NVIDIA AI Enterprise

User Guide

DU-10617-001 _v5.0 | October 2024

Table of Contents

List of Figures

List of Tables

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

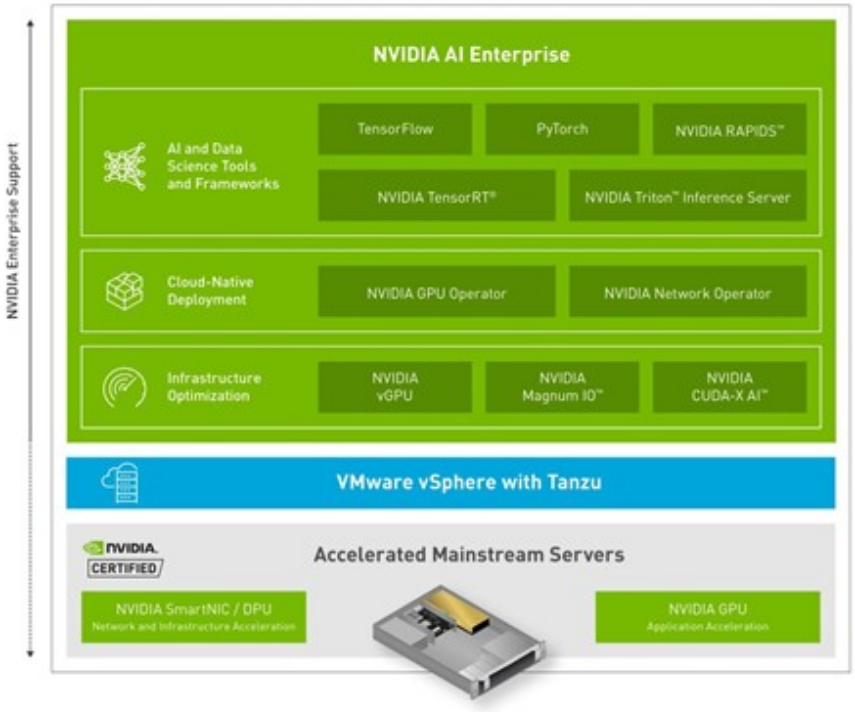
The software in the NVIDIA AI Enterprise suite is organized into separate layers for infrastructure optimization software, cloud native deployment software, and AI and data science frameworks.

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.

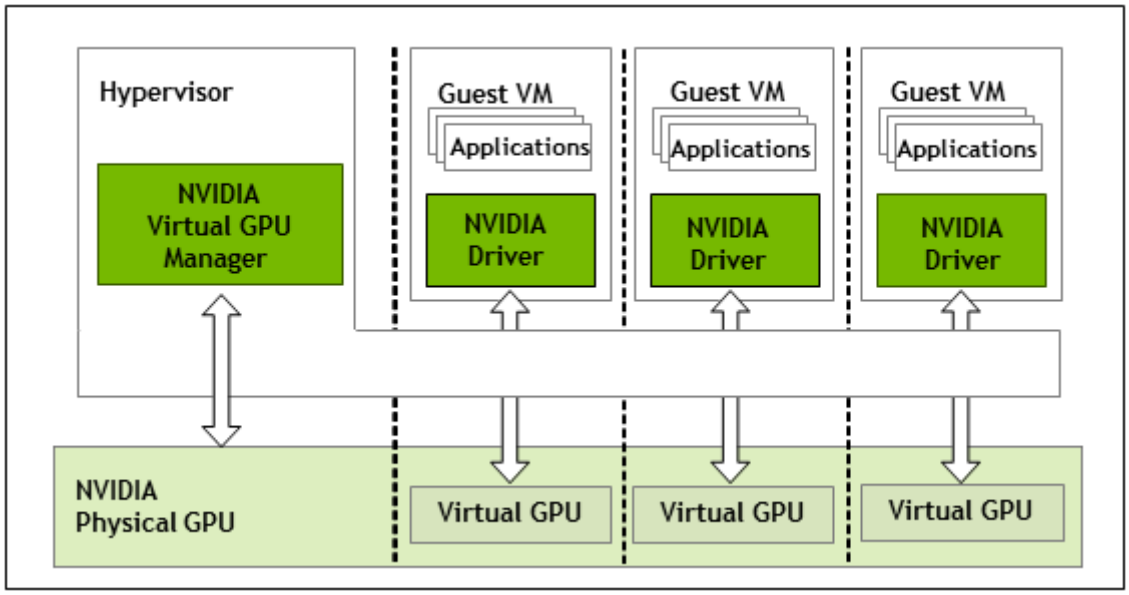
2.1. About NVIDIA Virtual GPUs

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 and the GPU virtualization software, NVIDIA Virtual GPU Manager supports different types of vGPU on a physical GPU:

- 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 and NVIDIA AI Enterprise, MIG-backed vGPUs are supported. The MIG feature is introduced on GPUs that are based on the NVIDIA Ampere GPU architecture.

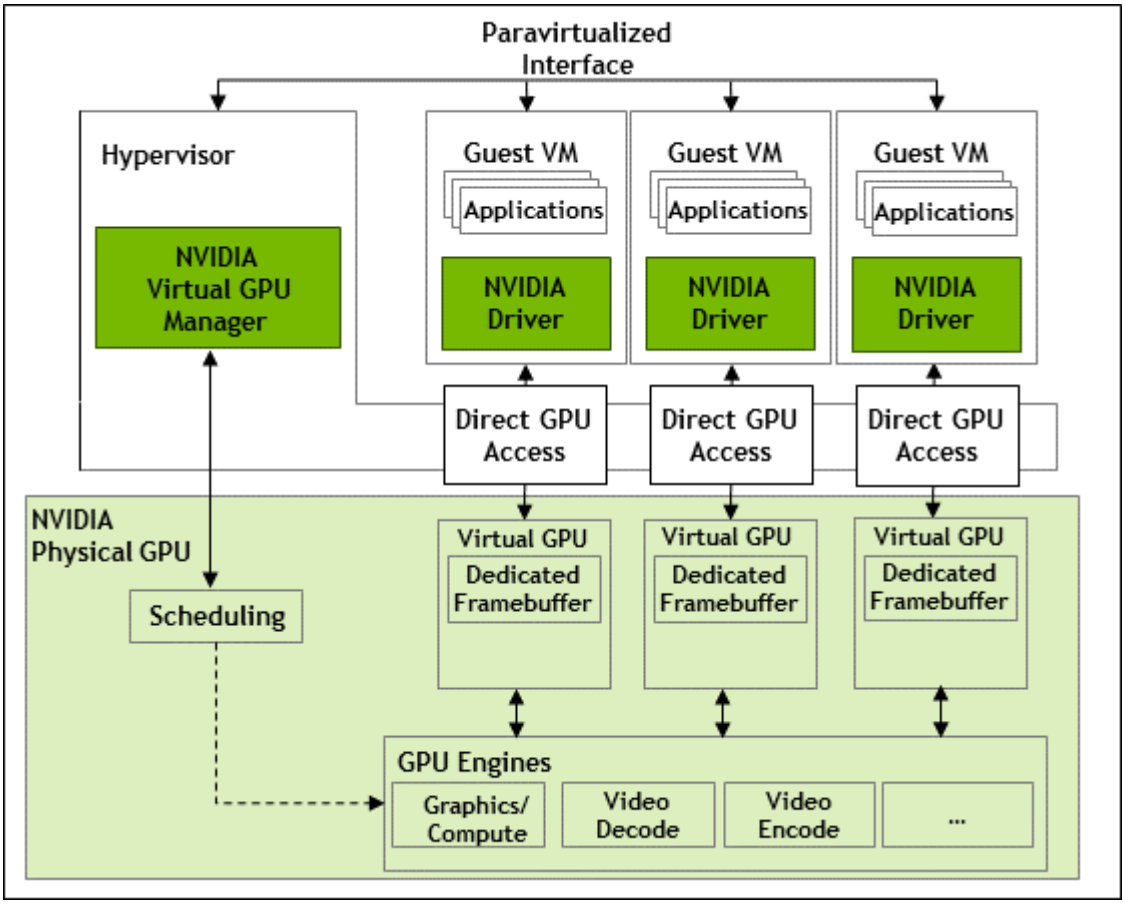
Note: Although earlier releases of NVIDIA AI Enterprise supported GPUs that support the MIG feature, such GPUs are **not** supported on this release of NVIDIA AI Enterprise. GPUs that support the MIG feature are supported **only** on NVIDIA AI Enterprise.

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

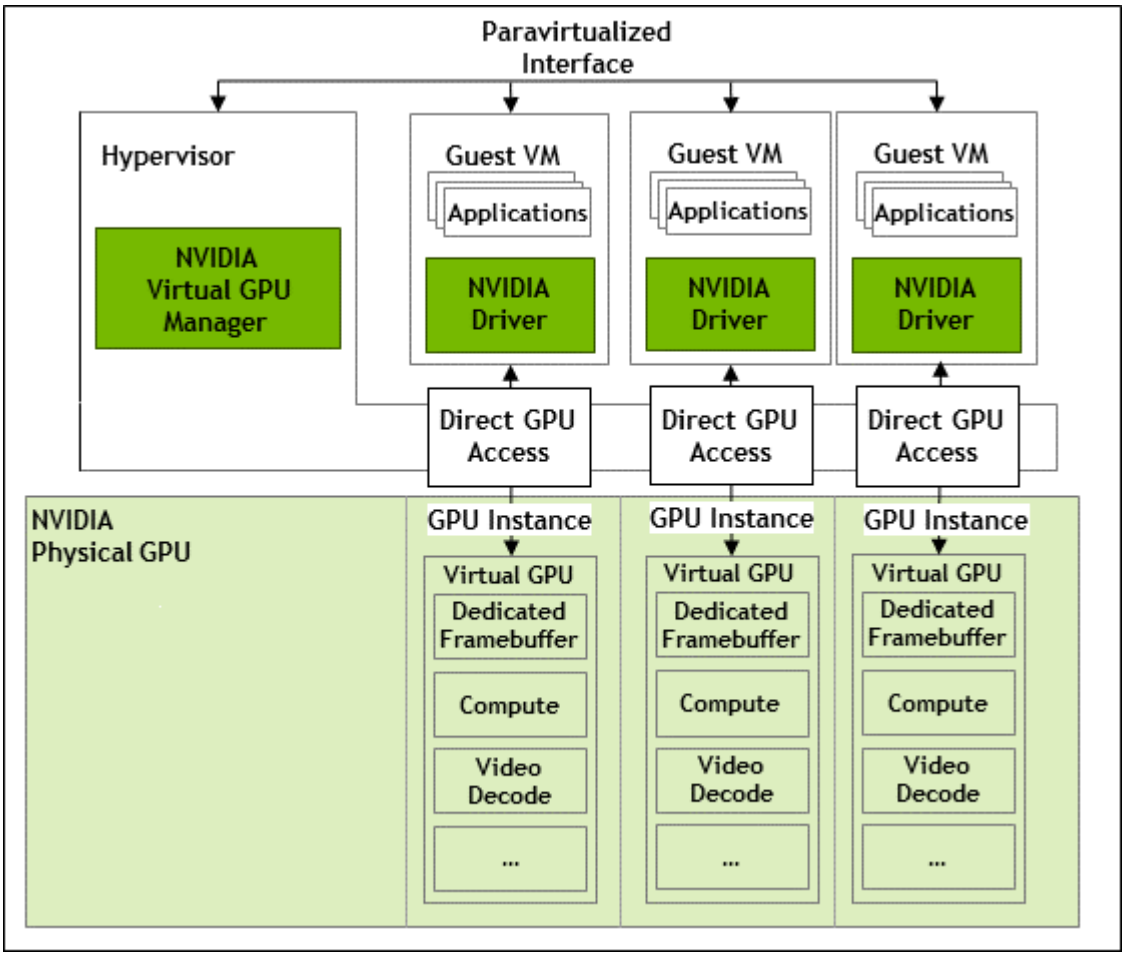


2.1.1.2. MIG-Backed NVIDIA vGPU Internal Architecture

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. 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
C-series	Compute-intensive server workloads, such as artificial intelligence (AI), deep learning, or high-performance computing (HPC)[1, 2]

[1] 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.

[2] The maximum number of NVIDIA Virtual Compute Server vGPUs is limited to **12** vGPUs per physical GPU, irrespective of the available hardware resources of the physical GPU.

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 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. 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

NVIDIA AI Enterprise supports a mixture of different types of time-sliced vGPUs on the same physical GPU. Any combination of A-series, B-series, and Q-series vGPUs with any amount of frame buffer can reside on the same physical GPU simultaneously. The total amount of frame buffer allocated to the vGPUs on a physical GPU must not exceed the amount of frame buffer that the physical GPU has.

For example, the following combinations of vGPUs can reside on the same physical GPU simultaneously:

- A40-2B and A40-2Q
- A40-2Q and A40-4Q
- A40-2B and A40-4Q

By default, a GPU supports only vGPUs with the same amount of frame buffer and, therefore, is in equal-size mode. To support vGPUs with different amounts of frame buffer, the GPU must be put into mixed-size mode. When a GPU is in mixed-size mode,

the maximum number of some types of vGPU allowed on a GPU is less than when the GPU is in equal-size mode. For more information, refer to the following topics:

- Putting a GPU Into Mixed-Size Mode
- Virtual GPU Types for Supported GPUs

Not all hypervisors and GPUs support a mixture of different types of time-sliced vGPUs on the same physical GPU. To determine if your chosen hypervisor supports this feature with your chosen GPU, consult the release notes for your hypervisor at NVIDIA AI Enterprise Documentation.

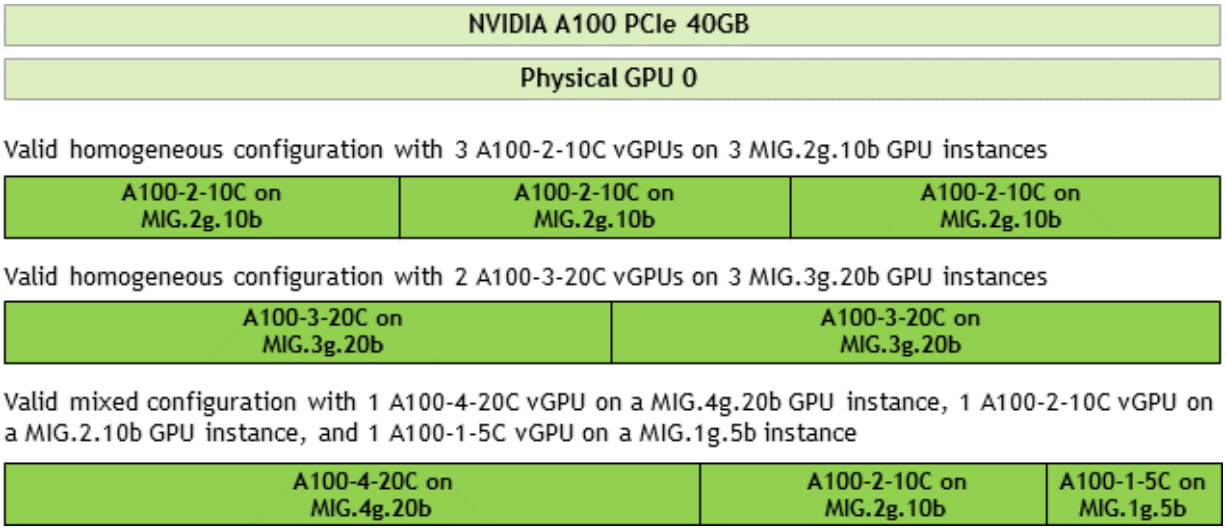
2.1.3.2. Valid MIG-Backed Virtual GPU Configurations on a Single GPU

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 display-off and display-enabled modes but must be used in NVIDIA AI Enterprise deployments in display-off 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 display-off mode, but other GPUs are supplied in a display-enabled mode.

GPU	Mode as Supplied from the Factory
NVIDIA A40	Display-off
NVIDIA L40	Display-off
NVIDIA L40S	Display-off
NVIDIA L20	Display-off
NVIDIA L20 liquid cooled	Display-off
NVIDIA RTX 5000 Ada	Display enabled
NVIDIA RTX 6000 Ada	Display enabled
NVIDIA RTX A5000	Display enabled
NVIDIA RTX A5500	Display enabled
NVIDIA RTX A6000	Display enabled

A GPU that is supplied from the factory in display-off 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 GPUs listed in the table support the `displaymodeselector` tool. 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. Downloading NVIDIA AI Enterprise

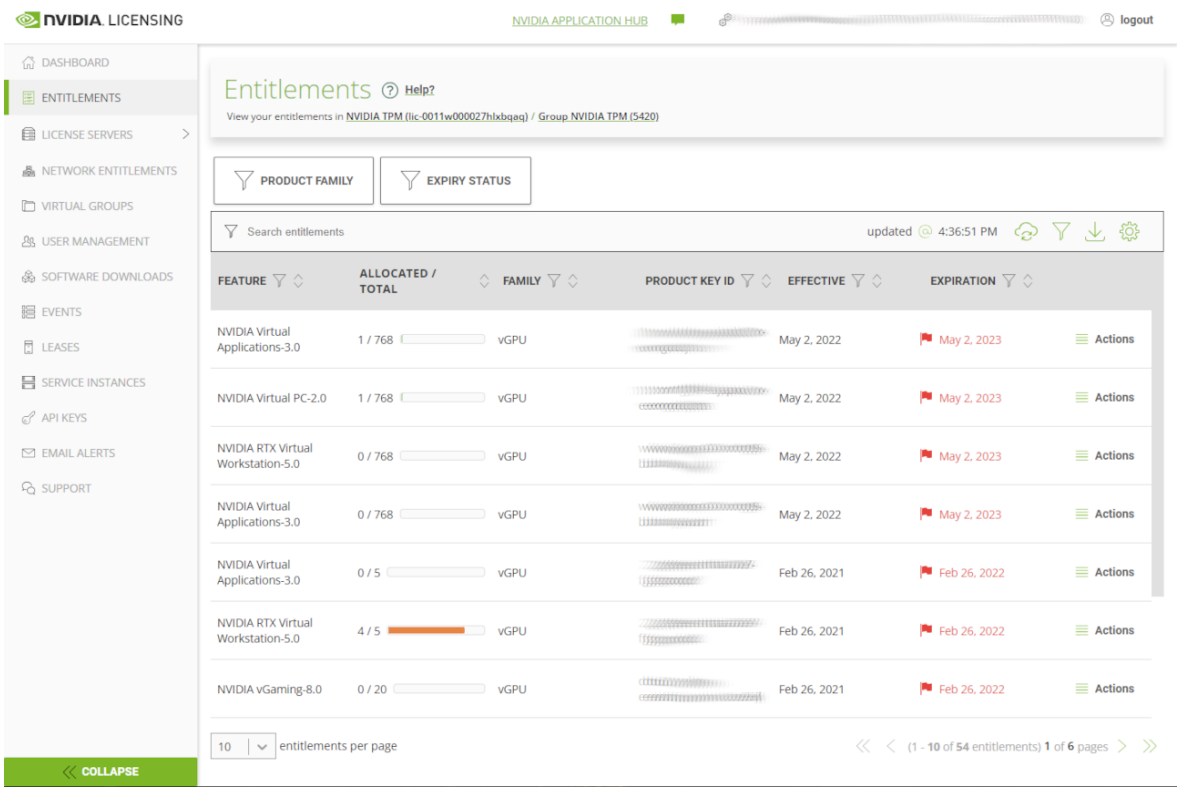
Before you begin, ensure that you have your order confirmation message and have created an NVIDIA Enterprise Account.

1. Visit the NVIDIA Application Hub by following the **Login** link in the instructions for using your NVIDIA Entitlement Certificate or when prompted after setting the password for your NVIDIA Enterprise Account.
2. When prompted, provide your e-mail address and password, and click **LOGIN**.
3. On the **NVIDIA APPLICATION HUB** page that opens, click **NVIDIA LICENSING PORTAL**.

The NVIDIA Licensing Portal dashboard page opens.

Note: Your entitlement might not appear on the NVIDIA Licensing Portal dashboard page until 24 business hours after you set your password during the initial registration process.

4. In the left navigation pane of the NVIDIA Licensing Portal dashboard page, click **ENTITLEMENTS** to view details of the NVIDIA AI Enterprise entitlements that you purchased.



5. In the left navigation pane of the NVIDIA Licensing Portal dashboard page, click **SOFTWARE DOWNLOADS**.
6. On the **Software Downloads** page that opens, download the NVIDIA AI Enterprise drivers that you require.
   a). Ensure that the **Driver downloads** tab is selected.
   b). Set the **PRODUCT FAMILY** option to **NVAIE**.
   c). Follow the **Download** link for the brand and version of your chosen hypervisor for the release of NVIDIA AI Enterprise that you are using.
   For example: NVIDIA AI Enterprise for vSphere 7.0.3 for NVIDIA AI Enterprise release 17.1.

   If the brand and version of your chosen hypervisor for the release of NVIDIA AI Enterprise that you are using aren't displayed, click **ALL AVAILABLE** to display a list

of all available NVIDIA AI Enterprise downloads. Set filters on columns in the table to filter the software listed.

d). When prompted to accept the license for the software that you are downloading, click **AGREE & DOWNLOAD**.

7. If necessary, download the standalone **NVIDIA Control Panel** installer.
   a). Ensure that the **Driver downloads** tab is selected.
   b). Set the filter on the **DESCRIPTION** column to **`control`**.
   c). Follow the **Download** link for the standalone **NVIDIA Control Panel** installer.
   d). When prompted to accept the license for the software that you are downloading, click **AGREE & DOWNLOAD**.
8. Download any additional, non-driver software that you need for your NVIDIA AI Enterprise deployment.
   a). Click the **Non-Driver downloads** tab.
   b). **Optional:** Use the **CATEGORY** filter to list only the category of software that you are interested in, for example, **DLS**.
   c). Follow the **Download** link for the software that you want to download.
   - If you are using Delegated License Service (DLS) instances to serve licenses, follow the link to the DLS release for your chosen platform, for example, **NLS License Server (DLS) 3.2 for VMware vSphere**.

     For information about installing and configuring DLS instances, refer to *NVIDIA License System User Guide*.
   - If you are using NVIDIA GPU Operator, follow the **vGPU Driver Catalog** link.

   d). When prompted to accept the license for the software that you are downloading, click **AGREE & DOWNLOAD**.

2.4. Installing the Virtual GPU Manager Package for Linux KVM

Before installing the Virtual GPU Manager package for Linux KVM, ensure that the following prerequisites are met:

- The following packages are installed on the Linux KVM server:
  - The `x86_64` build of the GNU Compiler Collection (GCC)
  - Linux kernel headers
- The package file is copied to a directory in the file system of the Linux KVM server.

If the Nouveau driver for NVIDIA graphics cards is present, disable it before installing the package.

1. Change to the directory on the Linux KVM server that contains the package file.

```
# cd package-file-directory
```

package-file-directory
The path to the directory that contains the package file.

2. Make the package file executable.

```
# chmod +x package-file-name
```

package-file-name
The name of the file that contains the Virtual GPU Manager package for Linux KVM, for example `NVIDIA-Linux-x86_64-390.42-vgpu-kvm.run`.

3. Run the package file as the root user.

```
# sudo sh./package-file-name
```

The package file should launch and display the license agreement.

4. Accept the license agreement to continue with the installation.
5. When installation has completed, select **OK** to exit the installer.
6. Reboot the Linux KVM server.

```
# systemctl reboot
```

2.5. Installing and Configuring the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM

The following topics step you through the process of setting up a single Red Hat Enterprise Linux Kernel-based Virtual Machine (KVM) VM to use NVIDIA vGPU.

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

Follow this sequence of instructions:

1. Installing the Virtual GPU Manager Package for Red Hat Enterprise Linux KVM
2. Verifying the Installation of the NVIDIA AI Enterprise for Red Hat Enterprise Linux KVM
3. **MIG-backed vGPUs only:** Configuring a GPU for MIG-Backed vGPUs
4. **vGPUs that support SR-IOV only:** Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor
5. **Optional:** Putting a GPU Into Mixed-Size Mode
6. Getting the BDF and Domain of a GPU on a Linux with KVM Hypervisor
7. Creating an NVIDIA vGPU on a Linux with KVM Hypervisor
8. Adding One or More vGPUs to a Linux with KVM Hypervisor VM
9. **Optional:** Placing a vGPU on a Physical GPU in Mixed-Size Mode
10. Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor

After the process is complete, you can install the graphics driver for your guest OS and license any NVIDIA AI Enterprise licensed products that you are using.

2.5.1. Installing the Virtual GPU Manager Package for Red Hat Enterprise Linux KVM

The NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM is provided as a `.rpm` file.

> **CAUTION:** NVIDIA Virtual GPU Manager and guest VM drivers must be compatible. If you update vGPU Manager to a release that is incompatible with the guest VM drivers, guest VMs will boot with vGPU disabled until their guest vGPU driver is updated to a compatible version.

Before installing the RPM package for Red Hat Enterprise Linux KVM, ensure that the `sshd` service on the Red Hat Enterprise Linux KVM server is configured to permit root login. If the Nouveau driver for NVIDIA graphics cards is present, disable it before installing the package. For instructions, see How to disable the Nouveau driver and install the Nvidia driver in RHEL 7 (Red Hat subscription required).

Some versions of Red Hat Enterprise Linux KVM have z-stream updates that break Kernel Application Binary Interface (kABI) compatibility with the previous kernel or the GA kernel. For these versions of Red Hat Enterprise Linux KVM, the following Virtual GPU Manager RPM packages are supplied:

- A package for the GA Linux KVM kernel
- A package for the updated z-stream kernel

To differentiate these packages, the name of each RPM package includes the kernel version. Ensure that you install the RPM package that is compatible with your Linux KVM kernel version.

1. Securely copy the RPM file from the system where you downloaded the file to the Red Hat Enterprise Linux KVM server.
   - From a Windows system, use a secure copy client such as WinSCP.
   - From a Linux system, use the `scp` command.
2. Use secure shell (SSH) to log in as root to the Red Hat Enterprise Linux KVM server.

   ```
   # ssh root@kvm-server
   ```

   kvm-server
   The host name or IP address of the Red Hat Enterprise Linux KVM server.
3. Change to the directory on the Red Hat Enterprise Linux KVM server to which you copied the RPM file.

   ```
   # cd rpm-file-directory
   ```

   rpm-file-directory
   The path to the directory to which you copied the RPM file.
4. Use the `rpm` command to install the package.

   ```
   # rpm -iv NVIDIA-vGPU-rhel-8.9-550.54.16.x86_64.rpm
   Preparing packages for installation...
   ```

```
NVIDIA-vGPU-rhel-8.9-550.54.16
#
```

5. Reboot the Red Hat Enterprise Linux KVM server.

```
# systemctl reboot
```

2.5.2. Verifying the Installation of the NVIDIA AI Enterprise for Red Hat Enterprise Linux KVM

After the Red Hat Enterprise Linux KVM server has rebooted, verify the installation of the NVIDIA AI Enterprise package for Red Hat Enterprise Linux KVM.

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

```
# lsmod | grep vfio
nvidia_vgpu_vfio       27099  0
nvidia              12316924  1 nvidia_vgpu_vfio
vfio_mdev              12841  0
mdev                   20414  2 vfio_mdev,nvidia_vgpu_vfio
vfio_iommu_type1       22342  0
vfio                   32331  3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1
#
```

2. Verify that the `libvirtd` service is active and running.

```
# service libvirtd status
```

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.

```
# nvidia-smi
Fri Mar 22 18:46:50 2024
+-----------------------------------------------------------+
| NVIDIA-SMI 550.54.16    Driver Version: 550.54.16         |
|-------------------------------+----------------------+----------------------+
| 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   | 0000:85:00.0     Off |                  Off |
| N/A   23C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           On   | 0000:86:00.0     Off |                  Off |
| N/A   29C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P40           On   | 0000:87:00.0     Off |                  Off |
| N/A   21C    P8    18W / 250W |     53MiB / 24575MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
#
```

If `nvidia-smi` fails to run or doesn't produce the expected output for all the NVIDIA GPUs in your system, see Troubleshooting for troubleshooting steps.

2.6. Installing and Configuring the NVIDIA Virtual GPU Manager for Ubuntu

Follow this sequence of instructions to set up a single Ubuntu VM to use NVIDIA vGPU.

