Technical Report

# AI at Scale with Trident, Kubernetes, and Kubeflow

## Execute AI Workloads at Scale with Trident, Kubernetes, and Kubeflow

Mike Oglesby, NetApp
October 2019 | TR-4798


## Abstract

In today's digital economy, artificial intelligence (AI) is becoming critical to business success. As organizations increase their use of AI, they face two major challenges: data availability and workload scalability. This document demonstrates how you can overcome these challenges by using Kubeflow running on Kubernetes as a platform to execute AI workloads and NetApp® Trident to provide seamless access to persistent data across nodes and regions. We also walk through the setup of a Kubernetes and Trident environment for AI, including the deployment of Kubeflow, and provide examples and demonstrations of Kubernetes-based AI jobs.


NetApp®

## TABLE OF CONTENTS



## LIST OF TABLES



## LIST OF FIGURES



# 1 Introduction

In today's digital economy, artificial intelligence (AI) is becoming increasingly critical to business success. Two of the major challenges that organizations face as they adopt AI are data availability and workload scalability. This document describes how you can meet these challenges by using Kubeflow running on Kubernetes and NetApp Trident. Furthermore, this report walks you through the setup of a Kubernetes and Trident environment for AI, including the deployment of Kubeflow, and includes examples and demonstrations of Kubernetes-based AI jobs. Kubeflow makes it simple to deploy and scale AI workloads across multiple GPUs and nodes, and NetApp Trident provides seamless access to persistent data across nodes and regions. With Trident, you can quickly and easily make data volumes, potentially containing petabytes of data, available to Kubernetes-based workloads. Additionally, Trident is a Kubernetes-native app. Trident allows users and administrators to provision and manage storage using standard Kubernetes tools and APIs; no NetApp or NetApp ONTAP® expertise is required.

# 2 Concepts and Components

## 2.1 Artificial Intelligence

AI is a computer science discipline in which computers are trained to mimic the cognitive functions of the human mind. AI aims to train computers to learn and to solve problems in a manner that is similar to, or even superior to, humans. Deep learning (DL) and machine learning (ML) are subfields of AI. Organizations are increasingly adopting AI, ML, and DL to support their critical business needs. Some examples are as follows:

- Analyzing large amounts of data to unearth previously unknown business insights
- Interacting directly with customers by using natural language processing
- Automating various business processes and functions

Modern AI training and inference workloads require massively parallel computing capabilities. Therefore, GPUs are increasingly being used to execute AI operations because the parallel processing capabilities of GPUs are vastly superior to those of general-purpose CPUs.

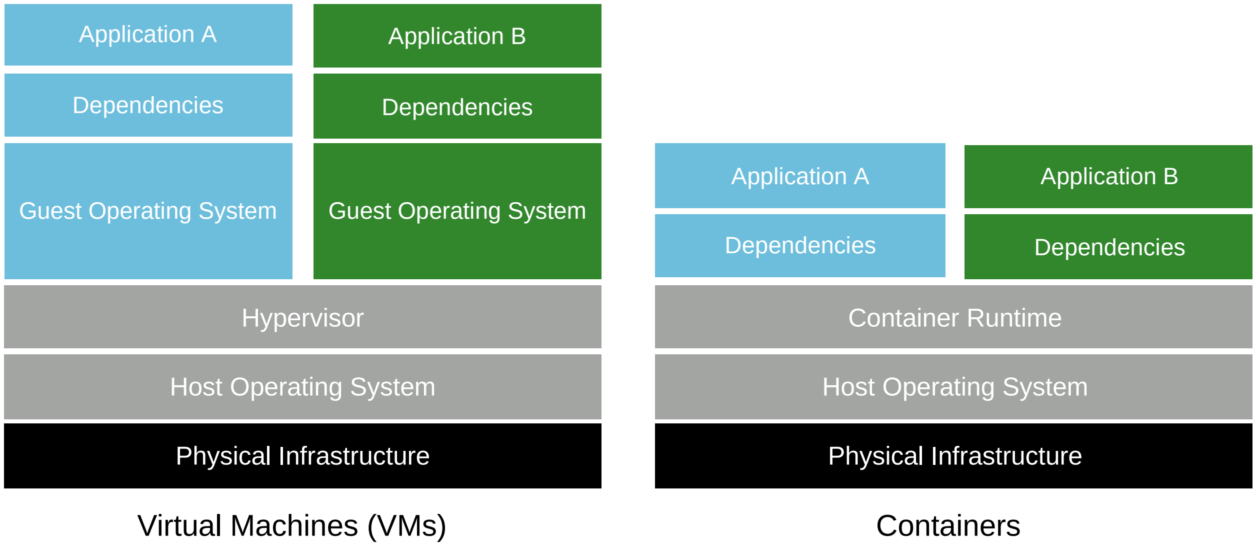
## 2.2 Containers

Containers are isolated user-space instances that run on top of a shared host operating system kernel. The adoption of containers is increasing rapidly. Containers offer many of the same application sandboxing benefits that virtual machines (VMs) offer. However, because the hypervisor and guest operating system layers that VMs rely on have been eliminated, containers are far more lightweight. See Figure 1 for a visualization.

Containers also allow the efficient packaging of application dependencies, run times, and so on, directly with an application. The most commonly used container packaging format is the Docker container. An application that has been containerized in the Docker container format can be executed on any machine that can run Docker containers. This is true even if the application's dependencies are not present on the machine because all dependencies are packaged in the container itself. For more information, visit the Docker website.

**Figure 1) VMs versus containers.**




## 2.3 Kubernetes

Kubernetes is an open-source, distributed, container orchestration platform that was originally designed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes enables the automation of deployment, management, and scaling functions for containerized applications. In recent years, Kubernetes has emerged as the dominant container orchestration platform. Although other container packaging formats and run times are supported, Kubernetes is most often used as an orchestration system for Docker containers. For more information, visit the Kubernetes website.

## 2.4 NetApp Trident

Trident is an open source storage orchestrator developed and maintained by NetApp that greatly simplifies the creation, management, and consumption of persistent storage for Kubernetes workloads. Trident, itself a Kubernetes-native application, runs directly within a Kubernetes cluster. With Trident, Kubernetes users (developers, data scientists, Kubernetes administrators, and so on) can create, manage, and interact with persistent storage volumes in the standard Kubernetes format that they are already familiar with. At the same time, they can take advantage of NetApp advanced data management capabilities and a data fabric that is powered by NetApp technology. Trident abstracts away the complexities of persistent storage and makes it simple to consume. For more information, visit the Trident website.

## 2.5 NVIDIA DeepOps

DeepOps is an open source project from NVIDIA that, by using Ansible, automates the deployment of GPU server clusters according to best practices. DeepOps is modular and can be used for various deployment tasks. For this document and the validation exercise that it describes, DeepOps is used to deploy a Kubernetes cluster that consists of GPU server worker nodes. For more information, visit the DeepOps website.

## 2.6 Kubeflow

Kubeflow is an open source AI and ML toolkit for Kubernetes that was originally developed by Google. The Kubeflow project seeks to make deployments of AI/ML workflows on Kubernetes simple, portable, and scalable. Kubeflow abstracts away the intricacies of Kubernetes, allowing data scientists to focus on what they know best—data science. See Figure 2 for a visualization. Kubeflow has been gaining significant traction as enterprise IT departments have increasingly standardized on Kubernetes. For more information, visit the [Kubeflow website](https://www.kubeflow.org/).

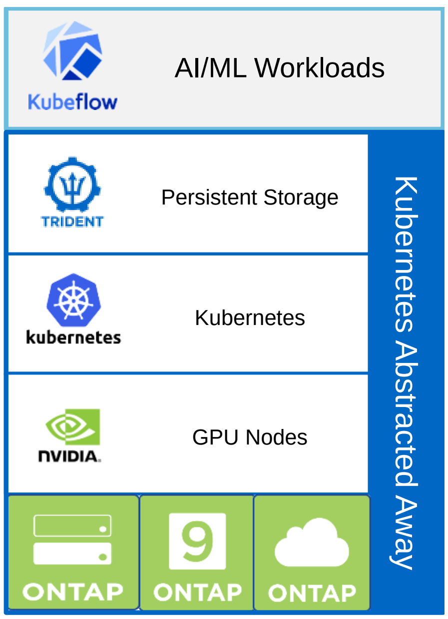
### Kubeflow Pipelines

Kubeflow Pipelines are a key component of Kubeflow. Kubeflow Pipelines are a platform and standard for defining and deploying portable and scalable AI and ML workflows. For more information, see the official Kubeflow documentation.

### Jupyter Notebook Server

A Jupyter Notebook Server is an open source web application that allows data scientists to create wiki-like documents called Jupyter Notebooks that contain live code as well as descriptive test. Jupyter Notebooks are widely used in the AI/ML community as a means of documenting, storing, and sharing AI and ML projects. Kubeflow simplifies the provisioning and deployment of Jupyter Notebook Servers on Kubernetes. For more information on Jupyter Notebooks, visit the Jupyter website. For more information about Jupyter Notebooks within the context of Kubeflow, see the official Kubeflow documentation.

**Figure 2) Kubeflow visualization.**




## 2.7 NetApp ONTAP 9

NetApp ONTAP 9 is the latest generation of storage management software from NetApp that enables businesses like yours to modernize infrastructure and to transition to a cloud-ready data center. With industry-leading data management capabilities, ONTAP enables you to manage and protect your data with a single set of tools regardless of where that data resides. You can also move data freely to wherever you need it: the edge, the core, or the cloud. ONTAP 9 includes numerous features that simplify data management, accelerate and protect your critical data, and future-proof your infrastructure across hybrid cloud architectures.

### Simplify Data Management

Data management is crucial for your enterprise IT operations so that you can use appropriate resources for your applications and datasets. ONTAP includes the following features to streamline and simplify your operations and reduce your total cost of operation:

- **Inline data compaction and expanded deduplication.** Data compaction reduces wasted space inside storage blocks, and deduplication significantly increases effective capacity.
- **Minimum, maximum, and adaptive quality of service (QoS).** Granular QoS controls help maintain performance levels for critical applications in highly shared environments.
- **ONTAP FabricPool.** This feature provides automatic tiering of cold data to public and private cloud storage options, including Amazon Web Services (AWS), Azure, and NetApp StorageGRID® object-based storage.

### Accelerate and Protect Data

ONTAP delivers superior levels of performance and data protection and extends these capabilities with the following features:

- **High performance and low latency.** ONTAP offers the highest possible throughput at the lowest possible latency.
- **NetApp ONTAP FlexGroup technology.** A FlexGroup volume is a high-performance data container that can scale linearly to up to 20PB and 400 billion files, providing a single namespace that simplifies data management.
- **Data protection.** ONTAP provides built-in data protection capabilities with common management across all platforms.
- **NetApp Volume Encryption.** ONTAP offers native volume-level encryption with both onboard and external key management support.

### Future-Proof Infrastructure

ONTAP 9 helps meet your demanding and constantly changing business needs:

- **Seamless scaling and nondisruptive operations.** ONTAP supports the nondisruptive addition of capacity to existing controllers and to scale-out clusters. You can upgrade to the latest technologies, such as NVMe and 32Gb FC, without costly data migrations or outages.
- **Cloud connection.** ONTAP is one of the most cloud-connected storage management software, with options for software-defined storage (ONTAP Select) and cloud-native instances (NetApp Cloud Volumes Service) in all public clouds.
- **Integration with emerging applications.** By using the same infrastructure that supports existing enterprise apps, ONTAP offers enterprise-grade data services for next-generation platforms and applications such as OpenStack, Hadoop, and MongoDB.

## 2.8 NetApp ONTAP FlexGroup Volumes

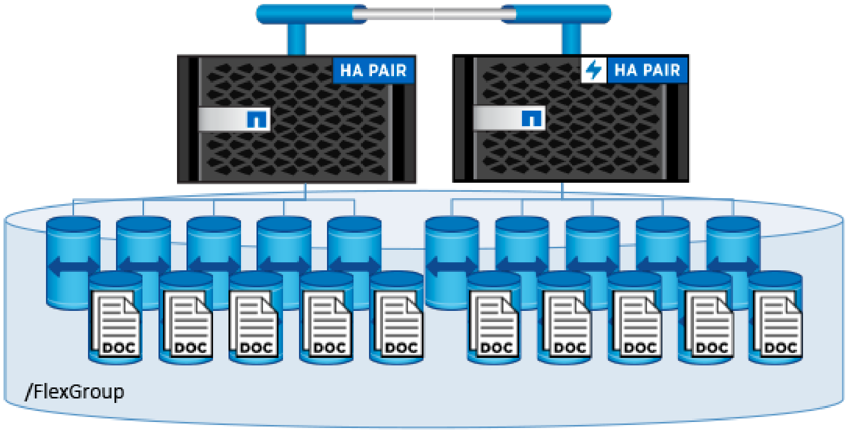
A training dataset can be a collection of potentially billions of files. Files can include text, audio, video, and other forms of unstructured data that must be stored and processed to be read in parallel. The storage system must store large numbers of small files and must read those files in parallel for sequential and random I/O.

A FlexGroup volume (Figure 3) is a single namespace that comprises multiple constituent member volumes. From a storage administrator viewpoint, a FlexGroup volume is managed and acts like a NetApp FlexVol® volume. Files in a FlexGroup volume are allocated to individual member volumes and are not striped across volumes or nodes. They enable the following capabilities:

- FlexGroup volumes provide multiple petabytes of capacity and predictable low latency for high-metadata workloads.
- They support up to 400 billion files in the same namespace.
- They support parallelized operations in NAS workloads across CPUs, nodes, aggregates, and constituent FlexVol volumes.

**Figure 3) NetApp FlexGroup volumes.**




## 3 Validation Environment

All configuration and validation procedures that are outlined in this document were performed on the NetApp ONTAP AI converged infrastructure solution. For more details on the ONTAP AI architecture, see NVA-1121. For this exercise, two bare-metal NVIDIA DGX-1 servers, each featuring eight NVIDIA GPUs, were used as Kubernetes worker nodes. A NetApp AFF A800 all-flash storage system provided a single persistent storage namespace across nodes, and two Cisco Nexus 3232C switches were used to provide network connectivity. Three VMs that ran on a separate physical server outside of the ONTAP AI pod were used as Kubernetes master nodes. See Table 1 for infrastructure details. See Table 2 for software version details.

**Table 1) Infrastructure details.**

| Component | Quantity | Details | Operating System |
|---|---|---|---|
| Deployment jump host | 1 | VM | Ubuntu 18.04.2 LTS |
| Kubernetes master nodes | 3 | VM | Ubuntu 18.04.2 LTS |
| Kubernetes worker nodes | 2 | NVIDIA DGX-1 (bare-metal) | NVIDIA DGX OS 4.0.5 (based on Ubuntu 18.04.2 LTS) |
| Storage | 1 | NetApp AFF A800 | NetApp ONTAP 9.5 P1 |
| Network connectivity | 2 | Cisco Nexus 3232C | Cisco NX-OS 7.0(3)I6(1) |

**Table 2) Software version details.**

| Component | Version |
|---|---|
| NVIDIA DeepOps | Pulled from GitHub repository (https://github.com/NVIDIA/deepops) on August 30, 2019 |
| NVIDIA DGX OS | 4.0.5 (based on Ubuntu 18.04.2 LTS) |
| Ubuntu | 18.04.2 LTS |
| Docker | 18.09.5-ce |
| Kubernetes | 1.14.3 |

| Component | Version |
|---|---|
| NetApp ONTAP | 9.5 P1 |
| NetApp Trident | 19.07 |
| Cisco NX-OS | 7.0(3)I6(1) |

# 4 Kubernetes Environment Configuration and Example Operations

This section describes the tasks that you must complete to configure a Kubernetes and Trident environment for scalable AI in the validation environment that is described in Section 3.

An NVIDIA DGX-1 server and a NetApp AFF A800 system were used for this validation exercise. However, the tasks that are outlined in this section should apply to any environment that contains a NetApp ONTAP appliance or instance. Examples include a NetApp AFF storage appliance, a NetApp ONTAP Select software-defined storage instance, or a NetApp Cloud Volumes ONTAP instance running in the cloud. The NetApp instance can be paired with servers or with instances that feature NVIDIA GPUs, including white-box servers that feature NVIDIA GPUs or cloud-compute instances that feature NVIDIA GPUs.

## 4.1 Prerequisites

Before you perform the configuration exercises that are outlined in this section, we assume that you have already performed the following tasks:

1. You have already configured the ONTAP appliance or instance and GPU servers or instances (Kubernetes worker nodes) according to their respective standard deployment instructions.

   **Note:** For the validation exercise that is described in this document, the NetApp AFF A800 storage appliance and NVIDIA DGX-1 servers have been configured according to the ONTAP AI converged infrastructure solution guidelines. See NVA-1121 for ONTAP AI deployment details.

2. You have installed a supported operating system on all Kubernetes master and worker nodes and on the deployment jump host. As of the time of writing, NVIDIA DeepOps supports the following Linux distributions:
   – NVIDIA DGX OS 4
   – Ubuntu 18.04 LTS
   – CentOS 7

   **Note:** For this validation exercise, NVIDIA DGX OS 4.0.5 was installed on the Kubernetes worker nodes according to the ONTAP AI converged infrastructure solution guidelines (see NVA-1121). Ubuntu 18.04.2 LTS was installed on the Kubernetes master nodes and deployment jump host.

## 4.2 Use NVIDIA DeepOps to Install and Configure Kubernetes

To deploy and configure your Kubernetes cluster with NVIDIA DeepOps, perform the following tasks on the deployment jump host:

1. Clone the NVIDIA DeepOps GitHub repository.

```
$ git clone https://github.com/NVIDIA/deepops
Cloning into 'deepops'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 6048 (delta 3), reused 1 (delta 0), pack-reused 6039
```

```
Receiving objects: 100% (6048/6048), 7.36 MiB | 21.83 MiB/s, done.
Resolving deltas: 100% (3498/3498), done.
$ cd ./deepops
```

2. Deploy Kubernetes in your cluster by following the instructions on the [Kubernetes Deployment Guide page] on the NVIDIA DeepOps GitHub site.

   **Note:** For the DeepOps Kubernetes deployment to work, the same user must exist on all Kubernetes master and worker nodes. Additionally, NetApp recommends that you set up passwordless Secure Shell (SSH) access to all Kubernetes nodes from the deployment jump host before you perform the deployment.

If the deployment fails, change the value of `kubectl_localhost` to `false` in `deepops/config/group_vars/k8s-cluster.yml` and repeat step 2. The `Copy kubectl binary to ansible host` task, which executes only when the value of `kubectl_localhost` is `true`, relies on the `fetch` Ansible module, which has known memory usage issues. These memory usage issues can sometimes cause the task to fail. If the task fails because of a memory issue, then the remainder of the deployment operation does not complete successfully.

   If the deployment completes successfully after you have changed the value of `kubectl_localhost` to `false`, then you must manually copy the kubectl binary from a Kubernetes master node to the deployment jump host. You can find the location of the kubectl binary on a specific master node by executing the command `which kubectl` directly on that node.

## 4.3 Install and Configure Trident

To install and configure NetApp Trident in your Kubernetes cluster, perform the following tasks on the deployment jump host:

1. Deploy Trident for Kubernetes in your cluster by following the [deployment instructions] in the Trident documentation.
2. Create a FlexGroup-enabled Trident back end for each data LIF (logical network interface that provides data access) that you want to use on your ONTAP system. The example commands that follow show the creation of two FlexGroup-enabled Trident back ends for two different data LIFs that are associated with the same ONTAP storage virtual machine (SVM). For more information about back ends, see the [Trident documentation].

   Due to NFS protocol limitations, a single NFS mount can provide only 1.5GBps to 2GBps of bandwidth. If you need more bandwidth for a job, Trident enables you to add multiple NFS mounts (mounting the same NFS volume multiple times) quickly and easily when you create a Kubernetes pod. For maximum performance, these multiple mounts should be distributed across different data LIFs. You must create a Trident back end for each data LIF that you want to use for these mounts. This example assumes that future AI and ML jobs use two mounts; therefore, it shows the creation of two Trident back ends that are distributed across two different data LIFs.

   The example commands that follow show the creation of two Trident back ends that use the `ontap-nas-flexgroup` storage driver. ONTAP supports two main data volume types: FlexVol and FlexGroup. FlexVol volumes are size-limited (as of this writing, the maximum size depends on the specific deployment). FlexGroup volumes, on the other hand, can scale linearly to up to 20PB and 400 billion files, providing a single namespace that greatly simplifies data management. Therefore, FlexGroup volumes are optimal for AI and ML workloads that rely on large amounts of data.

   If you are working with a small amount of data and want to use FlexVol volumes instead of FlexGroup volumes, you can create Trident back ends that use the `ontap-nas` storage driver instead of the `ontap-nas-flexgroup` storage driver.

```
$ cat << EOF > ./trident-backend-ontap-ai-flexgroups-iface1.json
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "backendName": "ontap-ai-flexgroups-iface1",
    "managementLIF": "10.61.218.100",
```

```
    "dataLIF": "192.168.11.11",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-ontap-ai-flexgroups-iface1.json -n trident
+----------------------------+---------------------+--------------------------------------+------
--+---------+
|            NAME            |   STORAGE DRIVER    |                 UUID                 | STATE
| VOLUMES |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
| ontap-ai-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd |
online |       0 |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
$ cat << EOF > ./trident-backend-ontap-ai-flexgroups-iface2.json
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "backendName": "ontap-ai-flexgroups-iface2",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.12.12",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-ontap-ai-flexgroups-iface2.json -n trident
+----------------------------+---------------------+--------------------------------------+------
--+---------+
|            NAME            |   STORAGE DRIVER    |                 UUID                 | STATE
| VOLUMES |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
| ontap-ai-flexgroups-iface2 | ontap-nas-flexgroup | 61814d48-c770-436b-9cb4-cf7ee661274d |
online |       0 |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
$ tridentctl get backend -n trident
+----------------------------+---------------------+--------------------------------------+------
--+---------+
|            NAME            |   STORAGE DRIVER    |                 UUID                 | STATE
| VOLUMES |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
| ontap-ai-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd |
online |       0 |
| ontap-ai-flexgroups-iface2 | ontap-nas-flexgroup | 61814d48-c770-436b-9cb4-cf7ee661274d |
online |       0 |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
```

3. **Optional:** Create one or more FlexVol-enabled Trident back ends. If you use FlexGroup volumes for training dataset storage, you might want to use FlexVol volumes for storing results, output, debug information, and so on. If you want to use FlexVol volumes, you must create one or more FlexVol-enabled Trident back ends. The example commands that follow show the creation of a single FlexVol-enabled Trident back end that uses a single data LIF.

```
$ cat << EOF > ./trident-backend-ontap-ai-flexvols.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "ontap-ai-flexvols",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.11.11",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
```

```
}
EOF
$ tridentctl create backend -f ./trident-backend-ontap-ai-flexvols.json -n trident
+------------------------------+---------------------+--------------------------------------+------
--+---------+
|             NAME             |   STORAGE DRIVER    |                 UUID                 | STATE
| VOLUMES |
+------------------------------+---------------------+--------------------------------------+------
--+---------+
| ontap-ai-flexvols            | ontap-nas           | 52bdb3b1-13a5-4513-a9c1-52a69657fabe |
online |       0 |
+------------------------------+---------------------+--------------------------------------+------
--+---------+
$ tridentctl get backend -n trident
+------------------------------+---------------------+--------------------------------------+------
--+---------+
|             NAME             |   STORAGE DRIVER    |                 UUID                 | STATE
| VOLUMES |
+------------------------------+---------------------+--------------------------------------+------
--+---------+
| ontap-ai-flexvols            | ontap-nas           | 52bdb3b1-13a5-4513-a9c1-52a69657fabe |
online |       0 |
| ontap-ai-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd |
online |       0 |
| ontap-ai-flexgroups-iface2 | ontap-nas-flexgroup | 61814d48-c770-436b-9cb4-cf7ee661274d |
online |       0 |
+------------------------------+---------------------+--------------------------------------+------
--+---------+
```

## 4.4 Create Kubernetes StorageClasses

To administer ONTAP volumes by using Kubernetes, you must create Kubernetes StorageClasses. To create the StorageClasses that you need, perform the following tasks on the deployment jump host:

1. Create a StorageClass that corresponds to each FlexGroup-enabled Trident back end that you created in section 4.3, step 2. These granular StorageClasses enable you to add NFS mounts that correspond to specific LIFs (the LIFs that you specified when you created the Trident back ends) as a particular back end that is specified in the StorageClass spec file. The example commands that follow show the creation of two StorageClasses that correspond to the two example back ends that were created in section 4.3, step 2. The highlighted text shows where the Trident back end is specified in the StorageClass spec file. For more information about StorageClasses, see the official Kubernetes documentation.

