Enabling VMware vSphere with Tanzu for Accelerated AI workloads with NVIDIA

Proof-of-Concept Deployment Guide

Justin Murray

Andrew Liu

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Authors | Description of Change |
| 0.7 | January 28, 2021 | J.Murray, A.Liu | Draft |

Table of Contents

[Document History 2](#_Toc62550943)

[Executive Summary 4](#_Toc62550944)

[Proof-of-Concept Overview 4](#_Toc62550945)

[Minimum Requirements 5](#_Toc62550946)

[Phase 1: vSphere Setup for Tanzu with NVIDIA virtual GPUs 5](#_Toc62550947)

[Setup the GPU host server with NVIDIA virtual GPU and MIG (MIG for A100 only) 5](#_Toc62550948)

[Create a Namespace and Tanzu Kubernetes Grid (TKG) cluster 6](#_Toc62550949)

[Creating an Ubuntu VM for the GPU-enabled node 7](#_Toc62550950)

[Enabling Access to a TKG Cluster Node via the Supervisor Cluster 9](#_Toc62550951)

[Accessing a TKG Cluster ControlPlane Node to generate a token, used to join the VM to the cluster 10](#_Toc62550952)

[Join the new Ubuntu VM to the TKG Cluster 10](#_Toc62550953)

[CNI Setup for manually created worker node 12](#_Toc62550954)

[Phase 2: Access the GPU-enabled Tanzu Kubernetes Grid cluster and deploy NVIDIA GPU Operator 13](#_Toc62550955)

[Build GPU Operator 1.5.1 with vGPU 12.0 Beta drivers 13](#_Toc62550956)

[Access the GPU-enabled TKG Cluster 13](#_Toc62550957)

[Deploy the NVIDIA GPU Operator with NVIDIA virtual GPU on the TKG cluster 14](#_Toc62550958)

[Phase 3: Deploy Sample AI and ML applications on the GPU-Enabled TKG Cluster 17](#_Toc62550959)

[Deploy the TF-Notebook Application 18](#_Toc62550960)

[Deploy the Intelligent Video Analytics (IVA) Application 18](#_Toc62550961)

[Appendix: 20](#_Toc62550962)

[Key Terms 20](#_Toc62550963)

[Related Materials 20](#_Toc62550964)

# Executive Summary

VMware announced a close partnership with NVIDIA at VMworld 2020 to bring the NVIDIA GPU Cloud (NGC) scaled AI/ML platform to vSphere 7 with Tanzu.  This collaboration is designed to bring the best accelerated computing experience to customers and to accelerate the adoption of ML/AI applications in the enterprise.

Tanzu Kubernetes Grid (TKG) is VMware’s enterprise-ready Kubernetes runtime that streamlines operations across multi-cloud infrastructure.  TKG provides platform operators with a consistent Kubernetes footprint on-premises and in the public cloud.  TKG is tightly integrated in the vSphere 7 with Tanzu product and is referred to as the “Tanzu Kubernetes Grid Service”. TKG can be deployed on vSphere 7.0 with a built-in management cluster.

As a part of this partnership, VMware will be developing native support within TKG to support NVIDIA virtual GPU with GPU Operator, on GPU-enabled host-systems. This development is scheduled to be released later this year, and as a result, this document provides the steps and guidance on how to set up an “interim solution” to manually enable TKG clusters for NVIDIA GPU support, and demonstrate how to deploy sample applications to perform AI training and inferencing on this cluster. This document also provides information on how to leverage the Multi-Instance-GPU capability supported by NVIDIA A100 GPUs, and how to create and manage MIG instances among multiple Kubernetes clusters with Tanzu.

# Proof-of-Concept Overview

**Phase 1:** vSphere Administrator steps

Setup Tanzu with vSphere 7.0 U2, set up NVIDIA virtual GPU with MIG (A100 only). Create a Tanzu Kubernetes Grid cluster and enable the cluster with a manually created node with NVIDIA Virtual Compute Server. Provide access to the Kubernetes/DevOps engineer using vSphere with Tanzu management capabilities.

**Phase 2:** Kubernetes/DevOps Engineer steps

Access the GPU-enabled TKG cluster provided by the vSphere admin. Build and install the GPU Operator for NVIDIA Virtual GPUs onto the Tanzu Kubernetes Grid cluster and check overall health of the GPU enabled cluster.

