TABLE OF CONTENTS

Chapter 1. Introduction to NVIDIA vGPU Software

1.1. How NVIDIA vGPU Software Is Used

1.1.2. GPU Pass-Through

1.1.3. Bare-Metal Deployment

1.2. How this Guide Is Organized

1.3. NVIDIA vGPU Architecture

1.4. Supported GPUs

1.4.1. Virtual GPU Types

1.4.1.1. Tesla M60 Virtual GPU Types

1.4.1.2. Tesla M10 Virtual GPU Types

1.4.1.3. Tesla M6 Virtual GPU Types

1.4.1.4. Tesla P100 PCIe 12GB Virtual GPU Types

1.4.1.5. Tesla P100 PCIe 16GB Virtual GPU Types

1.4.1.6. Tesla P100 SXM2 Virtual GPU Types

1.4.1.7. Tesla P40 Virtual GPU Types

1.4.1.8. Tesla P6 Virtual GPU Types

1.4.1.9. Tesla P4 Virtual GPU Types

1.4.1.10. Tesla V100 SXM2 Virtual GPU Types

1.4.1.11. Tesla V100 SXM2 32GB Virtual GPU Types

1.4.1.12. Tesla V100 PCIe Virtual GPU Types

1.4.1.13. Tesla V100 PCIe 32GB Virtual GPU Types

1.4.1.14. Tesla V100 FHHL Virtual GPU Types

1.4.2. Homogeneous Virtual GPUs

1.5. Guest VM Support

1.5.1. Windows Guest VM Support

1.5.2. Linux Guest VM support

1.6. NVIDIA vGPU Software Features

1.6.1. API Support on NVIDIA vGPU

1.6.2. NVIDIA CUDA Toolkit and OpenCL Support on NVIDIA vGPU Software

1.6.3. Additional Quadro vDWS Features

1.6.4. NVIDIA GPU Cloud (NGC) Containers Support on NVIDIA vGPU Software

Chapter 2. Installing and Configuring NVIDIA Virtual GPU Manager

2.1. Prerequisites for Using NVIDIA vGPU

2.2. Switching the Mode of a Tesla M60 or M6 GPU

2.3. Installing and Configuring the NVIDIA Virtual GPU Manager for Citrix XenServer

2.3.1. Installing and Updating the NVIDIA Virtual GPU Manager for XenServer

2.3.1.1. Installing the RPM package for XenServer

2.3.1.2. Updating the RPM Package for XenServer

2.3.1.3. Installing or Updating the Supplemental Pack for XenServer

2.3.1.4. Verifying the Installation of the NVIDIA vGPU Software for XenServer Package
A.4. Changing the vGPU Scheduling Policy for Select GPUs .......................................... 135
A.5. Restoring Default vGPU Scheduler Settings .......................................................... 137

Appendix B. XenServer Basics ..............................................................................138
B.1. Opening a dom0 shell .................................................................................... 138
   B.1.1. Accessing the dom0 shell through XenCenter ............................................. 138
   B.1.2. Accessing the dom0 shell through an SSH client ......................................... 139
B.2. Copying files to dom0 .................................................................................... 139
   B.2.1. Copying files by using an SCP client .......................................................... 139
   B.2.2. Copying files by using a CIFS-mounted file system ..................................... 140
B.3. Determining a VM’s UUID ..............................................................................140
   B.3.1. Determining a VM’s UUID by using xe vm-list ............................................. 141
   B.3.2. Determining a VM’s UUID by using XenCenter ............................................ 141
B.4. Using more than two vCPUs with Windows client VMs ..................................... 142
B.5. Pinning VMs to a specific CPU socket and cores ............................................. 142
B.6. Changing dom0 vCPU Default configuration ............................................... 143
   B.6.1. Changing the number of dom0 vCPUs ....................................................... 144
   B.6.2. Pinning dom0 vCPUs ............................................................................. 144
B.7. How GPU locality is determined ..................................................................... 144
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>NVIDIA vGPU System Architecture</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>NVIDIA vGPU Internal Architecture</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example vGPU Configurations on Tesla M60</td>
<td>21</td>
</tr>
<tr>
<td>Figure 4</td>
<td>NVIDIA vGPU Manager supplemental pack selected in XenCenter</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Successful installation of NVIDIA vGPU Manager supplemental pack</td>
<td>29</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Using Citrix XenCenter to configure a VM with a vGPU</td>
<td>31</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Shared default graphics type</td>
<td>51</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Host graphics settings for vGPU</td>
<td>52</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Shared graphics type</td>
<td>52</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Graphics device settings for a physical GPU</td>
<td>53</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Shared direct graphics type</td>
<td>53</td>
</tr>
<tr>
<td>Figure 12</td>
<td>VM settings for vGPU</td>
<td>55</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Using XenCenter to configure a pass-through GPU</td>
<td>60</td>
</tr>
<tr>
<td>Figure 14</td>
<td>NVIDIA driver installation</td>
<td>72</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Verifying NVIDIA driver operation using NVIDIA Control Panel</td>
<td>73</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Update xorg.conf settings</td>
<td>74</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Verifying operation with nvidia-settings</td>
<td>75</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Managing vGPU licensing in NVIDIA Control Panel</td>
<td>77</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Using XenCenter to remove a vGPU configuration from a VM</td>
<td>81</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Modifying GPU placement policy in XenCenter</td>
<td>83</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Breadth-first allocation scheme setting for vGPU-enabled VMs</td>
<td>84</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Host graphics settings for vGPU</td>
<td>85</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Depth-first allocation scheme setting for vGPU-enabled VMs</td>
<td>86</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Using Citrix XenCenter to monitor GPU performance</td>
<td>102</td>
</tr>
</tbody>
</table>
Figure 25  Using nvidia-smi from a Windows guest VM to get total resource usage by all applications................................................................. 104

Figure 26  Using nvidia-smi from a Windows guest VM to get resource usage by individual applications................................................................. 105

Figure 27  Using Windows Performance Monitor to monitor GPU performance ......................... 106

Figure 28  Using WMI Explorer to monitor GPU performance .............................................. 107

Figure 29  Physical GPU display in XenCenter ................................................................. 111

Figure 30  Editing a GPU's enabled vGPU types using XenCenter ........................................ 119

Figure 31  Using a custom GPU group within XenCenter .................................................. 121

Figure 32  Cloning a VM using XenCenter ................................................................. 122

Figure 33  A NUMA server platform ........................................................................ 125

Figure 34  Including NVIDIA logs in a XenServer status report ......................................... 132

Figure 35  Connecting to the dom0 shell by using XenCenter ............................................. 139

Figure 36  Using XenCenter to determine a VM's UUID .................................................. 142
Chapter 1.
INTRODUCTION TO NVIDIA VGPU SOFTWARE

NVIDIA vGPU software is a graphics virtualization platform that provides virtual machines (VMs) access to NVIDIA GPU technology.

1.1. How NVIDIA vGPU Software Is Used

NVIDIA vGPU software can be used in several ways.

1.1.1. NVIDIA vGPU

NVIDIA Virtual GPU (vGPU™) enables multiple virtual machines (VMs) to have simultaneous, direct access to a single physical GPU, using the same NVIDIA graphics drivers that are deployed on non-virtualized operating systems. By doing this, NVIDIA vGPU provides VMs with unparalleled graphics performance and application compatibility, together with the cost-effectiveness and scalability brought about by sharing a GPU among multiple workloads.

For more information, see Installing and Configuring NVIDIA Virtual GPU Manager.

1.1.2. GPU Pass-Through

In GPU pass-through mode, an entire physical GPU is directly assigned to one VM, bypassing the NVIDIA Virtual GPU Manager. In this mode of operation, the GPU is accessed exclusively by the NVIDIA driver running in the VM to which it is assigned. The GPU is not shared among VMs.

For more information, see Using GPU Pass-Through.

1.1.3. Bare-Metal Deployment

In a bare-metal deployment, you can use NVIDIA vGPU software graphics drivers with Quadro vDWS and GRID Virtual Applications licenses to deliver remote virtual
desktops and applications. If you intend to use Tesla boards without a hypervisor for this purpose, use NVIDIA vGPU software graphics drivers, not other NVIDIA drivers.

To use NVIDIA vGPU software drivers for a bare-metal deployment, complete these tasks:

1. Install the driver on the physical host.
   For instructions, see Installing the NVIDIA vGPU Software Graphics Driver.
2. License any NVIDIA vGPU software that you are using.
   For instructions, see Virtual GPU Client Licensing User Guide.
3. Configure the platform for remote access.
   To use graphics features with Tesla GPUs, you must use a supported remoting solution, for example, RemoteFX, XenApp, VNC, or similar technology.
4. Use the display settings feature of the host OS to configure the Tesla GPU as the primary display.
   NVIDIA Tesla generally operates as a secondary device on bare-metal platforms.
5. If the system has multiple display adapters, disable display devices connected through adapters that are not from NVIDIA.
   You can use the display settings feature of the host OS or the remoting solution for this purpose. On NVIDIA GPUs, including Tesla GPUs, a default display device is enabled.
   Users can launch applications that require NVIDIA GPU technology for enhanced user experience only after displays that are driven by NVIDIA adapters are enabled.

1.2. How this Guide Is Organized

Virtual GPU Software User Guide is organized as follows:

- This chapter introduces the architecture and features of NVIDIA vGPU software.
- Installing and Configuring NVIDIA Virtual GPU Manager provides a step-by-step guide to installing and configuring vGPU on supported hypervisors.
- Using GPU Pass-Through explains how to configure a GPU for pass-through on supported hypervisors.
- Installing the NVIDIA vGPU Software Graphics Driver explains how to install NVIDIA vGPU software graphics driver on Windows and Linux operating systems.
- Licensing an NVIDIA vGPU explains how to license NVIDIA vGPU licensed products on Windows and Linux operating systems.
- Modifying a VM’s NVIDIA vGPU Configuration explains how to remove a VM’s vGPU configuration and modify GPU assignments for vGPU-enabled vMware vSphere VMs.
- XenServer vGPU Management covers vGPU management on XenServer.
- XenServer Performance Tuning covers vGPU performance optimization on XenServer.
- Troubleshooting provides guidance on troubleshooting.
1.3. 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](image)

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.

All vGPUs resident on a physical GPU share access to the GPU’s engines including the graphics (3D), video decode, and video encode engines.
1.4. Supported GPUs

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

1.4.1. 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. Virtual GPU 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 at which they are targeted. Each series is identified by the last letter of the vGPU type name.

- Q-series virtual GPU types are targeted at designers and power users.
- B-series virtual GPU types are targeted at power users.
- A-series virtual GPU types are targeted at virtual applications users.¹

¹ A-series NVIDIA vGPUs support a single display at low resolution to be used as the console display in remote application environments such as RDSH and XenApp. The maximum resolution for the A-series NVIDIA vGPUs applies only to the console display. The maximum resolution of each RDSH or XenApp session is not restricted by the maximum resolution of the vGPU.
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 M60-2Q is allocated 2048 Mbytes of frame buffer on a Tesla M60 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, a Tesla M60 board can support up to 4 M60-2Q vGPUs on each of its two physical GPUs, for a total of 8 vGPUs, but only 2 M60-4Q vGPUs, for a total of 4 vGPUs.

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

- Q-series virtual GPU types require a Quadro vDWS license.
- B-series virtual GPU types require a GRID Virtual PC license but can also be used with a Quadro vDWS license.
- A-series vGPU types require a GRID Virtual Applications license.

NVIDIA vGPUs with less than 1 Gbyte of frame buffer support only 1 virtual display head on a Windows 10 guest OS.

### 1.4.1.1. Tesla M60 Virtual GPU Types

Physical GPUs per board: 2

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>M60-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M60-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M60-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M60-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M60-0Q</td>
<td>Power User, Designer</td>
<td>512</td>
<td>2</td>
<td>2560×1600</td>
<td>16</td>
<td>32</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M60-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>4</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>M60-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>4</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>Virtual GPU Type</td>
<td>Intended Use Case</td>
<td>Frame Buffer (Mbytes)</td>
<td>Virtual Display Heads</td>
<td>Maximum Resolution per Display Head</td>
<td>Maximum vGPUs per GPU</td>
<td>Maximum vGPUs per Board</td>
<td>Required License Edition</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
<td>-------------------------------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>M60-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>8</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>M60-0B</td>
<td>Power User</td>
<td>512</td>
<td>2</td>
<td>2560×1600</td>
<td>16</td>
<td>32</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>M60-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280×1024</td>
<td>1</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>M60-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280×1024</td>
<td>2</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>M60-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280×1024</td>
<td>4</td>
<td>8</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>M60-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280×1024</td>
<td>8</td>
<td>16</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

1.4.1.2. Tesla M10 Virtual GPU Types

Physical GPUs per board: 4

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>M10-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M10-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M10-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M10-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>32</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M10-0Q</td>
<td>Power User, Designer</td>
<td>512</td>
<td>2</td>
<td>2560×1600</td>
<td>16</td>
<td>64</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M10-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>4</td>
<td>16</td>
<td>GRID Virtual PC</td>
</tr>
</tbody>
</table>
### 1.4.1.3. Tesla M6 Virtual GPU Types

Physical GPUs per board: 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>M6-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M6-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M6-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>M6-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
</tbody>
</table>
### 1.4.1.4. Tesla P100 PCIe 12GB Virtual GPU Types

Physical GPUs per board: 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>P100C-12Q</td>
<td>Designer</td>
<td>12288</td>
<td>4</td>
<td>4096x2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100C-6Q</td>
<td>Designer</td>
<td>6144</td>
<td>4</td>
<td>4096x2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>Virtual GPU Type</td>
<td>Intended Use Case</td>
<td>Frame Buffer (Mbytes)</td>
<td>Virtual Display Heads</td>
<td>Maximum Resolution per Display Head</td>
<td>Maximum vGPUs per GPU</td>
<td>Maximum vGPUs per Board</td>
<td>Required License Edition</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
<td>-------------------------------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>P100C-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>3</td>
<td>3</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100C-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>6</td>
<td>6</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100C-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>12</td>
<td>12</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100C-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>6</td>
<td>6</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100C-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>6</td>
<td>6</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100C-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>12</td>
<td>12</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100C-12A</td>
<td>Virtual Application User</td>
<td>12288</td>
<td>1</td>
<td>1280×1024</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100C-6A</td>
<td>Virtual Application User</td>
<td>6144</td>
<td>1</td>
<td>1280×1024</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100C-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280×1024</td>
<td>3</td>
<td>3</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100C-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280×1024</td>
<td>6</td>
<td>6</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100C-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280×1024</td>
<td>12</td>
<td>12</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

### 1.4.1.5. Tesla P100 PCIe 16GB Virtual GPU Types

Physical GPUs per board: 1
<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>P100-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096x2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096x2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096x2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096x2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096x2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096x2160</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560x1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560x1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100-16A</td>
<td>Virtual Application User</td>
<td>16384</td>
<td>1</td>
<td>1280x1024[1]</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280x1024[1]</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280x1024[1]</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280x1024[1]</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280x1024[1]</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

1.4.1.6. Tesla P100 SXM2 Virtual GPU Types

Physical GPUs per board: 1
<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>P100X-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096x2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100X-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096x2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100X-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096x2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100X-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096x2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100X-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096x2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P100X-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096x2160</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100X-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560x1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100X-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560x1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P100X-16A</td>
<td>Virtual Application User</td>
<td>16384</td>
<td>1</td>
<td>1280x1024</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100X-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280x1024</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100X-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280x1024</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100X-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280x1024</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P100X-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280x1024</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

### 1.4.1.7. Tesla P40 Virtual GPU Types

Physical GPUs per board: 1
<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>P40-24Q</td>
<td>Designer</td>
<td>24576</td>
<td>4</td>
<td>4096x2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-12Q</td>
<td>Designer</td>
<td>12288</td>
<td>4</td>
<td>4096x2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096x2160</td>
<td>3</td>
<td>3</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-6Q</td>
<td>Designer</td>
<td>6144</td>
<td>4</td>
<td>4096x2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096x2160</td>
<td>6</td>
<td>6</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-3Q</td>
<td>Designer</td>
<td>3072</td>
<td>4</td>
<td>4096x2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096x2160</td>
<td>12</td>
<td>12</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096x2160</td>
<td>24</td>
<td>24</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P40-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096x2160</td>
<td>12</td>
<td>12</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P40-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560x1600</td>
<td>12</td>
<td>12</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P40-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560x1600</td>
<td>24</td>
<td>24</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P40-24A</td>
<td>Virtual Application User</td>
<td>24576</td>
<td>1</td>
<td>1280x1024</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P40-12A</td>
<td>Virtual Application User</td>
<td>12288</td>
<td>1</td>
<td>1280x1024</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P40-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280x1024</td>
<td>3</td>
<td>3</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>P40-6A</td>
<td>Virtual Application User</td>
<td>6144</td>
<td>1</td>
<td>1280x1024</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>
### 1.4.1.8. Tesla P6 Virtual GPU Types

Physical GPUs per board: 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P6-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P6-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P6-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P6-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P6-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P6-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P6-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
</tbody>
</table>
### 1.4.1.9. Tesla P4 Virtual GPU Types

Physical GPUs per board: 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P4-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P4-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P4-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>P4-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P4-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>P4-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC</td>
</tr>
</tbody>
</table>
## 1.4.1.10. Tesla V100 SXM2 Virtual GPU Types

Physical GPUs per board: 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100X-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100X-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100X-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100X-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100X-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100X-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100X-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100X-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC</td>
</tr>
</tbody>
</table>
### 1.4.1.11. Tesla V100 SXM2 32GB Virtual GPU Types

**Physical GPUs per board:** 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100DX-32Q</td>
<td>Designer</td>
<td>32768</td>
<td>4</td>
<td>4096x2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100DX-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096x2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100DX-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096x2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100DX-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096x2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100DX-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096x2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100DX-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096x2160</td>
<td>32</td>
<td>32</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100DX-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096x2160</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
</tbody>
</table>
1.4.1.12. Tesla V100 PCIe Virtual GPU Types

Physical GPUs per board: 1

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
</tbody>
</table>
### 1.4.1.13. Tesla V100 PCIe 32GB Virtual GPU Types

<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100-16A</td>
<td>Virtual Application User</td>
<td>16384</td>
<td>1</td>
<td>1280×1024</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280×1024</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280×1024</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280×1024</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280×1024</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

**1.4.1.13. Tesla V100 PCIe 32GB Virtual GPU Types**

Physical GPUs per board: 1
<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100D-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100D-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100D-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100D-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>32</td>
<td>32</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100D-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100D-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100D-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>32</td>
<td>32</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100D-32A</td>
<td>Virtual Application User</td>
<td>32768</td>
<td>1</td>
<td>1280×1024</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100D-16A</td>
<td>Virtual Application User</td>
<td>16384</td>
<td>1</td>
<td>1280×1024</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100D-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280×1024</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100D-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280×1024</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100D-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280×1024</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100D-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280×1024</td>
<td>32</td>
<td>32</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

1.4.1.14. Tesla V100 FHHL Virtual GPU Types
Physical GPUs per board: 1
<table>
<thead>
<tr>
<th>Virtual GPU Type</th>
<th>Intended Use Case</th>
<th>Frame Buffer (Mbytes)</th>
<th>Virtual Display Heads</th>
<th>Maximum Resolution per Display Head</th>
<th>Maximum vGPUs per GPU</th>
<th>Maximum vGPUs per Board</th>
<th>Required License Edition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100L-16Q</td>
<td>Designer</td>
<td>16384</td>
<td>4</td>
<td>4096×2160</td>
<td>1</td>
<td>1</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100L-8Q</td>
<td>Designer</td>
<td>8192</td>
<td>4</td>
<td>4096×2160</td>
<td>2</td>
<td>2</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100L-4Q</td>
<td>Designer</td>
<td>4096</td>
<td>4</td>
<td>4096×2160</td>
<td>4</td>
<td>4</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100L-2Q</td>
<td>Designer</td>
<td>2048</td>
<td>4</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100L-1Q</td>
<td>Power User, Designer</td>
<td>1024</td>
<td>2</td>
<td>4096×2160</td>
<td>16</td>
<td>16</td>
<td>Quadro vDWS</td>
</tr>
<tr>
<td>V100L-2B</td>
<td>Power User</td>
<td>2048</td>
<td>2</td>
<td>4096×2160</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100L-2B4</td>
<td>Power User</td>
<td>2048</td>
<td>4</td>
<td>2560×1600</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100L-1B</td>
<td>Power User</td>
<td>1024</td>
<td>4</td>
<td>2560×1600</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual PC or Quadro vDWS</td>
</tr>
<tr>
<td>V100L-16A</td>
<td>Virtual Application User</td>
<td>16384</td>
<td>1</td>
<td>1280×1024[1]</td>
<td>1</td>
<td>1</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100L-8A</td>
<td>Virtual Application User</td>
<td>8192</td>
<td>1</td>
<td>1280×1024[1]</td>
<td>2</td>
<td>2</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100L-4A</td>
<td>Virtual Application User</td>
<td>4096</td>
<td>1</td>
<td>1280×1024[1]</td>
<td>4</td>
<td>4</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100L-2A</td>
<td>Virtual Application User</td>
<td>2048</td>
<td>1</td>
<td>1280×1024[1]</td>
<td>8</td>
<td>8</td>
<td>GRID Virtual Application</td>
</tr>
<tr>
<td>V100L-1A</td>
<td>Virtual Application User</td>
<td>1024</td>
<td>1</td>
<td>1280×1024[1]</td>
<td>16</td>
<td>16</td>
<td>GRID Virtual Application</td>
</tr>
</tbody>
</table>

### 1.4.2. Homogeneous Virtual GPUs

This release of NVIDIA vGPU supports only homogeneous virtual GPUs. At any given time, the virtual GPUs resident on a single physical GPU must be all of the same type. However, this restriction doesn’t extend across physical GPUs on the same card.
Different physical GPUs on the same card may host different types of virtual GPU at the same time, provided that the vGPU types on any one physical GPU are the same.