   **Note:** So that a persistent volume isn't deleted when the corresponding PersistentVolumeClaim (PVC) is deleted, the following example uses a `reclaimPolicy` value of `Retain`. For more information about the `reclaimPolicy` field, see the official Kubernetes documentation.

```
$ cat << EOF > ./storage-class-ontap-ai-flexgroups-retain-iface1.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexgroups-retain-iface1
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas-flexgroup"
  storagePools: "ontap-ai-flexgroups-iface1:.*"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexgroups-retain-iface1.yaml
storageclass.storage.k8s.io/ontap-ai-flexgroups-retain-iface1 created
$ cat << EOF > ./storage-class-ontap-ai-flexgroups-retain-iface2.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexgroups-retain-iface2
provisioner: netapp.io/trident
parameters:
```

```
  backendType: "ontap-nas-flexgroup"
  storagePools: "ontap-ai-flexgroups-iface2:.*"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexgroups-retain-iface2.yaml
storageclass.storage.k8s.io/ontap-ai-flexgroups-retain-iface2 created
$ kubectl get storageclass
NAME                                PROVISIONER         AGE
ontap-ai-flexgroups-retain-iface1   netapp.io/trident   0m
ontap-ai-flexgroups-retain-iface2   netapp.io/trident   0m
```

2. **Optional:** Create a StorageClass that corresponds to the FlexVol-enabled Trident back end that you created in section 4.3, step 3. The example commands that follow show the creation of a single StorageClass for FlexVol volumes.

   **Note:** In the following example, a particular back end is not specified in the StorageClass spec file because only one FlexVol-enabled Trident back end was created in section 4.3, step 3. When you use Kubernetes to administer volumes that use this StorageClass, Trident attempts to use any available back end that uses the `ontap-nas` driver.

```
$ cat << EOF > ./storage-class-ontap-ai-flexvols-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexvols-retain
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexvols-retain.yaml
storageclass.storage.k8s.io/ontap-ai-flexvols-retain created
$ kubectl get storageclass
NAME                                PROVISIONER         AGE
ontap-ai-flexgroups-retain-iface1   netapp.io/trident   1m
ontap-ai-flexgroups-retain-iface2   netapp.io/trident   1m
ontap-ai-flexvols-retain            netapp.io/trident   0m
```

3. **Optional:** Create a generic StorageClass for FlexGroup volumes. The following example commands show the creation of a single generic StorageClass for FlexGroup volumes. Note that a particular back end is not specified in the StorageClass spec file. Therefore, when you use Kubernetes to administer volumes that use this StorageClass, Trident attempts to use any available back end that uses the `ontap-nas-flexgroup` driver.

```
$ cat << EOF > ./storage-class-ontap-ai-flexgroups-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexgroups-retain
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas-flexgroup"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexgroups-retain.yaml
storageclass.storage.k8s.io/ontap-ai-flexgroups-retain created
$ kubectl get storageclass
NAME                                PROVISIONER         AGE
ontap-ai-flexgroups-retain          netapp.io/trident   0m
ontap-ai-flexgroups-retain-iface1   netapp.io/trident   2m
ontap-ai-flexgroups-retain-iface2   netapp.io/trident   2m
ontap-ai-flexvols-retain            netapp.io/trident   1m
```

## 4.5 Import and Create Data Volumes

This exercise assumes that a FlexGroup volume that contains data to be used by AI and ML jobs already exists. To import this volume into your Kubernetes cluster so that you can interact with it in a Kubernetes-native format, perform the following tasks on the deployment jump host:

1. Use the Trident volume import functionality to create Kubernetes PersistentVolumeClaims (PVCs) for your existing FlexGroup volume that contains data to be used by AI and ML jobs. Create one PVC for each FlexGroup-enabled Trident back end that you created in section 4.3, step 2. This step enables you to mount the same data volume (your existing FlexGroup volume) multiple times across different LIFs, as described in section 4.3, step 2. The example commands that follow show the creation of two PVCs, one corresponding to each back end, for an existing volume named `pb_fg_all`. For more information about PersistentVolumeClaims, see the [official Kubernetes documentation](). For more information about the volume import functionality, see the [Trident documentation]().

   **Note:** An `accessModes` value of `ReadOnlyMany` is specified in the example PVC spec files. This value means that multiple pods can mount these PVCs in read-only mode at the same time. For more information about the `accessMode` field, see the [official Kubernetes documentation]().

   **Note:** The back-end names that are specified in the following example import commands are highlighted for reference. These names should correspond to the back ends that you create in section 4.3, step 2.

   **Note:** The StorageClass names that are specified in the following example PVC spec files are highlighted for reference. These names should correspond to the StorageClasses that you create in section 4.4, step 1.

```
$ cat << EOF > ./pvc-import-pb_fg_all-iface1.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pb-fg-all-iface1
  namespace: default
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexgroups-retain-iface1
EOF
$ tridentctl import volume ontap-ai-flexgroups-iface1 pb_fg_all -f ./pvc-import-pb_fg_all-
iface1.yaml -n trident
+--------------------------------+---------+-----------------------------------+----------+-------
------------------------------------+--------+---------+
|          NAME                  |  SIZE   |       STORAGE CLASS               | PROTOCOL |
BACKEND UUID                         | STATE  | MANAGED |
+--------------------------------+---------+-----------------------------------+----------+-------
-----------------------------------+--------+---------+
| default-pb-fg-all-iface1-7d9f1 | 10 TiB | ontap-ai-flexgroups-retain-iface1 | file     |
b74cbddb-e0b8-40b7-b263-b6da6dec0bdd | online | true    |
+--------------------------------+---------+-----------------------------------+----------+-------
------------------------------------+--------+---------+
$ cat << EOF > ./pvc-import-pb_fg_all-iface2.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pb-fg-all-iface2
  namespace: default
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexgroups-retain-iface2
EOF
$ tridentctl import volume ontap-ai-flexgroups-iface2 pb_fg_all -f ./pvc-import-pb_fg_all-
iface2.yaml -n trident
+--------------------------------+---------+-----------------------------------+----------+-------
------------------------------------+--------+---------+
```

```
|          NAME                        |  SIZE  |       STORAGE CLASS                | PROTOCOL |
BACKEND UUID                             | STATE  | MANAGED |
+---------------------------------+--------+------------------------------------+----------+-------
------------------------------------+--------+---------+
| default-pb-fg-all-iface2-85aee | 10 TiB | ontap-ai-flexgroups-retain-iface2 | file     |
61814d48-c770-436b-9cb4-cf7ee661274d | online | true    |
+---------------------------------+--------+------------------------------------+----------+-------
--------------------------------------+--------+---------+
$ tridentctl get volume -n trident
+-----------------------------------+---------+------------------------------------+----------+----
---------------------------------+--------+---------+
|              NAME                 |  SIZE   |           STORAGE CLASS            | PROTOCOL |
BACKEND UUID                | STATE   | MANAGED |
+-----------------------------------+---------+------------------------------------+----------+----
---------------------------------+--------+---------+
| default-pb-fg-all-iface1-7d9f1    | 10 TiB  | ontap-ai-flexgroups-retain-iface1 | file     |
b74cbddb-e0b8-40b7-b263-b6da6dec0bdd | online | true    |
| default-pb-fg-all-iface2-85aee    | 10 TiB  | ontap-ai-flexgroups-retain-iface2 | file     |
61814d48-c770-436b-9cb4-cf7ee661274d | online | true    |
+-----------------------------------+---------+------------------------------------+----------+----
---------------------------------+--------+---------+
$ kubectl get pvc
NAME                 STATUS   VOLUME                              CAPACITY         ACCESS MODES
STORAGECLASS                        AGE
pb-fg-all-iface1     Bound    default-pb-fg-all-iface1-7d9f1     10995116277760   ROX
ontap-ai-flexgroups-retain-iface1   25h
pb-fg-all-iface2     Bound    default-pb-fg-all-iface2-85aee     10995116277760   ROX
ontap-ai-flexgroups-retain-iface2   25h
```

2. **Optional:** Use Kubernetes and Trident to provision a new FlexVol volume to store results, output, debug information, and so on, by using the StorageClass that you created in section 4.4, step 2. The following example commands show the provisioning of a single new FlexVol volume to store TensorFlow results.

   **Note:** An `accessModes` value of `ReadWriteMany` is specified in the following example PVC spec file. This value means that multiple pods can mount this PVC in read/write mode at the same time. For more information about the `accessMode` field, see the [official Kubernetes documentation](https://kubernetes.io/docs/concepts/storage/persistent-volumes/).

```
$ cat << EOF > ./pvc-tensorflow-results.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tensorflow-results
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-ai-flexvols-retain
EOF
$ kubectl create -f ./pvc-tensorflow-results.yaml
persistentvolumeclaim/tensorflow-results created
$ kubectl get pvc
NAME                          STATUS     VOLUME                              CAPACITY
ACCESS MODES   STORAGECLASS                        AGE
pb-fg-all-iface1              Bound      default-pb-fg-all-iface1-7d9f1     10995116277760
ROX            ontap-ai-flexgroups-retain-iface1   26h
pb-fg-all-iface2              Bound      default-pb-fg-all-iface2-85aee     10995116277760
ROX            ontap-ai-flexgroups-retain-iface2   26h
tensorflow-results            Bound      default-tensorflow-results-2fd60   1073741824
RWX            ontap-ai-flexvols-retain            25h
```

## 4.6 Execute a Single-Node AI Workload

To execute a single-node AI and ML job in your Kubernetes cluster by taking advantage of data that is stored on an ONTAP volume, perform the following tasks on the deployment jump host. With Trident, you can quickly and easily make a data volume, potentially containing petabytes of data, that is available to a Kubernetes workload. To make such a data volume accessible from within a Kubernetes pod, simply specify a PVC, such as one of the PVCs that you created in section 4.5, in the pod specification. This step is a Kubernetes-native operation; no NetApp or ONTAP expertise is required.

**Note:** This section assumes that you have already containerized (in the Docker container format) the specific AI and ML workload that you are attempting to execute in your Kubernetes cluster.

1. Create a Kubernetes job for your containerized AI and ML workload.

   The following example commands show the execution of a TensorFlow benchmark job that uses the ImageNet dataset. For more information about the ImageNet dataset, see the ImageNet website. This example job requests eight GPUs and therefore can run on a single GPU worker node that features eight or more GPUs. Additionally, to provide the required amount of storage bandwidth, the volume that contains the needed training data (the volume that was imported in section 4.5, step 1) is mounted twice within the pod that this job creates. See the highlighted lines in the following job specification.

   The results volume that was created in section 4.5, step 2, is also mounted in the pod. These volumes are referenced in the job specification by using the names of the PVCs that were created in section 4.5. For more information about Kubernetes jobs, see the official Kubernetes documentation.

   See section 4.3, step 2, for details about why you might have to mount the same data volume multiple times. The number of mounts that you need depends on the amount of bandwidth that the specific job requires.

   This example job could be submitted in a cluster for which a worker node featuring eight or more GPUs is not present or is currently occupied with another workload. If so, then the job remains in a pending state until such a worker node becomes available.

   An `emptyDir` volume with a `medium` value of `Memory` is mounted to `/dev/shm` in the pod that this example job creates. The default size of the `/dev/shm` virtual volume that is automatically created by the Docker container run time can sometimes be insufficient for TensorFlow's needs. Mounting an `emptyDir` volume as in the following example provides a sufficiently large `/dev/shm` virtual volume. For more information about `emptyDir` volumes, see the official Kubernetes documentation.

   The single container that is specified in this example job spec is given a `securityContext > privileged` value of `true`. This value means that the container effectively has root access on the host. This annotation is used in this case because the specific workload that is being executed requires root access. Specifically, a clear cache operation that the workload performs requires root access. Whether or not this `privileged: true` annotation is necessary depends on the requirements of the specific workload that you are executing.

```
$ cat << EOF > ./netapp-tensorflow-single-imagenet.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-single-imagenet
spec:
  backoffLimit: 5
  template:
    spec:
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
```

```
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
dataset_dir=/mnt/mount_0/dataset/imagenet", "--dgx_version=dgx1", "--num_devices=8"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-single-imagenet.yaml
job.batch/netapp-tensorflow-single-imagenet created
$ kubectl get jobs
NAME                                       COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet          0/1           24s        24s
```

2. Confirm that the job that you created in step 1 is running correctly. The following example command confirms that a single pod was created for the job, as specified in the job spec, and that this pod is currently running on one of the GPU worker nodes.

```
$ kubectl get pods -o wide
NAME                                             READY   STATUS      RESTARTS   AGE
IP              NODE              NOMINATED NODE
netapp-tensorflow-single-imagenet-m7x92          1/1     Running     0          3m
10.233.68.61    10.61.218.154     <none>
```

3. Confirm that the job that you created in step 1 completes successfully. The following example commands confirm that the job completed successfully.

```
$ kubectl get jobs
NAME                                             COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet                1/1           5m42s      10m
$ kubectl get pods
NAME                                                   READY   STATUS      RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92                0/1     Completed   0          11m
$ kubectl logs netapp-tensorflow-single-imagenet-m7x92
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c
at line 702
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c
at line 711
Total images/sec = 6530.59125
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 1 -H localhost:1 bash -c 'sync; echo 1 > /proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 8 -H localhost:8 -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x
LD_LIBRARY_PATH -x PATH python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --
model=resnet50 --batch_size=256 --device=gpu --force_gpu_compatible=True --num_intra_threads=1 --
num_inter_threads=48 --variable_update=horovod --batch_group_size=20 --num_batches=500 --
nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True --use_tf_layers=False --
data_name=imagenet --use_datasets=True --data_dir=/mnt/mount_0/dataset/imagenet --
datasets_parallel_interleave_cycle_length=10 --datasets_sloppy_parallel_interleave=False --
num_mounts=2 --mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000 --
datasets_use_prefetch=True --datasets_num_private_threads=4 --horovod_device=gpu >
/tmp/20190814_105450_tensorflow_horovod_rdma_resnet50_gpu_8_256_b500_imagenet_nodistort_fp16_r10_
m2_nockpt.txt 2>&1
```

4. **Optional:** Clean up job artifacts. The following example commands show the deletion of the job object that was created in step 1.

**Note:** When you delete the job object, Kubernetes automatically deletes any associated pods.

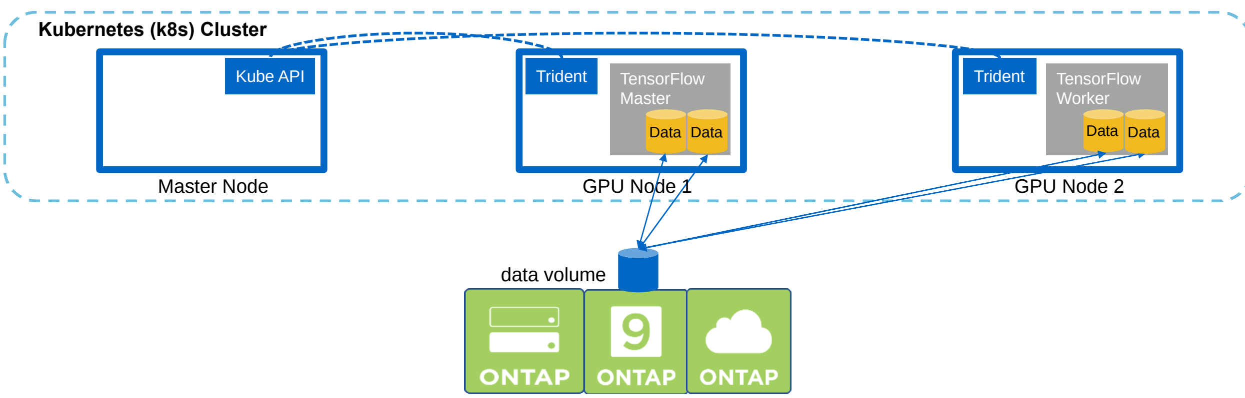
```
$ kubectl get jobs
NAME                                               COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet                  1/1           5m42s      10m
$ kubectl get pods
NAME                                                     READY   STATUS      RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92                  0/1     Completed   0          11m
$ kubectl delete job netapp-tensorflow-single-imagenet
job.batch "netapp-tensorflow-single-imagenet" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.
```

## 4.7 Execute a Synchronous Distributed AI Workload

To execute a synchronous multinode AI and ML job in your Kubernetes cluster, perform the following tasks on the deployment jump host. This process enables you to take advantage of data that is stored on an ONTAP volume and to use more GPUs than a single worker node can provide. See Figure 4 for a visualization.

**Note:** Synchronous distributed jobs can help increase performance and training accuracy compared with asynchronous distributed jobs. A discussion of the pros and cons of synchronous jobs versus asynchronous jobs is outside the scope of this document.

**Figure 4) Synchronous distributed AI job.**




1. Create a Kubernetes deployment for a worker that participates in the execution of the synchronous multinode job.

   The following example commands show the creation of one worker that participates in the synchronous distributed execution of the same TensorFlow benchmark job that was executed on a single node in section 4.6. In this specific example, only a single worker is deployed because the job is executed across two worker nodes. This example worker deployment requests eight GPUs and thus can run on a single GPU worker node that features eight or more GPUs. If your GPU worker nodes feature more than eight GPUs, to maximize performance, you might want to increase this number to be equal to the number of GPUs that your worker nodes feature. For more information about Kubernetes deployments, see the official Kubernetes documentation.

   A Kubernetes deployment is created in this example because this specific containerized worker would never complete on its own. Therefore, it doesn't make sense to deploy it by using the Kubernetes job construct. If your worker is designed or written to complete on its own, then it might make sense to use the job construct to deploy your worker.

   The pod that is specified in this example deployment specification is given a `hostNetwork` value of `true`. This value means that the pod uses the host worker node's networking stack instead of the virtual networking stack that Kubernetes usually creates for each pod. This annotation is used in this case because the specific workload relies on Open MPI, NCCL, and Horovod to execute the workload in a synchronous distributed manner. Therefore, it requires access to the host networking stack. A discussion about Open MPI, NCCL, and Horovod is outside the scope of this document. Whether or not this `hostNetwork: true` annotation is necessary depends on the requirements of the specific workload that you are executing. For more information about the `hostNetwork` field, see the [official Kubernetes documentation](official Kubernetes documentation).

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-worker.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: netapp-tensorflow-multi-imagenet-worker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: netapp-tensorflow-multi-imagenet-worker
  template:
    metadata:
      labels:
        app: netapp-tensorflow-multi-imagenet-worker
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["bash", "/netapp/scripts/start-slave-multi.sh", "22122"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-worker.yaml
deployment.apps/netapp-tensorflow-multi-imagenet-worker created
$ kubectl get deployments
NAME                                      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1            1           4s
```

2. Confirm that the worker deployment that you created in step 1 launched successfully. The following example commands confirm that a single worker pod was created for the deployment, as indicated in the deployment specification, and that this pod is currently running on one of the GPU worker nodes.

```
$ kubectl get pods -o wide
NAME                                                       READY   STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
```

```
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1     Running   0          60s
10.61.218.154   10.61.218.154   <none>
$ kubectl logs netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725
22122
```

3. Create a Kubernetes job for a master that kicks off, participates in, and tracks the execution of the synchronous multinode job. The following example commands create one master that kicks off, participates in, and tracks the synchronous distributed execution of the same TensorFlow benchmark job that was executed on a single node in section 4.6. This example master job requests eight GPUs and thus can run on a single GPU worker node that features eight or more GPUs. If your GPU worker nodes feature more than eight GPUs, to maximize performance, you might want to increase this number to be equal to the number of GPUs that your worker nodes feature.

   **Note:** The master pod that is specified in this example job specification is given a `hostNetwork` value of `true`, just as the worker pod was given a `hostNetwork` value of `true` in step 1. See step 1 for details about why this value is necessary.

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-master.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-multi-imagenet-master
spec:
  backoffLimit: 5
  template:
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
dataset_dir=/mnt/mount_0/dataset/imagenet", "--port=22122", "--num_devices=16", "--
dgx_version=dgx1", "--nodes=10.61.218.152,10.61.218.154"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-master.yaml
job.batch/netapp-tensorflow-multi-imagenet-master created
$ kubectl get jobs
NAME                                      COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   0/1           25s        25s
```

4. Confirm that the master job that you created in step 3 is running correctly. The following example command confirms that a single master pod was created for the job, as indicated in the job specification, and that this pod is currently running on one of the GPU worker nodes. You should also see that the worker pod that you originally saw in step 2 is still running and that the master and worker pods are running on different nodes.

```
$ kubectl get pods -o wide
NAME                                                        READY   STATUS    RESTARTS   AGE
IP                NODE              NOMINATED NODE
netapp-tensorflow-multi-imagenet-master-ppwwj               1/1     Running   0          45s
10.61.218.152     10.61.218.152     <none>
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725    1/1     Running   0          26m
10.61.218.154     10.61.218.154     <none>
```

5. Confirm that the master job that you created in step 3 completes successfully. The following example commands confirm that the job completed successfully.

```
$ kubectl get jobs
NAME                                       COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master    1/1           5m50s      9m18s
$ kubectl get pods
NAME                                                        READY   STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj               0/1     Completed   0          9m38s
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725    1/1     Running     0          35m
$ kubectl logs netapp-tensorflow-multi-imagenet-master-ppwwj
[10.61.218.152:00008] WARNING: local probe returned unhandled shell:unknown assuming bash
rm: cannot remove '/lib': Is a directory
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 702
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 711
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 702
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 711
Total images/sec = 12881.33875
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 2 -H 10.61.218.152:1,10.61.218.154:1 -mca pml ob1 -mca btl ^openib
-mca btl_tcp_if_include enp1s0f0 -mca plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" bash -c
'sync; echo 1 > /proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 16 -H 10.61.218.152:8,10.61.218.154:8 -bind-to none -map-by slot -x
NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl ^openib -mca btl_tcp_if_include
enp1s0f0 -x NCCL_IB_HCA=mlx5 -x NCCL_NET_GDR_READ=1 -x NCCL_IB_SL=3 -x NCCL_IB_GID_INDEX=3 -x
NCCL_SOCKET_IFNAME=enp5s0.3091,enp12s0.3092,enp132s0.3093,enp139s0.3094 -x NCCL_IB_CUDA_SUPPORT=1
-mca orte_base_help_aggregate 0 -mca plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --
model=resnet50 --batch_size=256 --device=gpu --force_gpu_compatible=True --num_intra_threads=1 --
num_inter_threads=48 --variable_update=horovod --batch_group_size=20 --num_batches=500 --
nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True --use_tf_layers=False --
data_name=imagenet --use_datasets=True --data_dir=/mnt/mount_0/dataset/imagenet --
datasets_parallel_interleave_cycle_length=10 --datasets_sloppy_parallel_interleave=False --
num_mounts=2 --mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000 --
datasets_use_prefetch=True --datasets_num_private_threads=4 --horovod_device=gpu >
/tmp/20190814_161609_tensorflow_horovod_rdma_resnet50_gpu_16_256_b500_imagenet_nodistort_fp16_r10
_m2_nockpt.txt 2>&1
```

6. Delete the worker deployment when you no longer need it. The following example commands show the deletion of the worker deployment object that was created in step 1.

