p +61 1300 113 112
e info@hyperscalers.com

Solving Information Technology's **Complexity**

TECHNICAL WHITEPAPER

Virtual GPU Appliance

REFERENCE ARCHITECTURE

The vGPU appliance by Hyperscalers enables system administrators, data scientist and developers with everything they need to run both 3D graphics applications and compute workloads in a virtualized GPU environment 'out of the box.'

July 2021

Contents

Project Overview:

Enterprise size organizations around the world have a growing need for GPU acceleration. This demand comes from data scientists and developers who are asking their system administrators to provide them with GPU capable environments. However, when these GPU environments are delivered in the physical bare metal sense as part of either workstations or in servers, they typically sit in silo-like environments which become poorly utilized. A survey of enterprises has shown that GPU are utilised only 25-30% of the time.

With the intent to solve the problem of underutilisation without sacrificing performance, Hyperscalers together with Red Hat and NVIDIA have teamed to build an appliance that enables systems administrators to provide the data scientist and developers with everything they need to run both 3D graphics applications and compute workloads in a virtualized environment 'out of the box'.

This reference architecture by Hyperscalers is a step-by-step guide to deploying virtual GPU using Red Hat Enterprise Linux (RHEL) and NVIDIA vGPU on S5BV GPU server.

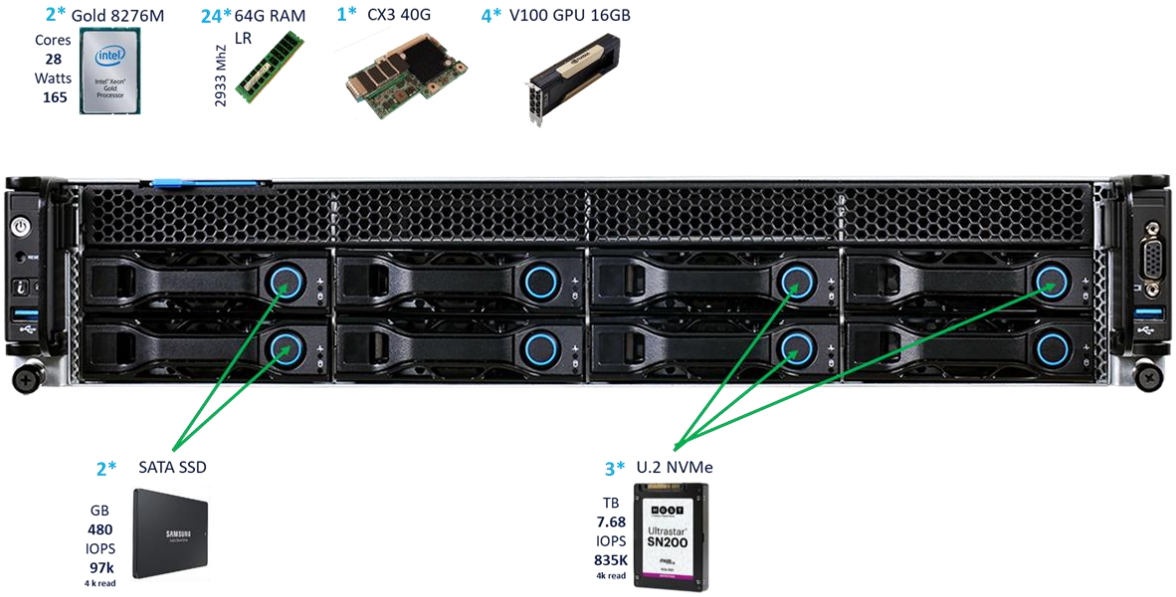
Key Deliverables: GPU compute and graphics capability shared by VMs (Virtual Machines)

Tools used:

- IP Appliance Design Process document Click here
- Appliance Optimizer Utility Click here
- Anydesk – Remote desktop with recording tool and player
- Cuda-Z
- GeekBench
- Regression Testing Tool
- Shell script tool

Hardware Specification:

