MAK TECHNOLOGIES

Running **MAK ONE** Products on Virtual Machines or Docker Containers

ST Engineering
North America

Running MAK ONE Products on Virtual Machines or Docker Containers

This document describes MAK Technologies' support for virtual machines and two configurations for running MAK ONE products on a virtual machine. It also describes how to run MAK ONE products using Docker containers.

See the appropriate topic to learn more.

- "Virtual Machine Overview" and "Docker Container Overview" on the next page
- "1. Setting Up a Public Cloud" on page 5
- "2. Setting Up a Private Cloud" on page 11
- "3. Running VR-Vantage and VR-Forces with Docker" on page 29

Document ID: MAK-0-5-221115

Virtual Machine Overview

Many MAK customers want to run our applications in the cloud or on virtual machines. Applications that do not require 3D graphics, such as VR-Link or MAK Data Logger, can run on virtual machines without any special configuration work. However, applications such as VR-Vantage, VR-Forces, and VR-Engage, which have high quality 3D graphics, are difficult to set up because the virtual machines do not have easy access to the graphics card. Some customers using particular hardware and software configurations have been able to run graphics-intensive MAK ONE applications in the cloud and on virtual machines.

MAK has tested some of these configurations so that we can provide assistance to customers who want to run our software virtually. Due to the special hardware required and many possible configurations of it, our ability to fully test, debug, and support our products on these platforms is extremely limited. This document describes specific hardware configurations (using Amazon Web Services or a local virtual machine) that we have tested. If you are using other hardware or software environments, we will probably be unable to assist you if you run into problems. Please ask your chosen software and hardware vendors for help.

We update this document periodically as the technology changes and we achieve more success supporting MAK ONE products on virtual platforms.

Docker Container Overview

Docker containers allow you to package an application in a containers, which makes it portable to Linux or Windows. For example, you can run MAK ONE applications built for Linux Red Hat 8 on machines running the Linux Red Hat 7 OS.

1. Setting Up a Public Cloud

This section explains how to run MAK ONE products in the cloud, specifically, Amazon Web Services (AWS).

1.1. Setting Up a Virtual Machine Using Amazon Web Services 6
- 1.1.1 Create an AWS VM Instance 6
- 1.1.2 Connect to the Running VM 8
- 1.1.3 Install the NVIDIA Driver Card 8
- 1.1.4 Activate NVIDIA GRID Capabilities 8
- 1.1.5 Optimize GPU Settings 8
- 1.1.6 Install Software on VM 8
- 1.1.7 Set Up the MAK RTI 8
- 1.1.8 Run MAK Software 9

1.2. Installing Desktop Cloud Visualization with NICE DCV 9

1.1. Setting Up a Virtual Machine Using Amazon Web Services

1.1.1 Create an AWS VM Instance

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance.

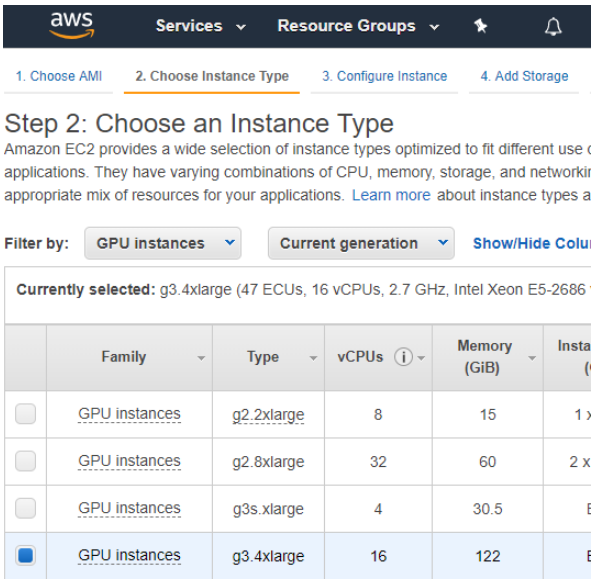
1. Create an AWS account.
2. Sign in to the AWS Console: https://console.aws.amazon.com
3. Navigate to EC2 Dashboard.
4. Find Instances Link - EC2 Dashboard->INSTANCES->Instances.
5. Create a G2 Instance with at least 150 GB storage.
6. Click the Launch Instance button. This starts the wizard to select all portions of the AMI.

Define the AMI

1. Click 1. Choose AMI.
2. Select Microsoft Windows Server 2016 Base. This is the Amazon Machine Image (AMI).
3. Select 2. Choose Instance Type (Figure 1).
4. In the Filter by list, select GPU Instances.

   GPU instances provide graphics processing units (GPUs) along with high CPU and network performance for applications benefiting from highly parallelized processing, including 3D graphics, HPC, rendering, and media processing applications.

Figure 1. Choosing instance type



5. In the list of GPU Instances, select the row Family: G2 Instance Type: g3.4xlarge.

   G2 Instances are backed by 1 x NVIDIA GRID GPU (Kepler GK104) and 8 x hardware hyperthreads from an Intel Xeon E5-2670.

6. Click 3. Configure Instance Details. Keep the default settings.
7. Click 4. Add Storage (Figure 2).
8. Change the size to at least 150 GB.

Figure 2. Add Storage

9. Click 6. Configure Security Group.
10. Set the security group to allow your public addresses to access AWS. Table 1 shows the setup used by MAK to test use of AWS.

Table 1. Security group setup

	Type	Protocol	Port	Source
Remote Desktop Connection	RDP	TCP	3389	Your public IP address/32
MAK RTI	Custom TCP Rule	TCP	4000	Your public IP address/32
MAK RTI	Custom TCP Rule	TCP	5000	Your public IP address/32
TightVNC	Custom TCP Rule	TCP	5800	Your public IP address/32
TightVNC	Custom TCP Rule	TCP	5900	Your public IP address/32

11. Click 7. Review.
12. Log in to the virtual machine:
    a. Create and Save a Key-Pair file (make sure that you save this file).
    b. Once the G2 Instance is up and running, go to the Connect button.
    c. Click the Download Remote Desktop File button. This makes an easy access launch file. *.RDP.

d. Click the Get Password button, and follow instructions to get a password using the key-pair file. This password is for logging into the VM as Administrator.

1.1.2 Connect to the Running VM

Once the VM is up and running, connect to it:

1. Log in using the RDP file or by launching Remote Desktop with the VM information given from the Get Password button.
2. Download and install TightVNC (TightVNC runs over ports 5900/5800).

   TightVNC will allow the setup of the NVIDIA GRID M60 graphics card (version of card used on AWS G3 instance as of August 5, 2019). Remote Desktop Connection will only use a basic graphics driver and so cannot be used to install NVIDIA drivers.
3. Continue on the VM using TightVNC.

1.1.3 Install the NVIDIA Driver Card

➤ To install the NVIDIA driver on Windows, see the instructions “Installing the NVIDIA Driver on Windows” at https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/install-NVIDIA-driver-windows.html.

1.1.4 Activate NVIDIA GRID Capabilities

➤ Activate NVIDIA GRID capabilities as described in “Activate NVIDIA GRID Virtual Applications” at https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/activate_grid.html

1.1.5 Optimize GPU Settings

➤ Optimize the GPU settings for G3 as described in “Optimizing GPU Settings” at https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/optimize_gpu.html.

1.1.6 Install Software on VM

We recommend that you install Chrome and Notepad++ or a similar text editor.

1. Download VR-Vantage, VR-Forces, or VR-Engage with data.
2. If you are using HLA, download the MAK RTI.
3. Install MAK software.
4. Test that VR-Vantage, VR-Forces, or VR-Engage runs on the VM.
5. Shut down MAK software.

1.1.7 Set Up the MAK RTI

To get MAK RTI to work between the AWS VM and your local machine, you must change the *rid.mtl* on both machines.

Local Machine - MAK RTI

Edit the following lines in the *rid.mtl* file:

```
(setqb RTI_configureConnectionWithRid 1)
(setqb RTI_useRtiExec 1)
;; Set to the public IP address of the VM
(setqb RTI_tcpForwarderAddr "AWS VM Public IP Address")
(setqb RTI_fomDataTransportTypeControl 2)
;; Set to your local machine network interface
(setqb RTI_networkInterfaceAddr "Local IP Address")
```

AWS VM - MAK RTI

Set the following lines in the *rid.mtl* file to:

```
(setqb RTI_configureConnectionWithRid 1)
(setqb RTI_useRtiExec 1)
;; Set to the private IP address of the AWS VM
(setqb RTI_tcpForwarderAddr "AWS Local IP ADdress")
(setqb RTI_fomDataTransportTypeControl 2)
;; Set to the private ip address of the AWS VM
(setqb RTI_networkInterfaceAddr "AWS Local IP ADdress")
```

Start the MAK RTI

➤ **To start the MAK RTI:**

1. On the AWS VM, start the RTI Forwarder.
2. Start the rtiexec.
3. On the local machine, start any federate using HLA, such as MAK Data Logger or VR-Forces.

1.1.8 Run MAK Software

➤ **To start MAK 3D intensive software:**

1. On the AWS VM, start VR-Vantage or VR-Forces using HLA.
2. On the local machine, start any federate using HLA.

1.2. Installing Desktop Cloud Visualization with NICE DCV

While Remote Desktop and TightVNC allow you to connect to the AWS instance, we have found that they do not perform well when using a 3D-intensive graphics application such as VR-Forces, VR-Vantage, or VR-Engage, especially when moving through a 3D world.