   **Note:** When you delete the worker deployment object, Kubernetes automatically deletes any associated worker pods.

```
$ kubectl get deployments
NAME                                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker    1         1         1            1           43m
$ kubectl get pods
NAME                                                        READY   STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj               0/1     Completed   0          17m
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725    1/1     Running     0          43m
$ kubectl delete deployment netapp-tensorflow-multi-imagenet-worker
deployment.extensions "netapp-tensorflow-multi-imagenet-worker" deleted
$ kubectl get deployments
```

```
No resources found.
$ kubectl get pods
NAME                                          READY   STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj   0/1     Completed   0          18m
```

7. **Optional:** Clean up the master job artifacts. The following example commands show the deletion of the master job object that was created in step 3.

   **Note:** When you delete the master job object, Kubernetes automatically deletes any associated master pods.

```
$ kubectl get jobs
NAME                                      COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      19m
$ kubectl get pods
NAME                                          READY   STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj   0/1     Completed   0          19m
$ kubectl delete job netapp-tensorflow-multi-imagenet-master
job.batch "netapp-tensorflow-multi-imagenet-master" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.
```

## 4.8 Enable Volume Snapshot Feature

At the time of writing, the volume snapshot feature within Kubernetes is turned off by default. If you want to create volume snapshots using standard Kubernetes tools and/or APIs, then you must enable the feature by completing the following tasks:

1. Enable the volume snapshot feature gate within the Kubelet config file on each of your Kubernetes nodes (all master and worker nodes). If your nodes are running Ubuntu, this file should be located at `/etc/default/kubelet`. If your nodes are running RHEL or CentOS, this file should be located at `/etc/sysconfig/kubelet`. In the following example, there is no existing Kubelet config file, so one is added as follows:

```
$ sudo -i
$ cat << EOF > /etc/default/kubelet
KUBELET_EXTRA_ARGS=--feature-
gates=VolumeSnapshotDataSource=true,CSIDriverRegistry=true,CSINodeInfo=true
EOF
$ systemctl restart kubelet
```

2. Enable the volume snapshot feature gate within the `kube-apiserver` config file on each of your Kubernetes master nodes. This file should be located at `/etc/kubernetes/manifests/kube-apiserver.yaml`. In the following example, the highlighted text is added to the file.

```
$ vi /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=10.61.218.131
    - --allow-privileged=true
    - --apiserver-count=3
    - --authorization-mode=Node,RBAC
    - --bind-address=0.0.0.0
    - --client-ca-file=/etc/kubernetes/ssl/ca.crt
```

```
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --endpoint-reconciler-type=lease
    - --etcd-cafile=/etc/ssl/etcd/ssl/ca.pem
    - --etcd-certfile=/etc/ssl/etcd/ssl/node-mgmt01.pem
    - --etcd-keyfile=/etc/ssl/etcd/ssl/node-mgmt01-key.pem
    - --etcd-
servers=https://10.61.218.131:2379,https://10.61.218.132:2379,https://10.61.218.133:2379
    - --insecure-port=0
    - --kubelet-client-certificate=/etc/kubernetes/ssl/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/ssl/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalDNS,InternalIP,Hostname,ExternalDNS,ExternalIP
    - --proxy-client-cert-file=/etc/kubernetes/ssl/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/ssl/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/ssl/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --runtime-config=
    - --secure-port=6443
    - --service-account-key-file=/etc/kubernetes/ssl/sa.pub
    - --service-cluster-ip-range=10.233.0.0/18
    - --service-node-port-range=30000-32767
    - --storage-backend=etcd3
    - --tls-cert-file=/etc/kubernetes/ssl/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/ssl/apiserver.key
    - --feature-gates=VolumeSnapshotDataSource=true,CSIDriverRegistry=true,CSINodeInfo=true
    image: gcr.io/google-containers/kube-apiserver:v1.14.3
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 10.61.218.131
        path: /healthz
        port: 6443
        scheme: HTTPS
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: kube-apiserver
    resources:
      requests:
        cpu: 250m
    volumeMounts:
    - mountPath: /etc/ssl/certs
      name: ca-certs
      readOnly: true
    - mountPath: /etc/ca-certificates
      name: etc-ca-certificates
      readOnly: true
    - mountPath: /etc/ssl/etcd/ssl
      name: etcd-certs-0
      readOnly: true
    - mountPath: /etc/kubernetes/ssl
      name: k8s-certs
      readOnly: true
    - mountPath: /usr/local/share/ca-certificates
      name: usr-local-share-ca-certificates
      readOnly: true
    - mountPath: /usr/share/ca-certificates
      name: usr-share-ca-certificates
      readOnly: true
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /etc/ssl/certs
      type: DirectoryOrCreate
    name: ca-certs
  - hostPath:
      path: /etc/ca-certificates
```

```
      type: DirectoryOrCreate
    name: etc-ca-certificates
  - hostPath:
      path: /etc/ssl/etcd/ssl
      type: DirectoryOrCreate
    name: etcd-certs-0
  - hostPath:
      path: /etc/kubernetes/ssl
      type: DirectoryOrCreate
    name: k8s-certs
  - hostPath:
      path: /usr/local/share/ca-certificates
      type: DirectoryOrCreate
    name: usr-local-share-ca-certificates
  - hostPath:
      path: /usr/share/ca-certificates
      type: ""
    name: usr-share-ca-certificates
status: {}
```

3. Enable the volume snapshot feature gate within the `kube-controller-manager` config file on each of your Kubernetes master nodes. This file should be located at /`etc/kubernetes/manifests/kube-controller-manager.yaml`. In the following example, the highlighted text is added to the file:

```
$ vi /etc/kubernetes/manifests/kube-controller-manager.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-controller-manager
    - --allocate-node-cidrs=true
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=0.0.0.0
    - --client-ca-file=/etc/kubernetes/ssl/ca.crt
    - --cluster-cidr=10.233.64.0/18
    - --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/ssl/ca.key
    - --configure-cloud-routes=false
    - --controllers=*,bootstrapsigner,tokencleaner
    - --kubeconfig=/etc/kubernetes/controller-manager.conf
    - --leader-elect=true
    - --node-cidr-mask-size=24
    - --node-monitor-grace-period=40s
    - --node-monitor-period=5s
    - --pod-eviction-timeout=5m0s
    - --requestheader-client-ca-file=/etc/kubernetes/ssl/front-proxy-ca.crt
    - --root-ca-file=/etc/kubernetes/ssl/ca.crt
    - --service-account-private-key-file=/etc/kubernetes/ssl/sa.key
    - --use-service-account-credentials=true
    - --feature-gates=VolumeSnapshotDataSource=true,CSIDriverRegistry=true,CSINodeInfo=true
    image: gcr.io/google-containers/kube-controller-manager:v1.14.3
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10252
        scheme: HTTP
      initialDelaySeconds: 15
```

```
      timeoutSeconds: 15
    name: kube-controller-manager
    resources:
      requests:
        cpu: 200m
    volumeMounts:
    - mountPath: /etc/ssl/certs
      name: ca-certs
      readOnly: true
    - mountPath: /etc/ca-certificates
      name: etc-ca-certificates
      readOnly: true
    - mountPath: /usr/libexec/kubernetes/kubelet-plugins/volume/exec
      name: flexvolume-dir
    - mountPath: /etc/kubernetes/ssl
      name: k8s-certs
      readOnly: true
    - mountPath: /etc/kubernetes/controller-manager.conf
      name: kubeconfig
      readOnly: true
    - mountPath: /usr/local/share/ca-certificates
      name: usr-local-share-ca-certificates
      readOnly: true
    - mountPath: /usr/share/ca-certificates
      name: usr-share-ca-certificates
      readOnly: true
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /etc/ssl/certs
      type: DirectoryOrCreate
    name: ca-certs
  - hostPath:
      path: /etc/ca-certificates
      type: DirectoryOrCreate
    name: etc-ca-certificates
  - hostPath:
      path: /usr/libexec/kubernetes/kubelet-plugins/volume/exec
      type: DirectoryOrCreate
    name: flexvolume-dir
  - hostPath:
      path: /etc/kubernetes/ssl
      type: DirectoryOrCreate
    name: k8s-certs
  - hostPath:
      path: /etc/kubernetes/controller-manager.conf
      type: FileOrCreate
    name: kubeconfig
  - hostPath:
      path: /usr/local/share/ca-certificates
      type: DirectoryOrCreate
    name: usr-local-share-ca-certificates
  - hostPath:
      path: /usr/share/ca-certificates
      type: DirectoryOrCreate
    name: usr-share-ca-certificates
status: {}
```

4. Enable the volume snapshot feature gate within the `kube-scheduler` config file on each of your Kubernetes master nodes. This file should be located at /`etc/kubernetes/manifests/kube-scheduler.yaml`. In the following example, the highlighted text is added to the file.

```
$ vi /etc/kubernetes/manifests/kube-scheduler.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-scheduler
```

```
    tier: control-plane
  name: kube-scheduler
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-scheduler
    - --bind-address=0.0.0.0
    - --kubeconfig=/etc/kubernetes/scheduler.conf
    - --leader-elect=true
    - --feature-gates=VolumeSnapshotDataSource=true,CSIDriverRegistry=true,CSINodeInfo=true
    image: gcr.io/google-containers/kube-scheduler:v1.14.3
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10251
        scheme: HTTP
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: kube-scheduler
    resources:
      requests:
        cpu: 100m
    volumeMounts:
    - mountPath: /etc/kubernetes/scheduler.conf
      name: kubeconfig
      readOnly: true
  hostNetwork: true
  priorityClassName: system-cluster-critical
  volumes:
  - hostPath:
      path: /etc/kubernetes/scheduler.conf
      type: FileOrCreate
    name: kubeconfig
status: {}
```

## 4.9 Create a Volume Snapshot

To create a snapshot of a Trident-controlled volume from within your Kubernetes environment, perform the following tasks on the deployment jump host. This operation takes advantage of NetApp Snapshot™ technology but is undertaken using standard Kubernetes tooling; no NetApp or NetApp ONTAP expertise is required.

1. Create a volume snapshot class for Trident. Before creating a snapshot of a Trident-controlled volume, you must set up a volume snapshot class. The example commands that follow show the creation of a volume snapshot class named `csi-snapclass`.

```
$ cat << EOF > ./snapshot-class.yaml
apiVersion: snapshot.storage.k8s.io/v1alpha1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
snapshotter: csi.trident.netapp.io
EOF
$ kubectl create -f ./snapshot-class.yaml
volumesnapshotclass.snapshot.storage.k8s.io/csi-snapclass created
$ kubectl get volumesnapshotclass
NAME            AGE
csi-snapclass   1m
```

2. Use standard Kubernetes tooling to create a snapshot of a FlexVol volume. Note that, at the time of writing, Trident does not support snapshots for FlexGroup volumes. The example commands that follow show the creation of a snapshot for the FlexVol volume that was created in section 4.5, step 2.

```
$ cat << EOF > ./snap-tensorflow-results.yaml
```

```
apiVersion: snapshot.storage.k8s.io/v1alpha1
kind: VolumeSnapshot
metadata:
  name: tensorflow-results-snap1
spec:
  snapshotClassName: csi-snapclass
  source:
    name: tensorflow-results
    kind: PersistentVolumeClaim
EOF
$ kubectl create -f ./snap-tensorflow-results.yaml
volumesnapshot.snapshot.storage.k8s.io/tensorflow-results-snap1 created
$ kubectl get volumesnapshot
NAME                       AGE
tensorflow-results-snap1   15s
$ kubectl describe volumesnapshot tensorflow-results-snap1
Name:         tensorflow-results-snap1
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  snapshot.storage.k8s.io/v1alpha1
Kind:         VolumeSnapshot
Metadata:
  Creation Timestamp:  2019-09-13T18:52:40Z
  Finalizers:
    snapshot.storage.kubernetes.io/volumesnapshot-protection
  Generation:        3
  Resource Version:  2927664
  Self Link:
/apis/snapshot.storage.k8s.io/v1alpha1/namespaces/default/volumesnapshots/tensorflow-results-
snap1
  UID:               a9a28907-d657-11e9-a043-00505681a82d
Spec:
  Snapshot Class Name:    csi-snapclass
  Snapshot Content Name:  snapcontent-a9a28907-d657-11e9-a043-00505681a82d
  Source:
    API Group:  <nil>
    Kind:       PersistentVolumeClaim
    Name:       tensorflow-results
Status:
  Creation Time:  2019-09-13T18:52:41Z
  Ready To Use:   true
  Restore Size:   1Gi
Events:           <none>
```

## 4.10 Provision a New Volume from a Snapshot

To provision a new volume that is a clone of a snapshot that was created within your Kubernetes environment, perform the following tasks on the deployment jump host. This operation takes advantage of NetApp FlexClone technology but is undertaken using standard Kubernetes tooling; no NetApp or NetApp ONTAP expertise is required.

1. Use standard Kubernetes tooling to provision a new volume that is a clone of a snapshot. The example commands that follow show the creation of a new volume that is a clone of the snapshot that was created in section 4.9, step 2.

```
$ cat << EOF > ./pvc-from-tensorflow-results-snap1.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-tensorflow-results-snap1
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ontap-ai-flexvols-retain
  resources:
    requests:
      storage: 1Gi
  dataSource:
```

```
    name: tensorflow-results-snap1
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
$ kubectl create -f ./pvc-from-tensorflow-results-snap1.yaml
persistentvolumeclaim/pvc-from-tensorflow-results-snap1 created
$ kubectl get pvc
NAME                                 STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS                AGE
pvc-from-tensorflow-results-snap1    Bound    pvc-998a4762-d8b9-11e9-a043-00505681a82d   1Gi
RWX            ontap-ai-flexvols-retain   5s
tensorflow-results                   Bound    pvc-2c4d45e9-d4a0-11e9-9b9d-00505681a82d   1Gi
RWX            ontap-ai-flexvols-retain   5d5h
```

# 5 Kubeflow Configuration and Example Operations

This section describes the tasks that you must complete to deploy and configure Kubeflow within the Kubernetes cluster that was provisioned in section 4 in the validation environment described in section 3.

An NVIDIA DGX-1 server and a NetApp AFF A800 system were used for this validation exercise. However, the tasks that are outlined in this section should apply to any environment that contains a NetApp ONTAP appliance or instance. Examples include a NetApp AFF storage appliance, a NetApp ONTAP Select software-defined storage instance, or a NetApp Cloud Volumes ONTAP instance running in the cloud. The NetApp instance can be paired with servers or with instances that feature NVIDIA GPUs, including white-box servers that feature NVIDIA GPUs or cloud-compute instances that feature NVIDIA GPUs.

## 5.1 Prerequisites

Before you perform the configuration exercises that are outlined in this section, we assume that you have already configured your Kubernetes cluster by performing the tasks outlined in sections 4.1 through 4.5. You must also create a default snapshot class within your Kubernetes environment by performing the task outlined in section 4.9, step 1.

## 5.2 Set Default Kubernetes StorageClass

Before you deploy Kubeflow, you must designate a default StorageClass within your Kubernetes cluster. The Kubeflow deployment process attempts to provision new PVCs using the default StorageClass. If no StorageClass is designated as the default StorageClass, then the deployment fails. To designate a default StorageClass within your cluster, perform the following task on the deployment jump host:

1. Designate one of your existing StorageClasses as the default StorageClass. The example commands that follow show the designation of the `ontap-ai-flexvols-retain` StorageClass that was created in section 4.4, step 2 as the default StorageClass.

   **Note:** The `ontap-nas-flexgroup` backendType has a minimum PVC size of 800GB. By default, Kubeflow attempts to provision PVCs that are smaller than 800GB. Therefore, you should not designate a StorageClass that utilizes the `ontap-nas-flexgroup` backendType as the default StorageClass for the purposes of Kubeflow deployment.

```
$ kubectl get sc
NAME                                 PROVISIONER             AGE
ontap-ai-flexgroups-retain           csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface1    csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface2    csi.trident.netapp.io   25h
ontap-ai-flexvols-retain             csi.trident.netapp.io   3s
$ kubectl patch storageclass ontap-ai-flexvols-retain -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
storageclass.storage.k8s.io/ontap-ai-flexvols-retain patched
$ kubectl get sc
NAME                                   PROVISIONER             AGE
ontap-ai-flexgroups-retain             csi.trident.netapp.io   25h
ontap-ai-flexgroups-retain-iface1      csi.trident.netapp.io   25h
```

```
ontap-ai-flexgroups-retain-iface2    csi.trident.netapp.io   25h
ontap-ai-flexvols-retain (default)   csi.trident.netapp.io   54s
```

## 5.3 Deploy Kubeflow

To deploy Kubeflow in your Kubernetes cluster, perform the following tasks on the deployment jump host:

1. Deploy Kubeflow in your cluster by following the Deploy Kubeflow instructions found on the Kubeflow Deployment with kfctl_k8s_istio page in the official Kubeflow documentation.

   If the deployment process fails, NetApp recommends removing all leftover artifacts (instructions follow) and rerunning the deployment process. Occasionally, Kubernetes takes too long to download the needed container images, causing the deployment to fail. Rerunning the deployment process usually fixes the issue.

   To remove all leftover artifacts, execute the following commands from within the `KFAPP` directory created as a part of the deployment process:

```
$ kfctl delete all -V
INFO[0000] Downloading /home/cpoc/kubeflow01/app.yaml to /tmp/856959178/app.yaml
filename="v1alpha1/application_types.go:334"
INFO[0000] Writing stripped KfDef to /home/cpoc/kubeflow01/app.yaml
filename="v1alpha1/application_types.go:626"
INFO[0000] Downloading /home/cpoc/kubeflow01/app.yaml to /tmp/631186337/app.yaml
filename="v1alpha1/application_types.go:334"
INFO[0000] Initializing a default restConfig for Kubernetes
filename="kustomize/kustomize.go:249"
INFO[0000] deleting namespace: kubeflow                  filename="kustomize/kustomize.go:547"
$ kubectl delete ns kubeflow-anonymous
namespace "kubeflow-anonymous" deleted
$ kubectl delete deployment --all -n istio-system
deployment.extensions "grafana" deleted
deployment.extensions "istio-citadel" deleted
deployment.extensions "istio-egressgateway" deleted
deployment.extensions "istio-galley" deleted
deployment.extensions "istio-ingressgateway" deleted
deployment.extensions "istio-pilot" deleted
deployment.extensions "istio-policy" deleted
deployment.extensions "istio-sidecar-injector" deleted
deployment.extensions "istio-telemetry" deleted
deployment.extensions "istio-tracing" deleted
deployment.extensions "kiali" deleted
deployment.extensions "prometheus" deleted
$ kubectl delete svc --all -n istio-system
service "grafana" deleted
service "istio-citadel" deleted
service "istio-egressgateway" deleted
service "istio-galley" deleted
service "istio-ingressgateway" deleted
service "istio-pilot" deleted
service "istio-policy" deleted
service "istio-sidecar-injector" deleted
service "istio-telemetry" deleted
service "jaeger-agent" deleted
service "jaeger-collector" deleted
service "jaeger-query" deleted
service "kiali" deleted
service "prometheus" deleted
service "tracing" deleted
service "zipkin" deleted
$ kubectl delete job --all -n istio-system
job.batch "istio-cleanup-secrets-1.1.6" deleted
job.batch "istio-grafana-post-install-1.1.6" deleted
job.batch "istio-security-post-install-1.1.6" deleted
$ kubectl delete horizontalpodautoscaler --all -n istio-system
horizontalpodautoscaler.autoscaling "istio-egressgateway" deleted
horizontalpodautoscaler.autoscaling "istio-ingressgateway" deleted
horizontalpodautoscaler.autoscaling "istio-pilot" deleted
horizontalpodautoscaler.autoscaling "istio-policy" deleted
```

```
horizontalpodautoscaler.autoscaling "istio-telemetry" deleted
$ kubectl get all -n kubeflow
No resources found.
$ kubectl get all -n kubeflow-anonymous
No resources found.
$ kubectl get all -n istio-system
No resources found.
```

**Note:** If your default StorageClass uses a `reclaimPolicy` value of `Retain`, you might also need to remove some leftover persistent volumes (PVs). Execute `kubectl get pv` to get a list of all PVs within your cluster. Any PV that shows a `STATUS` of `Released` was likely created by the Kubeflow deployment process. To remove a PV, execute `kubectl delete pv <pv_name>`.