For example, a Tesla M60 card has two physical GPUs, and can support several types of virtual GPU. Figure 3 shows the following examples of valid and invalid virtual GPU configurations on Tesla M60:

- A valid configuration with M60-2Q vGPUs on GPU 0 and M60-4Q vGPUs on GPU 1
- A valid configuration with M60-1B vGPUs on GPU 0 and M60-2Q vGPUs on GPU 1
- An invalid configuration with mixed vGPU types on GPU 0

Figure 3 Example vGPU Configurations on Tesla M60

1.5. Guest VM Support

NVIDIA vGPU supports Windows and Linux guest VM operating systems. The supported vGPU types depend on the guest VM OS.

For details of the supported releases of Windows and Linux, and for further information on supported configurations, see the driver release notes for your hypervisor at NVIDIA Virtual GPU Software Documentation.

1.5.1. Windows Guest VM Support

Windows guest VMs are supported on all NVIDIA vGPU types.

1.5.2. Linux Guest VM Support

64-bit Linux guest VMs are supported only on Q-series and B-series NVIDIA vGPUs.

1.6. NVIDIA vGPU Software Features

NVIDIA vGPU software includes Quadro vDWS, GRID Virtual PC, and GRID Virtual Applications.
1.6.1. API Support on NVIDIA vGPU

NVIDIA vGPU includes support for the following APIs:
- DirectX 12, Direct2D, and DirectX Video Acceleration (DXVA)
- OpenGL 4.5
- NVIDIA vGPU software SDK (remote graphics acceleration)
- Vulkan 1.0

1.6.2. NVIDIA CUDA Toolkit and OpenCL Support on NVIDIA vGPU Software

OpenCL and CUDA applications are supported on the following NVIDIA vGPU types:
- The 8Q vGPU type on Tesla M6, Tesla M10, and Tesla M60 GPUs
- All Q-series vGPU types on the following GPUs:
  - Tesla P4
  - Tesla P6
  - Tesla P40
  - Tesla P100 SXM2
  - Tesla P100 PCIe 16 GB
  - Tesla P100 PCIe 12 GB
  - Tesla V100 SXM2
  - Tesla V100 PCIe
  - Tesla V100 FHHL

NVIDIA vGPU does not support the following NVIDIA CUDA Toolkit features:
- Unified Memory
- Dynamic page retirement
- Error-correcting code (ECC)
- Peer-to-peer
- GPUDirect remote direct memory access (RDMA)
- Development tools such as IDEs, debuggers, profilers, and utilities as listed under CUDA Toolkit Major Components in CUDA Toolkit 10.0 Release Notes for Windows, Linux, and Mac OS

These features are supported in GPU pass-through mode and in bare-metal deployments.

For more information about NVIDIA CUDA Toolkit, see CUDA Toolkit 10.0 Documentation.
1.6.3. Additional Quadro vDWS Features

In addition to the features of GRID Virtual PC and GRID Virtual Applications, Quadro vDWS provides the following features:

- Workstation-specific graphics features and accelerations
- Certified drivers for professional applications
- GPU pass through for workstation or professional 3D graphics

In pass-through mode, Quadro vDWS supports up to four virtual display heads at 4K resolution.

1.6.4. NVIDIA GPU Cloud (NGC) Containers Support on NVIDIA vGPU Software

NVIDIA vGPU software supports NGC containers in NVIDIA vGPU and GPU pass-through deployments on all supported hypervisors.

In NVIDIA vGPU deployments, the following vGPU types are supported:

- P4-8Q
- P40–24Q
- P100–16Q
- V100–32Q

In GPU pass-through deployments, all GPUs based on the NVIDIA Pascal™ and Volta™ architectures that support NVIDIA vGPU software are supported.

The Ubuntu guest operating systems is supported.

For more information about setting up NVIDIA vGPU software for use with NGC containers, see Using NGC with NVIDIA Virtual GPU Software Setup Guide.
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 vGPU software licensed products that you are using.

2.1. Prerequisites for Using NVIDIA vGPU

Before proceeding, ensure that these prerequisites are met:

‣ You have a server platform that is capable of hosting your chosen hypervisor and NVIDIA GPUs that support NVIDIA vGPU software.
‣ One or more NVIDIA GPUs that support NVIDIA vGPU software is installed in your server platform.
‣ You have downloaded the NVIDIA vGPU software package for your chosen hypervisor, which consists of the following software:
  ▶ NVIDIA Virtual GPU Manager for your hypervisor
  ▶ NVIDIA vGPU software graphics drivers for supported guest operating systems
‣ The following software is installed according to the instructions in the software vendor’s documentation:
  ▶ Your chosen hypervisor, for example, Citrix XenServer, Red Hat Enterprise Linux KVM, Red Hat Virtualization (RHV), or VMware vSphere Hypervisor (ESXi)
  ▶ The software for managing your chosen hypervisor, for example, Citrix XenCenter management GUI, or VMware vCenter Server
The virtual desktop software that you will use with virtual machines (VMs) running NVIDIA Virtual GPU, for example, Citrix XenDesktop, or VMware Horizon

If you are using VMware vSphere Hypervisor (ESXi), ensure that the ESXi host on which you will configure a VM with NVIDIA vGPU is not a member of a VMware Distributed Resource Scheduler (DRS) cluster.

A VM to be enabled with vGPU is created.

Red Hat Enterprise Linux KVM supports multiple vGPUs in a VM. However, other hypervisors covered in this guide support only a single vGPU in a VM.

Your chosen guest OS is installed in the VM.

For information about supported hardware and software, and any known issues for this release of NVIDIA vGPU software, refer to the Release Notes for your chosen hypervisor:

- Virtual GPU Software for Citrix XenServer Release Notes
- Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes
- Virtual GPU Software for VMware vSphere Release Notes

2.2. Switching the Mode of a Tesla M60 or M6 GPU

Tesla M60 and M6 GPUs support compute mode and graphics mode. NVIDIA vGPU requires GPUs that support both modes to operate in graphics mode.

Only Tesla M60 and M6 GPUs require and support mode switching. Other GPUs that support NVIDIA vGPU do not require or support mode switching.

Recent Tesla M60 GPUs and M6 GPUs are supplied in graphics mode. However, your GPU might be in compute mode if it is an older Tesla M60 GPU or M6 GPU, or if its mode has previously been changed.

If your GPU supports both modes but is in compute mode, you must use the gpumodeswitch tool to change the mode of the GPU to graphics mode. If you are unsure which mode your GPU is in, use the gpumodeswitch tool to find out the mode.

For more information, see gpumodeswitch User Guide.

2.3. Installing and Configuring the NVIDIA Virtual GPU Manager for Citrix XenServer

The following topics step you through the process of setting up a single Citrix XenServer VM to use NVIDIA vGPU. After the process is complete, you can install the graphics driver for your guest OS and license any NVIDIA vGPU software licensed products that you are using.
These setup steps assume familiarity with the XenServer skills covered in XenServer Basics.

### 2.3.1. Installing and Updating the NVIDIA Virtual GPU Manager for XenServer

The NVIDIA Virtual GPU Manager runs in XenServer's dom0. The NVIDIA Virtual GPU Manager for Citrix XenServer is supplied as an RPM file and as a Supplemental Pack.

**Caution** NVIDIA Virtual GPU Manager and Guest VM drivers must be matched from the same main driver branch. If you update vGPU Manager to a release from another driver branch, guest VMs will boot with vGPU disabled until their guest vGPU driver is updated to match the vGPU Manager version. Consult Virtual GPU Software for Citrix XenServer Release Notes for further details.

#### 2.3.1.1. Installing the RPM package for XenServer

The RPM file must be copied to XenServer’s dom0 prior to installation (see Copying files to dom0).

1. Use the `rpm` command to install the package:

   ```bash
   [root@xenserver ~]# rpm -iv NVIDIA-vGPU-xenserver-7.0-410.68.x86_64.rpm
   Preparing packages for installation...
   NVIDIA-vGPU-xensuser-7.0-410.68
   [root@xenserver ~]#
   ```

2. Reboot the XenServer platform:

   ```bash
   [root@xenserver ~]# shutdown –r now
   Broadcast message from root (pts/1) (Fri Oct 12 14:24:11 2018):
   The system is going down for reboot NOW!
   [root@xenserver ~]#
   ```

#### 2.3.1.2. Updating the RPM Package for XenServer

If an existing NVIDIA Virtual GPU Manager is already installed on the system and you want to upgrade, follow these steps:

1. Shut down any VMs that are using NVIDIA vGPU.
2. Install the new package using the `-U` option to the `rpm` command, to upgrade from the previously installed package:

   ```bash
   [root@xenserver ~]# rpm -Uv NVIDIA-vGPU-xensuser-7.0-410.68.x86_64.rpm
   Preparing packages for installation...
   NVIDIA-vGPU-xensuser-7.0-410.68
   [root@xenserver ~]#
   ```

   You can query the version of the current NVIDIA Virtual GPU Manager package using the `rpm –q` command:
3. Reboot the XenServer platform:

```
[root@xenserver ~]# shutdown -r now
Broadcast message from root (pts/1) (Fri Oct 12 14:24:11 2018):
The system is going down for reboot NOW!
[root@xenserver ~]#
```

2.3.1.3. Installing or Updating the Supplemental Pack for XenServer

XenCenter can be used to install or update Supplemental Packs on XenServer hosts. The NVIDIA Virtual GPU Manager supplemental pack is provided as an ISO.

1. Select **Install Update** from the **Tools** menu.
2. Click **Next** after going through the instructions on the **Before You Start** section.
3. Click **Select update or supplemental pack from disk** on the **Select Update** section and open NVIDIA's XenServer Supplemental Pack ISO.
Figure 4 NVIDIA vGPU Manager supplemental pack selected in XenCenter

4. Click Next on the Select Update section.
5. In the Select Servers section select all the XenServer hosts on which the Supplemental Pack should be installed on and click Next.
6. Click Next on the Upload section once the Supplemental Pack has been uploaded to all the XenServer hosts.
7. Click Next on the Prechecks section.
8. Click Install Update on the Update Mode section.
9. Click Finish on the Install Update section.
2.3.1.4. Verifying the Installation of the NVIDIA vGPU Software for XenServer Package

After the XenServer platform has rebooted, verify the installation of the NVIDIA vGPU software package for XenServer.

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

   ```bash
   [root@xenserver ~]# lsmod | grep nvidia
   nvidia               9522927  0
   i2c_core               20294  2 nvidia,i2c_i801
   [root@xenserver ~]#
   ```

2. 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.

   ```bash
   [root@xenserver ~]# nvidia-smi
   Fri Nov 16 18:46:50 2018
   +-----------------------------------------------------------------------------+
   | NVIDIA-SMI 410.68   Driver Version: 410.71   |
   +-----------------------------------------------------------------------------+
   ```
Installing and Configuring NVIDIA Virtual GPU Manager

<table>
<thead>
<tr>
<th>GPU Name</th>
<th>Persistence-M</th>
<th>Bus-Id</th>
<th>Volatile Uncorr. ECC</th>
<th>GPU-Util</th>
<th>Compute M.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Tesla M60</td>
<td>On</td>
<td>00000000:05:00.0 Off</td>
<td>0% Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>25C</td>
<td>P8</td>
<td>24W / 150W</td>
<td>13MiB / 8191MiB</td>
<td></td>
</tr>
<tr>
<td>1 Tesla M60</td>
<td>On</td>
<td>00000000:06:00.0 Off</td>
<td>0% Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>24C</td>
<td>P8</td>
<td>24W / 150W</td>
<td>13MiB / 8191MiB</td>
<td></td>
</tr>
<tr>
<td>2 Tesla M60</td>
<td>On</td>
<td>00000000:86:00.0 Off</td>
<td>0% Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>25C</td>
<td>P8</td>
<td>25W / 150W</td>
<td>13MiB / 8191MiB</td>
<td></td>
</tr>
<tr>
<td>3 Tesla M60</td>
<td>On</td>
<td>00000000:87:00.0 Off</td>
<td>0% Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>28C</td>
<td>P8</td>
<td>24W / 150W</td>
<td>13MiB / 8191MiB</td>
<td></td>
</tr>
<tr>
<td>Processes:</td>
<td></td>
<td>GPU Memory</td>
<td>Usage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU PID Type</td>
<td>Process name</td>
<td>Usage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No running processes found</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[root@xenserver ~]#

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.

2.3.2. Configuring a Citrix XenServer VM with Virtual GPU

XenServer supports configuration and management of virtual GPUs using XenCenter, or the xe command line tool that is run in a XenServer dom0 shell. Basic configuration using XenCenter is described in the following sections. Command line management using xe is described in XenServer vGPU Management.

1. Ensure the VM is powered off.
2. Right-click the VM in XenCenter, select Properties to open the VM's properties, and select the GPU property.
   The available GPU types are listed in the GPU type drop-down list:
After you have configured a XenServer VM with a vGPU, start the VM, either from XenCenter or by using `xe vm-start` in a dom0 shell. You can view the VM's console in XenCenter.

After the VM has booted, install the NVIDIA vGPU software graphics driver as explained in Installing the NVIDIA vGPU Software Graphics Driver.

2.4. Installing and Configuring the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM or RHV

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

Red Hat Enterprise Linux KVM and RHV use the same Virtual GPU Manager package, but are configured with NVIDIA vGPU in different ways.

For RHV, follow this sequence of instructions:

1. Installing the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM or RHV
2. Adding a vGPU to a Red Hat Virtualization (RHV) VM

For Red Hat Enterprise Linux KVM, follow this sequence of instructions:

1. Installing the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM or RHV
2. Getting the BDF and Domain of a GPU on Red Hat Enterprise Linux KVM
3. Creating an NVIDIA vGPU on Red Hat Enterprise Linux KVM
4. Adding One or More vGPUs to a Red Hat Enterprise Linux KVM VM
5. Setting vGPU Plugin Parameters on Red Hat Enterprise Linux KVM

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

2.4.1. Installing the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM or RHV

The NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM and Red Hat Virtualization (RHV) is provided as a `.rpm` file.

Caution NVIDIA Virtual GPU Manager and Guest VM drivers must be matched from the same main driver branch. If you update vGPU Manager to a release from another driver branch, guest VMs will boot with vGPU disabled until their guest vGPU driver is updated to match the vGPU Manager version. Consult Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes for further details.

2.4.1.1. Installing the Virtual GPU Manager Package for Red Hat Enterprise Linux KVM or RHV

Before installing the RPM package for Red Hat Enterprise Linux KVM or RHV, ensure that the `sshd` service on the Red Hat Enterprise Linux KVM or RHV 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).

1. Securely copy the RPM file from the system where you downloaded the file to the Red Hat Enterprise Linux KVM or RHV 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 or RHV server.

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

`kvm-server`

  The host name or IP address of the Red Hat Enterprise Linux KVM or RHV server.

3. Change to the directory on the Red Hat Enterprise Linux KVM or RHV 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-7.5-410.68.x86_64.rpm
Preparing packages for installation...
NVIDIA-vGPU-rhel-7.5-410.68
#
```

5. Reboot the Red Hat Enterprise Linux KVM or RHV server.

```
# systemctl reboot
```

### 2.4.1.2. Verifying the Installation of the NVIDIA vGPU Software for Red Hat Enterprise Linux KVM or RHV

After the Red Hat Enterprise Linux KVM or RHV server has rebooted, verify the installation of the NVIDIA vGPU software package for Red Hat Enterprise Linux KVM or RHV.

1. Verify that the NVIDIA vGPU software 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 Nov 16 18:46:50 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.68       Driver Version: 410.71 |
|-------------------------------+----------------------+----------------------+
| 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 |
+-----------------------------------------------------------------------------+
```

www.nvidia.com
Virtual GPU Software
DU-06920-001_v7.0 | 33
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.

### 2.4.2. Adding a vGPU to a Red Hat Virtualization (RHV) VM

Ensure that the VM to which you want to add the vGPU is shut down.

1. Determine the mediated device type (mdev_type) identifiers of the vGPU types available on the RHV host.

```bash
# vdsm-client Host hostdevListByCaps
...
"mdev": {
  "nvidia-155": {
    "name": "GRID M10-2B",
    "available_instances": "4"
  },
  "nvidia-36": {
    "name": "GRID M10-0Q",
    "available_instances": "16"
  },
...
```

The preceding example shows the mdev_type identifiers of the following vGPU types:

- For the GRID M10-2B vGPU type, the mdev_type identifier is nvidia-155.
- For the GRID M10-0Q vGPU type, the mdev_type identifier is nvidia-36.

2. Note the mdev_type identifier of the vGPU type that you want to add.

3. Log in to the RHV Administration Portal.

4. From the Main Navigation Menu, choose Compute > Virtual Machines > `virtual-machine-name`.

   `virtual-machine-name`
   
   The name of the virtual machine to which you want to add the vGPU.

5. Click Edit.

6. In the Edit Virtual Machine window that opens, click Show Advanced Options and in the list of options, select Custom Properties.

7. From the drop-down list, select mdev_type.

8. In the text field, type the mdev_type identifier of the vGPU type that you want to add and click OK.
After adding a vGPU to an RHV VM, start the VM.

After the VM has booted, install the NVIDIA vGPU software graphics driver as explained in Installing the NVIDIA vGPU Software Graphics Driver.

