Technical Report

NetApp AI Control Plane

Pairing Popular Open-Source Tools with NetApp to Enable AI, ML, and DL Data and Experiment Management

Mike Oglesby, NetApp
October 2020 | TR-4798

Abstract

As organizations increase their use of artificial intelligence (AI), they face many challenges, including workload scalability and data availability. This document demonstrates how to address these challenges through the use of NetApp® AI Control Plane, a solution that pairs NetApp data management capabilities with popular open-source tools and frameworks that are used by data scientists and data engineers. In this document, we show you how to rapidly clone a data namespace just as you would a Git repo. We demonstrate how to define and implement AI training workflows that incorporate the near-instant creation of data and model baselines for traceability and versioning. We also show how to seamlessly replicate data across sites and regions and swiftly provision Jupyter Notebook workspaces with access to massive datasets.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

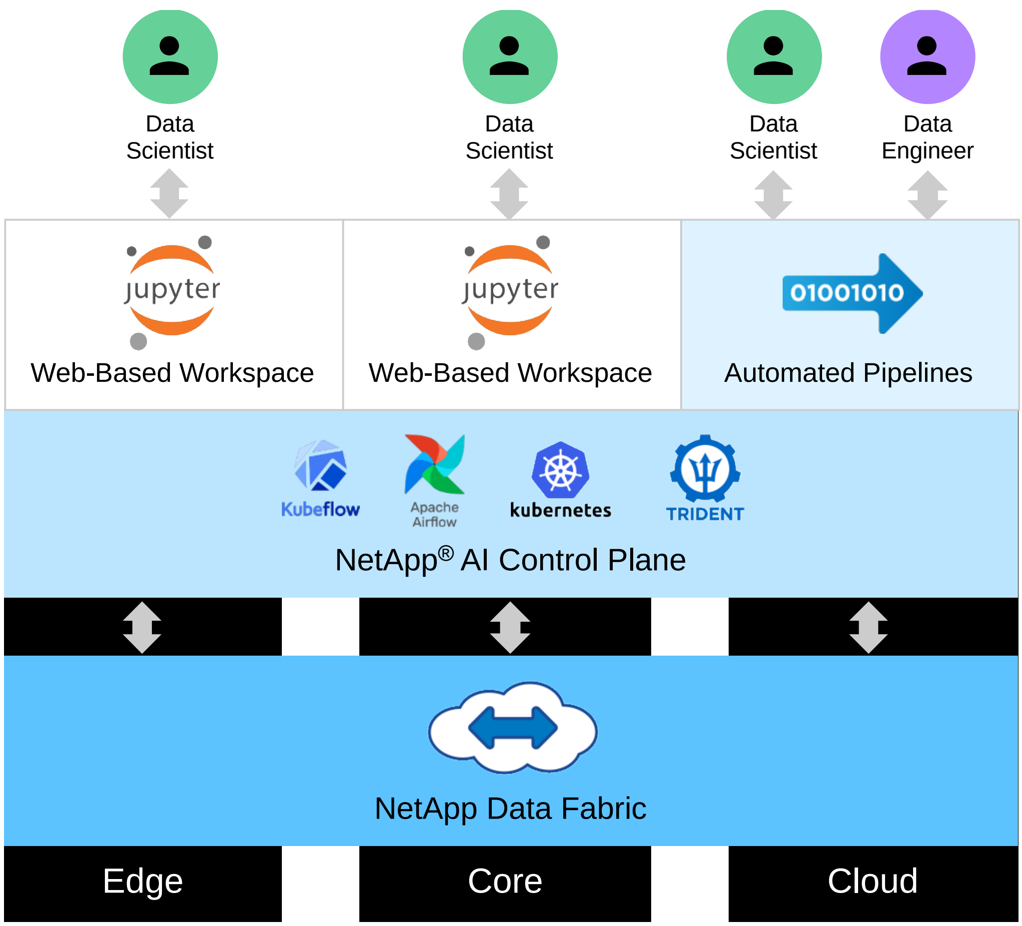
Introduction

Companies and organizations of all sizes and across many industries are turning to artificial intelligence (AI), machine learning (ML), and deep learning (DL) to solve real-world problems, deliver innovative products and services, and to get an edge in an increasingly competitive marketplace. As organizations increase their use of AI, ML, and DL, they face many challenges, including workload scalability and data availability. This document demonstrates how you can address these challenges by using the NetApp AI Control Plane, a solution that pairs NetApp data management capabilities with popular open-source tools and frameworks.

This report shows you how to rapidly clone a data namespace just as you would a Git repo. It also shows you how to seamlessly replicate data across sites and regions to create a cohesive and unified AI/ML/DL data pipeline. Additionally, it walks you through the defining and implementing of AI, ML, and DL training workflows that incorporate the near-instant creation of data and model baselines for traceability and versioning. With this solution, you can trace every model training run back to the exact dataset that was used to train and/or validate the model. Lastly, this document shows you how to swiftly provision Jupyter Notebook workspaces with access to massive datasets.

The NetApp AI Control Plane is targeted towards data scientists and data engineers, and, thus, minimal NetApp or NetApp ONTAP® expertise is required. With this solution, data management functions can be executed using simple and familiar tools and interfaces. If you already have NetApp storage in your environment, you can test drive the NetApp AI Control plane today. If you want to test drive the solution but you do not have already have NetApp storage, visit cloud.netapp.com, and you can be up and running with a cloud-based NetApp storage solution in minutes.

Figure 1) Solution visualization.



Concepts and Components

Artificial Intelligence

AI is a computer science discipline in which computers are trained to mimic the cognitive functions of the human mind. AI developers train computers to learn and to solve problems in a manner that is similar to, or even superior to, humans. Deep learning and machine learning 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

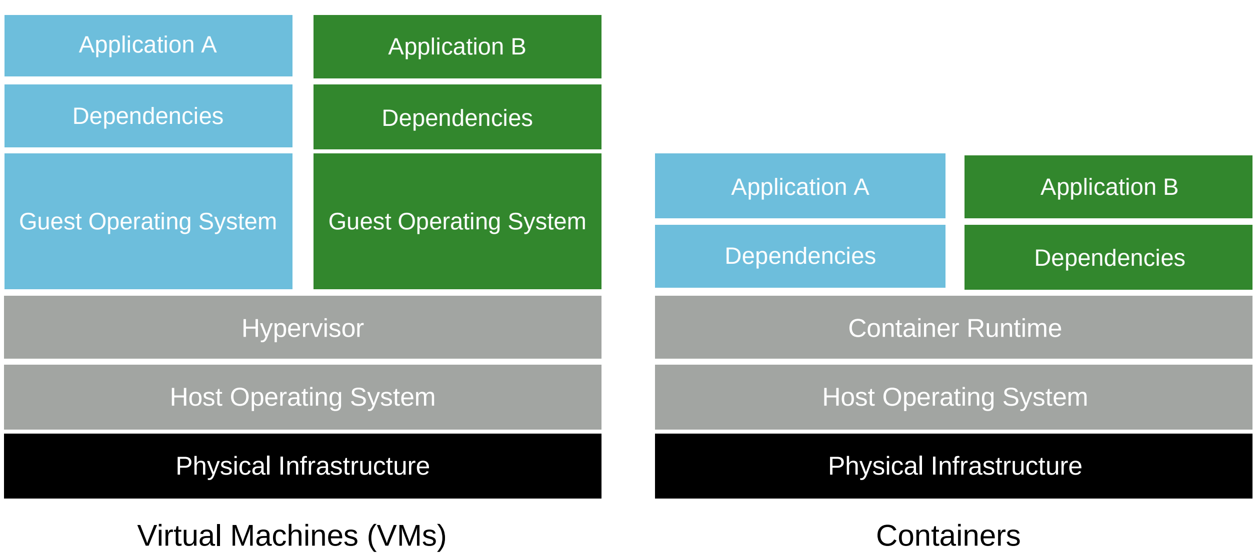
Modern AI training and inference workloads require massively parallel computing capabilities. Therefore, GPUs are increasingly being used to execute AI operations because the parallel processing capabilities of GPUs are vastly superior to those of general-purpose CPUs.

