; ROHS R6
   Device:        /dev/mst/mt4123_pciconf0

   Configurations:           Next Boot    New
   ATS_ENABLED               False(0)     True(1)
   Apply new Configuration? (y/n) [n] : y
   Applying... Done!
   -I- Please reboot machine to load new configurations.
   ```
2. After changing the ATS configuration to enabled on each guest VM on the node, turn off the power to the VMware vSphere ESXi host and turn the power back on again.

   Note:

   To apply the changed ATS configuration setting, you must turn off the power to the VMware vSphere ESXi host and turn the power back on again. Rebooting the host is insufficient to apply this change.
3. Start VMware vCenter Server on the hypervisor host.
4. For each VM on the node, perform the following steps:
   a). Turn on the power to the VM.
   b). Start Mellanox software tools.
   ```
   $ sudo mst start
   ```
   c). Determine whether ATS is enabled.
   ```
   $ sudo mlxconfig -d /dev/mst/mt4123_pciconf0 query | grep -i ATS
   ```
   If the installed version of the firmware supports ATS, output similar to the following example is displayed.
   ```
   ATS_ENABLED                          True(1)
   ```
   d). Obtain detailed information about all PCI buses and devices in the VM and confirm that the ATS capability of Mellanox ConnectX-6 device is shown as `Enable+`.
   ```
   $ sudo lspci -vvv
   ...
           Capabilities: [480 v1] Address Translation Service (ATS)
                   ATSCap: Invalidate Queue Depth: 00
                   ATSCtl: Enable+, Smallest Translation Unit: 00
   ...
   ```

4.7. Building and Installing the NVIDIA Peer Memory Driver

Perform this task on each guest VM on each hypervisor host.

1. If necessary, install the latest stable upstream version of Git.
   a). Add the `ppa:git-core/ppa` repository to your list of package sources.
   ```
   $ sudo add-apt-repository ppa:git-core/ppa
   ```
   b). Download information from all configured sources about the latest versions of the packages.
   ```
   $ sudo apt update
   ```
   c). Install the `git` package.
   ```
   $ sudo apt install git
   ```
2. Clone the Mellanox `nv_peer_memory` Git repository.
   ```
   $ git clone https://github.com/Mellanox/nv_peer_memory.git
   ```
3. Change to the `nv_peer_memory` directory.
   ```
   $ cd nv_peer_memory/
   ```
4. Build the NVIDIA peer memory driver software.
   ```
   $ ./build_module.sh
   ```
5. Change to the `/tmp` directory.
   ```
   $ cd /tmp/
   ```
6. Extract the NVIDIA peer memory driver software from the compressed tar archive that the build process created.
   ```
   $ tar xzf /tmp/nvidia-peer-memory_1.1.orig.tar.gz
   ```
7. Change to the `nvidia-peer-memory-1.1` directory.
   ```
   $ cd nvidia-peer-memory-1.1/
   ```
8. Build the NVIDIA peer memory driver package.
   ```
   $ dpkg-buildpackage -us -uc
   ```
9. Change to the parent of the current working directory.
   ```
   $ cd ..
   ```
10. Install the driver package that you built.
   ```
   $ sudo dpkg -i nvidia-peer-memory_1.1-0_all.deb
   ```

Chapter 5. Modifying a VM's NVIDIA vGPU Configuration

You can modify a VM's NVIDIA vGPU configuration by removing the NVIDIA vGPU configuration from a VM or by modifying GPU allocation policy.

5.1. Removing a VM's NVIDIA vGPU Configuration

Remove a VM's NVIDIA vGPU configuration when you no longer require the VM to use a virtual GPU.

5.1.1. Removing a vSphere VM's vGPU Configuration

To remove a vSphere vGPU configuration from a VM:

1. Select **Edit settings** after right-clicking on the VM in the vCenter Web UI.
2. Select the **Virtual Hardware** tab.
3. Mouse over the **PCI Device** entry showing **NVIDIA GRID vGPU** and click on the (**X**) icon to mark the device for removal.
4. Click **OK** to remove the device and update the VM settings.

5.2. Modifying GPU Allocation Policy

Citrix Hypervisor and VMware vSphere both support the *breadth first* and *depth-first* GPU allocation policies for vGPU-enabled VMs.

VMware vSphere supports the *breadth first* and *depth-first* GPU allocation policies for vGPU-enabled VMs.

breadth-first

The breadth-first allocation policy attempts to minimize the number of vGPUs running on each physical GPU. Newly created vGPUs are placed on the physical GPU that can support the new vGPU and that has the **fewest** vGPUs already resident on it. This policy generally

leads to higher performance because it attempts to minimize sharing of physical GPUs, but it may artificially limit the total number of vGPUs that can run.

depth-first
The depth-first allocation policy attempts to maximize the number of vGPUs running on each physical GPU. Newly created vGPUs are placed on the physical GPU that can support the new vGPU and that has the **most** vGPUs already resident on it. This policy generally leads to higher density of vGPUs, particularly when different types of vGPUs are being run, but may result in lower performance because it attempts to maximize sharing of physical GPUs.

Each hypervisor uses a different GPU allocation policy by default.

- Citrix Hypervisor uses the depth-first allocation policy.
- VMware vSphere ESXi uses the breadth-first allocation policy.

By default, VMware vSphere ESXi uses the breadth-first allocation policy.

If the default GPU allocation policy does not meet your requirements for performance or density of vGPUs, you can change it.

5.2.1. Modifying GPU Allocation Policy on VMware vSphere

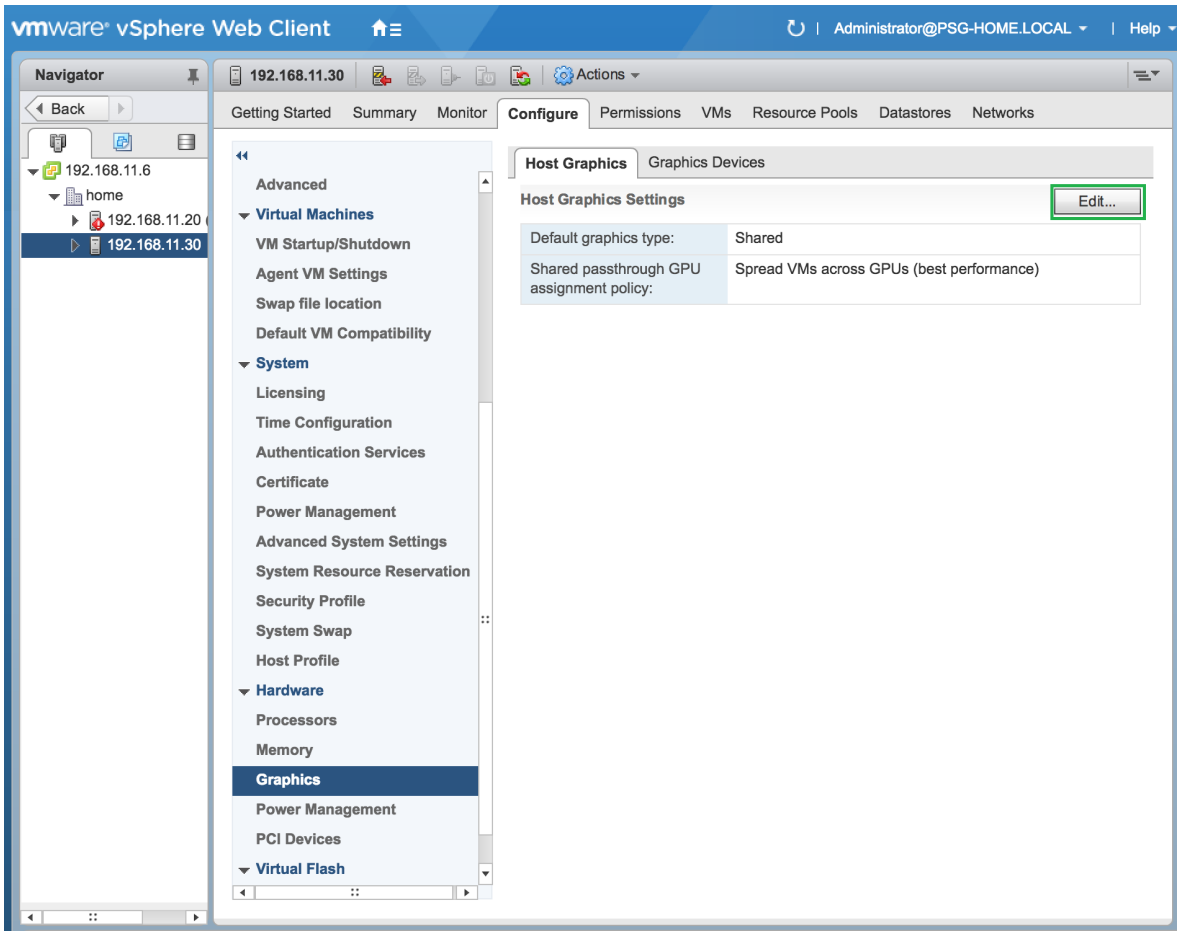
How to switch to a depth-first allocation scheme depends on the version of VMware vSphere that you are using.

- Supported versions earlier than 6.5: Add the following parameter to `/etc/vmware/config`:
  ```
  vGPU.consolidation = true
  ```
- Version 6.5: Use the vSphere Web Client.

Before using the vSphere Web Client to change the allocation scheme, ensure that the ESXi host is running and that all VMs on the host are powered off.

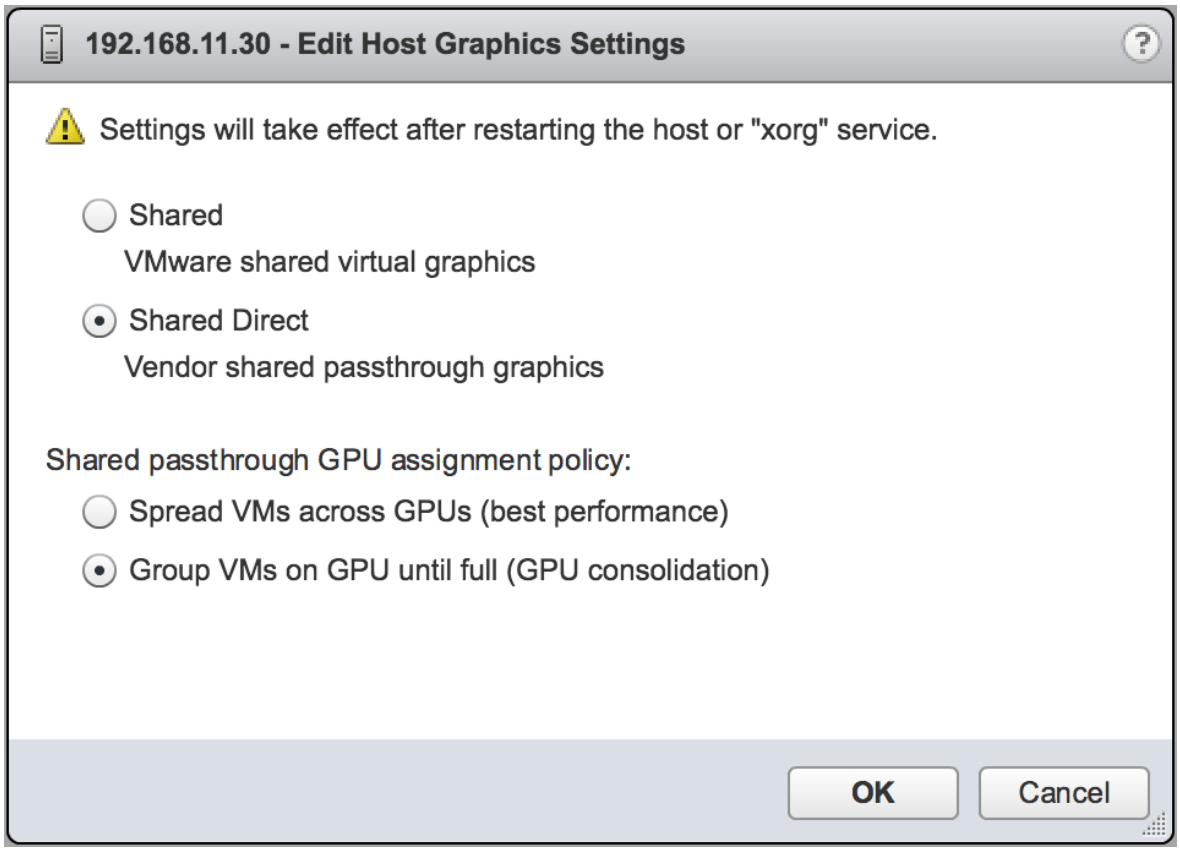
1. Log in to vCenter Server by using the vSphere Web Client.
2. In the navigation tree, select your ESXi host and click the **Configure** tab.
3. From the menu, choose **Graphics** and then click the **Host Graphics** tab.
4. On the **Host Graphics** tab, click **Edit**.

Figure 11. Breadth-first allocation scheme setting for vGPU-enabled VMs



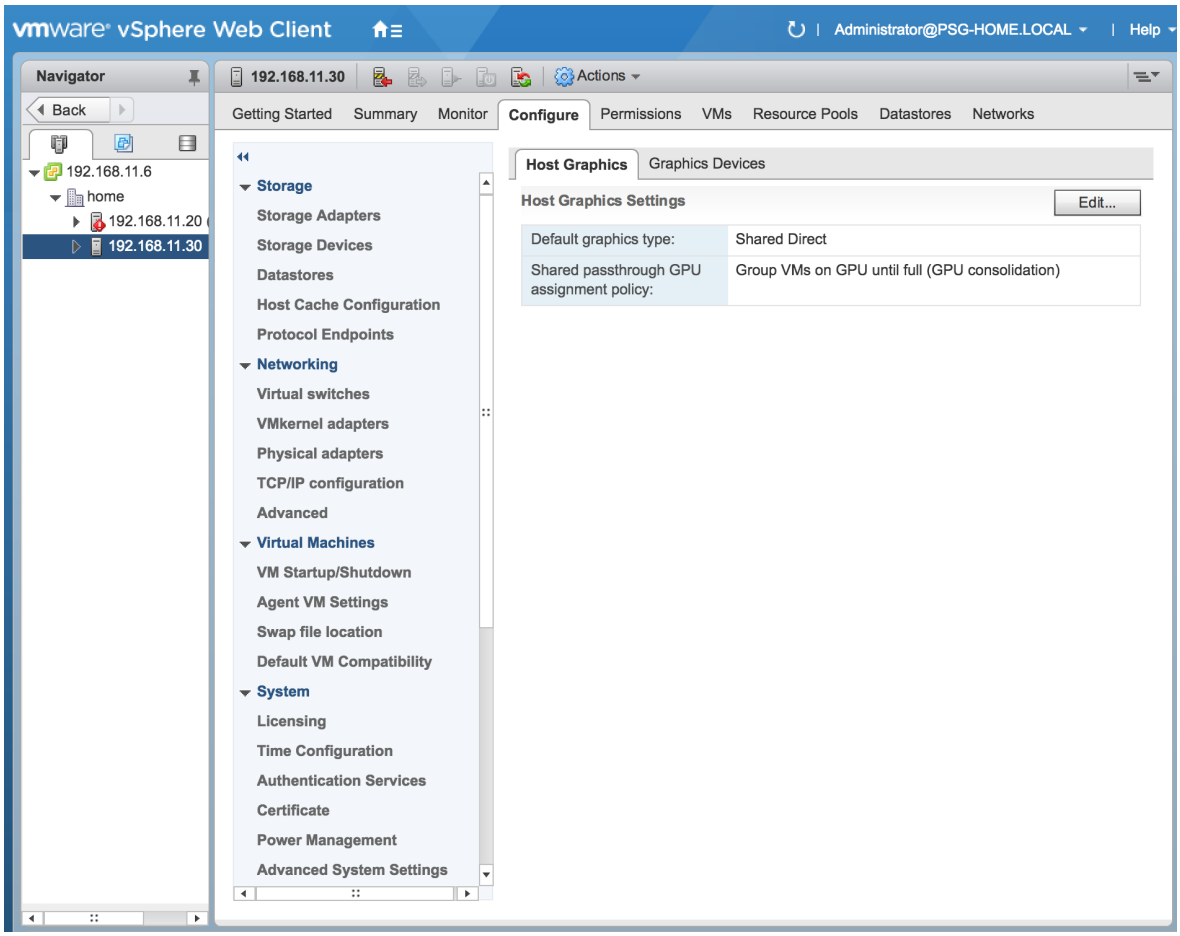
5. In the **Edit Host Graphics Settings** dialog box that opens, select these options and click **OK**.
   a). If not already selected, select **Shared Direct**.
   b). Select **Group VMs on GPU until full**.

Figure 12. Host graphics settings for vGPU



After you click OK, the default graphics type changes to Shared Direct and the allocation scheme for vGPU-enabled VMs is breadth-first.

Figure 13. Depth-first allocation scheme setting for vGPU-enabled VMs



6. Restart the ESXi host or the Xorg service on the host.

See also the following topics in the VMware vSphere documentation:

- Log in to vCenter Server by Using the vSphere Web Client
- Configuring Host Graphics

5.3. Migrating a VM Configured with vGPU

On some hypervisors, NVIDIA AI Enterprise supports migration of VMs that are configured with vGPU. Migration is supported for both time-sliced and MIG-backed vGPUs.

Before migrating a VM configured with vGPU, ensure that the following prerequisites are met:

- The VM is configured with vGPU.
- The VM is running.
- The VM obtained a suitable vGPU license when it was booted.

- The destination host has a physical GPU of the same type as the GPU where the vGPU currently resides.
- If the physical GPU supports the Multi-Instance GPU (MIG) feature, the MIG configuration of the GPU on both the source and destination hosts must be identical.
- ECC memory configuration (enabled or disabled) on both the source and destination hosts must be identical.
- The GPU topologies (including NVLink widths) on both the source and destination hosts must be identical.

Note: vGPU migration is disabled for a VM for which any of the following NVIDIA CUDA Toolkit features is enabled:

- Unified memory
- Debuggers
- Profilers

How to migrate a VM configured with vGPU depends on the hypervisor that you are using.

After migration, the vGPU type of the vGPU remains unchanged.

The time required for migration depends on the amount of frame buffer that the vGPU has. Migration for a vGPU with a large amount of frame buffer is slower than for a vGPU with a small amount of frame buffer.

5.3.1. Migrating a VM Configured with vGPU on VMware vSphere

NVIDIA AI Enterprise supports VMware vMotion for VMs that are configured with vGPU. VMware vMotion enables you to move a running virtual machine from one physical host machine to another host with very little disruption or downtime. For a VM that is configured with vGPU, the vGPU is migrated with the VM to an NVIDIA GPU on the other host. The NVIDIA GPUs on both host machines must be of the same type.

For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support suspend and resume, see *NVIDIA AI Enterprise Release Notes*.

Perform this task in the VMware vSphere web client by using the **Migration** wizard.

Before migrating a VM configured with vGPU on VMware vSphere, ensure that the following prerequisites are met:

- Your hosts are correctly configured for VMware vMotion. See Host Configuration for vMotion in the VMware documentation.
- The prerequisites listed for all supported hypervisors in Migrating a VM Configured with vGPU are met.

- NVIDIA vGPU migration is configured. See Configuring VMware vMotion with vGPU for VMware vSphere.

1. Context-click the VM and from the menu that opens, choose **Migrate**.
2. For the type of migration, select **Change compute resource only** and click **Next**.

   If you select **Change both compute resource and storage**, the time required for the migration increases.
3. Select the destination host and click **Next**.

   The destination host must have a physical GPU of the same type as the GPU where the vGPU currently resides. Furthermore, the physical GPU must be capable of hosting the vGPU. If these requirements are not met, no available hosts are listed.
4. Select the destination network and click **Next**.
5. Select the migration priority level and click **Next**.
6. Review your selections and click **Finish**.

For more information, see the following topics in the VMware documentation:

- Migrate a Virtual Machine to a New Compute Resource
- Using vMotion to Migrate vGPU Virtual Machines

If NVIDIA vGPU migration is not configured, any attempt to migrate a VM with an NVIDIA vGPU fails and a window containing the following error message is displayed:

```
Compatibility Issue/Host
Migration was temporarily disabled due to another
migration activity.
vGPU hot migration is not enabled.
```

The window appears as follows:

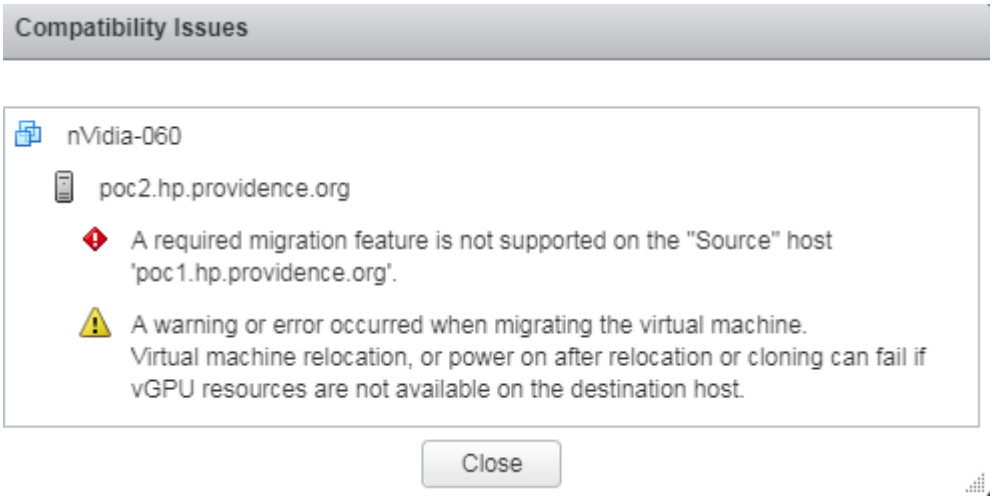
If you see this error, configure NVIDIA vGPU migration as explained in Configuring VMware vMotion with vGPU for VMware vSphere.

If your version of VMware vSpehere ESXi does not support vMotion for VMs configured with NVIDIA vGPU, any attempt to migrate a VM with an NVIDIA vGPU fails and a window containing the following error message is displayed:

```
Compatibility Issues
...
A required migration feature is not supported on the "Source" host 'host-name'.
```

```
A warning or error occurred when migrating the virtual machine.
Virtual machine relocation, or power on after relocation or cloning can fail if
vGPU resources are not available on the destination host.
```

The window appears as follows:



For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support suspend and resume, see *NVIDIA AI Enterprise Release Notes*.

5.3.2. Suspending and Resuming a VM Configured with vGPU on VMware vSphere

NVIDIA AI Enterprise supports suspend and resume for VMs that are configured with vGPU.

For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support suspend and resume, see *NVIDIA AI Enterprise Release Notes*.

Perform this task in the VMware vSphere web client.

- To suspend a VM, context-click the VM that you want to suspend, and from the context menu that pops up, choose **Power** > **Suspend** .
- To resume a VM, context-click the VM that you want to resume, and from the context menu that pops up, choose **Power** > **Power On** .

5.4. Modifying a MIG-Backed vGPU's Configuration

If compute instances weren't created within the GPU instances when the GPU was configured for MIG-backed vGPUs, you can add the compute instances for an individual vGPU from within the guest VM. If you want to replace the compute instances that were created when the GPU

was configured for MIG-backed vGPUs, you can delete them before adding the compute instances from within the guest VM.

Ensure that the following prerequisites are met:

- You have root user privileges in the guest VM.
- The GPU instance is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

Perform this task in a guest VM command shell.

1. Open a command shell as the root user in the guest VM.

   You can use secure shell (SSH) for this purpose.
2. List the available GPU instance.

```
$ nvidia-smi mig -lgi
+-------------------------------------------------------+
| GPU instances:                                        |
| GPU   Name          Profile  Instance   Placement     |
|                       ID       ID       Start:Size    |
|=======================================================|
|   0  MIG 2g.10gb       0        0          0:8       |
+-------------------------------------------------------+
```

3. **Optional:** If compute instances were created when the GPU was configured for MIG-backed vGPUs that you no longer require, delete them.

```
$ nvidia-smi mig -dci -ci compute-instance-id -gi gpu-instance-id
```

   compute-instance-id
   The ID of the compute instance that you want to delete.

   gpu-instance-id
   The ID of the GPU instance from which you want to delete the compute instance.

   > **Note:** If the GPU instance is being used by another process, this command fails. In this situation, stop all processes that are using the GPU instance and retry the command.

   This example deletes compute instance 0 from GPU instance 0 on GPU 0.

```
$ nvidia-smi mig -dci -ci 0 -gi 0
Successfully destroyed compute instance ID  0 from GPU  0 GPU instance ID  0
```

4. List the compute instance profiles that are available for your GPU instance.

```
$ nvidia-smi mig -lcip
```

   This example shows that one `MIG 2g.10gb` compute instance or two `MIG 1c.2g.10gb` compute instances can be created within the GPU instance.