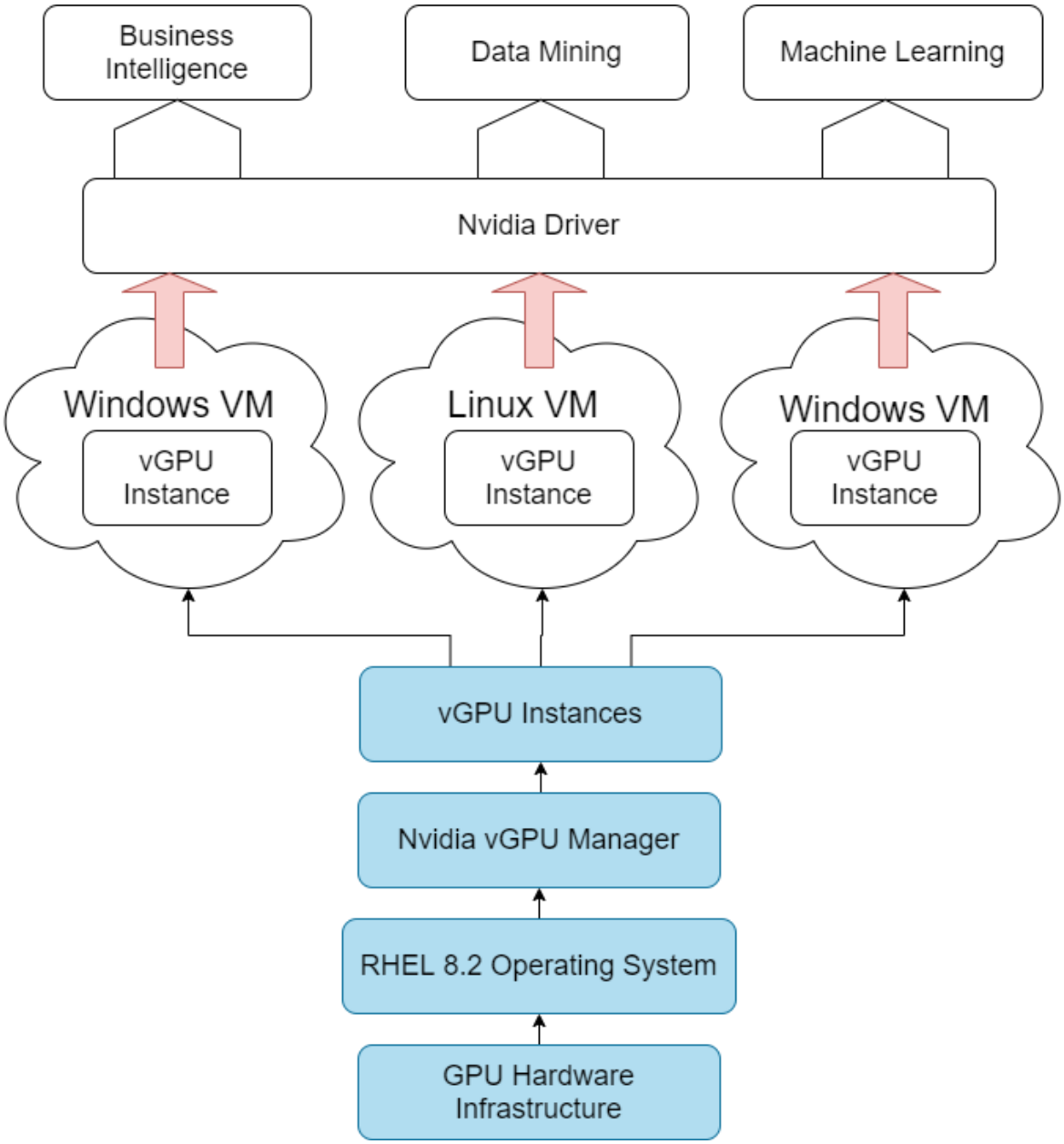
- Networking – 1U 100G Switch (BMS T7032-IX1)
- Server – D52BV-2U (S5BV)



