

NVIDIA Virtual GPU Software User Guide

Release 20.0-20.4

NVIDIA Corporation

Nov 19, 2026

Contents

1 Introduction to NVIDIA vGPU Software 1
1.1 How NVIDIA vGPU Software Is Used . . . 1
1.1.1 NVIDIA vGPU . . . 1
1.1.2 GPU Pass-Through . . . 1
1.1.3 Bare-Metal Deployment . . . 1
1.2 Primary Display Adapter Requirements for NVIDIA vGPU Software Deployments . . . 2
1.3 NVIDIA vGPU Software Features . . . 3
1.3.1 GPU Instance Support on NVIDIA vGPU Software . . . 3
1.3.2 API Support on NVIDIA vGPU . . . 4
1.3.3 NVIDIA CUDA Toolkit and OpenCL Support on NVIDIA vGPU Software . . . 5
1.3.3.1 OpenCL and CUDA Application Support . . . 5
1.3.3.2 NVIDIA CUDA Toolkit Development Tool Support . . . 6
1.3.3.3 Supported NVIDIA CUDA Toolkit Features . . . 8
1.3.3.4 NVIDIA CUDA Toolkit Features Not Supported by NVIDIA vGPU . . . 9
1.3.4 Additional vWS Features . . . 9
1.3.5 NVIDIA GPU Cloud (NGC) Containers Support on NVIDIA vGPU Software . . . 9
1.3.6 NVIDIA GPU Operator Support . . . 9
1.4 How this Guide Is Organized . . . 10

2 Installing and Configuring NVIDIA Virtual GPU Manager 11
2.1 About NVIDIA Virtual GPUs . . . 11
2.1.1 NVIDIA vGPU Architecture . . . 11
2.1.1.1 Time-Sliced NVIDIA vGPU Internal Architecture for vGPUs on a Single-Instance GPU . . . 12
2.1.1.2 MIG-Backed NVIDIA vGPU Internal Architecture for vGPUs that Occupy an Entire GPU Instance . . . 12
2.1.1.3 Time-Sliced, MIG-Backed NVIDIA vGPU Internal Architecture . . . 12
2.1.2 About Virtual GPU Types . . . 15
2.1.3 Virtual Display Resolutions for Q-series and B-series vGPUs . . . 16
2.1.4 Valid Time-Sliced Virtual GPU Configurations on a Single GPU . . . 17
2.1.5 Guest VM Support . . . 17
2.1.5.1 Windows Guest VM Support . . . 18
2.1.5.2 Linux Guest VM support . . . 18
2.2 Prerequisites for Using NVIDIA vGPU . . . 18
2.3 Switching the Mode of a GPU that Supports Multiple Display Modes . . . 19
2.4 Installing and Configuring the NVIDIA Virtual GPU Manager for XenServer . . . 20
2.4.1 Installing and Updating the NVIDIA Virtual GPU Manager for XenServer . . . 21
2.4.1.1 Installing or Updating the Supplemental Pack for XenServer . . . 21
2.4.1.2 XenServer 8.4 Only: Installing the RPM package for XenServer . . . 21
2.4.1.3 XenServer 8.4 Only: Updating the RPM Package for XenServer . . . 24
2.4.1.4 Verifying the Installation of the NVIDIA vGPU Software for XenServer Package 25
2.4.2 Configuring a XenServer VM with Virtual GPU . . . 26
2.4.3 Setting vGPU Plugin Parameters on XenServer . . . 27

2.5 Installing the Virtual GPU Manager Package for Linux KVM . . . 28
2.6 Installing and Configuring the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server . . . 29
2.6.1 Installing the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server . . . 30
2.6.2 Setting the vGPU Series Allowed on a GPU . . . 32
2.6.3 Adding a vGPU to a Microsoft Azure Local or Microsoft Windows Server VM . . . 33
2.6.4 Uninstalling the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server . . . 35
2.7 Installing and Configuring the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM . . . 36
2.7.1 Installing the Virtual GPU Manager Package for Red Hat Enterprise Linux KVM . . . 36
2.7.2 Verifying the Installation of the NVIDIA vGPU Software for Red Hat Enterprise Linux KVM . . . 37
2.8 Installing and Configuring the NVIDIA Virtual GPU Manager for Ubuntu . . . 39
2.8.1 Installing the NVIDIA Virtual GPU Manager for Ubuntu . . . 39
2.8.1.1 Installing the Virtual GPU Manager Package for Ubuntu . . . 40
2.8.1.2 Verifying the Installation of the NVIDIA vGPU Software for Ubuntu . . . 40
2.9 Installing and Configuring the NVIDIA Virtual GPU Manager for VMware vSphere . . . 42
2.9.1 NVIDIA vGPU Instructions . . . 42
2.9.2 Requirements for Configuring NVIDIA vGPU in a DRS Cluster . . . 42
2.9.3 Installing and Updating the NVIDIA Virtual GPU Manager for VMware vSphere . . . 43
2.9.3.1 Installing the NVIDIA Virtual GPU Manager on VMware vSphere . . . 43
2.9.3.2 Updating the NVIDIA Virtual GPU Manager for VMware vSphere . . . 44
2.9.3.3 Verifying the Installation of the NVIDIA vGPU Software Package for vSphere . . . 45
2.9.3.4 Managing the NVIDIA GPU Management Daemon for VMware vSphere . . . 47
2.9.4 Configuring VMware vMotion with vGPU for VMware vSphere . . . 47
2.9.5 Configuring Fast Suspend-Resume for a VMware vSphere VM . . . 48
2.9.6 Changing the Graphics Type in VMware vSphere . . . 49
2.9.7 Configuring a vSphere VM with NVIDIA vGPU . . . 53
2.9.8 Setting vGPU Plugin Parameters on VMware vSphere . . . 56
2.10 Configuring the vGPU Manager for a Linux with KVM Hypervisor . . . 56
2.10.1 Getting the BDF and Domain of a GPU on a Linux with KVM Hypervisor . . . 57
2.10.2 Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor . . . 58
2.10.3 Creating an NVIDIA vGPU on a Linux with KVM Hypervisor . . . 60
2.10.3.1 Creating a Legacy NVIDIA vGPU on a Linux with KVM Hypervisor . . . 61
2.10.3.2 Creating an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor 63
2.10.3.3 Creating an NVIDIA vGPU on a Linux with KVM Hypervisor that Uses a Vendor-Specific VFIO Framework . . . 65
2.10.4 Adding One or More vGPUs to a Linux with KVM Hypervisor VM . . . 68
2.10.4.1 Adding One or More vGPUs to a Linux with KVM Hypervisor VM by Using `virsh` 68
2.10.4.2 Adding One or More vGPUs to a Linux with KVM Hypervisor VM by Using the QEMU Command Line . . . 71
2.10.5 Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor . . . 72
2.10.6 Deleting a vGPU on a Linux with KVM Hypervisor . . . 74
2.10.6.1 Deleting a vGPU on a Linux with KVM Hypervisor that Uses the `mdev` VFIO Framework . . . 74
2.10.6.2 Deleting a vGPU on a Linux with KVM Hypervisor that Uses a Vendor-Specific VFIO Framework . . . 75
2.10.7 Preparing a GPU Configured for Pass-Through for Use with vGPU . . . 76
2.10.8 NVIDIA vGPU Information in the `sysfs` File System . . . 78
2.10.8.1 NVIDIA vGPU Information in the `sysfs` File System for Hypervisors that Use the `mdev` VFIO Framework . . . 78

2.10.8.2 NVIDIA vGPU Information in the sysfs File System for Hypervisors that Use a Vendor-Specific VFIO Framework . . . 81
2.11 Putting a GPU or GPU Instance into Mixed-Size Mode . . . 82
2.11.1 Putting a GPU into Mixed-Size Mode on a Linux with KVM Hypervisor . . . 83
2.11.2 Putting a GPU Instance into Mixed-Size Mode on a Linux with KVM Hypervisor . . . 84
2.11.3 Putting a GPU Into Mixed-Size Mode on VMware vSphere . . . 85
2.11.3.1 Putting a GPU Into Mixed-Size Mode on VMware vSphere by Using vCenter Server 86
2.11.3.2 Putting a GPU Into Mixed-Size Mode on VMware vSphere by Using the esxcli Command . . . 86
2.12 Placing a vGPU on a Physical GPU or a GPU Instance on a Linux with KVM Hypervisor . 86
2.12.1 Placing a vGPU on a Physical GPU on a Linux with KVM Hypervisor . . . 87
2.12.2 Placing a vGPU on a GPU Instance on a Linux with KVM Hypervisor . . . 89
2.13 Configuring a GPU for MIG-Backed vGPUs . . . 92
2.13.1 Enabling MIG Mode for a GPU . . . 92
2.13.2 Creating GPU Instances on a MIG-Enabled GPU . . . 94
2.13.3 Creating Compute Instances in a GPU instance . . . 96
2.14 Disabling MIG Mode for One or More GPUs . . . 99
2.15 Disabling and Enabling ECC Memory . . . 101
2.15.1 Disabling ECC Memory . . . 102
2.15.2 Enabling ECC Memory . . . 103
2.16 Configuring a vGPU VM for Use with NVIDIA GPUDirect Storage Technology . . . 105

3 Using GPU Pass-Through 107
3.1 Display Resolutions for Physical GPUs . . . 108
3.1.1 NVIDIA RTX Virtual Workstation Physical GPU Resolutions . . . 108
3.1.2 NVIDIA Virtual Applications or NVIDIA vGPU for Compute Physical GPU Resolutions 108
3.2 Using GPU Pass-Through on XenServer . . . 109
3.2.1 Configuring a VM for GPU Pass Through by Using XenCenter . . . 109
3.2.2 Configuring a VM for GPU Pass Through by Using xe . . . 109
3.3 Using GPU Pass-Through on a Linux with KVM Hypervisor . . . 110
3.3.1 Configuring a VM for GPU Pass-Through by Using Virtual Machine Manager (virt-manager) . . . 111
3.3.2 Configuring a VM for GPU Pass-Through by Using virsh . . . 111
3.3.3 Configuring a VM for GPU Pass-Through by Using the QEMU Command Line . . . 113
3.3.4 Preparing a GPU Configured for vGPU for Use in Pass-Through Mode . . . 114
3.4 Using GPU Pass-Through on Microsoft Windows Server . . . 117
3.4.1 Assigning a GPU to a VM on Microsoft Windows Server with Hyper-V . . . 117
3.4.2 Returning a GPU to the Host OS from a VM on Windows Server with Hyper-V . . . 118
3.5 Using GPU Pass-Through on VMware vSphere . . . 120

4 Installing the NVIDIA vGPU Software Graphics Driver 123
4.1 Installing the NVIDIA vGPU Software Graphics Driver and NVIDIA Control Panel on Windows . . . 123
4.1.1 Installing the NVIDIA vGPU Software Graphics Driver on Windows . . . 123
4.1.2 Installing the Standalone NVIDIA Control Panel App . . . 125
4.2 Installing the NVIDIA vGPU Software Graphics Driver on Linux . . . 126
4.2.1 Installing the NVIDIA vGPU Software Graphics Driver on Linux from a .run File . . . 127
4.2.2 Installing the NVIDIA vGPU Software Graphics Driver on Ubuntu from a Debian Package . . . 129
4.2.3 Installing the NVIDIA vGPU Software Graphics Driver on Red Hat Distributions from an RPM Package . . . 130
4.2.4 Disabling the Nouveau Driver for NVIDIA Graphics Cards . . . 130
4.2.5 Disabling the Wayland Display Server Protocol for Red Hat Enterprise Linux . . . 131
4.2.6 Disabling GSP Firmware . . . 132

4.2.7 Configuring the Wayland Display Server Protocol for Use with NVIDIA vGPU Software 133
4.2.7.1 Enabling Remote Desktop Mode for the GNOME Remote Desktop 134
4.2.7.2 Configuring GNOME Remote Desktop for Use with NVIDIA vGPU 134
4.2.7.3 Remote Desktop User: Enabling Access to GNOME Remote Desktop Over RDP 135
4.2.7.4 Remote Desktop Users: Turning Off Screen Lock 136
4.2.7.5 Ubuntu OS Only: Configuring the Mutter Compositor 137
4.2.7.6 Configuring the Wayland Display Server Protocol for MIG-Backed vGPUs 137

5 Modifying a VM's NVIDIA vGPU Configuration 139
5.1 Removing a VM's NVIDIA vGPU Configuration 139
5.1.1 Removing a Citrix Virtual Apps and Desktops VM's vGPU configuration 139
5.1.1.1 Removing a VM's vGPU configuration by using XenCenter 139
5.1.1.2 Removing a VM's vGPU configuration by using `xe` 139
5.1.2 Removing a vSphere VM's vGPU Configuration 141
5.2 Modifying GPU Allocation Policy 141
5.2.1 Modifying GPU Allocation Policy on XenServer 142
5.2.1.1 Modifying GPU Allocation Policy by Using `xe` 142
5.2.1.2 Modifying GPU Allocation Policy GPU by Using XenCenter 142
5.2.2 Modifying GPU Allocation Policy on VMware vSphere 143
5.3 Migrating a VM Configured with vGPU 146
5.3.1 Migrating a VM Configured with vGPU on XenServer 146
5.3.2 Migrating a VM Configured with vGPU on a Linux with KVM Hypervisor 147
5.3.3 Suspending and Resuming a VM Configured with vGPU on a Linux with KVM Hypervisor 149
5.3.4 Migrating a VM Configured with vGPU on Microsoft Azure Local or Microsoft Windows Server 149
5.3.5 Migrating a VM Configured with vGPU on VMware vSphere 150
5.3.6 Suspending and Resuming a VM Configured with vGPU on VMware vSphere 152
5.4 Modifying a MIG-Backed vGPU's Configuration from a Guest VM 152
5.5 Enabling Unified Memory for a vGPU 157
5.5.1 Enabling Unified Memory for a vGPU on XenServer 157
5.5.2 Enabling Unified Memory for a vGPU on Red Hat Enterprise Linux KVM 158
5.5.3 Enabling Unified Memory for a vGPU on VMware vSphere 158
5.6 Enabling NVIDIA CUDA Toolkit Development Tools for NVIDIA vGPU 158
5.6.1 Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU 159
5.6.1.1 Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU on XenServer 159
5.6.1.2 Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU on Red Hat Enterprise Linux KVM 159
5.6.1.3 Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU on on VMware vSphere 159
5.6.2 Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU 160
5.6.2.1 Supported NVIDIA CUDA Toolkit Profiler Features 160
5.6.2.2 Clock Management for a vGPU VM for Which NVIDIA CUDA Toolkit Profilers Are Enabled 160
5.6.2.3 Limitations on the Use of NVIDIA CUDA Toolkit Profilers with NVIDIA vGPU . . . 160
5.6.2.4 Enabling NVIDIA CUDA Toolkit Profilers for a vGPU VM 160
5.7 Enabling the TCC Driver Model for a vGPU 161

6 Monitoring GPU Performance 163
6.1 NVIDIA System Management Interface `nvidia-smi` 163
6.2 Monitoring GPU Performance from a Hypervisor 164
6.2.1 Using `nvidia-smi` to Monitor GPU Performance from a Hypervisor 164
6.2.1.1 Getting a Summary of all Physical GPUs in the System 164
6.2.1.2 Getting a Summary of all vGPUs in the System 166
6.2.1.3 Getting Physical GPU Details 167

6.2.1.4 Getting vGPU Details . . . 171
6.2.1.5 Monitoring vGPU engine usage . . . 173
6.2.1.6 Monitoring vGPU engine usage by applications . . . 174
6.2.1.7 Monitoring Encoder Sessions . . . 175
6.2.1.8 Monitoring Frame Buffer Capture (FBC) Sessions . . . 176
6.2.1.9 Monitoring MIG-backed vGPU activity . . . 181
6.2.1.10 Listing Supported vGPU Types . . . 183
6.2.1.11 Listing the vGPU Types that Can Currently Be Created . . . 185
6.2.2 Using Citrix XenCenter to monitor GPU performance . . . 185
6.3 Monitoring GPU Performance from a Guest VM . . . 185
6.3.1 Using `nvidia-smi` to Monitor GPU Performance from a Guest VM . . . 186
6.3.2 Using Windows Performance Counters to monitor GPU performance . . . 189
6.3.3 Using NVWMI to monitor GPU performance . . . 190
6.3.3.1 `Gpu` instance properties that do not apply to vGPUs . . . 191
6.3.3.2 `PcieLink` instance properties that do not apply to vGPUs . . . 191

7 Changing Scheduling Behavior for Time-Sliced vGPUs and MIG-Backed, Time-Sliced vGPUs 193
7.1 Scheduling Policies for Time-Sliced vGPUs . . . 193
7.2 Scheduler Time Slice for Time-Sliced vGPUs . . . 195
7.3 Getting Information about the Scheduling Behavior of Time-Sliced vGPUs . . . 195
7.3.1 Getting Time-Sliced vGPU Scheduler Capabilities . . . 195
7.3.2 Getting Time-Sliced vGPU Scheduler State Information . . . 196
7.3.2.1 vGPU Scheduler State Information for a VM that Is Not Running . . . 197
7.3.2.2 vGPU Scheduler State Information for a Running VM . . . 197
7.3.2.3 vGPU Scheduler State Information for a GPU Instance on a VM that Is Not Running 197
7.3.2.4 vGPU Scheduler State Information for a GPU Instance on a Running VM . . . 197
7.3.3 Getting Time-Sliced vGPU Scheduler Work Logs . . . 197
7.3.3.1 vGPU Scheduler Work Logs for a VM that Is Not Running . . . 198
7.3.3.2 vGPU Scheduler Work Logs for a Running VM with a GPU in Equal-Size Mode . . 198
7.3.3.3 vGPU Scheduler Work Logs for a Running VM with a GPU in Mixed-Size Mode . 199
7.3.3.4 vGPU Scheduler Work Logs for a GPU Instance on a VM that Is Not Running . . 199
7.3.3.5 vGPU Scheduler Work Logs for a GPU Instance on a Running VM . . . 199
7.3.3.6 vGPU Scheduler Work Logs in CSV Format for a GPU Instance on a VM that Is Not Running . . . 200
7.3.3.7 vGPU Scheduler Work Logs in CSV Format for a GPU Instance on a Running VM 200
7.3.4 Linux-Based Hypervisors Only: Getting the Current Time-Sliced vGPU Scheduling Policy for All GPUs . . . 201
7.4 Tools for Changing Scheduling Behavior for Time-Sliced vGPUs . . . 202
7.5 Changing Scheduling Behavior for Time-Sliced vGPUs by Using the `nvidia-smi` Command . . . 203
7.5.1 Setting the Scheduling Policy for a Single GPU . . . 205
7.5.2 Enabling a Strict Round-Robin Scheduling Policy for an Equal Share Scheduler for All GPUs . . . 205
7.5.3 Enabling a Strict Round-Robin Scheduling Policy for a Fixed Share Scheduler for All GPUs . . . 205
7.5.4 Enabling a Strict Round-Robin Scheduling Policy for an Equal Share Scheduler with Custom Properties for All GPUs . . . 205
7.5.5 Enabling a Strict Round-Robin Scheduling Policy for a Fixed Share Scheduler with Custom Properties for All GPUs . . . 206
7.5.6 Restoring Default Time-Sliced vGPU Scheduler Settings . . . 206
7.5.7 Setting the Scheduling Policy for a Single GPU Instance . . . 206
7.5.8 Restoring Default Time-Sliced vGPU Scheduler Settings for a Single GPU Instance . 206
7.6 Changing Scheduling Behavior for Time-Sliced vGPUs by Using the `RmPVMRL` Registry Key . . . 206

7.6.1 Linux-Based Hypervisors: Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key . . . 207
7.6.1.1 Linux-Based Hypervisors: Changing the Time-Sliced vGPU Scheduling Behavior for All GPUs by Using the RmPVMRL Registry Key . . . 207
7.6.1.2 Linux-Based Hypervisors: Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs by Using the RmPVMRL Registry Key . . . 208
7.6.1.3 Linux-Based Hypervisors: Restoring Default Time-Sliced vGPU Scheduler Settings by Using the RmPVMRL Registry Key . . . 210
7.6.2 Windows Hypervisors: Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key . . . 211
7.6.2.1 Windows Hypervisors: Getting the Driver Key of a GPU . . . 211
7.6.2.2 Windows Hypervisors: Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs by Using the RmPVMRL Registry Key . . . 212
7.6.2.3 Windows Hypervisors: Restoring Default Time-Sliced vGPU Scheduler Settings by Using the RmPVMRL Registry Key . . . 212
7.6.3 RmPVMRL Registry Key . . . 213
7.6.3.1 Type . . . 213
7.6.3.2 Contents . . . 214
7.6.3.3 Examples . . . 215

8 Troubleshooting 217
8.1 Known issues . . . 217
8.2 Troubleshooting steps . . . 217
8.2.1 Verifying the NVIDIA Kernel Driver Is Loaded . . . 217
8.2.2 Verifying that nvidia-smi works . . . 218
8.2.3 Examining NVIDIA kernel driver output . . . 218
8.2.4 Examining NVIDIA Virtual GPU Manager Messages . . . 219
8.2.4.1 Examining XenServer vGPU Manager Messages . . . 219
8.2.4.2 Examining Red Hat Enterprise Linux KVM vGPU Manager Messages . . . 219
8.2.4.3 Examining VMware vSphere vGPU Manager Messages . . . 220
8.3 Capturing configuration data for filing a bug report . . . 221
8.3.1 Capturing configuration data by running nvidia-bug-report.sh . . . 221
8.3.2 Capturing Configuration Data by Creating a XenServer Status Report . . . 222
8.4 Gathering Troubleshooting Information for XID 119 Errors . . . 223

9 Virtual GPU Types Reference 225
9.1 Virtual GPU Types for Supported GPUs . . . 225
9.1.1 NVIDIA A40 Virtual GPU Types . . . 225
9.1.1.1 Q-Series Virtual GPU Types for NVIDIA A40 . . . 225
9.1.1.2 B-Series Virtual GPU Types for NVIDIA A40 . . . 227
9.1.1.3 A-Series Virtual GPU Types for NVIDIA A40 . . . 227
9.1.2 NVIDIA A16 Virtual GPU Types . . . 228
9.1.2.1 Q-Series Virtual GPU Types for NVIDIA A16 . . . 228
9.1.2.2 B-Series Virtual GPU Types for NVIDIA A16 . . . 229
9.1.2.3 A-Series Virtual GPU Types for NVIDIA A16 . . . 230
9.1.3 NVIDIA A10 Virtual GPU Types . . . 231
9.1.3.1 Q-Series Virtual GPU Types for NVIDIA A10 . . . 231
9.1.3.2 B-Series Virtual GPU Types for NVIDIA A10 . . . 232
9.1.3.3 A-Series Virtual GPU Types for NVIDIA A10 . . . 233
9.1.4 NVIDIA A2 Virtual GPU Types . . . 234
9.1.4.1 Q-Series Virtual GPU Types for NVIDIA A2 . . . 234
9.1.4.2 B-Series Virtual GPU Types for NVIDIA A2 . . . 235
9.1.4.3 A-Series Virtual GPU Types for NVIDIA A2 . . . 236
9.1.5 NVIDIA L40 Virtual GPU Types . . . 237

9.1.5.1 Q-Series Virtual GPU Types for NVIDIA L40 . . . 237
9.1.5.2 B-Series Virtual GPU Types for NVIDIA L40 . . . 239
9.1.5.3 A-Series Virtual GPU Types for NVIDIA L40 . . . 239
9.1.6 NVIDIA L40S Virtual GPU Types . . . 240
9.1.6.1 Q-Series Virtual GPU Types for NVIDIA L40S . . . 240
9.1.6.2 B-Series Virtual GPU Types for NVIDIA L40S . . . 242
9.1.6.3 A-Series Virtual GPU Types for NVIDIA L40S . . . 242
9.1.7 NVIDIA L20 and NVIDIA L20 Liquid Cooled Virtual GPU Types . . . 243
9.1.7.1 Q-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled . . . 243
9.1.7.2 B-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled . . . 245
9.1.7.3 A-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled . . . 245
9.1.8 NVIDIA L4 Virtual GPU Types . . . 246
9.1.8.1 Q-Series Virtual GPU Types for NVIDIA L4 . . . 246
9.1.8.2 B-Series Virtual GPU Types for NVIDIA L4 . . . 247
9.1.8.3 A-Series Virtual GPU Types for NVIDIA L4 . . . 248
9.1.9 NVIDIA L2 Virtual GPU Types . . . 249
9.1.9.1 Q-Series Virtual GPU Types for NVIDIA L2 . . . 249
9.1.9.2 B-Series Virtual GPU Types for NVIDIA L2 . . . 250
9.1.9.3 A-Series Virtual GPU Types for NVIDIA L2 . . . 251
9.1.10 NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled Virtual GPU Types . . . 252
9.1.10.1 MIG-Backed Q-Series Virtual GPU Types for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled 252
9.1.10.2 Time-Sliced Q-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled . . . 254
9.1.10.3 MIG-Backed B-Series Virtual GPU Types for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled 256
9.1.10.4 Time-Sliced B-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled . . . 256
9.1.10.5 MIG-Backed A-Series Virtual GPU Types for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled 257
9.1.10.6 Time-Sliced A-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled . . . 258
9.1.11 **Since 20.2:** NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB Virtual GPU Types . . . 259
9.1.11.1 MIG-Backed Q-Series Virtual GPU Types for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB . . . 259
9.1.11.2 Time-Sliced Q-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB . . . 261
9.1.11.3 MIG-Backed B-Series Virtual GPU Types for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB . . . 262
9.1.11.4 Time-Sliced B-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB . . . 263
9.1.12 NVIDIA RTX PRO 4500 Blackwell Server Edition Virtual GPU Types . . . 264
9.1.12.1 MIG-Backed Q-Series Virtual GPU Types for NVIDIA RTX PRO 4500 Blackwell Server Edition . . . 264
9.1.12.2 Time-Sliced Q-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 4500 Blackwell Server Edition . . . 265
9.1.12.3 MIG-Backed B-Series Virtual GPU Types for NVIDIA RTX PRO 4500 Blackwell Server Edition . . . 266

9.1.12.4 Time-Sliced B-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 4500 Blackwell Server Edition . . . 267
9.1.12.5 MIG-Backed A-Series Virtual GPU Types for NVIDIA RTX PRO 4500 Blackwell Server Edition . . . 268
9.1.12.6 Time-Sliced A-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 4500 Blackwell Server Edition . . . 269
9.1.13 NVIDIA RTX 6000 Ada Virtual GPU Types . . . 269
9.1.13.1 Q-Series Virtual GPU Types for NVIDIA RTX 6000 Ada . . . 270
9.1.13.2 B-Series Virtual GPU Types for NVIDIA RTX 6000 Ada . . . 271
9.1.13.3 A-Series Virtual GPU Types for NVIDIA RTX 6000 Ada . . . 272
9.1.14 NVIDIA RTX 5880 Ada Virtual GPU Types . . . 273
9.1.14.1 Q-Series Virtual GPU Types for NVIDIA RTX 5880 Ada . . . 273
9.1.14.2 B-Series Virtual GPU Types for NVIDIA RTX 5880 Ada . . . 275
9.1.14.3 A-Series Virtual GPU Types for NVIDIA RTX 5880 Ada . . . 276
9.1.15 NVIDIA RTX 5000 Ada Virtual GPU Types . . . 277
9.1.15.1 Q-Series Virtual GPU Types for NVIDIA RTX 5000 Ada . . . 277
9.1.15.2 B-Series Virtual GPU Types for NVIDIA RTX 5000 Ada . . . 278
9.1.15.3 A-Series Virtual GPU Types for NVIDIA RTX 5000 Ada . . . 279
9.1.16 NVIDIA RTX A6000 Virtual GPU Types . . . 280
9.1.16.1 Q-Series Virtual GPU Types for NVIDIA RTX A6000 . . . 280
9.1.16.2 B-Series Virtual GPU Types for NVIDIA RTX A6000 . . . 282
9.1.16.3 A-Series Virtual GPU Types for NVIDIA RTX A6000 . . . 282
9.1.17 NVIDIA RTX A5500 Virtual GPU Types . . . 283
9.1.17.1 Q-Series Virtual GPU Types for NVIDIA RTX A5500 . . . 283
9.1.17.2 B-Series Virtual GPU Types for NVIDIA RTX A5500 . . . 284
9.1.17.3 A-Series Virtual GPU Types for NVIDIA RTX A5500 . . . 285
9.1.18 NVIDIA RTX A5000 Virtual GPU Types . . . 286
9.1.18.1 Q-Series Virtual GPU Types for NVIDIA RTX A5000 . . . 286
9.1.18.2 B-Series Virtual GPU Types for NVIDIA RTX A5000 . . . 287
9.1.18.3 A-Series Virtual GPU Types for NVIDIA RTX A5000 . . . 288
9.1.19 Tesla T4 Virtual GPU Types . . . 289
9.1.19.1 Q-Series Virtual GPU Types for Tesla T4 . . . 289
9.1.19.2 B-Series Virtual GPU Types for Tesla T4 . . . 290
9.1.19.3 A-Series Virtual GPU Types for Tesla T4 . . . 291
9.2 Mixed Display Configurations for B-Series and Q-Series vGPUs . . . 292
9.2.1 Mixed Display Configurations for B-Series vGPUs . . . 292
9.2.2 Mixed Display Configurations for Q-Series vGPUs . . . 293
9.3 vGPU Placements for GPUs . . . 293
9.3.1 vGPU Placements for GPUs with 96 GB of Frame Buffer . . . 293
9.3.1.1 vGPU Placements on a Single-Instance GPU . . . 294
9.3.1.2 vGPU Placements on a Full-Partition GPU Instance (4g.96gb+gfx) . . . 295
9.3.1.3 vGPU Placements on a Half-Partition GPU Instance (2g.48gb+gfx) . . . 295
9.3.1.4 vGPU Placements on a Quarter-Partition GPU Instance (1g.24gb+gfx) . . . 296
9.3.2 vGPU Placements for GPUs with 94 GB of Frame Buffer . . . 296
9.3.3 vGPU Placements for GPUs with 80 GB of Frame Buffer . . . 297
9.3.4 vGPU Placements for GPUs with 64 GB of Frame Buffer . . . 298
9.3.5 vGPU Placements for GPUs with 48 GB of Frame Buffer . . . 299
9.3.6 vGPU Placements for GPUs with 40 GB of Frame Buffer . . . 301
9.3.7 vGPU Placements for GPUs with 32 GB of Frame Buffer . . . 302
9.3.8 vGPU Placements for GPUs with 24 GB of Frame Buffer . . . 303
9.3.9 vGPU Placements for GPUs with 20 GB of Frame Buffer . . . 304
9.3.10 vGPU Placements for GPUs with 16 GB of Frame Buffer . . . 304

10 Allocation Strategies **307**

10.1 NUMA Considerations . . . 307
10.1.1 Obtaining Best Performance on a NUMA Platform with XenServer . . . 308
10.1.2 Obtaining Best Performance on a NUMA Platform with VMware vSphere ESXi . . . 308
10.2 Maximizing Performance . . . 308

11 Configuring x11vnc for Checking the GPU in a Linux Server 311
11.1 Configuring the Xorg Server on the Linux Server . . . 311
11.2 Installing and Configuring x11vnc on the Linux Server . . . 313
11.3 Using a VNC Client to Connect to the Linux Server . . . 314

12 Disabling NVIDIA Notification Icon for Citrix Published Application User Sessions 317
12.1 Disabling NVIDIA Notification Icon for All Users' Citrix Published Application Sessions . 319
12.2 Disabling NVIDIA Notification Icon for your Citrix Published Application User Sessions . 319

13 XenServer Basics 321
13.1 Opening a dom0 shell . . . 321
13.1.1 Accessing the dom0 shell through XenCenter . . . 321
13.1.2 Accessing the dom0 shell through an SSH client . . . 321
13.2 Copying files to dom0 . . . 321
13.3 Determining a VM's UUID . . . 322
13.3.1 Determining a VM's UUID by using `xe vm-list` . . . 323
13.3.2 Determining a VM's UUID by using XenCenter . . . 323
13.4 Using more than two vCPUs with Windows client VMs . . . 323
13.5 How GPU locality is determined . . . 324

14 XenServer vGPU Management 325
14.1 Management objects for GPUs . . . 325
14.1.1 `pgpu` - Physical GPU . . . 325
14.1.1.1 Listing the `pgpu` Objects Present on a Platform . . . 325
14.1.1.2 Viewing Detailed Information About a `pgpu` Object . . . 326
14.1.1.3 Viewing physical GPUs in XenCenter . . . 326
14.1.2 `vgpu-type` - Virtual GPU Type . . . 326
14.1.2.1 Listing the `vgpu-type` Objects Present on a Platform . . . 327
14.1.2.2 Viewing Detailed Information About a `vgpu-type` Object . . . 331
14.1.3 `gpu-group` - collection of physical GPUs . . . 332
14.1.3.1 Listing the `gpu-group` Objects Present on a Platform . . . 332
14.1.3.2 Viewing Detailed Information About a `gpu-group` Object . . . 332
14.1.4 `vgpu` - Virtual GPU . . . 333
14.2 Creating a vGPU Using `xe` . . . 333
14.3 Controlling vGPU allocation . . . 334
14.3.1 Determining the Physical GPU on Which a Virtual GPU is Resident . . . 334
14.3.2 Controlling the vGPU types enabled on specific physical GPUs . . . 335
14.3.2.1 Controlling vGPU types enabled on specific physical GPUs by using XenCenter . 336
14.3.2.2 Controlling vGPU Types Enabled on Specific Physical GPUs by Using `xe` . . . 336
14.3.3 Creating vGPUs on Specific Physical GPUs . . . 337
14.4 Cloning vGPU-Enabled VMs . . . 338
14.4.1 Cloning a vGPU-enabled VM by using xe . . . 338
14.4.2 Cloning a vGPU-enabled VM by using XenCenter . . . 339

15 XenServer Performance Tuning 341
15.1 XenServer Tools . . . 341
15.2 Using Remote Graphics . . . 341
15.2.1 Disabling Console VGA . . . 341

16 Notices 343

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, compute 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 vWS and vApps 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 Software 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, Citrix Virtual Apps and Desktops, 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.

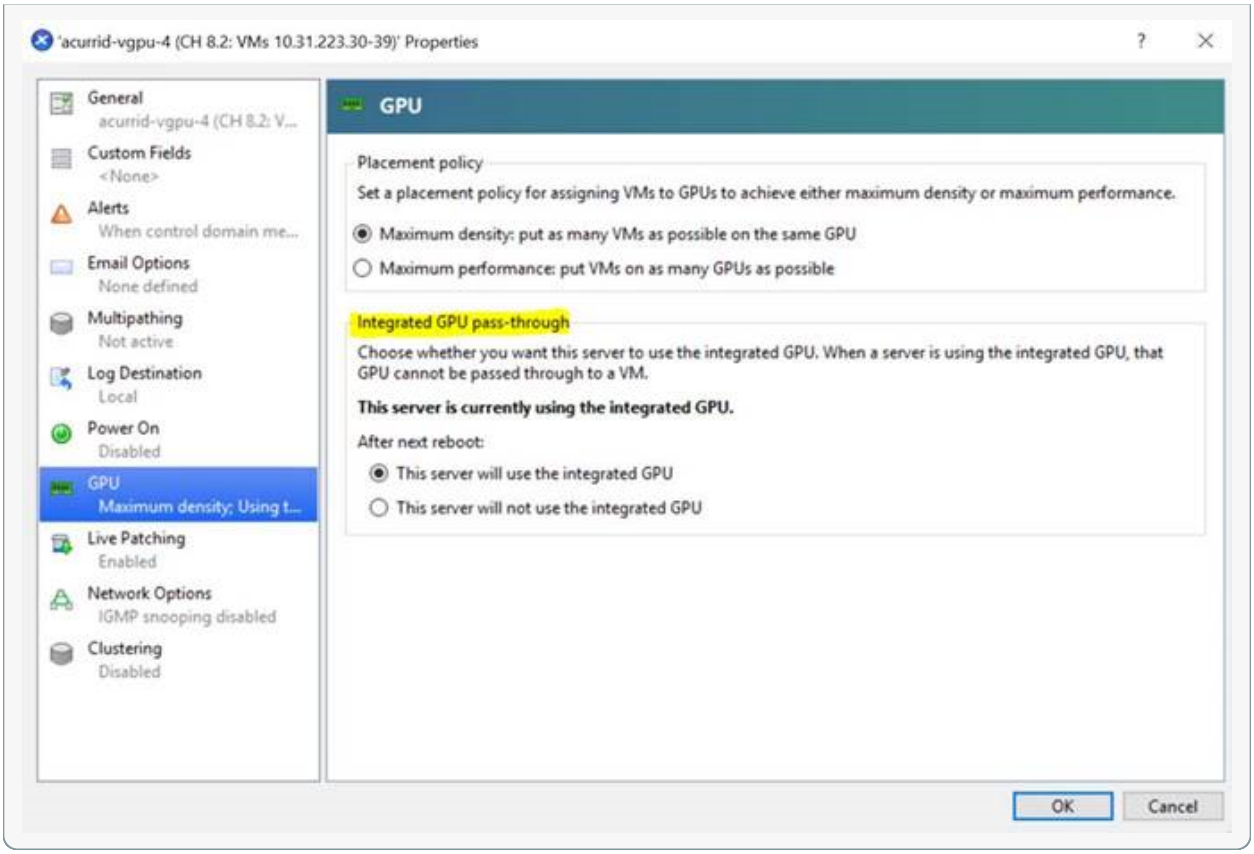
   Users can launch applications that require NVIDIA GPU technology for enhanced user experience only after displays that are driven by NVIDIA adapters are enabled.

1.2. Primary Display Adapter Requirements for NVIDIA vGPU Software Deployments

The GPU that is set as the primary display adapter cannot be used for NVIDIA vGPU deployments or GPU pass through deployments. The primary display is the boot display of the hypervisor host, which displays SBIOS console messages and then boot of the OS or hypervisor.

Any GPU that is being used for NVIDIA vGPU deployments or GPU pass through deployments must be set as a **secondary** display adapter.

> **Note**
>
> XenServer provides a specific setting to allow the primary display adapter to be used for GPU pass through deployments.



Only the following GPUs are supported as the primary display adapter:

- Tesla M6
- Quadro RTX 6000
- Quadro RTX 8000

All other GPUs that support NVIDIA vGPU software cannot function as the primary display adapter because they are 3D controllers, not VGA devices.

If the hypervisor host does not have an extra graphics adapter, consider installing a low-end display adapter to be used as the primary display adapter. If necessary, ensure that the primary display adapter is set correctly in the BIOS options of the hypervisor host.

1.3. NVIDIA vGPU Software Features

NVIDIA vGPU software includes vWS, vPC, and vApps.

1.3.1. GPU Instance Support on NVIDIA vGPU Software

NVIDIA vGPU software supports GPU instances on GPUs that support the Multi-Instance GPU (MIG) feature in NVIDIA vGPU and GPU pass through deployments. MIG enables a physical GPU to be securely partitioned into multiple separate GPU instances, providing multiple users with separate GPU resources to accelerate their applications.

In addition to providing all the benefits of MIG, NVIDIA vGPU software adds virtual machine security and management for workloads. Single Root I/O Virtualization (SR-IOV) virtual functions enable full IOMMU protection for the virtual machines that are configured with vGPUs.

Figure 1.1 shows a GPU that is split into three GPU instances of different sizes, with each instance mapped to one vGPU. Although each GPU instance is managed by the hypervisor host and is mapped to one vGPU, each virtual machine can further subdivide the compute resources into smaller compute instances and run multiple containers on top of them in parallel, even within each vGPU.

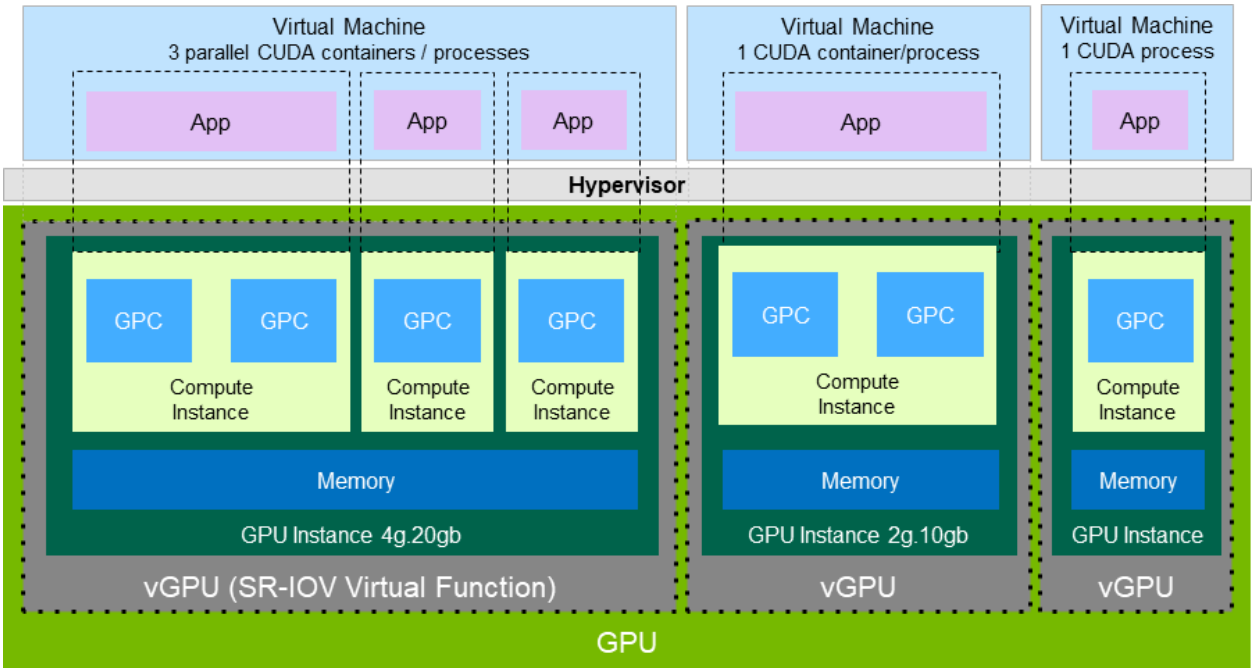


Figure 1.1: GPU Instances Configured with NVIDIA vGPU

NVIDIA vGPU software supports a single-slice MIG-backed vGPU with DEC, JPG, and OFA support. Only one MIG-backed vGPU with DEC, JPG, and OFA support can reside on a GPU. The instance can be placed identically to a single-slice instance without DEC, JPG, and OFA support.

Not all hypervisors support GPU instances in NVIDIA vGPU deployments. To determine if your chosen hypervisor supports GPU instances in NVIDIA vGPU deployments, consult the release notes for your hypervisor at NVIDIA Virtual GPU Software Documentation.

To support GPU instances with NVIDIA vGPU, a GPU must be configured with MIG mode enabled and GPU instances must be created and configured on the physical GPU. For more information, see *Configuring a GPU for MIG-Backed vGPUs*. For general information about the MIG feature, see: NVIDIA Multi-Instance GPU User Guide.

1.3.2. API Support on NVIDIA vGPU

NVIDIA vGPU includes support for the following APIs:

- Open Computing Language (OpenCL™ software) 3.0
- OpenGL® 4.6
- Vulkan® 1.3
- DirectX 11

- DirectX 12 (Windows 10)
- Direct2D
- DirectX Video Acceleration (DXVA)
- NVIDIA® CUDA® 13.2
- NVIDIA vGPU software SDK (remote graphics acceleration)
- NVIDIA RTX (on GPUs based on the NVIDIA Volta graphic architecture and later architectures)

Note

These APIs are backwards compatible. Older versions of the API are also supported.

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

NVIDIA CUDA Toolkit and OpenCL are supported with NVIDIA vGPU only on a subset of vGPU types and supported GPUs.

For more information about NVIDIA CUDA Toolkit, see CUDA Toolkit Documentation 13.2.

Note

If you are using NVIDIA vGPU software with CUDA on Linux, avoid conflicting installation methods by installing CUDA from a distribution-independent runfile package. Do not install CUDA from a distribution-specific RPM or Deb package.

To ensure that the NVIDIA vGPU software graphics driver is not overwritten when CUDA is installed, deselect the CUDA driver when selecting the CUDA components to install.

For more information, see NVIDIA CUDA Installation Guide for Linux.

1.3.3.1 OpenCL and CUDA Application Support

OpenCL and CUDA applications are supported on the following NVIDIA vGPU types:

- All Q-series vGPU types on the following GPUs:
  - NVIDIA L2
  - NVIDIA L4
  - NVIDIA L20
  - NVIDIA L40
  - NVIDIA L40S
  - NVIDIA RTX 5000 Ada
  - NVIDIA RTX 6000 Ada
  - NVIDIA A2
  - NVIDIA A10

- NVIDIA A16
- NVIDIA A40
- NVIDIA RTX A5000
- NVIDIA RTX A5500
- NVIDIA RTX A6000
- RTX PRO 4500 Blackwell Server Edition
- **Since 20.2:** NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB
- NVIDIA RTX PRO 6000 Blackwell Server Edition
- NVIDIA RTX PRO 6000 Blackwell Server Edition liquid cooled
- Tesla T4

1.3.3.2 NVIDIA CUDA Toolkit Development Tool Support

NVIDIA vGPU supports the following NVIDIA CUDA Toolkit development tools on some GPUs:

- Debuggers:
  - CUDA-GDB
  - Compute Sanitizer
- Profilers:
  - The Activity, Callback, and Profiling APIs of the CUDA Profiling Tools Interface (CUPTI)

    Other CUPTI APIs, such as the Event and Metric APIs, are not supported.
  - NVIDIA Nsight™ Compute
  - NVIDIA Nsight Systems
  - NVIDIA Nsight plugin
  - NVIDIA Nsight Visual Studio plugin

  Other CUDA profilers, such as `nvprof` and NVIDIA Visual Profiler, are not supported.

These tools are supported **only** in Linux guest VMs.

NVIDIA CUDA Toolkit profilers are supported and can be enabled on a VM for which unified memory is enabled.

> **Note**
>
> By default, NVIDIA CUDA Toolkit development tools are disabled on NVIDIA vGPU. If used, you must enable NVIDIA CUDA Toolkit development tools individually for each VM that requires them by setting vGPU plugin parameters. For instructions, see *Enabling NVIDIA CUDA Toolkit Development Tools for NVIDIA vGPU*.

The following table lists the GPUs on which NVIDIA vGPU supports these debuggers and profilers.

GPU	vGPU Mode	Debuggers	Profilers
NVIDIA L2	Time-sliced	✓	✓
NVIDIA L4	Time-sliced	✓	✓
NVIDIA L20	Time-sliced	✓	✓
NVIDIA L40	Time-sliced	✓	✓
NVIDIA L40S	Time-sliced	✓	✓
NVIDIA RTX 5000 Ada	Time-sliced	✓	✓
NVIDIA RTX 6000 Ada	Time-sliced	✓	✓
NVIDIA A2	Time-sliced	✓	✓
NVIDIA A10	Time-sliced	✓	✓
NVIDIA A16	Time-sliced	✓	✓
NVIDIA A40	Time-sliced	✓	✓
NVIDIA RTX A5000	Time-sliced	✓	✓
NVIDIA RTX A5500	Time-sliced	✓	✓
NVIDIA RTX A6000	Time-sliced	✓	✓
NVIDIA RTX PRO 4500 Blackwell Server Edition	Time-sliced	✓	✓
	MIG-backed	✓	✓
	MIG-backed, time-sliced	×	×
Since 20.2: NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB	Time-sliced	✓	✓
	MIG-backed	✓	✓
	MIG-backed, time-sliced	×	×
NVIDIA RTX PRO 6000 Blackwell Server Edition	Time-sliced	✓	✓
	MIG-backed	✓	✓
	MIG-backed, time-sliced	×	×
NVIDIA RTX PRO 6000 Blackwell Server Edition liquid cooled	Time-sliced	✓	✓
	MIG-backed	✓	✓
	MIG-backed, time-sliced	×	×
Tesla T4	Time-sliced	✓	✓

✓ Feature is supported

× Feature is not supported

1.3.3.3 Supported NVIDIA CUDA Toolkit Features

NVIDIA vGPU supports the following NVIDIA CUDA Toolkit features if the vGPU type, physical GPU, and the hypervisor software version support the feature:

- Error-correcting code (ECC) memory
- Peer-to-peer CUDA transfers over NVLink

> **Note**
>
> To determine the NVLink topology between physical GPUs in a host or vGPUs assigned to a VM, run the following command from the host or VM:
>
> ```
> $ nvidia-smi topo -m
> ```

- Unified Memory

> **Note**
>
> Unified memory is disabled by default. If used, you must enable unified memory individually for each vGPU that requires it by setting a vGPU plugin parameter. For instructions, see *Enabling Unified Memory for a vGPU*.

- NVIDIA Nsight Systems GPU context switch trace
- NVIDIA NSight Systems video hardware profiling on the following GPUs:
  - NVIDIA L2
  - NVIDIA L4
  - NVIDIA L20
  - NVIDIA L40
  - NVIDIA L40S
  - NVIDIA RTX 5000 Ada
  - NVIDIA RTX 6000 Ada
  - NVIDIA RTX PRO 4500 Blackwell Server Edition
  - **Since 20.2:** NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB
  - NVIDIA RTX PRO 6000 Blackwell Server Edition
  - NVIDIA RTX PRO 6000 Blackwell Server Edition liquid cooled
  - NVIDIA A2
  - NVIDIA A10
  - NVIDIA A16
  - NVIDIA A40

Dynamic page retirement is supported for all vGPU types on physical GPUs that support ECC memory, even if ECC memory is disabled on the physical GPU.

1.3.3.4 NVIDIA CUDA Toolkit Features Not Supported by NVIDIA vGPU

NVIDIA vGPU does not support the NVIDIA Nsight Graphics feature of NVIDIA CUDA Toolkit.

> **Note**
>
> The NVIDIA Nsight Graphics feature is supported in GPU pass-through mode and in bare-metal deployments.

1.3.4. Additional vWS Features

In addition to the features of vPC and vApps, vWS 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, vWS supports multiple virtual display heads at resolutions up to 8K and flexible virtual display resolutions based on the number of available pixels. For details, see *Display Resolutions for Physical GPUs*.
- 10-bit color for Windows users. (HDR/10-bit color is not currently supported on Linux, NvFBC capture is supported but deprecated.)

1.3.5. 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, Q-series vGPU types are supported on all GPUs that support NVIDIA vGPU software.

In GPU pass-through deployments, all GPUs that support NVIDIA vGPU software are supported.

NVIDIA vGPU software supports NGC containers on any guest operating system listed in Supported Platforms - NVIDIA Container Toolkit that is also supported by NVIDIA vGPU software.

For more information about setting up NVIDIA vGPU software for use with NGC containers, see Using NGC with NVIDIA Virtual GPU Software Setup Guide.

1.3.6. NVIDIA GPU Operator Support

NVIDIA GPU Operator simplifies the deployment of NVIDIA vGPU software on software container platforms that are managed by the Kubernetes container orchestration engine. It automates the installation and update of NVIDIA vGPU software graphics drivers for container platforms running in guest VMs that are configured with NVIDIA vGPU.

Any drivers to be installed by NVIDIA GPU Operator must be downloaded from the NVIDIA Licensing Portal to a local computer. Automated access to the NVIDIA Licensing Portal by NVIDIA GPU Operator is not supported.

NVIDIA GPU Operator supports automated configuration of NVIDIA vGPU software and provides telemetry support through DCGM Exporter running in a guest VM.

NVIDIA GPU Operator is supported only on specific combinations of hypervisor software release, container platform, vGPU type, and guest OS release. To determine if your configuration supports NVIDIA GPU Operator with NVIDIA vGPU deployments, consult the release notes for your chosen hypervisor at NVIDIA Virtual GPU Software Documentation.

For more information, see NVIDIA GPU Operator Overview on the NVIDIA documentation portal.

1.4. How this Guide Is Organized

Virtual GPU Software User Guide is organized as follows:

- This chapter introduces the capabilities 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.
- *Modifying a VM's NVIDIA vGPU Configuration* explains how to remove a VM's vGPU configuration and modify GPU assignments for vGPU-enabled VMs.
- *Monitoring GPU Performance* covers performance monitoring of physical GPUs and virtual GPUs from the hypervisor and from within individual guest VMs.
- *Changing Scheduling Behavior for Time-Sliced vGPUs and MIG-Backed, Time-Sliced vGPUs* describes the scheduling behavior of NVIDIA vGPUs and how to change it.
- *Troubleshooting* provides guidance on troubleshooting.
- *Virtual GPU Types Reference* provides details of each vGPU available from each supported GPU and provides examples of mixed virtual display configurations for B-series and Q-series vGPUs.
- *Configuring x11vnc for Checking the GPU in a Linux Server* explains how to use x11vnc to confirm that the NVIDIA GPU in a Linux server to which no display devices are directly connected is working as expected.
- *Disabling NVIDIA Notification Icon for Citrix Published Application User Sessions* explains how to ensure that the NVIDIA Notification Icon application does not prevent the Citrix Published Application user session from being logged off even after the user has quit all other applications.
- *XenServer Basics* explains how to perform basic operations on XenServer to install and configure NVIDIA vGPU software and optimize XenServer operation with vGPU.
- *XenServer vGPU Management* covers vGPU management on XenServer.
- *XenServer Performance Tuning* covers vGPU performance optimization on XenServer.

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. About NVIDIA Virtual GPUs

2.1.1. NVIDIA vGPU Architecture

The high-level architecture of NVIDIA vGPU is illustrated in Figure 2.1. 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.

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

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

- On all GPUs that support NVIDIA vGPU software, time-sliced vGPUs on a single-GPU instance can be created.
- Additionally, on GPUs that support the Multi-Instance GPU (MIG) feature, the following types of MIG-backed vGPU are supported:
  - MIG-backed vGPUs that vGPUs that occupy an entire GPU instance
  - Time-sliced, MIG-backed vGPUs

Note

Not all hypervisors support MIG-backed vGPUs on GPUs that support MIG. For information about whether a hypervisor supports MIG-backed vGPUs, refer to the release notes for the hypervisor at

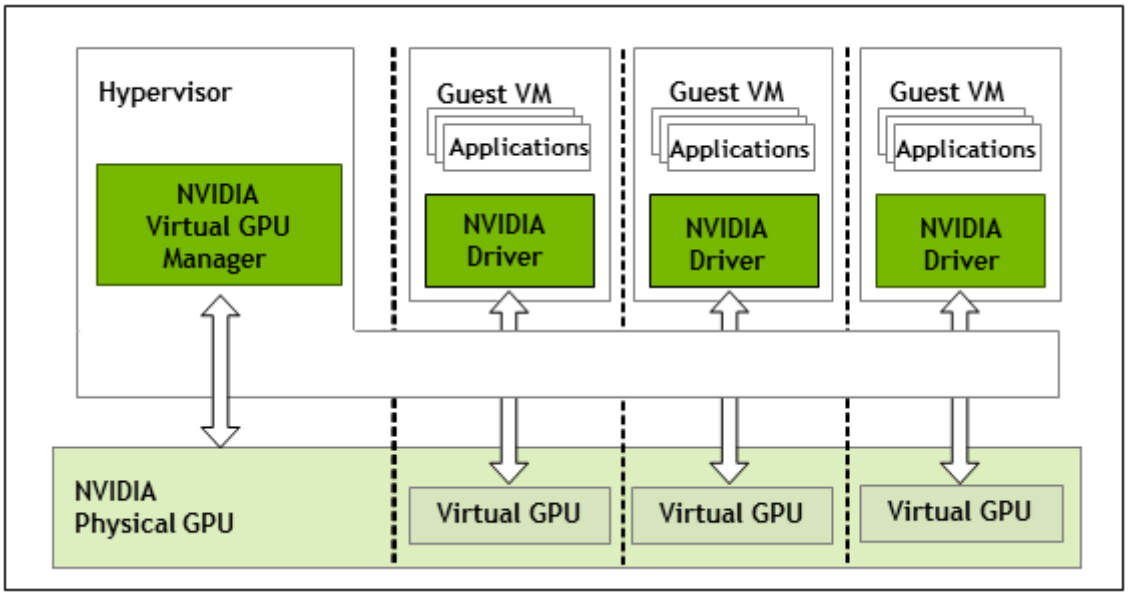


Figure 2.1: NVIDIA vGPU System Architecture

NVIDIA Virtual GPU Software Documentation.

2.1.1.1 Time-Sliced NVIDIA vGPU Internal Architecture for vGPUs on a Single-Instance GPU

A vGPU on a single-instance GPU is a time-sliced vGPU because it shares access to the GPU's engines, including the graphics (3D), video decode, and video encode engines, with all other vGPUs on the GPU. A single-instance GPU is a GPU that is not partitioned into multiple GPU instances.

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

2.1.1.2 MIG-Backed NVIDIA vGPU Internal Architecture for vGPUs that Occupy an Entire GPU Instance

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

In a MIG-backed vGPU, processes that run on the vGPU run in parallel with processes running on other vGPUs in other GPU instances on the GPU. If all the vGPUS resident on a physical GPU occupy an entire GPU instance, processes run on all vGPUs simultaneously.

2.1.1.3 Time-Sliced, MIG-Backed NVIDIA vGPU Internal Architecture

A time-sliced, MIG-backed vGPU is a vGPU that occupies only a fraction of a GPU instance in a MIG-capable physical GPU. Each time-sliced, MIG-backed vGPU shares access to the *GPU instance's* engines, including the compute and video decode engines, with all other vGPUs on the *GPU instance*.