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 2 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 2) Virtual machines versus containers.



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.

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.

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.

Kubeflow

Kubeflow is an open source AI and ML toolkit for Kubernetes that was originally developed by Google. The Kubeflow project makes deployments of AI and 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 3 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.

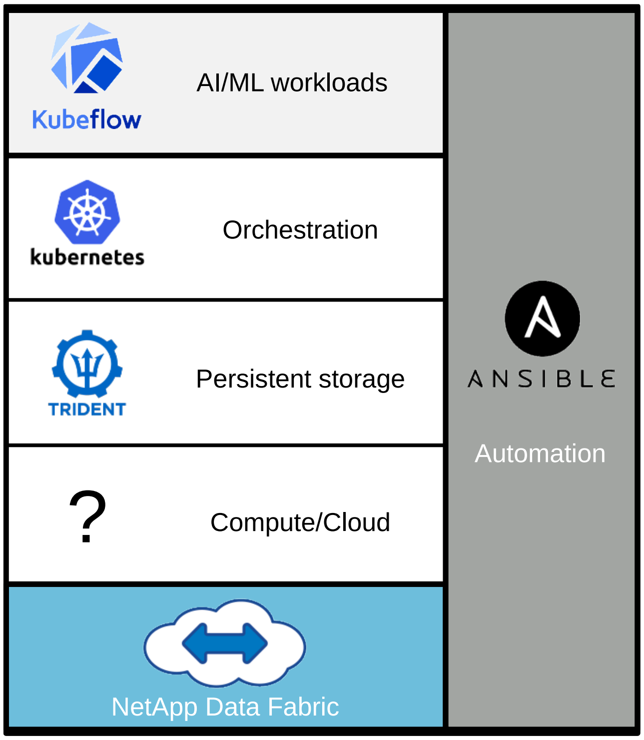
Kubeflow Pipelines

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 and 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 3) Kubeflow visualization.



Apache Airflow

Apache Airflow is an open-source workflow management platform that enables programmatic authoring, scheduling, and monitoring for complex enterprise workflows. It is often used to automate ETL and data pipeline workflows, but it is not limited to these types of workflows. The Airflow project was started by Airbnb but has since become very popular in the industry and now falls under the auspices of The Apache Software Foundation. Airflow is written in Python, Airflow workflows are created via Python scripts, and Airflow is designed under the principle of "configuration as code." Many enterprise Airflow users now run Airflow on top of Kubernetes.

Directed Acyclic Graphs (DAGs)

In Airflow, workflows are called Directed Acyclic Graphs (DAGs). DAGs are made up of tasks that are executed in sequence, in parallel, or a combination of the two, depending on the DAG definition. The Airflow scheduler executes individual tasks on an array of workers, adhering to the task-level dependencies that are specified in the DAG definition. DAGs are defined and created via Python scripts.

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.

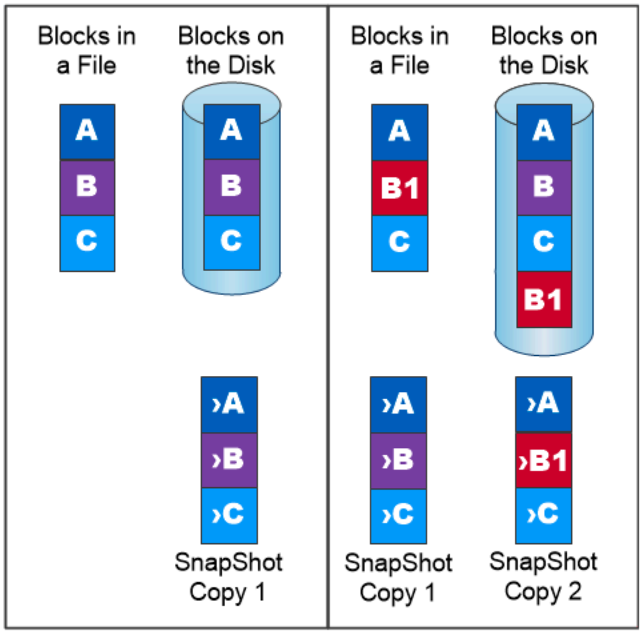
NetApp Snapshot Copies

A NetApp Snapshot™ copy is a read-only, point-in-time image of a volume. The image consumes minimal storage space and incurs negligible performance overhead because it only records changes to files create since the last Snapshot copy was made.

Snapshot copies owe their efficiency to the core ONTAP storage virtualization technology, the Write Anywhere File Layout (WAFL). Like a database, WAFL uses metadata to point to actual data blocks on disk. But, unlike a database, WAFL does not overwrite existing blocks. It writes updated data to a new block and changes the metadata. It's because ONTAP references metadata when it creates a Snapshot copy, rather than copying data blocks, that Snapshot copies are so efficient. Doing so eliminates the "seek time" that other systems incur in locating the blocks to copy, as well as the cost of making the copy itself.

You can use a Snapshot copy to recover individual files or LUNs or to restore the entire contents of a volume. ONTAP compares pointer information in the Snapshot copy with data on disk to reconstruct the missing or damaged object, without downtime or a significant performance cost.

Figure 4) NetApp Snapshot copies.

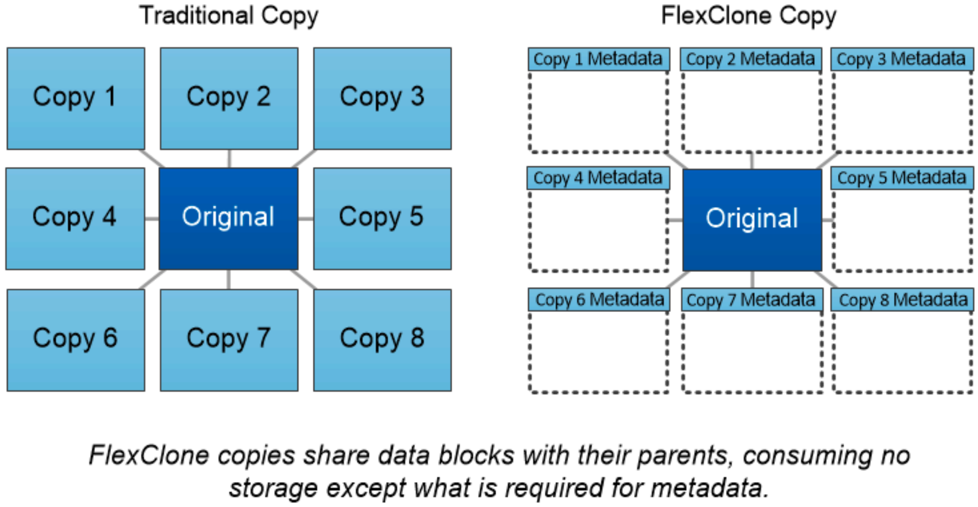


A Snapshot copy records only changes to the active file system since the last Snapshot copy.