1. Installing the NVIDIA Virtual GPU Manager for Ubuntu
2. **MIG-backed vGPUs only:** Configuring a GPU for MIG-Backed vGPUs
3. Getting the BDF and Domain of a GPU on a Linux with KVM Hypervisor
4. **vGPUs that support SR-IOV only:** Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor
5. **Optional:** Putting a GPU Into Mixed-Size Mode
6. Creating an NVIDIA vGPU on a Linux with KVM Hypervisor
7. Adding One or More vGPUs to a Linux with KVM Hypervisor VM
8. **Optional:** Placing a vGPU on a Physical GPU in Mixed-Size Mode
9. Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor

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

After the process is complete, you can install the graphics driver for your guest OS and license any NVIDIA AI Enterprise licensed products that you are using.

2.6.1. Installing the NVIDIA Virtual GPU Manager for Ubuntu

The NVIDIA Virtual GPU Manager for Ubuntu is provided as a Debian package (`.deb`) file.

> **CAUTION:** NVIDIA Virtual GPU Manager and guest VM drivers must be compatible. If you update vGPU Manager to a release that is incompatible with the guest VM drivers, guest VMs will boot with vGPU disabled until their guest vGPU driver is updated to a compatible version. Consult for further details.

2.6.1.1. Installing the Virtual GPU Manager Package for Ubuntu

Before installing the Debian package for Ubuntu, ensure that the `sshd` service on the Ubuntu server is configured to permit root login. If the Nouveau driver for NVIDIA graphics cards is present, disable it before installing the package.

1. Securely copy the Debian package file from the system where you downloaded the file to the Ubuntu server.
   - From a Windows system, use a secure copy client such as WinSCP.
   - From a Linux system, use the `scp` command.
2. Use secure shell (SSH) to log in as root to the Ubuntu server.

   ```
   # ssh root@ubuntu-server
   ```

   ubuntu-server
   The host name or IP address of the Ubuntu server.
3. Change to the directory on the Ubuntu server to which you copied the Debian package file.

   ```
   # cd deb-file-directory
   ```

   deb-file-directory
   The path to the directory to which you copied the Debian package file.
4. Use the `apt` command to install the package.

   ```
   # apt install ./.deb
   ```
5. Reboot the Ubuntu server.

   ```
   # systemctl reboot
   ```

2.6.1.2. Verifying the Installation of the NVIDIA AI Enterprise for Ubuntu

After the Ubuntu server has rebooted, verify the installation of the NVIDIA AI Enterprise package for Ubuntu.

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

   ```
   # lsmod | grep vfio
   nvidia_vgpu_vfio       27099  0
   nvidia              12316924  1 nvidia_vgpu_vfio
   vfio_mdev              12841  0
   mdev                   20414  2 vfio_mdev,nvidia_vgpu_vfio
   vfio_iommu_type1       22342  0
   vfio                   32331  3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1
   #
   ```
2. Verify that the `libvirtd` service is active and running.

   ```
   # service libvirtd status
   ```
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.

```
# nvidia-smi
Fri Mar 22 18:46:50 2024
+-----------------------------------------------------------+
| NVIDIA-SMI 550.54.16    Driver Version: 550.54.16         |
|-----------------------------------+----------------------+----------------------+
| 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    | 0000:85:00.0     Off |                  Off |
| N/A   23C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default |
+-----------------------------------+----------------------+----------------------+
|   1  Tesla M60             On    | 0000:86:00.0     Off |                  Off |
| N/A   29C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default |
+-----------------------------------+----------------------+----------------------+
|   2  Tesla P40             On    | 0000:87:00.0     Off |                  Off |
| N/A   21C    P8    18W / 250W |     53MiB / 24575MiB |      0%      Default |
+-----------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------------+
| Processes:                                                                        |
|  GPU       PID  Type  Process name                                  GPU Memory |
|                                                                     Usage      |
|===================================================================================|
|  No running processes found                                                       |
+-----------------------------------------------------------------------------------+
#
```

If `nvidia-smi` fails to run or doesn't produce the expected output for all the NVIDIA GPUs in your system, see Troubleshooting for troubleshooting steps.

2.7. 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.

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.

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, to ensure that the automation level of the

cluster supports VMs configured with NVIDIA vGPU, you must set the automation level to **Partially Automated** or **Manual**.

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

2.7.1. Installing the NVIDIA Virtual GPU Manager on VMware vSphere

To install the NVIDIA Virtual GPU Manager 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 software components for the NVIDIA Virtual GPU Manager have been extracted from the downloaded ZIP archive.

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

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

3. Install the NVIDIA vGPU hypervisor host driver and the NVIDIA GPU Management daemon from their software component files.
   a). Run the `esxcli` command to install the NVIDIA vGPU hypervisor host driver from its software component file.

```
$ esxcli software vib install -d /vmfs/volumes/datastore/host-driver-component.zip
```

   b). Run the `esxcli` command to install the NVIDIA GPU Management daemon from its software component file.

```
$ esxcli software vib install -d /vmfs/volumes/datastore/gpu-management-daemon-component.zip
```

   datastore
   The name of the VMFS datastore to which you copied the software components.
   host-driver-component
   The name of the file that contains the NVIDIA vGPU hypervisor host driver 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 7.0.3, *host-driver-component* is **NVD-VMware-x86_64-525.125.03-1OEM.703.0.0.17630552-bundle-*build-number***.
   gpu-management-daemon-component
   The name of the file that contains the NVIDIA GPU Management daemon 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 7.0.3, *gpu-management-daemon-component* is **VMW-esx-7.0.2-nvd-gpu-mgmt-daemon-1.0-0.0.0001**.
4. Exit maintenance mode.

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

5. Reboot the ESXi host.

```
$ reboot
```

2.7.2. Updating the NVIDIA Virtual GPU Manager for VMware vSphere

Update the NVIDIA Virtual GPU Manager if you want to install a new version of NVIDIA Virtual GPU Manager on a system where an existing version is already installed.

To update the vGPU Manager VIB 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.

Note: Before proceeding with the vGPU Manager update, make sure that all VMs are powered off and the ESXi host is placed in maintenance mode. Refer to VMware's documentation on how to place an ESXi host in maintenance mode

1. Stop the NVIDIA GPU Management Daemon.

```
$ /etc/init.d/nvdGpuMgmtDaemon stop
```

2. Update the NVIDIA vGPU hypervisor host driver and the NVIDIA GPU Management daemon.
   a). Run the `esxcli` command to update the NVIDIA vGPU hypervisor host driver from its software component file.

```
$ esxcli software vib update -d /vmfs/volumes/datastore/host-driver-component.zip
```

   b). Run the `esxcli` command to update the NVIDIA GPU Management daemon from its software component file.

```
$ esxcli software vib update -d /vmfs/volumes/datastore/gpu-management-daemon-component.zip
```

   datastore
   The name of the VMFS datastore to which you copied the software components.
   host-driver-component
   The name of the file that contains the NVIDIA vGPU hypervisor host driver 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 7.0.3, *host-driver-component* is **`NVD-VMware-x86_64-525.125.03-1OEM.703.0.0.17630552-bundle-build-number`**.
   gpu-management-daemon-component
   The name of the file that contains the NVIDIA GPU Management daemon 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 7.0.3, *gpu-management-daemon-component* is **`VMW-esx-7.0.2-nvd-gpu-mgmt-daemon-1.0-0.0.0001`**.
3. Reboot the ESXi host and remove it from maintenance mode.

2.7.3. 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 GPU Management daemon has started.

```
$ /etc/init.d/nvdGpuMgmtDaemon status
```

4. 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 Mar 22 17:56:22 2024
+-----------------------------------------------------------+
| NVIDIA-SMI 550.54.16    Driver Version: 550.54.16         |
|-----------------------------------+----------------------+----------------------+
| 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.

2.7.4. Managing the NVIDIA GPU Management Daemon for VMware vSphere

The NVIDIA GPU Management Daemon for VMware vSphere is a service that is controlled through scripts in the `/etc/init.d` directory. You can use these scripts to start the daemon, stop the daemon, and get its status.

To start the NVIDIA GPU Management Daemon, enter the following command:

```
$ /etc/init.d/nvdGpuMgmtDaemon start
```

To stop the NVIDIA GPU Management Daemon, enter the following command:

```
$ /etc/init.d/nvdGpuMgmtDaemon stop
```

To get the status of the NVIDIA GPU Management Daemon, enter the following command:

```
$ /etc/init.d/nvdGpuMgmtDaemon status
```

2.7.5. Configuring VMware vMotion with vGPU for VMware vSphere

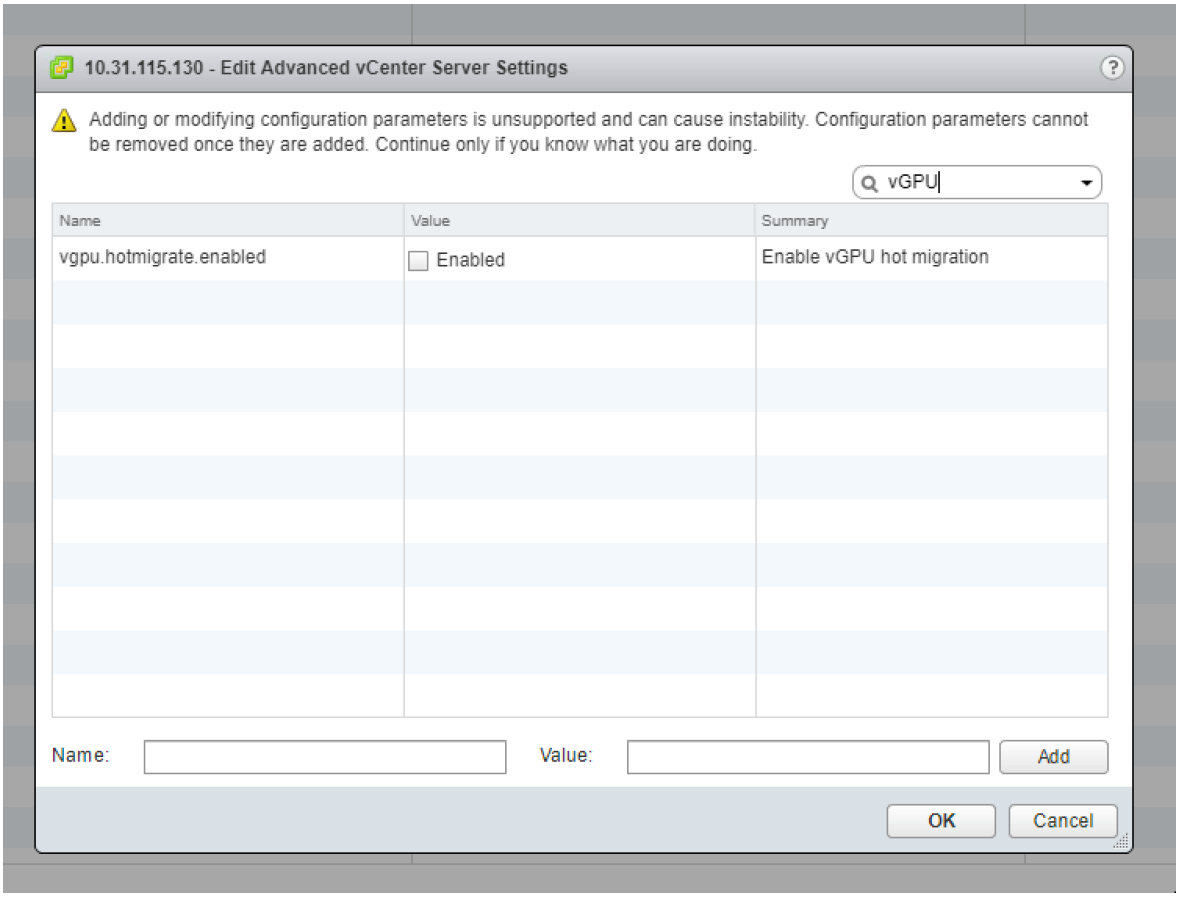
NVIDIA AI Enterprise supports vGPU migration, which includes VMware vMotion and suspend-resume, for VMs that are configured with vGPU. To enable VMware vMotion with vGPU, an advanced **vCenter Server** setting must be enabled. However, suspend-resume for VMs that are configured with vGPU is enabled by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

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**.



2.7.6. Changing the Default Graphics Type in VMware vSphere

After the vGPU Manager VIB for VMware vSphere VIB is installed, the default graphics type is Shared. To enable vGPU support for VMs in VMware vSphere, 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.
```

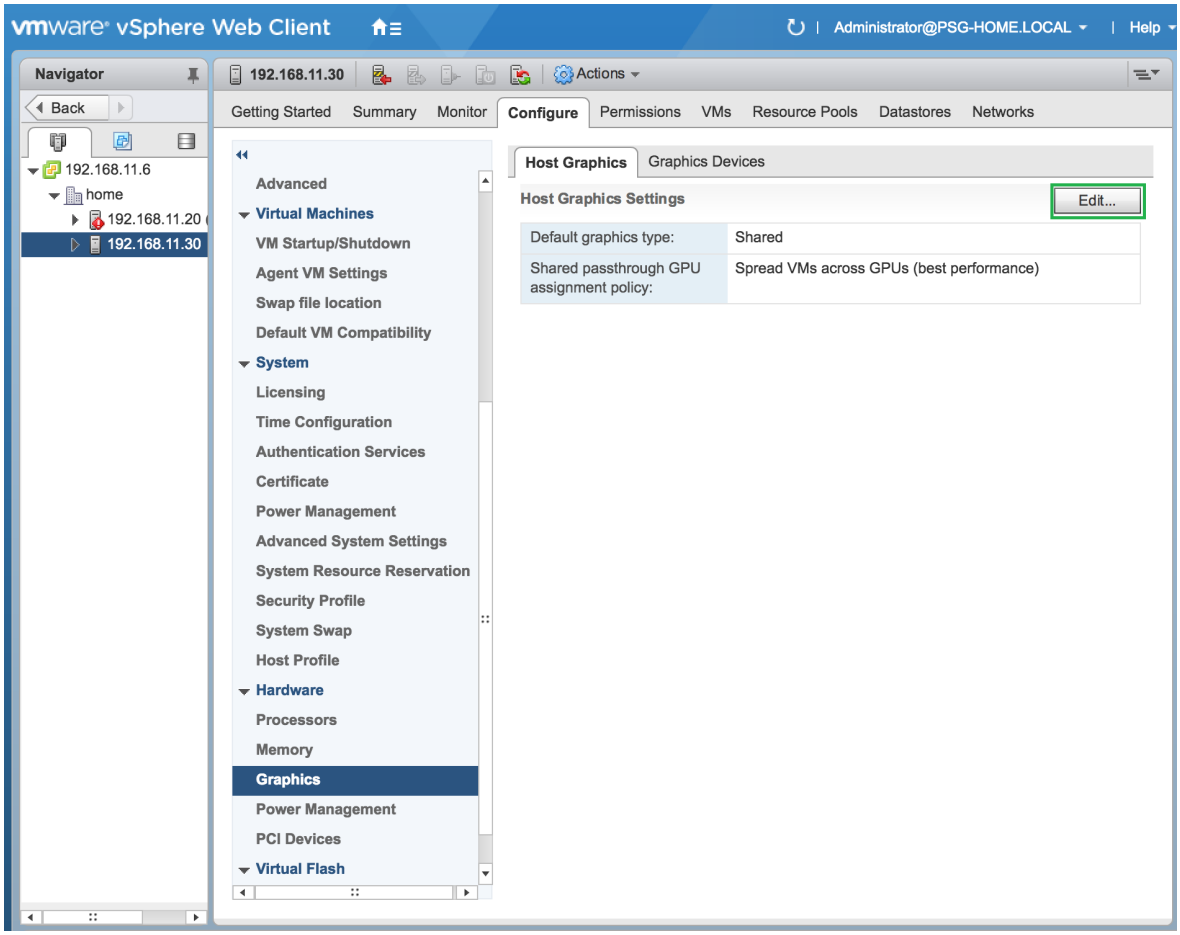
Note: Change the default graphics type **before** configuring vGPU. Output from the VM console in the VMware vSphere Web Client is not available for VMs that are running vGPU.

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.

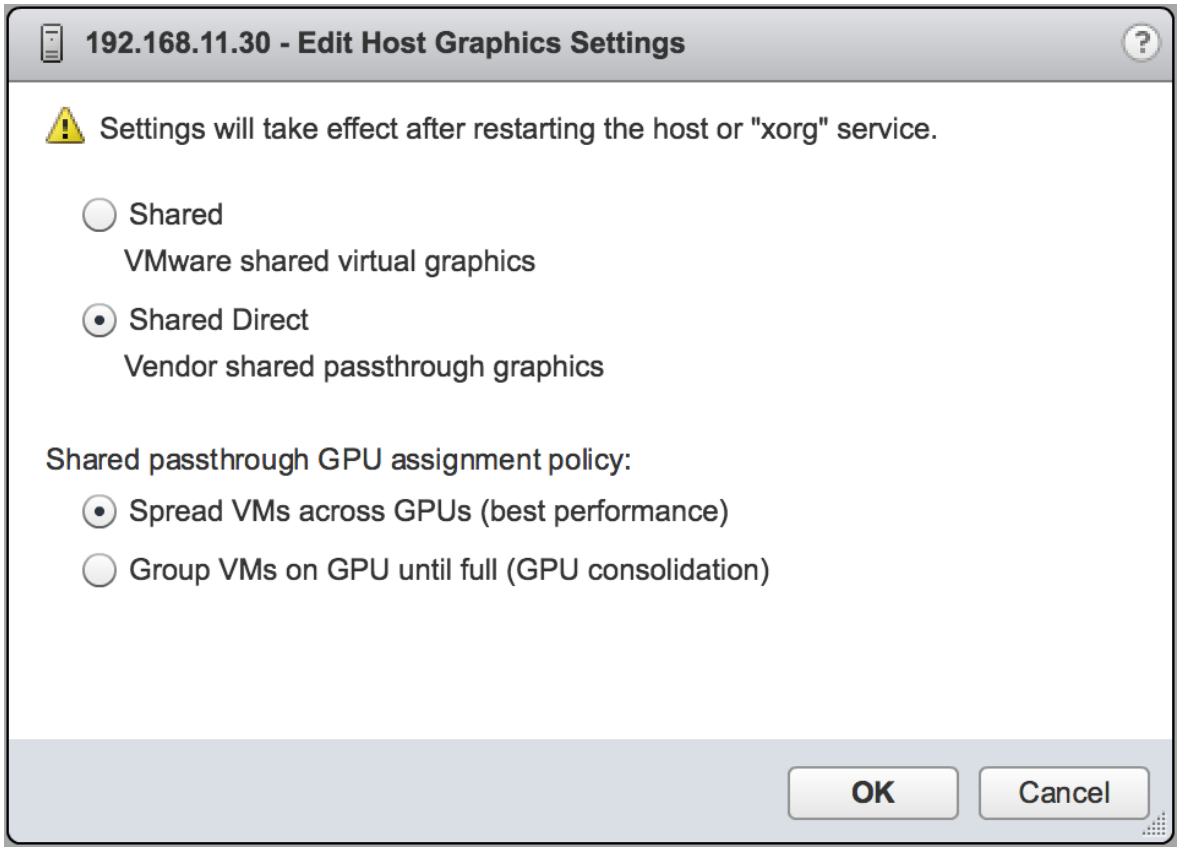
3. From the menu, choose **Graphics** and then click the **Host Graphics** tab.
4. On the **Host Graphics** tab, click **Edit**.

Figure 5. Shared default graphics type



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

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.

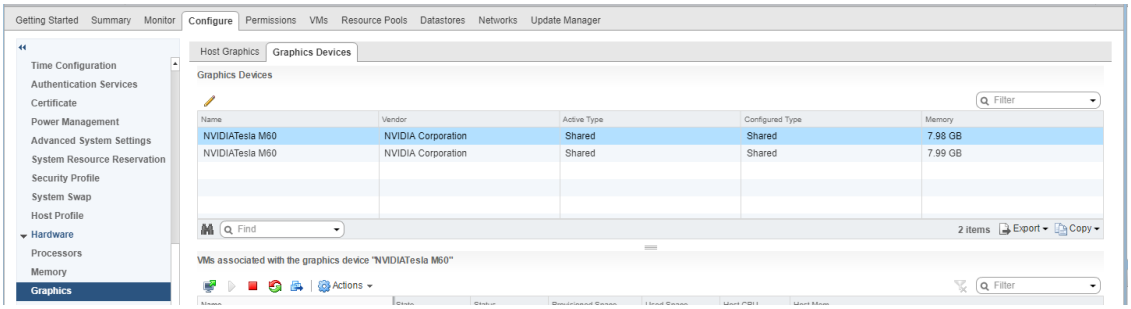
After you click OK, the default graphics type changes to Shared Direct.

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:

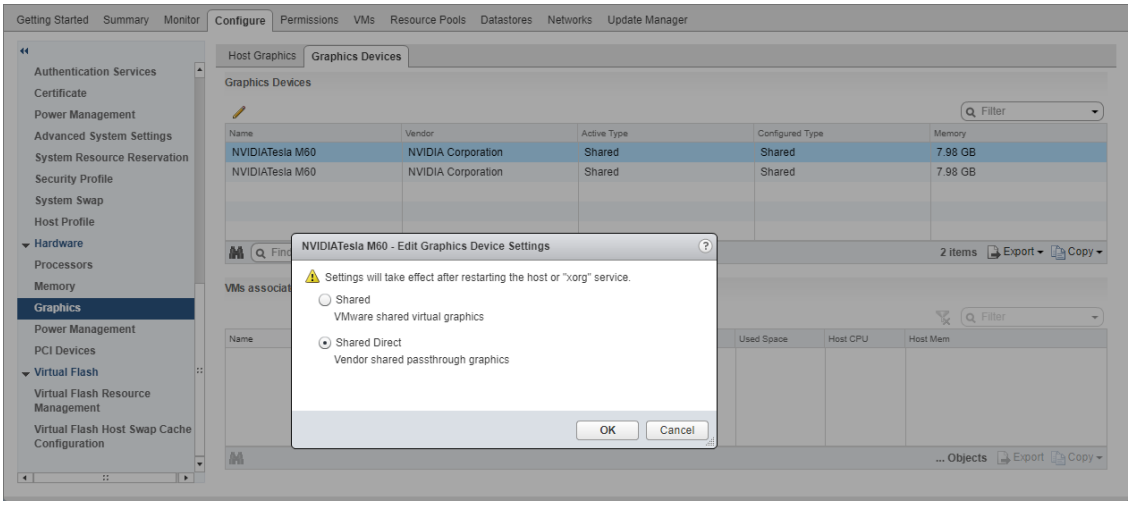
   a). On the **Graphics Devices** tab, select the physical GPU and click the **Edit icon**.

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 `nv-hostengine` on the ESXi host.

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

   a). Stop `nv-hostengine`.

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

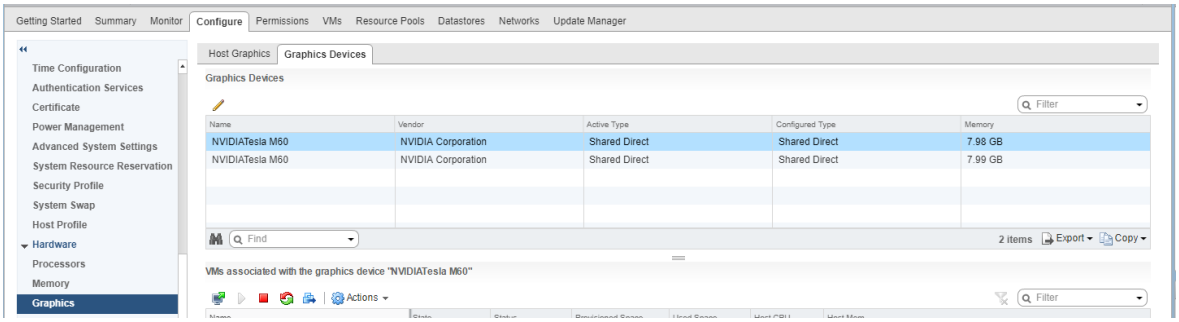
   b). Wait for 1 second to allow `nv-hostengine` to stop.

   c). Start `nv-hostengine`.

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

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

2.7.7. 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.

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.

How to configure a vSphere VM with a vGPU depends on your VMware vSphere version as explained in the following topics:

- Configuring a vSphere 8 VM with NVIDIA vGPU
- Configuring a vSphere 7 VM with NVIDIA vGPU

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 Installing and Licensing NVIDIA AI Enterprise Software Components.

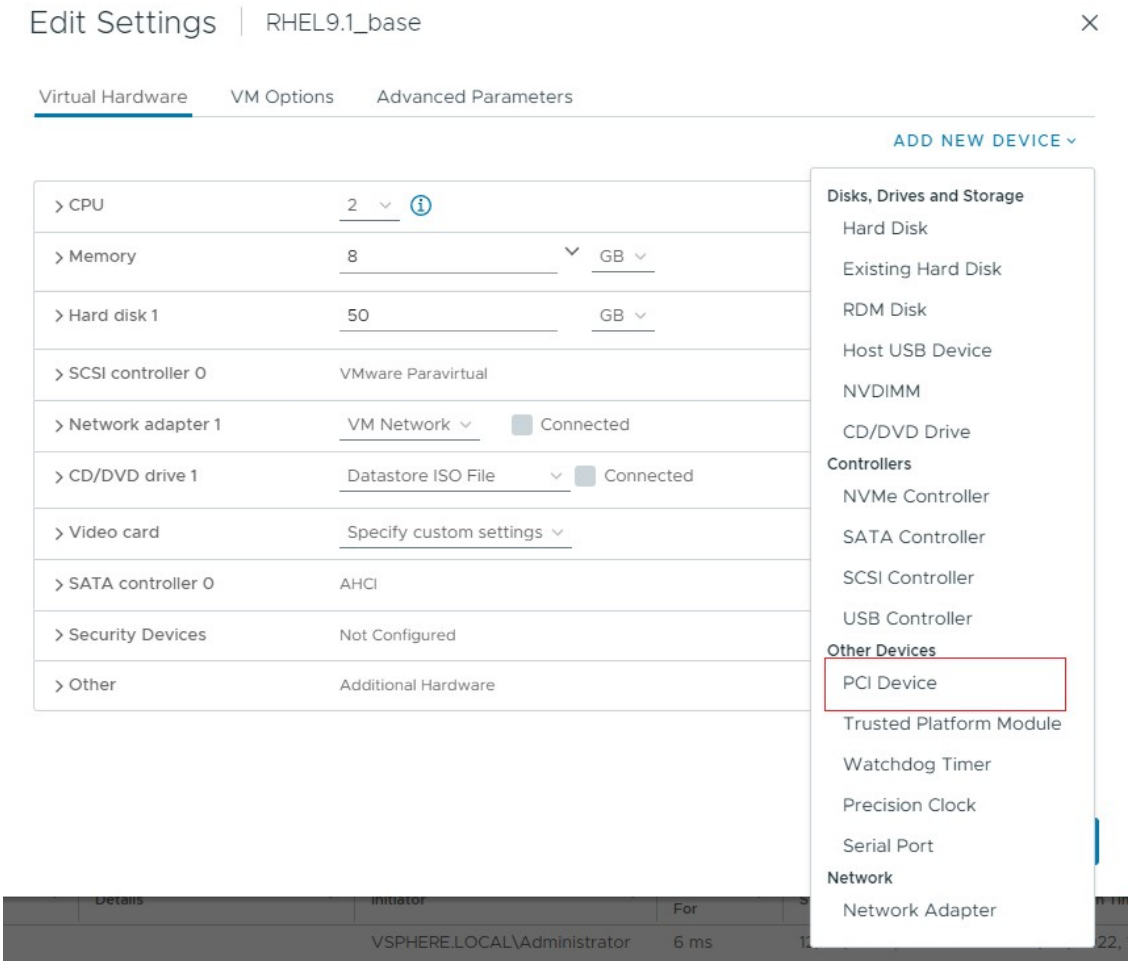
2.7.7.1. Configuring a vSphere 8 VM with NVIDIA vGPU

1. Open the vCenter Web UI.
2. In the vCenter Web UI, right-click the VM and choose **Edit Settings**.
3. In the **Edit Settings** window that opens, configure the vGPUs that you want to add to the VM.

   Add each vGPU that you want to add to the VM as follows:

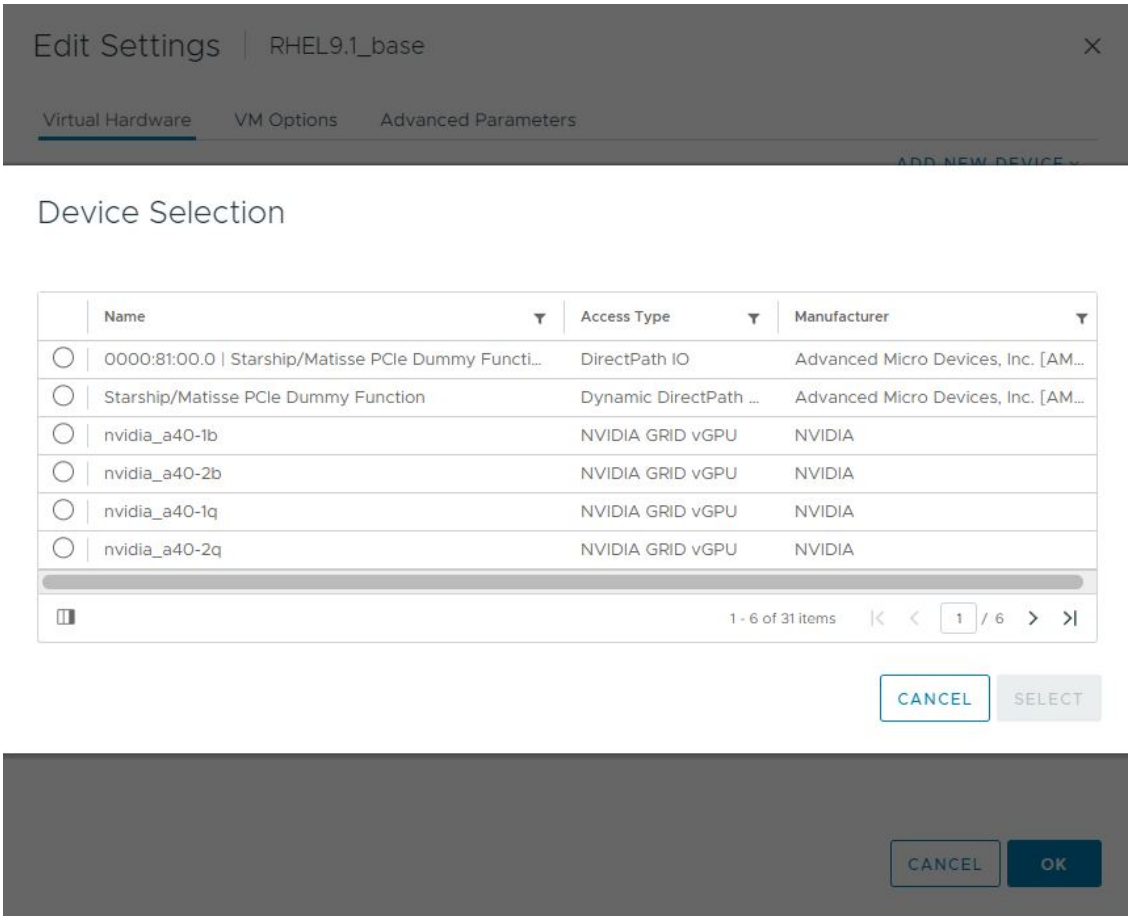
   a). From the **ADD NEW DEVICE** menu, choose **PCI Device**.

Figure 10. Command for Adding a PCI Device



b). In the **Device Selection** window that opens, select the type of vGPU you want to configure and click **SELECT**.

Figure 11. VM Device Selections for vGPU



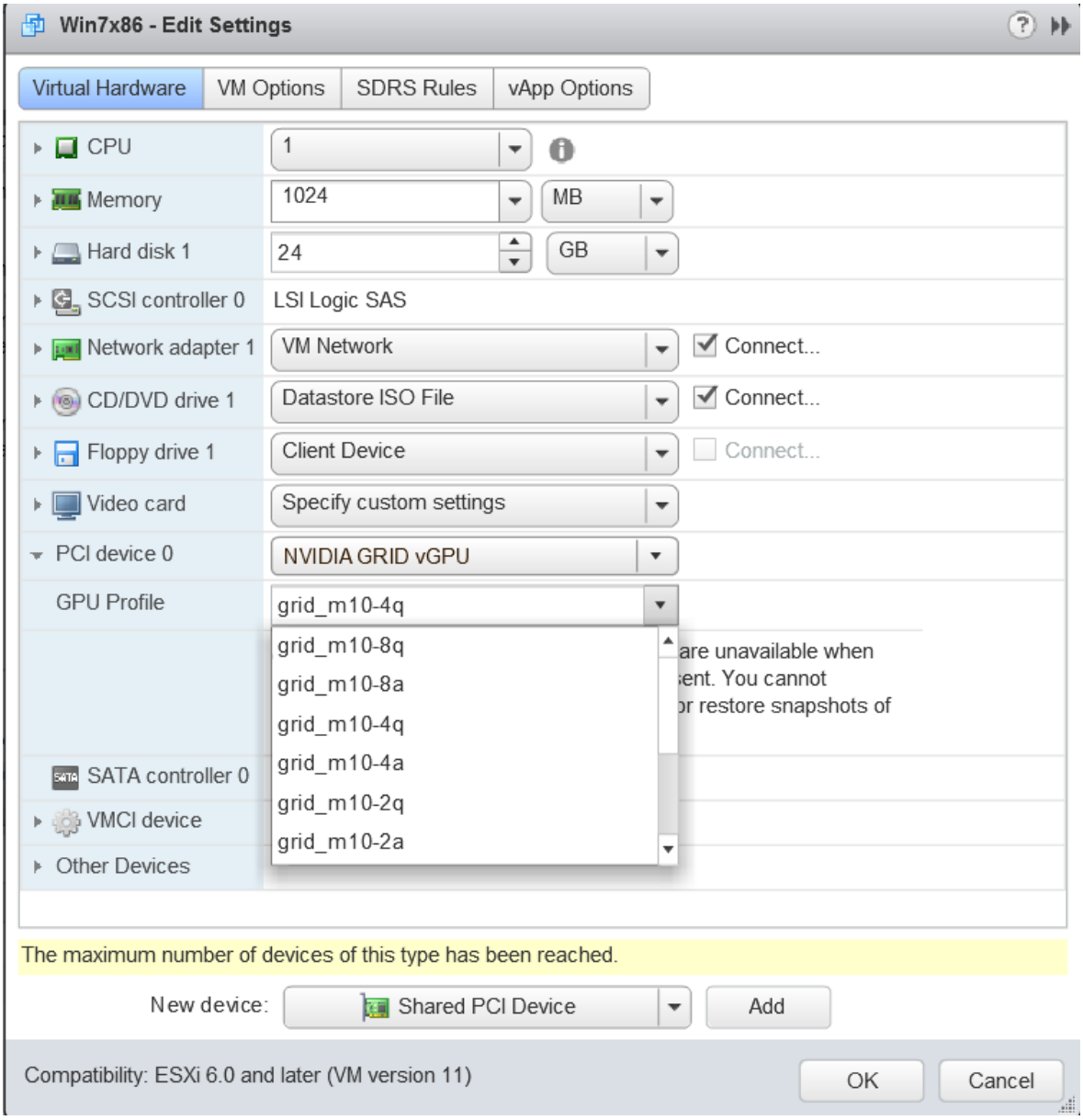
4. Back in the **Edit Settings** window, click **OK**.

2.7.7.2. Configuring a vSphere 7 VM with NVIDIA vGPU

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

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 12. VM settings for vGPU



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

2.7.8. 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.

2.8. Configuring the vGPU Manager for a Linux with KVM Hypervisor

NVIDIA AI Enterprise supports the following Linux with KVM hypervisors: Red Hat Enterprise Linux with KVM and Ubuntu.

2.8.1. Getting the BDF and Domain of a GPU on a Linux with KVM Hypervisor

Sometimes when configuring a physical GPU for use with NVIDIA AI Enterprise, you must find out which directory in the `sysfs` file system represents the GPU. This directory is identified by the domain, bus, slot, and function of the GPU.

For more information about the directory in the `sysfs` file system that represents a physical GPU, see NVIDIA vGPU Information in the sysfs File System.

1. Obtain the PCI device bus/device/function (BDF) of the physical GPU.

   ```
   # lspci | grep NVIDIA
   ```

   The NVIDIA GPUs listed in this example have the PCI device BDFs `06:00.0` and `07:00.0`.

   ```
   # lspci | grep NVIDIA
   06:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M10] (rev
    a1)
   07:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M10] (rev
    a1)
   ```

2. Obtain the full identifier of the GPU from its PCI device BDF.

   ```
   # virsh nodedev-list --cap pci| grep transformed-bdf
   ```

   transformed-bdf
   The PCI device BDF of the GPU with the colon and the period replaced with underscores, for example, `06_00_0`.

   This example obtains the full identifier of the GPU with the PCI device BDF `06:00.0`.

   ```
   # virsh nodedev-list --cap pci| grep 06_00_0
   pci_0000_06_00_0
   ```

3. Obtain the domain, bus, slot, and function of the GPU from the full identifier of the GPU.

   ```
   virsh nodedev-dumpxml full-identifier| egrep 'domain|bus|slot|function'
   ```

   full-identifier
   The full identifier of the GPU that you obtained in the previous step, for example, `pci_0000_06_00_0`.

   This example obtains the domain, bus, slot, and function of the GPU with the PCI device BDF `06:00.0`.

   ```
   # virsh nodedev-dumpxml pci_0000_06_00_0| egrep 'domain|bus|slot|function'
     <domain>0x0000</domain>
     <bus>0x06</bus>
     <slot>0x00</slot>
     <function>0x0</function>
   ```