We have found software that is available to use with AWS called Desktop Cloud Visualization (DCV) from NICE Software. The use of DCV is free when used with your AWS instance. Installing NICE DCV on your instance allows you to access your instance from a browser or a simple desktop client application. Using a browser means there is no need to install any software on your remote client system. Through

the browser, you can control the instance and installed MAK software. You can move through the 3D world by switching the mouse control from absolute to relative mouse mode.

The browser method currently allows only the use of the mouse and keyboard. To use joysticks and gamepads, you must use the desktop client. Another reason to use the desktop client application instead of the browser is that there is an increase in performance. In tests, we have seen an improvement in latency when connecting through the client over the browser.

➤ To install the NICE DCV server on your instance, follow the "Licensing the NICE DCV Server" instructions at https://docs.aws.amazon.com/dcv/latest/adminguide/setting-up-license.html.

The *NICE DCV Administrator Guide* at https://docs.aws.amazon.com/dcv/latest/adminguide/dcv-ag.pdf provides further instructions for setting up NICE DCV Server on your instance.

The *NICE DCV User Guide* at https://docs.aws.amazon.com/dcv/latest/userguide/dcv-ug.pdf includes more details on how to use the DCV browser or client software to access your instance.

2. Setting Up a Private Cloud

This section explains how to run MAK ONE products on local machines that have virtual machines (VM) using VMWare. In "2.1. Setting Up a Local Virtual Machine" on the next page, you set up a single VM using the “pass-through” method so that the virtual machine can access the graphics card. "2.2. Implementing a Private Cloud using NVIDIA GRID and VMware" on page 19 describes the pieces used to define multiple VMs using VMWare and NVIDIA grid based graphics cards to create a private cloud environment.

2.1. Setting Up a Local Virtual Machine 12
- 2.1.1 Hardware and Software 12
- 2.1.2 Setting Up the VMware ESXi Hypervisor Host 12
- 2.1.3 Setting up Windows in VMware 13
- 2.1.4 Configure Virtual Machine Video Card 3D Capabilities 14
- 2.1.5 Enable NVIDIA Passthrough Card on a VM 17

2.2. Implementing a Private Cloud using NVIDIA GRID and VMware 19
- 2.2.1 Setting up VMware ESXi Hypervisor Host 20
- 2.2.2 Setting up a VM on VMware ESXi Hypervisor Host 23
- 2.2.3 Setting up NVIDIA Licensing for GRID Technology 26
- 2.2.4 Setting up VMware vCenter 27

2.3. Installing Desktop Cloud Visualization with VMWare Horizon 27

2.1. Setting Up a Local Virtual Machine

This section explains how to run MAK ONE products on a local machine that has a VMware virtual machine (VM). It sets up a "pass-through" so that the virtual machine can access the graphics card.

2.1.1 Hardware and Software

We used the following hardware and software:

- RAVE Ignition ATX Tower
- Intel Core i7-4790K 8M Cache, 4.40GHz
- 8 GB RAM
- NVIDIA Quadro M6000 graphics card
- VMware vSphere/ESXi 6.5

These instructions using passthrough are only for Quadro-based cards.

2.1.2 Setting Up the VMware ESXi Hypervisor Host

1. Download VMware vSphere ESXi.
2. Create a bootable image of the VMware vSphere ESXi and make a disk (DVD).
3. Insert the bootable disk into your host machine and turn it on. (You may need to enable booting from a disk in your BIOS.)

   The disk will lead you through the installation of VMware vSphere ESXi onto the host computer.
4. On the host BIOS, enable Intel Virtualization:
   a. Enable Intel Virtualization Technology.
   b. Enable VT-d.

   The location of these two settings in the BIOS may differ between machines.

Accessing the ESXi Host Remotely

In a web browser, log in to the host:

`https://IP_Address/ui/#/login`

Example: `https://10.0.2.129/ui/#/login`

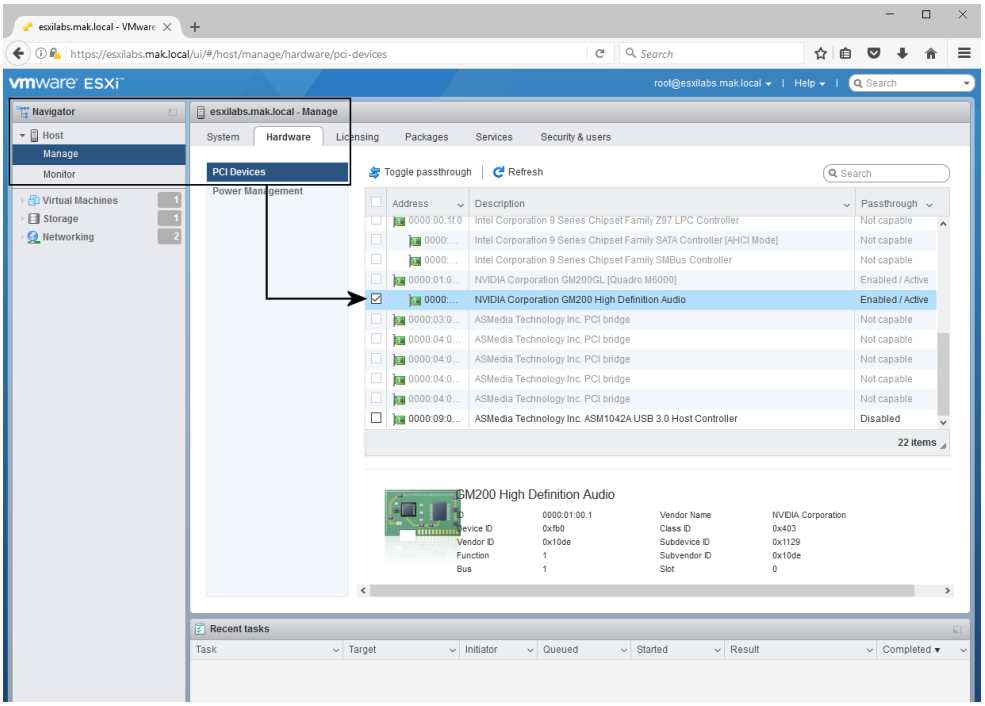
If you register a DNS name, you can use the following type of URL:

`https://DNS_name/ui/#/login`

Example: `https://esxilabs.mak.local/ui/#/login`

The VMware ESXi host interface opens (Figure 3).

Figure 3. VMware ESXi host interface



Enable the NVIDIA PCI Device

1. In the Navigator Panel, expand the Host entry.
2. Select Manage.
3. Select the Hardware tab.
4. Select PCI Devices.
5. In the list of PCI devices, select the NVIDIA card. If you cannot locate the Quadro (M6000) card in the PCI device list, make sure that you have enabled Intel Virtualization in the BIOS.
6. Click the “Toggle passthrough” button.

2.1.3 Setting up Windows in VMware

Create a Virtual Machine in VMware. We recommend 60 GB hard drive space or larger. Once the VM is created, install Windows 10 x64.

To get VMware SVGA graphics (and not use basic Windows graphics):

1. Go to VM > Settings.
2. Select CD/DVD and set to auto detect.
3. Select Floppy and set to auto detect.

2.1.4 Configure Virtual Machine Video Card 3D Capabilities

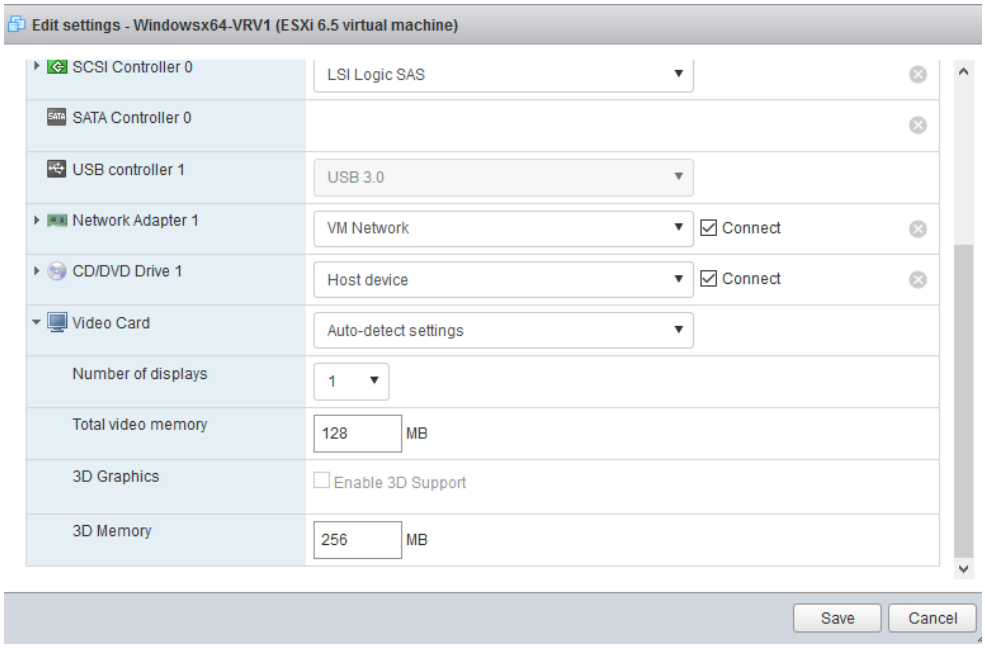
➤ **To configure virtual machine video card 3D capabilities using the vSphere Client:**

1. Connect to the ESXi host, as described in "Accessing the ESXi Host Remotely" on page 12.
2. Select the virtual machine.
3. Select the Summary tab.
4. Under Commands, click Edit Settings.
5. Under Hardware, click Video card (Figure 4).
6. Under Displays and video memory, set the video card 3D capabilities.