NetApp FlexClone Technology

NetApp FlexClone® technology references Snapshot metadata to create writable, point-in-time copies of a volume. Copies share data blocks with their parents, consuming no storage except what is required for metadata, until changes are written to the copy. Where traditional copies can take minutes or even hours to create, FlexClone software lets you copy even the largest datasets almost instantaneously. That makes it ideal for situations in which you need multiple copies of identical datasets (a development workspace, for example) or temporary copies of a dataset (testing an application against a production dataset).

Figure 5) NetApp FlexClone technology.

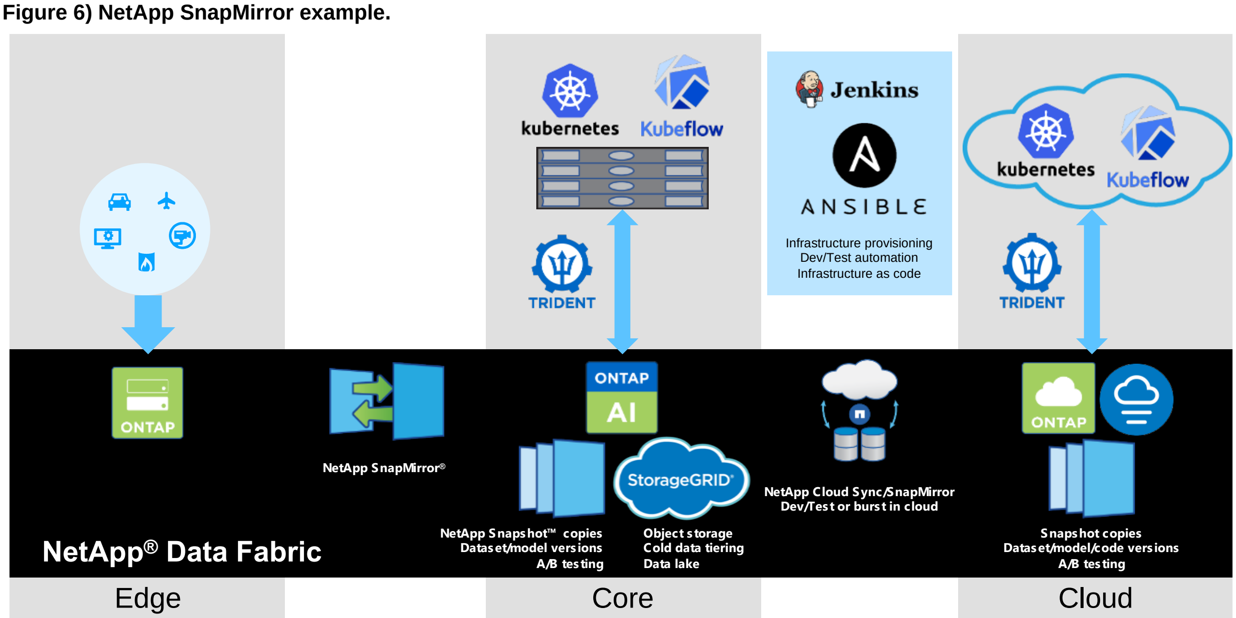


NetApp SnapMirror Data Replication Technology

NetApp SnapMirror® software is a cost-effective, easy-to-use unified replication solution across the data fabric. It replicates data at high speeds over LAN or WAN. It gives you high data availability and fast data replication for applications of all types, including business critical applications in both virtual and traditional environments. When you replicate data to one or more NetApp storage systems and continually update the secondary data, your data is kept current and is available whenever you need it. No external replication servers are required. See Figure 6 for an example of an architecture that leverages SnapMirror technology.

SnapMirror software leverages NetApp ONTAP storage efficiencies by sending only changed blocks over the network. SnapMirror software also uses built-in network compression to accelerate data transfers and reduce network bandwidth utilization by up to 70%. With SnapMirror technology, you can leverage one thin replication data stream to create a single repository that maintains both the active mirror and prior point-in-time copies, reducing network traffic by up to 50%.

Figure 6) NetApp SnapMirror example.



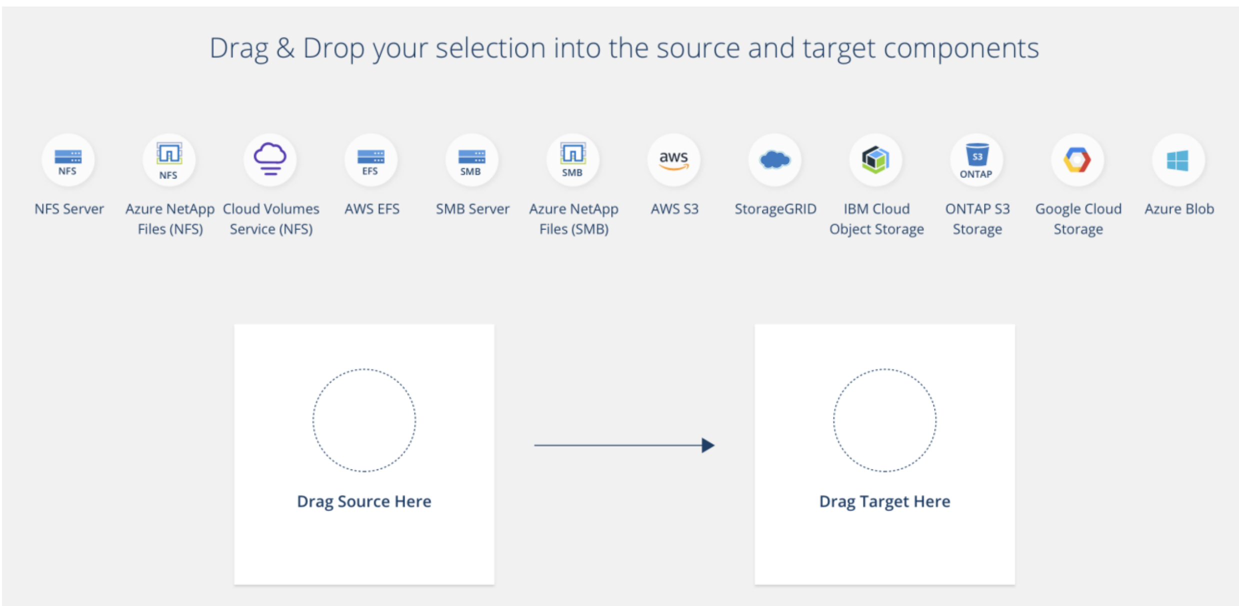
NetApp Cloud Sync

Cloud Sync is a NetApp service for rapid and secure data synchronization. Whether you need to transfer files between on-premises NFS or SMB file shares, NetApp StorageGRID, NetApp ONTAP S3, NetApp Cloud Volumes Service, Azure NetApp Files, AWS S3, AWS EFS, Azure Blob, Google Cloud Storage, or IBM Cloud Object Storage, Cloud Sync moves the files where you need them quickly and securely.