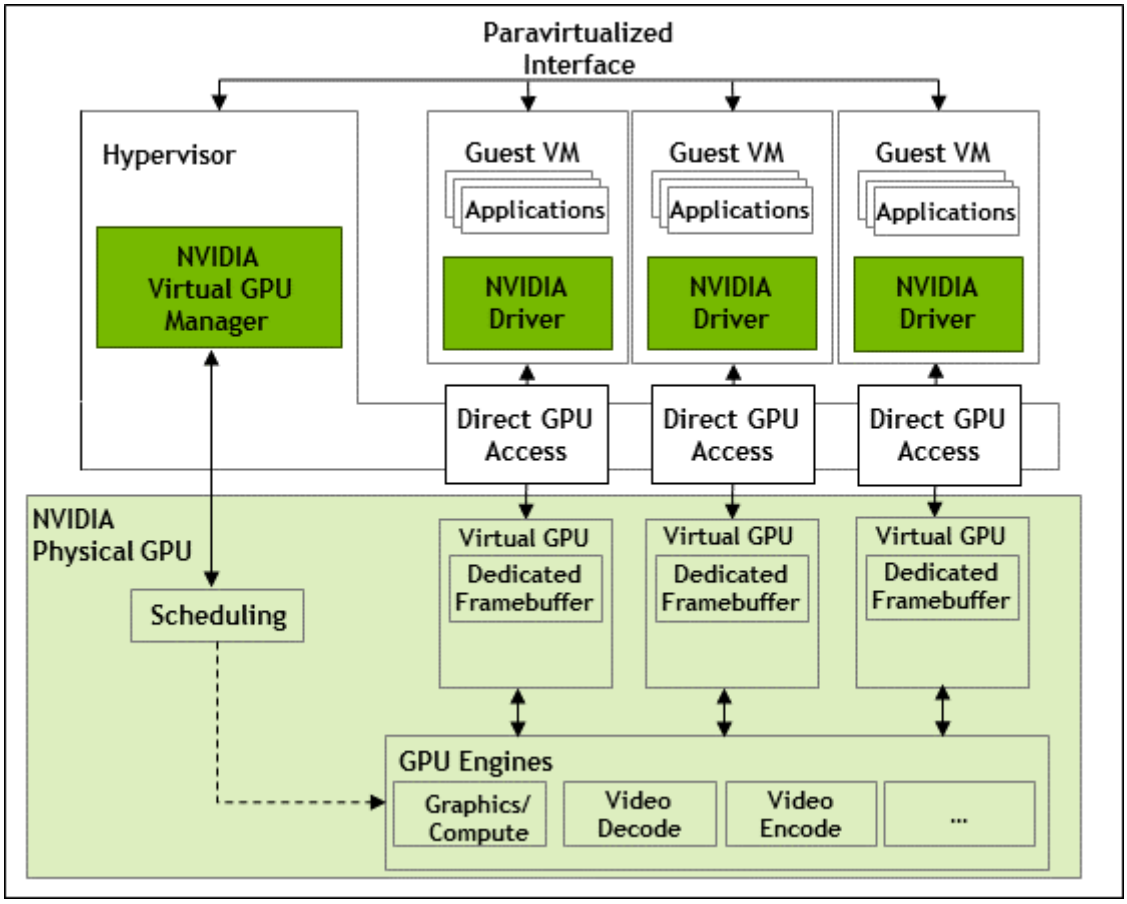


Figure 2.2: Time-Sliced NVIDIA vGPU Internal Architecture for Single-Instance GPUs

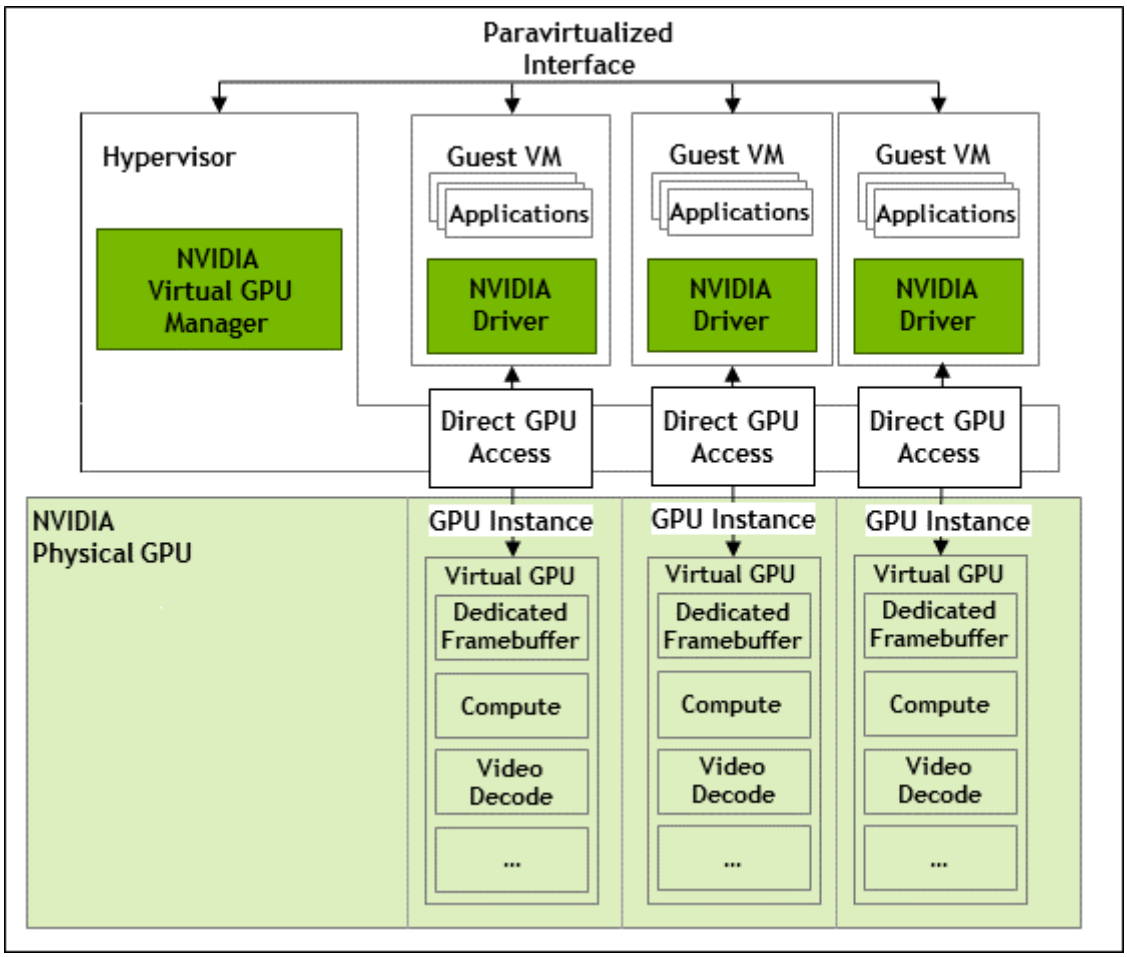


Figure 2.3: MIG-Backed NVIDIA vGPU Internal Architecture

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

Note

You can determine whether time-sliced, MIG-backed vGPUs are supported with your GPU on your chosen hypervisor by running the `nvidia-smi -q` command.

```
$ nvidia-smi -q
    vGPU Device Capability
        MIG Time-Slicing                : Supported
        MIG Time-Slicing Mode           : Enabled
```

- If `MIG Time-Slicing` is shown as `Supported`, the GPU supports time-sliced, MIG-backed vGPUs.
- If `MIG Time-Slicing Mode` is shown as `Enabled`, your chosen hypervisor supports time-sliced, MIG-backed vGPUs on GPUs that also support this feature.

2.1.2. About Virtual GPU Types

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

Series	Optimal Workload
Q-series	Virtual workstations for creative and technical professionals who require the performance and features of Quadro technology
B-series	Virtual desktops for business professionals and knowledge workers
A-series	App streaming or session-based solutions for virtual applications users[1]

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

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

[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 Citrix Virtual Apps and Desktops. The maximum resolution and number of virtual display heads for the A-series NVIDIA vGPUs applies only to the console display. The maximum resolution of each RDSH or Citrix Virtual Apps and Desktops session is determined by the remoting solution and is **not** restricted by the maximum resolution of the vGPU. Similarly, the number of virtual display heads supported by each session is determined by the remoting solution and is **not** restricted by the vGPU.

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

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

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

Note

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

- Q-series vGPU types require a vWS license.
- B-series vGPU types require a vPC license but can also be used with a vWS license.
- A-series vGPU types require a vApps license.

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

2.1.3. Virtual Display Resolutions for Q-series and B-series vGPUs

Instead of a fixed maximum resolution per display, Q-series and B-series vGPUs support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPUs.

The number of virtual displays that you can use depends on a combination of the following factors:

- Virtual GPU series
- GPU architecture
- vGPU frame buffer size
- Display resolution

Note

You cannot use more than the maximum number of displays that a vGPU supports even if the combined resolution of the displays is less than the number of available pixels from the vGPU. For example, because -0Q and -0B vGPUs support a maximum of only two displays, you cannot use four 1280×1024 displays with these vGPUs even though the combined resolution of the displays (6220800) is less than the number of available pixels from these vGPUs (8192000).

Various factors affect the consumption of the GPU frame buffer, which can impact the user experience. These factors include and are not limited to the number of displays, display resolution, workload and applications deployed, remoting solution, and guest OS. The ability of a vGPU to drive a certain combination of displays does not guarantee that enough frame buffer remains free for all applications to run. If applications run out of frame buffer, consider changing your setup in one of the following ways:

- Switching to a vGPU type with more frame buffer
- Using fewer displays
- Using lower resolution displays

The maximum number of displays per vGPU listed in *Virtual GPU Types for Supported GPUs* is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

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

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

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

By default, a GPU or GPU instance supports only vGPUs with the same amount of frame buffer and, therefore, is in equal-size mode. To support vGPUs with different amounts of frame buffer, the GPU or GPU instance must be put into mixed-size mode. In mixed-size mode, the maximum number of some types of vGPU allowed on a GPU or GPU instance is less than in equal-size mode. For more information, refer to the following topics:

- *Putting a GPU or GPU Instance into Mixed-Size Mode*
- *Virtual GPU Types for Supported GPUs*

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

2.1.5. 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.

2.1.5.1 Windows Guest VM Support

Windows guest VMs are supported on all NVIDIA vGPU types, namely: Q-series, B-series, and A-series NVIDIA vGPU types.

2.1.5.2 Linux Guest VM support

Linux guest VMs are supported on all NVIDIA vGPU types, namely: Q-series, B-series, and A-series NVIDIA vGPU types.

2.2. 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.
- If you are using GPUs based on the NVIDIA Ampere architecture or later architectures, the following BIOS settings are enabled on your server platform:
  - VT-D/IOMMU
  - SR-IOV
  - Alternative Routing ID Interpretation (ARI)
- 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, XenServer, Red Hat Enterprise Linux KVM, 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 Virtual Apps and Desktops, or Omnissa Horizon

  Note

  If you are using VMware vSphere Hypervisor (ESXi), the ESXi host on which you configure a VM with NVIDIA vGPU can be a member of a fully automated VMware Distributed Resource Scheduler (DRS) cluster. Earlier restrictions on fully automated DRS clusters no longer apply. For more information, see *Requirements for Configuring NVIDIA vGPU in a DRS Cluster*.

- A VM to be enabled with one or more virtual GPUs is created.

> **Note**
>
> If the VM uses UEFI boot and you plan to install a Linux guest OS in the VM, either ensure that **secure boot** is **disabled** or self-sign the NVIDIA vGPU software graphics driver for Linux. For instructions for self-signing the driver, refer to the Linux `README` file that is included in the `.run` file distribution of the driver.

- 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 XenServer Release Notes
- Virtual GPU Software for Microsoft Azure Local Release Notes
- Virtual GPU Software for Microsoft Windows Server Release Notes
- Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes
- Virtual GPU Software for Ubuntu Release Notes
- Virtual GPU Software for VMware vSphere Release Notes

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

Some GPUs support display-off and display-enabled modes but must be used in NVIDIA vGPU software deployments in display-off mode.

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

GPU	Mode as Supplied from the Factory
NVIDIA A40	Display-off
NVIDIA L40	Display-off
NVIDIA L40S	Display-off
NVIDIA L20	Display-off
NVIDIA L20 liquid cooled	Display-off
NVIDIA RTX 5000 Ada	Display enabled
NVIDIA RTX 6000 Ada	Display enabled
NVIDIA RTX 5880 Ada	Display enabled
NVIDIA RTX A5000	Display enabled
NVIDIA RTX A5500	Display enabled
NVIDIA RTX A6000	Display enabled
NVIDIA RTX PRO 6000 Blackwell Server Edition	Display-off
NVIDIA RTX PRO 6000 Blackwell Server Edition liquid cooled	Display-off

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

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

Note

Only the GPUs listed in the table support the `displaymodeselector` tool. Other GPUs that support NVIDIA vGPU software do not support the `displaymodeselector` tool and, unless otherwise stated, do not require display mode switching.

2.4. Installing and Configuring the NVIDIA Virtual GPU Manager for XenServer

The following topics step you through the process of setting up a single 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*.

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

The NVIDIA Virtual GPU Manager runs in the XenServer dom0 domain.

The NVIDIA Virtual GPU Manager for XenServer is supplied as follows:

- **Since XenServer 9:** Supplemental Pack **only**
- **XenServer 8.4 only:** Supplemental Pack and RPM file

Note

RPM package installation of NVIDIA Virtual GPU Manager for XenServer 8.4 is deprecated. Use Supplemental Pack installation for new deployments and updates.

Caution

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

2.4.1.1 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.
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 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.

2.4.1.2 XenServer 8.4 Only: Installing the RPM package for XenServer

Note

RPM package installation of NVIDIA Virtual GPU Manager for XenServer 8.4 is deprecated. Use Supplemental Pack installation for new deployments and updates.

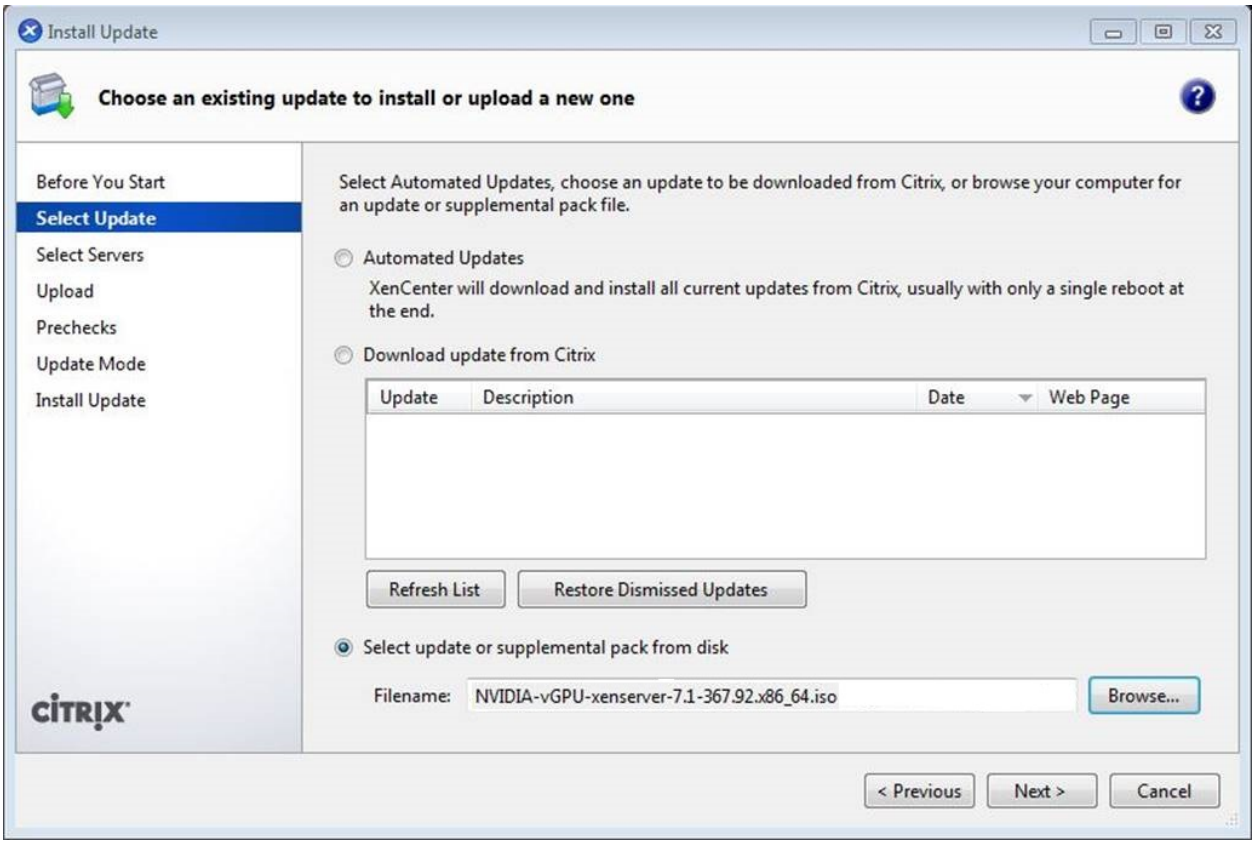


Figure 2.4: NVIDIA vGPU Manager supplemental pack selected in XenCenter

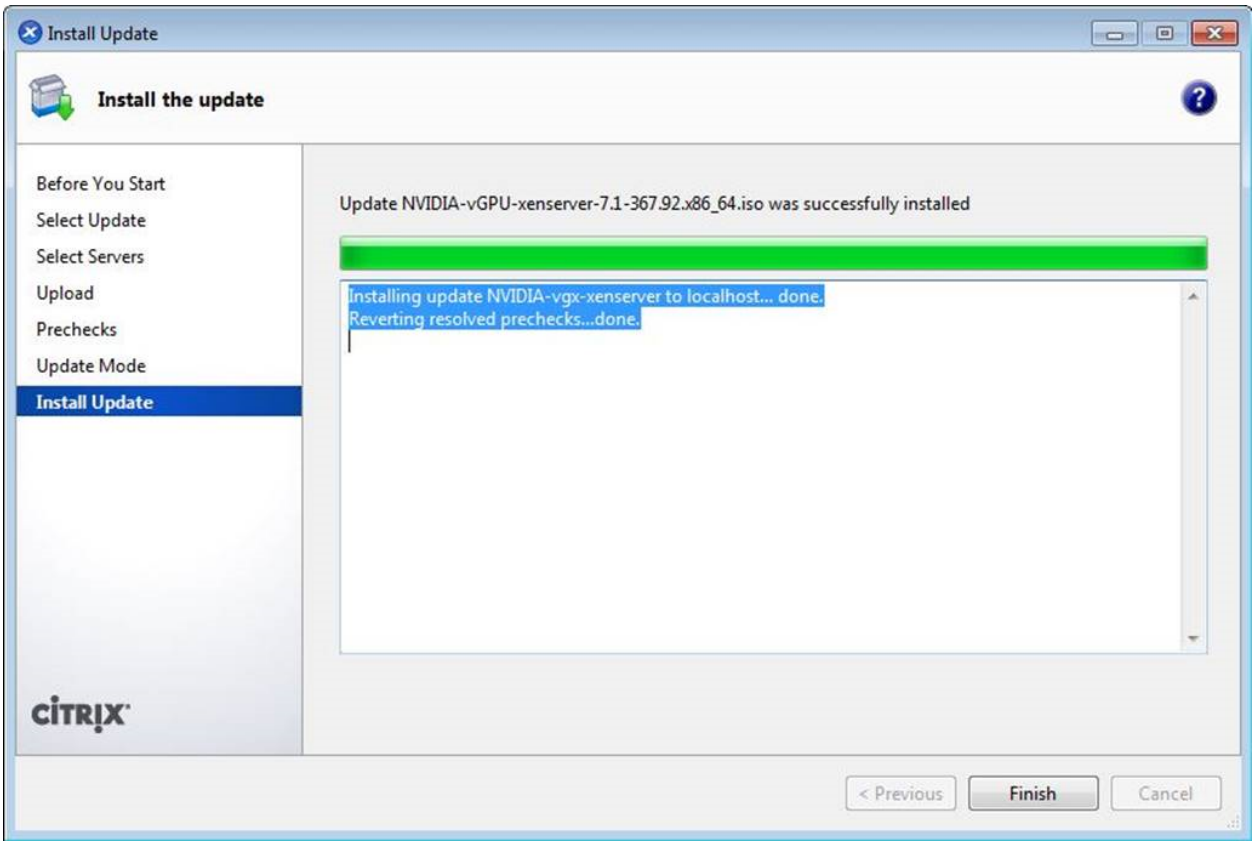


Figure 2.5: Successful installation of NVIDIA vGPU Manager supplemental pack

The RPM file must be copied to the XenServer dom0 domain prior to installation (see *Copying files to dom0*).

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

```
[root@xenserver ~]# rpm -iv NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-
↪595.112.08.x86_64.rpm
Preparing packages for installation...
NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-595.112.08
[root@xenserver ~]#
```

2. Reboot the XenServer platform:

```
[root@xenserver ~]# shutdown –r now

Broadcast message from root (pts/1) (Fri May 15 14:24:11 2026):

The system is going down for reboot NOW!
[root@xenserver ~]#
```

2.4.1.3 XenServer 8.4 Only: Updating the RPM Package for XenServer

Note

RPM package installation of NVIDIA Virtual GPU Manager for XenServer 8.4 is deprecated. Use Supplemental Pack installation for new deployments and updates.

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:

```
[root@xenserver ~]# rpm -Uv NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-
↪595.112.08.x86_64.rpm
Preparing packages for installation...
NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-595.112.08
[root@xenserver ~]#
```

Note

You can query the version of the current NVIDIA Virtual GPU Manager package using the `rpm -q` command:

```
[root@xenserver ~]# rpm -q NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.
↪2-595.112.08
[root@xenserver ~]#
If an existing NVIDIA GRID package is already installed and you don't
↪select the upgrade (-U) option when installing a newer GRID package,
↪the rpm command will return many conflict errors.
Preparing packages for installation...
        file /usr/bin/nvidia-smi from install of NVIDIA-vGPU-NVIDIA-
```

```
↪vGPU-CitrixHypervisor-8.2-595.112.08.x86_64 conflicts with file from
↪package NVIDIA-vGPU-xenserver-8.2-595.91.04.x86_64
       file /usr/lib/libnvidia-ml.so from install of NVIDIA-vGPU-
↪NVIDIA-vGPU-CitrixHypervisor-8.2-595.112.08.x86_64 conflicts with
↪file from package NVIDIA-vGPU-xenserver-8.2-595.91.04.x86_64
       ...
```

3. Reboot the XenServer platform:

```
[root@xenserver ~]# shutdown -r now
Broadcast message from root (pts/1) (Fri May 15 14:24:11 2026):

The system is going down for reboot NOW!
[root@xenserver ~]#
```

2.4.1.4 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.

```
[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.

```
[root@xenserver ~]# nvidia-smi
Fri May 15 18:46:50 2026
+-----------------------------------------------------------+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08   |
|-----------------------------+----------------------+----------------------
↪+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
↪|
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
↪|
|=============================+======================+======================|
|   0  Tesla M60           On   | 00000000:05:00.0 Off |                  Off
↪|
| N/A   25C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default
↪|
+-----------------------------+----------------------+----------------------
↪+
```

(continues on next page)

(continued from previous page)

```
|   1  Tesla M60           On    | 00000000:06:00.0 Off |                  Off
↪|
| N/A   24C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   2  Tesla M60           On    | 00000000:86:00.0 Off |                  Off
↪|
| N/A   25C    P8    25W / 150W |     13MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   3  Tesla M60           On    | 00000000:87:00.0 Off |                  Off
↪|
| N/A   28C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+

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

[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.

2.4.2. Configuring a XenServer VM with Virtual GPU

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

For details about which XenServer versions and NVIDIA vGPUs support the assignment of multiple vGPUs to a VM, see Virtual GPU Software for XenServer Release Notes.

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*.

Note

If you need to assign plugin configuration parameters, create vGPUs using the `xe` command as

explained in *Creating a vGPU Using xe*.

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:

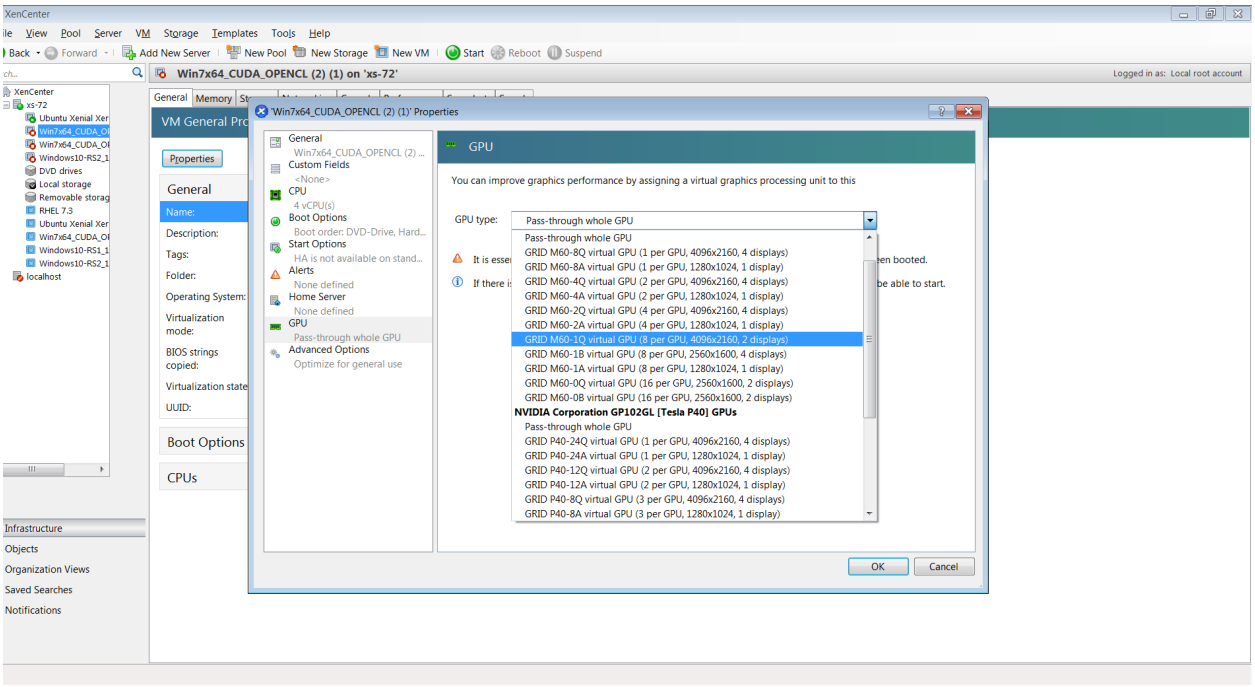


Figure 2.6: Using Citrix XenCenter to configure a VM with a vGPU

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*.

2.4.3. Setting vGPU Plugin Parameters on XenServer

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 command shell in the XenServer dom0 domain.

1. Get the UUIDs of all VMs on the hypervisor host and use the output from the command to identify the VM to which the vGPU is assigned.

```
[root@xenserver ~] xe vm-list
...
uuid ( RO)           : 7f6c855d-5635-2d57-9fbc-b1200172162f
     name-label ( RW): RHEL8.3
    power-state ( RO): running
...
```

2. Get the UUIDs of all vGPUs on the hypervisor host and from the UUID of the VM to which the vGPU is assigned, determine the UUID of the vGPU.

```
[root@xenserver ~] xe vgpu-list
...
uuid ( RO)              : d15083f8-5c59-7474-d0cb-fbc3f7284f1b
           vm-uuid ( RO): 7f6c855d-5635-2d57-9fbc-b1200172162f
            device ( RO): 0
    gpu-group-uuid ( RO): 3a2fbc36-827d-a078-0b2f-9e869ae6fd93
...
```

3. Use the `xe` command to set each vGPU plugin parameter that you want to set.

```
[root@xenserver ~] xe vgpu-param-set uuid=<vgpu-uuid> extra_args='
↪<parameter>=<value>'
```

`<vgpu-uuid>`
The UUID of the vGPU, which you obtained in the previous step.

`<parameter>`
The name of the vGPU plugin parameter that you want to set.

`<value>`
The value to which you want to set the vGPU plugin parameter.

This example sets the `enable_uvm` vGPU plugin parameter to 1 for the vGPU that has the UUID `d15083f8-5c59-7474-d0cb-fbc3f7284f1b`. This parameter setting enables unified memory for the vGPU.

```
[root@xenserver ~] xe vgpu-param-set uuid=d15083f8-5c59-7474-d0cb-
↪fbc3f7284f1b extra_args='enable_uvm=1'
```

2.5. Installing the Virtual GPU Manager Package for Linux KVM

NVIDIA vGPU software for Linux Kernel-based Virtual Machine (KVM) (Linux KVM) is intended **only** for use with supported versions of Linux KVM hypervisors. For details about which Linux KVM hypervisor versions are supported, see Virtual GPU Software for Generic Linux with KVM Release Notes.

Note

If you are using Red Hat Enterprise Linux KVM, follow the instructions in *Installing and Configuring the NVIDIA Virtual GPU Manager for Red Hat Enterprise Linux KVM*.

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

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

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

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

   ```
   # cd <package-file-directory>
   ```

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

2. Make the package file executable.

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

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

3. Run the package file as the root user.

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

   The package file should launch and display the license agreement.

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

   ```
   # systemctl reboot
   ```

2.6. Installing and Configuring the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server

Before you begin, ensure that the prerequisites in *Prerequisites for Using NVIDIA vGPU* are met and the Microsoft Azure Local or Microsoft Windows Server host is configured as follows:

- The OS that you are using is installed as explained in the relevant articles on the Microsoft documentation site.
  - **Microsoft Azure Local:** Deploy the Azure Stack HCI operating system
  - **Microsoft Windows Server:**
    - Install Windows Server from installation media

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

- The following BIOS settings are enabled:
  - Virtualization support, for example, Intel Virtualization Technology (VT-D) or AMD Virtualization (AMD-V)
  - SR-IOV
  - Above 4G Decoding
  - **For Supermicro servers:** ASPM Support
  - **For servers that have an AMD CPU:**
    - Alternative Routing ID Interpretation (ARI)
    - Access Control Service (ACS)
    - Advanced Error Reporting (AER)

Follow this sequence of instructions to set up a single Microsoft Azure Local or Microsoft Windows Server VM to use NVIDIA vGPU.

1. *Installing the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server*
2. *Setting the vGPU Series Allowed on a GPU*
3. *Adding a vGPU to a Microsoft Azure Local or Microsoft Windows Server VM*

These instructions assume familiarity with the Microsoft Windows PowerShell commands covered in Manage VMs on Azure Local using Windows PowerShell on the Microsoft documentation site.

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

2.6.1. Installing the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server

The driver package for the Virtual GPU Manager is distributed as an archive file. You must extract the contents of this archive file to enable the package to be added to the driver store from a setup information file.

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

1. Download the archive file in which the driver package for the Virtual GPU Manager is distributed.
2. Extract the contents of the archive file to a directory that is accessible from the Microsoft Azure Local or Microsoft Windows Server host.
3. Change to the directory that you extracted from the archive file.

   The directory depends on the OS that you are using.

OS	Directory
Microsoft Azure Local	`GridSW-Azure-Local`
Microsoft Windows Server	`GridSW-Windows-Server-GPU-P`

4. Use the `PnPUtil` tool to add the driver package for the Virtual GPU Manager to the driver store from the setup information file for your OS.

   In the command for adding the driver package, also set the options to traverse subdirectories for driver packages and reboot the Microsoft Azure Local or Microsoft Windows Server host if necessary to complete the operation.

   - **Microsoft Azure Local:** Add the driver package to the driver store from the `nvgridswhci.inf` setup information file.

     ```
     PS C:> pnputil /add-driver nvgridswhci.inf /subdirs /install /reboot
     ```

   - **Microsoft Windows Server:** Add the driver package to the driver store from the `nvgridswhostserver.inf` setup information file.

     ```
     PS C:> pnputil /add-driver nvgridswhostserver.inf /subdirs /install /
     ↪reboot
     ```

5. After the host has rebooted, verify that the NVIDIA Virtual GPU Manager can successfully communicate with the NVIDIA physical GPUs in your system.

   Run the `nvidia-smi` command with no arguments for this purpose. Running the `nvidia-smi` command should produce a listing of the GPUs in your platform.

6. Confirm that the Microsoft Azure Local or Microsoft Windows Server host has GPU adapters that can be partitioned by listing the GPUs that support GPU-P.

   ```
   PS C:> Get-VMHostPartitionableGpu
   ```

   If the NVIDIA Virtual GPU Manager is correctly installed, each GPU in the host GPU is listed. The numbers of partitions that each GPU supports and the unique name for referencing each GPU are also listed.

7. For each GPU, set the number of partitions that the GPU should support to the maximum number of vGPUs that can be added to the GPU.

   ```
   PS C:> Set-VMHostPartitionableGpu -Name "<gpu-name>" -PartitionCount
   ↪<partitions>
   ```

   `<gpu-name>`
   The unique name for referencing the GPU that you obtained in the previous step.

   `<partitions>`
   The maximum number of vGPUs that can be added to the GPU. This number depends on the virtual GPU type. For example, the maximum number of each type of vGPU that can be added to the NVIDIA A16 GPU is as follows:

Virtual GPU Type	Maximum vGPUs per GPU
A16-16Q	
A16-16A	1
A16-8Q	
A16-8A	2
A16-4Q	
A16-4A	4
A16-2Q	
A16-2B	
A16-2A	8
A16-1Q	
A16-1B	
A16-1A	16

2.6.2. Setting the vGPU Series Allowed on a GPU

The Virtual GPU Manager allows virtual GPUs (vGPUs) to be created on a GPU from only one vGPU series. By default, only Q-series vGPUs may be created on a GPU. You can change the vGPU series allowed on a GPU by setting the `GridGpupProfileType` value for the GPU in the Windows registry.

This task requires administrator user privileges.

1. Use Windows PowerShell to get the driver key of the GPU on which you want to set the allowed vGPU series.

   You will need this information in the next step to identify the Windows registry key in which information about the GPU is stored.

   1. Get the `InstanceID` property of all available NVIDIA GPUs in your system.

```
PS C:\> Get-PnpDevice -PresentOnly |
>> Where-Object {$_.InstanceId -like "PCI\VEN_10DE*" }  |
>> Select-Object -Property FriendlyName,InstanceId |
>> Format-List

...
FriendlyName : NVIDIA A40
InstanceId   : PCI\VEN_10DE&DEV_2235&SUBSYS_145A10DE&REV_A1\6&
↪2D4D2F51&0&00800018
...
```

> **Note**
>
> If multiple NVIDIA GPUs are available on the system, analyze the output from this command to get the `InstanceID` property of the GPU on which you want to set the allowed

vGPU series.

2. Get the `DEVPKEY_Device_Driver` property of the GPU from the `InstanceID` property that you got in the previous step.

```
PS C:\> Get-PnpDeviceProperty -InstanceId "<instance-id>" |
>> where {$_.KeyName -eq "DEVPKEY_Device_Driver"} |
>> Select-Object -Property Data

Data
----
{4d36e968-e325-11ce-bfc1-08002be10318}\0001
```

`<instance-id>`
The `InstanceID` property of the GPU that you got in the previous step, for example, `PCI\VEN_10DE&DEV_2235&SUBSYS_145A10DE&REV_A1\6&2D4D2F51&0&00800018`.

2. Set the `GridGpupProfileType` DWord (`REG_DWORD`) registry value in the Windows registry key `HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class\<driver-key>`.

`<driver-key>`
The driver key for the GPU that you got in the previous step, for example, `{4d36e968-e325-11ce-bfc1-08002be10318}\0001`.

The value to set depends on the vGPU series that you want to be allowed on the GPU.

vGPU Series	Value
Q-series	1
A-series	2
B-series	3

3. Reboot the Microsoft Azure Local or Microsoft Windows Server host.

2.6.3. Adding a vGPU to a Microsoft Azure Local or Microsoft Windows Server VM

You add a vGPU to a Microsoft Azure Local or Microsoft Windows Server VM by adding a GPU-P adapter to a VM.

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

1. Set the variable `$vm` to the name of the virtual machine to which you are adding a vGPU.

```
PS C:> $vm = "<vm-name>"
```

`<vm-name>`
The name of the virtual machine to which you are adding a vGPU.

2. Allow the VM to control cache types for MMIO access.

```
PS C:> Set-VM -GuestControlledCacheTypes $true -VMName $vm
```

3. Set the lower MMIO space to 1 GB to allow sufficient MMIO space to be mapped.

```
PS C:> Set-VM -LowMemoryMappedIoSpace 1Gb -VMName $vm
```

This amount is twice the amount that the device must allow for alignment. Lower MMIO space is the address space below 4 GB and is required for any device that has 32-bit BAR memory.

4. Set the upper MMIO space to 32 GB to allow sufficient MMIO space to be mapped.

```
PS C:> Set-VM –HighMemoryMappedIoSpace 32GB –VMName $vm
```

This amount is twice the amount that the device must allow for alignment. Upper MMIO space is the address space above approximately 64 GB.

5. Confirm that the Microsoft Azure Local or Microsoft Windows Server host has a GPU that supports the GPU-P adapter that you want to create.

```
PS C:> get-VMHostPartitionableGpu
```

The maximum and minimum values that you can specify for the properties of the GPU that you want to create are also listed.

6. Add a GPU-P adapter to the VM.

```
PS C:> Add-VMGpuPartitionAdapter –VMName $vm `
–MinPartitionVRAM <min-ram> `
-MaxPartitionVRAM <max-ram> `
-OptimalPartitionVRAM <opt-ram> `
-MinPartitionEncode <min-enc> `
-MaxPartitionEncode <max-enc> `
-OptimalPartitionEncode <opt-enc> `
-MinPartitionDecode <min-dec> `
-MaxPartitionDecode <max-dec> `
-OptimalPartitionDecode <opt-dec> `
-MinPartitionCompute <min-compute> `
-MaxPartitionCompute <max-compute> `
-OptimalPartitionCompute <opt-compute>
```

Note

Because partitions are resolved only when the VM is started, this command cannot validate that the Microsoft Azure Local or Microsoft Windows Server host has a GPU that supports the GPU-P adapter that you want to create. The values that you specify must be within the maximum and minimum values that were listed in the previous step.

7. List the adapters assigned to the VM to confirm that the GPU-P adapter has been added to the VM.

```
PS C:> Get-VMGpuPartitionAdapter –VMName $vm
```

This command also returns the adapter ID to use for reconfiguring or deleting a GPU partition.

8. Connect to and start the VM.

2.6.4. Uninstalling the NVIDIA Virtual GPU Manager for Microsoft Azure Local or Microsoft Windows Server

If you no longer require the Virtual GPU Manager on your Microsoft Azure Local or Microsoft Windows Server server, you can uninstall the driver package for the Virtual GPU Manager.

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

1. Determine the published name of the driver package for the Virtual GPU Manager by enumerating all third-party driver packages in the driver store.

   ```
   PS C:> pnputil /enum-drivers
   ```

   Information similar to the following example is displayed. In this example, the published name of the driver package for the Virtual GPU Manager is `oem5.inf`.

   ```
   Microsoft PnP Utility
   ...

   Published name :            oem5.inf
   Driver package provider :   NVIDIA
   Class :                     Display adapters
   Driver date and version :   01/01/2023 31.0.15.2807
   Signer name :               Microsoft Windows Hardware Compatibility
   ↪Publisher

   ...
   ```

2. Delete and uninstall the driver package for the Virtual GPU Manager.

   ```
   PS C:> pnputil /delete-driver <vgpu-manager-package-published-name> /
   ↪uninstall /reboot
   ```

   `<vgpu-manager-package-published-name>`
   The published name of the driver package for the Virtual GPU Manager that you obtained in the previous step, for example, `oem5.inf`.

   This example deletes and uninstalls the driver package for which the published name is `oem5.inf`.

   ```
   PS C:> pnputil.exe /delete-driver oem5.inf /uninstall /reboot
   Microsoft PnP Utility

   Driver package uninstalled.
   Driver package deleted successfully.
   ```

   If necessary, the Microsoft Azure Local or Microsoft Windows Server host is rebooted.

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

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

Follow this sequence of instructions:

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

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

Note

If you are using a generic Linux KVM hypervisor, follow the instructions in *Installing the Virtual GPU Manager Package for Linux KVM*.

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

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

Caution

NVIDIA Virtual GPU Manager and guest VM drivers must be compatible. If you update vGPU Manager to a release that is incompatible with the guest VM drivers, guest VMs will boot with vGPU disabled until their guest vGPU driver is updated to a compatible version. Consult Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes for further details.

Before installing the RPM package for Red Hat Enterprise Linux KVM, ensure that the `sshd` service on the Red Hat Enterprise Linux KVM server is configured to permit root login. If the Nouveau driver for

NVIDIA graphics cards is present, disable it before installing the package. For instructions, see How to disable the Nouveau driver and install the Nvidia driver in RHEL 7 (Red Hat subscription required).

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

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

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

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

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

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

   ```
   # cd <rpm-file-directory>
   ```

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

   ```
   # rpm -iv NVIDIA-vGPU-rhel-8.9-595.112.08.x86_64.rpm
   Preparing packages for installation...
   NVIDIA-vGPU-rhel-8.9-595.112.08
   #
   ```
5. Reboot the Red Hat Enterprise Linux KVM server.

   ```
   # systemctl reboot
   ```

2.7.2. Verifying the Installation of the NVIDIA vGPU Software for Red Hat Enterprise Linux KVM

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

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 May 15 18:46:50 2026
+-----------------------------------------------------------+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08   |
|-------------------------------+----------------------+----------------------
↪+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
↪|
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
↪|
|===============================+======================+======================|
|   0  Tesla M60           On   | 0000:85:00.0     Off |                  Off
↪|
| N/A   23C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   1  Tesla M60           On   | 0000:86:00.0     Off |                  Off
↪|
| N/A   29C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   2  Tesla P40           On   | 0000:87:00.0     Off |                  Off
↪|
| N/A   21C    P8    18W / 250W |     53MiB / 24575MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
+-----------------------------------------------------------------------------
↪+
| Processes:                                                         GPU Memory
```

(continues on next page)

(continued from previous page)

```
↪|
|  GPU       PID  Type  Process name                                  Usage
↪|
|=============================================================================|
|  No running processes found
↪|
+-----------------------------------------------------------------------------
↪+
#
```

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

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

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

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

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

2.8.1. Installing the NVIDIA Virtual GPU Manager for Ubuntu

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

Caution

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

2.8.1.1 Installing the Virtual GPU Manager Package for Ubuntu

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

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

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

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

   ```
   # cd <deb-file-directory>
   ```

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

   ```
   # apt install ./nvidia-vgpu-ubuntu-595.112.08_amd64.deb
   ```
5. Reboot the Ubuntu server.

   ```
   # systemctl reboot
   ```

2.8.1.2 Verifying the Installation of the NVIDIA vGPU Software for Ubuntu

After the Ubuntu server has rebooted, verify the installation of the NVIDIA vGPU software package for Ubuntu.

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 May 15 18:46:50 2026
+-----------------------------------------------------------+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08   |
|-------------------------------+----------------------+----------------------
 ↪+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
 ↪|
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
 ↪|
|===============================+======================+======================|
|   0  Tesla M60           On   | 0000:85:00.0     Off |                  Off
 ↪|
| N/A   23C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+
|   1  Tesla M60           On   | 0000:86:00.0     Off |                  Off
 ↪|
| N/A   29C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+
|   2  Tesla P40           On   | 0000:87:00.0     Off |                  Off
 ↪|
| N/A   21C    P8    18W / 250W |     53MiB / 24575MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+
+-----------------------------------------------------------------------------
 ↪+
| Processes:                                                       GPU Memory
 ↪|
|  GPU       PID  Type  Process name                               Usage
 ↪|
|=============================================================================|
|  No running processes found
 ↪|
+-----------------------------------------------------------------------------
 ↪+
#
```

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

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

You can use the NVIDIA Virtual GPU Manager for VMware vSphere to set up a VMware vSphere VM to use NVIDIA vGPU.

Note

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

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

2.9.1. NVIDIA vGPU Instructions

Note

The Xorg service is not required for graphics devices in NVIDIA vGPU mode. For more information, see *Installing and Updating the NVIDIA Virtual GPU Manager for VMware vSphere*.

To set up a VMware vSphere VM to use NVIDIA vGPU, follow this sequence of instructions:

1. *Installing and Updating the NVIDIA Virtual GPU Manager for VMware vSphere*
2. *Configuring VMware vMotion with vGPU for VMware vSphere*
3. *Configuring Fast Suspend-Resume for a VMware vSphere VM*
4. *Changing the Default Graphics Type in VMware vSphere*
5. **MIG-backed vGPUs only:** *Configuring a GPU for MIG-Backed vGPUs*
6. *Configuring a vSphere VM with NVIDIA vGPU*
7. **Optional:** *Setting vGPU Plugin Parameters on VMware vSphere*

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.

2.9.2. Requirements for Configuring NVIDIA vGPU in a DRS Cluster

You can configure a VM with NVIDIA vGPU on an ESXi host in a VMware Distributed Resource Scheduler (DRS) cluster. Any automation level of the cluster is supported.

If the cluster automation level is **Fully Automated**, see Broadcom Knowledge Base Article: vGPU Virtual Machine automated migration for Host Maintenance Mode in a DRS Cluster for required configuration settings and possible known issues.

For more information about the settings for automation levels in a DRS cluster, see Edit Cluster Settings in the VMware documentation.

2.9.3. Installing and Updating the NVIDIA Virtual GPU Manager for VMware vSphere

The NVIDIA Virtual GPU Manager runs on the ESXi host. It is distributed as a number of software components in a ZIP archive.

The NVIDIA Virtual GPU Manager software components are as follows:

- A software component for the NVIDIA vGPU hypervisor host driver
- A software component for the NVIDIA GPU Management daemon

You can install these software components in one of the following ways:

- By copying the software components to the ESXi host and then installing them as explained in *Installing the NVIDIA Virtual GPU Manager on VMware vSphere*
- By importing the software components into the vSphere Lifecycle Manager depot as explained in Import Updates to the vSphere Lifecycle Manager Depot in the VMware vSphere documentation
- By downloading the software components through NVIDIA GPU Manager for VMware vCenter and synchronizing them with vSphere Lifecycle Manager as explained in Downloading and Synchronizing NVIDIA GPU Drivers in the *NVIDIA GPU Manager for VMware vCenter User Guide*

Caution

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

Note

Installing or updating the Virtual GPU Manager for VMware vSphere by using the `esxcli software` might fail with the following error:

```
[InstallationError]
The pending transaction requires 524 MB free space, however the maximum
↪supported size is 489 MB.
```

For information about how to work around this issue, refer to Broadcom Knowledge Base Article: ESXi upgrade or installation failure: Host not compatible with the image. The bootbank partition on the host has a capacity of □size□, the image requires □size□.

2.9.3.1 Installing the NVIDIA Virtual GPU Manager on VMware vSphere

To install the NVIDIA Virtual GPU Manager you need to access the ESXi host via the ESXi Shell or SSH. Refer to VMware's documentation on how to enable ESXi Shell or SSH for an ESXi host.

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

- The ZIP archive that contains NVIDIA vGPU software has been downloaded from the NVIDIA Licensing Portal.
- The software components for the NVIDIA Virtual GPU Manager have been extracted from the downloaded ZIP archive.

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

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

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

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

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

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

<datastore>
The name of the VMFS datastore to which you copied the software components.

<host-driver-component>
The name of the file that contains the NVIDIA vGPU hypervisor host driver in the form of a software component. Ensure that you specify the file that was extracted from the downloaded ZIP archive. For example, for VMware vSphere 8.0.0, <host-driver-component> is `NVD-VMware-x86_64-595.112.08-1OEM.800.1.0.20613240-bundle-<build-number>`.

<gpu-management-daemon-component>
The name of the file that contains the NVIDIA GPU Management daemon in the form of a software component. Ensure that you specify the file that was extracted from the downloaded ZIP archive. For example, for VMware vSphere 8.0.0, <gpu-management-daemon-component> is `VMW-esx-8.0.0-nvd-gpu-mgmt-daemon-1.0-0.0.0001`.

4. Exit maintenance mode.

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

5. Reboot the ESXi host.

```
$ reboot
```

2.9.3.2 Updating the NVIDIA Virtual GPU Manager for VMware vSphere

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

To update the vGPU Manager VIB you need to access the ESXi host via the ESXi Shell or SSH. Refer to VMware's documentation on how to enable ESXi Shell or SSH for an ESXi host.

Note

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

1. Stop the NVIDIA GPU Management Daemon.

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

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

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

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

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

<datastore>
The name of the VMFS datastore to which you copied the software components.

<host-driver-component>
The name of the file that contains the NVIDIA vGPU hypervisor host driver in the form of a software component. Ensure that you specify the file that was extracted from the downloaded ZIP archive. For example, for VMware vSphere 8.0.0, <host-driver-component> is NVD-VMware-x86_64-595.112.08-10EM.800.1.0.20613240-bundle-<build-number>.

<gpu-management-daemon-component>
The name of the file that contains the NVIDIA GPU Management daemon in the form of a software component. Ensure that you specify the file that was extracted from the downloaded ZIP archive. For example, for VMware vSphere 8.0.0, <gpu-management-daemon-component> is VMW-esx-8.0.0-nvd-gpu-mgmt-daemon-1.0-0.0.0001.

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

2.9.3.3 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 <module>
nvidia                  5    8420
```

<module> depends on the GPU architecture of your GPU:

- For the NVIDIA Ada Lovelace GPU architecture and later GPU architectures, <module> is nvidia-gpu.

- For GPU architectures before NVIDIA Ada Lovelace, `<module>` is `nvidia`.

2. If the NVIDIA driver is not listed in the output, check `dmesg` for any load-time errors reported by the driver.
3. Verify that the NVIDIA GPU Management daemon has started.

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

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

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

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

```
[root@esxi:~] nvidia-smi
Fri May 15 17:56:22 2026
+-----------------------------------------------------------+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08   |
|-------------------------------+----------------------+----------------------
 ↪+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
 ↪|
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
 ↪|
|===============================+======================+======================|
|   0  Tesla M60           On   | 00000000:05:00.0 Off |                   Off
 ↪|
| N/A   25C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+
|   1  Tesla M60           On   | 00000000:06:00.0 Off |                   Off
 ↪|
| N/A   24C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+
|   2  Tesla M60           On   | 00000000:86:00.0 Off |                   Off
 ↪|
| N/A   25C    P8    25W / 150W |     13MiB /  8191MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+
|   3  Tesla M60           On   | 00000000:87:00.0 Off |                   Off
 ↪|
| N/A   28C    P8    24W / 150W |     13MiB /  8191MiB |      0%      Default
 ↪|
+-------------------------------+----------------------+----------------------
 ↪+

+-----------------------------------------------------------------------------
 ↪+
| Processes:                                                         GPU Memory
```

(continues on next page)

(continued from previous page)