```
$ nvidia-smi mig -lcip
+-----------------------------------------------------------------------------+
| Compute instance profiles:                                                  |
| GPU    GPU      Name          Profile  Instances   Exclusive       Shared   |
|      Instance                   ID     Free/Total     SM       DEC   ENC   OFA |
|        ID                                                       CE    JPEG   |
|=============================================================================|
|   0     0       MIG 1c.2g.10gb   0      2/2           14        1     0     0  |
|                                                                   2     0      |
+-----------------------------------------------------------------------------+
|   0     0       MIG 2g.10gb      1*     1/1           28        1     0     0  |
|                                                                   2     0      |
+-----------------------------------------------------------------------------+
```

5. Create the compute instances that you need within the available GPU instance.

Create each compute instance individually by running the following command.

```
$ nvidia-smi mig -cci compute-instance-profile-id -gi gpu-instance-id
```

compute-instance-profile-id
The compute instance profile ID that specifies the compute instance.

gpu-instance-id
The GPU instance ID that specifies the GPU instance within which you want to create the compute instance.

> **Note:** If the GPU instance is being used by another process, this command fails. In this situation, stop all processes that are using the GPU and retry the command.

This example creates a `MIG 2g.10gb` compute instance on GPU instance 0.

```
$ nvidia-smi mig -cci 1 -gi 0
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  0 using
 profile MIG 2g.10gb (ID  1)
```

This example creates two `MIG 1c.2g.10gb` compute instances on GPU instance 0 by running the same command twice.

```
$ nvidia-smi mig -cci 0 -gi 0
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  0 using
 profile MIG 1c.2g.10gb (ID  0)
$ nvidia-smi mig -cci 0 -gi 0
Successfully created compute instance ID  1 on GPU  0 GPU instance ID  0 using
 profile MIG 1c.2g.10gb (ID  0)
```

6. Verify that the compute instances were created within the GPU instance.

   Use the `nvidia-smi` command for this purpose.

   This example confirms that a `MIG 2g.10gb` compute instance was created on GPU instance 0.

```
nvidia-smi
Mon Aug 30 19:01:24 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63      Driver Version: 470.63     CUDA Version:  11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GRID A100X-2-10C     On  | 00000000:00:08.0 Off |                    On |
| N/A   N/A    P0    N/A /  N/A |   1058MiB / 10235MiB |     N/A      Default |
|                               |                      |              Enabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared       |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                      |
|==================+======================+===========+======================|
|  0    0   0   0  |   1058MiB / 10235MiB | 28      0 |  2   0    1    0    0 |
|                  |      0MiB /  4096MiB |           |                      |
+------------------+----------------------+-----------+----------------------+

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

```
+-----------------------------------------------------------------------------+
```

This example confirms that two `MIG 1c.2g.10gb` compute instances were created on GPU instance 0.

```
$ nvidia-smi
Mon Aug 30 19:01:24 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63      Driver Version: 470.63     CUDA Version:  11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GRID A100X-2-10C     On  | 00000000:00:08.0 Off |                   On |
| N/A   N/A    P0    N/A /  N/A |   1058MiB / 10235MiB |     N/A      Default |
|                               |                      |              Enabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    0   0   0  |   1058MiB / 10235MiB | 14      0 |  2   0    1    0    0 |
|                  |      0MiB /  4096MiB |           |                       |
+------------------+                      +-----------+-----------------------+
|  0    0   1   1  |                      | 14      0 |  2   0    1    0    0 |
|                  |                      |           |                       |
+------------------+----------------------+-----------+-----------------------+

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

5.5. Enabling Unified Memory for a vGPU

Unified memory is disabled by default. If used, you must enable unified memory individually for each vGPU that requires it by setting a vGPU plugin parameter. How to enable unified memory for a vGPU depends on the hypervisor that you are using.

5.5.1. Enabling Unified Memory for a vGPU on VMware vSphere

On VMware vSphere, enable unified memory by setting the **`pciPassthru`*`vgpu-id`*`.cfg.enable_uvm`** configuration parameter in advanced VM attributes.

Ensure that the VM to which the vGPU is assigned is powered off.

Perform this task in the **vSphere Client** for each vGPU that requires unified memory.

In advanced VM attributes, set the **`pciPassthru`*`vgpu-id`*`.cfg.enable_uvm`** vGPU plugin parameter for the vGPU to 1 as explained in Setting vGPU Plugin Parameters on VMware vSphere.

vgpu-id

A positive integer that identifies the vGPU assigned to a VM. For the first vGPU assigned to a VM, *vgpu-id* is **0**. For example, if two vGPUs are assigned to a VM and you are enabling unified memory for both vGPUs, set **pciPassthru0.cfg.enable_uvm** and **pciPassthru1.cfg.enable_uvm** to 1.

5.6. Enabling NVIDIA CUDA Toolkit Development Tools for NVIDIA vGPU

By default, NVIDIA CUDA Toolkit development tools are disabled on NVIDIA vGPU. If used, you must enable NVIDIA CUDA Toolkit development tools individually for each VM that requires them by setting vGPU plugin parameters. One parameter must be set for enabling NVIDIA CUDA Toolkit debuggers and a different parameter must be set for enabling NVIDIA CUDA Toolkit profilers.

5.6.1. Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU

By default, NVIDIA CUDA Toolkit debuggers are disabled. If used, you must enable them for each vGPU VM that requires them by setting a vGPU plugin parameter. How to set the parameter to enable NVIDIA CUDA Toolkit debuggers for a vGPU VM depends on the hypervisor that you are using.

You can enable NVIDIA CUDA Toolkit debuggers for any number of VMs configured with vGPUs on the same GPU. When NVIDIA CUDA Toolkit debuggers are enabled for a VM, the VM cannot be migrated.

Perform this task for each VM for which you want to enable NVIDIA CUDA Toolkit debuggers.

Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU on on VMware vSphere

Ensure that the VM for which you want to enable NVIDIA CUDA Toolkit debuggers is powered off.

In advanced VM attributes, set the **pciPassthru*vgpu-id*.cfg.enable_debugging** vGPU plugin parameter for the vGPU that is assigned to the VM to 1 as explained in Setting vGPU Plugin Parameters on VMware vSphere.

vgpu-id

A positive integer that identifies the vGPU assigned to the VM. For the first vGPU assigned to a VM, *vgpu-id* is **0**. For example, if two vGPUs are assigned to a VM and you are enabling debuggers for both vGPUs, set **pciPassthru0.cfg.enable_debugging** and **pciPassthru1.cfg.enable_debugging** to 1.

The setting of this parameter is preserved after a guest VM is restarted. However, this parameter is reset to its default value after the hypervisor host is restarted.

5.6.2. Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU

By default, only GPU workload trace is enabled. If you want to use all NVIDIA CUDA Toolkit profiler features that NVIDIA vGPU supports, you must enable them for each vGPU VM that requires them.

Note: Enabling profiling for a VM gives the VM access to the GPU's global performance counters, which may include activity from other VMs executing on the same GPU. Enabling profiling for a VM also allows the VM to lock clocks on the GPU, which impacts all other VMs executing on the same GPU.

5.6.2.1. Supported NVIDIA CUDA Toolkit Profiler Features

You can enable the following NVIDIA CUDA Toolkit profiler features for a vGPU VM:

- NVIDIA Nsight™ Compute
- NVIDIA Nsight Systems
- CUDA Profiling Tools Interface (CUPTI)

5.6.2.2. Clock Management for a vGPU VM for Which NVIDIA CUDA Toolkit Profilers Are Enabled

Clocks are not locked for periodic sampling use cases such as NVIDIA Nsight Systems profiling.

Clocks are locked for multipass profiling such as:

- NVIDIA Nsight Compute kernel profiling
- CUPTI range profiling

Clocks are locked automatically when profiling starts and are unlocked automatically when profiling ends.

5.6.2.3. Limitations on the Use of NVIDIA CUDA Toolkit Profilers with NVIDIA vGPU

The following limitations apply when NVIDIA CUDA Toolkit profilers are enabled for NVIDIA vGPU:

- NVIDIA CUDA Toolkit profilers can be used on only one VM at a time.
- Multiple CUDA contexts cannot be profiled simultaneously.
- Profiling data is collected separately for each context.

- A VM for which NVIDIA CUDA Toolkit profilers are enabled cannot be migrated.

Because NVIDIA CUDA Toolkit profilers can be used on only one VM at a time, you should enable them for only one VM assigned a vGPU on a GPU. However, NVIDIA AI Enterprise cannot enforce this requirement. If NVIDIA CUDA Toolkit profilers are enabled on more than one VM assigned a vGPU on a GPU, profiling data is collected only for the first VM to start the profiler.

5.6.2.4. Enabling NVIDIA CUDA Toolkit Profilers for a vGPU VM

You enable NVIDIA CUDA Toolkit profilers for a vGPU VM by setting a vGPU plugin parameter. How to set the parameter to enable NVIDIA CUDA Toolkit profilers for a vGPU VM depends on the hypervisor that you are using.

Perform this task for the VM for which you want to enable NVIDIA CUDA Toolkit profilers.

Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU on on VMware vSphere

Ensure that the VM for which you want to enable NVIDIA CUDA Toolkit profilers is powered off.

In advanced VM attributes, set the **`pciPassthru`*`vgpu-id`*`.cfg.enable_profiling`** vGPU plugin parameter for the vGPU that is assigned to the VM to 1 as explained in Setting vGPU Plugin Parameters on VMware vSphere.

vgpu-id

A positive integer that identifies the vGPU assigned to the VM. For the first vGPU assigned to a VM, *vgpu-id* is **`0`**. For example, if two vGPUs are assigned to a VM and you are enabling profilers for the second vGPU, set **`pciPassthru1.cfg.enable_profiling`** to 1.

The setting of this parameter is preserved after a guest VM is restarted. However, this parameter is reset to its default value after the hypervisor host is restarted.

Chapter 6. Monitoring GPU Performance

NVIDIA AI Enterprise enables you to monitor the performance of physical GPUs and virtual GPUs from the hypervisor and from within individual guest VMs.

You can use several tools for monitoring GPU performance:

- From any supported hypervisor, and from a guest VM that is running a 64-bit edition of Windows or Linux, you can use NVIDIA System Management Interface, `nvidia-smi`.
- From Citrix Hypervisor, you can use Citrix XenCenter.
- From a Windows guest VM, you can use these tools:
  - Windows Performance Monitor
  - Windows Management Instrumentation (WMI)

6.1. NVIDIA System Management Interface `nvidia-smi`

NVIDIA System Management Interface, `nvidia-smi`, is a command-line tool that reports management information for NVIDIA GPUs.

The `nvidia-smi` tool is included in the following packages:

- NVIDIA Virtual GPU Manager package for each supported hypervisor
- NVIDIA driver package for each supported guest OS

The scope of the reported management information depends on where you run `nvidia-smi` from:

- From a hypervisor command shell, such as the Citrix Hypervisor dom0 shell or VMware ESXi host shell, `nvidia-smi` reports management information for NVIDIA physical GPUs and virtual GPUs present in the system.

> **Note:** When run from a hypervisor command shell, `nvidia-smi` will not list any GPU that is currently allocated for GPU pass-through.

- From a guest VM, `nvidia-smi` retrieves usage statistics for vGPUs or pass-through GPUs that are assigned to the VM.

  From a Windows guest VM, you can run `nvidia-smi` from a command prompt by changing to the `C:\Program Files\NVIDIA Corporation\NVSMI` folder and running the `nvidia-smi.exe` command.

6.2. Using `nvidia-smi` to Monitor GPU Performance from a Hypervisor

You can get management information for the NVIDIA physical GPUs and virtual GPUs present in the system by running `nvidia-smi` from a hypervisor command shell such as the Citrix Hypervisor dom0 shell or the VMware ESXi host shell.

Without a subcommand, `nvidia-smi` provides management information for **physical** GPUs. To examine **virtual** GPUs in more detail, use `nvidia-smi` with the `vgpu` subcommand.

From the command line, you can get help information about the `nvidia-smi` tool and the `vgpu` subcommand.

Help Information	Command
A list of subcommands supported by the `nvidia-smi` tool. Note that not all subcommands apply to GPUs that support NVIDIA AI Enterprise.	**`nvidia-smi -h`**
A list of all options supported by the `vgpu` subcommand.	**`nvidia-smi vgpu -h`**

6.2.1. Getting a Summary of all Physical GPUs in the System

To get a summary of all physical GPUs in the system, along with PCI bus IDs, power state, temperature, current memory usage, and so on, run `nvidia-smi` without additional arguments.

Each vGPU instance is reported in the `Compute processes` section, together with its physical GPU index and the amount of frame-buffer memory assigned to it.

In the example that follows, three vGPUs are running in the system: One vGPU is running on each of the physical GPUs 0, 1, and 2.

