NVIDIA Virtual Compute Server for Red Hat Enterprise Linux with KVM

Deployment Guide

Document History

DU-10130-001_v01

Version	Date	Authors	Description of Change
01	September 4, 2020	AS, EA	Initial Release
02	October 2, 2020	AS, EA, DS	RHEL build out
03	December 16, 2020	AS	Technical feedback and RH feedback

Table of Contents

Chapter 1. Executive Summary

This document provides insights into how to deploy NVIDIA Virtual Compute Server on Red Hat Virtualization/Red Hat Enterprise Linux (RHEL) and serves as a technical resource for understanding system pre-requisites, installation, and configuration.

1.1 What is NVIDIA Virtual Compute Server

NVIDIA Virtual Compute Server enables the benefits of hypervisor-based server virtualization for GPU accelerated servers. Data center admins are now able to power any compute-intensive workload with GPUs in a virtual machine (VM).

NVIDIA Virtual Compute Server software virtualizes NVIDIA GPUs to accelerate large workloads, including more than 600 GPU accelerated applications for AI, deep learning, and high-performance computing (HPC). With GPU sharing, multiple VMs can be powered by a single GPU, maximizing utilization and throughput, or a single VM can be powered by multiple virtual GPUs, making even the most intensive workloads manageable. With support for all major hypervisor virtualization platforms, including Red Hat RHV/RHEL and VMware vSphere, data center administrators can use the same management tools for their GPU-accelerated servers as they do for the rest of their data center.

NVIDIA Virtual Compute Server supports the NVIDIA NGC (NVIDIA GPU Cloud) GPU-optimized repository for deep learning, machine learning, and HPC. NGC software includes containers for the top AI and data science software, tuned, tested, and optimized by NVIDIA, as well as fully tested containers for HPC applications and data analytics.

NVIDIA Virtual Compute Server is not tied to a user with a display. It is licensed per GPU as a 1-year subscription with NVIDIA enterprise support included. This allows a number of compute workloads in multiple VMs to be run on a single GPU, maximizing utilization of resources and ROI.

For more information regarding NVIDIA Virtual Compute Server please refer to the NVIDIA Virtual Compute Server Solution Overview.

1.2 Why NVIDIA vGPU?

NVIDIA Virtual Compute Server (NVIDIA vCS) can power the most compute-intensive workloads with virtual GPUs. NVIDIA vCS software is based upon NVIDIA virtual GPU (vGPU) technology, and includes the NVIDIA compute driver, which is required by compute-intensive operations. NVIDIA vGPU enables

multiple virtual machines (VMs) to have simultaneous, direct access to a single physical GPU, or GPUs can be aggregated within a single VM. vGPU uses the same NVIDIA drivers that are deployed on non-virtualized operating systems. By doing so, NVIDIA vGPU provides VMs with high performance compute and application compatibility, as well as cost-effectiveness and scalability since multiple VMs can be customized to specific tasks that may demand more or less GPU compute or memory.
With NVIDIA vCS you can gain access to the most powerful GPUs in a virtualized environment and gain vGPU software features such as:

- Management and monitoring – streamline data center manageability by leveraging hypervisor-based tools.
- Security – Extend the benefits of server virtualization to GPU workloads.
- Multi-Tenant – Isolate workloads and securely support multiple users.

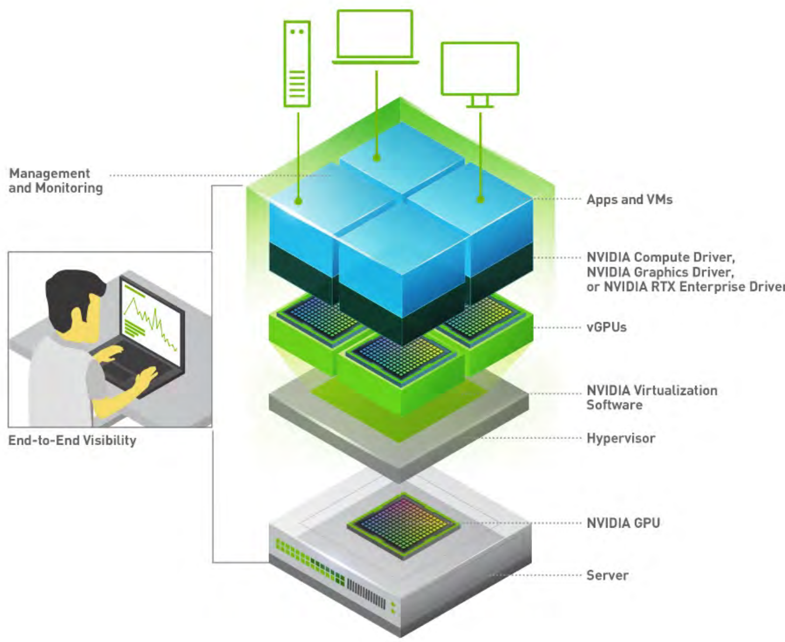
1.3 NVIDIA vGPU Architecture

The high-level architecture of an NVIDIA virtual GPU-enabled VDI environment is illustrated below in Figure 1.1. Here, the GPUs in the server, and the NVIDIA vGPU Manager software (.RPM file) is installed on the host server. This software enables multiple VMs to share a single GPU, or if there are multiple GPU's in the server, they can be aggregated so that a single VM can access multiple GPUs.

This GPU enabled environment provides not only unprecedented performance, but also enables support for more users on a server because work that is typically done by the CPU can be offloaded to the GPU. Physical NVIDIA GPUs can support multiple *virtual* GPUs (vGPUs) and be assigned directly to guest VMs under the control of NVIDIA's Virtual GPU Manager running in a hypervisor.

Guest VMs use NVIDIA vGPUs in the same manner as physical GPUs that have been passed through by the hypervisor. For NVIDIA vGPU deployments, the NVIDIA vGPU software automatically selects the correct type of license based on the vGPU type assigned.

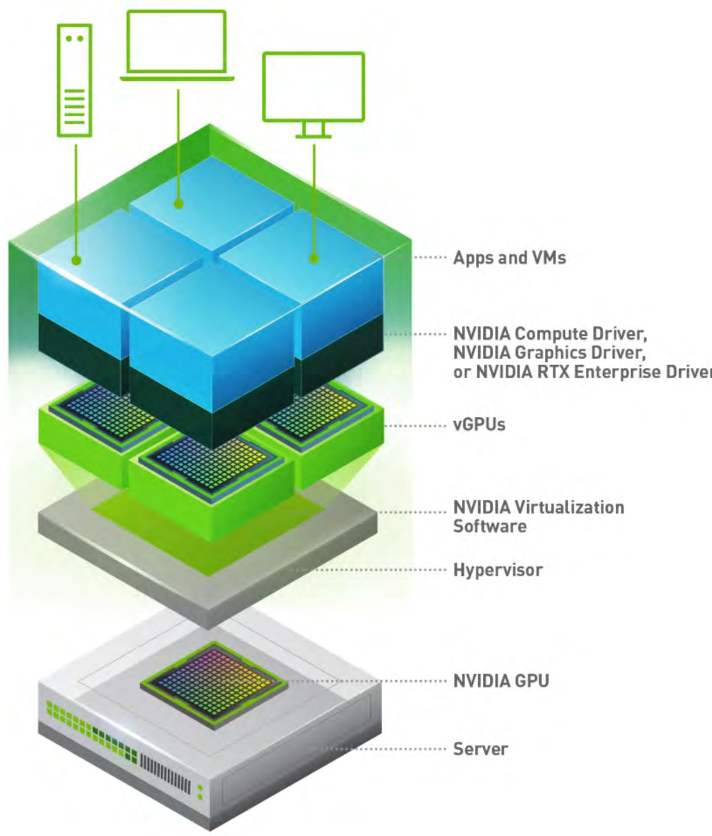
Figure 1.1 NVIDIA vGPU Platform Solution Architecture



NVIDIA vGPUs are comparable to conventional GPUs in that they have a fixed amount of GPU Memory and one or more virtual display outputs or *heads*. Multiple heads support multiple displays. Managed by the NVIDIA vGPU Manager installed in the hypervisor, the vGPU Memory is allocated out of the physical GPU frame buffer at the time the vGPU is created. The vGPU retains exclusive use of that GPU Memory until it is destroyed.

All vGPUs resident on a physical GPU share access to the GPU's engines, including the graphics (3D) and video decode and encode engines. Figure 1.2 shows the vGPU internal architecture. The VM's guest OS leverages direct access to the GPU for performance and critical fast paths. Non-critical performance management operations use a para-virtualized interface to the NVIDIA Virtual GPU Manager.

Figure 1.2 NVIDIA vGPU Internal Architecture



1.4 Supported GPUs

NVIDIA virtual GPU software is supported with NVIDIA data center GPUs. For a list of certified servers with NVIDIA GPUs, consult the NVIDIA vGPU Certified Servers page. Please refer to the NVIDIA vCS solution brief for a full list of recommended and supported GPUs. Each card requires auxiliary power cables connected to it (except NVIDIA P4 & T4).

Most industry standard servers require an enablement kit for proper mounting of NVIDIA cards. Check with your server OEM of choice for more specific requirements.
The maximum number of vGPUs that can be created simultaneously on a physical GPU varies on a card-by-card basis. A complete list of maximum vGPUs per GPU is located here. For example, an NVIDIA V100 PCIe 32 GB GPU that has 32 GB of GPU Memory, can support up to six 8C profiles (32 GB total with 4 GB per VM). You cannot oversubscribe GPU memory, and it must be shared equally for each physical GPU. If you have multiple GPUs installed in a server, you have the flexibility to allocate each physical GPU appropriately to meet your users demands.

1.5 Virtual GPU Types

vGPUs have a fixed amount of GPU Memory, number of supported displays, and maximum resolution. vGPU types 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 NVIDIA RTX Enterprise drivers
C-series	Compute-intensive server workloads, such as artificial intelligence (AI), deep learning (DL), or high-performance computing (HPC)
B-series	Virtual desktops for business professionals and knowledge workers
A-series	App streaming or session-based solutions for virtual applications users

NVIDIA vCS uses the C-Series vGPU profiles. Please refer to the NVIDIA vCS solution brief for more information regarding the available profiles.

1.6 General Prerequisites

Prior to installing and configuring vGPU software for NVIDIA vCS it is important to document an evaluation plan. This can consist of the following:

- A list of your business drivers and goals
- A list of all the user groups, their workloads, and applications with current, and future projections in consideration
- Current end-user experience measurements and analysis
- ROI / Density goals

If you are new to virtualization it is recommended to review Red Hat Enterprise Linux Visualization Deployment and Administration Guide.
The following elements are required to install and configure vGPU software on Red Hat Enterprise Linux with KVM.

- NVIDIA certified servers with NVIDIA GPUs
    - High-speed RAM
    - Fast networking
    - If using local storage, IOPS plays a major role in performance
    - Intel Xeon E5-2600 v4, Intel Xeon Scalable Processor Family with 2.6GHz CPU or faster.
- Select the appropriate NVIDIA GPU for your use case. Please refer to the NVIDIA vCS solution brief for a full list of recommended and supported GPUs.
- Red Hat Enterprise Linux with KVM. For a list of supported versions, please refer to the vGPU software documentation.
- NVIDIA vCS software and license (free trial license available).
- NVIDIA vGPU Manager RPM

- NVIDIA WDDM guest driver
- NVIDIA Linux guest driver

Note: The vGPU Manager RPM is loaded like a driver in the RHEL hypervisor.

For testing and benchmarking you may leverage the NVIDIA System Management interface (NV-SMI) management and monitoring tool.

1.6.1 Server Configuration

The following server configuration details are considered best practices:

- Hyperthreading – Enabled
- Power Setting or System Profile – High Performance
- CPU Performance (if applicable) – Enterprise or High Throughput
- Memory Mapped I/O above 4 GB - Enabled (if applicable)

Note: If NVIDIA card detection does not include all of the installed GPUs, set SR-IOV option to Enabled.

Chapter 2. Installing Red Hat Enterprise with KVM

This chapter covers the following RHEL with KVM installation topics:

- Choosing the Installation Method
- Preparing USB Boot Media
- Installing RHEL KVM
- Initial Host Configuration

> Note: This deployment guide assumes you are building an environment as proof of concept and not for production deployment. Consequently, some choices are made to speed up and ease the process. See the corresponding guides for each technology, and make choices appropriate for your needs, before building your production environment.

For this guide, RHEL 8.3 with KVM is used, Red Hat Virtualization (RHV) installation setups are reasonably similar.

2.1 Choosing the Installation Method

RHEL can be installed from USB boot media, from optical media, or over a network. NVIDIA's lab used Supermicro's IPMI and virtual media to boot from an ISO file and install on local storage. Network installation via PXE booting is beyond the scope of this guide but be aware of it as it can ease mass deployments in environments like datacenters.

2.2 Preparing USB Boot Media

For more information, see the Red Hat installation guide.
Installation via boot media is the easiest way to install RHEL to a local server.

1. Download the latest ISO file from Red Hat's site. Download the latest ISO file from the Red Hat's site.
2. Insert your USB device in your computer.
3. From terminal, use the dd command to image the USB device:

```
dd if=/<download_directory>/<latest_image>.iso of=/dev/[device] bs=512k
```

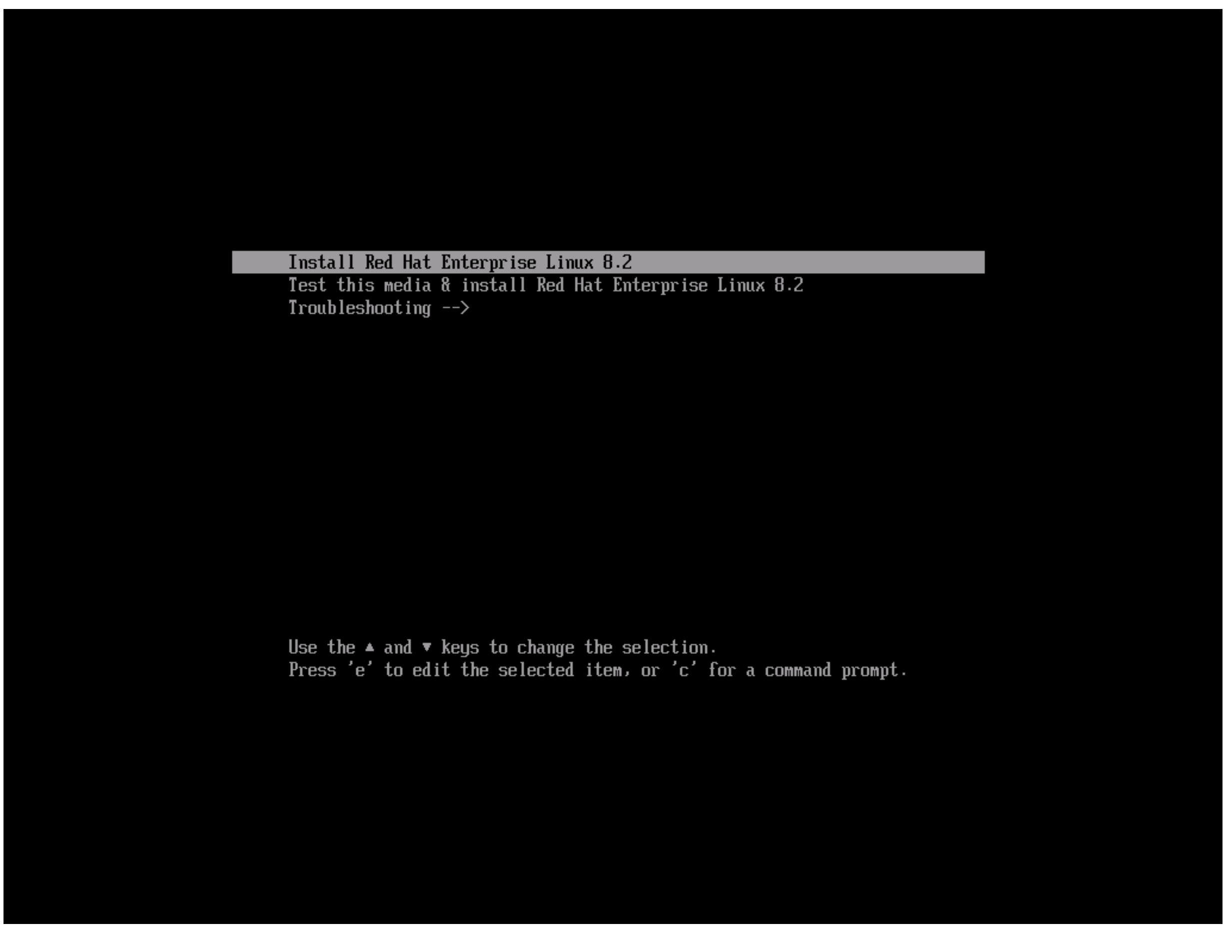
4. Replace <download_directory> with the location of the ISO file and <latest_image> with the name of the latest image. *Device* is the name of your USB devices mount point.

> Note: This deployment guide assumes you are using a Linux environment. If you need to create a USB installation media from Windows or Mac OS, Red Hat recommends you use the Fedora Media builder. Instructions are available from Red Hat. Instructions are available from Red Hat. RUFUS is also a recommended option for Windows users.

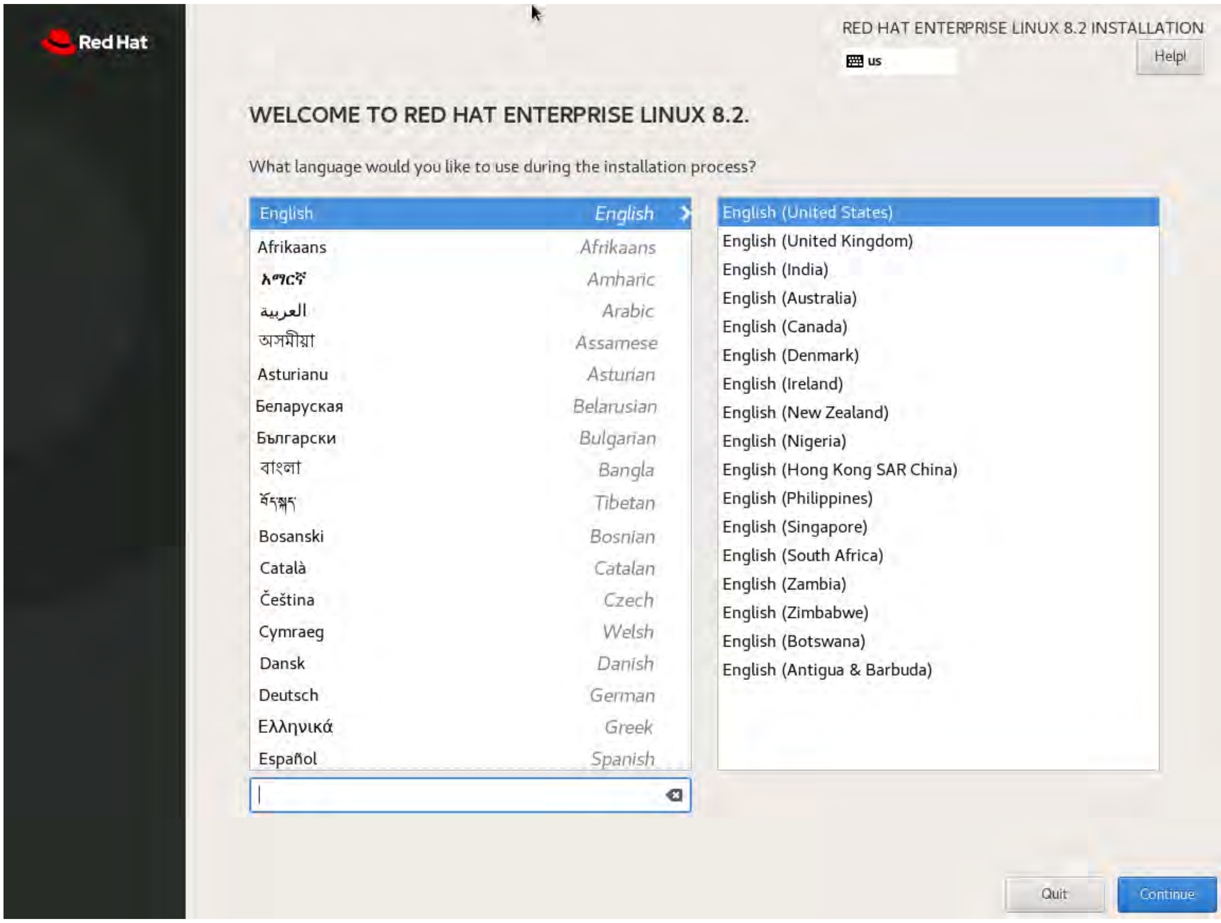
2.3 Installing RHEL with KVM

Use the following procedure to install RHEL with KVM. Select the USB boot media with the RHEL ISO from your host's boot menu.

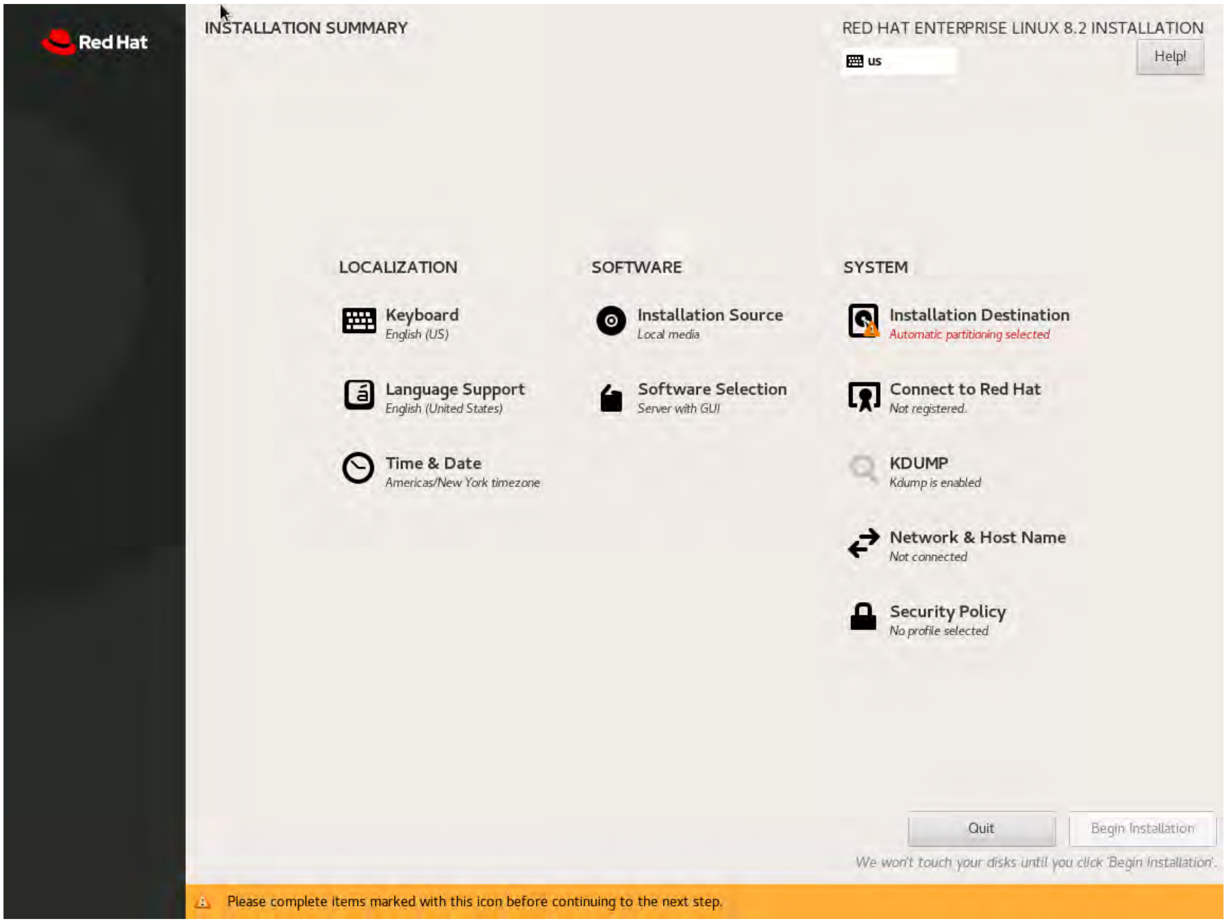
1. Apply power to start the host and select your USB media to boot. Consult your server vendor's documentation to set boot options.
2. Select Install Red Hat Enterprise Linux from the boot menu and press ENTER.