```
↪|
|  GPU       PID   Type   Process name                              Usage
↪|
|=============================================================================|
|  No running processes found
↪|
+-----------------------------------------------------------------------------
↪+
```

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

2.9.3.4 Managing the NVIDIA GPU Management Daemon for VMware vSphere

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

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

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

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

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

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

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

2.9.4. 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.

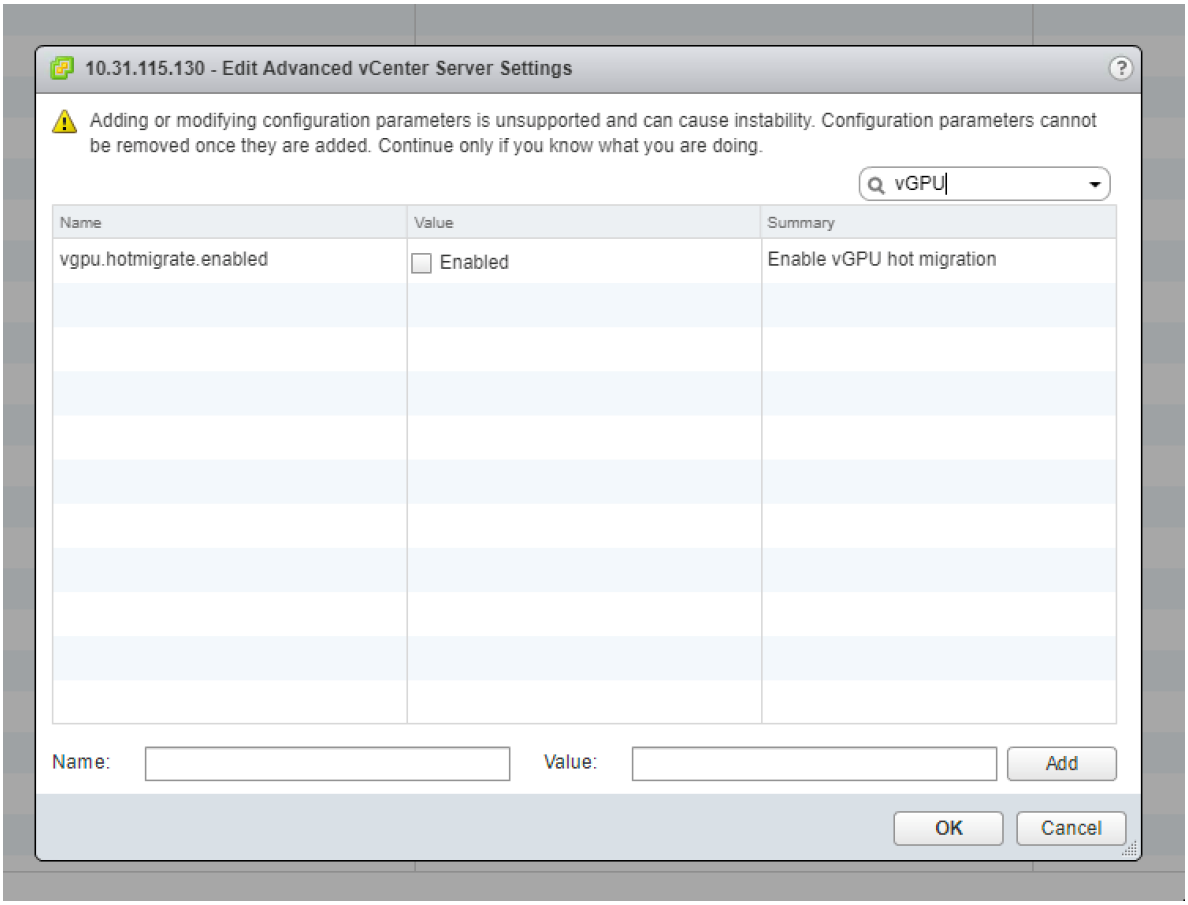
Before configuring VMware vMotion with vGPU for an ESXi host, ensure that the current NVIDIA Virtual GPU Manager for VMware vSphere package is installed on the host.

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

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

3. Click the **Configure** tab.

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



2.9.5. Configuring Fast Suspend-Resume for a VMware vSphere VM

NVIDIA vGPU software supports Fast Suspend-Resume for VMs that are configured with vGPU. Fast Suspend-Resume enables changes to a VM, such as adding a network adapter, with minimal disruption to the VM. To enable Fast Suspend-Resume for a VMware vSphere VM, an advanced VM setting must be enabled.

For details about which VMware vSphere versions, NVIDIA GPUs, and guest OS releases support Fast Suspend-Resume for VMware vSphere, see Virtual GPU Software for VMware vSphere Release Notes.

Before configuring Fast Suspend-Resume for a VMware vSphere VM, ensure that the current NVIDIA Virtual GPU Manager for VMware vSphere package is installed on the ESXi host.

1. Log in to vCenter Server by using the vSphere Web Client.
2. In the inventory, context-click the VM and choose **Edit Settings**.
3. Click the **Advanced Parameters** tab.
4. Add the `vmiop.allowWarmUpdate` attribute with the value TRUE, click **Add** and then click **OK**.

5. Boot the VM.

2.9.6. Changing the Graphics Type in VMware vSphere

After NVIDIA Virtual GPU Manager is installed, the graphics type is Shared. To enable vGPU support for VMs in VMware vSphere, you must change the graphics type to Shared Direct.

If you do not change the graphics type, VMs to which a vGPU is assigned fail to start and the following error message is displayed:

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

Note

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

Note

On VMware Cloud Foundation (VCF) 9 releases, you can change the graphics type only for individual physical GPUs. On earlier VMware vSphere releases, you can change the default graphics type for the host.

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**.
4. **Releases earlier than VCF 9 only:** Set the default graphics type for the host to Shared Direct.
    1. Click the **Host Graphics** tab.
    2. On the **Host Graphics** tab, click **Edit**.
    3. In the Edit Host Graphics Settings dialog box that opens, select **Shared Direct** and click **OK**.

        Note

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

        After you click OK, the default graphics type changes to Shared Direct.
5. 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.
    1. Open the **Graphics Devices** tab.
    2. For each physical GPU for which the configured type is Shared, change its configured type to Shared Direct.

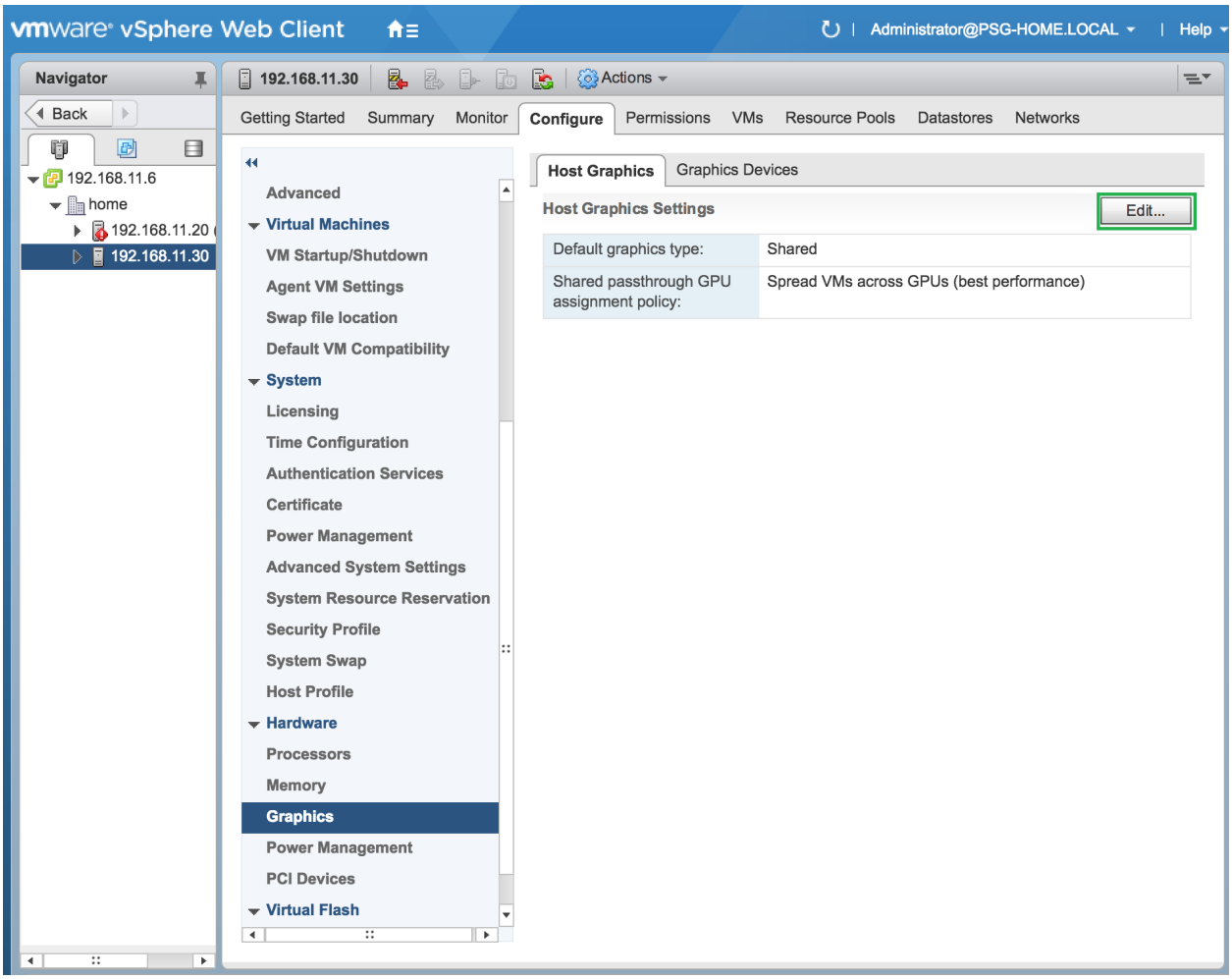


Figure 2.7: Shared default graphics type

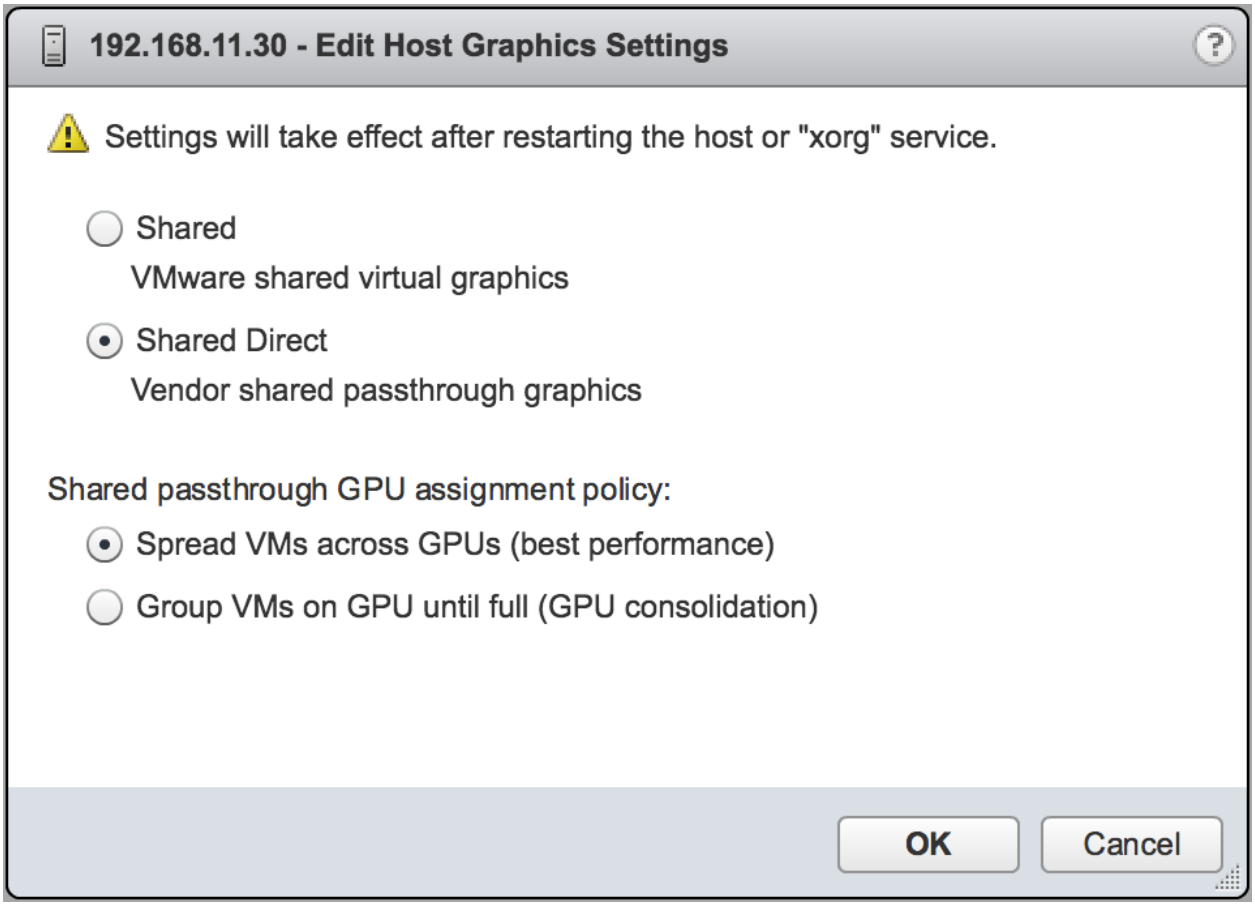


Figure 2.8: Host graphics settings for vGPU

1. Select the physical GPU and click the **Edit icon**.

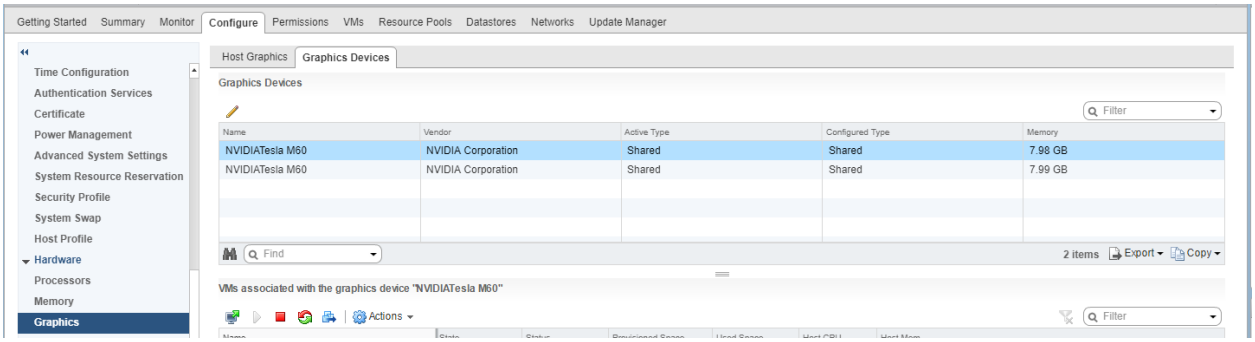


Figure 2.9: Shared graphics type

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

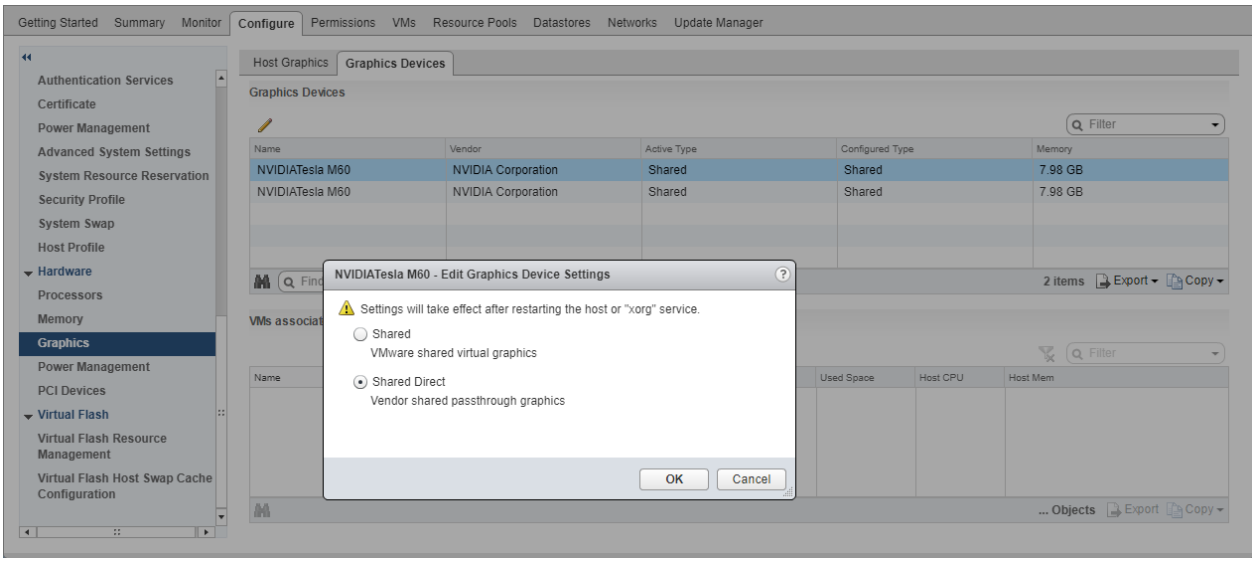


Figure 2.10: Graphics device settings for a physical GPU

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

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

   1. Stop `nv-hostengine`.

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

   2. Wait for 1 second to allow `nv-hostengine` to stop.
   3. Start `nv-hostengine`.

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

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

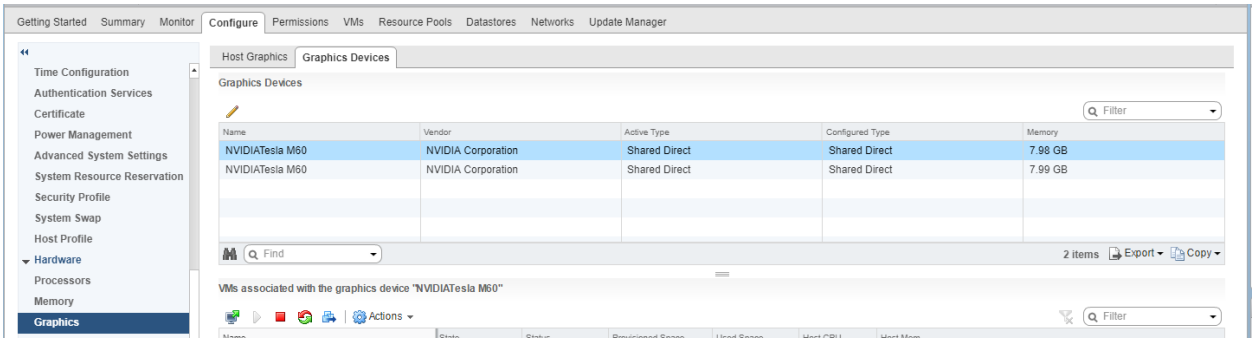


Figure 2.11: Shared direct graphics type

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

See also the following topics in the VMware vSphere documentation:

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

2.9.7. Configuring a vSphere VM with NVIDIA vGPU

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

For details about which VMware vSphere versions and NVIDIA vGPUs support the assignment of multiple vGPUs to a VM, see Virtual GPU Software for VMware vSphere Release Notes.

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 Omnissa 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.

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

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

   1. From the **ADD NEW DEVICE** menu, choose **PCI Device**.
   2. In the Device Selection window that opens, select the type of vGPU you want to configure and click **SELECT**.

      Note

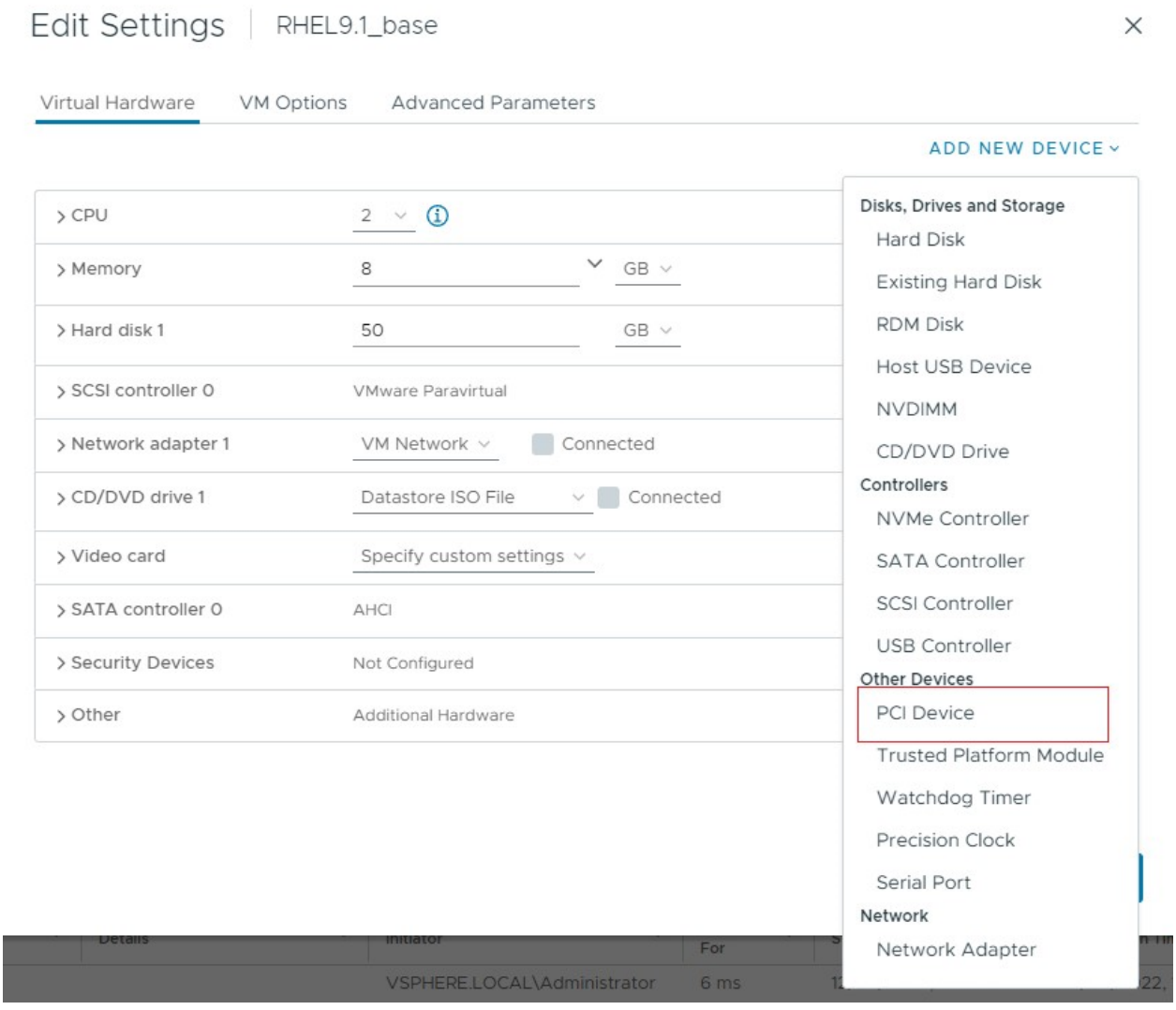


Figure 2.12: Command for Adding a PCI Device

NVIDIA vGPU software does **not** support NVIDIA vGPU for Compute on VMware vSphere. Therefore, C-series vGPU types are not available for selection in the Device Selection window.

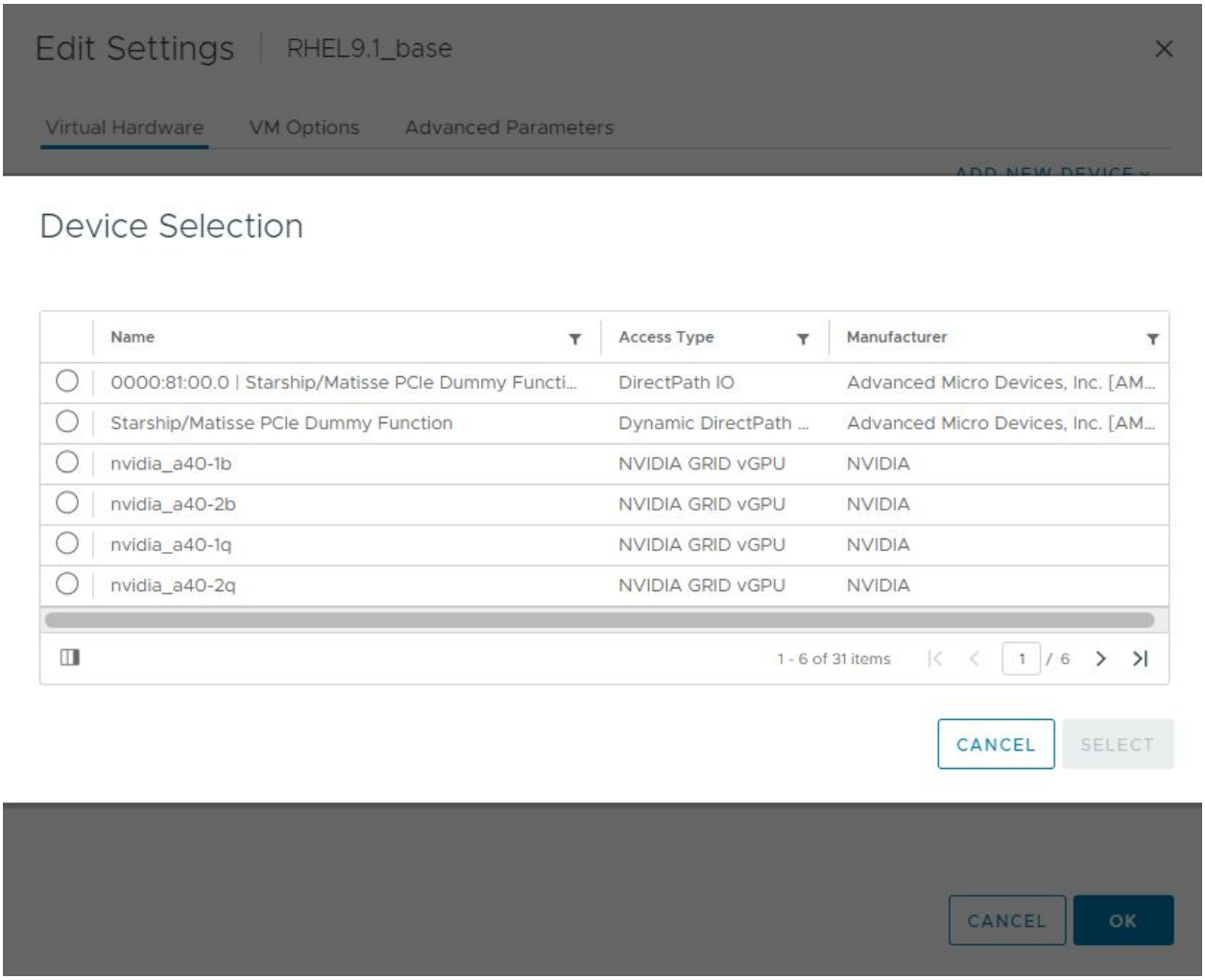


Figure 2.13: VM Device Selections for vGPU

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

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 Omnissa 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*.

2.9.8. Setting vGPU Plugin Parameters on VMware vSphere

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

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

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

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

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

   - `pciPassthru0.cfg.<parameter>`
   - `pciPassthru1.cfg.<parameter>`

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

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

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

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

If you're configuring an NVIDIA vGPU that requires a large BAR address space on a UEFI VM, refer to NVIDIA vGPU software graphics driver fails to load on KVM-based hypervisors for a workaround to ensure that BAR resources are mapped into the VM.

> **Note**
>
> This workaround involves setting an experimental QEMU parameter.

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

Sometimes when configuring a physical GPU for use with NVIDIA 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 that represents a physical GPU, see *NVIDIA vGPU Information in the sysfs File System*.

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

   ```
   # lspci | grep NVIDIA
   ```

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

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

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

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

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

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

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

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

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

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

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

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

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

An NVIDIA vGPU that supports SR-IOV resides on a physical GPU that supports SR-IOV, such as a GPU based on the NVIDIA Ampere architecture. Before creating an NVIDIA vGPU on a GPUthat supports SR-IOV, you must enable the virtual functions of the GPU and obtain the domain, bus, slot, and function of the specific virtual function on which you want to create the vGPU.

Before performing this task, ensure that the GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

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

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

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

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

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

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

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

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

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

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

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

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

   ```
   # ls -l /sys/bus/pci/devices/0000:41:00.0/ | grep virtfn
   lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn0 -> ../0000:41:00.
   ↪4
   lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn1 -> ../0000:41:00.
   ↪5
   ```

(continues on next page)

(continued from previous page)

```
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn10 -> ../
↪0000:41:01.6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn11 -> ../
↪0000:41:01.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn12 -> ../
↪0000:41:02.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn13 -> ../
↪0000:41:02.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn14 -> ../
↪0000:41:02.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn15 -> ../
↪0000:41:02.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn16 -> ../
↪0000:41:02.4
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn17 -> ../
↪0000:41:02.5
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn18 -> ../
↪0000:41:02.6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn19 -> ../
↪0000:41:02.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn2 -> ../0000:41:00.
↪6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn20 -> ../
↪0000:41:03.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn21 -> ../
↪0000:41:03.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn22 -> ../
↪0000:41:03.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn23 -> ../
↪0000:41:03.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn24 -> ../
↪0000:41:03.4
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn25 -> ../
↪0000:41:03.5
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn26 -> ../
↪0000:41:03.6
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn27 -> ../
↪0000:41:03.7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn28 -> ../
↪0000:41:04.0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn29 -> ../
↪0000:41:04.1
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn3 -> ../0000:41:00.
↪7
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn30 -> ../
↪0000:41:04.2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn31 -> ../
↪0000:41:04.3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn4 -> ../0000:41:01.
↪0
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn5 -> ../0000:41:01.
↪1
```

(continues on next page)

(continued from previous page)

```
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn6 -> ../0000:41:01.
↪2
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn7 -> ../0000:41:01.
↪3
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn8 -> ../0000:41:01.
↪4
lrwxrwxrwx. 1 root root           0 Jul 16 04:42 virtfn9 -> ../0000:41:01.
↪5
```

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

On systems configured with NVLink, the `sriov-manage` script might not be able to enable the virtual functions for the physical GPU because the initialization of the Virtual GPU Manager has not completed. In this situation, the `sriov-manage` script writes the following error message to the log file on the hypervisor host:

```
NVRM: Timeout occurred in event processing by vgpu_mgr daemon
```

If this error message appears in the log file, try again later to run the `sriov-manage` script to enable the virtual functions for the physical GPU.

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

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

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

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

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

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

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

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

- Red Hat Enterprise Linux with KVM 10.0
- Ubuntu 24.04

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

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

On systems configured with NVLink, the vGPU might not be created because the initialization of the Virtual GPU Manager has not completed. In this situation, the following error message is written to the log file on the hypervisor host:

```
NVRM: kvgpumgrCreateRequestVgpu: GPU is not initialized yet
```

If this error message appears in the log file, try again later to create the vGPU.

2.10.3.1 Creating a Legacy NVIDIA vGPU on a Linux with KVM Hypervisor

A legacy NVIDIA vGPU does not support SR-IOV.

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

   ```
   # cd /sys/class/mdev_bus/<domain>\:<bus>\:<slot>.<function>/mdev_
   ↪supported_types/
   ```

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

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

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

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

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

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

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

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

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

```
# cat <subdirectory>/available_instances
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
# mdevctl define --auto --uuid <uuid>
```

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

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

7. Confirm that the vGPU was created.

    1. Confirm that the `/sys/bus/mdev/devices/` directory contains the `mdev` device file for the vGPU.

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

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

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

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

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

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

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

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

    ```
    # cd /sys/class/mdev_bus/<domain>\:<bus>\:<vf-slot>.<v-function>/mdev_
    ↪supported_types/
    ```

    `<domain>`, `<bus>`
    The domain and bus of the GPU, without the `0x` prefix.

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

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

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

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

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

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

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

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

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

```
# cat <subdirectory>/available_instances
```

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

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

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

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

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

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

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

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

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

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

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

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

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

6. **MIG-backed vGPUs only:** Ensure that the vGPU is created on the GPU instance that you want.

If you are creating multiple vGPUs on the same GPU instance, perform this step for each vGPU to ensure that all vGPUs are on the same GPU instance.

1. Determine the GPU instance on which the vGPU was created.

```
$ cat /sys/bus/mdev/devices/<uuid>/nvidia/gpu_instance_id
```

2. If the vGPU was **not** created on the GPU instance that you want, write the GPU instance ID that you do want to the `gpu_instance_id` file for the vGPU.

```
$ echo <gpu-instance-id> /sys/bus/mdev/devices/<uuid>/nvidia/gpu_
↪instance_id
```

If the vGPU was created on the GPU instance that you want, omit this step.

<gpu-instance-id>
The ID of the GPU instance on which you want the vGPU to be created.

<uuid>
The UUID of the vGPU that you created.

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

```
# mdevctl define --auto --uuid <uuid>
```

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

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

8. Confirm that the vGPU was created.
   1. Confirm that the `/sys/bus/mdev/devices/` directory contains a symbolic link to the `mdev` device file.

```
# ls -l /sys/bus/mdev/devices/
total 0
lrwxrwxrwx. 1 root root 0 Jul 16 05:57 aa618089-8b16-4d01-a136-
↪25a0f3c73123 -> ../../../devices/pci0000:40/0000:40:01.1/0000:41:00.
↪4/aa618089-8b16-4d01-a136-25a0f3c73123
```

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

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

2.10.3.3 Creating an NVIDIA vGPU on a Linux with KVM Hypervisor that Uses a Vendor-Specific VFIO Framework

A hypervisor uses a vendor-specific VFIO framework **only** for an NVIDIA vGPU that supports SR-IOV. For a legacy NVIDIA vGPU, the hypervisor uses the standard VFIO framework. A vendor-specific VFIO framework does not support the mediated VFIO `mdev` driver framework.

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

- Red Hat Enterprise Linux with KVM 10.0
- Ubuntu 24.04

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

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

1. Change to the directory in the `sysfs` file system that contains the files for vGPU management on the virtual function on which you want to create the vGPU.

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

`<domain>`, `<bus>`
The domain and bus of the GPU, without the `0x` prefix.

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

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

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

2. Confirm that the directory contains the files for vGPU management on the virtual function, namely `creatable_vgpu_types` and `current_vgpu_type`.

```
# ll
-r--r--r-- 1 root root       4096 Aug  3 00:39 creatable_vgpu_types
-rw-r--r-- 1 root root       4096 Aug  3 00:39 current_vgpu_type
...
```

3. Confirm that a vGPU has not already been created on the virtual function.

```
# cat current_vgpu_type
0
```

If the current vGPU type is 0, a vGPU has not already been created on the virtual function.

> **Note**
> If the current vGPU type is not 0, a vGPU **cannot** be created on the virtual function because a vGPU has already been created on it and only one vGPU can be created on a virtual function.

4. Determine the NVIDIA vGPU types that can be created on the virtual function and the integer ID that represents each vGPU type in the `sysfs` file system.

```
# cat creatable_vgpu_types
NVIDIA A40-1Q           557
NVIDIA A40-2Q           558
NVIDIA A40-3Q           559
NVIDIA A40-4Q           560
NVIDIA A40-6Q           561
```

5. Write the ID that represents the type of the NVIDIA vGPU that you want to create to the `current_vgpu_type` file.

```
# echo <vgpu-type-id> > current_vgpu_type
```

<vgpu-type-id>
The ID that represents the type of the NVIDIA vGPU that you want to create in the `sysfs` file system.

> **Note**
> You must specify an valid ID. If you specify an invalid ID, the write operation fails and current vGPU type is set to `0`.

This example creates an instance of the A40-4Q vGPU type.

```
# echo 560 > current_vgpu_type
```

6. Confirm that current vGPU type on the virtual function matches the type of the vGPU that you created in the previous step.

```
# cat current_vgpu_type
560
```

7. Confirm that the `creatable_vgpu_types` file is empty, signifying that no vGPUs can be created on the virtual function.

```
# cat creatable_vgpu_types
```

8. **MIG-backed vGPUs only:** Ensure that the vGPU is created on the GPU instance that you want.

   If you are creating multiple vGPUs on the same GPU instance, perform this step for each vGPU to ensure that all vGPUs are on the same GPU instance.

   1. Determine the GPU instance on which the vGPU was created.

```
$ cat gpu_instance_id
```

   2. If the vGPU was **not** created on the GPU instance that you want, write the GPU instance ID that you do want to the `gpu_instance_id` file for the vGPU.

```
$ echo <gpu-instance-id> gpu_instance_id
```

      If the vGPU was created on the GPU instance that you want, omit this step.

<gpu-instance-id>
The ID of the GPU instance on which you want the vGPU to be created.

To reconfigure the vGPU on a virtual function, the existing vGPU must first be deleted as explained in *Deleting a vGPU on a Linux with KVM Hypervisor that Uses a Vendor-Specific VFIO Framework*.

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

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

For details about which hypervisor 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 and Virtual GPU Software for Ubuntu 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 a Linux with KVM Hypervisor*.

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

- The `virsh` command
- The QEMU command line

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

```
# virsh start <vm-name>
```

`<vm-name>`
The name of the VM that you added the vGPUs to.

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

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

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

   ```
   # virsh edit <vm-name>
   ```

   `<vm-name>`
   The name of the VM to that you want to add the vGPUs to.

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

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

   For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

   - Red Hat Enterprise Linux with KVM 10.0
   - Ubuntu 24.04
   - For a hypervisor that uses the **`mdev` VFIO framework**, add a device entry that identifies the vGPU through its UUID as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

   - Red Hat Enterprise Linux with KVM 10.0
   - Ubuntu 24.04
   - For each vGPU on a hypervisor that uses the **`mdev` VFIO framework**, add a `-device` option that identifies the vGPU through its UUID.

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

     `<vgpu-uuid>`
     The UUID that was assigned to the vGPU when the vGPU was created.

   - For each vGPU on a hypervisor that uses a **vendor-specific VFIO framework**, add a `-device` option that identifies the vGPU through the virtual function on which the vGPU is created.

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

     `<domain>`, `<bus>`
     The domain and bus of the GPU, without the `0x` prefix.

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

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

   ```
   -uuid <vm-uuid>
   ```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

- Red Hat Enterprise Linux with KVM 10.0
- Ubuntu 24.04
- For a hypervisor that uses the **mdev VFIO framework**, change to the `nvidia` subdirectory of the `mdev` device directory that represents the vGPU.

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

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

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

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

<domain>, <bus>
The domain and bus of the GPU, without the `0x` prefix.

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

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

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

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

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

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

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

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

This example enables unified memory for a vGPU.

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

This example enables NVIDIA CUDA Toolkit debuggers for a vGPU.

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

This example enables NVIDIA CUDA Toolkit profilers for a vGPU.

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

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

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

2.10.6. Deleting a vGPU on a Linux with KVM Hypervisor

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

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

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

- Red Hat Enterprise Linux with KVM 10.0
- Ubuntu 24.04

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

NVIDIA vGPU Type	VFIO Framework	Instructions
Legacy: SR-IOV not supported	`mdev`	*Deleting a vGPU on a Linux with KVM Hypervisor that Uses the mdev VFIO Framework*
SR-IOV supported	`mdev`	*Deleting a vGPU on a Linux with KVM Hypervisor that Uses the mdev VFIO Framework*
SR-IOV supported	Vendor specific	*Deleting a vGPU on a Linux with KVM Hypervisor that Uses a Vendor-Specific VFIO Framework*

2.10.6.1 Deleting a vGPU on a Linux with KVM Hypervisor that Uses the `mdev` VFIO Framework

For each vGPU that you want to delete, perform this task in a Linux command shell on the Linux with KVM hypervisor 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 a Linux with KVM Hypervisor*.
- 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
```

2.10.6.2 Deleting a vGPU on a Linux with KVM Hypervisor that Uses a Vendor-Specific VFIO Framework

A hypervisor uses a vendor-specific VFIO framework **only** for an NVIDIA vGPU that supports SR-IOV. For a legacy NVIDIA vGPU, the hypervisor uses the `mdev` VFIO framework. A vendor-specific VFIO framework does not support the mediated VFIO `mdev` driver framework.

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

- Red Hat Enterprise Linux with KVM 10.0
- Ubuntu 24.04

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

- You have the following information:
  - The domain and bus of the GPU where the vGPU that you want to delete resides. For instructions, see *Getting the BDF and Domain of a GPU on a Linux with KVM Hypervisor*.
  - The slot and function of the virtual function on which the vGPU that you want to delete was created.
- The VM to which the vGPU is assigned is shut down.

1. Change to the directory in the `sysfs` file system that contains the files for vGPU management on the virtual function on which the vGPU was created.

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

<domain>, <bus>
The domain and bus of the GPU, without the `0x` prefix.

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

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

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

2. Confirm that the directory contains the files for vGPU management on the virtual function, namely `creatable_vgpu_types` and `current_vgpu_type`.

```
# ll
-r--r--r-- 1 root root       4096 Aug  3 00:39 creatable_vgpu_types
-rw-r--r-- 1 root root       4096 Aug  3 00:39 current_vgpu_type
...
```

3. Confirm that the current vGPU type on the virtual function is the ID that represents the type of the vGPU that you want to delete.

```
# cat current_vgpu_type
560
```

4. Write `0` to the `current_vgpu_type` file.

```
# echo 0 > current_vgpu_type
```

5. Confirm that current vGPU type on the virtual function is `0`, signifying that the vGPU has been deleted.

```
# cat current_vgpu_type
0
```

6. Confirm that the `creatable_vgpu_types` file is no longer empty, signifying that the vGPU has been deleted and that a vGPU can again be created on the virtual function.

```
# cat creatable_vgpu_types
NVIDIA A40-1Q           557
NVIDIA A40-2Q           558
NVIDIA A40-3Q           559
NVIDIA A40-4Q           560
NVIDIA A40-6Q           561
```

2.10.7. 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 a Linux with KVM Hypervisor*.

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.

   1. Change to the `sysfs` directory that represents the `vfio-pci` kernel module.

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

   2. 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
   ```

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

   1. 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
   ```

   2. Write the kernel module name `nvidia` to the `driver_override` file in this directory.

      ```
      # echo nvidia > driver_override
      ```

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

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

4. 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`.

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

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*.

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

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

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

Note

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

For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

- Red Hat Enterprise Linux with KVM 10.0
- Ubuntu 24.04

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

- *NVIDIA vGPU Information in the sysfs File System for Hypervisors that Use the mdev VFIO Framework*
- *NVIDIA vGPU Information in the sysfs File System for Hypervisors that Use a Vendor-Specific VFIO Framework*

2.10.8.1 NVIDIA vGPU Information in the `sysfs` File System for Hypervisors that Use the `mdev` VFIO Framework

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

```
/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:

```
# ll /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
lrwxrwxrwx. 1 root root 0 Dec 12 03:20 0000:07:00.0 -> ../../devices/
 ↪pci0000:00/0000:00:03.0/0000:03:00.0/0000:04:10.0/0000:07:00.0
lrwxrwxrwx. 1 root root 0 Dec 12 03:20 0000:08:00.0 -> ../../devices/
 ↪pci0000:00/0000:00:03.0/0000:03:00.0/0000:04:11.0/0000:08:00.0
```

`mdev_supported_types`

A directory named `mdev_supported_types` is required under the `sysfs` directory for each physical GPU that will be configured with NVIDIA vGPU. How this directory is created for a GPU depends on whether the GPU supports SR-IOV.

- For a GPU that does not support SR-IOV, this directory is created automatically after the Virtual GPU Manager is installed on the host and the host has been rebooted.
- For a GPU that supports SR-IOV, such as a GPU based on the NVIDIA Ampere architecture, you must create this directory by enabling the virtual function for the GPU as explained in *Creating an NVIDIA vGPU on a Linux with KVM Hypervisor*. The `mdev_supported_types` directory itself is never visible on the physical function.

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:

```
# ll 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
```

nvidia-<vgputype-id>
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.

> **Note**
>
> When a time-sliced vGPU is created, the content of the `available_instances` for all other time-sliced vGPU types on the physical GPU is set to 0. This behavior enforces the requirement that all time-sliced vGPUs on a physical GPU must be of the same type. However, this requirement does not apply to MIG-backed vGPUs. Therefore, when a MIG-backed vGPU is created, `available_instances` for all other MIG-backed vGPU types on the physical GPU is not set to 0

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
```

2.10.8.2 NVIDIA vGPU Information in the sysfs File System for Hypervisors that Use a Vendor-Specific VFIO Framework

A vendor-specific VFIO framework does not support the mediated VFIO `mdev` driver framework. 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/bus/pci/devices/` directory.

The organization of the `sysfs` directory for each virtual function on a physical GPU is as follows:

```
/sys/bus/pci/devices/
           |-<virtual-function>
             |-nvidia
               |-creatable_vgpu_types
               |-current_vgpu_type
               |-vgpu_params
```

<virtual-function>
Each virtual function on each physical GPU on the host is represented by a subdirectory of the `/sys/bus/pci/devices/` directory.

The name of each subdirectory is as follows:

`<domain>\:<bus>\:<vf-slot>.<v-function>`

`<domain>` and `<bus>` are the domain and bus of the GPU. `<vf-slot>` and `<v-function>` are the slot and function of the virtual function. For example: `0000\:3d\:00.4`.

You must create this directory by enabling the virtual function for the GPU as explained in *Creating an NVIDIA vGPU on a Linux with KVM Hypervisor*. This directory is **not** created automatically.

nvidia
The `nvidia` directory contains the files for vGPU management on the virtual function. These files are as follows:

creatable_vgpu_types
This file contains the NVIDIA vGPU types that can be created on the virtual function and the integer ID that represents each vGPU type in the `sysfs` file system. For example:

```
# cat creatable_vgpu_types
NVIDIA A40-1Q           557
NVIDIA A40-2Q           558
NVIDIA A40-3Q           559
NVIDIA A40-4Q           560
NVIDIA A40-6Q           561
```

This file is **not** a static list of all the NVIDIA vGPU types that the GPU supports. It is updated dynamically in response to changes to the `current_vgpu_type` file for this virtual function and for other virtual functions on the same GPU.

- If a vGPU has been created on this virtual function, this file is empty.
- If a vGPU has been created on another virtual function on the same GPU, this file contains only the vGPU types that can reside on the same GPU as the existing vGPU.
- If the maximum number of vGPUs that the GPU supports has been created on other virtual functions for the GPU, this file is empty.

> **Note**
>
> When a time-sliced vGPU is created on a GPU in equal-size mode, the content of the `creatable_vgpu_types` for all virtual functions on the physical GPU is set to only the vGPU types with the same amount of frame buffer as the vGPU that was created. This behavior enforces the requirement that all time-sliced vGPUs on the physical GPU must have the same amount of frame buffer. However, this requirement does not apply to time-sliced vGPUs created on a GPU in mixed-size mode or to MIG-backed vGPUs.

`current_vgpu_type`

This file contains the integer ID that represents the vGPU type in the `sysfs` file system of the vGPU that is created on this virtual function. For example, if an NVIDIA A40-4Q vGPU has been created on this virtual function, this file contains the integer `560`:

```
# cat current_vgpu_type
560
```

If no vGPU is created on the virtual function, this file contains the integer 0. When this file is created, its contents are set to the default value of 0.

This file is used for creating and deleting a vGPU on the virtual function.

- A vGPU is created by writing the integer ID that represents the vGPU type in the `sysfs` file system to this file.
- A vGPU is deleted by writing 0 to this file.

`vgpu_params`

This file is used for setting plugin parameters for the vGPU on the virtual function to control its behavior. Plugin parameters are set by writing a list of parameter-value pairs to this file. For more information, refer to *Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor*.

2.11. Putting a GPU or GPU Instance into Mixed-Size Mode

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

Note

- How a GPU or GPU instance in mixed-size mode behaves if the hypervisor host is rebooted, the NVIDIA Virtual GPU Manager is reloaded, or the GPU is reset depends on the hypervisor:
  - On a Linux with KVM hypervisor, a GPU or GPU instance in mixed-size mode reverts to its default mode.
  - On VMware vSphere, a GPU in mixed-size mode remains in mixed-size mode. The GPU **doesn't** revert to its default mode
- When a GPU or GPU instance is in mixed-size mode, only the best effort and equal share schedulers are supported. The fixed share scheduler is **not** supported.

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

How to put a GPU or GPU instance into mixed-size mode depends on the hypervisor that you are using as explained in the following topics:

- *Putting a GPU into Mixed-Size Mode on a Linux with KVM Hypervisor*
- *Putting a GPU Instance into Mixed-Size Mode on a Linux with KVM Hypervisor*
- *Putting a GPU Into Mixed-Size Mode on VMware vSphere*

2.11.1. Putting a GPU into Mixed-Size Mode on a Linux with KVM Hypervisor

Perform this task only for a single-instance GPU. If you try to perform this task for a GPU instance on a MIG-enabled GPU, the `nvidia-smi` commands fail or might not return meaningful information.

For a GPU instance on a MIG-enabled GPU, perform the task in *Putting a GPU Instance into Mixed-Size Mode on a Linux with KVM Hypervisor*.

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

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

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

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

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

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

> **Note**
> To put a GPU into equal size mode, run this command with the `-shm 0` option.

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

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

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

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

2.11.2. Putting a GPU Instance into Mixed-Size Mode on a Linux with KVM Hypervisor

Perform this task only for a GPU instance on a MIG-enabled GPU. If you try to perform this task for a single-instance GPU, the `nvidia-smi vgpu` commands fail or do not return meaningful information.

For a single instance GPU, perform the task in *Putting a GPU into Mixed-Size Mode on a Linux with KVM Hypervisor*.

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

- MIG mode has been enabled for the GPU as explained in *Enabling MIG Mode for a GPU*.
- The GPU instance that you want to put into mixed-size mode has been created as explained in *Creating GPU Instances on a MIG-Enabled GPU*.
- If you are using a GPU that supports SR-IOV, ensure that the virtual functions for the physical GPU in the `sysfs` file system are enabled as explained in *Preparing the Virtual Function for an NVIDIA vGPU that Supports SR-IOV on a Linux with KVM Hypervisor*.

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

```
# nvidia-smi -q
...
Attached GPUs                         : 1
GPU 00000000:C7:00.0
...
        Heterogeneous Time-Slice Sizes    : Supported
...
```

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

```
# nvidia-smi vgpu -i <id> –gi <gpu-instance-id> -shm 1
```

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

<gpu-instance-id>
The identifier of the GPU instance that you want to put into mixed-size mode.

> **Note**
> To put a GPU instance into equal size mode, run this command with the `-shm 0` option.

This example puts the GPU instance 3 on the GPU with index `00000000:C7:00.0` into mixed-size mode.

```
# nvidia-smi vgpu -i 0 -gi 3 -shm 1
Enabled vGPU heterogeneous mode on GI Id 3
```

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

```
# nvidia-smi vgpu -i <id> –gi <gpu-instance-id> -ghm
```

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

<gpu-instance-id>
The identifier of the GPU instance that you want to put into mixed-size mode.

This example confirms that GPU instance 3 on the GPU with index `00000000:C7:00.0` is in mixed-size mode.

```
# nvidia-smi vgpu -i 0 -gi 3 -ghm
GPU 00000000:C7:00.0
    GPU Instance ID: 3
        vGPU Heterogeneous Mode           : Enabled
```

2.11.3. Putting a GPU Into Mixed-Size Mode on VMware vSphere

On VMware vSphere, you can put a GPU into mixed-size mode either by using vCenter Server or by using the `esxcli` command.

For instructions, refer to the following topics:

- *Putting a GPU Into Mixed-Size Mode on VMware vSphere by Using vCenter Server*
- *Putting a GPU Into Mixed-Size Mode on VMware vSphere by Using the esxcli Command*

2.11.3.1 Putting a GPU Into Mixed-Size Mode on VMware vSphere by Using vCenter Server

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 navigation tree in the **Configure** tab, select **Hardware** > **Graphics**.
4. Select the GPU and click **Edit**.
5. In the pop-up window that opens, set **vGPU Mode** to **Mixed Size**.

> **Note**
>
> Do **not** set the option to restart the Xorg server. This option is required only when the device type is changed, **not** when the vGPU mode of a GPU is changed.

2.11.3.2 Putting a GPU Into Mixed-Size Mode on VMware vSphere by Using the `esxcli` Command

Perform this task from ESXi Shell on the hypervisor host.

1. Run the `esxcli` command to change the vGPU mode of the GPU to mixed size.

```
$ esxcli graphics device set --device-id=<slot>:<bus>:<domain>.<function>
↪--type SharedPassthru --vgpu-mode=MixedSize
```

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

> **Note**
>
> To put a GPU into equal size mode, run this command with the `--vgpu-mode=SameSize` option.

2. Refresh the host and rebuild the assignable hardware tree.

```
$ esxcli graphics host refresh
```

3. Confirm that the vGPU mode of the GPU has been changed.

```
$ esxcli graphics device list
```

2.12. Placing a vGPU on a Physical GPU or a GPU Instance on a Linux with KVM Hypervisor

By default, the Virtual GPU Manager determines where a vGPU is placed on a GPU or GPU instance by assigning a placement ID when the `mdev` that represents the vGPU is created. On a Linux with KVM hypervisor, you can control the placement of vGPUs on a GPU or GPU instance. The benefits of controlling the placement of vGPUs depend on whether the GPU or GPU instance is in mixed-size mode or equal-size mode.

- In **mixed-size mode**, you can fit as many vGPUs as possible on the GPU or GPU instance by ensuring that no gaps that can be occupied by a vGPU are left in the placement region.

- In **equal-size mode**, you can fix the location in the frame buffer at which the vGPU's virtual frame buffer is allocated so that you can verify that frame buffer content is cleared when a vGPU is shut down and restarted.

The vGPU placements that a GPU supports depend on the total amount of frame buffer that the GPU has and whether the GPU is in mixed-size mode or equal-size mode. For details, refer to *vGPU Placements for GPUs*.

Note

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

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

- The vGPU that you want to place on the physical GPU or GPU instance has been created as explained in *Creating an NVIDIA vGPU on a Linux with KVM Hypervisor*.
- The VM to which the vGPU is assigned is **not** running. If the VM is running when you try to place the vGPU, an error occurs.

For detailed instructions, refer to the following topics:

- *Placing a vGPU on a Physical GPU on a Linux with KVM Hypervisor*
- *Placing a vGPU on a GPU Instance on a Linux with KVM Hypervisor*

2.12.1. Placing a vGPU on a Physical GPU on a Linux with KVM Hypervisor

Perform this task only for a single-instance GPU. If you try to perform this task for a GPU instance on a MIG-enabled GPU, the commands fail or might not return meaningful information.

For a GPU instance on a MIG-enabled GPU, perform the task in *Placing a vGPU on a GPU Instance on a Linux with KVM Hypervisor*.

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

1. If you want to place a vGPU on a GPU in equal-size mode, confirm that the GPU supports **Homogeneous Placements**.

```
# nvidia-smi -q | grep -A 4 "vGPU Device Capability"
    vGPU Device Capability
        Fractional Multi-vGPU             : Supported
        ...
        Homogeneous Placements            : Supported
```

2. Confirm that the GPU is in the mode that you want (mixed-size mode or equal-size mode).

```
# nvidia-smi -q | grep "vGPU Heterogeneous Mode"
        vGPU Heterogeneous Mode           : Disabled
```

   If necessary, change the mode to mixed-size mode or equal-size mode as explained in *Putting a GPU into Mixed-Size Mode on a Linux with KVM Hypervisor*.

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

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

Note

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

```
# nvidia-smi vgpu -s -v
...
    vGPU Type ID                            : 0x392
        Name                                : NVIDIA L4-6Q
...
        Placement Size                      : 6
        Supported Placement IDs             : 0 6 12 18
...
```

The number of supported placement IDs is the maximum number of vGPUs of the type that are allowed on the GPU.

4. Determine the current placement ID of the vGPU by viewing the contents of the `sysfs` file that represents the placement ID of the vGPU.

   - For hypervisors that use the `mdev` VFIO framework, run the following command:

     ```
     # cat /sys/bus/mdev/devices/<uuid>/nvidia/placement_id
     ```

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

   - For hypervisors that use a vendor-specific VFIO framework, run the following command:

     ```
     # cat /sys/bus/pci/devices/<domain>:<bus>:<slot>.<function>/nvidia/
     ↪placement_id
     ```

   `<domain>`, `<bus>`, `<slot>`, `<function>`
   The domain, bus, slot, and function of the vGPU, without the `0x` prefix, for example, `0000\:3d\:00.4`.

5. Write the new placement ID that you want to the `sysfs` file that represents the placement ID of the vGPU.

   - For hypervisors that use the `mdev` VFIO framework, run the following command:

     ```
     # echo <placement-id> > /sys/bus/mdev/devices/<uuid>/nvidia/placement_
     ↪id
     ```

   - For hypervisors that use a vendor-specific VFIO framework, run the following command:

```
# echo <placement-id> > /sys/bus/pci/devices/<domain>:<bus>:<slot>.
↪<function>/nvidia/placement_id
```

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

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

<domain>, <bus>, <slot>, <function>
The domain, bus, slot, and function of the vGPU, without the `0x` prefix, for example, `0000\:3d\:00.4`.

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

```
# echo 6> /sys/bus/mdev/devices/aa618089-8b16-4d01-a136-25a0f3c73123/
↪nvidia/placement_id
```

This example sets the placement ID for the vGPU that has the domain, bus, slot, and function `0000\:3d\:00.4` to 6.

```
# echo 6> /sys/bus/pci/devices/0000\:3d\:00.4/nvidia/placement_id
```

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

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

2.12.2. Placing a vGPU on a GPU Instance on a Linux with KVM Hypervisor

Perform this task only for a GPU instance on a MIG-enabled GPU. If you try to perform this task for a single-instance GPU, the commands fail or might not return meaningful information.

For a single instance GPU, perform the task in *Placing a vGPU on a Physical GPU on a Linux with KVM Hypervisor*.

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

1. If you want to place a vGPU on a GPU instance in equal-size mode, confirm that the underlying physical GPU supports **Homogeneous Placements**.

```
# nvidia-smi -q | grep -A 4 "vGPU Device Capability"
    vGPU Device Capability
        Fractional Multi-vGPU             : Supported
        ...
        Homogeneous Placements            : Supported
```

2. Confirm that the GPU instance is in the mode that you want (mixed-size mode or equal-size mode).

```
# nvidia-smi vgpu -i <id> –gi <gpu-instance-id> -ghm
```

<id>
The index of the GPU.

<gpu-instance-id>
The identifier of the GPU instance on which you want to place a vGPU.

If necessary, change the mode to mixed-size mode or equal-size mode as explained in *Putting a GPU Instance into Mixed-Size Mode on a Linux with KVM Hypervisor*.

This example confirms that GPU instance 3 on the GPU with index `00000000:C7:00.0` is in mixed-size mode.

```
# nvidia-smi vgpu -i 0 -gi 3 -ghm
GPU 00000000:C7:00.0
    GPU Instance ID: 3
        vGPU Heterogeneous Mode           : Enabled
```

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

```
# nvidia-smi vgpu -i <id> –gi <gpu-instance-id> -c -v
```

<id>
The index of the GPU.

<gpu-instance-id>
The identifier of the GPU instance on which you want to place a vGPU.

The number of supported placement IDs is the maximum number of vGPUs of the type that are allowed on the GPU instance.

This example lists the placement size and available placement IDs for the NVIDIA RTX Pro 6000 Blackwell DC-1-2Q vGPU type on GPU instance 3 on the GPU with index `00000000:C7:00.0`.

```
# nvidia-smi vgpu -i 0 -gi 3 -c -v
GPU 00000000:C7:00.0
   GPU Instance ID 3
    vGPU Type ID                          : 0x60a
        Name                              : NVIDIA RTX Pro 6000 Blackwell
↪DC-1-2Q
...
        Placement Size                    : 2
        Placement IDs                     : 0 2 4 6 8 10 12 14 16 18 20 22
```

4. Determine the current placement ID of the vGPU on the GPU instance by viewing the contents of the `sysfs` file that represents the placement ID of the vGPU.

   - For hypervisors that use the `mdev` VFIO framework, run the following command:

```
# cat /sys/bus/mdev/devices/<uuid>/nvidia/gpu_instance_id/placement_id
```

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

   - For hypervisors that use a vendor-specific VFIO framework, run the following command:

```
# cat /sys/bus/pci/devices/<domain>:<bus>:<slot>.<function>/gpu_
↪instance_id/nvidia/placement_id
```

<domain>, <bus>, <slot>, <function>
The domain, bus, slot, and function of the vGPU, without the `0x` prefix, for example, `0000\:3d\:00.4`.

5. Write the new placement ID that you want to the `sysfs` file that represents the placement ID of the vGPU.

   - For hypervisors that use the `mdev` VFIO framework, run the following command:

   ```
   # echo <placement-id> > /sys/bus/mdev/devices/<uuid>/nvidia/gpu_
   ↪instance_id/placement_id
   ```

   - For hypervisors that use a vendor-specific VFIO framework, run the following command:

   ```
   # echo <placement-id> > /sys/bus/pci/devices/<domain>:<bus>:<slot>.
   ↪<function>/nvidia/gpu_instance_id/placement_id
   ```

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

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

   <domain>, <bus>, <slot>, <function>
   The domain, bus, slot, and function of the vGPU, without the `0x` prefix, for example, `0000\:3d\:00.4`.

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

   ```
   # echo 6> /sys/bus/mdev/devices/aa618089-8b16-4d01-a136-25a0f3c73123/
   ↪nvidia/gpu_instance_id/placement_id
   ```

   This example sets the placement ID for the vGPU that has the domain, bus, slot, and function `0000\:3d\:00.4` to 6.

   ```
   # echo 6> /sys/bus/pci/devices/0000\:3d\:00.4/nvidia/gpu_instance_id/
   ↪placement_id
   ```

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

```
# nvidia-smi vgpu -i <id> –gi <gpu-instance-id> -q
```

<id>
The index of the GPU.

<gpu-instance-id>
The identifier of the GPU instance on which you placed the vGPU.

For example:

```
# nvidia-smi vgpu -i 0 -gi 3 -q
GPU 00000000:41:00.0
    Active vGPUs                          : 1
```

(continues on next page)

(continued from previous page)

```
    vGPU ID                                 : 3251634224
        VM UUID                             : 604de595-d65c-4e6f-acb5-
↪b178dec1da0d
        ...
        Placement ID                        : 6
        ...
```

2.13. Configuring a GPU for MIG-Backed vGPUs

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

Ensure that the following prerequisites are met:

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

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

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

1. *Enabling MIG Mode for a GPU*

   Note

   For VMware vSphere, only enabling MIG mode is required because VMware vSphere creates the GPU instances and, after the VM is booted and guest driver is installed, one compute instance is automatically created in the VM.

2. *Creating GPU Instances on a MIG-Enabled GPU*
3. *Creating Compute Instances in a GPU instance*

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

2.13.1. Enabling MIG Mode for a GPU

Perform this task in your hypervisor command shell.

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

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

2. Determine whether MIG mode is enabled.

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

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

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------
↪----+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08  CUDA Version:  13.2
↪    |
|-------------------------------+----------------------+-----------------
↪----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
↪ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
↪Compute M. |
|                               |                      |
↪MIG M. |
|===============================+======================+=====================|
|   0 NVIDIA RTX PRO 6000...On  | 00000000:9B:00.0 Off |
↪Off |
| N/A  35C  P8     40W /  600W  |     0MiB /  97887MiB |       0%
↪Default |
+-------------------------------+----------------------+-----------------
↪----+
|                               |                      |
↪Disabled |
+-------------------------------+----------------------+-----------------
↪----+
```

3. If MIG mode is disabled, enable it.

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

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

This example enables MIG mode on GPU 0.

```
$ nvidia-smi -i 0 -mig 1
Enabled MIG Mode for GPU 00000000:9B:00.0
All done.
```

Note

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

4. **VMware vSphere ESXi with GPUs based on the NVIDIA Ampere architecture only:** Reboot the hypervisor host.

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

5. Query the GPUs on which you enabled MIG mode to confirm that MIG mode is enabled.

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

```
$ nvidia-smi -i 0 --query-gpu=pci.bus_id,mig.mode.current --format=csv
pci.bus_id, mig.mode.current
00000000:9B:00.0, Enabled
```

2.13.2. Creating GPU Instances on a MIG-Enabled GPU

Note

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

Perform this task in your hypervisor command shell.

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

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

```
$ nvidia-smi mig -lgip
+-----------------------------------------------------------------------------
↪----+
| GPU instance profiles:
↪     |
| GPU   Name               ID   Instances   Memory     P2P    SM    DEC
↪ENC  |
|                               Free/Total   GiB              CE    JPEG
↪OFA  |
|=============================================================================|
|   0  MIG 1g.24gb         14   4/4         23.12      No     46     1
↪1    |
|                                                              1     1
↪0    |
+-----------------------------------------------------------------------------
↪----+
|   0  MIG 1g.24gb+me      21   1/1         23.12      No     46     1
↪1    |
|                                                              1     1
↪1    |
+-----------------------------------------------------------------------------
↪----+
|   0  MIG 1g.24gb+gfx     47   4/4         23.12      No     46     1
↪1    |
```

(continues on next page)

(continued from previous page)

```
|                                                                          1      1
↪0    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 1g.24gb+me.all 65   1/1          23.12      No     46      4
↪4    |
|                                                                          1      4
↪1    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 1g.24gb-me     67    4/4          23.12      No     46      0
↪0    |
|                                                                          1      0
↪0    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 2g.48gb         5    2/2          46.50      No     94      2
↪2    |
|                                                                          2      2
↪0    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 2g.48gb+gfx    35    2/2          46.50      No     94      2
↪2    |
|                                                                          2      2
↪0    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 2g.48gb+me.all 64    1/1          46.50      No     94      4
↪4    |
|                                                                          2      4
↪1    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 2g.48gb-me     66    2/2          46.50      No     94      0
↪0    |
|                                                                          2      0
↪0    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 4g.96gb         0    1/1          93.38      No     188     4
↪4    |
|                                                                          4      4
↪1    |
+------------------------------------------------------------------------------
↪----+
|   0  MIG 4g.96gb+gfx    32    1/1          93.38      No     188     4
↪4    |
|                                                                          4      4
↪1    |
+------------------------------------------------------------------------------
↪----+
```

> **Note**
>
> On GPUs that support the Multi-Instance GPU (MIG) feature, NVIDIA vGPU software supports only GPU instance profiles for graphics. GPU instance profiles for graphics are denoted by the `+gfx` suffix, for example, `MIG 4g.96gb+gfx`.

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

```
$ nvidia-smi mig -cgi <gpu-instance-profile-ids> -C
```

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

> **Note**
>
> NVIDIA vGPU software supports only the default compute instance, which spans the complete GPU instance.If you omit the `-C` option from the command to create the GPU instances, they are created without compute instances. In this situation, you must add the compute instances manually as explained in *Creating Compute Instances in a GPU instance*.

This example creates one GPU instance of type `2g.48gb+gfx`, which has profile ID 35, with the default compute instance.

```
$ nvidia-smi mig -cgi 35 -C
Successfully created GPU instance ID  1 on GPU  0 using profile MIG 2g.
↪48gb+gfx (ID 35)
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  1
↪using profile MIG 2g.48gb (ID  1)
```

2.13.3. Creating Compute Instances in a GPU instance

If you did **not** create GPU instances with the default compute instance, you must add the default compute instance explicitly. Otherwise, omit this task. The default compute instance spans the complete GPU. If you plan to use MIG-backed vGPUs that occupy the entire GPU instance, you can reconfigure the compute instances later for individual vGPUs from within the guest VMs.

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

Perform this task in your hypervisor command shell.

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

```
$ nvidia-smi mig -lgi
+-------------------------------------------------------+
| GPU instances:                                        |
| GPU   Name             Profile  Instance   Placement  |
|                          ID       ID       Start:Size |
|=======================================================|
|   0  MIG 2g.48gb+gfx     35        1          0:6     |
+-------------------------------------------------------+
```

3. List the compute instance profiles that are available in the GPU instances within which you want to create the compute instances.

```
$ nvidia-smi mig -lcip -gi <gpu-instance-ids>
```

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

If a compute instance has already been created, an asterisk (*) is displayed against the the profile ID.

This example lists the compute instance profiles that are available in GPU instance 1.

```
$ nvidia-smi mig -lcip -gi 1
+-----------------------------------------------------------------------------
 ↪-----+
| Compute instance profiles:
 ↪      |
| GPU    GPU       Name       Profile Instances   Exclusive       Shared
 ↪      |
|      Instance                 ID    Free/Total     SM       DEC   ENC
 ↪OFA  |
|        ID                                                    CE    JPEG
 ↪      |
|=============================================================================|
|   0     1       MIG 1c.2g.48gb 0     2/2           46        2     2
 ↪ 0    |
|                                                              2     2
 ↪      |
+-----------------------------------------------------------------------------
 ↪-----+
|   0      1      MIG 1c.2g.48gb 7     1/1           80        2     2
 ↪ 0    |
|                                                              2     2
 ↪      |
+-----------------------------------------------------------------------------
 ↪-----+
|   0      1      MIG 2g.48gb    1*    1/1           94        2     2
 ↪ 0    |
|                                                              2     2
 ↪      |
+-----------------------------------------------------------------------------
 ↪-----+
```

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

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

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

NVIDIA vGPU software supports only the default compute instance, which spans the complete GPU instance.

Caution

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

This example creates the default compute instance on GPU instance 1.

```
$ nvidia-smi mig -cci -gi 1
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  1
 ↪using profile MIG 2g.48gb (ID  1)
```

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

```
$ nvidia-smi
+-----------------------------------------------------------------------------
 ↪----+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08  CUDA Version:  13.2
 ↪    |
|-------------------------------+----------------------+------------------
 ↪----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
 ↪ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
 ↪Compute M. |
|                               |                      |
 ↪MIG M. |
|===============================+======================+======================|
|   0 NVIDIA RTX PRO 6000...On  | 00000000:9B:00.0 Off |
 ↪Off |
| N/A  35C  P8     40W /  600W  |     0MiB /  97887MiB |       0%
 ↪Default |
+-------------------------------+----------------------+------------------
 ↪----+
|                               |                      |
 ↪Enabled |
+-------------------------------+----------------------+------------------
 ↪----+

+-----------------------------------------------------------------------------
 ↪----+
| MIG devices:
 ↪    |
```

(continues on next page)

(continued from previous page)

```
+------------------+----------------------+-----------+--------------------
↪----+
| GPU  GI  CI  MIG |  Shared Memory-Usage |        Vol|         Shared
↪    |
|      ID  ID  Dev |    Shared BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA
↪ JPG|
|                  |                      |        ECC|
↪    |
|==================+======================+===========+======================|
|  0    1   0   0  | 144MiB / 47616MiB    | 94    N/A |  2   2    2    0
↪  2 |
|                  |   0MiB / 17736MiB    |           |
↪    |
+------------------+-- ------------------+-----------+--------------------
↪----+

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

Note

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

2.14. Disabling MIG Mode for One or More GPUs

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

Ensure that the following prerequisites are met:

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

Perform this task in your hypervisor command shell.