2. Confirm that all pods deployed within the Kubeflow namespace show a `STATUS` of `Running` and confirm that no components deployed within the namespace are in an error state.

```
$ kubectl get all -n kubeflow
NAME                                                            READY   STATUS    RESTARTS   AGE
pod/admission-webhook-bootstrap-stateful-set-0                  1/1     Running   0          95s
pod/admission-webhook-deployment-6b89c84c98-vrtbh               1/1     Running   0          91s
pod/application-controller-stateful-set-0                       1/1     Running   0          98s
pod/argo-ui-5dcf5d8b4f-m2wn4                                    1/1     Running   0          97s
pod/centraldashboard-cf4874ddc-7hcr8                            1/1     Running   0          97s
pod/jupyter-web-app-deployment-685b455447-gjhh7                 1/1     Running   0          96s
pod/katib-controller-88c97d85c-kgq66                            1/1     Running   1          95s
pod/katib-db-8598468fd8-5jw2c                                   1/1     Running   0          95s
pod/katib-manager-574c8c67f9-wtrf5                              1/1     Running   1          95s
pod/katib-manager-rest-778857c989-fjbzn                         1/1     Running   0          95s
pod/katib-suggestion-bayesianoptimization-65df4d7455-qthmw      1/1     Running   0          94s
pod/katib-suggestion-grid-56bf69f597-98vwn                      1/1     Running   0          94s
pod/katib-suggestion-hyperband-7777b76cb9-9v6dq                 1/1     Running   0          93s
pod/katib-suggestion-nasrl-77f6f9458c-2qzxq                     1/1     Running   0          93s
pod/katib-suggestion-random-77b88b5c79-l64j9                    1/1     Running   0          93s
pod/katib-ui-7587c5b967-nd629                                   1/1     Running   0          95s
pod/metacontroller-0                                            1/1     Running   0          96s
pod/metadata-db-5dd459cc-swzkm                                  1/1     Running   0          94s
pod/metadata-deployment-6cf77db994-69fk7                        1/1     Running   3          93s
pod/metadata-deployment-6cf77db994-mpbjt                        1/1     Running   3          93s
pod/metadata-deployment-6cf77db994-xg7tz                        1/1     Running   3          94s
pod/metadata-ui-78f5b59b56-qb6kr                                1/1     Running   0          94s
pod/minio-758b769d67-llvdr                                      1/1     Running   0          91s
pod/ml-pipeline-5875b9db95-g8t2k                                1/1     Running   0          91s
pod/ml-pipeline-persistenceagent-9b69ddd46-bt9r9                1/1     Running   0          90s
pod/ml-pipeline-scheduledworkflow-7b8d756c76-7x56s              1/1     Running   0          90s
pod/ml-pipeline-ui-79ffd9c76-fcwpd                              1/1     Running   0          90s
pod/ml-pipeline-viewer-controller-deployment-5fdc87f58-b2t9r    1/1     Running   0          90s
pod/mysql-657f87857d-l5k9z                                      1/1     Running   0          91s
pod/notebook-controller-deployment-56b4f59bbf-8bvnr             1/1     Running   0          92s
pod/profiles-deployment-6bc745947-mrdkh                         2/2     Running   0          90s
pod/pytorch-operator-77c97f4879-hmlrv                           1/1     Running   0          92s
pod/seldon-operator-controller-manager-0                        1/1     Running   1          91s
pod/spartakus-volunteer-5fdfddb779-l7qkm                        1/1     Running   0          92s
pod/tensorboard-6544748d94-nh8b2                                1/1     Running   0          92s
pod/tf-job-dashboard-56f79c59dd-6w59t                           1/1     Running   0          92s
pod/tf-job-operator-79cbfd6dbc-rb58c                            1/1     Running   0          91s
pod/workflow-controller-db644d554-cwrnb                         1/1     Running   0          97s


NAME                                                    TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)             AGE
service/admission-webhook-service                       ClusterIP   10.233.51.169   <none>
443/TCP             97s
service/application-controller-service                  ClusterIP   10.233.4.54     <none>
443/TCP             98s
service/argo-ui                                         NodePort    10.233.47.191   <none>
80:31799/TCP        97s
service/centraldashboard                                ClusterIP   10.233.8.36     <none>
80/TCP              97s
```

```
service/jupyter-web-app-service                     ClusterIP   10.233.1.42     <none>
80/TCP            97s
service/katib-controller                            ClusterIP   10.233.25.226   <none>
443/TCP           96s
service/katib-db                                    ClusterIP   10.233.33.151   <none>
3306/TCP          97s
service/katib-manager                               ClusterIP   10.233.46.239   <none>
6789/TCP          96s
service/katib-manager-rest                          ClusterIP   10.233.55.32    <none>
80/TCP            96s
service/katib-suggestion-bayesianoptimization       ClusterIP   10.233.49.191   <none>
6789/TCP          95s
service/katib-suggestion-grid                       ClusterIP   10.233.9.105    <none>
6789/TCP          95s
service/katib-suggestion-hyperband                  ClusterIP   10.233.22.2     <none>
6789/TCP          95s
service/katib-suggestion-nasrl                      ClusterIP   10.233.63.73    <none>
6789/TCP          95s
service/katib-suggestion-random                     ClusterIP   10.233.57.210   <none>
6789/TCP          95s
service/katib-ui                                    ClusterIP   10.233.6.116    <none>
80/TCP            96s
service/metadata-db                                 ClusterIP   10.233.31.2     <none>
3306/TCP          96s
service/metadata-service                            ClusterIP   10.233.27.104   <none>
8080/TCP          96s
service/metadata-ui                                 ClusterIP   10.233.57.177   <none>
80/TCP            96s
service/minio-service                               ClusterIP   10.233.44.90    <none>
9000/TCP          94s
service/ml-pipeline                                 ClusterIP   10.233.41.201   <none>
8888/TCP,8887/TCP   94s
service/ml-pipeline-tensorboard-ui                  ClusterIP   10.233.36.207   <none>
80/TCP            93s
service/ml-pipeline-ui                              ClusterIP   10.233.61.150   <none>
80/TCP            93s
service/mysql                                       ClusterIP   10.233.55.117   <none>
3306/TCP          94s
service/notebook-controller-service                 ClusterIP   10.233.10.166   <none>
443/TCP           95s
service/profiles-kfam                               ClusterIP   10.233.33.79    <none>
8081/TCP          92s
service/pytorch-operator                            ClusterIP   10.233.37.112   <none>
8443/TCP          95s
service/seldon-operator-controller-manager-service   ClusterIP   10.233.30.178   <none>
443/TCP           92s
service/tensorboard                                 ClusterIP   10.233.58.151   <none>
9000/TCP          94s
service/tf-job-dashboard                            ClusterIP   10.233.4.17     <none>
80/TCP            94s
service/tf-job-operator                             ClusterIP   10.233.60.32    <none>
8443/TCP          94s
service/webhook-server-service                      ClusterIP   10.233.32.167   <none>
443/TCP           87s


NAME                                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/admission-webhook-deployment            1/1     1            1           97s
deployment.apps/argo-ui                                 1/1     1            1           97s
deployment.apps/centraldashboard                        1/1     1            1           97s
deployment.apps/jupyter-web-app-deployment              1/1     1            1           97s
deployment.apps/katib-controller                        1/1     1            1           96s
deployment.apps/katib-db                                1/1     1            1           97s
deployment.apps/katib-manager                           1/1     1            1           96s
deployment.apps/katib-manager-rest                      1/1     1            1           96s
deployment.apps/katib-suggestion-bayesianoptimization   1/1     1            1           95s
deployment.apps/katib-suggestion-grid                   1/1     1            1           95s
deployment.apps/katib-suggestion-hyperband              1/1     1            1           95s
deployment.apps/katib-suggestion-nasrl                  1/1     1            1           95s
deployment.apps/katib-suggestion-random                 1/1     1            1           95s
deployment.apps/katib-ui                                1/1     1            1           96s
```

```
deployment.apps/metadata-db                                 1/1     1            1           96s
deployment.apps/metadata-deployment                         3/3     3            3           96s
deployment.apps/metadata-ui                                 1/1     1            1           96s
deployment.apps/minio                                       1/1     1            1           94s
deployment.apps/ml-pipeline                                 1/1     1            1           94s
deployment.apps/ml-pipeline-persistenceagent                1/1     1            1           93s
deployment.apps/ml-pipeline-scheduledworkflow               1/1     1            1           93s
deployment.apps/ml-pipeline-ui                              1/1     1            1           93s
deployment.apps/ml-pipeline-viewer-controller-deployment   1/1     1            1           93s
deployment.apps/mysql                                       1/1     1            1           94s
deployment.apps/notebook-controller-deployment              1/1     1            1           95s
deployment.apps/profiles-deployment                         1/1     1            1           92s
deployment.apps/pytorch-operator                            1/1     1            1           95s
deployment.apps/spartakus-volunteer                         1/1     1            1           94s
deployment.apps/tensorboard                                 1/1     1            1           94s
deployment.apps/tf-job-dashboard                            1/1     1            1           94s
deployment.apps/tf-job-operator                             1/1     1            1           94s
deployment.apps/workflow-controller                         1/1     1            1           97s

NAME                                                                  DESIRED   CURRENT   READY
AGE
replicaset.apps/admission-webhook-deployment-6b89c84c98              1         1         1
97s
replicaset.apps/argo-ui-5dcf5d8b4f                                    1         1         1
97s
replicaset.apps/centraldashboard-cf4874ddc                            1         1         1
97s
replicaset.apps/jupyter-web-app-deployment-685b455447                 1         1         1
97s
replicaset.apps/katib-controller-88c97d85c                            1         1         1
96s
replicaset.apps/katib-db-8598468fd8                                   1         1         1
97s
replicaset.apps/katib-manager-574c8c67f9                              1         1         1
96s
replicaset.apps/katib-manager-rest-778857c989                         1         1         1
96s
replicaset.apps/katib-suggestion-bayesianoptimization-65df4d7455     1         1         1
95s
replicaset.apps/katib-suggestion-grid-56bf69f597                      1         1         1
95s
replicaset.apps/katib-suggestion-hyperband-7777b76cb9                 1         1         1
95s
replicaset.apps/katib-suggestion-nasrl-77f6f9458c                     1         1         1
95s
replicaset.apps/katib-suggestion-random-77b88b5c79                    1         1         1
95s
replicaset.apps/katib-ui-7587c5b967                                   1         1         1
96s
replicaset.apps/metadata-db-5dd459cc                                  1         1         1
96s
replicaset.apps/metadata-deployment-6cf77db994                        3         3         3
96s
replicaset.apps/metadata-ui-78f5b59b56                                1         1         1
96s
replicaset.apps/minio-758b769d67                                      1         1         1
93s
replicaset.apps/ml-pipeline-5875b9db95                                1         1         1
93s
replicaset.apps/ml-pipeline-persistenceagent-9b69ddd46                1         1         1
92s
replicaset.apps/ml-pipeline-scheduledworkflow-7b8d756c76              1         1         1
91s
replicaset.apps/ml-pipeline-ui-79ffd9c76                              1         1         1
91s
replicaset.apps/ml-pipeline-viewer-controller-deployment-5fdc87f58   1         1         1
91s
replicaset.apps/mysql-657f87857d                                      1         1         1
92s
replicaset.apps/notebook-controller-deployment-56b4f59bbf             1         1         1
94s
```

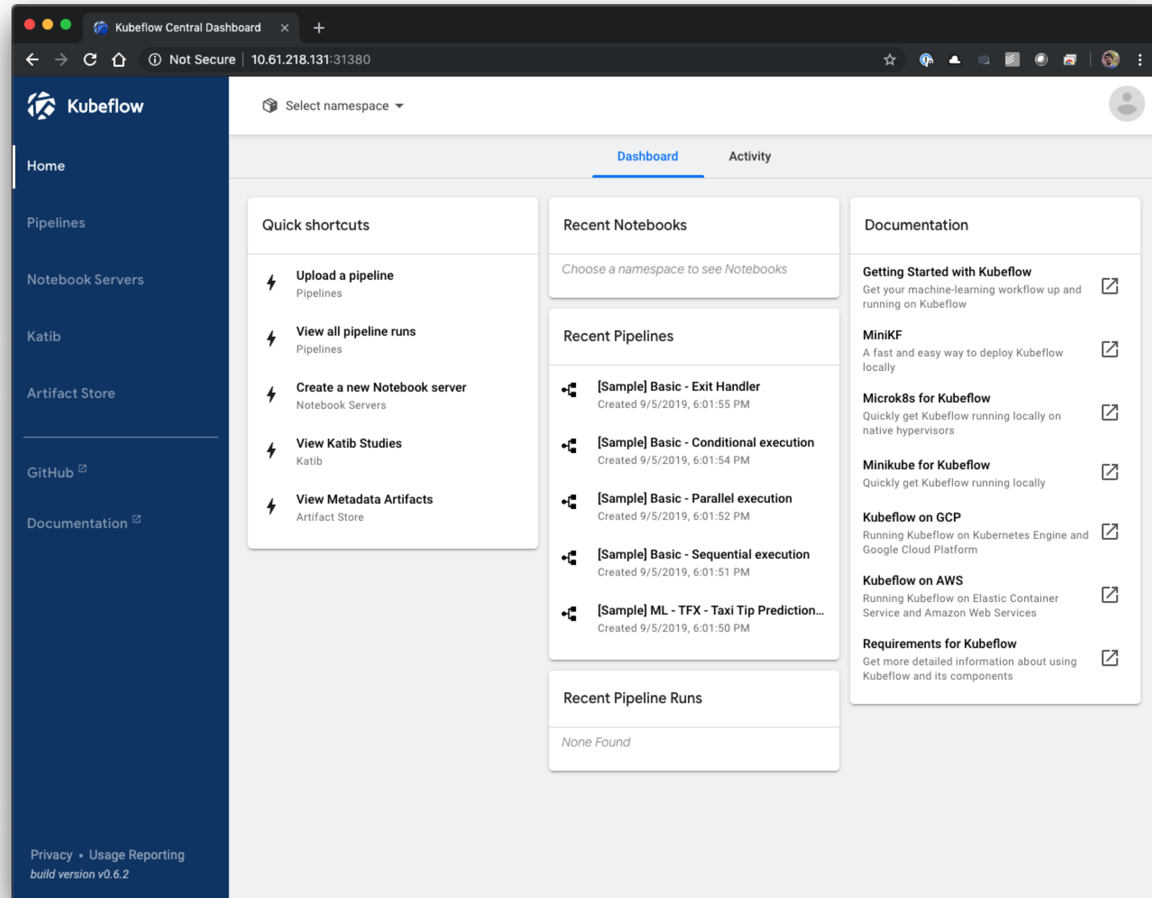
```
replicaset.apps/profiles-deployment-6bc745947                    1         1         1
91s
replicaset.apps/pytorch-operator-77c97f4879                      1         1         1
94s
replicaset.apps/spartakus-volunteer-5fdfddb779                   1         1         1
94s
replicaset.apps/tensorboard-6544748d94                           1         1         1
93s
replicaset.apps/tf-job-dashboard-56f79c59dd                      1         1         1
93s
replicaset.apps/tf-job-operator-79cbfd6dbc                       1         1         1
93s
replicaset.apps/workflow-controller-db644d554                    1         1         1
97s

NAME                                                        READY   AGE
statefulset.apps/admission-webhook-bootstrap-stateful-set   1/1     97s
statefulset.apps/application-controller-stateful-set        1/1     98s
statefulset.apps/metacontroller                             1/1     98s
statefulset.apps/seldon-operator-controller-manager         1/1     92s

$ kubectl get pvc -n kubeflow
NAME             STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS               AGE
katib-mysql      Bound    pvc-b07f293e-d028-11e9-9b9d-00505681a82d   10Gi       RWO
ontap-ai-flexvols-retain   27m
metadata-mysql   Bound    pvc-b0f3f032-d028-11e9-9b9d-00505681a82d   10Gi       RWO
ontap-ai-flexvols-retain   27m
minio-pv-claim   Bound    pvc-b22727ee-d028-11e9-9b9d-00505681a82d   20Gi       RWO
ontap-ai-flexvols-retain   27m
mysql-pv-claim   Bound    pvc-b2429afd-d028-11e9-9b9d-00505681a82d   20Gi       RWO
ontap-ai-flexvols-retain   27m
```

3. Retrieve the port number of the port that is mapped to the Kubeflow central dashboard with Istio. You are looking for the port that is mapped to port 80. In the following example, port 31380 is mapped to port 80.

```
$ kubectl get svc istio-ingressgateway -n istio-system
NAME                   TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
istio-ingressgateway   NodePort   10.233.44.218   <none>
15020:32276/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30522/TCP,15030:32354/TCP,15031:
31606/TCP,15032:31452/TCP,15443:30930/TCP   29m
```

4. Access the Kubeflow central dashboard by navigating to `http://<ip_address_of_any_kubernetes_worker_node>:<port_number_retrieved_in_step_3>` in your web browser.




## 5.4 Provision a Jupyter Notebook Server

To provision a new Jupyter Notebook Server with Kubeflow, perform the following tasks. For more information about Jupyter Notebooks within the Kubeflow context, see the official Kubeflow documentation.

1. Use the Trident volume import functionality to import any existing dataset volumes that you want to mount on your new Jupyter Notebook Server. The volume(s) must be imported in the namespace that the new Jupyter Notebook Server is created in (see step 4 below).

   The example commands that follow show the importing of the same FlexGroup volume containing data to be used by AI jobs that was imported in section 4.5, step 1. This time, however, the volume is imported in the `kubeflow-anonymous` namespace because that is the namespace that the new Jupyter Notebook Server is created in in step 4. To mount this existing volume on the new Jupyter Notebook Server using Kubeflow, a PVC must exist for the volume in the same namespace.

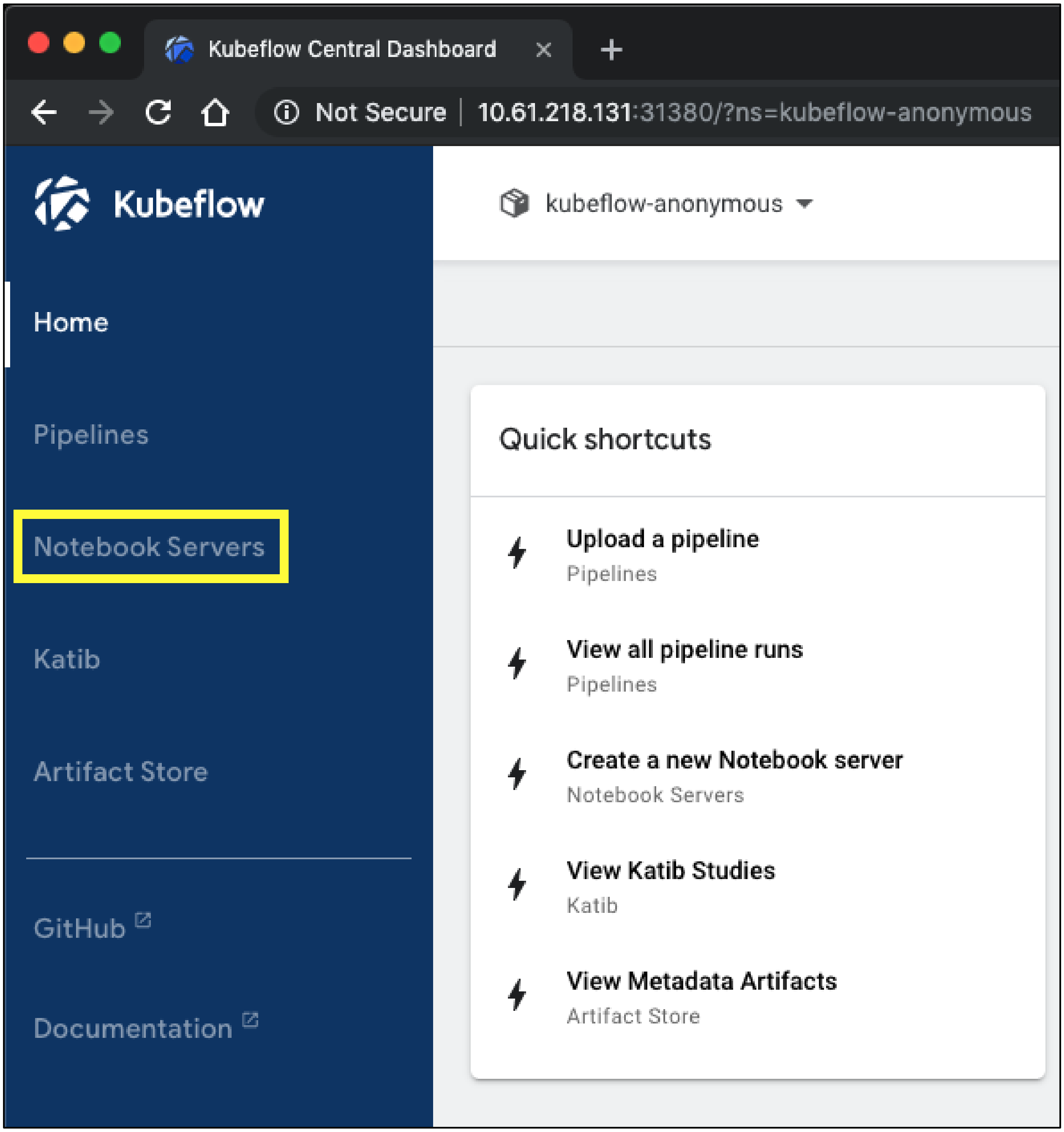
```
$ cat << EOF > ./pvc-import-pb_fg_all-kubeflow-anonymous.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pb-fg-all
  namespace: kubeflow-anonymous
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexgroups-retain
```

```
EOF
$ tridentctl import volume ontap-ai-flexgroups-iface1 pb_fg_all -f ./pvc-import-pb_fg_all-
kubeflow-anonymous.yaml -n trident
+------------------------------------------+---------+----------------------------+----------+----
----------------------------------+---------+---------+
|                   NAME                   |  SIZE   |       STORAGE CLASS        | PROTOCOL |
BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------+----------------------------+----------+----
----------------------------------+---------+---------+
| pvc-1ed071be-d5a6-11e9-8278-00505681feb6 | 10 TiB | ontap-ai-flexgroups-retain | file     |
12f4f8fa-0500-4710-a023-d9b47e86a2ec | online | true    |
+------------------------------------------+---------+----------------------------+----------+----
----------------------------------+---------+---------+
$ kubectl get pvc -n kubeflow-anonymous
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS                 AGE
pb-fg-all   Bound    pvc-1ed071be-d5a6-11e9-8278-00505681feb6   10Ti       ROX            ontap-
ai-flexgroups-retain   14s
```

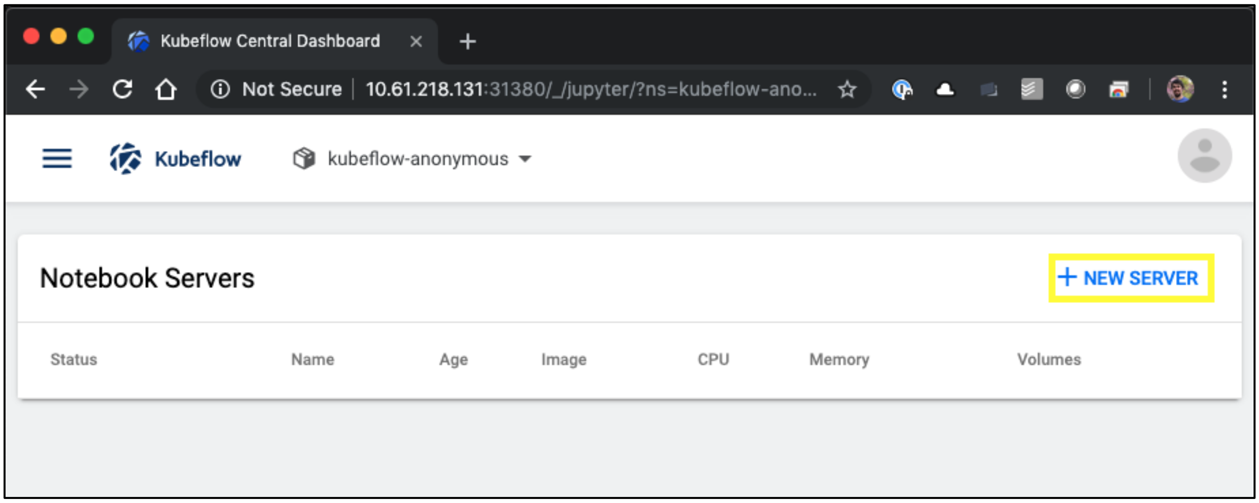
2. From the Kubeflow central dashboard, click Notebook Servers in the main menu to navigate to the Jupyter Notebook Server administration page.