```
      <address domain='0x0000' bus='0x06' slot='0x00' function='0x0'/>
```

2.8.2. Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor

An NVIDIA vGPU that supports SR-IOV resides on a physical GPU that supports SR-IOV, such as a GPU based on the NVIDIA Ampere architecture. Before creating an NVIDIA vGPU on a GPUthat supports SR-IOV, you must enable the virtual functions of the GPU and obtain the domain, bus, slot, and function of the specific virtual function on which you want to create the vGPU.
Before performing this task, ensure that the GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

1. Enable the virtual functions for the physical GPU in the `sysfs` file system.

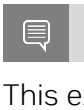
   

   Note: The virtual functions for the physical GPU in the `sysfs` file system are disabled after the hypervisor host is rebooted or if the driver is reloaded or upgraded.

   Use **only** the custom script `sriov-manage` provided by NVIDIA AI Enterprise for this purpose. Do **not** try to enable the virtual function for the GPU by any other means.

   ```
   # /usr/lib/nvidia/sriov-manage -e domain:bus:slot.function
   ```

   domain
   bus
   slot
   function
   The domain, bus, slot, and function of the GPU, without the `0x` prefix.

   Note: Only one `mdev` device file can be created on a virtual function.

   This example enables the virtual functions for the GPU with the domain `00`, bus `41`, slot `0000`, and function `0`.

   ```
   # /usr/lib/nvidia/sriov-manage -e 00:41:0000.0
   ```

2. Obtain the domain, bus, slot, and function of the available virtual functions on the GPU.

   ```
   # ls -l /sys/bus/pci/devices/domain\:bus\:slot.function/ | grep virtfn
   ```

   domain
   bus
   slot
   function
   The domain, bus, slot, and function of the GPU, without the `0x` prefix.

   This example shows the output of this command for a physical GPU with slot `00`, bus `41`, domain `0000`, and function `0`.

   ```
   # ls -l /sys/bus/pci/devices/0000:41:00.0/ | grep virtfn
   lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn0 -> ../0000:41:00.4
   lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn1 -> ../0000:41:00.5
   lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn10 -> ../0000:41:01.6
   ```

```
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn11 -> ../0000:41:01.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn12 -> ../0000:41:02.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn13 -> ../0000:41:02.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn14 -> ../0000:41:02.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn15 -> ../0000:41:02.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn16 -> ../0000:41:02.4
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn17 -> ../0000:41:02.5
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn18 -> ../0000:41:02.6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn19 -> ../0000:41:02.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn2 -> ../0000:41:00.6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn20 -> ../0000:41:03.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn21 -> ../0000:41:03.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn22 -> ../0000:41:03.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn23 -> ../0000:41:03.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn24 -> ../0000:41:03.4
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn25 -> ../0000:41:03.5
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn26 -> ../0000:41:03.6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn27 -> ../0000:41:03.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn28 -> ../0000:41:04.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn29 -> ../0000:41:04.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn3 -> ../0000:41:00.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn30 -> ../0000:41:04.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn31 -> ../0000:41:04.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn4 -> ../0000:41:01.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn5 -> ../0000:41:01.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn6 -> ../0000:41:01.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn7 -> ../0000:41:01.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn8 -> ../0000:41:01.4
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn9 -> ../0000:41:01.5
```

3. Choose the available virtual function on which you want to create the vGPU and note its domain, bus, slot, and function.

2.8.3. Creating an NVIDIA vGPU on a Linux with KVM Hypervisor

For each vGPU that you want to create, perform this task in a Linux command shell on the a Linux with KVM hypervisor host.

Before you begin, ensure that you have the domain, bus, slot, and function of the GPU on which you are creating the vGPU. For instructions, see Getting the BDF and Domain of a GPU on a Linux with KVM Hypervisor.

How to create an NVIDIA vGPU on a Linux with KVM hypervisor depends on the following factors:

- Whether the NVIDIA vGPU supports single root I/O virtualization (SR-IOV)
- Whether the hypervisor uses a vendor-specific Virtual Function I/O (VFIO) framework for an NVIDIA vGPU that supports SR-IOV

> **Note:** A hypervisor that uses a vendor-specific VFIO framework uses it **only** for an NVIDIA vGPU that supports SR-IOV. The hypervisor still uses the mediated VFIO `mdev` driver framework for a legacy NVIDIA vGPU.

A vendor-specific VFIO framework does not support the mediated VFIO `mdev` driver framework.

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in Ubuntu release 24.04.

To determine which instructions to follow for the NVIDIA vGPU that you are creating, refer to the following table.

NVIDIA vGPU Type	VFIO Framework	Instructions
Legacy: SR-IOV not supported	`mdev`	Creating a Legacy NVIDIA vGPU on a Linux with KVM Hypervisor
SR-IOV supported	`mdev`	Creating an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor
SR-IOV supported	Vendor specific	#unique_53

2.8.3.1. Creating a Legacy NVIDIA vGPU on a Linux with KVM Hypervisor

A legacy NVIDIA vGPU does not support SR-IOV.

1. Change to the `mdev_supported_types` directory for the physical GPU.

   ```
   # cd /sys/class/mdev_bus/domain\:bus\:slot.function/mdev_supported_types/
   ```

   domain
   bus
   slot
   function
   The domain, bus, slot, and function of the GPU, without the `0x` prefix.

   This example changes to the `mdev_supported_types` directory for the GPU with the domain `0000` and PCI device BDF `06:00.0`.

   ```
   # cd /sys/bus/pci/devices/0000\:06\:00.0/mdev_supported_types/
   ```

2. Find out which subdirectory of `mdev_supported_types` contains registration information for the vGPU type that you want to create.

   ```
   # grep -l "vgpu-type" nvidia-*/name
   ```

   vgpu-type
   The vGPU type, for example, `M10-2Q`.

   This example shows that the registration information for the M10-2Q vGPU type is contained in the `nvidia-41` subdirectory of `mdev_supported_types`.

   ```
   # grep -l "M10-2Q" nvidia-*/name
   nvidia-41/name
   ```

3. Confirm that you can create an instance of the vGPU type on the physical GPU.

   ```
   # cat subdirectory/available_instances
   ```

   subdirectory
   The subdirectory that you found in the previous step, for example, `nvidia-41`.

   The number of available instances must be at least 1. If the number is 0, either an instance of another vGPU type already exists on the physical GPU, or the maximum number of allowed instances has already been created.

This example shows that four more instances of the M10-2Q vGPU type can be created on the physical GPU.

```
# cat nvidia-41/available_instances
4
```

4. Generate a correctly formatted universally unique identifier (UUID) for the vGPU.

```
# uuidgen
aa618089-8b16-4d01-a136-25a0f3c73123
```

5. Write the UUID that you obtained in the previous step to the `create` file in the registration information directory for the vGPU type that you want to create.

```
# echo "uuid"> subdirectory/create
```

uuid
The UUID that you generated in the previous step, which will become the UUID of the vGPU that you want to create.

subdirectory
The registration information directory for the vGPU type that you want to create, for example, `nvidia-41`.

This example creates an instance of the M10-2Q vGPU type with the UUID `aa618089-8b16-4d01-a136-25a0f3c73123`.

```
# echo "aa618089-8b16-4d01-a136-25a0f3c73123" > nvidia-41/create
```

An `mdev` device file for the vGPU is added to the parent physical device directory of the vGPU. The vGPU is identified by its UUID.

The `/sys/bus/mdev/devices/` directory contains a symbolic link to the `mdev` device file.

6. Make the `mdev` device file that you created to represent the vGPU persistent.

```
# mdevctl define --auto --uuid uuid
```

uuid
The UUID that you specified in the previous step for the vGPU that you are creating.

Note: Not all Linux with KVM hypervisor releases include the `mdevctl` command. If your release does not include the `mdevctl` command, you can use standard features of the operating system to automate the re-creation of this device file when the host is booted. For example, you can write a custom script that is executed when the host is rebooted.

7. Confirm that the vGPU was created.
   a). Confirm that the `/sys/bus/mdev/devices/` directory contains the `mdev` device file for the vGPU.

```
# ls -l /sys/bus/mdev/devices/
total 0
lrwxrwxrwx. 1 root root 0 Nov 24 13:33 aa618089-8b16-4d01-a136-25a0f3c73123 -
> ../../../devices/
pci0000:00/0000:00:03.0/0000:03:00.0/0000:04:09.0/0000:06:00.0/
aa618089-8b16-4d01-a136-25a0f3c73123
```

   b). If your release includes the `mdevctl` command, list the active mediated devices on the hypervisor host.

```
# mdevctl list
aa618089-8b16-4d01-a136-25a0f3c73123 0000:06:00.0 nvidia-41
```

2.8.3.2. Creating an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor

An NVIDIA vGPU that supports SR-IOV resides on a physical GPU that supports SR-IOV, such as a GPU based on the NVIDIA Ampere architecture.

Before performing this task, ensure that the virtual function on which you want to create the vGPU has been prepared as explained in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

If you want to support vGPUs with different amounts of frame buffer, also ensure that the GPU has been put into mixed-size mode as explained in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

1. Change to the `mdev_supported_types` directory for the virtual function on which you want to create the vGPU.

   ```
   # cd /sys/class/mdev_bus/domain\:bus\:vf-slot.v-function/mdev_supported_types/
   ```

   domain
   bus
   The domain and bus of the GPU, without the `0x` prefix.
   vf-slot
   v-function
   The slot and function of the virtual function that you noted in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

   This example changes to the `mdev_supported_types` directory for the first virtual function (`virtfn0`) for the GPU with the domain `0000` and bus `41`. The first virtual function (`virtfn0`) has slot `00` and function `4`.

   ```
   # cd /sys/class/mdev_bus/0000\:41\:00.4/mdev_supported_types
   ```

2. Find out which subdirectory of `mdev_supported_types` contains registration information for the vGPU type that you want to create.

   ```
   # grep -l "vgpu-type" nvidia-*/name
   ```

   vgpu-type
   The vGPU type, for example, `A40-2Q`.

   This example shows that the registration information for the A40-2Q vGPU type is contained in the `nvidia-558` subdirectory of `mdev_supported_types`.

   ```
   # grep -l "A40-2Q" nvidia-*/name
   nvidia-558/name
   ```

3. Confirm that you can create an instance of the vGPU type on the virtual function.

   ```
   # cat subdirectory/available_instances
   ```

   subdirectory
   The subdirectory that you found in the previous step, for example, `nvidia-558`.

   The number of available instances must be 1. If the number is 0, a vGPU has already been created on the virtual function. Only one instance of any vGPU type can be created on a virtual function.

This example shows that an instance of the A40-2Q vGPU type can be created on the virtual function.

```
# cat nvidia-558/available_instances
1
```

4. Generate a correctly formatted universally unique identifier (UUID) for the vGPU.

```
# uuidgen
aa618089-8b16-4d01-a136-25a0f3c73123
```

5. Write the UUID that you obtained in the previous step to the `create` file in the registration information directory for the vGPU type that you want to create.

```
# echo "uuid"> subdirectory/create
```

uuid
The UUID that you generated in the previous step, which will become the UUID of the vGPU that you want to create.

subdirectory
The registration information directory for the vGPU type that you want to create, for example, `nvidia-558`.

This example creates an instance of the A40-2Q vGPU type with the UUID `aa618089-8b16-4d01-a136-25a0f3c73123`.

```
# echo "aa618089-8b16-4d01-a136-25a0f3c73123" > nvidia-558/create
```

An `mdev` device file for the vGPU is added to the parent virtual function directory of the vGPU. The vGPU is identified by its UUID.

6. **Time-sliced vGPUs only:** Make the `mdev` device file that you created to represent the vGPU persistent.

```
# mdevctl define --auto --uuid uuid
```

uuid
The UUID that you specified in the previous step for the vGPU that you are creating.

Note:

- If you are using a GPU that supports SR-IOV, the `mdev` device file persists after a host reboot only if you enable the virtual functions for the GPU as explained in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor before rebooting any VM that is configured with a vGPU on the GPU.
- You **cannot** use the `mdevctl` command to make the `mdev` device file for a MIG-backed vGPU persistent. The `mdev` device file for a MIG-backed vGPU is not retained after the host is rebooted because MIG instances are no longer available.
- Not all Linux with KVM hypervisor releases include the `mdevctl` command. If your release does not include the `mdevctl` command, you can use standard features of the operating system to automate the re-creation of this device file when the host is booted. For example, you can write a custom script that is executed when the host is rebooted.

7. Confirm that the vGPU was created.

a). Confirm that the `/sys/bus/mdev/devices/` directory contains a symbolic link to the `mdev` device file.

```
# ls -l /sys/bus/mdev/devices/
total 0
```

```
lrwxrwxrwx. 1 root root 0 Jul 16 05:57 aa618089-8b16-4d01-a136-25a0f3c73123
 -> ../../../devices/pci0000:40/0000:40:01.1/0000:41:00.4/aa618089-8b16-4d01-
a136-25a0f3c73123
```

b). If your release includes the `mdevctl` command, list the active mediated devices on the hypervisor host.

```
# mdevctl list
aa618089-8b16-4d01-a136-25a0f3c73123 0000:06:00.0 nvidia-558
```

2.8.4. Adding One or More vGPUs to a Linux with KVM Hypervisor VM

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

Ensure that the following prerequisites are met:

- The VM to which you want to add the vGPUs is shut down.
- The vGPUs that you want to add have been created as explained in Creating an NVIDIA vGPU on a Linux with KVM Hypervisor.

You can add vGPUs to a Linux with KVM hypervisor VM by using any of the following tools:

- The `virsh` command
- The QEMU command line

After adding vGPUs to a Linux with KVM hypervisor VM, start the VM.

```
# virsh start vm-name
```

vm-name

The name of the VM that you added the vGPUs to.

After the VM has booted, install the NVIDIA AI Enterprise graphics driver as explained in Installing and Licensing NVIDIA AI Enterprise Components Natively.

2.8.4.1. Adding One or More vGPUs to a Linux with KVM Hypervisor VM by Using `virsh`

1. In `virsh`, open for editing the XML file of the VM that you want to add the vGPU to.

   ```
   # virsh edit vm-name
   ```

   vm-name

   The name of the VM to that you want to add the vGPUs to.

2. For each vGPU that you want to add to the VM, add a device entry in the form of an `address` element inside the `source` element to add the vGPU to the guest VM.

   The content of the device entry depends on whether the hypervisor uses a vendor-specific VFIO framework for an NVIDIA vGPU that supports SR-IOV.

   For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in Ubuntu release 24.04.

- For a hypervisor that uses the `mdev` **VFIO framework**, add a device entry that identifies the vGPU through its UUID as follows:

```
<device>
...
  <hostdev mode='subsystem' type='mdev' model='vfio-pci'>
    <source>
      <address uuid='uuid'/>
    </source>
  </hostdev>
</device>
```

uuid
The UUID that was assigned to the vGPU when the vGPU was created.

This example adds a device entry for the vGPU with the UUID `a618089-8b16-4d01-a136-25a0f3c73123`.

```
<device>
...
  <hostdev mode='subsystem' type='mdev' model='vfio-pci'>
    <source>
      <address uuid='a618089-8b16-4d01-a136-25a0f3c73123'/>
    </source>
  </hostdev>
</device>
```

This example adds device entries for two vGPUs with the following UUIDs:

- `c73f1fa6-489e-4834-9476-d70dabd98c40`
- `3b356d38-854e-48be-b376-00c72c7d119c`

```
<device>
...
  <hostdev mode='subsystem' type='mdev' model='vfio-pci'>
    <source>
      <address uuid='c73f1fa6-489e-4834-9476-d70dabd98c40'/>
    </source>
  </hostdev>
  <hostdev mode='subsystem' type='mdev' model='vfio-pci'>
    <source>
      <address uuid='3b356d38-854e-48be-b376-00c72c7d119c'/>
    </source>
  </hostdev>
</device>
```

- For a hypervisor that uses a **vendor-specific VFIO framework**, add a device entry that identifies the vGPU through the virtual function on which the vGPU is created as follows:

```
<hostdev mode='subsystem' type='pci' managed='no'>
  <source>
    <address domain='domain' bus='bus' slot='vf-slot' function='v-function'/>
  </source>
</hostdev>
```

domain
bus
The domain and bus of the GPU, including the `0x` prefix.

vf-slot
v-function
The slot and function of the virtual function that you noted in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

Note: A vGPU is supported only in unmanaged `libvirt` mode. Therefore, ensure that in the `hostdev` element, the `managed` attribute is set to `no`.

This example adds a device entry for the vGPU that is created on the virtual function `0000:3d:00.4`.

```
<device>
...
  <hostdev mode='subsystem' type='pci' managed='no'>
    <source>
      <address domain='0x0000' bus='0x3d' slot='0x00' function='0x4'/>
    </source>
  </hostdev>
</device>
```

This example adds device entries for two vGPUs that are created on the following virtual functions:

- `0000:3d:00.4`
- `0000:3d:00.5`

```
<device>
...
  <hostdev mode='subsystem' type='pci' managed='no'>
    <source>
      <address domain='0x0000' bus='0x3d' slot='0x00' function='0x4'/>
    </source>
  </hostdev>
  <hostdev mode='subsystem' type='pci' managed='no'>
    <source>
      <address domain='0x0000' bus='0x3d' slot='0x00' function='0x5'/>
    </source>
  </hostdev>
</device>
```

3. **Optional:** Add a `video` element that contains a `model` element in which the `type` attribute is set to `none`.

```
<video>
<model type='none'/>
</video>
```

Adding this `video` element prevents the default video device that `libvirt` adds from being loaded into the VM. If you don't add this `video` element, you must configure the Xorg server or your remoting solution to load only the vGPU devices you added and not the default video device.

2.8.4.2. Adding One or More vGPUs to a Linux with KVM Hypervisor VM by Using the QEMU Command Line

This task involves adding options to the QEMU command line that identify the vGPUs that you want to add and the VM to which you want to add them.

1. For each vGPU that you want to add to the VM, add one `-device` option that identifies the vGPU.

   The format of each `-device` option depends on whether the hypervisor uses a vendor-specific VFIO framework for an NVIDIA vGPU that supports SR-IOV.

   For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in Ubuntu release 24.04.

   - For each vGPU on a hypervisor that uses the `mdev` **VFIO framework**, add a `-device` option that identifies the vGPU through its UUID.

     ```
     -device vfio-pci,sysfsdev=/sys/bus/mdev/devices/vgpu-uuid
     ```

     vgpu-uuid
     The UUID that was assigned to the vGPU when the vGPU was created.
   - For each vGPU on a hypervisor that uses a **vendor-specific VFIO framework**, add a `-device` option that identifies the vGPU through the virtual function on which the vGPU is created.

     ```
     -device vfio-pci,sysfsdev=/sys/bus/pci/devices/domain\:bus\:vf-slot.v-function/
     ```

     domain
     bus
     The domain and bus of the GPU, without the `0x` prefix.
     vf-slot
     v-function
     The slot and function of the virtual function that you noted in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

2. Add a `-uuid` option to specify the VM to which you want to add the vGPUs.

   ```
   -uuid vm-uuid
   ```

   vm-uuid
   The UUID that was assigned to the VM when the VM was created.

Adding One vGPU to a VM on a Hypervisor that Uses the `mdev` VFIO Framework

This example adds the vGPU with the UUID `aa618089-8b16-4d01-a136-25a0f3c73123` to the VM with the UUID `ebb10a6e-7ac9-49aa-af92-f56bb8c65893`.

