Running Kubeflow on AI-Ready Enterprise Platform on VMware vSphere 7 with VMware Tanzu Kubernetes Grid

Overview

Table of Contents

Overview

A typical machine learning (ML) workflow usually includes stages such as data verification, feature engineering, model training, and deployment in a scalable fashion. Kubeflow provides a collection of cloud native components for developing and automating and maintaining all the stages of the ML process in a Kubernetes cluster either on-premises or in the cloud.

VMWare vSphere 7 delivers Artificial Intelligence (AI) and Developer-Ready infrastructure, scales without compromise, and simplifies operations, is helping in the adoption of AI in the enterprise. VMware and NVIDIA AI-Ready Enterprise software suite is an end-to-end cloud-native suite of AI tools and frameworks, optimized and exclusively certified by NVIDIA to run on VMware vSphere. This software suite handles the complexity associated with AI and ML efforts, giving organizations the confidence to update their infrastructure for AI and utilize AI to transform their business.

This paper will provide a general design and deployment guidance for running Kubeflow on VMware vSphere® 7 with VMware Tanzu® Kubernetes Grid™ with GPU access empowered by NVIDIA Artificial Intelligence Enterprise (NVAIE). We will also validate the core component functions to demonstrate that Kubeflow enables repeatable and reproducible machine learning workflows that can be shared between different teams such as data scientists, machine learning engineers, and DevOps.

Technology Overview

The technology components in this solution are:

- VMware vSphere
- VMware Tanzu Kubernetes Grid
- VMware vSAN File Service
- Kubeflow

VMware vSphere

VMware vSphere is industry's leading virtualization and workload platform, vSphere 7 brings efficiency, scale, and security to AI and modern applications. AI infrastructure is now a part of a managed environment within IT to provision specific GPU accelerators, compute, storage, and network resources for AI workload needs.

vSphere 7 delivers powerful support for the most modern GPUs such as NVIDIA Ampere-based A100 GPUs, including enhancements to performance boosting GPUDirect communications, vSphere also supports NVIDIA Multi-Instance GPU (MIG) technology to allow for partitioning of GPUs, which further increases utilization while strictly separating the virtual machines (VMs) sharing the GPU hardware.

With vSphere 7, developers and DevOps teams can use Kubernetes commands to provision VMs on hosts or Tanzu Kubernetes Grid clusters with vGPUs. This will help customers build and run their AI apps on GPU-enabled hardware using a self-service model. customers will have at their fingertips the power to build scalable AI applications.

VMware Tanzu Kubernetes Grid

VMware Tanzu Kubernetes Grid (TKG) provides organizations with a consistent, upstream-compatible, regional Kubernetes substrate that is ready for end-user workloads and ecosystem integrations. You can deploy Tanzu Kubernetes Grid across software-defined datacenters (SDDC) and public cloud environments, including vSphere, Microsoft Azure, and Amazon EC2.

Tanzu Kubernetes Grid provides the services such as networking, authentication, ingress control, and logging that a production Kubernetes environment requires. It can simplify operations of large-scale, multi-cluster Kubernetes environments, and keep your workloads properly isolated. It also automates lifecycle management to reduce your risk and shift your focus to more strategic work.

This document describes the use of VMware Tanzu Kubernetes Grid Service to support machine learning workloads that are distributed across the nodes and servers in the cluster. The Tanzu Kubernetes Grid Service provides self-service lifecycle management of Tanzu Kubernetes clusters. You use the Tanzu Kubernetes Grid Service to create and manage Tanzu Kubernetes clusters in a declarative manner that is familiar to Kubernetes operators and developers.

VMware vSAN File Service

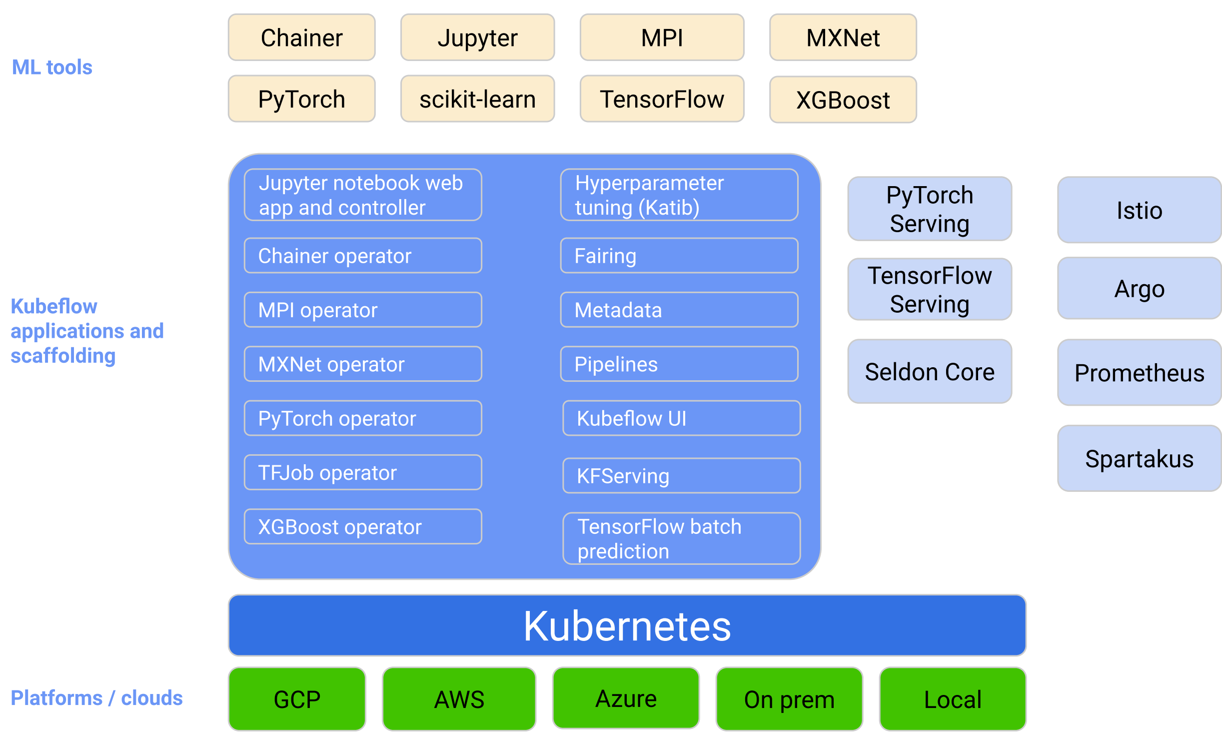
vSAN helps reduce the complexity of monitoring and maintaining infrastructure and enables administrators to rapidly provision a file share in a single workflow for Kubernetes-orchestrated cloud native applications. See VMware vSAN doc and VMware vSAN 7.0 Update 3 Release Notes for more information.

vSAN File Services is a layer that sits on top of vSAN to provide file sharing services. It currently supports SMB, NFSv3, and NFSv4.1 file shares. vSAN File Service brings in the capability to host the file shares directly on the vSAN cluster. See vSAN File Services.

The NFS feature of the vSAN File service was used to provide ReadWriteMany (RWM) volumes for this solution.

Kubeflow

Kubeflow is a free and open-source end-to-end machine learning platform designed to enable machine learning pipelines to orchestrate complicated workflows running on Kubernetes. Kubeflow provides components for each stage in the machine learning lifecycle, from exploration through to training and deployment.



This drawing is courtesy of the Kubeflow project website.

Figure 1: Kubeflow Application

Table 1 lists the main pillars of Kubeflow.

Table 1 Kubeflow Main Pillars

Component Name	Description
Central Dashboard	The central user interface (UI) in Kubeflow.
Kubeflow Notebooks	Kubeflow Notebooks provides a way to run web-based development environments inside your Kubernetes cluster by running them inside pods.
Kubeflow Pipelines	Documentation for Kubeflow pipelines
Katib	Katib is a project that is agnostic to machine learning frameworks. It can tune hyperparameters of applications written in any language of the users' choice and natively supports many machine learning frameworks, such as TensorFlow, MXNet, PyTorch, XGBoost, and others.
Training Operators	Training of machine learning models in Kubeflow through operators.
Kserve	Kserve allows you to serve your models as scalable APIs effortlessly and even do canary releases.
Multi-Tenancy	Multi-user isolation and identity access management (IAM)

These Kubeflow components can support multi-user isolation: central dashboard, notebooks, pipelines, AutoML (Katib), KServe. Furthermore, resources created by the notebooks (for example, training jobs and deployments) also inherit the same access.

Kubeflow can organize loosely-coupled microservices as a single unit and deploy them to a variety of locations, including on a laptop, on-premises, or in the cloud. It is a platform for data scientists to build and experiment with machine learning pipelines, also for machine learning engineers and operational teams who want to deploy machine learning systems to various environments for development, testing, and production-level serving. See kubeflow website for more information.

Configuration

Architecture

The Tanzu Kubernetes cluster was provisioned on top of vSphere consisting of multiple worker nodes, where each node is implemented as a virtual machine. Worker nodes that did not have a vGPU associated with them, were used for Kubeflow components. Worker nodes equipped with a vGPU are for pod deployment with GPU requirements. The NVIDIA GPU operator v1.9.1 was installed in the Tanzu Kubernetes cluster to allow users to manage the GPU nodes in the cluster. One ReadWriteMany (RWM) persistent volume from the vSAN file service was configured for shared data.

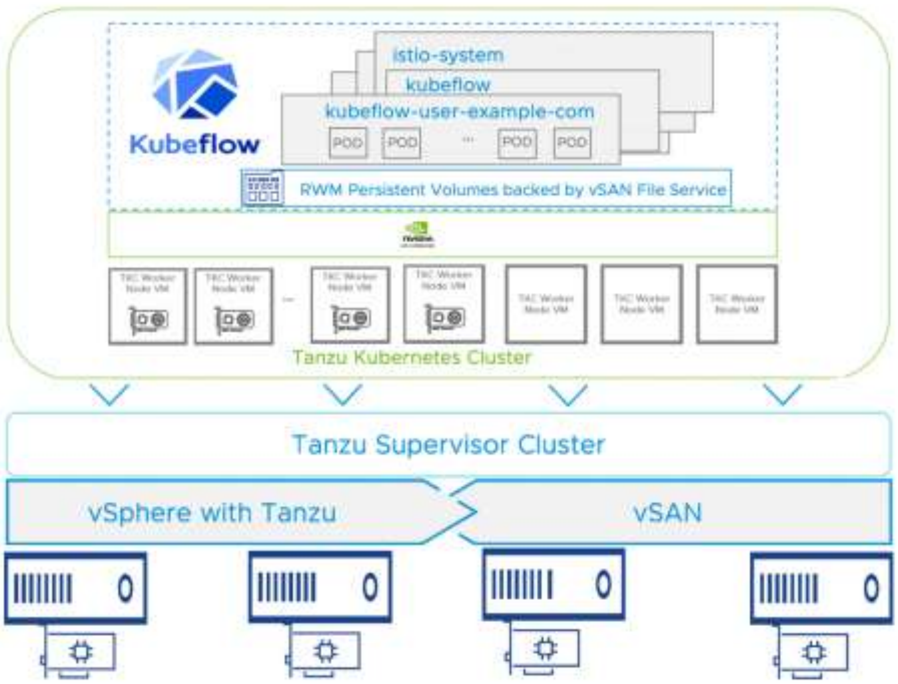


Figure 2: Solution Architecture

Hardware Resource

Server
A minimum of three servers that are approved on both the VMware Hardware Compatibility List and the NVIDIA Virtual GPU Certified Servers List are required.

