Talk Nerdy to Me, Using Python to Create VMs with vGPUs for AI Workloads

VMWARE {code}

Tony Foster

Principal Technical Marketing Engineer

Dell Technologies – Integrated Solutions Group

CODE2778

#vmworld #CODE2778

vmware

©2021 VMware, Inc.

Required Disclaimer for All Presentations

- This presentation may contain product features or functionality that are currently under development.
- This overview of new technology represents no commitment from VMware to deliver these features in any generally available product.
- Features are subject to change, and must not be included in contracts, purchase orders, or sales agreements of any kind.
- Technical feasibility and market demand will affect final delivery.
- Pricing and packaging for any new features/functionality/technology discussed or presented, have not been determined.

At a glance

Overview of vGPUs

Requirements

The MOB

Basic Operations

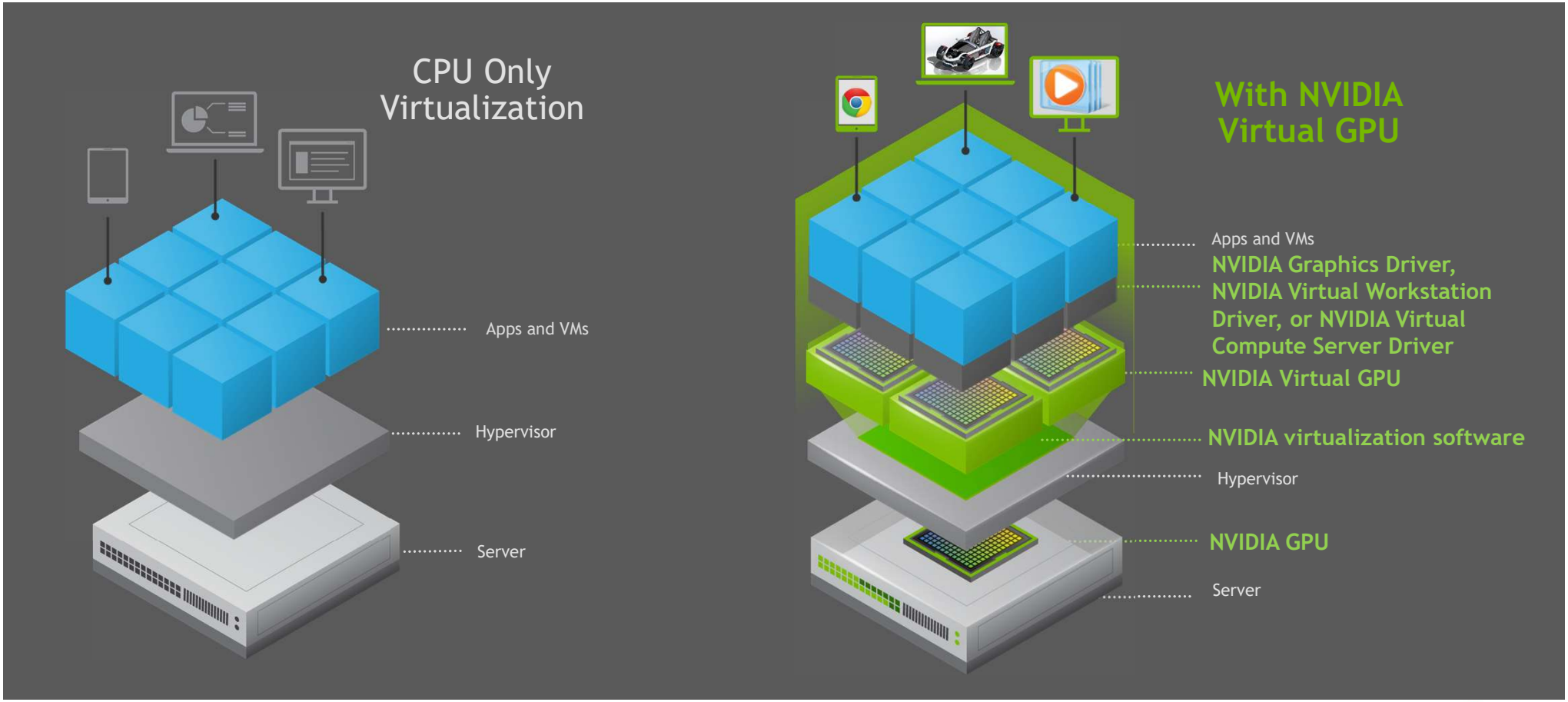
- Getting GPUs per host
- Getting vGPU profiles available
- Getting VMs with vGPUs
- Add a vGPU to a VM
- Remove a vGPU

Where Next?