```
[root@vgpu ~]# nvidia-smi
Fri Aug 27 09:26:18 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63                 Driver Version: 470.63                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla M60           On   | 0000:83:00.0     Off |                    Off |
| N/A   31C    P8    23W / 150W |   1889MiB /  8191MiB |      7%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           On   | 0000:84:00.0     Off |                    Off |
| N/A   26C    P8    23W / 150W |    926MiB /  8191MiB |      9%      Default |
```

```
+-------------------------------+----------------------+----------------------+
|   2  Tesla M10           On   | 0000:8A:00.0     Off |                  N/A |
| N/A   23C    P8    10W /  53W |   1882MiB /  8191MiB |     12%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla M10           On   | 0000:8B:00.0     Off |                  N/A |
| N/A   26C    P8    10W /  53W |     10MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Tesla M10           On   | 0000:8C:00.0     Off |                  N/A |
| N/A   34C    P8    10W /  53W |     10MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Tesla M10           On   | 0000:8D:00.0     Off |                  N/A |
| N/A   32C    P8    10W /  53W |     10MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU       PID  Type  Process name                                  GPU Memory |
|=============================================================================|
|    0     11924  C+G   /usr/lib64/xen/bin/vgpu                        1856MiB |
|    1     11903  C+G   /usr/lib64/xen/bin/vgpu                         896MiB |
|    2     11908  C+G   /usr/lib64/xen/bin/vgpu                        1856MiB |
+-----------------------------------------------------------------------------+
[root@vgpu ~]#
```

6.2.2. Getting a Summary of all vGPUs in the System

To get a summary of the vGPUs currently that are currently running on each physical GPU in the system, run `nvidia-smi vgpu` without additional arguments.

```
[root@vgpu ~]# nvidia-smi vgpu
Fri Aug 27 09:27:06 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63                 Driver Version: 470.63                    |
|-------------------------------+-------------------------------+------------+
| GPU  Name                     | Bus-Id                        | GPU-Util   |
|      vGPU ID    Name          | VM ID    VM Name              | vGPU-Util  |
|===============================+===============================+============|
|   0  Tesla M60                | 0000:83:00.0                  |   7%       |
|      11924      GRID M60-2Q   | 3        Win7-64 GRID test 2  |       6%   |
+-------------------------------+-------------------------------+------------+
|   1  Tesla M60                | 0000:84:00.0                  |   9%       |
|      11903      GRID M60-1B   | 1        Win8.1-64 GRID test 3 |       8%   |
+-------------------------------+-------------------------------+------------+
|   2  Tesla M10                | 0000:8A:00.0                  |  12%       |
|      11908      GRID M10-2Q   | 2        Win7-64 GRID test 1  |      10%   |
+-------------------------------+-------------------------------+------------+
|   3  Tesla M10                | 0000:8B:00.0                  |   0%       |
+-------------------------------+-------------------------------+------------+
|   4  Tesla M10                | 0000:8C:00.0                  |   0%       |
+-------------------------------+-------------------------------+------------+
|   5  Tesla M10                | 0000:8D:00.0                  |   0%       |
+-------------------------------+-------------------------------+------------+
[root@vgpu ~]#
```

6.2.3. Getting vGPU Details

To get detailed information about all the vGPUs on the platform, run `nvidia-smi vgpu` with the `-q` or `--query` option.

To limit the information retrieved to a subset of the GPUs on the platform, use the `-i` or `--id` option to select one or more vGPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -q -i 1
GPU 00000000:86:00.0
    Active vGPUs                      : 1
    vGPU ID                           : 3251634178
        VM ID                         : 1
        VM Name                       : Win7
        vGPU Name                     : GRID M60-8Q
        vGPU Type                     : 22
        vGPU UUID                     : b8c6d0e1-d167-11e8-b8c9-55705e5a54a6
        Guest Driver Version          : 411.81
        License Status                : Unlicensed
        Accounting Mode               : Disabled
        Accounting Buffer Size: 4000
        Frame Rate Limit              : 3 FPS
        FB Memory Usage               :
            Total                     : 8192 MiB
            Used                      : 675 MiB
            Free                      : 7517 MiB
        Utilization                   :
            Gpu                       : 3 %
            Memory                    : 0 %
            Encoder                   : 0 %
            Decoder                   : 0 %
        Encoder Stats                 :
            Active Sessions           : 0
            Average FPS               : 0
            Average Latency           : 0
        FBC Stats                     :
            Active Sessions           : 1
            Average FPS               : 227
            Average Latency           : 4403
[root@vgpu ~]#
```

6.2.4. Monitoring vGPU engine usage

To monitor vGPU engine usage across multiple vGPUs, run `nvidia-smi vgpu` with the `-u` or `--utilization` option.

For each vGPU, the usage statistics in the following table are reported once every second. The table also shows the name of the column in the command output under which each statistic is reported.

Statistic	Column
3D/Compute	`sm`
Memory controller bandwidth	`mem`
Video encoder	`enc`
Video decoder	`dec`

Each reported percentage is the percentage of the physical GPU's capacity that a vGPU is using. For example, a vGPU that uses 20% of the GPU's graphics engine's capacity will report 20%.

To modify the reporting frequency, use the `-l` or `--loop` option.

To limit monitoring to a subset of the GPUs on the platform, use the `-i` or `--id` option to select one or more vGPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -u
# gpu     vgpu     sm   mem   enc   dec
# Idx       Id      %     %     %     %
    0    11924      6     3     0     0
    1    11903      8     3     0     0
    2    11908     10     4     0     0
    3        -      -     -     -     -
    4        -      -     -     -     -
    5        -      -     -     -     -
    0    11924      6     3     0     0
    1    11903      9     3     0     0
    2    11908     10     4     0     0
    3        -      -     -     -     -
    4        -      -     -     -     -
    5        -      -     -     -     -
    0    11924      6     3     0     0
    1    11903      8     3     0     0
    2    11908     10     4     0     0
    3        -      -     -     -     -
    4        -      -     -     -     -
    5        -      -     -     -     -
^C[root@vgpu ~]#
```

6.2.5. Monitoring vGPU engine usage by applications

To monitor vGPU engine usage by applications across multiple vGPUs, run `nvidia-smi vgpu` with the `-p` option.

For each application on each vGPU, the usage statistics in the following table are reported once every second. Each application is identified by its process ID and process name. The table also shows the name of the column in the command output under which each statistic is reported.

Statistic	Column
3D/Compute	`sm`
Memory controller bandwidth	`mem`
Video encoder	`enc`
Video decoder	`dec`

Each reported percentage is the percentage of the physical GPU's capacity used by an application running on a vGPU that resides on the physical GPU. For example, an application that uses 20% of the GPU's graphics engine's capacity will report 20%.

To modify the reporting frequency, use the `-l` or `--loop` option.

To limit monitoring to a subset of the GPUs on the platform, use the `-i` or `--id` option to select one or more vGPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -p
# GPU     vGPU process         process    sm   mem   enc   dec
# Idx       Id      Id            name     %     %     %     %
    0    38127    1528         dwm.exe     0     0     0     0
    1    37408    4232   DolphinVS.exe    32    25     0     0
    1   257869    4432     FurMark.exe    16    12     0     0
```

```
    1  257969    4552    FurMark.exe   48   37    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1   37408    4232  DolphinVS.exe   16   12    0    0
    1  257911     656  DolphinVS.exe   32   24    0    0
    1  257969    4552    FurMark.exe   48   37    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1  257869    4432    FurMark.exe   38   30    0    0
    1  257911     656  DolphinVS.exe   19   14    0    0
    1  257969    4552    FurMark.exe   38   30    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1  257848    3220    Balls64.exe   16   12    0    0
    1  257869    4432    FurMark.exe   16   12    0    0
    1  257911     656  DolphinVS.exe   16   12    0    0
    1  257969    4552    FurMark.exe   48   37    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1  257911     656  DolphinVS.exe   32   25    0    0
    1  257969    4552    FurMark.exe   64   50    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1   37408    4232  DolphinVS.exe   16   12    0    0
    1  257911     656  DolphinVS.exe   16   12    0    0
    1  257969    4552    FurMark.exe   64   49    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1   37408    4232  DolphinVS.exe   16   12    0    0
    1  257869    4432    FurMark.exe   16   12    0    0
    1  257969    4552    FurMark.exe   64   49    0    0
[root@vgpu ~]#
```

6.2.6. Monitoring Encoder Sessions

Note: Encoder sessions can be monitored **only** for vGPUs assigned to Windows VMs. No encoder session statistics are reported for vGPUs assigned to Linux VMs.

To monitor the encoder sessions for processes running on multiple vGPUs, run `nvidia-smi vgpu` with the `-es` or `--encodersessions` option.

For each encoder session, the following statistics are reported once every second:

- GPU ID
- vGPU ID
- Encoder session ID
- PID of the process in the VM that created the encoder session
- Codec type, for example, H.264 or H.265
- Encode horizontal resolution
- Encode vertical resolution
- One-second trailing average encoded FPS
- One-second trailing average encode latency in microseconds

To modify the reporting frequency, use the `-l` or `--loop` option.

To limit monitoring to a subset of the GPUs on the platform, use the `-i` or `--id` option to select one or more vGPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -es
# GPU    vGPU Session Process   Codec       H       V Average     Average
# Idx      Id      Id      Id    Type     Res     Res     FPS Latency(us)
```

```
    1   21211       2    2308   H.264    1920    1080     424        1977
    1   21206       3    2424   H.264    1920    1080       0           0
    1   22011       1    3676   H.264    1920    1080     374        1589
    1   21211       2    2308   H.264    1920    1080     360         807
    1   21206       3    2424   H.264    1920    1080     325        1474
    1   22011       1    3676   H.264    1920    1080     313        1005
    1   21211       2    2308   H.264    1920    1080     329        1732
    1   21206       3    2424   H.264    1920    1080     352        1415
    1   22011       1    3676   H.264    1920    1080     434        1894
    1   21211       2    2308   H.264    1920    1080     362        1818
    1   21206       3    2424   H.264    1920    1080     296        1072
    1   22011       1    3676   H.264    1920    1080     416        1994
    1   21211       2    2308   H.264    1920    1080     444        1912
    1   21206       3    2424   H.264    1920    1080     330        1261
    1   22011       1    3676   H.264    1920    1080     436        1644
    1   21211       2    2308   H.264    1920    1080     344        1500
    1   21206       3    2424   H.264    1920    1080     393        1727
    1   22011       1    3676   H.264    1920    1080     364        1945
    1   21211       2    2308   H.264    1920    1080     555        1653
    1   21206       3    2424   H.264    1920    1080     295         925
    1   22011       1    3676   H.264    1920    1080     372        1869
    1   21211       2    2308   H.264    1920    1080     326        2206
    1   21206       3    2424   H.264    1920    1080     318        1366
    1   22011       1    3676   H.264    1920    1080     464        2015
    1   21211       2    2308   H.264    1920    1080     305        1167
    1   21206       3    2424   H.264    1920    1080     445        1892
    1   22011       1    3676   H.264    1920    1080     361         906
    1   21211       2    2308   H.264    1920    1080     353        1436
    1   21206       3    2424   H.264    1920    1080     354        1798
    1   22011       1    3676   H.264    1920    1080     373        1310
^C[root@vgpu ~]#
```

6.2.7. Listing Supported vGPU Types

To list the virtual GPU types that the GPUs in the system support, run `nvidia-smi vgpu` with the `-s` or `--supported` option.

To limit the retrieved information to a subset of the GPUs on the platform, use the `-i` or `--id` option to select one or more vGPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -s -i 0
GPU 0000:83:00.0
    GRID M60-0B
    GRID M60-0Q
    GRID M60-1A
    GRID M60-1B
    GRID M60-1Q
    GRID M60-2A
    GRID M60-2Q
    GRID M60-4A
    GRID M60-4Q
    GRID M60-8A
    GRID M60-8Q
[root@vgpu ~]#
```

To view detailed information about the supported vGPU types, add the `-v` or `--verbose` option:

```
[root@vgpu ~]# nvidia-smi vgpu -s -i 0 -v   | less
GPU 00000000:83:00.0
    vGPU Type ID                : 0xb
        Name                    : GRID M60-0B
        Class                   : NVS
        Max Instances           : 16
        Device ID               : 0x13f210de
        Sub System ID           : 0x13f21176
        FB Memory               : 512 MiB
```

```
        Display Heads          : 2
        Maximum X Resolution   : 2560
        Maximum Y Resolution   : 1600
        Frame Rate Limit       : 45 FPS
        GRID License           : GRID-Virtual-PC,2.0;GRID-Virtual-WS,2.0;GRID-
Virtual-WS-Ext,2.0;Quadro-Virtual-DWS,5.0
    vGPU Type ID               : 0xc
        Name                   : GRID M60-0Q
        Class                  : Quadro
        Max Instances          : 16
        Device ID              : 0x13f210de
        Sub System ID          : 0x13f2114c
        FB Memory              : 512 MiB
        Display Heads          : 2
        Maximum X Resolution   : 2560
        Maximum Y Resolution   : 1600
        Frame Rate Limit       : 60 FPS
        GRID License           : GRID-Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0;Quadro-
Virtual-DWS,5.0
    vGPU Type ID               : 0xd
        Name                   : GRID M60-1A
        Class                  : NVS
        Max Instances          : 8
…
[root@vgpu ~]#
```

6.2.8. Listing the vGPU Types that Can Currently Be Created

To list the virtual GPU types that can currently be created on GPUs in the system, run `nvidia-smi vgpu` with the `-c` or `--creatable` option.