**Phase 3:** Data Scientist/Developer workload examples

Deploy two applications using Helm onto the GPU-enabled Tanzu Kubernetes Grid cluster and demonstrate their working functionality.

1. The TensorFlow/Jupyter Notebook is a good example test showcasing single node AI training and inference development capabilities.
2. The Intelligent Video Analytics (IVA) demo application from NVIDIA NGC showcases AI in action, performing real-time AI inferencing for IVA using NVIDIA’s DeepStream SDK and pre-built models.

**Phase 4 (Future):** Multi-node AI training on a GPU-enabled cluster with multiple GPU worker nodes connected by NVIDIA Mellanox networking

# Minimum Requirements

* VMware vSphere 7.0 U2 Beta with Tanzu installed and configured for Workload Management
* At least one NVIDIA A100 or T4 GPU in one NVIDIA-Certified server (NOTE: Three servers are recommended for vSphere with Tanzu. Only one server requires an NVIDIA GPU)
* NVIDIA GPU Operator version 1.5.1
* NVIDIA vGPU Host and Guest Driver Software 12.0 with Virtual Compute Server licenses
* NVIDIA Virtual GPU license server
* Access to the NVIDIA NGC Private Registry for this POC exercise
* Access to the Management network with administrator access to vSphere with Tanzu
* Ubuntu 20.04 Server ISO.

**Note:** Phase 4 (Future) will require additional hardware and software requirements, such as multiple servers, with NVIDIA Mellanox NICs and NVIDIA GPUs on the same root complex/PCIe switch on each system, connected to a 100/200G Mellanox switch.

# Phase 1: vSphere Setup for Tanzu with NVIDIA virtual GPUs

## Setup the GPU host server with NVIDIA virtual GPU and MIG (MIG for A100 only)

1. Start with Administrator access to a working vSphere setup where the Workload Management feature is enabled and at least one host contains NVIDIA GPUs. The vSphere with Tanzu Quickstart Guide provides details for setting up the Workload Management functionality in vSphere. The Guide is located at:

<https://core.vmware.com/resource/vsphere-tanzu-quick-start-guide-v1a#_Toc53677530>

This document also assumes that a TKG namespace and cluster are already created and running. For guidance on how to create and deploy a TKG cluster, please see the following document:

https://github.com/vsphere-tmm/vsphere-with-tanzu-quick-start

1. Choose a host server on which your GPU-enabled VM will run. This host does not need to have vSphere with Tanzu installed on it. Enable SSH to this host and open a session to it.
2. Check that the host server you plan to use has the latest NVIDIA host driver installed in it (a VIB or VMware Installable Bundle is the mechanism here). When logged in to the host server, enter the following command:

esxcli software vib list |grep -I NVIDIA

1. If the VIB is installed for the host driver, it should be at version 460.32.04
2. If the Host Driver is not installed, then download the VIB from the NVIDIA site and install it using the following procedure:

esxcli system maintenanceMode set --enable true

esxcli software acceptance set --level=CommunitySupported

esxcli software vib install --no-sig-check -v /tmp/NVIDIA\_bootbank\_NVIDIA- NVIDIA\_bootbank\_NVIDIA-VMware\_ESXi\_7.0\_Host\_Driver\_460.32.04-1OEM.700.0.0.15525992.vib

1. After installing the host driver, you will need to set the host graphics to Shared Direct mode. To do so, select the host in vCenter > Configure > Graphics, then edit each GPU and set the graphics mode to Shared Direct.
2. Now reboot the host, then exit maintenance mode.

esxcli system maintenanceMode set --enable false

1. Ensure that MIG is enabled for all of the A100s on the host server. You can see this at the ESXi host level by typing “nvidia-smi" and checking the MIG state on the output. If MIG is not enabled, then you can enable it using the instructions here:

<https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html>

Create a Namespace and Tanzu Kubernetes Grid (TKG) cluster

1. Once MIG is enabled, we can then begin the process of creating a Tanzu Kubernetes Grid (TKG) cluster. Start by following the instructions in the vSphere with Tanzu Quickstart Guide linked above to create a namespace and TKG cluster.
2. In the example given below, there are two separate K8s namespaces with two different TKG clusters in them. We will work here in just one of these TKG clusters and namespaces. We do that by specifying the name of the namespace and cluster in the “kubectl vsphere login” step to the TKG cluster – details of this are given below.