### 2.4.3. Getting the BDF and Domain of a GPU on Red Hat Enterprise Linux KVM

Sometimes when configuring a physical GPU for use with NVIDIA vGPU software, 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 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.

   ```bash
   # 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.

```bash
# 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.

```bash
# 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.

```bash
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.

```bash
# 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>
```

2.4.4. Creating an NVIDIA vGPU on Red Hat Enterprise Linux KVM

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

The `mdev` device file that you create to represent the vGPU does not persist when the host is rebooted and must be re-created after the host is rebooted. If necessary, 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, by writing a custom script that is executed when the host is rebooted.
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 Red Hat Enterprise Linux KVM.

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

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

   ```bash
   # 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.

   ```bash
   # 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`.

   ```bash
   # 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.

   ```bash
   # 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.

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

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

   ```bash
   # 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.

   ```bash
   # 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 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. Confirm that the vGPU was created.

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

### 2.4.5. Adding One or More vGPUs to a Red Hat Enterprise Linux KVM VM

To support applications and workloads that are compute or graphics intensive, you can add multiple vGPUs to a single VM. If you are adding multiple vGPUs to a single VM, all vGPUs must be of the same type.

For details about which Red Hat Enterprise Linux KVM versions and NVIDIA vGPUs support the assignment of multiple vGPUs to a VM, see [Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes](#).

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 Red Hat Enterprise Linux KVM](#).

You can add vGPUs to a Red Hat Enterprise Linux KVM VM by using any of the following tools:

- The `virsh` command
- The QEMU command line

After adding vGPUs to a Red Hat Enterprise Linux KVM 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 vGPU software graphics driver as explained in Installing the NVIDIA vGPU Software Graphics Driver.

### 2.4.5.1. Adding One or More vGPUs to a Red Hat Enterprise Linux KVM 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.

```
<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>
```
2.4.5.2. Adding One or More vGPUs to a Red Hat Enterprise Linux KVM VM by Using the QEMU Command Line

Add the following options to the QEMU command line:

- For each vGPU that you want to add to the VM, add one `-device` option in the following format:

  ```
  -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.

- Add a `-uuid` option to specify the VM as follows:

  ```
  -uuid vm-uuid
  ```

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

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
  ```

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
  ```

2.4.6. Setting vGPU Plugin Parameters on Red Hat Enterprise Linux KVM

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 Red Hat Enterprise Linux KVM host.

1. Change to the `nvidia` subdirectory of the `mdev` device directory that represents the vGPU.
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
```

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

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

2.4.7. Deleting a vGPU on Red Hat Enterprise Linux KVM

For each vGPU that you want to delete, perform this task in a Linux command shell on the Red Hat Enterprise Linux KVM host.

Before you begin, ensure that the following prerequisites are met:

- You have the domain, bus, slot, and function of the GPU where the vGPU that you want to delete resides. For instructions, see Getting the BDF and Domain of a GPU on Red Hat Enterprise Linux KVM.
- The VM to which the vGPU is assigned is shut down.

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 PCI device BDF 06:00.0.

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

2. Change to the subdirectory of mdev_supported_types that contains registration information for the vGPU.

```
# cd `find . -type d -name uuid`
```
uuid

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

3. Write the value 1 to the remove file in the registration information directory for the vGPU that you want to delete.

   # echo "1" > remove

On the Red Hat Virtualization (RHV) kernel, if you try to remove a vGPU device while its VM is running, the vGPU device might not be removed even if the remove file has been written to successfully. To confirm that the vGPU device is removed, confirm that the UUID of the vGPU is not found in the sysfs file system.

2.4.8. Preparing a GPU Configured for Pass-Through for Use with vGPU

The mode in which a physical GPU is being used determines the Linux kernel module to which the GPU is bound. If you want to switch the mode in which a GPU is being used, you must unbind the GPU from its current kernel module and bind it to the kernel module for the new mode. After binding the GPU to the correct kernel module, you can then configure it for vGPU.

A physical GPU that is passed through to a VM is bound to the vfio-pci kernel module. A physical GPU that is bound to the vfio-pci kernel module can be used only for pass-through. To enable the GPU to be used for vGPU, the GPU must be unbound from vfio-pci kernel module and bound to the nvidia kernel module.

Before you begin, ensure that you have the domain, bus, slot, and function of the GPU that you are preparing for use with vGPU. For instructions, see Getting the BDF and Domain of a GPU on Red Hat Enterprise Linux KVM.

1. Determine the kernel module to which the GPU is bound by running the lspci command with the -k option on the NVIDIA GPUs on your host.

   # lspci -d 10de: -k

   The Kernel driver in use: field indicates the kernel module to which the GPU is bound.

   The following example shows that the NVIDIA Tesla M60 GPU with BDF 06:00.0 is bound to the vfio-pci kernel module and is being used for GPU pass through.

   06:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
   Subsystem: NVIDIA Corporation Device 115e
   Kernel driver in use: vfio-pci

2. Unbind the GPU from vfio-pci kernel module.
a) Change to the `sysfs` directory that represents the `vfio-pci` kernel module.

```
# cd /sys/bus/pci/drivers/vfio-pci
```

b) Write the domain, bus, slot, and function of the GPU to the `unbind` file in this directory.

```
# echo domain:bus:slot.function > unbind
```

3. Bind the GPU to the `nvidia` kernel module.

a) Change to the `sysfs` directory that contains the PCI device information for the physical GPU.

```
# cd /sys/bus/pci/devices/domain\:bus\:slot.function
```

d) Write the domain, bus, slot, and function of the GPU to the `bind` file in this directory.

```
# echo domain:bus:slot.function > bind
```

www.nvidia.com
Virtual GPU Software

DU-06920-001_v7.0 | 43
You can now configure the GPU with vGPU as explained in Installing and Configuring the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM or RHV.

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

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

All physical GPUs on the host are registered with the `mdev` kernel module. Information about the physical GPUs and the vGPU types that can be created on each physical GPU is stored in directories and files under the `/sys/class/mdev_bus/` directory.

The `sysfs` directory for each physical GPU is at the following locations:

- `/sys/bus/pci/devices/`
- `/sys/class/mdev_bus/`

Both directories are a symbolic link to the real directory for PCI devices in the `sysfs` file system.

The organization the `sysfs` directory for each physical GPU is as follows:

```plaintext
/sys/class/mdev_bus/
    |-parent-physical-device
    |   |-mdev_supported_types
    |   |   |-nvidia-vgputype-id
    |   |   |   |-available_instances
    |   |   |   |-create
    |   |   |   |-description
    |   |   |   |-device_api
    |   |   |   |-devices
    |   |   |-name
```

`parent-physical-device`

Each physical GPU on the host is represented by a subdirectory of the `/sys/class/mdev_bus/` directory.

The name of each subdirectory is as follows:

`domain:bus:slot.function`

`domain`, `bus`, `slot`, `function` are the domain, bus, slot, and function of the GPU, for example, `0000:06:00.0`.

Each directory is a symbolic link to the real directory for PCI devices in the `sysfs` file system. For example:

```bash
# 11 /sys/class/mdev_bus/
total 0
lrwxrwxrwx. 1 root root 0 Dec 12 03:20 0000:05:00.0 -> ../..devices/pci0000:00/0000:00:03.0/0000:03:00.0/0000:04:08.0/0000:05:00.0
lrwxrwxrwx. 1 root root 0 Dec 12 03:20 0000:06:00.0 -> ../..devices/pci0000:00/0000:00:03.0/0000:03:00.0/0000:04:09.0/0000:06:00.0
```
Installing and Configuring NVIDIA Virtual GPU Manager

After the Virtual GPU Manager is installed on the host and the host has been rebooted, a directory named `mdev_supported_types` is created under the `sysfs` directory for each physical GPU. The `mdev_supported_types` directory contains a subdirectory for each vGPU type that the physical GPU supports. The name of each subdirectory is `nvidia-vgputype-id`, where `vgputype-id` is an unsigned integer serial number. For example:

```
# l/ mdev_supported_types/
    total 0
  drwxr-xr-x  3 root  root 0 Dec  6 01:37 nvidia-35
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-36
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-37
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-38
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-39
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-40
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-41
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-42
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-43
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-44
  drwxr-xr-x  3 root  root 0 Dec  5 10:43 nvidia-45
```

Each directory represents an individual vGPU type and contains the following files and directories:

- **available_instances**: This file contains the number of instances of this vGPU type that can still be created. This file is updated any time a vGPU of this type is created on or removed from the physical GPU.

  *When a vGPU is created, the content of the available_instances for all other vGPU types on the physical GPU is set to 0. This behavior enforces the requirement that all vGPUs on a physical GPU must be of the same type.*

- **create**: This file is used for creating a vGPU instance. A vGPU instance is created by writing the UUID of the vGPU to this file. The file is write only.

- **description**: This file contains the following details of the vGPU type:
  - The maximum number of virtual display heads that the vGPU type supports
  - The frame rate limiter (FRL) configuration in frames per second
  - The frame buffer size in Mbytes
  - The maximum resolution per display head
  - The maximum number of vGPU instances per physical GPU

  For example:

  ```
  # cat description
  num_heads=4, frl_config=60, framebuffer=2048M, max_resolution=4096x2160, max_instance=4
  ```
device_api
   This file contains the string `vfio_pci` to indicate that a vGPU is a PCI device.

devices
   This directory contains all the mdev devices that are created for the vGPU type. For example:

   ```
   # ll devices
   total 0
   lrwxrwxrwx 1 root root 0 Dec 6 01:52 aa618089-8b16-4d01-a136-25a0f3c73123
   -> ../../../aa618089-8b16-4d01-a136-25a0f3c73123
   ```

name
   This file contains the name of the vGPU type. For example:

   ```
   # cat name
   GRID M10-2Q
   ```

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

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

For NVIDIA vGPU, follow this sequence of instructions:

1. Installing and Updating the NVIDIA Virtual GPU Manager for vSphere
2. Configuring VMware vMotion with vGPU for VMware vSphere
3. Changing the Default Graphics Type in VMware vSphere 6.5 and Later
4. Configuring a vSphere VM with NVIDIA vGPU

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

For VMware vSGA, follow this sequence of instructions:

1. Installing and Updating the NVIDIA Virtual GPU Manager for vSphere
2. Configuring a vSphere VM with VMware vSGA

Installation of the NVIDIA vGPU software graphics driver for the guest OS is not required for vSGA.

2.5.1. Installing and Updating the NVIDIA Virtual GPU Manager for vSphere

The NVIDIA Virtual GPU Manager runs on the ESXi host. It is provided in the following formats:

- As a VIB file, which must be copied to the ESXi host and then installed
As an offline bundle that you can import manually as explained in Import Patches Manually in the VMware vSphere documentation

**Caution** NVIDIA Virtual GPU Manager and Guest VM drivers must be matched from the same main driver branch. If you update vGPU Manager to a release from another driver branch, guest VMs will boot with vGPU disabled until their guest vGPU driver is updated to match the vGPU Manager version. Consult Virtual GPU Software for VMware vSphere Release Notes for further details.

2.5.1.1. Installing the NVIDIA Virtual GPU Manager Package for vSphere

To install 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.

Before proceeding with the vGPU Manager installation 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. Use the `esxcli` command to install the vGPU Manager package:

   ```
   [root@esxi:~] esxcli software vib install -v directory/NVIDIA-vGPU-VMware_ESXi_6.0_Host_Driver_410.68-1OEM.600.0.0.2159203.vib
   Installation Result
   Message: Operation finished successfully.
   Reboot Required: false
   VIBs Installed: NVIDIA-vGPU-VMware_ESXi_6.0_Host_Driver_410.68-1OEM.600.0.0.2159203
   VIBs Removed:
   VIBs Skipped:
   ```

   *directory* is the absolute path to the directory that contains the VIB file. You must specify the absolute path even if the VIB file is in the current working directory.

2. Reboot the ESXi host and remove it from maintenance mode.

2.5.1.2. Updating the NVIDIA Virtual GPU Manager Package for vSphere

Update the vGPU Manager VIB package 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.

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. Use the `esxcli` command to update the vGPU Manager package:

   ```
   [root@esxi:~] esxcli software vib update -v directory/NVIDIA-vGPU-
   VMware_ESXi_6.0_Host_Driver_410.68-1OEM.600.0.0.2159203.vib
   ```

   **Installation Result**
   
   Message: Operation finished successfully.
   Reboot Required: false
   VIBs Installed: NVIDIA-vGPU-
   VMware_ESXi_6.0_Host_Driver_410.68-1OEM.600.0.0.2159203
   VIBs Removed: NVIDIA-vGPU-
   VMware_ESXi_6.0_Host_Driver_390.94-1OEM.600.0.0.2159203
   VIBs Skipped:

   `directory` is the path to the directory that contains the VIB file.

2. Reboot the ESXi host and remove it from maintenance mode.

2.5.1.3. Verifying the Installation of the NVIDIA vGPU Software Package for vSphere

After the ESXi host has rebooted, verify the installation of the NVIDIA vGPU software package for vSphere.

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

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

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

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

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

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

   ```
   [root@esxi:~] nvidia-smi
   Fri Nov 16 17:56:22 2018
   +---------------------------------------------------------------------------+
   | NVIDIA-SMI 410.68       Driver Version: 410.71                   |
   +---------------------------------------------------------------------------+
   | GPU   Name     Persistence-M | Bus-Id   Disp.A | Volatile Uncorr. ECC |                  |
   +---------------------------------------------------------------------------+
   | Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
   +---------------------------------------------------------------------------+
   | | | | | | | | |
   +---------------------------------------------------------------------------+
   ```
2.5.2. Configuring VMware vMotion with vGPU for VMware vSphere

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

For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support vGPU migration, see Virtual GPU Software for VMware vSphere Release Notes.

Perform this task for each ESXi host between which you want to migrate VMs configured with vGPU.

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 ESXi host.
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.
2.5.3. Changing the Default Graphics Type in VMware vSphere 6.5 and Later

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

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

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

Change the default graphics type before configuring vGPU. Output from the VM console in the VMware vSphere Web Client is not available for VMs that are running vGPU.
Before changing the default graphics type, ensure that the ESXi host is running and that all VMs on the host are powered off.

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

![Image of vSphere Web Client screen showing Host Graphics settings]

Figure 7  Shared default graphics type

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

   b) In the **Edit Graphics Device Settings** dialog box that opens, select **Shared Direct** and click **OK**.
7. Restart the ESXi host or stop and restart the Xorg service and `nv-hostengine` on the ESXi host.

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

a) Stop the Xorg service.

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

b) Stop `nv-hostengine`.

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

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

d) Start `nv-hostengine`.

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

e) Start the Xorg service.

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

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

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:
2.5.4. Configuring a vSphere VM with NVIDIA vGPU

**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.

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

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

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 vGPU software graphics driver as explained in Installing the NVIDIA vGPU Software Graphics Driver.
2.5.5. Configuring a vSphere VM with VMware vSGA

Virtual Shared Graphics Acceleration (vSGA) is a feature of VMware vSphere that enables multiple virtual machines to share the physical GPUs on ESXi hosts.

If you are configuring a VM to use NVIDIA vGPU, omit this task.

Before configuring a vSphere VM with vSGA, ensure that these prerequisites are met:

- VMware tools are installed on the VM.
- The VM is powered off.
- The NVIDIA Virtual GPU Manager package for vSphere is installed.

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 device list, expand the Video card node and set the following options:
   a) Select the Enable 3D support option.
   b) Set the 3D Renderer to Hardware.
   For more information, see Configure 3D Graphics and Video Cards in the VMware Horizon documentation.
5. Start the VM.
6. After the VM has booted, verify that the VM has been configured correctly with vSGA.
   a) Under the Display Adapter section of Device Manager, confirm that VMware SVGA 3D is listed.
   b) Verify that the virtual machine is using the GPU card.

```bash
# gpwd
```

The output from the command is similar to the following example for a VM named samplevm1:

```
Xserver unix:0, GPU maximum memory 4173824KB
pid 21859, VM samplevm1, reserved 131072KB of GPU memory.
GPU memory left 4042752KB.
```

The memory reserved for the VM and the GPU maximum memory depend on the GPU installed in the host and the 3D memory allocated to the virtual machine.

Installation of the NVIDIA vGPU software graphics driver for the guest OS is not required for vSGA.
2.6. Disabling ECC Memory

Tesla M60, Tesla M6, and GPUs based on the Pascal GPU architecture, for example Tesla P100 or Tesla P4, support error correcting code (ECC) memory for improved data integrity. Tesla M60 and M6 GPUs in graphics mode are supplied with ECC memory disabled by default, but it may subsequently be enabled using `nvidia-smi`. GPUs based on the Pascal GPU architecture are supplied with ECC memory enabled.

However, NVIDIA vGPU does not support ECC memory. If ECC memory is enabled, NVIDIA vGPU fails to start. Therefore, you must ensure that ECC memory is disabled on all GPUs if you are using NVIDIA vGPU.

Before you begin, ensure that NVIDIA Virtual GPU Manager is installed on your hypervisor.

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

```
# nvidia-smi -q

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

Timestamp                           : Tue Dec 19 18:36:45 2017
Driver Version                      : 384.99
Attached GPUs                       : 1
GPU 0000:02:00.0

[...]                

    Ecc Mode               
    Current               : Enabled
    Pending               : Enabled

[...]
```

2. Change the ECC status to off on each GPU for which ECC is enabled.

   ▶ If you want to change the ECC status to off for all GPUs on your host machine, run this command:

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

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

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

id is the index of the GPU 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.

4. Confirm that ECC is now disabled for the GPU.

```
# nvidia-smi -q
```
If you later need to enable ECC on your GPUs, run one of the following commands:

- If you want to change the ECC status to on for all GPUs on your host machine, run this command:

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

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

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

  *id* is the index of the GPU as reported by *nvidia-smi*.

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

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

After changing the ECC status to on, reboot the host.
Chapter 3.
USING GPU PASS-THROUGH

GPU pass-through is used to directly assign an entire physical GPU to one VM, bypassing the NVIDIA Virtual GPU Manager. In this mode of operation, the GPU is accessed exclusively by the NVIDIA driver running in the VM to which it is assigned; the GPU is not shared among VMs.

In pass-through mode, the Tesla P4, Tesla P6, Tesla P40, and Tesla P100 GPUs support error-correcting code (ECC).

GPU pass-through can be used in a server platform alongside NVIDIA vGPU, with some restrictions:

- A physical GPU can host NVIDIA vGPUs, or can be used for pass-through, but cannot do both at the same time. Some hypervisors, for example VMware vSphere ESXi, require a host reboot to change a GPU from pass-through mode to vGPU mode.
- A single VM cannot be configured for both vGPU and GPU pass-through at the same time.
- The performance of a physical GPU passed through to a VM can be monitored only from within the VM itself. Such a GPU cannot be monitored by tools that operate through the hypervisor, such as XenCenter or nvidia-smi (see Monitoring GPU Performance).
- The following BIOS settings must be enabled on your server platform:
  - VT-D/IOMMU
  - SR-IOV in Advanced Options

3.1. Using GPU Pass-Through on Citrix XenServer

You can configure a GPU for pass-through on Citrix XenServer by using XenCenter or by using the xe command.

The following additional restrictions apply when GPU pass-through is used in a server platform alongside NVIDIA vGPU:

- The performance of a physical GPU passed through to a VM cannot be monitored through XenCenter.
Using GPU Pass-Through

- `nvidia-smi` in dom0 no longer has access to the GPU.
- Pass-through GPUs do not provide console output through XenCenter’s **VM Console** tab. Use a remote graphics connection directly into the VM to access the VM’s OS.