```
-device vfio-pci,sysfsdev=/sys/bus/mdev/devices/aa618089-8b16-4d01-a136-25a0f3c73123
 \
-uuid ebb10a6e-7ac9-49aa-af92-f56bb8c65893
```

Adding Two vGPUs to a VM on a Hypervisor that Uses the `mdev` VFIO Framework

This example adds device entries for two vGPUs with the following UUIDs:

- `676428a0-2445-499f-9bfd-65cd4a9bd18f`
- `6c5954b8-5bc1-4769-b820-8099fe50aaba`

The entries are added to the VM with the UUID `ec5e8ee0-657c-4db6-8775-da70e332c67e`.

```
-device vfio-pci,sysfsdev=/sys/bus/mdev/devices/676428a0-2445-499f-9bfd-65cd4a9bd18f
 \
-device vfio-pci,sysfsdev=/sys/bus/mdev/devices/6c5954b8-5bc1-4769-b820-8099fe50aaba
 \
-uuid ec5e8ee0-657c-4db6-8775-da70e332c67e
```

Adding One vGPU to a VM on a Hypervisor that Uses a Vendor-Specific VFIO Framework

This example adds the vGPU that is created on the virtual function `0000:3d:00.4` to the VM with the UUID `ebb10a6e-7ac9-49aa-af92-f56bb8c65893`.

```
-device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000\:3d\:00.4 \
-uuid ebb10a6e-7ac9-49aa-af92-f56bb8c65893
```

Adding Two vGPUs to a VM on a Hypervisor that Uses a Vendor-Specific VFIO Framework

This example adds device entries for two vGPUs that are created on the following virtual functions:

- `0000:3d:00.4`
- `0000:3d:00.5`

The entries are added to the VM with the UUID `ec5e8ee0-657c-4db6-8775-da70e332c67e`.

```
-device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000\:3d\:00.4 \
-device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000\:3d\:00.5 \
-uuid ec5e8ee0-657c-4db6-8775-da70e332c67e
```

2.8.5. Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor

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.

For each vGPU for which you want to set plugin parameters, perform this task in a Linux command shell on the Linux with KVM hypervisor host.

1. Change to the directory in the `sysfs` file system that contains the `vgpu_params` file for the vGPU for which you want to set vGPU plugin parameters.

   The directory depends on whether the hypervisor uses a vendor-specific VFIO framework for an NVIDIA vGPU that supports SR-IOV.

   For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in Ubuntu release 24.04.

   - For a hypervisor that uses the `mdev` **VFIO framework**, change to the `nvidia` subdirectory of the `mdev` device directory that represents the vGPU.

     ```
     # cd /sys/bus/mdev/devices/uuid/nvidia
     ```

     uuid
     The UUID of the vGPU, for example, `aa618089-8b16-4d01-a136-25a0f3c73123`.

   - For a hypervisor that uses a **vendor-specific VFIO framework**, change to the directory in the `sysfs` file system that contains the files for vGPU management on the virtual function on which the vGPU was created.

     ```
     # cd /sys/bus/pci/devices/domain\:bus\:vf-slot.v-function/nvidia
     ```

     domain
     bus
     The domain and bus of the GPU, without the `0x` prefix.
     vf-slot
     v-function
     The slot and function of the virtual function that you noted in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

     This example changes to the `nvidia` directory for the first virtual function (`virtfn0`) for the GPU with the domain `0000` and bus `3d`. The first virtual function (`virtfn0`) has slot `00` and function `4`.

     ```
     # cd /sys/bus/pci/devices/0000\:3d\:00.4/nvidia
     ```

2. Write the plugin parameters that you want to set to the `vgpu_params` file in the directory that you changed to in the previous step.

   ```
   # echo "plugin-config-params" > vgpu_params
   ```

   plugin-config-params
   A comma-separated list of parameter-value pairs, where each pair is of the form ***parameter-name=value***.

   This example disables frame rate limiting and console VNC for a vGPU.

   ```
   # echo "frame_rate_limiter=0, disable_vnc=1" > vgpu_params
   ```

   This example enables unified memory for a vGPU.

   ```
   # echo "enable_uvm=1"  > vgpu_params
   ```

   This example enables NVIDIA CUDA Toolkit debuggers for a vGPU.

   ```
   # echo "enable_debugging=1"  > vgpu_params
   ```

   This example enables NVIDIA CUDA Toolkit profilers for a vGPU.

   ```
   # echo "enable_profiling=1"  > vgpu_params
   ```

To clear any vGPU plugin parameters that were set previously, write a space to the `vgpu_params` file for the vGPU.

```
# echo " " > vgpu_params
```

2.8.6. Deleting a vGPU on a Linux with KVM Hypervisor

How to delete a vGPU on a Linux with KVM hypervisor depends on whether the hypervisor uses a vendor-specific VFIO framework for an NVIDIA vGPU that supports SR-IOV.

Note: A hypervisor that uses a vendor-specific VFIO framework uses it **only** for an NVIDIA vGPU that supports SR-IOV. The hypervisor still uses the mediated VFIO `mdev` driver framework for a legacy NVIDIA vGPU.

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in Ubuntu release 24.04.

To determine which instructions to follow for the NVIDIA vGPU that you are deleting, refer to the following table.

NVIDIA vGPU Type	VFIO Framework	Instructions
Legacy: SR-IOV not supported	`mdev`	#unique_58
SR-IOV supported	`mdev`	
SR-IOV supported	Vendor specific	#unique_59

2.8.7. NVIDIA vGPU Information in the `sysfs` File System

Information about the NVIDIA vGPU types supported by each physical GPU in a Linux with KVM hypervisor host is stored in the `sysfs` file system.

How NVIDIA vGPU information is stored in the `sysfs` file system depends on whether the hypervisor uses a vendor-specific VFIO framework for an NVIDIA vGPU that supports SR-IOV.

Note: A hypervisor that uses a vendor-specific VFIO framework for an NVIDIA vGPU that supports SR-IOV uses the `mdev` VFIO framework for a legacy NVIDIA vGPU.

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in Ubuntu release 24.04.

For more detailed information about how NVIDIA vGPU information is stored in the `sysfs` file system, refer to the following topics:

- #unique_60
- #unique_61

2.9. Putting a GPU Into Mixed-Size Mode

By default, a GPU supports only vGPUs with the same amount of frame buffer and, therefore, is in equal-size mode. To support vGPUs with different amounts of frame buffer, the GPU must be put into mixed-size mode. When a GPU is in mixed-size mode, the maximum number of some types of vGPU allowed on a GPU is less than when the GPU is in equal-size mode.

Note:

- A GPU in mixed-size mode reverts to its default mode if the hypervisor host is rebooted, the NVIDIA Virtual GPU Manager is reloaded, or the GPU is reset.
- When a GPU is in mixed-size mode, only the best effort and equal share schedulers are supported. The fixed share scheduler is **not** supported.

Before performing this task, ensure that no vGPUs are running on the GPU and that the GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

If you are using a GPU that supports SR-IOV on a Linux with KVM hypervisor, also ensure that the virtual functions for the physical GPU in the `sysfs` file system are enabled as explained in Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor.

1. Use `nvidia-smi` to list the status of all physical GPUs, and check that heterogeneous time-sliced vGPU sizes are noted as supported.

```
# nvidia-smi -q
...
Attached GPUs                         : 1
GPU 00000000:41:00.0
...
        Heterogeneous Time-Slice Sizes    : Supported
...
```

2. Put each GPU that you want to support vGPUs with different amounts of frame buffer into mixed-size mode.

```
# nvidia-smi vgpu -i id -shm 1
```

id

The index of the GPU as reported by `nvidia-smi`.

This example puts the GPU with index `00000000:41:00.0` into mixed-size mode.

```
# nvidia-smi vgpu -i 0 -shm 1
Enabled vGPU heterogeneous mode for GPU 00000000:41:00.0
```

3. Confirm that the GPU is now in mixed-size mode by using `nvidia-smi` to check that vGPU heterogeneous mode is enabled.

```
# nvidia-smi -q
        ...
        vGPU Heterogeneous Mode           : Enabled
        ...
```

2.10. Placing a vGPU on a Physical GPU in Mixed-Size Mode

By default, the Virtual GPU Manager determines where a vGPU is placed on a GPU. To fit as many vGPUs as possible on the GPU, you can control the placement of vGPUs on a GPU in mixed-size mode. By controlling the placement of vGPUs on the GPU, you can ensure that no gaps that cannot be occupied by a vGPU are left in the placement region on the GPU.

The vGPU placements that a GPU in mixed-size mode supports depend on the total amount of frame buffer that the GPU has. For details, refer to vGPU Placements for GPUs in Mixed-Size Mode.

Note: This task is optional. If you want the Virtual GPU Manager to determine where a vGPU is placed on a GPU, omit this task.

Before performing this task, ensure that following prerequisites are met:

- The GPU has been put into mixed-size mode as explained in Putting a GPU Into Mixed-Size Mode.
- The vGPU that you want to place on the physical GPU has been created as explained in Creating an NVIDIA vGPU on a Linux with KVM Hypervisor.

Perform this task in a command shell on the hypervisor host.

1. Use `nvidia-smi` to list the placement size and available placement IDs for the type of the vGPU.

```
# nvidia-smi vgpu -c -v
...
    vGPU Type ID                          : 0x392
        Name                              : NVIDIA L4-6Q
...
        Placement Size                    : 6
        Creatable Placement IDs           : 6 18
...
```

Note:

Some supported placement IDs for the vGPU type might be unavailable because they are already in use by another vGPU. To list the placement size and all supported placement IDs for the type of the vGPU, run the following command:

```
# nvidia-smi vgpu -s -v
...
    vGPU Type ID                          : 0x392
        Name                              : NVIDIA L4-6Q
```

```
...
        Placement Size                      : 6
        Supported Placement IDs             : 0 6 12 18
...
```

The number of supported placement IDs is the maximum number of vGPUs of the type that are allowed on the GPU in mixed-size mode.

2. Set the `vgpu-placement-id` vGPU plugin parameter for the vGPU to the placement ID that you want.

   For a Linux with KVM hypervisor, write the parameter to the `vgpu_params` file in the `nvidia` subdirectory of the `mdev` device directory that represents the vGPU.

   ```
   # echo "vgpu-placement-id=placement-id" > /sys/bus/mdev/devices/uuid/nvidia/vgpu_params
   ```

   placement-id
   The placement ID that you want to set for the vGPU.

   uuid
   The UUID of the vGPU, for example, `aa618089-8b16-4d01-a136-25a0f3c73123`.

   This example sets the placement ID for the vGPU that has the UUID `aa618089-8b16-4d01-a136-25a0f3c73123` to 6.

   ```
   # echo "vgpu-placement-id=6" > \
   /sys/bus/mdev/devices/aa618089-8b16-4d01-a136-25a0f3c73123/nvidia/vgpu_params
   ```

When the VM to which the vGPU is assigned is rebooted, the Virtual GPU Manager validates the placement ID that you assigned to the vGPU. If the placement ID is invalid or unavailable, the VM fails to boot.

After the VM to which the vGPU is assigned has been rebooted, you can confirm that the vGPU has been assigned the correct placement ID.

```
# nvidia-smi vgpu -q
GPU 00000000:41:00.0
    Active vGPUs                        : 1
    vGPU ID                             : 3251719533
        VM ID                           : 2150987
        ...
        Placement ID                    : 6
        ...
```

2.11. 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.
- The GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

To configure a GPU for MIG-backed vGPUs, follow these instructions:

1. Enabling MIG Mode for a GPU

   Note: For VMware vSphere, only enabling MIG mode is required because VMware vSphere creates the GPU instances and, after the VM is booted and guest driver is installed, one compute instance is automatically created in the VM.
2. Creating GPU Instances on a MIG-Enabled GPU
3. Optional: Creating Compute Instances in a GPU instance

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

2.11.1. 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.
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 550.54.16   Driver Version: 550.54.16    CUDA Version:  12.3     |
|-------------------------------+----------------------+----------------------+
| 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 with GPUs based on the NVIDIA Ampere architecture only:** Reboot the hypervisor host.

   If you are using any other hypervisor or GPUs that are based on the NVIDIA Hopper™ GPU architecture or a later architecture, omit this step.

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
```

2.11.2. Creating GPU Instances on a MIG-Enabled GPU

Note: If you are using VMware vSphere, omit this task. VMware vSphere creates the GPU instances automatically.

Perform this task in your hypervisor command shell.

1. If necessary, open a command shell as the root user on your hypervisor host machine.
2. List the GPU instance profiles that are available on your GPU.

   You will need to specify the profiles by their IDs, not their names, when you create them.

```
$ nvidia-smi mig -lgip
+-----------------------------------------------------------------------------+
| GPU instance profiles:                                                      |
| GPU   Name          ID    Instances   Memory     P2P    SM    DEC   ENC   |
|                           Free/Total   GiB              CE    JPEG  OFA   |
|=============================================================================|
|   0  MIG 1g.5gb     19     7/7         4.95       No     14     0     0   |
|                                                          1     0     0   |
+-----------------------------------------------------------------------------+
|   0  MIG 2g.10gb    14     3/3         9.90       No     28     1     0   |
|                                                          2     0     0   |
+-----------------------------------------------------------------------------+
```

```
|   0  MIG 3g.20gb     9     2/2        19.79      No     42     2     0    |
|                                                          3     0     0    |
+-----------------------------------------------------------------------------+
|   0  MIG 4g.20gb     5     1/1        19.79      No     56     2     0    |
|                                                          4     0     0    |
+-----------------------------------------------------------------------------+
|   0  MIG 7g.40gb     0     1/1        39.59      No     98     5     0    |
|                                                          7     1     1    |
+-----------------------------------------------------------------------------+
```

3. Create the GPU instances that correspond to the vGPU types of the MIG-backed vGPUs that you will create.

```
$ nvidia-smi mig -cgi gpu-instance-profile-ids
```

gpu-instance-profile-ids
A comma-separated list of GPU instance profile IDs that specifies the GPU instances that you want to create.

This example creates two GPU instances of type `2g.10gb`, which has profile ID 14.

```
$ nvidia-smi mig -cgi 14,14
Successfully created GPU instance ID  5 on GPU  2 using profile MIG 2g.10gb (ID
 14)
Successfully created GPU instance ID  3 on GPU  2 using profile MIG 2g.10gb (ID
 14)
```

2.11.3. Optional: Creating Compute Instances in a GPU instance

Creating compute instances within GPU instances is optional. If you don't create compute instances within the GPU instances, they can be added later for individual vGPUs from within the guest VMs.

Note: If you are using VMware vSphere, omit this task. After the VM is booted and guest driver is installed, one compute instance is automatically created in the VM.

Perform this task in your hypervisor command shell.

1. If necessary, open a command shell as the root user on your hypervisor host machine.
2. List the available GPU instances.

```
$ nvidia-smi mig -lgi
+-------------------------------------------------------+
| GPU instances:                                        |
| GPU   Name          Profile  Instance   Placement  |
|                       ID       ID       Start:Size |
|=======================================================|
|   2  MIG 2g.10gb      14        3          0:2     |
+-------------------------------------------------------+
|   2  MIG 2g.10gb      14        5          4:2     |
+-------------------------------------------------------+
```

3. Create the compute instances that you need within each GPU instance.

```
$ nvidia-smi mig -cci -gi gpu-instance-ids
```

gpu-instance-ids
A comma-separated list of GPU instance IDs that specifies the GPU instances within which you want to create the compute instances.

> **CAUTION:** To avoid an inconsistent state between a guest VM and the hypervisor host, do **not** create compute instances from the hypervisor on a GPU instance on which an active guest VM is running. Instead, create the compute instances from within the guest VM as explained in Modifying a MIG-Backed vGPU's Configuration.

This example creates a compute instance on each of GPU instances 3 and 5.

```
$ nvidia-smi mig -cci -gi 3,5
Successfully created compute instance on GPU  0 GPU instance ID  1 using profile
 ID  2
Successfully created compute instance on GPU  0 GPU instance ID  2 using profile
 ID  2
```

4. Verify that the compute instances were created within each GPU instance.

```
$ nvidia-smi
+-------------------------------------------------------------------------------+
| MIG devices:                                                                  |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  2    3   0   0  |      0MiB /  9984MiB | 28      0 |  2   0    1    0    0 |
|                  |      0MiB / 16383MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+
|  2    5   0   1  |      0MiB /  9984MiB | 28      0 |  2   0    1    0    0 |
|                  |      0MiB / 16383MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+

+-------------------------------------------------------------------------------+
| Processes:                                                                    |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|===============================================================================|
```

> **Note:** Additional compute instances that have been created in a VM are destroyed when the VM is shut down or rebooted. After the shutdown or reboot, only one compute instance remains in the VM. This compute instance is created automatically after the NVIDIA AI Enterprise graphics driver is installed.

2.12. 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.

   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.
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 550.54.16    Driver Version: 550.54.16   CUDA Version:  12.3      |
|-------------------------------+----------------------+----------------------+
| 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 550.54.16    Driver Version: 550.54.16   CUDA Version:  12.3      |
|-------------------------------+----------------------+----------------------+
| 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 |
```

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

2.13. 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.

2.13.1. 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 Mar 25 18:36:45 2024
Driver Version                      : 550.54.16

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 Mar 25 18:37:53 2024
Driver Version                      : 550.54.16

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.

2.13.2. 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 Mar 25 18:36:45 2024
Driver Version                      : 550.54.16

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 Mar 25 18:37:53 2024
Driver Version                      : 550.54.16

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.

2.14. Configuring a vGPU VM for Use with NVIDIA GPUDirect Storage Technology

To use NVIDIA® GPUDirect Storage® technology with NVIDIA vGPU, you must install all the required software in the VM that is configured with NVIDIA vGPU.
Ensure that the prerequisites in Prerequisites for Using NVIDIA AI Enterprise are met.

1. Install and configure the NVIDIA Virtual GPU Manager as explained in Installing and Configuring the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM.
2. As root, log in to the VM that you configured with NVIDIA vGPU in the previous step.
3. Install the Mellanox OpenFabrics Enterprise Distribution for Linux (MLNX_OFED) in the VM as explained in Installation Procedure in *Installing Mellanox OFED*.

   In the command to run the installation script, specify the following options:

   - `--with-nvmf`
   - `--with-nfsrdma`
   - `--enable-gds`
   - `--add-kernel-support`

4. Install the NVIDIA AI Enterprise graphics driver for Linux in the VM from a distribution-specific package.

   Note: GPUDirect Storage technology does **not** support installation of the NVIDIA AI Enterprise graphics driver for Linux from a `.run` file.

   Follow the instructions for the Linux distribution that is installed in the VM:

   - Installing the NVIDIA AI Enterprise Graphics Driver on Ubuntu from a Debian Package
   - Installing the NVIDIA AI Enterprise Graphics Driver on Red Hat Distributions from an RPM Package

5. Install NVIDIA CUDA Toolkit from a `.run` file, deselecting the CUDA driver when selecting the CUDA components to install.

   Note: To avoid overwriting the NVIDIA AI Enterprise graphics driver that you installed in the previous step, do **not** install NVIDIA CUDA Toolkit from a distribution-specific package.

   For instructions, refer to Runfile Installation in *NVIDIA CUDA Installation Guide for Linux*.

6. Use the package manager of the Linux distribution that is installed in the VM to install the GPUDirect Storage technology packages, omitting the installation of the NVIDIA CUDA Toolkit packages.

   Follow the instructions in *NVIDIA CUDA Installation Guide for Linux* for the Linux distribution that is installed in the VM:

   - RHEL 8/Rocky 8

     In the step to install CUDA, execute **only** the command to include all GPUDirect Storage technology packages:

     ```
     sudo dnf install nvidia-gds
     ```

   - Ubuntu

     In the step to install CUDA, execute **only** the command to include all GPUDirect Storage technology packages:

     ```
     sudo apt-get install nvidia-gds
     ```

After you configure a vGPU VM for use with NVIDIA GPUDirect Storage technology, you can license the NVIDIA AI Enterprise licensed products that you are using. For instructions, refer to *NVIDIA AI Enterprise Client Licensing User Guide*.

Chapter 3. Installing and Licensing NVIDIA AI Enterprise Software Components

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 AI Enterprise Graphics Driver Natively

Perform this task in the guest VM by following this sequence of instructions:

- Installing the NVIDIA AI Enterprise Graphics Driver on Linux
- Configuring a Licensed Client of NVIDIA License System
- Installing NVIDIA Container Toolkit

3.2.2. Installing NVIDIA AI Enterprise Software, Applications, and Deep Learning Framework Components by Using Docker

NVIDIA AI Enterprise software components in the infrastructure optimization and cloud native deployment layers are distributed through the NVIDIA AI Enterprise Infra Release 5 collection on NVIDIA NGC. Applications and deep learning framework components for NVIDIA AI Enterprise are distributed **exclusively** through the NGC Public Catalog.

The container image for each 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.

Obtain the Docker `pull` command for downloading each of the following applications and deep learning framework components from the listing for the application or component in the NGC Public Catalog.

- Applications:
  - NVIDIA Clara Parabricks
  - NVIDIA DeepStream
  - NVIDIA Riva
  - MONAI - Medical Open Network for Artificial Intelligence
  - RAPIDS
  - RAPIDS Accelerator for Apache Spark
  - TAO
- Deep learning framework components:
  - NVIDIA TensorRT
  - NVIDIA Triton Inference Server
  - PyTorch
  - TensorFlow 2

Obtain the command for downloading each of the following NVIDIA AI Enterprise software components from the listing for the component in the NVIDIA AI Enterprise Infra Release 5 collection on NVIDIA NGC.

- GPU Operator
- Network Operator
- NVIDIA Base Command Manager Essentials
- vGPU Guest Driver, Ubuntu 22.04

3.3. Installing NVIDIA GPU Operator by Using a Bash Shell Script

A bash shell script for installing NVIDIA GPU Operator with the NVIDIA vGPU guest driver is available for download from NVIDIA NGC.
Before performing this task, ensure that the following prerequisites are met:

- A client configuration token has been generated for the client on which the script will install the vGPU guest driver.
- The API key of the NVIDIA NGC user to be used for creating the image pull secret has been generated.
- The following environment variables are set:

  NGC_API_KEY
  The API key of the NVIDIA NGC user to be used for creating the image pull secret

  For example:

  ```
  export
   NGC_API_KEY="RLh1zerCiG4wPGWWt4Tyj2VMyd7T8MnDyCT95pygP5VJFv8en4eLvdXVZzjm"
  ```

  NGC_USER_EMAIL
  The email address of the NVIDIA NGC user to be used for creating the image pull secret

  For example:

  ```
  export NGC_USER_EMAIL="ada.lovelace@example.com"
  ```

1. Download the NVIDIA GPU Operator - Deploy Installer Script from NVIDIA NGC.
2. Ensure that the file access modes of the script allow the owner to execute the script.
   a). Change to the directory that contains the script.
   ```
   # cd script-directory
   ```
   script-directory
   The directory to which you downloaded the script in the previous step.
   b). Determine the current file access modes of the script.
   ```
   # ls -l gpu-operator-nvaie.sh
   ```
   c). If necessary, grant execute permission to the owner of the script.
   ```
   # chmod u+x gpu-operator-nvaie.sh
   ```
3. Copy the client configuration token to the directory that contains the script.
4. Rename the client configuration token to `client_configuration_token.tok`.
   The client configuration token is generated with a file name that includes a time stamp, namely: `client_configuration_token_mm-dd-yyyy-hh-mm-ss.tok`.
5. From the directory that contains the script, start the script, specifying the option to install the NVIDIA vGPU guest driver.
   ```
   # bash gpu-operator-nvaie.sh install
   ```

3.4. Installing and Licensing NVIDIA AI Enterprise Components Natively

3.4.1. Installing the NVIDIA AI Enterprise Graphics Driver on Windows

After you create a Windows VM on the hypervisor and boot the VM, the VM should boot to a standard Windows desktop in VGA mode at 800×600 resolution. You can use the Windows screen resolution control panel to increase the resolution to other standard resolutions, but to fully enable GPU operation, the NVIDIA AI Enterprise graphics driver must be installed. Windows guest VMs are supported on all NVIDIA vGPU types, namely: Q-series, C-series, B-series, and A-series NVIDIA vGPU types.

1. Copy the NVIDIA Windows driver package to the guest VM where you are installing the driver.
2. Execute the package to unpack and run the driver installer.

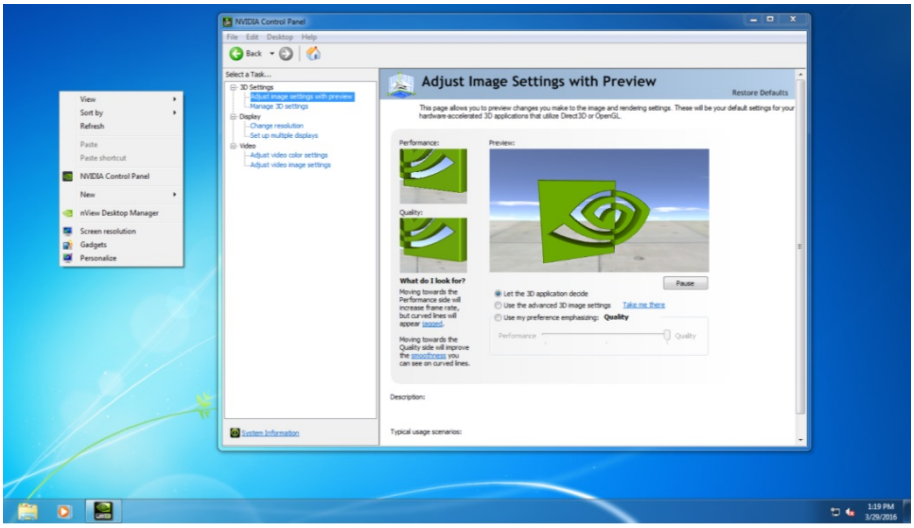
Figure 13. NVIDIA driver installation

3. Click through the license agreement.
4. Select **Express Installation** and click **NEXT**.
   After the driver installation is complete, the installer may prompt you to restart the platform.
5. If prompted to restart the platform, do one of the following:
   - Select **Restart Now** to reboot the VM.
   - Exit the installer and reboot the VM when you are ready.

   After the VM restarts, it boots to a Windows desktop.
6. Verify that the NVIDIA driver is running.
   a). Right-click on the desktop.
   b). From the menu that opens, choose **NVIDIA Control Panel**.
   c). In the **NVIDIA Control Panel**, from the **Help** menu, choose **System Information**.

   NVIDIA Control Panel reports the vGPU or physical GPU that is being used, its capabilities, and the NVIDIA driver version that is loaded.

Figure 14. Verifying NVIDIA driver operation using **NVIDIA Control Panel**



After you install the NVIDIA AI Enterprise graphics driver, you can license any NVIDIA AI Enterprise licensed products that you are using. For instructions, refer to *NVIDIA AI Enterprise Client Licensing User Guide*.

Note: The graphics driver for Windows in this release of NVIDIA AI Enterprise is distributed in a DCH-compliant package. A DCH-compliant package differs from a driver package that is not DCH compliant in the following ways:

- The Windows registry key for license settings for a DCH-compliant package is different than the key for a driver package that is not DCH compliant. If you are upgrading from a driver package that is not DCH compliant in a VM that was previously licensed, you must reconfigure the license settings for the VM. Existing license settings are not propagated to the new Windows registry key for a DCH-compliant package.
- NVIDIA System Management Interface, `nvidia-smi`, is installed in a folder that is in the default executable path.
- The NVWMI binary files are installed in the Windows Driver Store under `%SystemDrive%:\Windows\System32\DriverStore\FileRepository\`.
- NVWMI help information in Windows Help format is not installed with graphics driver for Windows guest OSes.

3.4.2. Installing the NVIDIA AI Enterprise Graphics Driver on Linux

The NVIDIA AI Enterprise graphics driver for Linux is distributed as a Debian package for Ubuntu distributions and as an RPM package for Red Hat distributions. The procedure for installing the driver is the same in a VM and on bare metal.

If you are using a Linux OS for which the Wayland display server protocol is enabled by default, disable it as explained in Disabling the Wayland Display Server Protocol for Red Hat Enterprise Linux.

How to install the NVIDIA AI Enterprisegraphics driver on Linux depends on the distribution format from which you are installing the driver. For detailed instructions, refer to:

- Installing the NVIDIA AI Enterprise Graphics Driver on Ubuntu from a Debian Package
- Installing the NVIDIA AI Enterprise Graphics Driver on Red Hat Distributions from an RPM Package

3.4.2.1. 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-550_550.54.15_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-550_550.54.15_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
```

3.4.2.2. Installing the NVIDIA AI Enterprise Graphics Driver on Red Hat Distributions from an RPM Package

The NVIDIA AI Enterprise graphics driver for Red Hat Distributions is distributed as an RPM package file.
This task requires `root` user privileges.