GPU
A minimum of one NVIDIA GPU installed in one of the servers:

- Ampere class GPU (A100, A30, A0, or A10) (A100 and A30 are MIG capable, recommended, A40 is mainly focused on graphics)
- Turing class GPU (T4)

- Additional supported GPUs can be found here

In our validation environment, we used the following GPU resource: 1 x NVIDIA Ampere A100 40GB PCIe/server.

PROPERTY	SPECIFICATION
Server Model	Dell VxRail P670F
CPU	2 x Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz, 28 core each
RAM	512GB
Network Resources	1 x Intel(R) Ethernet Controller E810-XXV, 25Gbit/s, dual ports 1 x NVIDIA ConnectX-5 Ex, 100Gbit/s dual ports
Storage Resources	1 x Dell HBA355i disk controller 2 x P5600 1.6TB as vSAN Cache Devices 8 x 3.84TB Read Intensive SAS SSDs as vSAN Capacity Devices
GPU Resources	1 x NVIDIA Ampere A100 40GB PCIe

Software Resource

Table 1: Software List

Software	Version
vSphere	7.0 update 3c
Tanzu Kubernetes Release	v1.20.8+vmware.2
NVAIE	1.1
Kubeflow	v1.5
Helm	3.7.2

Network Design

The 25GbE NICs were used for vSphere management, vMotion, vSAN, and the Tanzu Kubernetes Grid management network. The 100GbE NICs were used for the vSAN file service and the Tanzu Kubernetes Grid workload network. In this case, the workload cluster was physically separated from the management and vSAN network. The workload cluster used the higher network bandwidth for both node-to-node interactions and read or write data on the vSAN file share.

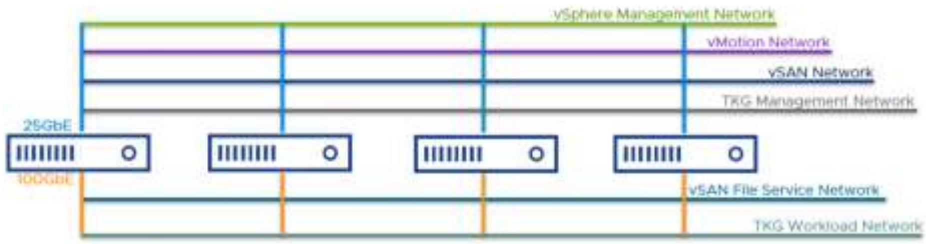


Figure 3: Network

vSphere Configuration

In this solution, a vSphere cluster should be pre-configured with vSAN enabled, and the ESXi hosts in the cluster should have NVIDIA GPUs installed.

Enable vGPU on ESXi Hosts

The vSphere administrator can follow *this article* to install the NVIDIA Virtual GPU Manager from NVAIE 1.1 package and enable vGPU support on the ESXi hosts that have GPU installed.

Configure vSAN File Service for Network File System (NFS)

With the vSAN file service enabled, we can create native vSAN NFS File shares without extra storage on the vSphere cluster. Most machine learning platforms need a data lake, a centralized repository to store all the structured and unstructured data. The Tanzu Kubernetes cluster can be configured with an NFS-backed ReadWriteMany (RWM) persistent volume across the pods to share and store data.

In Cluster **Configure->vSAN->File Service**, click **Enable** and follow the wizard to enable the File Service.

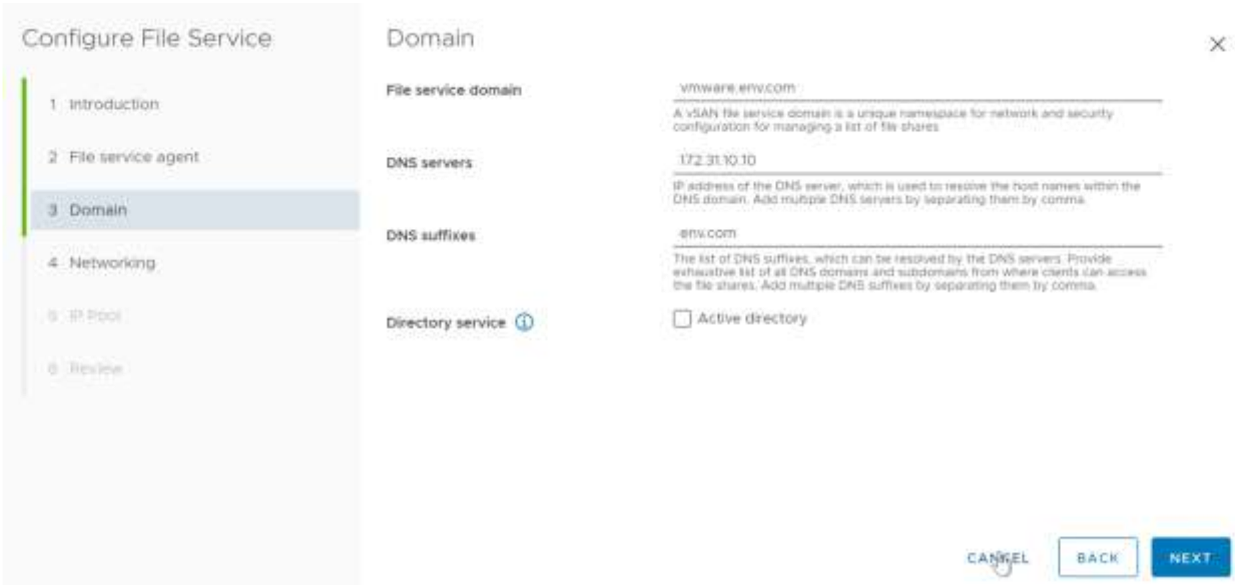


Figure 4: Configure File Service

In this solution, we created a vSAN NFS file share MLData with a size of 1TB. The storage policy was configured with RAID 1 with StripeWidth=8 to guarantee the best performance by distributing the data across all the vSAN disk groups while not compromising data resiliency.

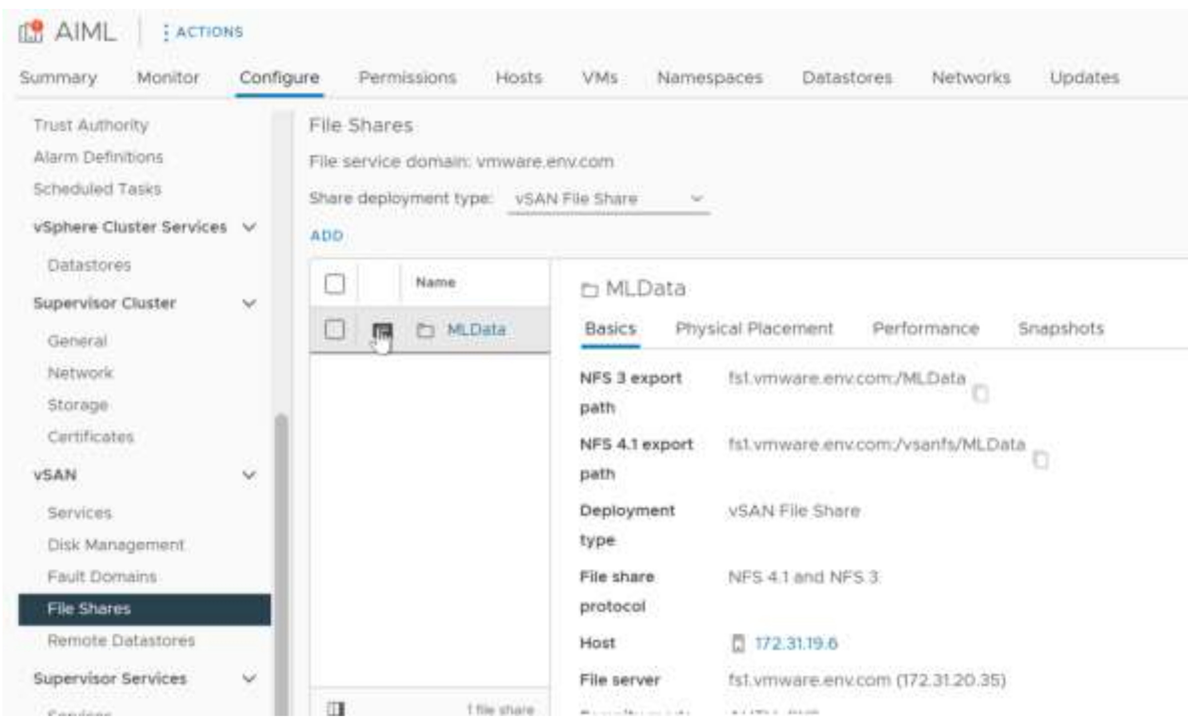


Figure 5: NFS File Share

Note: RWM volume is not natively supported with vSAN File Services in the current version. We can configure an RWM persistent volume according to [Using ReadWriteMany Volumes on TKG Clusters.](#) See the example [here](#).

For more information regarding vSAN file service, visit the link here.

Provision the Tanzu Kubernetes Cluster

While provisioning the Tanzu Kubernetes cluster, we defined the control planes and worker nodes as follows.

Table 2 Tanzu Kubernetes Cluster Definition

Role	Replicas	Storage Class	VM Class	Tanzu Kubernetes Release (TKR)
Control Plane	3	vsan-r1	best-effort-small	v1.20.8---vmware.1-tkg.2
GPU Worker Nodes	6	vsan-r1	gpuclass-a100	v1.20.8---vmware.1-tkg.2
Non-GPU Worker Nodes	3	vsan-r1	best-effort-xlarge	v1.20.8---vmware.1-tkg.2

Additionally, for each of the worker nodes, we configured a 50GB storage volume for container and a 50GB volume for the kubelet. The YAML file we used in this example for Tanzu Kubernetes cluster deployment can be found here.

Configure a Node in a Tanzu Kubernetes Cluster with vGPU Access

To configure Tanzu Kubernetes cluster with vGPU access, the vSphere administrator should first enable Workload Management in the vSphere Client, create the supervisor cluster and content library that will be subscribed to https://wp-content.vmware.com/v2/latest/lib.json, create the VM classes with vGPU access and create a new namespace with the VM classes configured with vGPU. Visit the link here for more details on the procedures and steps involved in this section.

In this solution, we configured the Tanzu Supervisor Cluster with haproxy v0.2.0 for load balancing. We added the pre-defined best-effort-small, best-effort-large, and best-effort-2xlarge VM classes to the namespace. To give the worker nodes vGPU access, we created and added a VM class (named gpuclass-a100) with the following specifications to the namespace:

Figure 6: VM Class Details

From the storage perspective, we added two vSAN storage policies to the namespace, one was vsan-r1 that is with RAID 1 configured, the other was stripe that is configured with RAID 5 and StripeWidth=8 to maximize the performance for the Tanzu Kubernetes cluster worker nodes.