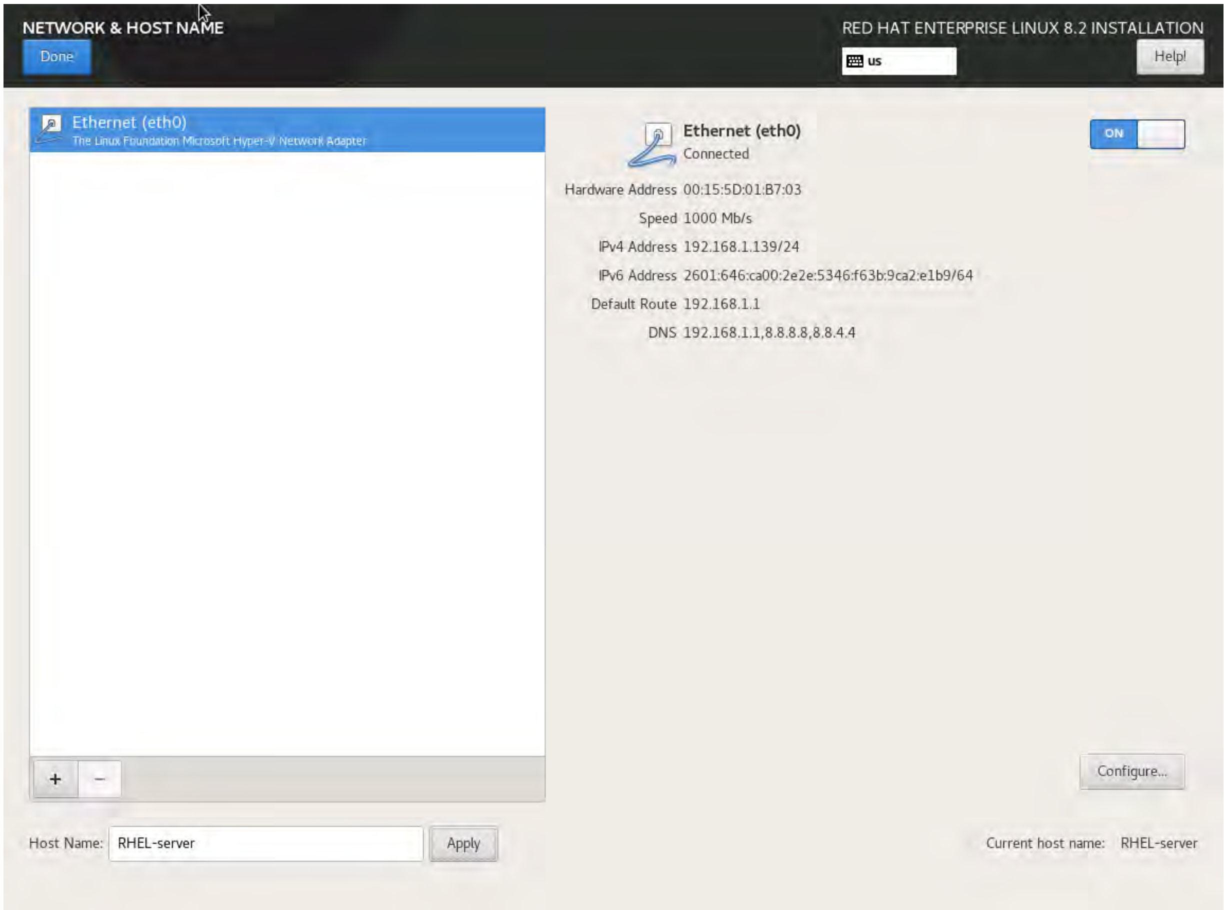
3. Select your desired language. This guide uses English (United States).



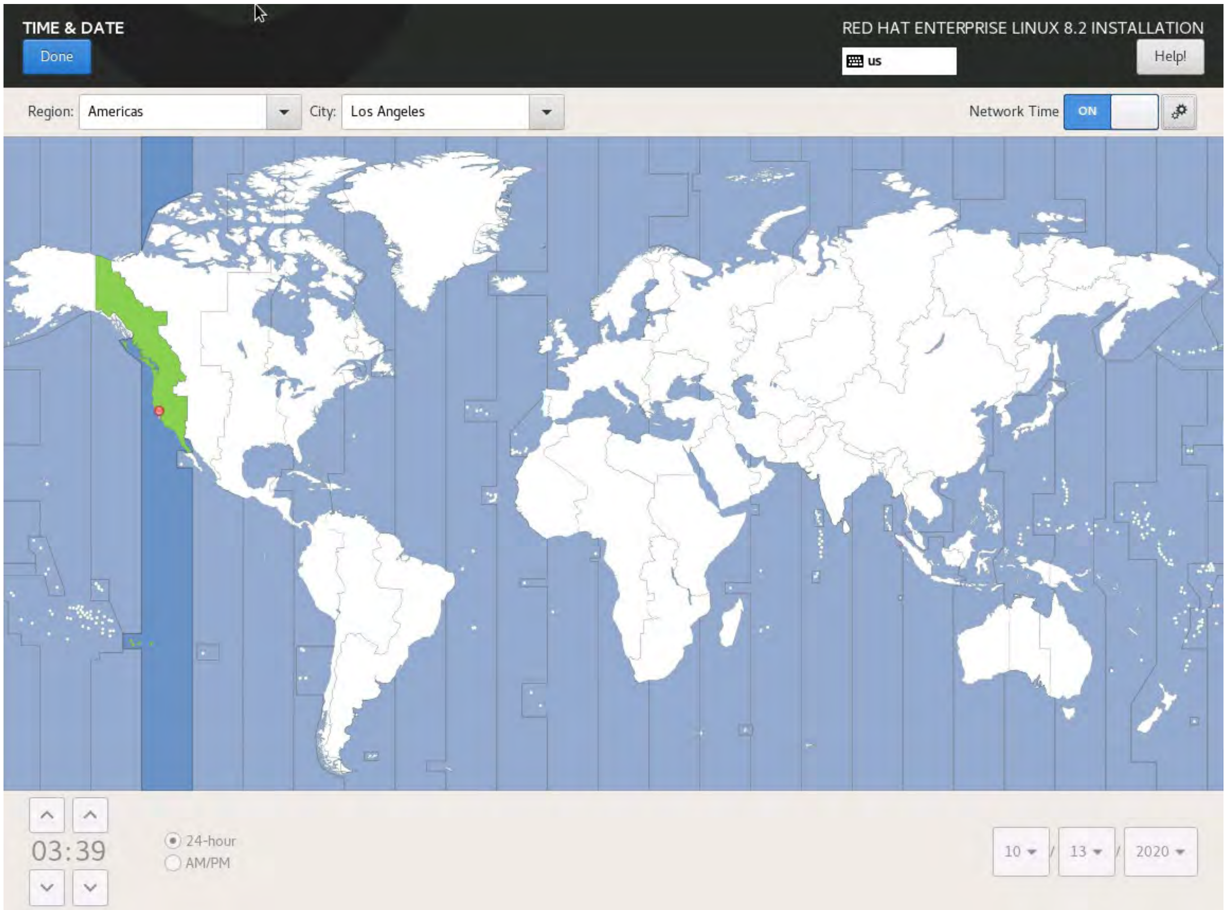
4. Once you arrive at the main installation screen, there are a few options to configure.



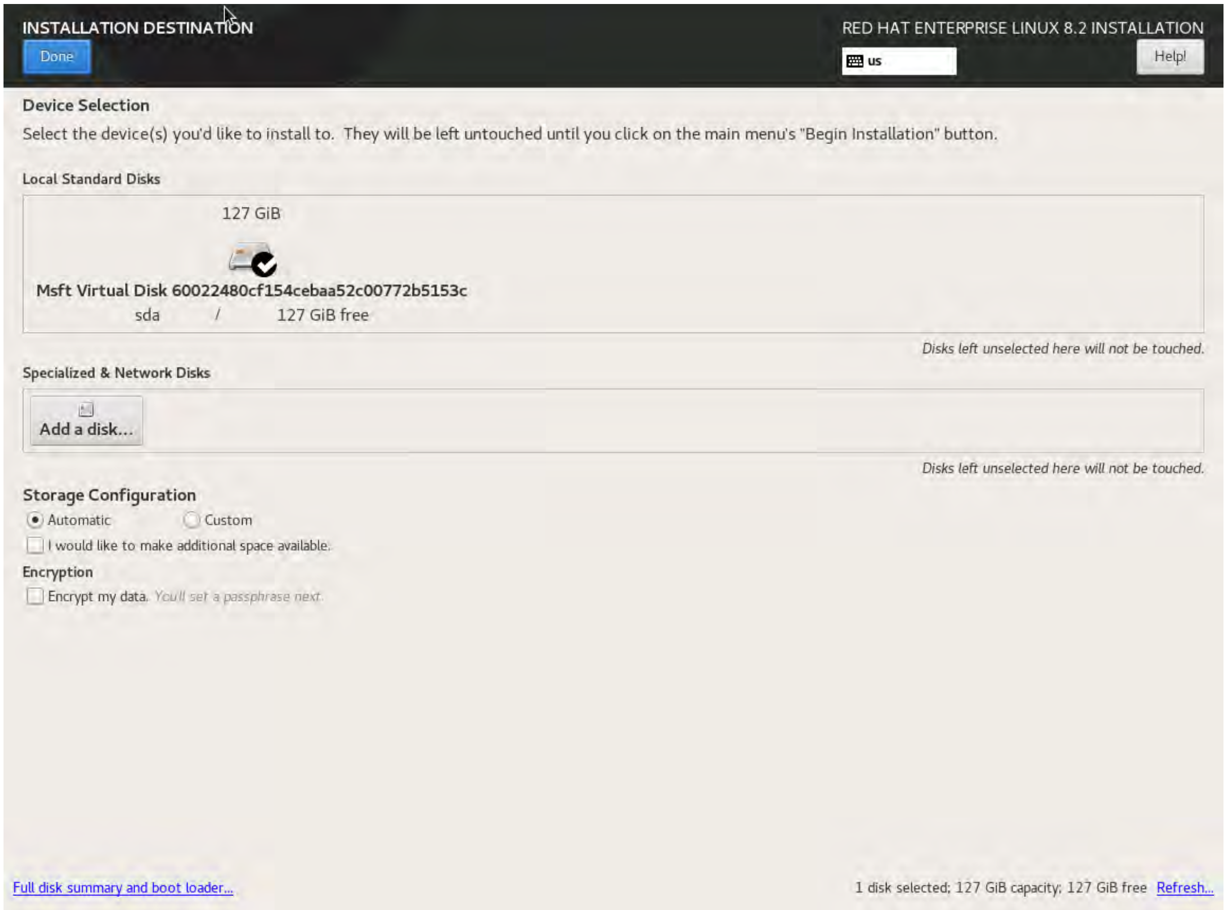
5. Networking is the first thing to configure. Click Network & Host Name.



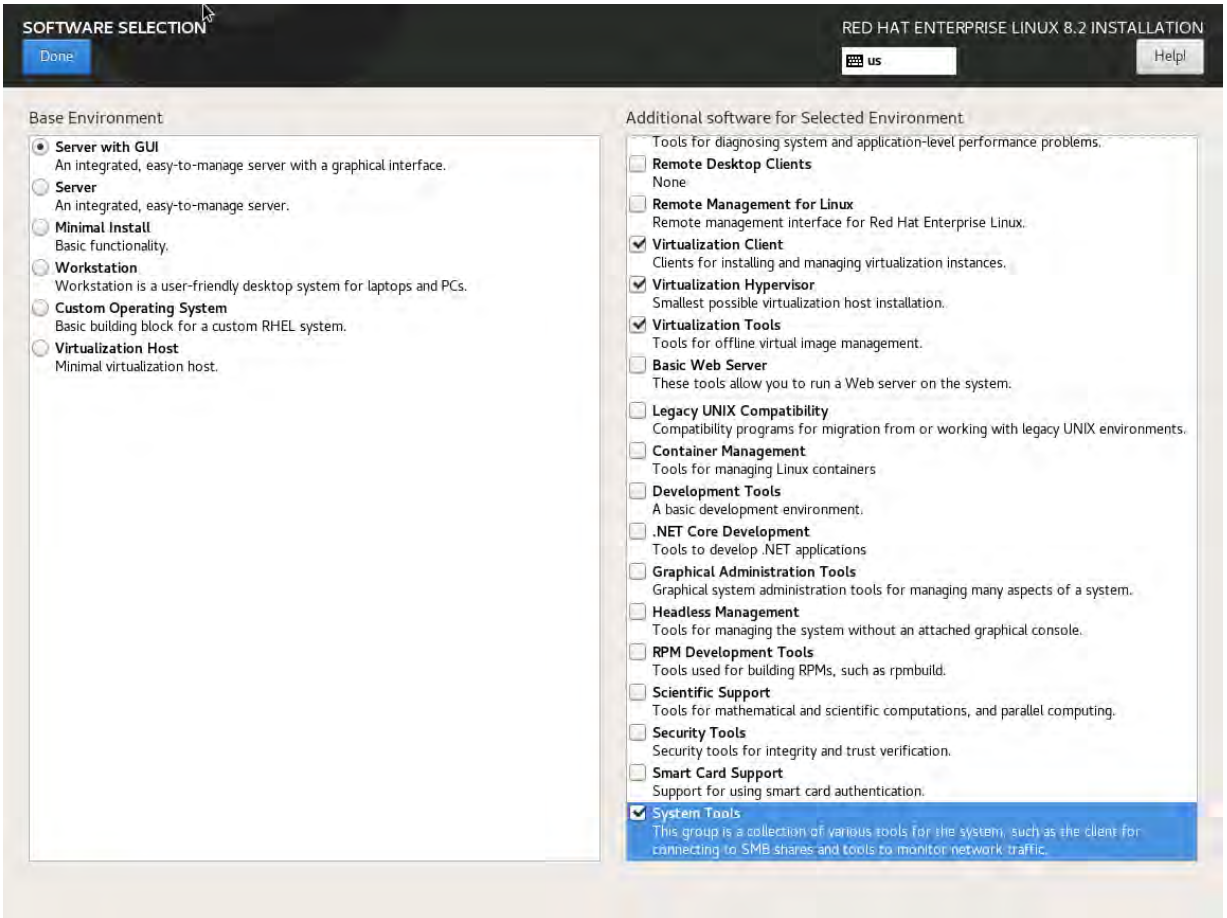
6. Give your server a unique hostname in the lower right corner. Enable the network adapter that provides Internet server to your server (in the above example, eth0). If no DHCP is available on your network, enter your network details manually. When the Ethernet configuration is complete, click Done in the top left corner.
7. Select Date & Time to set time and date.



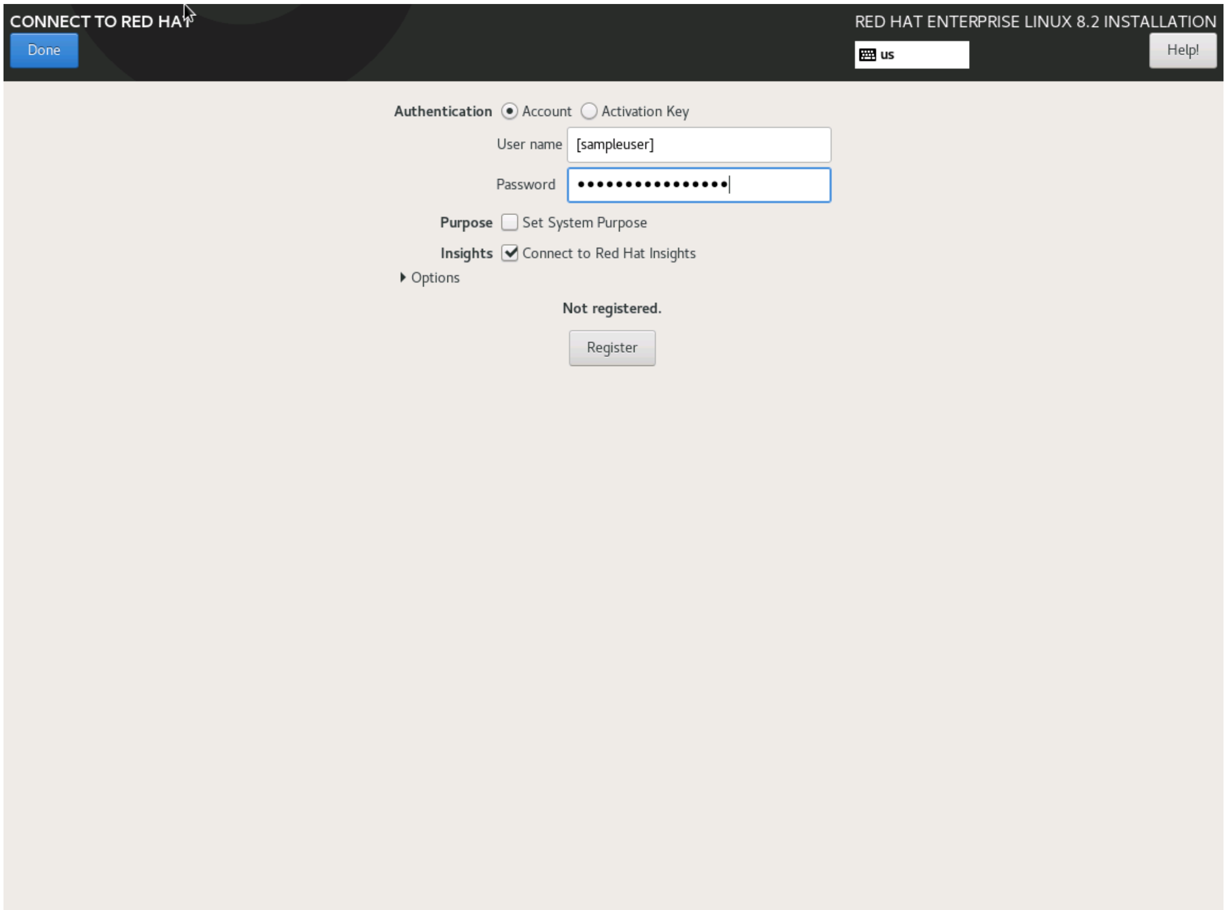
8. Set your time zone and enable Network Time (in the top right corner) as well. If no NTP connection is available, set the date and time manually. An incorrect setting can lead to issues with SSH, YUM, and other certificate-based services. When you are finished, click Done.
9. Now it is time to set a destination for our RHEL installation. Click Installation Destination.



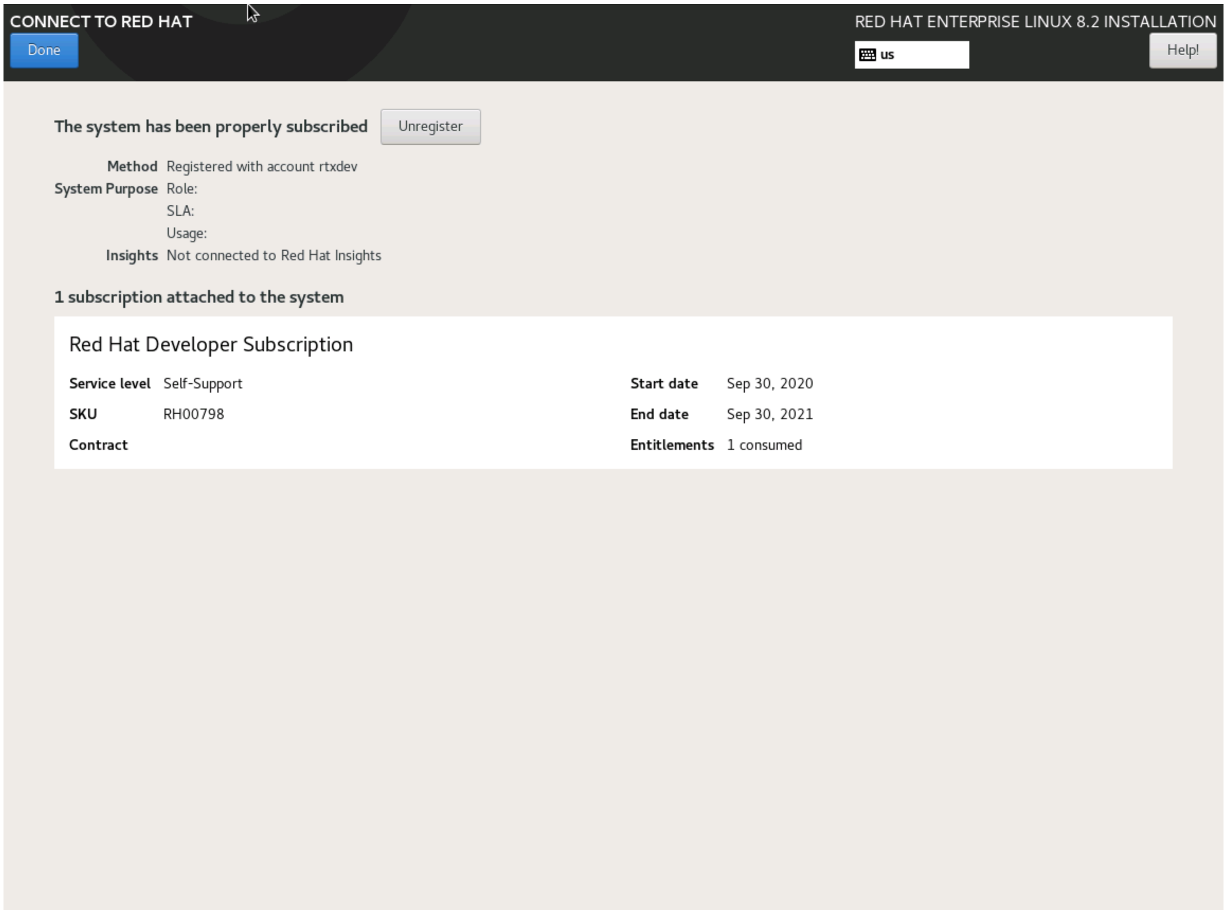
10. By default, the installer selects your first logical drive for your virtual disk and uses the whole drive. You may want to select a different logical drive and a smaller space allocation for a real installation. Be aware that the installation process erases this drive, and *any data previously on the drive will be lost*. Click Done.
11. Next you configure the installer to install the virtualization platform you need to leverage vGPU.



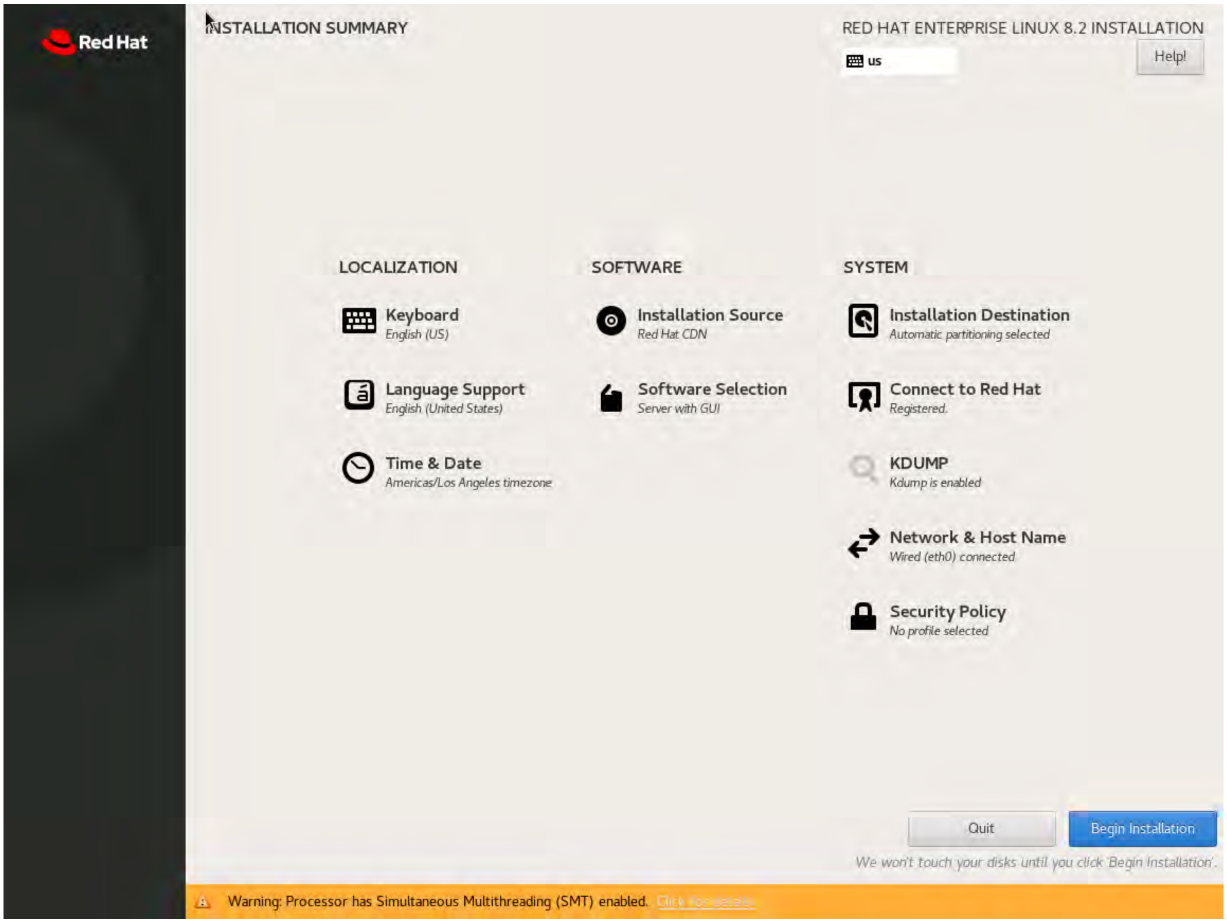
12. From the Base Environment list, select Server with GUI. In the Additional software list, check Virtualization Client, Virtualization Hypervisor, Virtualization Tools, and System Administration Tools. This gives us basic hypervisor setup. Click Done. The installer checks dependencies to determine what packages it needs to download. Then it verifies that it can download all of them.
13. Select Connect to Red Hat.