1. Copy the NVIDIA AI Enterprise Linux driver package, for example `nvidia-linux-grid-550.54.15_amd64.rpm`, to the guest VM where you are installing the driver.
2. Log in to the guest VM as a user with `root` user 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.

```
$ rpm -iv ./nvidia-linux-grid-550.54.15_amd64.rpm
```

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
```

3.4.2.3. Disabling the Nouveau Driver for NVIDIA Graphics Cards

If the Nouveau driver for NVIDIA graphics cards is present, disable it before installing the NVIDIA AI Enterprise graphics driver.

Note: If you are using SUSE Linux Enterprise Server, you can skip this task because the Nouveau driver is not present in SUSE Linux Enterprise Server.

Run the following command and if the command prints any output, the Nouveau driver is present and must be disabled.

```
$ lsmod | grep nouveau
```

1. Create the file `/etc/modprobe.d/blacklist-nouveau.conf` with the following contents:

```
blacklist nouveau
options nouveau modeset=0
```

2. Regenerate the kernel initial RAM file system (initramfs).

   The command to run to regenerate the kernel initramfs depends on the Linux distribution that you are using.

Linux Distribution	Command
CentOS	`$ sudo dracut --force`
Debian	`$ sudo update-initramfs -u`
Red Hat Enterprise Linux	`$ sudo dracut --force`
Ubuntu	`$ sudo update-initramfs -u`

3. Reboot the host or guest VM.

3.4.2.4. Disabling the Wayland Display Server Protocol for Red Hat Enterprise Linux

Starting with Red Hat Enterprise Linux Desktop 8.0, the Wayland display server protocol is used by default on supported GPU and graphics driver configurations. However, the NVIDIA AI Enterprise graphics driver for Linux requires the X Window System. Before installing the driver, you must disable the Wayland display server protocol to revert to the X Window System.

Perform this task from the host or guest VM that is running Red Hat Enterprise Linux Desktop.

This task requires administrative access.

1. In a plain text editor, edit the file `/etc/gdm/custom.conf` and remove the comment from the option `WaylandEnable=false`.
2. Save your changes to `/etc/gdm/custom.conf`.
3. Reboot the host or guest VM.

3.4.2.5. Disabling GSP Firmware

Some GPUs include a GPU System Processor (GSP), which may be used to offload GPU initialization and management tasks. In GPU pass through and bare-metal deployments on Linux, GSP is supported only for vCS. If you are using any other product in a GPU pass through or bare-metal deployment on Linux, you must disable the GSP firmware.

Note:

For NVIDIA vGPU deployments on Linux and all NVIDIA AI Enterprise deployments on Windows, omit this task.

GSP firmware is supported with NVIDIA vGPU deployments on GPUs that are based on the NVIDIA Ada Lovelace GPU architecture. For NVIDIA vGPU deployments on Linux and all NVIDIA AI Enterprise deployments on Windows on GPUs based on earlier GPU architectures, GSP is also not supported but GSP firmware is already disabled.

For each NVIDIA AI Enterprise product, the following table lists whether GSP is supported in deployments in which GSP firmware can be enabled. The table also summarizes the behavior of NVIDIA AI Enterprise if a VM or host requests a license when GSP firmware is enabled. The deployments in which GSP firmware can be enabled are GPU pass through and bare-metal deployments on Linux.

Product	GSP	License Request	Error Message
vCS	Supported	Allowed	Not applicable
vApps	Not supported	Blocked	Printed
vWS	Not supported	Blocked	Printed

When a license request is blocked, the following error message is written to the licensing event log file at the location given in *NVIDIA AI Enterprise Client Licensing User Guide*:

```
Invalid feature requested for the underlying GSP firmware configuration.
Disable GSP firmware to use this feature.
```

Perform this task on the VM to which the GPU is passed through or on the bare-metal host.

Ensure that the NVIDIA AI Enterprise graphics driver for Linux is installed on the VM or bare-metal host.

1. Log in to the VM or bare-metal host and open a command shell.
2. Determine whether GSP firmware is enabled.

   ```
   $ nvidia-smi -q
   ```

   - If GSP firmware is enabled, the command displays the GSP firmware version, for example:

     ```
     GSP Firmware Version                    : 550.54.15
     ```

   - Otherwise, the command displays `N/A` as the GSP firmware version.
3. If GSP firmware is enabled, disable it by setting the NVIDIA module parameter `NVreg_EnableGpuFirmware` to 0.

   Set this parameter by adding the following entry to the `/etc/modprobe.d/nvidia.conf` file:

   ```
   options nvidia NVreg_EnableGpuFirmware=0
   ```

   If the `/etc/modprobe.d/nvidia.conf` file does not already exist, create it.
4. Reboot the VM or bare-metal host.

If you later need to enable GSP firmware, set the NVIDIA module parameter `NVreg_EnableGpuFirmware` to 1.

3.4.3. Configuring a Licensed Client of NVIDIA License System

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.

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.

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.

3.4.3.1. Proxy Server Requirements and Firewall Rules for a CLS Instance

To enable communication between a licensed client and a CLS instance through a proxy server, the proxy server must meet certain requirements. To enable communication through a firewall, firewall rules that allow traffic on specific URLs through specific ports must be defined.

The processes for configuring a proxy server and defining firewall rules are separate from the process for configuring a CLS instance. Use the standard interfaces of the proxy server and the firewall that you are using to perform these processes.

Proxy Server Requirements for a CLS Instance

NVIDIA License System supports transparent proxy servers and non-transparent proxy servers.

- **A transparent proxy server** identifies itself to the server and does **not** modify client requests and responses.
- **A non-transparent proxy server** does not reveal the IP address of the client and modifies client requests and responses.

Any proxy server between a licensed client and a CLS instance must allow programmatic calls to the URL `api.cls.licensing.nvidia.com`.

Non-Transparent Proxy Server Support

NVIDIA License System supports both authenticated and unauthenticated non-transparent proxy servers.

The following authenticated proxy servers are supported:

- Squid

The following authentication methods are supported for authenticated proxy servers:

- Basic

- Microsoft Windows Challenge/Response (Microsoft NTLM) (Windows clients only)
- Kerberos (only for clients that are a member of an Active Directory domain)

Firewall Rules for a CLS Instance

To enable communication between a licensed client and a CLS instance through a firewall, firewall rules that allow traffic on the URLs through the ports specified in the following table must be defined.

URL	Port	Traffic
`api.cls.licensing.nvidia.com`	443	• Licensing operations, namely, the borrowing, renewal, and return of a license. • Licensed client authentication
`api.licensing.nvidia.com`	80	License return from a Windows licensed client that has not been shut down cleanly

3.4.3.2. Configuring a Licensed Client on Windows with Default Settings

Perform this task from the client.

1. Copy the client configuration token to the `%SystemDrive%:\Program Files\NVIDIA Corporation\vGPU Licensing\ClientConfigToken` folder.
2. Restart the `NvDisplayContainer` service.

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

3.4.3.3. Configuring a Licensed Client 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

This example shows how to configure a licensed Linux client for NVIDIA Virtual Compute Server.

```
# /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=4
...
```

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.

3.4.3.4. Generating an Encrypted Credentials File

Some authentication methods require a licensed client to provide user credentials when the client authenticates with a proxy server. To enable the client to provide these credentials securely without input from a user, you must generate a file that contains these credentials in an encrypted form that the client can read.

The following authentication methods require an encrypted credentials file:

- Basic authentication

- Microsoft Windows Challenge/Response (NTLM) authentication for a client that is not a member of an Active Directory domain

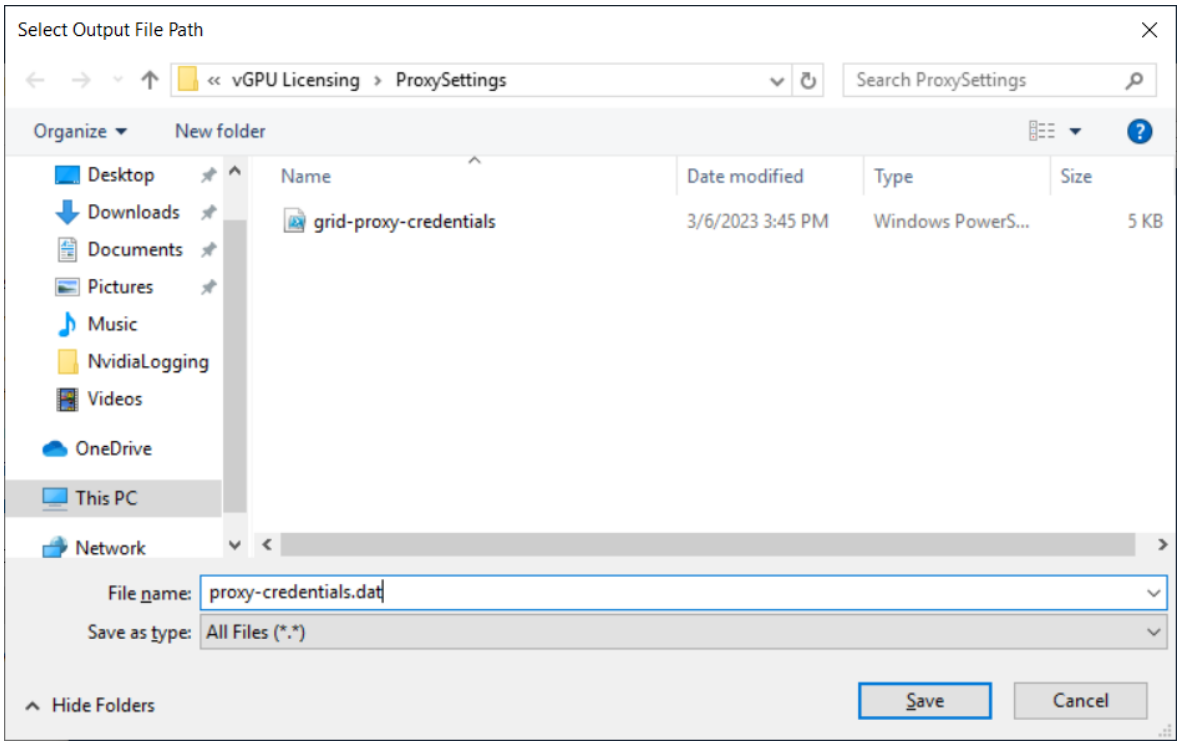
How to generate an encrypted credentials file depends on the OS that client is running. For detailed instructions, refer to the following topics:

- Generating an Encrypted Credentials File on Windows
- Generating an Encrypted Credentials File on Linux

3.4.3.4.1. Generating an Encrypted Credentials File on Windows

Perform this task in a **Windows PowerShell** window as the Administrator user on the client.

1. Change to the `C:\Program Files\NVIDIA Corporation\vGPU Licensing\ProxySettings` folder.
   ```
   PS C:\> cd "C:\Program Files\NVIDIA Corporation\vGPU Licensing\ProxySettings"
   ```
2. Run the `grid-proxy-credentials` Windows PowerShell script.
   ```
   PS C:\> .\grid-proxy-credentials.ps1
   ```
3. In the **Select Output File Path** window that opens, navigate to the directory in which you want to generate the credentials file, enter the file name, and click **Save**.



4. When prompted in the **Windows PowerShell** window, specify the password of the user that will log in to the proxy server when the licensed client requests a license.

Provide the path to this file when configuring a licensed client that will use the file as explained in Configuring a Licensed Client on Windows with Default Settings.

3.4.3.4.2. Generating an Encrypted Credentials File on Linux

Perform this task in a Linux command shell on the client.

1. Run the `grid-proxy-credentials.sh` command.

   ```
   # /usr/lib/nvidia/grid-proxy-credentials.sh -o output-file-path
   ```

   output-file-path
   The full path to the credentials file that you are generating. Ensure that the directory in the path exists.

   Tip: To get help information for this command, type **`/usr/lib/nvidia/grid-proxy-credentials.sh --help`**.

   This example creates the credentials file `/etc/nvidia/proxy-credentials.dat`.

   ```
   # /usr/lib/nvidia/grid-proxy-credentials.sh -o /etc/nvidia/proxy-credentials.dat
   ```

2. When prompted, specify the password of the user that will log in to the proxy server when the licensed client requests a license.

Provide the path to this file when configuring a licensed client that will use the file as explained in Configuring a Licensed Client on Linux with Default Settings.

3.4.3.5. 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.

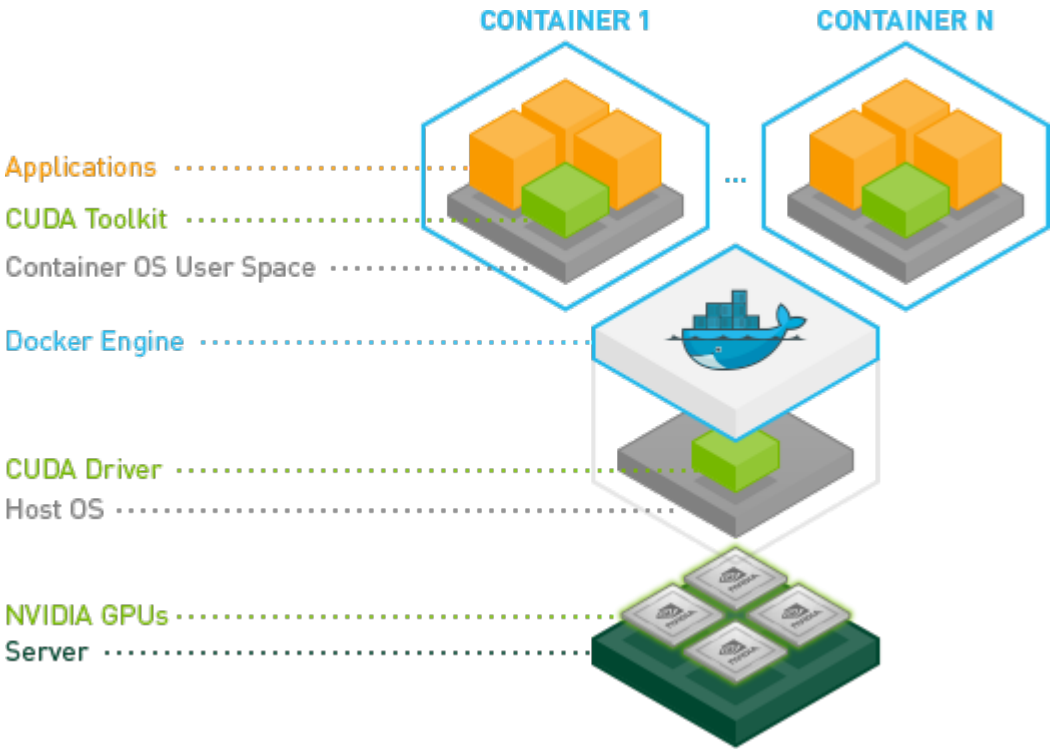
To verify the license status of a licensed client, run `nvidia-smi` with the `-q` or `--query` optionfrom the licensed client, **not** the hypervisor host. If the product is licensed, the expiration date is shown in the license status.

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

Timestamp                                 : Wed Nov 23 10:52:59 2022
Driver Version                            : 525.60.06
CUDA Version                              : 12.0

Attached GPUs                             : 2
GPU 00000000:02:03.0
    Product Name                          :
    Product Brand                         : NVIDIA Virtual Compute Server
    Product Architecture                  : Ampere
    Display Mode                          : Enabled
    Display Active                        : Disabled
    Persistence Mode                      : Enabled
    MIG Mode
        Current                           : Disabled
        Pending                           : Disabled
    Accounting Mode                       : Disabled
    Accounting Mode Buffer Size           : 4000
    Driver Model
        Current                           : N/A
```

```
        Pending                               : N/A
    Serial Number                             : N/A
    GPU UUID                                  : GPU-ba5b1e9b-1dd3-11b2-be4f-98ef552f4216
    Minor Number                              : 0
    VBIOS Version                             : 00.00.00.00.00
    MultiGPU Board                            : No
    Board ID                                  : 0x203
    Board Part Number                         : N/A
    GPU Part Number                           : 25B6-890-A1
    Module ID                                 : N/A
    Inforom Version
        Image Version                         : N/A
        OEM Object                            : N/A
        ECC Object                            : N/A
        Power Management Object               : N/A
    GPU Operation Mode
        Current                               : N/A
        Pending                               : N/A
    GSP Firmware Version                      : N/A
    GPU Virtualization Mode
        Virtualization Mode                   : VGPU
        Host VGPU Mode                        : N/A
    vGPU Software Licensed Product
        Product Name                          : NVIDIA Virtual Compute Server
        License Status                        : Licensed (Expiry: 2022-11-23 10:41:16
GMT)
    …
    …
```

3.4.4. 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
```

3.4.5. Verifying the Installation of NVIDIA Container Toolkit

1. Run the `nvidia-smi` command contained in the latest official NVIDIA CUDA Toolkit image that is compatible with the release of the NVIDIA CUDA Toolkit driver that is running on your machine.

Note: Do not use a release of the NVIDIA CUDA Toolkit image later than the release of the NVIDIA CUDA Toolkit driver that is running on your machine. For a list of all NVIDIA CUDA Toolkit images, refer to nvidia/cuda on Docker Hub.

```
$ docker run --gpus all nvidia/cuda:12.3.0-base-ubuntu22.04 nvidia-smi
```

2. Start a GPU-enabled container on any two available GPUs.

```
$ docker run --gpus 2 nvidia/cuda:12.3.0-base-ubuntu22.04 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:12.3.0-base-ubuntu22.04 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:12.3.0-base-ubuntu22.04 nvidia-
smi
```

5. Specify a GPU capability for the container.

```
$ docker run --gpus all,capabilities=utility nvidia/cuda:12.3.0-base-ubuntu22.04 nvidia-
smi
```

3.4.6. Installing Software Distributed as Container Images

The NGC container images accessed through the NVIDIA NGC Catalog include the AI and data science applications and frameworks. 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.

Obtain the Docker `pull` command for downloading each of the following applications and deep learning framework components from the listing for the application or component in the NGC Public Catalog.

- Applications:
  - NVIDIA Clara Parabricks
  - NVIDIA DeepStream
  - NVIDIA Riva
  - MONAI - Medical Open Network for Artificial Intelligence
  - RAPIDS
  - RAPIDS Accelerator for Apache Spark
  - TAO
- Deep learning framework components:
  - NVIDIA TensorRT
  - NVIDIA Triton Inference Server
  - PyTorch
  - TensorFlow 2

3.4.7. Running ResNet-50 with TensorRT

1. Launch the NVIDIA TensorRT container image on all GPUs in interactive mode, specifying that the container will be deleted when it is stopped.

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

2. 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
   ```

3. 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
   ```
4. Press **Ctrl**+**P**, **Ctrl**+**Q** to exit the container runtime and return to the Linux command shell.

3.4.8. Running ResNet-50 with TensorFlow

1. Launch the **TensorFlow 1** container image on all GPUs in interactive mode, specifying that the container will be deleted when it is stopped.
   ```
   $ sudo docker run --gpus all -it --rm \
   nvcr.io/nvidia/tensorflow:21.07-tf1-py3
   ```
2. From within the container runtime, change to the directory that contains test data for `cnn` example.
   ```
   # cd /workspace/nvidia-examples/cnn
   ```
3. Run the ResNet-50 training test with FP16 precision.
   ```
   # python resnet.py --layers 50 -b 64 -i 200 -u batch --precision fp16
   ```
4. Confirm that all operations on the application are performed correctly and that a set of results is reported when the test is completed.
5. Press **Ctrl**+**P**, **Ctrl**+**Q** to exit the container runtime and return to the Linux command shell.

3.4.9. 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.

3.5. The NVIDIA NGC Catalog

NVIDIA AI Enterprise components are distributed through the NVIDIA NGC Catalog. Infrastructure and workload management components are distributed as resources in the NVIDIA AI Enterprise Infra Release 5 collection. Tools for AI development and use cases are available from the NVIDIA AI Enterprise Software Suite.

3.5.1. Resources

Infrastructure and workload management components of NVIDIA AI Enterprise are distributed as resources in the NVIDIA AI Enterprise Infra Release 5 collection.

The **NVIDIA AI Enterprise Infra Release 5** collection contains the following resources:

- GPU Operator
- Network Operator
- NVIDIA Base Command Manager Essentials
- vGPU Guest Driver, Ubuntu 22.04

Before downloading any NVIDIA AI Enterprise software assets, ensure that you have signed in to NVIDIA NGC from the NVIDIA NGC Sign In page.

1. Go to the NVIDIA AI Enterprise Infra Release 5 collection on NVIDIA NGC.
2. Click the **Entities** tab and select the resource that you are interested in.
3. Click **Download** and, from the menu that opens, choose to download the resource by using a direct download in the browser, the displayed `wget` command, or the CLI.

3.5.2. Container Images

You obtain AI and data science container images by using Docker to pull the images from the NVIDIA NGC container registry.
For each container image that you want, perform this task from the VM or host where you want to run the container image.

1. Generate your API key.
2. Access the NVIDIA NGC container registry.
   a). Log in to the NVIDIA NGC container registry.
   ```
   sudo docker login nvcr.io
   ```
   b). When prompted for your username, enter the text **`$oauthtoken`**.
   ```
   Username: $oauthtoken
   ```
   c). When prompted for your password, enter your NVIDIA NGC API key.
   ```
   Password: my-api-key
   ```
3. For each AI or data science application that you are interested in, load the container.
   ```
   sudo docker pull nvcr.io/nvaie/tensorflow:21.02-tf2-py3
   ```

3.5.3. Helm Charts

The **NVIDIA AI Enterprise Infra Release 5** collection contains Helm charts for use by NVIDIA GPU Operator and NVIDIA Network Operator to deploy and manage GPU resources and network resources for deployments based on Kubernetes.
Before downloading any NVIDIA AI Enterprise software assets, ensure that you have signed in to NVIDIA NGC from the NVIDIA NGC Sign In page.

1. Go to the NVIDIA AI Enterprise Infra Release 5 collection on NVIDIA NGC.
2. Click the **Entities** tab and select the Helm chart that you are interested in.
3. Download the Helm chart as explained in Manage Helm Charts Using the Helm CLI in *NGC Private Registry User Guide*.

3.5.4. Models

The Feature Branches and Models collection contains pretrained unencrypted models for MONAI and Tao Toolkit.

1. Go to the Feature Branches and Models collection.
2. Click the **Entities** tab and select the model that you are interested in.
3. On the page that opens for the model that you selected, click **Download** and choose to download the resource by using a direct download in the browser, the displayed `wget` command, or the CLI.

3.5.5. Accessing the NVIDIA AI Enterprise Software Suite

Tools for AI development and use cases are available from the NVIDIA AI Enterprise Software Suite, which is distributed through the NVIDIA NGC Catalog.
Before downloading any NVIDIA AI Enterprise software assets, ensure that you have signed in to NVIDIA NGC from the NVIDIA NGC Sign In page.

1. View the NVIDIA AI Enterprise Software Suite on NVIDIA NGC.
   - Go to the NVIDIA AI Enterprise Supported page on NVIDIA NGC.
   - Visit the NVIDIA NGC site and set the **NVIDIA AI Enterprise Support** filter.
2. Browse the NVIDIA AI Enterprise Software Suite to find software assets that you are interested in.
3. For each software asset that you are interested in, click the asset to learn more about or download the asset.

3.5.6. Adding Additional Users from Your Organization to the Enterprise Catalog (Admins Only)

As an admin, you are responsible for giving members of your organization access to the Enterprise Catalog.

1. Make sure you are signed in.
2. Make sure to select your company's organization from the user menu on the top right.
3. On the left side menu, select **Organization** and click on **Users**, then click the + icon at the bottom of the screen and then click the **Invite New User** icon.
4. Provide the name and email address of the user you would like to add.
5. Provision user roles for the new user:
   a). To give the new user access to the entities in the Enterprise Catalog, provide them with the user role **NVIDIA AI Enterprise Viewer**.
   b). To make the user an admin that can add additional users to the Enterprise Catalog, provision the user roles: **NVIDIA AI Enterprise Viewer** and **User Admin**.
   c). To give the user access to your organization's Private Registry, see Accessing Your NGC Private Registry. Provisioning access to the Enterprise Catalog and your organization's Private Registry can be done in one or two steps.

3.6. The NGC Private Registry

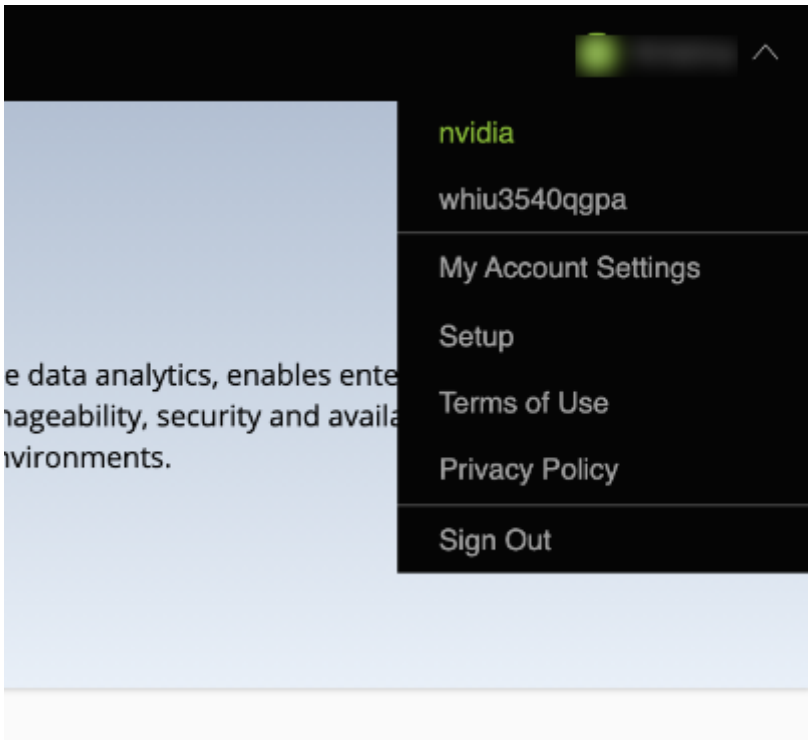
As an NVIDIA AI Enterprise user, you have exclusive access to your organization's own NGC Private Registry, which gives authorized users within your organization privileges

to store your company's proprietary software and tools, including custom models, frameworks, and helm charts, in one location.

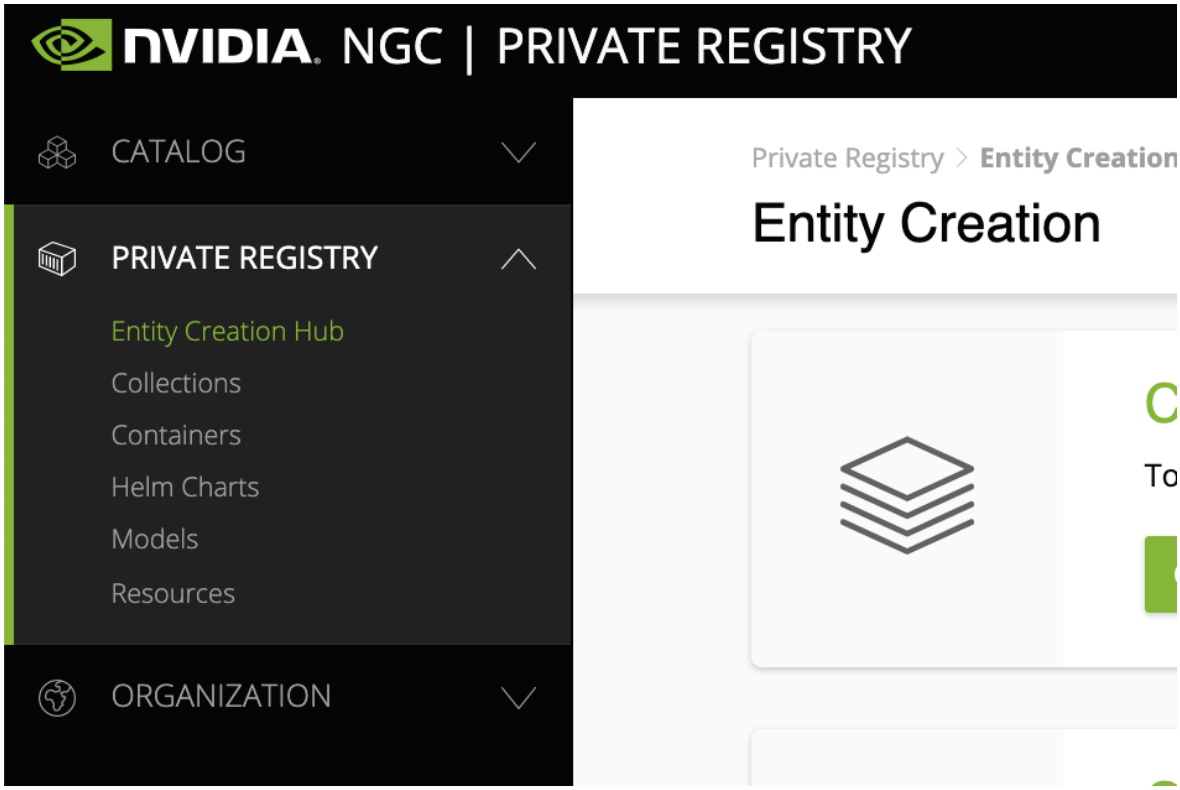
The complete NGC Private Registry user guide can be found here.

3.6.1. Accessing Your NGC Private Registry

1. To access your NGC Private Registry, sign in with your NGC Account.
2. In the top right corner, click your user account icon and select the **orgname**.



3. To view artifacts in your NGC Private Registry, select **Private Registry** in the left-hand menu.



4. You can access the content of the NGC Private Registry by selecting one of the entity types (Collections, Containers, Helm Charts, Models, Resources).
5. To upload entities to your NGC Private Registry, click on **Entity Creation Hub**.

3.6.2. Managing Teams and Users

As an admin, you can add users to your organization's NGC Private Registry and create teams within the NGC Private Registry.

Before adding users and teams, familiarize yourself with the following definitions of each role here.

3.6.2.1. Creating Teams

Creating teams allows users to share images within a team while keeping them invisible to other teams in the same organization. Only organization administrators can create teams.

Here is how you create a team.

3.6.2.2. Creating Users

As the organization administrator, you must create user accounts to allow others to use the NGC container registry within the organization.

Here is how you create a new user.

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.

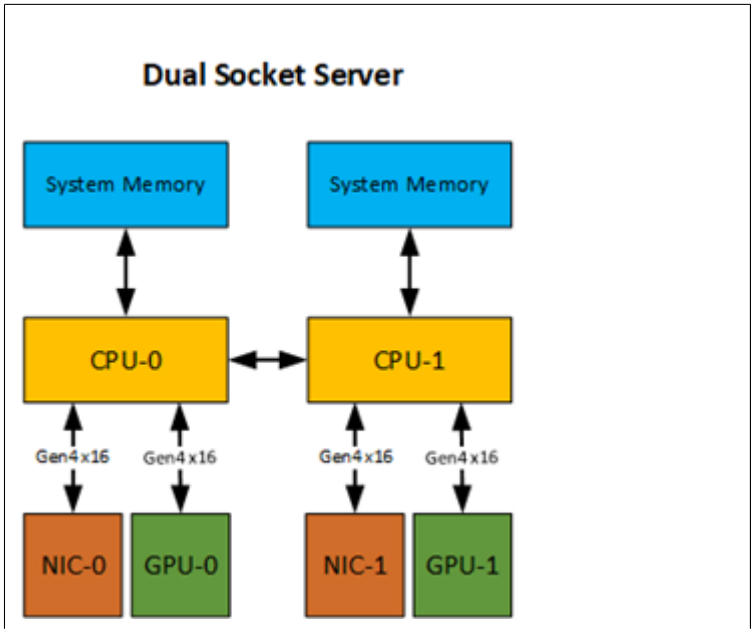
4.2. Configuring NUMA Affinity for the VMs

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:.

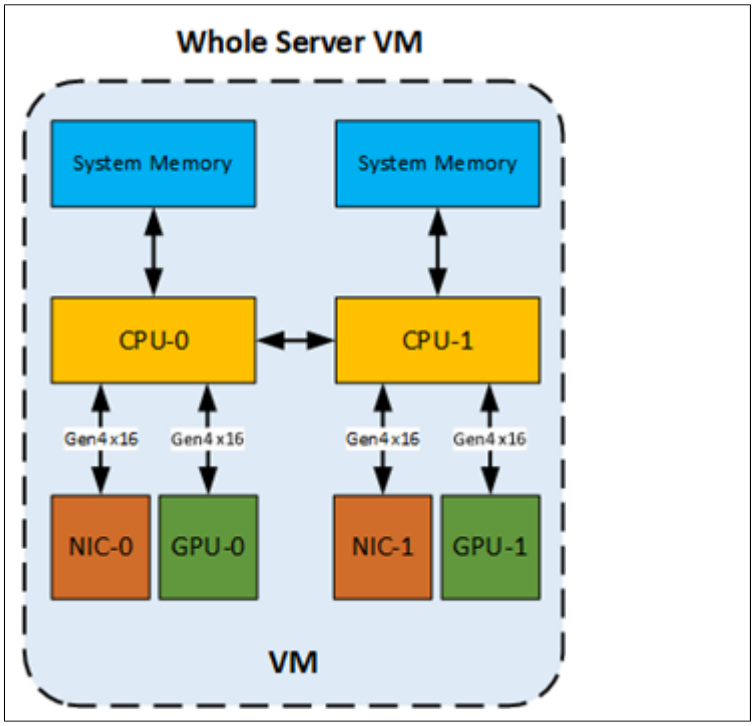
- Whole-server VM with two GPUs and two NICs across both NUMA nodes
- Per-socket VM with one GPU and one NIC paired on a single NUMA node

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:



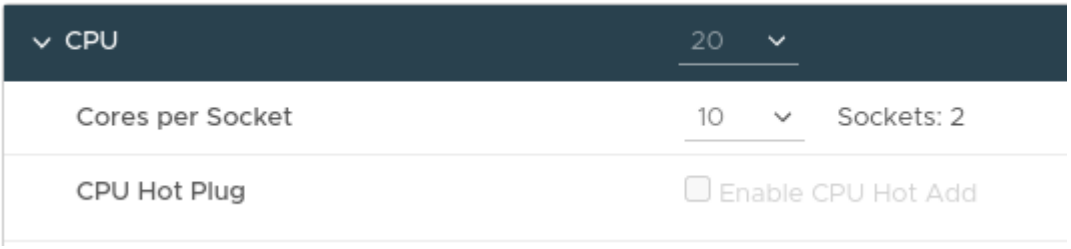
Perform this task on each hypervisor host.

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.

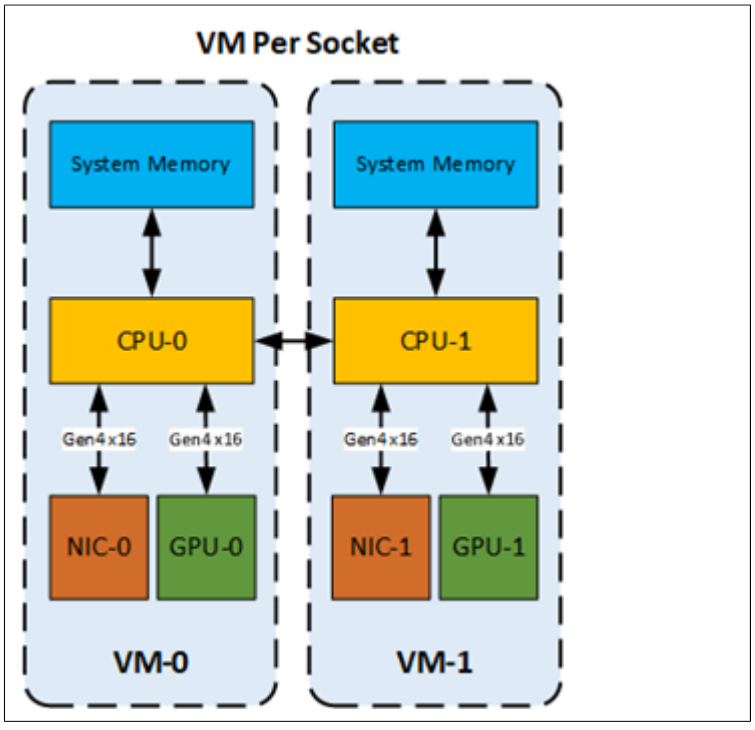


3. With two GPUs and NICs in the VM across NUMA nodes, set the NUMA affinity in the VM configuration to include both NUMA nodes 0 and 1.