### 3.1.1. Configuring a VM for GPU Pass Through by Using XenCenter

Select the **Pass-through whole GPU** option as the GPU type in the VM’s Properties:

![Using XenCenter to configure a pass-through GPU](image)

After configuring a Citrix XenServer VM for GPU pass through, install the NVIDIA graphics driver in the guest OS on the VM as explained in **Installing the NVIDIA vGPU Software Graphics Driver**.

### 3.1.2. Configuring a VM for GPU Pass Through by Using `xe`

Create a `vgpu` object with the `passthrough` vGPU type:

```bash
[root@xenserver ~]# xe vgpu-type-list model-name="passthrough"
uuid (RO) : fa50b0f0-9705-6c59-689e-ea62a3d35237
  vendor-name (RO):
  model-name (RO): passthrough
  framebuffer-size (RO): 0
```

```bash
[root@xenserver ~]# xe vgpu-create vm-uuid=753e77a9-e10d-7679-f674-65c078abb2eb vgpu-type-uuid=fa50b0f0-9705-6c59-689e-ea62a3d35237 gpu-group-uuid=585877ef-5a6c-66af-fc56-7bd525bdc2f6 6aa530ec-8f27-86bd-b8e4-fe4fde8f08f9
```
Caution

Do not assign pass-through GPUs using the legacy `other-config:pci` parameter setting. This mechanism is not supported alongside the XenCenter UI and `xe vgpu` mechanisms, and attempts to use it may lead to undefined results.

After configuring a Citrix XenServer VM for GPU pass through, install the NVIDIA graphics driver in the guest OS on the VM as explained in Installing the NVIDIA vGPU Software Graphics Driver.

3.2. Using GPU Pass-Through on Red Hat Enterprise Linux KVM

You can configure a GPU for pass-through on Red Hat Enterprise Linux Kernel-based Virtual Machine (KVM) by using any of the following tools:

- The Virtual Machine Manager (`virt-manager`) graphical tool
- The `virsh` command
- The QEMU command line

Before configuring a GPU for pass-through on Red Hat Enterprise Linux KVM, ensure that the following prerequisites are met:

- Red Hat Enterprise Linux KVM is installed.
- A virtual disk has been created.

Do not create any virtual disks in `/root`.

- A virtual machine has been created.

3.2.1. Configuring a VM for GPU Pass-Through by Using Virtual Machine Manager (`virt-manager`)

For more information about using Virtual Machine Manager, see the following topics in the documentation for Red Hat Enterprise Linux 7:

- Managing Guests with the Virtual Machine Manager (virt-manager)
- Starting `virt-manager`
- Assigning a PCI Device with virt-manager

1. Start `virt-manager`.
2. In the `virt-manager` main window, select the VM that you want to configure for pass-through.
3. From the Edit menu, choose Virtual Machine Details.
4. In the virtual machine hardware information window that opens, click Add Hardware.
5. In the Add New Virtual Hardware dialog box that opens, in the hardware list on the left, select PCI Host Device.
6. From the Host Device list that appears, select the GPU that you want to assign to the VM and click Finish.

If you want to remove a GPU from the VM to which it is assigned, in the virtual machine hardware information window, select the GPU and click Remove.

After configuring a Red Hat Enterprise Linux KVM for GPU pass through, install the NVIDIA graphics driver in the guest OS on the VM as explained in Installing the NVIDIA vGPU Software Graphics Driver.

### 3.2.2. Configuring a VM for GPU Pass-Through by Using virsh

For more information about using virsh, see the following topics in the documentation for Red Hat Enterprise Linux 7:

- Managing Guest Virtual Machines with virsh
- Assigning a PCI Device with virsh

1. Verify that the vfio-pci module is loaded.

```
# lsmod | grep vfio-pci
```

2. Obtain the PCI device bus/device/function (BDF) of the GPU that you want to assign in pass-through mode to a VM.

```
# lspci | grep NVIDIA
```

The NVIDIA GPUs listed in this example have the PCI device BDFs `85:00.0` and `86:00.0`.

```
# lspci | grep NVIDIA
85:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
86:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
```

3. 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, `85_00_0`.

This example obtains the full identifier of the GPU with the PCI device BDF `85:00.0`.

```
# virsh nodedev-list --cap pci| grep 85_00_0
pci_0000_85_00_0
```
4. Obtain the domain, bus, slot, and function 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_85_00_0`.

This example obtains the domain, bus, slot, and function of the GPU with the PCI device BDF `85:00.0`.

```
# virsh nodedev-dumpxml pci_0000_85_00_0| egrep 'domain|bus|slot|function'
<domain>0x0000</domain>
<bus>0x85</bus>
<slot>0x00</slot>
<function>0x0</function>
<address domain='0x0000' bus='0x85' slot='0x00' function='0x0'/>
```

5. In `virsh`, open for editing the XML file of the VM that you want to assign the GPU to.

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

**vm-name**

The name of the VM to that you want to assign the GPU to.

6. Add a device entry in the form of an `address` element inside the `source` element to assign the GPU to the guest VM.

   You can optionally add a second address element after the `source` element to set a fixed PCI device BDF for the GPU in the guest operating system.

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='domain' bus='bus' slot='slot' function='function'/>
  </source>
  <address type='pci' domain='domain' bus='bus' slot='slot' function='function'/>
</hostdev>
```

**domain**

**bus**

**slot**

**function**

The domain, bus, slot, and function of the GPU, which you obtained in the previous step.

This example adds a device entry for the GPU with the PCI device BDF `85:00.0` and fixes the BDF for the GPU in the guest operating system.

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x85' slot='0x00' function='0x0'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x85' slot='0x05' function='0x0'/>
</hostdev>
```

7. Start the VM that you assigned the GPU to.

```
# virsh start vm-name
```
The name of the VM that you assigned the GPU to.

After configuring a Red Hat Enterprise Linux KVM for GPU pass through, install the NVIDIA graphics driver in the guest OS on the VM as explained in Installing the NVIDIA vGPU Software Graphics Driver.

### 3.2.3. Configuring a VM for GPU Pass-Through by Using the QEMU Command Line

1. Obtain the PCI device bus/device/function (BDF) of the GPU that you want to assign in pass-through mode to a VM.

   ```bash
   # lspci | grep NVIDIA
   
   The NVIDIA GPUs listed in this example have the PCI device BDFs 85:00.0 and 86:00.0.
   
   # lspci | grep NVIDIA
   
   85:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
   86:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
   
2. Add the following option to the QEMU command line:

   ```bash
   -device vfio-pci,host=bdf
   
   bdf
   
   The PCI device BDF of the GPU that you want to assign in pass-through mode to a VM, for example, 85:00.0.
   
   This example assigns the GPU with the PCI device BDF 85:00.0 in pass-through mode to a VM.

   ```bash
   -device vfio-pci,host=85:00.0
   
   After configuring a Red Hat Enterprise Linux KVM for GPU pass through, install the NVIDIA graphics driver in the guest OS on the VM as explained in Installing the NVIDIA vGPU Software Graphics Driver.

### 3.2.4. Preparing a GPU Configured for vGPU for Use in Pass-Through Mode

The mode in which a physical GPU is being used determines the Linux kernel module to which the GPU is bound. If you want to switch the mode in which a GPU is being used, you must unbind the GPU from its current kernel module and bind it to the kernel module for the new mode. After binding the GPU to the correct kernel module, you can then configure it for pass-through.

When the Virtual GPU Manager is installed on a Red Hat Enterprise Linux KVM host, the physical GPUs on the host are bound to the nvidia kernel module. A physical GPU
that is bound to the nvidia kernel module can be used only for vGPU. To enable the GPU to be passed through to a VM, the GPU must be unbound from nvidia kernel module and bound to the vfio-pci kernel module.

Before you begin, ensure that you have the domain, bus, slot, and function of the GPU that you are preparing for use in pass-through mode. For instructions, see Getting the BDF and Domain of a GPU on Red Hat Enterprise Linux KVM.

1. Determine the kernel module to which the GPU is bound by running the lspci command with the \(-k\) option on the NVIDIA GPUs on your host.

   
   # lspci -d 10de: -k
   
   The Kernel driver in use: field indicates the kernel module to which the GPU is bound.

   The following example shows that the NVIDIA Tesla M60 GPU with BDF 06:00.0 is bound to the nvidia kernel module and is being used for vGPU.

   | 06:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60]  
   | (rev a1)  
   | Subsystem: NVIDIA Corporation Device 115e  
   | Kernel driver in use: nvidia

2. To ensure that no clients are using the GPU, acquire the unbind lock of the GPU.

   a) Ensure that no VM is running to which a vGPU on the physical GPU is assigned and that no process running on the host is using that GPU.
   
   Processes on the host that use the GPU include the nvidia-smi command and all processes based on the NVIDIA Management Library (NVML).
   b) Change to the directory in the proc file system that represents the GPU.

   # cd /proc/driver/nvidia/gpus/domain:bus:slot.function
   
   domain    
   bus       
   slot      
   function  

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

   This example changes to the directory in the proc file system that represents the GPU with the domain 0000 and PCI device BDF 06:00.0.

   # cd /proc/driver/nvidia/gpus/0000\:06\:00.0

   c) Write the value 1 to the unbindLock file in this directory.

   # echo 1 > unbindLock
   
   d) Confirm that the unbindLock file now contains the value 1.

   # cat unbindLock
   1

   If the unbindLock file contains the value 0, the unbind lock could not be acquired because a process or client is using the GPU.

3. Unbind the GPU from nvidia kernel module.
a) Change to the `sysfs` directory that represents the `nvidia` kernel module.

```
# cd /sys/bus/pci/drivers/nvidia
```

b) Write the domain, bus, slot, and function of the GPU to the `unbind` file in this directory.

```
# echo domain:bus:slot.function > unbind
domain
bus
slot
function
```

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

This example writes the domain, bus, slot, and function of the GPU with the domain `0000` and PCI device BDF `06:00.0`.

```
# echo 0000:06:00.0 > unbind
```

4. Bind the GPU to the `vfio-pci` kernel module.

a) Change to the `sysfs` directory that contains the PCI device information for the physical GPU.

```
# cd /sys/bus/pci/devices/domain:bus:slot.function
```

```
domain
bus
slot
function
```

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

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

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

b) Write the kernel module name `vfio-pci` to the `driver_override` file in this directory.

```
# echo vfio-pci > driver_override
```

c) Change to the `sysfs` directory that represents the `nvidia` kernel module.

```
# cd /sys/bus/pci/drivers/vfio-pci
```

d) Write the domain, bus, slot, and function of the GPU to the `bind` file in this directory.

```
# echo domain:bus:slot.function > bind
domain
bus
slot
function
```

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

This example writes the domain, bus, slot, and function of the GPU with the domain `0000` and PCI device BDF `06:00.0`.
You can now configure the GPU for use in pass-through mode as explained in Using GPU Pass-Through on Red Hat Enterprise Linux KVM.

### 3.3. Using GPU Pass-Through on Microsoft Windows Server

On supported versions of Microsoft Windows Server with Hyper-V role, you can use Discrete Device Assignment (DDA) to enable a VM to access a GPU directly.

#### 3.3.1. Assigning a GPU to a VM on Microsoft Windows Server with Hyper-V

Perform this task in Windows PowerShell. If you do not know the location path of the GPU that you want to assign to a VM, use Device Manager to obtain it.

Ensure that the following prerequisites are met:

- Windows Server with Desktop Experience and the Hyper-V role are installed and configured on your server platform, and a VM is created.

  For instructions, refer to the following articles on the Microsoft technical documentation site:

  - Install Server with Desktop Experience
  - Install the Hyper-V role on Windows Server 2016
  - Create a virtual switch for Hyper-V virtual machines
  - Create a virtual machine in Hyper-V

- The guest OS is installed in the VM.

- The VM is powered off.

1. Obtain the location path of the GPU that you want to assign to a VM.
   
   a) In the device manager, context-click the GPU and from the menu that pops up, choose Properties.
   
   b) In the Properties window that opens, click the Details tab and in the Properties drop-down list, select Location paths.

   An example location path is as follows:

   ```bash
   PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)
   ```
2. Dismount the GPU from host to make it unavailable to the host so that it can be used solely by the VM.

```powershell
Dismount-VMHostAssignableDevice -LocationPath gpu-device-location -force
```

*gpu-device-location*
   
The location path of the GPU that you obtained in the previous step.

This example dismounts the GPU at the location path `PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)`.

```powershell
Dismount-VMHostAssignableDevice -LocationPath "PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)" -force
```

3. Assign the GPU that you dismounted in the previous step to the VM.

```powershell
Add-VMAssignableDevice -LocationPath gpu-device-location -VMName vm-name
```

*gpu-device-location*
   
The location path of the GPU that you dismounted in the previous step.

*vm-name*
   
The name of the VM to which you are attaching the GPU.

You can assign a pass-through GPU to only one virtual machine at a time.

This example assigns the GPU at the location path `PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)` to the VM VM1.

```powershell
Add-VMAssignableDevice -LocationPath "PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)" -VMName VM1
```

4. Power on the VM.
   
The guest OS should now be able to use the GPU.

After assigning a GPU to a VM, install the NVIDIA graphics driver in the guest OS on the VM as explained in Installing the NVIDIA vGPU Software Graphics Driver.

### 3.3.2. Returning a GPU to the Host OS from a VM on Windows Server with Hyper-V

Perform this task in the Windows PowerShell.

1. List the GPUs that are currently assigned to the virtual machine (VM).

```powershell
Get-VMAssignableDevice -VMName vm-name
```

*vm-name*
   
The name of the VM whose assigned GPUs you want to list.

2. Shut down the VM to which the GPU is assigned.

3. Remove the GPU from VM to which it is assigned.

```powershell
Remove-VMAssignableDevice -LocationPath gpu-device-location -VMName vm-name
```
Using GPU Pass-Through

**gpu-device-location**
The location path of the GPU that you are removing, which you obtained in the previous step.

**vm-name**
The name of the VM from which you are removing the GPU.

This example removes the GPU at the location path `PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)` from the VM VM1.

```
Remove-VMAssignableDevice -LocationPath "PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)" -VMName VM1
```

After the GPU is removed from the VM, it is unavailable to the host operating system (OS) until you remount it on the host OS.

4. Remount the GPU on the host OS.

```
Mount-VMHostAssignableDevice -LocationPath gpu-device-location
```

**gpu-device-location**
The location path of the GPU that you are remounting, which you specified in the previous step to remove the GPU from the VM.

This example remounts the GPU at the location path `PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)` on the host OS.