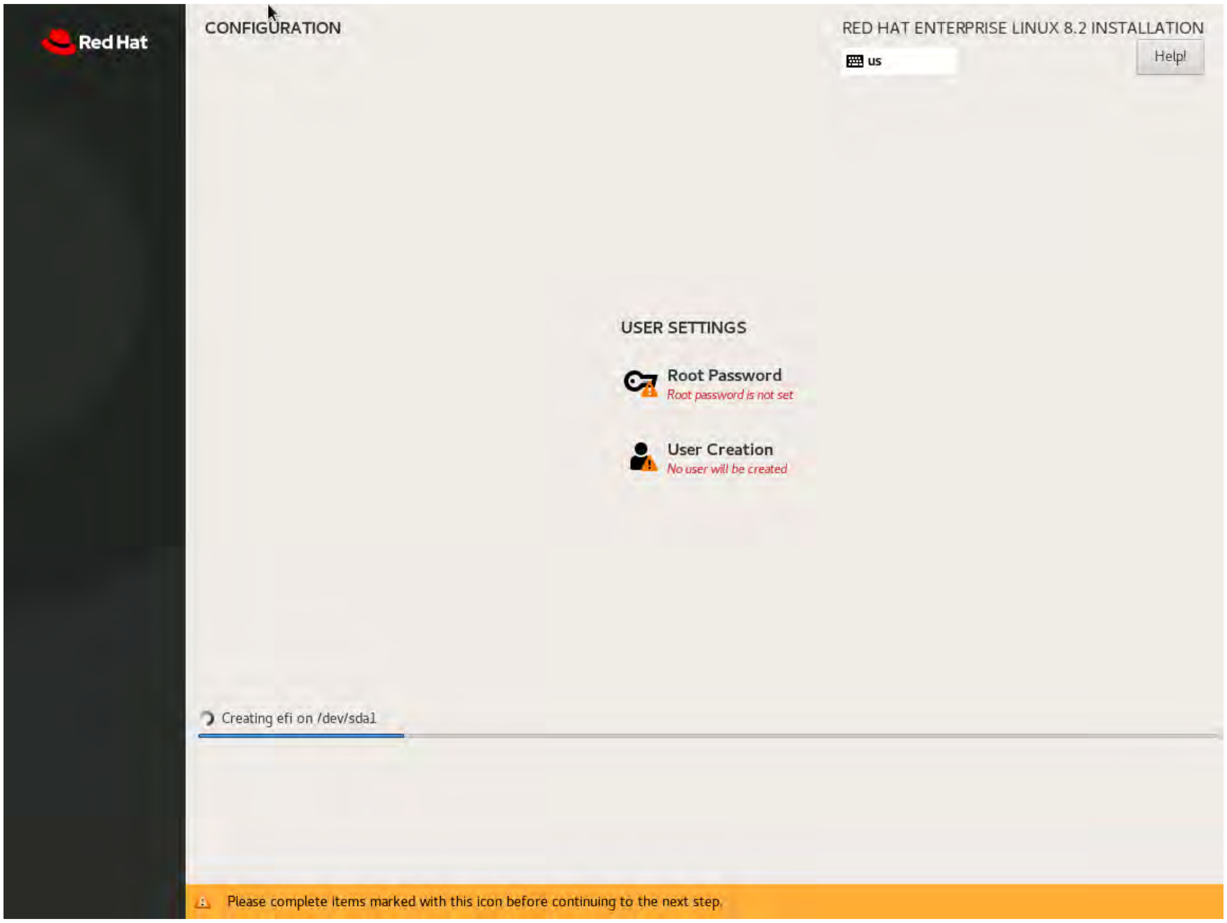
14. You must now register this system with Red Hat's entitlement server to receive updates and packages from their repo. Enter the credentials for your Red Hat account or Red Hat Developer Program membership. Then click Register. Red Hat gives you a list of subscriptions that are available for you to attach. Choose the appropriate one, then click Next. Once you have successfully registered with Red Hat, the Connect to Red Hat interface will automatically refresh with the information below:



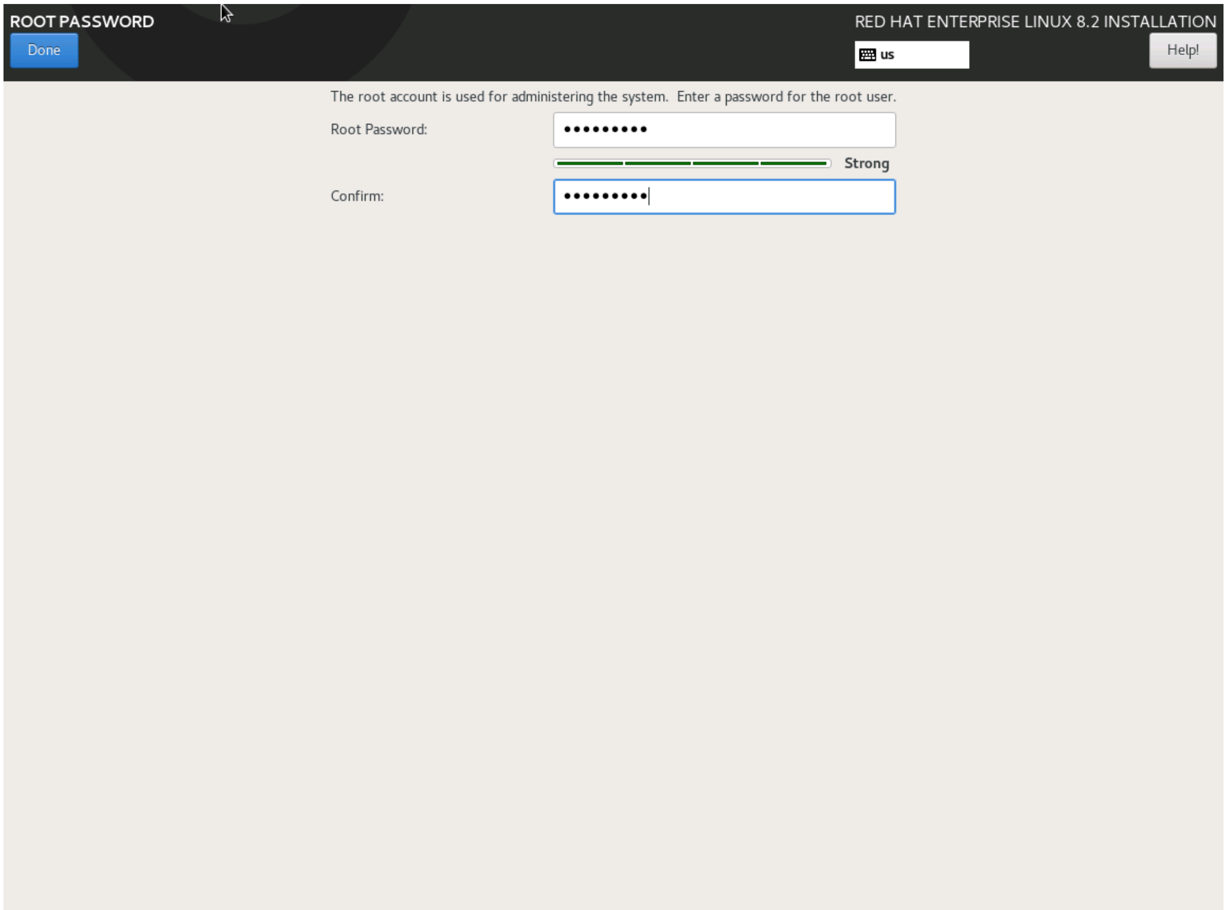
15. You are now ready to start the installation. Click Begin Installation.



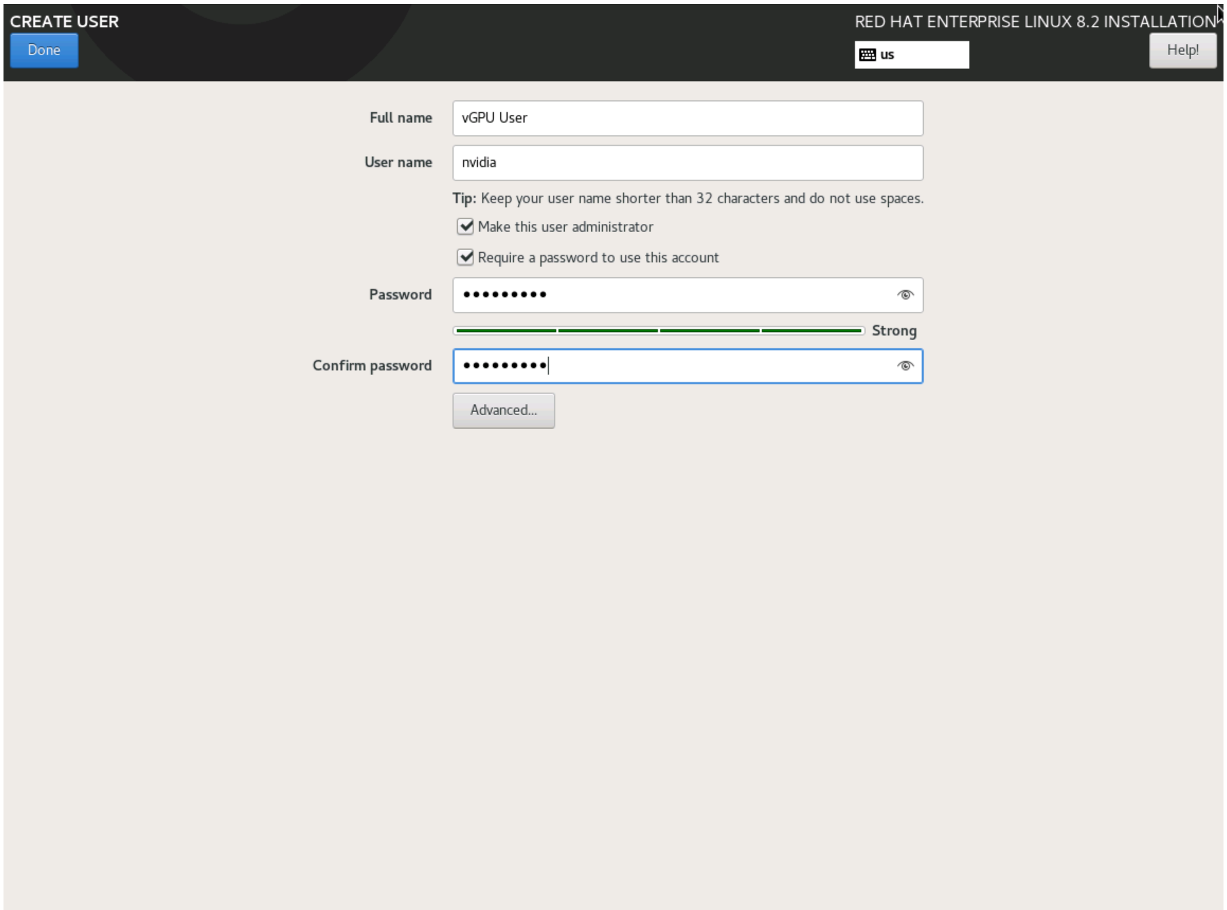
16. While you are waiting for installation to be completed, you can complete two critical tasks.



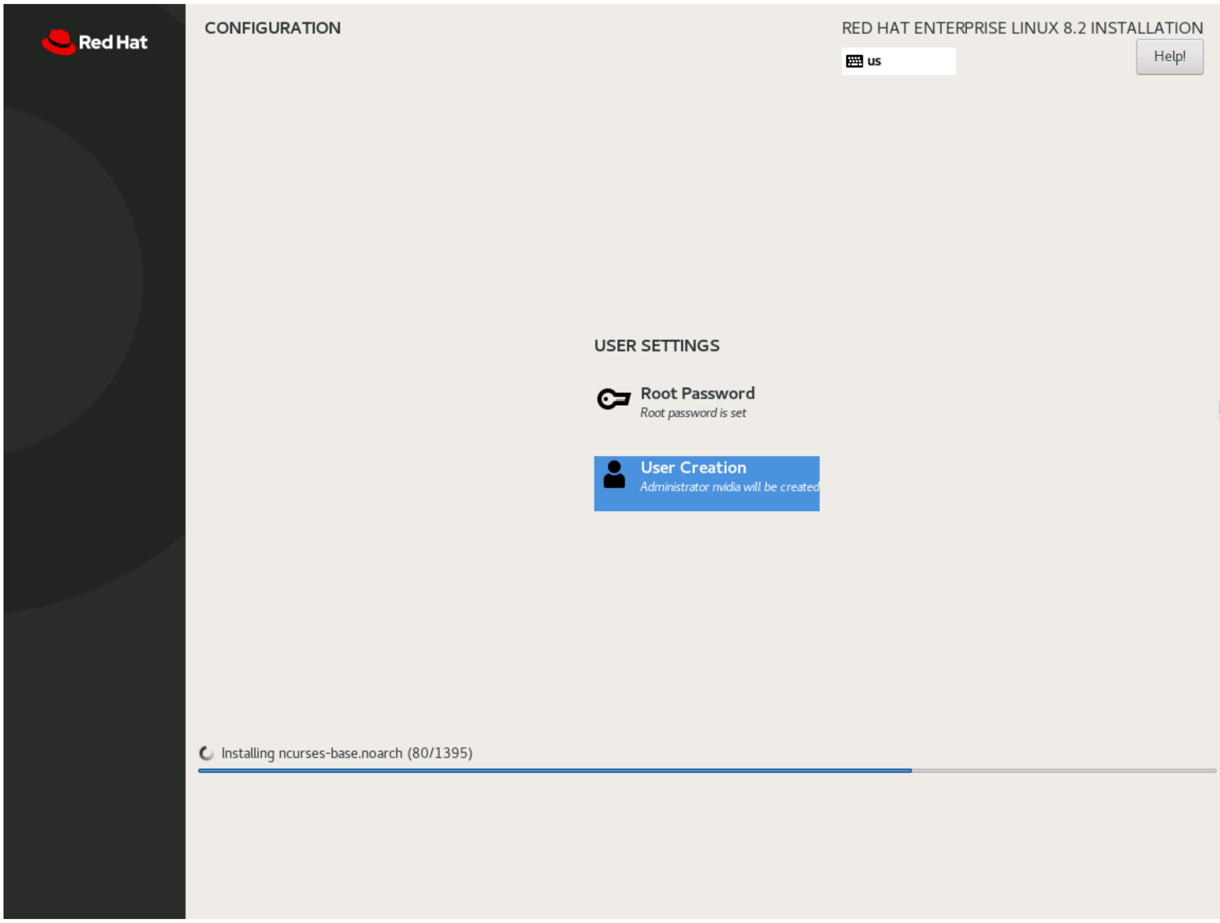
17. First, we need to set a root password. Choose a secure password. This password will grant you root privileges.



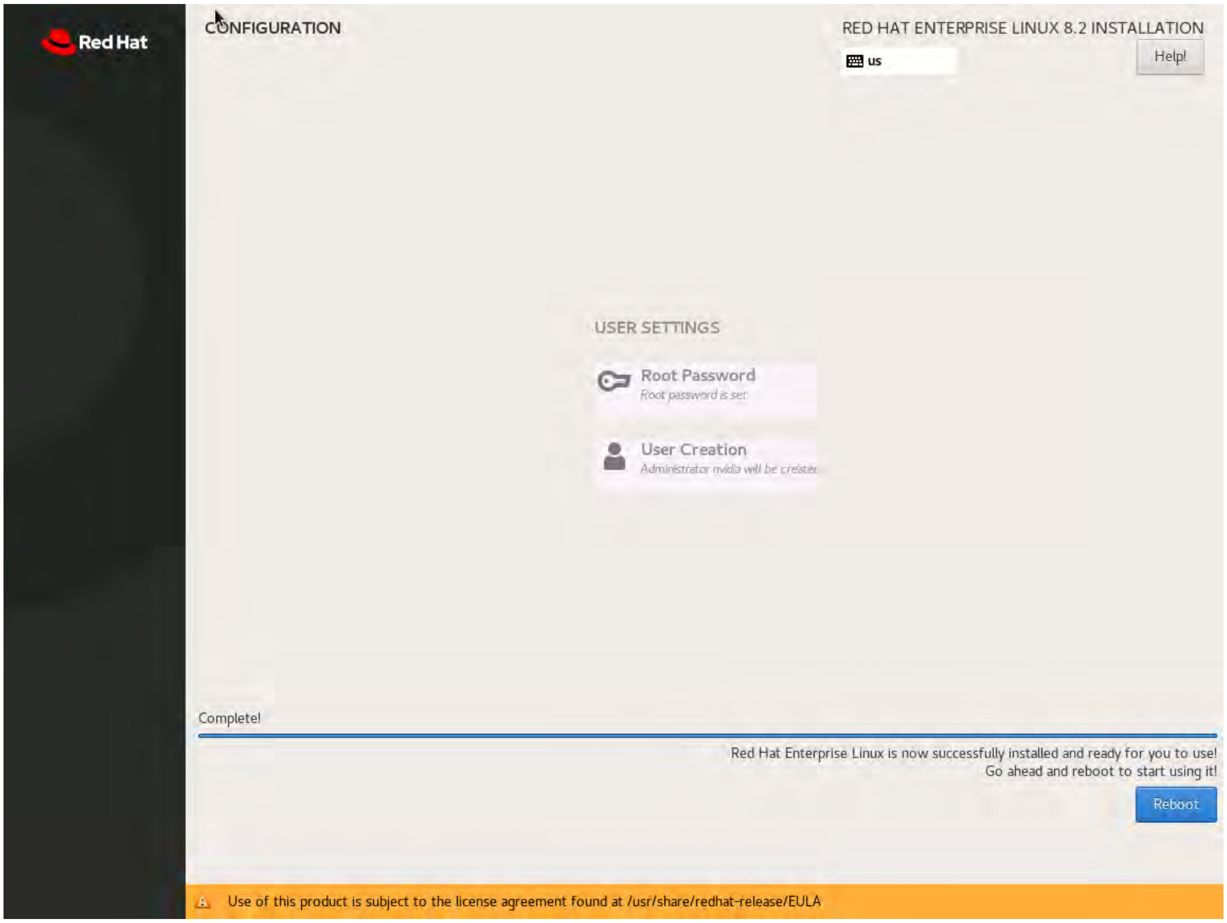
18. Second, create our primary user account.



19. Choose an appropriate username and password. Best practice is to use a different password from your root password. Make sure to check Make this user administrator and Require a password to use this account. Click Done when finished. Now we wait for the installation to finish.



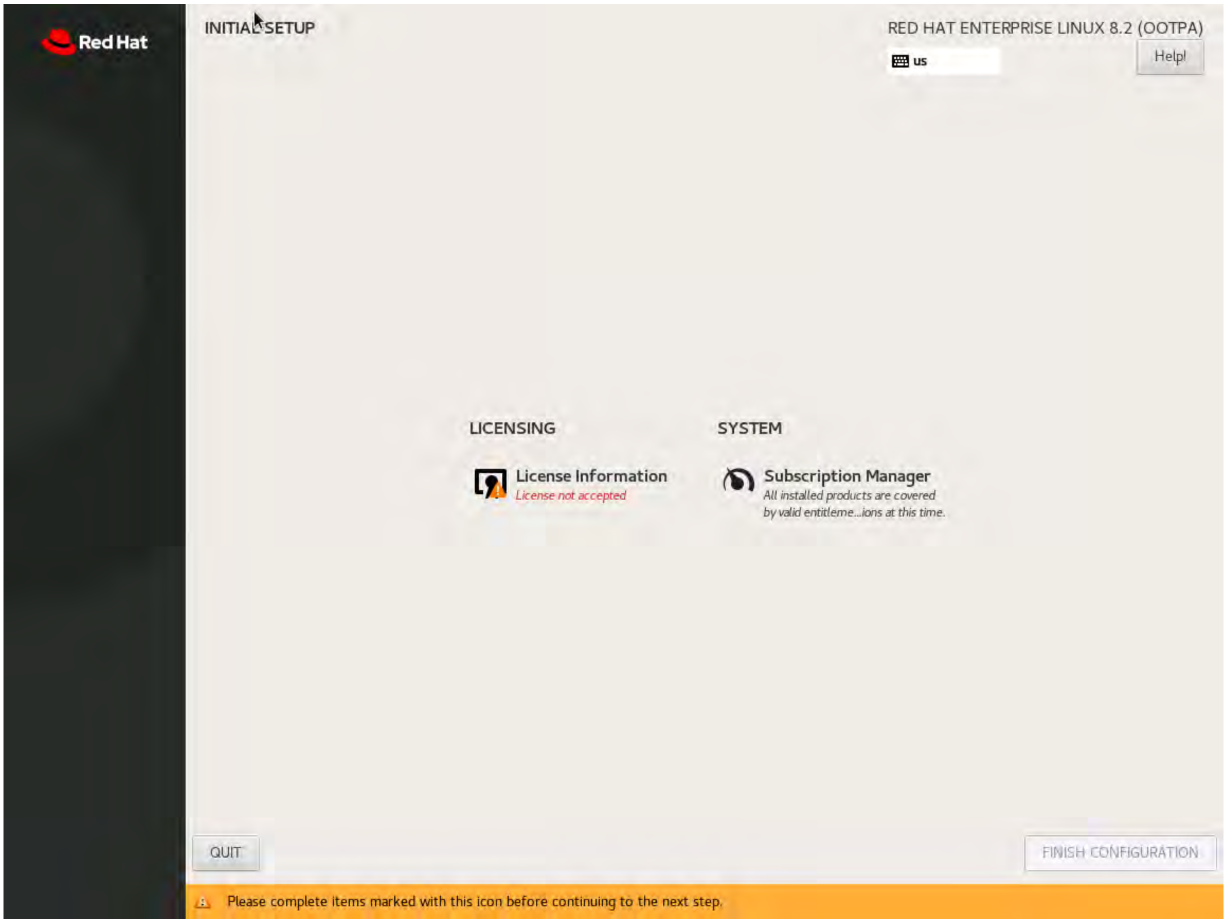
20. The installation is now complete. Click Reboot. RHEL install is complete and ready to use.



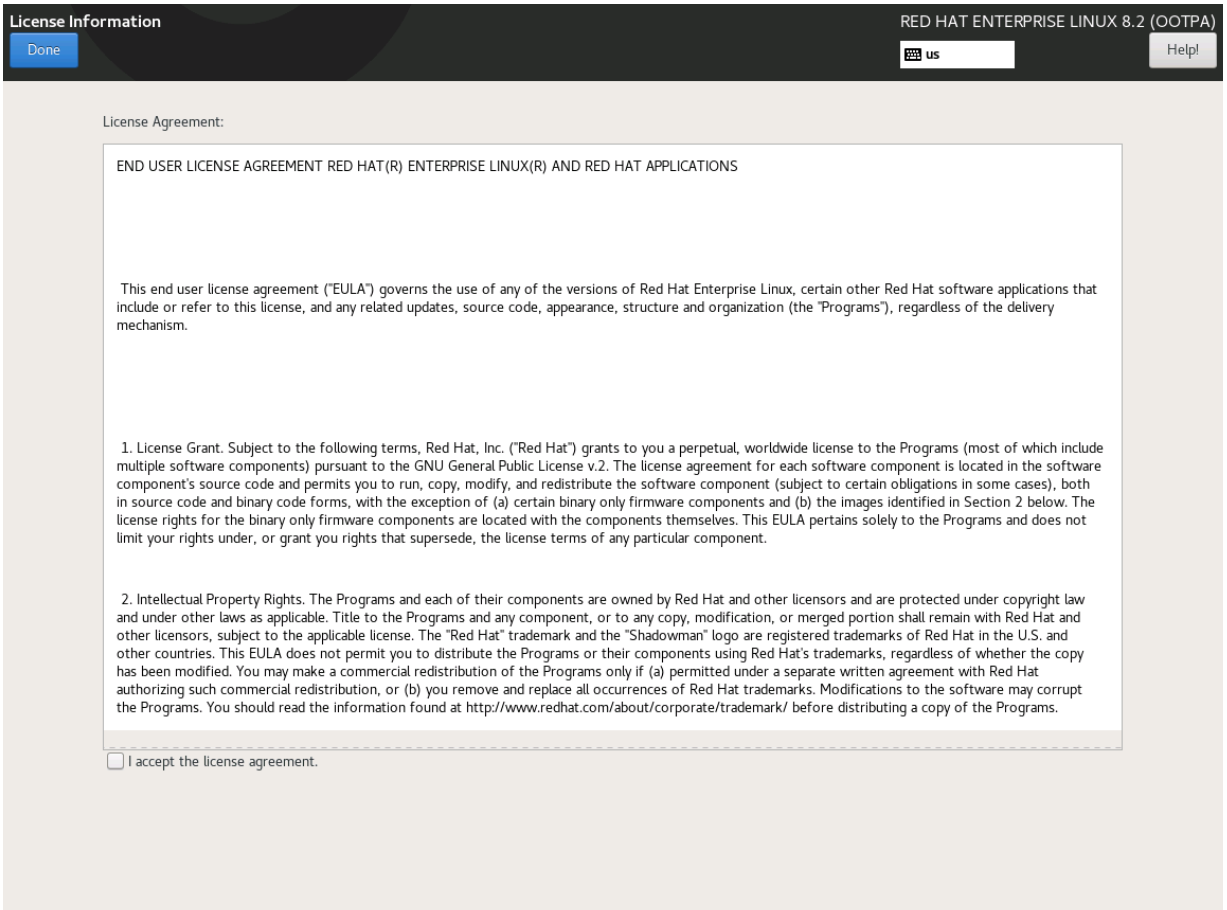
2.4 Initial Host Configuration

Now that we are finished with the installation of the host OS itself, we need to configure it for use as a virtualization host.

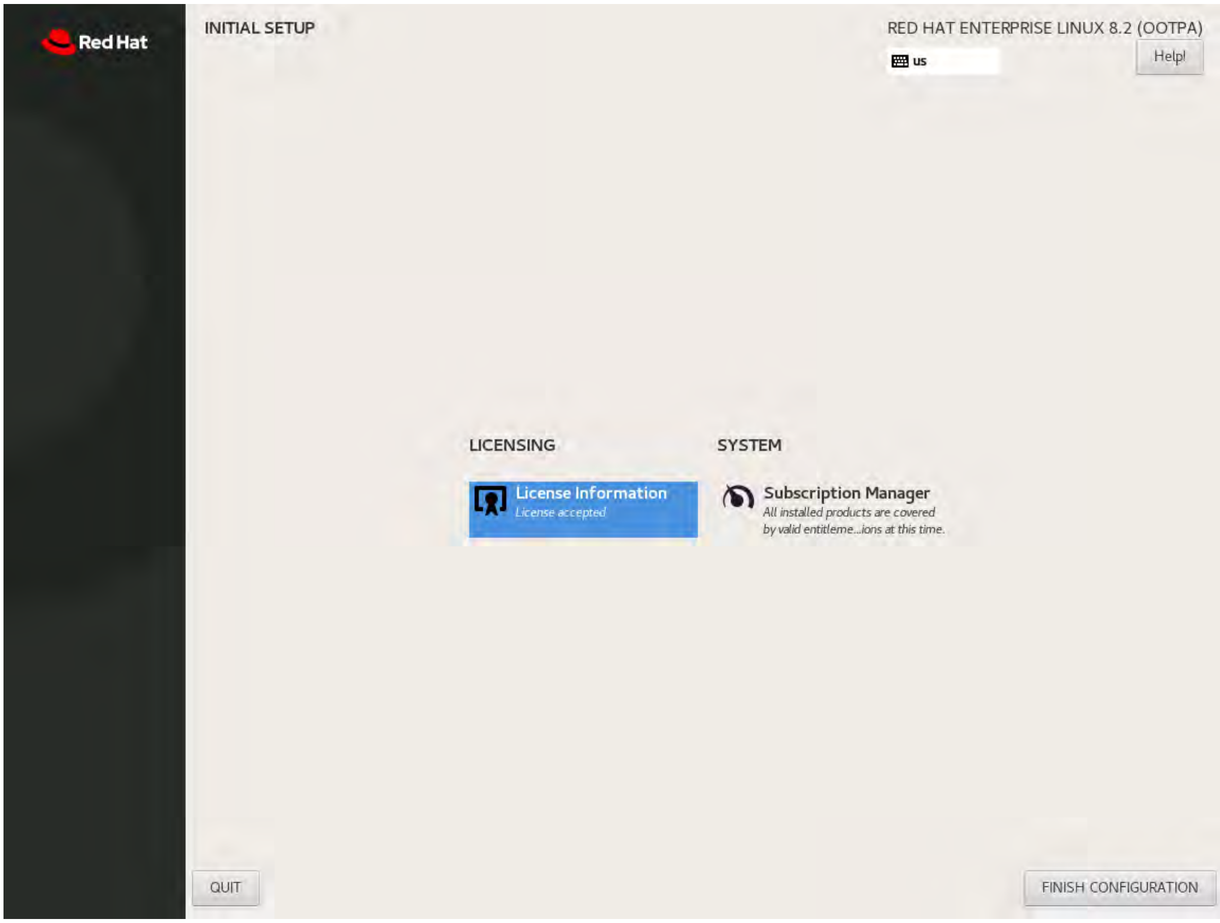
1. Once the server finishes rebooting, you will be presented with the initial setup screen.