Find it on GitHub

Resources

NVIDIA vGPU


CPU Only Virtualization
Apps and VMs
Hypervisor
Server
With NVIDIA Virtual GPU
Apps and VMs
NVIDIA Graphics Driver, NVIDIA Virtual Workstation Driver, or NVIDIA Virtual Compute Server Driver
NVIDIA Virtual GPU
NVIDIA virtualization software
Hypervisor
NVIDIA GPU
Server

Requirements

vSphere environment

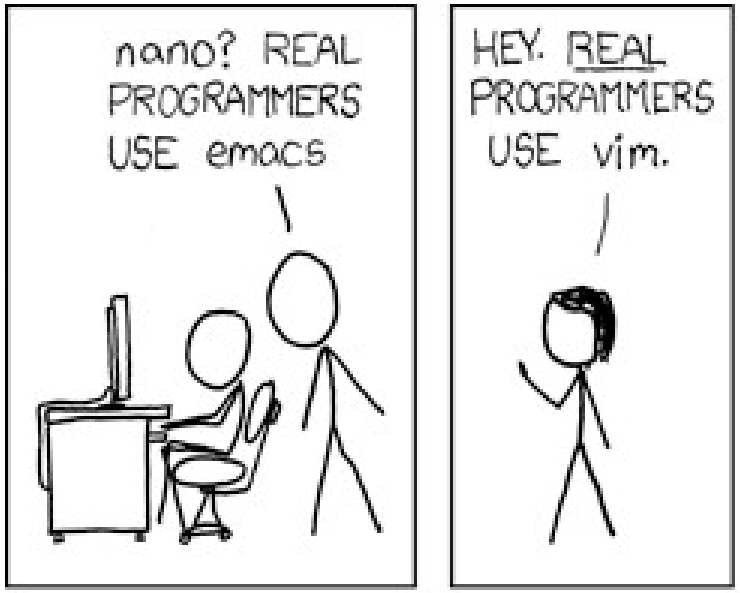
- At least one ESXi host with a supported GPU
- Managed Object Browser (MOB) *https://[vcenter]/mob*

Python

Modules

- pyVmomi
- pyVim

Text editor (Nano, Vim, emacs, etc) or IDE



https://xkcd.com/378

Managed Object Browser

To get to hosts:

- content > rootFolder > childEntity[] > hostFolder > childEntity[]
- Supports object types:
  - "vim.Folder"
  - "vim.ComputeResource"
- Child Entity object reference type: HostSystem[]

GPU details:

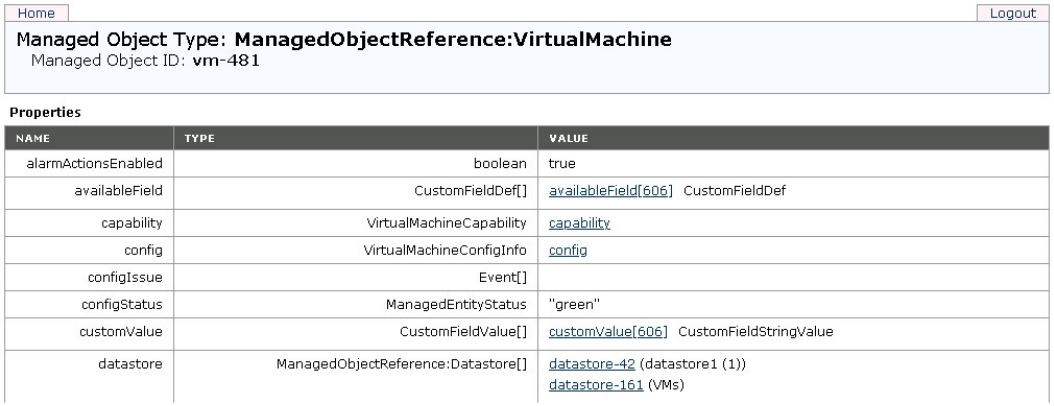
- *<<above path>>* > host > config >
  - sharedPassthruGpuTypes[]
  - sharedGpuCapabilities (supported vGPU attributes)

