

Deploying AI/ML Platform on a GPU-enabled Dell APEX Private Cloud with Rancher Prime

August 2023

H19751

White Paper

Abstract

This white paper provides guidance for deploying Kubeflow on a GPU-enabled Dell APEX Private Cloud with Rancher Prime, for customers who require an on-premises AI/ML platform solution.

Copyright

The information in this publication is provided as is. Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

Copyright © 2023 Dell Inc. or its subsidiaries. Published in the USA August 2023 H19751.

Dell Inc. believes the information in this document is accurate as of its publication date. The information is subject to change without notice.

Contents

Executive summary 4

Dell APEX Private Cloud..... 6

Rancher Prime for Kubernetes 7

Solution architecture 8

Configuring Dell APEX Private Cloud and installing Kubeflow 9

Conclusion..... 21

References..... 22

Executive summary

Overview

Applications are the lifeblood of the modern enterprise, and organizations need the flexibility to run their applications in the manner that best aligns with their business requirements. Virtualization fundamentally shifted how this flexibility was achieved, and virtualized infrastructure quickly became a standard feature of enterprise data centers. As organizations embrace cloud-native architectures and containerized workloads orchestrated by Kubernetes, we are witnessing the next evolution in application architectures.

Container technologies enable development teams to provision isolated applications quickly. Customers who want to boost their productivity and reduce the time to value can use containers with departments focused on software development.

Kubernetes orchestration provides capabilities such as auto-scaling, security, and management of containerized applications. A persistent and stable datastore is required to run containerized applications within a Kubernetes cluster that can survive a pod's lifetime or the node on which it is running.

Rancher Prime is a Kubernetes management platform that simplifies cluster installation and operations. The platform can be on-premises, in the cloud, or at the edge, allowing the DevOps team to build and run containerized applications anywhere.

Dell APEX delivers cloud services for various data and workload requirements, enabling you to simplify transformation, adapt to evolving conditions, and control your data. APEX is based on innovative Dell Technologies infrastructure built with Intel flexibility and performance.

Dell APEX Cloud Services simplifies multicloud, providing a consistent experience with robust security across your clouds for compute, data storage, and data protection.

Dell APEX Private and Hybrid Clouds deliver options to automate the deployment of modern application infrastructure with Rancher Prime to provide a production-ready Kubernetes environment. This automation enables you to leverage a consistent infrastructure operational model across your Kubernetes deployments to accelerate time to cloud-native application development. With support for traditional and cloud-native applications on the same infrastructure, you can capitalize on the next evolution in enterprise applications.

Dell APEX Cloud environments with GPU configurations provide critical resources for VDI and AI ML workloads. Kubeflow is an open-source platform for machine learning and MLOps on Kubernetes. Kubeflow streamlines ML workflows by providing an intuitive UI and ease of configuration for new ML environments.

Audience

The audience for this paper includes sales engineers, field consultants, IT administrators, customers, and anyone interested in deploying Kubeflow in a Dell APEX Private Cloud environment with GPUs. This deployment uses RKE2 and Rancher Prime.

Readers are expected to have an understanding and working knowledge of containers, Kubernetes, NVIDIA, and APEX Private Cloud.

Revisions

Date	Part number/ revision	Description
August 2023	H19751	Initial release

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on this document. Contact the Dell Technologies team by [email](#).

Author: Juan Carlos Reyes

Contributors: Alex Renzetti, Gerson Guevara (SUSE Team)





Note: For links to other documentation for this topic, see the [Dell APEX Info Hub](#).

Dell APEX Private Cloud

Overview

[APEX Private Cloud](#) delivers integrated compute, storage, networking, and virtualization resources. With virtualization from VMware (vSphere and vSAN) powered by Dell VxRail hyperconverged infrastructure, you can deploy a simple and scalable private cloud solution on-premises or at an edge location. You can automate deployments and provide full-stack, single-click life cycle management through VxRail Manager.

Node types are standardized combinations of compute, memory, storage, and networking resources—defined by a fixed physical memory to a physical core ratio and powered by VxRail. They are optimized for your virtualized and containerized workload requirements, ranging from small (4 GB) to extra-large (32 GB) memory-to-CPU core ratios. We also offer special-purpose node types with graphics processing units (GPUs) from NVIDIA to address artificial intelligence and machine learning (AI/ML) and virtual desktop infrastructure (VDI).

 <p>Compute optimized</p> <p>Delivers high performance for running compute intensive workloads</p>	 <p>General purpose</p> <p>Includes multi-Instance GPU (MIG) to partition the GPU, allowing each fully isolated GPU instance with its own high-bandwidth memory, cache, and compute cores</p>	 <p>Memory optimized</p> <p>Built on NVIDIA Ampere architecture and PCIe Generation 4 (64 GB/s) to double the bandwidth of previous PCIe Generation 3</p>	 <p>Large scale optimized</p> <p>Delivers fast performance using an extra high memory-to-core ratio for workloads that process large data sets in memory</p>
4GB memory/core	8GB memory/core	16GB memory/core	32GB memory/core
<p>Use cases</p> <ul style="list-style-type: none"> High performance computing (HPC) Mainstream web servers Batch processing applications Network appliances Media encoding servers Online gaming servers 	<p>Use cases</p> <ul style="list-style-type: none"> Low-medium traffic web servers Database application servers Development and test servers Unstructured data and NoSQL databases Log and data processing 	<p>Use cases</p> <ul style="list-style-type: none"> Relational databases (MySQL, MariaDB, PostgreSQL, etc.) Large in-memory databases (SAP/HANA) Data mining Large web scale in-memory caches (Memcached) Smaller enterprise Java applications 	<p>Use cases</p> <ul style="list-style-type: none"> High performance relational databases (Oracle, Microsoft SQL, MySQL, etc.) Midsized in-memory databases (SAP/HANA) Web scale in-memory caches (Memcached) Enterprise Java applications Data mining

Optional GPU types and details

<p>Use cases</p> <ul style="list-style-type: none"> VDI density optimized 	<p>Use cases</p> <ul style="list-style-type: none"> General compute AI inference optimized 	<p>Use cases</p> <ul style="list-style-type: none"> VDI performance optimized Video encoding/decoding AI training and inference
Model NVIDIA A16 (or similar)	Model NVIDIA A30 (or similar)	Model NVIDIA A40 (or similar)
Built on NVIDIA Ampere architecture, providing double the user density compared to previous generation	Includes multi-Instance GPU (MIG) to partition the GPU, allowing each fully isolated GPU instance with its own high-bandwidth memory, cache, and compute cores	Built on NVIDIA Ampere architecture and PCIe Generation 4 (64 GB/s) to double the bandwidth of previous PCIe Generation 3

The solutions described in this white paper are built on four general-purpose nodes with an all-flash cluster storage capacity.