Install the NVIDIA GPU Operator

After the Tanzu Kubernetes cluster is up and running, the developer logs into the Tanzu Kubernetes cluster that was created and follows the instructions in this link to install NVIDIA GPU operator on the Tanzu Kubernetes cluster. The installation process needs the developer to provide the NVIDIA CLS or DLS license token and the NGC account information. Refer to the NVIDIA Licensing Guide here.

In this solution, we installed GPU operator v1.9.1 and the script we used for installing NVIDIA GPU operator can be found here.

Monitoring Tools

Kubeflow Central Dashboard

The Kubeflow central dashboard provides quick access to the Kubeflow components deployed in your cluster where you can see a list of recent pipelines, notebooks, metrics, and an overview of your jobs as they are processed. See Central Dashboard to learn more.

vSAN Performance Service

The vSAN Performance Service is for monitoring the performance of the vSAN environment and helping users to investigate potential problems. The performance service collects and analyzes performance statistics and displays the data in a graphical format. You can use the performance charts to manage your workload and determine the root cause of problems.

Kubeflow Deployment

Introduction

This document provides instructions for deploying Kubeflow on Tanzu Kubernetes cluster.

Scope and Steps

Kubeflow provides components for each stage in the machine learning lifecycle, from exploration through to training and deployment. Operators can choose what is best for their users, there is no requirement to deploy every component of Kubeflow.

Prerequisites

NOTE: All prerequisites must be installed and configured before creating the Tanzu Kubernetes cluster.

Perform the following steps:

1. Download and Install kubectl for vSphere in our validation for Kubeflow version 1.5 of kubectl requires v1.21+.
2. Make sure you first create a Tanzu Kubernetes cluster and install GPU Operator on your Tanzu Kubernetes cluster in the configuration session.
3. Install Kustomize for Kubeflow installation

Deploy Kubeflow

We used the manifests for installation, perform the following steps to deploy Kubeflow 1.5.0 on your Tanzu Kubernetes cluster:

1. The following kubectl command creates a ClusterRoleBinding that grants access to authenticated users to run a privileged set of workloads using the default PSP vmware-system-privileged.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

2. Set the default storageclass for pv claims of kubeflow components such as MinIO and MySQL:

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

```
root@photon-HCIBench [ ~/manifests/example ]# kubectl get sc
NAME              PROVISIONER                                      RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
nfs-external      cluster.local/nfs-subdir-external-provisioner    Delete          Immediate           true                   26d
stripe            csi.vsphere.vmware.com                           Delete          Immediate           true                   23d
vsan-r1 (default) csi.vsphere.vmware.com                           Delete          Immediate           true                   33d
```

Figure 7: Set Default Storageclass

3. Download the scripts to deploy kubeflow by cloning the Github repository:

```
git clone https://github.com/kubeflow/manifests.git

git checkout v1.5-branch
```

4. You can install kubeflow official components by using either of the two options, Install with a single command or Install individual components. **Note:** Individual components may have dependencies. If all the individual commands are executed, the result is the same as the single command installation.

5. Verify all the pods are running. The kubectl apply commands may fail on the first try. This is inherent in how Kubernetes and kubectl work. Try to rerun the command until it succeeds.

   To check that all Kubeflow-related pods are ready, use the following commands:

```
kubectl get pods -n cert-manager

kubectl get pods -n istio-system

kubectl get pods -n auth

kubectl get pods -n knative-eventing

kubectl get pods -n knative-serving

kubectl get pods -n kubeflow

kubectl get pods -n kubeflow-user-example-com
```

   The following diagram shows the pods deployed in the Istio namespace:

```
kubectl get pod -n istio-system
NAME                                     READY        STATUS.     RESTARTS    AGE
authservice-0                            1/1          RUNNING      0          23h
cluster-local-gateway-7796d7bc87-9qb5v   1/1          Running      0          24h
istio-ingressgateway-64b7899489-ft5gn    1/1          Running      0          24h
istio-5d9bb9cb4-5zvzz                    1/1          Running      0          24h
```

Figure 8: Pods in istio-system Namespace

Figure 9 shows the pods deployed in the kubeflow namespace:

```
ubuntu@vmware-tanzu-jumpbox          kubectl get pod -n kubeflow

NAME                                                        READY    STATUS    RESTARTS   AGE
admission-webhook-deployment-7df7558c67
-gltpf                                                      1/1      Running   0          23d
cache-deployer-deployment-6f4bcc969-7j2jk                   1/1      Running   0          23d
cache-server-7cc6cbbf55-8f6m9                               1/1      Running   0          23d
centraldashboard-5dd4f57bbd-2k7f7                           2/2      Running   0          22d
jupyter-web-app-deployment-8d96db4cd-7n4g5                  1/1      Running   0          23d
katib-controller-58ddb4b856-fafhw                           1/1      Running   0          23d
katib-db-manager-6df878f5b8-27545                           1/1      Running   0          23d
katib-mysql-6dcb447c6f-xp8fc                                1/1      Running   0          23d
katib-ui-f787b9d88-gglr5                                    1/1      Running   0          23d
kfserving-controller-manager-                               1/1      Running   0          23d
kfserving-models-web-app-5d6cd6b5dd-58q6d                   1/1      Running   0          23d
kserve-models-web-app-6f45769bb6-5adpz                      1/1      Running   0          23d
kubeflow-pipelines-profile-controller-7fd7c77c5d-kx4591/1          Running   0          23d
metacontroller-0                                            1/1      Running   0          23d
metadata-envoy-deployment-76847ff6c5-2bdbz                  1/1      Running   0          23d
metadata-grpc-deployment-6f6f7776c5-btchf                   2/2      Running   0          23d
metadata-writer-78fc7d5bb8-7s9c9                            1/1      Running   0          23d
minio-5665df66c9-hfjm8                                      2/2      Running   0          23d
ml-pipeline-6bccbd7bd-5m6n6                                 2/2      Running   0          23d
ml-pipeline-persistenceagent-87b6888c4-bxlcb                2/2      Running   0          23d
ml-pipeline-scheduledworkflow-665847bb9-pj91m               2/2      Running   0          23d
ml-pipeline-ui-68cc764f66-w7gww                             2/2      Running   0          23d
ml-pipeline-viewer-crd-68777557fb-g7sms                     2/2      Running   0          23d
ml-pipeline-visualizationserver-58ccb76855-dlmwn            2/2      Running   0          23d
mysql-f7b9b7dd4-k65vv                                       2/2      Running   0          23d
notebook-controller-deployment-5d9c6c656c-4prq4             2/2      Running   0          23d
profiles-deployment-78ffd649f5-q7bk9                        3/3      Running   0          22d
tensorboard-controller-controller-manager-6848cb6846-       3/3      Running   0          23d
9h4sn
tensorboards-web-app-deployment-7c5db448d7-9ggp7            1/1      Running   0          23d
training-operator-7b8cc9865d-hffbp                          1/1      Running   0          23d
volumes-web-app-deployment-87484c848-62t9n                  1/1      Running   0          23d
workflow-controller-6fc6f67d66-5zpgx                        2/2      Running   2          22d
```

Figure 10: Pods in Kubeflow Namespace

6. Access the Kubeflow central dashboard:

   - **Option 1**: Port forward: The default way of accessing Kubeflow is via port-forward.

     ```
     kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80
     ```

     Example: http://localhost:8080

   - **Option 2:** NodePort/LoadBalancer/Ingress: since many of the Kubeflow web apps (for example, Tensorboard Web App, Jupyter Web App, Katib UI) use secure cookies, we need to set up HTTPS.

     We can access the dashboard using the LoadBalancer external IP address：

     - Change the type of the istio-ingressgateway service to LoadBalancer:

       ```
       kubectl -n istio-system patch service istio-ingressgateway -p '{"spec":
       {"type": "LoadBalancer"}}'
       ```

```
kubectl get svc -n istio-system
NAME            TYPE            CLUSTER-IP     EXTERNAL-IP    PORT (S)
Authservice   ClusterIP       10.100.82.68    ‹none>         8080/TCP
cluster-local-gateway ClusterIP 10.101.213.134 ‹none> 15020/TCP,80/TCP
istio-ingressgateway LoadBalancer 10.104.45.33 172.16.20.72
15021:32506/TCP,80:31917/TCP,443:32332/TCP,314

istiod               ClusterIP  10.103.211.151 (none>

5010/TCP.15012/TCP,443/TCP,15014/TCP

knative-local-gateway ClusterIP 10.111.221.131 ‹none> 80/TCP
```

Figure 4: Change istio-ingressgateway Service Type to Loadbalancer

And make changes to set up HTTPS configuration.

Configure HTTPS

Make the following changes:

- Update Istio Gateway to expose port 443 with HTTPS and make port 80 redirected to 443:

```
kubectl -n kubeflow edit gateways.networking.istio.io kubeflow-gateway

servers:

- hosts:

  -“*”

  port:

    name: http

    number: 80

    protocol: HTTP

  tls:

    httpsRedirect: true

-hosts:

 -“*”

 port:

   name: https

   number: 443

   protocol: HTTPS
```

```
tls:

  mode: SIMPLE

  privatekey:/etc/istio/ingressgateway-certs/tls.key

  serverCertificate:/etc/istio/ingressgateway-certs/tls.crt
```

Figure 5: Update istio Gateway Attributes

- Change the REDIRECT_URL in oidc-authservice-parameters configmap.

In our example, 172.16.20.72 is the IP address of the istio-ingressgateway.

```
kubectl -n istio-system edit configmap oidc-authservice-parameters

OIDC SCOPES: profile email groups
PORT: '"8080"'
REDIRECT URL: https://172.16.20.72/login/oide
SKIP AUTH URI: / dex
STORE PATH: /var/lib/authservice/data.db
```

Figure 11: Change REDIRECT_URL to Loadbalancer IP Address

Append the same to the redirectURIs list in dex configmap:

```
kubectl -n auth edit configmap dex
```

- Rollout restart authservice and dex

```
kubectl -n istio-system rollout restart statefulset authservice

kubectl -n auth rollout restart deployment dex
```

- Create a certificate.yaml with the YAML in **Figure 12** to create a self-signed certificate:

```
kubectl -n istio-system apply -f certificate.yaml

apiVersion:

cert-manager.io/vlalpha2

kind: Certificate

metadata:

name: istio-ingressgateway-certs

namespace: istio-system

spec:

commonName: istio-ingressgateway.istio-system.svc
```

```
ipAddresses:

- 172.16.20.72

isCA: true

issuerRef:

kind: ClusterIssuer

name: kubeflow-self-signing-issuer

secretName:istio-ingressgateway-certs
```

Figure 13: Create istio-ingressgateway Certificate

- We can access the Kubeflow Central Dashboard from https:// IP address of the istio-ingressgateway.