Notice also the separate “control plane” VMs for each TKG cluster that are within their respective namespaces. You can also see the SupervisorControlPlane VMs that run outside of the TKG clusters and that perform the management of resources for those TKG clusters.

Graphical user interface

Description automatically generated

Figure 1: vSphere client showing two namespaces with a TKG cluster in each one

Create an Ubuntu VM to use as a Jumper VM

1. We will need to create a Jumper VM to use with the vSphere Kubernetes CLI, and to build the driver container images for the GPU Operator.

Recommended VM configuration:

* + 4 vCPU
  + 16 GB RAM
  + 500 GB Thin Provisioned disk
  + VMXNet3 NIC connected to Management network
  + VMXNet3 NIC connected to Workload network
  + (If applicable) VMXNet3 NIC connected to Frontend network
  + Ubuntu Server 20.04 Server 64-bit
  + Configured for EFI boot
* Install the vSphere Kubernetes CLI tools from the control plane IP provided. We will use this Jumper VM to interface with the TKG cluster and add the manually created node below. The process for doing this is at:

https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-0F6E45C4-3CB1-4562-9370-686668519FCA.html

<https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-0F6E45C4-3CB1-4562-9370-686668519FCA.html>

## Create an Ubuntu VM for the GPU-enabled node

1. Create an Ubuntu VM such that it is bootable using EFI. This will appear in the VM Options -> Boot Options field in vSphere. NOTE: The EFI boot option needs to be chosen prior to the guest OS installation time. Changing the Boot Options for an already installed non-EFI VM will not suffice.

Recommended VM configuration:

* 16 vCPU
* 64 GB RAM
* 500 GB Thin Provisioned disk
* VMXNet3 NIC connected to Workload network
* NVIDIA vCS profile attached (A100-40C as an example)
* Ubuntu Server 20.04 Server 64-bit
* Configured for EFI boot

1. In the vSphere Client, set the VM Options – Advanced – Edit Configuration for two GPU-specific arguments.

* pciPassthru.use64BitMMIO = TRUE and
* pciPassthru.64bitMMIOSizeGB=128 (or 256 depending on the memory size of your GPU)
* For details on these settings see the blog article at: <https://blogs.vmware.com/apps/2018/09/using-gpus-with-virtual-machines-on-vsphere-part-2-vmdirectpath-i-o.html>

1. In the vSphere Client, choose the VM, use Edit Settings – Virtual Hardware – Add New Device and for PCI Device, select NVIDIA GRID vGPU and choose the appropriate vCS profile that you want this VM to have assigned to it.
2. Now power on the VM and install Ubuntu 20.04.
3. Give the VM a static IP address and supply it with the Gateway and DNS addresses so that it can reach the internet.
4. Setup NTP synchronization on the VM
5. Turn swapping off by running:

sudo swapoff -a

1. Next edit the “swap.img” line in */etc/fstab* to have a comment or # against the first swap.img entry. Example below

# /boot/efi was on /dev/sda1 during curtin installation

/dev/disk/by-uuid/476D-E9*94 /boot/efi vfat defaults 0 0*

#/swap.img none swap sw 0 0

1. Disable the firewall on Ubuntu using the command

sudo systemctl disable ufw

1. Install containerd and kubeadm onto this VM, so that it can participate in the TKG cluster.

* containerd: <https://kubernetes.io/docs/setup/production-environment/container-runtimes/>
* kubeadm : <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

1. In additiona, to ensure that containerd is running correctly along with the kubelet process, the KUBELET\_EXTRA\_ARGS variable on the kubelet command line (seen using a ps -eaf) should be set to the values as seen here:

KUBELET\_EXTRA\_ARGS="--container-runtime=remote --runtime-request-timeout=15m --container-runtime-endpoint=unix:///run/containerd/containerd.sock"

1. The line given above can be placed into a file named /etc/default/kubelet if required.
2. Following that file editing, the commands following should be used to restart the processes:

systemctl daemon-reload

systemctl restart containerd

systemctl restart kubelet

1. On your Ubuntu VM, run the command “sudo ctr version” This should produce output similar to the following:

Client:

Version: 1.3.3-0ubuntu2.2

Revision:

Server:

Version: 1.3.3-0ubuntu2.2

Revision:

UUID: 29d71fe6-8896-4da4-a3bf-bdde7807a72d

This ensures that your Ubuntu node is running correctly with the appropriate container runtime (CRI), that is containerd for Tanzu.