   Set the Total Video Memory to a value between 64 MB and 128 MB. Most applications should work with 128 MB. Video memory values larger than 128 MB are available only with virtual machines with hardware version 9 or 10.
7. Click Save.

You cannot set the 3D renderer from the vSphere Client. Use the vSphere Web Client to configure video card 3D capabilities.

Figure 4. Configuring video card

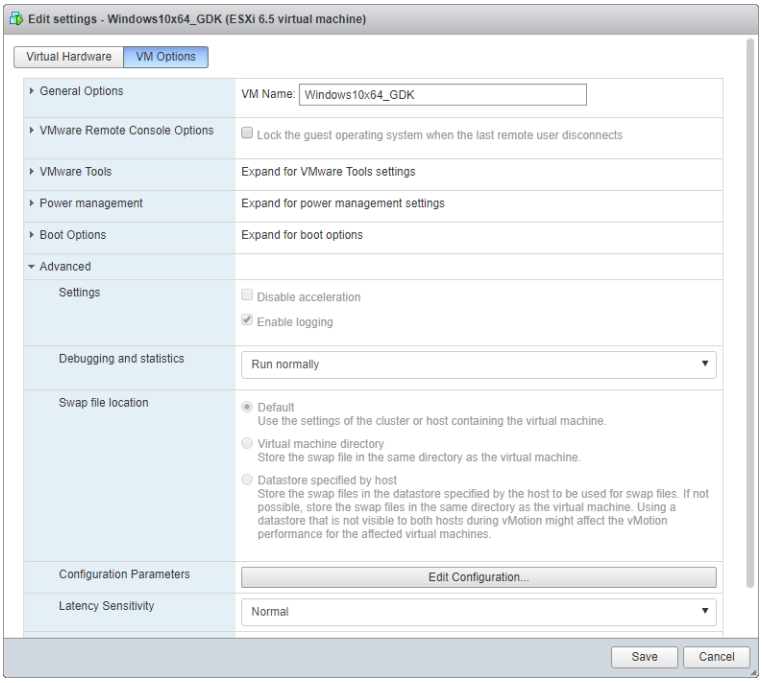


Configure the Hypervisor CPU ID

Setting the hypervisor CPU ID to false tricks the hypervisor into believing it is not running a VM. This allows us to pass-through the NVIDIA graphics card. The downside to this is that there can only be one VM machine run at one time.

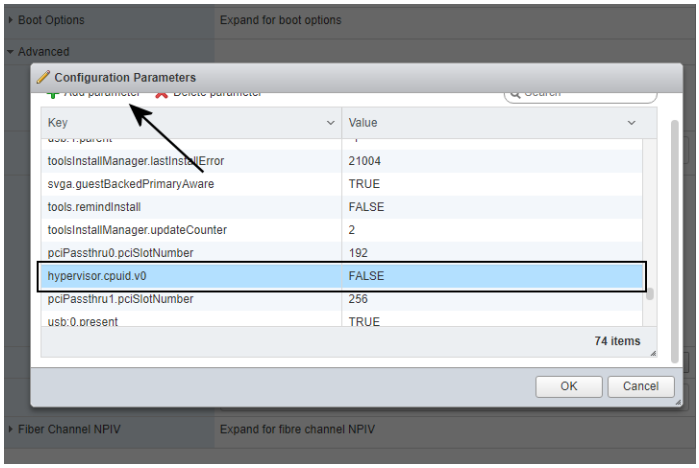
1. In the Edit Settings window, click the VM Options button.
2. Expand the Advanced tab (Figure 5).

Figure 5. Advanced VM Options



3. In the Configuration Parameters section, click Edit Configuration. The Configuration Parameters dialog box opens.
4. Click Add Parameter.
5. Select hypervisor.cpuid.vo and set it to False (Figure 6).

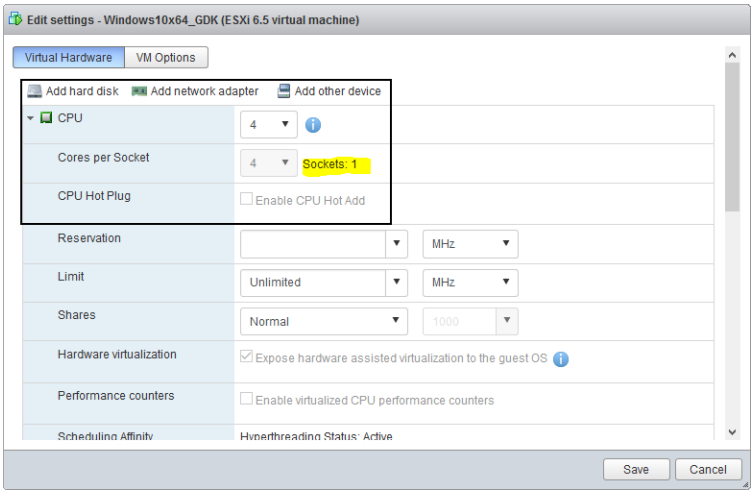
Figure 6. Set hypervisor.cpuid.v0



6. Click OK.
7. In the Edit Settings dialog box, click Save.

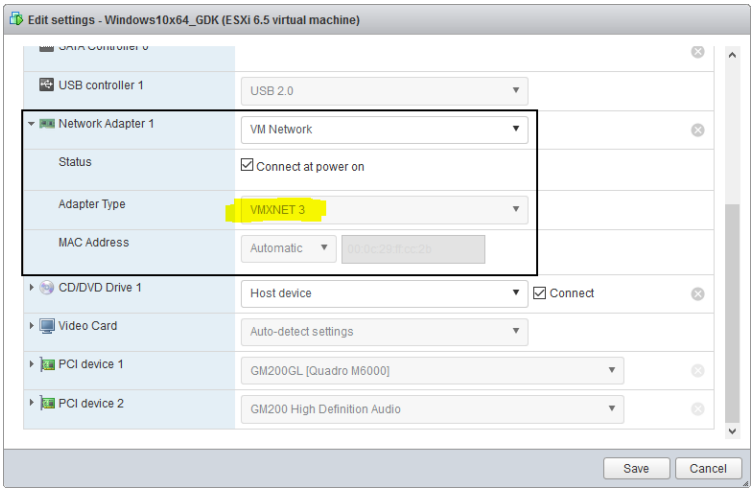
8. Click the Virtual Hardware button.
9. Expand the CPU section.
10. Make sure that on the Cores per Socket line, the number of Sockets is 1 (Figure 7).

Figure 7. CPU settings



11. Expand the Network Adapter section.
12. Make sure the Adapter Type is set to VMXNET3.

Figure 8. Adapter Type



> Setting the Network Adapter to VMXNET means that VMWare Tools is installed. This is done once the VM is running.

13. In the VMware Remote Console, choose **VMRC > Manage > Install VMware Tools** (or **Reinstall VMware Tools**).

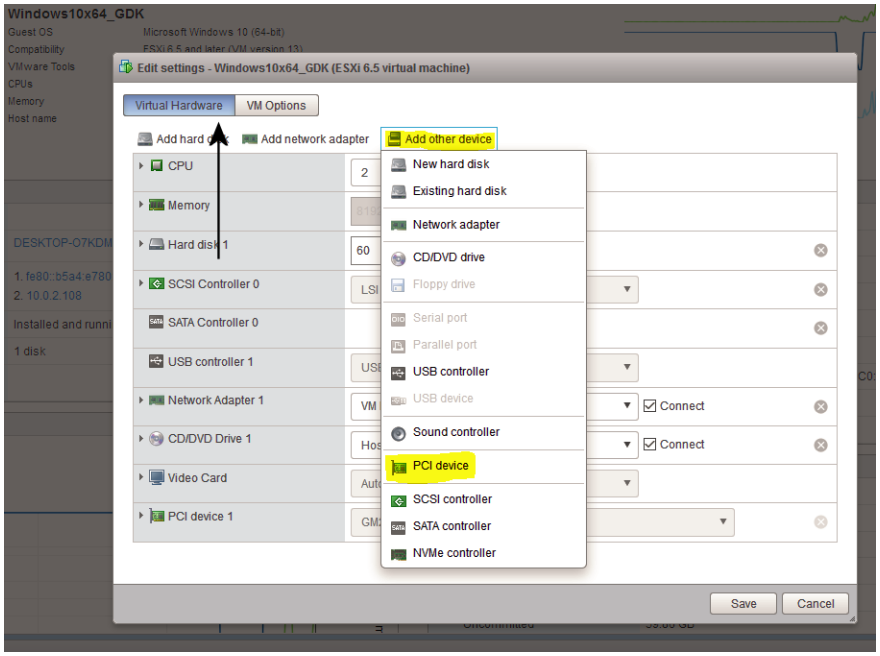
2.1.5 Enable NVIDIA Passthrough Card on a VM

1. Load the VM.
2. Install TightVNC or some other application for remote viewing, because the basic VMware console will not work. The VMware console will launch, but the screen will remain black and there will be no interface for user control once you enable the NVIDIA Quadro graphics card.
3. Install the NVIDIA software for the installed Quadro card and follow the installation wizard.
4. Restart the VM after the NVIDIA driver is installed.