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

   You can use secure shell (SSH) for this purpose.

2. Determine whether MIG mode is disabled.

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

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

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------
↪----+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08  CUDA Version:  13.2
↪     |
|-------------------------------+----------------------+------------------
↪----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
↪ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
↪Compute M. |
|                               |                      |
↪MIG M. |
|===============================+======================+======================|
|   0 NVIDIA RTX PRO 6000...On  | 00000000:9B:00.0 Off |
↪Off |
| N/A  35C  P8     40W /  600W  |   144MiB /  97887MiB |      N/A
↪Default |
+-------------------------------+----------------------+------------------
↪----+
|                               |                      |
↪Enabled |
+-------------------------------+----------------------+------------------
↪----+
```

3. If MIG mode is enabled, disable it.

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

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

This example disables MIG mode on GPU 0.

```
$ sudo nvidia-smi -i 0 -mig 0
Disabled MIG Mode for GPU 00000000:9B:00.0
All done.
```

4. Confirm that MIG mode was disabled.

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

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

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------
↪----+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08  CUDA Version:  13.2
↪    |
|-------------------------------+----------------------+-----------------
↪----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
↪ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
↪Compute M. |
|                               |                      |
↪MIG M. |
|===============================+======================+=====================|
|   0 NVIDIA RTX PRO 6000...On  | 00000000:9B:00.0 Off |
↪Off |
| N/A  35C  P8     40W /  600W  |   144MiB /  97887MiB |      0%
↪Default |
|                               |                      |
↪Disabled |
+-------------------------------+----------------------+-----------------
↪----+
```

2.15. Disabling and Enabling ECC Memory

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

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

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

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

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

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

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

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

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

2.15.1. Disabling ECC Memory

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

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

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

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

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

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

```
# nvidia-smi -q

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

Timestamp                           : Mon May 18 18:36:45 2026
Driver Version                      : 595.112.08

Attached GPUs                       : 1
GPU 0000:02:00.0

[...]

    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled

[...]
```

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

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

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

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

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

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

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

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

```
# nvidia-smi -q

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

Timestamp                           : Mon May 18 18:37:53 2026
Driver Version                      : 595.112.08

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

    Ecc Mode
        Current                     : Disabled
        Pending                     : Disabled

[...]
```

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

2.15.2. Enabling ECC Memory

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

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

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

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

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

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

```
# nvidia-smi -q

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

Timestamp                           : Mon May 18 18:36:45 2026
Driver Version                      : 595.112.08

Attached GPUs                       : 1
GPU 0000:02:00.0

[...]

    Ecc Mode
        Current                     : Disabled
        Pending                     : Disabled

[...]
```

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

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

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

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

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

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

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

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

```
# nvidia-smi -q

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

Timestamp                           : Mon May 18 18:37:53 2026
Driver Version                      : 595.112.08

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

    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled
```

(continues on next page)

(continued from previous page)

```
[...]
```

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

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

To use NVIDIA® GPUDirect Storage® technology with NVIDIA vGPU, you must install all the required software in the VM that is configured with NVIDIA vGPU.

Ensure that the prerequisites in *Prerequisites for Using NVIDIA vGPU* are met.

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

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

   - `--with-nvmf`
   - `--with-nfsrdma`
   - `--enable-gds`
   - `--add-kernel-support`
4. Install the NVIDIA vGPU software graphics driver for Linux in the VM from a distribution-specific package.

   Note

   GPUDirect Storage technology does **not** support installation of the NVIDIA vGPU software graphics driver for Linux from a `.run` file.

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

   - *Installing the NVIDIA vGPU Software Graphics Driver on Ubuntu from a Debian Package*
   - *Installing the NVIDIA vGPU Software Graphics Driver on Red Hat Distributions from an RPM Package*
5. Install NVIDIA CUDA Toolkit from a `.run` file, deselecting the CUDA driver when selecting the CUDA components to install.

   Note

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

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

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

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

   - RHEL8/Rocky 8

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

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

   - Ubuntu

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

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

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

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, all GPUs that support NVIDIA vGPU software 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**
- All GPUs directly connected to each other through NVLink must be assigned to the same VM.

You can assign multiple physical GPUs to one VM. The maximum number of physical GPUs that you can assign to a VM depends on the maximum number of PCIe pass-through devices per VM that your chosen hypervisor can support. For more information, refer to the documentation for your hypervisor, for example:

- **XenServer:** Configuration limits
- **Red Hat Enterprise Linux:**
  - **Red Hat Enterprise Linux 9 releases:** Assigning a GPU to a virtual machine, *Known Issues*
  - **Red Hat Enterprise Linux 8 releases:** Assigning a GPU to a virtual machine, *Known Issues*
  - **Red Hat Enterprise Linux 7 releases:** GPU PCI Device Assignment
- **VMware vSphere:** VMware Configuration Maximums

Note

If you intend to configure all GPUs in your server platform for pass-through, you do not need to install the NVIDIA Virtual GPU Manager.

3.1. Display Resolutions for Physical GPUs

The display resolutions supported by a physical GPU depend on the NVIDIA vGPU software license that is applied to the GPU.

3.1.1. NVIDIA RTX Virtual Workstation Physical GPU Resolutions

GPUs that are licensed with an NVIDIA RTX Virtual Workstation license support a maximum combined resolution based on the number of available pixels. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these GPUs.

The following table lists the maximum number of displays per GPU at each supported display resolution for configurations in which all displays have the same resolution.

Available Pixels	Display Resolution	Displays per GPU
66355200	7680×4320	2
	5120×2880 or lower	4

The following table provides examples of configurations with a mixture of display resolutions.

Available Pixels	Available Pixel Basis	Maximum Displays	Sample Mixed Display Configurations
66355200	2 7680×4320 displays	4	1 7680×4320 display plus 2 5120×2880 displays
			1 7680×4320 display plus 3 4096×2160 displays

Note

You cannot use more than four displays even if the combined resolution of the displays is less than the number of available pixels from the GPU. For example, you cannot use five 4096×2160 displays even though the combined resolution of the displays (44236800) is less than the number of available pixels from the GPU (66355200).

3.1.2. NVIDIA Virtual Applications or NVIDIA vGPU for Compute Physical GPU Resolutions

GPUs that are licensed with an NVIDIA Virtual Applications or an NVIDIA vGPU for Compute license support a single display with a fixed maximum resolution. The maximum resolution depends on the following factors:

- The NVIDIA vGPU software license that is applied to the GPU

- The operating system that is running on the system to which the GPU is assigned

License	Operating System	Maximum Display Resolution	Displays per GPU
NVIDIA Virtual Applications	Linux	2560×1600	1
	Windows	1280×1024	1

3.2. Using GPU Pass-Through on XenServer

You can configure a GPU for pass-through on 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.
- `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.

3.2.1. 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:

After configuring a 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*.

3.2.2. Configuring a VM for GPU Pass Through by Using xe

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

```
[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

[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
[root@xenserver ~]#
```

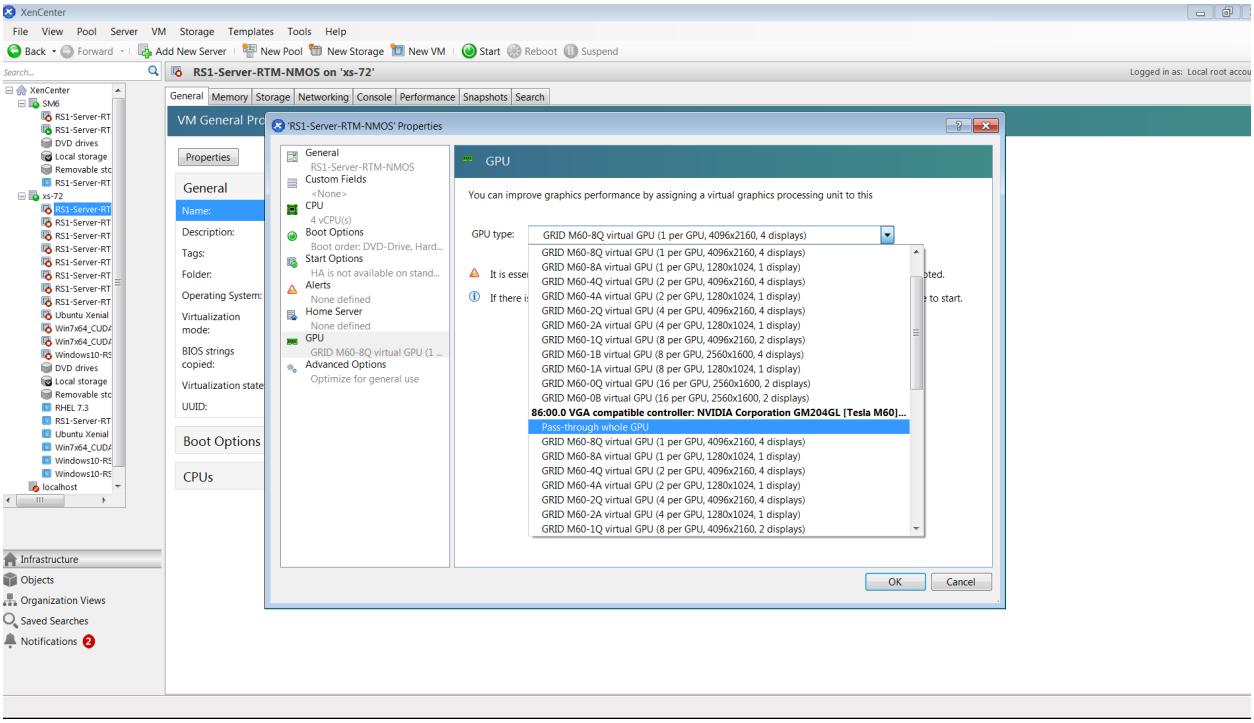


Figure 3.1: Using XenCenter to configure a pass-through GPU

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 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*.

3.3. Using GPU Pass-Through on a Linux with KVM Hypervisor

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

You can configure a GPU for pass-through on a Linux with KVM hypervisor 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 or Ubuntu, ensure that the following prerequisites are met:

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

> **Note**
>
> Do not create any virtual disks in `/root`.

- A virtual machine has been created.

If you're configuring a pass-through GPU that requires a large BAR address space on a UEFI VM, refer to NVIDIA vGPU software graphics driver fails to load on KVM-based hypervisors for a workaround to ensure that BAR resources are mapped into the VM.

> **Note**
>
> This workaround involves setting an experimental QEMU parameter.

3.3.1. 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 the 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*.

3.3.2. 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 VG compatible controller: NVIDIA Corporation GM204GL [Tesla
↪M60] (rev a1)
 86:00.0 VG 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='0x0000' bus='0x00' slot='0x05' function=
↪'0x0'/>
</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='0x00' slot='0x05' function=
↪'0x0'/>
</hostdev>
```

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

```
# virsh start <vm-name>
```

   `<vm-name>`
   The name of the VM that you assigned the GPU to.

After configuring the 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*.

3.3.3. 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.

```
# 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:

```
-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.

```
-device vfio-pci,host=85:00.0
```

After configuring the 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*.

3.3.4. 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 or Ubuntu 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 a Linux with KVM Hypervisor*.

1. If you are using a GPU that supports SR-IOV, such as a GPU based on the NVIDIA Ampere architecture, disable the virtual function for the GPU in the `sysfs` file system.

   If your GPU does not support SR-IOV, omit this step.

   > **Note**
   > Before performing this step, ensure that the GPU is not being used by any other processes, such as CUDA applications, monitoring applications, or the `nvidia-smi` command.

   Use the custom script `sriov-manage` provided by NVIDIA vGPU software for this purpose.

```
# /usr/lib/nvidia/sriov-manage -d <domain>:<bus>:<slot>.<function>
```

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

This example disables the virtual function for the GPU with the domain `00`, bus `06`, slot `0000`, and function `0`.

```
# /usr/lib/nvidia/sriov-manage -d 00:06:0000.0
```

2. 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
```

3. To ensure that no clients are using the GPU, acquire the unbind lock of the GPU.
   1. 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).

   2. 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
```

   3. Write the value 1 to the `unbindLock` file in this directory.

```
# echo 1 > unbindLock
```

   4. 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.

4. Unbind the GPU from `nvidia` kernel module.
   1. Change to the `sysfs` directory that represents the `nvidia` kernel module.

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

   2. 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
```

5. Bind the GPU to the `vfio-pci` kernel module.
   1. 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
```

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

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

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

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

   4. 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`.

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

   5. Change back to the `sysfs` directory that contains the PCI device information for the physical GPU.

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

   6. Clear the content of the `driver_override` file in this directory.

```
# echo > driver_override
```

You can now configure the GPU for use in pass-through mode as explained in *Using GPU Pass-Through on a Linux with KVM Hypervisor*.

3.4. Using GPU Pass-Through on Microsoft Windows Server

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

3.4.1. 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.

If you are using an actively cooled NVIDIA Quadro graphics card such as the RTX 8000 or RTX 6000, you must also pass through the audio device on the graphics card.

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
  - 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.
   1. In the device manager, context-click the GPU and from the menu that pops up, choose **Properties**.
   2. 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:

   ```
   PCIROOT(80)#PCI(0200)#PCI(0000)#PCI(1000)#PCI(0000)
   ```

2. If you are using an actively cooled NVIDIA Quadro graphics card, obtain the location path of the audio device on the graphics card and disable the device.
   1. In the device manager, from the **View** menu, choose **Devices by connection**.
   2. Navigate to **ACPI x64-based PC** > **Microsoft ACPI-Compliant System** > **PCI Express Root Complex** > **PCI-to-PCI Bridge**.
   3. Context-click **High Definition Audio Controller** and from the menu that pops up, choose **Properties**.
   4. In the Properties window that opens, click the **Details** tab and in the **Properties** drop-down list, select **Location paths**.

5. Context-click **High Definition Audio Controller** again and from the menu that pops up, choose **Disable device**.

3. Dismount the GPU and, if present, the audio device from host to make them unavailable to the host so that they can be used solely by the VM.

   For each device that you are dismounting, type the following command:

```
Dismount-VMHostAssignableDevice -LocationPath <gpu-device-location> -force
```

   <gpu-device-location>
   The location path of the GPU or the audio device that you obtained previously.

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

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

4. Assign the GPU and, if present, the audio device that you dismounted in the previous step to the VM.

   For each device that you are assigning, type the following command:

```
Add-VMAssignableDevice -LocationPath <gpu-device-location> -VMName <vm-
↪name>
```

   <gpu-device-location>
   The location path of the GPU or the audio device that you dismounted in the previous step.

   <vm-name>
   The name of the VM to which you are attaching the GPU or the audio device.

   Note

   You can assign a pass-through GPU and, if present, its audio device 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(00` to the VM `VM1`.

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

5. Power on the VM.

   The guest OS should now be able to use the GPU and, if present, the audio device.

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*.

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

Perform this task in the Windows PowerShell.

If you are using an actively cooled NVIDIA Quadro graphics card such as the RTX 8000 or RTX 6000, you must also return the audio device on the graphics card.

1. List the GPUs and, if present, the audio devices that are currently assigned to the virtual machine (VM).

```
Get-VMAssignableDevice -VMName <vm-name>
```

<vm-name>
The name of the VM whose assigned GPUs and audio devices you want to list.

2. Shut down the VM to which the GPU and any audio devices are assigned.
3. Remove the GPU and, if present, the audio device from the VM to which they are assigned.

   For each device that you are removing, type the following command:

```
Remove-VMAssignableDevice –LocationPath <gpu-device-location> -VMName <vm-
↪name>
```

<gpu-device-location>
The location path of the GPU or the audio device that you are removing, which you obtained previously.

<vm-name>
The name of the VM from which you are removing the GPU or the audio device.

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 and, if present, its audio device are removed from the VM, they are unavailable to the host operating system (OS) until you remount them on the host OS.

4. Remount the GPU and, if present, its audio device on the host OS.

   For each device that you are remounting, type the following command:

```
Mount-VMHostAssignableDevice –LocationPath <gpu-device-location>
```

<gpu-device-location>
The location path of the GPU or the audio device that you are remounting, which you specified in the previous step to remove the GPU or the audio device 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 and, if present, its audio device.

3.5. 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.

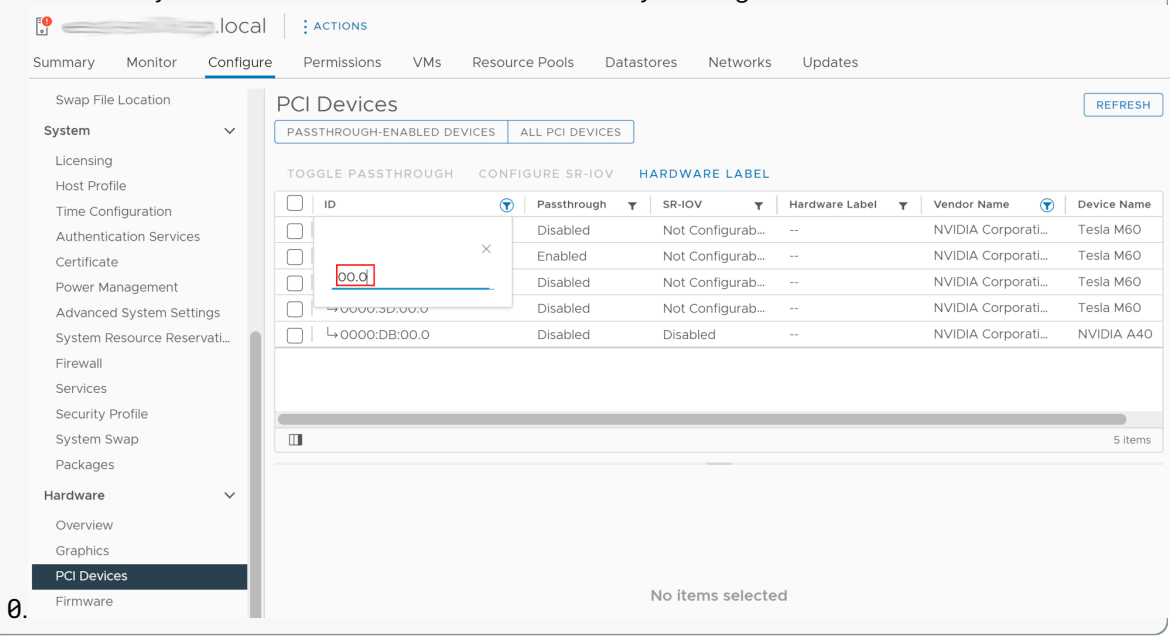
Before configuring a vSphere VM with vDGA, ensure that these prerequisites are met

- 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 **Configure**.
3. From the **Hardware** menu, choose **PCI Devices**.
4. On the PCI Devices page that opens, click **ALL PCI DEVICES** and in the table of devices, select the GPU.

Note

When selecting the GPU to pass through, you must select only the **physical** device. To list only NVIDIA physical devices, set the filter on the **Vendor Name** field to `NVIDIA` and filter out any virtual function devices of the GPU by setting the filter on the **ID** field to `00.`



0.

5. Click **TOGGLE PASSTHROUGH**.
6. Reboot the ESXi host.
7. After the ESXi host has booted, right-click the VM and choose **Edit Settings**.
8. From the **New Device** menu, choose **PCI Device** and click **Add**.

9. On the page that opens, from the **New Device** drop-down list, select the GPU.
10. Click **Reserve all memory** and click **OK**.
11. 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*.

Chapter 4. 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 and NVIDIA Control Panel on Windows

To fully enable GPU operation in a VM or on a bare-metal host, the NVIDIA vGPU software graphics driver must be installed. If the NVIDIA Control Panel app is not installed when the graphics driver is installed, you can install it separately from the graphics driver.

4.1.1. 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, namely: Q-series, B-series, and A-series 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.

The procedure for installing the driver is the same in a VM and on bare metal.

1. Copy the 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.



Figure 4.1: NVIDIA driver installation

3. Click through the license agreement.
4. Select **Express Installation** and click **NEXT**.

   After the driver installation is complete, the installer may prompt you to restart the platform.
5. If prompted to restart the platform, do one of the following:
   - Select **Restart Now** to reboot the VM 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.
   1. Right-click on the desktop.
   2. From the menu that opens, choose **NVIDIA Control Panel**.
   3. 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 Software Client Licensing User Guide.

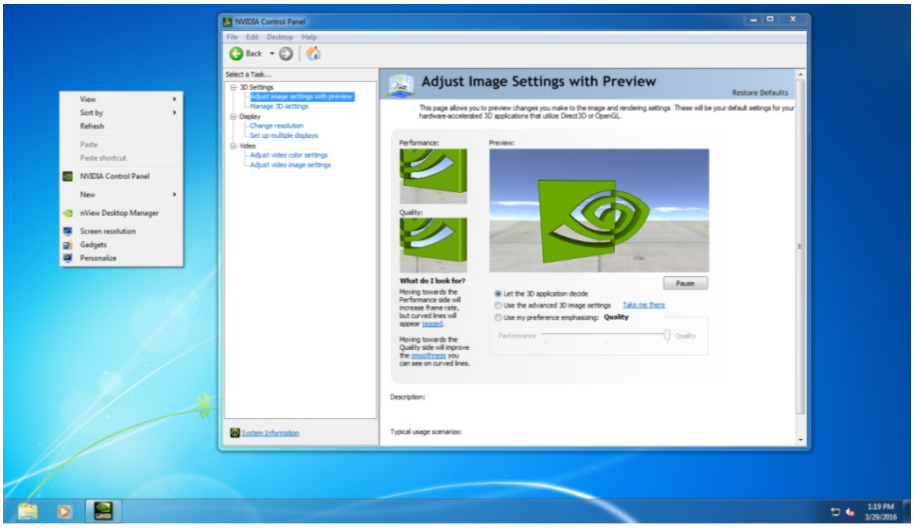


Figure 4.2: Verifying NVIDIA driver operation using NVIDIA Control Panel

Note

The graphics driver for Windows in this release of NVIDIA vGPU software is distributed in a DCH-compliant package. A DCH-compliant package differs from a driver package that is not DCH compliant in the following ways:- The Windows registry key for license settings for a DCH-compliant package is different than the key for a driver package that is not DCH compliant. If you are upgrading from a driver package that is not DCH compliant in a VM that was previously licensed, you must reconfigure the license settings for the VM. Existing license settings are not propagated to the new Windows registry key for a DCH-compliant package.

- NVIDIA System Management Interface, `nvidia-smi`, is installed in a folder that is in the default executable path.
- The NVWMI binary files are installed in the Windows Driver Store under `%SystemDrive%\Windows\System32\DriverStore\FileRepository\`.
- NVWMI help information in Windows Help format is not installed with graphics driver for Windows guest OSes.

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

4.1.2. Installing the Standalone NVIDIA Control Panel App

The NVIDIA Control Panel app is now distributed through the Microsoft Store. If your system does not allow the installation apps from the Microsoft Store, the NVIDIA Control Panel app is not installed when the NVIDIA vGPU software graphics driver for Windows is installed.

Your system might not allow the installation apps from the Microsoft Store for any of the following reasons:

- The Microsoft Store app is disabled.
- Your system is not connected to the Internet

- Installation of apps from the Microsoft Store is blocked by your system settings.

You can install the NVIDIA Control Panel app separately from the graphics driver by downloading and running the standalone NVIDIA Control Panel installer that is available from NVIDIA Licensing Portal.

1. Download and extract the standalone NVIDIA Control Panel installer from NVIDIA Licensing Portal.
2. Copy the extracted standalone NVIDIA Control Panel installer to the guest VM or physical host where you are installing the NVIDIA Control Panel app.
3. Double-click the installer executable file to start the installer.
4. When asked if you want to allow the installer app to make changes to your device, click **Yes**.
5. Accept the NVIDA software license agreement.
6. Select the **Express** installation option and click **NEXT**.
7. When the installation is complete, click **CLOSE** to close the installer.

   The NVIDIA Control Panel app opens.

4.2. Installing the NVIDIA vGPU Software Graphics Driver on Linux

The NVIDIA vGPU software graphics driver for Linux is distributed as a `.run` file that can be installed on all supported Linux distributions. The driver is also distributed as a Debian package for Ubuntu distributions and as an RPM package for Red Hat distributions.

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. Linux guest VMs are supported on all NVIDIA vGPU types, namely: Q-series, B-series, and A-series NVIDIA vGPU types.

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.

In addition to the proprietary release of the NVIDIA vGPU software graphics driver for Linux, a release that is based on NVIDIA Linux open GPU kernel modules is also available. The release that is based on NVIDIA Linux open GPU kernel modules is compatible with the following NVIDIA vGPU software deployments:

- NVIDIA vGPU deployments on GPUs that are based on the NVIDIA Ada Lovelace GPU architecture or later architectures
- Bare-metal deployments on GPUs that are based on the NVIDIA Turing GPU architecture or later architectures

The release that is based on NVIDIA Linux open GPU kernel modules can be installed **only** from the `.run` file, **not** from a Debian package or RPM package.

The procedure for installing the driver is the same in a VM and on bare metal.

Before installing the NVIDIA vGPU software graphics driver, ensure that the following prerequisites are met:

- OpenSSL is installed in the VM. If OpenSSL is not installed, the VM will not be able to obtain NVIDIA vGPU software licenses.

- NVIDIA Direct Rendering Manager Kernel Modesetting (DRM KMS) is disabled. By default, DRM KMS is disabled. However, if it has been enabled, remove `nvidia-drm.modeset=1` from the kernel command-line options.
- If the VM uses UEFI boot, do one of the following:
  - Ensure that **secure boot** is **disabled**.
  - Self-sign the driver as explained in the Linux `README` file that is included in the `.run` file distribution of the driver.
- If the Nouveau driver for NVIDIA graphics cards is present, disable it. For instructions, refer to *Disabling the Nouveau Driver for NVIDIA Graphics Cards*.
- If you are using the Wayland display server protocol, refer to *Configuring the Wayland Display Server Protocol for Use with NVIDIA vGPU Software* for additional prerequisites for installing the NVIDIA vGPU software graphics driver.
- If you are using a Linux OS for which the Wayland display server protocol is enabled by default, and either of the following conditions is met, disable it as explained in *Disabling the Wayland Display Server Protocol for Red Hat Enterprise Linux*.
  - Wayland isn't supported on NVIDIA vGPU software with your Linux OS.
  - You don't want to use Wayland.

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

- *Installing the NVIDIA vGPU Software Graphics Driver on Linux from a .run File*
- *Installing the NVIDIA vGPU Software Graphics Driver on Ubuntu from a Debian Package*
- *Installing the NVIDIA vGPU Software Graphics Driver on Red Hat Distributions from an RPM Package*

Installation in a VM: After you install the NVIDIA vGPU software graphics driver, perform the following tasks:

1. If you are using the Wayland display server protocol, complete the configuration of Wayland for use with NVIDIA vGPU software as explained in *Configuring the Wayland Display Server Protocol for Use with NVIDIA vGPU Software*.
2. License any NVIDIA vGPU software licensed products that you are using as explained in Virtual GPU Software 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*.

4.2.1. Installing the NVIDIA vGPU Software Graphics Driver on Linux from a `.run` File

You can use the `.run` file to install the NVIDIA vGPU software graphics driver on any supported Linux distribution.

Installation of the NVIDIA vGPU software graphics driver for Linux from a `.run` file requires:

- Compiler toolchain
- Kernel headers

If a driver has previously been installed on the guest VM or physical host from a Debian package or RPM package, uninstall that driver before installing the driver from a `.run` file.

If Dynamic Kernel Module Support (DKMS) is enabled, ensure that the `dkms` package is installed.

1. Copy the NVIDIA vGPU software Linux driver package, for example `NVIDIA-Linux_x86_64-595.112.07-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. Switch to a console login prompt.
        - If you have access to the terminal's function keys, press **CTRL-ALT-F1**.
        - If you are accessing the guest VM or physical host through VNC or a web browser and do not have access to the terminal's function keys, run the `chvt` command of the OS as root.

          ```
          [nvidia@localhost ~]$ sudo chvt 3
          ```

     2. Log in and shut down the display manager:
        - For Ubuntu 18 and later releases, stop the `gdm` service.

          ```
          [nvidia@localhost ~]$ sudo service gdm stop
          ```

        - For releases earlier than Ubuntu 18, stop the `lightdm` service.

          ```
          [nvidia@localhost ~]$ sudo service lightdm stop
          ```

3. From a console shell, run the driver installer as the root user.
   - To install the proprietary release of the driver, run the driver installer without any additional options.

     ```
     sudo sh ./NVIDIA-Linux_x86_64-595.112.07-grid.run
     ```

   - To install the release that is based on NVIDIA Linux open GPU kernel modules, run the driver installer with the `-m=kernel-open` option.

     ```
     sudo sh ./NVIDIA-Linux_x86_64-595.112.07-grid.run -m=kernel-open
     ```

   If DKMS is enabled, set the `-dkms` option. This option requires the `dkms` package to be installed.

   ```
   sudo sh ./NVIDIA-Linux_x86_64-595.112.07-grid.run -dkms
   ```

   In some instances, the installer may fail to detect the installed kernel headers and sources. In this situation, rerun the installer, specifying the kernel source path with the `--kernel-source-path` option.

   ```
   sudo sh ./NVIDIA-Linux_x86_64-595.112.07-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`).

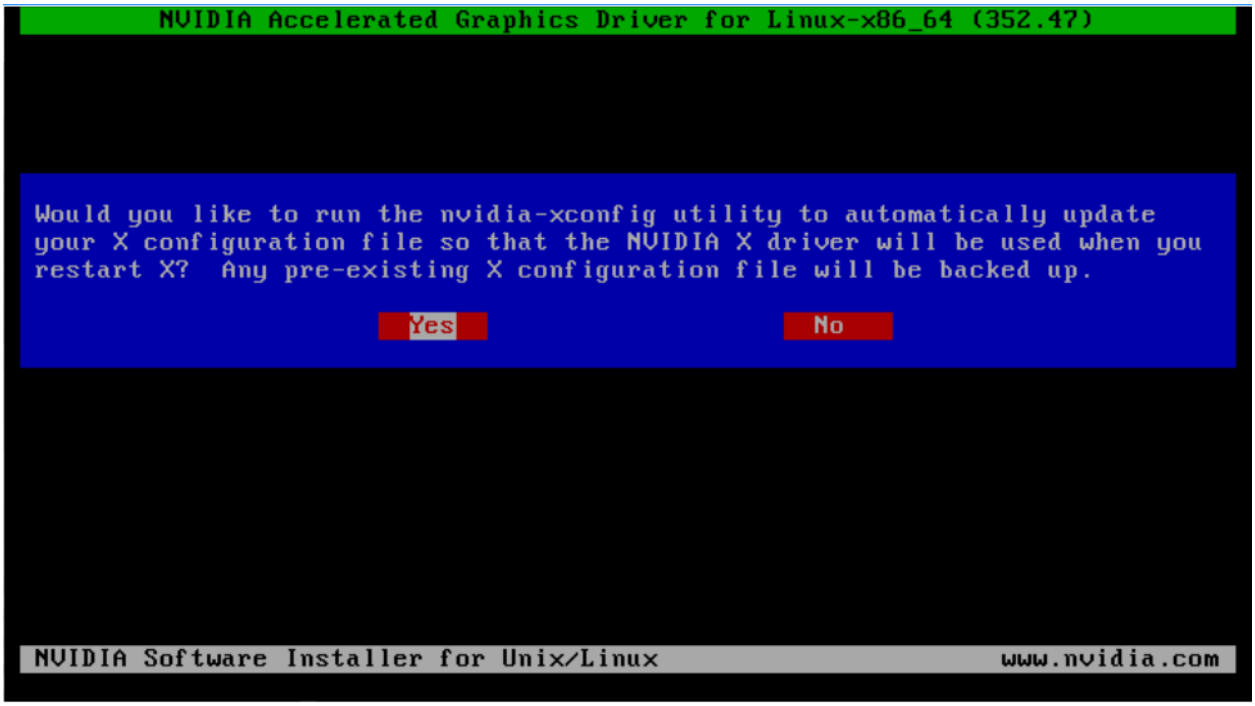


Figure 4.3: Update `xorg.conf` settings

5. After the installation is complete, select **OK** to exit the installer.
6. Verify that the NVIDIA driver is operational.
   1. Reboot the system and log in.
   2. Run `nvidia-settings`.

      ```
      [nvidia@localhost ~]$ nvidia-settings
      ```

      The NVIDIA X Server Settings dialog box opens to show that the NVIDIA driver is operational.

4.2.2. Installing the NVIDIA vGPU Software Graphics Driver on Ubuntu from a Debian Package

The NVIDIA vGPU software graphics driver for Ubuntu is distributed as a Debian package file.

This task requires `sudo` privileges.

1. Copy the NVIDIA vGPU software Linux driver package, for example `nvidia-linux-grid-595_595.112.07_amd64.deb`, to the guest VM where you are installing the driver.
2. Log in to the guest VM as a user with `sudo` privileges.
3. Open a command shell and change to the directory that contains the NVIDIA vGPU software Linux driver package.
4. From the command shell, run the command to install the package.

```
$ sudo apt-get install ./nvidia-linux-grid-595_595.112.07_amd64.deb
```

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

      ```
      $ nvidia-smi
      ```

4.2.3. Installing the NVIDIA vGPU Software Graphics Driver on Red Hat Distributions from an RPM Package

The NVIDIA vGPU software graphics driver for Red Hat Distributions is distributed as an RPM package file.

This task requires `root` user privileges.

1. Copy the NVIDIA vGPU software Linux driver package, for example `nvidia-linux-grid-595_595.112.07-1.x86_64.rpm`, to the guest VM where you are installing the driver.
2. Log in to the guest VM as a user with `root` user privileges.
3. Open a command shell and change to the directory that contains the NVIDIA vGPU software Linux driver package.
4. From the command shell, run the command to install the package.

   ```
   $ rpm -iv ./nvidia-linux-grid-595_595.112.07-1.x86_64.rpm
   ```

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

      ```
      $ nvidia-smi
      ```

4.2.4. Disabling the Nouveau Driver for NVIDIA Graphics Cards

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

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

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

```
$ lsmod | grep nouveau
```

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

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

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

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

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

3. Reboot the host or guest VM.

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

Starting with Red Hat Enterprise Linux Desktop 8.0, the Wayland display server protocol is used by default on supported GPU and graphics driver configurations. If you don't want to use Wayland or if Wayland isn't supported on NVIDIA vGPU software with your Linux OS, disable Wayland before installing the driver. If supported, the OS reverts to the X Window System. Otherwise, the OS can be used only in console and command-line interface (CLI) mode

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

This task requires administrative access.

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

4.2.6. Disabling GSP Firmware

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

Note

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

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

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

Product	GSP	License Request	Error Message
NVIDIA vGPU for Compute	Supported	Allowed	Not applicable
NVIDIA Virtual Applications	Not supported	Blocked	Printed
NVIDIA RTX Virtual Workstation	Not supported	Blocked	Printed

When a license request is blocked, the following error message is written to the licensing event log file at the location given in Virtual GPU Software Client Licensing User Guide:

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

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

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

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

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

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

     ```
     GSP Firmware Version                    : 595.112.07
     ```

   - Otherwise, the command displays `N/A` as the GSP firmware version.

3. If GSP firmware is enabled, disable it by setting the NVIDIA module parameter `NVreg_EnableGpuFirmware` to 0.

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

   ```
   options nvidia NVreg_EnableGpuFirmware=0
   ```

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

4. Reboot the VM or bare-metal host.

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

4.2.7. Configuring the Wayland Display Server Protocol for Use with NVIDIA vGPU Software

To enable remote desktops to operate with the security protocols in the Wayland display server protocol, you must configure Wayland for use with NVIDIA vGPU software. Perform tasks that require the regular Virtual Network Computing (VNC) console before installing the NVIDIA vGPU software graphics driver.

If you are using Red Hat Enterprise Linux 10.1 or 10.0, upgrade the installed GNOME Remote Desktop packages to their latest versions.

Before installing the NVIDIA vGPU software graphics driver, follow this sequence of instructions:

1. *Enabling Remote Desktop Mode for the GNOME Remote Desktop*
2. *Configuring GNOME Remote Desktop for Use with NVIDIA vGPU*

Then ask each remote desktop user to follow this sequence of instructions:

1. *Remote Desktop User: Enabling Access to GNOME Remote Desktop Over RDP*

   The first time this task is performed in a VM, the user is prompted for a password that can be provided only through the VNC console. However, after the NVIDIA vGPU software graphics driver is installed, the VNC console is blank. Therefore, each remote desktop user must perform this task once **before** the NVIDIA vGPU software graphics driver is installed.

2. *Remote Desktop Users: Turning Off Screen Lock*

After each remote desktop user has completed these tasks, continue with the configuration of Wayland for use with NVIDIA vGPU software as follows:

1. Ensure that the generic video card is removed from the configuration on the hypervisor host and only NVIDIA GPUs are attached.
2. Follow the instructions in one of the following topics to install the graphics driver:
   - *Installing the NVIDIA vGPU Software Graphics Driver on Linux from a .run File*
   - *Installing the NVIDIA vGPU Software Graphics Driver on Ubuntu from a Debian Package*
   - *Installing the NVIDIA vGPU Software Graphics Driver on Red Hat Distributions from an RPM Package*
3. If the guest OS is Ubuntu, follow the instructions in *Ubuntu OS Only: Configuring the Mutter Compositor*.

After installing the graphics driver and, if necessary, configuring the Mutter compositor, complete the configuration of Wayland for use with NVIDIA vGPU software as follows:

1. **MIG-backed vGPUs only:** Follow the instructions in *Configuring the Wayland Display Server Protocol for MIG-Backed vGPUs*.
2. Ask each remote desktop user to follow the instructions in *Remote Desktop User: Enabling Access to GNOME Remote Desktop Over RDP* again after every reboot of the VM on which the GNOME Remote Desktop is running.

4.2.7.1 Enabling Remote Desktop Mode for the GNOME Remote Desktop

If you are using the Wayland display server protocol, enable remote desktop mode for each user that will use the GNOME Remote Desktop.

Use the regular Virtual Network Computing (VNC) console to perform this task in a VM in which the NVIDIA vGPU software graphics driver **has not yet been installed**.

1. In the Settings app, choose **System** and click **Remote Desktop**.
2. Click the **Desktop Sharing** tab and ensure that the **Desktop Sharing** and **Remote Control** options are selected.
3. In **Login Details**, set the user name and password of the remote desktop user for whom you are enabling remote desktop mode.
4. Click the **Remote Login** tab and ensure that the **Remote Login** option is deselected.

   This option does not use the path that the NVIDIA vGPU software graphics driver supports.

After enabling remote desktop mode, configure GNOME Remote Desktop for use with NVIDIA vGPU as explained in *Configuring GNOME Remote Desktop for Use with NVIDIA vGPU*.

4.2.7.2 Configuring GNOME Remote Desktop for Use with NVIDIA vGPU

Perform this task for each user for whom remote desktop mode was enabled in *Enabling Remote Desktop Mode for the GNOME Remote Desktop*.

This task requires administrative access.

1. Ensure that GNOME Desktop Manager is configured to start at boot.

   ```
   # sudo systemctl set-default graphical.target
   ```

2. If necessary, enable the Wayland display server protocol.

   If you are using a Linux OS for which the Wayland display server protocol is enabled by default, such as Red Hat Enterprise Linux, and Wayland hasn't previously been disabled, skip this step.

   1. In a plain text editor, edit the file `/etc/gdm/custom.conf` and, under the `[daemon]` section, add the option `WaylandEnable=true`.
   2. If Wayland has previously been disabled, delete the uncommented `WaylandEnable=false` option.
   3. Save your changes to `/etc/gdm/custom.conf`.

3. Enable the user for whom you are performing this task to log in automatically to GNOME Desktop Manager.

   1. In a plain text editor, edit the file `/etc/gdm/custom.conf` and, under the `[daemon]` section, add the following lines:

      ```
      AutomaticLoginEnable = true
      AutomaticLogin = <username>
      ```

<username>
The user name of the user for whom you are performing this task for logging in to the VM on which GNOME Remote Desktop is running.

2. Save your changes to `/etc/gdm/custom.conf`.

4. **Ubuntu OS only:** To prevent `udev` rules from restricting the use of Wayland with the NVIDIA vGPU software graphics driver, comment out all `udev` rules from the file `/usr/lib/udev/rules.d/61-gdm.rules`.

5. Reboot the guest VM.

After configuring GNOME Remote Desktop for use with NVIDIA vGPU, ask each remote desktop user to follow this sequence of instructions:

1. *Remote Desktop User: Enabling Access to GNOME Remote Desktop Over RDP*
2. *Remote Desktop Users: Turning Off Screen Lock*

4.2.7.3 Remote Desktop User: Enabling Access to GNOME Remote Desktop Over RDP

Each remote desktop user must unlock the default GNOME keyring and set a new Remote Desktop Protocol (RDP) password. This task must be performed once as part of the initial setup before the NVIDIA vGPU software graphics driver is installed **and** after every reboot of the VM in which the GNOME Remote Desktop is running.

Perform this task with normal user privileges as the remote desktop user for whom remote desktop mode was enabled in *Enabling Remote Desktop Mode for the GNOME Remote Desktop*.

Ensure that you have secure shell (SSH) access to the VM in which GNOME Remote Desktop is running.

1. Use SSH to log in to the VM in which GNOME Remote Desktop is running.

```
$ ssh <username>@<desktop-vm>
```

<username>
Your user name for logging in to the VM in which GNOME Remote Desktop is running.

<desktop-vm>
The host name or IP address of the VM in which GNOME Remote Desktop is running.

2. Unlock the default keyring by sending your password to the GNOME Keyring daemon and restarting the daemon.

```
$ echo '<password>' | gnome-keyring-daemon --replace --unlock
```

<password>
Your password for logging in to the VM in which GNOME Remote Desktop is running.

If you are performing this task for the first time in the VM, a pop-up window opens in the VNC console to prompt you for your password.

If you have previously performed this task and are performing it again after the VM has been rebooted, you are **not** prompted for your password.

3. If you are prompted for your password, provide it in the pop-up window that opened in the VNC console.

4. Restart the GNOME Remote Desktop service for only yourself to apply your changes.

```
$ systemctl --user restart gnome-remote-desktop.service
```

5. Set your RDP login credentials for connecting to the remote desktop.

```
$ grdctl rdp set-credentials <username> '<password>'
```

<username>
Your RDP user name for connecting to the remote desktop.

<password>
Your RDP password for connecting to the remote desktop.

6. Turn on RDP in GNOME Remote Desktop to enable the session to accept incoming RDP connections.

```
$ grdctl rdp enable
```

7. Restart the GNOME Remote Desktop service so that it runs with the new credentials and RDP enabled.

```
$ systemctl --user restart gnome-remote-desktop.service
```

8. Use an RDP client to connect to your remote desktop.

What to do next depends on whether you have just performed this task for the first time in the VM in which the GNOME Remote Desktop is running or after a reboot of the VM:

- **First time in the VM:** Turn off screen lock as explained in *Remote Desktop Users: Turning Off Screen Lock*.
- **After a reboot of the VM:** No further action is required. Your remote desktop is now ready for use.

4.2.7.4 Remote Desktop Users: Turning Off Screen Lock

To prevent being locked out while connected over RDP, each remote desktop user must turn off screen lock. This task is required only as part of the initial setup before the NVIDIA vGPU software graphics driver is installed.

Use a Remote Desktop Protocol (RDP) client to perform this task with normal user privileges as the remote desktop user for whom remote desktop mode was enabled in *Enabling Remote Desktop Mode for the GNOME Remote Desktop*.

Ensure that you are connected to your remote desktop.

1. In the Settings app, choose **Privacy & Security**.
2. Click **Screen Lock** and ensure that the **Automatic Screen Lock** option is deselected.
3. Set **Blank Screen Delay** to **Never**.

The system administrator can now remove the generic video card from the configuration of the hypervisor host and then install the NVIDIA vGPU software graphics driver by following the instructions in one of the following topics:

- *Installing the NVIDIA vGPU Software Graphics Driver on Linux from a .run File*
- *Installing the NVIDIA vGPU Software Graphics Driver on Ubuntu from a Debian Package*
- *Installing the NVIDIA vGPU Software Graphics Driver on Red Hat Distributions from an RPM Package*

If the guest OS is Ubuntu, after installing the NVIDIA vGPU software graphics driver, the system administrator must follow the instructions in *Ubuntu OS Only: Configuring the Mutter Compositor*.

4.2.7.5 Ubuntu OS Only: Configuring the Mutter Compositor

If you are using the Ubuntu OS with the Wayland display server protocol, configure the Mutter compositor to use the NVIDIA GPU as primary display device.

Perform this task for each user for whom remote desktop mode was enabled in *Enabling Remote Desktop Mode for the GNOME Remote Desktop*.

This task requires administrative access.

1. Get the main DRM device node of the NVIDIA GPU in the form `card<ID>`, for example, `card0` or `card1`.

```
# ls -al /dev/dri/by-path/
total 0
drwxr-xr-x 2 root root 200 Oct  3 15:22 .
drwxr-xr-x 3 root root 220 Oct  3 15:22 ..
lrwxrwxrwx 1 root root   8 Oct  3 15:22 pci-0000:07:00.0-card -> ../card0
lrwxrwxrwx 1 root root  13 Oct  3 15:22 pci-0000:07:00.0-render -> ../
↪renderD128
lrwxrwxrwx 1 root root   8 Oct  3 15:22 pci-0000:08:00.0-card -> ../card1
lrwxrwxrwx 1 root root  13 Oct  3 15:22 pci-0000:08:00.0-render -> ../
↪renderD129
```

2. In a plain text editor, create the file `/etc/udev/rules.d/61-mutter-primary-gpu.rules` and add the following line to the file:

```
ENV{DEVNAME}=="/dev/dri/<drm-device-node>", TAG+="mutter-device-preferred-
↪primary"
```

`<drm-device-node>`
The DRM device node of the NVIDIA GPU in the form `card<ID>` that you just obtained.

3. Save your changes to `/etc/udev/rules.d/61-mutter-primary-gpu.rules`.
4. Reboot the guest VM.

4.2.7.6 Configuring the Wayland Display Server Protocol for MIG-Backed vGPUs

To use the Wayland display server protocol with MIG-backed vGPUs, you must reload the `nvidia-drm` and `nvidia-modeset` kernel modules.

After installing the NVIDIA vGPU software graphics driver, perform this task from the VM to which the MIG-backed vGPUs are assigned.

This task requires administrative access.

1. Stop the GNOME Display Manager (GDM) service.

```
# sudo service gdm stop
```

2. Unload the `nvidia-drm` and `nvidia-modeset` kernel modules.

```
# sudo rmmod nvidia_drm nvidia_modeset
```

3. Load the `nvidia-drm` and `nvidia-modeset` kernel modules again.

```
# sudo modprobe nvidia_drm nvidia_modeset
```

4. Start the GDM service again.

```
# sudo service gdm start
```

Chapter 5. Modifying a VM's NVIDIA vGPU Configuration

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

5.1. Removing a VM's NVIDIA vGPU Configuration

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

5.1.1. Removing a Citrix Virtual Apps and Desktops 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.

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

5.1.1.1 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 5.1.
2. Click **OK**.

5.1.1.2 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
```

(continues on next page)

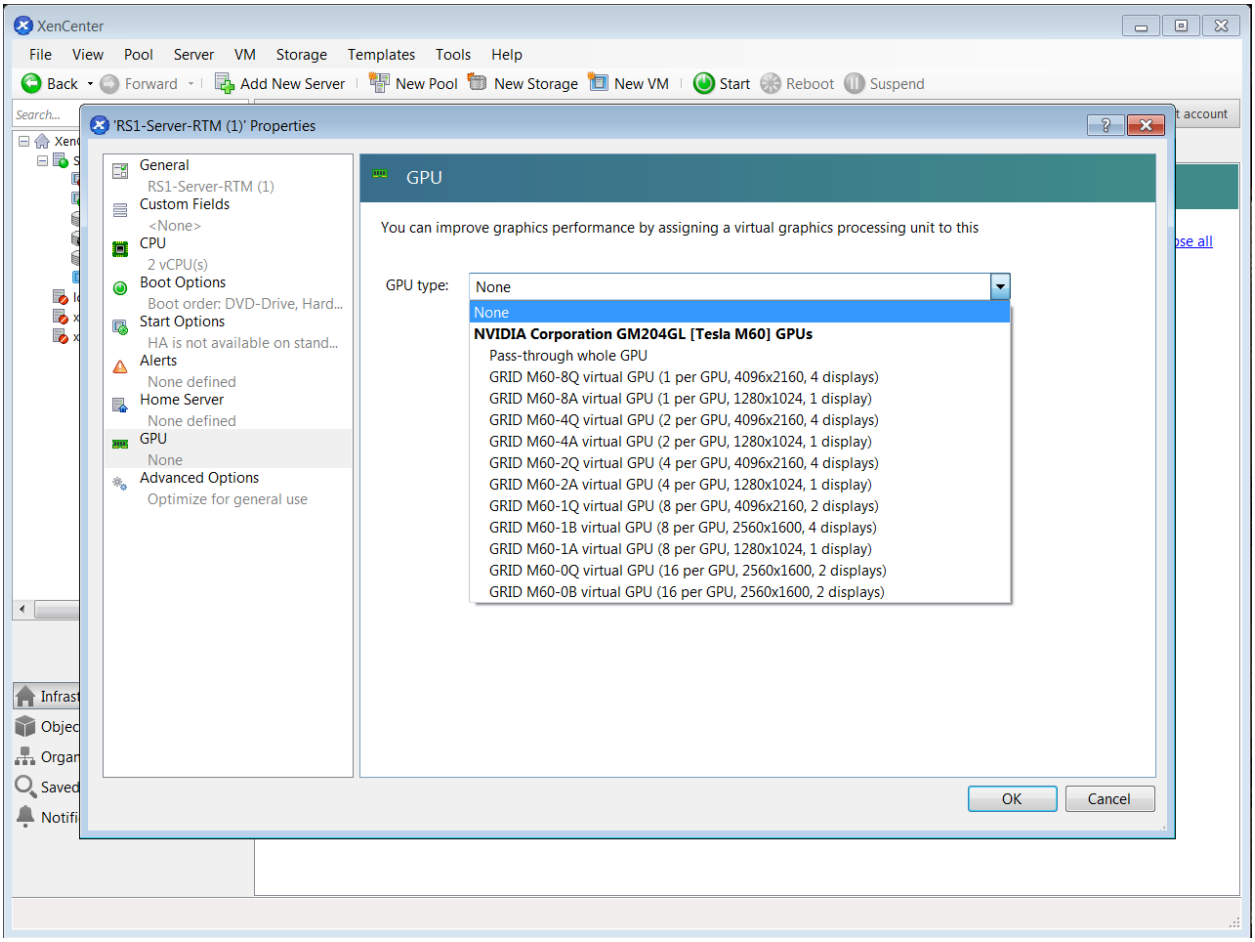


Figure 5.1: Using XenCenter to remove a vGPU configuration from a VM

(continued from previous page)

```
          vm-uuid ( RO): e71afda4-53f4-3a1b-6c92-a364a7f619c2
    gpu-group-uuid ( RO): d53526a9-3656-5c88-890b-5b24144c3d96
```

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 ~]#
```

5.1.2. 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.

5.2. Modifying GPU Allocation Policy

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

On VMware vSphere, the GPU allocation policy does not apply to MIG-backed vGPUs or to a GPU in mixed-size mode.

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.

- 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.