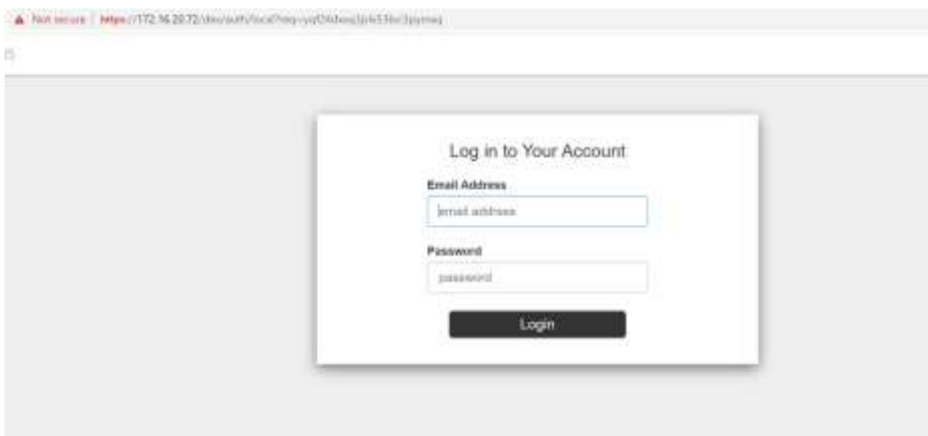


Figure 14: Kubeflow Login Page

Log in with the default user's credential. The default email address is user@example.com and the default password is 12341234. The default user's namespace is Kubeflow-user-example-com.

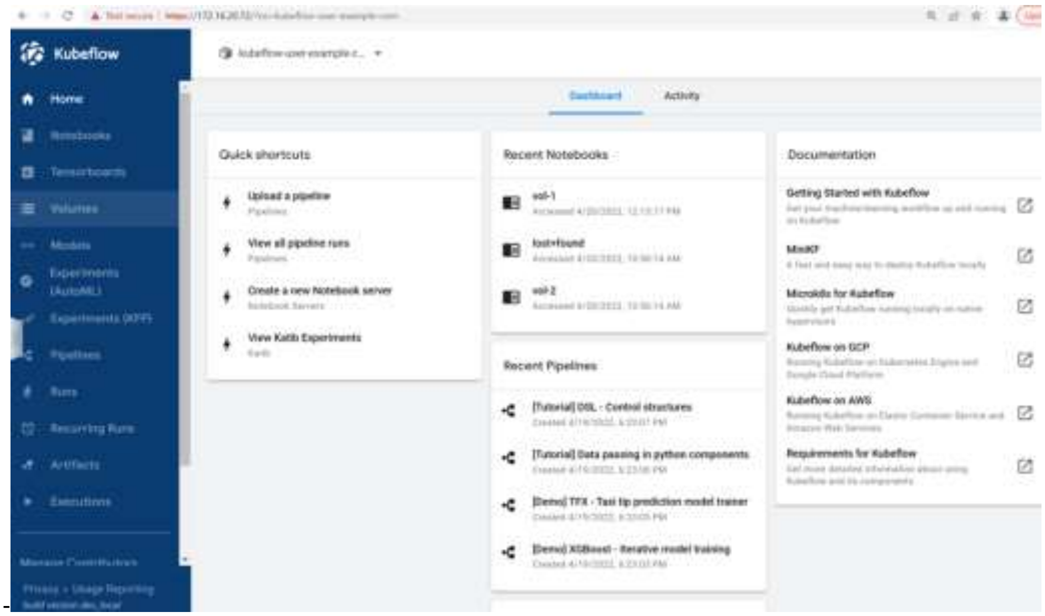


Figure 15: Kubeflow Central Dashboard

Add New Users

Add new user: users are managed by Kubeflow profile module:

```
cat <<EOF | kubectl apply -f

apiVersion: kubeflow.org/v1beta1

kind: Profile

metadata:

name: newuser’s namesmespacename   # replace with the name of profile you want

spec:

owner:

kind: User

name: newuser@example.com   # replace with the user email

EOF
```

Add the user credentials in dex in Kubeflow for basic authentication. Generate the hash by using bcrypt in the dex configmap:

```
kubectl edit cm dex -o yaml -n auth
```

Add the new user under the staticPasswords section:

```
-email: newuser@example.com
 hash: $2v$12$4K/VkmDdla10rb3xAt82zu8qk7Ad6ReFR4ICP9UeYE9ONLiN9D£72
```

```
 username: newuser
```

Figure 16: Add New User in Dex Configmap

For more information, refer to Kubeflow Getting Started.

Kubeflow Function Validation

Introduction

Kubeflow allows a notebook-based modeling system to easily integrate with the data preparation on a local data lake or in the cloud in a similar way. Kubeflow supports multi-tenant machine learning environments by managing the container orchestration aspect of the infrastructure that enables simple and effective sharing.

We validated the core functions from Notebooks to Pipelines and model serving and showcased an integrated end-to-end Pipeline example:

- Kubeflow Notebooks
- Run TensorFlow example
- Run PyTorch example
- Run Pipeline example
- KServe inference example
- End-to-end Pipeline example

Kubeflow Notebooks

Kubeflow Notebooks provides a way to run web-based development environments inside your Kubernetes cluster by running inside pods. It provides several default images. System administrators can provide customized notebook images for their organization with required packages pre-installed.

Access control is managed by Kubeflow's RBAC, enabling easier notebook sharing across the organization. Users can create notebook containers directly in the cluster.

Creating a Kubeflow Notebook

Data scientists can create notebook servers for their data preparation and model development.

To spin up a notebook, perform the following steps:

Click the **Central Dashboard Notebooks** tab and click **New Notebook**:

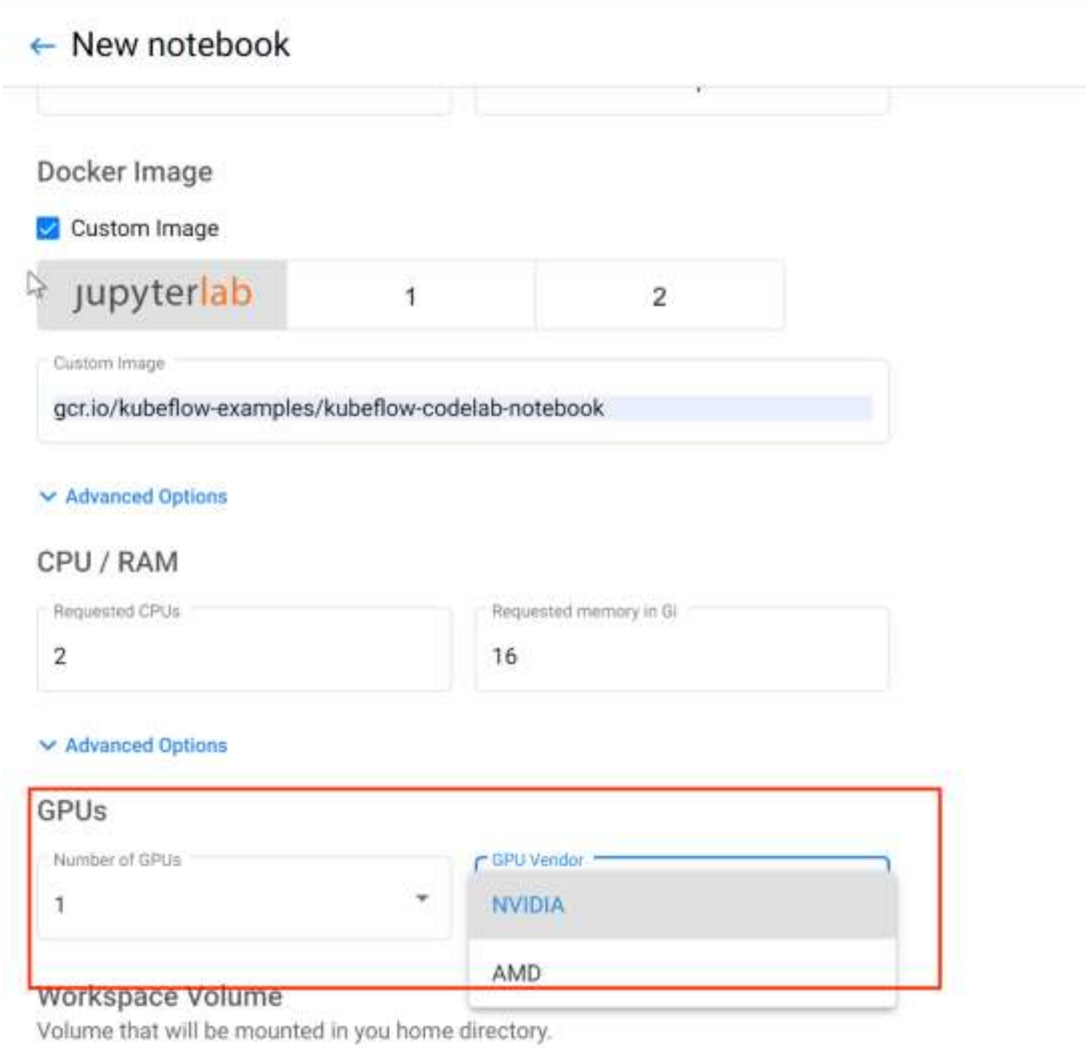


Figure 17: New Notebook Wizard

Note: Kubeflow uses "limits" in pod requests to provision GPUs onto the notebook pods (details about scheduling GPUs can be found in the [Kubernetes Documentation](#)). If we want to enable GPU on your notebook, in the GPU drop-down list, specify any "GPU Vendor" devices that your notebook server requests. In our environment, as **Figure 18** shows, we select NVIDIA GPU.

We can configure a ReadWriteMany persistent volume according to [Using ReadWriteMany Volumes on TKG Clusters.](#) See example [here](#).

Note: RWM volume is not natively supported with vSAN File Services in the current version.

Figure 19 shows the "external-nfs-pvc" volume in the Volumes Web UI, which is provisioned in the configuration section, also other ReadWriteOnce volumes in the user namespace are in the list.

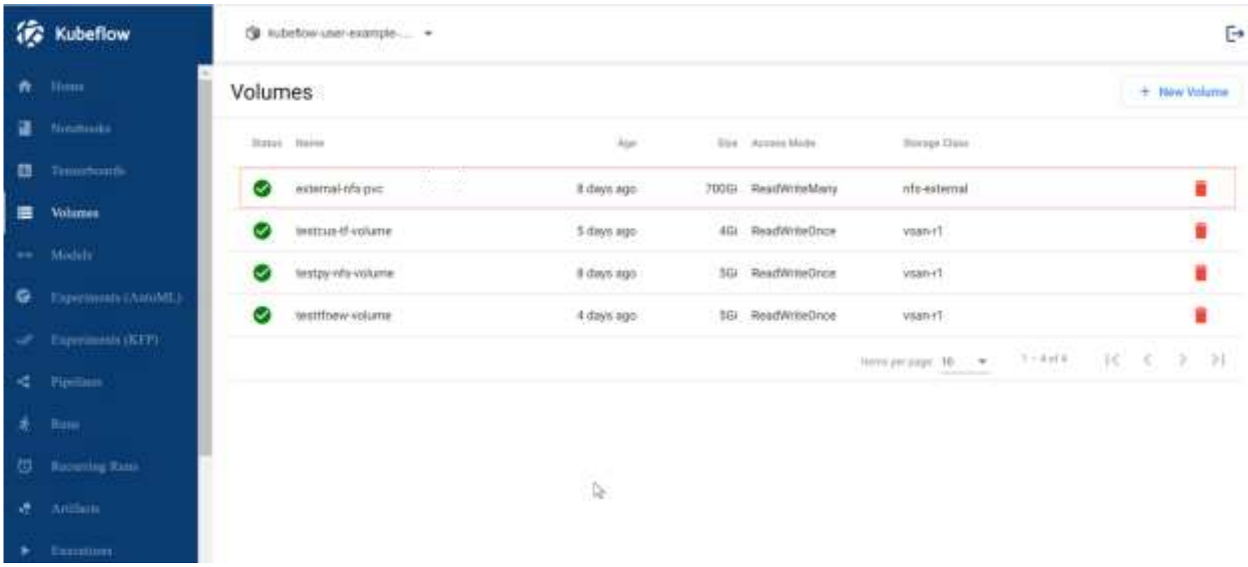


Figure 20: RWM Volume Backed by vSAN File Service

We can attach the existing RWM volume to the new notebook.