Add a PCI Device for Video

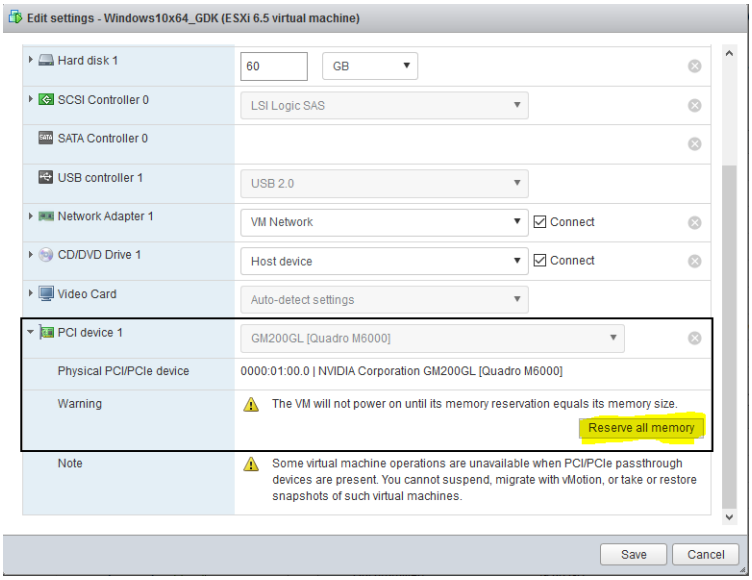
1. Navigate to the **VM > Edit or Actions > Edit Settings**.
2. Select the Virtual Hardware Tab (Figure 9).

Figure 9. Virtual Hardware - Add Other Device



3. Click Add Other Device.
4. Select PCI Device. This adds a PCI Device entry for video.
5. Select the NVIDIA card that was passed through (Figure 10).
6. Scroll down and click the Reserve all memory button.

Figure 10. Reserve All Memory

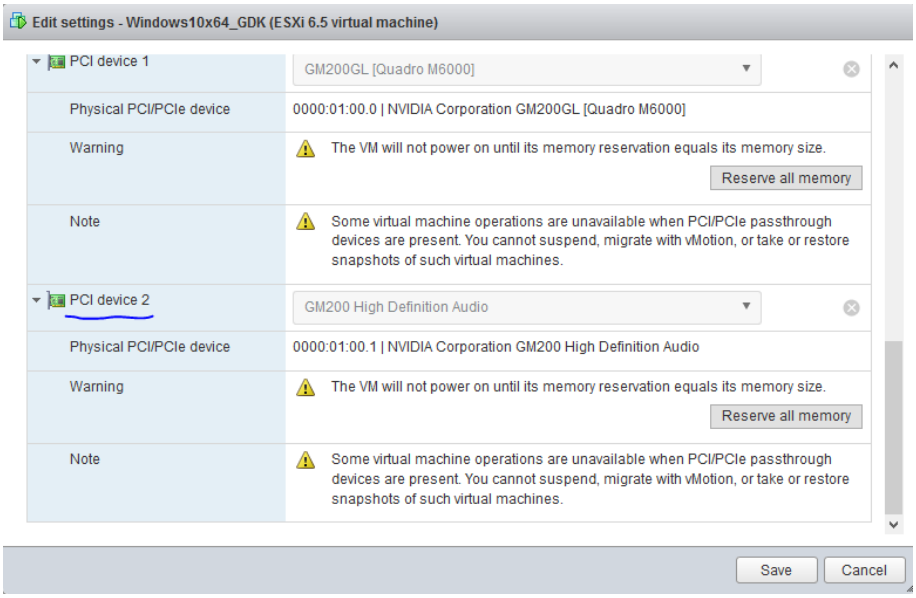


7. Save the settings.

Add a PCI Device for Audio

1. Select the Virtual Hardware Tab (Figure 9).
2. Click Add Other Device.
3. Select PCI Device. This adds a PCI Device entry for audio (Figure 11).
4. Select the NVIDIA card audio entry that was passed through.

Figure 11. PCI device for audio



5. Save the settings.

Start Virtual Machine and Install Applications

1. Power on the VM and wait for it to load.
2. Launch your viewer (for example, TightVNC) and log in.
3. Right click on the VM desktop to verify that the NVIDIA panel is present. This should indicate that you are using the NVIDIA card.
4. Install and run MAK ONE products.

2.2. Implementing a Private Cloud using NVIDIA GRID and VMware

This section explains how to set up a private cloud using NVIDIA GRID hardware and VMware. We used the following computer from AMAX with the following specifications:

- ServMax 2U Intel Xeon Server:
  - Intel Server System R2312WF0ZSSPP, 2U 12x 3.5" Bay
  - 2x - Intel® Xeon® Platinum 8160 Processor - 24core 2.1Ghz
  - 4x - 32GB DDR4 2666MHz x4 DR RDIMM CT32G
  - Performance Optimized BIOS - Amax
- Graphics Card:
  - NVIDIA Tesla P40 24GB GDDR5
  - NVIDIA GRID version 390.72
  - NVIDIA driver version 391.81
- Virtual Machine Software:
  - VMware vSphere/ESXi 6.7
  - VMware vCenter
  - VMware Horizon Client - Desktop Virtualization Client

The hardware for setting up NVIDIA GRID-based virtual machines must fully support Intel Virtualization Technology and VT-d (Intel ® VT for Directed I/O). In general, Intel Core i# (example: i7) cannot enable Intel ® VT for Directed I/O. Therefore, Intel Xeon processors are recommended. However, you must still verify that your chosen processor fully supports the Virtualization Technology.

2.2.1 Setting up VMware ESXi Hypervisor Host

1. On the ESXi host machine, enable Intel Virtualization Technology and VT-d (Intel ® VT for Directed I/O) on the BIOS.

   The location of these items in the BIOS may differ between machines. If these BIOS settings are not enabled, VMware displays the pass-through capability as grayed-out. This is one indication that the BIOS settings are not set correctly.

2. Download VMware ESXi and vSphere server v6.7.
3. Download the NVIDIA vGPU package that supports VMware vSphere. Documentation is available at https://docs.NVIDIA.com/grid/latest/pdf/grid-vgpu-release-notes-vmware-vsphere.pdf.
4. Create bootable media (USB or DVD) for VMware ESXi 6.7.
5. Set the host to boot from your chosen media and run the installer.
6. Set up the hostname, datastores, and networking as described in the VMware documentation.
7. Upload your ISO image of choice, which will be used when you create your VMs, to a datastore on the ESXi host.

   We created Windows 10 VMs during our tests. You can also create ISO install disks if you prefer, but datastores are a good way to keep your ISO images for use rather than physical media that needs to be placed into the physical machine before using.

8. Install the NVIDIA vGPU package you downloaded in step 3 onto the ESXi host as per NVIDIA's Virtual GPU Software.

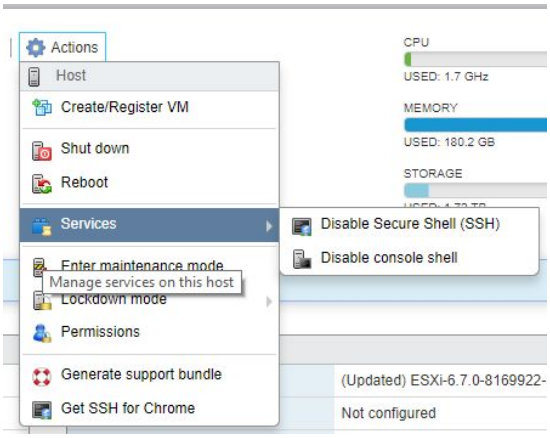
   Please see “Installing and Configuring the NVIDIA Virtual GPU Manager for VMware vSphere” at https://docs.NVIDIA.com/grid/latest/grid-vgpu-user-guide/index.html#vmware-vsphere-install-configure-vgpu for instructions.

Now that the VMware vSphere/ESXi is installed and running, check that the NVIDIA Virtual GPU Manager is installed on the hypervisor. The easiest way to do this is to SSH into the ESXi host.

1. Enable SSH from the vSphere Web Client:
   a. Select the host.
   b. Click Manage and keep Settings selected.
   c. Click Security Profile.
   d. In the Services section, click Edit.
   e. Select SSH.
      — To temporarily start or stop the service, click the Start or Stop button.
      — To change the Startup policy across reboots, select Start and stop with host and reboot the host.
   f. Click OK.

g. Log in to the ESXi Shell remotely and run ESXi Shell commands.