This property is a dynamic property that varies for each GPU depending on whether MIG mode is enabled for the GPU.

- If MIG mode is **not** enabled for the GPU, or if the GPU does not support MIG, this property reflects the number and type of vGPUs that are already running on the GPU.
  - If no vGPUs are running on the GPU, all vGPU types that the GPU supports are listed.
  - If one or more vGPUs are running on the GPU, but the GPU is not fully loaded, only the type of the vGPUs that are already running is listed.
  - If the GPU is fully loaded, no vGPU types are listed.
- If MIG mode is enabled for the GPU, the result reflects the number and type of GPU instances on which no vGPUs are already running.
  - If no GPU instances have been created, no vGPU types are listed.
  - If GPU instances have been created, only the vGPU types that correspond to GPU instances on which no vGPU is running are listed.
  - If a vGPU is running on every GPU instance, no vGPU types are listed.

To limit the retrieved information to a subset of the GPUs on the platform, use the `-i` or `--id` option to select one or more vGPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -c -i 0
GPU 0000:83:00.0
```

```
    GRID M60-2Q
[root@vgpu ~]#
```

To view detailed information about the vGPU types that can currently be created, add the `-v` or `--verbose` option.

6.3. Monitoring GPU Performance from a Guest VM

You can use monitoring tools within an individual guest VM to monitor the performance of vGPUs or pass-through GPUs that are assigned to the VM. The scope of these tools is limited to the guest VM within which you use them. You cannot use monitoring tools within an individual guest VM to monitor any other GPUs in the platform.

For a vGPU, only these metrics are reported in a guest VM:

- 3D/Compute
- Memory controller
- Video encoder
- Video decoder
- Frame buffer usage

Other metrics normally present in a GPU are not applicable to a vGPU and are reported as zero or `N/A`, depending on the tool that you are using.

6.3.1. Using `nvidia-smi` to Monitor GPU Performance from a Guest VM

In guest VMs, you can use the `nvidia-smi` command to retrieve statistics for the total usage by all applications running in the VM and usage by individual applications of the following resources:

- GPU
- Video encoder
- Video decoder
- Frame buffer

To use `nvidia-smi` to retrieve statistics for the total resource usage by all applications running in the VM, run the following command:

```
nvidia-smi dmon
```

The following example shows the result of running `nvidia-smi dmon` from within a Windows guest VM.

Figure 14. Using `nvidia-smi` from a Windows guest VM to get total resource usage by all applications

```
Command Prompt
C:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi dmon
# gpu   pwr  temp    sm   mem   enc   dec  mclk  pclk
# Idx     W     C     %     %     %     %   MHz   MHz
    0     -     -     9     0     0     0  3704  1531
    0     -     -     9     0     0     0  3704  1531
    0     -     -     9     0     0     0  3704  1531
    0     -     -     9     0     0     0  3704  1531
    0     -     -     9     0     0     0  3704  1531
    0     -     -     9     0     0     0  3704  1531
    0     -     -     9     0     0     0  3704  1531
    0     -     -     8     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     7     0     0     0  3704  1531
    0     -     -     6     0     0     0  3704  1531
    0     -     -     6     0     0     0  3704  1531

C:\Program Files\NVIDIA Corporation\NVSMI>
```

To use `nvidia-smi` to retrieve statistics for resource usage by individual applications running in the VM, run the following command:

```
nvidia-smi pmon
```

Figure 15. Using `nvidia-smi` from a Windows guest VM to get resource usage by individual applications

```
C:\Windows\system32\cmd.exe
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi  pmon
# gpu      pid  type    sm   mem   enc   dec   command
# Idx        #   C/G     %     %     %     %   name
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     1     1     0     0   Balls64.exe
    0     4472   C+G    23    20     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    22    19     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    23    20     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    22    19     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    19    16     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    19    16     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    20    17     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    20    17     0     0   FurMark.exe
    0     4868   C+G     0     0     0     0   Balls64.exe
    0      656   C+G     0     0     0     0   DolphinVS.exe
    0     2520   C+G     0     0     0     0   chrome.exe
    0     4216   C+G     0     0     0     0   Balls64.exe
    0     4472   C+G    20    17     0     0   FurMark.exe
```

Chapter 7. Changing Scheduling Behavior for Time-Sliced vGPUs

NVIDIA GPUs based on the NVIDIA Maxwell™ graphic architecture implement a best effort vGPU scheduler that aims to balance performance across vGPUs. The best effort scheduler allows a vGPU to use GPU processing cycles that are not being used by other vGPUs. Under some circumstances, a VM running a graphics-intensive application may adversely affect the performance of graphics-light applications running in other VMs.

GPUs based on NVIDIA GPU architectures **after** the Maxwell architecture additionally support equal share and fixed share vGPU schedulers. To address this issue with the best effort vGPU scheduler, NVIDIA GPUs additionally support equal share and fixed share vGPU schedulers. These schedulers impose a limit on GPU processing cycles used by a vGPU, which prevents graphics-intensive applications running in one VM from affecting the performance of graphics-light applications running in other VMs. On GPUs that support multiple vGPU schedulers, you can select the vGPU scheduler to use. You can also set the length of the time slice for the equal share and fixed share vGPU schedulers.

Note: If you use the equal share or fixed share vGPU scheduler, the frame-rate limiter (FRL) is disabled.

The best effort scheduler is the default scheduler for all supported GPU architectures.

If you are unsure of the NVIDIA GPU architecture of your GPU, consult the release notes for your hypervisor at NVIDIA AI Enterprise Documentation.

7.1. Scheduling Policies for Time-Sliced vGPUs

In addition to the default best effort scheduler, GPUs based on NVIDIA GPU architectures **after** the Maxwell architecture support equal share and fixed share vGPU schedulers.

Equal share scheduler

The physical GPU is shared equally amongst the running vGPUs that reside on it. As vGPUs are added to or removed from a GPU, the share of the GPU's processing cycles allocated to each vGPU changes accordingly. As a result, the performance of a vGPU may increase as

other vGPUs on the same GPU are stopped, or decrease as other vGPUs are started on the same GPU.

Fixed share scheduler

Each vGPU is given a fixed share of the physical GPU's processing cycles, the amount of which depends on the vGPU type, which in turn determines the maximum number of vGPUs per physical GPU. For example, the maximum number of T4-4C vGPUs per physical GPU is 4. When the scheduling policy is fixed share, each T4-4C vGPU is given one quarter, or 25%, the physical GPU's processing cycles. As vGPUs are added to or removed from a GPU, the share of the GPU's processing cycles allocated to each vGPU remains constant. As a result, the performance of a vGPU remains unchanged as other vGPUs are stopped or started on the same GPU.

7.2. Scheduler Time Slice for Time-Sliced vGPUs

When multiple VMs access the vGPUs on a single GPU, the GPU performs the work for each VM **serially**. The vGPU scheduler time slice represents the amount of time that the work of a VM is allowed to run on the GPU before it is preempted and the work of the next VM is performed.

For the equal share and fixed share vGPU schedulers, you can set the length of the time slice. The length of the time slice affects latency and throughput. The optimal length of the time slice depends the workload that the GPU is handling.

- For workloads that require low latency, a shorter time slice is optimal. Typically, these workloads are applications that must generate output at a fixed interval, such as graphics applications that generate output at a frame rate of 60 FPS. These workloads are sensitive to latency and should be allowed to run at least once per interval. A shorter time slice reduces latency and improves responsiveness by causing the scheduler to switch more frequently between VMs.
- For workloads that require maximum throughput, a longer time slice is optimal. Typically, these workloads are applications that must complete their work as quickly as possible and do not require responsiveness, such as CUDA applications. A longer time slice increases throughput by preventing frequent switching between VMs.

7.3. `RmPVMRL` Registry Key

The `RmPVMRL` registry key controls the scheduling behavior for NVIDIA vGPUs by setting the scheduling policy and the length of the time slice.

Note: You can change the vGPU scheduling behavior only on GPUs that support multiple vGPU schedulers, that is, GPUs based on NVIDIA GPU architectures **after** the Maxwell architecture.

Type

`Dword`

Contents

Value	Meaning
`0x00` (default)	Best effort scheduler
`0x01`	Equal share scheduler with the default time slice length
`0x00`*`TT`*`0001`	Equal share scheduler with a user-defined time slice length *TT*
`0x11`	Fixed share scheduler with the default time slice length
`0x00`*`TT`*`0011`	Fixed share scheduler with a user-defined time slice length *TT*

The default time slice length depends on the maximum number of vGPUs per physical GPU allowed for the vGPU type.

Maximum Number of vGPUs	Default Time Slice Length
Less than or equal to 8	2 ms
Greater than 8	1 ms

TT

Two hexadecimal digits in the range 01 to 1E that set the length of the time slice in milliseconds (ms) for the equal share and fixed share schedulers. The minimum length is 1 ms and the maximum length is 30 ms.

If *TT* is 00, the length is set to the default length for the vGPU type.

If *TT* is greater than 1E, the length is set to 30 ms.

Examples

This example sets the vGPU scheduler to equal share scheduler with the default time slice length.

```
RmPVMRL=0x01
```

This example sets the vGPU scheduler to equal share scheduler with a time slice that is 3 ms long.

```
RmPVMRL=0x00030001
```

This example sets the vGPU scheduler to fixed share scheduler with the default time slice length.

```
RmPVMRL=0x11
```

This example sets the vGPU scheduler to fixed share scheduler with a time slice that is 24 (0x18) ms long.