  - 2x Platinum 8276M Processor 28c 2.20 - 4.00 GHz 38.5 MB 165W DDR4 2933
  - 24x DDR4_LR 23400(2933MHz) 64GB Register Samsung M386A8K40CM2-CVF
  - 2x SSD - 2.5" SATA 6Gb\s 480GB Samsung PM883 HSMZ7LH480HAHQ (480GB DWPD = 1.3)
  - 3x 2.5" U.2 NVMe SSD HGST ULTRASTAR SN200 7.6TB HUSMR7676BDP3Y1 0TS1357 (7.6TB DWPD = 1)
  - 1x OCP Mezz 40Gb QSFP+ 2 Port Quanta on 40Gbe CX3 ConnectX 3364ELB0000 2 PORT 40G QSFP+
  - 4x Nvidia Tesla V100 GV100 250W Gen 3 PCI-E X16 32GB FHFL Dual Slot ADLB2G50015 900-2G500-0010-000
- System management - IPMI v2.0 Compliant, on board "KVM over IP" support
- Operating environment –
  - Operating temperature: 5°C to 35°C (41°F to 95°F)
  - Non-operating temperature: -40°C to 65°C (-40°F to 149°F)
  - Operating relative humidity: 50% to 85%RH.
  - Non-operating relative humidity: 20% to 90%RH
- Video - Integrated Aspeed AST2400 with 8MB DDR3 video memory

Architecture:

A bottom-up approach is followed to build the virtual GPU appliance and it is seen as below:



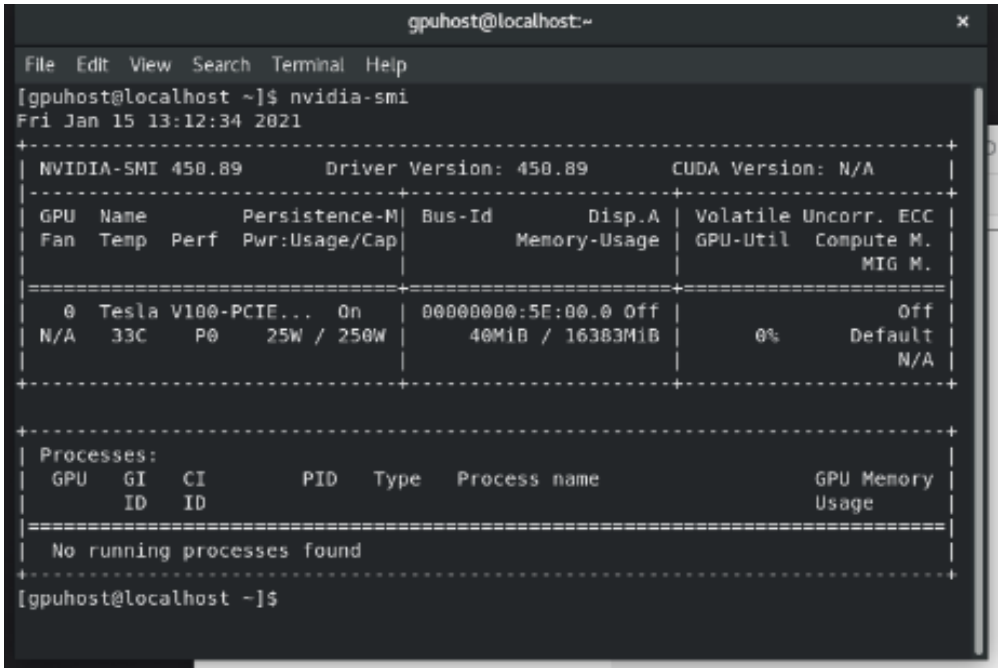
Operating System and pre-requisite:

1. In BIOS, Navigate to Advance -> PCI subsystem settings.
2. Enable VT-D or IOMMU option in BIOS.
3. Enable SR-IOV options in the BIOS.
4. Install RHEL 8.2 with "Workstation" capability.
5. Install GCC with the terminal command "sudo dnf install gcc"
6. Go to the file destination "/etc/default/grub" and open terminal at the location. Type the command "sudo gedit grub" and enter. Modify the grub file variable GRUB_CMBLINE_LINUX by adding "modprobe.blacklist=nouveau" to the end.