Rancher Prime for Kubernetes

Overview

Rancher Prime is an enterprise computing platform for running Kubernetes on-premises, in the cloud, and at the edge. It addresses the operational and security challenges of managing multiple Kubernetes clusters anywhere. Rancher Prime also provides IT operators and development teams with integrated tools for building, deploying, and running cloud-native workloads.

Rancher Prime deploys production-grade Kubernetes clusters from the data center to the cloud and edge, and unites them with centralized authentication, access control, and observability. Rancher Prime lets you streamline cluster deployment on bare metal, edge devices, private clouds, public clouds, and vSphere. It also secures the clusters using global security policies. You can use Helm or the Rancher Prime catalog to deploy and manage applications across all these environments, ensuring multicluster consistency with a single deployment.

The following figure shows the Rancher architecture. For more details, see the [Rancher Prime Technical Architecture Guide](#).

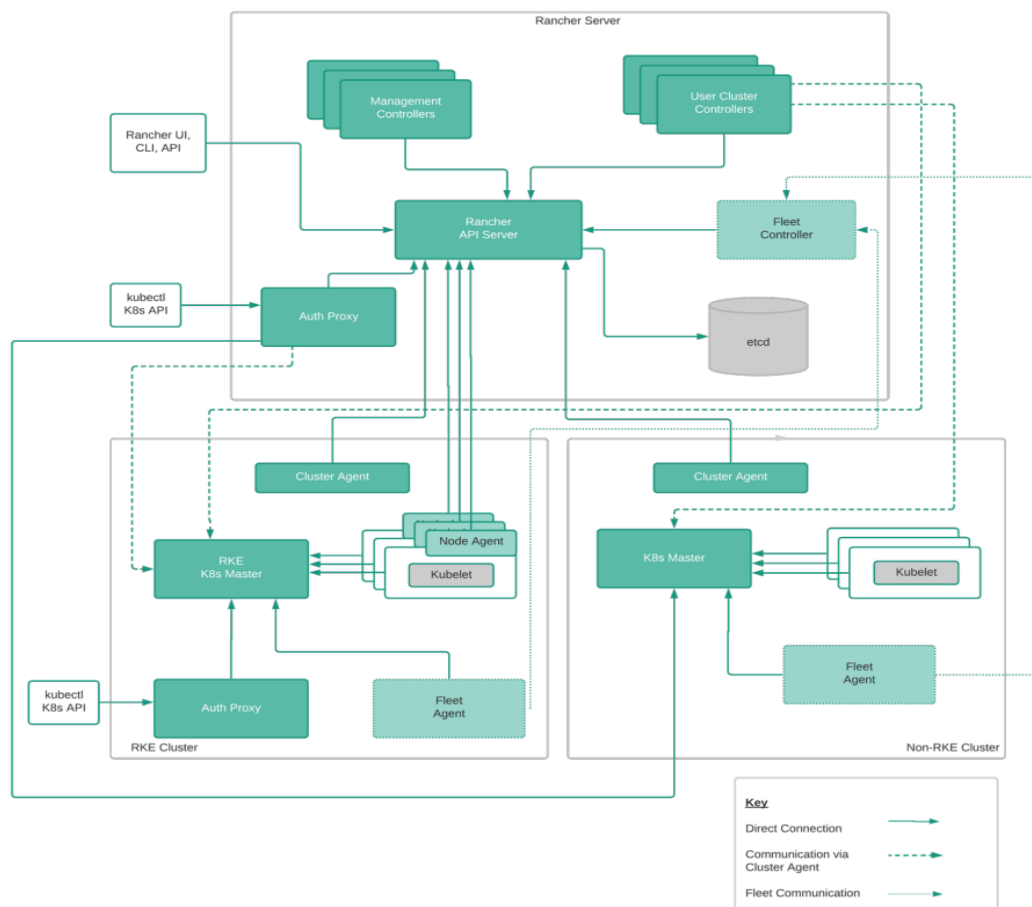


Figure 1. Rancher Prime architecture

Solution architecture

APEX Cloud Private Service is equipped with the Dell VxRail HCI integrated system. On top of the infrastructure, VMs are created to host an RKE2 cluster with Rancher Prime. Single-server and multiple-server RKE2 clusters can be deployed through the Rancher Prime UI. Through the Rancher Prime UI, users can deploy applications using the Rancher marketplace and manage them. Adding private and public container repositories increases the number of applications managed through Rancher Prime.