Enabling Access to a TKG Cluster Node via the Supervisor Cluster

1. We first login to a TKG cluster ControlPlane Node (VM), so that we can generate a token to allow your Ubuntu VM to “kubeadm join” into the TKG cluster and be recognized as a node within it.
2. In order to login to any TKG cluster node, we will need a password. To get that password we first login to the VMware vSphere **Supervisor** cluster via one of the **SupervisorControlPlane** VMs. We use the first one, whose IP address is 10.196.180.22 in the example. We could use any of the three SupervisorControlPlane VMs.
3. Login to the **Supervisor Cluster** from your Jumper VM using the “kubectl vsphere login” command below.

Use the administrator credentials to do this in a command similar to the following:

kubectl vsphere login –-vsphere-username [administrator@vsphere.local](mailto:administrator@vsphere.local) -–server 10.196.180.22 -–insecure-skip-tls-verify

**Note:** There are **two** minus signs before the arguments above

This login step is fully described at : <https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-F5114388-1838-4B3B-8A8D-4AE17F33526A.html>

1. Once successfully logged in to the Supervisor cluster, use the command below to set your context/namespace:

kubectl config use-context <example-context-name>

1. Next, to get the password to allow you to login with ssh into one the TKG cluster nodes, follow the steps at: <https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-37DC1DF2-119B-4E9E-8CA6-C194F39DDEDA.html>

Save the password that you got from this process in a secure location. You will use it shortly to login to a TKG cluster node.

1. Issue a kubectl vsphere logoutcommand to exit the Supervisor cluster, once you have saved the ssh password securely.

## Accessing a TKG Cluster ControlPlane Node to generate a token, used to join the VM to the cluster

1. Choose one of the TKG ControlPlane VMs for your TKG cluster. We chose the “endor-control-plane-q2nwv” in our example, that is the VM whose IP address is 10.196.186.20. Your details of VM name and IP address will differ for your own vSphere with Tanzu setup.
2. Use the command on your Jumper VM: ssh [vmware-system-user@10.196.186.20](mailto:vmware-system-user@10.196.186.20) to login to the TKG ControlPlane node. You will be asked for the password and you will supply the one you saved earlier
3. Once logged in to the TKG ControlPlane node, issue the commands below to generate a token that allows a new node to join the cluster (note: you need to be in sudo mode to do this):

sudo su

kubeadm token create --print-join-command > join-command.txt

1. Copy (scp) that “join-command.txt” file to the GPU-enabled Ubuntu VM so that the join command in that file can be used there.c

## Join the new Ubuntu VM to the TKG Cluster

1. Return to the Ubuntu VM and issue the command using the join-command.txt that you copied over from the TKG ControlPlane Node. The syntax will be similar to the following:

kubeadm join 10.196.186.210:6443 --token k10xhnz --discovery-token-ca-cert-hash sha256:12-f5

**Note:** Where the token and the --discovery-token-ca-cert-hash will be long strings of characters (These values here are shortened for example).

1. Once that command completes, you should see a message that your new node has joined the TKG cluster.
2. Logout from the TKC Control Plane VM on your Jumper VM using `kubectl vsphere logout`
3. You can now login to the TKG cluster from your Jumper VM using the command

kubectl vsphere login -–insecure-skip-tls-verify --server=SUPERVISOR-CLUSTER-CONTROL-PLANE-IP

--tanzu-kubernetes-cluster-name TANZU-KUBERNETES-CLUSTER-NAME

--tanzu-kubernetes-cluster-namespace SUPERVISOR-NAMESPACE-WHERE-THE-CLUSTER-IS-DEPLOYED

--vsphere-username VCENTER-SSO-USER-NAME

For example:

kubectl vsphere login --insecure-skip-tls-verify --vsphere-username administrator@vsphere.local --server=10.196.180.21 --tanzu-kubernetes-cluster-name endor --tanzu-kubernetes-cluster-namespace face-melter

**Note:** The 10.196.180.21 IP address belongs to the ControlPlane nodes for the TKG cluster. Note that here you are being explicit about the TKG cluster name and namespace you want to be in. You will be asked for your vSphere administrator password to login.

1. Once logged in, you will see a message with a set of contexts that you have access to, such as:

Logged in successfully.