To get to VMs:

- content > rootFolder > childEntity[] > vmFolder > childEntity[]
- Supports object types:
  - "vim.Folder"
  - "vim.VirtualMachine"
  - "vim.VirtualApp“
- Child entity object reference type: ManagedEntity[]

vGPU details:

- *<<above path>>* > config > hardware > device[]
  - VirtualPCIPassthrough
    - backing
    - deviceInfo



Lab Time

- Establishing a connection
- Getting GPUs in a host
- Getting vGPU profiles available
- Getting VMs with vGPUs
- Add a vGPU to a VM
- Remove a vGPU

Establishing a connection

Standard connection script

Setup to handle self signed certificates

Additional code inserted starting at line 28

vCenter connection is “si”

If it is unable to connect it prints a message

```
01: from __future__ import print_function
02: from pyVim.connect import SmartConnect, Disconnect
03: from pyVmomi import vim
04: from pyVmomi import vmodl
05:
06: import argparse
07: import atexit
08: import getpass
09: import ssl
10: from pyVim.task import WaitForTask
11: def main():
12:     context = None
13:     if hasattr(ssl, '_create_unverified_context'):
14:         context = ssl._create_unverified_context()
15:     si = SmartConnect(host="vsa01.wondernerd.local",
16:                       user="UserName",
17:                       pwd="PassWord",
18:                       port=443,
19:                       sslContext=context)
20:     if not si:
21:         print("Could not connect to the specified host using specified "
22:               "username and password")
23:         return -1
24:
25:     atexit.register(Disconnect, si)
26:
27:     HostContent=si.content
28:     ###### Code Modules Here #####
29:
30:     ###################################
31:     # end of code modules
32:     return 0
33: if __name__ == "__main__":
34:    main()
```

GPUs in a Host

Code snipit starting at line 29 of the establishing a connection slide

Creates a container view

Iterates through the hosts

Check for a valid config

Checks for a valid array of sharedPassthruGpuTypes using the *config.graphicsInfo*

For each GPU in the array print its config.deviceName

```
01: ###### Code Modules Here #####
02:
03: HostContent=si.content
04: TempHold = HostContent.viewManager.CreateContainerView(
05:     HostContent.rootFolder,[vim.HostSystem], True)
06: for managed_object_ref in TempHold.view:
07:     print(managed_object_ref.name)
08:     try:
09:         if isinstance(managed_object_ref.config, NoneType) == False:
10:             if managed_object_ref.config.graphicsInfo != []:
11:                 for GPU_Config in managed_object_ref.config.graphicsInfo:
12:                     print(GPU_Config.deviceName)
13:             else:
14:                 print("No GPUs found")
15:         else:
16:             print("Host powered off")
17:     except:
18:         print("Error retrieving information")
19:
20: ###################################
21: # end of code modules
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
```

vGPU Profiles Available

Code snipit starting at line 29 of the establishing a connection slide

Almost same code as GPUs in a Host

- Instead of *config.graphicsInfo* it uses *config.SharedPassthruGpuTypes* which has each GPU profile

```
01: ###### Code Modules Here #####
02:
03: HostContent=si.content
04: TempHold = HostContent.viewManager.CreateContainerView(
05:     HostContent.rootFolder,[vim.HostSystem], True)
06: for managed_object_ref in TempHold.view:
07:     print(managed_object_ref.name)
08:     try:
09:         if isinstance(managed_object_ref.config, NoneType) == False:
10:             if managed_object_ref.config.sharedPassthruGpuTypes != []:
11:                 for GPU_Profile in \
12:                     managed_object_ref.config.sharedPassthruGpuTypes:
13:
14:                     print(GPU_Profile)
15:             else:
16:                 print("No GPUs found")
17:         else:
18:             print("Host powered off")
19:     except:
20:         print("Error retrieving information")
21:
22: ###################################
23: # end of code modules
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
```