```
RmPVMRL=0x00180011
```

7.4. Getting the Current Time-Sliced vGPU Scheduling Behavior for All GPUs

Get the current scheduling behavior before changing the scheduling behavior of one or more GPUs to determine if you need to change it or after changing it to confirm the change.

Perform this task in your hypervisor command shell.

1. Open a command shell on your hypervisor host machine.

   On all supported hypervisors, you can use secure shell (SSH) for this purpose. Individual hypervisors may provide additional means for logging in. For details, refer to the documentation for your hypervisor.

   You can use secure shell (SSH) for this purpose.
2. Open a command shell on your hypervisor host machine.
3. Use the `dmesg` command to display messages from the kernel that contain the strings `NVRM` and `scheduler`.

   ```
   $ dmesg | grep NVRM | grep scheduler
   ```

   The scheduling behavior is indicated in these messages by the following strings:

   - `BEST_EFFORT`
   - `EQUAL_SHARE`
   - `FIXED_SHARE`

   If the scheduling behavior is equal share or fixed share, the scheduler time slice in ms is also displayed.

   This example gets the scheduling behavior of the GPUs in a system in which the behavior of one GPU is set to best effort, one GPU is set to equal share, and one GPU is set to fixed share.

   ```
   $ dmesg | grep NVRM | grep scheduler
   2020-10-05T02:58:08.928Z cpu79:2100753)NVRM: GPU at 0000:3d:00.0 has software
    scheduler DISABLED with policy BEST_EFFORT.
   2020-10-05T02:58:09.818Z cpu79:2100753)NVRM: GPU at 0000:5e:00.0 has software
    scheduler ENABLED with policy EQUAL_SHARE.
   NVRM: Software scheduler timeslice set to 1 ms.
   2020-10-05T02:58:12.115Z cpu79:2100753)NVRM: GPU at 0000:88:00.0 has software
    scheduler ENABLED with policy FIXED_SHARE.
   NVRM: Software scheduler timeslice set to 1 ms.
   ```

7.5. Changing the Time-Sliced vGPU Scheduling Behavior for All GPUs

Note: You can change the vGPU scheduling behavior only on GPUs that support multiple vGPU schedulers, that is, GPUs based on NVIDIA GPU architectures **after** the Maxwell architecture.

Perform this task in your hypervisor command shell.

1. Open a command shell on your hypervisor host machine.

   On all supported hypervisors, you can use secure shell (SSH) for this purpose. Individual hypervisors may provide additional means for logging in. For details, refer to the documentation for your hypervisor.

   You can use secure shell (SSH) for this purpose.
2. Open a command shell on your hypervisor host machine.
3. Set the `RmPVMRL` registry key to the value that sets the GPU scheduling policy and the length of the time slice that you want.

   Add the following entry to the `/etc/modprobe.d/nvidia.conf` file:

   ```
   options nvidia NVreg_RegistryDwords="RmPVMRL=value"
   ```

   If the `/etc/modprobe.d/nvidia.conf` file does not already exist, create it.

   - On Citrix Hypervisor, Red Hat Enterprise Linux KVM, or Red Hat Virtualization (RHV), add the following entry to the `/etc/modprobe.d/nvidia.conf` file.

     ```
     options nvidia NVreg_RegistryDwords="RmPVMRL=value"
     ```

     If the `/etc/modprobe.d/nvidia.conf` file does not already exist, create it.
   - On VMware vSphere, use the `esxcli set` command.

     ```
     # esxcli system module parameters set -m nvidia -p
      "NVreg_RegistryDwords=RmPVMRL=value"
     ```

   value

   The value that sets the GPU scheduling policy and the length of the time slice that you want, for example:

   0x01
   Sets the vGPU scheduling policy to equal share scheduler with the default time slice length.

   0x00030001
   Sets the GPU scheduling policy to equal share scheduler with a time slice that is 3 ms long.

   0x11
   Sets the vGPU scheduling policy to fixed share scheduler with the default time slice length.

   0x00180011
   Sets the GPU scheduling policy to fixed share scheduler with a time slice that is 24 (0x18) ms long.

For all supported values, see RmPVMRL Registry Key.

4. Reboot your hypervisor host machine.

Confirm that the scheduling behavior was changed as required as explained in Getting the Current Time-Sliced vGPU Scheduling Behavior for All GPUs.

7.6. Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs

Note: You can change the vGPU scheduling behavior only on GPUs that support multiple vGPU schedulers, that is, GPUs based on NVIDIA GPU architectures **after** the Maxwell architecture.

Perform this task in your hypervisor command shell.

1. Open a command shell on your hypervisor host machine.

   On all supported hypervisors, you can use secure shell (SSH) for this purpose. Individual hypervisors may provide additional means for logging in. For details, refer to the documentation for your hypervisor.

   You can use secure shell (SSH) for this purpose.
2. Open a command shell on your hypervisor host machine.
3. Use the `lspci` command to obtain the PCI domain and bus/device/function (BDF) of each GPU for which you want to change the scheduling behavior.

   ```
   # lspci -D -d 10de:
   ```

   - On Citrix Hypervisor, Red Hat Enterprise Linux KVM, or Red Hat Virtualization (RHV), add the `-D` option to display the PCI domain and the `-d 10de:` option to display information only for NVIDIA GPUs.

     ```
     # lspci -D -d 10de:
     ```
   - On VMware vSphere, pipe the output of `lspci` to the `grep` command to display information only for NVIDIA GPUs.

     ```
     # lspci | grep NVIDIA
     ```

   The NVIDIA GPU listed in this example has the PCI domain `0000` and BDF `86:00.0`.

   ```
   0000:86:00.0 3D controller: NVIDIA Corporation GP104GL [Tesla P4] (rev a1)
   ```
4. Use the module parameter `NVreg_RegistryDwordsPerDevice` to set the `pci` and `RmPVMRL` registry keys for each GPU.

   Add the following entry to the `/etc/modprobe.d/nvidia.conf` file

   ```
   options nvidia NVreg_RegistryDwordsPerDevice="pci=pci-domain:pci-
   bdf;RmPVMRL=value
   [;pci=pci-domain:pci-bdf;RmPVMRL=value...]"
   ```

   If the `/etc/modprobe.d/nvidia.conf` file does not already exist, create it.

- On Citrix Hypervisor, Red Hat Enterprise Linux KVM, or RHV, add the following entry to the `/etc/modprobe.d/nvidia.conf` file.

  ```
  options nvidia NVreg_RegistryDwordsPerDevice="pci=pci-domain:pci-bdf;RmPVMRL=value
  [;pci=pci-domain:pci-bdf;RmPVMRL=value...]"
  ```

  If the `/etc/modprobe.d/nvidia.conf` file does not already exist, create it.
- On VMware vSphere, use the `esxcli set` command.

  ```
  # esxcli system module parameters set -m nvidia \
  -p "NVreg_RegistryDwordsPerDevice=pci=pci-domain:pci-bdf;RmPVMRL=value\
  [;pci=pci-domain:pci-bdf;RmPVMRL=value...]"
  ```

For each GPU, provide the following information:

pci-domain
: The PCI domain of the GPU.

pci-bdf
: The PCI device BDF of the GPU.

value

: The value that sets the GPU scheduling policy and the length of the time slice that you want, for example:

  0x01
  : Sets the GPU scheduling policy to equal share scheduler with the default time slice length.

  0x00030001
  : Sets the GPU scheduling policy to equal share scheduler with a time slice that is 3 ms long.

  0x11
  : Sets the GPU scheduling policy to fixed share scheduler with the default time slice length.

  0x00180011
  : Sets the GPU scheduling policy to fixed share scheduler with a time slice that is 24 (0x18) ms long.

  For all supported values, see RmPVMRL Registry Key.

This example adds an entry to the `/etc/modprobe.d/nvidia.conf` file to change the scheduling behavior of a single GPU. The entry sets the GPU scheduling policy of the GPU at PCI domain `0000` and BDF `86:00.0` to fixed share scheduler with the default time slice length.

```
options nvidia NVreg_RegistryDwordsPerDevice=
"pci=0000:86:00.0;RmPVMRL=0x11"
```

This example adds an entry to the `/etc/modprobe.d/nvidia.conf` file to change the scheduling behavior of a single GPU. The entry sets the scheduling policy of the GPU at PCI domain `0000` and BDF `86:00.0` to fixed share scheduler with a time slice that is 24 (0x18) ms long.

```
options nvidia NVreg_RegistryDwordsPerDevice=
"pci=0000:86:00.0;RmPVMRL=0x00180011"
```

This example changes the scheduling behavior of a single GPU on a hypervisor host that is running VMware vSphere. The command sets the scheduling policy of the GPU at PCI domain `0000` and BDF `15:00.0` to fixed share scheduler with the default time slice length.

```
# esxcli system module parameters set -m nvidia -p \
"NVreg_RegistryDwordsPerDevice=pci=0000:15:00.0;RmPVMRL=0x11[;pci=0000:15:00.0;RmPVMRL=0x11]"
```

This example changes the scheduling behavior of a single GPU on a hypervisor host that is running VMware vSphere. The command sets the scheduling policy of the GPU at PCI domain `0000` and BDF `15:00.0` to fixed share scheduler with a time slice that is 24 (0x18) ms long.

```
# esxcli system module parameters set -m nvidia -p \
"NVreg_RegistryDwordsPerDevice=pci=0000:15:00.0;RmPVMRL=0x11[;pci=0000:15:00.0;RmPVMRL=0x00180011]"
```

5. Reboot your hypervisor host machine.

Confirm that the scheduling behavior was changed as required as explained in Getting the Current Time-Sliced vGPU Scheduling Behavior for All GPUs.

7.7. Restoring Default Time-Sliced vGPU Scheduler Settings

Perform this task in your hypervisor command shell.

1. Open a command shell on your hypervisor host machine.

   On all supported hypervisors, you can use secure shell (SSH) for this purpose. Individual hypervisors may provide additional means for logging in. For details, refer to the documentation for your hypervisor.

   You can use secure shell (SSH) for this purpose.
2. Open a command shell on your hypervisor host machine.
3. Unset the `RmPVMRL` registry key.

   Comment out the entries in the `/etc/modprobe.d/nvidia.conf` file that set `RmPVMRL` by prefixing each entry with the `#` character.

   - On Citrix Hypervisor, Red Hat Enterprise Linux KVM, or Red Hat Virtualization (RHV), comment out the entries in the `/etc/modprobe.d/nvidia.conf` file that set `RmPVMRL` by prefixing each entry with the `#` character.
   - On VMware vSphere, set the module parameter to an empty string.

     ```
     # esxcli system module parameters set -m nvidia -p "module-parameter="
     ```

     module-parameter

     The module parameter to set, which depends on whether the scheduling behavior was changed for all GPUs or select GPUs:

     - For all GPUs, set the `NVreg_RegistryDwords` module parameter.
     - For select GPUs, set the `NVreg_RegistryDwordsPerDevice` module parameter.

     For example, to restore default vGPU scheduler settings after they were changed for all GPUs, enter this command:

     ```
     # esxcli system module parameters set -m nvidia -p "NVreg_RegistryDwords="
     ```
4. Reboot your hypervisor host machine.

Chapter 8. Troubleshooting

This chapter describes basic troubleshooting steps for NVIDIA vGPU on Citrix Hypervisor, Red Hat Enterprise Linux KVM, Red Hat Virtualization (RHV), and VMware vSphere, and how to collect debug information when filing a bug report.

8.1. Known issues

Before troubleshooting or filing a bug report, review the release notes that accompany each driver release, for information about known issues with the current release, and potential workarounds.

8.2. Troubleshooting steps

If a vGPU-enabled VM fails to start, or doesn't display any output when it does start, follow these steps to narrow down the probable cause.

8.2.1. Verifying the NVIDIA Kernel Driver Is Loaded

1. Use the command that your hypervisor provides to verify that the kernel driver is loaded:
   - On Citrix Hypervisor, Red Hat Enterprise Linux KVM, and RHV, use `lsmod`:

     ```
     [root@xenserver ~]# lsmod|grep nvidia
     nvidia                 9604895  84
     i2c_core                 20294  2 nvidia,i2c_i801
     [root@xenserver ~]#
     ```

   - On VMware vSphere, use `vmkload_mod`:

     ```
     [root@esxi:~] vmkload_mod -l | grep nvidia
     nvidia                     5    8420
     ```

2. Run the `vmkload_mod` command.

   ```
   [root@esxi:~] vmkload_mod -l | grep nvidia
   nvidia                     5    8420
   ```

3. If the `nvidia` driver is not listed in the output, check `dmesg` for any load-time errors reported by the driver (see Examining NVIDIA kernel driver output).
4. On Citrix Hypervisor, Red Hat Enterprise Linux KVM, and RHV, also use the **`rpm -q`** command to verify that the NVIDIA GPU Manager package is correctly installed.

   ```
   rpm -q vgpu-manager-rpm-package-name
   ```

vgpu-manager-rpm-package-name

The RPM package name of the NVIDIA GPU Manager package, for example `NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-470.63` for Citrix Hypervisor.

This example verifies that the NVIDIA GPU Manager package for Citrix Hypervisor is correctly installed.

```
[root@xenserver ~]# rpm -q NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-470.63
[root@xenserver ~]#
If an existing NVIDIA GRID package is already installed and you don't select the
 upgrade (-U) option when installing a newer GRID package, the rpm command will
 return many conflict errors.
Preparing packages for installation...
        file /usr/bin/nvidia-smi from install of NVIDIA-vGPU-NVIDIA-vGPU-
CitrixHypervisor-8.2-470.63.x86_64 conflicts with file from package NVIDIA-vGPU-
xenserver-8.2-460.73.02.x86_64
        file /usr/lib/libnvidia-ml.so from install of NVIDIA-vGPU-NVIDIA-vGPU-
CitrixHypervisor-8.2-470.63.x86_64 conflicts with file from package NVIDIA-vGPU-
xenserver-8.2-460.73.02.x86_64
        ...
```

8.2.2. Verifying that `nvidia-smi` works

If the NVIDIA kernel driver is correctly loaded on the physical GPU, run `nvidia-smi` and verify that all physical GPUs not currently being used for GPU pass-through are listed in the output. For details on expected output, see NVIDIA System Management Interface nvidia-smi.

If `nvidia-smi` fails to report the expected output, check `dmesg` for NVIDIA kernel driver messages.

8.2.3. Examining NVIDIA kernel driver output

Information and debug messages from the NVIDIA kernel driver are logged in kernel logs, prefixed with `NVRM` or `nvidia`.

Run `dmesg` on Citrix Hypervisor, Red Hat Enterprise Linux KVM, RHV, and VMware vSphere and check for the `NVRM` and `nvidia` prefixes:

```
[root@xenserver ~]# dmesg | grep -E "NVRM|nvidia"
[   22.054928] nvidia: module license 'NVIDIA' taints kernel.
[   22.390414] NVRM: loading
[   22.829226] nvidia 0000:04:00.0: enabling device (0000 -> 0003)
[   22.829236] nvidia 0000:04:00.0: PCI INT A -> GSI 32 (level, low) -> IRQ 32
[   22.829240] NVRM: This PCI I/O region assigned to your NVIDIA device is invalid:
[   22.829241] NVRM: BAR0 is 0M @ 0x0 (PCI:0000:00:04.0)
[   22.829243] NVRM: The system BIOS may have misconfigured your GPU.
```

8.2.4. Examining NVIDIA Virtual GPU Manager Messages

Information and debug messages from the NVIDIA Virtual GPU Manager are logged to the hypervisor's log files, prefixed with `vmiop`.

8.2.4.1. Examining VMware vSphere vGPU Manager Messages

For VMware vSphere, NVIDIA Virtual GPU Manager messages are written to the `vmware.log` file in the guest VM's storage directory.

Look in the `vmware.log` file for the `vmiop` prefix:

```
[root@esxi:~] grep vmiop /vmfs/volumes/datastore1/win7-vgpu-test1/vmware.log
2021-08-27T14:02:21.275Z| vmx| I120: DICT   pciPassthru0.virtualDev = "vmiop"
2021-08-27T14:02:21.344Z| vmx| I120: GetPluginPath testing /usr/lib64/vmware/plugin/
libvmx-vmiop.so
2021-08-27T14:02:21.344Z| vmx| I120: PluginLdr_LoadShared: Loaded shared plugin
 libvmx-vmiop.so from /usr/lib64/vmware/plugin/libvmx-vmiop.so
2021-08-27T14:02:21.344Z| vmx| I120: VMIOP: Loaded plugin libvmx-
vmiop.so:VMIOP_InitModule
2021-08-27T14:02:21.359Z| vmx| I120: VMIOP: Initializing plugin vmiop-display
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: gpu-pci-id : 0000:04:00.0
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: vgpu_type : quadro
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: Framebuffer: 0x74000000
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: Virtual Device Id: 0x11B0:0x101B
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: ######## vGPU Manager Information:
 ########
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: Driver Version: 470.63
2021-08-27T14:02:21.365Z| vmx| I120: vmiop_log: VGX Version: 1.0
2021-08-27T14:02:21.445Z| vmx| I120: vmiop_log: Init frame copy engine: syncing...
2021-08-27T14:02:37.031Z| vthread-12| I120: vmiop_log: ######## Guest NVIDIA Driver
 Information: ########
2021-08-27T14:02:37.031Z| vthread-12| I120: vmiop_log: Driver Version: 471.68
2021-08-27T14:02:37.031Z| vthread-12| I120: vmiop_log: VGX Version: 1.0
2021-08-27T14:02:37.093Z| vthread-12| I120: vmiop_log: Clearing BAR1 mapping
2021-08-27T23:39:55.726Z| vmx| I120: VMIOP: Shutting down plugin vmiop-display
[root@esxi:~]
```

8.3. Capturing configuration data by running `nvidia-bug-report.sh`

The `nvidia-bug-report.sh` script captures debug information into a gzip-compressed log file on the server.

Run `nvidia-bug-report.sh` from the Citrix Hypervisor dom0 shell, the Red Hat Enterprise Linux KVM host shell, the Red Hat Virtualization (RHV) host shell, or the VMware ESXi host shell.

This example runs `nvidia-bug-report.sh` on Citrix Hypervisor, but the procedure is the same on Red Hat Enterprise Linux KVM, RHV, or VMware vSphere ESXi.

```
[root@xenserver ~]# nvidia-bug-report.sh

nvidia-bug-report.sh will now collect information about your
system and create the file 'nvidia-bug-report.log.gz' in the current
directory.  It may take several seconds to run.  In some
cases, it may hang trying to capture data generated dynamically
by the Linux kernel and/or the NVIDIA kernel module.  While
the bug report log file will be incomplete if this happens, it
may still contain enough data to diagnose your problem.
```

```
For Xen open source/XCP users, if you are reporting a domain issue,
please run: nvidia-bug-report.sh --domain-name <"domain_name">

Please include the 'nvidia-bug-report.log.gz' log file when reporting
your bug via the NVIDIA Linux forum (see devtalk.nvidia.com)
or by sending email to 'linux-bugs@nvidia.com'.

Running nvidia-bug-report.sh...

If the bug report script hangs after this point consider running with
--safe-mode command line argument.