```
Mount-VMHostAssignableDevice -LocationPath "PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)"
```

The host OS should now be able to use the GPU.

3.4. Using GPU Pass-Through on VMware vSphere

On VMware vSphere, you can use Virtual Dedicated Graphics Acceleration (vDGA) to enable a VM to access a GPU directly. vDGA is a feature of VMware vSphere that dedicates a single physical GPU on an ESXi host to a single virtual machine.

Before configuring a vSphere VM with vDGA, ensure that these prerequisites are met:

- The VM and the ESXi host are configured as explained in Preparing for vDGA Capabilities in the VMware Horizon documentation.
- The VM is powered off.

1. Open the vCenter Web UI.
2. In the vCenter Web UI, right-click the ESXi host and choose Settings.
3. From the Hardware menu, choose PCI Devices and click the Edit icon.
4. Select all NVIDIA GPUs and click OK.
5. Reboot the ESXi host.
6. After the ESXi host has booted, right-click the VM and choose Edit Settings.
7. From the New Device menu, choose PCI Device and click Add.
8. On the page that opens, from the New Device drop-down list, select the GPU.
9. Click Reserve all memory and click OK.
10. Start the VM.

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

- Configuring 3D Rendering for Desktops
- Configure RHEL 6 for vDGA

After configuring a vSphere VM with vDGA, install the NVIDIA graphics driver in the guest OS on the VM as explained in Installing the NVIDIA vGPU Software Graphics Driver.
The process for installing the NVIDIA vGPU software graphics driver depends on the OS that you are using. However, for any OS, the process for installing the driver is the same in a VM configured with vGPU, in a VM that is running pass-through GPU, or on a physical host in a bare-metal deployment.

After you install the NVIDIA vGPU software graphics driver, you can license any NVIDIA vGPU software licensed products that you are using.

4.1. Installing the NVIDIA vGPU Software Graphics Driver on Windows

**Installation in a VM:** 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 vGPU software graphics driver must be installed. Windows guest VMs are supported on all NVIDIA vGPU types.

**Installation on bare metal:** When the physical host is booted before the NVIDIA vGPU software graphics driver is installed, boot and the primary display are handled by an on-board graphics adapter. To install the NVIDIA vGPU software graphics driver, access the Windows desktop on the host by using a display connected through the on-board graphics adapter.

1. Copy the 32-bit or 64-bit NVIDIA Windows driver package to the guest VM or physical host where you are installing the driver.
2. Execute the package to unpack and run the driver installer.
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 or physical host.
   - Exit the installer and reboot the VM or physical host when you are ready.

After the VM or physical host 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.
Installation in a VM: After you install the NVIDIA vGPU software graphics driver, you can license any NVIDIA vGPU software licensed products that you are using. For instructions, refer to Virtual GPU Client Licensing User Guide.

Installation on bare metal: After you install the NVIDIA vGPU software graphics driver, complete the bare-metal deployment as explained in Bare-Metal Deployment.

### 4.2. Installing the NVIDIA vGPU Software Graphics Driver on Linux

**Installation in a VM:** After you create a Linux VM on the hypervisor and boot the VM, install the NVIDIA vGPU software graphics driver in the VM to fully enable GPU operation. 64-bit Linux guest VMs are supported only on Q-series and B-series NVIDIA vGPUs.

**Installation on bare metal:** When the physical host is booted before the NVIDIA vGPU software graphics driver is installed, the *vesa* Xorg driver starts the X server. If a primary display device is connected to the host, use the device to access the desktop. Otherwise, use secure shell (SSH) to log in to the host from a remote host. If the Nouveau driver for NVIDIA graphics cards is present, disable it before installing the NVIDIA vGPU software graphics driver.

Installation of the NVIDIA vGPU software graphics driver for Linux requires:

- Compiler toolchain
- Kernel headers
1. Copy the NVIDIA vGPU software Linux driver package, for example NVIDIA-Linux_x86_64-410.71-grid.run, to the guest VM or physical host where you are installing the driver.

2. Before attempting to run the driver installer, exit the X server and terminate all OpenGL applications.
   
   ▶ On Red Hat Enterprise Linux and CentOS systems, exit the X server by transitioning to runlevel 3:
   
   ```
   [nvidia@localhost ~]$ sudo init 3
   ```
   
   ▶ On Ubuntu platforms, do the following:
   
   1. Use `CTRL-ALT-F1` to switch to a console login prompt.
   2. Log in and shut down the display manager:
      
      ```
      [nvidia@localhost ~]$ sudo service lightdm stop
      ```
   
   3. From a console shell, run the driver installer as the root user.
      
      ```
      sudo sh ./ NVIDIA-Linux_x86_64-352.47-grid.run
      ```
      
      In some instances the installer may fail to detect the installed kernel headers and sources. In this situation, re-run the installer, specifying the kernel source path with the `--kernel-source-path` option:
      
      ```
      sudo sh ./ NVIDIA-Linux_x86_64-352.47-grid.run --kernel-source-path=/usr/src/kernels/3.10.0-229.11.1.el7.x86_64
      ```
   
   4. When prompted, accept the option to update the X configuration file (`xorg.conf`).

   ![Update xorg.conf settings](image)

   **Figure 16 Update xorg.conf settings**

   5. Once installation has completed, select OK to exit the installer.

   6. Verify that the NVIDIA driver is operational.
      
      a) Reboot the system and log in.
      
      b) Run `nvidia-settings`. 
The **NVIDIA X Server Settings** dialog box opens to show that the NVIDIA driver is operational.

![NVIDIA X Server Settings dialog box](image)

**Figure 17 Verifying operation with nvidia-settings**

**Installation in a VM:** After you install the NVIDIA vGPU software graphics driver, you can license any NVIDIA vGPU software licensed products that you are using. For instructions, refer to *Virtual GPU Client Licensing User Guide*.

**Installation on bare metal:** After you install the NVIDIA vGPU software graphics driver, complete the bare-metal deployment as explained in *Bare-Metal Deployment*.
NVIDIA vGPU is a licensed product. When booted on a supported GPU, a vGPU runs at reduced capability until a license is acquired.

The performance of an unlicensed vGPU is restricted as follows:

- Frame rate is capped at 3 frames per second.
- GPU resource allocations are limited, which will prevent some applications from running correctly.
- On vGPUs that support CUDA, CUDA is disabled.

These restrictions are removed when a license is acquired.

After you license NVIDIA vGPU, the VM that is set up to use NVIDIA vGPU is capable of running the full range of DirectX and OpenGL graphics applications.

If licensing is configured, the virtual machine (VM) obtains a license from the license server when a vGPU is booted on these GPUs. The VM retains the license until it is shut down. It then releases the license back to the license server. Licensing settings persist across reboots and need only be modified if the license server address changes, or the VM is switched to running GPU pass through.

For complete information about configuring and using NVIDIA vGPU software licensed features, including vGPU, refer to **Virtual GPU Client Licensing User Guide**.

### 5.1. Licensing an NVIDIA vGPU on Windows

Perform this task from the guest VM to which the vGPU is assigned.

The **NVIDIA Control Panel** tool that you use to perform this task detects that a vGPU is assigned to the VM and, therefore, provides no options for selecting the license type. After you license the vGPU, NVIDIA vGPU software automatically selects the correct type of license based on the vGPU type.

1. Open **NVIDIA Control Panel**:
1. Right-click on the Windows desktop and select **NVIDIA Control Panel** from the menu.
2. Open **Windows Control Panel** and double-click the **NVIDIA Control Panel** icon.

2. In **NVIDIA Control Panel**, select the **Manage License** task in the **Licensing** section of the navigation pane.

   - **If the Licensing section and Manage License task are not displayed in NVIDIA Control Panel, the system has been configured to hide licensing controls in NVIDIA Control Panel. For information about registry settings, see Virtual GPU Client Licensing User Guide.**

   The **Manage License** task pane shows that NVIDIA vGPU is currently unlicensed.

3. In the **Primary License Server** field, enter the address of your primary NVIDIA vGPU software License Server.
   
   The address can be a fully-qualified domain name such as `gridlicense1.example.com`, or an IP address such as `10.31.20.45`.
   
   If you have only one license server configured, enter its address in this field.

4. Leave the **Port Number** field under the **Primary License Server** field unset.
   
   The port defaults to `7070`, which is the default port number used by NVIDIA vGPU software License Server.
5. In the **Secondary License Server** field, enter the address of your secondary NVIDIA vGPU software License Server.
   If you have only one license server configured, leave this field unset.
   The address can be a fully-qualified domain name such as gridlicense2.example.com, or an IP address such as 10.31.20.46.

6. Leave the **Port Number** field under the **Secondary License Server** field unset.
   The port defaults to 7070, which is the default port number used by NVIDIA vGPU software License Server.

7. Click **Apply** to assign the settings.
   The system requests the appropriate license for the current vGPU from the configured license server.

The vGPU within the VM should now exhibit full frame rate, resolution, and display output capabilities. The VM is now capable of running the full range of DirectX and OpenGL graphics applications.

If the system fails to obtain a license, see *Virtual GPU Client Licensing User Guide* for guidance on troubleshooting.

### 5.2. Licensing an NVIDIA vGPU on Linux

Perform this task from the guest VM to which the vGPU is assigned.

The **NVIDIA X Server Settings** tool that you use to perform this task detects that a vGPU is assigned to the VM and, therefore, provides no options for selecting the license type. After you license the vGPU, NVIDIA vGPU software automatically selects the correct type of license based on the vGPU type.

1. Start **NVIDIA X Server Settings** by using the method for launching applications provided by your Linux distribution.
   For example, on Ubuntu Desktop, open the **Dash**, search for **NVIDIA X Server Settings**, and click the **NVIDIA X Server Settings** icon.

2. In the **NVIDIA X Server Settings** window that opens, click **Manage GRID License**.
   The **License Edition** section of the **NVIDIA X Server Settings** window shows that NVIDIA vGPU is currently unlicensed.

3. In the **Primary Server** field, enter the address of your primary NVIDIA vGPU software License Server.
   The address can be a fully-qualified domain name such as gridlicense1.example.com, or an IP address such as 10.31.20.45.
   If you have only one license server configured, enter its address in this field.

4. Leave the **Port Number** field under the **Primary Server** field unset.
   The port defaults to 7070, which is the default port number used by NVIDIA vGPU software License Server.

5. In the **Secondary Server** field, enter the address of your secondary NVIDIA vGPU software License Server.
If you have only one license server configured, leave this field unset. The address can be a fully-qualified domain name such as gridlicense2.example.com, or an IP address such as 10.31.20.46.

6. Leave the Port Number field under the Secondary Server field unset. The port defaults to 7070, which is the default port number used by NVIDIA vGPU software License Server.

7. Click Apply to assign the settings. The system requests the appropriate license for the current vGPU from the configured license server.

The vGPU within the VM should now exhibit full frame rate, resolution, and display output capabilities. The VM is now capable of running the full range of DirectX and OpenGL graphics applications.

If the system fails to obtain a license, see Virtual GPU Client Licensing User Guide for guidance on troubleshooting.
Chapter 6.
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.

6.1. 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.

6.1.1. Removing a XenServer VM’s vGPU configuration

You can remove a virtual GPU assignment from a VM, such that it no longer uses a virtual GPU, by using either XenCenter or the `xe` command.

The VM must be in the powered-off state in order for its vGPU configuration to be modified or removed.

6.1.1.1. Removing a VM’s vGPU configuration by using XenCenter

1. Set the **GPU type** to **None** in the VM’s **GPU Properties**, as shown in Figure 19.
6.1.1.2. Removing a VM’s vGPU configuration by using `xe`

1. Use `vgpu-list` to discover the vGPU object UUID associated with a given VM:

```
[root@xenserver ~]# xe vgpu-list vm-uuid=e71afda4-53f4-3a1b-6c92-a364a7f619c2
```
```
uuid (RO)              : c1c7c43d-4c99-af76-5051-119f1c2b4188
vm-uuid (RO): e71afda4-53f4-3a1b-6c92-a364a7f619c2
```

2. Use `vgpu-destroy` to delete the virtual GPU object associated with the VM:

```
[root@xenserver ~]# xe vgpu-destroy uuid=c1c7c43d-4c99-af76-5051-119f1c2b4188
[root@xenserver ~]#
```

6.1.2. 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.

6.2. Modifying GPU Allocation Policy

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

**breadth-first**

The breadth-first allocation policy attempts to minimize the number of vGPUs running on each physical GPU. Newly created vGPUs are placed on the physical GPU that can support the new vGPU and that has the fewest vGPUs already resident on it. This policy generally leads to higher performance because it attempts to minimize sharing of physical GPUs, but it may artificially limit the total number of vGPUs that can run.

**depth-first**

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

Each hypervisor uses a different GPU allocation policy by default.

- Citrix XenServer uses the depth-first allocation policy
- 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.

6.2.1. Modifying GPU Allocation Policy on Citrix XenServer

You can modify GPU allocation policy on Citrix XenServer by using XenCenter or the `xe` command.

6.2.1.1. Modifying GPU Allocation Policy by Using `xe`

The allocation policy of a GPU group is stored in the `allocation-algorithm` parameter of the `gpu-group` object.

To change the allocation policy of a GPU group, use `gpu-group-param-set`:

```
[root@xenserver ~]# xe gpu-group-param-get uuid=be825ba2-01d7-8d51-9780-f82cfaa64924 param-name=allocation-algorithmdepth-first
[root@xenserver ~]# xe gpu-group-param-set uuid=be825ba2-01d7-8d51-9780-f82cfaa64924 allocation-algorithm=breadth-first
[root@xenserver ~]#
```
6.2.1.2. Modifying GPU Allocation Policy GPU by Using XenCenter

You can modify GPU allocation policy from the GPU tab in XenCenter.

![Modifying GPU placement policy in XenCenter](image)

**Figure 20** Modifying GPU placement policy in XenCenter

6.2.2. Modifying GPU Allocation Policy on VMware vSphere

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

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

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

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

![Host Graphics tab](image)

**Figure 21** 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 22  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.
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

### 6.3. Migrating a VM Configured with vGPU

On some hypervisors, NVIDIA vGPU software supports migration of VMs that are configured with vGPU.

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.
- **VMware vSphere only:** vGPU migration is configured for the host where the VM is running and the destination host. See Configuring VMware vMotion with vGPU for VMware vSphere.
- The destination host has a physical GPU of the same type as the GPU where the vGPU currently resides.

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.

### 6.3.1. Migrating a VM Configured with vGPU on Citrix XenServer

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

For details about which Citrix XenServer versions, NVIDIA GPUs, and guest OS releases support XenMotion with vGPU, see *Virtual GPU Software for Citrix XenServer Release Notes*.

For best performance, the physical hosts should be configured to use the following:

- Shared storage, such as NFS, iSCSI, or Fiberchannel
  
  If shared storage is not used, migration can take a very long time because vDISK must also be migrated.

- 10 GB networking.

1. In Citrix XenCenter, context-click the VM and from the menu that opens, choose **Migrate**.
2. From the list of available hosts, select the destination host to which you want to migrate the VM.
   
   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.
6.3.2. Migrating a VM Configured with vGPU on VMware vSphere

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

For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support suspend and resume, see Virtual GPU Software for VMware vSphere Release Notes.

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

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 Migrate a Virtual Machine to a New Compute Resource in the VMware documentation.

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

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

For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support suspend and resume, see Virtual GPU Software for VMware vSphere Release Notes.

Perform this task in the VMware vSphere web client.

- To suspend a VM, context-click the VM that you want to suspend, and from the context menu that pops up, choose Power > Suspend.
To resume a VM, context-click the VM that you want to resume, and from the context menu that pops up, choose **Power > Power On**.
Chapter 7. MONITORING GPU PERFORMANCE

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

You can use several tools for monitoring GPU performance:

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

7.1. 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 XenServer dom0 shell or VMware ESXi host shell, nvidia-smi reports management information for NVIDIA physical GPUs and virtual GPUs present in the system.

When run from a hypervisor command shell, nvidia-smi will not list any GPU that is currently allocated for GPU pass-through.
From a guest VM that is running Windows or Linux, `nvidia-smi` retrieves usage statistics for vGPUs or pass-through GPUs that are assigned to the VM.

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

### 7.2. Monitoring GPU Performance from a Hypervisor

You can monitor GPU performance from any supported hypervisor by using the NVIDIA System Management Interface `nvidia-smi` command-line utility. On Citrix XenServer platforms, you can also use Citrix XenCenter to monitor GPU performance.

**You cannot** monitor from the hypervisor the performance of GPUs that are being used for GPU pass-through. You can monitor the performance of pass-through GPUs only from within the guest VM that is using them.

#### 7.2.1. 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 XenServer 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.

<table>
<thead>
<tr>
<th>Help Information</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>A list of subcommands supported by the <code>nvidia-smi</code> tool. Note that not all subcommands apply to GPUs that support NVIDIA vGPU software.</td>
<td><code>nvidia-smi -h</code></td>
</tr>
<tr>
<td>A list of all options supported by the <code>vgpu</code> subcommand.</td>
<td><code>nvidia-smi vgpu -h</code></td>
</tr>
</tbody>
</table>

#### 7.2.1.1. 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 Nov 16 09:26:18 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.68                 Driver Version: 410.71                    |
|-------------------------------+----------------------+----------------------+
| 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 | 1889MiB /  8191MiB |    7%  Default |
|   1  Tesla M60           On   | 0000:84:00.0     Off |  926MiB /  8191MiB |    9%  Default |
|   2  Tesla M10           On   | 0000:8A:00.0     Off | 1882MiB /  8191MiB |   12%  Default |
|   3  Tesla M10           On   | 0000:8B:00.0     Off | 10MiB /   8191MiB |   12%  Default |
|   4  Tesla M10           On   | 0000:8C:00.0     Off | 10MiB /   8191MiB |   12%  Default |
|   5  Tesla M10           On   | 0000:8D:00.0     Off | 10MiB /   8191MiB |   12%  Default |
|-------------------------------+----------------------+----------------------+
| Processes:                                                       GPU Memory |
|  GPU  PID  Type  Process name                               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 ~]#  
```

7.2.1.2. 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 Nov 16 09:27:06 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.68                 Driver Version: 410.71                    |
|-------------------------------+--------------------------------+------------+
| GPU Name                      | Bus-Id                         | GPU-Util   |
| vGPU ID  Name                 | VM ID    VM Name               | vGPU-Util  |
|===============================+================================+============|
|   0  Tesla M60                | 0000:83:00.0                   |   7%       |
|   1  Tesla M60                | 0000:8A:00.0                   |   9%       |
|   2  Tesla M10                | 0000:8B:00.0                   |  12%       |
|   3  Tesla M10                | 0000:8C:00.0                   |   0%       |
|   4  Tesla M10                | 0000:8D:00.0                   |   0%       |
7.2.1.3. Getting vGPU Details

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

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

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

7.2.1.4. Monitoring vGPU engine usage

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

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

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D/Compute</td>
<td>sm</td>
</tr>
<tr>
<td>Memory controller bandwidth</td>
<td>mem</td>
</tr>
</tbody>
</table>
Each reported percentage is the percentage of the physical GPU’s capacity that a vGPU is using. For example, a vGPU that uses 20% of the GPU’s graphics engine’s capacity will report 20%.

To modify the reporting frequency, use the \(-l\) or \(--loop\) option.

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

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

### 7.2.1.5. Monitoring vGPU engine usage by applications

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

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

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D/Compute</td>
<td>sm</td>
</tr>
<tr>
<td>Memory controller bandwidth</td>
<td>mem</td>
</tr>
<tr>
<td>Video encoder</td>
<td>enc</td>
</tr>
<tr>
<td>Video decoder</td>
<td>dec</td>
</tr>
</tbody>
</table>
Each reported percentage is the percentage of the physical GPU’s capacity used by an application running on a vGPU that resides on the physical GPU. For example, an application that uses 20% of the GPU’s graphics engine’s capacity will report 20%.

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

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

```
[root@vgpu ~]# nvidia-smi vgpu -p
# GPU   vGPU process        process   sm  mem  enc  dec
# Idx  Id     Id           name    %    %    %    %
0   38127    1528        dwm.exe    0    0    0    0
1   37408    4232  DolphinVS.exe   32   25    0    0
1  257869    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  257969    4552  FurMark.exe   48   37    0    0
0  257869    4432    dwm.exe    0    0    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  257869    4552  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   49    0    0
0  38127    1528    dwm.exe    0    0    0    0
1  257911     656  DolphinVS.exe   16   12    0    0
1  257869    4432  FurMark.exe   16   12    0    0
1  257969    4552  FurMark.exe   64   49    0    0
```

7.2.1.6. Monitoring Encoder Sessions

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
Monitoring GPU Performance

- Encode horizontal resolution
- Encode vertical resolution
- One-second trailing average encoded FPS
- One-second trailing average encode latency in microseconds

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

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

```
[root@vgpu ~]# nvidia-smi vgpu -es
```

<table>
<thead>
<tr>
<th>#</th>
<th>GPU Idx</th>
<th>vGPU Idx</th>
<th>FBC Session Idx</th>
<th>Process Id</th>
<th>Codec</th>
<th>H Res</th>
<th>V Res</th>
<th>Average FPS</th>
<th>Average Encode Latency(us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>424</td>
<td>1977</td>
<td>1997</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>374</td>
<td>1589</td>
<td>1598</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>360</td>
<td>807</td>
<td>817</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>325</td>
<td>1474</td>
<td>1483</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>313</td>
<td>1005</td>
<td>1011</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>329</td>
<td>1912</td>
<td>1921</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>352</td>
<td>1415</td>
<td>1425</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>434</td>
<td>1894</td>
<td>1904</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>362</td>
<td>1818</td>
<td>1828</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>296</td>
<td>1072</td>
<td>1082</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>416</td>
<td>1994</td>
<td>2004</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>444</td>
<td>1912</td>
<td>1922</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>330</td>
<td>1261</td>
<td>1271</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>436</td>
<td>1644</td>
<td>1654</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>344</td>
<td>1500</td>
<td>1510</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>393</td>
<td>1727</td>
<td>1737</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>364</td>
<td>1945</td>
<td>1955</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>555</td>
<td>1653</td>
<td>1663</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>295</td>
<td>925</td>
<td>935</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>372</td>
<td>1869</td>
<td>1879</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>326</td>
<td>2206</td>
<td>2216</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>318</td>
<td>1366</td>
<td>1376</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>464</td>
<td>2015</td>
<td>2025</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>305</td>
<td>1167</td>
<td>1177</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>445</td>
<td>1892</td>
<td>1897</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>361</td>
<td>906</td>
<td>916</td>
</tr>
<tr>
<td>1</td>
<td>21211</td>
<td>2</td>
<td>2308</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>353</td>
<td>1436</td>
<td>1446</td>
</tr>
<tr>
<td>1</td>
<td>21206</td>
<td>3</td>
<td>2424</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>354</td>
<td>1798</td>
<td>1803</td>
</tr>
<tr>
<td>1</td>
<td>22011</td>
<td>1</td>
<td>3676</td>
<td>H.264</td>
<td>1920</td>
<td>1080</td>
<td>373</td>
<td>1310</td>
<td>1315</td>
</tr>
</tbody>
</table>

&[^C[root@vgpu ~]#]

7.2.1.7. Monitoring Frame Buffer Capture (FBC) Sessions

To monitor the FBC sessions for processes running on multiple vGPUs, run `nvidia-smi vgpu` with the `-fs` or `--fbcsessions` option.