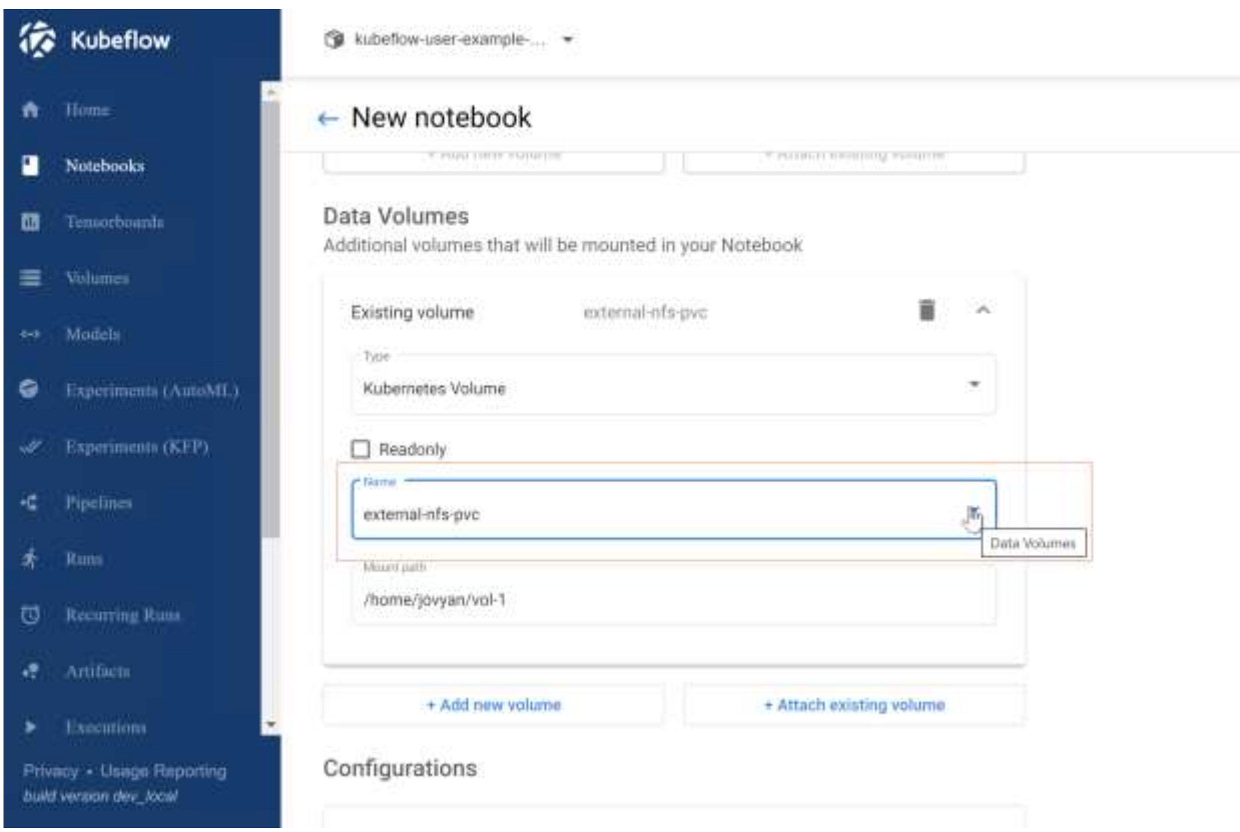


Figure 21: Attach the Existing Volume to New Notebook

For more information, see the notebooks quickstart guide.

Run TensorFlow Example

We use the BERT for TensorFlow Jupyter Notebook for testing. Bidirectional Embedding Representations from Transformers (BERT) is a method of pre-training language representations, which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. NVIDIA's BERT is an optimized version of Google's official implementation. The notebook provides a worked example for utilizing the BERT for TensorFlow model scripts.

After deploying the tensorflow-cuda image notebook, click on CONNECT, since the scripts are based on TensorFlow 1.15 version, either change some of the deprecated API to new ones or build a customized image on the same tensorflow version to make the code pass.

We chose the notebook server image with tensorflow+cuda 11.

Follow the steps to run an example use case of the BERT model for end user applications. **Figure 22** shows inference using GPU.

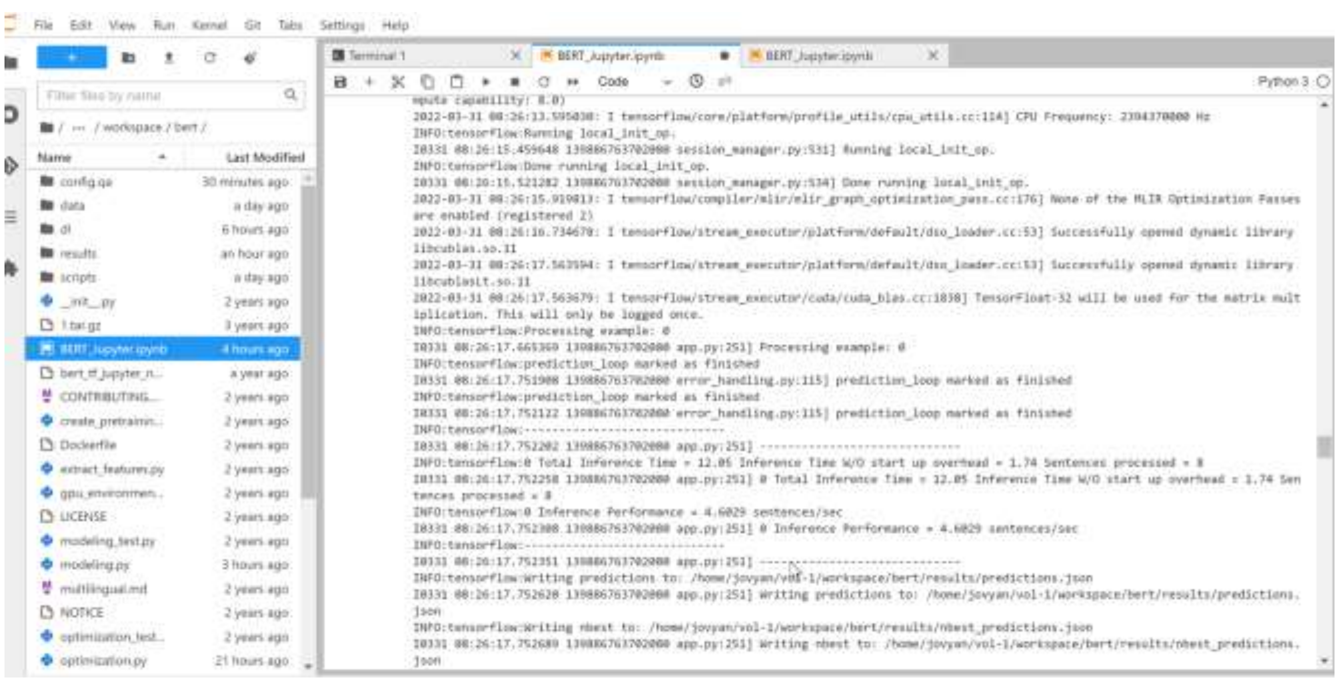


Figure 23: BERT_Jupyter Notebook using GPU

Figure 24 is an example prediction result for using the BERT for TensorFlow.

4b. Display Results:

```
display_results(predict_file, output_prediction_file)
```

Id	Question	Answer
Q1	What project put the first Americans into space?	Project Mercury
Q2	What program was created to carry out these projects and missions?	The Apollo program
Q3	What year did the first manned Apollo flight occur?	1968
Q4	What President is credited with the notion of putting Americans on the moon?	John F. Kennedy
Q5	Who did the U.S. collaborate with on an Earth orbit mission in 1975?	Soviet Union
Q6	How long did Project Apollo run?	1961 to 1972
Q7	What program helped develop space travel techniques that Project Apollo used?	Gemini missions
Q8	What space station supported three manned missions in 1973-1974?	Skylab

Figure 25: Prediction Result

Run Pytorch YOLOV5 Example

We use YOLOV5 to verify the inference and validation, which is a family of object detection architectures and models pre-trained on the COCO dataset.

Notes: Require customized image to have pycocotools library installed (which needs gcc library installed, this is not included in the default kubeflow notebook images).

In our validation, the GPU is NVDIA A100, we installed pytorch with cuda v 11.3. In the Notebook, we first installed below:

```
pip3 install torch torchvision torchaudio --extra-index-url
https://download.pytorch.org/whl/cu113
pip3 install pycocotools
```

Then we followed the [tutorial notebook]() to run the validation and inference case.

```
# Run YOLOv5x on COCO val
!python val.py --weights yolov5x.pt --data coco.yaml --img 640 --iou 0.65 --half
```

```
val: data=/home/jovyan/vol-1/yolov5/data/coco.yaml, weights=['yolov5x.pt'], batch_size=32, imgsz=640, conf_thres=0.001, iou_t
hres=0.65, task=val, device=, workers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, s
ave_conf=False, save_json=True, project=runs/val, name=exp, exist_ok=False, half=True, dnn=False
Python 3.7.0 required by YOLOv5, but Python 3.6.7 is currently installed
YOLOv5 🚀 v6.1-135-g7926afc torch 1.10.2+cu113 CUDA:0 (GRID A100-2-10C MIG 2g.10gb, 10236MiB)

Fusing layers...
YOLOv5x summary: 444 layers, 86705005 parameters, 0 gradients, 205.7 GFLOPs
val: Scanning '/home/jovyan/vol-1/datasets/coco/val2017.cache' images and labels
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all       5000      36335      0.743      0.626      0.683      0.496
Speed: 0.1ms pre-process, 11.2ms inference, 1.3ms NMS per image at shape (32, 3, 640, 640)

Evaluating pycocotools mAP... saving runs/val/exp/yolov5x_predictions.json...
Python 3.7.0 required by YOLOv5, but Python 3.6.7 is currently installed
loading annotations into memory...
Done (t=0.64s)
creating index...
index created!
```

Figure 26: YOLOV5 Validation on coco Dataset Screenshot Using A100 MIG

```
In [3]:
!python detect.py --weights yolov5s.pt --img 640 --conf 0.25 --source data/images
display.Image(filename='runs/detect/exp/zidane.jpg', width=600)

Downloading https://ultralytics.com/assets/Arial.ttf to /home/jovyan/.config/Ultralytics/Arial.ttf...
detect: weights=['yolov5s.pt'], source=data/images, data=data/coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45,
max_det=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_n
ms=False, augment=False, visualize=False, update=False, project=runs/detect, name=exp, exist_ok=False, line_thickness=3, hide_l
abels=False, hide_conf=False, half=False, dnn=False
Python 3.7.0 required by YOLOv5, but Python 3.6.7 is currently installed
YOLOv5 🚀 v6.1-135-g7926afc torch 1.10.2+cu113 CUDA:0 (GRID A100-2-10C MIG 2g.10gb, 10236MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.5 GFLOPs
image 1/2 /home/jovyan/vol-1/yolov5/data/images/bus.jpg: 640x480 4 persons, 1 bus, Done. (0.013s)
image 2/2 /home/jovyan/vol-1/yolov5/data/images/zidane.jpg: 384x640 2 persons, 2 ties, Done. (0.014s)
Speed: 0.7ms pre-process, 13.6ms inference, 36.7ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp

Out[3]:
```

Figure 27: YOLOV5 Inference Screenshot

Run Pipeline Example

A Kubeflow Pipeline is a portable and scalable definition of a machine learning workflow, based on containers. Kubeflow Pipelines are reusable end-to-end machine learning workflows composed of a set of input parameters and a list of the steps using the Kubeflow Pipelines SDK.