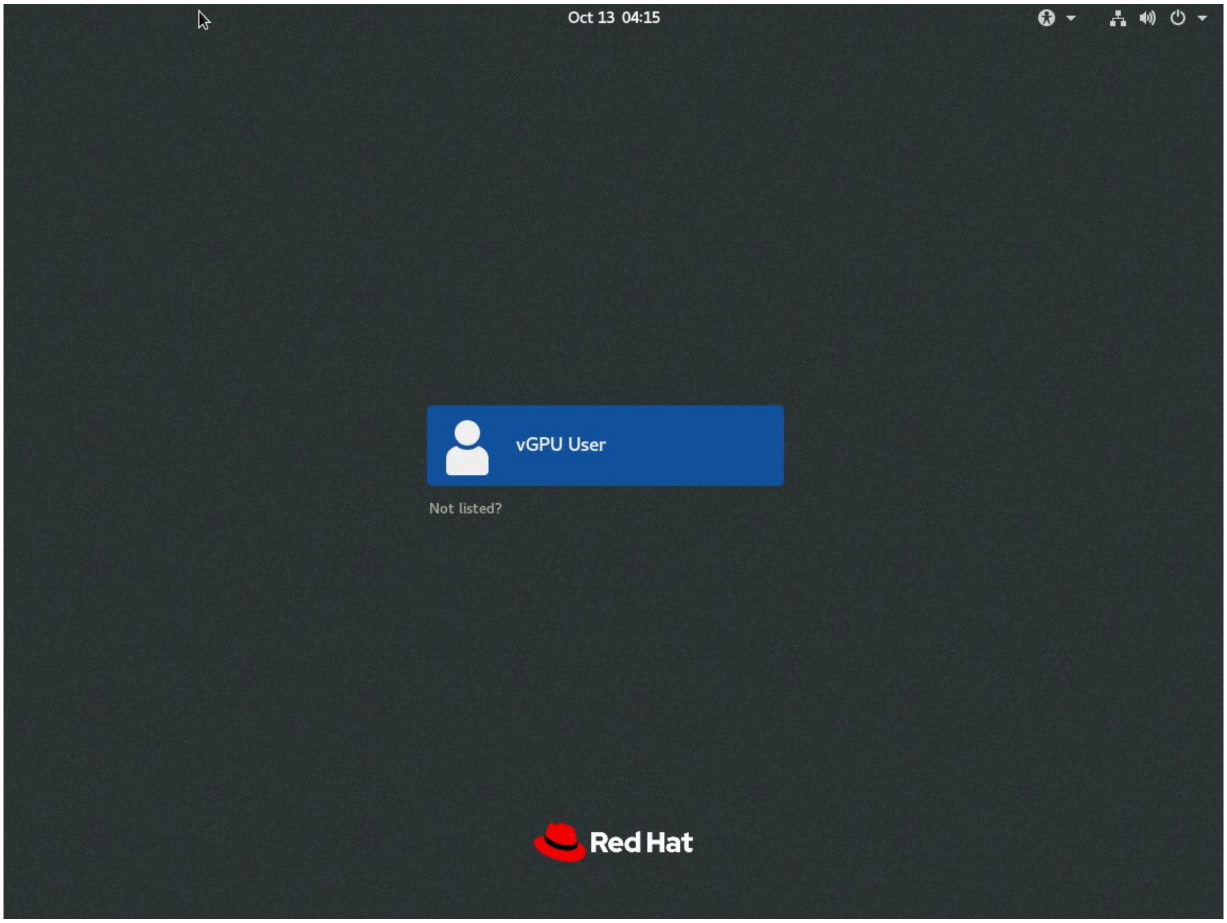
2. We need to accept the terms of the software license. Click License Information. Then accept the license agreements if you accept the terms.



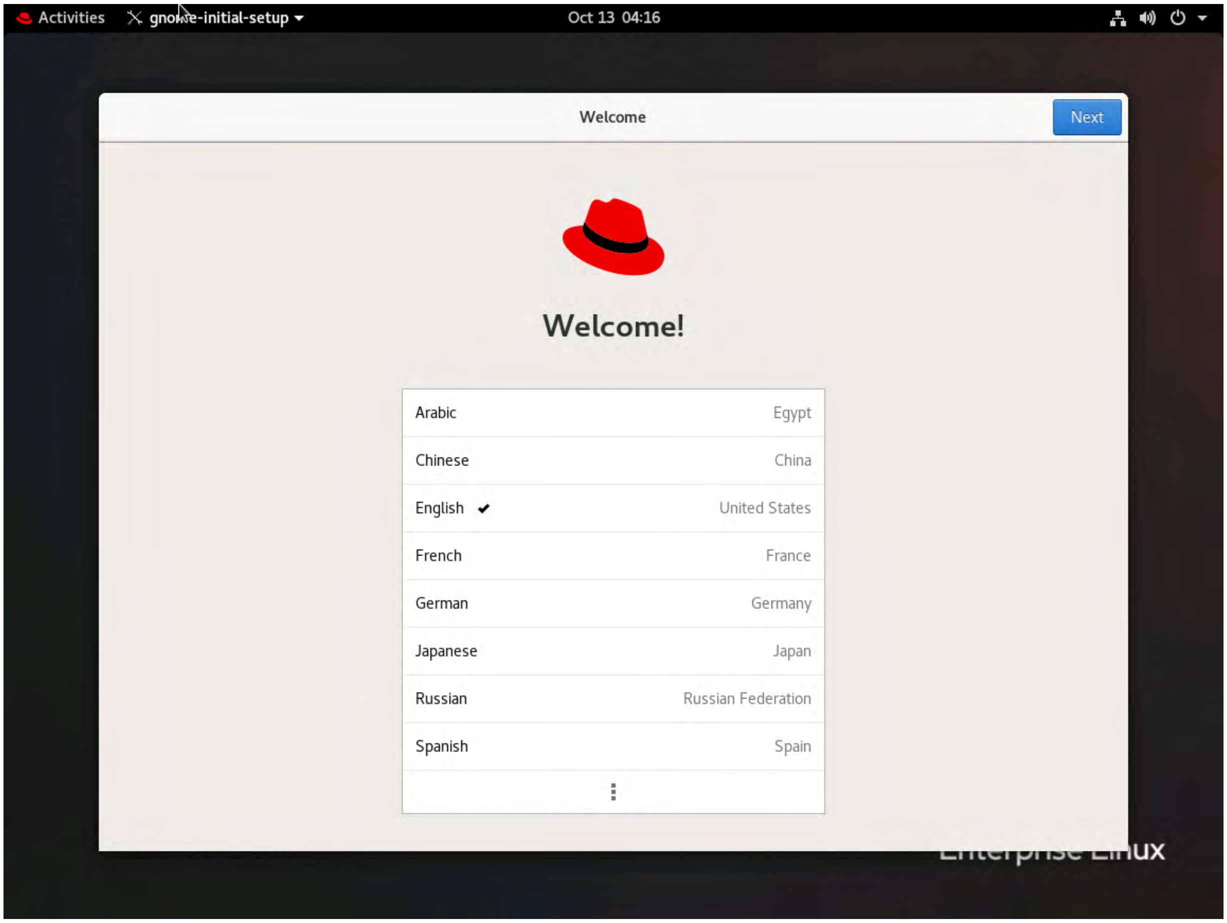
3. Click Done, then click Finish Configuration.



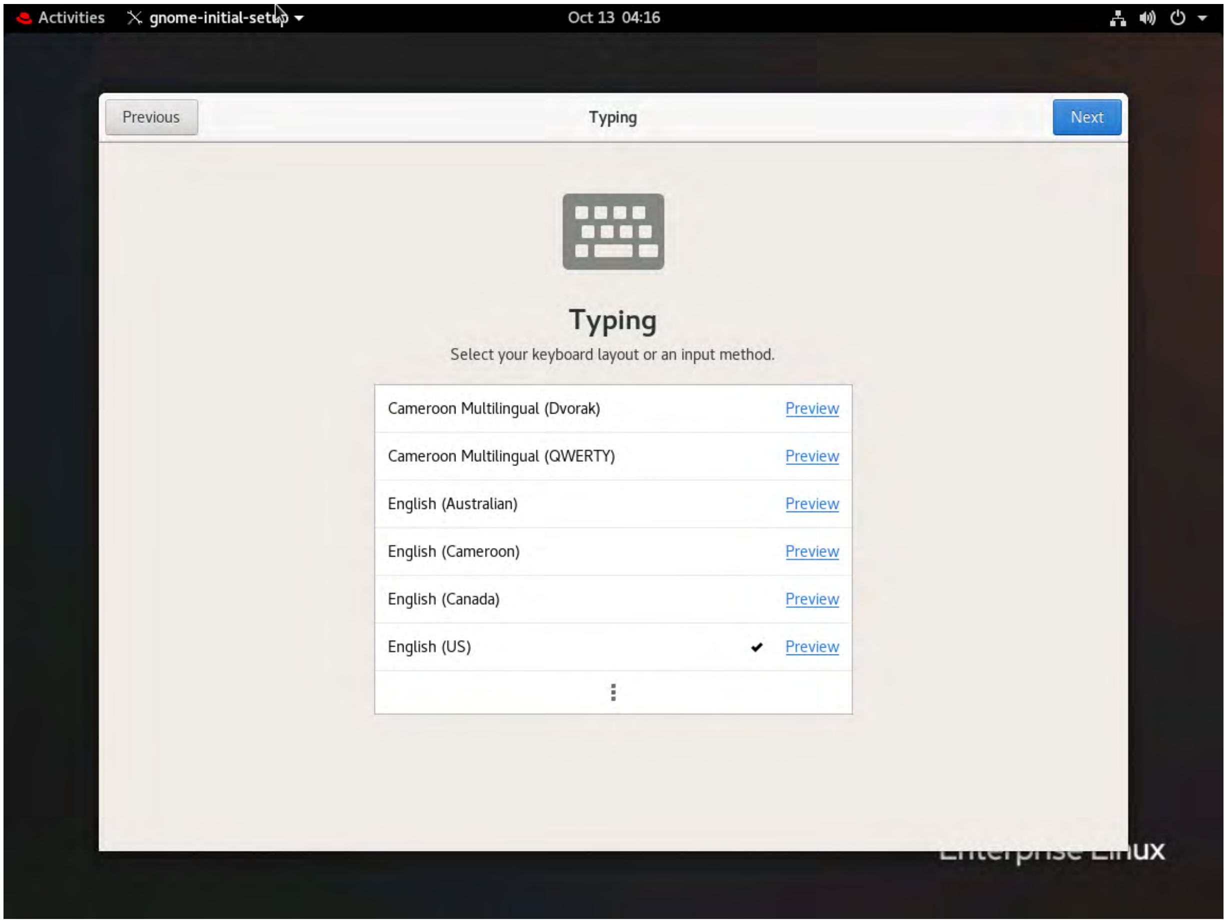
4. Once you have completed the initial setup, you will be presented with a login screen. Log in with the credentials you created in Step 2.3.18.



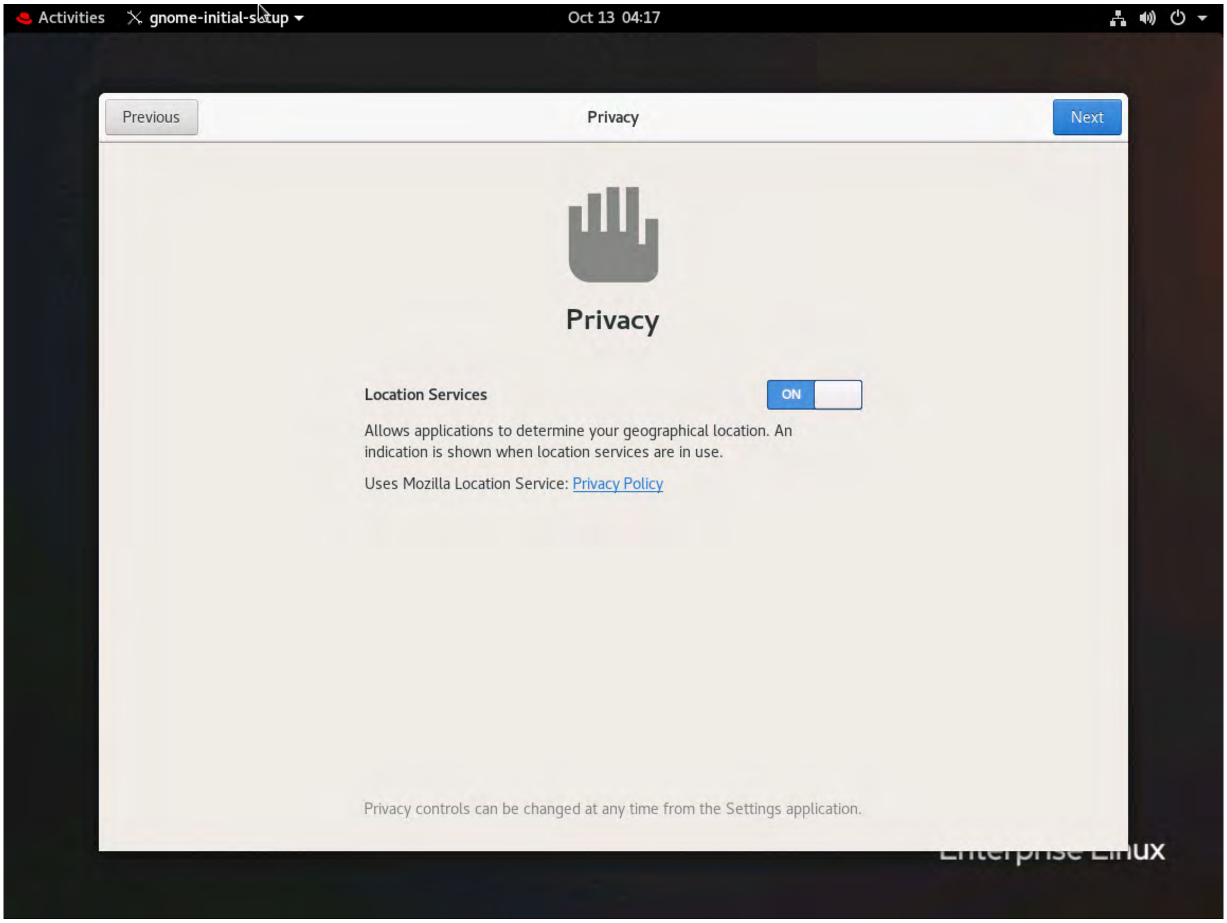
5. We will need to complete basic user account setup.



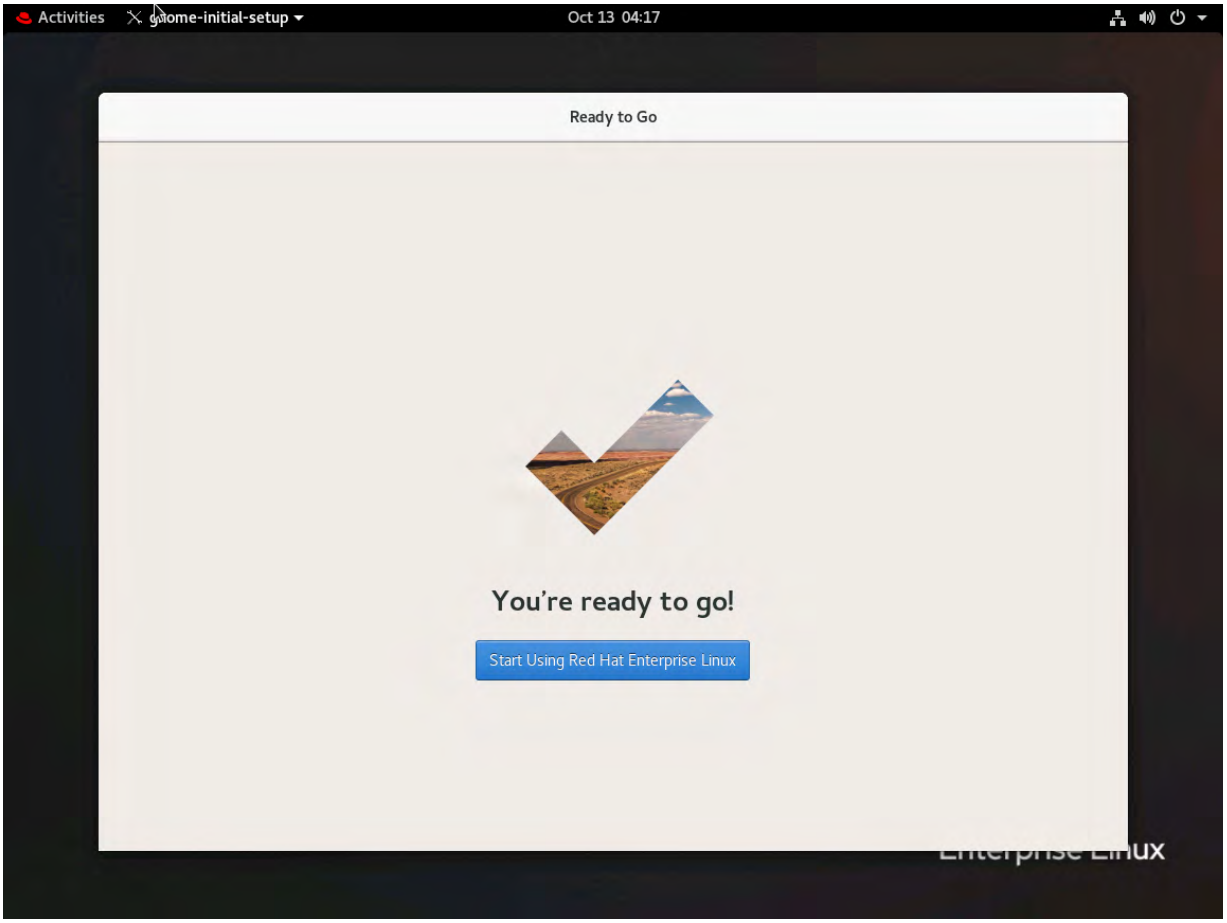
6. Select the language of your choice. This guide assumes you choose English. Click Next.



7. Select your keyboard layout of choice and click Next. This guide will proceed with the default (English US).
8. Click Next for location services.

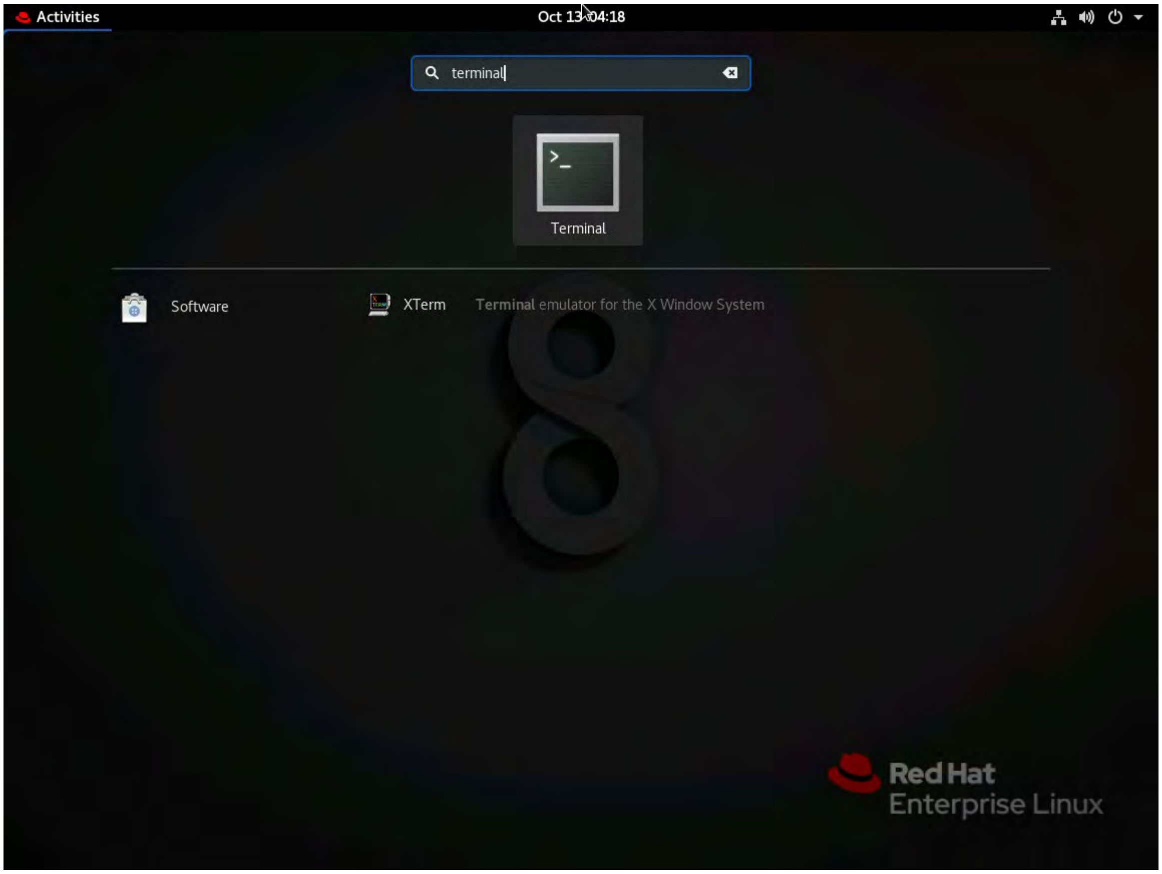


9. Click Skip to skip connecting online accounts. NVIDIA does not recommend attaching online personal accounts to a server.
10. Click Start Using Red Hat Enterprise Linux Server. Exit the Getting Started guide.



11. Run the Terminal app. You can search for it in the Activities menu.

It can be found from the Applications menu (top left of desktop) in the category System Tools.

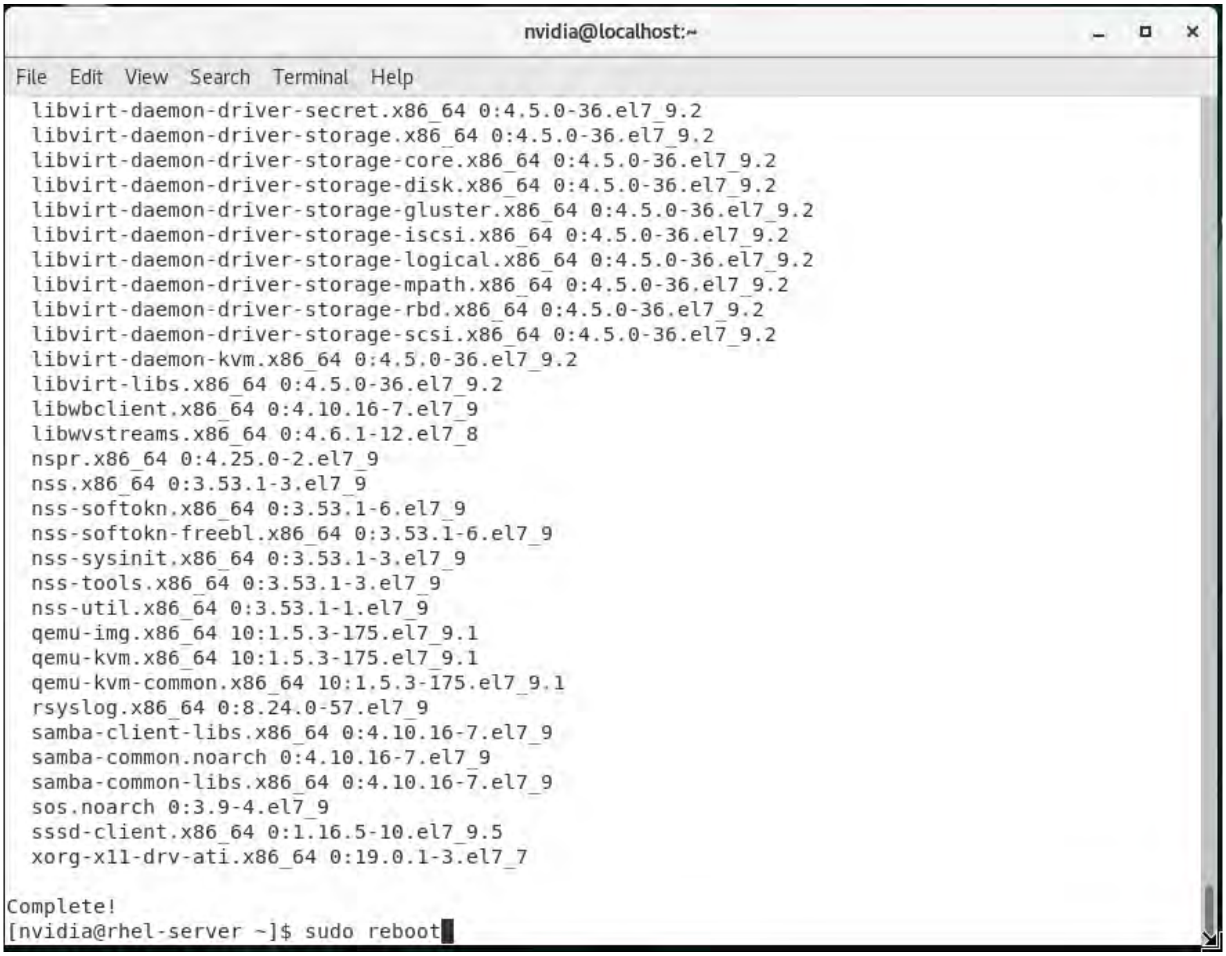


Note: The terminal is a fundamental part of system administration in Linux-based distributions like RHEL. You can find more information and links to guides in this Red Hat article.