5.2.1. Modifying GPU Allocation Policy on XenServer

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

5.2.1.1 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 ~]#
```

5.2.1.2 Modifying GPU Allocation Policy GPU by Using XenCenter

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

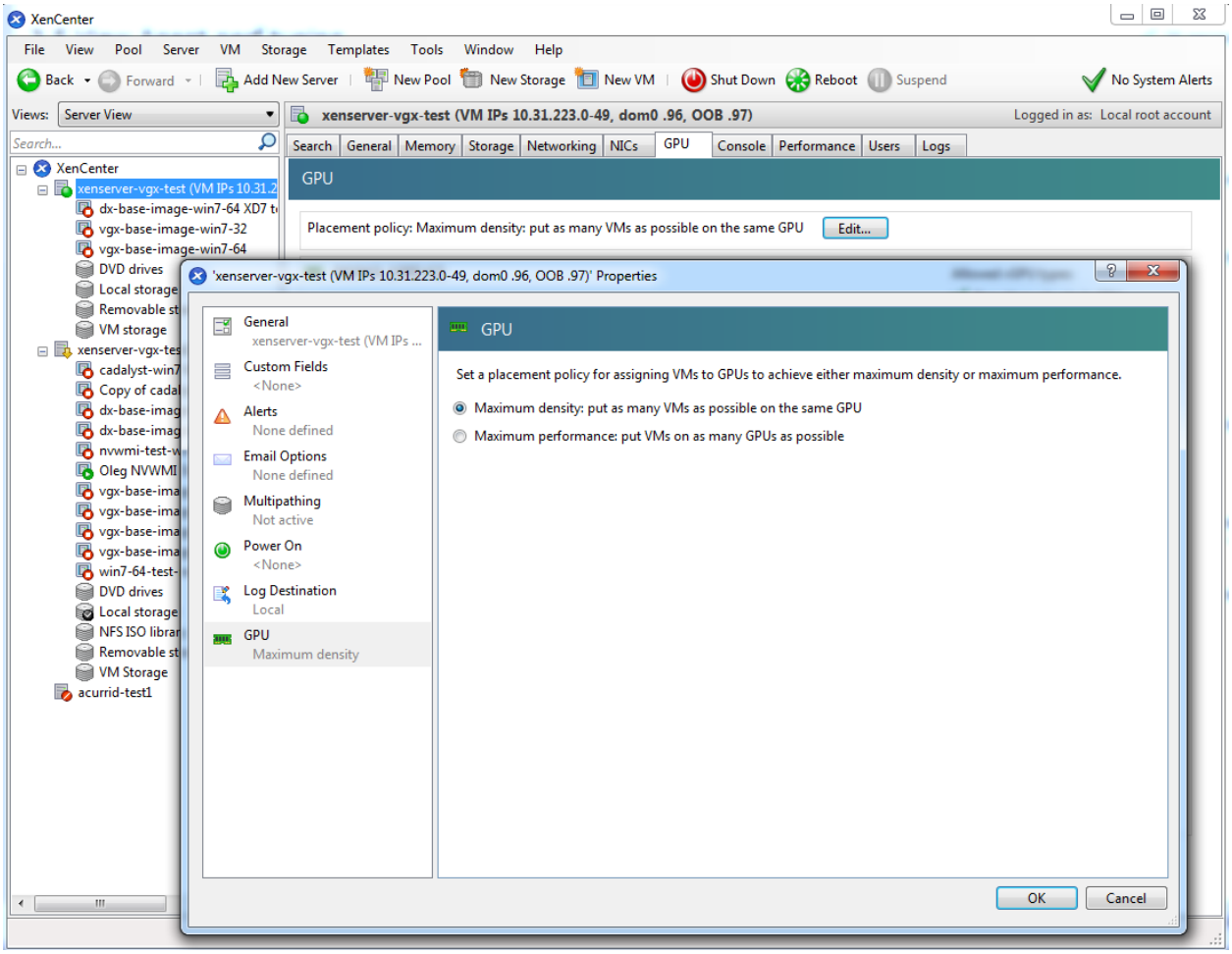


Figure 5.2: Modifying GPU placement policy in XenCenter

5.2.2. Modifying GPU Allocation Policy on VMware vSphere

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

This procedure does not apply to MIG-backed vGPUs or to a GPU in mixed-size mode.

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

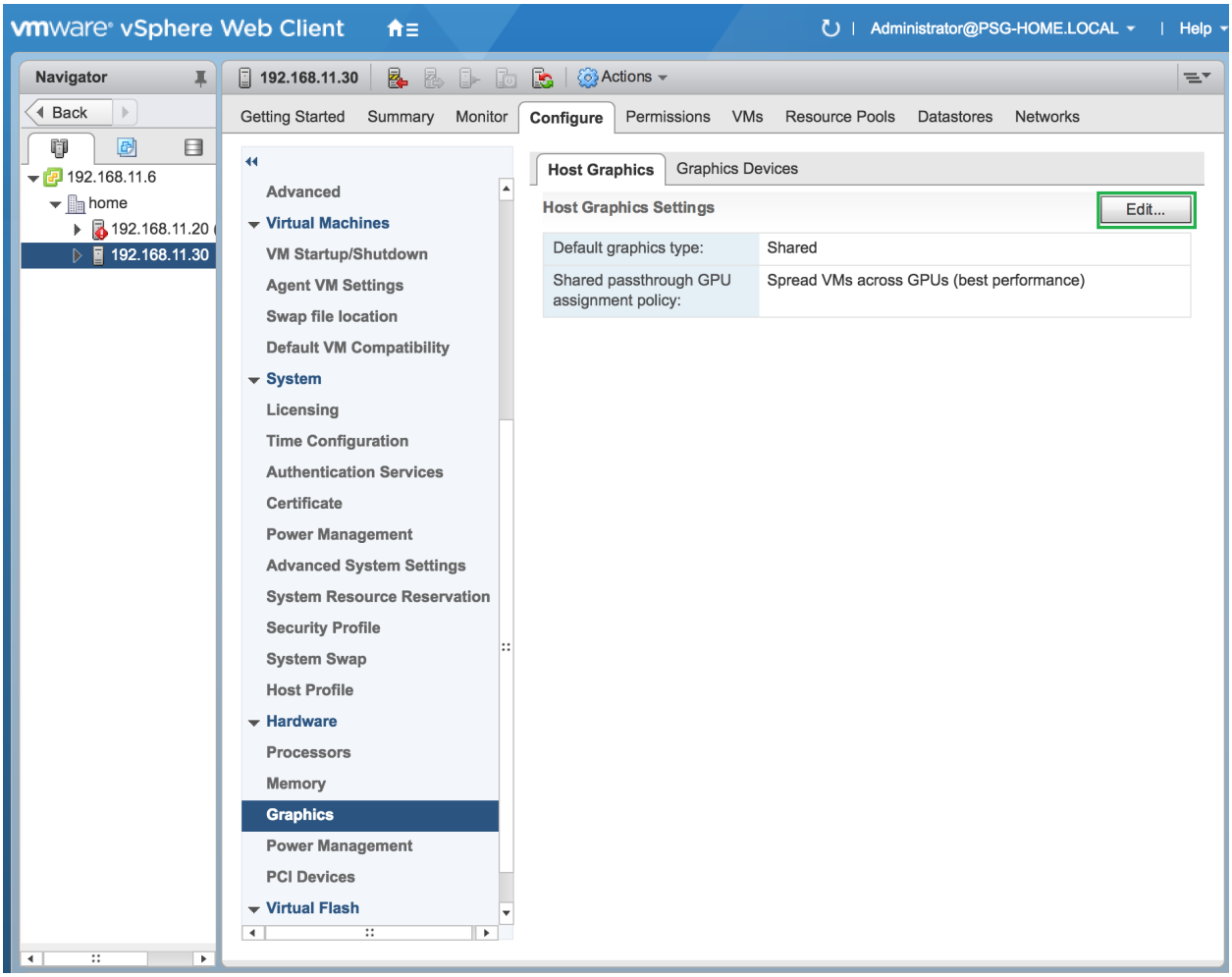


Figure 5.3: Breadth-first allocation scheme setting for vGPU-enabled VMs

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

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

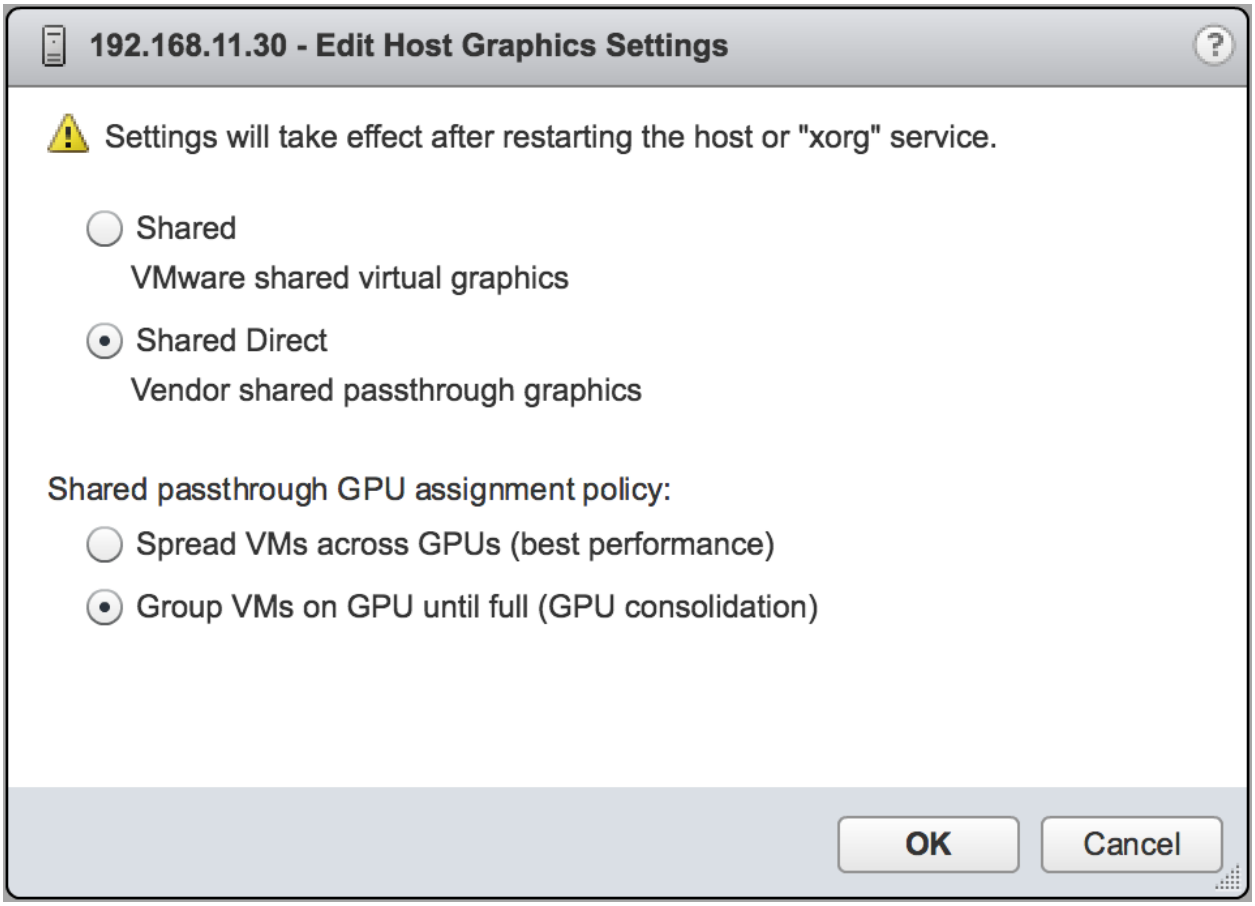


Figure 5.4: Host graphics settings for vGPU

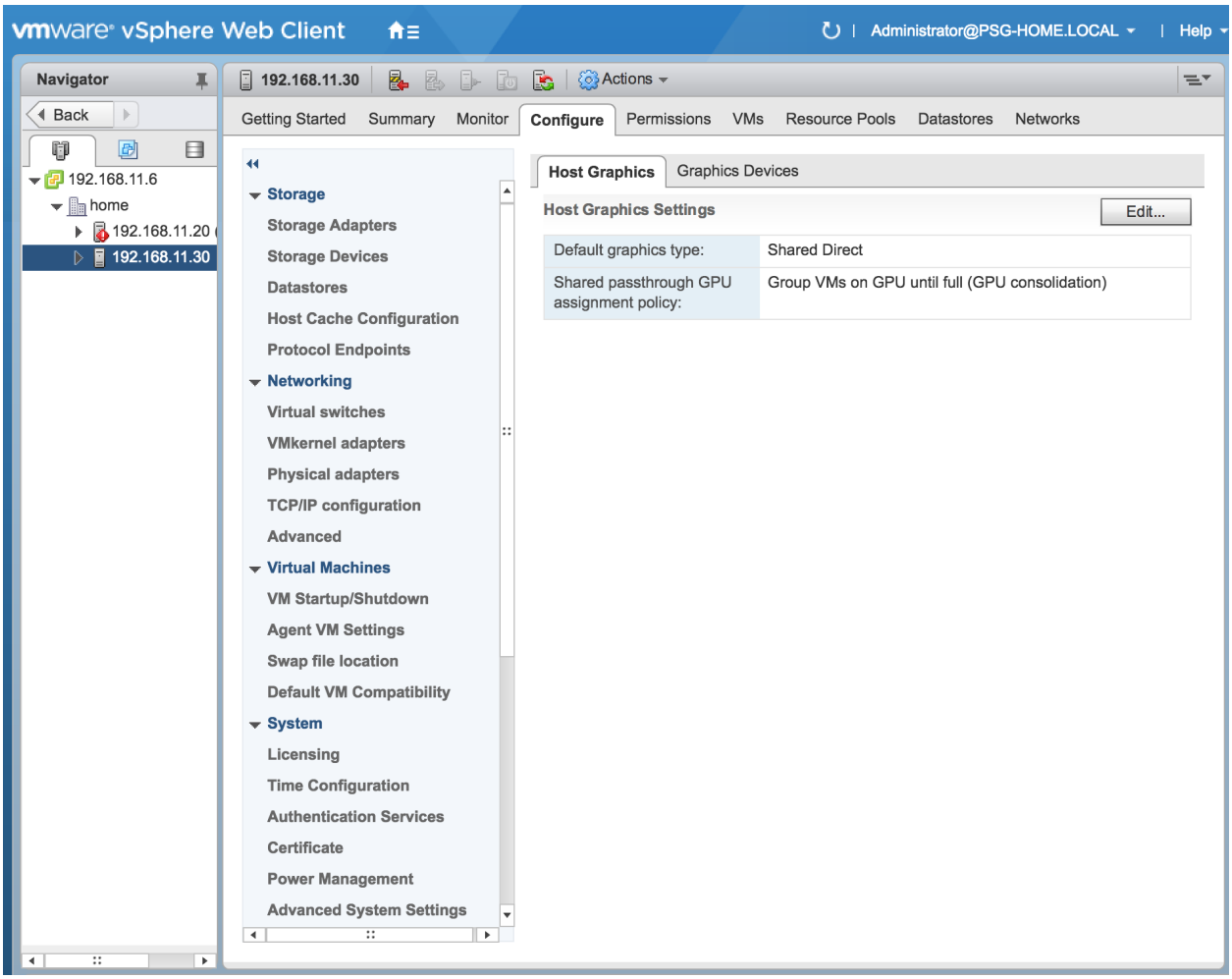


Figure 5.5: Depth-first allocation scheme setting for vGPU-enabled VMs

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

See also the following topics in the VMware vSphere documentation:

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

5.3. Migrating a VM Configured with vGPU

On some hypervisors, NVIDIA 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 destination host has a physical GPU of the same type as the GPU where the vGPU currently resides.
- ECC memory configuration (enabled or disabled) on both the source and destination hosts must be identical.
- The GPU topologies (including NVLink widths) on both the source and destination hosts must be identical.

Note

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

- Unified memory
- Debuggers
- Profilers

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

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

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

5.3.1. Migrating a VM Configured with vGPU on 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 XenServer versions, NVIDIA GPUs, and guest OS releases support XenMotion with vGPU, see Virtual GPU Software for 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.

5.3.2. Migrating a VM Configured with vGPU on a Linux with KVM Hypervisor

NVIDIA vGPU software supports vGPU Migration for VMs that are configured with vGPU. vGPU Migration 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.

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

For details about which Linux with KVM hypervisor versions, NVIDIA GPUs, and guest OS releases support vGPU Migration, refer to the following documentation:

- Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes
- Virtual GPU Software for Ubuntu Release Notes

Perform this task in a Linux command shell on the Linux with KVM hypervisor host on which the VM to be migrated is running.

Before migrating a VM configured with vGPU on a Linux with KVM hypervisor, ensure that the prerequisites listed for all supported hypervisors in *Migrating a VM Configured with vGPU* are met.

1. Set the maximum downtime of the VM to a length of time that is greater than the time required to complete the migration.

   If the VM is heavily loaded, migration might not be completed within the default maximum downtime. To ensure that migration of the VM is completed, ensure that the maxim downtime exceeds the time required to complete the migration.

   ```
   # virsh migrate-setmaxdowntime --domain <vm-name> --downtime <length>
   ```

   <vm-name>
   The name of the VM on the local host that you want to migrate.

   <length>
   The maximum downtime of the VM in milliseconds.

   This example sets the maximum downtime of the VM named `guestvm` on the local host to 10 s (10,000 ms).

```
# virsh migrate-setmaxdowntime --domain guestvm --downtime 10000
```

2. Run the `virsh migrate` command for migrating the VM.

   The command to run depends on whether the hypervisor on the local host and the hypervisor on remote host both use the same VFIO framework for an NVIDIA vGPU that supports SR-IOV.

   - If both hypervisors use the same VFIO framework, run the following command:

     ```
     # virsh migrate --live <vm-name> <destination-url> --verbose
     ```

   - If one hypervisor uses the `mdev` VFIO framework and the other hypervisor uses a vendor-specific VFIO framework, run the following command:

     ```
     # virsh migrate --live <vm-name> <destination-url> --verbose \
     --xml <remote-libvirt-xml-file>
     ```

     For GPUs that support SR-IOV, use of a vendor-specific VFIO framework is introduced in the following hypervisor software releases:

     - Red Hat Enterprise Linux with KVM 10.0
     - Ubuntu 24.04

`<vm-name>`
The name of the VM on the local host that you want to migrate.

`<destination-url>`
The URL of the connection to the remote host to which you want to migrate the VM. For example, to migrate the VM to the system connection of the remote host at IP v4 address 192.0.2.12 by using an SSH tunnel, specify `<destination-url>` as `qemu+ssh://root@192.0.2.12/system`.

`<remote-libvirt-xml-file>`
The full path to the `libvirt` XML file on the remote host, for example, `/root/destinationvm.xml`.

This example uses an SSH tunnel to migrate the VM named `guestvm` on the local host to the system connection of the remote host at IP v4 address 192.0.2.12.

```
# virsh migrate --live guestvm qemu+ssh://root@192.0.2.12/system --verbose
```

This example uses an SSH tunnel to migrate the VM named `guestvm` on the local host to the system connection of the remote host at IP v4 address 192.0.2.14. The hypervisor on the remote host uses a different VFIO framework. The full path to the `libvirt` XML file on the remote host is `/root/destinationvm.xml`.

```
# virsh migrate --live guestvm qemu+ssh://root@192.0.2.14/system --
↪verbose \
--xml /root/rhel9.4-vgpu-base_dest.xml
```

For more information, refer to Migrating virtual machines in the product documentation for Red Hat Enterprise Linux 9.

5.3.3. Suspending and Resuming a VM Configured with vGPU on a Linux with KVM Hypervisor

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

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

For details about which Linux with KVM hypervisor versions, NVIDIA GPUs, and guest OS releases support suspend and resume, refer to the following documentation:

- Virtual GPU Software for Red Hat Enterprise Linux with KVM Release Notes
- Virtual GPU Software for Ubuntu Release Notes

Perform this task in a Linux command shell on the Linux with KVM hypervisor host on which the VM to be suspended is running or on which the VM to be resumed will run.

- To suspend a VM, use the `virsh save` command to save the state of the VM to a file.

  ```
  # virsh save <vm-name> <vm-state-file>
  ```

  <vm-name>
  The name of the VM on the local host that you want to suspend.

  <vm-state-file>
  The name of the file to which you want to save the state of the VM.

  This example suspends the VM named `guestvm` on the local host to by saving its state to the file `guestvm-state.save`.

  ```
  # virsh save guestvm guestvm-state.save
  ```

- To resume a VM, use the `virsh restore` command to restore the VM from a file to which the state of the VM has previously been saved.

  ```
  # virsh restore <vm-state-file>
  ```

  <vm-state-file>
  The name of the file to which the state of the VM has previously been saved.

  This example resumes the VM named `guestvm` on the local host to by restoring its state from the file `guestvm-state.save`.

  ```
  # virsh restore guestvm-state.save
  ```

5.3.4. Migrating a VM Configured with vGPU on Microsoft Azure Local or Microsoft Windows Server

NVIDIA vGPU software supports vGPU Migration for VMs that are configured with vGPU. vGPU Migration 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 Microsoft Azure Local or Microsoft Windows Server versions, NVIDIA GPUs, and guest OS releases support vGPU Migration, refer to the following documentation:

- Virtual GPU Software for Microsoft Azure Local Release Notes
- Virtual GPU Software for Microsoft Windows Server Release Notes

Before migrating a VM configured with vGPU on Microsoft Azure Local or Microsoft Windows Server, ensure that the following prerequisites are met:

- Your hosts are correctly configured for vGPU Migration. Refer to the following articles on the Microsoft technical documentation site:
  - Set up hosts for live migration without Failover Clustering
  - Failover Clustering in Windows Server and Azure Local
- Your CPU supports IOMMU DMA bit tracking.

  For platform and configuration requirements, refer to CPU requirements on the Microsoft Learn website.
- The prerequisites listed for all supported hypervisors in *Migrating a VM Configured with vGPU* are met.

Use Hyper-V Manager or Windows PowerShell to migrate the VM as explained in Failover Clustering in Windows Server and Azure Local on the Microsoft technical documentation site.

5.3.5. 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.

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

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

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

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

The destination host must have a physical GPU of the same type as the GPU where the vGPU currently resides. Furthermore, the physical GPU must be capable of hosting the vGPU. If these requirements are not met, no available hosts are listed.

4. Select the destination network and click **Next**.
5. Select the migration priority level and click **Next**.
6. Review your selections and click **Finish**.

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

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

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

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

The window appears as follows:



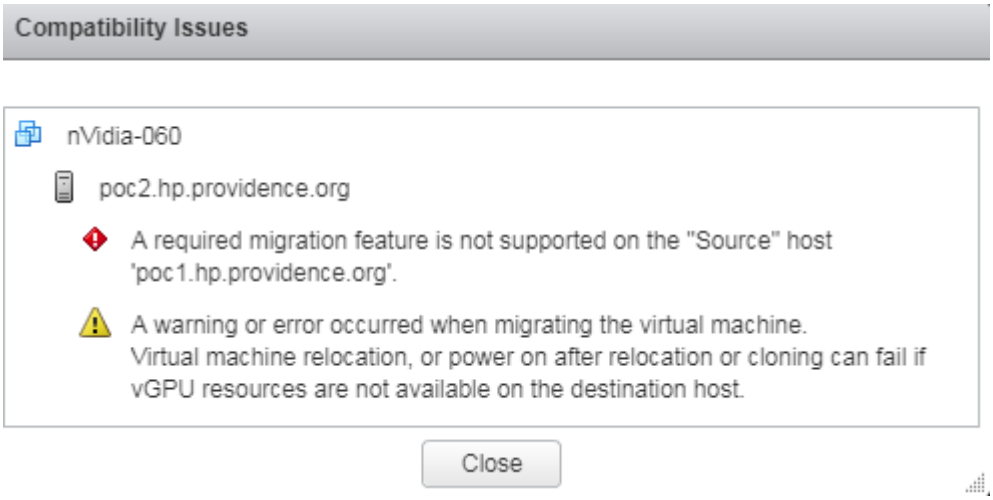
If you see this error, configure NVIDIA vGPU migration as explained in *Configuring VMware vMotion with vGPU for VMware vSphere*.

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

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

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

The window appears as follows:



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

5.3.6. 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**.

5.4. Modifying a MIG-Backed vGPU's Configuration from a Guest VM

From within a **Linux** guest VM, you can modify the configuration **only** of MIG-backed vGPUs that occupy an entire GPU instance. You **cannot** modify the configuration of time-sliced, MIG-backed vGPUs. Furthermore, from within a Windows guest VM, you **cannot** modify the configuration of any vGPUs, irrespective of whether they occupy an entire GPU instance.

> **Note**
>
> A MIG-backed vGPU that occupies an entire GPU instance is assigned all of the instance's frame buffer. For such vGPUs, the maximum vGPUs per GPU instance in the tables in *Virtual GPU Types Reference* is always 1.

Ensure that the following prerequisites are met:

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

Perform this task in a guest VM command shell.

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

   You can use secure shell (SSH) for this purpose.

2. Ensure that console virtual network computing (VNC) is disabled for the vGPU.

   To disable VNC, set the `disable_vnc` vGPU plugin parameter to 1 as explained in *Setting vGPU Plugin Parameters on a Linux with KVM Hypervisor*.

3. List the available GPU instance.

```
$ nvidia-smi mig -lgi
+-------------------------------------------------------+
| GPU instances:                                        |
| GPU   Name          Profile  Instance   Placement  |
|                       ID       ID       Start:Size |
|=======================================================|
|   0  MIG 2g.48gb+gfx  32        0          1:0     |
+-------------------------------------------------------+
```

4. Delete any instances were created when the GPU was configured for MIG-backed vGPUs.

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

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

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

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

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

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

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

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

This example shows that one `MIG 2g.48gb` compute instance or two `MIG 1c.2g.48gb` (profile ID 0) compute instances can be created within the GPU instance.

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

(continues on next page)

(continued from previous page)

```
↪------+
| Compute instance profiles:
↪       |
| GPU    GPU      Name          Profile  Instances   Exclusive
↪Shared      |
|      Instance                   ID     Free/Total     SM      DEC   ENC
↪  OFA |
|        ID                                                     CE
↪JPEG      |
|=============================================================================|
|   0      0       MIG 1c.2g.48gb       0      2/2      46      2     2
↪  0   |
|                                                               2     2
↪       |
+-----------------------------------------------------------------------------
↪------+
|   0      0       MIG 1c.2g.48gb       7      1/1      80      2     2
↪  0   |
|                                                               2     2
↪       |
+-----------------------------------------------------------------------------
↪------+
|   0      0       MIG 2g.48gb          1*     1/1      94      2     2
↪  0   |
|                                                               2     2
↪       |
+-----------------------------------------------------------------------------
↪------+
```

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

   Create each compute instance individually by running the following command.

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

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

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

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

This example creates a `MIG 2g.48gb` compute instance on GPU instance 0.

```
$ nvidia-smi mig -cci 1 -gi 0
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  0
↪using profile MIG 2g.48gb (ID  1)
```

This example creates two `MIG 1c.2g.48gb` compute instances on GPU instance 0 by running the same command twice.

```
$ nvidia-smi mig -cci 0 -gi 0
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  0
 ↪using profile MIG 1c.2g.48gb (ID  0)
$ nvidia-smi mig -cci 0 -gi 0
Successfully created compute instance ID  1 on GPU  0 GPU instance ID  0
 ↪using profile MIG 1c.2g.48gb (ID  0)
```

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

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

   This example confirms that a `MIG 2g.48gb` compute instance was created on GPU instance 0.

```
nvidia-smi
Mon May 18 19:01:24 2026
+-----------------------------------------------------------------------------
 ↪----+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08  CUDA Version:  13.2
 ↪    |
|-------------------------------+----------------------+------------------
 ↪----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
 ↪ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
 ↪Compute M. |
|                               |                      |
 ↪MIG M. |
|===============================+======================+======================|
|   0  NVIDIA RTX PRO 6000... On| 00000000:07:00.0 Off |
 ↪N/A |
| N/A   N/A    P8   N/A /   N/A |     1MiB /  49152MiB |     N/A
 ↪Default |
|                               |                      |
 ↪Enabled |
+-------------------------------+----------------------+------------------
 ↪----+

+-----------------------------------------------------------------------------
 ↪----+
| MIG devices:
 ↪    |
+------------------+----------------------+-----------+------------------
 ↪----+
| GPU  GI  CI  MIG |  Shared Memory-Usage |        Vol|         Shared
 ↪    |
|      ID  ID  Dev |    Shared BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA
 ↪ JPG|
|                  |                      |        ECC|
 ↪    |
|==================+======================+===========+======================|
|  0    0   0   0  |   1MiB / 46741MiB    | 94    N/A |  2   2    2    0
 ↪  2 |
```

(continues on next page)

(continued from previous page)

```
|                  |   0MiB /   256MiB    |           |
 ↪    |
+------------------+----------------------+-----------+-------------------
 ↪----+

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

This example confirms that two `MIG 1c.2g.48gb` compute instances were created on GPU instance 0.

```
$ nvidia-smi
Mon May 18 19:01:24 2026
+-----------------------------------------------------------------------------
 ↪----+
| NVIDIA-SMI 595.112.08   Driver Version: 595.112.08  CUDA Version:  13.2
 ↪    |
|-------------------------------+----------------------+------------------
 ↪----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
 ↪ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
 ↪Compute M. |
|                               |                      |
 ↪MIG M. |
|===============================+======================+======================|
|   0  NVIDIA RTX PRO 6000... On| 00000000:07:00.0 Off |
 ↪N/A |
| N/A   N/A    P8    N/A /  N/A |     1MiB /  49152MiB |     N/A
 ↪Default |
|                               |                      |
 ↪Enabled |
+-------------------------------+----------------------+------------------
 ↪----+

+-----------------------------------------------------------------------------
 ↪----+
| MIG devices:
 ↪    |
+------------------+----------------------+-----------+-------------------
 ↪----+
```

(continues on next page)

(continued from previous page)

```
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared
 ↪     |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA
 ↪ JPG|
|                  |                      |        ECC|
 ↪     |
|==================+======================+===========+======================|
|  0    0   0   0  |      1MiB / 46741MiB | 46    N/A |  2   2    2    0
 ↪  2 |
|                  |      0MiB /  4096MiB |           |
 ↪     |
+------------------+----------------------+-----------+-------------------
 ↪----+
|  0    0   1   1  |      1MiB / 46741MiB | 46    N/A |  2   2    2    0
 ↪  2 |
|                  |      0MiB /   256MiB |           |
 ↪     |
+------------------+----------------------+-----------+-------------------
 ↪----+

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

5.5. Enabling Unified Memory for a vGPU

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

5.5.1. Enabling Unified Memory for a vGPU on XenServer

On XenServer, enable unified memory by setting the `enable_uvm` vGPU plugin parameter.

Perform this task for each vGPU that requires unified memory by using the `xe` command.

Set the `enable_uvm` vGPU plugin parameter for the vGPU to `1` as explained in *Setting vGPU Plugin Parameters on XenServer*.

This example enables unified memory for the vGPU that has the UUID `d15083f8-5c59-7474-d0cb-fbc3f7284f1b`.

```
[root@xenserver ~] xe vgpu-param-set uuid=d15083f8-5c59-7474-d0cb-
↪fbc3f7284f1b extra_args='enable_uvm=1'
```

5.5.2. Enabling Unified Memory for a vGPU on Red Hat Enterprise Linux KVM

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

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

Perform this task for each vGPU that requires unified memory.

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

5.5.3. Enabling Unified Memory for a vGPU on VMware vSphere

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

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

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

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

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

5.6. Enabling NVIDIA CUDA Toolkit Development Tools for NVIDIA vGPU

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

5.6.1. Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU

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

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

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

5.6.1.1 Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU on XenServer

Set the `enable_debugging` vGPU plugin parameter for the vGPU that is assigned to the VM to `1` as explained in *Setting vGPU Plugin Parameters on XenServer*.

This example enables NVIDIA CUDA Toolkit debuggers for the vGPU that has the UUID `d15083f8-5c59-7474-d0cb-fbc3f7284f1b`.

```
[root@xenserver ~] xe vgpu-param-set uuid=d15083f8-5c59-7474-d0cb-
↪fbc3f7284f1b extra_args='enable_debugging=1'
```

The setting of this parameter is preserved after a guest VM is restarted and after the hypervisor host is restarted.

5.6.1.2 Enabling NVIDIA CUDA Toolkit Debuggers for NVIDIA vGPU on Red Hat Enterprise Linux KVM

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

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

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

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

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

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

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

5.6.2. Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU

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

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

5.6.2.1 Supported NVIDIA CUDA Toolkit Profiler Features

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

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

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

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

Clocks are locked for multipass profiling such as:

- NVIDIA Nsight Compute kernel profiling
- CUPTI range profiling

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

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

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

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

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

5.6.2.4 Enabling NVIDIA CUDA Toolkit Profilers for a vGPU VM

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

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

5.6.2.4.1 Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU on XenServer

Set the `enable_profiling` vGPU plugin parameter for the vGPU that is assigned to the VM to `1` as explained in *Setting vGPU Plugin Parameters on XenServer*.

This example enables NVIDIA CUDA Toolkit profilers for the vGPU that has the UUID `d15083f8-5c59-7474-d0cb-fbc3f7284f1b`.

```
[root@xenserver ~] xe vgpu-param-set uuid=d15083f8-5c59-7474-d0cb-
↪fbc3f7284f1b extra_args='enable_profiling=1'
```

The setting of this parameter is preserved after a guest VM is restarted and after the hypervisor host is restarted.

5.6.2.4.2 Enabling NVIDIA CUDA Toolkit Profilers for NVIDIA vGPU on Red Hat Enterprise Linux KVM

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

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

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

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

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

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

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

5.7. Enabling the TCC Driver Model for a vGPU

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

This task requires administrator privileges.

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

Note

Only Q-series vGPUs support the TCC driver model.

1. Log on to the VM to which the vGPU is assigned.
2. Set the driver model of the vGPU to the TCC driver model.

```
nvidia-smi -g <vgpu-id> -dm 1
```

<vgpu-id>
The ID of the vGPU for which you want to enable the TCC driver model. If the `-g` is omitted, the TCC driver model is enabled for all vGPUs that are assigned to the VM.

3. Reboot the VM.

Chapter 6. Monitoring GPU Performance

NVIDIA 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 XenServer, you can use Citrix XenCenter.
- From a Windows guest VM, you can use these tools:
  - Windows Performance Monitor
  - Windows Management Instrumentation (WMI)

6.1. NVIDIA System Management Interface `nvidia-smi`

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

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

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

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

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

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

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

  In a Windows guest VM, `nvidia-smi` is installed in a folder that is in the default executable path. Therefore, you can run `nvidia-smi` from a command prompt from any folder by running the `nvidia-smi.exe` command.

6.2. 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 XenServer platforms, you can also use Citrix XenCenter to monitor GPU performance.

> **Note**
>
> 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.

6.2.1. 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 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.

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

6.2.1.1 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 May 15 09:26:18 2026
+-----------------------------------------------------------------------------
↪+
| NVIDIA-SMI 595.112.08             Driver Version: 595.112.08
↪|
|-------------------------------+----------------------+----------------------
↪+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
↪|
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
↪|
|===============================+======================+======================|
|   0  Tesla M60           On   | 0000:83:00.0     Off |                  Off
↪|
| N/A   31C    P8    23W / 150W |   1889MiB /  8191MiB |      7%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   1  Tesla M60           On   | 0000:84:00.0     Off |                  Off
↪|
| N/A   26C    P8    23W / 150W |    926MiB /  8191MiB |      9%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   2  Tesla M10           On   | 0000:8A:00.0     Off |                  N/A
↪|
| N/A   23C    P8    10W /  53W |   1882MiB /  8191MiB |     12%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   3  Tesla M10           On   | 0000:8B:00.0     Off |                  N/A
↪|
| N/A   26C    P8    10W /  53W |     10MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   4  Tesla M10           On   | 0000:8C:00.0     Off |                  N/A
↪|
| N/A   34C    P8    10W /  53W |     10MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
|   5  Tesla M10           On   | 0000:8D:00.0     Off |                  N/A
↪|
| N/A   32C    P8    10W /  53W |     10MiB /  8191MiB |      0%      Default
↪|
+-------------------------------+----------------------+----------------------
↪+
+-----------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪+
| 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 ~]#
```

6.2.1.2 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 May 15 09:27:06 2026
+-----------------------------------------------------------------------------
↪+
| NVIDIA-SMI 595.112.08             Driver Version: 595.112.08
↪|
|---------------------------------+------------------------------+------------
↪+
| GPU  Name                       | Bus-Id                       | GPU-Util
↪|
|      vGPU ID    Name            | VM ID    VM Name             | vGPU-Util
↪|
|=================================+==============================+============|
|   0  Tesla M60                  | 0000:83:00.0                 |   7%
↪|
|      11924      GRID M60-2Q     | 3        WinD-64 VGPU test 2 |       6%
↪|
+---------------------------------+------------------------------+------------
↪+
|   1  Tesla M60                  | 0000:84:00.0                 |   9%
↪|
|      11903      GRID M60-1B     | 1        WinDv2-64 VGPU test 3 |       8%
↪|
+---------------------------------+------------------------------+------------
↪+
|   2  Tesla M10                  | 0000:8A:00.0                 |  12%
↪|
|      11908      GRID M10-2Q     | 2        WinD-64 VGPU test 1 |      10%
↪|
+---------------------------------+------------------------------+------------
↪+
```

(continues on next page)

(continued from previous page)

```
|   3  Tesla M10                 | 0000:8B:00.0                      |   0%
 ↪|
+-------------------------------+-----------------------------------+-----------
 ↪+
|   4  Tesla M10                 | 0000:8C:00.0                      |   0%
 ↪|
+-------------------------------+-----------------------------------+-----------
 ↪+
|   5  Tesla M10                 | 0000:8D:00.0                      |   0%
 ↪|
+-------------------------------+-----------------------------------+-----------
 ↪+
[root@vgpu ~]#
```

For a MIG-enabled GPU, the `nvidia-smi vgpu` command also lists the GPU instance ID.

```
[root@vgpu ~]# nvidia-smi vgpu
Fri May 15 09:27:06 2026
+-----------------------------------------------------------------------------
 ↪+
| NVIDIA-SMI 595.112.08              Driver Version: 595.112.08
 ↪|
|-------------------------------+-----------------------------------+-----------
 ↪+
| GPU  Name                     | Bus-Id                            | GPU-Util
 ↪|
|  GI  vGPU ID    Name          | VM ID    VM Name                  | vGPU-Util
 ↪|
|===============================+===================================+============|
|  0   NVIDIA RTX PRO 6000   ...| 00000000:C7:00.0                  |  N/A
 ↪|
|   3  3251634224  NVIDIA R ... | 604d...  linux2022                |   N/A
 ↪|
+-------------------------------+-----------------------------------+-----------
 ↪+
[root@vgpu ~]#
```

6.2.1.3 Getting Physical GPU Details

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

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

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

Attached GPUs                             : 3
```

(continues on next page)

(continued from previous page)

```
GPU 00000000:C1:00.0
    Product Name                          : Tesla T4
    Product Brand                         : NVIDIA
    Product Architecture                  : Turing
    Display Mode                          : Enabled
    Display Active                        : Disabled
    Persistence Mode                      : Enabled
    vGPU Device Capability
        Fractional Multi-vGPU             : Supported
        Heterogeneous Time-Slice Profiles : Supported
        Heterogeneous Time-Slice Sizes    : Not Supported
    MIG Mode
        Current                           : N/A
        Pending                           : N/A
    Accounting Mode                       : Enabled
    Accounting Mode Buffer Size           : 4000
    Driver Model
        Current                           : N/A
        Pending                           : N/A
    Serial Number                         : 1321120031291
    GPU UUID                              : GPU-9084c1b2-624f-2267-4b66-
↪345583fbd981
    Minor Number                          : 1
    VBIOS Version                         : 90.04.38.00.03
    MultiGPU Board                        : No
    Board ID                              : 0xc100
    Board Part Number                     : 900-2G183-0000-001
    GPU Part Number                       : 1EB8-895-A1
    Module ID                             : 0
    Inforom Version
        Image Version                     : G183.0200.00.02
        OEM Object                        : 1.1
        ECC Object                        : 5.0
        Power Management Object           : N/A
    GPU Operation Mode
        Current                           : N/A
        Pending                           : N/A
    GSP Firmware Version                  : N/A
    GPU Virtualization Mode
        Virtualization Mode               : Host VGPU
        Host VGPU Mode                    : Non SR-IOV
    IBMNPU
        Relaxed Ordering Mode             : N/A
    PCI
        Bus                               : 0xC1
        Device                            : 0x00
        Domain                            : 0x0000
        Device Id                         : 0x1EB810DE
        Bus Id                            : 00000000:C1:00.0
        Sub System Id                     : 0x12A210DE
        GPU Link Info
            PCIe Generation
```

(continues on next page)

(continued from previous page)

```
                Max                       : 3
                Current                   : 1
                Device Current            : 1
                Device Max                : 3
                Host Max                  : N/A
            Link Width
                Max                       : 16x
                Current                   : 16x
        Bridge Chip
            Type                          : N/A
            Firmware                      : N/A
        Replays Since Reset               : 0
        Replay Number Rollovers           : 0
        Tx Throughput                     : 0 KB/s
        Rx Throughput                     : 0 KB/s
        Atomic Caps Inbound               : N/A
        Atomic Caps Outbound              : N/A
    Fan Speed                             : N/A
    Performance State                     : P8
    Clocks Throttle Reasons
        Idle                              : Active
        Applications Clocks Setting       : Not Active
        SW Power Cap                      : Not Active
        HW Slowdown                       : Not Active
            HW Thermal Slowdown           : Not Active
            HW Power Brake Slowdown       : Not Active
        Sync Boost                        : Not Active
        SW Thermal Slowdown               : Not Active
        Display Clock Setting             : Not Active
    FB Memory Usage
        Total                             : 15360 MiB
        Reserved                          : 0 MiB
        Used                              : 3859 MiB
        Free                              : 11500 MiB
    BAR1 Memory Usage
        Total                             : 256 MiB
        Used                              : 17 MiB
        Free                              : 239 MiB
    Compute Mode                          : Default
    Utilization
        Gpu                               : 0 %
        Memory                            : 0 %
        Encoder                           : 0 %
        Decoder                           : 0 %
    Encoder Stats
        Active Sessions                   : 0
        Average FPS                       : 0
        Average Latency                   : 0
    FBC Stats
        Active Sessions                   : 0
        Average FPS                       : 0
        Average Latency                   : 0
```

(continues on next page)

(continued from previous page)

```
    Ecc Mode
        Current                           : Enabled
        Pending                           : Enabled
    ECC Errors
        Volatile
            SRAM Correctable              : 0
            SRAM Uncorrectable            : 0
            DRAM Correctable              : 0
            DRAM Uncorrectable            : 0
        Aggregate
            SRAM Correctable              : 0
            SRAM Uncorrectable            : 0
            DRAM Correctable              : 0
            DRAM Uncorrectable            : 0
    Retired Pages
        Single Bit ECC                    : 0
        Double Bit ECC                    : 0
        Pending Page Blacklist            : No
    Remapped Rows                         : N/A
    Temperature
        GPU Current Temp                  : 35 C
        GPU Shutdown Temp                 : 96 C
        GPU Slowdown Temp                 : 93 C
        GPU Max Operating Temp            : 85 C
        GPU Target Temperature            : N/A
        Memory Current Temp               : N/A
        Memory Max Operating Temp         : N/A
    Power Readings
        Power Management                  : Supported
        Power Draw                        : 16.57 W
        Power Limit                       : 70.00 W
        Default Power Limit               : 70.00 W
        Enforced Power Limit              : 70.00 W
        Min Power Limit                   : 60.00 W
        Max Power Limit                   : 70.00 W
    Clocks
        Graphics                          : 300 MHz
        SM                                : 300 MHz
        Memory                            : 405 MHz
        Video                             : 540 MHz
    Applications Clocks
        Graphics                          : 585 MHz
        Memory                            : 5001 MHz
    Default Applications Clocks
        Graphics                          : 585 MHz
        Memory                            : 5001 MHz
    Deferred Clocks
        Memory                            : N/A
    Max Clocks
        Graphics                          : 1590 MHz
        SM                                : 1590 MHz
        Memory                            : 5001 MHz
```

(continues on next page)

(continued from previous page)

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

6.2.1.4 Getting vGPU Details

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

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

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

(continues on next page)

(continued from previous page)

```
        Utilization
            Gpu                               : 0 %
            Memory                            : 0 %
            Encoder                           : 0 %
            Decoder                           : 0 %
        Encoder Stats
            Active Sessions                   : 0
            Average FPS                       : 0
            Average Latency                   : 0
        FBC Stats
            Active Sessions                   : 0
            Average FPS                       : 0
            Average Latency                   : 0
[root@vgpu ~]#
```

For a MIG-enabled GPU, use the `–gi` or `--gpu-instance-id` option with the `–i` or `--id` option to select one or more GPU instances.

```
[root@vgpu ~]# nvidia-smi vgpu -q -gi 3 -i 0
GPU 00000000:C7:00.0
    Active vGPUs                          : 1
    vGPU ID                               : 3251634224
        VM UUID                           : 604de595-d65c-4e6f-acb5-
↪b178dec1da0d
        VM Name                           : linux2022
        vGPU Name                         : NVIDIA RTX Pro 6000 Blackwell DC-
↪1-24Q
        vGPU Type                         : 1561
        vGPU UUID                         : 40884b72-6c60-11f0-aab8-
↪2b7e4b395804
        MDEV UUID                         : 3bdbac5d-0850-457a-918d-
↪5b7ecd1c8afd
        Guest Driver Version              : N/A
        License Status                    : N/A (Expiry: N/A)
        GPU Instance ID                   : 3
        Placement ID                      : 0
        Accounting Mode                   : N/A
        ECC Mode                          : Disabled
        Accounting Buffer Size            : 4000
        Frame Rate Limit                  : N/A
        PCI
            Bus Id                        : 00000000:00:00.0
        FB Memory Usage
            Total                         : 24576 MiB
            Used                          : 0 MiB
            Free                          : 24576 MiB
        Utilization
            GPU                           : N/A
            Memory                        : N/A
            Encoder                       : N/A
            Decoder                       : N/A
            Jpeg                          : N/A
```

(continues on next page)

(continued from previous page)

```
            Ofa                               : N/A
        Encoder Stats
            Active Sessions                   : 0
            Average FPS                       : 0
            Average Latency                   : 0
        FBC Stats
            Active Sessions                   : 0
            Average FPS                       : 0
            Average Latency                   : 0
[root@vgpu ~]#
```

6.2.1.5 Monitoring vGPU engine usage

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
    1   11903     8     3     0     0
    2   11908    10     4     0     0
    3       -     -     -     -     -
    4       -     -     -     -     -
    5       -     -     -     -     -
^C[root@vgpu ~]#
```

6.2.1.6 Monitoring vGPU engine usage by applications

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

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

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

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

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

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

```
[root@vgpu ~]# nvidia-smi vgpu -p
# GPU    vGPU process        process   sm  mem  enc  dec
# Idx      Id      Id           name    %    %    %    %
    0   38127    1528        dwm.exe    0    0    0    0
    1   37408    4232  DolphinVS.exe   32   25    0    0
    1  257869    4432    FurMark.exe   16   12    0    0
    1  257969    4552    FurMark.exe   48   37    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1   37408    4232  DolphinVS.exe   16   12    0    0
    1  257911     656  DolphinVS.exe   32   24    0    0
    1  257969    4552    FurMark.exe   48   37    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1  257869    4432    FurMark.exe   38   30    0    0
    1  257911     656  DolphinVS.exe   19   14    0    0
    1  257969    4552    FurMark.exe   38   30    0    0
    0   38127    1528        dwm.exe    0    0    0    0
    1  257848    3220    Balls64.exe   16   12    0    0
    1  257869    4432    FurMark.exe   16   12    0    0
```

(continues on next page)

(continued from previous page)

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

6.2.1.7 Monitoring Encoder Sessions

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

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

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

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

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

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

```
[root@vgpu ~]# nvidia-smi vgpu -es
# GPU    vGPU Session Process   Codec       H       V Average     Average
# Idx      Id      Id      Id    Type     Res     Res     FPS Latency(us)
    1   21211       2    2308   H.264    1920    1080     424        1977
    1   21206       3    2424   H.264    1920    1080       0           0
    1   22011       1    3676   H.264    1920    1080     374        1589
```

(continues on next page)

(continued from previous page)

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

6.2.1.8 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 GPUs.

```
[root@vgpu ~]# nvidia-smi vgpu -fs
# GPU        vGPU   Session   Process   Display   Session Diff. Map  Class.
 ↪Map       Capture   Max H   Max V     H     V   Average       Average
# Idx          Id        Id        Id   Ordinal      Type     State
 ↪State          Mode     Res     Res   Res   Res       FPS   Latency(us)
    0           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    1  3251634178         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    2           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    0           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    1  3251634178         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    2           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    0           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    1  3251634178         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    2           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    0           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    1  3251634178         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    2           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    0           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0         0             0
    2           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    0           -         -         -         -         -         -
↪-              -       -       -     -     -         -           -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0         0             0
    2           -         -         -         -         -         -
```

(continues on next page)

(continued from previous page)

```
↪-             -       -       -     -     -          -             -
# GPU        vGPU   Session   Process   Display   Session Diff. Map  Class.
↪Map       Capture   Max H   Max V     H     V   Average       Average
# Idx          Id        Id        Id   Ordinal      Type     State
↪State          Mode     Res     Res   Res   Res       FPS   Latency(us)
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled       Unknown    4096    2160     0     0          0             0
    2           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    0           -         -         -         -         -          -
↪-             -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160  1600   900         25         39964
    2           -         -         -         -         -          -
```

(continues on next page)

(continued from previous page)

```
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160  1600   900        25         39964
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
# GPU        vGPU   Session   Process   Display   Session Diff. Map  Class.
↪Map       Capture   Max H   Max V     H     V   Average       Average
# Idx          Id        Id        Id   Ordinal      Type     State
↪State          Mode     Res     Res   Res   Res       FPS   Latency(us)
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -              -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160  1600   900        135          7400
    2           -         -         -         -          -         -
```

(continues on next page)

(continued from previous page)

```
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160  1600   900       227          4403
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160  1600   900       227          4403
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0         0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0         0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
# GPU        vGPU   Session   Process   Display   Session Diff. Map  Class.
 ↪Map       Capture   Max H   Max V     H     V   Average       Average
# Idx          Id        Id        Id   Ordinal      Type     State
 ↪State          Mode     Res     Res   Res   Res       FPS   Latency(us)
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0         0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0         0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0         0             0
    2           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    0           -         -         -         -          -         -
↪-              -       -       -     -     -          -             -
    1  3251634178         1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0         0             0
    2           -         -         -         -          -         -
```

(continues on next page)

(continued from previous page)

```
↪-              -       -       -     -     -          -              -
    0           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    1  3251634178        1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0              0
    2           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    0           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    1  3251634178        1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0              0
    2           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    0           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    1  3251634178        1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0              0
    2           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    0           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    1  3251634178        1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0              0
    2           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    0           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    1  3251634178        1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0              0
    2           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    0           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
    1  3251634178        1      3984         0     ToSys  Disabled
↪Disabled      Blocking    4096    2160     0     0          0              0
    2           -       -       -       -       -          -
↪-              -       -       -     -     -          -              -
^C[root@vgpu ~]#
```

6.2.1.9 Monitoring MIG-backed vGPU activity

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

The `--gpm-metrics` option is supported only on MIG-backed vGPUs that are allocated all of the GPU instance's frame buffer.

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

Statistic	ID	Column
Graphics activity	1	gract
Streaming multiprocessor (SM) activity	2	smutil
SM occupancy	3	smocc
Integer activity	4	intutil
Tensor activity	5	mmaact
Double-precision fused multiply-add (DFMA) tensor activity	6	dfmat
Half matrix multiplication and accumulation (HMMA) tensor activity	7	hmmat
Integer matrix multiplication and accumulation (IMMA) tensor activity	9	immat
Dynamic random-access memory (DRAM) activity	10	dram
Double-precision 64-bit floating-point (FP64) activity	11	fp64
Single-precision 32-bit floating-point (FP32) activity	12	fp32
Half-precision 16-bit FP16 activity	13	fp16
NVDEC 0-7 Activity	30-37	nvdec0-nvdec7
NVJPG 0-7 Activity	40-47	nvjpg0-nvjpg7
NVOFA 0-1 Activity	50-51	nvofa0-nvofa01
HOSTMEM CACHE HIT	160	hmhit
HOSTMEM CACHE MISS	161	hmmiss
PEERMEM CACHE HIT	162	pmhit
PEERMEM CACHE MISS	163	pmmiss
DRAM CACHE HIT	164	dmhit
DRAM CACHE MISS	165	dmmiss
NVENC 0-3 Activity	166-169	nvenc0-nvenc3

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

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

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

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

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

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

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

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

6.2.1.10 Listing Supported vGPU Types

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

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

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

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

```
[root@vgpu ~]# nvidia-smi vgpu -s -i 0 -v   | less
GPU 00000000:40:00.0
    vGPU Type ID                            : 0xc
        Name                                : GRID M60-0Q
        Class                               : Quadro
        GPU Instance Profile ID             : N/A
        Max Instances                       : 16
        Max Instances Per VM                : 1
        Multi vGPU Exclusive                : False
        vGPU Exclusive Type                 : False
        vGPU Exclusive Size                 : False
        Device ID                           : 0x13f210de
        Sub System ID                       : 0x13f2114c
        FB Memory                           : 512 MiB
        Display Heads                       : 2
        Maximum X Resolution                : 2560
        Maximum Y Resolution                : 1600
        Frame Rate Limit                    : 60 FPS
        GRID License                        : Quadro-Virtual-DWS,5.0;GRID-
↪Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0
    vGPU Type ID                            : 0xf
        Name                                : GRID M60-1Q
        Class                               : Quadro
        GPU Instance Profile ID             : N/A
        Max Instances                       : 8
        Max Instances Per VM                : 1
        Multi vGPU Exclusive                : False
        vGPU Exclusive Type                 : False
        vGPU Exclusive Size                 : False
        Device ID                           : 0x13f210de
        Sub System ID                       : 0x13f2114d
        FB Memory                           : 1024 MiB
        Display Heads                       : 4
        Maximum X Resolution                : 5120
        Maximum Y Resolution                : 2880
        Frame Rate Limit                    : 60 FPS
        GRID License                        : Quadro-Virtual-DWS,5.0;GRID-
↪Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0
    vGPU Type ID                            : 0x12
        Name                                : GRID M60-2Q
        Class                               : Quadro
        GPU Instance Profile ID             : N/A
        Max Instances                       : 4
        Max Instances Per VM                : 1
        Multi vGPU Exclusive                : False
        vGPU Exclusive Type                 : False
        vGPU Exclusive Size                 : False
…
[root@vgpu ~]#
```

6.2.1.11 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 reflects the number and type of vGPUs that are already running on the GPU.

- If no vGPUs are running on the GPU, all vGPU types that the GPU supports are listed.
- If one or more vGPUs are running on the GPU, but the GPU is not fully loaded, only the type of the vGPUs that are already running is listed.
- If the GPU is fully loaded, no vGPU types are listed.

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

```
[root@vgpu ~]# nvidia-smi vgpu -c -i 0
GPU 0000:83:00.0
    GRID M60-2Q
[root@vgpu ~]#
```

For a MIG-enabled GPU, use the `–gi` or `--gpu-instance-id` option with the `–i` or `--id` option to select one or more GPU instances.

```
[root@vgpu ~]# nvidia-smi vgpu -c -gi 3 -i 0
GPU 00000000:C7:00.0
    GPU Instance ID: 3
      NVIDIA RTX Pro 6000 Blackwell DC-1-2Q
      ...
```

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

6.2.2. Using Citrix XenCenter to monitor GPU performance

If you are using 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.

6.3. Monitoring GPU Performance from a Guest VM

You can use monitoring tools within an individual guest VM to monitor the performance of vGPUs or pass-through GPUs that are assigned to the VM. The scope of these tools is limited to the guest VM

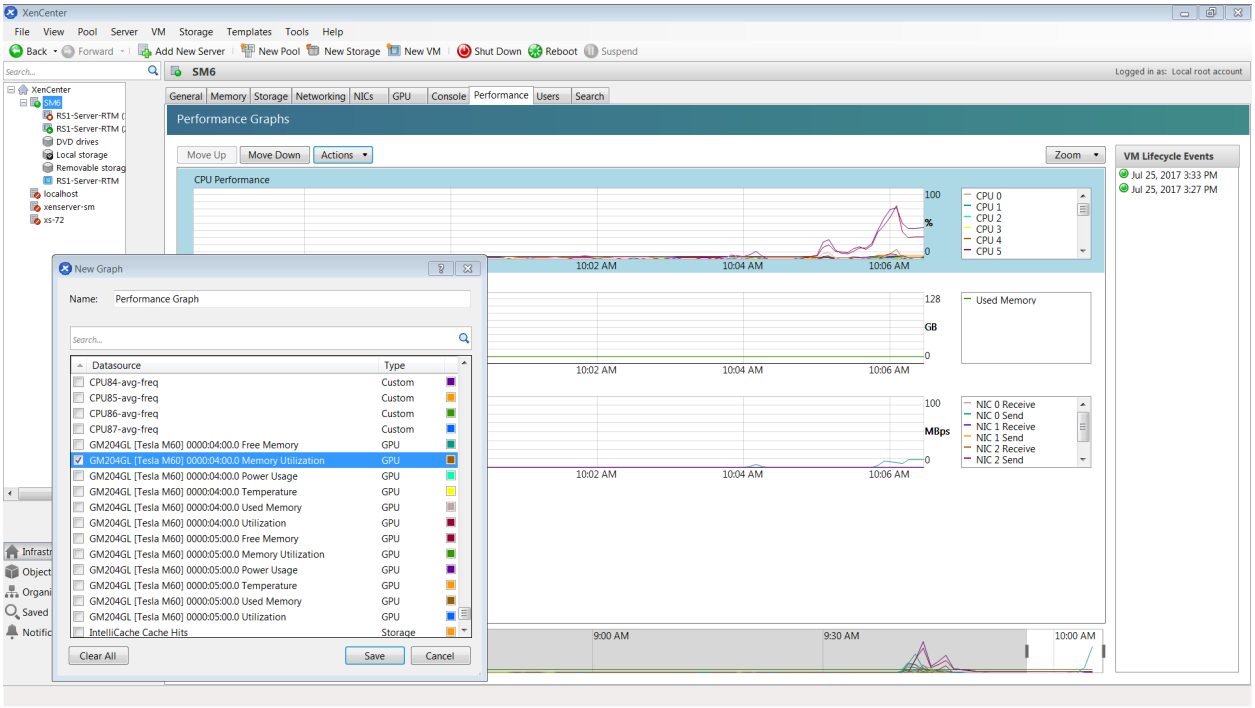


Figure 6.1: Using Citrix XenCenter to monitor GPU performance

within which you use them. You cannot use monitoring tools within an individual guest VM to monitor any other GPUs in the platform.

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

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

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

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

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

- GPU
- Video encoder
- Video decoder
- Frame buffer

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

```
nvidia-smi dmon
```

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

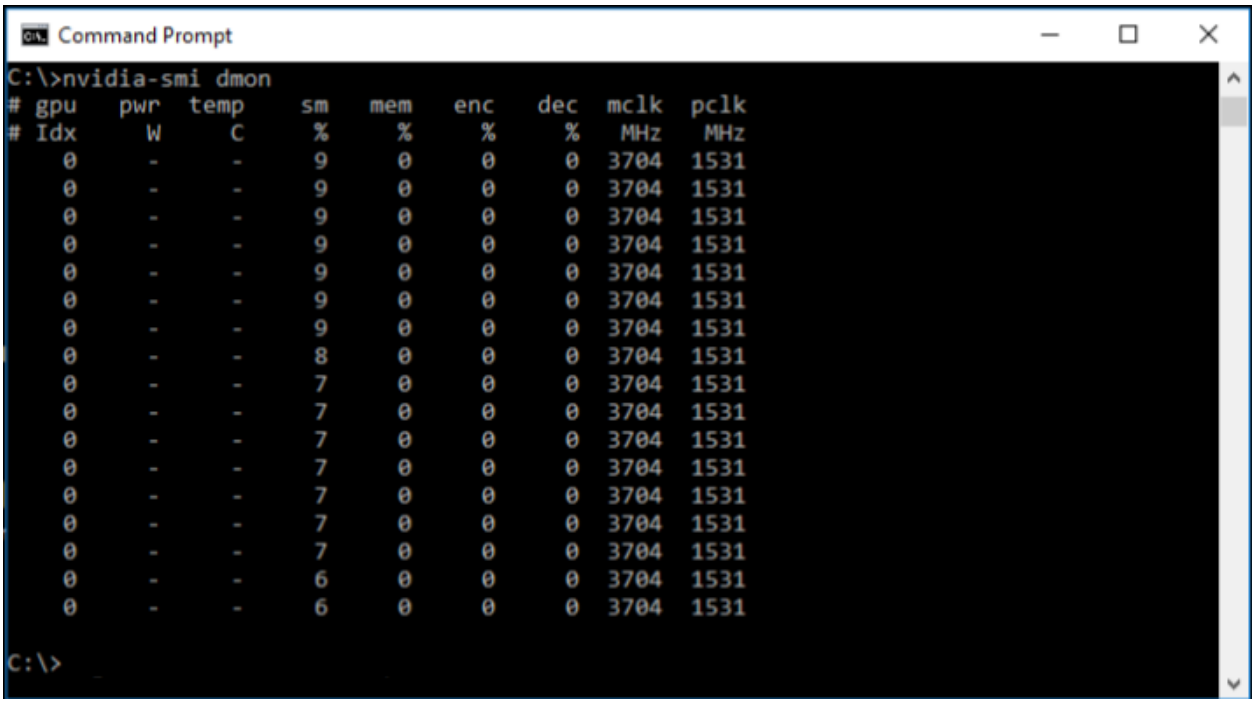


Figure 6.2: Using `nvidia-smi` from a Windows guest VM to get total resource usage by all applications

To use `nvidia-smi` to monitor MIG-backed vGPU activity from a VM, run `nvidia-smi dmon` with the following options:

- `--gpm-metrics <ID-list>`

  `<ID-list>`
  A comma-separated list of integer IDs that specify the statistics to monitor as described in *Monitoring MIG-backed vGPU activity*.

  Note

  The `--gpm-metrics` option is **not** supported on Windows.

- `--gpm-options m`

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