```
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

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.

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

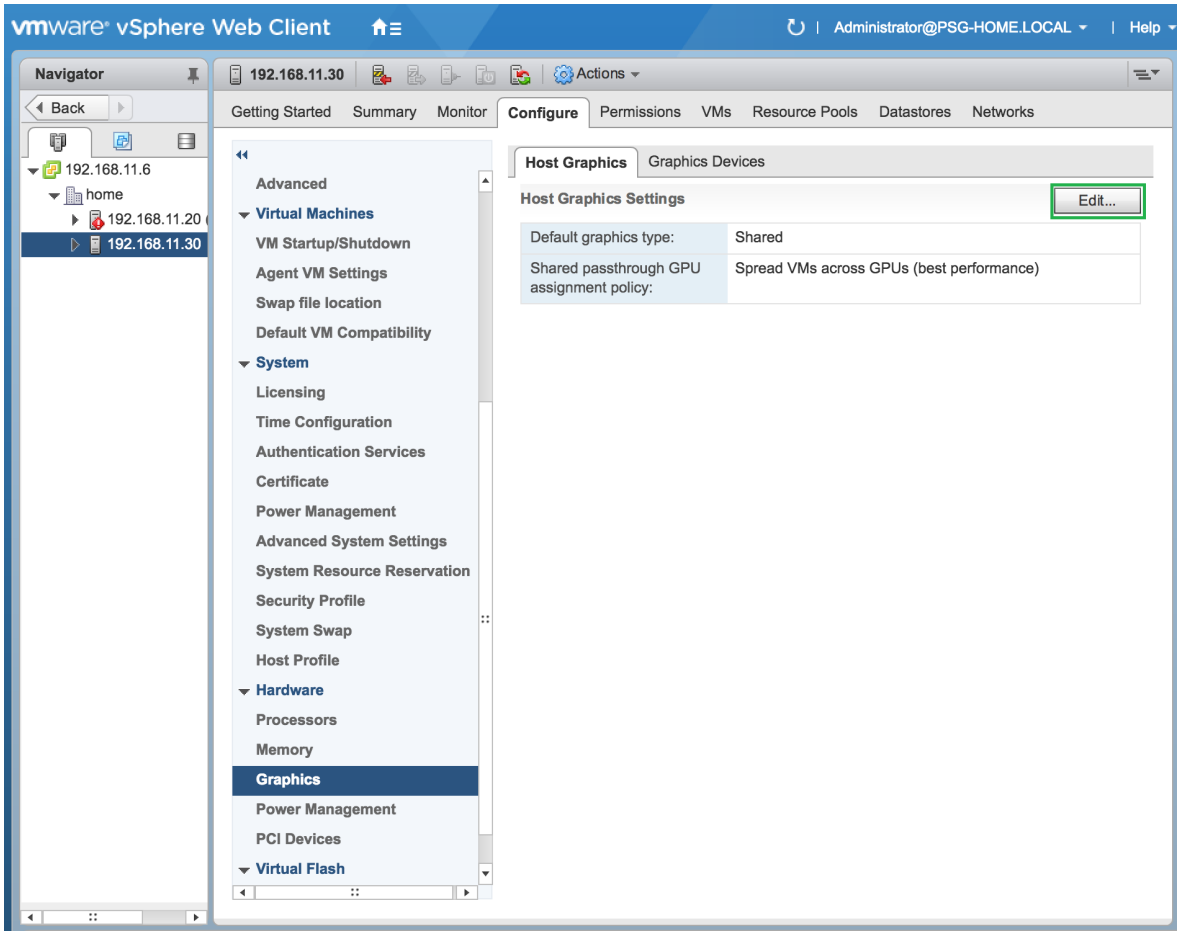
If the default GPU allocation policy does not meet your requirements for performance or density of vGPUs, you can change it.

5.2.1. Modifying GPU Allocation Policy on VMware vSphere

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.

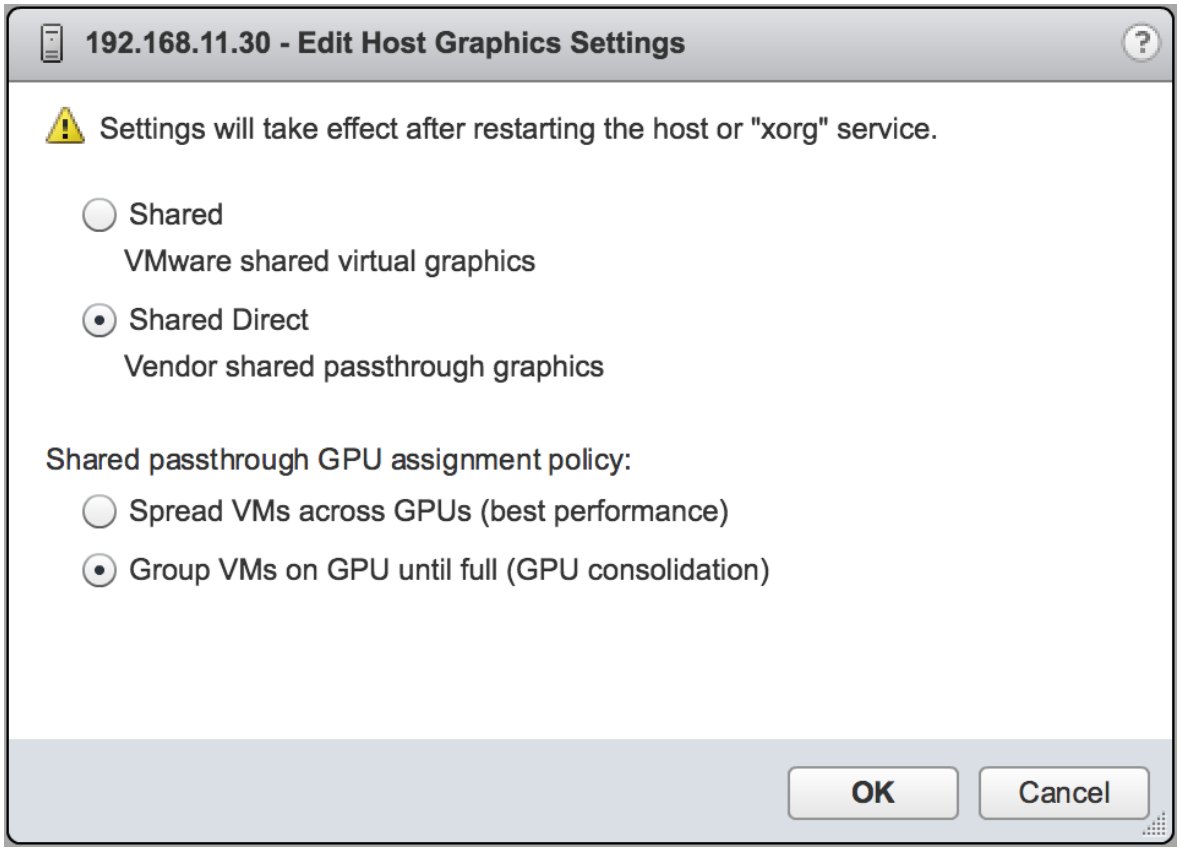
1. Log in to vCenter Server by using the vSphere Web Client.
2. In the navigation tree, select your ESXi host and click the **Configure** tab.
3. From the menu, choose **Graphics** and then click the **Host Graphics** tab.
4. On the **Host Graphics** tab, click **Edit**.

Figure 15. Breadth-first allocation scheme setting for vGPU-enabled VMs



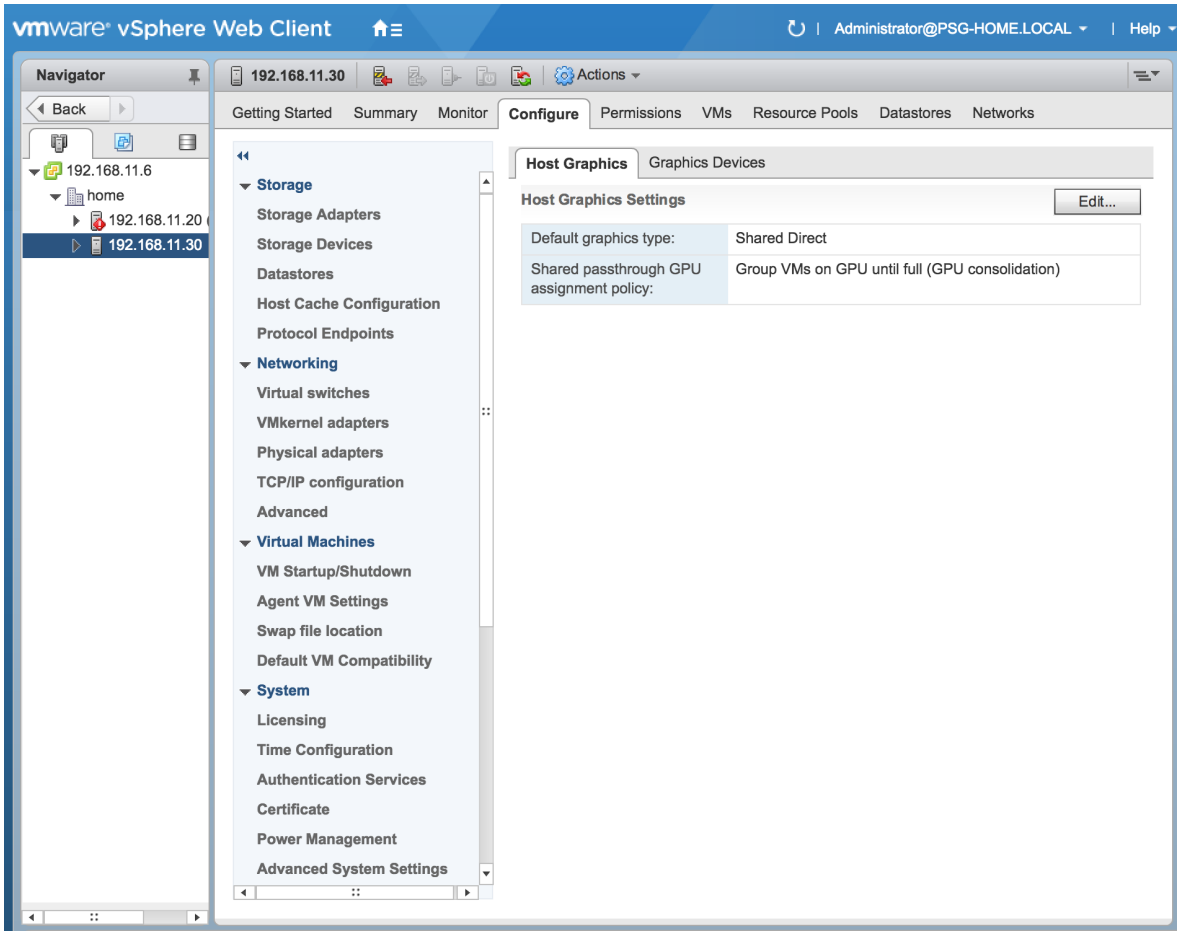
5. In the **Edit Host Graphics Settings** dialog box that opens, select these options and click **OK**.
   a). If not already selected, select **Shared Direct**.
   b). Select **Group VMs on GPU until full**.

Figure 16. 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 17. 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.

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.
```

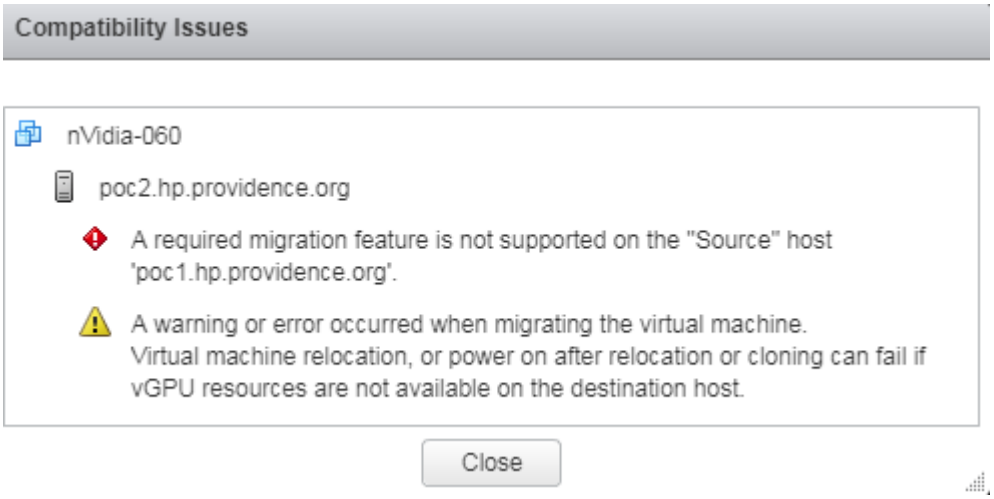
The window appears as follows:

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.

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.

   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.
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 Mar 25 19:01:24 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.16    Driver Version: 550.54.16   CUDA Version:  12.3     |
|-------------------------------+----------------------+----------------------+
| 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 Mar 25 19:01:24 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.16    Driver Version: 550.54.16   CUDA Version:  12.3      |
|-------------------------------+----------------------+----------------------+
| 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 Red Hat Enterprise Linux KVM

On Red Hat Enterprise Linux KVM, enable unified memory by setting the **`enable_uvm`** vGPU plugin parameter.

Ensure that the `mdev` device file that represents the vGPU has been created as explained in Creating an NVIDIA vGPU on a Linux with KVM Hypervisor.

Perform this task for each vGPU that requires unified memory.

Set the **`enable_uvm`** vGPU plugin parameter for the `mdev` device file that represents the vGPU to 1 as explained in Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor.

5.5.2. 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 Red Hat Enterprise Linux KVM

Set the **`enable_debugging`** vGPU plugin parameter for the `mdev` device file that represents the vGPU that is assigned to the VM to 1 as explained in Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor.

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.

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 Red Hat Enterprise Linux KVM

Set the **`enable_profiling`** vGPU plugin parameter for the `mdev` device file that represents the vGPU that is assigned to the VM to `1` as explained in Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor.

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.

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.

5.7. Enabling the TCC Driver Model for a vGPU

The Tesla Compute Cluster (TCC) driver model supports CUDA C/C++ applications. This model is optimized for compute applications and reduces kernel launch times on Windows. By default, the driver model of a vGPU that is assigned to a Windows VM is Windows Display Driver Model (WDDM). If you want to use the TCC driver model, you must enable it explicitly.
This task requires administrator privileges.
Perform this task from the VM to which the vGPU is assigned.

Note: Only Q-series vGPUs support the TCC driver model.

1. Log on to the VM to which the vGPU is assigned.
2. Set the driver model of the vGPU to the TCC driver model.
   ```
   nvidia-smi -g vgpu-id -dm 1
   ```
   vgpu-id
   The ID of the vGPU for which you want to enable the TCC driver model. If the `-g` is omitted, the TCC driver model is enabled for all vGPUs that are assigned to the VM.
3. Reboot the VM.

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.

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 VMware ESXi host shell, `nvidia-smi` reports management information for NVIDIA physical GPUs and virtual GPUs present in the system.
- From a guest VM, `nvidia-smi` retrieves usage statistics for vGPUs or pass-through GPUs that are assigned to the VM.

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 Mar 22 09:26:18 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.16              Driver Version: 550.54.16                 |
|-------------------------------+----------------------+----------------------+
| 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 |
|                                                                   Usage      |
|=============================================================================|
|    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 Mar 22 09:27:06 2024
+---------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.16              Driver Version: 550.54.16                     |
|---------------------------------+------------------------------+-----------+
| 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 Physical GPU Details

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

```
[root@vgpu ~]# nvidia-smi -q
==============NVSMI LOG==============

Timestamp                                 : Tue Nov 22 10:33:26 2022
Driver Version                            : 525.60.06
CUDA Version                              : Not Found
vGPU Driver Capability
        Heterogenous Multi-vGPU           : Supported

Attached GPUs                             : 3
GPU 00000000:C1:00.0
    Product Name                          : Tesla T4
    Product Brand                         : NVIDIA
    Product Architecture                  : Turing
    Display Mode                          : Enabled
    Display Active                        : Disabled
    Persistence Mode                      : Enabled
    vGPU Device Capability
        Fractional Multi-vGPU             : Supported
        Heterogeneous Time-Slice Profiles : Supported
        Heterogeneous Time-Slice Sizes    : Not Supported
    MIG Mode
        Current                           : N/A
```

```
        Pending                           : N/A
    Accounting Mode                       : Enabled
    Accounting Mode Buffer Size           : 4000
    Driver Model
        Current                           : N/A
        Pending                           : N/A
    Serial Number                         : 1321120031291
    GPU UUID                              : GPU-9084c1b2-624f-2267-4b66-345583fbd981
    Minor Number                          : 1
    VBIOS Version                         : 90.04.38.00.03
    MultiGPU Board                        : No
    Board ID                              : 0xc100
    Board Part Number                     : 900-2G183-0000-001
    GPU Part Number                       : 1EB8-895-A1
    Module ID                             : 0
    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
    GSP Firmware Version                  : N/A
    GPU Virtualization Mode
        Virtualization Mode               : Host VGPU
        Host VGPU Mode                    : Non SR-IOV
    IBMNPU
        Relaxed Ordering Mode             : N/A
    PCI
        Bus                               : 0xC1
        Device                            : 0x00
        Domain                            : 0x0000
        Device Id                         : 0x1EB810DE
        Bus Id                            : 00000000:C1:00.0
        Sub System Id                     : 0x12A210DE
        GPU Link Info
            PCIe Generation
                Max                       : 3
                Current                   : 1
                Device Current            : 1
                Device Max                : 3
                Host Max                  : N/A
            Link Width
                Max                       : 16x
                Current                   : 16x
        Bridge Chip
            Type                          : N/A
            Firmware                      : N/A
        Replays Since Reset               : 0
        Replay Number Rollovers           : 0
        Tx Throughput                     : 0 KB/s
        Rx Throughput                     : 0 KB/s
        Atomic Caps Inbound               : N/A
        Atomic Caps Outbound              : N/A
    Fan Speed                             : N/A
    Performance State                     : P8
    Clocks Throttle Reasons
        Idle                              : Active
        Applications Clocks Setting       : Not Active
        SW Power Cap                      : Not Active
        HW Slowdown                       : Not Active
            HW Thermal Slowdown           : Not Active
            HW Power Brake Slowdown       : Not Active
        Sync Boost                        : Not Active
        SW Thermal Slowdown               : Not Active
        Display Clock Setting             : Not Active
```

```
    FB Memory Usage
        Total                             : 15360 MiB
        Reserved                          : 0 MiB
        Used                              : 3859 MiB
        Free                              : 11500 MiB
    BAR1 Memory Usage
        Total                             : 256 MiB
        Used                              : 17 MiB
        Free                              : 239 MiB
    Compute Mode                          : Default
    Utilization
        Gpu                               : 0 %
        Memory                            : 0 %
        Encoder                           : 0 %
        Decoder                           : 0 %
    Encoder Stats
        Active Sessions                   : 0
        Average FPS                       : 0
        Average Latency                   : 0
    FBC Stats
        Active Sessions                   : 0
        Average FPS                       : 0
        Average Latency                   : 0
    Ecc Mode
        Current                           : Enabled
        Pending                           : Enabled
    ECC Errors
        Volatile
            SRAM Correctable              : 0
            SRAM Uncorrectable            : 0
            DRAM Correctable              : 0
            DRAM Uncorrectable            : 0
        Aggregate
            SRAM Correctable              : 0
            SRAM Uncorrectable            : 0
            DRAM Correctable              : 0
            DRAM Uncorrectable            : 0
    Retired Pages
        Single Bit ECC                    : 0
        Double Bit ECC                    : 0
        Pending Page Blacklist            : No
    Remapped Rows                         : N/A
    Temperature
        GPU Current Temp                  : 35 C
        GPU Shutdown Temp                 : 96 C
        GPU Slowdown Temp                 : 93 C
        GPU Max Operating Temp            : 85 C
        GPU Target Temperature            : N/A
        Memory Current Temp               : N/A
        Memory Max Operating Temp         : N/A
    Power Readings
        Power Management                  : Supported
        Power Draw                        : 16.57 W
        Power Limit                       : 70.00 W
        Default Power Limit               : 70.00 W
        Enforced Power Limit              : 70.00 W
        Min Power Limit                   : 60.00 W
        Max Power Limit                   : 70.00 W
    Clocks
        Graphics                          : 300 MHz
        SM                                : 300 MHz
        Memory                            : 405 MHz
        Video                             : 540 MHz
    Applications Clocks
        Graphics                          : 585 MHz
        Memory                            : 5001 MHz
    Default Applications Clocks
```

```
        Graphics                          : 585 MHz
        Memory                            : 5001 MHz
    Deferred Clocks
        Memory                            : N/A
    Max Clocks
        Graphics                          : 1590 MHz
        SM                                : 1590 MHz
        Memory                            : 5001 MHz
        Video                             : 1470 MHz
    Max Customer Boost Clocks
        Graphics                          : 1590 MHz
    Clock Policy
        Auto Boost                        : N/A
        Auto Boost Default                : N/A
    Voltage
        Graphics                          : N/A
    Fabric
        State                             : N/A
        Status                            : N/A
    Processes
        GPU instance ID                   : N/A
        Compute instance ID               : N/A
        Process ID                        : 2103065
            Type                          : C+G
            Name                          : Win11SV2_View87
            Used GPU Memory               : 3810 MiB
[root@vgpu ~]#
```

6.2.4. 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 GPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -q -i 1
GPU 00000000:C1:00.0
    Active vGPUs                      : 1
    vGPU ID                           : 3251634327
        VM ID                         : 2103066
        VM Name                       : Win11SV2_View87
        vGPU Name                     : GRID T4-4Q
        vGPU Type                     : 232
        vGPU UUID                     : afdcf724-1dd2-11b2-8534-624f22674b66
        Guest Driver Version          : 527.15
        License Status                : Licensed (Expiry: 2022-11-23 5:2:12 GMT)
        GPU Instance ID               : N/A
        Accounting Mode               : Disabled
        ECC Mode                      : Enabled
        Accounting Buffer Size        : 4000
        Frame Rate Limit              : 60 FPS
        PCI
            Bus Id                    : 00000000:02:04.0
        FB Memory Usage
            Total                     : 4096 MiB
            Used                      : 641 MiB
            Free                      : 3455 MiB
        Utilization
            Gpu                       : 0 %
            Memory                    : 0 %
            Encoder                   : 0 %
            Decoder                   : 0 %
        Encoder Stats
            Active Sessions           : 0
```

```
            Average FPS                       : 0
            Average Latency                   : 0
        FBC Stats
            Active Sessions                   : 0
            Average FPS                       : 0
            Average Latency                   : 0
[root@vgpu ~]#
```

6.2.5. 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 GPUs.

```
[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 ~]#
```

6.2.6. 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 GPUs.

```
[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 ~]#
```

6.2.7. 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 GPUs.

```
[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 ~]#
```

6.2.8. Monitoring MIG-backed vGPU activity

Note: MIG-backed vGPU activity **cannot** be monitored on GPUs based on the NVIDIA Ampere GPU architecture because the required hardware feature is not present on these GPUs.