3. Click NEW SERVER to provision a new Jupyter Notebook Server.

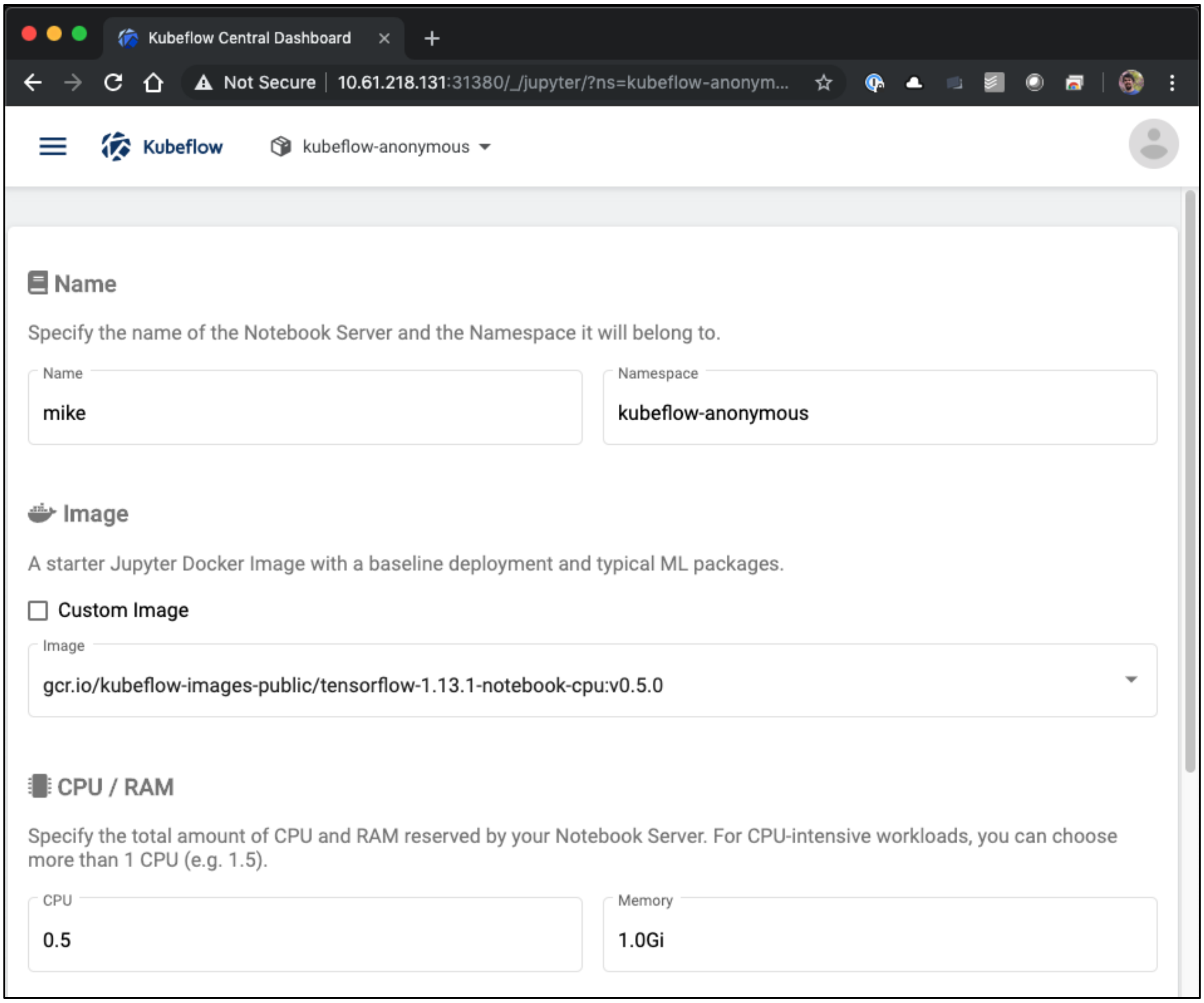
 



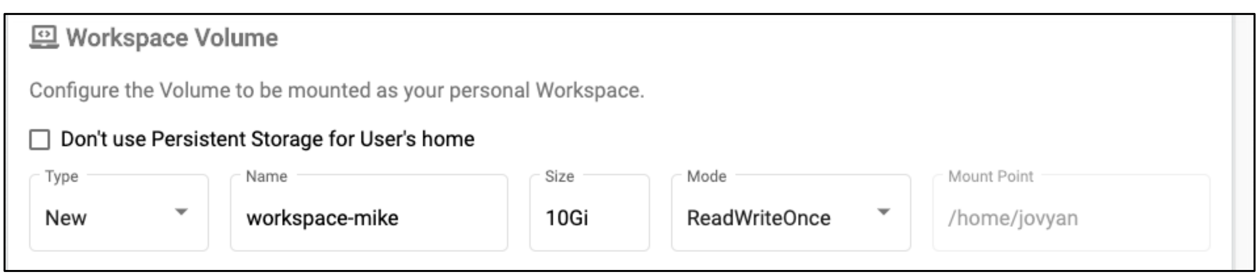

4. Give your new server a name, choose the Docker image that you want your server to be based on, and specify the amount of CPU and RAM to be reserved by your server. If the Namespace field is blank, use the Select Namespace menu in the page header to choose a namespace. The Namespace field is then auto-populated with the chosen namespace.

   In the following example, the `kubeflow-anonymous` namespace is chosen. In addition, the default values for Docker image, CPU, and RAM are accepted.

   **Note:** At the time of writing, Kubeflow only supports the `kubeflow-anonymous` namespace by default. Multiuser isolation must be configured to enable multiple namespaces within Kubeflow. For more information about multiuser isolation, see the official Kubeflow documentation.

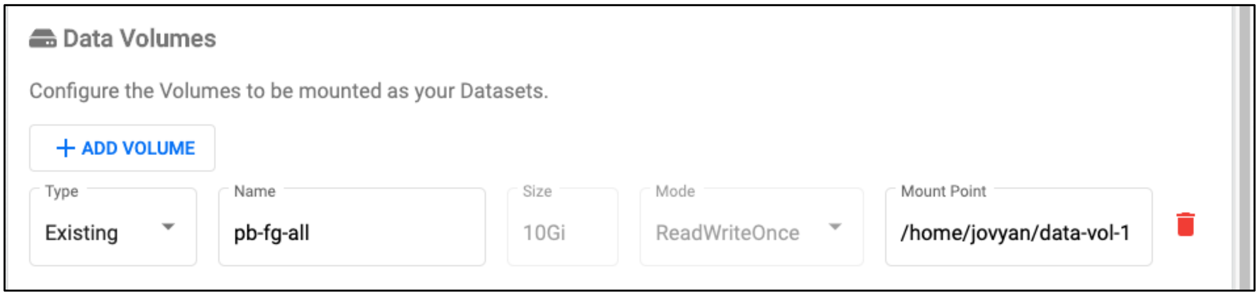



5. Specify the workspace volume details. If you choose to create a new volume, then that volume/PVC is provisioned using the default StorageClass. Because a StorageClass utilizing Trident was designated as the default StorageClass in section 5.2, the volume/PVC is provisioned with Trident. This volume is automatically mounted as the default workspace within the Jupyter Notebook Server container. Any notebooks that a user creates on the server that are not saved to a separate data volume are automatically saved to this workspace volume. Therefore, the notebooks is persistent across reboots.

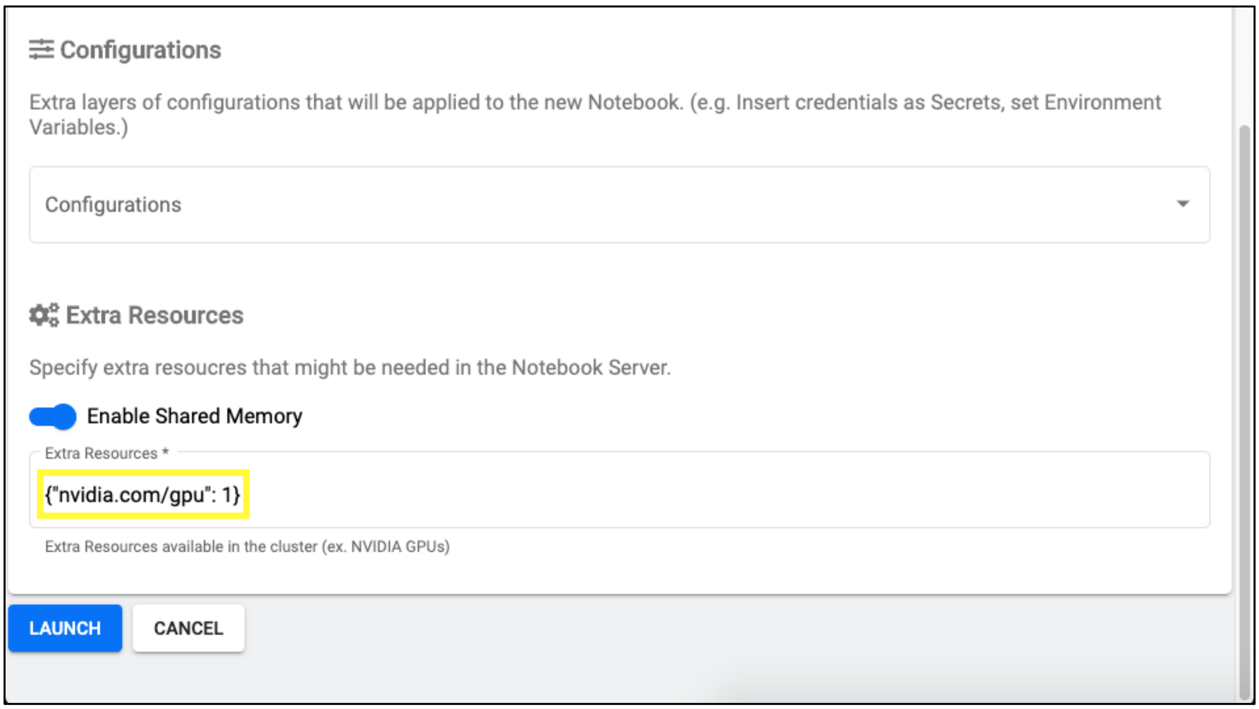



6. Add dataset volumes. The following example specifies the existing dataset volume/PVC that was imported in step 1 and accepts the default mount point.

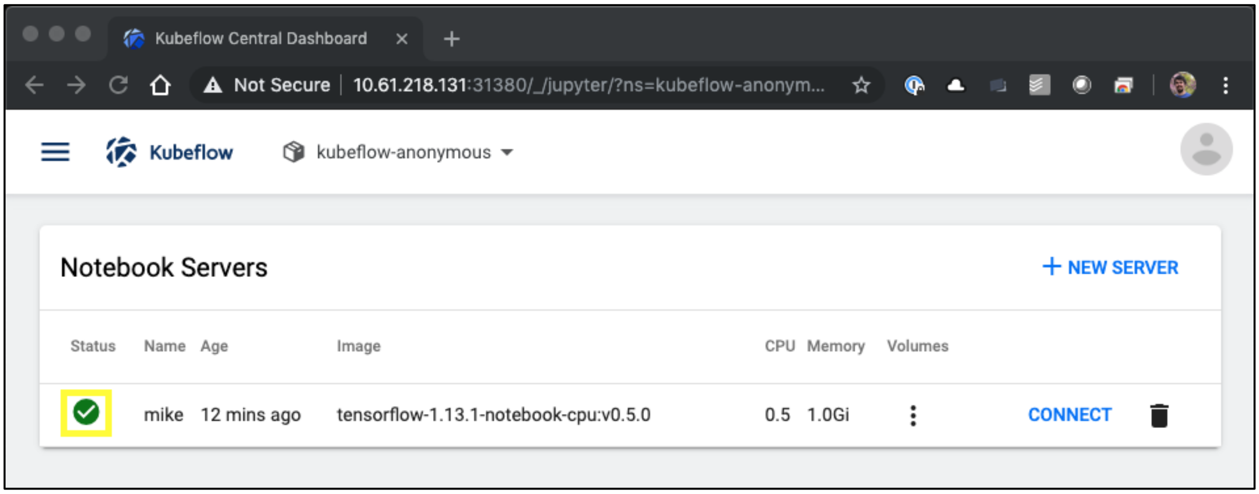



7. Request that the desired number of GPUs be allocated to your notebook server. In the following example, one GPU is requested.

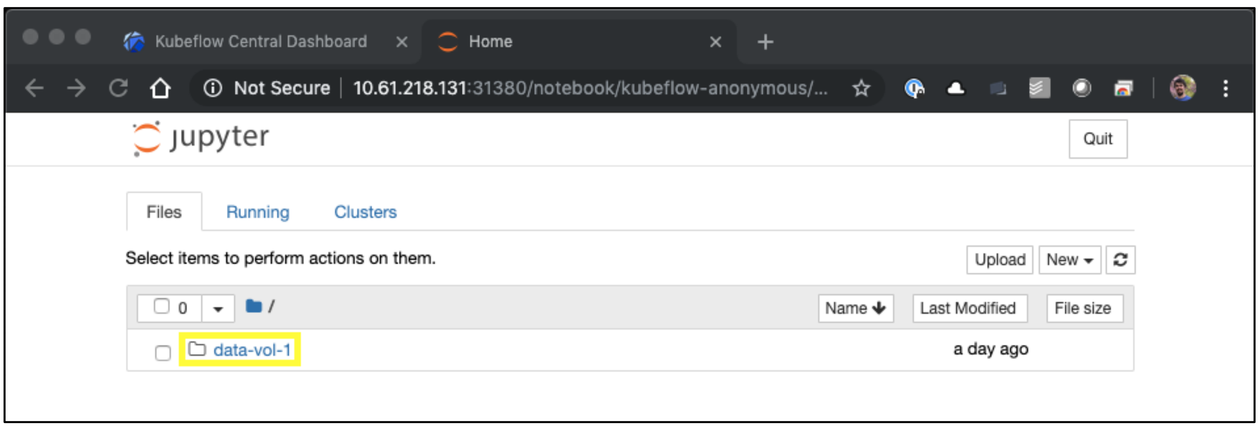



8. Click Launch to provision your new notebook server.
9. Wait for your notebook server to be fully provisioned. This can take several minutes if you have never provisioned a server using the Docker image that you specified in step 4. When your server has been fully provisioned, you see a green check-mark graphic in the Status column on the Jupyter Notebook Server administration page.

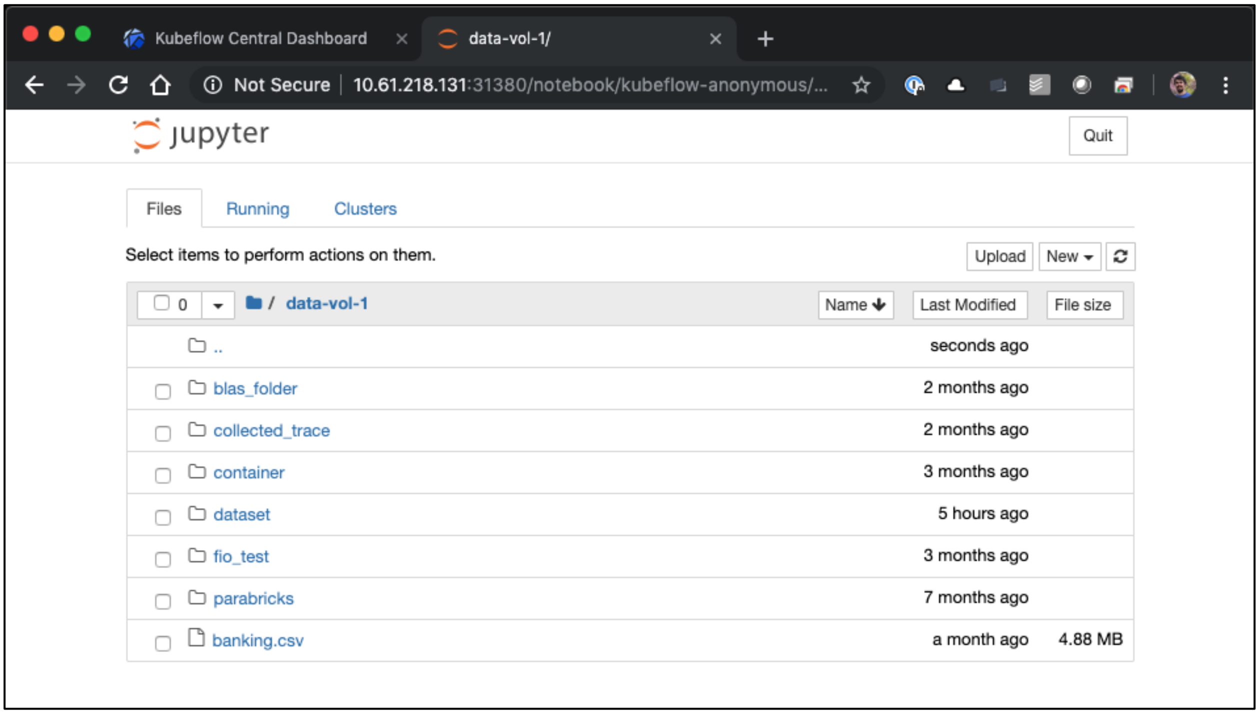
 




10. Click Connect to connect to your new server's web interface.
11. Confirm that the dataset volume that was specified in step 6 is mounted on the server. Note that this volume is mounted within the default workspace by default. From the perspective of the user, this is just another folder within the workspace. The user, who is likely a data scientist and not an infrastructure expert, does not need to possess any storage expertise in order to use this volume.

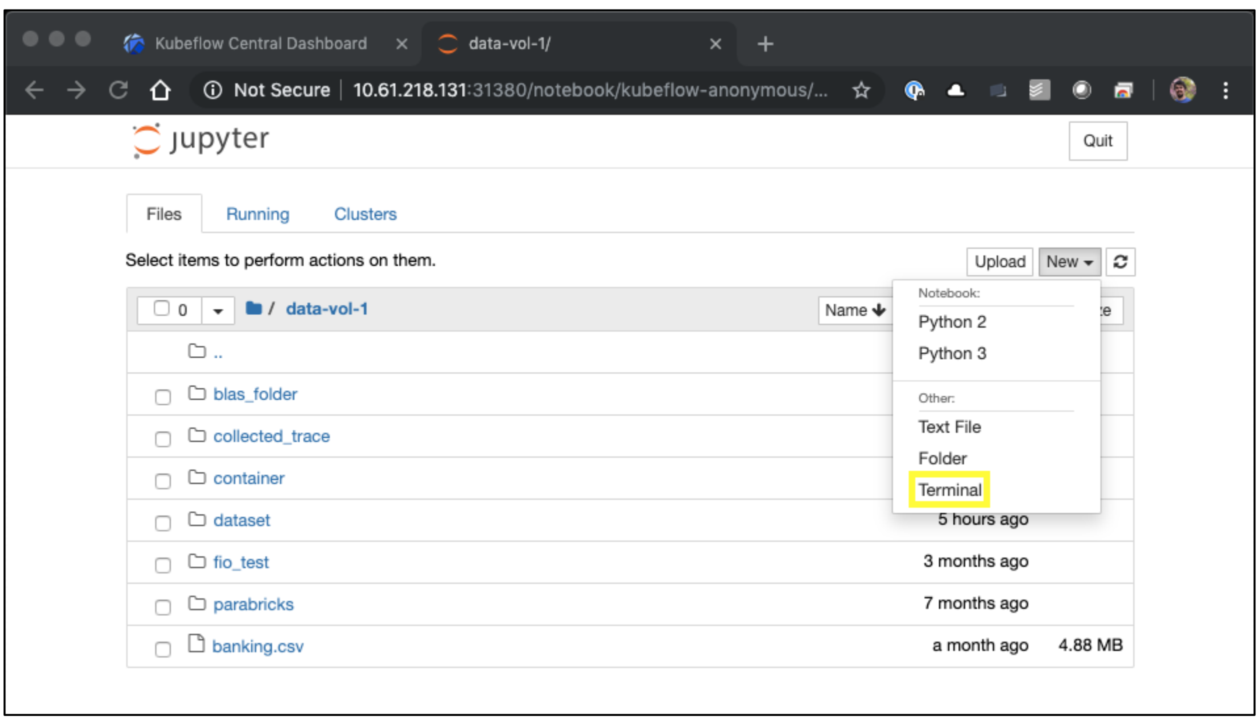


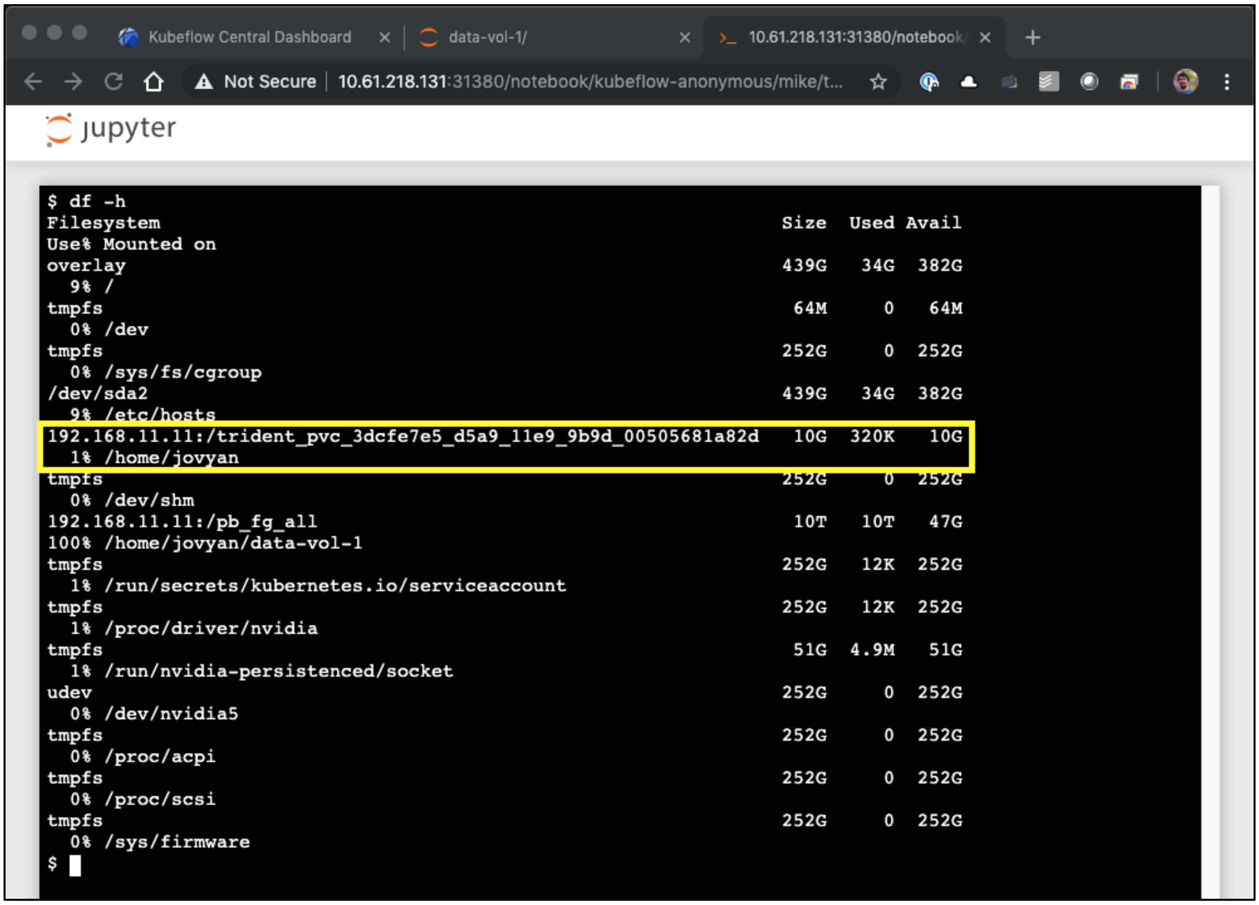



12. Open a Terminal and, assuming that a new volume was requested in step 5, execute `df -h` to confirm that a new Trident-provisioned persistent volume is mounted as the default workspace.