```
nvidia-smi pmon
```

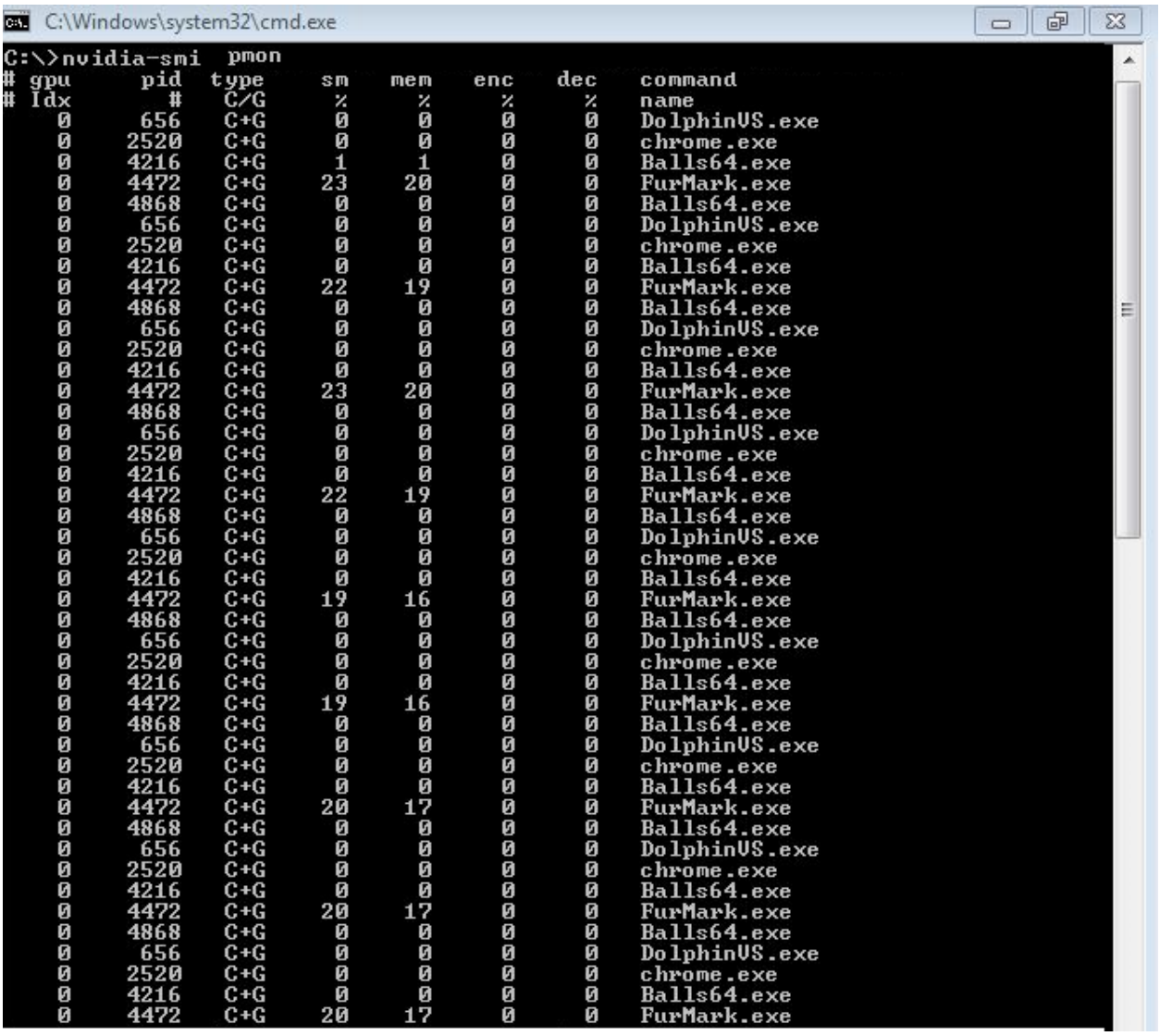


Figure 6.3: Using `nvidia-smi` from a Windows guest VM to get resource usage by individual applications

6.3.2. 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.

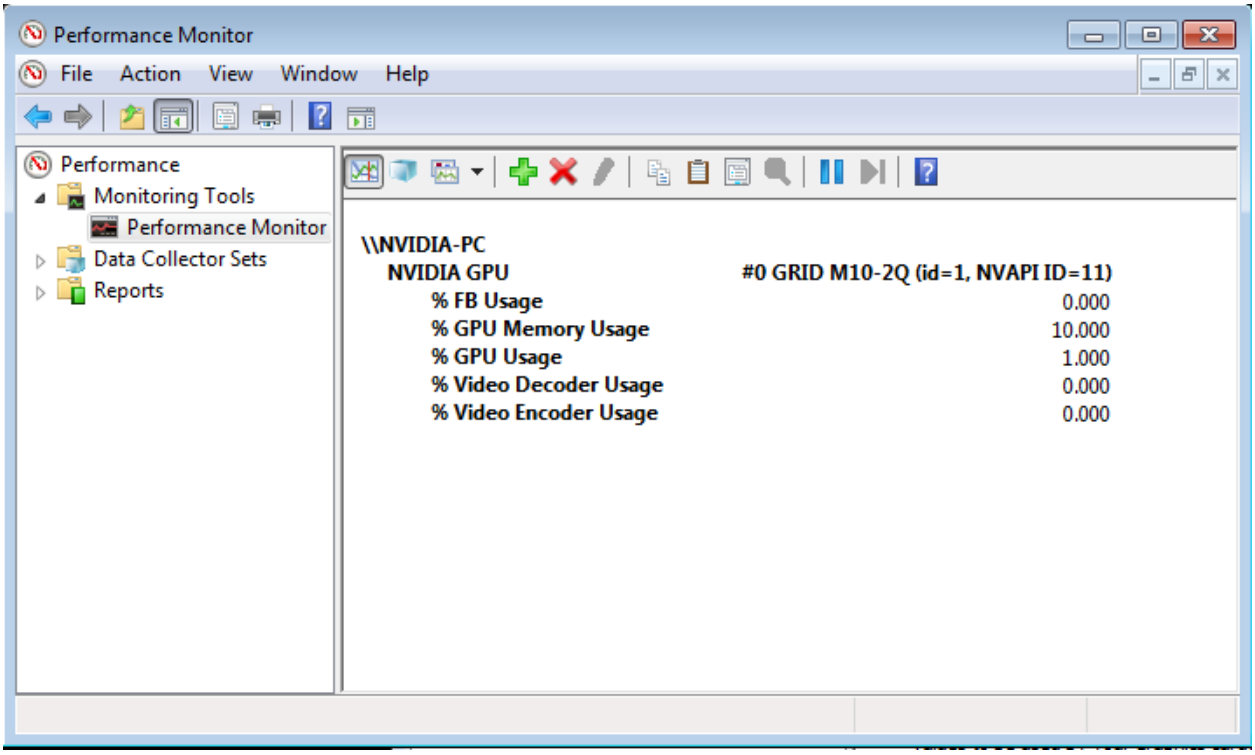


Figure 6.4: Using Windows Performance Monitor to monitor GPU performance

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

6.3.3. 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. The NVWMI API Reference in Windows Help format is available for download from the NVIDIA website.

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 from the WMI Explorer on GitHub page.

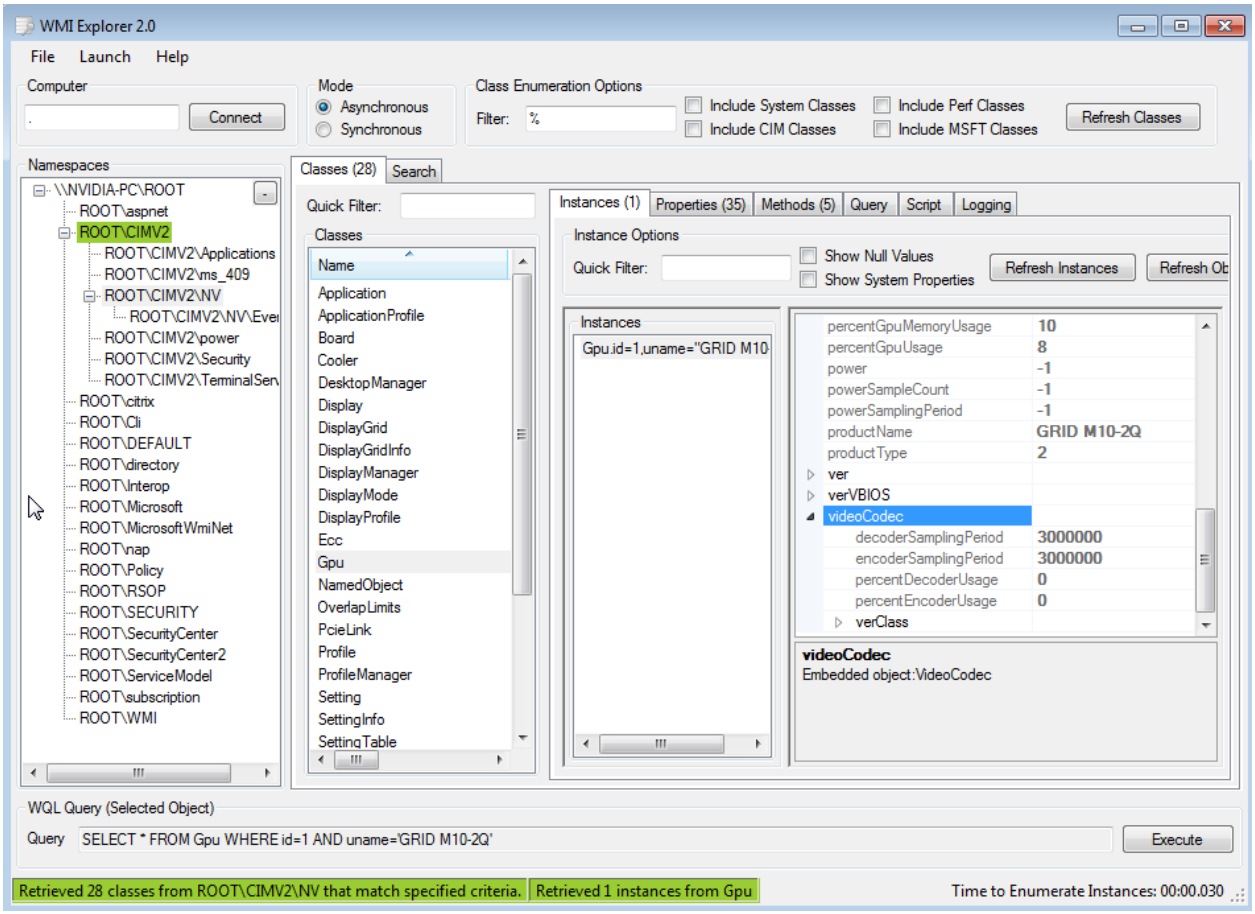


Figure 6.5: Using WMI Explorer to monitor GPU performance

On vGPUs, some instance properties of the following classes do not apply to vGPUs:

- `Gpu`
- `PcieLink`

6.3.3.1 Gpu instance properties that do not apply to vGPUs

Gpu Instance Property	Value reported on vGPU
gpuCoreClockCurrent	-1
memoryClockCurrent	-1
pciDownstreamWidth	0
pcieGpu.curGen	0
pcieGpu.curSpeed	0
pcieGpu.curWidth	0
pcieGpu.maxGen	1
pcieGpu.maxSpeed	2500
pcieGpu.maxWidth	0
power	-1
powerSampleCount	-1
powerSamplingPeriod	-1
verVBIOS.orderedValue	0
verVBIOS.strValue	—
verVBIOS.value	0

6.3.3.2 PcieLink instance properties that do not apply to vGPUs

No instances of PcieLink are reported for vGPU.

Chapter 7. Changing Scheduling Behavior for Time-Sliced vGPUs and MIG-Backed, Time-Sliced vGPUs

All GPUs that support NVIDIA vGPU software implement several vGPU schedulers, which control how GPU processing cycles are shared among vGPUs that reside on the same physical GPU.

On a single-instance GPU, you can change the scheduling behavior for all time-sliced vGPUs on the GPU. On a GPU that supports the Multi-Instance GPU (MIG) feature and is partitioned into multiple GPU instances, you can change the scheduling behavior for MIG-backed, time-sliced vGPUs within each GPU instance. The scheduling behavior for each GPU instance is set independently of the behavior for other GPU instances on the same GPU. The scheduling behavior that is set for a GPU instance applies to all the vGPUs that reside on the GPU instance

7.1. Scheduling Policies for Time-Sliced vGPUs

All GPUs that support NVIDIA vGPU software implement the best effort, equal share, and fixed share vGPU schedulers. The best effort scheduler is the default scheduler for all supported GPU architectures.

Best effort scheduler
: The physical GPU's processing cycles are shared in a way 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 might adversely affect the performance of graphics-light applications running in other VMs. To address this issue, the **equal share** and **fixed share** schedulers impose a limit on GPU processing cycles used by a vGPU.

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 might 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. 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.

The size of the share given to a vGPU depends on whether the GPU is in equal-size or mixed-size mode.

- In **equal-size mode**, the size of the share given to a vGPU is inversely proportional to the maximum number of vGPUs of its type allowed per physical GPU.

For example, a maximum of three A10-8Q vGPUs are allowed on a physical GPU. Therefore, an A10-8Q vGPU is given one third (33%) of the physical GPU's processing cycles.

- In **mixed-size mode**, the size of the share given to a vGPU is proportional to the percentage of nominal GPU frame buffer memory allocated to the vGPU, which in turn depends on the vGPU type.

For example, in a configuration with an A10-8Q and an A10-4Q on the same GPU, the physical GPU's processing cycles are shared as follows:

- The A10-8Q vGPU is given one half (50%) of the physical GPU's processing cycles.
- The A10-4Q vGPU is given one quarter (25%) of the physical GPU's processing cycles.

Note

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

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

Note

Unlike in earlier NVIDIA vGPU software releases, strict round robin policy **cannot** be disabled.

For schedulers that impose a strict round-robin scheduling policy, the adjustment to the time slice is based on the **scheduling frequency** and an **averaging factor**.

Scheduling frequency

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

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

Averaging factor

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

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

To calculate the amount by which to shorten the next time slice for a vGPU VM, the scheduler maintains a running total of the accrued overshoot time for each vGPU VM. This amount is equal to the running total divided by the averaging factor that you specify. The calculated amount is also subtracted from the accrued overshoot time. A high value for the averaging factor enforces the scheduling frequency less strictly by spreading the compensation for the accrued overshoot time over a longer period.

7.2. Scheduler Time Slice for Time-Sliced vGPUs

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

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

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

7.3. Getting Information about the Scheduling Behavior of Time-Sliced vGPUs

On all supported hypervisors, the `nvidia-smi` command provides options for getting detailed information about the scheduling behavior of time-sliced vGPUs. On Linux-based hypervisors, you can also use the hypervisor's `dmesg` command to get the current time-sliced vGPU scheduling policy for all GPUs.

7.3.1. Getting Time-Sliced vGPU Scheduler Capabilities

The scheduler capabilities of a time-sliced vGPU are a set of values that define how you can configure the vGPU to allocate the work for each VM that is configured with NVIDIA vGPU. These capability values depend on the vGPU engine type and, for vGPUs that support multiple scheduling policies, whether the vGPU supports and enforces a strict round-robin scheduling policy.

- If the vGPU engine type is graphics, the vGPU scheduler capability values consist of the supported scheduling policies and other values that affect how the work for each VM that is configured with NVIDIA vGPU is allocated. The capability values that are applicable depend on whether the vGPU supports and enforces a strict round-robin scheduling policy.
  - If the vGPU supports and enforces a strict round-robin scheduling policy, the values for the scheduling frequency and averaging factor are applicable.

- Otherwise, the values for the supported time slice range applicable.

- If the vGPU engine type is any type **other than** graphics, the only vGPU scheduler capability value indicates support for the best effort scheduling policy. All other capability values are zero.

To get the scheduler capabilities of all time-sliced vGPUs on the platform, run `nvidia-smi vgpu` with the `–sc` or `--schedulercaps` option.

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

```
[root@vgpu ~]# nvidia-smi vgpu -sc
vGPU scheduler capabilities
        Supported Policies              : Best Effort
                                          Equal Share
                                          Fixed Share
        ARR Mode                        : Supported
        Supported Timeslice Range
                Maximum Timeslice       : 30000000 ns
                Minimum Timeslice       : 1000000 ns
        Supported Scheduling Frequency
                Maximum Frequency       : 960
                Minimum Frequency       : 63
        Supported ARR Averaging Factor
                Maximum Avg Factor      : 60
                Minimum Avg Factor      : 1
```

7.3.2. Getting Time-Sliced vGPU Scheduler State Information

The scheduler state information for a time-sliced vGPU consists of the scheduling policy set for the vGPU and the values of properties that control how the work for the VM that is configured with the vGPU is allocated. The properties available depend on the scheduling policy that is set for the vGPU.

The scheduler state information that can be retrieved for a vGPU depends on whether the VM that is configured with the vGPU is running.

To get scheduler state information for all time-sliced vGPUs on the platform, run `nvidia-smi vgpu` with the `–ss` or `--schedulerstate` option.

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

For a MIG-enabled GPU, you must use the `–gi` or `--gpu-instance-id` option with the `–i` or `--id` option to select one or more GPU instances. If you use the `–i` or `--id` option without the `–gi` or `--gpu-instance-id` option, the `nvidia-smi` displays the following error message:

```
vGPU scheduler state not supported on device.
```

The following examples show the scheduler state information that is retrieved for a vGPU when the VM that is configured with the vGPU is not running and is running. In these examples, the scheduling policy is equal share scheduler, which imposes a strict round-robin scheduling policy, with the default scheduling frequency and averaging factor.

7.3.2.1 vGPU Scheduler State Information for a VM that Is Not Running

> **Note**
>
> For a VM that is not running, `Average Factor` and `Time Slice` are not listed.

```
[root@vgpu ~]# nvidia-smi vgpu -ss
GPU 00000000:65:00.0
    Active vGPUs                          : 0
    Scheduler Policy                      : Equal Share
```

7.3.2.2 vGPU Scheduler State Information for a Running VM

```
[root@vgpu ~]# nvidia-smi vgpu -ss
GPU 00000000:65:00.0
    Active vGPUs                          : 1
    Scheduler Policy                      : Equal Share
    Average Factor                        : 33
    Time Slice(ns)                        : 2083333
```

7.3.2.3 vGPU Scheduler State Information for a GPU Instance on a VM that Is Not Running

In this example, the scheduling policy is equal share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor.

```
$ nvidia-smi vgpu -ss -gi 1 -i 0
GPU 00000000:C7:00.0
    GPU Instance ID                       : 1
    Active vGPUs                          : 0
    Scheduler Policy                      : Equal Share
```

7.3.2.4 vGPU Scheduler State Information for a GPU Instance on a Running VM

```
$ nvidia-smi vgpu -ss -gi 1 -i 0
GPU 00000000:C7:00.0
    GPU Instance ID                       : 1
    Active vGPUs                          : 1
    Scheduler Policy                      : Equal Share
    Average Factor                        : 33
    Time Slice(ns)                        : 2083333
```

7.3.3. Getting Time-Sliced vGPU Scheduler Work Logs

The scheduler work logs for a time-sliced vGPU provide information about the allocation at runtime of the work for the VM that is configured with the vGPU.

The information in the scheduler work logs that can be retrieved for a vGPU depends on whether the VM that is configured with the vGPU is running.

To get scheduler work logs for all time-sliced vGPUs on the platform, run `nvidia-smi vgpu` with the `-sl` or `--schedulerlogs` option.

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

For a MIG-enabled GPU, you must use the `-gi` or `--gpu-instance-id` option with the `-i` or `--id` option to select one or more GPU instances. If you use the `-i` or `--id` option without the `-gi` or `--gpu-instance-id` option, the `nvidia-smi` displays the following error message:

```
Not supported on the device
```

To get the scheduler work logs in comma-separated values (CSV) format, run `nvidia-smi vGPU --query-gpu-instance-vgpu-scheduler-logs` with its options.

7.3.3.1 vGPU Scheduler Work Logs for a VM that Is Not Running

```
[root@vgpu ~]#  nvidia-smi vgpu -sl
+-----------------------------------------------------------------------------
↪----------------------------+
Engine Id          1
Scheduler Policy   Equal Share
GPU at deviceIndex 0 has no active VM runlist.
+-----------------------------------------------------------------------------
↪----------------------------+
```

7.3.3.2 vGPU Scheduler Work Logs for a Running VM with a GPU in Equal-Size Mode

```
[root@vgpu ~]# nvidia-smi vgpu -sl
+-----------------------------------------------------------------------------
↪-----------------------------------+
GPU Id             0
Engine Id          1
Scheduler Policy   Equal Share
Avg Factor         33
Time Slice         2083333
+-----------------------------------------------------------------------------
↪-----------------------------------+
 GPU  SW Runlist      Weight                Time          Cumulative
↪Prev Timeslice         Target Time          Cumulative
 Idx          Id        (0x)               Stamp            Run Time
↪   Runtime               Slice        Preempt Time
   0           0     5555555 1770095639091125856         79042283264
↪   2085184             1999627              2764156
   0           1     5555555 1770095639093203904         79039958880
↪   2078048             1998198              2804176
   0           2     5555555 1770095639095287040         79038201344
↪   2083136             1998560              2797323
   0           0     5555555 1770095639097371008         79044367232
↪   2083968             1999571              2764791
   0           1     5555555 1770095639099454656         79042042528
↪   2083648             1998358              2804491
   0           2     5555555 1770095639101535744         79040282432
↪   2081088             1998566              2795078
   0           0     5555555 1770095639103621696         79046453184
↪   2085952             1999552              2767410
```

7.3.3.3 vGPU Scheduler Work Logs for a Running VM with a GPU in Mixed-Size Mode

```
[root@vgpu ~]# nvidia-smi vgpu -sl
+-----------------------------------------------------------------------------
↪-----------------------------------+
GPU Id              0
Engine Id           1
Scheduler Policy    Fixed Share
Avg Factor          33
Time Slice          1041666
+-----------------------------------------------------------------------------
↪-----------------------------------+
 GPU  SW Runlist      Weight                Time          Cumulative
↪Prev Timeslice         Target Time          Cumulative
 Idx          Id        (0x)               Stamp             Run Time
↪    Runtime               Slice        Preempt Time
   0           2     4000000 1770095265490639840         12658108992
↪    1044928             1016752               825426
   0          33     7555556 1770095265491677824        506677753280
↪    1037984             1019889               714966
   0          33     7555556 1770095265492707232        506678782688
↪    1029408             1020001               702708
   0          33     7555556 1770095265493751456        506679826912
↪    1044224             1020372               705266
   0           2     4000000 1770095265494795328         12659152864
↪    1043872             1016654               827632
   0           1     2aaaaaa 1770095265495834656          8583022176
↪    1039328             1016386               831912
   0          33     7555556 1770095265496888416        506680880672
↪    1053760             1020295               717360
```

7.3.3.4 vGPU Scheduler Work Logs for a GPU Instance on a VM that Is Not Running

```
$ nvidia-smi vgpu -sl -gi 1 -i 0
+-----------------------------------------------------------------------------
↪----------------------------+
Engine Id           1
GPU Instance ID     1
Scheduler Policy    Equal Share
GPU at deviceIndex 0 has no active VM runlist.
+-----------------------------------------------------------------------------
↪----------------------------+
```

7.3.3.5 vGPU Scheduler Work Logs for a GPU Instance on a Running VM

```
$ nvidia-smi vgpu -sl -gi 1 -i 0
+-----------------------------------------------------------------------------
↪----------------------------+
GPU Id              0
GPU Instance ID     1
Engine Id           1
Scheduler Policy    Equal Share
```

(continues on next page)

(continued from previous page)

```
Avg Factor         33
Time Slice         2083333
+-------------------------------------------------------------------------------
↪-------------------------------------+
 GPU  SW Runlist      Weight                Time          Cumulative
↪Prev Timeslice         Target Time          Cumulative
 Idx          Id        (0x)               Stamp            Run Time
↪    Runtime               Slice        Preempt Time
   0           0     4000000 1770095769748761120          3658665120
↪    2080448              2057586               846768
   0          33     4000000 1770095769750841248        534416170560
↪    2080128              2056502               882227
   0           2     4000000 1770095769752920160          3661329728
↪    2078912              2055513               913647
   0           1     4000000 1770095769754997600          3661458880
↪    2077440              2054029               961163
   0           0     4000000 1770095769757134976          3660802496
↪    2137376              2057674               900811
   0          33     4000000 1770095769759217088        534418252672
↪    2082112              2056599               881006
   0           2     4000000 1770095769761296672          3663409312
↪    2079584              2055647               909898
```

7.3.3.6 vGPU Scheduler Work Logs in CSV Format for a GPU Instance on a VM that Is Not Running

Note

Line breaks have been added to this example to enhance readability.

```
$ nvidia-smi vgpu --query-gpu-instance-vgpu-scheduler-logs=gpu_bus_id,gi_id,
↪engine_id,scheduler_policy,
time_slice,sw_runlist_id,time_stamp,cumulative_run_time,previous_timeslice_
↪runtime,target_time_slice,
cumulative_preempt_time -gi 1 -i 0 --format=csv
gpu_bus_id, gi_id, engine_id, scheduler_policy, time_slice, sw_runlist_id,
↪time_stamp, cumulative_run_time,
previous_timeslice_runtime, target_time_slice, cumulative_preempt_time
00000000:C7:00.0, 1, 1, Equal Share, -, -, -, -, -, -, -
```

7.3.3.7 vGPU Scheduler Work Logs in CSV Format for a GPU Instance on a Running VM

Note

Line breaks have been added to this example to enhance readability.

```
$ nvidia-smi vgpu --query-gpu-instance-vgpu-scheduler-logs=gpu_bus_id,gi_id,
↪engine_id,scheduler_policy,
```

(continues on next page)

(continued from previous page)

```
time_slice,sw_runlist_id,time_stamp,cumulative_run_time,previous_timeslice_
↪runtime,target_time_slice,
cumulative_preempt_time -gi 1 -i 0 --format=csv
gpu_bus_id, gi_id, engine_id, scheduler_policy, time_slice, sw_runlist_id,
↪time_stamp, cumulative_run_time,
previous_timeslice_runtime, target_time_slice, cumulative_preempt_time
00000000:C7:00.0, 1, 1, Equal Share, 2083333, 0, 1751920964898612528,
↪481321948480, 2084800, 2047252, 1192158
00000000:C7:00.0, 1, 1, Equal Share, 2083333, 0, 1751920964900693296,
↪481324029248, 2080768, 2047207, 1189593
00000000:C7:00.0, 1, 1, Equal Share, 2083333, 0, 1751920964902778160,
↪481326114112, 2084864, 2047285, 1191124
00000000:C7:00.0, 1, 1, Equal Share, 2083333, 0, 1751920964904863024,
↪481328198976, 2084864, 2047239, 1192655
```

7.3.4. Linux-Based Hypervisors Only: Getting the Current Time-Sliced vGPU Scheduling Policy for All GPUs

On Linux-based hypervisors, you can use the `dmesg` command to get the current time-sliced vGPU scheduling policy for all GPUs. Get this information before changing the scheduling behavior of one or more GPUs to determine if you need to change it or after changing it to confirm the change.

> **Note**
>
> You **cannot** perform this task on Windows hypervisors because the `dmesg` command is available only on Linux-based hypervisors.

Perform this task in your hypervisor command shell.

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

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

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

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

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

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

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

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

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

7.4. Tools for Changing Scheduling Behavior for Time-Sliced vGPUs

To change the scheduling behavior for time-sliced vGPUs, you can use the `nvidia-smi` command or the `RmPVMRL` registry key. The tool to use depends on whether you require the changes to be applied immediately or whether you require the changes to be persistent.

- If you require the changes to be applied immediately, use the `nvidia-smi` command.

  If you use the `nvidia-smi` command, you do not need to reload the driver or reboot the hypervisor host to apply your changes. However, your changes are volatile and do not persist in the following circumstances:

  - The driver is reloaded.
  - The hypervisor host is rebooted.
  - The `sriov-manage` script is run to enable the virtual functions for the physical GPU in the `sysfs` file system.

- If you require the changes to be persistent, use the `RmPVMRL` registry key.

  However, if you use the `RmPVMRL` registry key, you must reload the driver or reboot the hypervisor host to apply your changes.

For information about how to use these tools to change the scheduling behavior for time-sliced vGPUs, refer to the following topics:

- *Changing Scheduling Behavior for Time-Sliced vGPUs by Using the nvidia-smi Command*
- *Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key*

Note

Unlike in earlier NVIDIA vGPU software releases, strict round robin policy **cannot** be disabled.

7.5. Changing Scheduling Behavior for Time-Sliced vGPUs by Using the `nvidia-smi` Command

The `nvidia-smi` command provides the `vgpu set-scheduler-state` subcommand and associated options for changing the scheduling behavior of time-sliced vGPUs. Because the `nvidia-smi` command is available on all supported hypervisors, you can use this method for Linux-style hypervisors and for Windows hypervisors.

> **Note**
>
> You **cannot** use the `nvidia-smi` command to change the scheduling behavior of MIG-backed, time-sliced vGPUs in VMware vSphere ESXi. You must use the `RmPVMRL` registry key for this purpose. Furthermore, when you use the `RmPVMRL` registry key for this purpose, you can change the scheduling behavior **only** for all GPU instances on a GPU. You **cannot** change the scheduling behavior for select GPU instances.

Ensure that no vGPUs exist on any physical GPU for which you want to change the scheduling behavior for time-sliced vGPUs. Any change that you make affects vGPUs that will be created on the physical GPU after you make the change.

If you try to change the scheduling behavior for time-sliced vGPUs on a physical GPU or GPU instance on which a vGPU already exists, the attempt to change the scheduling behavior fails.

- For a physical GPU, the `nvidia-smi` command displays an error message similar to the following example:

```
Unable to set the vGPU scheduler state on GPU "00000000:1A:00.0".
vGPU scheduler state cannot be configured, if vGPU instance is currently
 ↪active on the device.
```

- For a GPU instance, the `nvidia-smi` command displays an error message similar to the following example:

```
Unable to set the vGPU scheduler state on GI Id 1.
vGPU scheduler state cannot be configured, if vGPU instance is currently
 ↪active on the GI.
```

To change the scheduling behavior for time-sliced vGPUs, run `nvidia-smi vgpu set-scheduler-state` with its associated options.

For more information about these options, refer to *Scheduling Policies for Time-Sliced vGPUs*.

`-i <gpu-id>, --id <gpu-id>`

`<gpu-id>` is the identifier of the GPU on which you want to change the scheduling behavior of time-sliced vGPUs in one of the following formats:

- The GPU's 0-based index in the natural enumeration returned by the driver
- The GPU's universally unique identifier (UUID)
- The GPU's PCI bus ID in the form `<domain>:<bus>:<device>.<function>` in hexadecimal.

This option is not mandatory. If it is omitted, the scheduling behavior of time-sliced vGPUs for all GPUs on the platform is changed.

-gi <gpu-instance-id>, --gpu-instance-id <gpu-instance-id>
<gpu-instance-id> is the identifier of the GPU instance on which you want to change the scheduling behavior of time-sliced vGPUs.

For a MIG-enabled GPU, you must use this option with the -i or --id option to select one or more GPU instances. If you use the -i or --id option without this option, the nvidia-smi displays the following error message:

```
Unable to set the vGPU scheduler state. Not supported on device "<device-
↪id>".
```

Note

This option is not supported by VMware vSphere ESXi.

-p <S>, --policy <S>
<S> is a decimal integer in the range 1-3 that sets the scheduler to use:

- 1: Best effort scheduler (default)
- 2: Equal share scheduler
- 3: Fixed share scheduler

If <S> is **not** a decimal integer in the range 1-3, the attempt to set the scheduler to use fails and the nvidia-smi command displays the following error message:

```
Unable to set the vGPU scheduler state. Not supported
```

-asf <frequency>, --arr-sched-frequency <frequency>
<frequency> is a decimal integer in the range 63 to 960 that sets the scheduling frequency in Hz for the equal share and fixed share schedulers with a strict round-robin scheduling policy.

If <frequency> is outside the range 63 to 960, the scheduling frequency is set as follows:

- If <frequency> is not set, the scheduling frequency is set to the default scheduling frequency for the vGPU type as listed in Table 7.1.
- If <frequency> is less than 63, the scheduling frequency is raised to 63.
- If <frequency> is greater than 960, the scheduling frequency is capped at 960.

-aavg <averaging-factor>, --arr-avg-factor <averaging-factor>
<averaging-factor> is a decimal integer in the range 1 to 60 that sets the averaging factor to ensure scheduling fairness for the equal share and fixed share schedulers with a strict round-robin scheduling policy.

The number of time slices over which the compensation for the accrued overshoot time is applied depends on the value of <averaging-factor>:

- If <averaging-factor> is 1, the compensation for the accrued overshoot time is applied in a single time slice.
- If <averaging-factor> is 60, the compensation for the accrued overshoot time is spread over 60 time slices.
- If <averaging-factor> is not set, the default value of 33 is used.

- If `<averaging-factor>` is greater than 60, the number of time slices over which the compensation is applied is capped at 60.

Note

The following options from earlier NVIDIA vGPU software releases have been removed:

- `--arr-mode <R>`. Adaptive Round Robin (ARR) scheduling cannot be disabled.
- `--time-slice <time-slice-length>`. The time-slice length could be set only when ARR scheduling was disabled but ARR scheduling can no longer be disabled.

7.5.1. Setting the Scheduling Policy for a Single GPU

This example sets the scheduling policy of the GPU at PCI domain `0000` and BDF `15:00.0` to fixed share scheduler with a strict round-robin scheduling policy and with the default time slice length.

```
# nvidia-smi vgpu set-scheduler-state -i 0000:15:00.0 -p 3
```

7.5.2. Enabling a Strict Round-Robin Scheduling Policy for an Equal Share Scheduler for All GPUs

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

```
# nvidia-smi vgpu set-scheduler-state -p 2
```

7.5.3. Enabling a Strict Round-Robin Scheduling Policy for a Fixed Share Scheduler for All GPUs

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

```
# nvidia-smi vgpu set-scheduler-state -p 3
```

7.5.4. Enabling a Strict Round-Robin Scheduling Policy for an Equal Share Scheduler with Custom Properties for All GPUs

This example sets the vGPU scheduler to equal share scheduler with a strict round-robin scheduling policy, an averaging factor of 60, and a scheduling frequency of 960 Hz for all GPUs on the platform.

```
# nvidia-smi vgpu set-scheduler-state -p 2 -aavg 60 -asf 960
```

7.5.5. Enabling a Strict Round-Robin Scheduling Policy for a Fixed Share Scheduler with Custom Properties for All GPUs

This example sets the vGPU scheduler to fixed share scheduler with a strict round-robin scheduling policy, an averaging factor of 60, and a scheduling frequency of 960 Hz for all GPUs on the platform.

```
# nvidia-smi vgpu set-scheduler-state -p 3 -aavg 60 -asf 960
```

7.5.6. Restoring Default Time-Sliced vGPU Scheduler Settings

This example restores default time-sliced vGPU scheduler settings by setting the vGPU scheduler to best effort scheduler.

```
# nvidia-smi vgpu set-scheduler-state -p 1
```

7.5.7. Setting the Scheduling Policy for a Single GPU Instance

This example sets the vGPU scheduler to equal share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor for GPU instance 1 on GPU 0.

```
$ nvidia-smi vgpu set-scheduler-state -p 2 -gi 1 -i 0
```

7.5.8. Restoring Default Time-Sliced vGPU Scheduler Settings for a Single GPU Instance

This example restores default time-sliced vGPU scheduler settings for GPU instance 1 on GPU 0 by setting the vGPU scheduler to best effort scheduler.

```
$ nvidia-smi vgpu set-scheduler-state -p 1 -gi 1 -i 0
```

7.6. Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key

The RmPVMRL registry key controls the scheduling behavior for NVIDIA vGPUs by setting the scheduling policy, the averaging factor and scheduling frequency for schedulers with a strict round-robin

scheduling policy. How to use the RmPVMRL registry key for this purpose depends on whether you are using a Linux-based hypervisor or a Windows hypervisor.

For detailed instructions, refer to the following topics:

- *Linux-Based Hypervisors: Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key*
- *Windows Hypervisors: Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key*

7.6.1. Linux-Based Hypervisors: Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key

To use the RmPVMRL registry key to change the scheduling behavior of time-sliced vGPUs, use the standard interfaces of your hypervisor to set the RmPVMRL registry key value.

7.6.1.1 Linux-Based Hypervisors: Changing the Time-Sliced vGPU Scheduling Behavior for All GPUs by Using the RmPVMRL Registry Key

Note

You **cannot** perform this task on Windows hypervisors because it relies on interfaces that are available only on Linux-based hypervisors.

Perform this task in your hypervisor command shell.

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

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

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

   - On XenServer or Red Hat Enterprise Linux KVM, 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 <module> -p "NVreg_
     ↪RegistryDwords=RmPVMRL=<value>"
     ```

     - For the NVIDIA Ada Lovelace GPU architecture and later GPU architectures, `<module>` is `nvidia-gpu`.
     - For GPU architectures before NVIDIA Ada Lovelace, `<module>` is `nvidia`.

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

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

0x3c3c0001
Sets the GPU scheduling policy to equal share scheduler with a strict round-robin scheduling policy, an averaging factor of 60 (0x3C), and a scheduling frequency of 960 (0x3C0) Hz.

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

0x3c3c0011
Sets the GPU scheduling policy to fixed share scheduler with a strict round-robin scheduling policy, an averaging factor of 60 (0x3C), and a scheduling frequency of 960 (0x3C0) Hz.

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

3. Reboot your hypervisor host machine.

Confirm that the scheduling behavior was changed as required as explained in *Linux-Based Hypervisors Only: Getting the Current Time-Sliced vGPU Scheduling Policy for All GPUs*.

7.6.1.2 Linux-Based Hypervisors: Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs by Using the RmPVMRL Registry Key

Perform this task in your hypervisor command shell.

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

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

2. Use the `lspci` command to obtain the PCI domain and bus/device/function (BDF) of each GPU for which you want to change the scheduling behavior.

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

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

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

     ```
     # lspci | grep NVIDIA
     ```

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

   ```
   0000:86:00.0 3D controller: NVIDIA Corporation GP104GL [Tesla P4] (rev a1)
   ```

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

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

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

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

- On VMware vSphere, use the `esxcli set` command.

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

  - For the NVIDIA Ada Lovelace GPU architecture and later GPU architectures, `<module>` is `nvidia-gpu`.
  - For GPU architectures before NVIDIA Ada Lovelace, `<module>` is `nvidia`.

For each GPU, provide the following information:

<pci-domain>
The PCI domain of the GPU.

<pci-bdf>
The PCI device BDF of the GPU.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Reboot your hypervisor host machine.

Confirm that the scheduling behavior was changed as required as explained in *Linux-Based Hypervisors Only: Getting the Current Time-Sliced vGPU Scheduling Policy for All GPUs*.

7.6.1.3 Linux-Based Hypervisors: Restoring Default Time-Sliced vGPU Scheduler Settings by Using the RmPVMRL Registry Key

Perform this task in your hypervisor command shell.

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

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

2. Unset the `RmPVMRL` registry key.

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

     ```
     # esxcli system module parameters set -m <module> -p "<module-
     ↪parameter>="
     ```

   `<module>`

   - For the NVIDIA Ada Lovelace GPU architecture and later GPU architectures, `<module>` is `nvidia-gpu`.
   - For GPU architectures before NVIDIA Ada Lovelace, `<module>` is `nvidia`.

   `<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.

7.6.2. Windows Hypervisors: Changing Scheduling Behavior for Time-Sliced vGPUs by Using the RmPVMRL Registry Key

To use the `RmPVMRL` registry key to change the scheduling behavior of time-sliced vGPUs, use the Windows Registry Editor to set the `RmPVMRL` registry key value.

7.6.2.1 Windows Hypervisors: Getting the Driver Key of a GPU

The `RmPVMRL` registry key of a GPU on a host running a Windows hypervisor is located in the GPU's driver key. When setting the `RmPVMRL` registry key value for a GPU, you need the GPU's driver key to identify the Windows registry key in which information about the GPU is stored.

Perform this task in a Windows PowerShell window on the hypervisor host.

1. Get the `InstanceID` property of all available NVIDIA GPUs in your system.

```
PS C:\> Get-PnpDevice -PresentOnly |
>> Where-Object {$_.InstanceId -like "PCI\VEN_10DE*" }  |
>> Select-Object -Property FriendlyName,InstanceId |
>> Format-List

...
FriendlyName : NVIDIA A40
InstanceId   : PCI\VEN_10DE&DEV_2235&SUBSYS_145A10DE&REV_A1\6&2D4D2F51&0&
↪00800018
...
```

> **Note**
>
> If multiple NVIDIA GPUs are available on the system, analyze the output from this command to get the `InstanceID` property of the GPU on which you want to set the allowed vGPU series.

2. Get the `DEVPKEY_Device_Driver` property of the GPU from the `InstanceID` property that you got in the previous step.

```
PS C:\> Get-PnpDeviceProperty -InstanceId "<instance-id>" |
>> where {$_.KeyName -eq "DEVPKEY_Device_Driver"} |
>> Select-Object -Property Data

Data
```

(continues on next page)

(continued from previous page)

```
----
{4d36e968-e325-11ce-bfc1-08002be10318}\0001
```

<instance-id>
The `InstanceID` property of the GPU that you got in the previous step, for example, `PCI\VEN_10DE&DEV_2235&SUBSYS_145A10DE&REV_A1\6&2D4D2F51&0&00800018`.

7.6.2.2 Windows Hypervisors: Changing the Time-Sliced vGPU Scheduling Behavior for Select GPUs by Using the RmPVMRL Registry Key

Perform this task on the hypervisor host.

1. Get the driver key of the GPU on which you want to change the scheduling behavior.

   For instructions, refer to *Windows Hypervisors: Getting the Driver Key of a GPU*.

   You will need this information in the next step to identify the Windows registry key in which information about the GPU is stored.

2. In the Windows Registry Editor, navigate to the Windows registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\<driver-key>`.

   <driver-key>
   The driver key for the GPU that you got in the previous step, for example:

   ```
   {4d36e968-e325-11ce-bfc1-08002be10318}\0001
   ```

3. Set the `RmPVMRL` DWord (`REG_DWORD`) registry value to a hexadecimal integer that specifies the GPU scheduling policy and the length of the time slice that you want.

   For example:

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

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

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

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

   For all supported values, refer to *RmPVMRL Registry Key*.

4. Reboot your hypervisor host machine.

Confirm that the scheduling behavior was changed as required as explained in *Getting Time-Sliced vGPU Scheduler State Information*.

7.6.2.3 Windows Hypervisors: Restoring Default Time-Sliced vGPU Scheduler Settings by Using the RmPVMRL Registry Key

Perform this task on the hypervisor host.

1. Get the driver key of the GPU on which you want to change the scheduling behavior.

   For instructions, refer to *Windows Hypervisors: Getting the Driver Key of a GPU*.

You will need this information in the next step to identify the Windows registry key in which information about the GPU is stored.

2. In the Windows Registry Editor, navigate to the Windows registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\<driver-key>`.

   `<driver-key>`
   The driver key for the GPU that you got in the previous step, for example:

   ```
   {4d36e968-e325-11ce-bfc1-08002be10318}\0001
   ```

3. Remove the `RmPVMRL` DWord (`REG_DWORD`) registry value.
4. Reboot your hypervisor host machine.

7.6.3. RmPVMRL Registry Key

The `RmPVMRL` registry key controls the scheduling behavior for NVIDIA vGPUs by setting the scheduling policy, the averaging factor and scheduling frequency for schedulers with a strict round-robin scheduling policy.

> **Note**
>
> Unlike in earlier NVIDIA vGPU software releases, strict round robin policy **cannot** be disabled.

7.6.3.1 Type

`Dword`

7.6.3.2 Contents

Value	Meaning
`0x00` (default)	Best effort scheduler
`0x01`	Equal share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor
`0x03`	Unlike in earlier NVIDIA vGPU software releases, this value is invalid because strict round robin policy **cannot** be disabled.
`0x<AA><FFF>001`	Equal share scheduler with a strict round-robin scheduling policy and a user-defined averaging factor <AA> and a user-defined scheduling frequency <FFF>
`0x00<TT>0003`	Unlike in earlier NVIDIA vGPU software releases, this value is invalid because strict round robin policy **cannot** be disabled and a user-defined time slice length <TT> **cannot** be set.
`0x11`	Fixed share scheduler with a strict round-robin scheduling policy and the default time slice length, scheduling frequency, and averaging factor
`0x13`	Unlike in earlier NVIDIA vGPU software releases, this value is invalid because strict round robin policy **cannot** be disabled.
`0x<AA><FFF>011`	Fixed share scheduler with a strict round-robin scheduling policy and a user-defined averaging factor <AA> and a user-defined scheduling frequency <FFF>
`0x00<TT>0013`	Unlike in earlier NVIDIA vGPU software releases, this value is invalid because strict round robin policy **cannot** be disabled and a user-defined time slice length <TT> **cannot** be set.

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

Table 7.1: Default Time Slice Length and Scheduling Frequency by vGPU Density

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

<AA>

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

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

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

- If <AA> is greater than 0x3C, the value is capped at 0x3C.

<FFF>

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

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

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

- If <FFF> is 000, the scheduling frequency is set to the default scheduling frequency for the vGPU type as listed in Table 7.1.
- If <FFF> is greater than 000 but less than 0x3F, the scheduling frequency is raised to 0x3F (decimal 63).
- If <FFF> is greater than 0x3C0, the scheduling frequency is capped at 0x3C0 (decimal 960).

7.6.3.3 Examples

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

```
RmPVMRL=0x01
```

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

```
RmPVMRL=0x11
```

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

```
RmPVMRL=0x3c3c0001
```

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

```
RmPVMRL=0x3c3c0011
```

Chapter 8. Troubleshooting

This chapter describes basic troubleshooting steps for NVIDIA vGPU on Linux-style hypervisors and how to collect debug information when filing a bug report.

8.1. Known issues

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

8.2. Troubleshooting steps

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

8.2.1. Verifying the NVIDIA Kernel Driver Is Loaded

1. Use the command that your hypervisor provides to verify that the kernel driver is loaded:
   - On Linux-style hypervisors except VMware vSphere, use `lsmod`:

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

   - On VMware vSphere, use `vmkload_mod`:

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

   `<module>` depends on the GPU architecture of your GPU:

   - For the NVIDIA Ada Lovelace GPU architecture and later GPU architectures, `<module>` is `nvidia-gpu`.
   - For GPU architectures before NVIDIA Ada Lovelace, `<module>` is `nvidia`.
2. If the `nvidia` or `nvidia-gpu` 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 XenServer and Red Hat Enterprise Linux KVM, also use the `rpm -q` command to verify that the NVIDIA GPU Manager package is correctly installed.

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

<vgpu-manager-rpm-package-name>
The RPM package name of the NVIDIA GPU Manager package, for example `NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-595.112.08` for XenServer.

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

```
[root@xenserver ~]# rpm -q NVIDIA-vGPU-NVIDIA-vGPU-CitrixHypervisor-8.2-
↪595.112.08
[root@xenserver ~]#
If an existing NVIDIA GRID package is already installed and you don't
↪select the upgrade (-U) option when installing a newer GRID package,
↪the rpm command will return many conflict errors.
Preparing packages for installation...
        file /usr/bin/nvidia-smi from install of NVIDIA-vGPU-NVIDIA-vGPU-
↪CitrixHypervisor-8.2-595.112.08.x86_64 conflicts with file from package
↪NVIDIA-vGPU-xenserver-8.2-595.91.04.x86_64
        file /usr/lib/libnvidia-ml.so from install of NVIDIA-vGPU-NVIDIA-
↪vGPU-CitrixHypervisor-8.2-595.112.08.x86_64 conflicts with file from
↪package NVIDIA-vGPU-xenserver-8.2-595.91.04.x86_64
        ...
```

8.2.2. Verifying that `nvidia-smi` works

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

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

8.2.3. Examining NVIDIA kernel driver output

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

Run `dmesg` on a supported Linux-style hypervisor and check for the `NVRM` and `nvidia` prefixes:

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

8.2.4. Examining NVIDIA Virtual GPU Manager Messages

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

8.2.4.1 Examining XenServer vGPU Manager Messages

For 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
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: gpu-pci-id :
↪0000:05:00.0
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: vgpu_type :
↪quadro
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Framebuffer:
↪0x74000000
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Virtual Device
↪Id: 0xl3F2:0xll4E
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: ######## vGPU
↪Manager Information: ########
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Driver Version:
↪595.112.08
May 18 10:34:03 localhost vgpu-ll[25698]: notice: vmiop_log: Init frame copy
↪engine: syncing...
May 18 10:35:31 localhost vgpu-ll[25698]: notice: vmiop_log: ######## Guest
↪NVIDIA Driver Information: ########
May 18 10:35:31 localhost vgpu-ll[25698]: notice: vmiop_log: Driver Version:
↪597.30
May 18 10:35:36 localhost vgpu-ll[25698]: notice: vmiop_log: Current max
↪guest pfn = 0xllbc84!
May 18 10:35:40 localhost vgpu-ll[25698]: notice: vmiop_log: Current max
↪guest pfn = 0xlleff0!
[root@xenserver ~]#
```

8.2.4.2 Examining Red Hat Enterprise Linux KVM vGPU Manager Messages

For Red Hat Enterprise Linux KVM , 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
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop-env:
↪guest_max_gpfn:0x11f7ff
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: pluginconfig: /
↪usr/share/nvidia/vgx/grid_m60-1q.conf,gpu-pci-id=0000:06:00.0
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: Loading
↪Plugin0: libnvidia-vgpu
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: Successfully
↪update the env symbols!
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log: gpu-
↪pci-id : 0000:06:00.0
```

(continues on next page)

(continued from previous page)

```
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log:
↪vgpu_type : quadro
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log:
↪Framebuffer: 0x38000000
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log:
↪Virtual Device Id: 0x13F2:0x114D
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log: ####
↪#### vGPU Manager Information: ########
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log:
↪Driver Version: 595.112.08
[2026-05-15 04:46:12] vmiop_log: [2026-05-15 04:46:12] notice: vmiop_log:
↪Init frame copy engine: syncing...
[2026-05-15 05:09:14] vmiop_log: [2026-05-15 05:09:14] notice: vmiop_log: ####
↪#### Guest NVIDIA Driver Information: ########
[2026-05-15 05:09:14] vmiop_log: [2026-05-15 05:09:14] notice: vmiop_log:
↪Driver Version: 597.30
[2026-05-15 05:09:14] vmiop_log: [2026-05-15 05:09:14] notice: vmiop_log:
↪Current max guest pfn = 0x11a71f!
[2026-05-15 05:12:09] vmiop_log: [2026-05-15 05:12:09] notice: vmiop_log:
↪vGPU license state: (0x00000001)
#
```

8.2.4.3 Examining VMware vSphere vGPU Manager Messages

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

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

```
[root@esxi:~] grep vmiop /vmfs/volumes/datastore1/wind-vgpu-test1/vmware.log
2026-05-15T14:02:21.275Z| vmx| I120: DICT   pciPassthru0.virtualDev = "vmiop"
2026-05-15T14:02:21.344Z| vmx| I120: GetPluginPath testing /usr/lib64/vmware/
↪plugin/libvmx-vmiop.so
2026-05-15T14:02:21.344Z| vmx| I120: PluginLdr_LoadShared: Loaded shared
↪plugin libvmx-vmiop.so from /usr/lib64/vmware/plugin/libvmx-vmiop.so
2026-05-15T14:02:21.344Z| vmx| I120: VMIOP: Loaded plugin libvmx-vmiop.
↪so:VMIOP_InitModule
2026-05-15T14:02:21.359Z| vmx| I120: VMIOP: Initializing plugin vmiop-display
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: gpu-pci-id : 0000:04:00.0
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: vgpu_type : quadro
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: Framebuffer: 0x74000000
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: Virtual Device Id:
↪0x11B0:0x101B
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: ######## vGPU Manager
↪Information: ########
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: Driver Version: 595.112.08
2026-05-15T14:02:21.365Z| vmx| I120: vmiop_log: VGX Version: 20.2
2026-05-15T14:02:21.445Z| vmx| I120: vmiop_log: Init frame copy engine:
↪syncing...
2026-05-15T14:02:37.031Z| vthread-12| I120: vmiop_log: ######## Guest NVIDIA
↪Driver Information: ########
2026-05-15T14:02:37.031Z| vthread-12| I120: vmiop_log: Driver Version: 597.30
```

(continues on next page)

(continued from previous page)

```
2026-05-15T14:02:37.031Z| vthread-12| I120: vmiop_log: VGX Version: 20.2
2026-05-15T14:02:37.093Z| vthread-12| I120: vmiop_log: Clearing BAR1 mapping
2026-05-18T23:39:55.726Z| vmx| I120: VMIOP: Shutting down plugin vmiop-display
[root@esxi:~]
```

8.3. 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 `nvidia-bug-report.sh`.
- On Citrix XenServer, create a XenServer server status report.

8.3.1. 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 XenServer dom0 shell, the host shell of a supported Linux with KVM hypervisor, or the VMware ESXi host shell.

This example runs `nvidia-bug-report.sh` on XenServer, but the procedure is the same on any supported Linux with KVM hypervisoror 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.
```

(continues on next page)

(continued from previous page)

```
complete

[root@xenserver ~]#
```

8.3.2. 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.

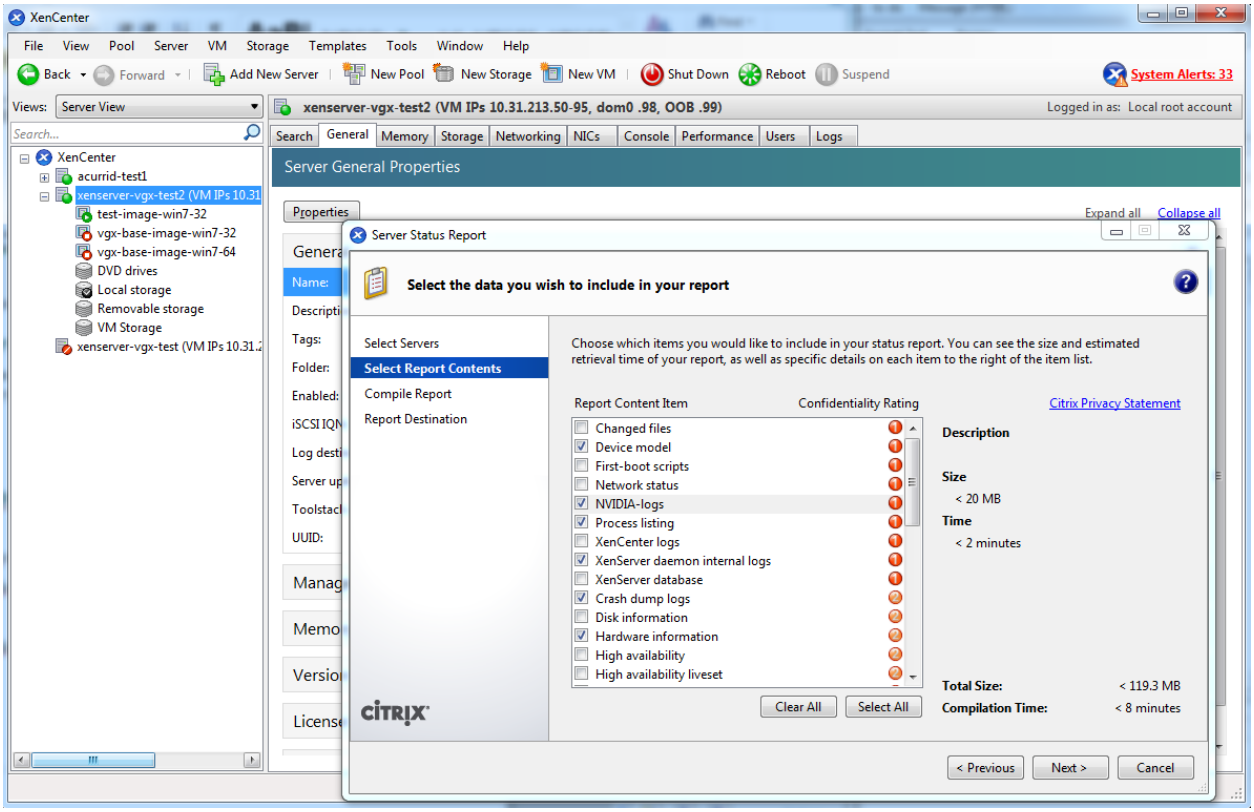


Figure 8.1: Including NVIDIA logs in a XenServer status report

8.4. Gathering Troubleshooting Information for XID 119 Errors

If you experience XID 119 errors, ensure that you have upgraded the NVIDIA vGPU software graphics driver to a release that provides information specifically for troubleshooting these errors. You can provide this information to NVIDIA Enterprise Support for troubleshooting these errors. Upgrading the NVIDIA vGPU software graphics driver does **not** provide a fix for these errors.

> **Note**
>
> XID 119 errors are caused by a GPU System Processor (GSP) RPC timeout. Therefore, these errors occur only with NVIDIA vGPU deployments on GPUs based on the NVIDIA Ada Lovelace and Hopper GPU architectures.

To gather the additional information about XID 119 errors that the NVIDIA vGPU software graphics drivers provide, download and run the script from NVIDIA Enterprise Support for this purpose. For more information about this script, refer to the article about capturing XID 119 and XID 120 error information in the NVIDIA knowledge base.

Chapter 9. Virtual GPU Types Reference

9.1. Virtual GPU Types for Supported GPUs

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

9.1.1. NVIDIA A40 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.1.1 Q-Series Virtual GPU Types for NVIDIA A40

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A40-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
A40-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
A40-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
A40-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
A40-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
A40-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
A40-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4
A40-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
A40-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
A40-1Q	1024	32[3]	30	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.1.2 B-Series Virtual GPU Types for NVIDIA A40

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A40-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A40-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A40-1B	1024	32	32	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[2]

9.1.1.3 A-Series Virtual GPU Types for NVIDIA A40

Intended use case: Virtual Applications

Required license edition: vApps

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

[3] The maximum number of vGPUs allowed on an NVIDIA A40 GPU is limited to 32.

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A40-48A	49152	1	1	1280×1024	1
A40-24A	24576	2	2	1280×1024	1
A40-16A	16384	3	2	1280×1024	1
A40-12A	12288	4	4	1280×1024	1
A40-8A	8192	6	4	1280×1024	1
A40-6A	6144	8	8	1280×1024	1
A40-4A	4096	12	8	1280×1024	1
A40-3A	3072	16	16	1280×1024	1
A40-2A	2048	24	16	1280×1024	1
A40-1A	1024	32[Page 227, 3]	32	1280×1024	1

9.1.2. NVIDIA A16 Virtual GPU Types

Physical GPUs per board: 4

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

9.1.2.1 Q-Series Virtual GPU Types for NVIDIA A16

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A16-16Q	16384	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
A16-8Q	8192	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
A16-4Q	4096	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
A16-2Q	2048	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
A16-1Q	1024	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.2.2 B-Series Virtual GPU Types for NVIDIA A16

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A16-3B	3072	5	4	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A16-2B	2048	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A16-1B	1024	16	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.2.3 A-Series Virtual GPU Types for NVIDIA A16

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A16-16A	16384	1	1	1280×1024	1
A16-8A	8192	2	2	1280×1024	1
A16-4A	4096	4	4	1280×1024	1
A16-2A	2048	8	8	1280×1024	1
A16-1A	1024	16	16	1280×1024	1

9.1.3. NVIDIA A10 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.3.1 Q-Series Virtual GPU Types for NVIDIA A10

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A10-24Q	24576	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
A10-12Q	12288	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
A10-8Q	8192	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
A10-6Q	6144	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
A10-4Q	4096	6	4	58982400	7680×4320	1
					5120×2880 or lower	4
A10-3Q	3072	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
A10-2Q	2048	12	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
A10-1Q	1024	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.3.2 B-Series Virtual GPU Types for NVIDIA A10

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of

high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A10-3B	3072	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A10-2B	2048	12	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A10-1B	1024	24	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.3.3 A-Series Virtual GPU Types for NVIDIA A10

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A10-24A	24576	1	1	1280×1024	1
A10-12A	12288	2	2	1280×1024	1
A10-8A	8192	3	2	1280×1024	1
A10-6A	6144	4	4	1280×1024	1
A10-4A	4096	6	4	1280×1024	1
A10-3A	3072	8	8	1280×1024	1
A10-2A	2048	12	8	1280×1024	1
A10-1A	1024	24	16	1280×1024	1

9.1.4. NVIDIA A2 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.4.1 Q-Series Virtual GPU Types for NVIDIA A2

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A2-16Q	16384	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
A2-8Q	8192	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
A2-4Q	4096	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
A2-2Q	2048	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
A2-1Q	1024	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.4.2 B-Series Virtual GPU Types for NVIDIA A2

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
A2-3B	3072	5	4	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A2-2B	2048	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
A2-1B	1024	16	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.4.3 A-Series Virtual GPU Types for NVIDIA A2

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
A2-16A	16384	1	1	1280×1024	1
A2-8A	8192	2	2	1280×1024	1
A2-4A	4096	4	4	1280×1024	1
A2-2A	2048	8	8	1280×1024	1
A2-1A	1024	16	16	1280×1024	1

9.1.5. NVIDIA L40 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.5.1 Q-Series Virtual GPU Types for NVIDIA L40

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L40-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
L40-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
L40-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
L40-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
L40-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
L40-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
L40-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4
L40-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L40-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L40-1Q	1024	32[4]	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.5.2 B-Series Virtual GPU Types for NVIDIA L40

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L40-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L40-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L40-1B	1024	32	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.5.3 A-Series Virtual GPU Types for NVIDIA L40

Intended use case: Virtual Applications

Required license edition: vApps

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

[4] The maximum number of vGPUs allowed on an NVIDIA L40 GPU is limited to 32.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L40-48A	49152	1	1	1280×1024	1
L40-24A	24576	2	2	1280×1024	1
L40-16A	16384	3	2	1280×1024	1
L40-12A	12288	4	4	1280×1024	1
L40-8A	8192	6	4	1280×1024	1
L40-6A	6144	8	8	1280×1024	1
L40-4A	4096	12	8	1280×1024	1
L40-3A	3072	16	16	1280×1024	1
L40-2A	2048	24	16	1280×1024	1
L40-1A	1024	32[Page 239, 4]	16	1280×1024	1

9.1.6. NVIDIA L40S Virtual GPU Types

Physical GPUs per board: 1

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

9.1.6.1 Q-Series Virtual GPU Types for NVIDIA L40S

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L40S-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
L40S-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
L40S-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
L40S-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
L40S-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
L40S-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
L40S-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4
L40S-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L40S-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L40S-1Q	1024	32[5]	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.6.2 B-Series Virtual GPU Types for NVIDIA L40S

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L40S-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L40S-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L40S-1B	1024	32	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.6.3 A-Series Virtual GPU Types for NVIDIA L40S

Intended use case: Virtual Applications

Required license edition: vApps

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

[5] The maximum number of vGPUs allowed on an NVIDIA L40S GPU is limited to 32.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L40S-48A	49152	1	1	1280×1024	1
L40S-24A	24576	2	2	1280×1024	1
L40S-16A	16384	3	2	1280×1024	1
L40S-12A	12288	4	4	1280×1024	1
L40S-8A	8192	6	4	1280×1024	1
L40S-6A	6144	8	8	1280×1024	1
L40S-4A	4096	12	8	1280×1024	1
L40S-3A	3072	16	16	1280×1024	1
L40S-2A	2048	24	16	1280×1024	1
L40S-1A	1024	32[Page 242, 5]	16	1280×1024	1

9.1.7. NVIDIA L20 and NVIDIA L20 Liquid Cooled Virtual GPU Types

Physical GPUs per board: 1

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

The virtual GPU types for the NVIDIA L20 and NVIDIA L20 liquid cooled GPUs are identical.