You have access to the following contexts:

10.196.180.21

10.196.180.22

a-new-hope

endor

face-melter

1. You now choose the context that you want to operate in

kubectl config use-context <cluster-name>

1. Use the commands to see the existing nodes and pods within your cluster.

kubectl get nodes

kubectl get pods -A

## Copy over /bin/depmod as a workaround

TBD, this is a temporary workaround that need to be fixed on ubuntu 20. Just find an ubuntu box, any box will do, copy /bin/depmod, and put it into /sbin/depmod on your ubuntu 20 gpu worker node. chmod +x.

## CNI Setup for manually created worker node

1. Next, we’ll need to manually import the CNI container images. Verify the CNI pods are not running on new node:

kubectl get pods -A -o wide | grep worker2

kube-system antrea-agent-xstlg 0/2 Init:ImagePullBackOff 0 8m37s 192.168.30.43 tkg01-worker2  
kube-system kube-proxy-7548b 1/1 Running 0 18m 192.168.30.43 tkg01-worker2   
vmware-system-csi vsphere-csi-node-t6rgj 0/3 ContainerCreating 0 18m <none> tkg01-worker2

1. SSH to the one of the running worker node, refer the section "Accessing a TKG Cluster ControlPlane Node to generate a token to join the VM to the cluster "
2. Now run the below commands to import the images using containerd as a tar file

sudo ctr -n=k8s.io i export antrea-photon-v0.9.2\_vmware.1.tar vmw

are.io/antrea/antrea-photon:v0.9.2\_vmware.1

sudo ctr -n=k8s.io i export vsphere-csi-v0.0.1.alpha\_vmware.tar vmware.io/vsphere-csi:v0.0.1.alpha\_vmware.79-7ecdcb1

sudo ctr -n=k8s.io i export csi-node-driver-registrar-v0.0.1.alpha\_vmware.tar vmware.io/csi-node-driver-registrar:v0.0.1.alpha\_vmware.79-7ecdcb1

sudo ctr -n=k8s.io i export csi-livenessprob-v0.0.1.alpha\_vmware.tar vmware.io/csi-livenessprobe/csi-livenessprobe:v0.0.1.alpha\_vmware.79-7ecdcb1

sudo chmod 666 antrea-photon-v0.9.2\_vmware.1.tar vsphere-csi-v0.0.1.alpha\_vmware.tar csi-node-driver-registrar-v0.0.1.alpha\_vmware.tar csi-livenessprob-v0.0.1.alpha\_vmware.tar kube-proxy:v1.18.10\_vmware.tar

sudo ctr -n k8s.io i export kube-proxy:v1.18.10\_vmware.tar localhost:5000/vmware.io/kube-proxy:v1.18.10\_vmware.1

1. Now copy all tar files to the new Ubuntu node with below command

scp \*.tar ubuntu-username@ubuntu-IP:~/

1. SSH to the New Ubuntu node and run the below commands to import the images to containerd

sudo ctr -n=k8s.io i import antrea-photon-v0.9.2\_vmware.1.tar

sudo ctr -n=k8s.io i import vsphere-csi-v0.0.1.alpha\_vmware.tar

sudo ctr -n=k8s.io i import csi-node-driver-registrar-v0.0.1.alpha\_vmware.tar

sudo ctr -n=k8s.io i import csi-livenessprob-v0.0.1.alpha\_vmware.tar

sudo ctr -n k8s.io i import kube-proxy:v1.18.10\_vmware.tar

1. Now verify the new node status and pods status

# Phase 2: Access the GPU-enabled Tanzu Kubernetes Grid cluster and deploy NVIDIA GPU Operator

## Build GPU Operator 1.5.1 with vGPU 12.0 drivers on Jumper VM

1. GPU Operator 1.5.1 requires containers to be built and configured for your own individual license server and driver versions. The containers are built using Docker, which we will install on the Ubuntu Jumper VM created earlier.For instructions on how to install docker, refer to the instructions here:

https://docs.docker.com/engine/install/ubuntu/

1. Access to an NGC private registry is also required. This access should have been provided as a part of this proof-of-concept exercise. You will need to create an API key to use to push and pull container images from this private registry. Instructions on how to create an API key can be found here:

<https://docs.nvidia.com/ngc/ngc-overview/index.html#generating-api-key>

1. Download the vGPU files needed from the NVIDIA virtual GPU licensing portal. This consists of the guest driver, host driver, and manifest file. You will also need vCSlicenses from the licensing portal added to your NVIDIA virtual GPU license server.
2. Once you have the files, licenses, access to a VM with Docker, and access to an NGC private registry, build and push the container images for GPU Operator 1.5.1 according to the instructions here:

https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#considerations-to-install-gpu-operator-with-nvidia-vgpu-driver

## Access the GPU-enabled TKG Cluster

1. Once the GPU Operator container images have been built and pushed to your NGC private registry, you can now login to the TKG cluster from your Jumper VM using the following command:

kubectl vsphere login -–insecure-skip-tls-verify --server=SUPERVISOR-CLUSTER-CONTROL-PLANE-IP

--tanzu-kubernetes-cluster-name TANZU-KUBERNETES-CLUSTER-NAME

--tanzu-kubernetes-cluster-namespace SUPERVISOR-NAMESPACE-WHERE-THE-CLUSTER-IS-DEPLOYED

--vsphere-username VCENTER-SSO-USER-NAME

**Note:** The 10.196.180.21 IP address belongs to the ControlPlane nodes for the TKG cluster. Note that here you are being explicit about the TKG cluster name and namespace you want to be in. You will be asked for your vSphere administrator password to login.

1. Once logged in, you will see a message with set of contexts that you have access to, such as:

Logged in successfully.

You have access to the following contexts:

10.196.180.21

10.196.180.22

a-new-hope

endor

face-melter

1. You now choose the context that you want to operate in

kubectl config use-context <cluster-name>

1. Use the commands to see the existing nodes and pods within your cluster.

kubectl get nodes

kubectl get pods -A

## Deploy the NVIDIA GPU Operator with NVIDIA virtual GPU on the TKG cluster

1. Prior to deploying GPU Operator, you first need to cordon off all of the non-GPU-enabled nodes to ensure all of the pods for the GPU Operator get deployed on the GPU-enabled node. Once the GPU Operator is deployed, we can then uncordon the nodes. To cordon the nodes, run the following commands:

kubectl cordon <node-name>

$ kubectl cordon tkg01-cluster-workers-wrbwb-ffcbf6f6d-rdbrr

$ kubectl cordon tkg01-cluster-control-plane-74sjp

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

tkg01-cluster-control-plane-74sjp Ready,SchedulingDisabled master 29h v1.18.5+vmware.1

tkg01-cluster-worker2 Ready <none> 28h v1.18.10+vmware.1

tkg01-cluster-workers-wrbwb-ffcbf6f6d-rdbrr Ready,SchedulingDisabled <none> 29h v1.18.5+vmware.1

1. Next, apply the below yaml to the TKG cluster:

<https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-4CCDBB85-2770-4FB8-BF0E-5146B45C9543.html>

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: psp:vmware-system-privileged:default

namespace: default

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: psp:vmware-system-privileged

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: Group

name: system:nodes

- apiGroup: rbac.authorization.k8s.io

kind: Group

name: system:serviceaccounts

1. Next, ensure that you are working with version 1.5.1 of the NVIDIA GPU Operator. Follow the instructions linked below to create your kubernetes secrets, create the gpu-operator-resources namespace, and set the correct environment variables to reference the GPU Operator container images created earlier.

<https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#considerations-to-install-gpu-operator-with-nvidia-vgpu-driver>

1. You will need to download and set up the Helm tool version 3 in order to do this. The commands to do this are located at https://helm.sh/docs/intro/install/
2. Once Helm v3 is set up on your Ubuntu VM, follow the commands for the vGPU section (tab) at /

in order to deploy the GPU Operator.

**Note:** Ensure that the environment variables are set correctly to reference your private registry and container images created for the GPU Operator.