Figure 12. Example of SSH (Secure Shell) Enabled from Actions menu in VMware ESXi host



2. Connect to the ESXi host via an SSH client. We used PuTTY.
3. In the SSH Shell, enter:

```
[root@gp40]: vmkload_mod -l | grep NVIDIA
```

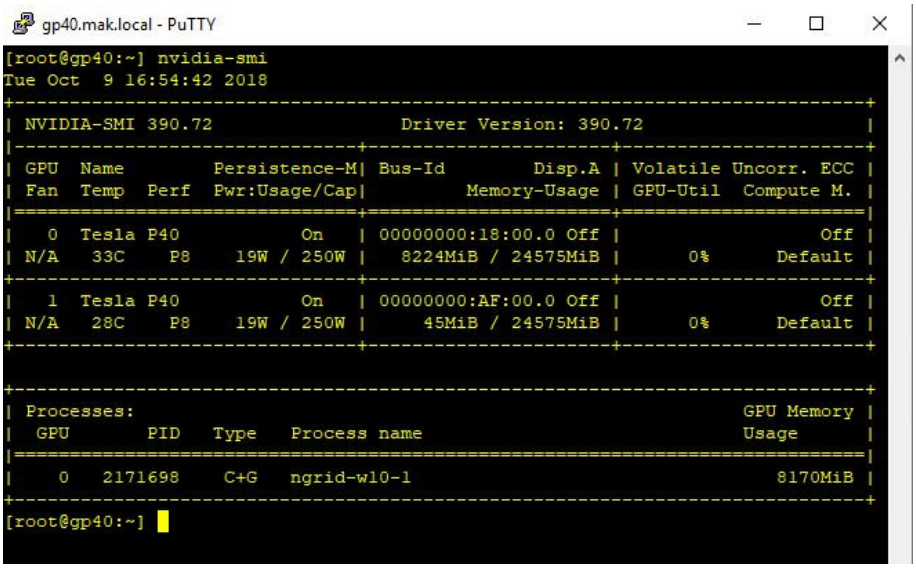
The result should look like:

```
NVIDIA 310    13816
```

Figure 13. vmkload output

4. Use NVIDIA-smi to list the status of all GPUs that are part of the current ESXi host system. This gives you a good indication of the NVIDIA vGPU software and driver installed on your system. It also shows the GPU count and their Bus-Id. The ID is a value that can be used later if you want to only modify the settings of a single GPU verses all GPUs in the system.

Figure 14. NVIDIA-smi example output

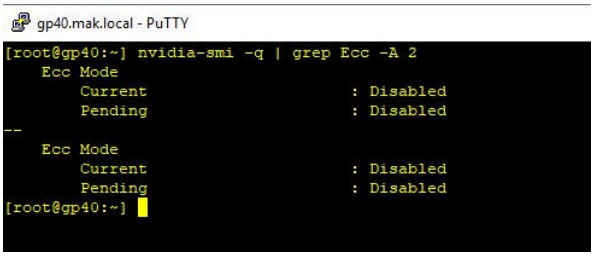


5. Check to see whether ECC memory is enabled. As per NVIDIA's Virtual GPU Software Documentation, Section 2.7. Disabling ECC Memory, check for ECC being enabled because “NVIDIA vGPU does not support ECC memory. If ECC memory is enabled, NVIDIA vGPU fails to start.” Type the following command (remember that grep is case sensitive):

   ```
   [root@gp40]: NVIDIA-smi -q | grep Ecc -A 2
   ```

   This command finds Ecc Mode and displays the next two lines that show whether it is enabled or disabled. There may be multiple results depending on the number of GPUs in the system.

Figure 15. Example Ecc Mode result indicating ECC memory is Disabled



6. To disable ECC memory for all GPUs, use the following command:

   ```
   [root@gp40]: NVIDIA-smi -e 0
   ```

7. To disable ECC memory for only a single GPU add the **-i Bus-Id** parameter to the command:

   ```
   NVIDIA-smi -i 00000000:18:00.0 -e 0
   ```

   where: Bus-Id=00000000:18:00.0 is the first GPU listed when running NVIDIA-smi command

8. Reboot the ESXi host.

9. Verify that the ECC memory is now disabled.

```
[root@gp40]: NVIDIA-smi -q | grep Ecc -A 2
```

The VMware ESXi host machine should now be ready to design and add virtual machines as per the VMware documentation.

2.2.2 Setting up a VM on VMware ESXi Hypervisor Host

Once you have logged in to your ESXi host and started to create a Windows-based VM, please revisit our hardware recommendations (listed on the MAK website at https://www.mak.com/support/hardware-recommendations) to help guide you through setting up the correct NVIDIA Virtual GPU Types.

For the test system running NVIDIA Tesla P40 cards, we set the Virtual GPU Type to P40-8Q.

This can be defined when specifying the shared PCI device as described in "Configuring a vSphere VM with NVIDIA vGPU" at https://docs.NVIDIA.com/grid/latest/grid-vgpu-user-guide/index.html#configure-vmware-vsphere-vm-with-vgpu.

If there are any questions, please contact us. We are happy to discuss your specific project/hardware requirements.

The following figures show how we set up our VM based on the above recommendations.