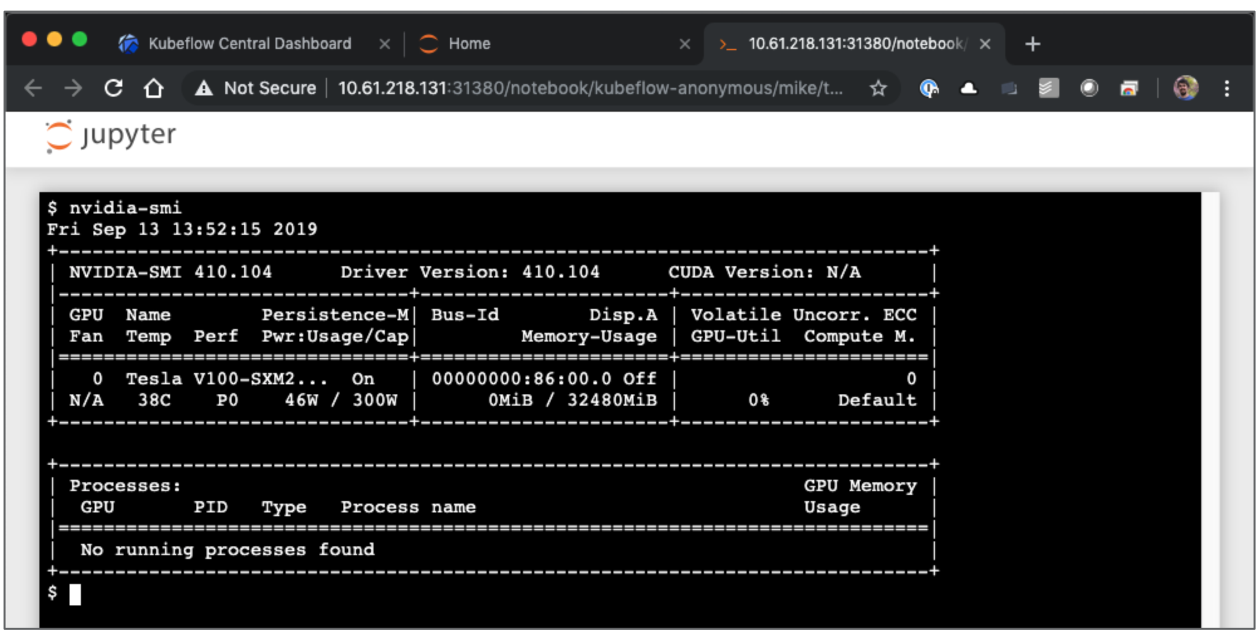
    **Note:** The default workspace directory is the base directory that you are presented with when you access the server's web interface. Therefore, any artifacts that the user creates using the web interface are stored on this Trident-provisioned persistent volume.






13. Using the terminal, execute `nvidia-smi` to confirm that the correct number of GPUs were allocated to the notebook server. In the following example, one GPU has been allocated to the notebook server as requested in step 7.



## 5.5 Create a Kubeflow Pipeline to Execute an AI Workload

To create and execute a new Kubeflow Pipeline that takes advantage of NetApp persistent storage and NetApp Snapshot technology, perform the following tasks. For more information about Kubeflow Pipelines, see the official Kubeflow documentation.

1. Use the Trident volume import functionality to import any existing dataset volumes that you want to perform operations on within your pipeline. The volume(s) must be imported in the `kubeflow` namespace because this is the namespace that pipelines are executed in.

   The example commands that follow show the importing of an existing FlexVol volume named `kfpdata`. A FlexVol volume is used here as opposed to a FlexGroup volume because the example pipeline that follows attempts to take a snapshot of this volume using Trident. At the time of writing, Trident does not support snapshots for FlexGroup volumes.

```
$ cat << EOF > ./pvc-import-kfpdata-kubeflow.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: kfpdata
  namespace: kubeflow
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexvols-retain
EOF
$ tridentctl import volume ontap-ai-flexvols kfpdata -f ./pvc-import-kfpdata-kubeflow.yaml -n
trident
+------------------------------------------+---------+--------------------------+----------+------
--------------------------------+--------+---------+
|                   NAME                   |  SIZE   |      STORAGE CLASS       | PROTOCOL |
BACKEND UUID                    | STATE  | MANAGED |
+------------------------------------------+---------+--------------------------+----------+------
--------------------------------+--------+---------+
| pvc-3c70ad14-d88f-11e9-b5e2-00505681f3d9 | 10 TiB | ontap-ai-flexvols-retain | file     |
2942d386-afcf-462e-bf89-1d2aa3376a7b | online | true    |
+------------------------------------------+---------+--------------------------+----------+------
--------------------------------+--------+---------+
$ kubectl get pvc -n kubeflow
NAME                                       STATUS    VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS               AGE
imagenet-benchmark-job-gblgq-kfpresults    Bound     pvc-a4e32212-d65c-11e9-a043-00505681a82d   1Gi
RWX            ontap-ai-flexvols-retain    2d19h
katib-mysql                                Bound     pvc-b07f293e-d028-11e9-9b9d-00505681a82d
10Gi       RWO            ontap-ai-flexvols-retain   10d
kfpdata                                    Bound     pvc-3c70ad14-d88f-11e9-b5e2-00505681f3d9
10Ti       ROX            ontap-ai-flexvols-retain   8s
metadata-mysql                             Bound     pvc-b0f3f032-d028-11e9-9b9d-00505681a82d
10Gi       RWO            ontap-ai-flexvols-retain   10d
minio-pv-claim                             Bound     pvc-b22727ee-d028-11e9-9b9d-00505681a82d
20Gi       RWO            ontap-ai-flexvols-retain   10d
mysql-pv-claim                             Bound     pvc-b2429afd-d028-11e9-9b9d-00505681a82d
20Gi       RWO            ontap-ai-flexvols-retain   10d
```

2. Define your Kubeflow Pipeline in Python using the Kubeflow Pipelines SDK. The example commands that follow show the creation of a pipeline definition for a pipeline that executes the following steps:
   a. Uses Trident to provision a new FlexVol volume. This new volume is used to store training results.
   b. Uses Trident to take a snapshot, using NetApp Snapshot technology, of the dataset volume that was imported in step 1.
   c. Executes the same ImageNet benchmark training job that was executed in section 4.6. This time however, the dataset volume is only mounted once.
   d. Uses Trident to take a snapshot, using NetApp Snapshot technology, of the results volume that was created in step 2, sub-step a.

```
$ pip3 install kfp
Requirement already satisfied: kfp in /usr/local/lib/python3.7/site-packages (0.1.29)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/site-packages (from kfp) (3.13)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/site-packages (from
kfp) (2.8.0)
Requirement already satisfied: google-auth>=1.6.1 in /usr/local/lib/python3.7/site-packages (from
kfp) (1.6.3)
Requirement already satisfied: urllib3<1.25,>=1.15 in /usr/local/lib/python3.7/site-packages
(from kfp) (1.24.1)
Requirement already satisfied: tabulate==0.8.3 in /usr/local/lib/python3.7/site-packages (from
kfp) (0.8.3)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.7/site-packages (from kfp)
(1.2.2)
Requirement already satisfied: kfp-server-api<=0.1.25,>=0.1.18 in /usr/local/lib/python3.7/site-
packages (from kfp) (0.1.18.3)
Requirement already satisfied: kubernetes<=9.0.0,>=8.0.0 in /usr/local/lib/python3.7/site-
packages (from kfp) (9.0.0)
Requirement already satisfied: argo-models==2.2.1a in /usr/local/lib/python3.7/site-packages
(from kfp) (2.2.1a0)
Requirement already satisfied: Deprecated in /usr/local/lib/python3.7/site-packages (from kfp)
(1.2.6)
Requirement already satisfied: cryptography>=2.4.2 in /usr/local/lib/python3.7/site-packages
(from kfp) (2.5)
Requirement already satisfied: click==7.0 in /usr/local/lib/python3.7/site-packages (from kfp)
(7.0)
Requirement already satisfied: google-cloud-storage>=1.13.0 in /usr/local/lib/python3.7/site-
packages (from kfp) (1.19.0)
Requirement already satisfied: requests-toolbelt>=0.8.0 in /usr/local/lib/python3.7/site-packages
(from kfp) (0.9.1)
Requirement already satisfied: jsonschema>=3.0.1 in /usr/local/lib/python3.7/site-packages (from
kfp) (3.0.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/site-packages (from kfp)
(2018.11.29)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/site-packages (from kfp)
(1.12.0)
Requirement already satisfied: PyJWT>=1.6.4 in /usr/local/lib/python3.7/site-packages (from kfp)
(1.7.1)
Requirement already satisfied: cachetools>=2.0.0 in /usr/local/lib/python3.7/site-packages (from
google-auth>=1.6.1->kfp) (3.1.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/site-packages
(from google-auth>=1.6.1->kfp) (0.2.6)
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.7/site-packages (from google-
auth>=1.6.1->kfp) (4.0)
Requirement already satisfied: websocket-client!=0.40.0,!=0.41.*,!=0.42.*,>=0.32.0 in
/usr/local/lib/python3.7/site-packages (from kubernetes<=9.0.0,>=8.0.0->kfp) (0.56.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/site-packages (from
kubernetes<=9.0.0,>=8.0.0->kfp) (2.21.0)
Requirement already satisfied: requests-oauthlib in /usr/local/lib/python3.7/site-packages (from
kubernetes<=9.0.0,>=8.0.0->kfp) (1.2.0)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.7/site-packages (from
kubernetes<=9.0.0,>=8.0.0->kfp) (41.0.1)
Requirement already satisfied: wrapt<2,>=1.10 in
/Users/moglesby/Library/Python/3.7/lib/python/site-packages (from Deprecated->kfp) (1.11.2)
Requirement already satisfied: cffi!=1.11.3,>=1.8 in /usr/local/lib/python3.7/site-packages (from
cryptography>=2.4.2->kfp) (1.11.5)
Requirement already satisfied: asn1crypto>=0.21.0 in /usr/local/lib/python3.7/site-packages (from
cryptography>=2.4.2->kfp) (0.24.0)
Requirement already satisfied: google-cloud-core<2.0dev,>=1.0.3 in /usr/local/lib/python3.7/site-
packages (from google-cloud-storage>=1.13.0->kfp) (1.0.3)
Requirement already satisfied: google-resumable-media>=0.3.1 in /usr/local/lib/python3.7/site-
packages (from google-cloud-storage>=1.13.0->kfp) (0.4.0)
Requirement already satisfied: pyrsistent>=0.14.0 in /usr/local/lib/python3.7/site-packages (from
jsonschema>=3.0.1->kfp) (0.15.4)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/site-packages (from
jsonschema>=3.0.1->kfp) (19.1.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/site-packages
(from pyasn1-modules>=0.2.1->google-auth>=1.6.1->kfp) (0.4.7)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/site-packages
(from requests->kubernetes<=9.0.0,>=8.0.0->kfp) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.7/site-packages (from
requests->kubernetes<=9.0.0,>=8.0.0->kfp) (2.8)
```

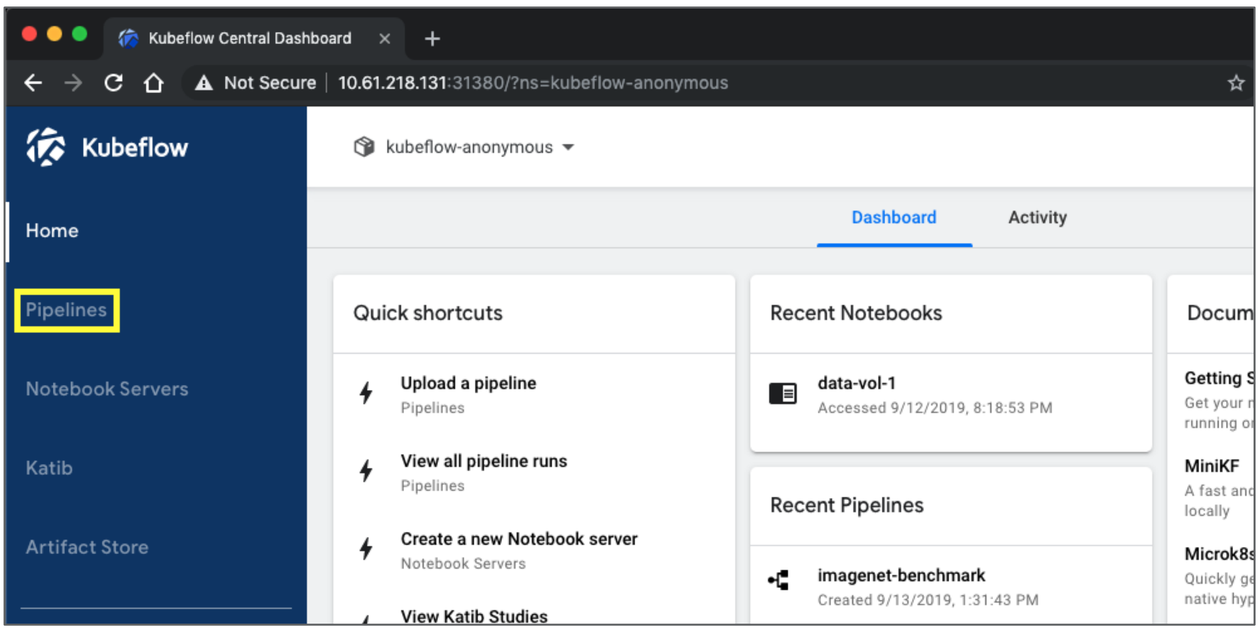
 

```
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/site-packages (from
requests-oauthlib->kubernetes<=9.0.0,>=8.0.0->kfp) (3.1.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/site-packages (from
cffi!=1.11.3,>=1.8->cryptography>=2.4.2->kfp) (2.19)
Requirement already satisfied: google-api-core<2.0.0dev,>=1.14.0 in
/usr/local/lib/python3.7/site-packages (from google-cloud-core<2.0dev,>=1.0.3->google-cloud-
storage>=1.13.0->kfp) (1.14.2)
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in
/usr/local/lib/python3.7/site-packages (from google-api-core<2.0.0dev,>=1.14.0->google-cloud-
core<2.0dev,>=1.0.3->google-cloud-storage>=1.13.0->kfp) (1.6.0)
Requirement already satisfied: protobuf>=3.4.0 in /usr/local/lib/python3.7/site-packages (from
google-api-core<2.0.0dev,>=1.14.0->google-cloud-core<2.0dev,>=1.0.3->google-cloud-
storage>=1.13.0->kfp) (3.9.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.7/site-packages (from google-api-
core<2.0.0dev,>=1.14.0->google-cloud-core<2.0dev,>=1.0.3->google-cloud-storage>=1.13.0->kfp)
(2019.2)
$ cat << EOF > ./imagenet-benchmark-pipeline.py
# Kubeflow Pipeline Definition: imagenet-benchmark-pipeline

import kfp.dsl as dsl
import kfp.onprem as onprem
import kubernetes.client.models as models
import datetime

@dsl.pipeline(
    # Define pipeline metadata
    name="ImageNet Benchmark Job",
    description="Demonstrate a full training pipeline"
)
def imagenet_benchmark(
    # Define variables that the user can set in the pipelines UI; set default values
    container_image="netapp/tensorflow-py2:19.03.0",
    dataset_volume_pvc_existing="kfpdata",
    dataset_volume_mountpoint="/mnt/mount_0",
    dataset_dir="/mnt/mount_0/dataset/imagenet/imagenet_train_copies",
    results_volume_pvc="kfpresults",
    results_volume_size="1Gi",
    dgx_version="dgx1"
):
    num_gpu = 8

    # create results volume/pvc with Trident
    results_volume = dsl.VolumeOp(
        name="create_results_vol",
        resource_name=results_volume_pvc,
        size=results_volume_size,
        modes=dsl.VOLUME_MODE_RWM # ReadWriteMany
    )

    # Take a snapshot of the dataset volume/pvc
    dataset_snapshot = dsl.VolumeSnapshotOp(
        name="dataset_vol_snapshot",
        resource_name="dataset",
        pvc=dataset_volume_pvc_existing,
        snapshot_class="csi-snapclass"
    )

    # Execute ImageNet benchmark training job
    train = dsl.ContainerOp(
        name="train",
        image=container_image,
        command=["python", "/netapp/scripts/run.py",
            "--dataset_dir", dataset_dir,
            "--dgx_version", dgx_version,
            "--num_devices", str(num_gpu),
            "--num_mounts=1"],
        pvolumes={"/tmp": results_volume.volume}
    )
    # Mount dataset volume/pvc
    train.apply(
        onprem.mount_pvc(dataset_volume_pvc_existing, 'datavol', dataset_volume_mountpoint)
```

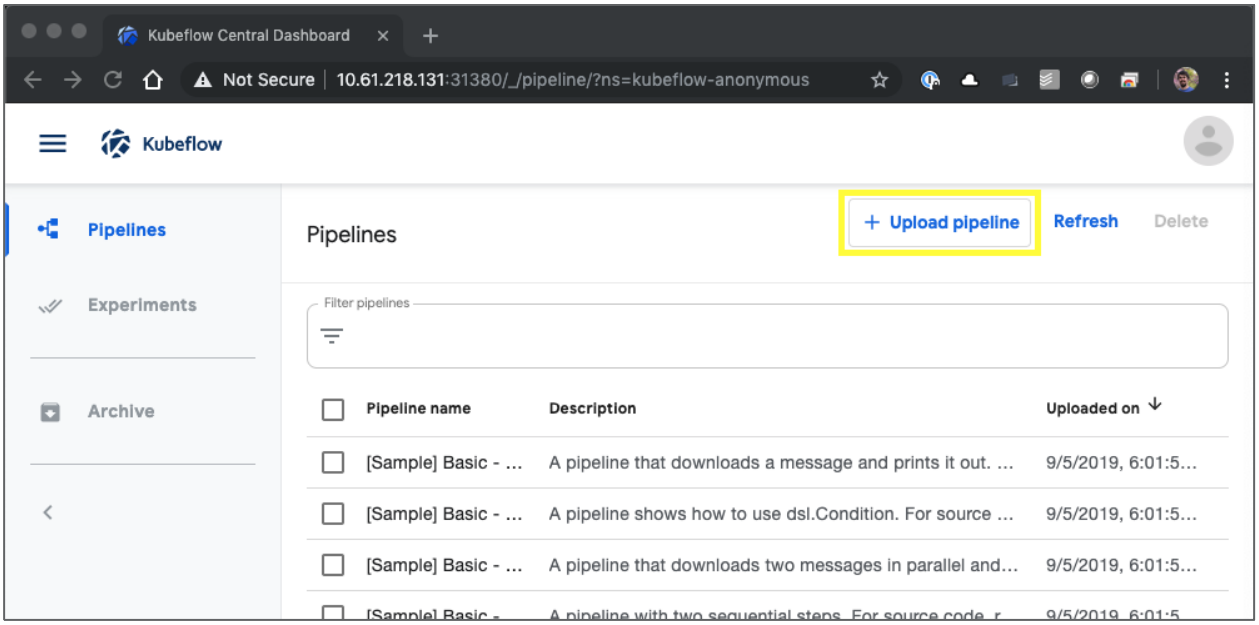
```
    )
    # Set security context of pod
    train.set_security_context(security_context = models.V1SecurityContext(privileged=True))
    # Request that GPUs be allocated to pod
    train.set_gpu_limit(num_gpu, 'nvidia')
    # State that training job should be executed after dataset volume snapshot is taken
    train.after(dataset_snapshot)

    # Take a snapshot of the results volume/pvc
    results_snapshot = dsl.VolumeSnapshotOp(
        name="results_vol_snapshot",
        resource_name="results",
        volume=train.pvolumes["/tmp"],
        snapshot_class="csi-snapclass"
    )

if __name__ == "__main__":
    import kfp.compiler as compiler
    compiler.Compiler().compile(imagenet_benchmark, __file__ + ".tar.gz")
EOF
$ python3 imagenet-benchmark-pipeline.py
$ ls imagenet-benchmark-pipeline.py.tar.gz
imagenet-benchmark-pipeline.py.tar.gz
```

3. From the Kubeflow central dashboard, click Pipelines in the main menu to navigate to the Kubeflow Pipelines administration page.