You can follow https://www.kubeflow.org/docs/components/pipelines/tutorials/build-pipeline/ to upload a compiled pipeline.

Kubeflow Pipelines offers a few samples that you can use to try out the pipelines quickly.

To run a basic pipeline, perform the following steps:

1. From the Kubeflow Pipeline UI

   Click the name of the sample XGBoost-iterative model training in Figure **28**, the source code is

   https://github.com/kubeflow/pipelines/tree/master/samples/core/train_until_good

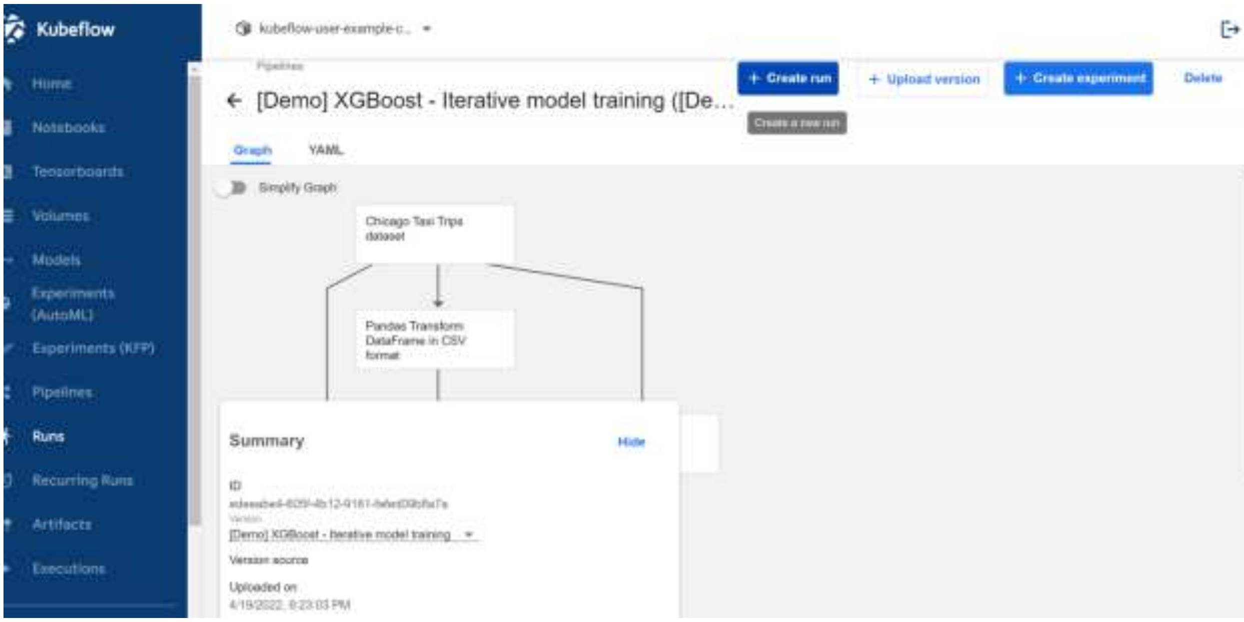


Figure 29: Example XGBoost Training Pipeline

A component in a pipeline can be responsible for data preprocessing, data transformation, model training, and so on.

The Artifacts include Pipeline packages, views, and large-scale metrics (time series). Use large-scale metrics to debug a pipeline run or investigate an individual run's performance. Kubeflow Pipeline installation stores the artifacts in an artifact store Minio server by default. Below is the pipeline running log stored in Artifacts:

Figure 30: Artifact Stores in MinIO Server

The lineage explorer displays the running flow of pipeline components:

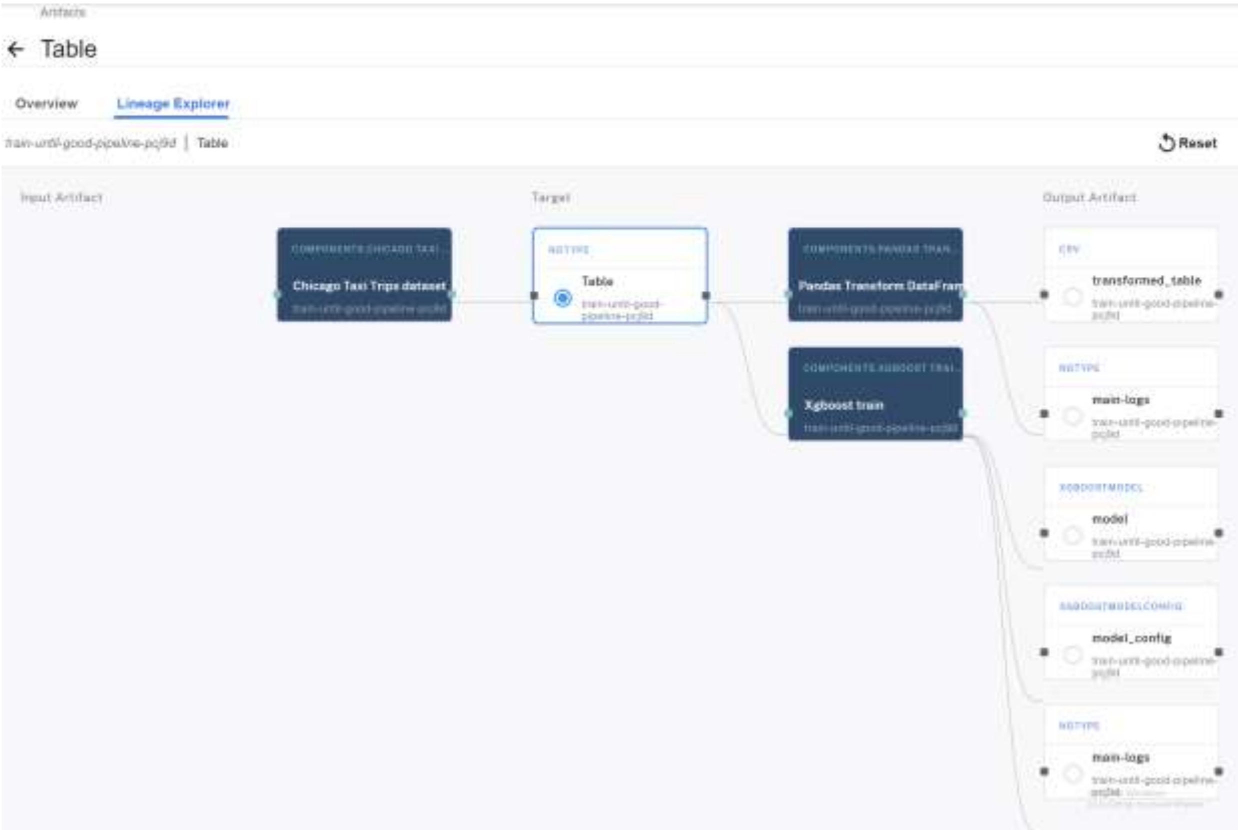
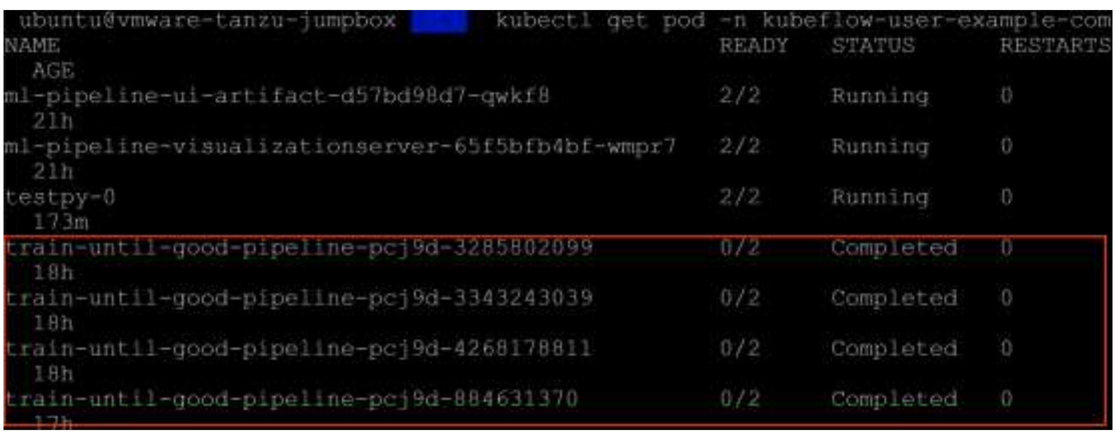


Figure 31: Artifacts for a Pipeline Running Log Lineage Explorer

From the Kubeflow-user-example-com namespace, we can also see the pipeline pod's status is completed.



```
ubuntu@vmware-tanzu-jumpbox      kubectl get pod -n kubeflow-user-example-com
NAME                                              READY   STATUS      RESTARTS
  AGE
ml-pipeline-ui-artifact-d57bd98d7-qwkf8           2/2     Running     0
  21h
ml-pipeline-visualizationserver-65f5bfb4bf-wmpr7  2/2     Running     0
  21h
testpy-0                                          2/2     Running     0
  173m
train-until-good-pipeline-pcj9d-3285802099        0/2     Completed   0
  18h
train-until-good-pipeline-pcj9d-3343243039        0/2     Completed   0
  18h
train-until-good-pipeline-pcj9d-4268178811        0/2     Completed   0
  18h
train-until-good-pipeline-pcj9d-884631370         0/2     Completed   0
  17h
```

Figure 32: Pipeline Pods Status Become Completed

For more details, see Kubeflow pipeline introduction.