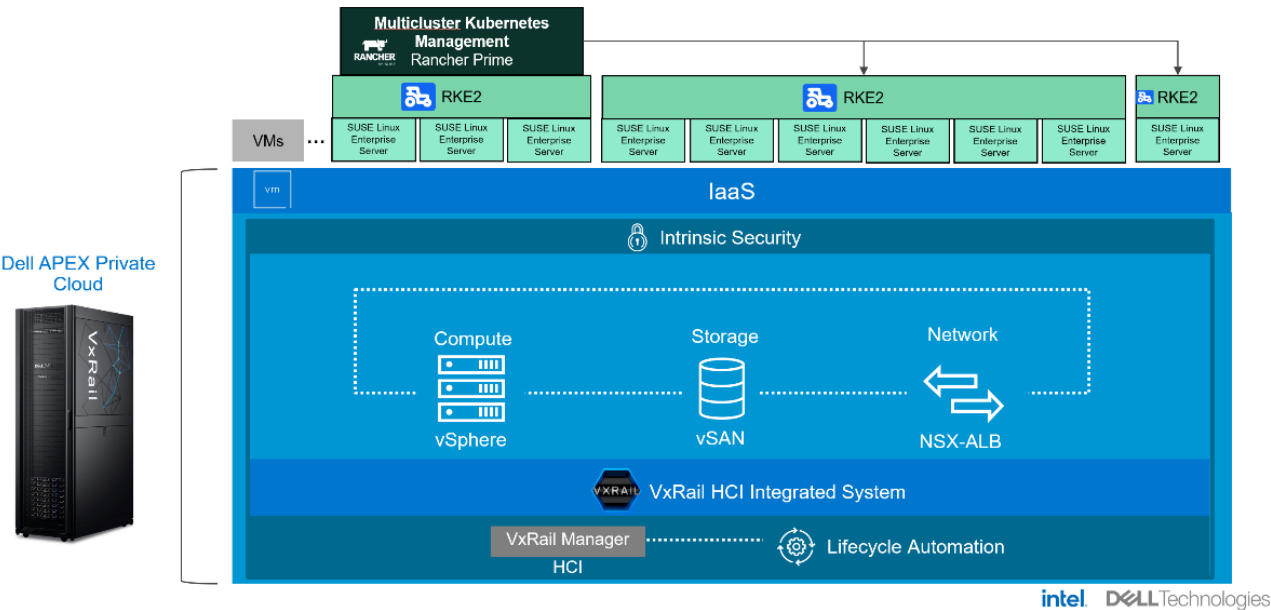


Figure 2. Dell APEX Private Cloud with Rancher Prime Kubernetes

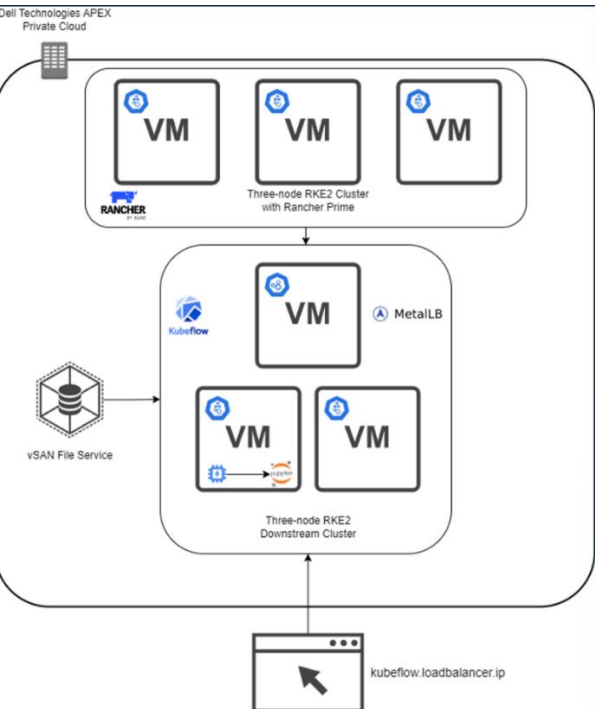


Figure 3. Environment and application architecture

Configuring Dell APEX Private Cloud and installing Kubeflow

Introduction

Follow the steps in this section to set up your Dell APEX Private Cloud to deploy the AIML platform Kubeflow. Kubeflow is an open-source platform for machine learning and MLOps on Kubernetes.

When we validated this solution, we used Rancher Prime v2.7.4, RKE2 release 2, Kubernetes v1.24, SLES15 SP4, and Ubuntu 20.04.

Requirements

The requirements for building this Kubeflow Platform are as follows:

Table 1. Requirements for Kubeflow platform on APEX Private Cloud

Name	Version	Description	Reference
Kubeflow	1.7	Open-source MLOps Platform	https://github.com/kubeflow/manifests#installation
Ubuntu Image	20.04	Cloud image for new RKE2 downstream cluster	https://cloud-images.ubuntu.com/focal/current/
NVIDIA vGPU Software	13.4	Install directly to ESXi hosts	https://docs.nvidia.com/grid/index.html
Rancher Server	2.7.4	Rancher server is used from Workstation VM.	https://www.suse.com/suse-rancher/support-matrix/all-supported-versions/
Rancher Kubernetes Engine Government (RKE2)	1.24.4+rke2r1	RKE2 is used from Workstation VM.	https://github.com/rancher/rke2/releases
3 SLES15 SP4 nodes *	SLES15 SP4	Ensure the nodes are accessed using SSH, and the required ports must be opened before the cluster installation.	https://rancher.com/docs/rke/latest/en/os/#ports

* A working DNS or fully qualified domain name (FQDN) must be appropriately configured for all the nodes.

Preparing the Dell APEX Private Cloud

The Dell APEX Private Cloud environment we are using comes with vSAN enabled. Each node has an NVIDIA T4 GPU.

An infrastructure administrator can follow [this documentation](#) to install the NVIDIA Virtual GPU Manager for the cluster.

vSAN File Services is [enabled](#) to create native vSAN file shares without extra storage on the Dell APC environment. With the NFS share, the Kubernetes clusters will have access to storage for ReadWriteMany persistent volumes. A storage policy configured with RAID 1 with Stripe Width 8 helps to guarantee the best performance by distributing the data across all the vSAN disk groups while not compromising data resiliency.

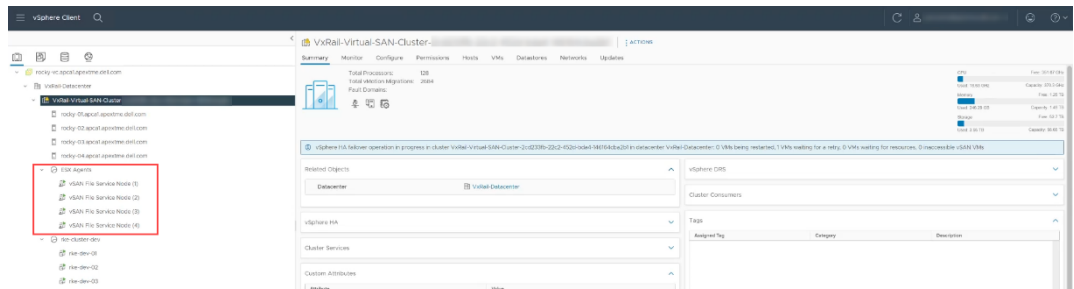


Figure 4. vSAN File Service configured for each node

Rancher Prime will use the cloud Ubuntu image templates to create new Kubernetes clusters. This [Ubuntu OVA image](#) is deployed to the Dell APC environment for this solution. After deployment, upgrade the VM image to the latest compatible version.