Figure 16. Example GRID VM Overview in VMware ESXi Host

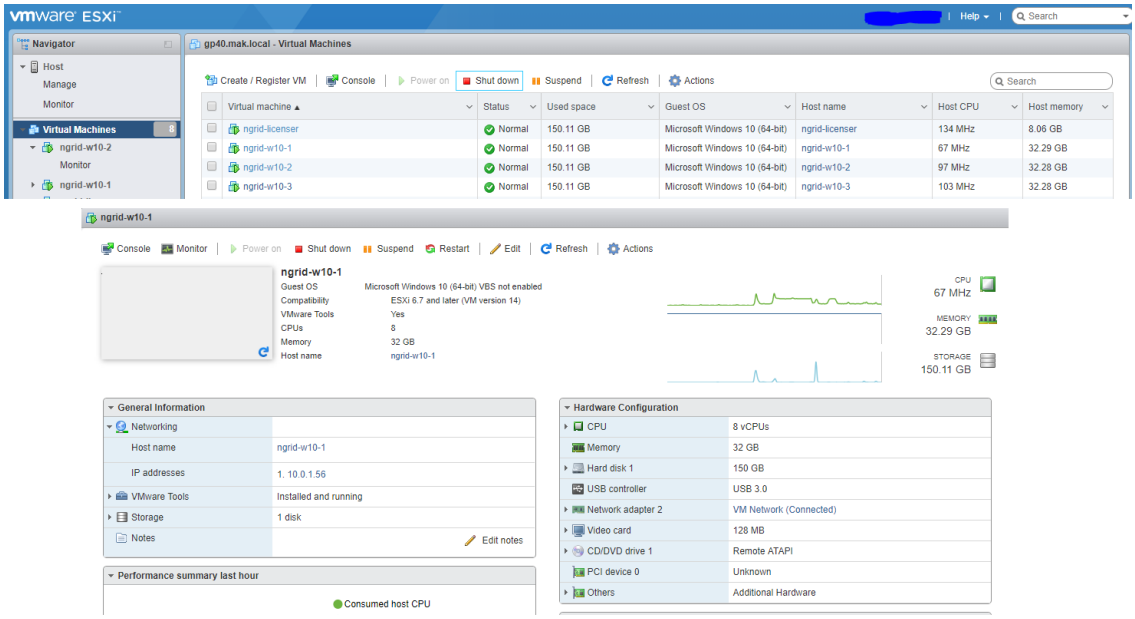


Figure 17. VM CPU Settings

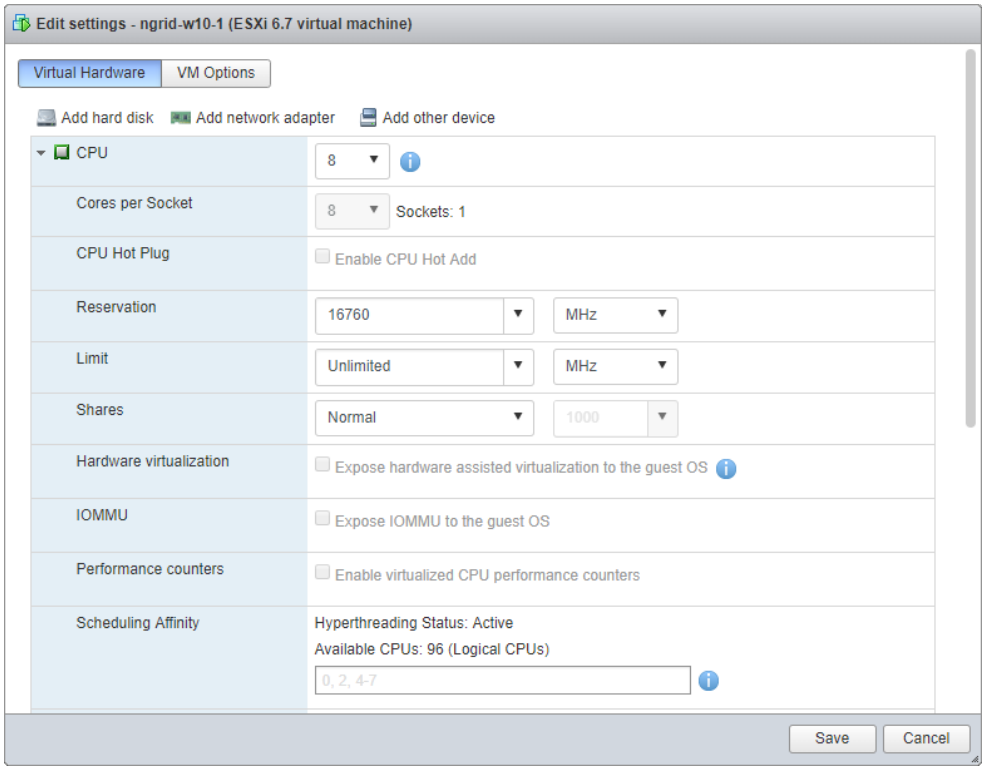


Figure 18. VM Memory Settings

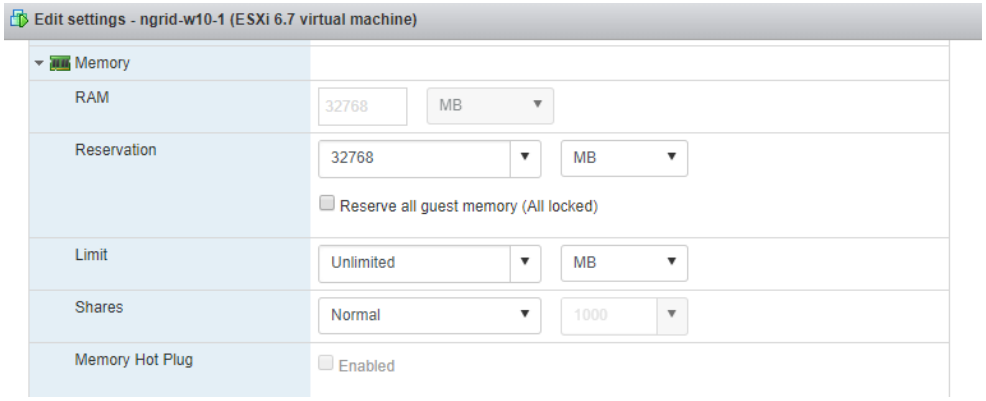


Figure 19. VM Hard Disk Settings

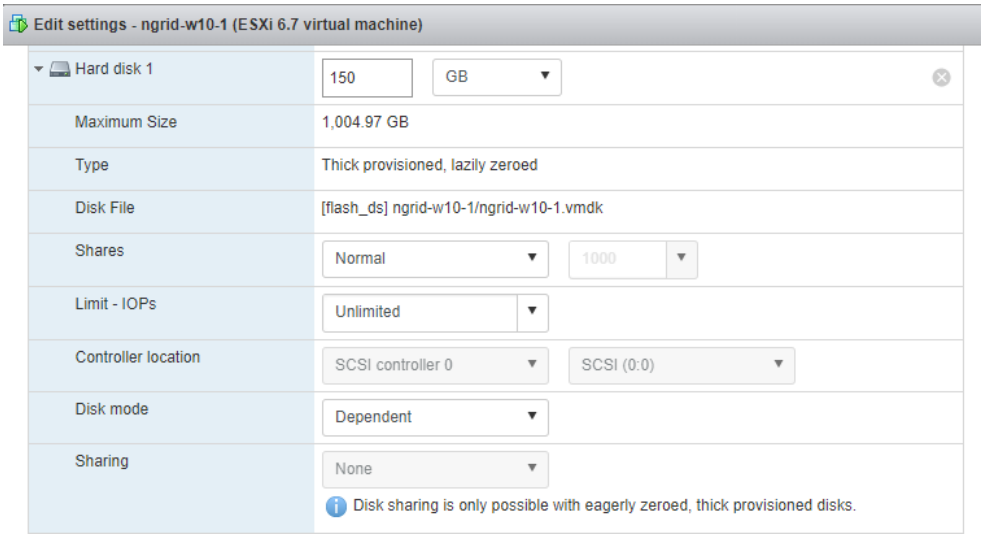


Figure 20. VM Network Adapter Settings

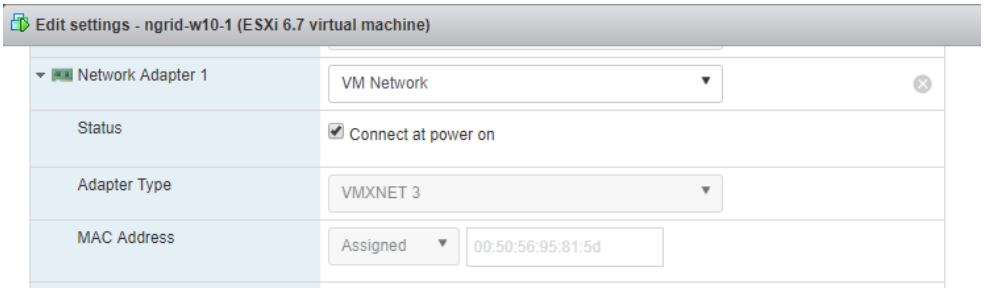


Figure 21. VM SCSI Controller Settings

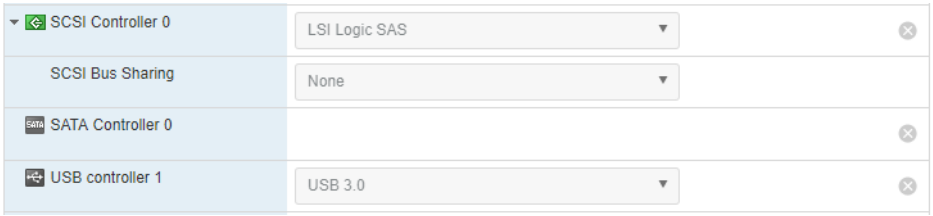
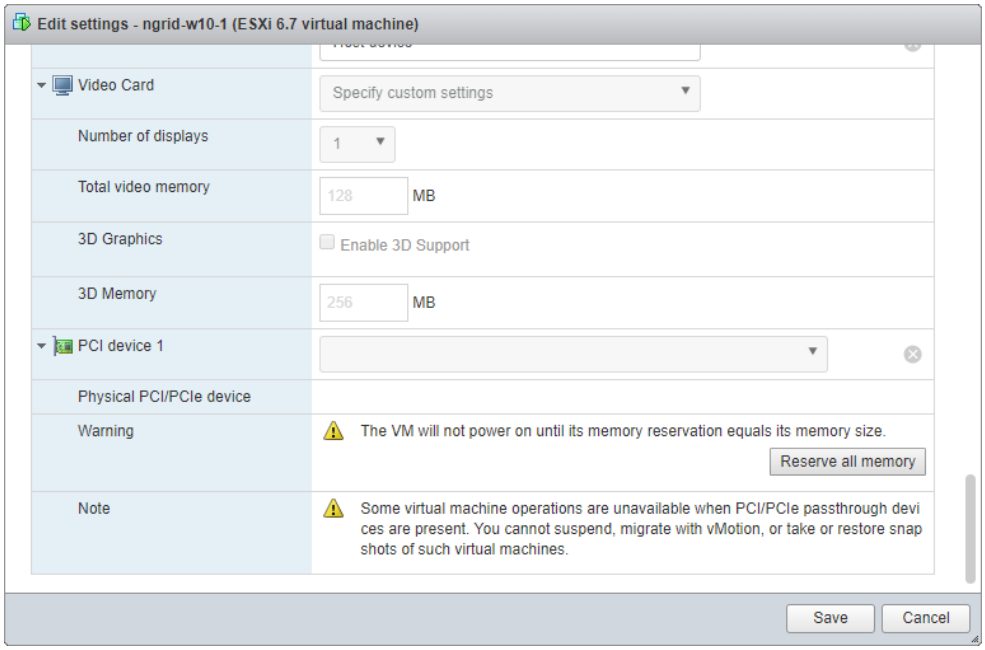


Figure 22. VM Video Card and PCI Device 1 Settings



2.2.3 Setting up NVIDIA Licensing for GRID Technology

You must install an NVIDIA license server. You can set it up on a separate machine, in which case the NVIDIA license server will not use up any resources from the NVIDIA GRID-based system, or you can run it on the same machine as the virtual machines.

If you install the license server on the same machine as the GRID-based VMs, make sure the machine has enough CPU and memory.

➤ **To install the NVIDIA license server:**

1. Download the license server from the NVIDIA Licensing Center at https://nvid.NVIDIA.com/dashboard/#/dashboard (We used an evaluation license to test this procedure.)
2. Generate a license for the MAC ID of the license server machine.
3. Upload the license file to the license server.
4. Point the server at the license.
5. Restart the fnls-server service to update the license.

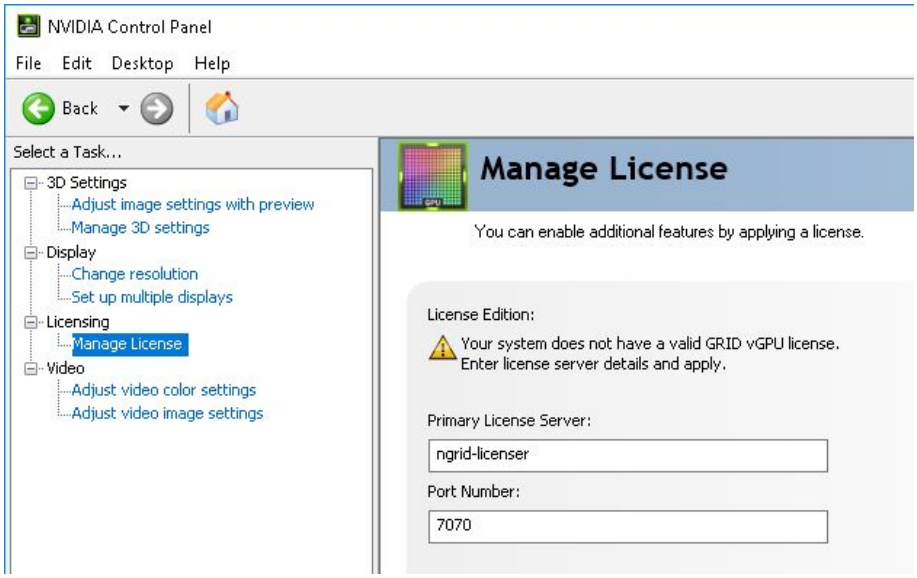
Set Up Each Virtual Machine to use the NVIDIA License

Once the license server is running with a valid license each, VM must be given access to the license server so it can check out a license.

➤ **To set up a virtual machine to use a license:**

1. Launch a VM and log in.
2. Right click on the desktop and open the NVIDIA Control Panel (Figure 23).

Figure 23. NVIDIA Control Panel



3. Under Licensing, select Manage License.
4. In the Primary License Server box, type the name of your license server.
5. In the Port Number box, type **7070**. The VM should then attempt to check out a license.