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

- GPU ID
- vGPU ID
- FBC session ID
- PID of the process in the VM that created the FBC session
- Display ordinal associated with the FBC session
- FBC session type
- FBC session flags
- Capture mode
- Maximum horizontal resolution supported by the session
- Maximum vertical resolution supported by the session
- Horizontal resolution requested by the caller in the capture call
- Vertical resolution requested by the caller in the capture call
- Moving average of new frames captured per second by the session
- Moving average new frame capture latency in microseconds for the session

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

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

```
[root@vgpu ~]# nvidia-smi vgpu -fs
```

<table>
<thead>
<tr>
<th>#</th>
<th>GPU</th>
<th>vGPU</th>
<th>Session</th>
<th>Process</th>
<th>Display</th>
<th>Session Diff. Map</th>
<th>Class. Map</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idx</td>
<td>Id</td>
<td>Id</td>
<td>Id</td>
<td>Ordinal</td>
<td>Type</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>Idx</td>
<td>Id</td>
<td>Id</td>
<td>Id</td>
<td>Ordinal</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>Idx</td>
<td>Id</td>
<td>Id</td>
<td>Id</td>
<td>Ordinal</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>Idx</td>
<td>Id</td>
<td>Id</td>
<td>Id</td>
<td>Ordinal</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>Idx</td>
<td>Id</td>
<td>Id</td>
<td>Id</td>
<td>Ordinal</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>Idx</td>
<td>Id</td>
<td>Id</td>
<td>Id</td>
<td>Ordinal</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td>#</td>
<td>GPU Idx</td>
<td>vGPU Idx</td>
<td>Session Id</td>
<td>Process Id</td>
<td>Display Id</td>
<td>Session Diff. Map</td>
<td>Class. Map</td>
</tr>
<tr>
<td>---</td>
<td>---------</td>
<td>----------</td>
<td>------------</td>
<td>------------</td>
<td>-----------</td>
<td>------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3251634178</td>
<td>1</td>
<td>3984</td>
<td>0</td>
<td>ToSys</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3251634178</td>
<td>1</td>
<td>3984</td>
<td>0</td>
<td>ToSys</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3251634178</td>
<td>1</td>
<td>3984</td>
<td>0</td>
<td>ToSys</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3251634178</td>
<td>1</td>
<td>3984</td>
<td>0</td>
<td>ToSys</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3251634178</td>
<td>1</td>
<td>3984</td>
<td>0</td>
<td>ToSys</td>
<td>Disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Idx</td>
<td>Mode</td>
<td>Res</td>
<td>Res</td>
<td>Res</td>
<td>Res</td>
<td>Ordinal</td>
<td>Type</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 0 0 0 0</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 0 0 0 0</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 0 0 0 0</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 0 0 0 0</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 1600 900 135 7400</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 1600 900 227 4403</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 1600 900 227 4403</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 0 0 0 0</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>ToSys Blocking 4096 2160 0 0 0 0</td>
<td>ToSys Disabled</td>
<td>Disabled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### 7.2.1.8. Listing Supported vGPU Types

To list the virtual GPU types that the GPUs in the system support, run `nvidia-smi vgpu` with the `-s` or `--supported` option.
To limit the retrieved information to a subset of the GPUs on the platform, use the –i or --id option to select one or more vGPUs.

```bash
[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:

```bash
[root@vgpu ~]# nvidia-smi vgpu -s -i 0 -v | less
GPU 00000000:83:00.0
  vGPU Type ID              : 0xb
  Name                  : GRID M60-0B
  Class                 : NVS
  Max Instances         : 16
  Device ID             : 0x13f210de
  Sub System ID         : 0x13f21176
  FB Memory             : 512 MiB
  Display Heads         : 2
  Maximum X Resolution  : 2560
  Maximum Y Resolution  : 1600
  Frame Rate Limit      : 45 FPS
  GRID License          : GRID-Virtual-PC,2.0;GRID-Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0;Quadro-Virtual-DWS,5.0
  vGPU Type ID              : 0xc
  Name                  : GRID M60-0Q
  Class                 : Quadro
  Max Instances         : 16
  Device ID             : 0x13f210de
  Sub System ID         : 0x13f21176
  FB Memory             : 512 MiB
  Display Heads         : 2
  Maximum X Resolution  : 2560
  Maximum Y Resolution  : 1600
  Frame Rate Limit      : 60 FPS
  GRID License          : GRID-Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0;Quadro-Virtual-DWS,5.0
  vGPU Type ID              : 0xd
  Name                  : GRID M60-1A
  Class                 : NVS
  Max Instances         : 8
```

### 7.2.1.9. 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 according to the vGPUs that are already running on each GPU.
To limit the retrieved information to a subset of the GPUs on the platform, use the \(-i\) or \(--i\) option to select one or more vGPUs.

```bash
[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.

### 7.2.2. Using Citrix XenCenter to monitor GPU performance

If you are using Citrix XenServer as your hypervisor, you can monitor GPU performance in XenCenter.

1. Click on a server’s **Performance** tab.
2. Right-click on the graph window, then select **Actions** and **New Graph**.
3. Provide a name for the graph.
4. In the list of available counter resources, select one or more GPU counters.

Counters are listed for each physical GPU not currently being used for GPU pass-through.

![Graph](image)

**Figure 24** Using Citrix XenCenter to monitor GPU performance
7.3. 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.

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

In VMs that are running Windows and 64-bit editions of Linux, you can use the `nvidia-smi` command to retrieve statistics for the total usage by all applications running in the VM and usage by individual applications of the following resources:

- GPU
- Video encoder
- Video decoder
- Frame buffer

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

```
nvidia-smi dmon
```

The following example shows the result of running `nvidia-smi dmon` from within a Windows guest VM.
Figure 25  Using `nvidia-smi` from a Windows guest VM to get total resource usage by all applications

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

```
nvidia-smi pmon
```
7.3.2. Using Windows Performance Counters to monitor GPU performance

In Windows VMs, GPU metrics are available as Windows Performance Counters through the NVIDIA GPU object.

Any application that is enabled to read performance counters can access these metrics. You can access these metrics directly through the Windows Performance Monitor application that is included with the Windows OS.

The following example shows GPU metrics in the Performance Monitor application.
On vGPUs, the following GPU performance counters read as 0 because they are not applicable to vGPUs:

- % Bus Usage
- % Cooler rate
- Core Clock MHz
- Fan Speed
- Memory Clock MHz
- PCI-E current speed to GPU Mbps
- PCI-E current width to GPU
- PCI-E downstream width to GPU
- Power Consumption mW
- Temperature C

7.3.3. Using NVWMI to monitor GPU performance

In Windows VMs, Windows Management Instrumentation (WMI) exposes GPU metrics in the `ROOT\CIMV2\NV` namespace through NVWMI. NVWMI is included with the NVIDIA driver package. After the driver is installed, NVWMI help information in Windows Help format is available as follows:

```
C:\Program Files\NVIDIA Corporation\NVIDIA WMI Provider>nvwmi.chm
```
Any WMI-enabled application can access these metrics. The following example shows GPU metrics in the third-party application WMI Explorer, which is available for download from the CodePlex WMI Explorer page.

![WMI Explorer](image)

**Figure 28 Using WMI Explorer to monitor GPU performance**

On vGPUs, some instance properties of the following classes do not apply to vGPUs:
- Ecc
- Gpu
- PcieLink

<table>
<thead>
<tr>
<th>Ecc Instance Property</th>
<th>Value reported on vGPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSupported</td>
<td>False</td>
</tr>
<tr>
<td>isWritable</td>
<td>False</td>
</tr>
<tr>
<td>isEnabled</td>
<td>False</td>
</tr>
<tr>
<td>isEnabledByDefault</td>
<td>False</td>
</tr>
<tr>
<td>aggregateDoubleBitErrors</td>
<td>0</td>
</tr>
</tbody>
</table>
### Ecc Instance Property

<table>
<thead>
<tr>
<th>Ecc Instance Property</th>
<th>Value reported on vGPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregateSingleBitErrors</td>
<td>0</td>
</tr>
<tr>
<td>currentDoubleBitErrors</td>
<td>0</td>
</tr>
<tr>
<td>currentSingleBitErrors</td>
<td>0</td>
</tr>
</tbody>
</table>

### Gpu Instance properties that do not apply to vGPUs

<table>
<thead>
<tr>
<th>Gpu Instance Property</th>
<th>Value reported on vGPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpuCoreClockCurrent</td>
<td>-1</td>
</tr>
<tr>
<td>memoryClockCurrent</td>
<td>-1</td>
</tr>
<tr>
<td>pciDownstreamWidth</td>
<td>0</td>
</tr>
<tr>
<td>pcieGpu.curGen</td>
<td>0</td>
</tr>
<tr>
<td>pcieGpu.curSpeed</td>
<td>0</td>
</tr>
<tr>
<td>pcieGpu.curWidth</td>
<td>0</td>
</tr>
<tr>
<td>pcieGpu.maxGen</td>
<td>1</td>
</tr>
<tr>
<td>pcieGpu.maxSpeed</td>
<td>2500</td>
</tr>
<tr>
<td>pcieGpu.maxWidth</td>
<td>0</td>
</tr>
<tr>
<td>power</td>
<td>-1</td>
</tr>
<tr>
<td>powerSampleCount</td>
<td>-1</td>
</tr>
<tr>
<td>powerSamplingPeriod</td>
<td>-1</td>
</tr>
<tr>
<td>verVBIOS.orderedValue</td>
<td>0</td>
</tr>
<tr>
<td>verVBIOS.strValue</td>
<td>-</td>
</tr>
<tr>
<td>verVBIOS.value</td>
<td>0</td>
</tr>
</tbody>
</table>

### PcieLink instance properties that do not apply to vGPUs

No instances of PcieLink are reported for vGPU.
This chapter describes Citrix XenServer advanced vGPU management techniques using XenCenter and `xe` command line operations.

8.1. Management objects for GPUs

XenServer uses four underlying management objects for GPUs: physical GPUs, vGPU types, GPU groups, and vGPUs. These objects are used directly when managing vGPU by using `xe`, and indirectly when managing vGPU by using XenCenter.

8.1.1. pgpu - Physical GPU

A `pgpu` object represents a physical GPU, such as one of the multiple GPUs present on a Tesla M60 or M10 card. XenServer automatically creates `pgpu` objects at startup to represent each physical GPU present on the platform.

8.1.1.1. Listing the pgpu Objects Present on a Platform

To list the physical GPU objects present on a platform, use `xe pgpu-list`.

For example, this platform contains a Tesla P40 card with a single physical GPU and a Tesla M60 card with two physical GPUs:

```plaintext
[root@xenserver ~]# xe pgpu-list
uuid (RO): f76d1c90-e443-4bfc-8f26-7959a7c85c68
  vendor-name (RO): NVIDIA Corporation
  device-name (RO): GP102GL [Tesla P40]
  gpu-group-uuid (RW): 134a7b71-5ceb-8066-ef1b-3b319fb2bef3

uuid (RO): 4c5e05d9-60fa-4fe5-9cfc-c641e95c8e85
  vendor-name (RO): NVIDIA Corporation
  device-name (RO): GM204GL [Tesla M60]
  gpu-group-uuid (RW): 3df80574-c303-f020-efb3-342f969da5de

uuid (RO): 4960e63c-c9fe-4a25-add4-ee697263e04c
  vendor-name (RO): NVIDIA Corporation
  device-name (RO): GM204GL [Tesla M60]
  gpu-group-uuid (RW): d32560f2-2158-42f9-d201-511691e1cb2b
[root@xenserver ~]#
```
8.1.1.2. Viewing Detailed Information About a pgpu Object

To view detailed information about a pgpu, use xe pgpu-param-list:

[root@xenserver ~]# xe pgpu-param-list uuid=4960e63c-c9fe-4a25-add4-ee697263e04c

uuid (RO)                        : 4960e63c-c9fe-4a25-add4-ee697263e04c
vendor-name (RO)                 : NVIDIA Corporation
device-name (RO)                 : GM204GL [Tesla M60]
dom0-access (RO)                 : enabled
is-system-display-device (RO)    : false
gpu-group-uuid (RW)              : d32560f2-2158-42f9-d201-511691e1cb2b
gpu-group-name-label (RO)        : 86:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
host-uuid (RO)                   : b55452df-1ee4-4e4e-bd97-3ae97b2123a
host-name-label (RO)             : xs7.1
pci-id (RO)                      : 0000:86:00.0
dependencies (SRO):              
other-config (MRW):              
supported-VGPU-types (RO):       5b9acd25-06fa-43e1-8b53-c35bceb8515c; 16326fcf-543f-4473-a4e-e2d30516a277b; 0f9fc39a-0758-43c8-88cc-54c8491aa4d4; cecb2033-3b4a-437c-a0c0-c4dfe692d9b; 095d8939-5f84-405d-a39a-684738f9b57; 56c335be-4036-4a38-816c-c246a0556ac; ef0a94fd-2230-4fd4-ae0-d6d3f6ced4ef; 11615f73-47b8-4494-806e-2a7b5e17be; db86f2ac-f548-4c0-804b-9133c9da8090; a33189f1-1417-4593-a77d-9784f259b53; 3f4a7373-3682-4897-a7ba-6334519f4c19; 99900aab-42b0-4cc-8382-560ff66b0231
enabled-VGPU-types (SRW):        5b9acd25-06fa-43e1-8b53-c35bceb8515c; 16326fcf-543f-4473-a4e-e2d30516a277b; 0f9fc39a-0758-43c8-88cc-54c8491aa4d4; cecb2033-3b4a-437c-a0c0-c4dfe692d9b; 095d8939-5f84-405d-a39a-684738f9b57; 56c335be-4036-4a38-816c-c246a0556ac; ef0a94fd-2230-4fd4-ae0-d6d3f6ced4ef; 11615f73-47b8-4494-806e-2a7b5e17be; db86f2ac-f548-4c0-804b-9133c9da8090; a33189f1-1417-4593-a77d-9784f259b53; 3f4a7373-3682-4897-a7ba-6334519f4c19; 99900aab-42b0-4cc-8382-560ff66b0231
resident-VGPUs (RO):             

[8.1.1.3. Viewing physical GPUs in XenCenter

To view physical GPUs in XenCenter, click on the server's GPU tab:
8.1.2. **vgpu-type - Virtual GPU Type**

A vgpu-type represents a type of virtual GPU, such as M60-0B, P40-8A, and P100-16Q. An additional, pass-through vGPU type is defined to represent a physical GPU that is directly assignable to a single guest VM.

XenServer automatically creates vgpu-type objects at startup to represent each virtual type supported by the physical GPUs present on the platform.

8.1.2.1. Listing the vgpu-type Objects Present on a Platform

To list the vgpu-type objects present on a platform, use `xe vgpu-type-list`.

For example, as this platform contains Tesla P100, Tesla P40, and Tesla M60 cards, the vGPU types reported are the types supported by these cards:

```
[root@xenserver ~]# xe vgpu-type-list
uuid ( RO)              : d27f84a2-53f8-4430-ad15-0eca225cd974
          vendor-name ( RO): NVIDIA Corporation
          model-name ( RO): GRID P40-12A
          max-heads ( RO): 1
          max-resolution ( RO): 1280x1024

uuid ( RO)              : 57bb231f-f61b-408e-a0c0-106bddd91019
          vendor-name ( RO): NVIDIA Corporation
          model-name ( RO): GRID P40-3Q
          max-heads ( RO): 4
          max-resolution ( RO): 4096x2160

uuid ( RO)              : 9b2eaba5-565f-4cb4-ad9b-6347c6b03e93
          vendor-name ( RO): NVIDIA Corporation
          model-name ( RO): GRID P40-2Q
```
max-heads (RO): 4
max-resolution (RO): 4096x2160

uuid (RO): af593219-0800-42da-a51d-d13b35f589e1
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-4A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 5b9acd25-06fa-43e1-8b53-c35bceb8515c
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): passthrough
  max-heads (RO): 0
  max-resolution (RO): 0x0

uuid (RO): af121387-0b58-43e1-8d04-fe0305e4308f
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-3A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 3b28a628-fd6c-4cda-80165699229e
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P100-4Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO): 99900aab-42b0-4cc4-560ff6b60231
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-1Q
  max-heads (RO): 2
  max-resolution (RO): 4096x2160

uuid (RO): 0f9c39a-0758-43c8-54c8491aa4d4
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-4A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 4017c9dd-373f-431a-b36f-50e4e5c9f0c0
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-6A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 125fbbdf-406e-4d7c-a7536aa1a838
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-24A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 88162a34-1151-49d3-98ae-afcd963f3105
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-2A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024
uuid (RO) : ad00a95c-d066-4158-b361-487abf57dd30
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-1A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO) : 11615f73-47b8-4494-806e-2a7b5e1d7bea
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-0Q
  max-heads (RO): 2
  max-resolution (RO): 2560x1600

uuid (RO) : 6ea0cd56-526c-4966-8f53-7e1721b95a5c
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-4Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO) : 095d8939-5f84-405d-a39a-684738f9b957
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-4Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO) : 9626e649-6802-4396-976d-94c0ead1f835
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-12Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO) : a33189f1-1417-4596-978c-f25b953
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-0B
  max-heads (RO): 2
  max-resolution (RO): 2560x1600

uuid (RO) : dbd8f2ac-f548-4c40-804b-9133cfda8090
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-1A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO) : ef0a94fd-2230-4fd4-aee0-d6d3f6ced4ef
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-8Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO) : 67fa06ab-554e-452b-a66e-a4048a5bfdf7
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-6Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO) : 739d7b8e-50e2-48a1-ae0d-5047aa490f0e
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-8A
  max-heads (RO): 1
max-resolution (RO): 1280x1024

uuid (RO): 9fb62f31-7dfb-46f8-a4a9-cca8db48147e
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P100-8Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO): 56c335be-4036-4a38-816c-c246a60556ac
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-1B
  max-heads (RO): 4
  max-resolution (RO): 2560x1600

uuid (RO): 3f437337-3682-4897-a7ba-6334519f4c19
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-8A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 25dbb2d3-a074-4f9f-92ce-b42d8b3d1de2
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-1B
  max-heads (RO): 4
  max-resolution (RO): 2560x1600

uuid (RO): cecb2033-3b4a-437c-a0c0-c9dfdb692d9b
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-2Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO): 16326fcb-543f-4473-a4ae-2d30516a2779
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID M60-2A
  max-heads (RO): 1
  max-resolution (RO): 1280x1024

uuid (RO): 7ca2399f-89ab-49dd-bf96-75071ced28fc
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-24Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO): 9611a3f4-d130-4a66-a61b-21d4a2ca4663
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-8Q
  max-heads (RO): 4
  max-resolution (RO): 4096x2160

uuid (RO): d0e4a116-a944-42ef-a8dc-62a54c4d2d77
  vendor-name (RO): NVIDIA Corporation
  model-name (RO): GRID P40-1Q
  max-heads (RO): 2
  max-resolution (RO): 4096x2160

[root@xenserver ~]#
8.1.2.2. Viewing Detailed Information About a vgpu-type Object

To see detailed information about a vgpu-type, use `xe vgpu-type-param-list`:

```bash
[root@xenserver ~]# xe vgpu-type-param-list uuid=7ca2399f-89ab-49dd-bf96-75071ced28fc

<table>
<thead>
<tr>
<th>uuid ( RO)</th>
<th>7ca2399f-89ab-49dd-bf96-75071ced28fc</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor-name ( RO)</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>model-name ( RO)</td>
<td>GRID P40-24Q</td>
</tr>
<tr>
<td>framebuffer-size ( RO)</td>
<td>24092082176</td>
</tr>
<tr>
<td>max-heads ( RO)</td>
<td>4</td>
</tr>
<tr>
<td>max-resolution ( RO)</td>
<td>4096x2160</td>
</tr>
<tr>
<td>supported-on-PGPUs ( RO)</td>
<td>f76d1c90-e443-4bfc-8f26-7959a7c85c68</td>
</tr>
<tr>
<td>enabled-on-PGPUs ( RO)</td>
<td>f76d1c90-e443-4bfc-8f26-7959a7c85c68</td>
</tr>
<tr>
<td>supported-on-GPU-groups ( RO)</td>
<td>134a7b71-5ceeb-8066-ef1b-3b319fb2bef3</td>
</tr>
<tr>
<td>enabled-on-GPU-groups ( RO)</td>
<td>134a7b71-5ceeb-8066-ef1b-3b319fb2bef3</td>
</tr>
<tr>
<td>VGPU-uuids ( RO)</td>
<td></td>
</tr>
<tr>
<td>experimental ( RO)</td>
<td>false</td>
</tr>
</tbody>
</table>
```

8.1.3. gpu-group - collection of physical GPUs

A gpu-group is a collection of physical GPUs, all of the same type. XenServer automatically creates gpu-group objects at startup to represent the distinct types of physical GPU present on the platform.

8.1.3.1. Listing the gpu-group Objects Present on a Platform

To list the gpu-group objects present on a platform, use `xe gpu-group-list`.