9.1.7.1 Q-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L20-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
L20-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
L20-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
L20-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
L20-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
L20-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
L20-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4
L20-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L20-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L20-1Q	1024	32[6]	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.7.2 B-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L20-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L20-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L20-1B	1024	32	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.7.3 A-Series Virtual GPU Types for NVIDIA L20 and NVIDIA L20 Liquid Cooled

Intended use case: Virtual Applications

Required license edition: vApps

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

[6] The maximum number of vGPUs allowed on an NVIDIA L20 GPU is limited to 32.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L20-48A	49152	1	1	1280×1024	1
L20-24A	24576	2	2	1280×1024	1
L20-16A	16384	3	2	1280×1024	1
L20-12A	12288	4	4	1280×1024	1
L20-8A	8192	6	4	1280×1024	1
L20-6A	6144	8	8	1280×1024	1
L20-4A	4096	12	8	1280×1024	1
L20-3A	3072	16	16	1280×1024	1
L20-2A	2048	24	16	1280×1024	1
L20-1A	1024	32[Page 245, 6]	16	1280×1024	1

9.1.8. NVIDIA L4 Virtual GPU Types

Physical GPUs per board: 1

9.1.8.1 Q-Series Virtual GPU Types for NVIDIA L4

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L4-24Q	24576	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
L4-12Q	12288	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
L4-8Q	8192	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
L4-6Q	6144	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
L4-4Q	4096	6	4	58982400	7680×4320	1
					5120×2880 or lower	4
L4-3Q	3072	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L4-2Q	2048	12	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L4-1Q	1024	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.8.2 B-Series Virtual GPU Types for NVIDIA L4

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of

high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L4-3B	3072	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L4-2B	2048	12	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L4-1B	1024	24	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.8.3 A-Series Virtual GPU Types for NVIDIA L4

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L4-24A	24576	1	1	1280×1024	1
L4-12A	12288	2	2	1280×1024	1
L4-8A	8192	3	2	1280×1024	1
L4-6A	6144	4	4	1280×1024	1
L4-4A	4096	6	4	1280×1024	1
L4-3A	3072	8	8	1280×1024	1
L4-2A	2048	12	8	1280×1024	1
L4-1A	1024	24	16	1280×1024	1

9.1.9. NVIDIA L2 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.9.1 Q-Series Virtual GPU Types for NVIDIA L2

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L2-24Q	24576	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
L2-12Q	12288	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
L2-8Q	8192	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
L2-6Q	6144	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
L2-4Q	4096	6	4	58982400	7680×4320	1
					5120×2880 or lower	4
L2-3Q	3072	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L2-2Q	2048	12	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
L2-1Q	1024	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.9.2 B-Series Virtual GPU Types for NVIDIA L2

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of

high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
L2-3B	3072	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L2-2B	2048	12	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
L2-1B	1024	24	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.9.3 A-Series Virtual GPU Types for NVIDIA L2

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
L2-24A	24576	1	1	1280×1024	1
L2-12A	12288	2	2	1280×1024	1
L2-8A	8192	3	2	1280×1024	1
L2-6A	6144	4	4	1280×1024	1
L2-4A	4096	6	4	1280×1024	1
L2-3A	3072	8	8	1280×1024	1
L2-2A	2048	12	8	1280×1024	1
L2-1A	1024	24	16	1280×1024	1

9.1.10. NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled Virtual GPU Types

Physical GPUs per board: 1

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

The full virtual GPU type names include the prefix **NVIDIA RTX PRO 6000 Blackwell**, for example **NVIDIA RTX PRO 6000 Blackwell DC-4-96Q**.

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

9.1.10.1 MIG-Backed Q-Series Virtual GPU Types for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-4-96Q MIG 4g.96gb+gfx	96	1	4	4	66355200	7680×432(	2
						5120×288(or lower	4
DC-2-48Q MIG 2g.48gb+gfx	48	1	2	2	66355200	7680×432(	2
						5120×288(or lower	4
DC-4-48Q MIG 4g.96gb+gfx	48	2	4	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-4-32Q MIG 4g.96gb+gfx	32	3	4	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-24Q MIG 1g.24gb+gfx	24	1	1	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-2-24Q MIG 2g.48gb+gfx	24	2	2	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-4-24Q MIG 4g.96gb+gfx	24	4	4	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-2-16Q MIG 2g.48gb+gfx	16	3	2	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-12Q MIG 1g.24gb+gfx	12	2	1	1	66355200	7680×432(	2
						5120×288(or lower	4

continues on next page

Table 9.1 – continued from previous page

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-2-12Q MIG 2g.48gb+gfx	12	4	2	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-8Q MIG 1g.24gb+gfx	8	3	1	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-6Q MIG 1g.24gb+gfx	6	4	1	1	58982400	7680×432(	1
						5120×288(or lower	4
DC-1-4Q MIG 1g.24gb+gfx	4	6	1	1	58982400	7680×432(	1
						5120×288(or lower	4
DC-1-3Q MIG 1g.24gb+gfx	3	8	1	1	36864000	7680×432(	1
						5120×288(	2
						3840×240(or lower	4
DC-1-2Q MIG 1g.24gb+gfx	2	12	1	1	36864000	7680×432(	1
						5120×288(	2
						3840×240(or lower	4

9.1.10.2 Time-Sliced Q-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display*

Configurations for B-Series and Q-Series vGPUs.

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-96Q	96	1	1	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-48Q	48	2	2	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-32Q	32	3	3	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-24Q	24	4	4	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-16Q	16	6	6	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-12Q	12	8	8	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-8Q	8	12	12	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-6Q	6	16	16	5898240(	7680×4320	1
					5120×2880 or lower	4
DC-4Q	4	24	24	5898240(	7680×4320	1
					5120×2880 or lower	4
DC-3Q	3	32	32	3686400(	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4

9.1.10.3 MIG-Backed B-Series Virtual GPU Types for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Max. Display Resolution	Virtual Displays per vGPU
DC-1-3B MIG 1g.24gb+gfx	3	8	1	1	18432000	5120×288(	1
						3840×240(	2
						3840×216(	2
						2560×160(or lower	4
DC-1-2B MIG 1g.24gb+gfx	2	12	1	1	18432000	5120×288(	1
						3840×240(	2
						3840×216(	2
						2560×160(or lower	4

9.1.10.4 Time-Sliced B-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-3B	3	32	32	1843200(	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
DC-2B	2	32	32	1843200(	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.10.5 MIG-Backed A-Series Virtual GPU Types for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled

Intended use case: Virtual Applications

Required license edition: vApps

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

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Max. Display Resolution	Virtual Displays per vGPU
DC-4-96A MIG 4g.96gb+gfx	96	1	4	4	1280×1024	1
DC-2-48A MIG 2g.48gb+gfx	48	1	2	2	1280×1024	1
DC-4-48A MIG 4g.96gb+gfx	48	2	4	1	1280×1024	1
DC-4-32A MIG 4g.96gb+gfx	32	3	4	1	1280×1024	1
DC-1-24A MIG 1g.24gb+gfx	24	1	1	1	1280×1024	1
DC-2-24A MIG 2g.48gb+gfx	24	2	2	1	1280×1024	1
DC-4-24A MIG 4g.96gb+gfx	24	4	2	1	1280×1024	1
DC-2-16A MIG 2g.48gb+gfx	16	3	2	1	1280×1024	1
DC-1-12A MIG 1g.24gb+gfx	12	2	1	1	1280×1024	1
DC-2-12A MIG 2g.48gb+gfx	12	4	2	1	1280×1024	1
DC-1-8A MIG 1g.24gb+gfx	8	3	1	1	1280×1024	1
DC-1-6A MIG 1g.24gb+gfx	6	4	1	1	1280×1024	1
DC-1-4A MIG 1g.24gb+gfx	4	6	1	1	1280×1024	1
DC-1-3A MIG 1g.24gb+gfx	3	8	1	1	1280×1024	1
DC-1-2A MIG 1g.24gb+gfx	2	12	1	1	1280×1024	1

9.1.10.6 Time-Sliced A-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 6000 Blackwell Server Edition and NVIDIA RTX PRO 6000 Blackwell Server Edition Liquid Cooled

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Display Resolution	Virtual Displays per vGPU
DC-96A	96	1	1	1280×1024	1
DC-48A	48	2	2	1280×1024	1
DC-32A	32	3	3	1280×1024	1
DC-24A	24	4	4	1280×1024	1
DC-16A	16	6	6	1280×1024	1
DC-12A	12	8	8	1280×1024	1
DC-8A	8	12	12	1280×1024	1
DC-6A	6	16	16	1280×1024	1
DC-4A	4	24	24	1280×1024	1
DC-3A	3	32	32	1280×1024	1

9.1.11. **Since 20.2:** NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB Virtual GPU Types

Physical GPUs per board: 1

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

The full virtual GPU type names include the prefix **NVIDIA RTX PRO 5000 72GB**, for example **NVIDIA RTX PRO 5000 72GB Blackwell-12Q**. The tables in this section use the short form of each name, for example **Blackwell-12Q**.

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

This GPU supports only Q-series virtual GPU types for NVIDIA RTX Virtual Workstation and B-series virtual GPU types for NVIDIA Virtual PC. Because this GPU is supported only in workstation systems, no A-series virtual GPU types are provided.

9.1.11.1 MIG-Backed Q-Series Virtual GPU Types for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Display Resolution	Virtual Displays per vGPU
Blackwell-1-2Q MIG 1g.36gb+gfx	2	18	1	1	36864000	7680×432(	1
						5120×288(	2
						3840×240(or lower	4
Blackwell-1-3Q MIG 1g.36gb+gfx	3	12	1	1	36864000	7680×432(	1
						5120×288(	2
						3840×240(or lower	4
Blackwell-1-4Q MIG 1g.36gb+gfx	4	9	1	1	58982400	7680×432(	1
						5120×288(or lower	4
Blackwell-1-6Q MIG 1g.36gb+gfx	6	6	1	1	58982400	7680×432(	1
						5120×288(or lower	4
Blackwell-1-9Q MIG 1g.36gb+gfx	9	4	1	1	66355200	7680×432(	2
						5120×288(or lower	4
Blackwell-1-12Q MIG 1g.36gb+gfx	12	3	1	1	66355200	7680×432(	2
						5120×288(or lower	4
Blackwell-1-18Q MIG 1g.36gb+gfx	18	2	1	1	66355200	7680×432(	2
						5120×288(or lower	4
Blackwell-1-36Q MIG 1g.36gb+gfx	36	1	1	1	66355200	7680×432(	2
						5120×288(or lower	4

continues on next page

Table 9.2 – continued from previous page

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Display Resolution	Virtual Displays per vGPU
Blackwell-2-18Q MIG 2g.72gb+gfx	18	4	2	1	66355200	7680×432(	2
						5120×288(or lower	4
Blackwell-2-24Q MIG 2g.72gb+gfx	24	3	2	1	66355200	7680×432(	2
						5120×288(or lower	4
Blackwell-2-36Q MIG 2g.72gb+gfx	36	2	2	1	66355200	7680×432(	2
						5120×288(or lower	4
Blackwell-2-72Q MIG 2g.72gb+gfx	72	1	2	1	66355200	7680×432(	2
						5120×288(or lower	4

9.1.11.2 Time-Sliced Q-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
Blackwell-72Q	72	1	1	6635520(	7680×4320	2
					5120×2880 or lower	4

continues on next page

Table 9.3 – continued from previous page

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
Blackwell-36Q	36	2	2	6635520(	7680×4320	2
					5120×2880 or lower	4
Blackwell-24Q	24	3	3	6635520(	7680×4320	2
					5120×2880 or lower	4
Blackwell-18Q	18	4	4	6635520(	7680×4320	2
					5120×2880 or lower	4
Blackwell-12Q	12	6	6	6635520(	7680×4320	2
					5120×2880 or lower	4
Blackwell-9Q	9	8	8	6635520(	7680×4320	2
					5120×2880 or lower	4
Blackwell-6Q	6	12	12	5898240(	7680×4320	1
					5120×2880 or lower	4
Blackwell-4Q	4	18	18	5898240(	7680×4320	1
					5120×2880 or lower	4
Blackwell-3Q	3	24	24	3686400(	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
Blackwell-2Q	2	32	32	3686400(	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4

9.1.11.3 MIG-Backed B-Series Virtual GPU Types for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels,

which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Max. Display Resolution	Virtual Displays per vGPU
Blackwell-1-3B MIG 1g.36gb+gfx	3	12	1	1	18432000	5120×288(	1
						3840×240(	2
						3840×216(	2
						2560×160(or lower	4
Blackwell-1-2B MIG 1g.36gb+gfx	2	18	1	1	18432000	5120×288(	1
						3840×240(	2
						3840×216(	2
						2560×160(or lower	4

9.1.11.4 Time-Sliced B-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 5000 Blackwell Workstation Edition 72 GB

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
Blackwell-3B	3	24	24	1843200	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
Blackwell-2B	2	32	32	1843200	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.12. NVIDIA RTX PRO 4500 Blackwell Server Edition Virtual GPU Types

Physical GPUs per board: 1

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

The full virtual GPU type names include the prefix **NVIDIA RTX PRO 4500 Blackwell**, for example **NVIDIA RTX PRO 4500 Blackwell DC-2-32Q**.

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

9.1.12.1 MIG-Backed Q-Series Virtual GPU Types for NVIDIA RTX PRO 4500 Blackwell Server Edition

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-2-32Q 2g.32gb+gfx	32	1	2	2	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-16Q 1g.16gb+gfx	16	1	1	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-2-16Q 2g.32gb+gfx	16	2	2	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-8Q 1g.16gb+gfx	8	2	1	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-2-8Q 2g.32gb+gfx	8	4	2	1	66355200	7680×432(	2
						5120×288(or lower	4
DC-1-4Q 1g.16gb+gfx	4	4	1	1	58982400	7680×432(	1
						5120×288(or lower	4
DC-1-2Q 1g.16gb+gfx	2	8	1	1	36864000	7680×432(	1
						5120×288(	2
						3840×240(or lower	4

9.1.12.2 Time-Sliced Q-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 4500 Blackwell Server Edition

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of

high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-32Q	32	1	1	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-16Q	16	2	2	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-8Q	8	4	4	6635520(	7680×4320	2
					5120×2880 or lower	4
DC-4Q	4	8	8	5898240(	7680×4320	1
					5120×2880 or lower	4

9.1.12.3 MIG-Backed B-Series Virtual GPU Types for NVIDIA RTX PRO 4500 Blackwell Server Edition

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Available Pixels	Max. Display Resolution	Virtual Displays per vGPU
DC-1-3B MIG 1g.16gb+gfx	3	5	1	1	18432000	5120×288(	1
						3840×240(	2
						3840×216(	2
						2560×160(or lower	4
DC-1-2B MIG 1g.16gb+gfx	2	8	1	1	18432000	5120×288(	1
						3840×240(	2
						3840×216(	2
						2560×160(or lower	4

9.1.12.4 Time-Sliced B-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 4500 Blackwell Server Edition

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
DC-3B	3	10	8	1843200	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
DC-2B	2	16	16	1843200	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.12.5 MIG-Backed A-Series Virtual GPU Types for NVIDIA RTX PRO 4500 Blackwell Server Edition

Intended use case: Virtual Applications

Required license edition: vApps

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

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

Each virtual GPU type is supported **only** on its corresponding GPU instance profile.

Virtual GPU Type and Corresponding GPU Instance Profile	Frame Buffer (GB)	Max. vGPUs per GPU Instance	Slices per vGPU	Compute Instances per vGPU	Max. Display Resolution	Virtual Displays per vGPU
DC-2-32A 2g.32gb+gfx	32	1	2	2	1280×1024	1
DC-1-16A 1g.16gb+gfx	16	1	1	1	1280×1024	1
DC-2-16A 2g.32gb+gfx	16	2	2	1	1280×1024	1
DC-1-8A 1g.16gb+gfx	8	2	1	1	1280×1024	1
DC-2-8A 2g.32gb+gfx	8	4	2	1	1280×1024	1
DC-1-4A 1g.16gb+gfx	4	4	1	1	1280×1024	1
DC-1-2A 1g.16gb+gfx	2	8	1	1	1280×1024	1

9.1.12.6 Time-Sliced A-Series Virtual GPU Types on Single-Instance GPUs for NVIDIA RTX PRO 4500 Blackwell Server Edition

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (GB)	Max. vGPUs per GPU in Equal-Size Mode	Max. vGPUs per GPU in Mixed-Size Mode	Display Resolution	Virtual Displays per vGPU
DC-32A	32	1	1	1280×1024	1
DC-16A	16	2	2	1280×1024	1
DC-8A	8	4	4	1280×1024	1
DC-4A	4	8	8	1280×1024	1

9.1.13. NVIDIA RTX 6000 Ada Virtual GPU Types

Physical GPUs per board: 1

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

9.1.13.1 Q-Series Virtual GPU Types for NVIDIA RTX 6000 Ada

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 6000 Ada-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 6000 Ada-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 6000 Ada-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 6000 Ada-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 6000 Ada-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 6000 Ada-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
RTX 6000 Ada-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4

continues on next page

Table 9.4 – continued from previous page

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 6000 Ada-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTX 6000 Ada-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTX 6000 Ada-1Q	1024	32[7]	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.13.2 B-Series Virtual GPU Types for NVIDIA RTX 6000 Ada

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

[7] The maximum number of vGPUs allowed on an NVIDIA RTX 6000 Ada GPU is limited to 32.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 6000 Ada-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTX 6000 Ada-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTX 6000 Ada-1B	1024	32	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.13.3 A-Series Virtual GPU Types for NVIDIA RTX 6000 Ada

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTX 6000 Ada-48A	49152	1	1	1280×1024	1
RTX 6000 Ada-24A	24576	2	2	1280×1024	1
RTX 6000 Ada-16A	16384	3	2	1280×1024	1
RTX 6000 Ada-12A	12288	4	4	1280×1024	1
RTX 6000 Ada-8A	8192	6	4	1280×1024	1
RTX 6000 Ada-6A	6144	8	8	1280×1024	1
RTX 6000 Ada-4A	4096	12	8	1280×1024	1
RTX 6000 Ada-3A	3072	16	16	1280×1024	1
RTX 6000 Ada-2A	2048	24	16	1280×1024	1
RTX 6000 Ada-1A	1024	32[Page 271, 7]	16	1280×1024	1

9.1.14. NVIDIA RTX 5880 Ada Virtual GPU Types

Physical GPUs per board: 1

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

9.1.14.1 Q-Series Virtual GPU Types for NVIDIA RTX 5880 Ada

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 5880 Ada-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5880 Ada-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5880 Ada-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5880 Ada-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5880 Ada-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5880 Ada-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
RTX 5880 Ada-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4
RTX 5880 Ada-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTX 5880 Ada-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2

continues on next page

Table 9.5 – continued from previous page

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
					3840×2400 or lower	4
RTX 5880 Ada-1Q	1024	32[8]	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.14.2 B-Series Virtual GPU Types for NVIDIA RTX 5880 Ada

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

[8] The maximum number of vGPUs allowed on an NVIDIA RTX 5880 Ada GPU is limited to 32.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 5880 Ada-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTX 5880 Ada-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTX 5880 Ada-1B	1024	32	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.14.3 A-Series Virtual GPU Types for NVIDIA RTX 5880 Ada

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTX 5880 Ada-48A	49152	1	1	1280×1024	1
RTX 5880 Ada-24A	24576	2	2	1280×1024	1
RTX 5880 Ada-16A	16384	3	2	1280×1024	1
RTX 5880 Ada-12A	12288	4	4	1280×1024	1
RTX 5880 Ada-8A	8192	6	4	1280×1024	1
RTX 5880 Ada-6A	6144	8	8	1280×1024	1
RTX 5880 Ada-4A	4096	12	8	1280×1024	1
RTX 5880 Ada-3A	3072	16	16	1280×1024	1
RTX 5880 Ada-2A	2048	24	16	1280×1024	1
RTX 5880 Ada-1A	1024	32[Page 275, 8]	16	1280×1024	1

9.1.15. NVIDIA RTX 5000 Ada Virtual GPU Types

Physical GPUs per board: 1

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

9.1.15.1 Q-Series Virtual GPU Types for NVIDIA RTX 5000 Ada

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 5000 Ada-32Q	32768	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5000 Ada-16Q	16384	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5000 Ada-8Q	8192	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTX 5000 Ada-4Q	4096	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
RTX 5000 Ada-2Q	2048	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTX 5000 Ada-1Q	1024	32	32	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.15.2 B-Series Virtual GPU Types for NVIDIA RTX 5000 Ada

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTX 5000 Ada-3B	3072	10	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTX 5000 Ada-2B	2048	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTX 5000 Ada-1B	1024	32	32	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.15.3 A-Series Virtual GPU Types for NVIDIA RTX 5000 Ada

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTX 5000 Ada-32A	32768	1	1	1280×1024	1
RTX 5000 Ada-16A	16384	2	2	1280×1024	1
RTX 5000 Ada-8A	8192	4	4	1280×1024	1
RTX 5000 Ada-4A	4096	8	8	1280×1024	1
RTX 5000 Ada-2A	2048	16	16	1280×1024	1
RTX 5000 Ada-1A	1024	32	32	1280×1024	1

9.1.16. NVIDIA RTX A6000 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.16.1 Q-Series Virtual GPU Types for NVIDIA RTX A6000

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA6000-48Q	49152	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA6000-24Q	24576	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA6000-16Q	16384	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA6000-12Q	12288	4	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA6000-8Q	8192	6	4	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA6000-6Q	6144	8	8	58982400	7680×4320	1
					5120×2880 or lower	4
RTXA6000-4Q	4096	12	8	58982400	7680×4320	1
					5120×2880 or lower	4
RTXA6000-3Q	3072	16	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTXA6000-2Q	2048	24	16	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTXA6000-1Q	1024	32[9]	30	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.16.2 B-Series Virtual GPU Types for NVIDIA RTX A6000

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA6000-3B	3072	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTXA6000-2B	2048	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTXA6000-1B	1024	32	30	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.16.3 A-Series Virtual GPU Types for NVIDIA RTX A6000

Intended use case: Virtual Applications

Required license edition: vApps

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

[9] The maximum number of vGPUs allowed on an NVIDIA RTX A6000 GPU is limited to 32.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA6000-48A	49152	1	1	1280×1024	1
RTXA6000-24A	24576	2	2	1280×1024	1
RTXA6000-16A	16384	3	2	1280×1024	1
RTXA6000-12A	12288	4	4	1280×1024	1
RTXA6000-8A	8192	6	4	1280×1024	1
RTXA6000-6A	6144	8	8	1280×1024	1
RTXA6000-4A	4096	12	8	1280×1024	1
RTXA6000-3A	3072	16	16	1280×1024	1
RTXA6000-2A	2048	24	16	1280×1024	1
RTXA6000-1A	1024	32[Page 282, 9]	30	1280×1024	1

9.1.17. NVIDIA RTX A5500 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.17.1 Q-Series Virtual GPU Types for NVIDIA RTX A5500

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5500-24Q	24576	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA5500-12Q	12288	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA5500-8Q	8192	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA5500-6Q	6144	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
RTXA5500-4Q	4096	6	4	58982400	7680×4320	1
					5120×2880 or lower	4
RTXA5500-3Q	3072	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTXA5500-2Q	2048	12	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTXA5500-1Q	1024	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.17.2 B-Series Virtual GPU Types for NVIDIA RTX A5500

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of

high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5500-3B	3072	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTXA5500-2B	2048	12	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTXA5500-1B	1024	24	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.17.3 A-Series Virtual GPU Types for NVIDIA RTX A5500

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5500-24A	24576	1	1	1280×1024	1
RTXA5500-12A	12288	2	2	1280×1024	1
RTXA5500-8A	8192	3	2	1280×1024	1
RTXA5500-6A	6144	4	4	1280×1024	1
RTXA5500-4A	4096	6	4	1280×1024	1
RTXA5500-3A	3072	8	8	1280×1024	1
RTXA5500-2A	2048	12	8	1280×1024	1
RTXA5500-1A	1024	24	16	1280×1024	1

9.1.18. NVIDIA RTX A5000 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.18.1 Q-Series Virtual GPU Types for NVIDIA RTX A5000

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5000-24Q	24576	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA5000-12Q	12288	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA5000-8Q	8192	3	2	66355200	7680×4320	2
					5120×2880 or lower	4
RTXA5000-6Q	6144	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
RTXA5000-4Q	4096	6	4	58982400	7680×4320	1
					5120×2880 or lower	4
RTXA5000-3Q	3072	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTXA5000-2Q	2048	12	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
RTXA5000-1Q	1024	24	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.18.2 B-Series Virtual GPU Types for NVIDIA RTX A5000

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of

high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
RTXA5000-3B	3072	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTXA5000-2B	2048	12	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
RTXA5000-1B	1024	24	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4[Page 227, 2]

9.1.18.3 A-Series Virtual GPU Types for NVIDIA RTX A5000

Intended use case: Virtual Applications

Required license edition: vApps

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

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
RTXA5000-24A	24576	1	1	1280×1024	1
RTXA5000-12A	12288	2	2	1280×1024	1
RTXA5000-8A	8192	3	2	1280×1024	1
RTXA5000-6A	6144	4	4	1280×1024	1
RTXA5000-4A	4096	6	4	1280×1024	1
RTXA5000-3A	3072	8	8	1280×1024	1
RTXA5000-2A	2048	12	8	1280×1024	1
RTXA5000-1A	1024	24	16	1280×1024	1

9.1.19. Tesla T4 Virtual GPU Types

Physical GPUs per board: 1

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

9.1.19.1 Q-Series Virtual GPU Types for Tesla T4

Intended use case: Virtual Workstations

Required license edition: vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
T4-16Q	16384	1	1	66355200	7680×4320	2
					5120×2880 or lower	4
T4-8Q	8192	2	2	66355200	7680×4320	2
					5120×2880 or lower	4
T4-4Q	4096	4	4	58982400	7680×4320	1
					5120×2880 or lower	4
T4-2Q	2048	8	8	36864000	7680×4320	1
					5120×2880	2
					3840×2400 or lower	4
T4-1Q	1024	16	16	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4

9.1.19.2 B-Series Virtual GPU Types for Tesla T4

Intended use case: Virtual Desktops

Required license edition: vPC or vWS

These vGPU types support a maximum combined resolution based on the number of available pixels, which is determined by their frame buffer size. You can choose between using a small number of high resolution displays or a larger number of lower resolution displays with these vGPU types. The maximum number of displays per vGPU is based on a configuration in which all displays have the same resolution. For examples of configurations with a mixture of display resolutions, see *Mixed Display Configurations for B-Series and Q-Series vGPUs*.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Available Pixels	Display Resolution	Virtual Displays per vGPU
T4-2B	2048	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
T4-2B4[1]	2048	8	8	18432000	5120×2880	1
					3840×2400	2
					3840×2160	2
					2560×1600 or lower	4
T4-1B	1024	16	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4 Page 227, 2
T4-1B4[1]	1024	16	16	16384000	5120×2880	1
					3840×2400	1
					3840×2160	1
					2560×1600 or lower	4 Page 227, 2

9.1.19.3 A-Series Virtual GPU Types for Tesla T4

Intended use case: Virtual Applications

Required license edition: vApps

These vGPU types support a single display with a fixed maximum resolution.

[1] The -1B4 and -2B4 vGPU types are deprecated in this release, and may be removed in a future release. In preparation for the possible removal of these vGPU types, use the following vGPU types, which provide equivalent functionality:

- Instead of -1B4 vGPU types, use -1B vGPU types.
- Instead of -2B4 vGPU types, use -2B vGPU types.

Virtual GPU Type	Frame Buffer (MB)	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Maximum Display Resolution	Virtual Displays per vGPU
T4-16A	16384	1	1	1280×1024	1
T4-8A	8192	2	2	1280×1024	1
T4-4A	4096	4	4	1280×1024	1
T4-2A	2048	8	8	1280×1024	1
T4-1A	1024	16	16	1280×1024	1

9.2. Mixed Display Configurations for B-Series and Q-Series vGPUs

9.2.1. Mixed Display Configurations for B-Series vGPUs

Virtual GPU Type	Available Pixels	Available Pixel Basis	Maximum Displays	Sample Mixed Display Configurations
-3B	18432000	2 3840×2400 displays	4	1 3840×2400 display plus 2 2560×1600 displays
-2B	18432000	2 3840×2400 displays	4	1 3840×2400 display plus 2 2560×1600 displays
-2B4	18432000	2 3840×2400 displays	4	1 3840×2400 display plus 2 2560×1600 displays
-1B	16384000	4 2560×1600 displays	4	1 4096×2160 display plus 1 2560×1600 display
-1B4	16384000	4 2560×1600 displays	4	1 4096×2160 display plus 1 2560×1600 display

9.2.2. Mixed Display Configurations for Q-Series vGPUs

Virtual GPU Type	Available Pixels	Available Pixel Basis	Maximum Displays	Sample Mixed Display Configurations
-8Q and above	66355200	2 7680×4320 displays	4	1 7680×4320 display plus 2 5120×2880 displays
				1 7680×4320 display plus 3 4096×2160 displays
-6Q	58982400	4 5120×2880 displays	4	1 7680×4320 display plus 1 5120×2880 display
-4Q	58982400	4 5120×2880 displays	4	1 7680×4320 display plus 1 5120×2880 display
-3Q	35389440	4 4096×2160 displays	4	1 5120×2880 display plus 2 4096×2160 displays
-2Q	35389440	4 4096×2160 displays	4	1 5120×2880 display plus 2 4096×2160 displays
-1Q	17694720	2 4096×2160 displays	4	1 4096×2160 display plus 2 2560×1600 displays

9.3. vGPU Placements for GPUs

The vGPU placements that a GPU supports depend on the total amount of frame buffer that the GPU has and whether the GPU is in mixed-size mode or equal-size mode.

In equal-size mode, the vGPU placement IDs are integer multiples of the placement size from zero to 1 fewer than the maximum vGPUs per GPU. For example, on a GPU with 40 GB of frame buffer, the maximum number of vGPUs with 10240 MB of frame buffer, which have a placement size of 10, is 4. The placement IDs for these vGPUs are as follows:

- 0 × 10 = 0
- 1 × 10 =10
- 2 × 10 = 20
- 3 × 10 = 30

9.3.1. vGPU Placements for GPUs with 96 GB of Frame Buffer

The placement region size depends on the number GPU instances on the GPU.

vGPU placement IDs for GPUs with 96 GB of frame buffer are the same in mixed-size mode and in equal-size mode.

Note

For MIG-backed vGPUs, the placement IDs and placement region are local to the GPU instance, **not** the physical GPU. Therefore, multiple vGPUs on different GPU instances on the same GPU can have the same placement ID.

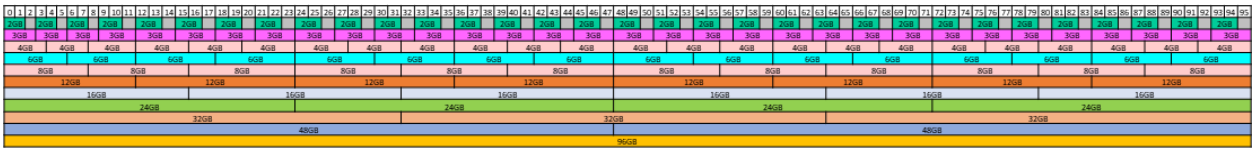
9.3.1.1 vGPU Placements on a Single-Instance GPU

Placement region size: 96

A single-instance GPU is a GPU that is not partitioned into multiple GPU instances.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
98304	96	1	1	0
49152	48	2	2	0, 48
32768	32	3	3	0, 32, 64
24576	24	4	4	0, 24, 48, 72
16384	16	6	6	0, 16, 32, 48, 64, 80
12288	12	8	8	0, 12, 24, 36, 48, 60, 72, 84
8192	8	12	12	0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88
6144	6	16	16	0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90
4096	4	24	24	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92
3072	3	32	32	0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93
2048	2	32	32	0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93

The following diagram shows the supported placements for each size of vGPU on a single-instance GPU with a total of 96 GB of frame buffer in mixed-size mode and in equal-size mode.



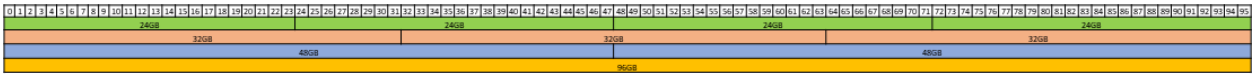
9.3.1.2 vGPU Placements on a Full-Partition GPU Instance (4g.96gb+gfx)

Placement region size: 96

A full partition GPU instance is assigned all of the GPU's frame buffer.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
98304	96	1	1	0
49152	48	2	2	0, 48
32768	32	3	3	0, 32, 64
24576	24	4	4	0, 24, 48, 72

The following diagram shows the supported placements for each size of vGPU on a full-partition GPU instance on a GPU with a total of 96 GB of frame buffer in mixed-size mode and in equal-size mode.



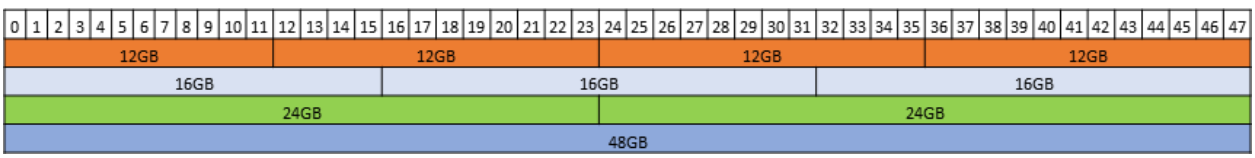
9.3.1.3 vGPU Placements on a Half-Partition GPU Instance (2g.48gb+gfx)

Placement region size: 48

A half- partition GPU instance is assigned half of the GPU's frame buffer.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
49152	48	1	1	0
24576	24	2	2	0, 24
16384	26	3	3	0, 16, 32
12288	12	4	4	0, 12, 24, 36

The following diagram shows the supported placements for each size of vGPU on a half-partition GPU instance on a GPU with a total of 96 GB of frame buffer in mixed-size mode and in equal-size mode.



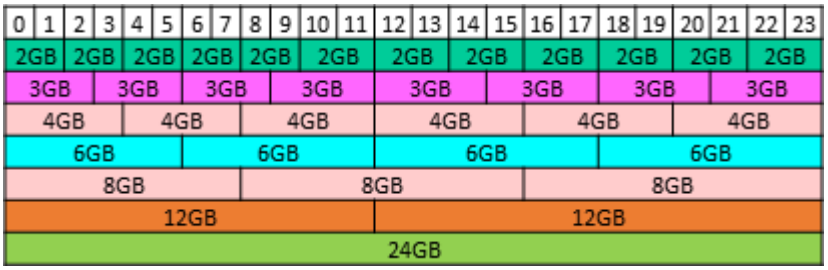
9.3.1.4 vGPU Placements on a Quarter-Partition GPU Instance (1g.24gb+gfx)

Placement region size: 48

A quarter- partition GPU instance is assigned one quarter of the GPU's frame buffer.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
24576	24	1	1	0
12288	12	2	2	0, 12
8192	8	3	3	0, 8, 16
6144	6	4	4	0, 6, 12, 18
4096	4	6	6	0, 4, 8, 12, 16, 20
3072	3	8	8	0, 3, 6, 9, 12, 15, 18, 21
2048	2	12	12	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22

The following diagram shows the supported placements for each size of vGPU on a quarter-partition GPU instance on a GPU with a total of 96 GB of frame buffer in mixed-size mode and in equal-size mode.

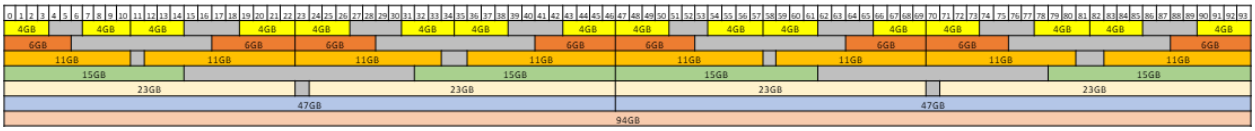


9.3.2. vGPU Placements for GPUs with 94 GB of Frame Buffer

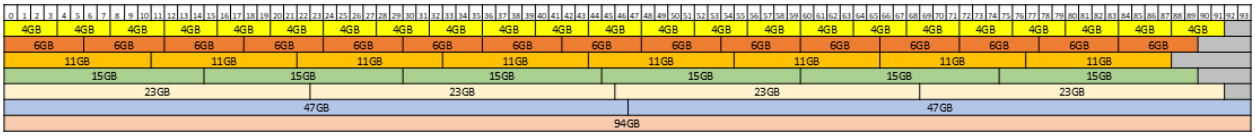
Placement region size: 94

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
96256	94	1	0	1	0
48128	47	2	0, 47	2	0, 47
23552	23	4	0, 23, 46, 79	4	0, 24, 47, 71
15360	15	6	0, 15, 30, 45, 60, 75	4	0, 32, 47, 79
11264	11	8	0, 11, 22, 33, 44, 55, 66, 77	8	0, 12, 23, 36, 47, 59, 70, 83
6,144	6	15	0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84	8	0, 17, 23, 41, 47, 64, 70, 88
4,096	4	23	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88	16	0, 7, 11, 19, 23, 31, 35, 43, 47, 54, 58, 66, 70, 78, 82, 90

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 94 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 94 GB of frame buffer in **equal-size mode**.

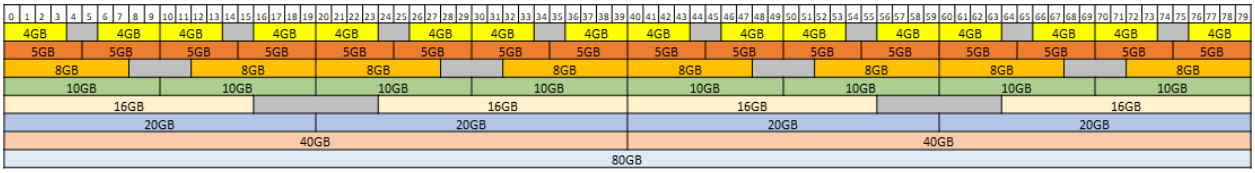


9.3.3. vGPU Placements for GPUs with 80 GB of Frame Buffer

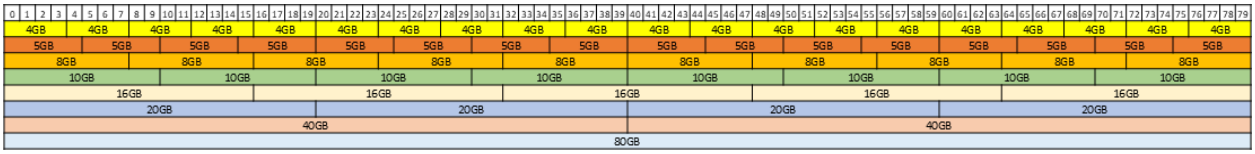
Placement region size: 80

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
81920	80	1	0	1	0
40960	40	2	0, 40	2	0, 40
20480	20	4	0, 20, 40, 60	4	0, 20, 40, 60
16384	16	5	0, 16, 32, 48, 64	4	0, 24, 40, 64
10240	10	8	0, 10, 20, 30, 40, 50, 60, 70	8	0, 10, 20, 30, 40, 50, 60, 70
8192	8	10	0, 8, 16, 24, 32, 40, 48, 56, 64, 72	8	0, 12, 20, 32, 40, 52, 60, 72
5120	5	16	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75	16	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75
4096	4	20	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 66, 70, 76	16	0, 6, 10, 16, 20, 26, 30, 36, 40, 46, 50, 56, 60, 66, 70, 76

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 80 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 80 GB of frame buffer in **equal-size mode**.



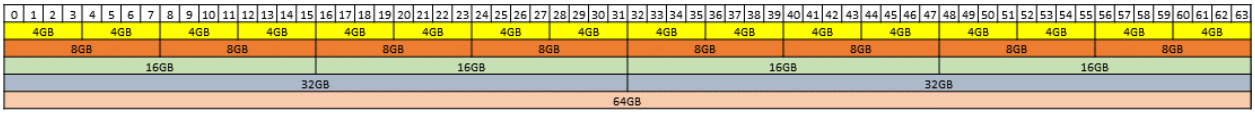
9.3.4. vGPU Placements for GPUs with 64 GB of Frame Buffer

Placement region size: 64

vGPU placement IDs for GPUs with 64 GB of frame buffer are the same in mixed-size mode and in equal-size mode.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs
65536	64	1	1	0
32768	32	2	2	0, 32
16384	16	4	4	0, 16, 32, 48
8192	8	8	8	0, 8, 16, 24, 32, 40, 48, 56
4096	4	16	16	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 46, 60

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 64 GB of frame buffer in mixed-size mode and in equal-size mode.



9.3.5. vGPU Placements for GPUs with 48 GB of Frame Buffer

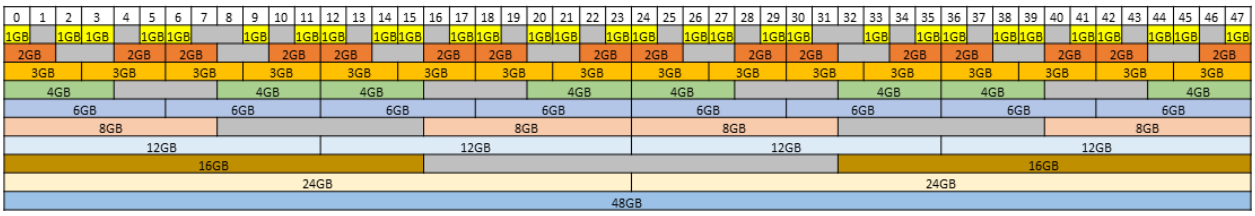
Placement region size: 48

Note

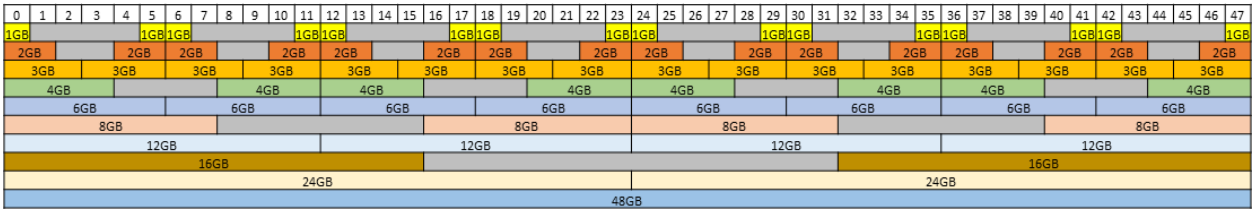
When in mixed-size mode, the maximum number of vGPUs with 1024 MB of frame buffer allowed on GPUs based on the Ada Lovelace GPU architecture is lower than for other GPU architectures. As a result, the supported placement IDs for these vGPUs on GPUs based on the Ada Lovelace GPU architecture are different than for other GPU architectures.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
49152	48	1	0	1	0
24576	24	2	0, 24	2	0, 24
16384	16	3	0, 16, 32	2	0, 32
12288	12	4	0, 12, 24, 36	4	0, 12, 24, 36
8192	8	6	0, 8, 16, 24, 32, 40	4	0, 16, 24, 40
6144	6	8	0, 6, 12, 18, 24, 30, 36, 42	8	0, 6, 12, 18, 24, 30, 36, 42
4096	4	12	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44	8	0, 8, 12, 20, 24, 32, 36, 44
3072	3	16	0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45	16	0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45
2048	2	24	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46	16	0, 4, 6, 10, 12, 16, 18, 22, 24, 28, 30, 34, 36, 40, 42, 46
1024	1	32	0 through 31	GPU architectures **except** Ada Lovelace: 30	GPU architectures **except** Ada Lovelace: 0, 2, 3, 5, 6, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23, 24, 26, 27, 29, 30, 33, 35, 36, 38, 39, 41, 42, 44, 45, 47
				Ada Lovelace GPU architecture: 16	Ada Lovelace GPU architecture: 0, 5, 6, 11, 12, 17, 18, 23, 24, 29, 30, 35, 36, 41, 42, 47

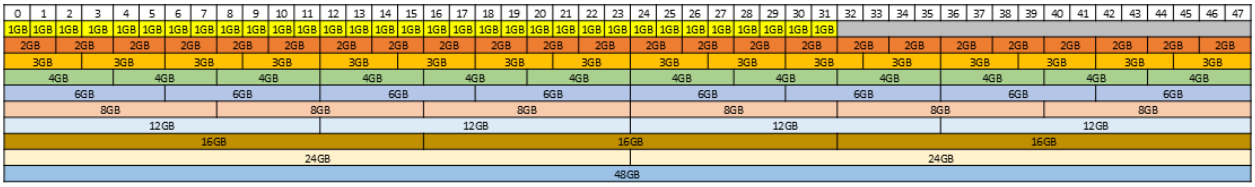
The following diagram shows the supported placements for each size of vGPU on a GPU based on a GPU architecture **except** Ada Lovelace with a total of 48 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU based on the **Ada Lovelace** GPU architecture with a total of 48 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 48 GB of frame buffer in **equal-size mode**.

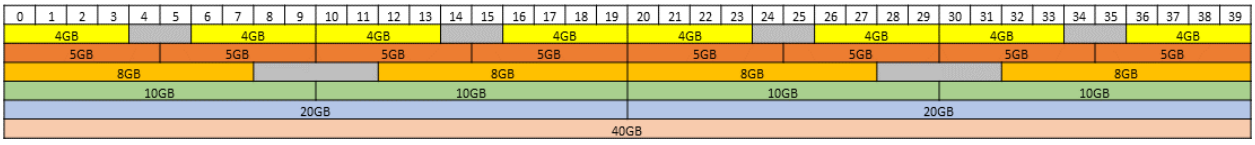


9.3.6. vGPU Placements for GPUs with 40 GB of Frame Buffer

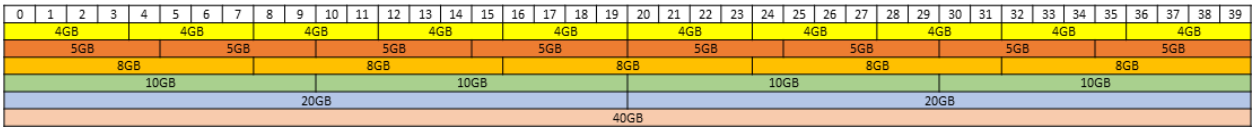
Placement region size: 40

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
40960	40	1	0	1	0
20480	20	2	0, 20	2	0, 20
10240	10	4	0, 10, 20, 30	4	0, 10, 20, 30
8192	8	5	0, 8, 16, 24, 32	4	0, 12, 20, 32
5120	5	8	0, 5, 10, 15, 20, 25, 30, 35	8	0, 5, 10, 15, 20, 25, 30, 35
4096	4	10	0, 4, 8, 12, 16, 20, 24, 28, 32, 36	8	0, 6, 10, 16, 20, 26, 30, 36

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 40 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 40 GB of frame buffer in **equal-size mode**.

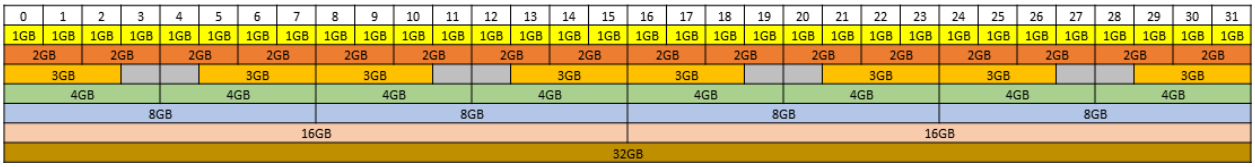


9.3.7. vGPU Placements for GPUs with 32 GB of Frame Buffer

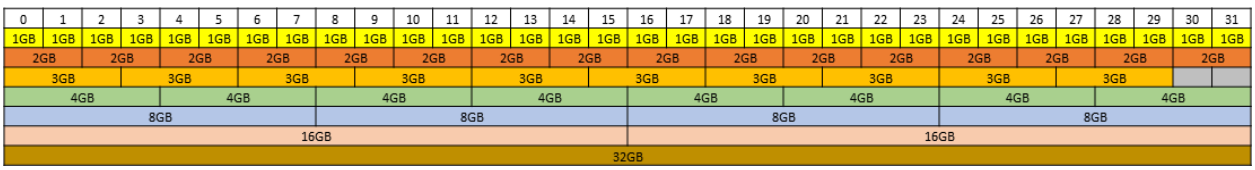
Placement region size: 32

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
32768	32	1	0	1	0
16384	16	2	0, 16	2	0, 16
8192	8	4	0, 8, 16, 24	4	0, 8, 16, 24
4096	4	8	0, 4, 8, 12, 16, 20, 24, 28	8	0, 4, 8, 12, 16, 20, 24, 28
3072	3	10	0, 3, 6, 9, 12, 15, 18, 21, 24, 27	8	0, 5, 8, 13, 16, 21, 24, 29
2048	2	16	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30	16	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
1024	1	32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31	32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 32 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 32 GB of frame buffer in **equal-size mode**.

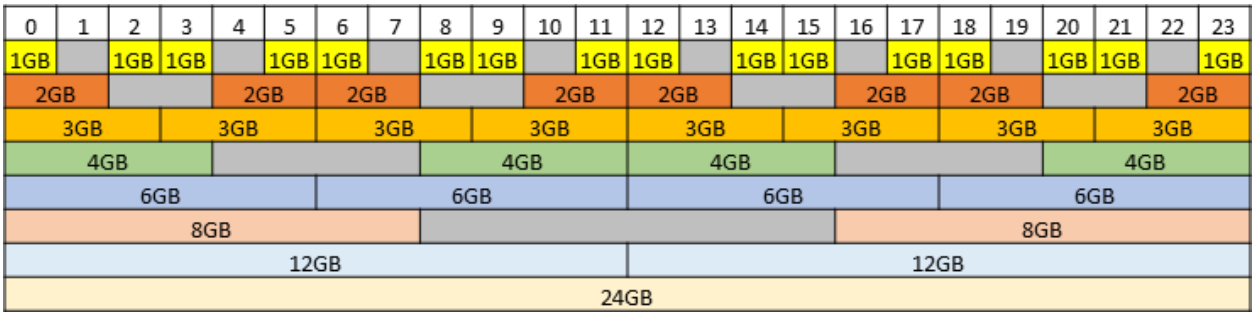


9.3.8. vGPU Placements for GPUs with 24 GB of Frame Buffer

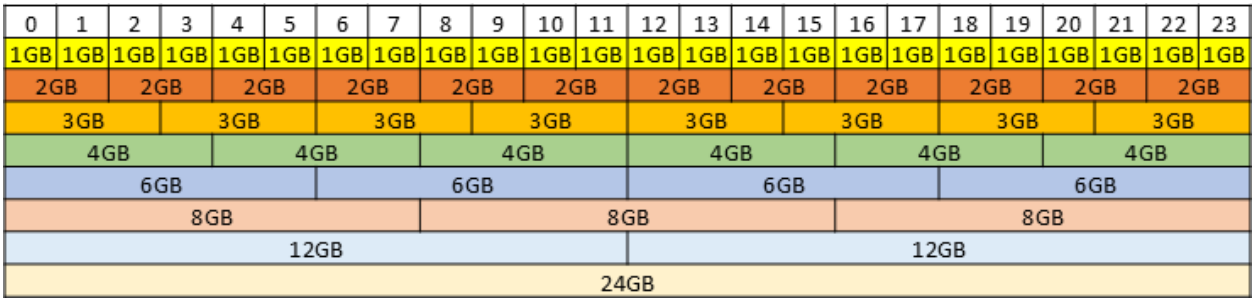
Placement region size: 24

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
24576	24	1	0	1	0
12288	12	2	0, 12	2	0, 12
8192	8	3	0, 8, 16	2	0, 16
6144	6	4	0, 6, 12, 18	4	0, 6, 12, 18
4096	4	6	0, 4, 8, 12, 16, 20	4	0, 8, 12, 20
3072	3	8	0, 3, 6, 9, 12, 15, 18, 21	8	0, 3, 6, 9, 12, 15, 18, 21
2048	2	12	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22	8	0, 4, 6, 10, 12, 16, 18, 22
1024	1	24	0 through 23	16	0, 2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18, 20, 21, 23

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 24 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 24 GB of frame buffer in **equal-size mode**.

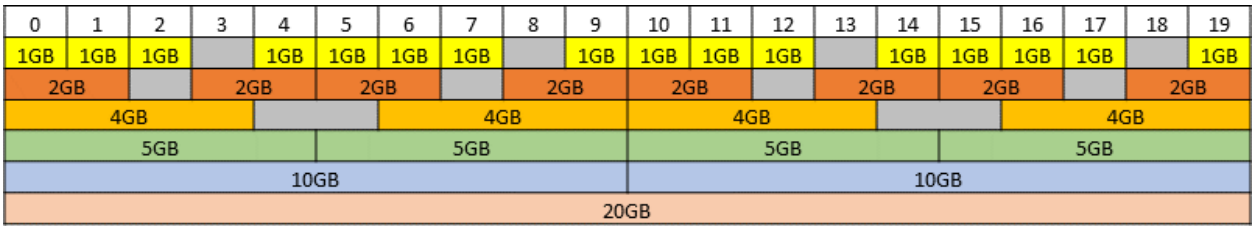


9.3.9. vGPU Placements for GPUs with 20 GB of Frame Buffer

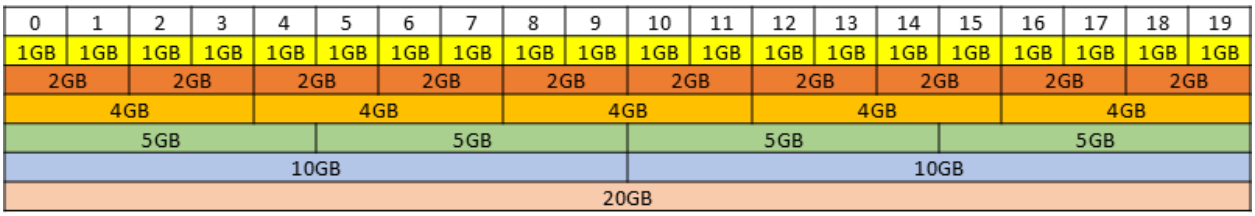
Placement region size: 20

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
20480	20	1	0	1	0
10240	10	2	0, 10	2	0, 10
5120	5	4	0, 5, 10, 15	4	0, 5, 10, 15
4096	4	5	0, 4, 8, 12, 16	4	0, 6, 10, 16
2048	2	10	0, 2, 4, 6, 8, 10, 12, 14, 16, 18	8	0, 3, 5, 8, 10, 13, 15, 18
1024	1	20	0 through 19	16	0, 1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17, 19

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 20 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 20 GB of frame buffer in **equal-size mode**.



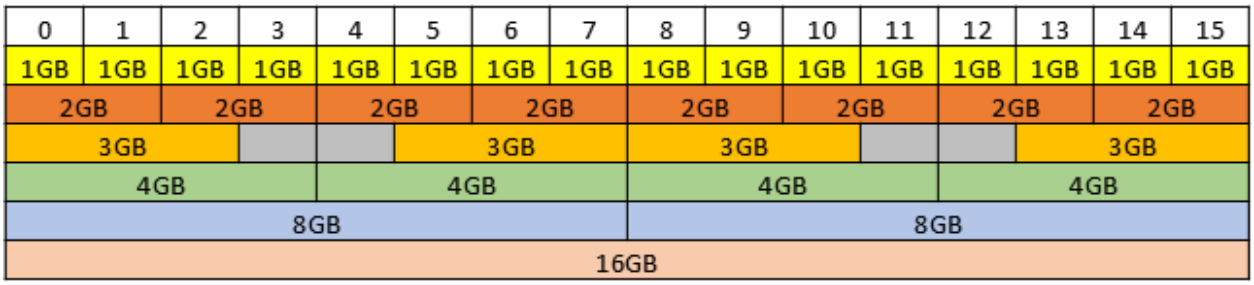
9.3.10. vGPU Placements for GPUs with 16 GB of Frame Buffer

Placement region size: 16

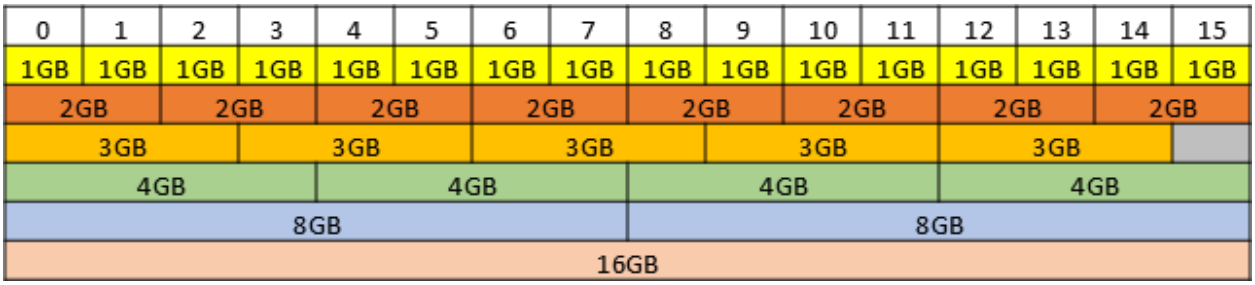
vGPU placement IDs for GPUs with 16 GB of frame buffer are the same in mixed-size mode and in equal-size mode.

vGPU Size (MB of Frame Buffer)	Placement Size	Maximum vGPUs per GPU in Equal-Size Mode	Supported Placement IDs in Equal-Size Mode	Maximum vGPUs per GPU in Mixed-Size Mode	Supported Placement IDs in Mixed-Size Mode
16384	16	1	0	1	0
8192	8	2	0, 8	2	0, 8
4096	4	4	0, 4, 8, 12	4	0, 4, 8, 12
3072	3	5	0, 3, 6, 9, 12	4	0, 5, 8, 13
2048	2	8	0, 2, 4, 6, 8, 10, 12, 14	8	0, 2, 4, 6, 8, 10, 12, 14
1024	1	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 16 GB of frame buffer in **mixed-size mode**.



The following diagram shows the supported placements for each size of vGPU on a GPU with a total of 16 GB of frame buffer in **equal-size mode**.



Chapter 10. Allocation Strategies

Strategies for allocating physical hardware resources to VMs and vGPUs can improve the performance of VMs running with NVIDIA vGPU. They include strategies for pinning VM CPU cores to physical cores on Non-Uniform Memory Access (NUMA) platforms, allocating VMs to CPUs, and allocating vGPUs to physical GPUs. These allocation strategies are supported by XenServer and VMware vSphere.

10.1. 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 10.1:

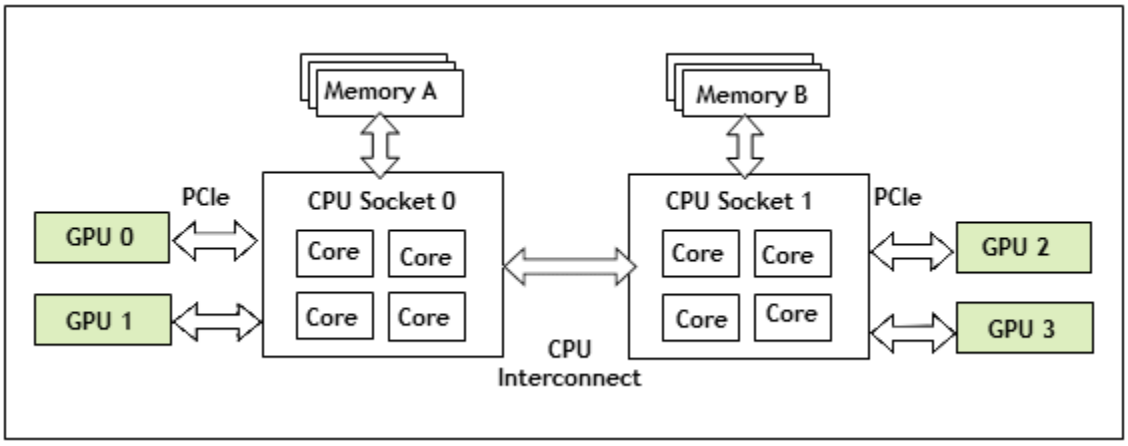


Figure 10.1: A NUMA Server Platform

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.

10.1.1. Obtaining Best Performance on a NUMA Platform with XenServer

To obtain best performance on a NUMA platform, place VMs on the same CPU socket as the physical GPU that hosts the VM's vGPU. On supported XenServer releases, use the Citrix NUMA feature for that placement.

See *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.

10.1.2. Obtaining Best Performance on a NUMA Platform with VMware vSphere ESXi

For some types of workloads or system configurations, you can optimize performance by specifying the placement of VMs explicitly. For best performance, pin each VM to the NUMA node to which the physical GPU hosting the VM's vGPU is attached.

The following types of workloads and system configurations benefit from explicit placement of VMs:

- Memory-intensive workloads, such as an in-memory database or an HPC application with a large data set
- A hypervisor host configured with a small number of virtual machines

VMware vSphere ESXi provides the **NUMA Node Affinity** option for specifying the placement of VMs explicitly. For general information about the options in VMware vSphere ESXi for NUMA placement, see Specifying NUMA Controls in the VMware documentation.