 complete

[root@xenserver ~]#
```

Chapter 9. Additional Information

The following table provides links to additional information about each application or framework in NVIDIA AI Enterprise.

Application or Framework	Additional Information
TensorFlow	▶ TensorFlow Release Notes ▶ TensorFlow User Guide
PyTorch	PyTorch Release Notes
NVIDIA Triton Inference Server	Triton Inference Server Documentation on Github
NVIDIA TensorRT	NVIDIA TensorRT Documentation
RAPIDS	RAPIDS Docs on the RAPIDS project site

Other Software	Additional Information
NVIDIA GPU Operator	NVIDIA GPU Operator Documentation
NVIDIA Network Operator	NVIDIA Network Operator Documentation

Appendix A. Virtual GPU Types for Supported GPUs

NVIDIA vGPU is available as a licensed product on supported NVIDIA GPUs. For a list of recommended server platforms and supported GPUs, consult the release notes for supported hypervisors at NVIDIA AI Enterprise Documentation.

A.1. NVIDIA A100 PCIe 40GB Virtual GPU Types

Physical GPUs per board: 1

This GPU supports MIG-backed virtual GPUs and time-sliced virtual GPUs.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A100 PCIe 40GB

Required license edition: vCS or vWS

For details of GPU instance profiles, see *NVIDIA Multi-Instance GPU User Guide*.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100-7-40C	Training Workloads	40960	1	7	7	MIG 7g.40gb
A100-4-20C	Training Workloads	20480	1	4	4	MIG 4g.20gb
A100-3-20C	Training Workloads	20480	2	3	3	MIG 3g.20gb
A100-2-10C	Training Workloads	10240	3	2	2	MIG 2g.10gb
A100-1-5C	Inference Workloads	5120	7	1	1	MIG 1g.5gb

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100-1-5CME	Inference Workloads	5120	1	1	1	MIG 1g.5gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A100 PCIe 40GB

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A100-40C	Training Workloads	40960	1	1	4096×2160[2]	1
A100-20C	Training Workloads	20480	2	2	4096×2160[2]	1
A100-10C	Training Workloads	10240	4	4	4096×2160[2]	1
A100-8C	Training Workloads	8192	5	5	4096×2160[2]	1
A100-5C	Inference Workloads	5120	8	8	4096×2160[2]	1
A100-4C	Inference Workloads	4096	10	10	4096×2160[2]	1

A.2. NVIDIA A100 HGX 40GB Virtual GPU Types

Physical GPUs per board: 1
This GPU supports MIG-backed virtual GPUs and time-sliced virtual GPUs.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A100 HGX 40GB

Required license edition: vCS or vWS

For details of GPU instance profiles, see *NVIDIA Multi-Instance GPU User Guide*.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100X-7-40C	Training Workloads	40960	1	7	7	MIG 7g.40gb
A100X-4-20C	Training Workloads	20480	1	4	4	MIG 4g.20gb
A100X-3-20C	Training Workloads	20480	2	3	3	MIG 3g.20gb
A100X-2-10C	Training Workloads	10240	3	2	2	MIG 2g.10gb
A100X-1-5C	Inference Workloads	5120	7	1	1	MIG 1g.5gb
A100X-1-5CME	Inference Workloads	5120	1	1	1	MIG 1g.5gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A100 HGX 40GB

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A100X-40C	Training Workloads	40960	1	1	4096×2160[2]	1
A100X-20C	Training Workloads	20480	2	2	4096×2160[2]	1
A100X-10C	Training Workloads	10240	4	4	4096×2160[2]	1
A100X-8C	Training Workloads	8192	5	5	4096×2160[2]	1
A100X-5C	Inference Workloads	5120	8	8	4096×2160[2]	1
A100X-4C	Inference Workloads	4096	10	10	4096×2160[2]	1

A.3. NVIDIA A100 PCIe 80GB and NVIDIA A100X Virtual GPU Types

Physical GPUs per board: 1

This GPU supports MIG-backed virtual GPUs and time-sliced virtual GPUs.

The virtual GPU types for the NVIDIA A100 PCIe 80GB and NVIDIA A100X GPUs are identical.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A100 PCIe 80GB and NVIDIA A100X

Required license edition: vCS or vWS

For details of GPU instance profiles, see *NVIDIA Multi-Instance GPU User Guide*.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100D-7-80C	Training Workloads	81920	1	7	7	MIG 7g.80gb
A100D-4-40C	Training Workloads	40960	1	4	4	MIG 4g.40gb
A100D-3-40C	Training Workloads	40960	2	3	3	MIG 3g.40gb
A100D-2-20C	Training Workloads	20480	3	2	2	MIG 2g.20gb
A100D-1-10C	Training Workloads	10240	7	1	1	MIG 1g.10gb
A100D-1-10CME	Training Workloads	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A100 PCIe 80GB and NVIDIA A100X

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A100D-80C	Training Workloads	81920	1	1	4096×2160[2]	1
A100D-40C	Training Workloads	40960	2	2	4096×2160[2]	1
A100D-20C	Training Workloads	20480	4	4	4096×2160[2]	1
A100D-16C	Inference Workloads	16384	5	5	4096×2160[2]	1
A100D-10C	Training Workloads	10240	8	8	4096×2160[2]	1
A100D-8C	Training Workloads	8192	10	10	4096×2160[2]	1
A100D-4C	Inference Workloads	4096	20	20	4096×2160[2]	1

A.4. NVIDIA A100 HGX 80GB Virtual GPU Types

Physical GPUs per board: 1

This GPU supports MIG-backed virtual GPUs and time-sliced virtual GPUs.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A100 HGX 80GB

Required license edition: vCS or vWS

For details of GPU instance profiles, see *NVIDIA Multi-Instance GPU User Guide*.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100DX-7-80C	Training Workloads	81920	1	7	7	MIG 7g.80gb
A100DX-4-40C	Training Workloads	40960	1	4	4	MIG 4g.40gb
A100DX-3-40C	Training Workloads	40960	2	3	3	MIG 3g.40gb

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100DX-2-20C	Training Workloads	20480	3	2	2	MIG 2g.20gb
A100DX-1-10C	Training Workloads	10240	7	1	1	MIG 1g.10gb
A100DX-1-10CME	Training Workloads	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A100 HGX 80GB

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A100DX-80C	Training Workloads	81920	1	1	4096×2160[2]	1
A100DX-40C	Training Workloads	40960	2	2	4096×2160[2]	1
A100DX-20C	Training Workloads	20480	4	4	4096×2160[2]	1
A100DX-16C	Inference Workloads	16384	5	5	4096×2160[2]	1
A100DX-10C	Training Workloads	10240	8	8	4096×2160[2]	1
A100DX-8C	Training Workloads	8192	10	10	4096×2160[2]	1
A100DX-4C	Inference Workloads	4096	20	20	4096×2160[2]	1

A.5. NVIDIA A40 Virtual GPU Types

Physical GPUs per board: 1

Q-Series Virtual GPU Types for NVIDIA A40

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
A40-48Q	Virtual Workstations	49152	1	1	66355200	7680×4320	2
						5120×2880 or lower	4
A40-24Q	Virtual Workstations	24576	2	2	66355200	7680×4320	2
						5120×2880 or lower	4
A40-16Q	Virtual Workstations	16384	3	3	66355200	7680×4320	2
						5120×2880 or lower	4
A40-12Q	Virtual Workstations	12288	4	4	66355200	7680×4320	2
						5120×2880 or lower	4
A40-8Q	Virtual Workstations	8192	6	6	66355200	7680×4320	2
						5120×2880 or lower	4
A40-6Q	Virtual Workstations	6144	8	8	58982400	7680×4320	1
						5120×2880 or lower	4
A40-4Q	Virtual Workstations	4096	12	12	58982400	7680×4320	1
						5120×2880 or lower	4
A40-3Q	Virtual Workstations	3072	16	16	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
A40-2Q	Virtual Workstations	2048	24	24	35389440	7680×4320	1
						5120×2880	2

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
						4096×2160 or lower	4
A40-1Q	Virtual Workstations	1024	32[7]	32	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4

B-Series Virtual GPU Types for NVIDIA A40

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
A40-2B	Virtual Desktops	2048	24	24	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4
A40-1B	Virtual Desktops	1024	32	32	16384000	5120×2880	1
						4096×2160	1
						3840×2160	1
						2560×1600 or lower	4[5]

[7] The maximum vGPUs per GPU is limited to 32.

C-Series Virtual GPU Types for NVIDIA A40

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A40-48C	Training Workloads	49152	1	1	4096×2160[2]	1
A40-24C	Training Workloads	24576	2	2	4096×2160[2]	1
A40-16C	Training Workloads	16384	3	3	4096×2160[2]	1
A40-12C	Training Workloads	12288	4	4	4096×2160[2]	1
A40-8C	Training Workloads	8192	6	6	4096×2160[2]	1
A40-6C	Training Workloads	6144	8	8	4096×2160[2]	1
A40-4C	Inference Workloads	4096	8[3]	12	4096×2160[2]	1

A-Series Virtual GPU Types for NVIDIA A40

Required license edition: vApps

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A40-48A	Virtual Applications	49152	1	1	1280×1024[6]	1[6]
A40-24A	Virtual Applications	24576	2	2	1280×1024[6]	1[6]
A40-16A	Virtual Applications	16384	3	3	1280×1024[6]	1[6]

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A40-12A	Virtual Applications	12288	4	4	1280×1024[6]	1[6]
A40-8A	Virtual Applications	8192	6	6	1280×1024[6]	1[6]
A40-6A	Virtual Applications	6144	8	8	1280×1024[6]	1[6]
A40-4A	Virtual Applications	4096	12	12	1280×1024[6]	1[6]
A40-3A	Virtual Applications	3072	16	16	1280×1024[6]	1[6]
A40-2A	Virtual Applications	2048	24	24	1280×1024[6]	1[6]
A40-1A	Virtual Applications	1024	32[7]	32	1280×1024[6]	1[6]

A.6. NVIDIA A30 and NVIDIA A30X Virtual GPU Types

Physical GPUs per board: 1

This GPU supports MIG-backed virtual GPUs and time-sliced virtual GPUs.

The virtual GPU types for the NVIDIA A30 and NVIDIA A30X GPUs are identical.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A30 and NVIDIA A30X

Required license edition: vCS or vWS

For details of GPU instance profiles, see *NVIDIA Multi-Instance GPU User Guide*.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A30-4-24C	Training Workloads	24576	1	4	4	MIG 4g.24gb
A30-2-12C	Training Workloads	12288	2	2	2	MIG 2g.12gb

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A30-1-6C	Inference Workloads	6144	4	1	1	MIG 1g.6gb
A30-1-6CME	Inference Workloads	6144	1	1	1	MIG 1g.6gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A30 and NVIDIA A30X

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A30-24C	Training Workloads	24576	1	1	4096×2160[2]	1
A30-12C	Training Workloads	12288	2	2	4096×2160[2]	1
A30-8C	Training Workloads	8192	3	3	4096×2160[2]	1
A30-6C	Inference Workloads	6144	4	4	4096×2160[2]	1
A30-4C	Inference Workloads	4096	6	6	4096×2160[2]	1

A.7. NVIDIA A10 Virtual GPU Types

Physical GPUs per board: 1

Q-Series Virtual GPU Types for NVIDIA A10

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
A10-24Q	Virtual Workstations	24576	1	1	66355200	7680×4320	2
						5120×2880 or lower	4
A10-12Q	Virtual Workstations	12288	2	2	66355200	7680×4320	2
						5120×2880 or lower	4
A10-8Q	Virtual Workstations	8192	3	3	66355200	7680×4320	2
						5120×2880 or lower	4
A10-6Q	Virtual Workstations	6144	4	4	58982400	7680×4320	1
						5120×2880 or lower	4
A10-4Q	Virtual Workstations	4096	6	6	58982400	7680×4320	1
						5120×2880 or lower	4
A10-3Q	Virtual Workstations	3072	8	8	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
A10-2Q	Virtual Workstations	2048	12	12	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
A10-1Q	Virtual Workstations	1024	24	24	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4

B-Series Virtual GPU Types for NVIDIA A10

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
A10-2B	Virtual Desktops	2048	12	12	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4
A10-1B	Virtual Desktops	1024	24	24	16384000	5120×2880	1
						4096×2160	1
						3840×2160	1
						2560×1600 or lower	4[5]

C-Series Virtual GPU Types for NVIDIA A10

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A10-24C	Training Workloads	24576	1	1	4096×2160[2]	1
A10-12C	Training Workloads	12288	2	2	4096×2160[2]	1
A10-8C	Training Workloads	8192	3	3	4096×2160[2]	1
A10-6C	Training Workloads	6144	4	4	4096×2160[2]	1

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A10-4C	Inference Workloads	4096	6	6	4096×2160[2]	1

A-Series Virtual GPU Types for NVIDIA A10

Required license edition: vApps

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
A10-24A	Virtual Applications	24576	1	1	1280×1024[6]	1[6]
A10-12A	Virtual Applications	12288	2	2	1280×1024[6]	1[6]
A10-8A	Virtual Applications	8192	3	3	1280×1024[6]	1[6]
A10-6A	Virtual Applications	6144	4	4	1280×1024[6]	1[6]
A10-4A	Virtual Applications	4096	6	6	1280×1024[6]	1[6]
A10-3A	Virtual Applications	3072	8	8	1280×1024[6]	1[6]
A10-2A	Virtual Applications	2048	12	12	1280×1024[6]	1[6]
A10-1A	Virtual Applications	1024	24	24	1280×1024[6]	1[6]

A.8. NVIDIA RTX A6000 Virtual GPU Types

Physical GPUs per board: 1

Q-Series Virtual GPU Types for NVIDIA RTX A6000

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA6000-48Q	Virtual Workstations	49152	1	1	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA6000-24Q	Virtual Workstations	24576	2	2	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA6000-16Q	Virtual Workstations	16384	3	3	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA6000-12Q	Virtual Workstations	12288	4	4	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA6000-8Q	Virtual Workstations	8192	6	6	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA6000-6Q	Virtual Workstations	6144	8	8	58982400	7680×4320	1
						5120×2880 or lower	4
RTXA6000-4Q	Virtual Workstations	4096	12	12	58982400	7680×4320	1
						5120×2880 or lower	4
RTXA6000-3Q	Virtual Workstations	3072	16	16	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
RTXA6000-2Q	Virtual Workstations	2048	24	24	35389440	7680×4320	1
						5120×2880	2

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
						4096×2160 or lower	4
RTXA6000-1Q	Virtual Workstations	1024	32[8]	32	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4

B-Series Virtual GPU Types for NVIDIA RTX A6000

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA6000-2B	Virtual Desktops	2048	24	24	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4
RTXA6000-1B	Virtual Desktops	1024	32	32	16384000	5120×2880	1
						4096×2160	1
						3840×2160	1
						2560×1600 or lower	4[5]

[8] The maximum vGPUs per GPU is limited to 32.

C-Series Virtual GPU Types for NVIDIA RTX A6000

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
RTXA6000-48C	Training Workloads	49152	1	1	4096×2160[2]	1
RTXA6000-24C	Training Workloads	24576	2	2	4096×2160[2]	1
RTXA6000-16C	Training Workloads	16384	3	3	4096×2160[2]	1
RTXA6000-12C	Training Workloads	12288	4	4	4096×2160[2]	1
RTXA6000-8C	Training Workloads	8192	6	6	4096×2160[2]	1
RTXA6000-6C	Training Workloads	6144	8	8	4096×2160[2]	1
RTXA6000-4C	Inference Workloads	4096	8[3]	12	4096×2160[2]	1

A-Series Virtual GPU Types for NVIDIA RTX A6000

Required license edition: vApps

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
RTXA6000-48A	Virtual Applications	49152	1	1	1280×1024[6]	1[6]
RTXA6000-24A	Virtual Applications	24576	2	2	1280×1024[6]	1[6]
RTXA6000-16A	Virtual Applications	16384	3	3	1280×1024[6]	1[6]

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
RTXA6000-12A	Virtual Applications	12288	4	4	1280×1024[6]	1[6]
RTXA6000-8A	Virtual Applications	8192	6	6	1280×1024[6]	1[6]
RTXA6000-6A	Virtual Applications	6144	8	8	1280×1024[6]	1[6]
RTXA6000-4A	Virtual Applications	4096	12	12	1280×1024[6]	1[6]
RTXA6000-3A	Virtual Applications	3072	16	16	1280×1024[6]	1[6]
RTXA6000-2A	Virtual Applications	2048	24	24	1280×1024[6]	1[6]
RTXA6000-1A	Virtual Applications	1024	32[8]	32	1280×1024[6]	1[6]

A.9. NVIDIA RTX A5000 Virtual GPU Types

Physical GPUs per board: 1

Q-Series Virtual GPU Types for NVIDIA RTX A5000

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5000-24Q	Virtual Workstations	24576	1	1	66355200	7680×4320	2
						5120×2880 or lower	4

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5000-12Q	Virtual Workstations	12288	2	2	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA5000-8Q	Virtual Workstations	8192	3	3	66355200	7680×4320	2
						5120×2880 or lower	4
RTXA5000-6Q	Virtual Workstations	6144	4	4	58982400	7680×4320	1
						5120×2880 or lower	4
RTXA5000-4Q	Virtual Workstations	4096	6	6	58982400	7680×4320	1
						5120×2880 or lower	4
RTXA5000-3Q	Virtual Workstations	3072	8	8	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
RTXA5000-2Q	Virtual Workstations	2048	12	12	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
RTXA5000-1Q	Virtual Workstations	1024	24	24	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4

B-Series Virtual GPU Types for NVIDIA RTX A5000

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which

all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5000-2B	Virtual Desktops	2048	12	12	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4
RTXA5000-1B	Virtual Desktops	1024	24	24	16384000	5120×2880	1
						4096×2160	1
						3840×2160	1
						2560×1600 or lower	4[5]

C-Series Virtual GPU Types for NVIDIA RTX A5000

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5000-24C	Training Workloads	24576	1	1	4096×2160[2]	1
RTXA5000-12C	Training Workloads	12288	2	2	4096×2160[2]	1
RTXA5000-8C	Training Workloads	8192	3	3	4096×2160[2]	1
RTXA5000-6C	Training Workloads	6144	4	4	4096×2160[2]	1
RTXA5000-4C	Inference Workloads	4096	6	6	4096×2160[2]	1

A-Series Virtual GPU Types for NVIDIA RTX A5000

Required license edition: vApps

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5000-24A	Virtual Applications	24576	1	1	1280×1024[6]	1[6]
RTXA5000-12A	Virtual Applications	12288	2	2	1280×1024[6]	1[6]
RTXA5000-8A	Virtual Applications	8192	3	3	1280×1024[6]	1[6]
RTXA5000-6A	Virtual Applications	6144	4	4	1280×1024[6]	1[6]
RTXA5000-4A	Virtual Applications	4096	6	6	1280×1024[6]	1[6]
RTXA5000-3A	Virtual Applications	3072	8	8	1280×1024[6]	1[6]
RTXA5000-2A	Virtual Applications	2048	12	12	1280×1024[6]	1[6]
RTXA5000-1A	Virtual Applications	1024	24	24	1280×1024[6]	1[6]

A.10. Tesla T4 Virtual GPU Types

Physical GPUs per board: 1

Q-Series Virtual GPU Types for Tesla T4

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
T4-16Q	Virtual Workstations	16384	1	1	66355200	7680×4320	2
						5120×2880 or lower	4
T4-8Q	Virtual Workstations	8192	2	2	66355200	7680×4320	2
						5120×2880 or lower	4
T4-4Q	Virtual Workstations	4096	4	4	58982400	7680×4320	1
						5120×2880 or lower	4
T4-2Q	Virtual Workstations	2048	8	8	35389440	7680×4320	1
						5120×2880	2
						4096×2160 or lower	4
T4-1Q	Virtual Desktops, Virtual Workstations	1024	16	16	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4

B-Series Virtual GPU Types for Tesla T4

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see #unique_118.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
T4-2B	Virtual Desktops	2048	8	8	17694720	5120×2880	1
						4096×2160	2

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Available Pixels	Display Resolution	Virtual Displays per vGPU
						3840×2160	2
						2560×1600 or lower	4
T4-2B4[4]	Virtual Desktops	2048	8	8	17694720	5120×2880	1
						4096×2160	2
						3840×2160	2
						2560×1600 or lower	4
T4-1B	Virtual Desktops	1024	16	16	16384000	5120×2880	1
						4096×2160	1
						3840×2160	1
						2560×1600 or lower	4[5]
T4-1B4[4]	Virtual Desktops	1024	16	16	16384000	5120×2880	1
						4096×2160	1
						3840×2160	1
						2560×1600 or lower	4[5]

C-Series Virtual GPU Types for Tesla T4

Required license edition: vCS or vWS

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
T4-16C	Training Workloads	16384	1	1	4096×2160[2]	1
T4-8C	Training Workloads	8192	2	2	4096×2160[2]	1
T4-4C	Inference Workloads	4096	4	4	4096×2160[2]	1

A-Series Virtual GPU Types for Tesla T4

Required license edition: vApps

These vGPU types support a single display with a fixed maximum resolution.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum vGPUs per Board	Maximum Display Resolution	Virtual Displays per vGPU
T4-16A	Virtual Applications	16384	1	1	1280×1024[6]	1[6]
T4-8A	Virtual Applications	8192	2	2	1280×1024[6]	1[6]
T4-4A	Virtual Applications	4096	4	4	1280×1024[6]	1[6]
T4-2A	Virtual Applications	2048	8	8	1280×1024[6]	1[6]
T4-1A	Virtual Applications	1024	16	16	1280×1024[6]	1[6]

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, NVIDIA Maxwell, NVIDIA Pascal, NVIDIA Turing, NVIDIA Volta, Quadro, and Tesla are trademarks 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

© 2024 NVIDIA Corporation & affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com