1. After installing the GPU Operator, apply the below yaml to create the necessary GPU resources:

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: psp:vmware-system-privileged:gpu-operator-resources

namespace: gpu-operator-resources

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: psp:vmware-system-privileged

subjects:

- kind: Group

apiGroup: rbac.authorization.k8s.io

name: system:serviceaccounts

1. We then need to edit the configuration of the Pod representing the “nvidia-driver-daemonset” using the command

kubectl edit daemonset nvidia-driver-daemonset -n gpu-operator-resources

Within the edit session:

* 1. Delete the entry that reads

hostnetwork: true

* 1. Delete the entry that reads

dnsPolicy: ClusterFirstWithHost

* 1. Write/Save the results and exit

1. Check your installation using the kubectl and the helm commands below and by using some of the sample applications, described below

1. Run the command in a terminal window on the Ubuntu node to show any activity that may be using the GPU. There may be no active processes on the GPU at this point. We will do this again while executing the sample TF-Notebook application (from the GPU Operator site) in a browser.

kubectl exec nvidia-container-toolkit-daemonset-sxkcm -n gpu-operator-resources -- nvidia-smi

1. At the end of the GPU Operator Setup, your TKG cluster should have a set of GPU-specific pods that are running or completed state as shown below:

kubectl get nodes

Table

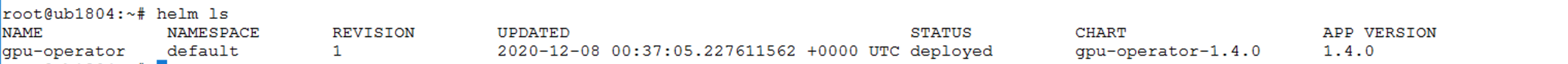
Description automatically generated

kubectl get pods -A| grep gpu

Graphical user interface, text

Description automatically generated

helm ls



1. Once the GPU Operator is deployed, uncordon the non-GPU enabled nodes using the following command:

kubectl uncordon <node name>

1. You can now proceed to further application testing on the infrastructure, using available Helm charts and containers.

# Phase 3: Deploy Sample AI and ML applications on the GPU-Enabled TKG Cluster

Once the TKG Cluster has the GPU Operator deployed and working successfully, access can be given to data scientists or AI researchers and developers to deploy sample applications and run GPU-enabled workloads. Two sample applications are given below.

## Deploy the TF-Notebook Application

1. The first demo we’ll use is the TF-notebook application as it is described on the NVIDIA GPU Operator Installation web page:

<https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#install-gpu-operator>

1. Deploy the application using kubectl as instructed. Once it is deployed, you can then bring up the Jupyter Notebook in a browser and run through sample code application for several different algorithms such as the classification example.
2. Instructions for accessing the Notebook in a browser (from your Jumpbox VM) are on the GPU-Operator installation page above. We use the “*kubectl exec nvidia….*” command given above in step 9 in order to check that the GPU is actually being used during the model.fit() training step in the ML application.
3. The different applications in the notebook should execute correctly to completion.

## Deploy the Intelligent Video Analytics (IVA) Application

The next application we’ll use is a pre-packaged IVA demo application from NVIDIA NGC. This application makes use of the DeepStream container and allows developers to build video processing applications that detect objects within the video stream and classifies them.

It is available on the NGC at:

<https://ngc.nvidia.com/catalog/helm-charts/nvidia:video-analytics-demo>

***NOTE:*** *You may need to delete the tf-notebook application before deploying this application, as the latter can take up a lot of your GPU’s memory. Use the below command to do this:*

kubectl delete -f https://nvidia.github.io/gpu-operator/notebook-example.yml

1. Use the instructions on the NGC page to bring this down to your Ubuntu VM/Kubernetes node. For example:

helm fetch <https://helm.ngc.nvidia.com/nvidia/charts/video-analytics-demo-0.1.5.tgz>

1. Once the .tgz file is downloaded, then execute tar xvf <filename> on it to extract the contents to a local directory called “video-analytics-demo”.
2. Change into that directory and you will see a file called values.yaml. Use your favorite editor on that file and add an entry as follows under the section entitled “nodeSelector”. You first remove the pair of curly braces that come after the original entry for “nodeSelector”.

Your entry should now look like the following (indentation is important here in YAML):

nodeSelector:

nvidia.com/gpu.present: "true"

1. Once you have made that change and saved the content to values.yaml, issue the command where the trailing dot indicates the current directory.

helm install video-analytics-demo .

This produces a set of output lines with one line indicating “Deployed”.

1. Once that is done, execute the command

kubectl get pods -A -o wide | grep video

Ensure that the pod for the video application itself was deployed on your Ubuntu node. The separate WebUI pod can be deployed on another worker node in the cluster.

1. You can then use a browser that can access the Ubuntu node to access the application at URL below to see the video play (it may take a few seconds to come up).

http://<IP address of the Ubuntu node>:31115/WebRTCApp/play.html?name=videoanalytics

You should be seeing images of cars and people being classified by the underlying ML application.