After your data is transferred, it is fully available for use on both source and target. Cloud Sync can sync data on-demand when an update is triggered or continuously sync data based on a predefined schedule. Regardless, Cloud Sync only moves the deltas, so time and money spent on data replication is minimized.

Cloud Sync is a software as a service (SaaS) tool that is extremely simple to set up and use. Data transfers that are triggered by Cloud Sync are carried out by data brokers. Cloud Sync data brokers can be deployed in AWS, Azure, Google Cloud Platform, or on-premises.

Figure 7) Cloud Sync.



NetApp XCP

NetApp XCP is client-based software for any-to-NetApp and NetApp-to-NetApp data migrations and file system insights. XCP is designed to scale and achieve maximum performance by utilizing all available system resources to handle high-volume datasets and high-performance migrations. XCP helps you to gain complete visibility into the file system with the option to generate reports.

NetApp XCP is available in a single package that supports NFS and SMB protocols. XCP includes a Linux binary for NFS data sets and a windows executable for SMB data sets.

NetApp XCP File Analytics is host-based software that detects file shares, runs scans on the file system, and provides a dashboard for file analytics. XCP File Analytics is compatible with both NetApp and non-NetApp systems and runs on Linux or Windows hosts to provide analytics for NFS and SMB-exported file systems.

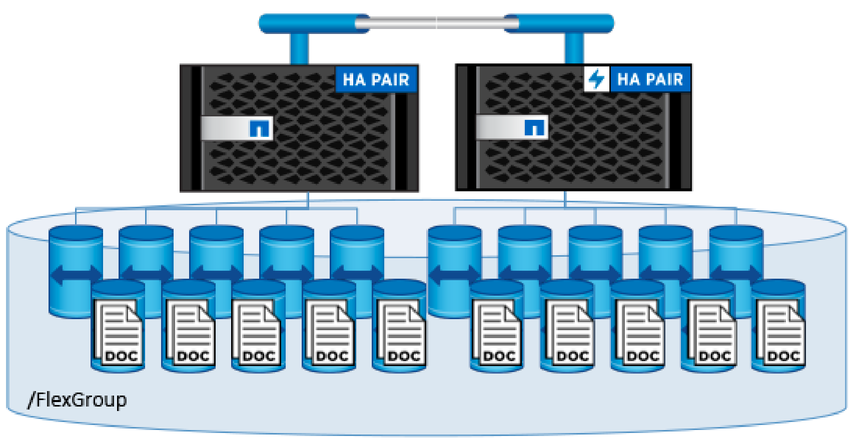
NetApp ONTAP FlexGroup Volumes

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 8) 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 8) NetApp FlexGroup volumes.



Hardware and Software Requirements

All procedures outlined in this document were validated on the NetApp ONTAP AI converged infrastructure solution described in [NVA-1121](). This verified architecture pairs a NetApp AFF A800 all-flash storage system with the NVIDIA DGX-1 Deep Learning System using Cisco Nexus networking. For this validation exercise, two bare-metal NVIDIA DGX-1 systems, each featuring eight NVIDIA V100 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 virtual machines (VMs) that ran on a separate physical server outside of the ONTAP AI pod were used as Kubernetes master nodes. See Table 1 for validation environment infrastructure details. See Table 2 for validation environment software version details.

Note, however, that the NetApp AI Control Plane solution that is outlined in this document is not dependent on this specific hardware. The solution is compatible with any NetApp physical storage appliance, software-defined instance, or cloud service, that supports the NFS protocol. Examples include a NetApp AFF storage system, Azure NetApp Files, NetApp Cloud Volumes Service, a NetApp ONTAP Select software-defined storage instance, or a NetApp Cloud Volumes ONTAP instance. Additionally, the solution can be implemented on any Kubernetes cluster as long as the Kubernetes version used is supported by Kubeflow and NetApp Trident. For a list of Kubernetes versions that are supported by Kubeflow, see the see the official Kubeflow documentation. For a list of Kubernetes versions that are supported by Trident, see the Trident documentation.

Table 1) Validation environment infrastructure details.

Component	Quantity	Details	Operating System
Deployment jump host	1	VM	Ubuntu 18.04.5 LTS
Kubernetes master nodes	3	VM	Ubuntu 18.04.5 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 HA Pair	NetApp AFF A800	NetApp ONTAP 9.6 P1
Network connectivity	2	Cisco Nexus 3232C	Cisco NX-OS 7.0(3)I6(1)

Table 2) Validation environment software version details.

Component	Version
Apache Airflow	1.10.12
Apache Airflow Helm Chart	7.10.1
Cisco NX-OS	7.0(3)I6(1)
Docker	18.09.7
Kubeflow	1.0
Kubernetes	1.17.9
NetApp ONTAP	9.6 P1
NetApp Trident	20.07
NVIDIA DeepOps	20.08.1
NVIDIA DGX OS	4.0.5 (based on Ubuntu 18.04.2 LTS)
Ubuntu	18.04.5 LTS

Support

NetApp does not offer enterprise support for Apache Airflow, Docker, Kubeflow, Kubernetes, or NVIDIA DeepOps. If you are interested in a fully supported solution with capabilities similar to the NetApp AI Control Plane solution, contact NetApp about fully supported AI/ML solutions that NetApp offers jointly with partners.

Kubernetes Deployment

This section describes the tasks that you must complete to deploy a Kubernetes cluster in which to implement the NetApp AI Control Plane solution. If you already have a Kubernetes cluster, then you can skip this section as long as you are running a version of Kubernetes that is supported by Kubeflow and NetApp Trident. For a list of Kubernetes versions that are supported by Kubeflow, see the see the official Kubeflow documentation. For a list of Kubernetes versions that are supported by Trident, see the Trident documentation.

For on-premises Kubernetes deployments that incorporate bare-metal nodes featuring NVIDIA GPU(s), NetApp recommends using NVIDIA's DeepOps Kubernetes deployment tool. This section outlines the deployment of a Kubernetes cluster using DeepOps.

Prerequisites

Before you perform the deployment exercise that is outlined in this section, we assume that you have already performed the following tasks:

1. You have already configured any bare-metal Kubernetes nodes (for example, an NVIDIA DGX system that is part of an ONTAP AI pod) according to standard configuration instructions.
2. You have installed a supported operating system on all Kubernetes master and worker nodes and on a deployment jump host. For a list of operating systems that are supported by DeepOps, see the DeepOps GitHub site.

Use NVIDIA DeepOps to Install and Configure Kubernetes

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

1. Download NVIDIA DeepOps by following the instructions on the Getting Started page on the NVIDIA DeepOps GitHub site.
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.

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.

NetApp Trident Deployment and Configuration

This section describes the tasks that you must complete to install and configure NetApp Trident in your Kubernetes cluster.

Prerequisites

Before you perform the deployment exercise that is outlined in this section, we assume that you have already performed the following tasks:

1. You already have a working Kubernetes cluster, and you are running a version of Kubernetes that is supported by Trident. For a list of supported versions, see the Trident documentation.
2. You already have a working NetApp storage appliance, software-defined instance, or cloud storage service, that supports the NFS protocol.