12. Best practice is to install the latest updates, then reboot. Use the following commands. Input your root password when it is requested.

```
sudo yum check-update
sudo yum update -y
sudo reboot
```

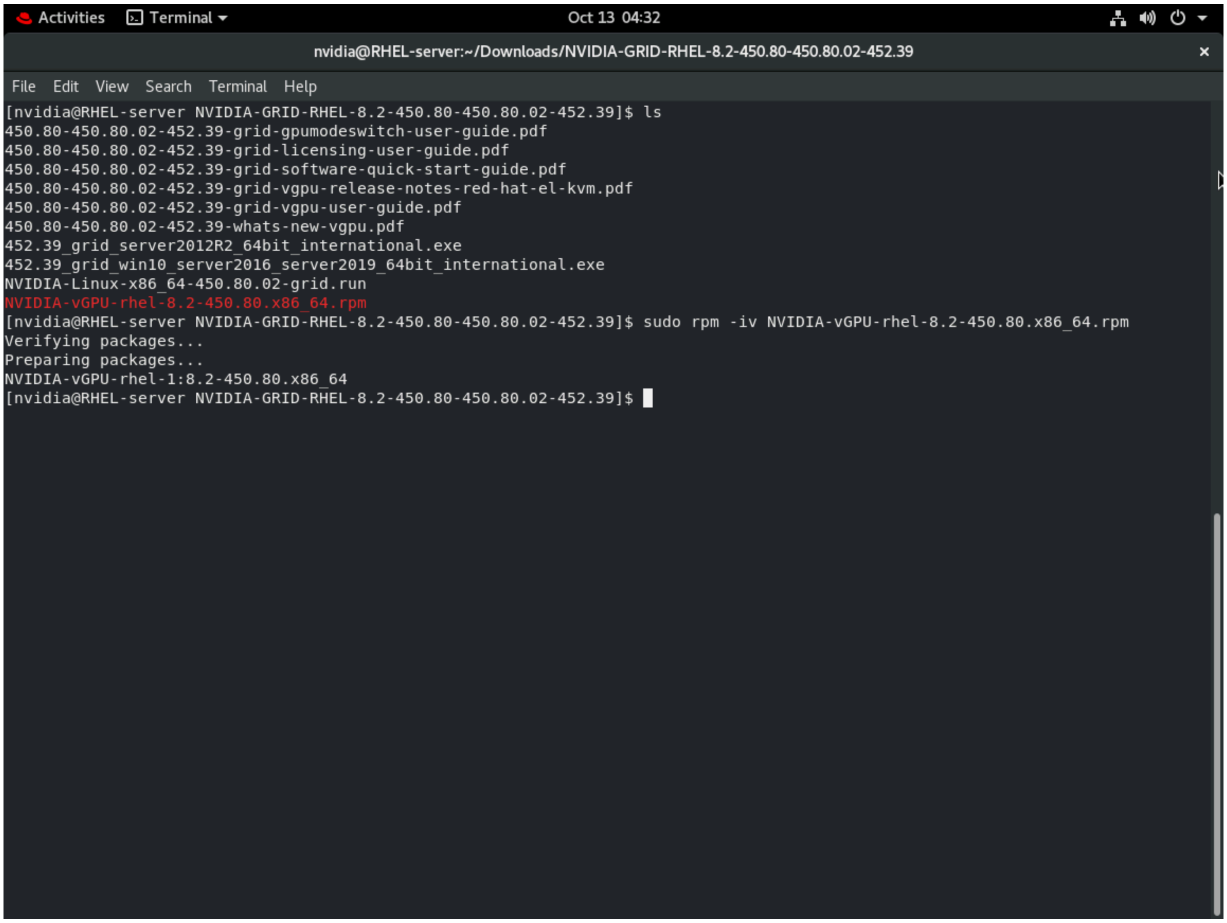
Note: The Sudo command (i.e., super user do) allows you to run commands at the superuser or root level. This is necessary for many tasks involving system level access, such as installing packages, system updates, configuration changes, and other critical functions. Sudo should only be used when you understand and trust the commands or programs being executed. You can cause damage or expose your server to security risks if you use the sudo command inappropriately or incorrectly.



13. Once the system is rebooted, log in again and reopen the Terminal app. We need to start installing the NVIDIA vGPU manager which comes in the form of an RPM package. Transfer the RPM file over your server into your working directory. Make sure you are in the same directory you uploaded the .RPM file into. Run the following commands:

```
sudo rpm -iv NVIDIA-vGPU-rhel-<version>.x86_64.rpm
```

Where <`version`> is the version number of the vGPU files you have.



Note: You can transfer the RPM file via scp if your files reside on a remote Linux machine. If you have a remote Windows machine, you can use WinSCP to transfer them (see Appendix A).

14. Once you have been returned to the command prompt, initiate another reboot.

```
sudo reboot
```

15. After the host completes its reboot, blacklist the Nouveau driver.

a) Open the blacklist.conf located in /etc/modprobe.d directory:

```
nano /etc/modprobe.d/blacklist-nouveau.conf
```

b) Add the following to the file:

```
blacklist nouveau
blacklist lbm-nouveau
options nouveau modeset=0
alias nouveau off
alias lbm-nouveau off
```

c) Next you will need to rebuild the GRUB file:

```
EFI:  grub2-mkconfig -o /boot/efi/EFI/Red Hat/grub.cfg
Legacy: grub2-mkconfig -o /boot/grub2/grub.cfg
```

d) After the GRUB file rebuild is complete, you will need to rebuild the initramfs:

```
dracut -f
```

e) The final step is to restart the server:

```
reboot
```

2.5 Verify Host Configuration

Now that all configuration is complete, we need to verify that everything is configured properly and that all necessary hardware is detected.

1. Verify that the libvirtd service is active and running.

```
Ssystemctl status libvirtd
```

2. Verify that the vGPU Manager package is installed correctly.

```
lsmod | grep nvidia_vgpu_vfio
```

Sample output:

```
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
```

3. Verify that your Nvidia GPU(s) are detected correctly.

```
nvidia-smi
```

Sample output:

```
[root@vgpu10:~] nvidia-smi
Wed Jan 13 19:48:05 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.55       Driver Version: 450.55       CUDA Version: N/A      |
|-------------------------------+----------------------+----------------------+
| 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  Tesla T4            On   | 00000000:81:00.0 Off |                  Off |
| N/A   33C    P8    15W /  70W |     79MiB / 16383MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            On   | 00000000:C5:00.0 Off |                  Off |
| N/A   31C    P8    15W /  70W |     79MiB / 16383MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+


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

Chapter 3. vGPU Configuration and Policies

This chapter covers configuring the NVIDIA vGPU Manager:

- Locating the hardware information of the physical GPU
- Creating a vGPU
- Changing the vGPU Scheduling Policy
- Disabling and enabling ECC memory on the vGPU

3.1 Getting the BDF and Domain of a GPU

Now that our host is installed, we must create the vGPU instance(s). To do so, we'll need to get the hardware information of our physical GPUs.

Sometimes when configuring a physical GPU for use with NVIDIA vGPU software, you must find out which directory in the sysfs file system represents the GPU. This directory is identified by the domain, bus, slot, and function of the GPU. For more information about the directory in the sysfs file system represents a physical GPU, see NVIDIA vGPU Information in the sysfs File System.

1. Open Terminal
2. Obtain the PCI device bus/device/function (BDF) of the physical GPU via command:

```
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 T4]
(rev a1)
07:00.0 VGA compatible controller: NVIDIA Corporation GM204GL [Tesla T4]
(rev a1)
```

3. Obtain the full identifier of the GPU from its PCI device BDF via command:

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

The `<transformed-bdf>` is 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
```

4. 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'/>
```

3.2 Creating the vGPU Instance(s)

Once you have the hardware information for the physical GPU, we can create the vGPU.

For each vGPU that you want to create, you will need to perform these steps from Terminal on the Red Hat Enterprise Linux KVM host. Please note that the mdev device file that you create to represent the vGPU does not persist when the host is rebooted but must be recreated after each reboot. If necessary, you can use standard features of the operating system to automate the creation of this device file when the host is booted, for example, by writing a custom script that is executed when the host is rebooted. Before you begin, ensure that you have the domain, bus, slot, and function of the GPU on which you are creating the vGPU.

1. Change to the mdev_supported_types directory for the physical GPU.
   ```
   # cd
   /sys/class/mdev_bus/[domain]\:[bus]\:slot.[function]/mdev_supported_types
   /
   ```
   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 contain registration information for the vGPU type that you want to create.

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

Replace [vgpu-type] with 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
```

Replace [subdirectory] with the directory 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 using uuidgen.

```
# 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
```

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

The [subdirectory] is the registration information directory for the vGPU type that you want to create, for example, `nvidia-41`. This example creates an instance of the T4-8Q vGPU type with the UUID aa618089-8b16-4d01-a136-25a0f3c73123:

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

An mdev device file for the vGPU is added is added to the parent physical device directory of the vGPU. The vGPU is identified by its UUID. The `/sys/bus/mdev/devices/` directory contains a symbolic link to the mdev device file.

6. Confirm that the vGPU was created with `ls -l /sys/bus/mdev/devices/`.

```
# 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
```

7. This vGPU is now ready for use. We will attach it to a Virtual Machine once the VM is created.

3.3 Changing the vGPU Scheduling Policy

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

GPUs, starting with the NVIDIA® Pascal™ architecture, also support equal-share and fixed-share vGPU schedulers. These schedulers impose a control on GPU processing cycles used by a vGPU which prevents graphics-intensive applications running in one VM from affecting the performance of graphics-light applications running in other VMs. The best-effort scheduler is the default scheduler for all supported GPU architectures.

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

The GPUs that are based on the Volta™ architecture are the NVIDIA V100 SXM2, NVIDIA V100 PCIe, NVIDIA V100 FHHL, and NVIDIA V100s.

The GPUs that are based on the Turing™ architecture are the NVIDIA T4, RTX 6000 and RTX 8000.

The GPUs that are based on the Ampere™ architecture are the NVIDIA A100, and A40.

3.3.1 vGPU Scheduling Policies

In addition to the default best effort scheduler, GPUs based on the Pascal and Volta architectures support equal share and fixed share vGPU schedulers.

- **Fixed share scheduling** always guarantees the same dedicated quality of service. The fixed share scheduling policies guarantee equal GPU performance across all vGPUs sharing the same physical GPU. Dedicated quality of service simplifies a POC since it allows the use of common benchmarks used to measure physical workstation performance such as SPECviewperf, to compare the performance with current physical or virtual workstations.

- **Best effort scheduling** provides consistent performance at a higher scale and therefore reduces the TCO per user. The best effort scheduler leverages a round-robin scheduling algorithm which shares GPU resources based on actual demand which results in optimal utilization of resources. This results in consistent performance with optimized user density. The best effort scheduling policy best utilizes the GPU during idle and not fully utilized times, allowing for optimized density and a good QoS.
- **Equal share scheduling** provides equal GPU resources to each running VM. As vGPUs are added or removed, the share of GPU processing cycles allocated changes, accordingly, resulting in performance to increase when utilization is low, and decrease when utilization is high.

3.3.2 RmPVMRL Registry Key

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

> Note: You can change the vGPU scheduling policy only on GPUs based on the Pascal, Volta, Turing, and Ampere architectures.

Type

`Dword`

Contents

Value	Meaning
0x00 (default)	Best effort scheduler
0x01	Equal share scheduler with the default time slice length
0x00TT0001	Equal share scheduler with a user-defined time slice length TT
0x11	Fixed share scheduler with the default time slice length
0x00TT0011	Fixed share scheduler with a user-defined time slice length TT

Examples

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

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

TT

- Two hexadecimal digits in the range 01 to 1E that set the length of the time slice in milliseconds (ms) for the equal share and fixed share schedulers. The minimum length is 1 ms and the maximum length is 30 ms.
- If *TT* is 00, the length is set to the default length for the vGPU type.

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

Examples

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

```
RmPVMRL=0x01
```

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

```
RmPVMRL=0x00030001
```

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

```
RmPVMRL=0x11
```

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

```
RmPVMRL=0x00180011
```

3.3.3 Changing the vGPU Scheduling Policy for All GPUs

> Note: You can change the vGPU scheduling policy only on GPUs based on the Pascal, Volta, Turing, and Ampere architectures.

Perform this task in your hypervisor command shell.

1. Open a command shell as the root user on your hypervisor host machine. On all supported hypervisors, you can use secure shell (SSH) for this purpose. Set the `RmPVMRL` registry key to specify the GPU scheduling policy you want.
2. Add an entry to the `/etc/modprobe.d/nvidia.conf` file.

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

Where <value> ***is*** the value that sets the vGPU scheduling policy you want, for example:

a) **0x00 -** Equal Share Scheduler with the default time slice length

b) **0x00030001** - Equal Share Scheduler with a time slice of 3 ms

c) **0x011 -** Fixed Share Scheduler with the default time slice length

d) **0x00180011** - Fixed Share Scheduler with a time slice of 24 ms (0x18)

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

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

For all supported values, see RmPVMRL Registry Key.

3. Reboot your hypervisor host machine.

3.3.4 Changing the vGPU Scheduling Policy for Selected GPUs

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

Perform this task in your hypervisor command shell.

1. Open a command shell as the root user on your hypervisor host machine. On all supported hypervisors, you can use secure shell (SSH) for this purpose.
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. 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:
   ```

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

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

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

4. 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...]"
   ```

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`
  - **0x00** - Sets the vGPU scheduling policy to Equal Share Scheduler with the default time slice length.
  - **0x00030001** - Sets the vGPU scheduling policy to Equal Share Scheduler with a time slice that is 3ms long.
  - **0x011** - Sets the vGPU scheduling policy to Fixed Share Scheduler with the default time slice length.
  - **0x00180011** - Sets the vGPU scheduling policy to Fixed Share Scheduler with a time slice that is 24 ms (0x18) 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 two GPUs as follows:

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

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

5. Reboot your hypervisor host machine.

3.3.5 Restoring Default vGPU Scheduler Settings

Perform this task in your hypervisor command shell.

1. Open a command shell as the root user on your hypervisor host machine. On all supported hypervisors, you can use secure shell (SSH) for this purpose.

2. Unset the `RmPVMRL` registry key by commenting out the entries in the `/etc/modprobe.d/nvidia.conf` file that set `RmPVMRL` by prefixing each entry with the # character.

3. Reboot your hypervisor host machine.

3.4 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 may 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 vGPUs 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 each 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. These GPUs are supplied with ECC memory enabled. 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.

3.4.1 Disabling ECC Memory

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

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

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

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

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

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 Jul 13 18:36:45 2020
Driver Version                      : 450.55

Attached GPUs                       : 1
GPU 0000:02:00.0

[...]

    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled

[...]
```

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

a) If you want to change the ECC status to off for all GPUs on your host machine or for vGPUs assigned to the VM, run this command:

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

b) 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
```

c) Where <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
```

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

```
# nvidia–smi –q

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

Timestamp                           : Mon Jul 13 18:37:53 2020
Driver Version                      : 450.55

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

    Ecc Mode
        Current                     : Disabled
        Pending                     : Disabled

[...]
```

3.4.2 Enabling ECC Memory

If ECC memory is suitable for your workloads and is supported by your hypervisor software and GPUs, but is disabled on your GPUs or vGPUs, enable it.
Where to perform this task depends on whether you are changing ECC memory settings for a physical GPU or a vGPU.

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

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

Before you begin, ensure that NVIDIA Virtual GPU Manager is installed on your hypervisor. If you are changing ECC memory settings for a vGPU, also ensure that the NVIDIA 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 Jul 13 18:36:45 2020
Driver Version                      : 450.55

Attached GPUs                       : 1
GPU 0000:02:00.0

[...]

    Ecc Mode
        Current                     : Disabled
        Pending                     : Disabled

[...]
```

2. Change the ECC status to "Disabled" for each GPU or vGPU for which ECC is enabled.
   a) 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
   ```
   b) 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
   ```
   c) <id> is the index of the GPU or vGPU as reported by nvidia-smi.
   d) 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 Jul 13 18:37:53 2020
Driver Version                      : 450.55