To monitor MIG-backed vGPU activity across multiple vGPUs, run `nvidia-smi vgpu` with the `--gpm-metrics` *ID-list* option.

ID-list

A comma-separated list of integer IDs that specify the statistics to monitor as shown in the following table. The table also shows the name of the column in the command output under which the statistic is reported.

Statistic	ID	Column
Graphics activity	1	`gract`
Streaming multiprocessor (SM) activity	2	`smutil`
SM occupancy	3	`smocc`
Integer activity	4	`intutil`
Tensor activity	5	`mmaact`
Double-precision fused multiply-add (DFMA) tensor activity	6	`dfmat`
Half matrix multiplication and accumulation (HMMA) tensor activity	7	`hmmat`
Integer matrix multiplication and accumulation (IMMA) tensor activity	9	`immat`
Dynamic random-access memory (DRAM) activity	10	`dram`
Double-precision 64-bit floating-point (FP64) activity	11	`fp64`
Single-precision 32-bit floating-point (FP32) activity	12	`fp32`
Half-precision 16-bit FP16 activity	13	`fp16`

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 DRAM capacity will report 20%.

For each vGPU, the specified statistics are reported once every second.

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 GPUs.

The following example reports graphics activity, SM activity, SM occupancy, and integer activity for one vGPU VM that is powered on and within which one application is running.

```
[root@vgpu ~]# nvidia-smi vgpu --gpm-metrics 1,2,3,4
# gpu        vgpu    mig_id       gi_id        ci_id       gract    smutil
 smocc    intutil
# Idx          Id       Idx         Idx          Idx           %         %
 %          %
   0  3251634249          0           2            0           -         -
 -          -
   0  3251634249          0           2            0          99        97
 26         13
   0  3251634249          0           2            0          99        96
 23         13
   0  3251634249          0           2            0          99        97
 27         13
```

When no vGPUs are active on the hypervisor host, no activity is reported.

```
[root@vgpu ~]# nvidia-smi vgpu --gpm-metrics 1,2,3,4
# gpu        vgpu    mig_id       gi_id        ci_id       gract    smutil
 smocc    intutil
# Idx          Id       Idx         Idx          Idx           %         %
 %          %
   0           -         -           -            -           -         -
 -          -
   0           -         -           -            -           -         -
 -          -
   0           -         -           -            -           -         -
 -          -
```

6.2.9. 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 GPUs.

```
[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 00000000:40:00.0
```

```
    vGPU Type ID                            : 0xc
        Name                                : GRID M60-0Q
        Class                               : Quadro
        GPU Instance Profile ID             : N/A
        Max Instances                       : 16
        Max Instances Per VM                : 1
        Multi vGPU Exclusive                : False
        vGPU Exclusive Type                 : False
        vGPU Exclusive Size                 : False
        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                        : Quadro-Virtual-DWS,5.0;GRID-Virtual-
WS,2.0;GRID-Virtual-WS-Ext,2.0
    vGPU Type ID                            : 0xf
        Name                                : GRID M60-1Q
        Class                               : Quadro
        GPU Instance Profile ID             : N/A
        Max Instances                       : 8
        Max Instances Per VM                : 1
        Multi vGPU Exclusive                : False
        vGPU Exclusive Type                 : False
        vGPU Exclusive Size                 : False
        Device ID                           : 0x13f210de
        Sub System ID                       : 0x13f2114d
        FB Memory                           : 1024 MiB
        Display Heads                       : 4
        Maximum X Resolution                : 5120
        Maximum Y Resolution                : 2880
        Frame Rate Limit                    : 60 FPS
        GRID License                        : Quadro-Virtual-DWS,5.0;GRID-Virtual-
WS,2.0;GRID-Virtual-WS-Ext,2.0
    vGPU Type ID                            : 0x12
        Name                                : GRID M60-2Q
        Class                               : Quadro
        GPU Instance Profile ID             : N/A
        Max Instances                       : 4
        Max Instances Per VM                : 1
        Multi vGPU Exclusive                : False
        vGPU Exclusive Type                 : False
        vGPU Exclusive Size                 : False
…
[root@vgpu ~]#
```

6.2.10. 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 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 GPUs.

```
[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
```

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

```
nvidia-smi pmon
```

Chapter 7. Changing Scheduling Behavior for Time-Sliced vGPUs

NVIDIA GPUs 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.

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.

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.

Note: For time-sliced vGPUs with different amounts of frame buffer on the same physical GPU, only the best effort and equal share schedulers are supported. The fixed share scheduler is **not** supported.

By default, these schedulers impose a strict round-robin scheduling policy. When this policy is enforced, the schedulers maintain scheduling fairness by adjusting the time slice for each VM that is configured with NVIDIA vGPU. The strict round-robin scheduling policy ensures more consistent scheduling of the work for VMs that are configured with NVIDIA vGPU and restricts the impact of GPU-intensive applications running in one VM on applications running in other VMs.

Instead of a strict round-robin scheduling policy, you can ensure scheduling fairness by scheduling the work for the vGPU that has spent the least amount of time in the scheduled state. This behavior was the default scheduling behavior in NVIDIA AI Enterprise releases before 15.0.

When a strict round-robin scheduling policy is enforced, the adjustment to the time slice is based on the **scheduling frequency** and an **averaging factor**.

Scheduling frequency

The number of times per second that work for a specific vGPU is scheduled. The default scheduling frequency depends on the number of vGPUs that reside on the physical GPU:

- If fewer than eight vGPUs reside on the physical GPU, the default is 480 Hz.
- If eight or more vGPUs reside on the physical GPU, the default is 960 Hz.

Averaging factor

A number that determines the moving average of time-slice overshoots accrued for each vGPU. This average controls the strictness with which the scheduling frequency is enforced. A high value for the averaging factor enforces the scheduling frequency less strictly than a low value.

Deviations from the specified scheduling frequency occur because the actual amount of time that a scheduler allocates to a VM might exceed, or overshoot, the time slice specified for the VM. A scheduler enforces the scheduling frequency by shortening the next time slice for each vGPU VM to compensate for the accrued overshoot time of the VM.

To calculate the amount by which to shorten the next time slice for a vGPU VM, the scheduler maintains a running total of the accrued overshoot time for each vGPU VM. This amount is equal to the running total divided by the averaging factor that you

specify. The calculated amount is also subtracted from the accrued overshoot time. A high value for the averaging factor enforces the scheduling frequency less strictly by spreading the compensation for the accrued overshoot time over a longer period.

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, the averaging factor and scheduling frequency for schedulers with a strict round-robin scheduling policy, and the length of the time slice for schedulers **without** a strict round-robin scheduling policy.

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 a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor
`0x03`	Equal share scheduler **without** a strict round-robin scheduling policy and the default time slice length
`0x`*`AAFFF`*`001`	Equal share scheduler with a strict round-robin scheduling policy and a user-defined averaging factor *AA* and a user-defined scheduling frequency *FFF*
`0x00`*`TT`*`0003`	Equal share scheduler **without** a strict round-robin scheduling policy and with a user-defined time slice length *TT*
`0x11`	Fixed share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor **Note:** This value cannot be set for time-sliced vGPUs on a physical GPU in mixed-size mode.
`0x13`	Fixed share scheduler **without** a strict round-robin scheduling policy and with the default time slice length **Note:** This value cannot be set for time-sliced vGPUs on a physical GPU in mixed-size mode.
`0x`*`AAFFF`*`011`	Fixed share scheduler with a strict round-robin scheduling policy and a user-defined averaging factor *AA* and a user-defined scheduling frequency *FFF* **Note:** This value cannot be set for time-sliced vGPUs on a physical GPU in mixed-size mode.
`0x00`*`TT`*`0013`	Fixed share scheduler **without** a strict round-robin scheduling policy and with a user-defined time slice length *TT* **Note:** This value cannot be set for time-sliced vGPUs on a physical GPU in mixed-size mode.

The default time slice length and scheduling frequency depend on the maximum number of vGPUs per physical GPU allowed for the vGPU type.

Table 1. Default Time Slice Length and Scheduling Frequency by vGPU Density

Maximum Number of vGPUs	Default Time Slice Length	Default Scheduling Frequency
Less than or equal to 8	2 ms	480 Hz
Greater than 8	1 ms	960 Hz

AA

Two hexadecimal digits in the range 0x01 to 0x3C (decimal 1-60) that set the averaging factor for the equal share and fixed share schedulers with a strict round-robin scheduling policy.

The number of time slices over which the compensation for the accrued overshoot time is applied depends on the value of *AA*:

- If *AA* is 0x01, the compensation for the accrued overshoot time is applied in a single time slice.
- If *AA* is 0x3C, the compensation for the accrued overshoot time is spread over 60 (0x3C) time slices.
- If *AA* is 0x00, the default value of 33 is used.
- If *AA* is greater than 0x3C, the value is capped at 0x3C.

FFF

Three hexadecimal digits in the range 0x3F to 0x3C0 (decimal 63-960) that set the scheduling frequency for the equal share and fixed share schedulers with a strict round-robin scheduling policy. The time slice is the inverse of scheduling frequency. For example, a frequency of 0x3F (63 Hz) yields a time slice of 1/63 s, or 15.873 ms.

A value of 0x100 for *FFF* sets the scheduling frequency to 256.

If *FFF* is outside the range 0x3F to 0x3C0, the scheduling frequency is set as follows:

- If *FFF* is 000, the scheduling frequency is set to the default scheduling frequency for the vGPU type as listed in Table 1.
- If *FFF* is greater than 000 but less than 0x3F, the scheduling frequency is raised to 0x3F (decimal 63).
- If *FFF* is greater than 0x3C0, the scheduling frequency is capped at 0x3C0 (decimal 960).

TT

Two hexadecimal digits in the range 0x01 to 0x1E (decimal 1-30) 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 outside the range 01 to 1E, the length is set as follows:

- If *TT* is 00, the length is set to the default time slice length for the vGPU type as listed in Table 1.
- If *TT* is greater than 0x1E (decimal 30), the length is capped at 30 ms.

Examples

This example sets the vGPU scheduler to equal share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor.

```
RmPVMRL=0x01
```

This example sets the vGPU scheduler to equal share scheduler **without** a strict round-robin scheduling policy and with a time slice that is 3 ms long.

```
RmPVMRL=0x00030003
```

This example sets the vGPU scheduler to fixed share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor.

```
RmPVMRL=0x11
```

This example sets the vGPU scheduler to fixed share scheduler **without** a strict round-robin scheduling policy and with a time slice that is 24 (0x18) ms long.

```
RmPVMRL=0x00180011
```

This example sets the vGPU scheduler to equal share scheduler with a strict round-robin scheduling policy, an averaging factor of 60 (0x3C), and a scheduling frequency of 960 (0x3C0) Hz.

```
RmPVMRL=0x3c3c0001
```

This example sets the vGPU scheduler to fixed share scheduler with a strict round-robin scheduling policy, an averaging factor of 60 (0x3C), and a scheduling frequency of 960 (0x3C0) Hz.

```
RmPVMRL=0x3c3c0011
```

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

You can use the hypervisor's `dmesg` command to get the current time-sliced vGPU scheduling policy for all GPUs. Get this information 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.

2. Use the `dmesg` command to display messages from the kernel that contain the strings `NVRM` and `scheduler`.

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

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

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

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

This example gets the scheduling policy of the GPUs in a system in which the policy 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 by Using the `RmPVMRL` Registry Key

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.

2. Set the `RmPVMRL` registry key to the value that sets the GPU scheduling policy and the length of the time slice that you want.

   - On , 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.

3. Reboot your hypervisor host machine.

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

7.6. Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs by Using the RmPVMRL Registry Key

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.
2. 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.
   - On Red Hat Enterprise Linux KVM, 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)
```

3. Use the module parameter `NVreg_RegistryDwordsPerDevice` to set the `pci` and `RmPVMRL` registry keys for each GPU.

   - On Red Hat Enterprise Linux KVM, 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]"
```

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 Policy for All GPUs.

7.7. Restoring Default Time-Sliced vGPU Scheduler Settings by Using the `RmPVMRL` Registry Key

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.
2. Unset the `RmPVMRL` registry key.

   - On Red Hat Enterprise Linux KVM, 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="
     ```