VMs with vGPUs

Code snipit starting at line 29 of the establishing a connection slide

This method navigates the folder structure instead of creating a view

Looks for VMs that are children of the DC

On each VM looks through the *config.hardware.device*

If the device is a *VirtualPCIPassthrough* device AND has a backing of "*vgpu*" then the VM has a vGPU associated with it

```
01: ###### Code Modules Here #####
02:
03: HostContent=si.content
04:
05: DataCenterContent = HostContent.rootFolder.childEntity[0] #Assume single DC
06: VMs = DataCenterContent.vmFolder.childEntity
07: for i in VMs:
08:     if isinstance(i,vim.Folder):
09:         #**************found a folder***************
10:         for ChildVM in i.childEntity:
11:
12:             # Does it have a vGPU
13:             for VMVirtDevice in ChildVM.config.hardware.device:
14:                 if isinstance(VMVirtDevice, vim.VirtualPCIPassthrough) and \
15:                     hasattr(VMVirtDevice.backing, "vgpu"):
16:
17:                     print("VM Name: "+ ChildVM.name)
18:                     print("In Folder: "+ ChildVM.parent.name)
19:                     print("Device Backing: " + VMVirtDevice.backing.vgpu)
20:                     print("Device Label: " + VMVirtDevice.deviceInfo.label)
21:                     print("Device Summary: " +
22:                         VMVirtDevice.deviceInfo.summary)
23:
24:                     print("************************************")
25:
26: ##################################
27: # end of code modules
28:
29:
30:
31:
32:
33:
34:
```

Add vGPU to a VM

Code snipit starting at line 29 of the establishing a connection slide

Creates a container view of *vim.VirtualMachine*

Iterates through the VMs

Finds the desired VM

Creates a *vim.vm.ConfigSpec*

- Adds a *vim.VirtualPCIPassthrough* device
- Sets the *deviceInfo.summary*
- Sets the *deviceInfo.label*
- Sets the *BackingInfo* to the correct vGPU profile (p4-4q)
  - This can also be done with the *backing.vgpu* line 25

The VM is then reconfigured with the *configSpec*

```
01: ###### Code Modules Here #####
02:
03: HostContent=si.content
04:
05: vm = None
06: TempVMlist = \
07: HostContent.viewManager.CreateContainerView(HostContent.rootFolder,\
08:         [vim.VirtualMachine], True)
09: for managed_VM_ref in TempVMlist.view: #Go thought VM list
10:     if managed_VM_ref.name == "Compute000": #find Desired VM
11:         print(managed_VM_ref)
12:         print(managed_VM_ref.name)
13:         vm = managed_VM_ref #Capture VM as an obj to use next
14: if vm != None: #Safety to make sure not added to null object
15:     cspec = vim.vm.ConfigSpec()
16:     cspec.deviceChange = [vim.VirtualDeviceConfigSpec()]
17:     cspec.deviceChange[0].operation = 'add'
18:     cspec.deviceChange[0].device = vim.VirtualPCIPassthrough()
19:     cspec.deviceChange[0].device.deviceInfo = vim.Description()
20:     cspec.deviceChange[0].device.deviceInfo.summary = 'NVIDIA GRID vGPU
21: grid_p4-4q'
22:     cspec.deviceChange[0].device.deviceInfo.label = 'New PCI device'
23:     cspec.deviceChange[0].device.backing = \
24:         vim.VirtualPCIPassthroughVmiopBackingInfo(vgpu='grid_p4-4q')
25:     #cspec.deviceChange[0].device.backing.vgpu =str('grid_p4-2q')
26:     WaitForTask(vm.Reconfigure(cspec))
27:
28: ################################
29: # end of code modules
30:
31:
32:
33:
34:
```

Remove vGPU From a VM

Code snipit starting at line 29 of the establishing a connection slide

Similar to adding a vGPU slide

Of note on line 16, it is necessary to test and make sure the VM has a vGPU to remove