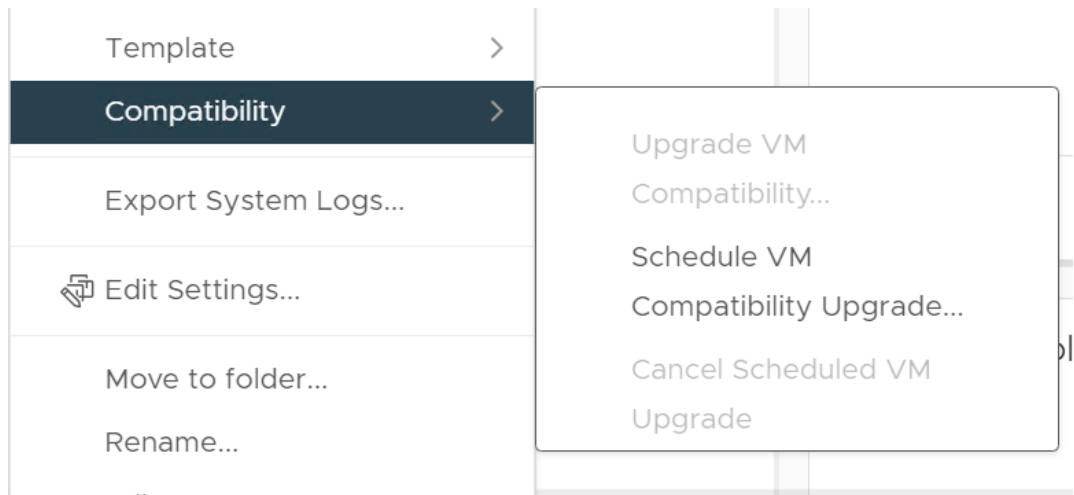
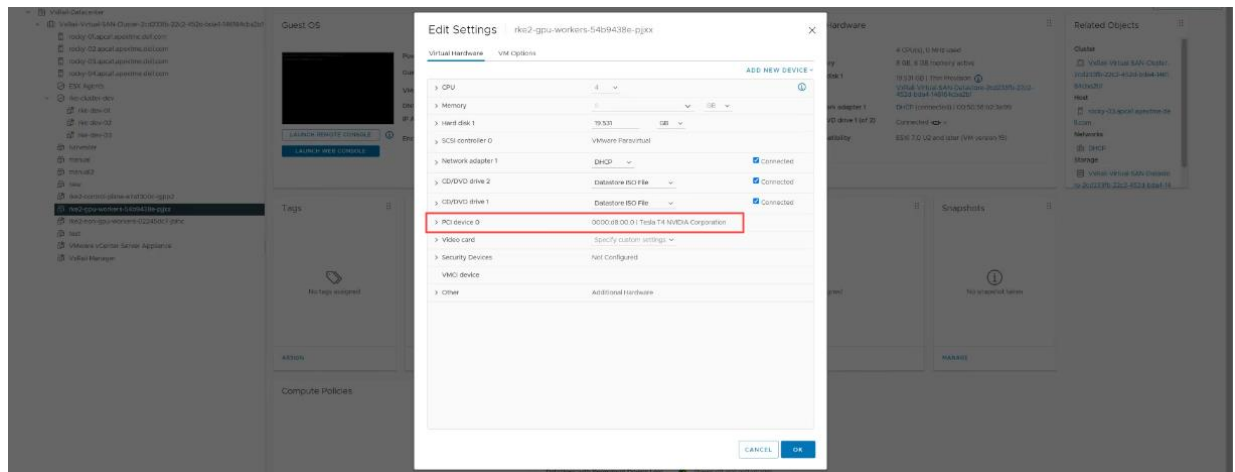


Figure 5. Upgrade the Ubuntu VM compatibility to ensure that all PCI devices work

Create a clone of the template, edit settings to customize the peripherals of the virtual machine, and add the GPU PCI device. Depending on the system's GPU version, you can choose from Dynamic, DirectPath IO, and NVIDIA GRID vGPU. Regular DirectPath IO was configured for this paper.



In the APC environment, a three-node RKE2 cluster is running and configured with Rancher Prime. More details on the Dell APC standard configuration and RKE2 cluster

can be found in this [white paper](#). The vSphere cloud credentials of a power-user account were added to Rancher Prime. This brings all the information of the Dell APC environment into the Rancher UI.

The main workstation has the following software and binaries already installed:

- Helm
- Kubectl
- Kustomize
- Go
- Kubeflow git

With this prep work complete, you can start installing and configuring Kubeflow.

Installing Kubeflow

From the Rancher Prime user interface (UI), we create a new RKE2 cluster. This cluster will have three pools: control plane, GPU workers, and non-GPU workers.

Scheduling
Choose what supervisor the virtual machine will be scheduled to

Data Center: VxRail-Datacenter
Data Store: VxRail-Virtual-SAN-Datastore-2cd233fb-22c2-452d-bda4-146164ca2b1
Host: Any

Resource Pool: host/VxRail-Virtual-SAN-Cluster-2cd233fb-22c2-452d-bda4-146164ca2b1/Resources
Folder:

Specific host to create VM on (leave blank for standalone ESX or for cluster with DRS)

Instance Options
Choose the size and OS of the virtual machine

CPU: 4 Cores
Memory: 8192 MiB
Disk: 20000 MiB
Operating System: Linux

Creation method: Deploy from template: Data Center

Cloud Config YAML:

```
1 #cloud-config
2 users:
3 - default
4 - name: juanreyes
5   sudo: ALL=(ALL) NOPASSWD:ALL
6   groups: sudo
7   shell: /bin/bash
8   login_password: false
9   plain_text_password: password
10 ssh_pwauth: true
11 runcmd:
12 - [ systemctl, daemon-reload ]
13 - [ systemctl, disable, supervisor.service ]
14 - [ systemctl, stop, supervisor.service ]
15 - [ systemctl, disable, firewalld.service ]
16 - [ systemctl, stop, firewalld.service ]
17 - [ systemctl, disable, swap.target ]
18
```

Template: SLES_15_SP4
SLES_15_SP4_Small
ubuntu
ubuntu-20.04-gpu-passthrough

Figure 6. vSphere environment with Rancher Prime cluster already configured

For this solution, only one node was added to each pool. The spec configurations were four CPUs, 8192 Mem, 2000 MiB storage, and so on.

1. For the cloud-config YAML, we added a user, enabled the NFS common package, and stopped certain services.

```
#Cloud-init
users:
- default
- name: apexuser
  sudo: ALL=(ALL) NOPASSWD:ALL
  groups: sudo
  shell: /bin/bash
```

```
lock_passwd: false
plain_text_passwd: password
ssh_pwauth: True
packages:
  - nfs-common
runcmd:
  - [ systemctl, daemon-reload ]
  - [ systemctl, disable, apparmor.service ]
  - [ systemctl, stop, apparmor.service ]
  - [ systemctl, disable, firewalld.service ]
  - [ systemctl, stop, firewalld.service ]
  - [ systemctl, disable, swap.target ]
```

2. When creating the GPU worker, select the template with the GPU PCI enabled.
3. After a successful deployment, we navigate to the new cluster in the UI. Under the Apps tab, we select Repositories and add the NVIDIA repository by specifying these values:

Repository: Create

Name *
nvidia

Description
Any text you want that better describes this resource

Target

☒ http(s) URL to an index generated by Helm
☐ Git repository containing Helm chart or cluster template definitions