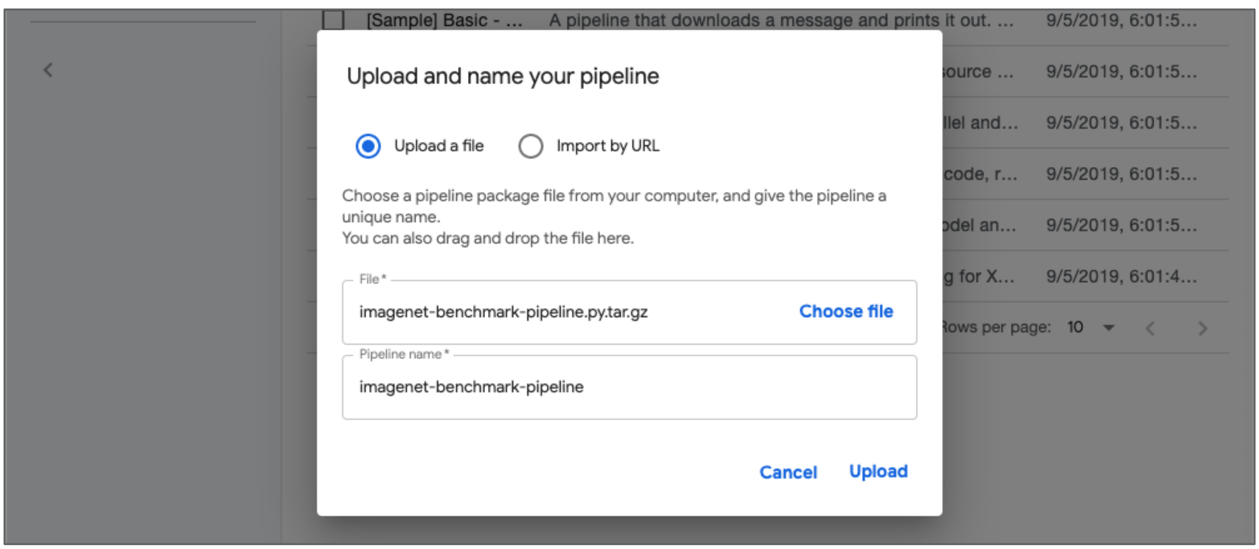



4. Click Upload Pipeline to upload your pipeline definition.

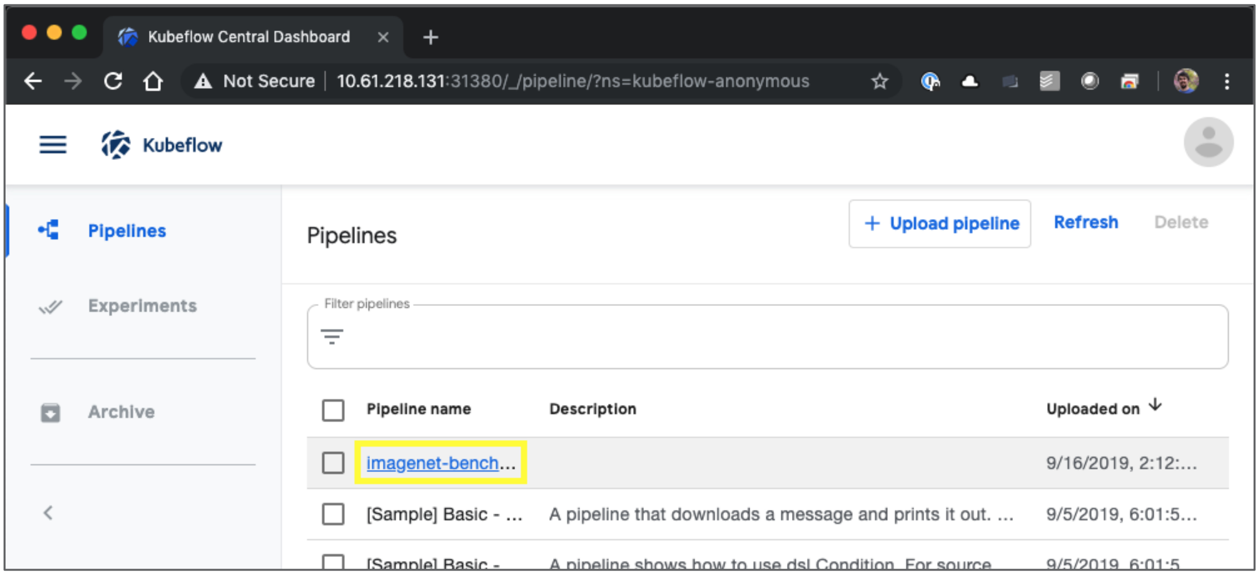



5. Choose the `.tar.gz` archive containing your pipeline definition that you created in step 2, give your pipeline a name, and click Upload.

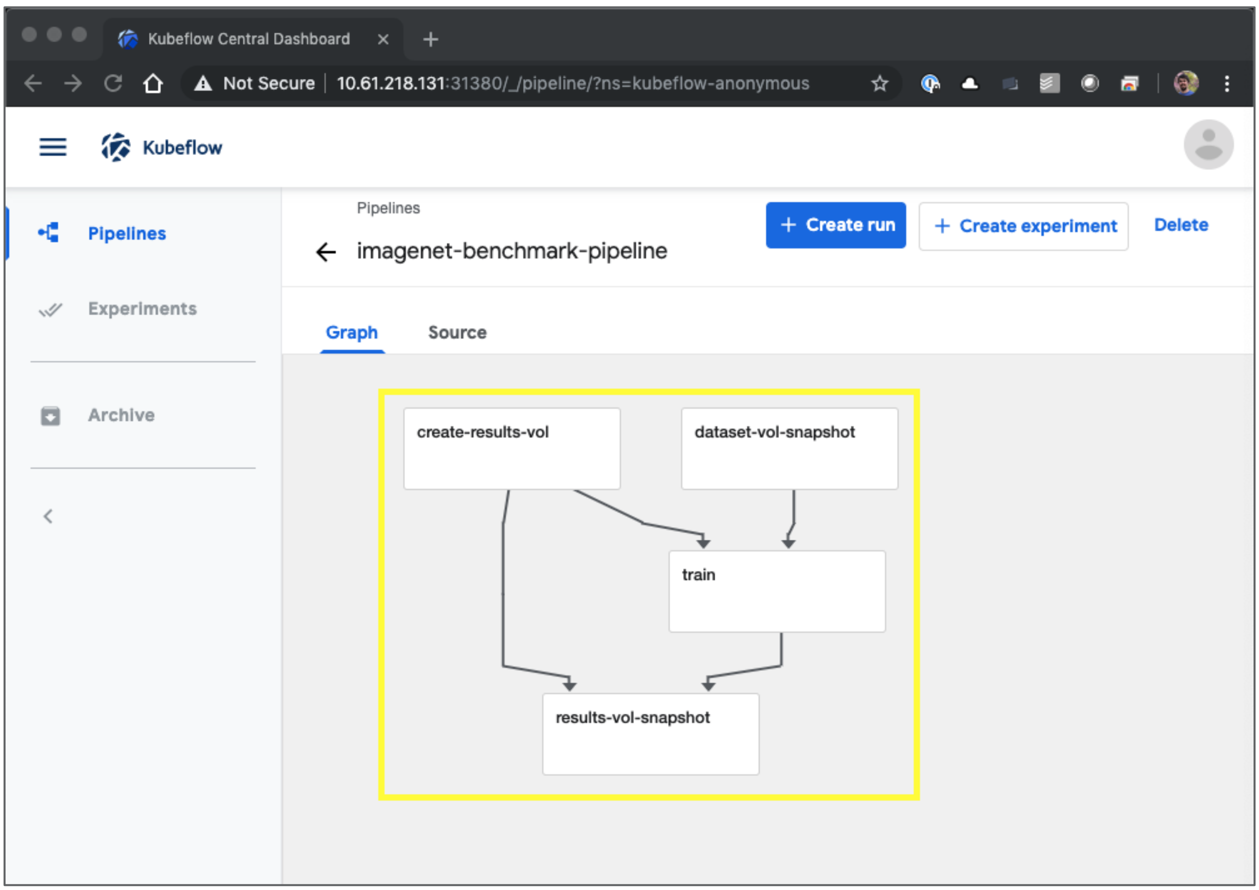



6. You should now see your new pipeline in the list of pipelines on the pipeline administration page. Click your pipeline's name to view it.

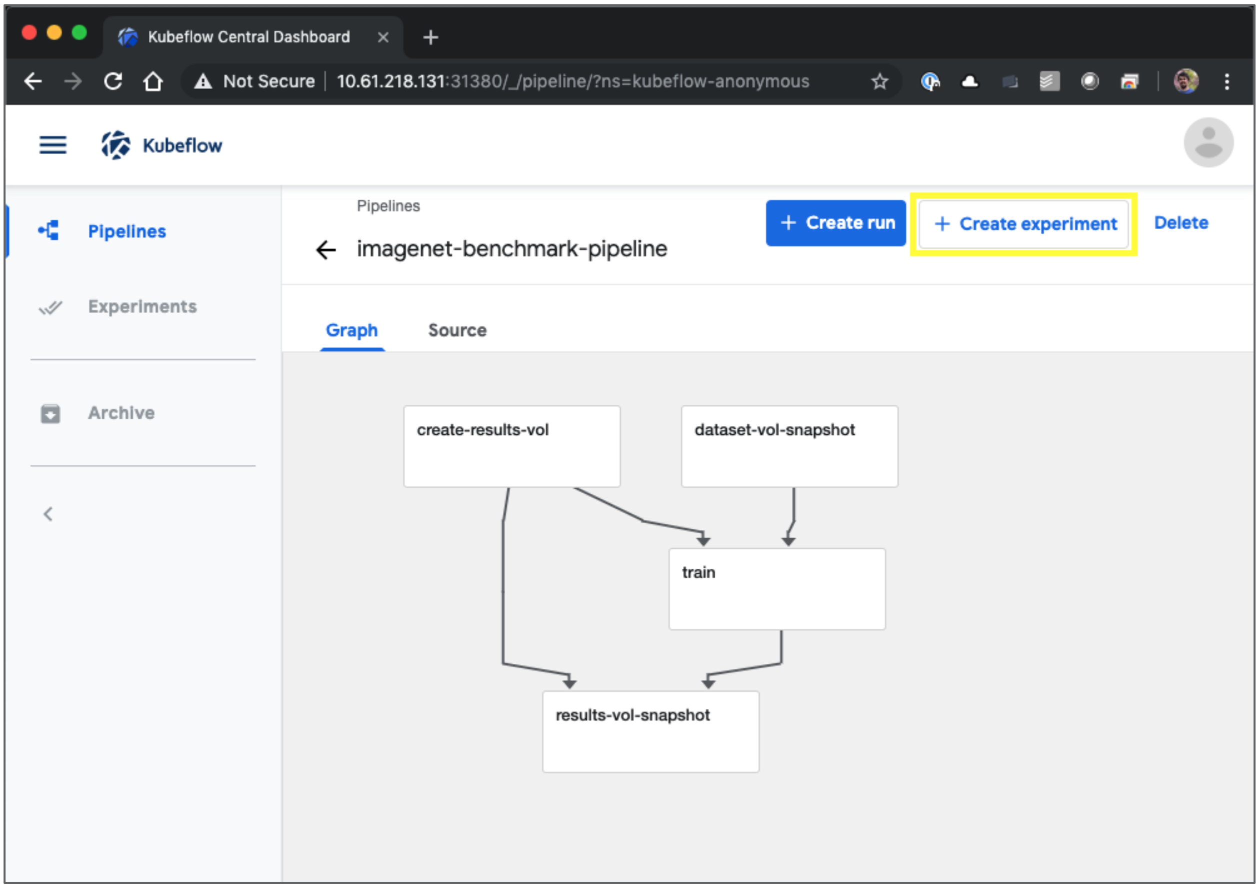



7. Review your pipeline to confirm that it looks correct.

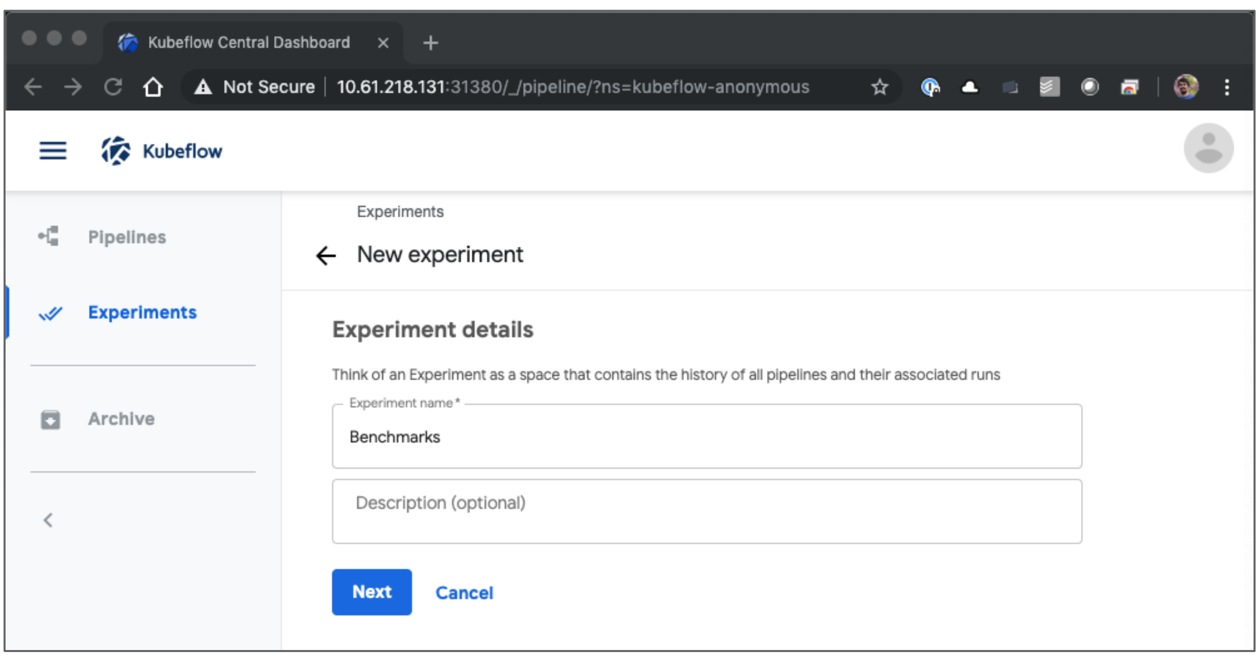



8. Click Create Experiment to create a new experiment. An experiment is a workspace in which you can run your pipelines. For more information, see the [official Kubeflow documentation](#).

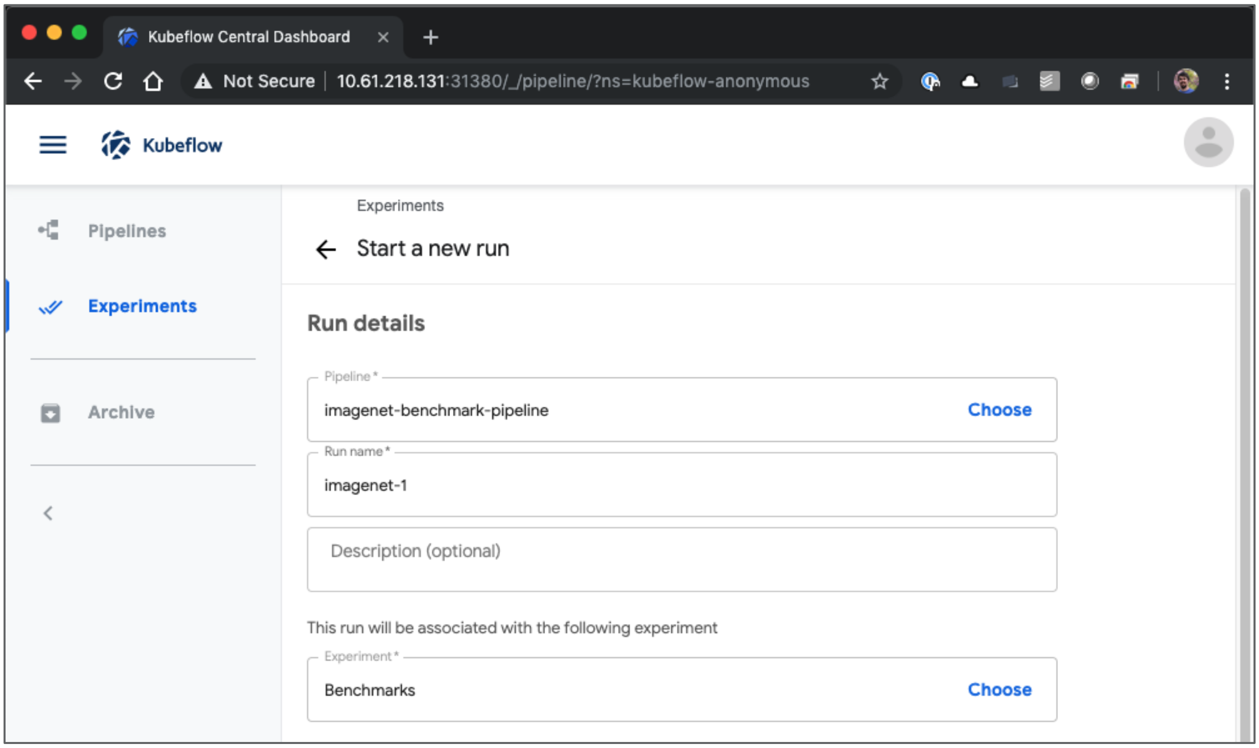



9. Give your experiment a name and then click Next.



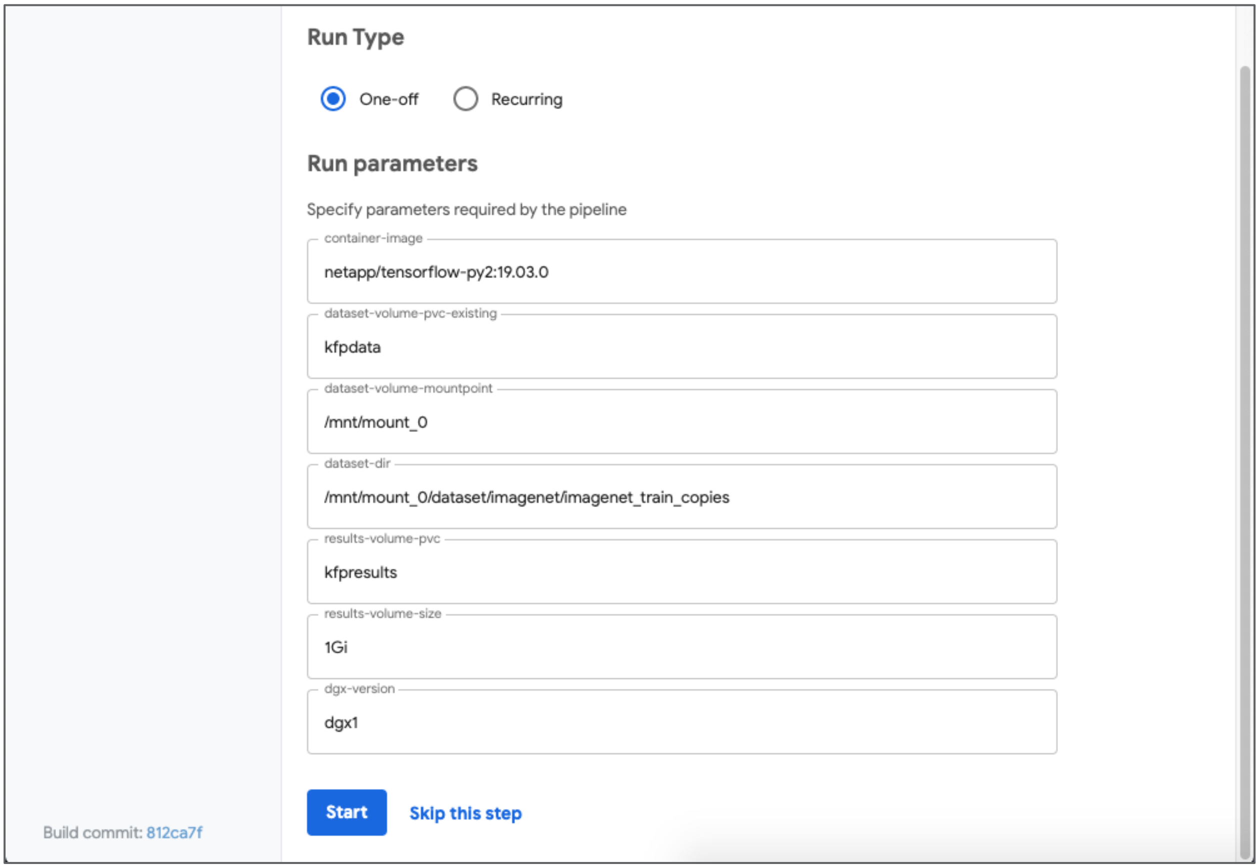

10. You are now presented with a screen from which you can start a pipeline run within your new experiment. Create a name for the run.



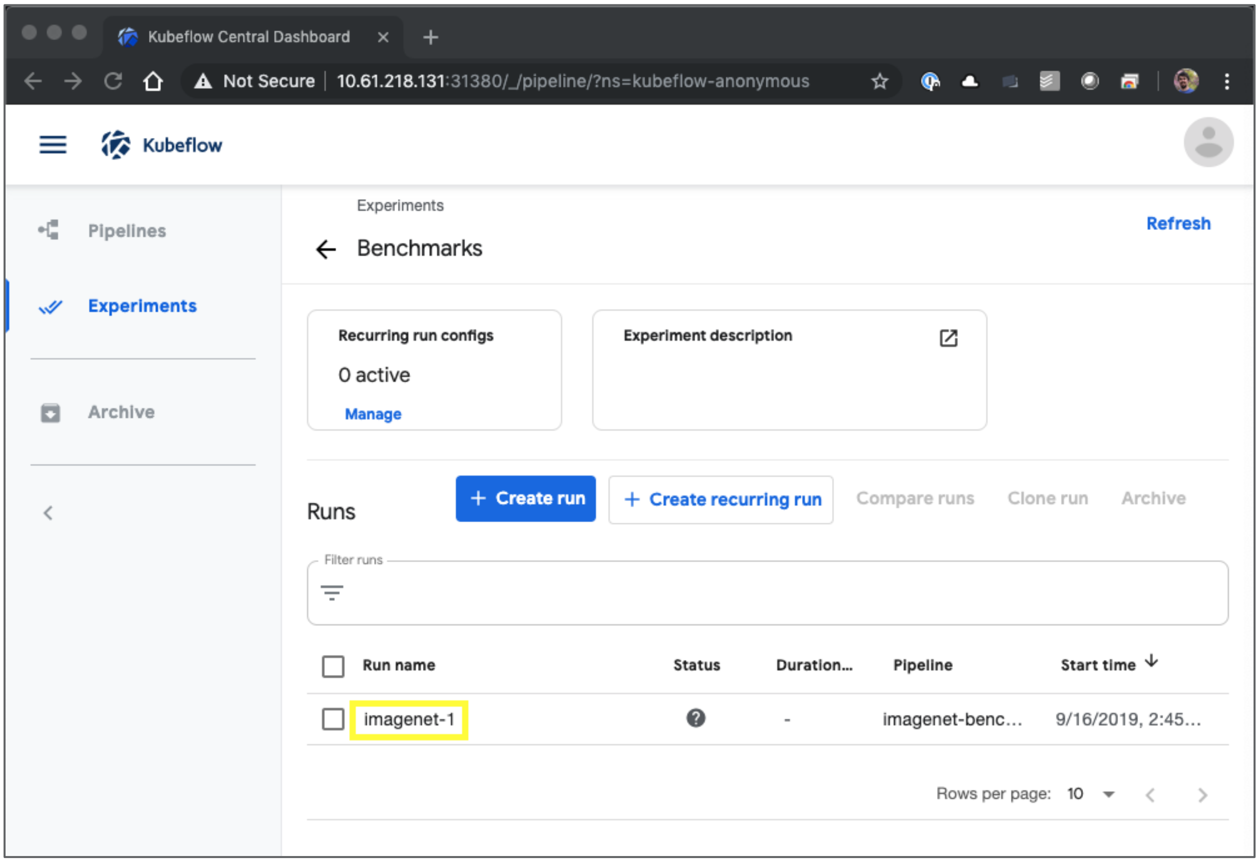

11. Define parameters for the run, and then click Start. In the following example, the default values are accepted for all parameters. Note that you defined the default values for the parameters within your pipeline definition (see step 2).