3. Reboot your hypervisor host machine.

Chapter 8. Troubleshooting

This chapter describes basic troubleshooting steps for NVIDIA vGPU 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. Run the `vmkload_mod` command.

```
[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 (see Examining NVIDIA kernel driver output).

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` 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
2024-03-22T14:02:21.275Z| vmx| I120: DICT   pciPassthru0.virtualDev = "vmiop"
2024-03-22T14:02:21.344Z| vmx| I120: GetPluginPath testing /usr/lib64/vmware/plugin/
libvmx-vmiop.so
2024-03-22T14:02:21.344Z| vmx| I120: PluginLdr_LoadShared: Loaded shared plugin
 libvmx-vmiop.so from /usr/lib64/vmware/plugin/libvmx-vmiop.so
2024-03-22T14:02:21.344Z| vmx| I120: VMIOP: Loaded plugin libvmx-
vmiop.so:VMIOP_InitModule
2024-03-22T14:02:21.359Z| vmx| I120: VMIOP: Initializing plugin vmiop-display
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: gpu-pci-id : 0000:04:00.0
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: vgpu_type : quadro
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: Framebuffer: 0x74000000
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: Virtual Device Id: 0x11B0:0x101B
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: ######## vGPU Manager Information:
 ########
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: Driver Version: 550.54.16
2024-03-22T14:02:21.365Z| vmx| I120: vmiop_log: VGX Version: 17.1
2024-03-22T14:02:21.445Z| vmx| I120: vmiop_log: Init frame copy engine: syncing...
2024-03-22T14:02:37.031Z| vthread-12| I120: vmiop_log: ######## Guest NVIDIA Driver
 Information: ########
2024-03-22T14:02:37.031Z| vthread-12| I120: vmiop_log: Driver Version: 551.78
2024-03-22T14:02:37.031Z| vthread-12| I120: vmiop_log: VGX Version: 17.1
2024-03-22T14:02:37.093Z| vthread-12| I120: vmiop_log: Clearing BAR1 mapping
2023-03-25T23:39:55.726Z| 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 VMware ESXi host shell.

Chapter 9. Additional Information

Additional information about the software components of NVIDIA AI Enterprise is available in the documentation for these entities.

Infrastructure and Workload Management Components

- NVIDIA virtual GPU software
- NVIDIA GPU Operator
- NVIDIA Network Operator
- NVIDIA Base Command™ Manager Essentials

Tools for AI Development and Use Cases

- NVIDIA Clara Parabricks
- NVIDIA DeepStream
- NVIDIA DGL
- NVIDIA Maxine
- NVIDIA Modulus
- MONAI (Medical Open Network for Artificial Intelligence) Enterprise
- NVIDIA NeMo™
- NVIDIA NIM
- PyTorch
- NVIDIA RAPIDS
- NVIDIA RAPIDS Accelerator for Apache Spark
- NVIDIA Riva
- TAO Toolkit
- NVIDIA TensorRT
- TensorFlow
- NVIDIA Triton Inference Server
- NVIDIA Triton Management Service

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 A800 PCIe 80GB, NVIDIA A800 PCIe 80GB Liquid Cooled, and NVIDIA AX800 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

The virtual GPU types for the NVIDIA A800 PCIe 80GB, NVIDIA A800 PCIe 80GB liquid cooled, and NVIDIA AX800 GPUs are identical.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A800 PCIe 80GB, NVIDIA A800 PCIe 80GB Liquid Cooled, and NVIDIA AX800

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A800D-7-80C	81920	1	7	7	MIG 7g.80gb

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A800D-4-40C	40960	1	4	4	MIG 4g.40gb
A800D-3-40C	40960	2	3	3	MIG 3g.40gb
A800D-2-20C	20480	3	2	2	MIG 2g.20gb
A800D-1-20C	20480	4	1	1	MIG 1g.20gb
A800D-1-10C	10240	7	1	1	MIG 1g.10gb
A800D-1-10CME	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A800 PCIe 80GB, NVIDIA A800 PCIe 80GB Liquid Cooled, and NVIDIA AX800

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A800D-80C	81920	1	1	3840×2400[1]	1
A800D-40C	40960	2	2	3840×2400[1]	1
A800D-20C	20480	4	4	3840×2400[1]	1
A800D-16C	16384	5	4	3840×2400[1]	1
A800D-10C	10240	8	8	3840×2400[1]	1
A800D-8C	8192	10	8	3840×2400[1]	1
A800D-4C	4096	20	16	3840×2400[1]	1

A.2. NVIDIA A800 PCIe 40GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA A800 PCIe 40GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A800-7-40C	40960	1	7	7	MIG 7g.40gb
A800-4-20C	20480	1	4	4	MIG 4g.20gb
A800-3-20C	20480	2	3	3	MIG 3g.20gb
A800-2-10C	10240	3	2	2	MIG 2g.10gb
A800-1-10C	10240	4	1	1	MIG 1g.10gb
A800-1-5C	5120	7	1	1	MIG 1g.5gb
A800-1-5CME	5120	1	1	1	MIG 1g.5gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A800 PCIe 40GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A800-40C	40960	1	1	3840×2400[1]	1
A800-20C	20480	2	2	3840×2400[1]	1
A800-10C	10240	4	4	3840×2400[1]	1
A800-8C	8192	5	4	3840×2400[1]	1
A800-5C	5120	8	8	3840×2400[1]	1
A800-4C	4096	10	8	3840×2400[1]	1

A.3. NVIDIA A800 HGX Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA A800 HGX 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A800DX-7-80C	81920	1	7	7	MIG 7g.80gb
A800DX-4-40C	40960	1	4	4	MIG 4g.40gb
A800DX-3-40C	40960	2	3	3	MIG 3g.40gb
A800DX-2-20C	20480	3	2	2	MIG 2g.20gb
A800DX-1-20C	20480	4	1	1	MIG 1g.20gb

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A800DX-1-10C	10240	7	1	1	MIG 1g.10gb
A800DX-1-10CME	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A800 HGX 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A800DX-80C	81920	1	1	3840×2400[1]	1
A800DX-40C	40960	2	2	3840×2400[1]	1
A800DX-20C	20480	4	4	3840×2400[1]	1
A800DX-16C	16384	5	4	3840×2400[1]	1
A800DX-10C	10240	8	8	3840×2400[1]	1
A800DX-8C	8192	10	8	3840×2400[1]	1
A800DX-4C	4096	20	16	3840×2400[1]	1

A.4. NVIDIA A100 PCIe 40GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A100-40C	40960	1	1	3840×2400[1]	1
A100-20C	20480	2	2	3840×2400[1]	1
A100-10C	10240	4	4	3840×2400[1]	1
A100-8C	8192	5	4	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A100-5C	5120	8	8	3840×2400[1]	1
A100-4C	4096	10	8	3840×2400[1]	1

A.5. NVIDIA A100 HGX 40GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A100X-40C	40960	1	1	3840×2400[1]	1
A100X-20C	20480	2	2	3840×2400[1]	1
A100X-10C	10240	4	4	3840×2400[1]	1
A100X-8C	8192	5	4	3840×2400[1]	1
A100X-5C	5120	8	8	3840×2400[1]	1
A100X-4C	4096	10	8	3840×2400[1]	1

A.6. NVIDIA A100 PCIe 80GB, NVIDIA A100 PCIe 80GB Liquid Cooled and NVIDIA A100X Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A100D-80C	81920	1	1	3840×2400[1]	1
A100D-40C	40960	2	2	3840×2400[1]	1
A100D-20C	20480	4	4	3840×2400[1]	1
A100D-16C	16384	5	4	3840×2400[1]	1
A100D-10C	10240	8	8	3840×2400[1]	1
A100D-8C	8192	10	8	3840×2400[1]	1
A100D-4C	4096	20	16	3840×2400[1]	1

A.7. NVIDIA A100 HGX 80GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A100DX-7-80C	81920	1	7	7	MIG 7g.80gb
A100DX-4-40C	40960	1	4	4	MIG 4g.40gb
A100DX-3-40C	40960	2	3	3	MIG 3g.40gb
A100DX-2-20C	20480	3	2	2	MIG 2g.20gb
A100DX-1-20C	20480	4	1	1	MIG 1g.20gb
A100DX-1-10C	10240	7	1	1	MIG 1g.10gb
A100DX-1-10CME	10240	1	1	1	MIG 1g.10gb+me

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

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A100DX-80C	81920	1	1	3840×2400[1]	1
A100DX-40C	40960	2	2	3840×2400[1]	1
A100DX-20C	20480	4	4	3840×2400[1]	1
A100DX-16C	16384	5	4	3840×2400[1]	1
A100DX-10C	10240	8	8	3840×2400[1]	1
A100DX-8C	8192	10	8	3840×2400[1]	1
A100DX-4C	4096	20	16	3840×2400[1]	1

A.8. NVIDIA A40 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA A40

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A40-48C	49152	1	1	3840×2400[1]	1
A40-24C	24576	2	2	3840×2400[1]	1
A40-16C	16384	3	2	3840×2400[1]	1
A40-12C	12288	4	4	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A40-8C	8192	6	4	3840×2400[1]	1
A40-6C	6144	8	8	3840×2400[1]	1
A40-4C	4096	12[2]	8	3840×2400[1]	1

A.9. NVIDIA A30, NVIDIA A30X, and NVIDIA A30 Liquid Cooled Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

The virtual GPU types for the NVIDIA A30, NVIDIA A30X, and NVIDIA A30 Liquid Cooled GPUs are identical.

MIG-Backed C-Series Virtual GPU Types for NVIDIA A30, NVIDIA A30X, and NVIDIA A30 Liquid Cooled

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A30-4-24C	24576	1	4	4	MIG 4g.24gb
A30-2-12C	12288	2	2	2	MIG 2g.12gb
A30-2-12CME	12288	1	2	2	MIG 2g.12gb+me
A30-1-6C	6144	4	1	1	MIG 1g.6gb

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
A30-1-6CME	6144	1	1	1	MIG 1g.6gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA A30, NVIDIA A30X, and NVIDIA A30 Liquid Cooled

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A30-24C	24576	1	1	3840×2400[1]	1
A30-12C	12288	2	2	3840×2400[1]	1
A30-8C	8192	3	2	3840×2400[1]	1
A30-6C	6144	4	4	3840×2400[1]	1
A30-4C	4096	6	4	3840×2400[1]	1

A.10. NVIDIA A16 Virtual GPU Types

Physical GPUs per board: 4

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA A16

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A16-16C	16384	1	1	3840×2400[1]	1
A16-8C	8192	2	2	3840×2400[1]	1
A16-4C	4096	4	4	3840×2400[1]	1

A.11. NVIDIA A10 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA A10

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A10-24C	24576	1	1	3840×2400[1]	1
A10-12C	12288	2	2	3840×2400[1]	1
A10-8C	8192	3	2	3840×2400[1]	1
A10-6C	6144	4	4	3840×2400[1]	1
A10-4C	4096	6	4	3840×2400[1]	1

A.12. NVIDIA H100 PCIe 94GB (H100 NVL) Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H100 PCIe 94GB (H100 NVL)

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H100L-7-94C	96246	1	7	7	MIG 7g.94gb
H100L-4-47C	48128	1	4	4	MIG 4g.47gb
H100L-3-47C	48128	2	3	3	MIG 3g.47gb
H100L-2-24C	24672	3	2	2	MIG 2g.24gb
H100L-1-24C	24672	4	1	1	MIG 1g.24gb
H100L-1-12C	12288	7	1	1	MIG 1g.12gb
H100L-1-12CME	12288	1	1	1	MIG 1g.12gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H100 PCIe 94GB (H100 NVL)

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H100L-94C	96246	1	1	3840×2400[1]	1
H100L-47C	48128	2	2	3840×2400[1]	1
H100L-23C	23552	4	4	3840×2400[1]	1
H100L-15C	15360	6	4	3840×2400[1]	1
H100L-11C	11264	8	8	3840×2400[1]	1
H100L-6C	6,144	15	8	3840×2400[1]	1
H100L-4C	4,096	23	16	3840×2400[1]	1

A.13. NVIDIA H100 SXM5 94GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H100 SXM5 94GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H100XL-7-94C	96246	1	7	7	MIG 7g.94gb
H100XL-4-47C	48128	1	4	4	MIG 4g.47gb
H100XL-3-47C	48128	2	3	3	MIG 3g.47gb
H100XL-2-24C	24672	3	2	2	MIG 2g.24gb

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H100XL-1-24C	24672	4	1	1	MIG 1g.24gb
H100XL-1-12C	12288	7	1	1	MIG 1g.12gb
H100XL-1-12CME	12288	1	1	1	MIG 1g.12gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H100 SXM5 94GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H100XL-94C	96246	1	1	3840×2400[1]	1
H100XL-47C	48128	2	2	3840×2400[1]	1
H100XL-23C	23552	4	4	3840×2400[1]	1
H100XL-15C	15360	6	4	3840×2400[1]	1
H100XL-11C	11264	8	8	3840×2400[1]	1
H100XL-6C	6,144	15	8	3840×2400[1]	1
H100XL-4C	4,096	23	16	3840×2400[1]	1

A.14. NVIDIA H100 PCIe 80GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H100 PCIe 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H100-7-80C	81920	1	7	7	MIG 7g.80gb
H100-4-40C	40960	1	4	4	MIG 4g.40gb
H100-3-40C	40960	2	3	3	MIG 3g.40gb
H100-2-20C	20480	3	2	2	MIG 2g.20gb
H100-1-20C	20480	4	1	1	MIG 1g.20gb
H100-1-10C	10240	7	1	1	MIG 1g.10gb
H100-1-10CME	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H100 PCIe 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H100-80C	81920	1	1	3840×2400[1]	1
H100-40C	40960	2	2	3840×2400[1]	1
H100-20C	20480	4	4	3840×2400[1]	1
H100-16C	16384	5	4	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H100-10C	10240	8	8	3840×2400[1]	1
H100-8C	8192	10	8	3840×2400[1]	1
H100-5C	5120	16	16	3840×2400[1]	1
H100-4C	4096	20	16	3840×2400[1]	1

A.15. NVIDIA H100 SXM5 80GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H100 SXM5 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H100XM-7-80C	81920	1	7	7	MIG 7g.80gb
H100XM-4-40C	40960	1	4	4	MIG 4g.40gb
H100XM-3-40C	40960	2	3	3	MIG 3g.40gb
H100XM-2-20C	20480	3	2	2	MIG 2g.20gb
H100XM-1-20C	20480	4	1	1	MIG 1g.20gb
H100XM-1-10C	10240	7	1	1	MIG 1g.10gb
H100XM-1-10CME	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H100 SXM5 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H100XM-80C	81920	1	1	3840×2400[1]	1
H100XM-40C	40960	2	2	3840×2400[1]	1
H100XM-20C	20480	4	4	3840×2400[1]	1
H100XM-16C	16384	5	4	3840×2400[1]	1
H100XM-10C	10240	8	8	3840×2400[1]	1
H100XM-8C	8192	10	8	3840×2400[1]	1
H100XM-5C	5120	16	16	3840×2400[1]	1
H100XM-4C	4096	20	16	3840×2400[1]	1

A.16. NVIDIA H100 SXM5 64GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H100 SXM5 64GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H100XS-7-64C	65536	1	7	7	MIG 7g.64gb
H100XS-4-32C	32768	1	4	4	MIG 4g.32gb
H100XS-3-32C	32768	2	3	3	MIG 3g.32gb
H100XS-2-16C	16384	3	2	2	MIG 2g.16gb
H100XS-1-16C	16384	4	1	1	MIG 1g.16gb
H100XS-1-8C	8192	7	1	1	MIG 1g.8gb
H100XS-1-8CME	8192	1	1	1	MIG 1g.8gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H100 SXM5 64GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H100XS-64C	65536	1	1	3840×2400[1]	1
H100XS-32C	32768	2	2	3840×2400[1]	1
H100XS-16C	16384	4	4	3840×2400[1]	1
H100XS-8C	8192	8	8	3840×2400[1]	1
H100XS-4C	4096	16	16	3840×2400[1]	1

A.17. NVIDIA H800 PCIe 94GB (H800 NVL) Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H800 PCIe 94GB (H800 NVL)

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H800L-7-94C	96246	1	7	7	MIG 7g.94gb
H800L-4-47C	48128	1	4	4	MIG 4g.47gb
H800L-3-47C	48128	2	3	3	MIG 3g.47gb
H800L-2-24C	24672	3	2	2	MIG 2g.24gb
H800L-1-24C	24672	4	1	1	MIG 1g.24gb
H800L-1-12C	12288	7	1	1	MIG 1g.12gb
H800L-1-12CME	12288	1	1	1	MIG 1g.12gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H800 PCIe 94GB (H800 NVL)

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H800L-94C	96246	1	1	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H800L-47C	48128	2	2	3840×2400[1]	1
H800L-23C	23552	4	4	3840×2400[1]	1
H800L-15C	15360	6	4	3840×2400[1]	1
H800L-11C	11264	8	8	3840×2400[1]	1
H800L-6C	6,144	15	8	3840×2400[1]	1
H800L-4C	4,096	23	16	3840×2400[1]	1

A.18. NVIDIA H800 PCIe 80GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H800 PCIe 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H800-7-80C	81920	1	7	7	MIG 7g.80gb
H800-4-40C	40960	1	4	4	MIG 4g.40gb
H800-3-40C	40960	2	3	3	MIG 3g.40gb
H800-2-20C	20480	3	2	2	MIG 2g.20gb
H800-1-20C	20480	4	1	1	MIG 1g.20gb

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H800-1-10C	10240	7	1	1	MIG 1g.10gb
H800-1-10CME	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H800 PCIe 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H800-80C	81920	1	1	3840×2400[1]	1
H800-40C	40960	2	2	3840×2400[1]	1
H800-20C	20480	4	4	3840×2400[1]	1
H800-16C	16384	5	4	3840×2400[1]	1
H800-10C	10240	8	8	3840×2400[1]	1
H800-8C	8192	10	8	3840×2400[1]	1
H800-5C	5120	16	16	3840×2400[1]	1
H800-4C	4096	20	16	3840×2400[1]	1

A.19. NVIDIA H800 SXM5 80GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

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

MIG-Backed C-Series Virtual GPU Types for NVIDIA H800 SXM5 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Slices per vGPU	Compute Instances per vGPU	Corresponding GPU Instance Profile
H800XM-7-80C	81920	1	7	7	MIG 7g.80gb
H800XM-4-40C	40960	1	4	4	MIG 4g.40gb
H800XM-3-40C	40960	2	3	3	MIG 3g.40gb
H800XM-2-20C	20480	3	2	2	MIG 2g.20gb
H800XM-1-20C	20480	4	1	1	MIG 1g.20gb
H800XM-1-10C	10240	7	1	1	MIG 1g.10gb
H800XM-1-10CME	10240	1	1	1	MIG 1g.10gb+me

Time-Sliced C-Series Virtual GPU Types for NVIDIA H800 SXM5 80GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H800XM-80C	81920	1	1	3840×2400[1]	1
H800XM-40C	40960	2	2	3840×2400[1]	1
H800XM-20C	20480	4	4	3840×2400[1]	1
H800XM-16C	16384	5	4	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
H800XM-10C	10240	8	8	3840×2400[1]	1
H800XM-8C	8192	10	8	3840×2400[1]	1
H800XM-5C	5120	16	16	3840×2400[1]	1
H800XM-4C	4096	20	16	3840×2400[1]	1

A.20. NVIDIA L40 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA L40

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L40-48C	49152	1	1	3840×2400[1]	1
L40-24C	24576	2	2	3840×2400[1]	1
L40-16C	16384	3	2	3840×2400[1]	1
L40-12C	12288	4	4	3840×2400[1]	1
L40-8C	8192	6	4	3840×2400[1]	1
L40-6C	6144	8	8	3840×2400[1]	1
L40-4C	4096	12[2]	8	3840×2400[1]	1

A.21. NVIDIA L40S Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA L40S

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L40S-48C	49152	1	1	3840×2400[1]	1
L40S-24C	24576	2	2	3840×2400[1]	1
L40S-16C	16384	3	2	3840×2400[1]	1
L40S-12C	12288	4	4	3840×2400[1]	1
L40S-8C	8192	6	4	3840×2400[1]	1
L40S-6C	6144	8	8	3840×2400[1]	1
L40S-4C	4096	12[2]	8	3840×2400[1]	1

A.22. NVIDIA L20 and NVIDIA L20 Liquid Cooled Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

The virtual GPU types for the NVIDIA L20 and NVIDIA L20 liquid cooled GPUs are identical.

C-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L20-48C	49152	1	1	3840×2400[1]	1
L20-24C	24576	2	2	3840×2400[1]	1
L20-16C	16384	3	2	3840×2400[1]	1
L20-12C	12288	4	4	3840×2400[1]	1
L20-8C	8192	6	4	3840×2400[1]	1
L20-6C	6144	8	8	3840×2400[1]	1
L20-4C	4096	12[2]	8	3840×2400[1]	1

A.23. NVIDIA L4 Virtual GPU Types

Physical GPUs per board: 1

C-Series Virtual GPU Types for NVIDIA L4

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L4-24C	24576	1	1	3840×2400[1]	1
L4-12C	12288	2	2	3840×2400[1]	1
L4-8C	8192	3	2	3840×2400[1]	1
L4-6C	6144	4	4	3840×2400[1]	1
L4-4C	4096	6	4	3840×2400[1]	1

A.24. NVIDIA L2 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA L2

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L2-24C	24576	1	1	3840×2400[1]	1
L2-12C	12288	2	2	3840×2400[1]	1
L2-8C	8192	3	2	3840×2400[1]	1
L2-6C	6144	4	4	3840×2400[1]	1
L2-4C	4096	6	4	3840×2400[1]	1

A.25. NVIDIA RTX 6000 Ada Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA RTX 6000 Ada

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTX 6000 Ada-48C	49152	1	1	3840×2400[1]	1
RTX 6000 Ada-24C	24576	2	2	3840×2400[1]	1
RTX 6000 Ada-16C	16384	3	2	3840×2400[1]	1
RTX 6000 Ada-12C	12288	4	4	3840×2400[1]	1
RTX 6000 Ada-8C	8192	6	4	3840×2400[1]	1
RTX 6000 Ada-6C	6144	8	8	3840×2400[1]	1
RTX 6000 Ada-4C	4096	12[2]	8	3840×2400[1]	1

A.26. NVIDIA RTX 5880 Ada Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA RTX 5880 Ada

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTX 5880 Ada-48C	49152	1	1	3840×2400[1]	1
RTX 5880 Ada-24C	24576	2	2	3840×2400[1]	1
RTX 5880 Ada-16C	16384	3	2	3840×2400[1]	1
RTX 5880 Ada-12C	12288	4	4	3840×2400[1]	1
RTX 5880 Ada-8C	8192	6	4	3840×2400[1]	1
RTX 5880 Ada-6C	6144	8	8	3840×2400[1]	1
RTX 5880 Ada-4C	4096	12[2]	8	3840×2400[1]	1

A.27. NVIDIA RTX 5000 Ada Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA RTX 5000 Ada

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTX 5000 Ada-32C	32768	1	1	3840×2400[1]	1
RTX 5000 Ada-16C	16384	2	2	3840×2400[1]	1
RTX 5000 Ada-8C	8192	4	4	3840×2400[1]	1
RTX 5000 Ada-4C	4096	8	8	3840×2400[1]	1

A.28. NVIDIA RTX A6000 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA RTX A6000

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA6000-48C	49152	1	1	3840×2400[1]	1
RTXA6000-24C	24576	2	2	3840×2400[1]	1
RTXA6000-16C	16384	3	2	3840×2400[1]	1
RTXA6000-12C	12288	4	4	3840×2400[1]	1
RTXA6000-8C	8192	6	4	3840×2400[1]	1
RTXA6000-6C	6144	8	8	3840×2400[1]	1
RTXA6000-4C	4096	12[2]	8	3840×2400[1]	1

A.29. NVIDIA RTX A5500 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

C-Series Virtual GPU Types for NVIDIA RTX A5500

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5500-24C	24576	1	1	3840×2400[1]	1
RTXA5500-12C	12288	2	2	3840×2400[1]	1
RTXA5500-8C	8192	3	2	3840×2400[1]	1
RTXA5500-6C	6144	4	4	3840×2400[1]	1
RTXA5500-4C	4096	6	4	3840×2400[1]	1

A.30. NVIDIA RTX A5000 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for NVIDIA RTX A5000

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5000-24C	24576	1	1	3840×2400[1]	1
RTXA5000-12C	12288	2	2	3840×2400[1]	1
RTXA5000-8C	8192	3	2	3840×2400[1]	1
RTXA5000-6C	6144	4	4	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5000-4C	4096	6	4	3840×2400[1]	1

A.31. Tesla T4 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla T4

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
T4-16C	16384	1	3840×2400[1]	1
T4-8C	8192	2	3840×2400[1]	1
T4-4C	4096	4	3840×2400[1]	1

A.32. Tesla V100 SXM2 Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla V100 SXM2

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
V100X-16C	16384	1	3840×2400[1]	1
V100X-8C	8192	2	3840×2400[1]	1
V100X-4C	4096	4	3840×2400[1]	1

A.33. Tesla V100 SXM2 32GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla V100 SXM2 32GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
V100DX-32C	32768	1	3840×2400[1]	1
V100DX-16C	16384	2	3840×2400[1]	1
V100DX-8C	8192	4	3840×2400[1]	1
V100DX-4C	4096	8	3840×2400[1]	1

A.34. Tesla V100 PCIe Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla V100 PCIe

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
V100-16C	16384	1	3840×2400[1]	1
V100-8C	8192	2	3840×2400[1]	1
V100-4C	4096	4	3840×2400[1]	1

A.35. Tesla V100 PCIe 32GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla V100 PCIe 32GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
V100D-32C	32768	1	3840×2400[1]	1
V100D-16C	16384	2	3840×2400[1]	1
V100D-8C	8192	4	3840×2400[1]	1
V100D-4C	4096	8	3840×2400[1]	1

A.36. Tesla V100S PCIe 32GB Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla V100S PCIe 32GB

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
V100S-32C	32768	1	3840×2400[1]	1
V100S-16C	16384	2	3840×2400[1]	1
V100S-8C	8192	4	3840×2400[1]	1
V100S-4C	4096	8	3840×2400[1]	1

A.37. Tesla V100 FHHL Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Tesla V100 FHHL

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

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
V100L-16C	Training Workloads	16384	1	1	3840×2400[1]	1
V100L-8C	Training Workloads	8192	2	2	3840×2400[1]	1
V100L-4C	Inference Workloads	4096	4	4	3840×2400[1]	1

A.38. Quadro RTX 8000 Passive Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Quadro RTX 8000 Passive

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads

- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
RTX8000P-48C	49152	1	3840×2400[1]	1
RTX8000P-24C	24576	2	3840×2400[1]	1
RTX8000P-16C	16384	3	3840×2400[1]	1
RTX8000P-12C	12288	4	3840×2400[1]	1
RTX8000P-8C	8192	6	3840×2400[1]	1
RTX8000P-6C	6144	8	3840×2400[1]	1
RTX8000P-4C	4096	8[2]	3840×2400[1]	1

A.39. Quadro RTX 6000 Passive Virtual GPU Types

Physical GPUs per board: 1

The maximum number of vGPUs per board is the product of the maximum number of vGPUs per GPU and the number of physical GPUs per board.

This GPU does **not** support mixed-size mode.

C-Series Virtual GPU Types for Quadro RTX 6000 Passive

Intended use cases:

- vGPUs with more than 4096 MB of frame buffer: Training Workloads
- vGPUs with 4096 MB of frame buffer: Inference Workloads

Required license edition: vCS or vWS

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
RTX6000P-24C	24576	1	3840×2400[1]	1
RTX6000P-12C	12288	2	3840×2400[1]	1

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU	Maximum Display Resolution	Virtual Displays per vGPU
RTX6000P-8C	8192	3	3840×2400[1]	1
RTX6000P-6C	6144	4	3840×2400[1]	1
RTX6000P-4C	4096	6	3840×2400[1]	1

Appendix B. vGPU Placements for GPUs in Mixed-Size Mode

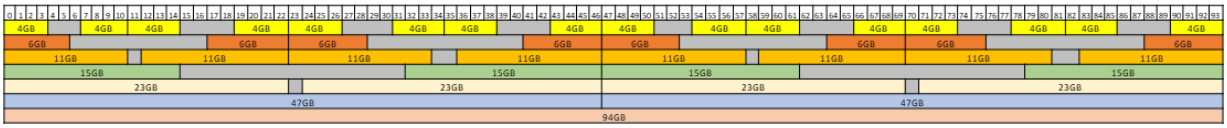
The vGPU placements that a GPU in mixed-size mode supports depend on the total amount of frame buffer that the GPU has.

B.1. vGPU Placements for GPUs with 94 GB of Frame Buffer

Placement region size: 94

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
96246	94	1	1	0
48128	47	2	2	0, 47
23552	23	4	4	0, 24, 47, 71
15360	15	6	4	0, 32, 47, 79
11264	11	8	8	0, 12, 23, 36, 47, 59, 70, 83
6,144	6	15	8	0, 17, 23, 41, 47, 64, 70, 88
4,096	4	23	16	0, 7, 11, 19, 23, 31, 35, 43, 47, 54, 58, 66, 70, 78, 82, 90

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 94 GB of frame buffer in mixed-size mode.

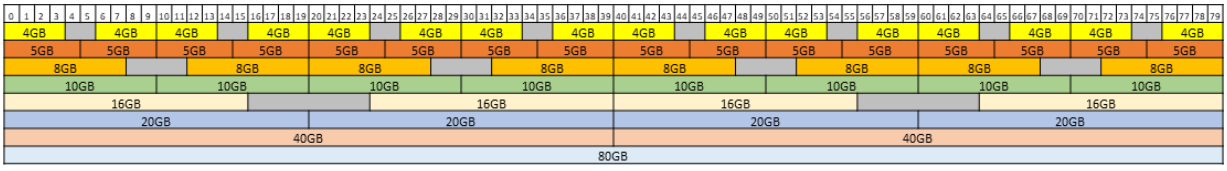


B.2. vGPU Placements for GPUs with 80 GB of Frame Buffer

Placement region size: 80

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
81920	80	1	1	0
40960	40	2	2	0, 40
20480	20	4	4	0, 20, 40, 60
16384	16	5	4	0, 24, 40, 64
10240	10	8	8	0, 10, 20, 30, 40, 50, 60, 70
8192	8	10	8	0, 12, 20, 32, 40, 52, 60, 72
5120	5	16	16	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75
4096	4	20	16	0, 6, 10, 16, 20, 26, 30, 36, 40, 46, 50, 56, 60, 66, 70, 76

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 80 GB of frame buffer in mixed-size mode.

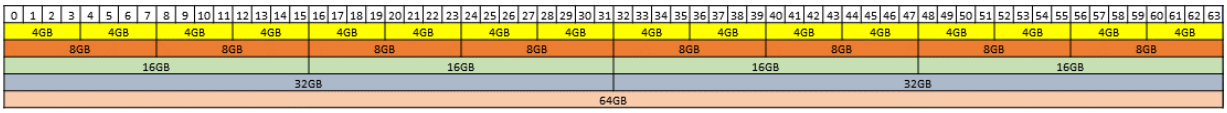


B.3. vGPU Placements for GPUs with 64 GB of Frame Buffer

Placement region size: 64

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
65536	64	1	1	0
32768	32	2	2	0, 32
16384	16	4	4	0, 16, 32, 48
8192	8	8	8	0, 8, 16, 24, 32, 40, 48, 56
4096	4	16	16	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 46, 60

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 64 GB of frame buffer in mixed-size mode.



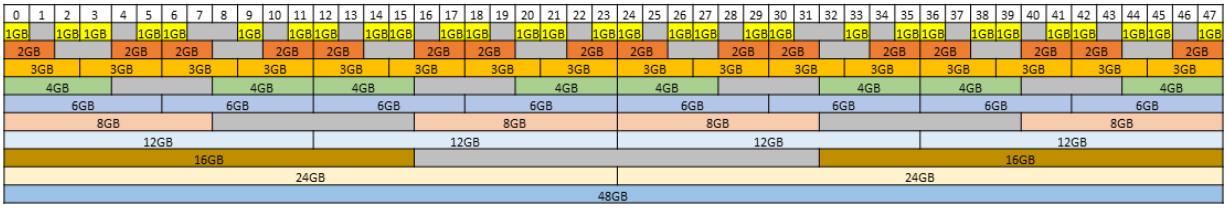
B.4. vGPU Placements for GPUs with 48 GB of Frame Buffer

Placement region size: 48

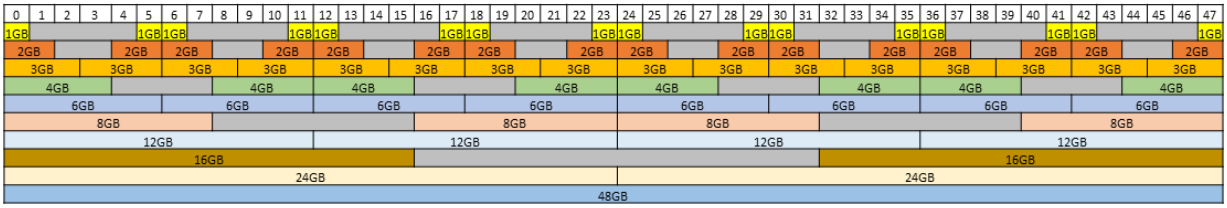
Note: When in mixed-size mode, the maximum number of vGPUs with 1024 MB of frame buffer allowed on GPUs based on the Ada Lovelace GPU architecture is lower than for other GPU architectures. As a result, the supported placement IDs for these vGPUs on GPUs based on the Ada Lovelace GPU architecture are different than for other GPU architectures.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
49152	48	1	1	0
24576	24	2	2	0, 24
16384	16	3	2	0, 32
12288	12	4	4	0, 12, 24, 36
8192	8	6	4	0, 16, 24, 40
6144	6	8	8	0, 6, 12, 18, 24, 30, 36, 42
4096	4	12	8	0, 8, 12, 20, 24, 32, 36, 44
3072	3	16	16	0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45
2048	2	24	16	0, 4, 6, 10, 12, 16, 18, 22, 24, 28, 30, 34, 36, 40, 42, 46
1024	1	32	GPU architectures **except** Ada Lovelace: 30	GPU architectures **except** Ada Lovelace: 0, 2, 3, 5, 6, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 33, 35, 36, 38, 39, 41, 42, 44, 45, 47
			Ada Lovelace GPU architecture: 16	Ada Lovelace GPU architecture: 0, 5, 6, 11, 12, 17, 18, 23, 24, 29, 30, 35, 36, 41, 42, 47

The following diagram shows the supported placements for each size of vGPU on a GPU based on a GPU architecture **except** Ada Lovelace with a total of 48 GB of frame buffer in mixed-size mode.



The following diagram shows the supported placements for each size of vGPU on a GPU based on the Ada Lovelace GPU architecture with a total of 48 GB of frame buffer in mixed-size mode.

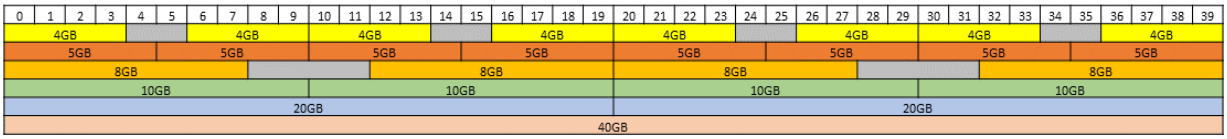


B.5. vGPU Placements for GPUs with 40 GB of Frame Buffer

Placement region size: 40

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
40960	40	1	1	0
20480	20	2	2	0, 20
10240	10	4	4	0, 10, 20, 30
8192	8	5	4	0, 12, 20, 32
5120	5	8	8	0, 5, 10, 15, 20, 25, 30, 35
4096	4	10	8	0, 6, 10, 16, 20, 26, 30, 36

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 40 GB of frame buffer in mixed-size mode.

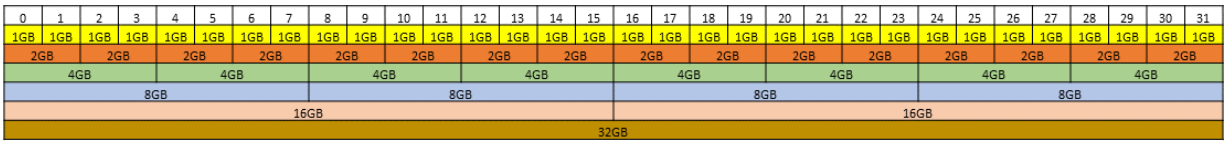


B.6. vGPU Placements for GPUs with 32 GB of Frame Buffer

Placement region size: 32

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
32768	32	1	1	0
16384	16	2	2	0, 16
8192	8	4	4	0, 8, 16, 24
4096	4	8	8	0, 4, 8, 12, 16, 20, 24, 28
2048	2	16	16	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
1024	1	32	32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 32 GB of frame buffer in mixed-size mode.

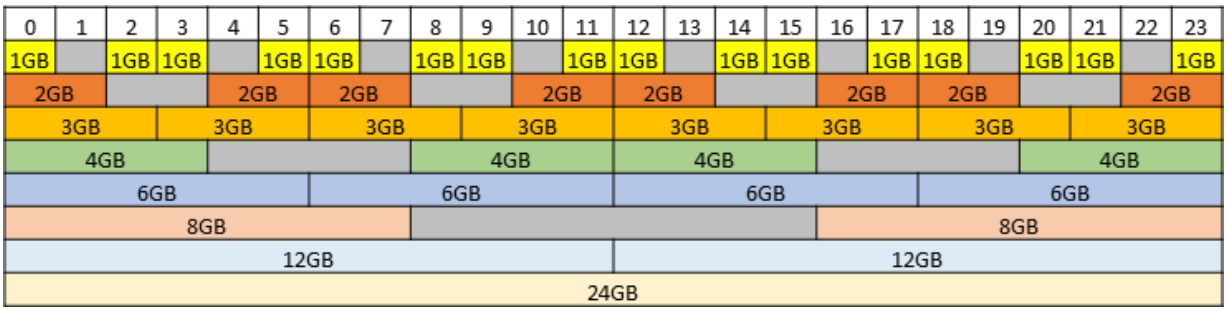


B.7. vGPU Placements for GPUs with 24 GB of Frame Buffer

Placement region size: 24

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
24576	24	1	1	0
12288	12	2	2	0, 12
8192	8	3	2	0, 16
6144	6	4	4	0, 6, 12, 18
4096	4	6	4	0, 8, 12, 20
3072	3	8	8	0, 3, 6, 9, 12, 15, 18, 21
2048	2	12	8	0, 4, 6, 10, 12, 16, 18, 22
1024	1	24	16	0, 2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 24 GB of frame buffer in mixed-size mode.



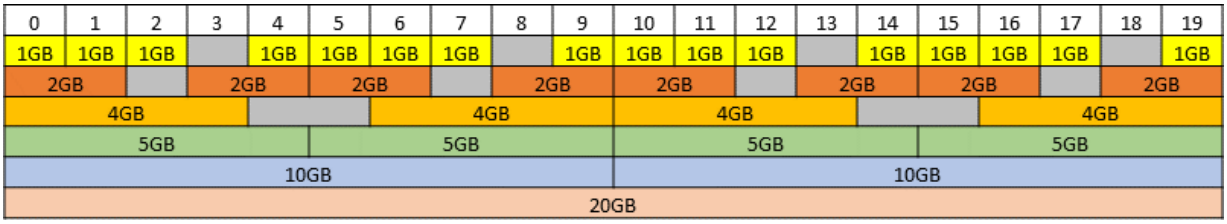
B.8. vGPU Placements for GPUs with 20 GB of Frame Buffer

Placement region size: 20

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
20480	20	1	1	0
10240	10	2	2	0, 10
5120	5	4	4	0, 5, 10, 15

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
4096	4	5	4	0, 6, 10, 16
2048	2	10	8	0, 3, 5, 8, 10, 13, 15, 18
1024	1	20	16	0, 1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17, 19

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 20 GB of frame buffer in mixed-size mode.

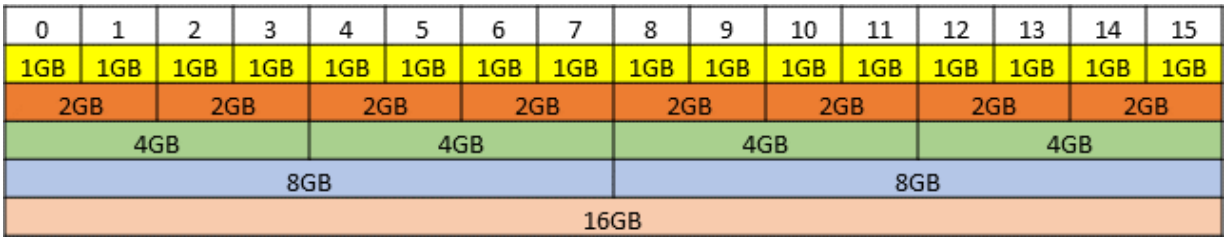


B.9. vGPU Placements for GPUs with 16 GB of Frame Buffer

Placement region size: 16

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
16384	16	1	1	0
8192	8	2	2	0, 8
4096	4	4	4	0, 4, 8, 12
2048	2	8	8	0, 2, 4, 6, 8, 10, 12, 14
1024	1	16	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 16 GB of frame buffer in mixed-size mode.


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB 1GB
2GB 2GB 2GB 2GB 2GB 2GB 2GB 2GB
4GB 4GB 4GB 4GB
8GB 8GB
16GB

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