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

    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled

[...]
```

Chapter 4. Deploying the NVIDIA vGPU Software License Server

This chapter covers deployment of the NVIDIA vGPU software license server, including:

- Platform Requirements
- Installing the Java Runtime Environment on Windows
- Installing the License Server Software on Windows

4.1 Platform Requirements

Before proceeding, ensure that you have a platform suitable for hosting the license server.

4.1.1 Hardware and Software Requirements

- The hosting platform may be a physical machine, an on-premises virtual machine (VM), or a VM on a supported cloud service. NVIDIA recommends using a host that is dedicated solely to running the license server.
- The recommended minimum configuration is 2 CPU cores and 4 GB of RAM. A high-end configuration of 4 or more CPU cores with 16 GB of RAM is suitable for handling up to 150,000 licensed clients.
- At least 1 GB of hard drive space is required.
- The hosting platform must run a supported operating system.
- On Window platforms, .NET Framework 4.5 or later is required.

4.1.2 Platform Configuration Requirements

- The platform must have a fixed (unchanging) IP address. The IP address may be assigned dynamically by DHCP or statically configured but must be constant.
- The platform must have at least one fixed Ethernet MAC address, to be used as a unique identifier when registering the server and generating licenses in the NVIDIA Licensing Portal.
- The platform's date and time must be set accurately. NTP is recommended.

4.1.3 Network Ports and Management Interface

The license server requires TCP port 7070 to be open in the platform's firewall to serve licenses to clients. By default, the installer automatically opens this port. The license server's management interface is web-based and uses TCP port 8080. The management interface itself does not implement access control; instead, the installer does not open port 8080 by default, so that the management interface is only available to web browsers running locally on the license server host. Access to the management interface is therefore controlled by limiting remote access (via VNC, RDP, etc.) to the license server platform.

> Note: If you choose to open port 8080 during license server installation, or at any time afterwards, the license server's management interface is unprotected.

4.2 Installing the NVIDIA vGPU Software License Server on Windows

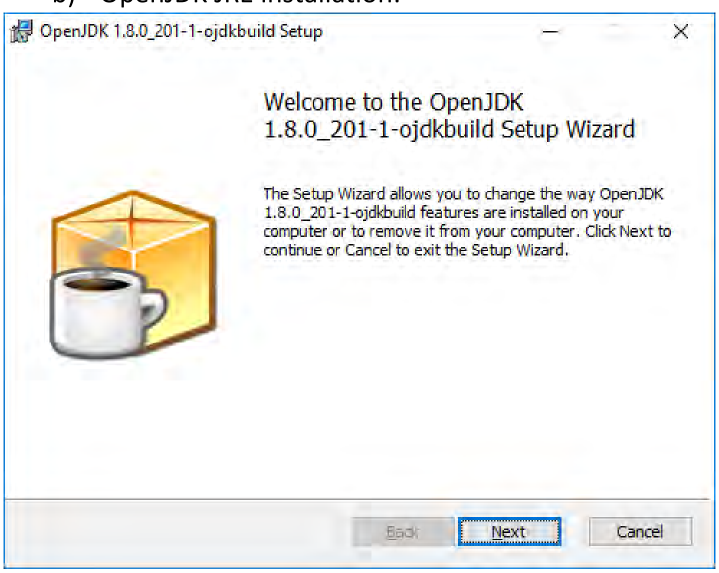
The license server requires a Java runtime environment, which must be installed separately before you install the license server.

4.2.1 Installing the Java Runtime Environment on Windows

If a suitable Java runtime environment (JRE) version is not already installed on your system install a supported JRE before running the NVIDIA license server installer.

1. Download a supported 64-bit JRE, either Oracle Java SE JRE or OpenJDK JRE.
   a) Download Oracle Java SE JRE from the Java Downloads for All Operating Systems page.
   b) Download Oracle Java SE JRE from the java.com: Java + You page
   c) Download OpenJDK JRE from the Community builds using source code from OpenJDK project on GitHub.
2. Install the JRE that you downloaded.
   a) Oracle Java SE JRE installation:

b) OpenJDK JRE installation:



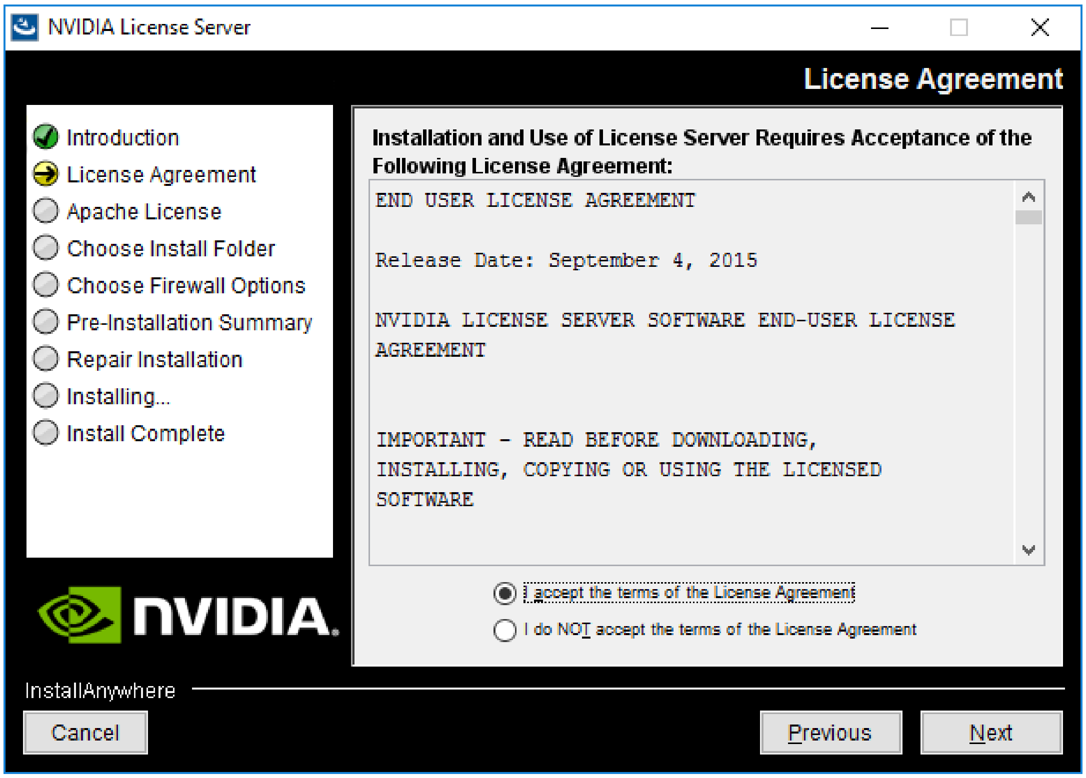
3. Set the JAVA_HOME system variable to the full path to the “jre…” folder of your JRE installation.
   a) **For 64-bit Oracle Java SE JRE**: `C:\Program Files\Java\jre1.8.0_191`
   b) **For 64-bit OpenJDK JRE**: `C:\Program Files\ojdkbuild\java-1.8.0-openjdk-1.8.0.201-1\jre`

   Ensure that the path does not include any trailing characters, such as a slash or a space.
   If you are upgrading to a new version of the JRE, update the value of `JAVA_HOME` to the full path to the jre folder of your new JRE version.
4. Ensure that the `path` system variable contains the path to the `java.exe` executable file.
   a) **For 64-bit Oracle Java SE JRE**: `C:\Program Files\Java\jre1.8.0_191\bin`

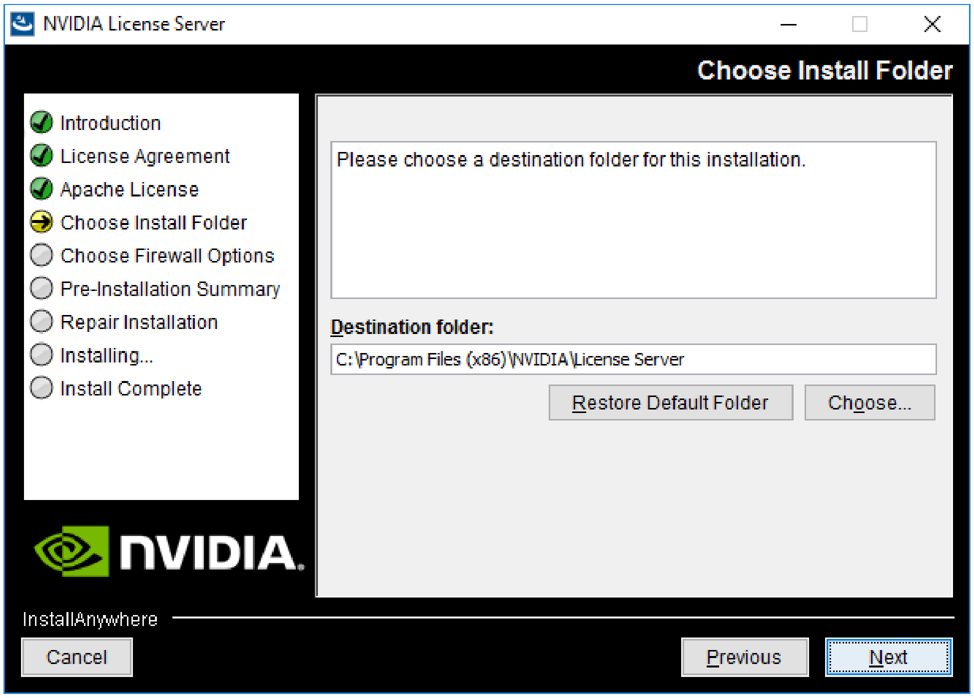
b) **For 64-bit OpenJDK JRE**: `C:\Program Files\ojdkbuild\java-1.8.0-openjdk-1.8.0.201-1\bin`

4.2.2 Installing the License Server Software on Windows

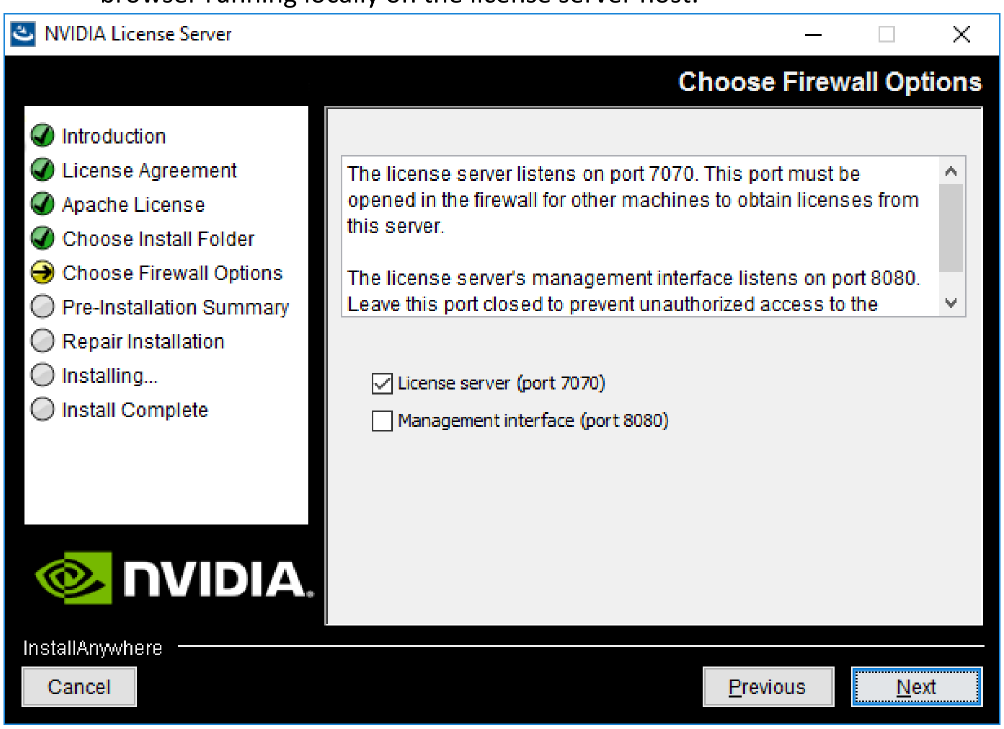
1. Unzip the license server installer and run `setup.exe`.
2. Accept the EULA for the license server software and the Apache Tomcat software used to support the license server's management interface.



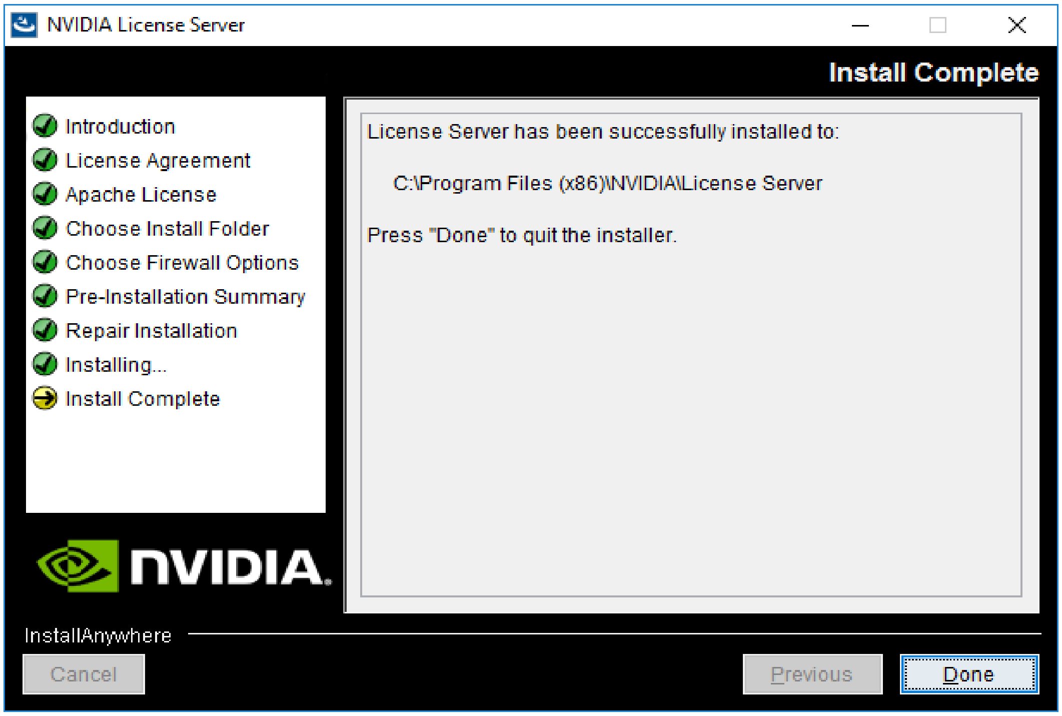
3. Choose the destination folder where you want the license server software to be installed.



4. In the Choose Firewall Options dialog box, select the ports to be opened in the firewall.
   To enable remote clients to access licenses from the server and prevent remote access to the management interface, use the default settings:
   a) Port 7070 is open to enable remote clients to access licenses from the server.
   b) Port 8080 is closed to ensure that the management interface is available only through a web browser running locally on the license server host.



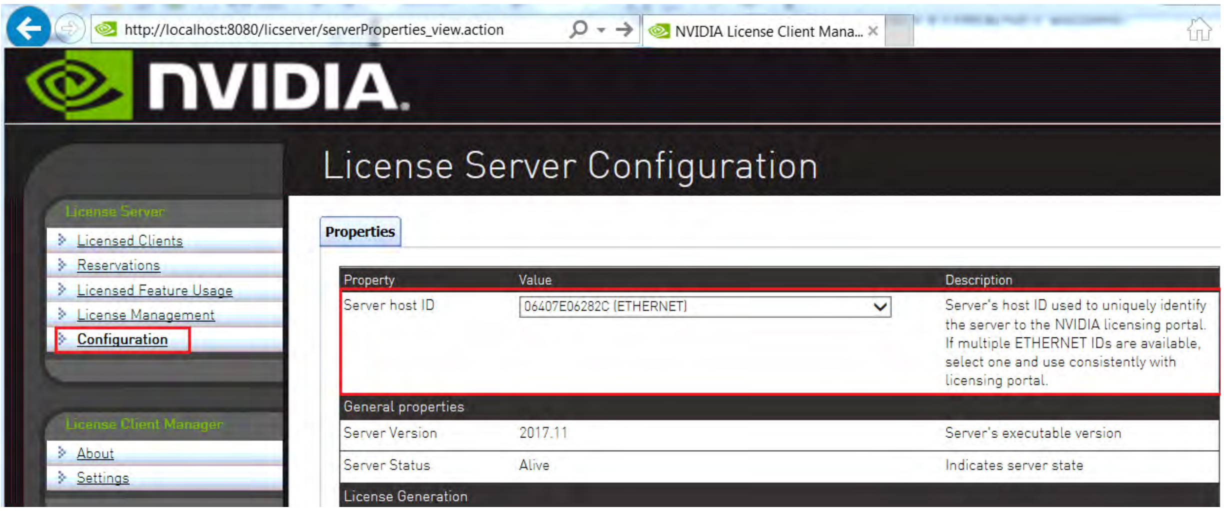
5. After installation has completed successfully, click Done to exit the installer.



4.2.3 Obtaining the License Server's MAC Address

The license server's Ethernet MAC address uniquely identifies your server to the NVIDIA Licensing Portal. You will need this address to register your license server with the NVIDIA Licensing Portal to generate license files.

1. Open a web browser on the license server host and connect to the URL http://localhost:8080/licserver.
2. In the license server management interface, select **Configuration**.
3. On the License Server Configuration page that opens, in the **Server host ID** drop-down list, select the platform's Ethernet address.



4.2.4 Managing your License Server and Getting your License Files

To be able to download NVIDIA vGPU software licenses, you must create at least one license server on the NVIDIA Licensing Portal and allocate licenses to the server. After creating a license server and allocating licenses to it, you can download your license file.

4.2.4.1 Creating a License Server on the NVIDIA Licensing Portal

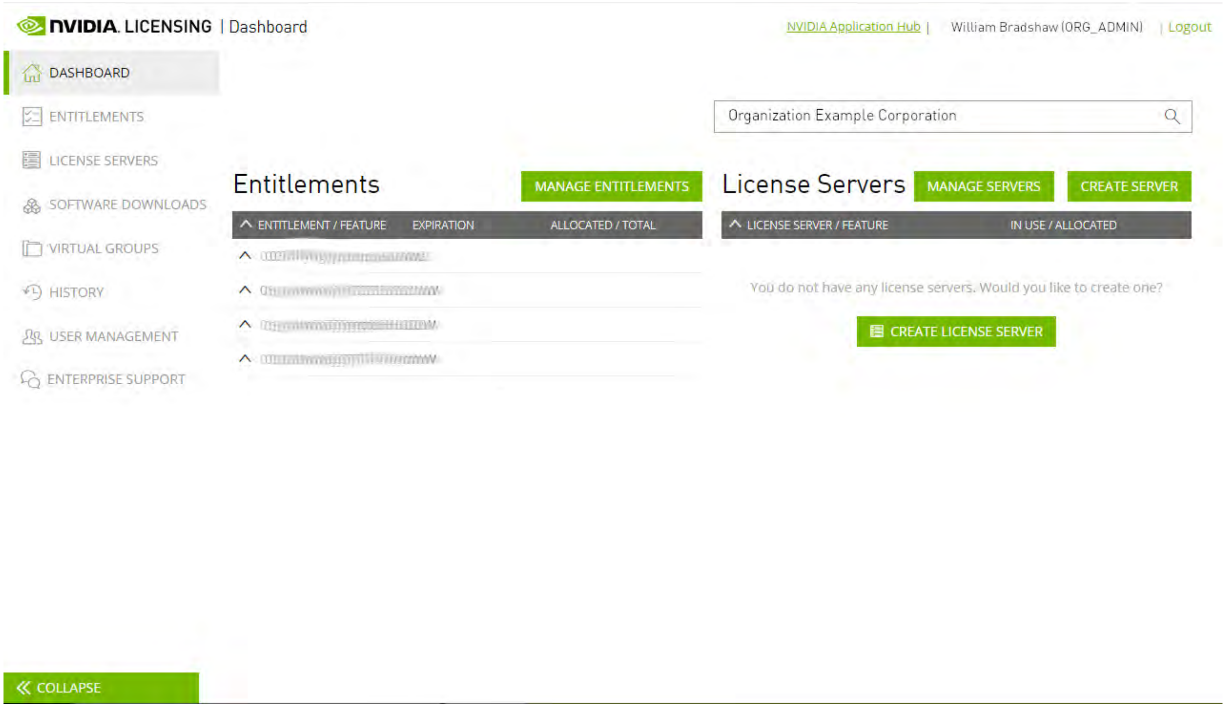
Creating a license server on the NVIDIA Licensing Portal registers your license server host with the NVIDIA Licensing Portal through the MAC address of the host.

1. In the NVIDIA Licensing Portal, navigate to the organization or virtual group for which you want to create the license server.

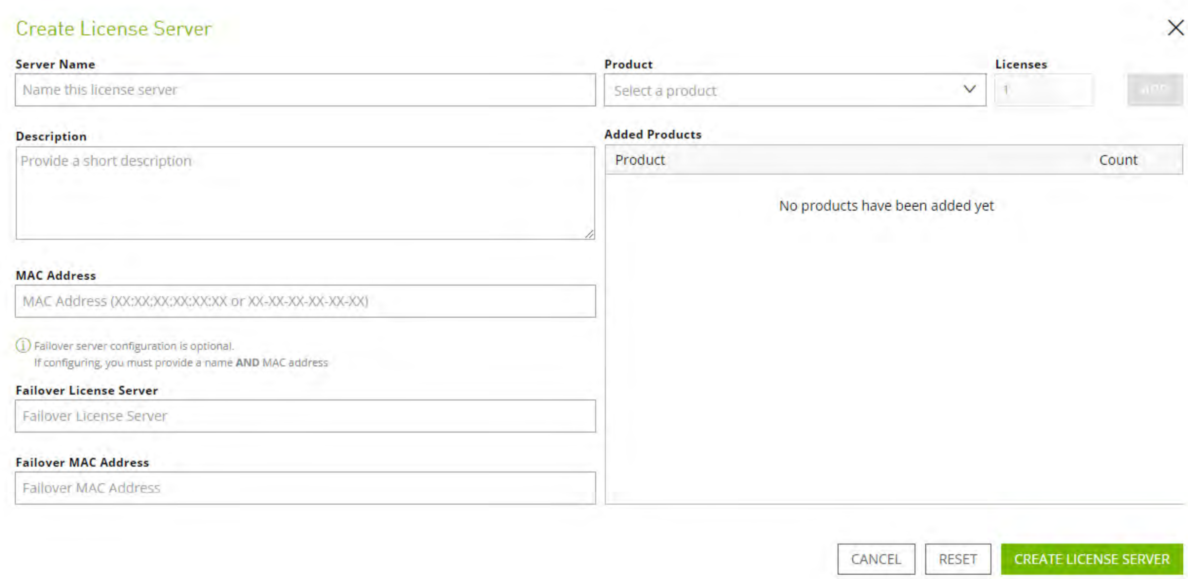
    a) If you have not already logged in, log in to the NVIDIA Enterprise Application Hub and click **NVIDIA LICENSING PORTAL** to go to the NVIDIA Licensing Portal.

    b) **Optional:** If your assigned roles give you access to multiple virtual groups, select the virtual group for which you are creating the license server from the list of virtual groups at the top right of the page.

    If no license servers have been created for your organization or virtual group, the NVIDIA Licensing Portal dashboard displays a message asking if you want to create a license server.



2. On the NVIDIA Licensing Portal dashboard, click **CREATE LICENSE SERVER**.
   The Create License Server pop-up window opens.



3. Provide the details of your license server.
   a) In the **Server Name** field, enter the host name of the license server.
   b) In the **Description** field, enter a text description of the license server. This description is required and will be displayed on the details page for the license server that you are creating.
   c) In the **MAC Address** field, enter the MAC address of the license server.
4. Add the licenses for the products that you want to allocate to this license server. For each product, add the licenses as follows:

   a) From the **Product** drop-down list, select the product for which you want to add licenses.
   b) In the **Licenses** field, enter the number of licenses for the product that you want to add.
   c) Click **ADD**.
5. Leave the **Failover License Server** and **Failover MAC Address** fields unset.
6. Click **CREATE LICENSE SERVER**.

4.2.4.2 Downloading a License File

Each license server that you create has a license file associated with it. The license file contains all of the licenses that you allocated to the license server. After you download the license file, you can install it on the license server host associated with the license server on the NVIDIA Licensing Portal.

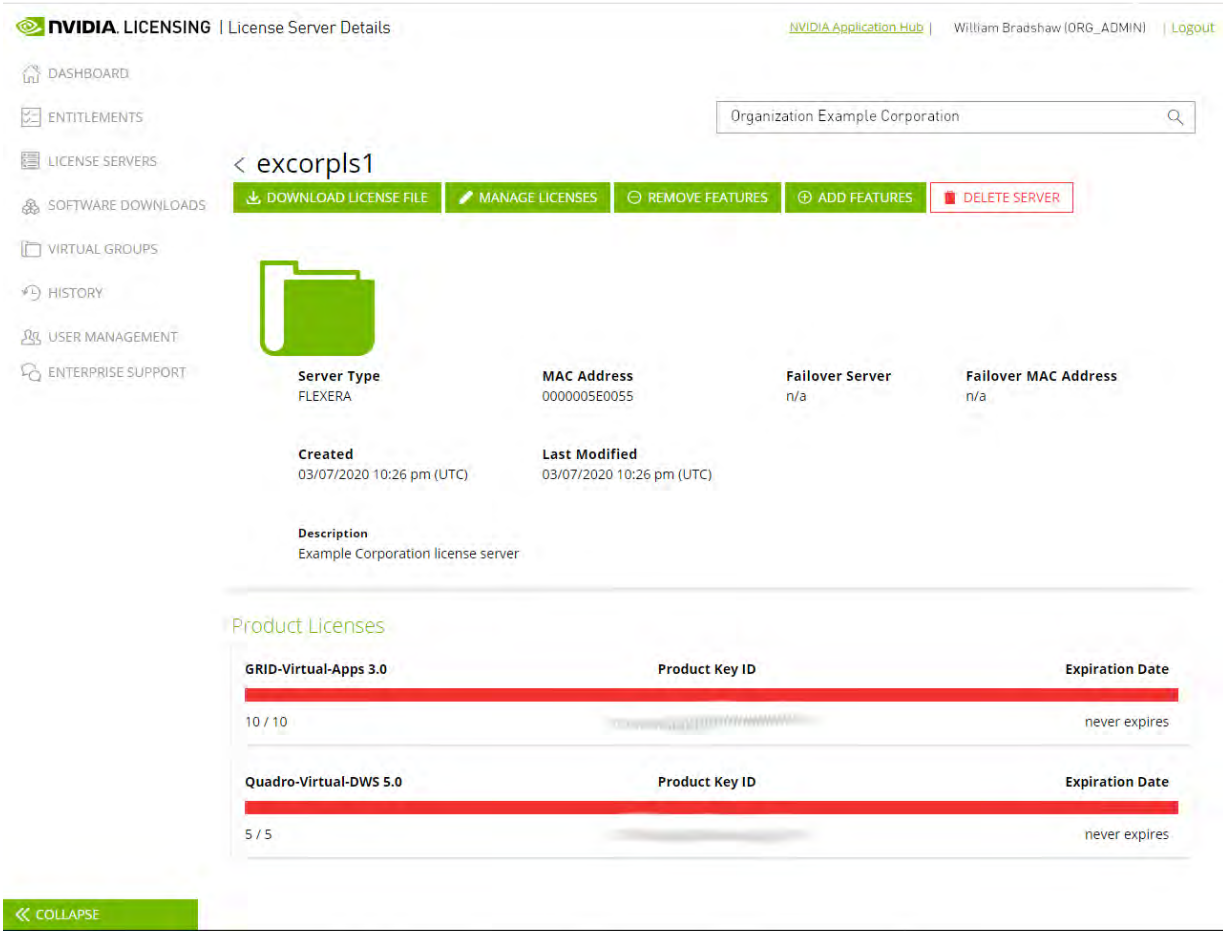
1. In the NVIDIA Licensing Portal, navigate to the organization or virtual group for which you want to download the license file.

   a) If you have not already logged in, log in to the NVIDIA Enterprise Application Hub and click **NVIDIA LICENSING PORTAL** to go to the NVIDIA Licensing Portal.

   b) **Optional:** If your assigned roles give you access to multiple virtual groups, select the virtual group for which you are downloading the license file from the list of virtual groups at the top right of the page.

2. In the list of license servers on the NVIDIA Licensing Portal dashboard, select the license server whose associated license file you want to download.

3. In the License Server Details page that opens, review the licenses allocated to the license server.



4. Click **DOWNLOAD LICENSE FILE** and save the `.bin` license file to your license server for installation.

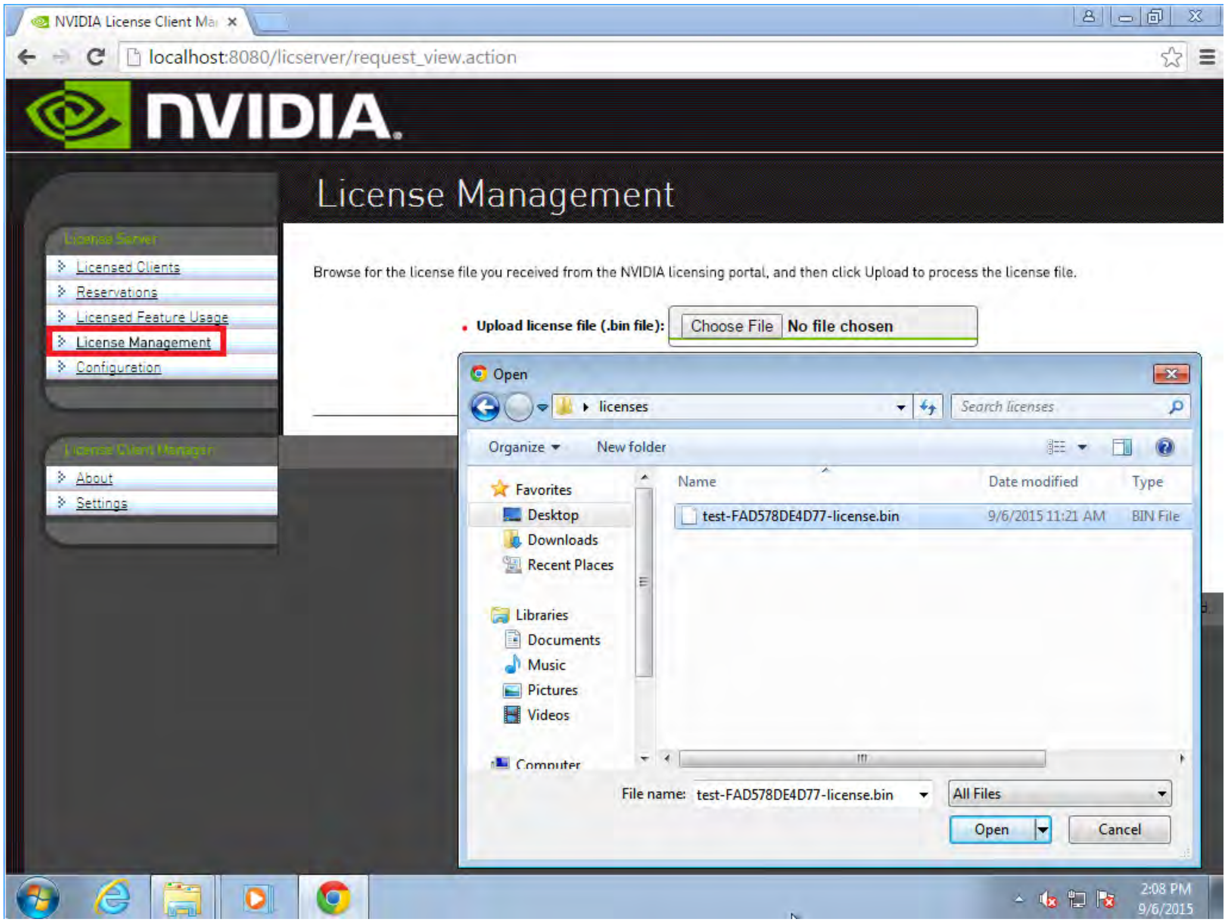
4.2.5 Installing a License

NVIDIA vGPU software licenses are distributed as `.bin` files for download from the NVIDIA Licensing Portal.

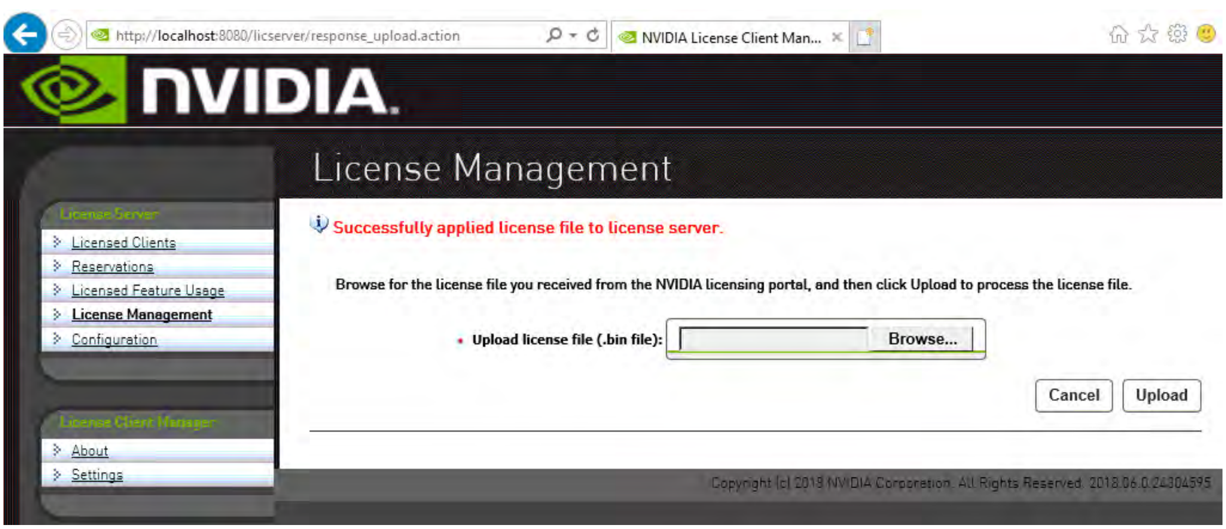
Before installing a license, ensure that you have downloaded the license file from the NVIDIA Licensing Portal.

1. In the license server management interface, select **License Management**.

2. On the License Management page that opens, click **Choose File**.



3. In the file browser that opens, select the `.bin` file and click **Open**.
4. Back on the License Management page, click **Upload** to install the license file on the license server. The license server displays a confirmation if the license file is installed successfully.



Note: For additional configuration options including Linux server deployment, securing your license server, and license provisioning, refer to the Virtual GPU Software License Server User Guide.

Chapter 5. Creating Your First NVIDIA Virtual Compute Server VM

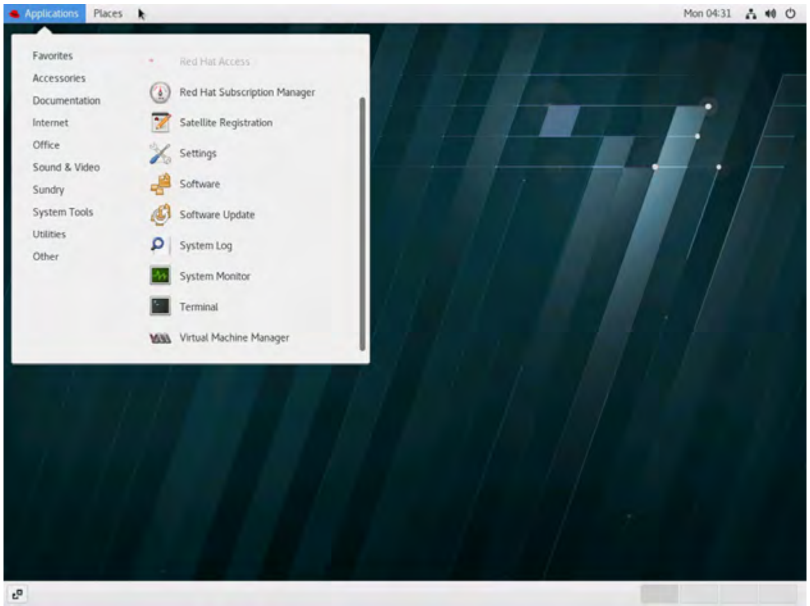
This chapter covers creating an NVIDIA Virtual Compute Server VM, including:

- Creating a Virtual Machine
- Creating the vGPU
- Attaching the vGPU to the VM
- Installing Ubuntu Server 18.04.5 LTS
- Enabling the NVIDIA vGPU
- Installing the NVIDIA Driver in the Ubuntu Virtual Machine
- Licensing an NVIDIA vGPU

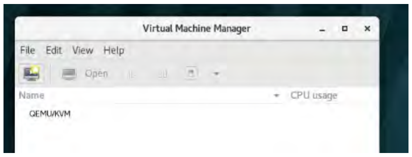
5.1 Creating the VM

Now that all configuration is complete, we can create our first VM.

1. Download Ubuntu Server OS.
2. Log in to the server and transfer the Ubuntu ISO to Virtual Machine Manager's (VMM's) location for images: `/var/lib/libvirt/images`.
3. Start VMM from the Applications menu. Enter your password when prompted.

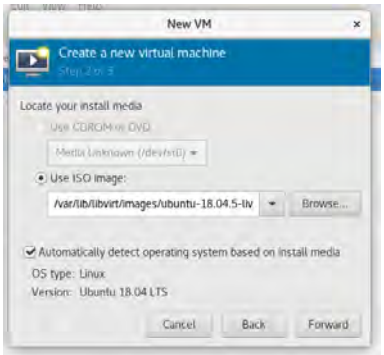


4. Click the Create a New Virtual Machine button.

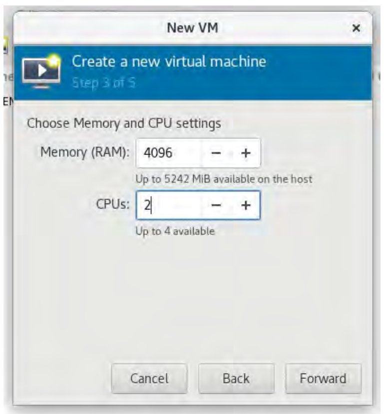


5. Leave the first option selected and click Forward.

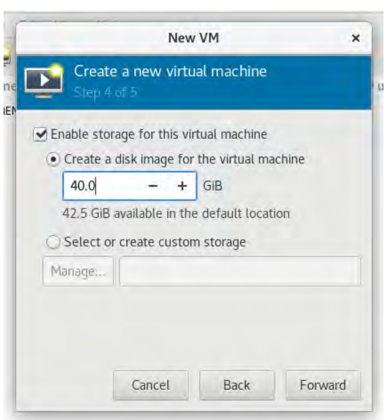
6. Select your Guest OS's ISO image and click Forward. The system normally auto-detects the correct OS, but if it does not, select the correct OS for the VM.



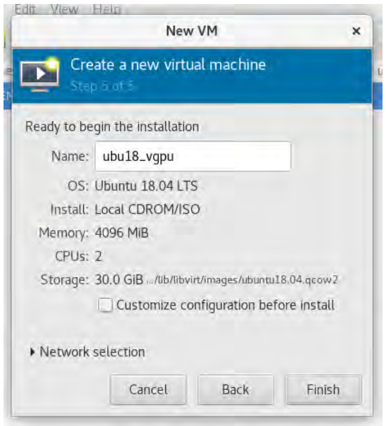
7. Enter the desired number of vCPUs and amount of RAM.



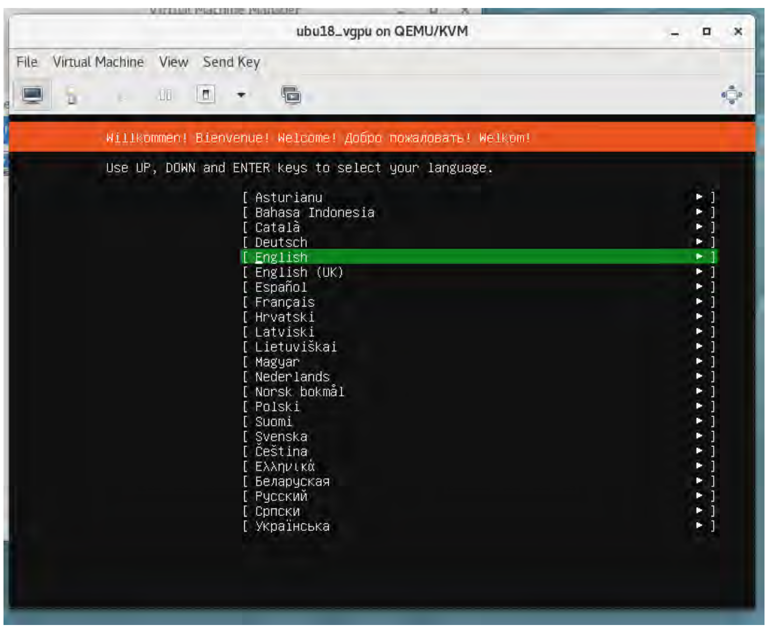
8. Enter the desired amount of storage.



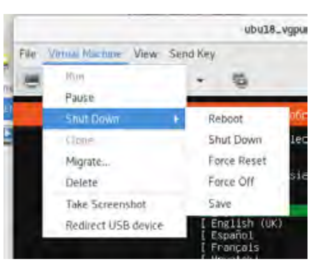
9. Enter an appropriate name for the VM, then click Finish.



10. VMM automatically turns on the VM and presents it in the View Manager. Verify that VM boots and can load the OS ISO.



11. Power off the VM for now, as some additional configuration is needed. Do not install the OS yet.



5.2 Attach the vGPU profile to the VM

Now that a VM has been created and has an available vGPU, it is time to combine the two. This is done by attaching the vGPU to the VM. Using virsh, a virtualization focused interactive terminal is the best way to accomplish this.

1. Use virsh to edit the VM.