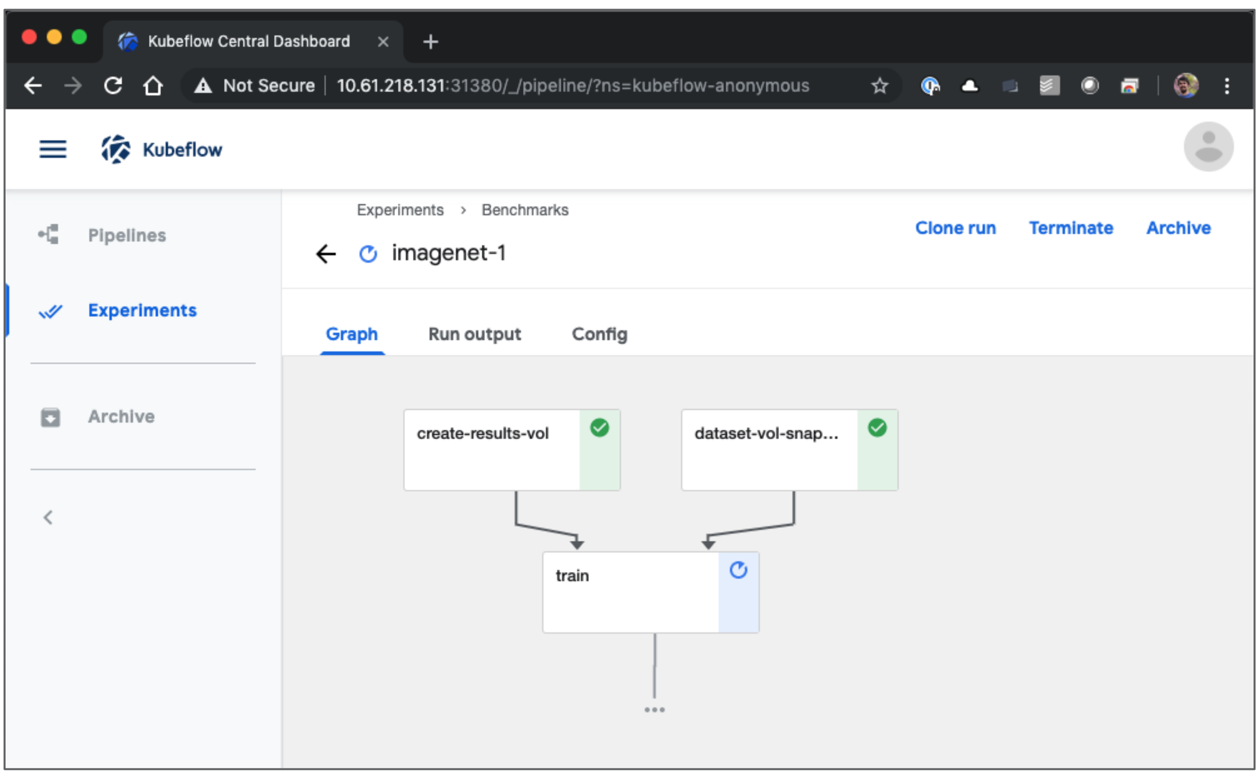



12. You are now presented with a screen listing all runs that fall under the specific experiment. Click the name of the run that you just started to view it.

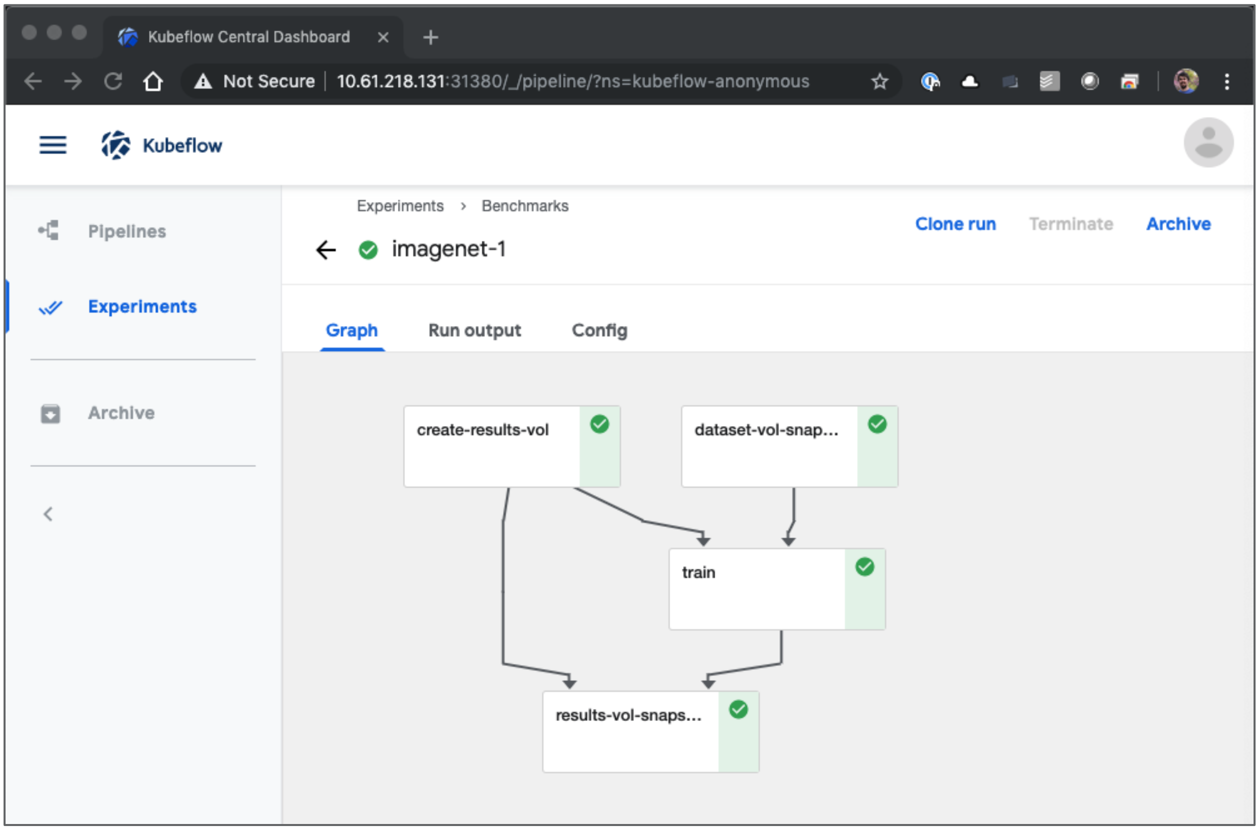



13. At this point, the run is likely still in progress.



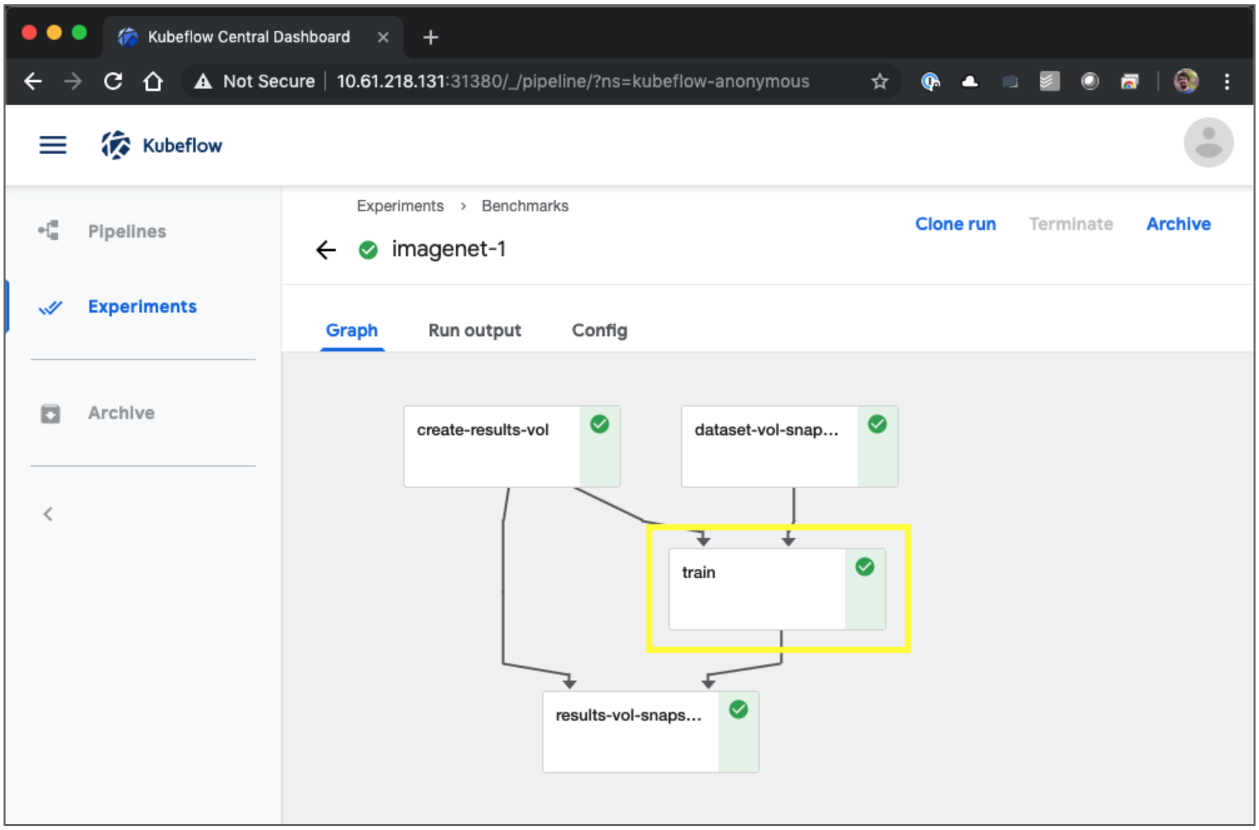

 

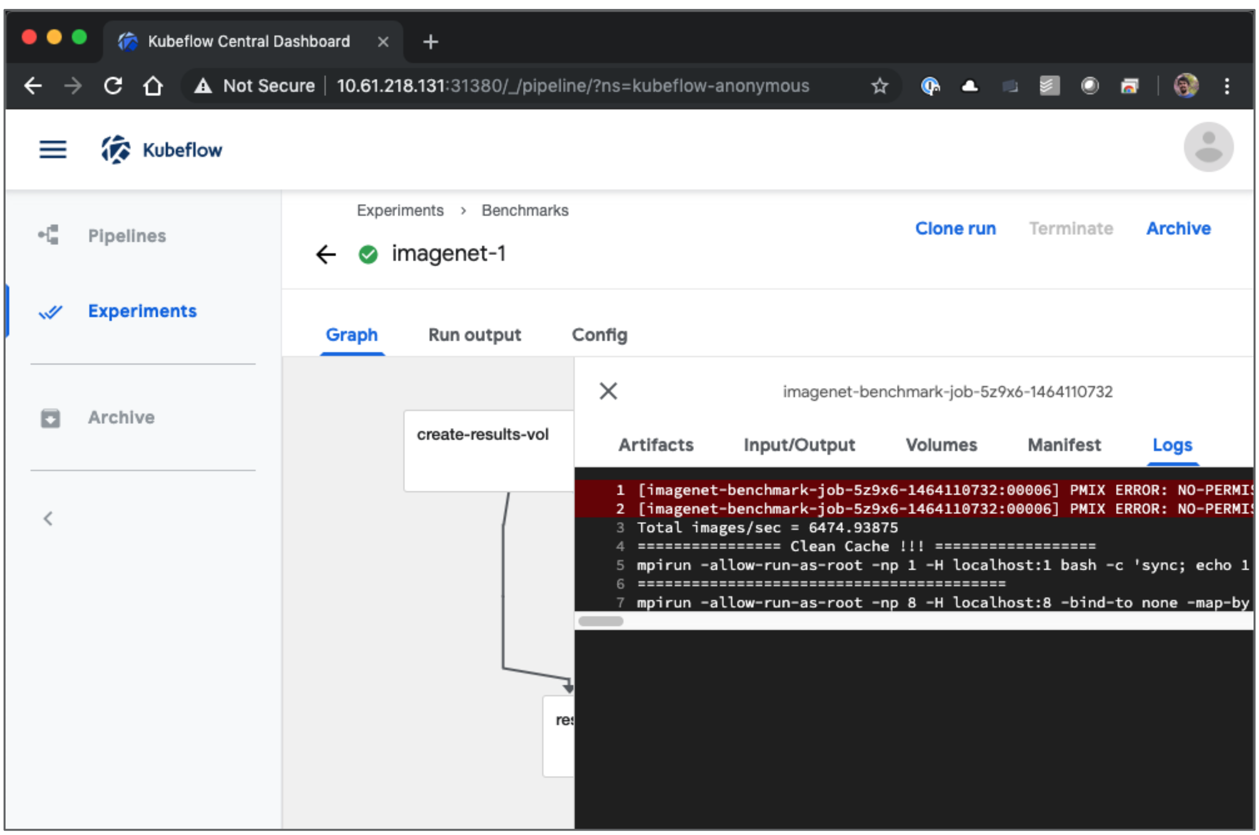
14. Confirm that the run completed successfully. When the run is complete, every stage of the pipeline shows a green check-mark icon.




15. Click the training stage, and then click on Logs to view logs for the training run.


Kubeflow Central Dashboard
Not Secure
10.61.218.131:31380/_/pipeline/?ns=kubeflow-anonymous
Kubeflow
Pipelines
Experiments
Archive
Experiments › Benchmarks
imagenet-1
Clone run
Terminate
Archive
Graph
Run output
Config
create-results-vol
dataset-vol-snap...
train
results-vol-snaps...


Kubeflow Central Dashboard
Not Secure
10.61.218.131:31380/_/pipeline/?ns=kubeflow-anonymous
Kubeflow
Pipelines
Experiments
Archive
Experiments › Benchmarks
imagenet-1
Clone run
Terminate
Archive
Graph
Run output
Config
create-results-vol
imagenet-benchmark-job-5z9x6-1464110732
Artifacts
Input/Output
Volumes
Manifest
Logs
1 [imagenet-benchmark-job-5z9x6-1464110732:00006] PMIX ERROR: NO-PERMI
2 [imagenet-benchmark-job-5z9x6-1464110732:00006] PMIX ERROR: NO-PERMI
3 Total images/sec = 6474.93875
4 =============== Clean Cache !!! =================
5 mpirun -allow-run-as-root -np 1 -H localhost:1 bash -c 'sync; echo 1
6 ========================================
7 mpirun -allow-run-as-root -np 8 -H localhost:8 -bind-to none -map-by

16. Confirm that a new results volume was provisioned during the pipeline run (see step 2, sub-step a, for details).

```
$ kubectl get pvc -n kubeflow
NAME                                         STATUS    VOLUME
CAPACITY    ACCESS MODES    STORAGECLASS                 AGE
imagenet-benchmark-job-5z9x6-kfpresults    Bound     pvc-30e882c9-d8b2-11e9-b5e2-00505681f3d9   1Gi
RWX           ontap-ai-flexvols-retain     20m
katib-mysql                                  Bound     pvc-b07f293e-d028-11e9-9b9d-00505681a82d
10Gi        RWO             ontap-ai-flexvols-retain   10d
kfpdata                                      Bound     pvc-3c70ad14-d88f-11e9-b5e2-00505681f3d9
10Ti        ROX             ontap-ai-flexvols-retain   4h30m
metadata-mysql                               Bound     pvc-b0f3f032-d028-11e9-9b9d-00505681a82d
10Gi        RWO             ontap-ai-flexvols-retain   10d
minio-pv-claim                               Bound     pvc-b22727ee-d028-11e9-9b9d-00505681a82d
20Gi        RWO             ontap-ai-flexvols-retain   10d
mysql-pv-claim                               Bound     pvc-b2429afd-d028-11e9-9b9d-00505681a82d
20Gi        RWO             ontap-ai-flexvols-retain   10d
```

17. Confirm that two snapshots were created during the pipeline run (see step 2, sub-steps b and d, for details).

```
$ kubectl get volumesnapshot -n kubeflow
NAME                                      AGE
imagenet-benchmark-job-5z9x6-dataset    22m
imagenet-benchmark-job-5z9x6-results    16m
```

# 6 Performance Testing

We performed a simple performance comparison as part of this validation exercise. We executed several standard NetApp benchmarking jobs by using Kubernetes, and we compared the benchmark results with executions that were performed by using a simple Docker run command. We did not see any noticeable differences in performance. Therefore, we concluded that the use of Kubernetes to orchestrate containerized jobs does not adversely affect performance. See Table 3 for the results of our performance comparison.

**Table 3) Performance comparison results.**

| Benchmark | Dataset | Docker Run (images/sec) | Kubernetes (images/sec) |
|---|---|---|---|
| Single-node TensorFlow | Synthetic data | 6,667.2475 | 6,661.93125 |
| Single-node TensorFlow | ImageNet | 6,570.2025 | 6,530.59125 |
| Synchronous distributed two-node TensorFlow | Synthetic data | 13,213.70625 | 13,218.288125 |
| Synchronous distributed two-node TensorFlow | ImageNet | 12,941.69125 | 12,881.33875 |

# 7 Conclusion

In today’s digital economy, AI is becoming increasingly critical for business success. As organizations increase their use of AI, they face two major challenges: data availability and workload scalability. Kubernetes and Kubeflow make it simple to deploy and scale AI workloads across multiple GPUs and nodes, and NetApp Trident provides seamless access to persistent data across nodes or regions. With Trident, you can quickly and easily make data volumes, potentially containing petabytes of data, available to Kubernetes-based workloads. Additionally, Trident is a Kubernetes-native app; no NetApp or NetApp ONTAP expertise is required.

## Acknowledgments

- David Arnette, Technical Marketing Engineer, NetApp
- Sung-Han Lin, Performance Analyst, NetApp
- Steve Guhr, Solutions Engineer, NetApp
- Muneer Ahmad, Solutions Architect, NetApp
- Nilesh Bagad, Senior Product Manager, NetApp
- Santosh Rao, Senior Technical Director, NetApp

## Where to Find Additional Information

To learn more about the information that is described in this document, see the following resources:

- NVIDIA DGX-1 servers:
  - NVIDIA DGX-1 servers
    https://www.nvidia.com/en-us/data-center/dgx-1/
  - NVIDIA Tesla V100 Tensor Core GPU
    https://www.nvidia.com/en-us/data-center/tesla-v100/
  - NVIDIA GPU Cloud (NGC)
    https://www.nvidia.com/en-us/gpu-cloud/
- NetApp AFF systems:
  - AFF datasheet
    https://www.netapp.com/us/media/ds-3582.pdf
  - NetApp FlashAdvantage for AFF
    https://www.netapp.com/us/media/ds-3733.pdf
  - ONTAP 9.x documentation
    http://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62286
  - NetApp FlexGroup technical report
    https://www.netapp.com/us/media/tr-4557.pdf
- NetApp persistent storage for containers:
  - NetApp Trident
    https://netapp.io/persistent-storage-provisioner-for-kubernetes/
- NetApp Interoperability Matrix:
  - NetApp Interoperability Matrix Tool
    http://support.netapp.com/matrix
- ONTAP AI networking:
  - Cisco Nexus 3232C Switches
    https://www.cisco.com/c/en/us/products/switches/nexus-3232c-switch/index.html
  - Mellanox Spectrum 2000 series switches
    http://www.mellanox.com/page/products_dyn?product_family=251&mtag=sn2000
- ML framework and tools:
  - DALI
    https://github.com/NVIDIA/DALI
  - TensorFlow: An Open-Source Machine Learning Framework for Everyone
    https://www.tensorflow.org/
  - Horovod: Uber's Open-Source Distributed Deep Learning Framework for TensorFlow
    https://eng.uber.com/horovod/

- Enabling GPUs in the Container Runtime Ecosystem
  https://devblogs.nvidia.com/gpu-containers-runtime/
- Docker
  https://docs.docker.com
- Kubernetes
  https://kubernetes.io/docs/home/
- NVIDIA DeepOps
  https://github.com/NVIDIA/deepops
- Kubeflow
  http://www.kubeflow.org/
- Jupyter Notebook Server
  http://www.jupyter.org/

- Dataset and benchmarks:
  - ImageNet
    http://www.image-net.org/
  - COCO
    http://cocodataset.org/
  - Cityscapes
    https://www.cityscapes-dataset.com/
  - nuScenes
    www.nuscenes.org
  - SECOND: Sparsely Embedded Convolutional Detection model
    https://pdfs.semanticscholar.org/5125/a16039cabc6320c908a4764f32596e018ad3.pdf
  - TensorFlow benchmarks
    https://github.com/tensorflow/benchmarks

# Version History

| Version | Date | Document Version History |
|---|---|---|
| Version 1.0 | September 2019 | Initial release. |
| Version 2.0 | September 2019 | Added sections on Snapshots/FlexClones (sections 4.8 - 4.10) and Kubeflow (sections 2.6 and 5.*); added Figure 4; updated DeepOps troubleshooting instructions (section 4.2, step 2). |

Refer to the [Interoperability Matrix Tool (IMT)]() on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.



NetApp