7. Rebuild the kernel with the command "sudo grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg" and the reboot.
8. Install the libraries by the terminal command "sudo yum install gcc make kernel-headers kernel -devel acpid libglvnd-glx libglvnd-opengl libglvnd-devel pkgconfig" and reboot.
9. backup the initramfs with the terminal command "sudo mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r)-nouveau.img
10. Build the kernel and reset with the terminal command "sudo dracut /boot/initramfs-$(uname -r).img $(uname -r)" and reboot.
11. Build the kernel and reset with the terminal command "sudo dracut /boot/initramfs-$(uname -r).img $(uname -r)" and reboot.

vGPU manager:

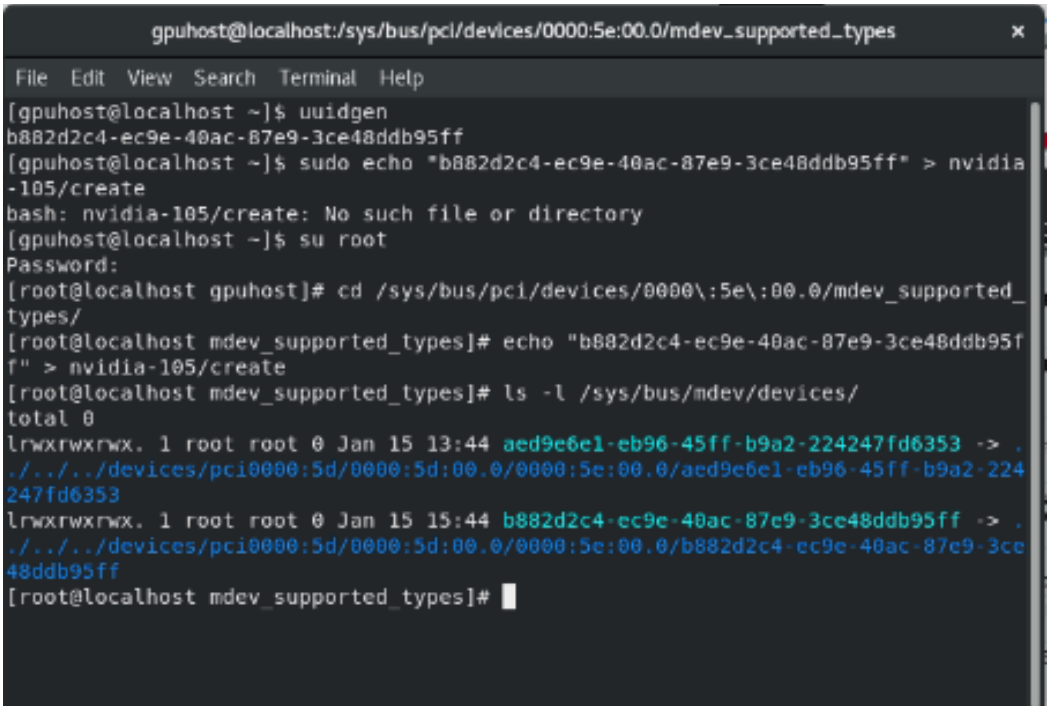
1. Exit the X server with the terminal command "sudo init 3" and login with your credentials.
2. Navigate to the directory with the VGPU rpm file and enter the command "sudo rpm -iv (installation file).rpm".
3. Reboot the system.
4. Run "nvidia-smi" on the terminal to validate the vGPU manager.



vGPU instances:

1. Verify the vfio modules are loaded with the command "lsmod | grep vfio"
2. Verify the libvirtd service is active and running with the command "service libvirtd status"
3. Retrieve and note the PCI bus details using the command "lspci | grep NVIDIA"
4. Install virsh using the command "libvirt-client" and get the full PCI identifier using the command "virsh nodedev-list –cap pci|grep <PCI bus details identified above>"

5. Obtain the bus and domain details with the command "virsh nodedev-dumpxml <full bus identifier> | egrep 'domain|bus|slot|function'"
6. Get a 'Q' series name and type of GPU instance based on the instances required and the graphics and compute capability required.
7. Enable sriov-manage on the PCI bus with the command "/usr/lib/nvidia/sriov-manage -e 00:5e:0000.0" the specific pci identifier which we already found.
8. Navigate to the directory of mdev using the command "cd /sys/bus/pci/devices/0000\:5e\:00.0/mdev_supported_types/" The bus and domain may change.
9. Enter the command "grep -l "<vgpu type>" <vgpu type id>/ and find a folder structure as "<vgpu-type-id>/name"
10. To get the info on the gpu manager and available instances, use the command "cat nvidia-105/name" and "cat nvidia-105/available_instances"
11. Generate an uuid using the command "uuidgen".
12. Use the terminal command "echo "<uuid>"> subdirectory/create" to add the instance to the gpu folder.
13. To confirm a vgpu instance is created, enter the command "ls -l /sys/bus/mdev/devices/" will list the vGPU's created.