Index URL *
https://helm.ngc.nvidia.com/nvidia

Authentication
Create a HTTP Basic Auth Secret

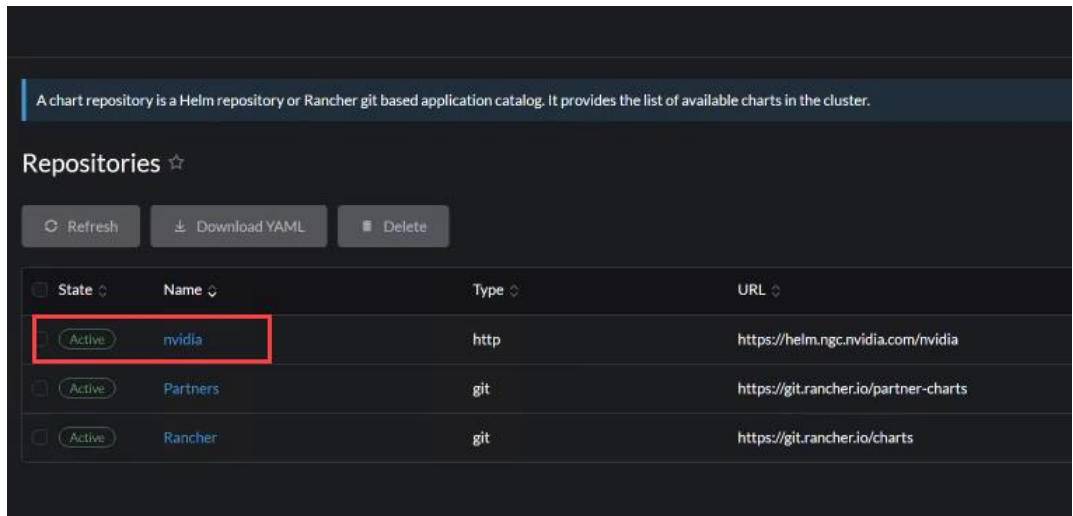
Username
\$oauthtoken

Index URL: https://helm.ngc.nvidia.com/nvidia

username: '\$oauthtoken'

password: [User's API key]

4. Make sure that you have a valid account.



- For bare-metal/passthrough with the pre-installed NVIDIA Container Toolkit (but no drivers), we need to update containerd to use NVIDIA as the default runtime and add the NVIDIA runtime configuration. To do this, we add the following config to `/var/lib/rancher/rke2/agent/etc/containerd/config.toml.tpl` and restart the containerd service.

```
root@rke2-control-plane-e7a1300c-gjpp2:~# ./patch-rke2-containerd.sh
grep: /etc/rancher/rke2/config.yaml: No such file or directory
version = 2
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
    enable_selinux = false
    sandbox_image = "index.docker.io/rancher/pause:3.6"
    stream_server_address = "127.0.0.1"
    stream_server_port = "10010"
  [plugins."io.containerd.grpc.v1.cri".containerd]
    disable_snapshot_annotations = true
    snapshotter = "overlayfs"
  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes]
    [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
      runtime_type = "io.containerd.runc.v2"
  [plugins."io.containerd.internal.v1.opt"]
    path = "/var/lib/rancher/rke2/agent/containerd"
root@rke2-control-plane-e7a1300c-gjpp2:~#
```

- Apply this patch to a control plane node.

```
#Patch Script
if grep -q 'data-dir' /etc/rancher/rke2/config.yaml; then
  DATA_DIR=$(grep 'data-dir' /etc/rancher/rke2/config.yaml
| awk '{print $2}')
else
  DATA_DIR=/var/lib/rancher/rke2
fi

CONTAINERD_CONFIG=$DATA_DIR/agent/etc/containerd/config.toml.
tpl
cat <<EOF | sudo tee $CONTAINERD_CONFIG
version = 2
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
```

```

enable_selinux = false
sandbox_image = "index.docker.io/rancher/pause:3.6"
stream_server_address = "127.0.0.1"
stream_server_port = "10010"
[plugins."io.containerd.grpc.v1.cri".containerd]
    disable_snapshot_annotations = true
    snapshotter = "overlayfs"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
    runtime_type = "io.containerd.runc.v2"
[plugins."io.containerd.internal.v1.opt"]
    path = "/var/lib/rancher/rke2/agent/containerd"
EOF

```

7. Now we can start the gpu-operator installation. Navigate to the Apps tab and select Charts. Here search for the gpu-operator.
8. We are going to make modifications to the default values.
 - a. First, we need to change the linux driver version that has been installed to our ESXi nodes. In this case version 470.141.03.
 - b. Next, to modify the default containerd location for the toolkit container configuration, add the following values:

toolkit:

```

env:
- name: CONTAINERD_CONFIG
  value:
/var/lib/rancher/k3s/agent/etc/containerd/config.toml.tmpl
- name: CONTAINERD_SOCKET
  value: /run/k3s/containerd/containerd.sock
- name: CONTAINERD_RUNTIME_CLASS
  value: nvidia
- name: CONTAINERD_SET_AS_DEFAULT
  value: "true"

```

With these modifications, all the components for the NVIDIA gpu-operator will run successfully.