KServe Inference Example

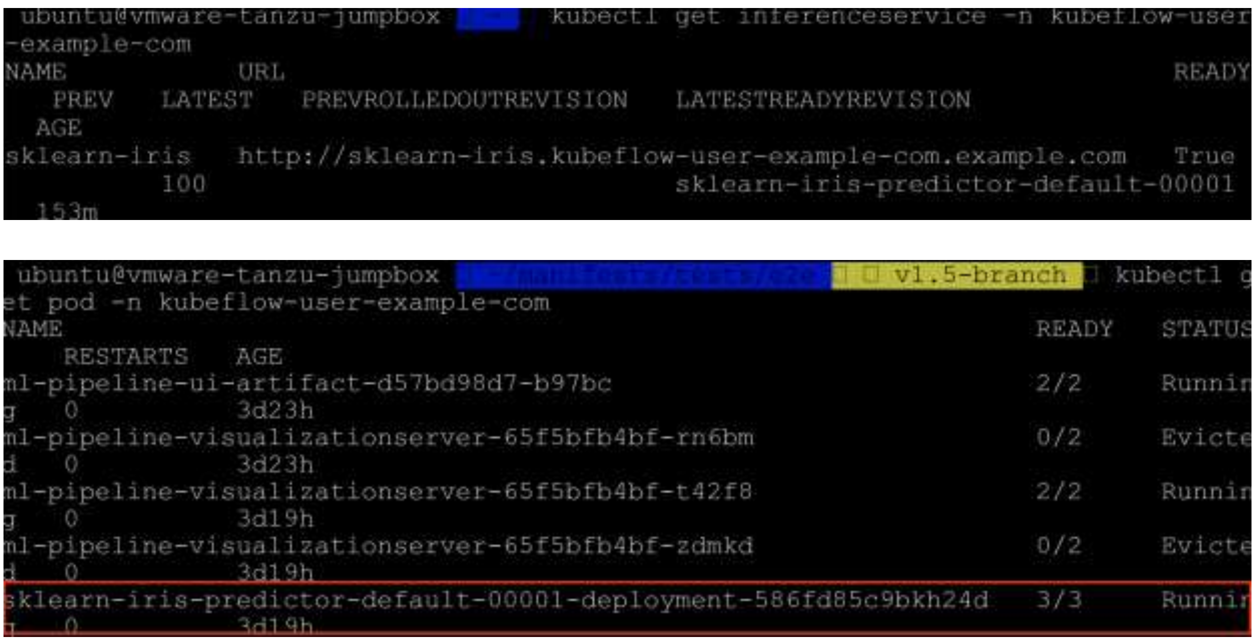
KServe enables serverless inferencing on Kubernetes and provides performant, high abstraction interfaces for common machine learning frameworks like TensorFlow, XGBoost, scikit-learn, PyTorch, and ONNX to handle production model serving use cases. For more details, visit the KServe website.

KServe provides a simple Kubernetes CRD to allow deploying single or multiple trained models onto model servers such as TFServing, TorchServe, ONNXRuntime, and Triton Inference Server. See samples for more information.

We validated the basic inference service which loads a simple iris machine learning model, sends a list of attributes, and prints the prediction for the class of iris plant, see the YAML file.

```
kubectl apply -f isvc.yaml -n kubeflow-user-example-com
```

The inference service will be ready as the figure shows.



```
ubuntu@vmware-tanzu-jumpbox  ~  kubectl get inferenceservice -n kubeflow-user
-example-com
NAME            URL                                                          READY
    PREV    LATEST    PREVROLLEDOUTREVISION    LATESTREADYREVISION
  AGE
sklearn-iris    http://sklearn-iris.kubeflow-user-example-com.example.com    True
          100                                  sklearn-iris-predictor-default-00001
  153m
```

```
ubuntu@vmware-tanzu-jumpbox          v1.5-branch  kubectl g
et pod -n kubeflow-user-example-com
NAME                                                              READY    STATUS
    RESTARTS    AGE
ml-pipeline-ui-artifact-d57bd98d7-b97bc                           2/2      Runnin
g    0          3d23h
ml-pipeline-visualizationserver-65f5bfb4bf-rn6bm                  0/2      Evicte
d    0          3d23h
ml-pipeline-visualizationserver-65f5bfb4bf-t42f8                  2/2      Runnin
g    0          3d19h
ml-pipeline-visualizationserver-65f5bfb4bf-zdmkd                  0/2      Evicte
d    0          3d19h
sklearn-iris-predictor-default-00001-deployment-586fd85c9bkh24d   3/3      Runnin
g    0          3d19h
```

Figure 33: inference Service Becomes Ready

You can also check the inference service from the **Model Servers** tab in **Figure 13.**

Figure 34: Deploy the Inference Service in the Model Servers Tab

Figure 35 is the screenshot of Model server details including service URL, Storage URI, and Predictor type.

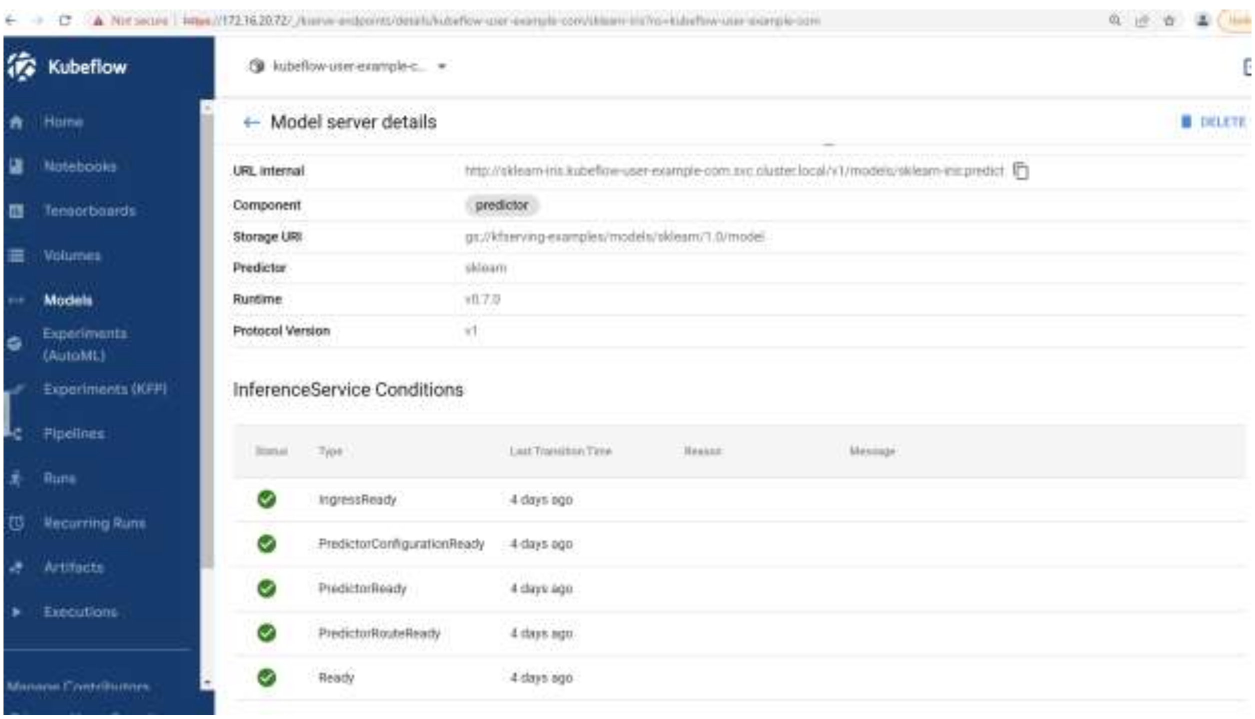


Figure 36: Inference Service Details

Figure 15 is an example [notebook](#) to do prediction using the deployed inference service.

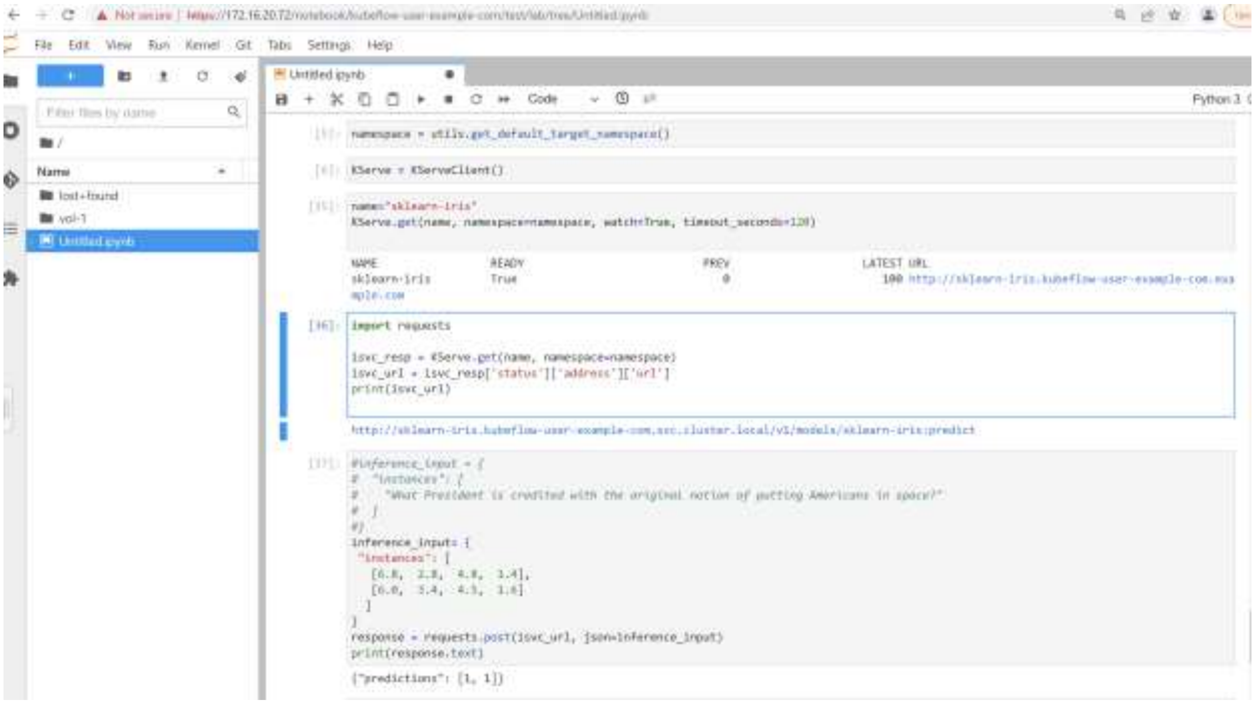


Figure 37: Call Inference Service for Prediction Example

Check the examples running [KServe on Istio/Dex](#) to access the endpoint outside the cluster.

End-to-End Pipeline Example

We validated an integrated [MNIST end-2-end pipeline test](#) to perform the following tasks:

- Hyperparameter tuning using Katib
- Distributive training with the best hyperparameters using TFJob
- Serve the trained model on local pvc using KServe

Before validation, make sure to set a default storageclass and install the python libraries in the requirements.txt, also you can set the parameters in the settings.py.