1. On the Ubuntu VM, while logged in to the TKG-S cluster, issue the command to see the GPU being used by the application, which is named in the output.

kubectl exec nvidia-container-toolkit-daemonset-sxkcm -n gpu-operator-resources -- nvidia-smi

# Appendix:

## Key Terms

* [A100](https://www.nvidia.com/en-us/data-center/a100/): New NVIDIA GPU platform that has the capability for Multi-Instance GPU support for higher physical isolation between consumers that are sharing a single device. The A100 demonstrates higher out-of-the-box performance than the previous high-end generation, the V100.
* Address Translation Services (ATS): The PCIe ATS capability enables a PCIe endpoint to request the DMA address translation from the IOMMU and cache the translation in the device side. By doing this, it can alleviate IOMMU pressure and improve the hardware performance in the I/O virtualization environment.
* Data Center GPU Manager (DCGM): NVIDIA DCGM is a suite of tools for managing and monitoring GPUs in cluster environments. It includes active health monitoring, comprehensive diagnostics, system alerts and governance policies including power and clock management. DCGM is built into the GPU Operator described below.
* EGX A100: NVIDIA EGXTM A100 combines the powerful performance of the NVIDIA Ampere architecture with the enhanced security and latency reduction capabilities of the NVIDIA Mellanox® ConnectX-6 Dx SmartNIC. The EGX cards pair a NVIDIA GPU and a Mellanox NIC or Smart NIC on the same PCIe board.
* GPU Operator: The GPU Operator, developed by NVIDIA, simplifies both the initial deployment and management of the following components: the GPU driver, K8s device plugin, container runtime and DCGM. By containerizing all the components and using standard Kubernetes APIs for automating and managing these components including versioning and upgrades. The GPU Operator, is fully open-source and is available in the NVIDIA ​[GitHub repo](file:///\\Users\jmurray\Downloads\versioning%20and%20upgrades.%20The%20GPU%20operator%20is%20fully%20open-source%20and%20is%20available%20on%20our%20%E2%80%8BGitHub%20repo%E2%80%8B.)​.
* TKG: Tanzu Kubernetes Grid is VMware’s signed and validated upstream Kubernetes distribution that is available for use on vSphere. This is part of the vSphere version 7 with Tanzu product. That vSphere 7 with Tanzu product includes a tight integration of Kubernetes into vSphere itself.
* Virtual Compute Server (vCS): Virtual Compute Server (vCS) is the NVIDIA vGPU software edition build for users of compute-intensive virtual servers for artificial intelligence (AI), deep learning, or high-performance computing (HPC) workloads.
* vSphere Installation Bundle (VIB): A software package that gets installed on a vSphere ESXi host.

## Related Materials

* [VMware vSphere with Tanzu QuickStart Guide](https://core.vmware.com/resource/vsphere-tanzu-quick-start-guide-v1a#_Toc53677530)
* Supporting MIG in [Kubernetes](https://docs.google.com/document/d/1mdgMQ8g7WmaI_XVVRrCvHPFPOMCm5LQD5JefgAh6N8g/edit)
* [Supported NVIDIA GPUs & Validated Server Platforms](https://docs.nvidia.com/grid/latest/grid-vgpu-release-notes-vmware-vsphere/index.html#hardware-configuration)
* [Steps to enabling MIG in Kubernetes](https://docs.google.com/document/d/1bshSIcWNYRZGfywgwRHa07C0qRyOYKxWYxClbeJM-WM/edit#heading=h.jw5js7865egx)
* [CUDA Toolkit 11.0 target platforms](https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64)
* [CUDA Toolkit 11.0 System Requirements](https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#system-requirements)
* [NVIDIA Breaks 16 AI Performance Records in Latest MLPerf Benchmarks](https://blogs.nvidia.com/blog/2020/07/29/mlperf-training-benchmark-records/)
* [MLPerf Training v0.7 Results](https://mlperf.org/training-results-0-7)
* NVIDIA NGC vGPU Software Support (Mark Johnson)
* [GPU monitoring](https://github.com/NVIDIA/gpu-monitoring-tools/blob/2.0.0-rc.0/etc/dcgm-exporter/dcp-metrics-included.csv)
* <https://www.dkube.io/>