Install Trident

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

1. Deploy Trident using one of the following methods:
   a. If you used NVIDIA DeepOps to deploy your Kubernetes cluster, you can also use NVIDIA DeepOps to deploy Trident in your Kubernetes cluster. To deploy Trident with DeepOps, follow the Trident deployment instructions on the NVIDIA DeepOps GitHub site.

b. If you did not use NVIDIA DeepOps to deploy your Kubernetes cluster or if you simply prefer to deploy Trident manually, you can deploy Trident by following the deployment instructions in the Trident documentation. Be sure to create at least one Trident backend and at least one Kubernetes StorageClass. For more information about backends and StorageClasses, see the Trident documentation.

Note: If you are deploying the NetApp AI Control Plane solution on an ONTAP AI pod, see the section “Example Trident Backends for ONTAP AI Deployments” for some examples of different Trident backends that you might want to create and the section “Example Kubernetes StorageClasses for ONTAP AI Deployments” for some examples of different Kubernetes StorageClasses that you might want to create.

Example Trident Backends for ONTAP AI Deployments

Before you can use Trident to dynamically provision storage resources within your Kubernetes cluster, you must create one or more Trident backends. The examples that follow represent different types of backends that you might want to create if you are deploying the NetApp AI Control Plane solution on an ONTAP AI pod. For more information about backends, see the Trident documentation.

1. NetApp recommends creating a FlexGroup-enabled Trident backend for each data LIF (logical network interface that provides data access) that you want to use on your NetApp AFF system. 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 backend for each data LIF that you want to use for these mounts.

   The example commands that follow show the creation of two FlexGroup-enabled Trident backends for two different data LIFs that are associated with the same ONTAP storage virtual machine (SVM). These backends 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 backends 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 |
+----------------------------+---------------------+--------------------------------------+------
--+---------+
```

2. NetApp also recommends creating one or more FlexVol-enabled Trident backends. 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 backends. The example commands that follow show the creation of a single FlexVol-enabled Trident backend 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 |
+----------------------------+--------------------+--------------------------------------+------
--+---------+
```

Example Kubernetes StorageClasses for ONTAP AI Deployments

Before you can use Trident to dynamically provision storage resources within your Kubernetes cluster, you must create one or more Kubernetes StorageClasses. The examples that follow represent different types of StorageClasses that you might want to create if you are deploying the NetApp AI Control Plane solution on an ONTAP AI pod. For more information about StorageClasses, see the Trident documentation.

1. NetApp recommends creating a separate StorageClass for each FlexGroup-enabled Trident backend that you created in the section “Example Trident Backends for ONTAP AI Deployments,” step 1. These granular StorageClasses enable you to add NFS mounts that correspond to specific LIFs (the LIFs that you specified when you created the Trident backends) as a particular backend 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 backends that were created in the section “Example Trident Backends for ONTAP AI Deployments,” step 1. The highlighted text shows where the Trident backend is specified in the StorageClass definition file. For more information about StorageClasses, see the Trident 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. NetApp also recommends creating a StorageClass that corresponds to the FlexVol-enabled Trident backend that you created in the section “Example Trident Backends for ONTAP AI Deployments,”

step 2. The example commands that follow show the creation of a single StorageClass for FlexVol volumes.

Note: In the following example, a particular backend is not specified in the StorageClass definition file because only one FlexVol-enabled Trident backend was created in the section "Install Trident," step 2. When you use Kubernetes to administer volumes that use this StorageClass, Trident attempts to use any available backend 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. NetApp also recommends creating 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 backend is not specified in the StorageClass definition file. Therefore, when you use Kubernetes to administer volumes that use this StorageClass, Trident attempts to use any available backend 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
```

Kubeflow Deployment

This section describes the tasks that you must complete to deploy Kubeflow in your Kubernetes cluster.

Prerequisites

Before you perform the deployment exercise that is outlined in this section, we assume that you have already performed the following tasks:

1. You already have a working Kubernetes cluster, and you are running a version of Kubernetes that is supported by Kubeflow. For a list of supported versions, see the official Kubeflow documentation.
2. You have already installed and configured NetApp Trident in your Kubernetes cluster as outlined in the section "NetApp Trident Deployment and Configuration."

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 persistent volumes 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 from the deployment jump host. If you have already designated a default StorageClass within your cluster, then you can skip this step.

1. Designate one of your existing StorageClasses as the default StorageClass. The example commands that follow show the designation of a StorageClass named `ontap-ai-flexvols-retain` as the default StorageClass.

   Note: The `ontap-nas-flexgroup` Trident backend type 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` backend type 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
```

Use NVIDIA DeepOps to Deploy Kubeflow

NetApp recommends using the Kubeflow deployment tool that is provided by NVIDIA DeepOps. To deploy Kubeflow in your Kubernetes cluster using the DeepOps deployment tool, perform the following tasks from the deployment jump host.

Note: Alternatively, you can deploy Kubeflow manually by following the installation instructions in the official Kubeflow documentation

1. Deploy Kubeflow in your cluster by following the Kubeflow deployment instructions on the NVIDIA DeepOps GitHub site.
2. Note down the Kubeflow Dashboard URL that the DeepOps Kubeflow deployment tool outputs.

```
$ ./scripts/k8s_deploy_kubeflow.sh
…
INFO[0007] Applied the configuration Successfully!       filename="cmd/apply.go:72"

Kubeflow app installed to: /home/ai/kubeflow

It may take several minutes for all services to start. Run 'kubectl get pods -n kubeflow' to
verify

To remove (excluding CRDs, istio, auth, and cert-manager), run: ./scripts/k8s_deploy_kubeflow.sh
-d

To perform a full uninstall : ./scripts/k8s_deploy_kubeflow.sh -D

Kubeflow Dashboard (HTTP NodePort): http://10.61.188.111:31380
```