Virtual Machines:

1. Go to the Box application on RHEL.
2. Create new virtual machine with the iso file of the OS image.
3. Load the OS machine and start the VM.

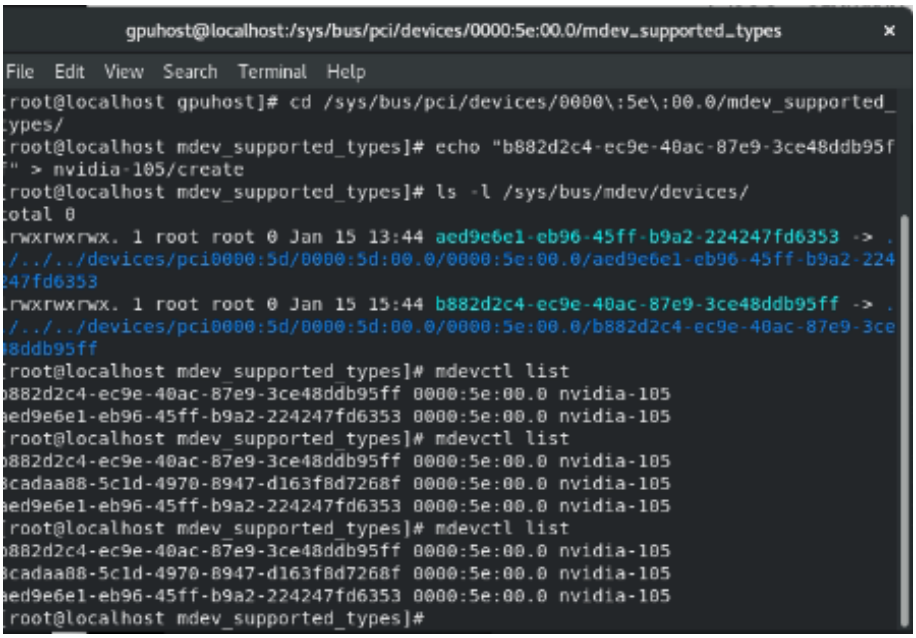
Assigning the vGPU instance to a VM:

1. Pre-requisite:
    a. Turn Wayland off

   b. Disable Nouveau
   c. Install GCC
   d. Disable ECC
2. Open the xml file of the VM using the command "virsh edit vm-name".
3. Add the vgpu instance to the VM devices list by adding the following lines:

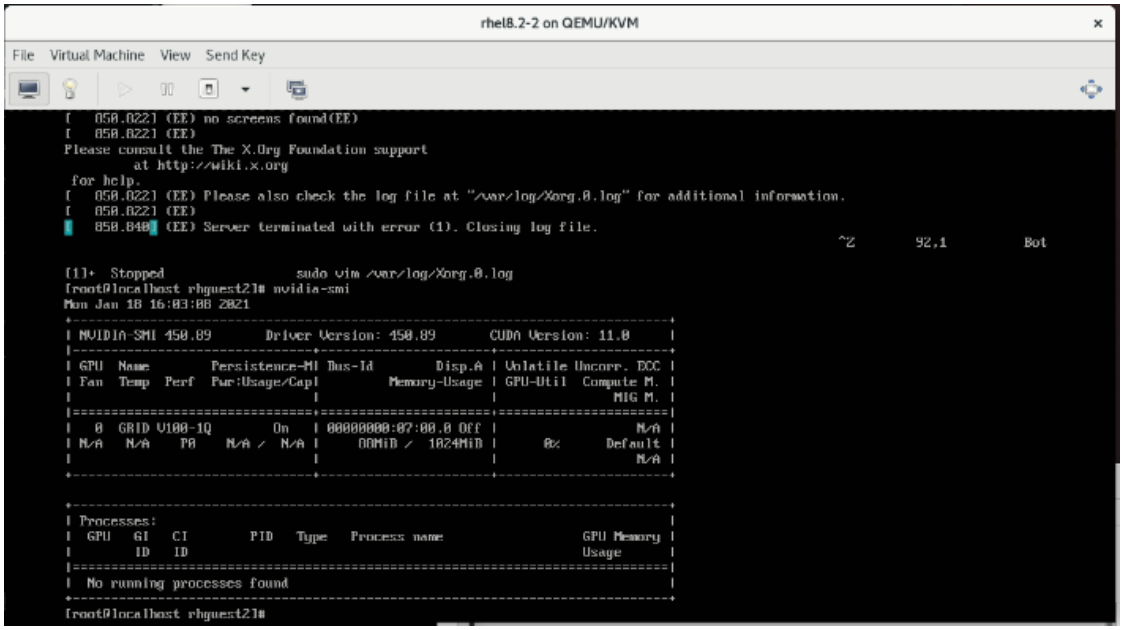
```
<hostdev mode='subsystem' type='mdev' model='vfio-pci'>
  <source>
    <address uuid='uuid'/>
  </source>
</hostdev>
```