The correct vGPU type is verified on line 20

A configSpec is defined to remove the vGPU

- This removal will remove the last added vGPU to the VM

```
01: ###### Code Modules Here #####
02:
03: HostContent=si.content
04:
05: vm = None
06: vGPUobj = None
07: TempVMlist = \
08:     HostContent.viewManager.CreateContainerView(HostContent.rootFolder,\
09:     [vim.VirtualMachine], True)
10: for managed_VM_ref in TempVMlist.view: #Go thought VM list
11:     if managed_VM_ref.name == "Compute000": #find Desired VM
12:         print(managed_VM_ref)
13:         print(managed_VM_ref.name)
14:         vm = managed_VM_ref #Capture VM as an obj to use next
15: if vm != None: #Safety to make sure not added to null object
16:     for VMVirtDevice in vm.config.hardware.device: #Go through vPCI find vGPU
17:         if isinstance(VMVirtDevice, vim.VirtualPCIPassthrough) and \
18:             hasattr(VMVirtDevice.backing, "vgpu"):
19:
20:             if VMVirtDevice.backing.vgpu == "grid_p4-4q":
21:                 vGPUobj = VMVirtDevice
22:                 print("Found vGPU: " + VMVirtDevice.backing.vgpu)
23:
24:     cspec = vim.vm.ConfigSpec()
25:     cspec.deviceChange = [vim.VirtualDeviceConfigSpec()]
26:     cspec.deviceChange[0].operation = 'remove'
27:     cspec.deviceChange[0].device = vGPUobj
28:     WaitForTask(vm.Reconfigure(cspec))
29:     print("Removed vGPU")
30:
31: ###################################
32: # end of code modules
33:
34:
```

Where Next?

Building a VM, read Alastair's (@demitasse) blog → https://demitasse.co.nz/2018/05/create-a-vm-with-pyvmomi/

- Use the pyVmomi Community Samples: https://github.com/vmware/pyvmomi-community-samples/tree/master/samples
- Use configSpec details to add vGPU in as part of the build

```
cspec = vim.vm.ConfigSpec()
    cspec.deviceChange = [vim.VirtualDeviceConfigSpec()]
    cspec.deviceChange[0].operation = 'add'
    cspec.deviceChange[0].device = vim.VirtualPCIPassthrough()
    cspec.deviceChange[0].device.deviceInfo = vim.Description()
    cspec.deviceChange[0].device.deviceInfo.summary = 'NVIDIA GRID vGPU grid_p4-4q'
    cspec.deviceChange[0].device.deviceInfo.label = 'New PCI device'
    cspec.deviceChange[0].device.backing = \
        vim.VirtualPCIPassthroughVmiopBackingInfo(vgpu='grid_p4-4q')
```

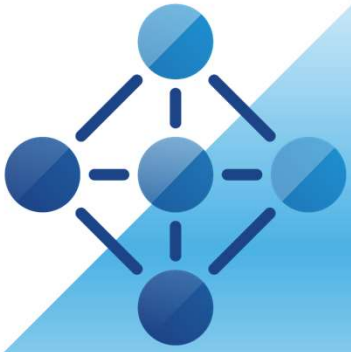
- Install an OS
- Install packages (SSH recommended unless using VDI)
- Install appropriate NVIDIA vGPU driver
- Install your AI/ML/DL package
  or for testing…
  Consider installing NVIDIA Container Toolkit https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker

Find it on GitHub

github.com/wondernerd/VMworld21

References

- VMware vHPC Tool-Kit
  https://github.com/vmware/vhpc-toolkit/
- First steps with Python and pyVmomi (vSphere SDK for Python)
  https://www.vcloudnine.de/first-steps-with-python-and-pyvmomi-vsphere-sdk-for-python/
- Adding vGPU using pyVmomi
  https://communities.vmware.com/t5/vSphere-Management-SDK/Adding-vGPU-using-pyVmomi/td-p/2730837
- New Sample: Addition of CPU Cores and Memory to the Existing machine #265
  https://github.com/vmware/pyvmomi-community-samples/issues/265



Resources

My blog: www.wondernerd.net

Get the code: www.github.com/wondernerd

Join the community http://code.vmware.com

Reach out:

Twitter @wonder_nerd

LinkedIn.com/in/wondernerd

vmware ©2021 VMware, Inc.

VMWARE {code}

Sessions you don’t want to miss!

[VMTN2835] Update to VDI by Day Compute by Night, Now with More vGPUs!

[EUS1289] VDI Nerdfest 2021: Demos That Make Admins Drool

[EUS3107] Nerd Tours: A Tech Deep Dive of the VDI NerdFest 2021 Extravaganza

[VI2222] Got GPUs? Learn How to Set Up Self-Service Access for AI/ML.

[VI1459] Best Practices for Running AI Workloads in VMs on VMware vSphere

[VI1559] vSphere Admin’s Guide to Virtual AI Infrastructure for Modern Data Science

Please take your survey.

vmware ©2021 VMware, Inc. #vmv

Thank you!

vmware ©2021 VMware, Inc. #vmv