3. 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
```

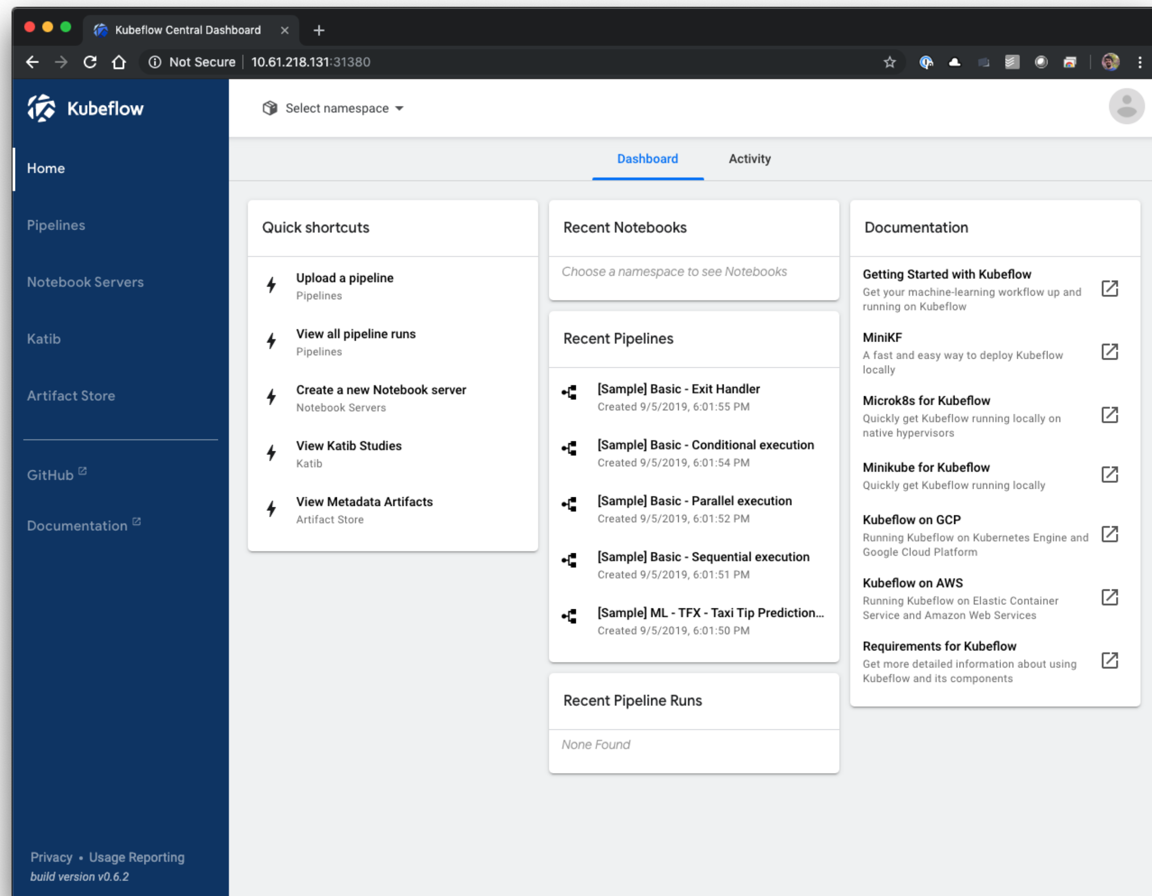


```
statefulset.apps/metacontroller                          1/1     98s
statefulset.apps/seldon-operator-controller-manager      1/1     92s

$ 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
```

4. In your web browser, access the Kubeflow central dashboard by navigating to the URL that you noted down in step 2.

   Note: The default username is `admin@kubeflow.org`, and the default password is `12341234`. To create additional users, follow the instructions in the official Kubeflow documentation.



Example Kubeflow Operations and Tasks

This section includes examples of various operations and tasks that you may want to perform using Kubeflow.

Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use

Kubeflow is capable of rapidly provisioning new Jupyter Notebook servers to act as data scientist workspaces. 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.

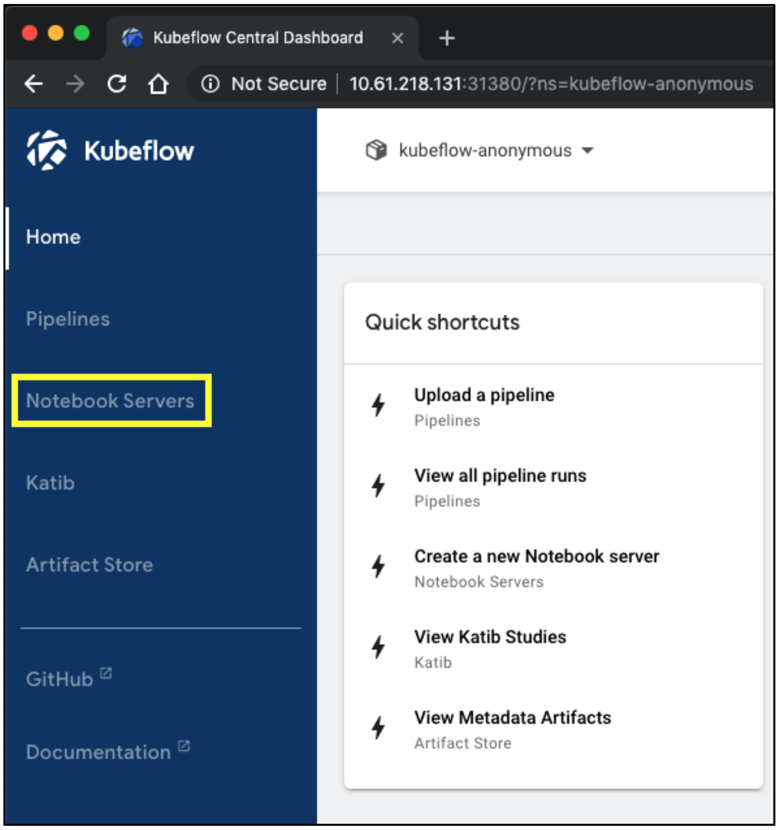
1. **Optional:** If there are existing volumes on your NetApp storage system that you want to mount on the new Jupyter Notebook server, but that are not tied to PersistentVolumeClaims (PVCs) in the namespace that the new server is going to be created in (see step 4 below), then you must import these volumes into that namespace. Use the Trident volume import functionality to import these volumes.

   The example commands that follow show the importing of an existing volume named `pb_fg_all` into the `kubeflow-anonymous` namespace. These commands create a PVC in the `kubeflow-anonymous` namespace that is tied to the volume on the NetApp storage system. For more information about PVCs, see the official Kubernetes documentation. For more information about the volume import functionality, see the Trident documentation. For a detailed example showing the importing of a volume using Trident, see the section “0.”

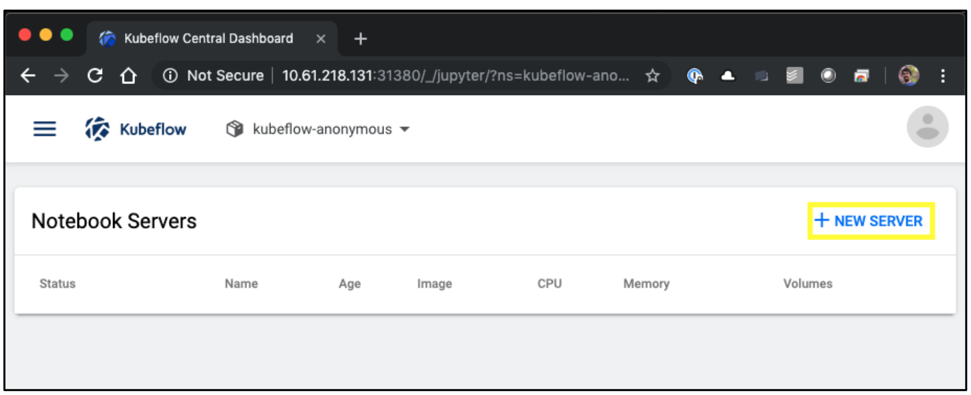
   Note: The volume is imported in the `kubeflow-anonymous` namespace because that is the namespace that the new Jupyter Notebook server is created in in step 4. To mount this existing volume on the new Jupyter Notebook server using Kubeflow, a PVC must exist for the volume in the same namespace.

```
$ 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.