4. Add multiple vgpu's by adding more hostdev with uuid of the vGPU instances.
5. Start the VM using the command "virsh start vm-name".



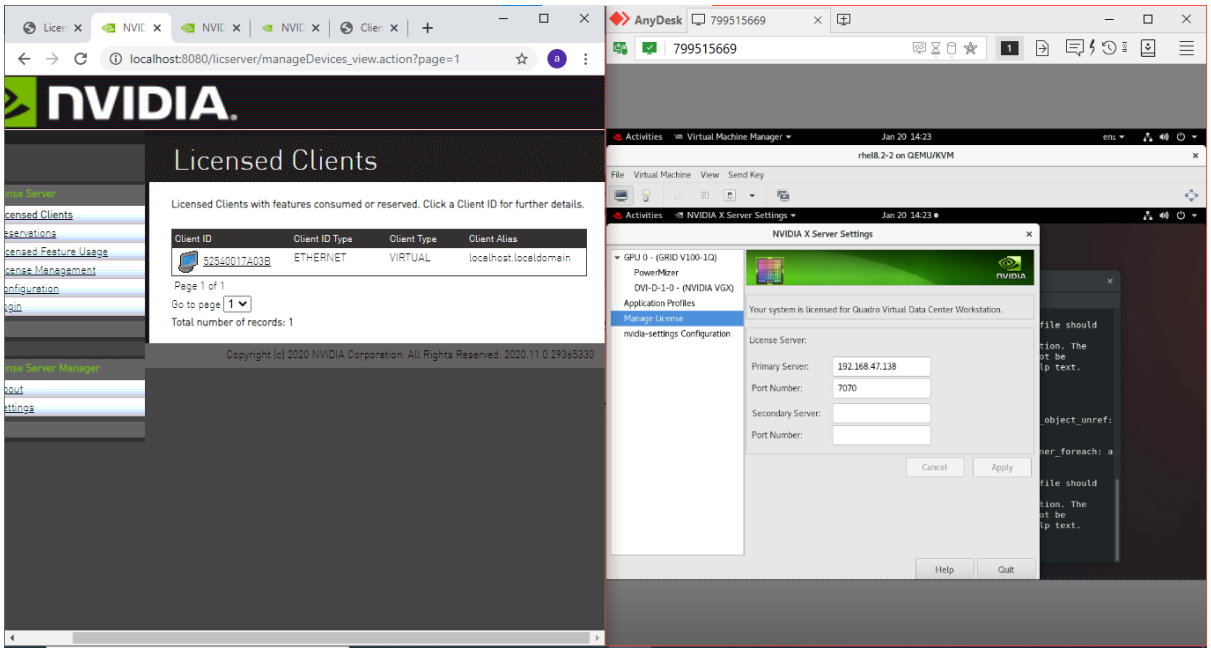
GPU driver on VM:

1. Download the linux driver package from nvidia.
2. Stop the display driver with the command "sudo init 3".
3. Run the installer with the command "sudo sh ./the installation file.run"
4. Select confirm to install the x configuration file.
5. Install the dependency packages and update the drivers and upgrade the system.
6. If any error in the install, see the nvidia log file and install the required libraries. Specifically, "sudo dnf install elfutils-libelf-devel".
7. Then re-run the driver installation.
8. Reboot the system to use the vGPU in the VM.