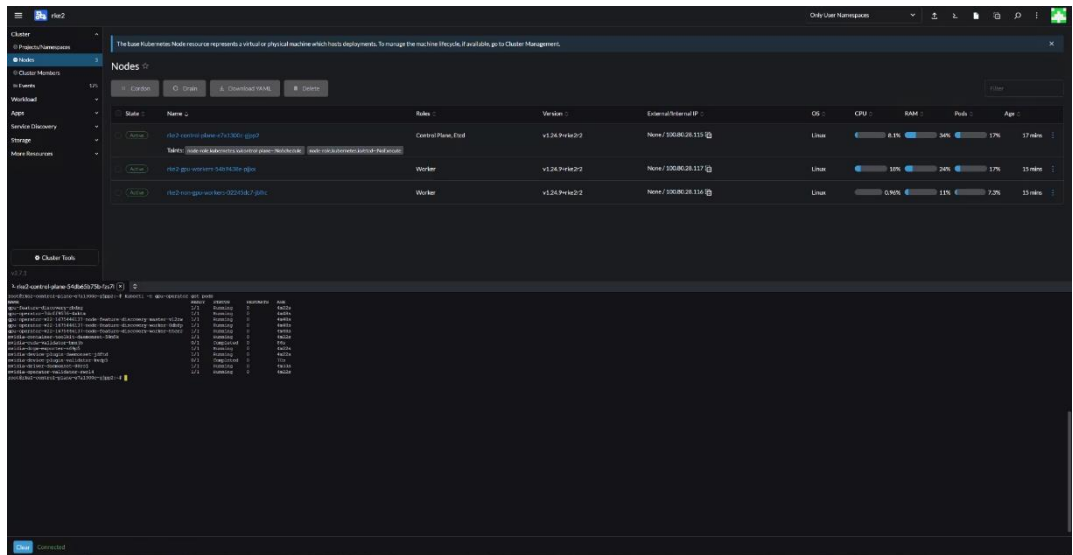


Figure 7. GPU operator successfully running

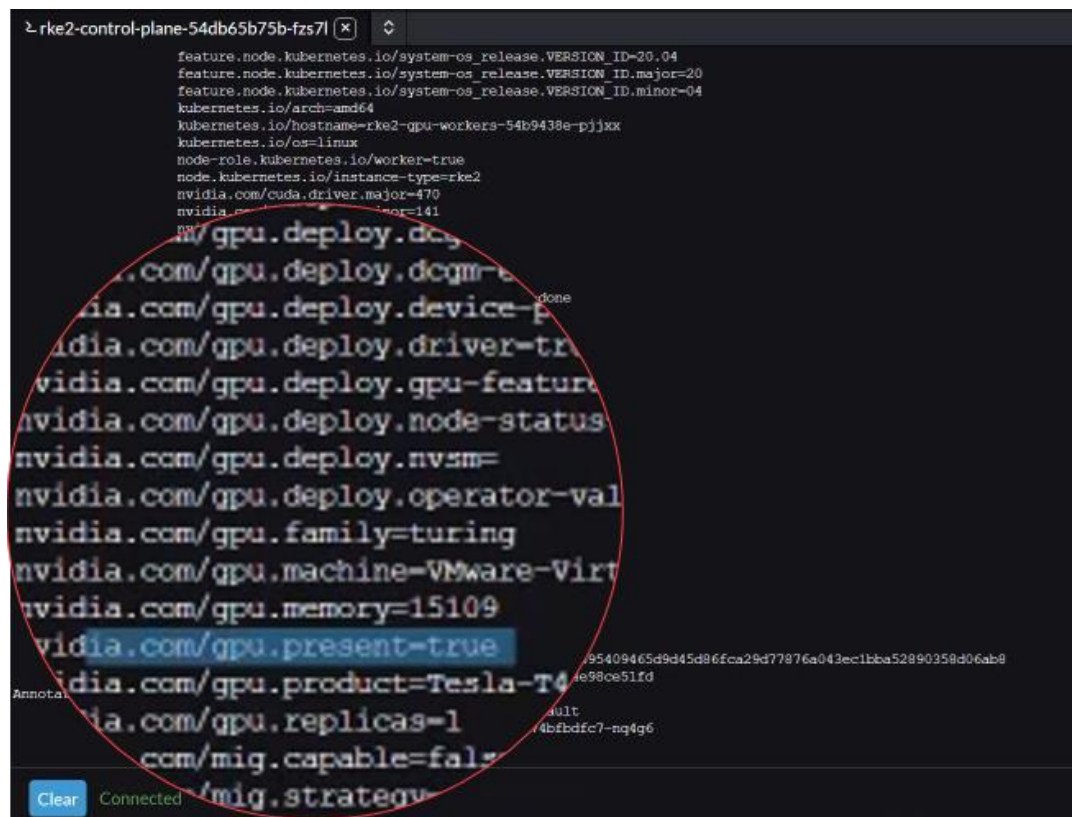


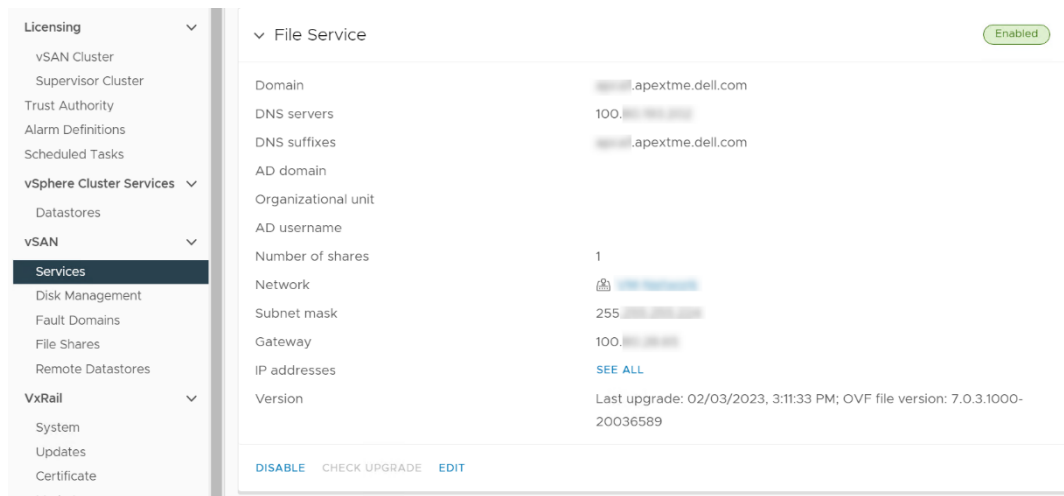
Figure 8. Users can also verify the node labels to see the new NVIDIA tags

- Next, we need to create a storage class with readwritemany capabilities. Jump to the workstation and connect to the Kubernetes cluster. You can then install the Kubernetes nfs subdir external provisioner following [these instructions](#).

```
helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner \
  --namespace infra \
```

```
--set nfs.server=nfs.apex.dell.com \  
--set nfs.path=/vsanfs/nfs-k8s-storage \  
--set storageClass.name=nfs-external \  
--set storageClass.accessModes=ReadWriteMany \  
--set podSecurityPolicy.enabled=false
```

Note: You can find the NFS file server IP and Path from the vSphere UI.



10. Patch the new storage class to make the default class with the following command:

```
kubectl patch storageclass nfs-external -p  
'{"metadata":{"annotations":{"storageclass.kubernetes.io/is-  
default-class":"true"}}}'
```

11. Now install Kubeflow. Navigate to where the Kubeflow git repo was downloaded.

```
git clone https://github.com/kubeflow/manifests.git  
git checkout v1.7-branch
```

Run the following command:

```
while ! kustomize build example | awk '!/well-defined/' |  
kubectl apply -f -; do echo "Retrying to apply resources";  
sleep 10; done
```

This installs all the components of the platform, and provides a great starting point to test out all the features available. In the official documentation, users can find how to install specific components. This is useful to avoid wasting resources on features that will not be used.