```
virsh edit [vm name]
```

   [vm name] is the name of the VM created in step 5 of section 5.1.

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.

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

[uuid] is 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
- b356d38-854e-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>
```

3. Now that the vGPU has been added, exit virsh.
4. Power on the VM, either via the VMM GUI or via virsh start [vm name] where [vm name] is the name of the VM we created in step 5.1.9. In our example:

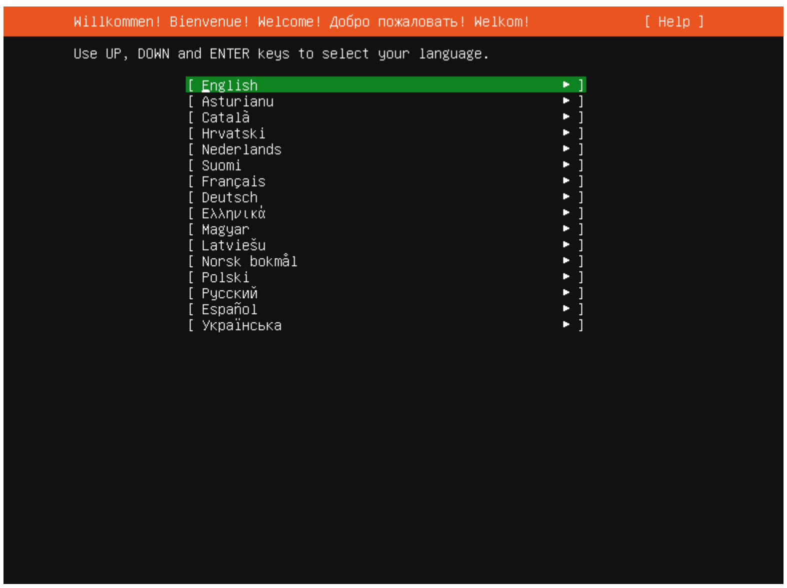
```
# Virsh start ubu18_vgpu
```

5.3 Installing Ubuntu Server 18.04.5 LTS

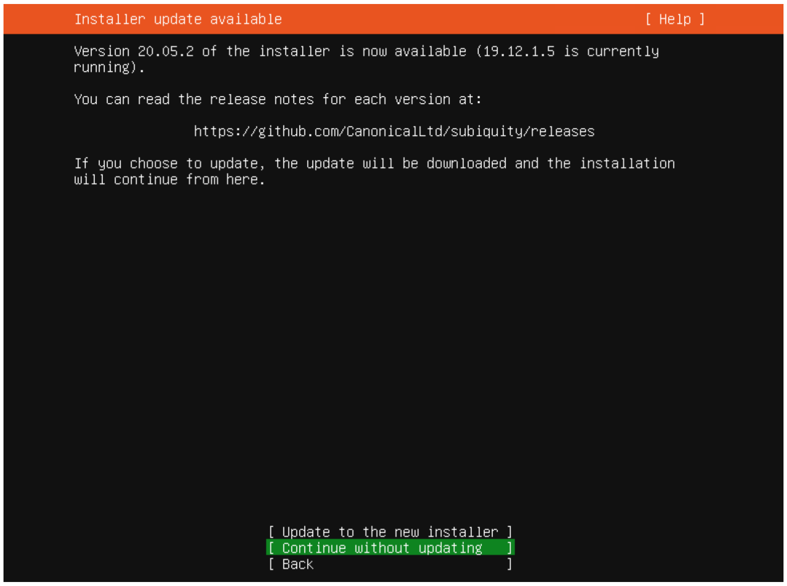
1. Power on the VM either with the VMM GUI or by entering:

   ```
   virsh start [vm name]
   ```

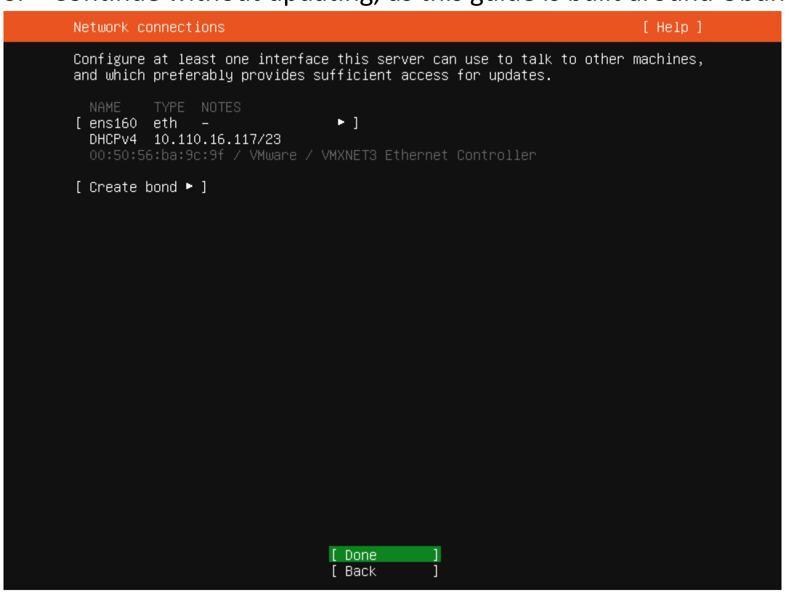
   Where [vm name] is the name of the VM we created in step 5.1.9. Wait for the installation screen to appear.



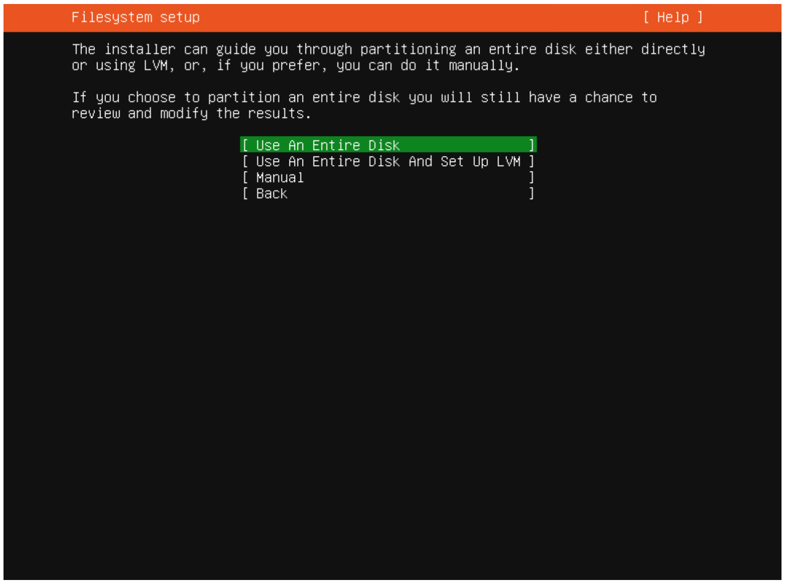
2. Select your preferred language and press ENTER.



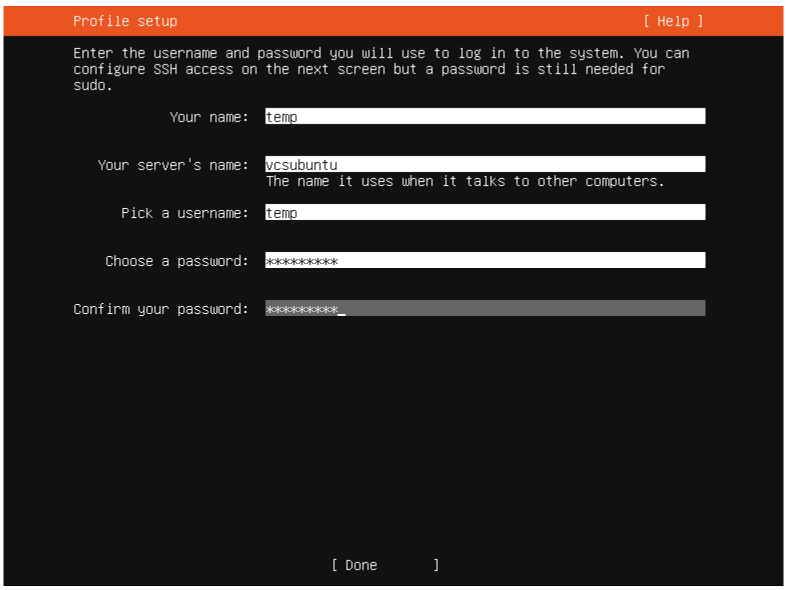
3. Continue without updating, as this guide is built around Ubuntu 18.04.



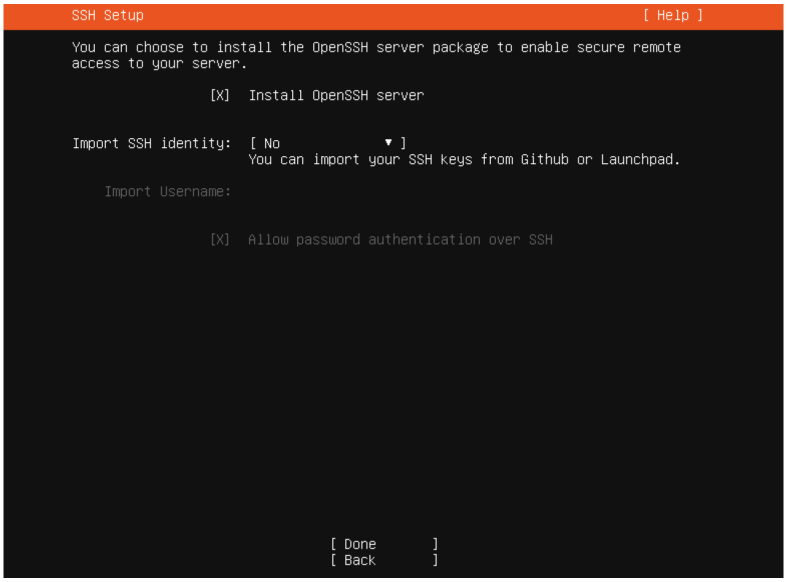
4. On this screen, select your network connection type and modify to fit your internal requirements. Select DHCP for our configuration.



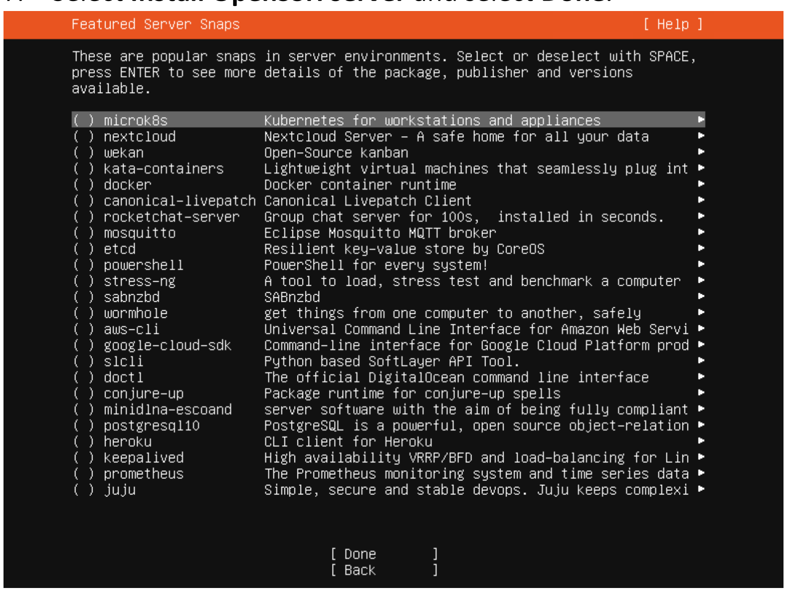
5. Format the entire disk.



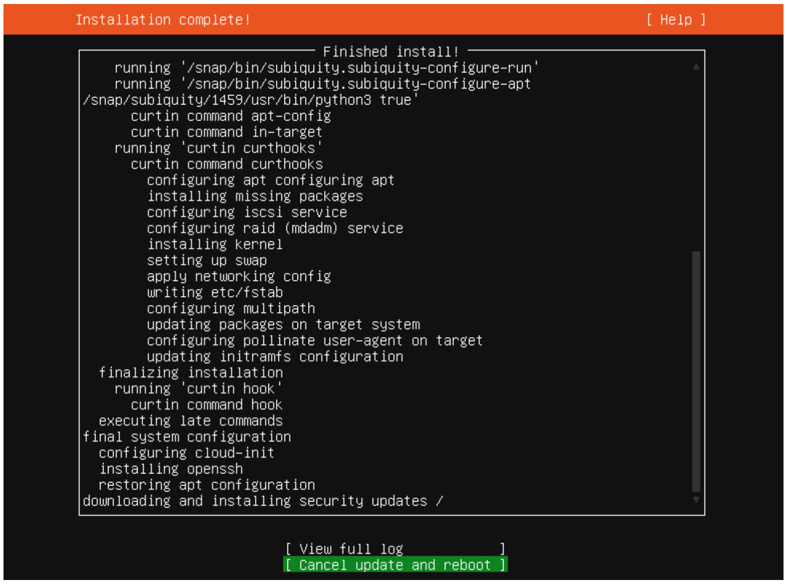
6. Configure the VM with a user account, name, and password.



7. Select **Install OpenSSH server** and select **Done**.



8. Select any server snaps that may be required for internal use in your environment and select **Done**.



9. Installation now runs to completion.

5.4 Installing the NVIDIA Driver on the Ubuntu Virtual Machine

After you create a Linux VM on the hypervisor and boot the VM, install the NVIDIA vGPU software display driver in the VM to enable GPU operation fully.
Installation of the NVIDIA vGPU software display driver for Linux requires:

- The compiler toolchain
- Kernel headers

Use the following procedure to install the NVIDIA driver on the Ubuntu VM:

1. Log in and shut down the display manager.
```
sudo service lightdm stop
```
2. From a console shell, run the driver installer as the root user.
```
sudo sh ./ NVIDIA-Linux_x86_64-440.87-grid.run
```
In some instances, the installer may fail to detect the installed kernel headers and sources. In this situation, re-run the installer, specifying the kernel source path with the --kernel-source-path option:
```
sudo sh ./ NVIDIA-Linux_x86_64-440.87-grid.run \
–kernel-source-path=/usr/src/kernels/3.10.0-229.11.1.el7.x86_64
```
3. When prompted, accept the option to update the X configuration file (`xorg.conf`).
4. Enable Persistence Mode.
```
sudo systemctl daemon-reload
sudo systemctl enable nvidia-persistenced.service
sudo systemctl start nvidia-persistenced.service
```
5. Reboot the system.
```
sudo reboot
```

6. After the system has rebooted, confirm that you can see your NVIDIA vGPU device in the output from `nvidia-smi`.
```
nvidia-smi
```

After you install the NVIDIA vGPU software graphics driver, you can license any NVIDIA vGPU software licensed products that you are using. For instructions, see Licensing an NVIDIA vGPU (update 11.0).

5.5 Licensing an NVIDIA vGPU

NVIDIA vGPU is a licensed product. When booted on a supported GPU, a vGPU initially operates at full capability but its performance is degraded over time if the VM fails to obtain a license. If the performance of a vGPU has been degraded, the full capability of the vGPU is restored when a license is acquired.

For complete information about configuring and using NVIDIA vGPU software licensed features, including vGPU, refer to Virtual GPU Client Licensing User Guide. Perform this task from the guest VM to which the vGPU is assigned.

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

1. Start NVIDIA X Server Settings by using the method for launching applications provided by your Linux distribution.
   For example, on Ubuntu Desktop, open the Dash, search for NVIDIA X Server Settings, and click the **NVIDIA X Server Settings** icon.
2. In the NVIDIA X Server Settings window that opens, click **Manage License**.
   The License Edition section of the NVIDIA X Server Settings window shows that NVIDIA vGPU is currently unlicensed.
3. In the **Primary Server** field, enter the address of your primary NVIDIA vGPU software License Server.
   The address can be a fully qualified domain name such as `nvidialicense1.example.com`, or an IP address. If you have only one license server configured, enter its address in this field.
4. Leave the **Port Number** field under the **Primary Server** field unset.
   The port defaults to 7070, which is the default port number used by NVIDIA vGPU software License Server.
5. In the **Secondary Server** field, enter the address of your secondary NVIDIA vGPU software License Server.
   The address can be a fully qualified domain name such as `nvidialicense2.example.com`, or an IP address. If you have only one license server configured, leave this field unset.
6. Leave the **Port Number** field under the **Secondary Server** field unset.
   The port defaults to 7070, which is the default port number used by NVIDIA vGPU software License Server.
7. Click **Apply** to assign the settings.

The system requests the appropriate license for the current vGPU from the configured license server.

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

Chapter 6. Selecting the Correct vGPU Profiles

Choosing the right vGPU profile to maximize your stakeholders' experience in the virtual instance is critical for ensuring expected performance and quality of service. Below, you will find guidance through the vGPU Manager and beyond to ensure your deployment is successful.

6.1 The Role of the vGPU Manager

NVIDIA vGPU profiles assign custom amounts of dedicated GPU memory for each user. NVIDIA vGPU Manager assigns the correct amount of memory to meet the specific needs within the workflow for said user. Every virtual machine has dedicated GPU memory and must be assigned accordingly, ensuring that it has the resources needed to handle the expected compute load.
NVIDIA vGPU Manager allows up to eight users to share each physical GPU by assigning the graphics resources of the available GPUs to virtual machines using a balanced approach. Depending on the number of GPUs within each line card, there can be multiple user types assigned.

6.2 vGPU Profiles for NVIDIA Virtual Compute Server

The profiles represent a very flexible deployment option of virtual GPUs, varying in size of GPU memory. The division of GPU memory defines the number of vGPUs that are possible per GPU. Please refer to the NVIDIA Virtual Compute Server Solution Brief for a full list of supported and recommended NVIDIA GPU's.

C-series vGPU types are NVIDIA vCS vGPU types, which are optimized for compute-intensive workloads. As a result, they support only a single display head at a maximum resolution of 4096×2160 and do not provide RTX graphics acceleration.

The following table illustrates examples of the NVIDIA vCS profiles and how they fractionalize.

Virtual GPU Type	Intended Use Case	Frame Buffer (MB)
48C	Training Workloads	49152
40C	Training Workloads	40960
32C	Training Workloads	32768
24C	Training Workloads	24576
20C	Training Workloads	20480
16C	Training Workloads	16384
12C	Training Workloads	12288
10C	Training Workloads	10240
8C	Training Workloads	8192
6C	Training Workloads	6144
5C	Training Workloads	5120
4C	Inference Workloads	4096

Chapter 7. GPU Aggregation for NVIDIA Virtual Compute Server

NVIDIA vCS supports GPU aggregation, a feature which allows one VM to access more than one GPU. GPU aggregations is often required for compute-intensive workloads.

NVIDIA vCS also supports both multi-vGPU and peer-to-peer computing.

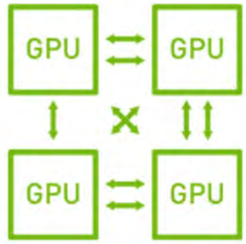
The following sections describe both technologies and explain how to deploy GPU aggregation within Red Hat Enterprise Linux with KVM.

7.1 Multi-vGPU

NVIDIA vCS supports multi-vGPU workloads which can offer a monumental improvement in virtual GPU performance by aggregating the power of up to 16 NVIDIA GPUs in a single virtual machine. With multi-vGPU, the GPUs are not directly connected to one another. The following graphic illustrates multi-vGPU and how a single VM can be assigned 4 vGPUs:

7.2 Peer-to-Peer NVIDIA NVLINK

NVIDIA vCS supports peer to peer computing where multiple GPU's are connected through NVIDIA NVLink. This enables a high speed, direct GPU-to-GPU interconnect that provides higher bandwidth for multi-vGPU system configurations than traditional PCIe-based solutions. The following graphic illustrates peer-to-peer NVLink:



This peer-to-peer communication allows access to device memory between GPU's from within the CUDA kernels and eliminates system memory allocation and copy overhead. It provides a more convenient means of multi-vGPU programming.