If the license is found, the VM is ready to use the NVIDIA GRID-based graphics card and any MAK ONE products that are installed.

If a GRID license is not found, an error message displays. If the license is not found and you are running VR-Engage, VR-Forces or VR-Vantage, the frame rate is restricted to 3 frames per second.

2.2.4 Setting up VMware vCenter

Setting up vCenter is an optional step and depends on your use case and budget. Since there are additional costs, you may decide not to add another layer when only running a single VMware ESXi host. However, if the design of your system is for multiple VMware ESXi hosts or you plan to set up more of a private cloud system, vCenter is designed to help manage all those VMware vSphere environments from a single interface.

2.3. Installing Desktop Cloud Visualization with VMWare Horizon

VMWare provides their own desktop cloud visualization software called Horizon Client, which comes with the VMWare Horizon Suite. It enables remote desktops to connect with the vSphere server to view your virtual machines. In our tests, it performed very well even when connecting to a virtual NVIDIA graphics machine running our 3D-intensive graphics applications such as VR-Forces, VR-Vantage, or VR-Engage.

To learn about installing VMWare Horizon and Horizon Client on your virtual machine, refer to the VMWare documentation center at https://docs.vmware.com/en/VMware-Horizon-7.

3. Running VR-Vantage and VR-Forces with Docker

This chapter is for users who want to run new versions of the MAK ONE graphical applications, such as VR-Vantage and VR-Forces, built for Red Hat 8 and running on Red Hat 7.

3.1. Prerequisites 30
- 3.1.1 Install the Latest Version of Docker Engine 30
- 3.1.2 Check the NVIDIA Graphics Driver 30
- 3.1.3 Install the Latest Version of Linux Kernel 30
- 3.1.4 Install NVIDIA-docker 30

3.2. Configuring Dockerfiles 30
- 3.2.1 Configuring the Centos8 Base Image 31
- 3.2.2 Building the OpenGL Runtime Container 32
- 3.2.3 Building the Base Container for MAK ONE Products 33

3.3. Running MAK ONE Applications 34
- 3.3.1 VR-Vantage 34
- 3.3.2 VR-Forces 37

3.4. Conclusion 39

3.1. Prerequisites

This chapter is specifically for Red Hat 7 host machines using Red Hat 8 builds of MAK ONE products. There are some prerequisites you must complete first.

3.1.1 Install the Latest Version of Docker Engine

The Docker package provided by YUM does not contain the necessary functionality required for passing through a GPU. You must download and install the latest version of Docker engine. See https://docs.docker.com/engine/install/centos to install the latest version of Docker.

3.1.2 Check the NVIDIA Graphics Driver

Because you will pass through the graphics card to your docker containers, you must ensure that you have the latest driver for your NVIDIA GPU. See this guide about updating your GPU driver: https://linuxconfig.org/how-to-install-the-NVIDIA-drivers-on-centos-7-linux.

> Make sure that you have the updated NVIDIA graphics driver before installing the latest version of the Linux kernel. You can run into graphical problems when you update the kernel if you do not have the latest driver.

3.1.3 Install the Latest Version of Linux Kernel

Docker uses the host machine's kernel to base the containers on. See https://phoenixnap.com/kb/how-to-upgrade-kernel-centos to update your kernel. For this guide, we used kernel version 4.4.225-1 (kernel-lt). You also need to install the kernel-lt-devel package.

3.1.4 Install NVIDIA-docker

NVIDIA develops and maintains libraries that you can use in docker containers. To install and configure NVIDIA-docker, follow the quick start guide here: https://github.com/NVIDIA/NVIDIA-docker.

> Since you will not use the docker version packaged through YUM, you can skip over the "RHEL Docker or Podman" section of the quick start guide.

3.2. Configuring Dockerfiles

Once you complete all of the prerequisites, the next step is to build your base docker containers. In this guide, you will create three different Dockerfiles: a base Centos8 image, a libglvnd image based on your Centos8 image, and an image containing the necessary libraries to run vrvStealth.

NVIDIA has an entire git repository dedicated to Docker images: https://gitlab.com/NVIDIA/container-images/opengl/-/tree/centos7. At this time,

NVIDIA does not provide Centos8 images for OpenGL in Docker. We predominantly use the same contents of the Dockerfiles in the repository mentioned above, with some slight changes.

3.2.1 Configuring the Centos8 Base Image

For your first image, create a base image that contains some necessary libraries required by your build of NVIDIA. Fortunately, the Dockerfile is identical to the one for Centos7: https://gitlab.com/NVIDIA/container-images/opengl/blob/centos7/base/Dockerfile.

```
  ARG from

  FROM ${from}
  LABEL maintainer "NVIDIA CORPORATION <cudatools@NVIDIA.com>"

  RUN yum install -y \
          libXau libXau.i686 \
          libXdmcp libXdmcp.i686 \
          libxcb libxcb.i686 \
          libXext libXext.i686 \
          libX11 libX11.i686 && \
          rm -rf /var/cache/yum/*

  # NVIDIA-container-runtime
  ENV NVIDIA_VISIBLE_DEVICES \
          ${NVIDIA_VISIBLE_DEVICES:-all}
  ENV NVIDIA_DRIVER_CAPABILITIES \
          ${NVIDIA_DRIVER_CAPABILITIES:+$NVIDIA_DRIVER_
CAPABILITIES,}graphics

  # NVIDIA-docker v1
  RUN echo "/usr/local/NVIDIA/lib" >> /etc/ld.so.conf.d/NVIDIA.conf &&
\
      echo "/usr/local/NVIDIA/lib64" >> /etc/ld.so.conf.d/NVIDIA.conf

  # Required for non-glvnd setups.
  ENV LD_LIBRARY_PATH /usr/local/NVIDIA/lib:/usr/local/NVIDIA/lib64

  # Required for non-glvnd setups.
  RUN echo '/usr/$LIB/libGL.so.1' > /etc/ld.so.preload && \
      echo '/usr/$LIB/libEGL.so.1' >> /etc/ld.so.preload

  # ld.so will print errors when this statement is executed, this is
normal.
  ONBUILD RUN rm /etc/ld.so.preload
```

After saving this Dockerfile out as centos8-NVIDIA.dockerfile, you can build this base image by running this command:

```
docker build --pull -t "centos8-NVIDIA" \
   --build-arg "from=centos:8" \
   -f centos8-NVIDIA.dockerfile .
```

In this code, you give the container a specific name and pass a build argument to it. The `--pull` argument tells the Docker build daemon to always pull the latest version of the base image.

3.2.2 Building the OpenGL Runtime Container

Next, create the libGL Dockerfile. This Dockerfile uses the centos8 base image you created to build the required libraries and run graphical applications. The following Dockerfile is based on https://gitlab.com/NVIDIA/container-images/opengl/blob/centos7/glvnd/runtime/Dockerfile with one change.

```
ARG from

# Build libglvnd
FROM centos:8 as glvnd

RUN yum install -y \
        git \
        make \
        libtool \
        gcc \
        pkgconfig \
        python3 \
        libXext-devel \
        libX11-devel \
        xorg-x11-proto-devel && \
    rm -rf /var/cache/yum/*

ARG LIBGLVND_VERSION

WORKDIR /opt/libglvnd
RUN git clone --branch="${LIBGLVND_VERSION}"
https://github.com/NVIDIA/libglvnd.git . && \
     ./autogen.sh && \
     ./configure --prefix=/usr/local --libdir=/usr/local/lib64 && \
     make -j"$(nproc)" install-strip && \
     find /usr/local/lib64 -type f -name 'lib*.la' -delete

RUN yum install --nogpgcheck -y \
        glibc-devel.i686 \
        libgcc.i686 \
        libXext-devel.i686 \
        libX11-devel.i686 && \
    rm -rf /var/cache/yum/*

# 32-bit libraries
```

```
  RUN make distclean && \
      ./autogen.sh && \
      ./configure --prefix=/usr/local --libdir=/usr/local/lib --
host=i386-linux-gnu "CFLAGS=-m32" "CXXFLAGS=-m32" "LDFLAGS=-m32" && \
      make -j"$(nproc)" install-strip && \
      find /usr/local/lib -type f -name 'lib*.la' -delete

  FROM ${from}
  LABEL maintainer "NVIDIA CORPORATION <cudatools@NVIDIA.com>"

  COPY --from=glvnd /usr/local/lib64 /usr/local/lib64
  COPY --from=glvnd /usr/local/lib /usr/local/lib

  COPY 10_NVIDIA.json /usr/local/share/glvnd/egl_vendor.d/10_
NVIDIA.json

  RUN echo '/usr/local/lib64' >> /etc/ld.so.conf.d/glvnd.conf && \
      echo '/usr/local/lib' >> /etc/ld.so.conf.d/glvnd.conf && \
      ldconfig && \
      echo '/usr/local/$LIB/libGL.so.1' >> /etc/ld.so.preload && \
      echo '/usr/local/$LIB/libEGL.so.1' >> /etc/ld.so.preload
```

Save this Dockerfile as libglvnd-centos8.dockerfile.