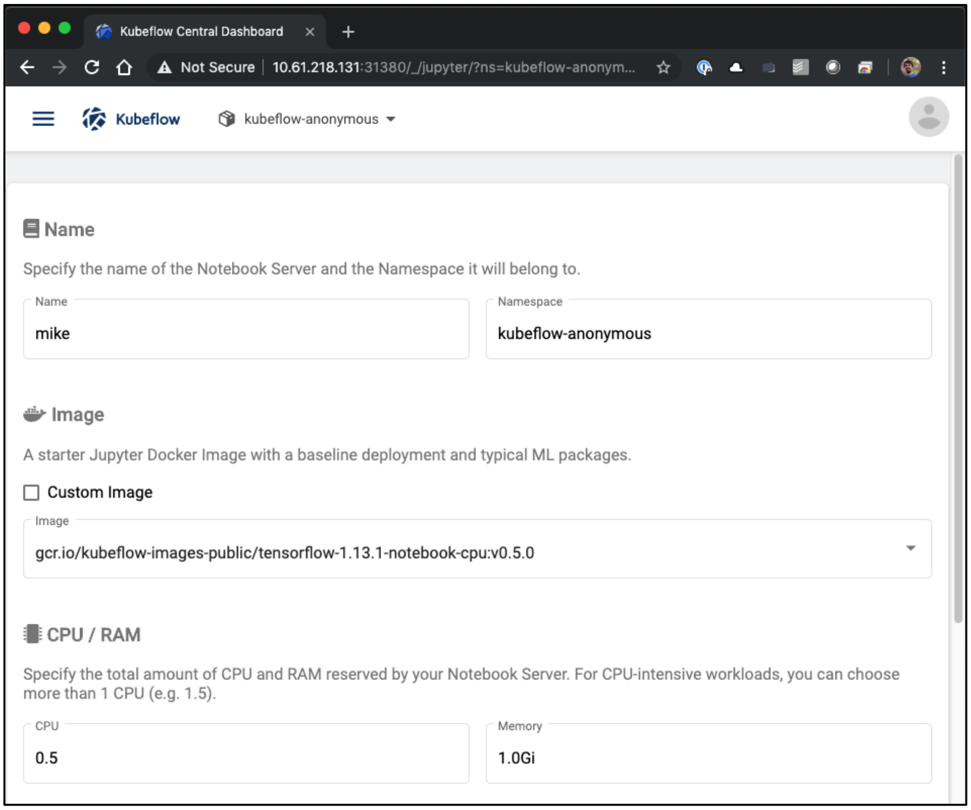


3. Click New Server to provision a new Jupyter Notebook server.

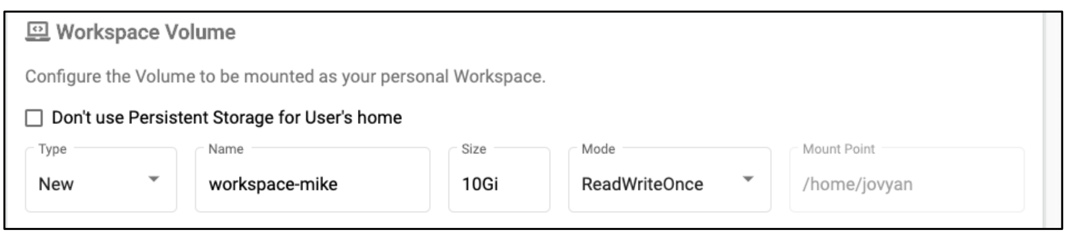


4. Give your new server a name, choose the Docker image that you want your server to be based on, and specify the amount of CPU and RAM to be reserved by your server. If the Namespace field is blank, use the Select Namespace menu in the page header to choose a namespace. The Namespace field is then auto-populated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

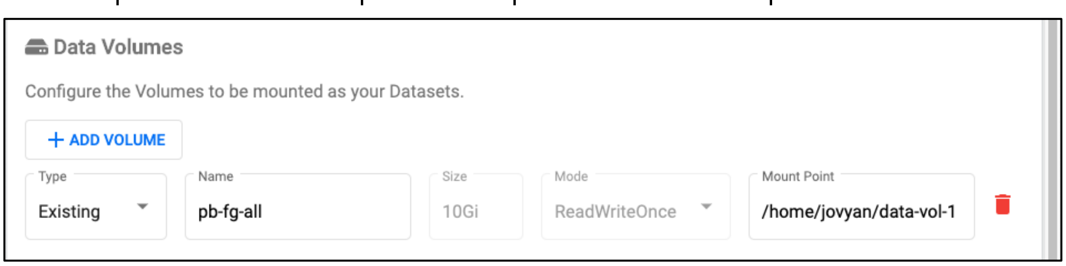
   In the following example, the `kubeflow-anonymous` namespace is chosen. In addition, the default values for Docker image, CPU, and RAM are accepted.



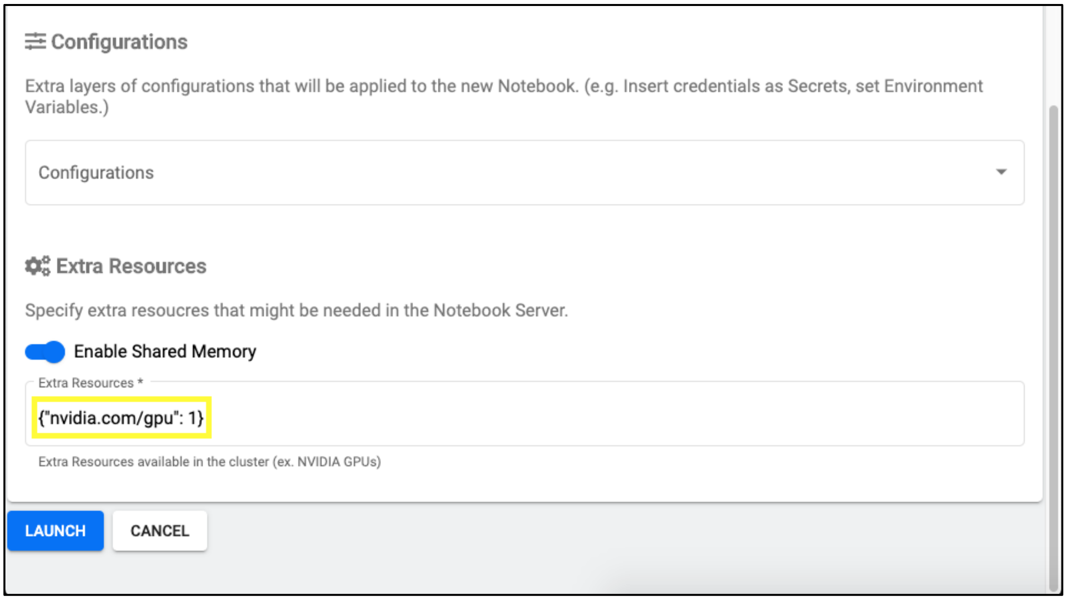
5. Specify the workspace volume details. If you choose to create a new volume, then that volume or PVC is provisioned using the default StorageClass. Because a StorageClass utilizing Trident was designated as the default StorageClass in the section “Set Default Kubernetes StorageClass,” the volume or 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 are persistent across reboots.