For example, a system with a single Tesla P100 card, a single Tesla P40 card, and two Tesla M60 cards contains a single GPU group of type Tesla P100, a single GPU group of type Tesla P40, and two GPU groups of type Tesla M60:

```bash
[root@xenserver ~]# xe gpu-group-list

<table>
<thead>
<tr>
<th>uuid ( RO)</th>
<th>3d652a59-beaf-ddb3-3b19-c8c77ef60605</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-label ( RW)</td>
<td>Group of NVIDIA Corporation GP100GL [Tesla P100 PCIe 16GB] GPUs</td>
</tr>
<tr>
<td>name-description ( RW):</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uuid ( RO)</th>
<th>3df80574-c303-f020-efb3-342f969da5de</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-label ( RW)</td>
<td>NVIDIA Corporation GM204GL [Tesla M60]</td>
</tr>
<tr>
<td>name-description ( RW):</td>
<td>NVIDIA Corporation GM204GL [Tesla M60] (rev al)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uuid ( RO)</th>
<th>134a7b71-5ceeb-8066-ef1b-3b319fb2bef3</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-label ( RW)</td>
<td>NVIDIA Corporation GP102GL [Tesla P40]</td>
</tr>
<tr>
<td>name-description ( RW):</td>
<td>NVIDIA Corporation GP102GL [Tesla P40] (rev al)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uuid ( RO)</th>
<th>d32560f2-2158-42f9-d201-511691e1cb2b</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-label ( RW)</td>
<td>NVIDIA Corporation GM204GL [Tesla M60]</td>
</tr>
<tr>
<td>name-description ( RW):</td>
<td>NVIDIA Corporation GM204GL [Tesla M60] (rev al)</td>
</tr>
</tbody>
</table>
```
8.1.3.2. Viewing Detailed Information About a gpu-group Object

To view detailed information about a gpu-group, use `xe gpu-group-param-list`:

```
[root@xenserver ~]# xe gpu-group-param-list uuid=134a7b71-5ceb-8066-ef1b-3b319f2b2ef3
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>134a7b71-5ceb-8066-ef1b-3b319f2b2ef3</td>
</tr>
<tr>
<td>name-label (RW)</td>
<td>87:00.0 3D controller: NVIDIA Corporation</td>
</tr>
<tr>
<td>name-description (RW)</td>
<td>87:00.0 3D controller: NVIDIA Corporation</td>
</tr>
<tr>
<td>VGPU-uuids (SRO)</td>
<td>101fb062-427f-1999-9e90-5a914075e9ca</td>
</tr>
<tr>
<td>supported-VGPU-types (RO)</td>
<td>d0e4a116-a944-42ef-a8dc-62a54cdedce</td>
</tr>
<tr>
<td>allocation-algorithm (RW)</td>
<td>depth-first</td>
</tr>
</tbody>
</table>

8.1.4. vgpu - Virtual GPU

A vgpu object represents a virtual GPU. Unlike the other GPU management objects, vgpu objects are not created automatically by XenServer. Instead, they are created as follows:

- When a VM is configured through XenCenter or through `xe` to use a vGPU
- By cloning a VM that is configured to use vGPU, as explained in Cloning vGPU-Enabled VMs

8.2. Creating a vGPU using `xe`

Use `xe vgpu-create` to create a vgpu object, specifying the type of vGPU required, the GPU group it will be allocated from, and the VM it is associated with:

```
[root@xenserver ~]# xe vgpu-create vm-uuid=e71afda4-53f4-3a1b-6c92-a364a7f6e9c2 gpu-group-uuid=be825ba2-01d7-8d51-9780-f82cfaa64924 vgpu-type-uuid=3f318889-7508-c9fd-7134-003d4d05ae56b73cb3d30-096f-8a9a-523e-a80062f4ca7
```

www.nvidia.com
Virtual GPU Software
Creating the vgpu object for a VM does not immediately cause a virtual GPU to be created on a physical GPU. Instead, the vgpu object is created whenever its associated VM is started. For more details on how vGPUs are created at VM startup, see Controlling vGPU allocation.

The owning VM must be in the powered-off state in order for the vgpu-create command to succeed.

A vgpu object’s owning VM, associated GPU group, and vGPU type are fixed at creation and cannot be subsequently changed. To change the type of vGPU allocated to a VM, delete the existing vgpu object and create another one.

8.3. Controlling vGPU allocation

Configuring a VM to use a vGPU in XenCenter, or creating a vgpu object for a VM using xe, does not immediately cause a virtual GPU to be created; rather, the virtual GPU is created at the time the VM is next booted, using the following steps:

- The GPU group that the vgpu object is associated with is checked for a physical GPU that can host a vGPU of the required type (i.e. the vgpu object’s associated vgpu-type). Because vGPU types cannot be mixed on a single physical GPU, the new vGPU can only be created on a physical GPU that has no vGPUs resident on it, or only vGPUs of the same type, and less than the limit of vGPUs of that type that the physical GPU can support.
- If no such physical GPUs exist in the group, the vgpu creation fails and the VM startup is aborted.
- Otherwise, if more than one such physical GPU exists in the group, a physical GPU is selected according to the GPU group’s allocation policy, as described in Modifying GPU Allocation Policy.

8.3.1. Determining the Physical GPU on Which a Virtual GPU is Resident

The vgpu object’s resident-on parameter returns the UUID of the pgpu object for the physical GPU the vGPU is resident on.

To determine the physical GPU that a virtual GPU is resident on, use vgpu-param-get:

```
[root@xenserver ~]# xe vgpu-param-get uuid=101fb062-427f-1999-9e90-5a914075e9ca param-name=resident-on
f76d1c90-e443-4bfc-8f26-7959a7c85c68
```

```
[root@xenserver ~]# xe pgpu-param-list uuid=f76d1c90-e443-4bfc-8f26-7959a7c85c68

uuid ( RO)                        : f76d1c90-e443-4bfc-8f26-7959a7c85c68
vendor-name ( RO): NVIDIA Corporation
device-name ( RO): GP102GL [Tesla P40]
gpu-group-uuid ( RW): 134a7b71-5ceb-8066-ef1b-3b319fb2bef3
gpu-group-name-label ( RO): 87:00.0 3D controller: NVIDIA Corporation GP102GL [TESLA P40] (rev a1)
```
If the vGPU is not currently running, the resident-on parameter is not instantiated for the vGPU, and the vgpu-param-get operation returns:

<not in database>

8.3.2. Controlling the vGPU types enabled on specific physical GPUs

Physical GPUs support several vGPU types, as defined in Supported GPUs and the “pass-through” type that is used to assign an entire physical GPU to a VM (see Using GPU Pass-Through on Citrix XenServer).

8.3.2.1. Controlling vGPU types enabled on specific physical GPUs by using XenCenter

To limit the types of vGPU that may be created on a specific physical GPU:

1. Open the server’s GPU tab in XenCenter.
2. Select the box beside one or more GPUs on which you want to limit the types of vGPU.
3. Select Edit Selected GPUs.
8.3.2.2. Controlling vGPU Types Enabled on Specific Physical GPUs by Using `xe`

The physical GPU's `pgpu` object's `enabled-vGPU-types` parameter controls the vGPU types enabled on specific physical GPUs.

To modify the `pgpu` object's `enabled-vGPU-types` parameter, use `xe pgpu-param-set`:

```
[root@xenserver ~]# xe pgpu-param-list uuid=cb08aaae-8e5a-47cb-888e-60dcc73c01d3

  vendor-name (RO): NVIDIA Corporation
device-name (RO): GP102GL [Tesla P40]
domO-access (RO): enabled
is-system-display-device (RO): false
gpu-group-uuid (RW): bfe16f3d-c526-05f3-e6f4-951485ef3b49
gpu-group-name-label (RO): 87:00.0 3D controller: NVIDIA Corporation GP102GL [Tesla P40] (rev al)
host-uuid (RO): fdeb6bbb-e460-4c1f-ad43-49a8c8c12040
host-name-label (RO): xs-72
pci-id (RO): 0000:87:00.0
dependencies (SRO): [other-config (MRW):
supported-VGPU-types (RO): 23e6b80b-1e5e-4c33-bedb-e6d1ae472f6c;
f583e39-2540-440d-a0ee-de9f0783abf; a1e84e6f-4d05-4322-b040-667ce77d78a8;
adel9a9-84e1-435f-b0e9-14c16e212f2b; 2560d066-054a-48a9-a44d-3f3f90493a00;
47858f38-045d-4a05-9b1c-9128fcee6b0ab; 1f52b7f6-493f-442b-abe2-94a6f49d49ce;
788b8e44-09ae-a4ac-8a9e-2b07ca585842; 18ed7e7-e8b7-496e-9784-8bae353ca3a3;
43853b8b-c4e5-4e39-95ff-c9ba2e8e484; cc35b9b-b4b3-400d-8c52-811948bb7f8c4;
8ed175a-e5df-4609-83ff-5f9bca9aace2; 840389a0-f511-4f90-8153-8a47d8a85e9e;
a2047242-da47-4613-a338-1d1f530d7cb9; 299e47c2-8f01-4edf-a331-e29d8b41b6; 4c95c636-06e6-47e-8a49-14b37d08922; 0524a5d0-7160-48c5-a9e1-cc33e66c5e0;
09043eb2-6d67-4443-b312-25688b7f13e012
enabled-VGPU-types (SRW): 23e6b80b-1e5e-4c33-bedb-e6d1ae472f6c;
f583e39-2540-440d-a0ee-de9f0783abf; a1e84e6f-4d05-4322-b040-667ce77d78a8;
adel9a9-84e1-435f-b0e9-14c16e212f2b; 2560d066-054a-48a9-a44d-3f3f90493a00;
```
8.3.3. Creating vGPUs on Specific Physical GPUs

To precisely control allocation of vGPUs on specific physical GPUs, create separate GPU groups for the physical GPUs you wish to allocate vGPUs on. When creating a virtual GPU, create it on the GPU group containing the physical GPU you want it to be allocated on.

For example, to create a new GPU group for the physical GPU at PCI bus ID 0000:87:00.0, follow these steps:

1. Create the new GPU group with an appropriate name:

   ```
   [root@xenserver ~]# xe gpu-group-create name-label="GRID P40 87:0.0"
   3f870244-41da-469f-71f3-22bc6d700e71
   [root@xenserver ~]#
   ```

2. Find the UUID of the physical GPU at 0000:87:0.0 that you want to assign to the new GPU group:

   ```
   [root@xenserver ~]# xe pgpu-list pci-id=0000:87:00.0
   uuid ( RO)              : f76d1c90-e443-4bfc-8f26-7959a7c85c68
   vendor-name ( RO): NVIDIA Corporation
   device-name ( RO): GP102GL [Tesla P40]
   gpu-group-uuid ( RW): 134a7b71-5ceb-8066-ef1b-3b319fb2ef3
   [root@xenserver ~]
   ```

   The `pci-id` parameter passed to the `pgpu-list` command must be in the exact format shown, with the PCI domain fully specified (for example, 0000) and the PCI bus and devices numbers each being two digits (for example, 87:00.0).

3. Ensure that no vGPUs are currently operating on the physical GPU by checking the `resident-VGPUs` parameter:

   ```
   [root@xenserver ~]# xe pgpu-param-get uuid=f76d1c90-e443-4bfc-8f26-7959a7c85c68 param-name=resident-VGPUs
   [root@xenserver ~]#
   ```

4. If any vGPUs are listed, shut down the VMs associated with them.

5. Change the `gpu-group-uuid` parameter of the physical GPU to the UUID of the newly-created GPU group:

   ```
   [root@xenserver ~]# xe pgpu-param-set uuid=7c1e3c0f-1429-0544-df3df8a0866f70a gpu-group-uuid=585877ef-5a6c-66af-fc56-7bd525b0c2f6
   [root@xenserver ~]#
   ```
Any vgpu object now created that specifies this GPU group UUID will always have its vGPUs created on the GPU at PCI bus ID 0000:05:0.0.

You can add more than one physical GPU to a manually-created GPU group - for example, to represent all the GPUs attached to the same CPU socket in a multi-socket server platform - but as for automatically-created GPU groups, all the physical GPUs in the group must be of the same type.

In XenCenter, manually-created GPU groups appear in the GPU type listing in a VM's GPU Properties. Select a GPU type within the group from which you wish the vGPU to be allocated:

![Figure 31 Using a custom GPU group within XenCenter](image)

### 8.4. Cloning vGPU-Enabled VMs

XenServer’s fast-clone or copying feature can be used to rapidly create new VMs from a “golden” base VM image that has been configured with NVIDIA vGPU, the NVIDIA driver, applications, and remote graphics software.

When a VM is cloned, any vGPU configuration associated with the base VM is copied to the cloned VM. Starting the cloned VM will create a vGPU instance of the same type as the original VM, from the same GPU group as the original vGPU.

#### 8.4.1. Cloning a vGPU-enabled VM by using xe

To clone a vGPU-enabled VM from the dom0 shell, use `vm-clone:`
8.4.2. Cloning a vGPU-enabled VM by using XenCenter

To clone a vGPU-enabled VM by using XenCenter, use the VM’s Copy VM command as shown in Figure 32.

![Figure 32 Cloning a VM using XenCenter](image)

```bash
[root@xenserver ~]# xe vm-clone new-name-label="new-vm" vm="base-vm-name" 7f7035cb-388d-1537-1465-1857fb6498e7
[root@xenserver ~]#```
This chapter provides recommendations on optimizing performance for VMs running with NVIDIA vGPU on Citrix XenServer.

### 9.1. XenServer Tools

To get maximum performance out of a VM running on Citrix XenServer, regardless of whether you are using NVIDIA vGPU, you must install Citrix XenServer tools within the VM. Without the optimized networking and storage drivers that the XenServer tools provide, remote graphics applications running on NVIDIA vGPU will not deliver maximum performance.

### 9.2. Using Remote Graphics

NVIDIA vGPU implements a console VGA interface that permits the VM’s graphics output to be viewed through XenCenter’s console tab. This feature allows the desktop of a vGPU-enabled VM to be visible in XenCenter before any NVIDIA graphics driver is loaded in the virtual machine, but it is intended solely as a management convenience; it only supports output of vGPU’s primary display and isn’t designed or optimized to deliver high frame rates.

To deliver high frames from multiple heads on vGPU, NVIDIA recommends that you install a high-performance remote graphics stack such as Citrix XenDesktop® with HDX 3D Pro remote graphics and, after the stack is installed, disable vGPU’s console VGA.

**Caution** Using Windows Remote Desktop (RDP) to access Windows 7 or Windows Server 2008 VMs running NVIDIA vGPU will cause the NVIDIA driver in the VM to be unloaded. GPU-accelerated DirectX, OpenGL, and the NVIDIA control panel will be unavailable whenever RDP is active. Installing a VNC server in the VM will allow for basic, low-performance remote access while leaving the NVIDIA driver loaded and...
vGPU active, but for high performance remote accesses, use an accelerated stack such as XenDesktop.

9.2.1. Disabling console VGA

The console VGA interface in vGPU is optimized to consume minimal resources, but when a system is loaded with a high number of VMs, disabling the console VGA interface entirely may yield some performance benefit.

Once you have installed an alternate means of accessing a VM (such as XenDesktop or a VNC server), its vGPU console VGA interface can be disabled by specifying `disable_vnc=1` in the VM’s `platform:vgpu_extra_args` parameter:

```
[root@xenserver ~]# xe vm-param-set uuid=e71afda4-53f4-3a1b-6c92-a364a7f619c2 platform:vgpu_extra_args="disable_vnc=1"
[root@xenserver ~]#
```

The new console VGA setting takes effect the next time the VM is started or rebooted. With console VGA disabled, the XenCenter console will display the Windows boot splash screen for the VM, but nothing beyond that.

**Caution**

If you disable console VGA before you have installed or enabled an alternate mechanism to access the VM (such as XenDesktop), you will not be able to interact with the VM once it has booted.

You can recover console VGA access by making one of the following changes:

- Removing the `vgpu_extra_args` key from the `platform` parameter
- Removing `disable_vnc=1` from the `vgpu_extra_args` key
- Setting `disable_vnc=0`, for example:

```
[root@xenserver ~]# xe vm-param-set uuid=e71afda4-53f4-3a1b-6c92-a364a7f619c2 platform:vgpu_extra_args="disable_vnc=0"
```

9.3. Allocation Strategies

Strategies for pinning VM CPU cores to physical cores on Non-Uniform Memory Access (NUMA) platforms and for allocating VMs to CPUs and vGPUs to physical GPUs can improve performance for VMs running with NVIDIA vGPU.

9.3.1. NUMA considerations

Server platforms typically implement multiple CPU sockets, with system memory and PCI Express expansion slots local to each CPU socket, as illustrated in Figure 33:
These platforms are typically configured to operate in Non-Uniform Memory Access (NUMA) mode; physical memory is arranged sequentially in the address space, with all the memory attached to each socket appearing in a single contiguous block of addresses. The cost of accessing a range of memory from a CPU or GPU varies; memory attached to the same socket as the CPU or GPU is accessible at lower latency than memory on another CPU socket, because accesses to remote memory must additionally traverse the interconnect between CPU sockets.

To obtain best performance on a NUMA platform, NVIDIA recommends pinning VM vCPU cores to physical cores on the same CPU socket to which the physical GPU hosting the VM’s vGPU is attached. For example, using as a reference, a VM with a vGPU allocated on physical GPU 0 or 1 should have its vCPUs pinned to CPU cores on CPU socket 0. Similarly, a VM with a vGPU allocated on physical GPU 2 or 3 should have its vCPUs pinned to CPU cores on socket 1.

See Pinning VMs to a specific CPU socket and cores for guidance on pinning vCPUs, and How GPU locality is determined for guidance on determining which CPU socket a GPU is connected to. Controlling the vGPU types enabled on specific physical GPUs describes how to precisely control which physical GPU is used to host a vGPU, by creating GPU groups for specific physical GPUs.

9.3.2. Maximizing performance

To maximize performance as the number of vGPU-enabled VMs on the platform increases, NVIDIA recommends adopting a breadth-first allocation: allocate new VMs on the least-loaded CPU socket, and allocate the VM’s vGPU on an available, least-loaded, physical GPU connected via that socket.
XenServer creates GPU groups with a default allocation policy of *depth-first*. See Modifying GPU Allocation Policy on Citrix XenServer for details on switching the allocation policy to *breadth-first*.

Due to vGPU’s requirement that only one type of vGPU can run on a physical GPU at any given time, not all physical GPUs may be available to host the vGPU type required by the new VM.
This chapter describes basic troubleshooting steps for NVIDIA vGPU on Citrix XenServer, Red Hat Enterprise Linux KVM, Red Hat Virtualization (RHV), and VMware vSphere, and how to collect debug information when filing a bug report.

10.1. 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.

10.2. 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.

10.2.1. Verifying the NVIDIA Kernel Driver Is Loaded

1. Use the command that your hypervisor provides to verify that the kernel driver is loaded:

   ▶ On Citrix XenServer, Red Hat Enterprise Linux KVM, and RHV, use `lsmod`:

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

   ▶ On VMware vSphere, use `vmkload_mod`:

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

2. 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).

3. On Citrix XenServer, Red Hat Enterprise Linux KVM, and RHV, also use the `rpm -q` command to verify that the NVIDIA GPU Manager package is correctly installed.
Troubleshooting

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

**vgpu-manager-rpm-package-name**
The RPM package name of the NVIDIA GPU Manager package, for example `NVIDIA-vGPU-xenserver-7.0-410.68` for Citrix XenServer.