The only change we made compared to the file that NIVIDIA provides for centos7 is on line 4, from:

```
  FROM centos:7 as glvnd
```

to:

```
  FROM centos:8 as glvnd
```

The instructions for building this docker container clones the libGL repo from git (https://github.com/NVIDIA/libglvnd) and builds the libraries from scratch. Docker then creates a combined base container that includes everything from our centos8-NVIDIA container and from the container above. The process of building this container is similar to the previous container you built:

```
  docker build --pull -t "centos8-NVIDIA-libglvnd-1.1" \
     --build-arg "from=centos8-NVIDIA" \
     --build-arg "LIBGLVND_VERSION=v1.1.1" \
     -f libglvnd-centos8.dockerfile .
```

Pass through two arguments to the Docker build daemon: the name of the base image you want to use and the LIBGLVND version to use.

3.2.3 Building the Base Container for MAK ONE Products

The last Dockerfile you need to create installs all of the dependencies required to run both VR-Vantage and VR-Forces applications.

```
FROM centos8-NVIDIA-libglvnd-1.1

ENV NVIDIA_DRIVER_CAPABILITIES ${NVIDIA_DRIVER_CAPABILITIES},display

RUN yum install -y --nogpgcheck \
   mesa-libGLU \
   harfbuzz \
   fontconfig \
   jasper-libs \
   libXcursor \
   libxcb \
   libXext \
   libXi \
   libXinerama \
   libXrandr \
   libXxf86vm \
   libgomp \
   libpng \
   libtiff \
   libxkbcommon \
   pulseaudio\
   unixODBC \
   && \
   rm -rf /var/cache/yum/*

CMD while true; do sleep 1000; done
```

The CMD section of this Dockerfile forces Docker to always run the container until stopped. This allow you to run applications and have the Docker container not close. You can now save this file as makBaseImage.dockerfile and build it.

```
docker build -t "mak-base-centos8-libgl " -f makBaseImage.dockerfile
.
```

3.3. Running MAK ONE Applications

Now that you have built the three containers, use our mak-bae-centos8-libgl container to run either VR-Forces or VR-Vantage applications. In the example, you first install and configure our VR-Vantage container to run vrvStealth, then go through VR-Forces to run vrfLauncher and vrfGui.

3.3.1 VR-Vantage

Installing VR-Vantage

If you have been reading through each Dockerfile, you might have noticed that they do not include installing or configuring VR-Vantage to run. Here we have a caveat: Docker limits the maximum size of a container. To get around this, install VR-Vantage and the data to a folder on the host machine and mount the directory to the docker

container. Create a folder, extract the VR-Vantage package and the data packages as you would install any other MAK ONE product. If you have a license file, you can put it in the same directory as your VR-Vantage install.

For this example, we use *./home/MAK/install/* as our base directory. The directory contains our VR-Vantage install with data and our license file. We access this directory by creating a volume mount within the container to the directory on our host machine.

Starting VR-Vantage

Before starting the container, give xhost permission to allow access to the desktop.

```
xhost +si:localuser:root
```

Next, start the container for VR-Vantage.

```
docker run -it \
     --runtime=NVIDIA \
     -v /home/MAK/install:/MAK/install \
     -v /tmp/.X11-unix:/tmp/.X11-unix \
     -v /run/user/1000/pulse:/run/user/1000/pulse \
     -e PULSE_SERVER=unix:/run/user/1000/pulse/native \
     -e XDG_RUNTIME_DIR=$XDG_RUNTIME_DIR \
     -e MAKLMGRD_LICENSE_FILE=/MAK/install/lic/DEMO-NOV-2021.lic \
     -e DISPLAY=unix$DISPLAY \
     -d \
     --name vrvantage \
     mak-base-centos8-libgl:latest
```

Table 2 explains the arguments in this run command.

Table 2. Arguments in the run command

Run command	Description
`--runtime=NVIDIA \`	We want to tell docker to use the NVIDIA runtime environment for our container. This allows NVIDIA to forward any graphics that are in the container to our desktop.
`-v /home/MAK/install:/MAK/install \`	Stated above, we need to give Docker access to our install of VR-Vantage. The section before the colon of the -v flag is the path to use on the host machine and the section after the colon is the path that Docker will have access to.

Table 2. Arguments in the run command (continued)

Run command	Description
`-v /tmp/.X11-unix:/tmp/.X11-unix \`	This forwards the desktop to our Docker container.
`-v /run/user/1000/pulse:/run/user/1000/pulse \`	Forward the Pulse audio server to the Docker container.
`-e XDG_RUNTIME_DIR=$XDG_RUNTIME_DIR \`	The XDG_RUNTIME_DIR is a required environment variable for running graphical applications.
`-e PULSE_SERVER=unix:/run/user/1000/pulse/native \`	Used for forwarding pulse audio through the docker container to the default system device.
`-e MAKLMGRD_LICENSE_FILE=/MAK/install/lic/DEMO-NOV-2021.lic \`	Set this to forward the license to vrvStealth in the container. If you are using a license server, you can set this to the server name of your license server.
`-e DISPLAY=unix$DISPLAY \`	Also required for forwarding the desktop to our Docker container.
`-d \`	Run the container in a detached state.
`--name vrvantage \`	Name the container for easy access.
`mak-base-centos8-libgl:latest`	The name of container image to use.

Running vrvStealth

After starting the VR-Vantage container, you can run vrvStealth. There are two ways you can start vrvStealth: opening a shell in the container or passing the run command through the docker exec command.

➤ **To run vrvStealth through a shell:**

1. Open an interactive bash shell in our docker container.

   ```
   docker exec -it vrvantage bash
   ```

2. Navigate to the bin64 directory.

   ```
   cd /MAK/install/vrvantage#/bin64
   ```

   where # is the version number for VR-Vantage.

3. Run vrvStealth.

```
./vrvStealth
```

You can also choose to run vrvStealth without starting a shell and then logging in to the container.

➤ To run vrvStealth through the docker exec command, enter:

```
docker exec vrvantage bash -c 'cd /MAK/install/vrvantage#/bin64/;
./vrvStealth'
```

Depending on the name of your VR-Vantage install directory, you may need to change the folder path accordingly. (This command almost mirrors the commands in the procedure for running vrvStealth through a shell.)

3.3.2 VR-Forces

The process for running a container for VR-Forces applications is similar, with only a few changes. There are a couple of additions to the run command and we run two containers, one for the vrfSim engine and one for the vrfGui.

Running the Container for vrfSim

Before starting the container, give xhost permission to allow access to your desktop.

```
xhost +si:localuser:root
```

Next, start the container for vrfSim. See Table 2 on page 35 for an description of each argument.

```
docker run -it \
   --runtime=NVIDIA \
   --net=host \
   -v /home/MAK/install:/MAK/install \
   -v /tmp/.X11-unix:/tmp/.X11-unix \
   -v /run/user/1000/pulse:/run/user/1000/pulse \
   -e PULSE_SERVER=unix:/run/user/1000/pulse/native \
   -e XDG_RUNTIME_DIR=$XDG_RUNTIME_DIR \
   -e MAKLMGRD_LICENSE_FILE=/MAK/install/lic/DEMO-NOV-2021.lic \
   -e DISPLAY=unix$DISPLAY \
   -d \
   --name vrforces-sim \
   mak-base-centos8-libgl:latest
```

The difference between the Docker run command for VR-Vantage and the one for VR-Forces sim engine is the addition of the `--net=host` argument. This allows Docker to directly interface with the network card.

Starting the VR-Forces Simulation Engine

For this example, run the simulation engine (sim) with the DIS protocol. We can start up the vrfLauncher with the `-B` argument to run the backend only.

As with VR-Vantage, you can run the launcher in one of two ways. Running the vrfLauncher through the docker exec command does not display anything to the console when the sim starts. If you want to see the messages from the sim, run vrfLauncher from the shell.

➤ To run vrfLauncher through a shell, open an interactive bash shell in the docker container, navigate to the bin64 directory, then run the vrfLauncher:

```
docker exec -it vrforces-sim bash
cd /MAK/install/vrforces#/bin64
./vrfLauncher -B
```

where # is the version number for VR-Forces.

➤ To run vrfLauncher through the docker exec command:

```
docker exec -it vrforces-sim bash -c 'cd
/MAK/install/vrforces#/bin64/; ./vrfLauncher -B'
```

Starting the VR-Forces GUI

Once the sim has started successfully, start a container for the vrfGui and run it.

```
docker run -it \
   --runtime=NVIDIA \
   --net=host \
   -v /home/tpc-rh7/mak:/MAK/install \
   -v /tmp/.X11-unix:/tmp/.X11-unix \
   -v /run/user/1000/pulse:/run/user/1000/pulse \
   -e PULSE_SERVER=unix:/run/user/1000/pulse/native \
   -e XDG_RUNTIME_DIR=$XDG_RUNTIME_DIR \
   -e MAKLMGRD_LICENSE_FILE=/MAK/install/lic/DEMO-NOV-2021.lic \
   -e DISPLAY=unix$DISPLAY \
   -d \
   --name vrforces-gui \
   mak-base-centos8-libgl:latest
```

As with the sim, you must pass the `--net=host` argument.

Starting the VR-Forces GUI

You can choose whether to start the GUI from shell or the docker exec command.

➤ To run the GUI through the shell:

```
docker exec -it vrforces-gui bash
cd /MAK/install/vrforces#/bin64
./vrfLauncher -F
```

➤ To run the GUI through the docker exec command:

```
docker exec -it vrforces-gui bash -c 'cd
/MAK/install/vrforces#/bin64/; ./vrfLauncher -F'
```

3.4. Conclusion

Congratulations! You have now learned to containerize and run VR-Forces and VR-Vantage. If you have any questions or comments, please contact MAK support: https://jira.mak.com/servicedesk/customer/portal/1.

MAK-0-5-221115