Before setting the **NUMA Node Affinity** option, run the `nvidia-smi topo -m` command in the ESXi host shell to determine the NUMA affinity of the GPU device.

After determining the NUMA affinity of the GPU device, set the **NUMA Node Affinity** option as explained in Associate Virtual Machines with Specified NUMA Nodes in the VMware documentation.

10.2. 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 and VMware vSphere ESXi use a different GPU allocation policy by default.

- XenServer creates GPU groups with a default allocation policy of *depth-first*.

  See *Modifying GPU Allocation Policy on XenServer* for details on switching the allocation policy to breadth-first.

- For time-sliced vGPUs on a GPU in equal-size mode, VMware vSphere ESXi creates GPU groups with a default allocation policy of *breadth-first*.

  See *Modifying GPU Allocation Policy on VMware vSphere* for details on switching the allocation policy to depth-first.

  The GPU allocation policy does not apply to MIG-backed vGPUs or to a GPU in mixed-size mode.

Note

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.

Chapter 11. Configuring x11vnc for Checking the GPU in a Linux Server

x11vnc is a virtual network computing (VNC) server that provides remote access to an existing X session with any VNC viewer. You can use x11vnc to confirm that the NVIDIA GPU in a Linux server to which no display devices are directly connected is working as expected. Examples of servers to which no display devices are directly connected include a VM that is configured with NVIDIA vGPU, a VM that is configured with a pass-through GPU, and a headless physical host in a bare-metal deployment.

Before you begin, ensure that the following prerequisites are met:

- The NVIDIA vGPU software software graphics driver for Linux is installed on the server.
- A secure shell (SSH) client is installed on your local system:
  - On Windows, you must use a third-party SSH client such as PuTTY.
  - On Linux, you can run the SSH client that is included with the OS from a shell or terminal window.

Configuring x11vnc involves following the sequence of instructions in these sections:

1. *Configuring the Xorg Server on the Linux Server*
2. *Installing and Configuring x11vnc on the Linux Server*
3. *Using a VNC Client to Connect to the Linux Server*

After connecting to the server, you can use NVIDIA X Server Settings to confirm that the NVIDIA GPU is working as expected.

11.1. Configuring the Xorg Server on the Linux Server

You must configure the Xorg server to specify which GPU or vGPU is to be used by the Xorg server if multiple GPUs are installed in your server and to allow the Xorg server to start even if no connected display devices can be detected.

1. Log in to the Linux server.
2. Determine the PCI bus identifier of the GPUs or vGPUs on the server.

```
# nvidia-xconfig --query-gpu-info
Number of GPUs: 1

GPU #0:
  Name      : GRID T4-2Q
  UUID      : GPU-ea80de2d-1dd8-11b2-8305-c955f034e718
  PCI BusID : PCI:2:2:0

  Number of Display Devices: 0
```

3. In a plain text editor, edit the `/etc/X11/xorg.conf` file to specify the GPU is to be used by the Xorg server and allow the Xorg server to start even if no connected display devices can be detected.
   1. In the `Device` section, add the PCI bus identifier of GPU to be used by the Xorg server.

```
Section "Device"
    Identifier     "Device0"
    Driver         "nvidia"
    VendorName     "NVIDIA Corporation"
    BusID          "PCI:2:2:0"
EndSection
```

> **Note**
>
> The three numbers in the `PCI BusID` obtained by `nvidia-xconfig` in the previous step are hexadecimal numbers. They must be converted to decimal numbers in the PCI bus identifier in the `Device` section. For example, if the PCI bus identifier obtained in the previous step is `PCI:A:10:0`, it must be specified as `PCI:10:16:0` in the PCI bus identifier in the `Device` section.

   2. In the `Screen` section, ensure that the `AllowEmptyInitialConfiguration` option is set to `True`.

```
Section "Screen"
    Identifier     "Screen0"
    Device         "Device0"
    Option         "AllowEmptyInitialConfiguration" "True"
EndSection
```

4. Restart the Xorg server in one of the following ways:
   - Restart the server.
   - Run the `startx` command.
   - If the Linux server is in run level 3, run the `init 5` command to run the server in graphical mode.
5. Confirm that the Xorg server is running.

```
# ps -ef | grep X
```

On Ubuntu, this command displays output similar to the following example.

```
root     16500 16499  2 03:01 tty2     00:00:00 /usr/lib/xorg/Xorg -
↪nolisten tcp :0 -auth /tmp/serverauth.s7CE4mMeIz
root      1140  1126  0 18:46 tty1     00:00:00 /usr/lib/xorg/Xorg vt1 -
↪displayfd 3 -auth /run/user/121/gdm/Xauthority -background none -
↪noreset -keeptty -verbose 3
root     17011 17108  0 18:50 pts/0    00:00:00 grep --color=auto X
```

On Red Hat Enterprise Linux, this command displays output similar to the following example.

```
root      5285  5181  0 16:29 pts/0    00:00:00 grep --color=auto X
root      5880     1  0 Jun13 ?        00:00:00 /usr/bin/abrt-watch-log -
↪F Backtrace /var/log/Xorg.0.log -- /usr/bin/abrt-dump-xorg -xD
root      7039  6289  0 Jun13 tty1     00:00:03 /usr/bin/X :0 -background
↪none -noreset -audit 4 -verbose -auth /run/gdm/auth-for-gdm-vr4MFC/
↪database -seat seat0 vt1
```

11.2. Installing and Configuring x11vnc on the Linux Server

Unlike other VNC servers, such as TigerVNC or Vino, x11vnc does not create an extra X session for remote access. Instead, x11vnc provides remote access to the existing X session on the Linux server.

1. Install the required x11vnc package and any dependent packages.
   - For distributions based on Red Hat, use the yum package manager to install the x11vnc package.

     ```
     # yum install x11vnc
     ```

   - For distributions based on Debian, use the apt package manager to install the x11vnc package.

     ```
     # sudo apt install x11vnc
     ```

   - For SuSE Linux distributions, install x11vnc from the x11vnc openSUSE Software page.
2. Get the display numbers of the servers for the Xorg server.

   ```
   # cat /proc/*/environ 2>/dev/null | tr '\0' '\n' | grep '^DISPLAY=:' |
   ↪uniq
   DISPLAY=:0
   DISPLAY=:100
   ```

3. Start the x11vnc server, specifying the display number to use.

   The following example starts the x11vnc server on display 0 on a Linux server that is running the Gnome desktop.

   ```
   # x11vnc -display :0 -auth /run/user/121/gdm/Xauthority -forever \
   -shared -ncache -bg
   ```

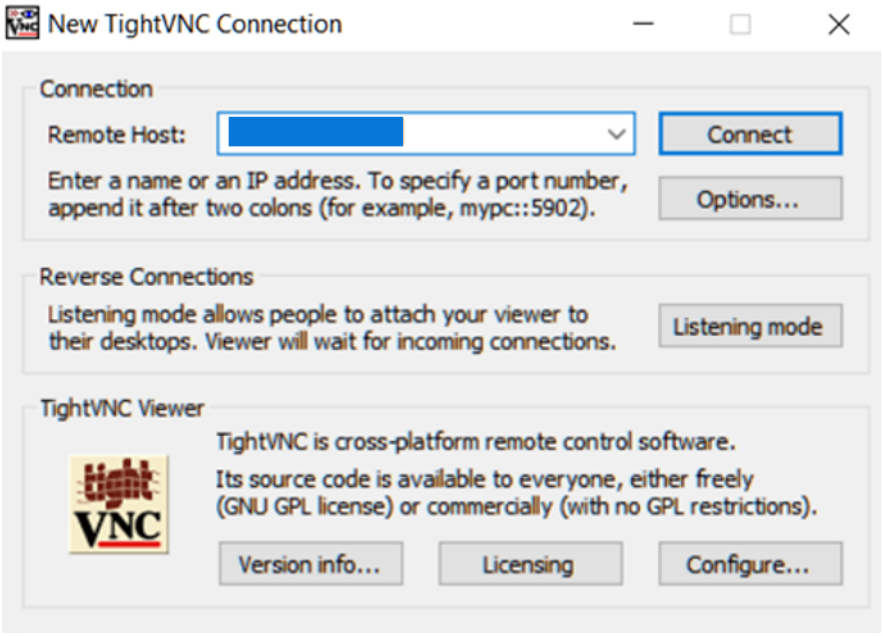
> **Note**
>
> If you are using a C-series vGPU, omit the `-ncache` option.

The `x11vnc` server starts on display `<hostname>:0`, for example, `my-linux-host:0`.

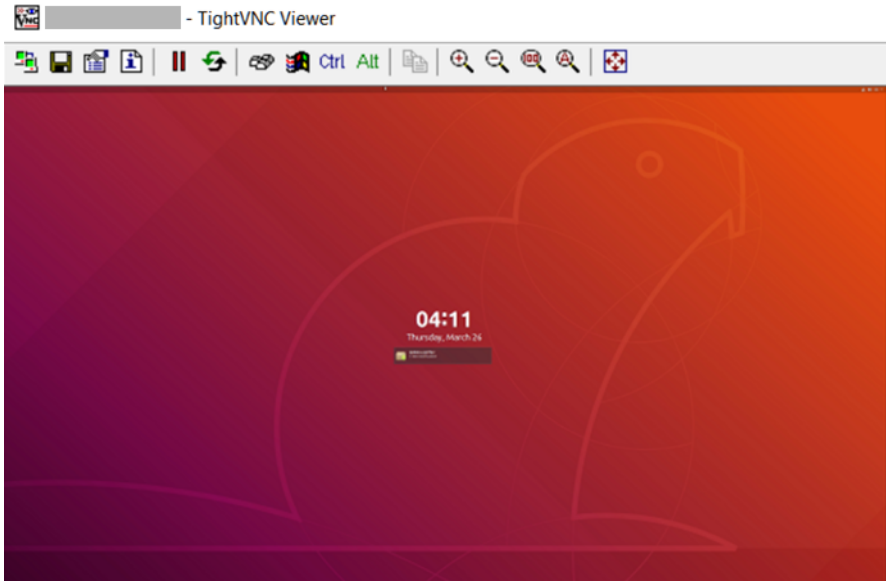
```
26/03/20200 04:23:13
The VNC desktop is:      my-linux-host:0
PORT=5900
```

11.3. Using a VNC Client to Connect to the Linux Server

1. On your client computer, install a VNC client such as TightVNC.
2. Start the VNC client and connect to the Linux server.



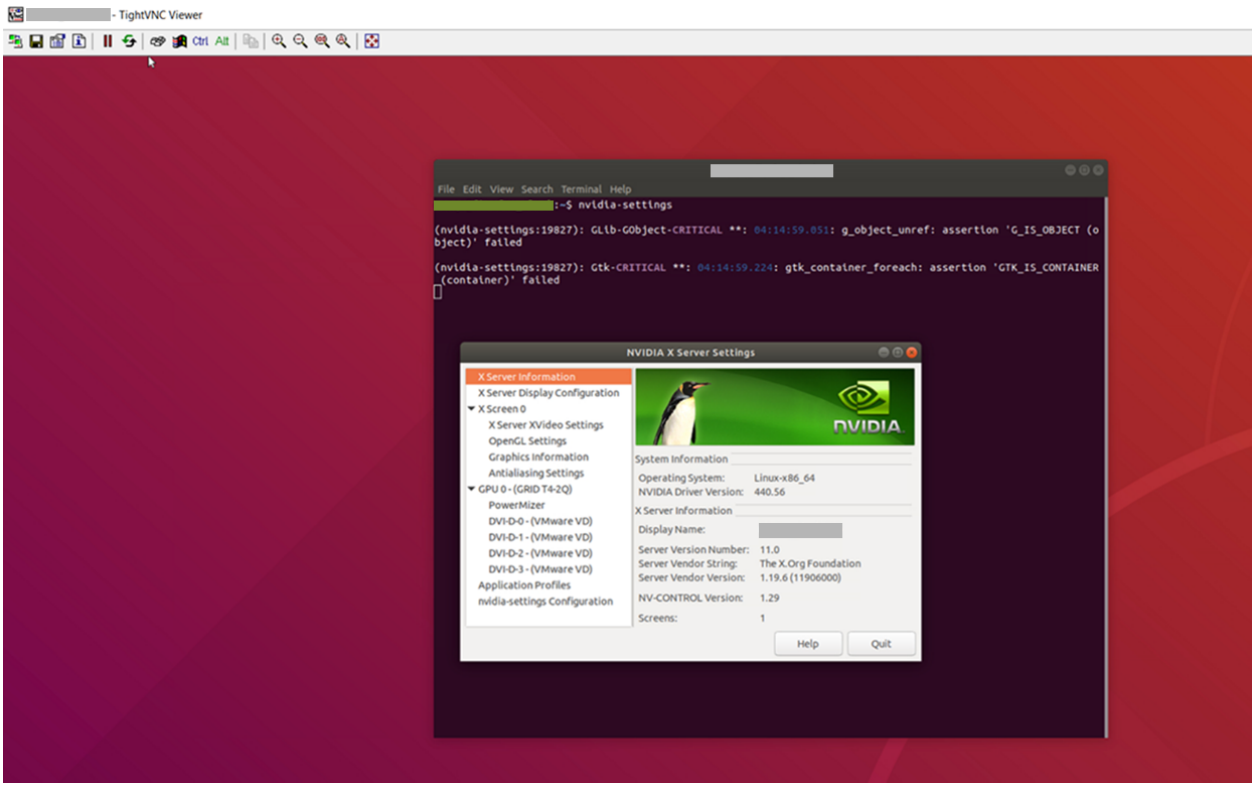
The X session on the server opens in the VNC client.



Troubleshooting: If your VNC client cannot connect to the server, change permissions **on the Linux server** as follows:

1. Allow the VNC client to connect to the server by making one of the following changes:
   - Disable the firewall and the `iptables` service.
   - Open the VNC port in the firewall.
2. Ensure that permissive mode is enabled for Security Enhanced Linux (SELinux).

After connecting to the server, you can use NVIDIA X Server Settings to confirm that the NVIDIA GPU is working as expected.

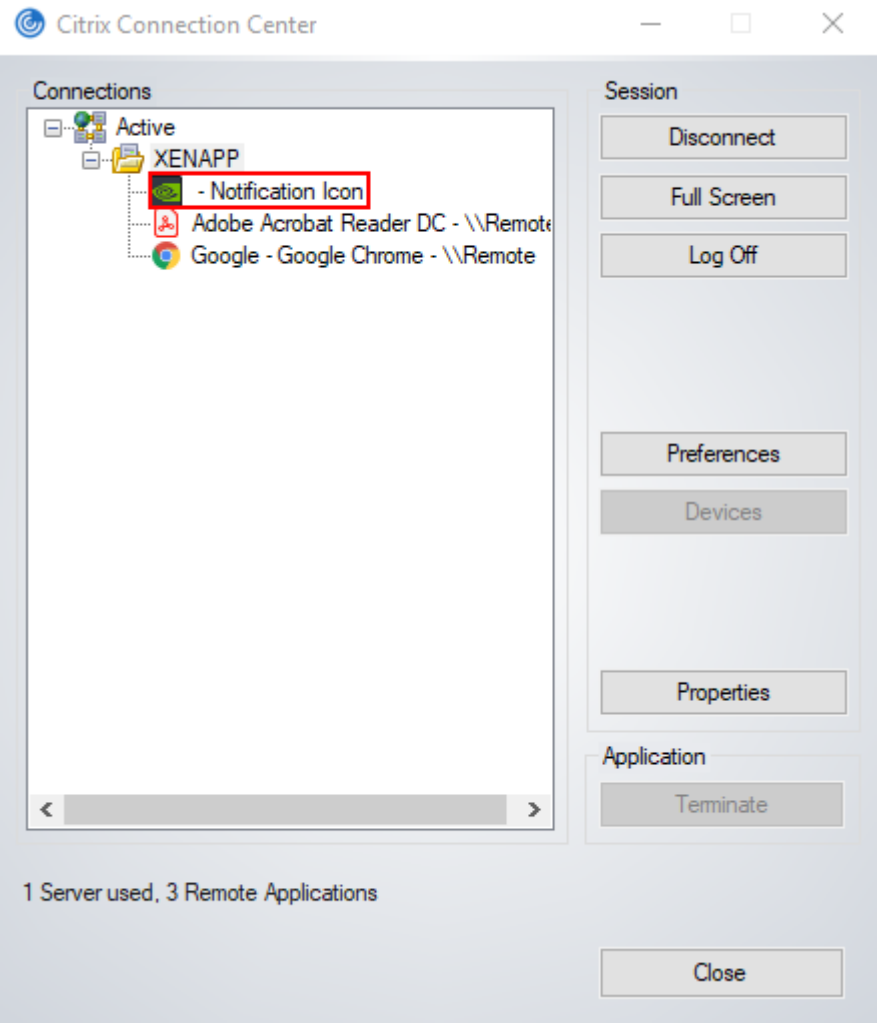

TightVNC Viewer
Ctrl Alt
File Edit View Search Terminal Help
:~$ nvidia-settings
(nvidia-settings:19827): GLib-GObject-CRITICAL **: 04:14:59.051: g_object_unref: assertion 'G_IS_OBJECT (object)' failed
(nvidia-settings:19827): Gtk-CRITICAL **: 04:14:59.224: gtk_container_foreach: assertion 'GTK_IS_CONTAINER (container)' failed
NVIDIA X Server Settings
X Server Information
X Server Display Configuration
X Screen 0
X Server XVideo Settings
OpenGL Settings
Graphics Information
Antialiasing Settings
GPU 0 - (GRID T4-2Q)
PowerMizer
DVI-D-0 - (VMware VD)
DVI-D-1 - (VMware VD)
DVI-D-2 - (VMware VD)
DVI-D-3 - (VMware VD)
Application Profiles
nvidia-settings Configuration
NVIDIA
System Information
Operating System: Linux-x86_64
NVIDIA Driver Version: 440.56
X Server Information
Display Name:
Server Version Number: 11.0
Server Vendor String: The X.Org Foundation
Server Vendor Version: 1.19.6 (11906000)
NV-CONTROL Version: 1.29
Screens: 1
Help
Quit

Chapter 12. Disabling NVIDIA Notification Icon for Citrix Published Application User Sessions

By default on Windows Server operating systems, the NVIDIA Notification Icon application is started with every Citrix Published Application user session. This application might prevent the Citrix Published Application user session from being logged off even after the user has quit all other applications.

The NVIDIA Notification Icon application appears in Citrix Connection Center on the endpoint client that is running Citrix Receiver or Citrix Workspace.

The following image shows the NVIDIA Notification Icon in Citrix Connection Center for a user session in which the Adobe Acrobat Reader DC and Google Chrome applications are published.



Administrators can disable the NVIDIA Notification Icon application for all users' sessions as explained in *Disabling NVIDIA Notification Icon for All Users' Citrix Published Application Sessions*.

Individual users can disable the NVIDIA Notification Icon application for their own sessions as explained in *Disabling NVIDIA Notification Icon for your Citrix Published Application User Sessions*.

Note

If an administrator has enabled the NVIDIA Notification Icon application for the administrator's own session, the application is enabled for all users' sessions, even the sessions of users who have previously disabled the application.

12.1. Disabling NVIDIA Notification Icon for All Users' Citrix Published Application Sessions

Administrators can set a registry key to disable the NVIDIA Notification Icon application for all users' Citrix Published Application sessions on a VM. To ensure that the NVIDIA Notification Icon application is disabled on any virtual delivery agent (VDA) that is created from a master image, set this key in the master image.

Perform this task from the VM on which the Citrix Published Application sessions will be created.

Before you begin, ensure that the NVIDIA vGPU software graphics driver is installed in the VM.

1. Set the system-level `StartOnLogin` Windows registry key to 0.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nvlddmkm\NvTray]
Value: "StartOnLogin"
Type: DWORD
Data: 00000000
```

   The data value 0 disables the NVIDIA Notification Icon, and the data value 1 enables it.

2. Restart the VM.

   You must restart the VM to ensure that the registry key is set before the NVIDIA service in the user session starts.

12.2. Disabling NVIDIA Notification Icon for your Citrix Published Application User Sessions

Individual users can disable the NVIDIA Notification Icon for their own Citrix Published Application sessions.

Before you begin, ensure that you are logged on to a Citrix Published Application session.

1. Set the current user's `StartOnLogin` Windows registry key to 0.

```
[HKEY_CURRENT_USER\SOFTWARE\NVIDIA Corporation\NvTray\]
Value: "StartOnLogin"
Type: DWORD
Data: 00000000
```

   The data value 0 disables the NVIDIA Notification Icon, and the data value 1 enables it.

2. Log off and log on again or restart the VM.

   You must log on and log off again or restart the VM to ensure that the registry key is set before the NVIDIA service in the user session starts.

Chapter 13. XenServer Basics

To install and configure NVIDIA vGPU software and optimize XenServer operation with vGPU, some basic operations on XenServer are needed.

13.1. Opening a dom0 shell

Most configuration commands must be run in a command shell in the XenServer dom0 domain. You can open a shell in the XenServer dom0 domain in any of the following ways:

- Using the console window in XenCenter
- Using a standalone secure shell (SSH) client

13.1.1. 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 console.
3. Press **Enter** to start a shell prompt.

13.1.2. 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.

13.2. Copying files to dom0

You can copy files to and from XenServer dom0 by mounting a CIFS/SMB file 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.

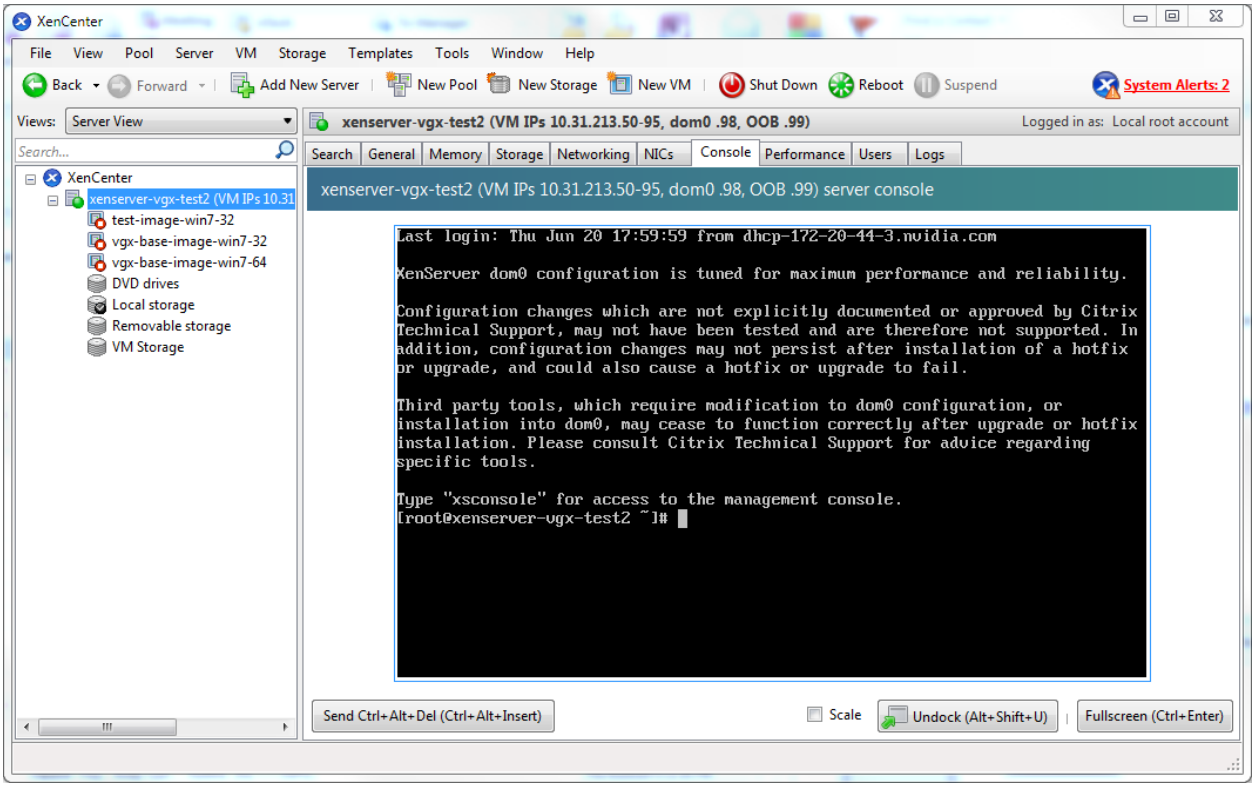


Figure 13.1: Connecting to the dom0 shell by using XenCenter

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-NVIDIA-vGPU-
↪CitrixHypervisor-8.2-595.112.08.x86_64.rpm .
[root@xenserver ~]#
```

13.3. 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

13.3.1. 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): vgx-base-image-wind-64
    power-state ( RO): halted


uuid ( RO)           : fa3d15c7-7e88-4886-c36a-cdb23ed8e275
     name-label ( RW): test-image-wind-64
    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-eee1-327f-02e4f40714fc
     name-label ( RW): vgx-base-image-wind-64
    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-wind-64
uuid ( RO)           : fa3d15c7-7e88-4886-c36a-cdb23ed8e275
     name-label ( RW): test-image-wind-64
    power-state ( RO): halted
```

13.3.2. 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.

13.4. 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.

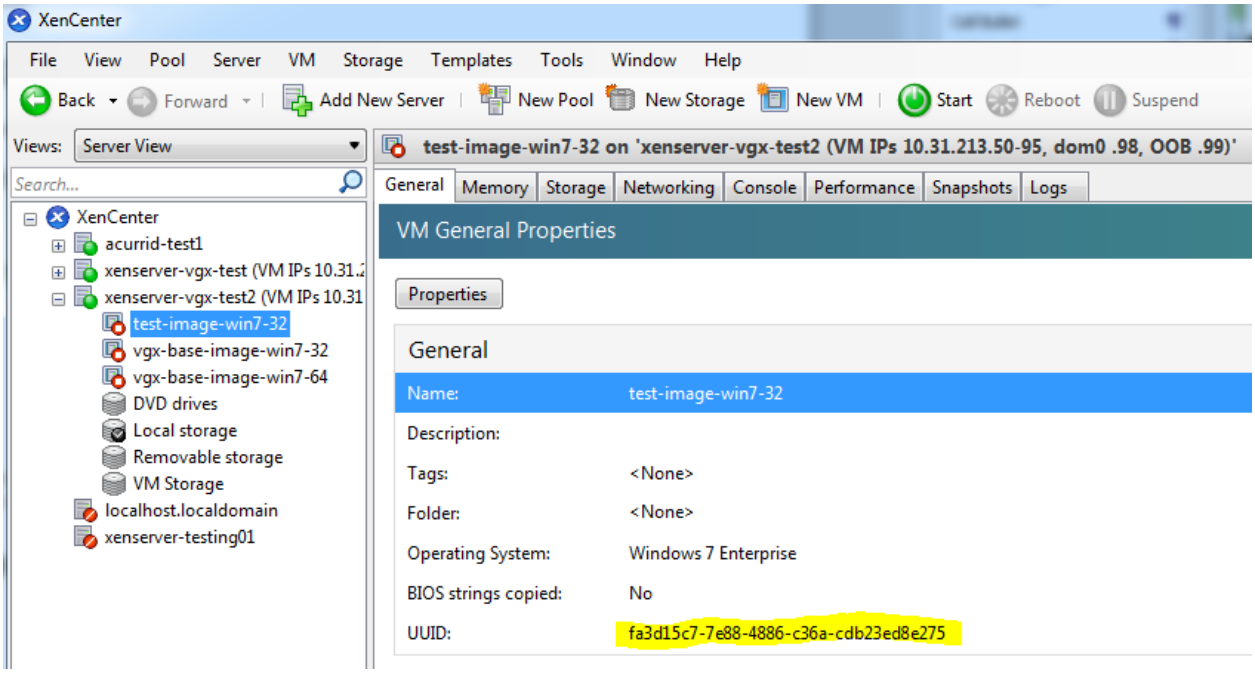


Figure 13.2: Using XenCenter to determine a VM's UUID

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*.

13.5. 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 for VMs to run on the same CPU socket as the physical GPU to which they are connected.

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.

Chapter 14. XenServer vGPU Management

You can perform XenServer advanced vGPU management techniques by using XenCenter and by using `xe` command line operations.

14.1. 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.

14.1.1. 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.

14.1.1.1 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:

```
[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]
```

(continues on next page)

(continued from previous page)

```
    gpu-group-uuid ( RW): d32560f2-2158-42f9-d201-511691e1cb2b
[root@xenserver ~]#
```

14.1.1.2 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-bd97-3aee97b2123a
            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;
↪16326fcb-543f-4473-a4ae-2d30516a2779; 0f9fc39a-0758-43c8-88cc-54c8491aa4d4;
↪cecb2033-3b4a-437c-a0c0-c9dfdb692d9b; 095d8939-5f84-405d-a39a-684738f9b957;
↪56c335be-4036-4a38-816c-c246a60556ac; ef0a94fd-2230-4fd4-aee0-d6d3f6ced4ef;
↪11615f73-47b8-4494-806e-2a7b5e1d7bea; dbd8f2ac-f548-4c40-804b-9133cfda8090;
↪a33189f1-1417-4593-aa7d-978c4f25b953; 3f437337-3682-4897-a7ba-6334519f4c19;
↪99900aab-42b0-4cc4-8832-560ff6b60231
         enabled-VGPU-types (SRW): 5b9acd25-06fa-43e1-8b53-c35bceb8515c;
↪16326fcb-543f-4473-a4ae-2d30516a2779; 0f9fc39a-0758-43c8-88cc-54c8491aa4d4;
↪cecb2033-3b4a-437c-a0c0-c9dfdb692d9b; 095d8939-5f84-405d-a39a-684738f9b957;
↪56c335be-4036-4a38-816c-c246a60556ac; ef0a94fd-2230-4fd4-aee0-d6d3f6ced4ef;
↪11615f73-47b8-4494-806e-2a7b5e1d7bea; dbd8f2ac-f548-4c40-804b-9133cfda8090;
↪a33189f1-1417-4593-aa7d-978c4f25b953; 3f437337-3682-4897-a7ba-6334519f4c19;
↪99900aab-42b0-4cc4-8832-560ff6b60231
             resident-VGPUs ( RO):
[root@xenserver ~]#
```

14.1.1.3 Viewing physical GPUs in XenCenter

To view physical GPUs in XenCenter, click on the server's **GPU** tab:

14.1.2. 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.

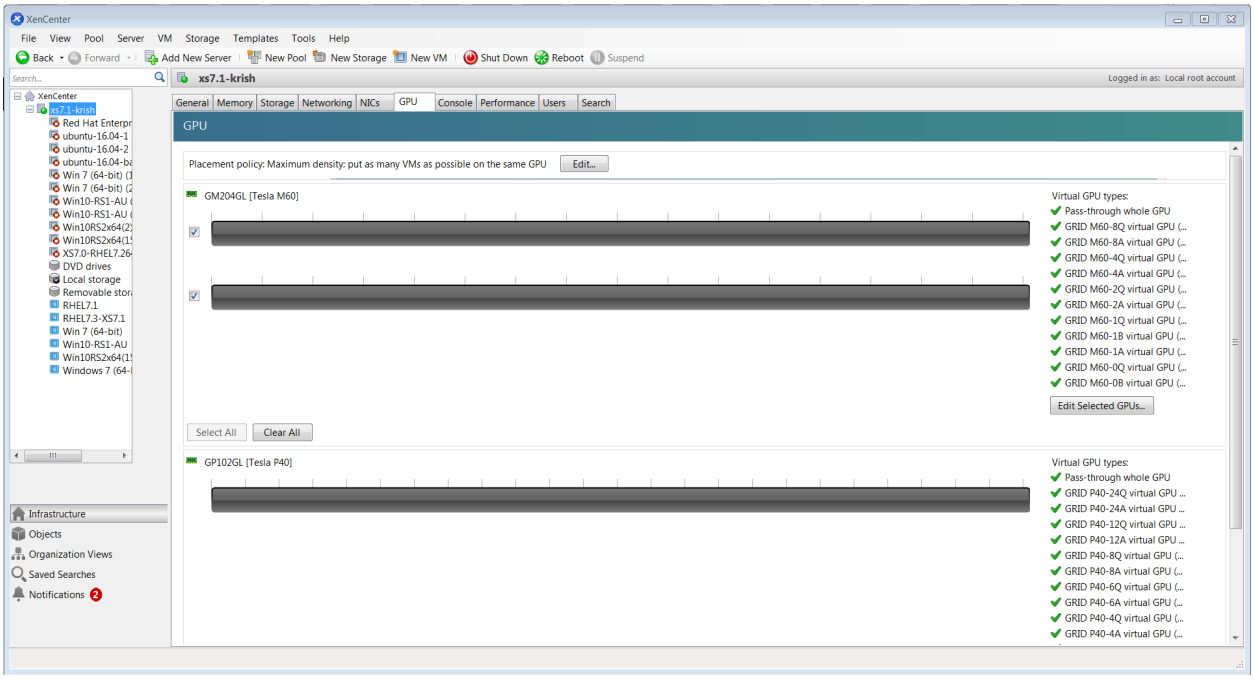


Figure 14.1: Physical GPU display in XenCenter

14.1.2.1 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-6347cfb03e93
       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
```

(continues on next page)

(continued from previous page)

```
       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):
        model-name ( RO): passthrough
         max-heads ( RO): 0
    max-resolution ( RO): 0x0


uuid ( RO)             : af121387-0b58-498a-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-b0fb-80165699229e
       vendor-name ( RO): NVIDIA Corporation
        model-name ( RO): GRID P100-4Q
         max-heads ( RO): 4
    max-resolution ( RO): 4096x2160


uuid ( RO)             : 99900aab-42b0-4cc4-8832-560ff6b60231
       vendor-name ( RO): NVIDIA Corporation
        model-name ( RO): GRID M60-1Q
         max-heads ( RO): 2
    max-resolution ( RO): 4096x2160


uuid ( RO)             : 0f9fc39a-0758-43c8-88cc-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-9de8-a7536aa1a838
       vendor-name ( RO): NVIDIA Corporation
        model-name ( RO): GRID P40-24A
         max-heads ( RO): 1
```

(continues on next page)

(continued from previous page)

```
    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)             : a33189f1-1417-4593-aa7d-978c4f25b953
       vendor-name ( RO): NVIDIA Corporation
        model-name ( RO): GRID M60-0B
         max-heads ( RO): 2
    max-resolution ( RO): 2560x1600
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
         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 ~]#
```

14.1.2.2 Viewing Detailed Information About a vgpu-type Object

To see detailed information about a `vgpu-type`, use `xe vgpu-type-param-list`:

```
[root@xenserver ~]# xe xe vgpu-type-param-list uuid=7ca2399f-89ab-49dd-bf96-
↪75071ced28fc
uuid ( RO)                     : 7ca2399f-89ab-49dd-bf96-75071ced28fc
               vendor-name ( RO): NVIDIA Corporation
                model-name ( RO): GRID P40-24Q
          framebuffer-size ( RO): 24092082176
                 max-heads ( RO): 4
            max-resolution ( RO): 4096x2160
```

(continues on next page)

(continued from previous page)

```
          supported-on-PGPUs ( RO): f76d1c90-e443-4bfc-8f26-7959a7c85c68
            enabled-on-PGPUs ( RO): f76d1c90-e443-4bfc-8f26-7959a7c85c68
      supported-on-GPU-groups ( RO): 134a7b71-5ceb-8066-ef1b-3b319fb2bef3
        enabled-on-GPU-groups ( RO): 134a7b71-5ceb-8066-ef1b-3b319fb2bef3
                    VGPU-uuids ( RO):
                  experimental ( RO): false
[root@xenserver ~]#
```

14.1.3. 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.

14.1.3.1 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:

```
[root@xenserver ~]# xe gpu-group-list
uuid ( RO)                  : 3d652a59-beaf-ddb3-3b19-c8c77ef60605
          name-label ( RW): Group of NVIDIA Corporation GP100GL [Tesla P100
↪PCIe 16GB] GPUs
    name-description ( RW):

uuid ( RO)                  : 3df80574-c303-f020-efb3-342f969da5de
          name-label ( RW): 85:00.0 VGA compatible controller: NVIDIA
↪Corporation GM204GL [Tesla M60] (rev a1)
    name-description ( RW): 85:00.0 VGA compatible controller: NVIDIA
↪Corporation GM204GL [Tesla M60] (rev a1)

uuid ( RO)                  : 134a7b71-5ceb-8066-ef1b-3b319fb2bef3
          name-label ( RW): 87:00.0 3D controller: NVIDIA Corporation GP102GL
↪[TESLA P40] (rev a1)
    name-description ( RW): 87:00.0 3D controller: NVIDIA Corporation GP102GL
↪[TESLA P40] (rev a1)

uuid ( RO)                  : d32560f2-2158-42f9-d201-511691e1cb2b
          name-label ( RW): 86:00.0 VGA compatible controller: NVIDIA
↪Corporation GM204GL [Tesla M60] (rev a1)
    name-description ( RW): 86:00.0 VGA compatible controller: NVIDIA
↪Corporation GM204GL [Tesla M60] (rev a1)
[root@xenserver ~]#
```

14.1.3.2 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-
↪3b319fb2bef3
uuid ( RO)                    : 134a7b71-5ceb-8066-ef1b-3b319fb2bef3
             name-label ( RW): 87:00.0 3D controller: NVIDIA Corporation
↪GP102GL [TESLA P40] (rev a1)
       name-description ( RW): 87:00.0 3D controller: NVIDIA Corporation
↪GP102GL [TESLA P40] (rev a1)
             VGPU-uuids (SRO): 101fb062-427f-1999-9e90-5a914075e9ca
             PGPU-uuids (SRO): f76d1c90-e443-4bfc-8f26-7959a7c85c68
           other-config (MRW):
     enabled-VGPU-types ( RO): d0e4a116-a944-42ef-a8dc-62a54c4d2d77;
↪9611a3f4-d130-4a66-a61b-21d4a2ca4663; 7ca2399f-89ab-49dd-bf96-75071ced28fc;
↪25dbb2d3-a074-4f9f-92ce-b42d8b3d1de2; 739d7b8e-50e2-48a1-ae0d-5047aa490f0e;
↪67fa06ab-554e-452b-a66e-a4048a5bfdf7; 9626e649-6802-4396-976d-94c0ead1f835;
↪6ea0cd56-526c-4966-8f53-7e1721b95a5c; ad00a95c-d066-4158-b361-487abf57dd30;
↪88162a34-1151-49d3-98ae-afcd963f3105; 125fbbdf-406e-4d7c-9de8-a7536aa1a838;
↪4017c9dd-373f-431a-b36f-50e4e5c9f0c0; af121387-0b58-498a-8d04-fe0305e4308f;
↪5b9acd25-06fa-43e1-8b53-c35bceb8515c; af593219-0800-42da-a51d-d13b35f589e1;
↪9b2eaba5-565f-4cb4-ad9b-6347cfb03e93; 57bb231f-f61b-408e-a0c0-106bddd91019;
↪d27f84a2-53f8-4430-ad15-0eca225cd974
    supported-VGPU-types ( RO): d0e4a116-a944-42ef-a8dc-62a54c4d2d77;
↪9611a3f4-d130-4a66-a61b-21d4a2ca4663; 7ca2399f-89ab-49dd-bf96-75071ced28fc;
↪25dbb2d3-a074-4f9f-92ce-b42d8b3d1de2; 739d7b8e-50e2-48a1-ae0d-5047aa490f0e;
↪67fa06ab-554e-452b-a66e-a4048a5bfdf7; 9626e649-6802-4396-976d-94c0ead1f835;
↪6ea0cd56-526c-4966-8f53-7e1721b95a5c; ad00a95c-d066-4158-b361-487abf57dd30;
↪88162a34-1151-49d3-98ae-afcd963f3105; 125fbbdf-406e-4d7c-9de8-a7536aa1a838;
↪4017c9dd-373f-431a-b36f-50e4e5c9f0c0; af121387-0b58-498a-8d04-fe0305e4308f;
↪5b9acd25-06fa-43e1-8b53-c35bceb8515c; af593219-0800-42da-a51d-d13b35f589e1;
↪9b2eaba5-565f-4cb4-ad9b-6347cfb03e93; 57bb231f-f61b-408e-a0c0-106bddd91019;
↪d27f84a2-53f8-4430-ad15-0eca225cd974
    allocation-algorithm ( RW): depth-first
[root@xenserver ~]
```

14.1.4. 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*

14.2. 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-
↪a364a7f619c2 gpu-group-uuid=be825ba2-01d7-8d51-9780-f82cfaa64924 vgpu-type-
↪uuid=3f318889-7508-c9fd-7134-003d4d05ae56b73cbd30-096f-8a9a-523e-
```

(continues on next page)

(continued from previous page)

```
↪a800062f4ca7
[root@xenserver ~]#
```

Creating the vgpu object for a VM does not immediately cause a virtual GPU to be created on a physical GPU. Instead, the virtual GPU is created whenever its associated VM is started. For more details on how vGPUs are created at VM startup, see *Controlling vGPU allocation*.

Note

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.

14.3. 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*.

14.3.1. 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]
```

(continues on next page)

(continued from previous page)

```
             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)
                 host-uuid ( RO): b55452df-1ee4-4e4e-bd97-3aee97b2123a
           host-name-label ( RO): xs7.1-krish
                    pci-id ( RO): 0000:87:00.0
              dependencies (SRO):
              other-config (MRW):
       supported-VGPU-types ( RO): 5b9acd25-06fa-43e1-8b53-c35bceb8515c;
↪88162a34-1151-49d3-98ae-afcd963f3105; 9b2eaba5-565f-4cb4-ad9b-6347cfb03e93;
↪739d7b8e-50e2-48a1-ae0d-5047aa490f0e; d0e4a116-a944-42ef-a8dc-62a54c4d2d77;
↪7ca2399f-89ab-49dd-bf96-75071ced28fc; 67fa06ab-554e-452b-a66e-a4048a5bfdf7;
↪9611a3f4-d130-4a66-a61b-21d4a2ca4663; d27f84a2-53f8-4430-ad15-0eca225cd974;
↪125fbbdf-406e-4d7c-9de8-a7536aa1a838; 4017c9dd-373f-431a-b36f-50e4e5c9f0c0;
↪6ea0cd56-526c-4966-8f53-7e1721b95a5c; af121387-0b58-498a-8d04-fe0305e4308f;
↪9626e649-6802-4396-976d-94c0ead1f835; ad00a95c-d066-4158-b361-487abf57dd30;
↪af593219-0800-42da-a51d-d13b35f589e1; 25dbb2d3-a074-4f9f-92ce-b42d8b3d1de2;
↪57bb231f-f61b-408e-a0c0-106bddd91019
        enabled-VGPU-types (SRW): 5b9acd25-06fa-43e1-8b53-c35bceb8515c;
↪88162a34-1151-49d3-98ae-afcd963f3105; 9b2eaba5-565f-4cb4-ad9b-6347cfb03e93;
↪739d7b8e-50e2-48a1-ae0d-5047aa490f0e; d0e4a116-a944-42ef-a8dc-62a54c4d2d77;
↪7ca2399f-89ab-49dd-bf96-75071ced28fc; 67fa06ab-554e-452b-a66e-a4048a5bfdf7;
↪9611a3f4-d130-4a66-a61b-21d4a2ca4663; d27f84a2-53f8-4430-ad15-0eca225cd974;
↪125fbbdf-406e-4d7c-9de8-a7536aa1a838; 4017c9dd-373f-431a-b36f-50e4e5c9f0c0;
↪6ea0cd56-526c-4966-8f53-7e1721b95a5c; af121387-0b58-498a-8d04-fe0305e4308f;
↪9626e649-6802-4396-976d-94c0ead1f835; ad00a95c-d066-4158-b361-487abf57dd30;
↪af593219-0800-42da-a51d-d13b35f589e1; 25dbb2d3-a074-4f9f-92ce-b42d8b3d1de2;
↪57bb231f-f61b-408e-a0c0-106bddd91019
             resident-VGPUs ( RO): 101fb062-427f-1999-9e90-5a914075e9ca
[root@xenserver ~]#
```

Note

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>
```

14.3.2. Controlling the vGPU types enabled on specific physical GPUs

Physical GPUs support several vGPU types, as defined in *Virtual GPU Types for 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 XenServer*).

14.3.2.1 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**.

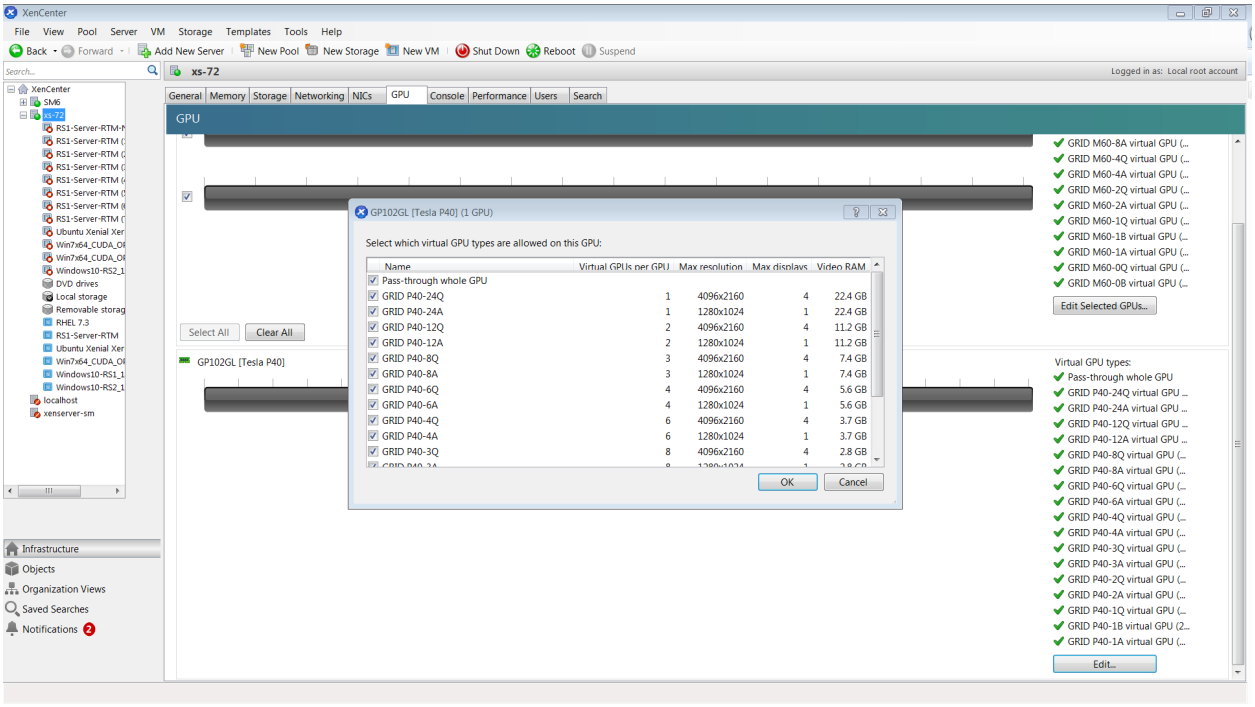


Figure 14.2: Editing a GPU's enabled vGPU types using XenCenter

14.3.2.2 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
uuid ( RO)                          : cb08aaae-8e5a-47cb-888e-60dcc73c01d3
             vendor-name ( RO): NVIDIA Corporation
                       device-name ( RO): GP102GL [Tesla P40]
             dom0-access ( RO): enabled
is-system-display-device ( RO): false
          gpu-group-uuid ( RW): bfe1603d-c526-05f3-e64f-951485ef3b49
        gpu-group-name-label ( RO): 87:00.0 3D controller: NVIDIA Corporation
↪GP102GL [Tesla P40] (rev a1)
                  host-uuid ( RO): fdeb6bbb-e460-4cf1-ad43-49ac81c20540
                host-name-label ( RO): xs-72
                 pci-id ( RO): 0000:87:00.0
```

(continues on next page)

(continued from previous page)

```
                        dependencies (SRO):
                        other-config (MRW):
    supported-VGPU-types ( RO): 23e6b80b-1e5e-4c33-bedb-e6dlae472fec;
↪f5583e39-2540-440d-a0ee-dde9f0783abf; al8e46ff-4d05-4322-b040-667ce77d78a8;
↪adell9a9-84el-435f-b0e9-14cl62e212fb; 2560d066-054a-48a9-a44d-3f3f90493a00;
↪47858f38-045d-4a05-9blc-9128fee6b0ab; Ifb527f6-493f-442b-abe2-94a6fafd49ce;
↪78b8e044-09ae-4a4c-8a96-b20c7a585842; 18ed7e7e-f8b7-496e-9784-8ba4e35acaa3;
↪48681d88-c4e5-4e39-85ff-c9bal2e8e484 ; cc3dbbfb-4b83-400d-8c52-811948b7f8c4;
↪ 8elad75a-ed5f-4609-83ff-5f9bca9aaca2; 840389a0-f511-4f90-8153-8a749d85b09e;
↪ a2042742-da67-4613-a538-ldl7d30dccb9; 299e47c2-8fcl-4edf-aa31-e29db84168c6;
↪ e95c636e-06e6-4 47e-8b49-14b37d308922; 0524a5d0-7160-48c5-a9el-
↪cc33e76dc0de; 09043fb2-6d67-4443-b312-25688f13e012
     enabled-VGPU-types (SRW): 23e6b80b-1e5e-4c33-bedb-e6dlae472fec;
↪f5583e39-2540-440d-a0ee-dde9f0783abf; al8e46ff-4d05-4322-b040-667ce77d78a8;
↪adell9a9-84el-435f-b0e9-14cl62e212fb; 2560d066-054a-48a9-a44d-3f3f90493a00;
↪47858f38-045d-4a05-9blc-9128fee6b0ab; Ifb527f6-493f-442b-abe2-94a6fafd49ce;
↪78b8e044-09ae-4a4c-8a96-b20c7a585842; 18ed7e7e-f8b7-496e-9784-8ba4e35acaa3;
↪48681d88-c4e5-4e39-85ff-c9bal2e8e484 ; cc3dbbfb-4b83-400d-8c52-811948b7f8c4;
↪ 8elad75a-ed5f-4609-83ff-5f9bca9aaca2; 840389a0-f511-4f90-8153-8a749d85b09e;
↪ a2042742-da67-4613-a538-ldl7d30dccb9; 299e47c2-8fcl-4edf-aa31-e29db84168c6;
↪ e95c636e-06e6-4 47e-8b49-14b37d308922; 0524a5d0-7160-48c5-a9el-
↪cc33e76dc0de; 09043fb2-6d67-4443-b312-25688f13e012
             resident-VGPUs ( RO):

[root@xenserver-vgx-test ~]#  xe pgpu-param-set uuid=cb08aaae-8e5a-47cb-888e-
↪60dcc73c01d3 enabled-VGPU-types=23e6b80b-1e5e-4c33-bedb-e6dlae472fec
```

14.3.3. 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-3b319fb2bef3
   [root@xenserver ~]
   ```

> **Note**
>
> 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=7c1e3cff-1429-0544-df3d-
↪bf8a086fb70a gpu-group-uuid=585877ef-5a6c-66af-fc56-7bd525bdc2f6
[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`.

> **Note**
>
> 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:

14.4. Cloning vGPU-Enabled VMs

The fast-clone or copying feature of XenServer 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.

14.4.1. Cloning a vGPU-enabled VM by using xe

To clone a vGPU-enabled VM from the dom0 shell, use `vm-clone`:

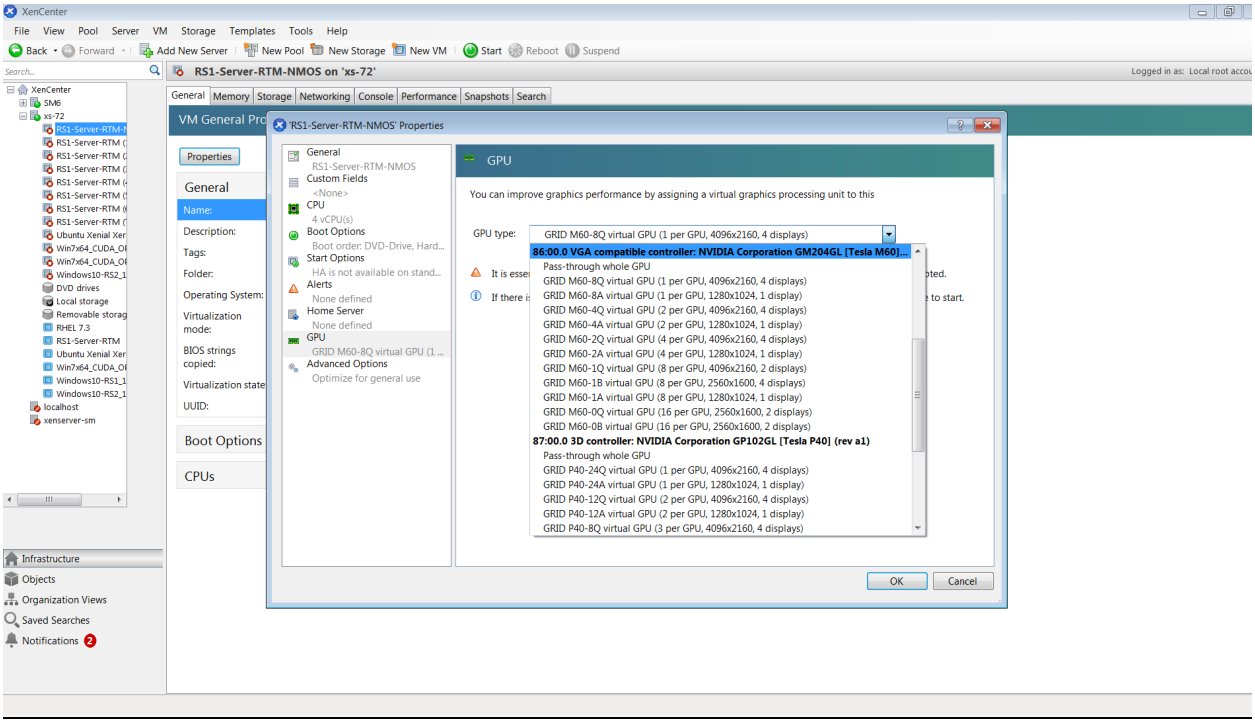


Figure 14.3: Using a custom GPU group within XenCenter

```
[root@xenserver ~]# xe vm-clone new-name-label="new-vm" vm="base-vm-name"
↪7f7035cb-388d-1537-1465-1857fb6498e7
[root@xenserver ~]#
```

14.4.2. 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 14.4.

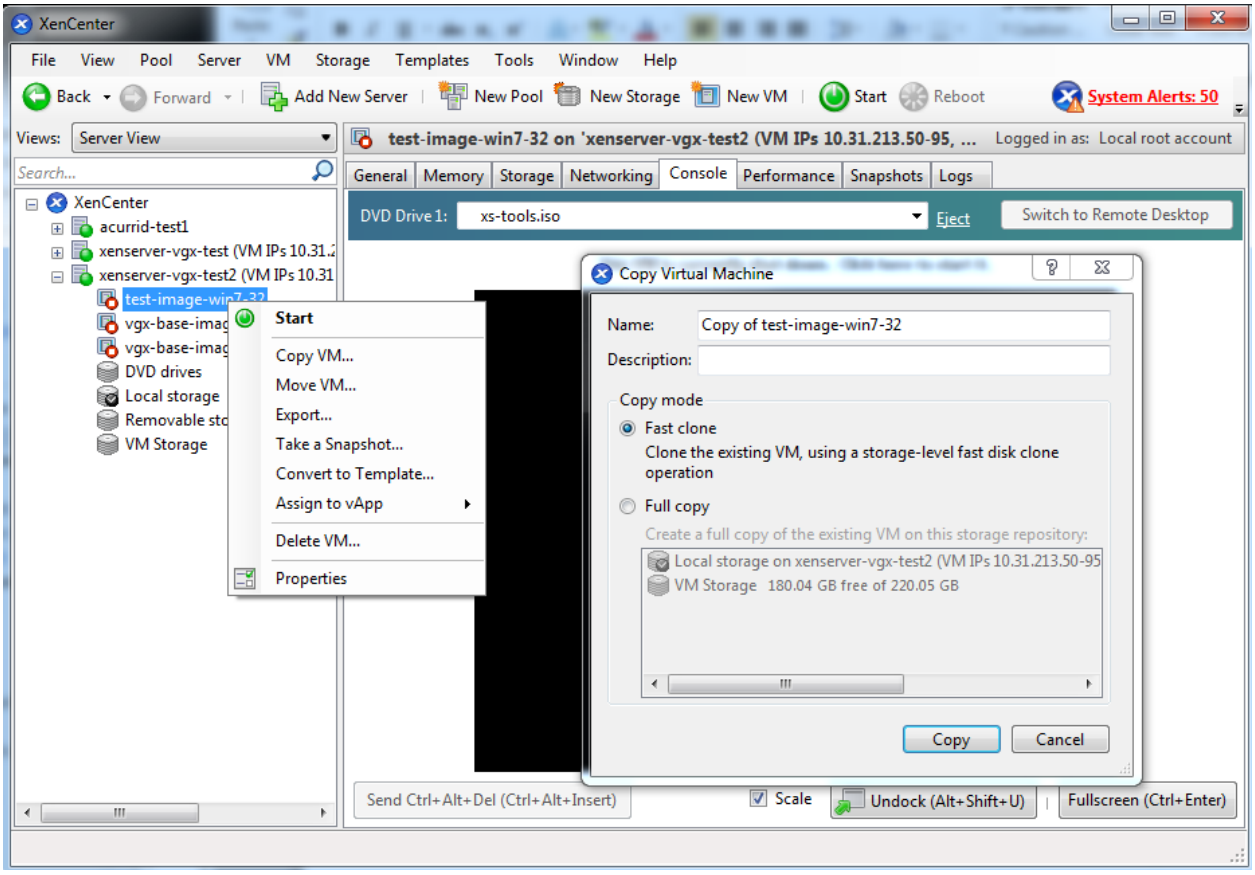


Figure 14.4: Cloning a VM using XenCenter

Chapter 15. XenServer Performance Tuning

This chapter provides recommendations on optimizing performance for VMs running with NVIDIA vGPU on XenServer.

15.1. XenServer Tools

To get maximum performance out of a VM running on XenServer, regardless of whether you are using NVIDIA vGPU, you must install 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.

15.2. 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 Virtual Apps and Desktops with HDX 3D Pro remote graphics and, after the stack is installed, disable vGPU's console VGA.

15.2.1. 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 Citrix Virtual Apps and Desktops or a VNC server), its vGPU console VGA interface can be disabled by creating the vGPU by using the `xe` command, and specifying plugin parameters for the group to which the vGPU belongs:

1. Create the vGPU.

```
[root@xenserver ~]# xe vgpu-create gpu-group-uuid=<gpu-group-uuid> vgpu-
↪type-uuid=<vgpu-type-uuid> vm-uuid=<vm-uuid>
```

This command returns <vgpu-uuid> as stored in XAPI.

2. Specify plugin parameters for the group to which the vGPU belongs.

```
[root@xenserver ~]# xe vgpu-param-set uuid=<vgpu-uuid> extra_args=disable_
↪vnc=1
```

Caution

If you disable console VGA before you have installed or enabled an alternate mechanism to access the VM (such as Citrix Virtual Apps and Desktops), 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 `extra_args` key the from group to which the vGPU belongs
- Removing `disable_vnc=1` from the `extra_args` or `vgpu_extra_args` key
- Setting `disable_vnc=0`, for example:

```
[root@xenserver ~]# xe vgpu-param-set uuid=<vgpu-uuid> extra_args=disable_
↪vnc=0
```

Chapter 16. Notices

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a

third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, NVIDIA GRID, NVIDIA GRID vGPU, NVIDIA Maxwell, NVIDIA Pascal, NVIDIA Turing, NVIDIA Volta, GPUDirect, 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

©2013-2026, NVIDIA Corporation