Peer-to-peer CUDA transfers over NVLink are supported for Linux only, not for Microsoft Windows. Currently vGPU does not support NVSwitch; therefore, only direct connections are supported. Peer-to-peer communication is supported only within a single VM. There is no SLI support; therefore, graphics is not included in this support, only CUDA. Peer-to-Peer CUDA Transfers over NVLink are supported only on a subset of vGPUs, Red Hat Enterprise Linux with KVM releases, and guest OS releases. Only C-series full frame buffer (1:1) vGPU profiles are supported with NVLink. Refer to the [vGPU latest release notes] for a listed of GPU's which are supported.

1. Connect to the RHEL host over SSH, using Putty, for example.
2. Type `nvidia-smi` in the command window.

```
[root@localhost:~] nvidia-smi
Thu Apr  9 16:21:46 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.53       Driver Version: 440.53       CUDA Version: N/A      |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM2...  On   | 00000000:15:00.0 Off |                    0 |
| N/A   34C    P0    44W / 300W |     61MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:16:00.0 Off |                    0 |
| N/A   34C    P0    42W / 300W |     61MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:3A:00.0 Off |                    0 |
| N/A   32C    P0    43W / 300W |     61MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:3B:00.0 Off |                    0 |
| N/A   33C    P0    43W / 300W |     61MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   4  Tesla V100-SXM2...  Off  | 00000000:89:00.0 Off |                    0 |
| N/A   33C    P0    44W / 300W |     59MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   5  Tesla V100-SXM2...  Off  | 00000000:8A:00.0 Off |                    0 |
| N/A   35C    P0    45W / 300W |     59MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   6  Tesla V100-SXM2...  Off  | 00000000:B2:00.0 Off |                    0 |
| N/A   35C    P0    44W / 300W |     59MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   7  Tesla V100-SXM2...  Off  | 00000000:B3:00.0 Off |                    0 |
| N/A   36C    P0    43W / 300W |     59MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Note: The form factor of the V100 graphics card in this example is SXM2.

3. Detect the topology between the GPUs by typing the following command:

```
$ nvidia-smi topo -m
```

```
[root@localhost:~] nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    CPU Affi
nity
GPU0     X      NV1     NV1     NV2     NV2     SYS     SYS     SYS
GPU1    NV1      X      NV2     NV1     SYS     NV2     SYS     SYS
GPU2    NV1     NV2      X      NV2     SYS     SYS     NV1     SYS
GPU3    NV2     NV1     NV2      X      SYS     SYS     SYS     NV1
GPU4    NV2     SYS     SYS     SYS      X      NV1     NV1     NV2
GPU5    SYS     NV2     SYS     SYS     NV1      X      NV2     NV1
GPU6    SYS     SYS     NV1     SYS     NV1     NV2      X      NV2
GPU7    SYS     SYS     SYS     NV1     NV2     NV1     NV2      X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA
 nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Hos
t Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
 CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCI
e Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```

4. Assign suitable 1:1 vGPU(s) to the VM.

The CUDA driver in the VM detects the peer-to-peer capability between the vGPUs and allows the CUDA application to use it.

Note: NVLink is supported in non-MIG mode, please refer to Chapter 9 for more information regarding NVIDIA Multi-Instance GPU's (MIG).

7.3 GPUDirect Technology Support

NVIDIA® GPUDirect® technology remote direct memory access (RDMA) enables network devices to access the vGPU frame buffer directly, bypassing CPU host memory altogether. GPUDirect technology is supported only on a subset of vGPUs and guest OS releases since vGPU release 11.1.

Only C-series vGPUs that are allocated all of the physical GPU's frame buffer on physical GPUs based on the NVIDIA Ampere architecture are supported. Both time-sliced and MIG-backed vGPUs that meet these requirements are supported. Please refer to the vGPU user guide for more information regarding supported OS and NVIDIA GPUs.

Chapter 8. Page Retirement and ECC

NVIDIA vCS supports ECC and dynamic page retirement on all supported GPUs. This feature "retires" bad frame buffer memory cells by retiring the page a cell belongs to. Dynamic page retirement is done automatically for cells that are degrading in quality. This feature can improve the longevity of an otherwise good board and is thus an important resiliency feature on supported products, especially in HPC and enterprise environments.

Page retirement may only occur when ECC is enabled. However, once a page has been retired it is permanently blacklisted, even if ECC is later disabled. Refer to the NVIDIA Developer Zone page retirement documentation for more information.

Chapter 9. NVIDIA Multi-Instance GPU Configuration for KVM

NVIDIA A100 Tensor Core GPU is based upon the NVIDIA Ampere architecture and accelerates compute workloads such as AI, data analytics, and HPC in the data center. MIG support on vGPUs began at the NVIDIA vGPU 11.1 software release and gives users the flexibility to use the NVIDIA A100 in MIG mode or non-MIG mode. When the NVIDIA A100 is in non-MIG mode, NVIDIA vCS software uses vGPU temporal partitioning and GPU time slice scheduling. MIG mode spatially partitions the hardware of GPU so that each MIG can be fully isolated with its own streaming multiprocessors (SM’s), high bandwidth, and memory. MIG can partition available GPU compute resources as well.

Each instance's processors have separate and isolated paths through the entire memory system. The on-chip crossbar ports, L2 cache banks, memory controllers, and DRAM address busses are all assigned uniquely to an individual instance. This ensures that an individual user's workload can run with predictable throughput and latency, using the same L2 cache allocation and DRAM bandwidth, even if other tasks are thrashing their own caches or saturating their DRAM interfaces.

A single NVIDIA A100 has eight usable GPU memory slices, each with 5 GB of memory, but there are only seven usable SM slices. There are seven SM slices, not eight, because some of the SMs are used to cover operational overhead when MIG mode is enabled. MIG mode is configured (or reconfigured) using nvidia-smi and has profiles that you can choose to meet the needs of HPC, deep learning, or accelerated computing workloads.

In summary, MIG spatially partitions the NVIDIA GPU into separate GPU instances but provides benefits of reduced latency over vGPU temporal partitioning for compute workloads. The following tables summarizes similarities as well as differences between A100 MIG capabilities and NVIDIA vGPU software, while also highlighting the additional flexibility when they are combined.

	NVIDIA A100 MIG-Backed Virtual GPU Types	NVIDIA A100 with NVIDIA vCS Virtual GPU Types
GPU Partitioning	Spatial (hardware)	Temporal (software)
Number of Partitions	7	10
Compute Resources	Dedicated	Shared
Compute Instance Partitioning	Yes	No
Address Space Isolation	Yes	Yes
Fault Tolerance	Yes (highest quality)	Yes
Low Latency Response	Yes (highest quality)	Yes
NVLink Support	No	Yes
Multi-Tenant	Yes	Yes
GPUDirect RDMA	Yes (GPU instances)	Yes
Heterogenous Profiles	Yes	No
Management - requires Super User	Yes	No

One of the new features introduced to vGPUs when VM's are using MIG0backed virtual GPUs is the ability to have different sized (heterogenous) partitioned GPU instances. The following table illustrates the 18 possible size combinations when NVIDIA A100 has MIG mode enabled.

<table>
<tr><th>Slice #1</th><th>Slice #2</th><th>Slice #3</th><th>Slice #4</th><th>Slice #5</th><th>Slice #6</th><th>Slice #7</th></tr>
<tr><td colspan="7">7</td></tr>
<tr><td colspan="4">4</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="4">4</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">2</td><td colspan="2">2</td><td colspan="3">3</td></tr>
<tr><td colspan="2">2</td><td>1</td><td>1</td><td colspan="3">3</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">2</td><td colspan="3">3</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="3">3</td></tr>
<tr><td colspan="3">3</td><td rowspan="3"></td><td colspan="3">3</td></tr>
<tr><td colspan="3">3</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="3">3</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">2</td><td colspan="2">2</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="2">2</td><td colspan="2">2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">2</td><td colspan="2">2</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">2</td><td>1</td><td>1</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="2">2</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="2">2</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Note: When using vCS and MIG mode is enabled, the vGPU software recognizes the MIG backed vGPU resource as if it were 1:1 or full GPU profile.

NVIDIA vGPU software supports MIG only with NVIDIA Virtual Compute Server and Linux guest operating systems. 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 prior to assigning the resource to a VM. For more information, see Configuring a GPU for MIG-Backed vGPUs in the Virtual GPU Software Documentation. For general information about the MIG feature, see NVIDIA Multi-Instance GPU User Guide.

9.1 Terminology

9.1.1 GPU Context

A GPU context is analogous to a CPU process. It encapsulates all of the resources necessary to execute operations on the GPU, including a distinct address space, memory allocations, etc. A GPU context has the following properties:

- Fault isolation
- Individual scheduling
- Distinct address space

9.1.2 GPU Engine

A GPU engine executes work on the GPU. The most commonly used engine is the Compute/Graphics engine, which executes the compute instructions. Other engines include the copy engine (CE), which is responsible for performing DMAs, NVDEC for video decoding, etc. Each engine can be scheduled independently and can execute work for different GPU contexts.

9.1.3 GPU Memory Slice

A GPU memory slice is the smallest fraction of the A100 GPU's memory, including the corresponding memory controllers and cache. A GPU memory slice is roughly one eighth of the total GPU memory resources, including both capacity and bandwidth.

9.1.4 GPU SM Slice

A GPU SM slice is the smallest fraction of the SMs on the A100 GPU. A GPU SM slice is roughly one seventh of the total number of SMs available in the GPU when configured in MIG mode.

9.1.5 GPU Slice

A GPU slice is the smallest fraction of the A100 GPU that combines a single GPU memory slice and a single GPU SM slice.

9.1.6 GPU Instance

A GPU instance (GI) is a combination of GPU slices and GPU engines (DMAs, NVDECs, etc.). Anything within a GPU instance always shares all the GPU memory slices and other GPU engines, but its SM slices can be further subdivided into compute instances (CIs). A GPU instance provides memory QoS. Each GPU slice includes dedicated GPU memory resources which limit both the available capacity and bandwidth, as well as provide memory QoS. Each GPU memory slice gets one eighth of the total GPU memory resources, and each GPU SM slice gets one seventh of the total number of SMs.

9.1.7 Compute Instance

A GPU instance can be subdivided into multiple compute instances. A compute instance (CI) contains a subset of the parent GPU instance's SM slices and other GPU engines (DMAs, NVDECs, etc.). The CIs share memory and engines.
The number of slices that a GI (GPU Instance) can be created with is not arbitrary. The NVIDIA driver APIs provide a number of "GPU Instance Profiles," and users can create GIs by specifying one of these profiles.
On a given GPU, multiple GIs can be created from a mix and match of these profiles, so long as enough slices are available to satisfy the request.

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	Number of Instances Available
MIG 1g.5gb	1/8	1/7	0 NVDECs	7
MIG 2g.10gb	2/8	2/7	1 NVDECs	3
MIG 3g.20gb	4/8	3/7	2 NVDECs	2
MIG 4g.20gb	4/8	4/7	2 NVDECs	1
MIG 7g.40gb	Full	7/7	5 NVDECs	1

9.2 MIG Prerequisites

The following prerequisites apply when using A100 in MIG mode.

- Supported only on NVIDIA A100 products and associated systems using A100 (see the vGPU certified servers page)
- Requires CUDA 11 and NVIDIA vGPU driver 450.51.05 or greater
- Requires CUDA 11 supported Linux operating system distributions

MIG can be managed programmatically using NVIDIA Management Library (NVML) APIs or its command-line-interface, `nvidia-smi`. Note that for brevity, some of the `nvidia-smi` output in the following examples may be cropped to showcase the relevant sections of interest.
For more information on the MIG commands, see the `nvidia-smi` man page or enter the command

```
nvidia-smi mig --help
```

For information on the MIG management APIs, see the NVML header (nvml.11.0.h) included in CUDA 11.

9.2.1 Enable MIG Mode

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 do not create compute instances within the GPU instances, they can be added later for individual vGPUs from 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.

1. Open a command shell as the root user on your hypervisor host machine. You can use secure shell (SSH) for this purpose.

2. Use the `nvidia-smi` command to determine whether MIG mode is enabled. By default, MIG mode is disabled. This example shows that MIG mode is disabled on GPU 0.

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.36.04    Driver Version: 450.36.04    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
```

```
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:36:00.0 Off |                    0 |
| N/A   29C    P0    62W / 400W |      0MiB / 40537MiB |      6%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

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 in the system.

This example enables MIG mode on GPU 0.

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

Note: If the GPU is being used by another process, nvidia-smi 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. Query the GPUs on which you enabled MIG mode to confirm that MIG mode is enabled.

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

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

9.2.2 List GPU Instance Profiles

1. List the GPU instance profiles that are available on your GPU. You must specify the profiles by their IDs, not their names, when you create them.

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

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

9.2.3 Creating GPU Instances

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

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

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

Supported profiles are 19, 14, 9, 5, and 0. These were listed in the previous step within the ID column.

For example, this command creates two 2g.10.gb GPU instances:

```
$ nvidia-smi mig -cgi 14,14 -gi 0
```

This command creates three 3g.20.gb GPU instances:

```
$ nvidia-smi mig -cgi 9,9,9 -gi 0
```

9.2.4 VM Configuration

Now that the MIG GPU instance has been created, next we will create appropriate mdev device for the MIG GPU instance.

For example, 2g.10gb, which has profile ID 14, was used to create two GPU instances in the previous step. Now we will create an mdev device for this profile (GRID A100-2-10C). Also, you must use different VFs for the two GPU instances. You can only assign one mdev device to a given VF.

> Note: An mdev is identified by its UUID. The /sys/bus/mdev/devices/ directory contains a symbolic link to the mdev device file.

```
echo "aa618089-8b16-4d01-a136-25a0f3c73123" >
/sys/bus/pci/devices/0000\:c1\:00.4/mdev_supported_types/nvidia-476/create
```

```
echo "1a6b0c78-0297-454f-b49a-65598e5e2e09" >
/sys/bus/pci/devices/0000\:c1\:00.5/mdev_supported_types/nvidia-476/create
```

Attach the mdev device to the VM.

```
virsh edit <VM Name>
<hostdev mode='subsystem' type='mdev' model='vfio-pci'>
<source>
<address uuid='<uuid>'/>
</source>
</hostdev>
```

Boot the VM with the MIG GPU Instance. In the next step we will create the Compute instance (optional).

9.2.5 Optional: Creating Compute Instances

You can add the compute instances for an individual vGPU from within the guest VM. If you want to replace the compute instances that were created when the GPU was configured for MIG-backed vGPUs, you can delete them before adding the compute instances from within the guest VM. Ensure that the following prerequisites are met:

- You have root user privileges on 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.

List the compute instances that can be created in a guest VM command shell:

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

```
+--------------------------------------------------------------------------------+
| Compute instance profiles:                                                     |
| GPU     GPU       Name             Profile  Instances   Exclusive       Shared   |
|       Instance                       ID     Free/Total     SM       DEC   ENC   OFA |
|         ID                                                          CE    JPEG      |
|================================================================================|
|   0      0       MIG 1c.3g.20gb      0        3/3         14         2     0     0  |
|                                                                     3     0        |
+--------------------------------------------------------------------------------+
|   0      0       MIG 2c.3g.20gb      1        1/1         28         2     0     0  |
|                                                                     3     0        |
+--------------------------------------------------------------------------------+
|   0      0       MIG 3g.20gb         2*       1/1         42         2     0     0  |
|                                                                     3     0        |
+--------------------------------------------------------------------------------+
```

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

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

Where <gpu-instance-ids> is a comma-separated list of GPU instance IDs that specifies the GPU instances within which you want to create the compute instances.
For example, to create compute instance with profile #2 (3g.20gb)

```
$ nvidia-smi mig -cci 2 -gi 0
```

If you want to create multiple compute instances and run apps in parallel, see the NVIDIA Multi-Instance GPU User Guide for more complex scenarios.

> 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 Since 11.1: Modifying a MIG-Backed vGPU's Configuration.

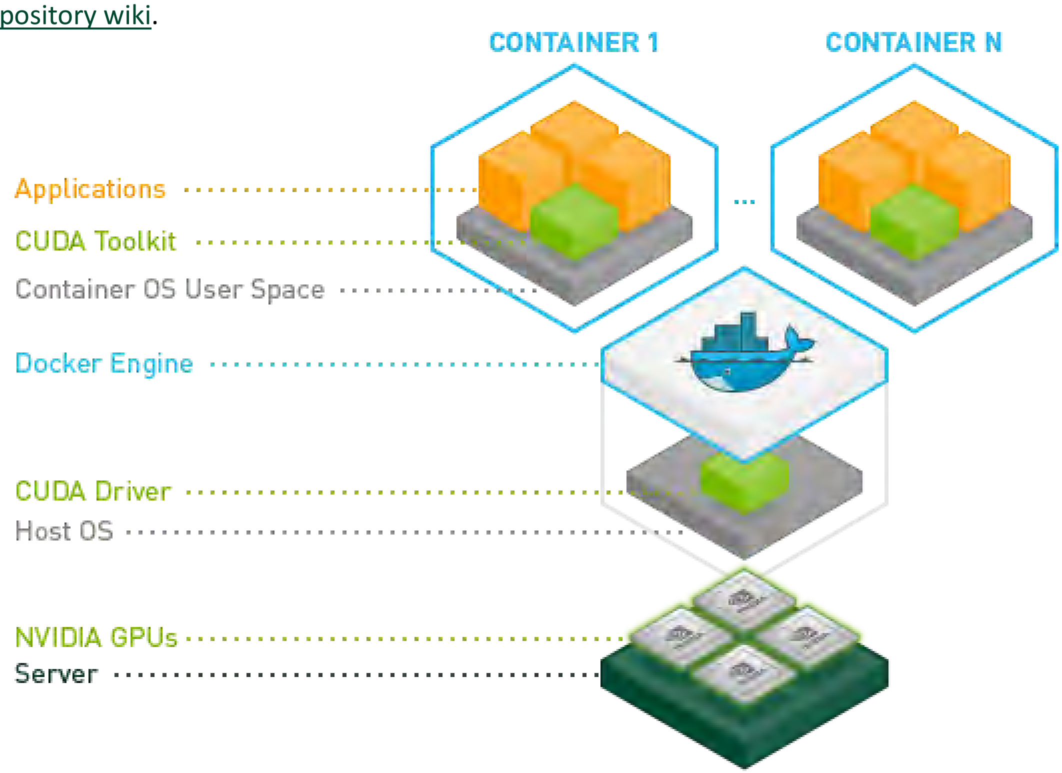
9.2.6 Optional: Update Containers for a MIG-Enabled vGPU

To run containers on a MIG-enabled vGPU you need to update the `nvidia-docker2` package. Follow the instructions NVIDIA Multi-Instance GPU User Guide.

```
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | apt-key add - |
curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu18.04/nvidia-
docker.list | tee /etc/apt/sources.list.d/nvidia-docker.list
apt-get update
apt-get install nvidia-docker2
```

Chapter 10. Installing Docker and the Docker Utility Engine for NVIDIA GPUs

The NVIDIA Container Toolkit allows users to build and run GPU accelerated Docker containers. The toolkit includes a container runtime library and utilities to configure containers automatically to leverage NVIDIA GPUs. Full documentation and frequently asked questions are available on the repository wiki.



10.1 Enabling the Docker Repository and Installing the NVIDIA Container Toolkit

Make sure you have installed the NVIDIA driver and Docker 19.03 for your Linux distribution note that you do not need to install the CUDA toolkit on the host, but the driver needs to be installed.

Note: With the release of Docker 19.03, nvidia-docker2 packages are deprecated since NVIDIA GPUs are now natively supported as devices in the Docker runtime.

For first-time users of Docker 19.03 and GPUs, continue with the instructions for getting started below.

1. Add the package repositories.
```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key
add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```
2. Download information from all configured sources about the latest versions of the packages. Install the `nvidia-container-toolkit` package.
```
sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit
```
3. Restart the docker service.
```
sudo systemctl restart docker
```

10.2 Testing Docker and NVIDIA Container Run Time

Use the commands below to test Docker and NVIDIA container run time.

```
#### Test nvidia-smi with the latest official CUDA image
docker run --gpus all nvidia/cuda:10.0-base nvidia-smi

# Start a GPU enabled container on two GPUs
docker run --gpus 2 nvidia/cuda:10.0-base nvidia-smi

# Starting a GPU enabled container on specific GPUs
docker run --gpus '"device=1,2"' nvidia/cuda:10.0-base nvidia-smi
docker run --gpus '"device=UUID-ABCDEF,1"' nvidia/cuda:10.0-base nvidia-smi

# Specifying a capability (graphics, compute, ...) for my container
# Note this is rarely if ever used this way
docker run --gpus all,capabilities=utility nvidia/cuda:10.0-base nvidia-smi
```

Chapter 11. Testing and Benchmarking

All deep learning frameworks are found on the NGC container registry, https://ngc.nvidia.com/container. NVIDIA is using the `19.04-py3` containers for each DL framework. Instructions for installing NVIDIA Docker can be found on the GitHub page at https://github.com/NVIDIA/nvidia-docker.

Note that most of the frameworks assume you have the dataset available on your system. NVIDIA is not allowed to distribute ImageNet (http://image-net.org/download) so customers must acquire it themselves. It is needed for all the RN50 training benchmarks.

Following are several examples with GNMT. While the dataset is the same, the preprocessing on the dataset is different for each case. Therefore, you cannot use the same dataset for each run. You must run the specific command to download and process the data to the benchmark example.

The following instructions are intended to be a shortcut to getting started with benchmarking. In the working directory of each benchmark. For each benchmark, a README file (`README.md` or `README.txt`) provides more details of data download, preprocessing, and running the code.

11.1 TensorRT RN50 Inference

- The container used in this example is `nvcr.io/nvidia/tensorrt:19.04-py3`.
- Required binary is included with the container at `/workspace/tensorrt/bin`.
- The Resnet50 model `prototxt` and `caffemodel` files are in the container at `/workspace/tensorrt/data/resnet50`.
- The command may take several minutes to run because NVIDIA® TensorRT™ is building the optimized plan prior to running. If you want to see what it is doing, add `--verbose` to the command.

11.1.1 Commands to Run the Test

Use the commands below to run the TensorRT RN50 Inference test.

```
$ docker pull nvcr.io/nvidia/tensorrt:19.04-py3
$ nvidia-docker run -it --rm -v $(pwd):/work nvcr.io/nvidia/tensorrt:19.04-
py3
# cd /workspace/tensorrt/data/resnet50
# /workspace/tensorrt/bin/trtexec --batch=128 --iterations=400 --
workspace=1024 --percentile=99
```

```
deploy=ResNet50_N2.prototxt --model=ResNet50_fp32.caffemodel --output=prob -
-int8
```

11.1.2 Interpreting the Results

Results are reported in time to infer the given batch size. To convert to images per second, compute BATCH_SIZE/AVERAGE_TIME.

11.2 TensorFlow RN50 Mixed Training

- The container used in this example is `nvcr.io/nvidia/tensorflow:19.04-py3`.
- The scripts for this test are in `/workspace/nvidia-examples/cnn`.
- The example is a synthetic training example, so no data is needed.
- `README.md` describes the functionality of this test.

11.2.1 Commands to Run the Test

```
$ docker pull nvcr.io/nvidia/tensorflow:19.04-py3
$ nvidia-docker run -it --rm -v $(pwd):/work
nvcr.io/nvidia/tensorflow:19.04-py3
# cd /workspace/nvidia-examples/cnn
# mpirun --allow-run-as-root -np 1 python -u ./resnet.py --batch_size 256 --
num_iter 800 --precision fp16 --iter_unit batch --layers 50
```

11.2.2 Interpreting the Results

This benchmark reports training performance in images per second at each reporting iteration. Use the last few values reported to represent training performance.

Chapter 12. Troubleshooting

12.1 Forums

NVIDIA forums are a very inclusive source of solutions to many problems that may be faced when deploying a virtualized environment. Search on the NVIDIA forums located at https://gridforums.nvidia.com/ first.

You may also wish to look through the NVIDIA Enterprise Services Knowledgebase to find further support articles and links at https://nvidia-esp.custhelp.com/app/answers/list/autologout/1

Keep in mind that not all issues within your deployment may be answered in the NVIDIA vGPU forums. You may also have to reference forums from the hardware supplier, the hypervisor and application themselves.

Some examples of other key forums to look through are as follows:

- HPE ProLiant Server Forums: https://community.hpe.com/t5/ProLiant/ct-p/proliant
- Dell Server Forums: https://www.dell.com/community/Servers/ct-p/ESServers
- Lenovo Server Forums: https://forums.lenovo.com/t5/Datacenter-Systems/ct-p/sv_eg

12.2 Filing a Bug Report

When filing a bug or requesting support assistance, it is critical to include information about the environment, so that the technical staff that can help you resolve the issue. NVIDIA includes the `nvidia-bug-report.sh` script in the RPM installation package to collect and package this critical information. The script collects the following information:

- RHEL version
- PCI information
- CPU information
- GPU information
- RPM information
- NVRM messages from vmkernel.log
- System `dmesg` output
- Which virtual machines have vGPU configured
- NSMI output

When running this script:

- You may specify the output location for the bug report using either the `-o` or `–output` switch followed by the output file name. If you do not specify an output directory, the script writes the bug report to the current directory.
- If you do not specify a file name, the script uses the default name `nvidia-bug-report.log.gz`.
- If the selected directory already contains a bug report file, the script changes the name of that file to `nvidia-bug-report.log.old.gz` before generating a new `nvidia-bug-report.log.gz` file.

To collect a bug report, issue the command:

```
$ nvidia-bug-report.sh
```

The system displays the following message during the collection process:

```
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 vSphere 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.
```

Be sure to include the **nvidia-bug-report.log.gz log** file when reporting problems to NVIDIA.

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

http://www.nvidia.com