12. Verify that the pods are running.


```

juancarlos@jc-ubuntu-jump:~/Documents/kubeflow/manifests$ kubectl get pods -n kubeflow
NAME                                READY   STATUS    RESTARTS   AGE
admission-webhook-deployment-75f57fbd7-gzwm6   1/1     Running   0           41s
cache-server-56669b8cc7-88mpv                 2/2     Running   0           41s
centraldashboard-7d86dfcd5-cz2n2              2/2     Running   0           40s
jupyter-web-app-deployment-9d44d8bbb-sxkvd      1/1     Running   0           43s
katib-controller-7f89984894-bjmqg             1/1     Running   0           42s
katib-db-manager-9d67d9d46-7hdl8              1/1     Running   0           42s
katib-mysql-db6dc68c-zbvb5                   1/1     Running   0           41s
katib-ui-57866d9cbd-xhrv4                    1/1     Running   0           39s
kserve-controller-manager-0                   2/2     Running   0           39s
kserve-models-web-app-66f74955cc-zr65n        2/2     Running   0           43s
kubeflow-pipelines-profile-controller-749d8ff66b-nbv25 1/1     Running   0           43s
metacontroller-0                             1/1     Running   0           39s
metadata-envoy-deployment-9f59cddc8-ns7j1      1/1     Running   0           43s
metadata-grpc-deployment-784b8b5fb4-jfk28      1/2     Running   3 (24s ago)  41s
metadata-writer-785b44578c-t7r6z              2/2     Running   0           40s
minio-65dff76b66-95qm9                       2/2     Running   0           40s
ml-pipeline-65fd978948-ggtxk                 2/2     Running   0           39s
ml-pipeline-persistenceagent-6d94869545-9d8nz   2/2     Running   0           39s
ml-pipeline-scheduledworkflow-7659dc7fdd-clsv  2/2     Running   0           39s
ml-pipeline-ui-75b9ddd4c9-d55q8              2/2     Running   0           42s
ml-pipeline-viewer-crd-55cd5f9b89-msjhl        2/2     Running   2 (28s ago)  42s
ml-pipeline-visualizationserver-7d5df79444-z28vg 2/2     Running   0           42s
mysql-67f7987d45-fjfqm                      2/2     Running   0           41s
notebook-controller-deployment-785f55b77d-kqs95 2/2     Running   2 (34s ago)  41s
profiles-deployment-8566976bc8-nht85          3/3     Running   2 (26s ago)  40s
tensorboard-controller-deployment-699f99c6bb-4t5qh 3/3     Running   2 (26s ago)  40s
tensorboards-web-app-deployment-6c9bc9c8b8-wgcwt 1/1     Running   0           40s
training-operator-5b7fcc959-zmlcb             1/1     Running   0           39s
volumes-web-app-deployment-565f9d5b89-tp5kn    1/1     Running   0           39s
workflow-controller-6547f784cd-x6zsd          2/2     Running   2 (30s ago)  38s
juancarlos@jc-ubuntu-jump:~/Documents/kubeflow/manifests$

```

From here, users can use port-forwarding to access the Kubeflow UI. However, we will install MetalLB as our load-balancer implementation to provide a reachable IP to our Kubeflow service.

- Back in the Rancher Prime UI, we navigate again to the Charts section and search for MetalLB. In this configuration, we leave the default values and install them. Once deployed, we must add a couple of custom resources to the cluster. In the workstation, create a new file and add the following code:

```

---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
    - 100.0.0.0-100.0.0.254
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - first-pool

```

- Use the `kubectl` command to apply these resources:

```
kubectl apply -f ip_pool.yaml
```

These resources capture the IP range you want Kubernetes to use for services.

Note: The cluster needs access to the network where the range is located. If the range is in a VLAN other than the VM network, add both networks when you create the cluster.

- With the load balancer properly configured, patch the istio ingressgateway service with a LoadBalancer type, replacing the default NodePort type.

```
juancarlos@jc-ubuntu-jump:~/Documents/kubeflow/metallb$ kubectl get svc -n istio-system
NAME                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)                                     AGE
authservice          ClusterIP            10.43.148.11     <none>            8080/TCP                                    9m7s
cluster-local-gateway ClusterIP            10.43.190.85     <none>            15020/TCP, 80/TCP                          9m7s
istio-ingressgateway NodePort            10.43.191.62     <none>            15021:31012/TCP, 80:31987/TCP, 443:30644/TCP, 31400:30941/TCP, 15443:31519/TCP 9m7s
istiod               ClusterIP            10.43.76.138     <none>            15010/TCP, 15012/TCP, 443/TCP, 15014/TCP    9m7s
knative-local-gateway ClusterIP            10.43.235.122    <none>            80/TCP                                       9m7s
juancarlos@jc-ubuntu-jump:~/Documents/kubeflow/metallb$ kubectl -n istio-system patch service istio-ingressgateway -p '{"spec": {"type": "LoadBalancer"}}'
service/istio-ingressgateway patched
juancarlos@jc-ubuntu-jump:~/Documents/kubeflow/metallb$ kubectl get svc -n istio-system
NAME                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)                                     AGE
authservice          ClusterIP            10.43.148.11     <none>            8080/TCP                                    9m31s
cluster-local-gateway ClusterIP            10.43.190.85     <none>            15020/TCP, 80/TCP                          9m31s
istio-ingressgateway LoadBalancer        10.43.191.62     100.100.100.100  15021:31012/TCP, 80:31987/TCP, 443:30644/TCP, 31400:30941/TCP, 15443:31519/TCP 9m31s
istiod               ClusterIP            10.43.76.138     <none>            15010/TCP, 15012/TCP, 443/TCP, 15014/TCP    9m31s
knative-local-gateway ClusterIP            10.43.235.122    <none>            80/TCP                                       9m31s
juancarlos@jc-ubuntu-jump:~/Documents/kubeflow/metallb$
```

The ingressgateway should now have an IP from the configured range.

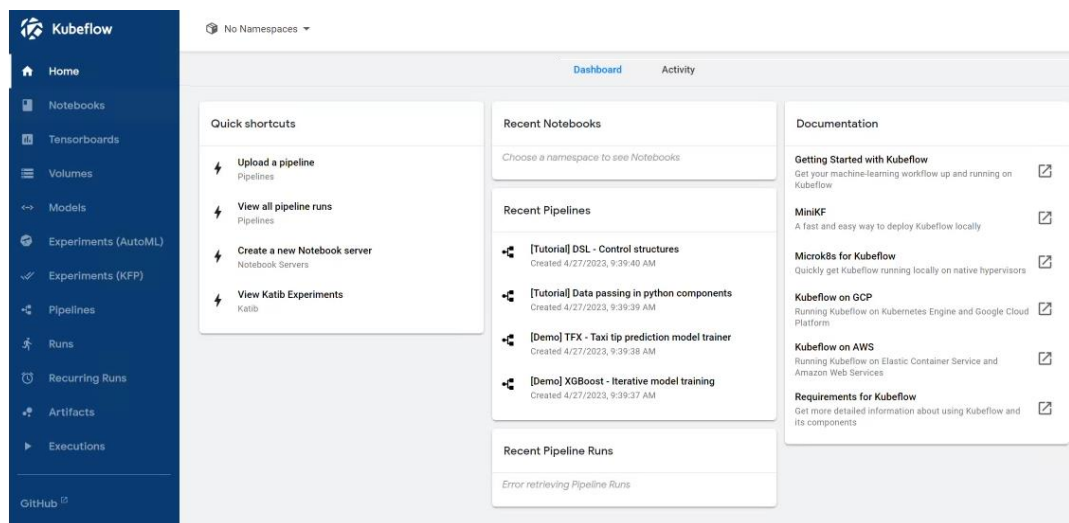
Jupyter Notebook example

Kubeflow is now running, and users can reach the web UI. Next, we will deploy a sample Jupyter Notebook and run an example job that shows some of the differences of running on GPU and non-GPU enabled nodes.