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

```
[root@xenserver ~]# rpm -q NVIDIA-vGPU-xenserver-7.0-410.68
[root@xenserver ~]# If an existing NVIDIA GRID package is already installed and you don’t select the upgrade (-U) option when installing a newer GRID package, the rpm command will return many conflict errors.
Preparing packages for installation...
  file /usr/bin/nvidia-smi from install of NVIDIA-vGPU-xenserver-7.0-410.68.x86_64 conflicts with file from package NVIDIA-vGPU-xenserver-7.0-390.94.x86_64
  file /usr/lib/libnvidia-ml.so from install of NVIDIA-vGPU-xenserver-7.0-410.68.x86_64 conflicts with file from package NVIDIA-vGPU-xenserver-7.0-390.94.x86_64
  ...
```

10.2.2. **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 past-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.

10.2.3. **Examining NVIDIA kernel driver output**

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

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

```
[root@xenserver ~]# dmesg | grep -E "NVRM|nvidia"
[ 22.390414] 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.
```

10.2.4. **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`.
10.2.4.1. Examining Citrix XenServer vGPU Manager Messages

For Citrix XenServer, NVIDIA Virtual GPU Manager messages are written to `/var/log/messages`.

Look in the `/var/log/messages` file for the `vmiop` prefix:

```
[root@xenserver ~]# grep vmiop /var/log/messages
Nov 19 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: gpu-pci-id : 0000:05:00.0
Nov 19 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: vgpu_type : quadro
Nov 19 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Framebuffer: 0x74000000
Nov 19 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Virtual Device Id: 0xl3F2:0x114E
Nov 19 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: # # # # # # vGPU Manager Information: # # # # # #
Nov 19 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Driver Version: 410.68
Nov 19 10:35:31 localhost vgpu-ll[25698]: notice: vmiop_log: Init frame copy engine: syncing...
Nov 19 10:35:31 localhost vgpu-ll[25698]: notice: vmiop_log: # # # # # # Guest NVIDIA Driver Information: # # # # # #
Nov 19 10:35:36 localhost vgpu-ll[25698]: notice: vmiop_log: Current max guest pfn = 0x11f7ff
Nov 19 10:35:40 localhost vgpu-ll[25698]: notice: vmiop_log: Current max guest pfn = 0x11eff0!
[root@xenserver ~]#
```

10.2.4.2. Examining Red Hat Enterprise Linux KVM vGPU Manager Messages

For Red Hat Enterprise Linux KVM and RHV, NVIDIA Virtual GPU Manager messages are written to `/var/log/messages`.

Look in these files for the `vmiop_log` prefix:

```
# grep vmiop_log /var/log/messages
```

10.2.4.3. Examining VMware vSphere vGPU Manager Messages

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

Look in the \texttt{vmware.log} file for the \texttt{vmiop} prefix:

[root@esxi:~] \texttt{grep vmiop /vmfs/volumes/datastore1/win7-vgpu-test1/vmware.log}
[root@esxi:~] \texttt{grep vmiop /vmfs/volumes/datastore1/win7-vgpu-test1/vmware.log}

10.3. Capturing configuration data for filing a bug report

When filing a bug report with NVIDIA, capture relevant configuration data from the platform exhibiting the bug in one of the following ways:

- On any supported hypervisor, run \texttt{nvidia-bug-report.sh}.
- On Citrix XenServer, create a XenServer server status report.

[www.nvidia.com](http://www.nvidia.com)  
Virtual GPU Software
10.3.1. Capturing configuration data by running nvidia-bug-report.sh

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

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

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

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

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

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

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

Running nvidia-bug-report.sh...

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

    complete

[root@xenserver ~]#

10.3.2. Capturing Configuration Data by Creating a XenServer Status Report

1. In XenCenter, from the Tools menu, choose Server Status Report.
2. Select the XenServer instance from which you want to collect a status report.
3. Select the data to include in the report.
4. To include NVIDIA vGPU debug information, select NVIDIA-logs in the Report Content Item list.
5. Generate the report.
Figure 34 Including NVIDIA logs in a XenServer status report
Appendix A.
CHANGING VGPU SCHEDULING POLICY

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

GPUs based on the NVIDIA Pascal™ architecture and the NVIDIA Volta architecture 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 based on the Pascal architecture, you can select the vGPU scheduler to use.

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

The GPUs that are based on the Pascal architecture are the Tesla P4, Tesla P6, Tesla P40, and Tesla P100.

The GPUs that are based on the Volta architecture are the Tesla V100 SXM2, Tesla V100 PCIe, and Tesla V100 FHHL.

A.1. vGPU Scheduling Policies

In addition to the default best effort scheduler, GPUs based on the Pascal and Volta architectures 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 P40-4Q vGPUs
Changing vGPU Scheduling Policy

per physical GPU is 6. When the scheduling policy is fixed share, each P40-4Q vGPU is given one sixth, or approximately 16.7%, of 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.

A.2. RmPVMRL Registry Key

The RmPVMRL registry key sets the scheduling policy for NVIDIA vGPUs.

- You can change the vGPU scheduling policy only on GPUs based on the Pascal and Volta architectures.

<table>
<thead>
<tr>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dword</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 (default)</td>
<td>Best Effort Scheduler</td>
</tr>
<tr>
<td>0x01</td>
<td>Equal Share Scheduler</td>
</tr>
<tr>
<td>0x11</td>
<td>Fixed Share Scheduler</td>
</tr>
</tbody>
</table>

Examples

This example sets the vGPU scheduler to Fixed Share Scheduler.

RmPVMRL=0x11

This example sets the vGPU scheduler to Equal Share Scheduler.

RmPVMRL=0x01

A.3. Changing the vGPU Scheduling Policy for All GPUs

- You can change the vGPU scheduling policy only on GPUs based on the Pascal and Volta architectures.

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. Set the `RmPVMRL` registry key to the value that sets the GPU scheduling policy that you want.
   - On Citrix XenServer, Red Hat Enterprise Linux KVM, or Red Hat Virtualization (RHV), add the following entry to the `/etc/modprobe.d/nvidia.conf` file.
     
     ```
     options nvidia NVreg_RegistryDwords="RmPVMRL=value"
     ```
     
     If the `/etc/modprobe.d/nvidia.conf` file does not already exist, create it.
   - On VMware vSphere, use the `esxcli set` command.
     ```
     # esxcli system module parameters set -m nvidia -p "NVreg_RegistryDwords=RmPVMRL=value"
     ```

   *value*

   The value that sets the vGPU scheduling policy that you want, for example:

   - 0x01
     Sets the vGPU scheduling policy to Equal Share Scheduler.
   - 0x11
     Sets the vGPU scheduling policy to Fixed Share Scheduler.

   For all supported values, see `RmPVMRL Registry Key`.

3. Reboot your hypervisor host machine.

### A.4. Changing the vGPU Scheduling Policy for Select GPUs

You can change the vGPU scheduling policy only on GPUs based on the Pascal and Volta architectures.

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. Use the `lspci` command to obtain the PCI domain and bus/device/function (BDF) of each GPU for which you want to change the scheduling behavior.
   ```
   # lspci -D -d 10de:
   ```

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

- On VMware vSphere, pipe the output of `lspci` to the `grep` command to display information only for NVIDIA GPUs.
  
  ```bash
  # lspci | grep NVIDIA
  ``

  The NVIDIA GPUs listed in this example have the PCI domain 0000 and BDFs 85:00.0 and 86:00.0.

  0000:85:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)
  0000:86:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla M60] (rev a1)

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

- On Citrix XenServer, Red Hat Enterprise Linux KVM, or RHV, add the following entry to the `/etc/modprobe.d/nvidia.conf` file.
  
  ```bash
  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.
  
  ```bash
  # 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 vGPU scheduling policy that you want, for example:
    
    0x01
    - Sets the GPU scheduling policy to Equal Share Scheduler.
    
    0x11
    - Sets the GPU scheduling policy to Fixed Share Scheduler.

  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 two GPUs as follows:

  - For the GPU at PCI domain 0000 and BDF 85:00.0, the vGPU scheduling policy is set to Equal Share Scheduler.
  - For the GPU at PCI domain 0000 and BDF 86:00.0, the vGPU scheduling policy is set to Fixed Share Scheduler.

  ```bash
  options nvidia NVreg_RegistryDwordsPerDevice="pci=0000:85:00.0;RmPVMRL=0x01;pci=0000:86:00.0;RmPVMRL=0x11"
  ``

4. Reboot your hypervisor host machine.
A.5. Restoring Default vGPU Scheduler Settings

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

     ```
     # esxcli system module parameters set -m nvidia -p "module-parameter="
     module-parameter
     The module parameter to set, which depends on whether the scheduling behavior was changed for all GPUs or select GPUs:
     - For all GPUs, set the NVreg_RegistryDwords module parameter.
     - For select GPUs, set the NVreg_RegistryDwordsPerDevice module parameter.
     
     For example, to restore default vGPU scheduler settings after they were changed for all GPUs, enter this command:
     ```

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

3. Reboot your hypervisor host machine.
Appendix B.
XENSERVER BASICS

To install and configure NVIDIA vGPU software and optimize XenServer operation with vGPU, some basic operations on XenServer that are needed.

B.1. Opening a dom0 shell

Most configuration commands must be run in a command shell on XenServer’s dom0. You can open a shell on XenServer’s dom0 in any of the following ways:

- Using the console window in XenCenter
- Using a standalone secure shell (SSH) client

B.1.1. Accessing the dom0 shell through XenCenter

1. In the left pane of the XenCenter window, select the XenServer host that you want to connect to.
2. Click on the Console tab to open the XenServer’s console.
3. Press Enter to start a shell prompt.
B.1.2. Accessing the dom0 shell through an SSH client

1. Ensure that you have an SSH client suite such as PuTTY on Windows, or the SSH client from OpenSSH on Linux.
2. Connect your SSH client to the management IP address of the XenServer host.
3. Log in as the root user.

B.2. Copying files to dom0

You can easily copy files to and from XenServer dom0 in any of the following ways:

- Using a Secure Copy Protocol (SCP) client
- Using a network-mounted file system

B.2.1. Copying files by using an SCP client

The SCP client to use for copying files to dom0 depends on where you are running the client from.

- If you are running the client from dom0, use the secure copy command `scp`.
  
  The `scp` command is part of the SSH suite of applications. It is implemented in dom0 and can be used to copy from a remote SSH-enabled server:
  
  ```
  [root@xenserver ~]# scp root@10.31.213.96:/tmp/somefile 
  ```
The authenticity of host '10.31.213.96 (10.31.213.96)' can't be established. RSA key fingerprint is 26:2d:9b:b9:bf:6c:81:70:36:76:13:02:82:3d:3c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.31.213.96' (RSA) to the list of known hosts.
root@10.31.213.96's password:
somefile 100% 532 0.5KB/s 00:00
[root@xenserver ~]#

If you are running the client from Windows, use the pscp program.

The pscp program is part of the PuTTY suite and can be used to copy files from a remote Windows system to XenServer:

C:\Users\nvidia> pscp somefile root@10.31.213.98:/tmp
root@10.31.213.98's password:
somefile | 80 kB | 80.1 kB/s | ETA: 00:00:00 | 100%
C:\Users\nvidia>

B.2.2. Copying files by using a CIFS-mounted file system

You can copy files to and from a CIFS/SMB file share by mounting the share from dom0.

The following example shows how to mount a network share \myserver.example.com\myshare at /mnt/myshare on dom0 and how to copy files to and from the share. The example assumes that the file share is part of an Active Directory domain called example.com and that user myuser has permissions to access the share.

1. Create the directory /mnt/myshare on dom0.

   [root@xenserver ~]# mkdir /mnt/myshare

2. Mount the network share \\myserver.example.com\myshare at /mnt/myshare on dom0.

   [root@xenserver ~]# mount -t cifs -o username=myuser,workgroup=example.com //myserver.example.com/myshare /mnt/myshare

   Password:
   [root@xenserver ~]#

3. When prompted for a password, enter the password for myuser in the example.com domain.

4. After the share has been mounted, copy files to and from the file share by using the cp command to copy them to and from /mnt/myshare:

   [root@xenserver ~]# cp /mnt/myshare/NVIDIA-vGPU-xenserver-7.0-410.68.x86_64.rpm .
   [root@xenserver ~]#

B.3. Determining a VM’s UUID

You can determine a virtual machine’s UUID in any of the following ways:

- Using the xe vm-list command in a dom0 shell
- Using XenCenter
B.3.1. Determining a VM’s UUID by using `xe vm-list`

Use the `xe vm-list` command to list all VMs and their associated UUIDs or to find the UUID of a specific named VM.

- To list all VMs and their associated UUIDs, use `xe vm-list` without any parameters:

```
[root@xenserver ~]# xe vm-list
uuid (RO)          : 6b5585f6-bd74-2e3e-0e11-03b9281c3ade
    name-label (RW): vxg-base-image-win7-64
    power-state (RO): halted

uuid (RO)          : fa3d15c7-7e88-4886-c36a-cdb23ed8e275
    name-label (RW): test-image-win7-32
    power-state (RO): halted

uuid (RO)          : 501bb598-a9b3-4afc-9143-ff85635d5dc3
    name-label (RW): Control domain on host: xenserver
    power-state (RO): running

uuid (RO)          : 8495adf7-be9d-eef1-327f-02e4f40714fc
    name-label (RW): vxg-base-image-win7-32
    power-state (RO): halted
```

- To find the UUID of a specific named VM, use the `name-label` parameter to `xe vm-list`:

```
[root@xenserver ~]# xe vm-list name-label=test-image-win7-32
uuid (RO)          : fa3d15c7-7e88-4886-c36a-cdb23ed8e275
    name-label (RW): test-image-win7-32
    power-state (RO): halted
```

B.3.2. Determining a VM’s UUID by using XenCenter

1. In the left pane of the XenCenter window, select the VM whose UUID you want to determine.
2. In the right pane of the XenCenter window, click the General tab.

The UUID is listed in the VM’s General Properties.
B.4. Using more than two vCPUs with Windows client VMs

Windows client operating systems support a maximum of two CPU sockets. When allocating vCPUs to virtual sockets within a guest VM, XenServer defaults to allocating one vCPU per socket. Any more than two vCPUs allocated to the VM won’t be recognized by the Windows client OS.

To ensure that all allocated vCPUs are recognized, set `platform:cores-per-socket` to the number of vCPUs that are allocated to the VM:

```
[root@xenserver ~]# xe vm-param-set uuid=vm-uuid platform:cores-per-socket=4
VCPUs-max=4 VCPUs-at-startup=4
```

`vm-uuid` is the VM's UUID, which you can obtain as explained in Determining a VM's UUID.

B.5. Pinning VMs to a specific CPU socket and cores

1. Use `xe host-cpu-info` to determine the number of CPU sockets and logical CPU cores in the server platform.
   In this example the server implements 32 logical CPU cores across two sockets:

   ```
   [root@xenserver ~]# xe host-cpu-info
cpu_count : 32
   ```
socket_count: 2
vendor: GenuineIntel
speed: 2600.064
modelname: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
family: 6
model: 45
stepping: 7
flags: fpu de tsc msr pae mce cx8 apic sep mtrr mca
pmov pat clflush acpi mmx mmx ss sse sse2 ss ht nx constant_tsc nonstop_tsc
aperfmpref pni pclmulqdq vmx est ssse3 sse4_1 sse4_2 x2apic popcnt aes
hypervisor ida arat tpr_shadow vmx flexpriority ept vpid
features: 17bee3ff-bfebfbff-00000001-2c100800
features_after_reboot: 17bee3ff-bfebfbff-00000001-2c100800
physical_features: 17bee3ff-bfebfbff-00000001-2c100800
maskable: full

2. Set **VCPUs-params:mask** to pin a VM's vCPUs to a specific socket or to specific cores within a socket.

This setting persists over VM reboots and shutdowns. In a dual socket platform with 32 total cores, cores 0-15 are on socket 0, and cores 16-31 are on socket 1.

In the examples that follow, **vm-uuid** is the VM's UUID, which you can obtain as explained in Determining a VM's UUID.

- To restrict a VM to only run on socket 0, set the mask to specify cores 0-15:

  ```
  [root@xenserver ~]# xe vm-param-set uuid=vm-uuid VCPUs-params:mask=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
  ```

- To restrict a VM to only run on socket 1, set the mask to specify cores 16-31:

  ```
  [root@xenserver ~]# xe vm-param-set uuid=vm-uuid VCPUs-params:mask=16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
  ```

- To pin vCPUs to specific cores within a socket, set the mask to specify the cores directly:

  ```
  [root@xenserver ~]# xe vm-param-set uuid=vm-uuid VCPUs-params:mask=16,17,18,19
  ```

3. Use **xl vcpu-list** to list the current assignment of vCPUs to physical CPUs:

```
[root@xenserver ~]# xl vcpu-list
Name                                ID  VCPU   CPU State  Time(s)   CPU  Affinity
Domain-0                             0     0   25   -b-   9188.4   any cpu
Domain-0                             0     1   19   r--   8908.4   any cpu
Domain-0                             0     2   30   -b-   6815.1   any cpu
Domain-0                             0     3   17   -b-   4881.4   any cpu
Domain-0                             0     4   22   -b-   4956.9   any cpu
Domain-0                             0     5   20   -b-   4319.2   any cpu
Domain-0                             0     6   28   -b-   5720.0   any cpu
Domain-0                             0     7   26   -b-   5736.0   any cpu
test-image-win7-32                  34     0    9   -b-     17.0   4-15
test-image-win7-32                  34     1    4   -b-     13.7   4-15
```

**B.6. Changing dom0 vCPU Default configuration**

By default, dom0 vCPUs are configured as follows:

- The number of vCPUs assigned to dom0 is 8.
The dom0 shell’s vCPUs are unpinned and able to run on any physical CPU in the system.

B.6.1. Changing the number of dom0 vCPUs

The default number of vCPUs assigned to dom0 is 8.

1. Modify the `dom0_max_vcpus` parameter in the Xen boot line.
   For example:
   
   ```
   [root@xenserver ~]# /opt/xensource/libexec/xen-cmdline --set-xen
   dom0_max_vcpus=4
   ```

2. After applying this setting, reboot the system for the setting to take effect by doing one of the following:
   - Run the following command:
     ```
     shutdown -r
     ```
   - Reboot the system from XenCenter.

B.6.2. Pinning dom0 vCPUs

By default, dom0’s vCPUs are unpinned, and able to run on any physical CPU in the system.

1. To pin dom0 vCPUs to specific physical CPUs, use `xl vcpu-pin`.
   For example, to pin dom0’s vCPU 0 to physical CPU 18, use the following command:
   
   ```
   [root@xenserver ~]# xl vcpu-pin Domain-0 0 18
   ```
   CPU pinnings applied this way take effect immediately but do not persist over reboots.

2. To make settings persistent, add `xl vcpu-pin` commands into `/etc/rc.local`
   For example:
   ```
   xl vcpu-pin 0 0 0-15
   xl vcpu-pin 0 1 0-15
   xl vcpu-pin 0 2 0-15
   xl vcpu-pin 0 3 0-15
   xl vcpu-pin 0 4 16-31
   xl vcpu-pin 0 5 16-31
   xl vcpu-pin 0 6 16-31
   xl vcpu-pin 0 7 16-31
   ```

B.7. How GPU locality is determined

As noted in NUMA considerations, current multi-socket servers typically implement PCIe expansion slots local to each CPU socket and it is advantageous to pin VMs to the same socket that their associated physical GPU is connected to.

For current Intel platforms, CPU socket 0 typically has its PCIe root ports located on bus 0, so any GPU below a root port located on bus 0 is connected to socket 0. CPU socket 1
has its root ports on a higher bus number, typically bus 0x20 or bus 0x80 depending on the specific server platform.
Notice

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.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

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 GRID, vGPU, Pascal, 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

© 2013-2018 NVIDIA Corporation. All rights reserved.