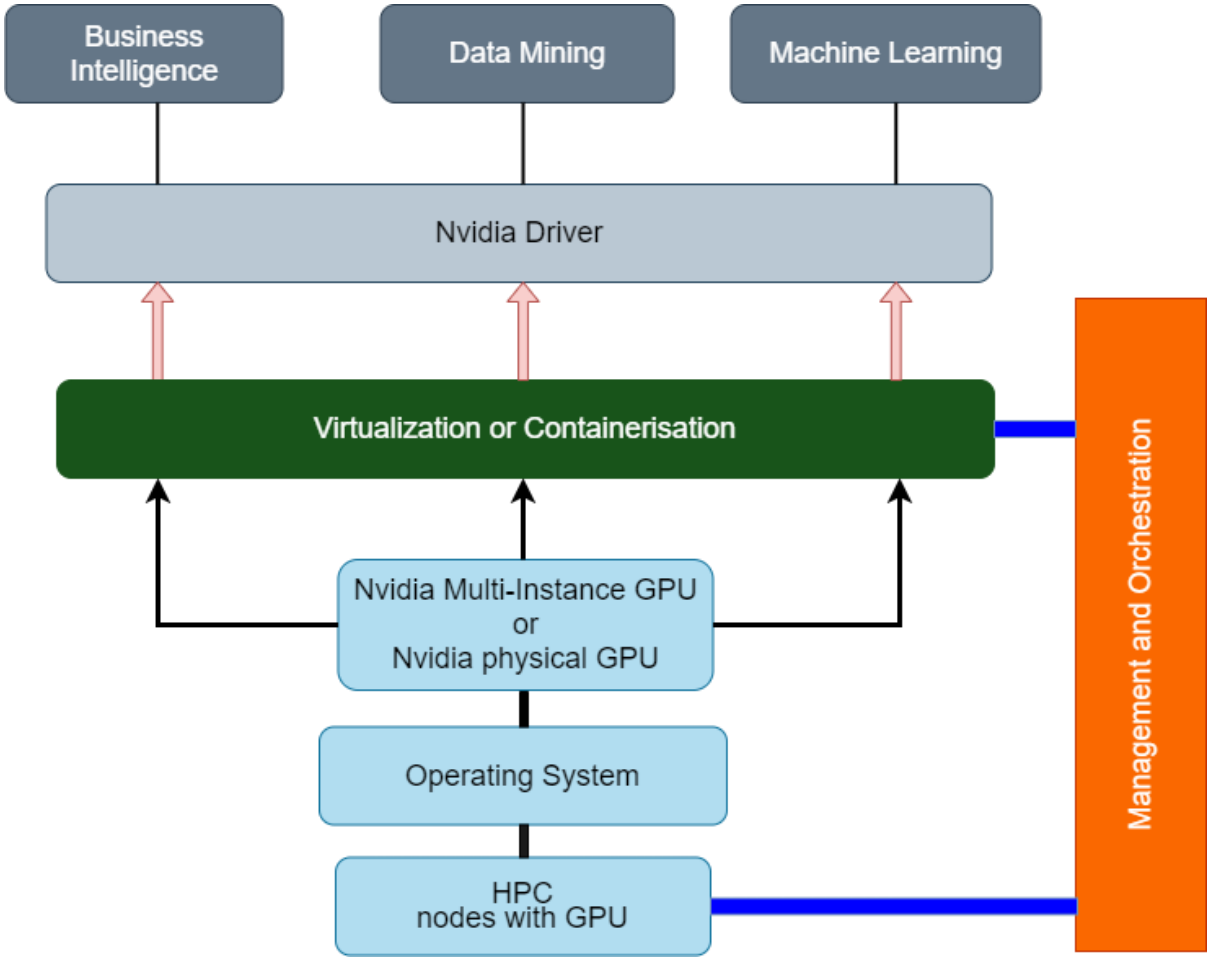
License Server:

To manage the subscription to the vGPU manager software, we need to setup a license server and host the application MAC address. Follow the grid-license-server-user-guide in the link.



Datacentre with Multi-GPU machines:

The future scope to the project is in adding the orchestration to the vGPU resources centralised to be integrated with the cluster management which may be VM clusters or containers. The architecture proposed on the datacentre level can be visualised from the architecture diagram below:



External Stakeholders

NVIDIA:
Michael L

Red Hat:
John N

Internal Staff

Hyperscalers:
Primary: Aaromal G
Secondary: Petar D