- Using the IP address of the istio-ingressgateway, we can navigate to the Kubeflow UI.
- Login with the default user and password.

Email address: user@example.com

Password: 12341234



- Click on the Notebooks tab and click New Notebook.

In this view, users can customize the configuration of the notebook with different images and resources.

For the demo, we use the following configuration:

Kubeflow

Home

Notebooks

Tensorboards

Volumes

Models

Experiments (AutoML)

Experiments (KFP)

Pipelines

Runs

Recurring Runs

Artifacts

Executions

Manage Contributors

GitHub

Documentation

Privacy • Usage Reporting
build version dev_local

kubeflow-user-example-c...

New notebook

kubeflownotebookswg/jupyter-tensorflow-cuda-full:v1.6.0-rc.1

Advanced Options

CPU / RAM

Requested CPUs: 2

Requested memory in Gi: 4

Advanced Options

GPUs

Number of GPUs: 1

GPU Vendor: NVIDIA

Workspace Volume

Volume that will be mounted in you home directory.

New volume ai-ml-volume, Empty, 10Gi

Type: Empty volume

Name: ai-ml-volume

Size in Gi: 10

Storage class

☐ Use default class

Class: nfs-external

Access mode

☐ ReadWriteOnce

☐ ReadOnlyMany

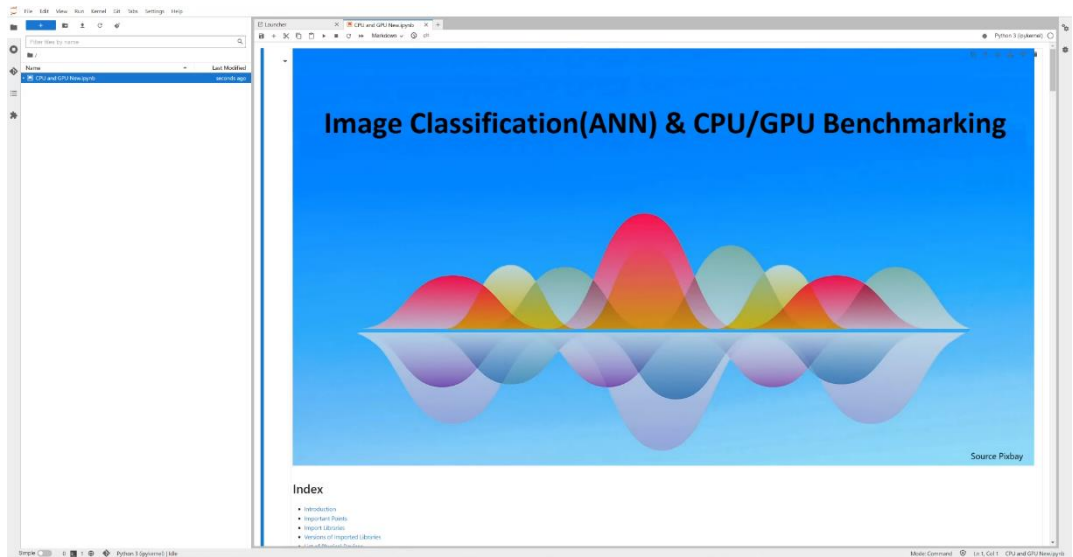
☒ ReadWriteMany

Mount path: /home/jovyan

+ Add new volume

+ Attach existing volume

4. Make sure to select the NFS storage class that was created earlier and use the readwritemany access mode.
5. After creating the new Notebook, click Connect.
6. Download the sample jupyter notebook from [here](#).



7. Run the entire notebook to see the results.

Note: If the training fails, one quick troubleshooting step is to increase the CPU in the notebook.

```
[17]: %%timeit -n1 -r1

# CPU benchmarking for 5 epoch
with tf.device('/CPU:0'):
    cpu_model = create_model()
    cpu_model.fit(X_train_scaled, y_train_categorical, epochs=5)

Epoch 1/5
1563/1563 [=====] - 53s 33ms/step - loss: 1.8108 - accuracy: 0.3522
Epoch 2/5
1563/1563 [=====] - 52s 33ms/step - loss: 1.6213 - accuracy: 0.4272
Epoch 3/5
1563/1563 [=====] - 51s 32ms/step - loss: 1.5410 - accuracy: 0.4545
Epoch 4/5
1563/1563 [=====] - 49s 32ms/step - loss: 1.4813 - accuracy: 0.4779
Epoch 5/5
1563/1563 [=====] - 49s 31ms/step - loss: 1.4297 - accuracy: 0.4970
4min 14s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

[18]: %%timeit -n1 -r1

# GPU benchmarking for 5 epoch
with tf.device('/GPU:0'):
    gpu_model = create_model()
    gpu_model.fit(X_train_scaled, y_train_categorical, epochs=5)

Epoch 1/5
1563/1563 [=====] - 4s 3ms/step - loss: 1.8143 - accuracy: 0.3547
Epoch 2/5
1563/1563 [=====] - 4s 3ms/step - loss: 1.6253 - accuracy: 0.4265
Epoch 3/5
1563/1563 [=====] - 4s 3ms/step - loss: 1.5463 - accuracy: 0.4547
Epoch 4/5
1563/1563 [=====] - 4s 3ms/step - loss: 1.4823 - accuracy: 0.4766
Epoch 5/5
1563/1563 [=====] - 4s 3ms/step - loss: 1.4333 - accuracy: 0.4957
22.2 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

Users can now keep testing different features and open new notebooks for data scientists to train new models.

Conclusion

The Kubernetes ecosphere continues to grow rapidly, providing more stability, security, and automatic service discovery. Using Rancher Prime and APEX Cloud Services streamlines some basic operations, such as setting up the Kubernetes cluster and deploying the workloads. This streamlining empowers administrators with the flexibility to deploy their Kubernetes environment quickly for developers and end users, enabling uninterrupted use of infrastructure and valuable resources such as GPUs. These solutions ensure that the Rancher Prime-managed Kubernetes downstream cluster workloads on APEX Private Cloud are available, consistent, and durable.

References

Dell Technologies documentation

The following Dell Technologies documentation provides other information related to this document. Access to these documents depends on your login credentials. If you cannot access a document, contact your Dell Technologies representative.

- [Dell APEX Info Hub](#)
- [Rancher Prime and RKE2 Kubernetes Cluster in APEX Private Cloud with PowerProtect Data Manager](#)

SUSE documentation

See also the following SUSE documentation.

- [Rancher Prime Technical Architecture Guide](#)

NVIDIA documentation

See also the following NVIDIA documentation.

- [NVIDIA Virtual GPU Software Documentation](#)

VMware documentation

See also the following VMware documentation.

- [Configure vSAN File Services](#)