As shown in **Figure 38,** start the pipeline ./runner.sh

```
ubuntu@vmware-tanzu-jumpbox  ~/manifests/tests/e2e   v1.5-branch ±  ./runner.sh
Installing necessary RBAC.
role.rbac.authorization.k8s.io/pipeline-runner unchanged
rolebinding.rbac.authorization.k8s.io/sa-pipeline-runner unchanged
serviceaccount/pipeline-runner unchanged
rolebinding.rbac.authorization.k8s.io/user-pipeline-runner configured
Setting up port-forward...
Started Istio port-forward, pid: 2004
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Started Pipelines port-forward, pid: 2013
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Running the tests.
Handling connection for 3000
Handling connection for 3000
Run ID:  ffeca68f-cad6-44f5-a4aa-0deb65395ef0
Waiting for mnist-e2e-exp experiments.kubeflow.org to get created...
```

Figure 39: Kickoff the E2E Pipeline

We can monitor the pipeline running from the central dashboard.

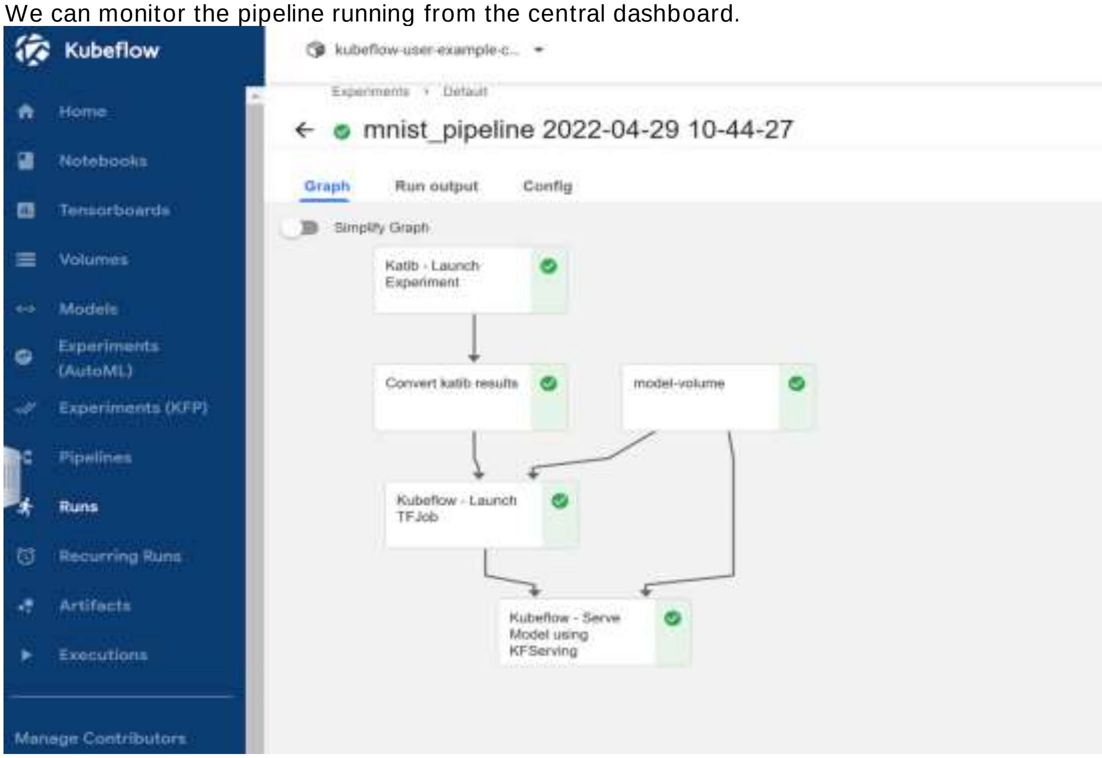


Figure 40: E2E mnist Pipeline Graph

Figure 41 shows the component running steps.

The first step is the Experiments to tune Hyperparameter using Katib. The Experiment uses a "random" algorithm and TFJob for the Trial's worker.

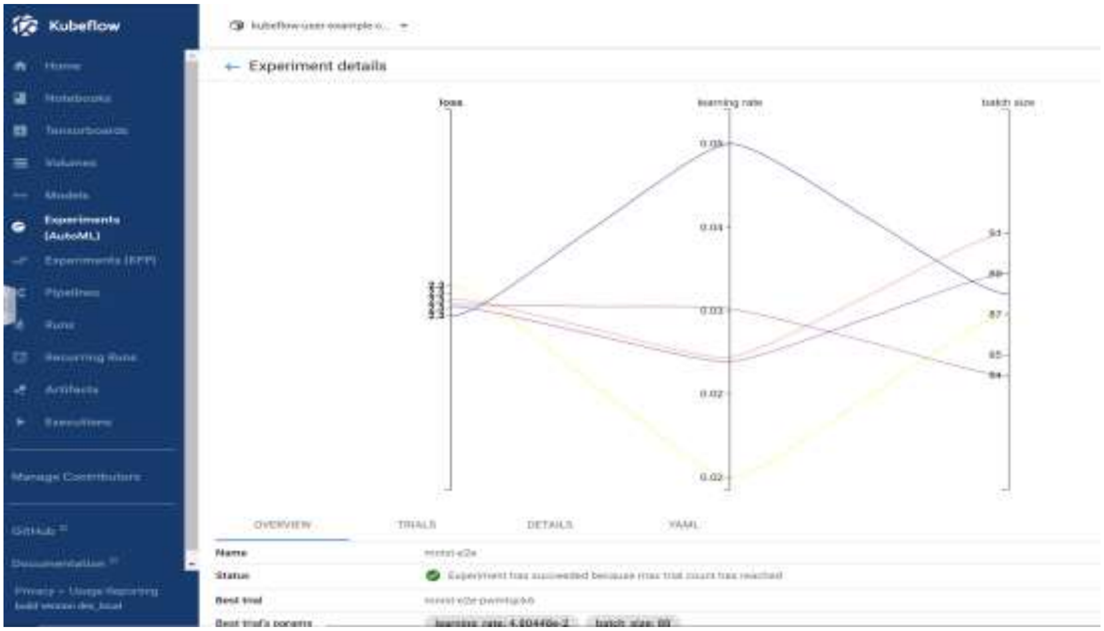


Figure 42: Katib AutoML Hyperparameter Tuning

Then the pipeline created a pvc to store the model. Next is the TFJob runs the Chief and Worker with 1 replica, and last is serving the model using the KServe inference service. And the pipeline runs status changed from running to success.

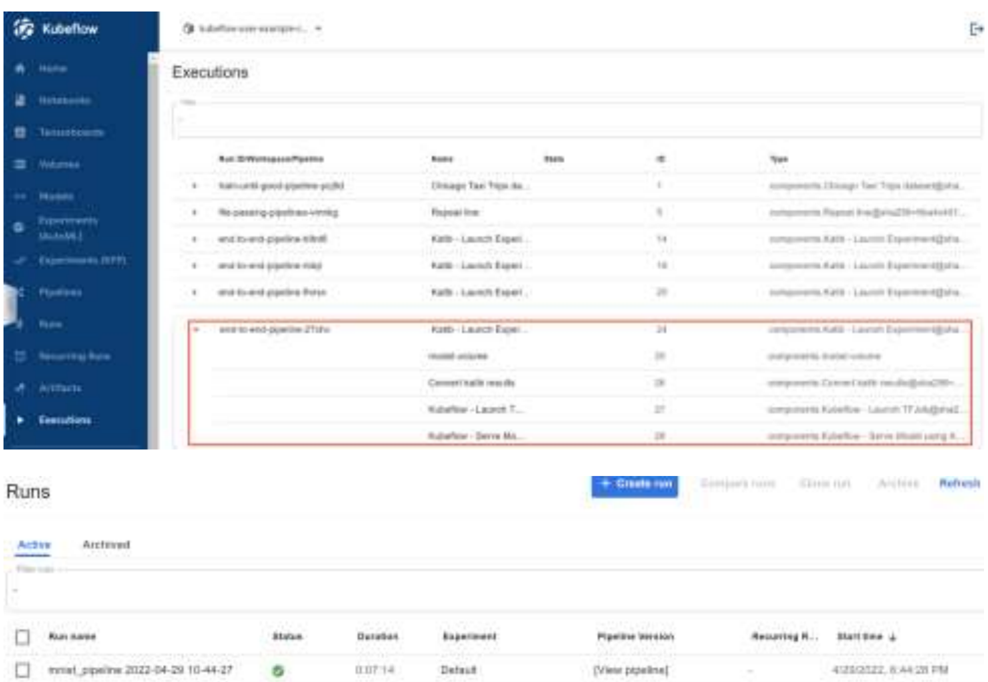


Figure 43: Pipeline Execution Steps and Status

Best Practices

The following recommendations provide the best practices and sizing guidance to run Kubeflow on the AI-Ready platform on vSphere 7 with Tanzu.

- Tanzu Kubernetes Grid:
  - Start with a smaller size of Tanzu Kubernetes cluster with fewer GPU worker nodes, since Kubeflow component pods do not consume GPU resources, and limited CPU resources. The NVIDIA GPU Operator automatically manages newly added GPU worker nodes. We can dynamically resize a Tanzu Kubernetes cluster with more GPU worker nodes and non-GPU worker nodes if there are more workloads running in the system.
  - Customize and pre-allocate enough CPU and memory resources for the Tanzu Kubernetes cluster. Refer to Performance Best Practices for Kubernetes with VMware Tanzu for sizing guidance for Tanzu Kubernetes Grid.
- vSAN Storage:
  - Using the vSAN file service for ReadWriteMany Persistent Volumes can easily scale out the file share and the security, failure tolerance, performance, and capacity-saving features. This architecture can also be easily balanced by manipulating the storage policy of the file share.
  - Failures to Tolerate (FTT) is recommended to set to 1 failure – RAID 1 (Mirroring), if considering space saving, use RAID 5, use stripe policy for a large file share.
  - Enable vSAN Trim/Unmap to allow space reclamation for persistent volumes.
- Kubeflow:
  - Use the latest stable version and match the Kubernetes cluster version and related tools version.
  - Request enough CPU and RAM resources for notebooks or pods to run machine learning workload if the workload is resource intensive.
  - If Kubeflow is deployed in a restricted internet access environment, it is recommended to use a private registry.
  - For GPU-enabled jobs, the CUDA version may not be compatible, so you may need to build a matching image for your cluster.
  - Kubeflow is a loosely-coupled platform. You can use individual components to serve your specific needs in the machine learning workflow.

Additional Resources

For more information about Kubeflow on AI-Ready Enterprise Platform on VMware vSphere 7 with VMware Tanzu Kubernetes grid, explore the following resources:

- VMware vSphere
- VMware vSAN
- VMware Tanzu Kubernetes Grid
- vSphere AI/ML solutions
- Kubeflow docs
- https://github.com/kubeflow

About the Author and Contributors

Ting Yin, Senior Technical Marketing Architect in the Workload Technical Marketing Team of the Cloud Infrastructure Big Group, wrote the original version of this paper. The following reviewers also contributed to the paper contents:

- *Justin Murray, Staff Technical Marketing Architect in VMware*
- *Ka Kit Wong, Staff Technical Marketing Architect in VMware*
- *Chen Wei, Senior Manager of Workload Technical Marketing in VMware*
- *Catherine Xu, Workload Technical Marketing Manager in VMware*