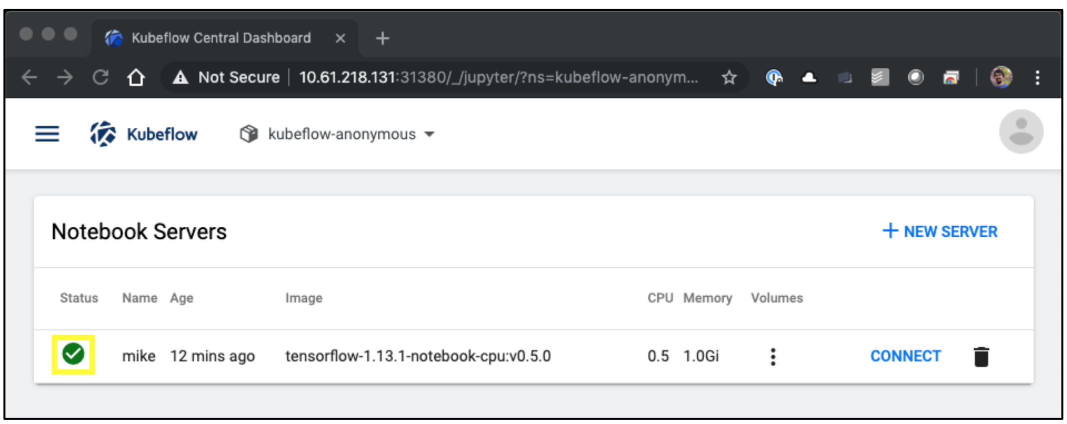
6. Add data volumes. The following example specifies the existing volume that was imported by the example commands in step 1 and accepts the default mount point.



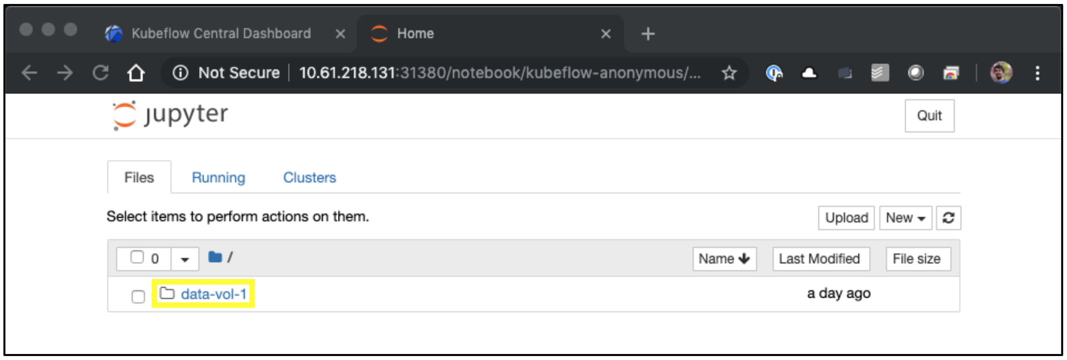
7. **Optional:** Request that the desired number of GPUs be allocated to your notebook server. In the following example, one GPU is requested.

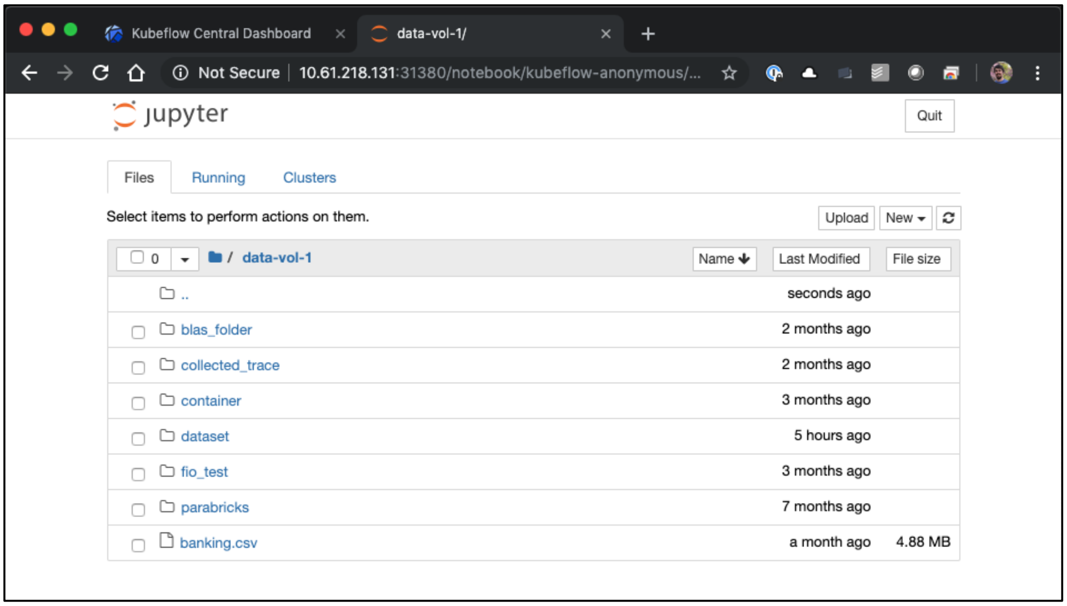


8. Click Launch to provision your new notebook server.
9. Wait for your notebook server to be fully provisioned. This can take several minutes if you have never provisioned a server using the Docker image that you specified in step 4 because the image needs to be downloaded. When your server has been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



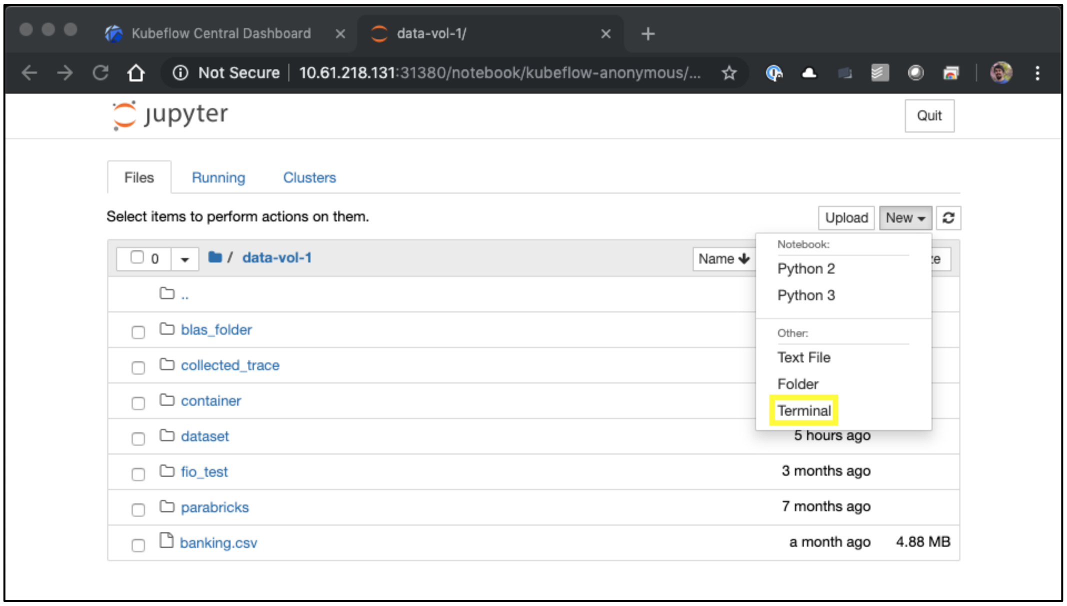
10. Click Connect to connect to your new server's web interface.
11. Confirm that the dataset volume that was specified in step 6 is mounted on the server. Note that this volume is mounted within the default workspace by default. From the perspective of the user, this is just another folder within the 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.

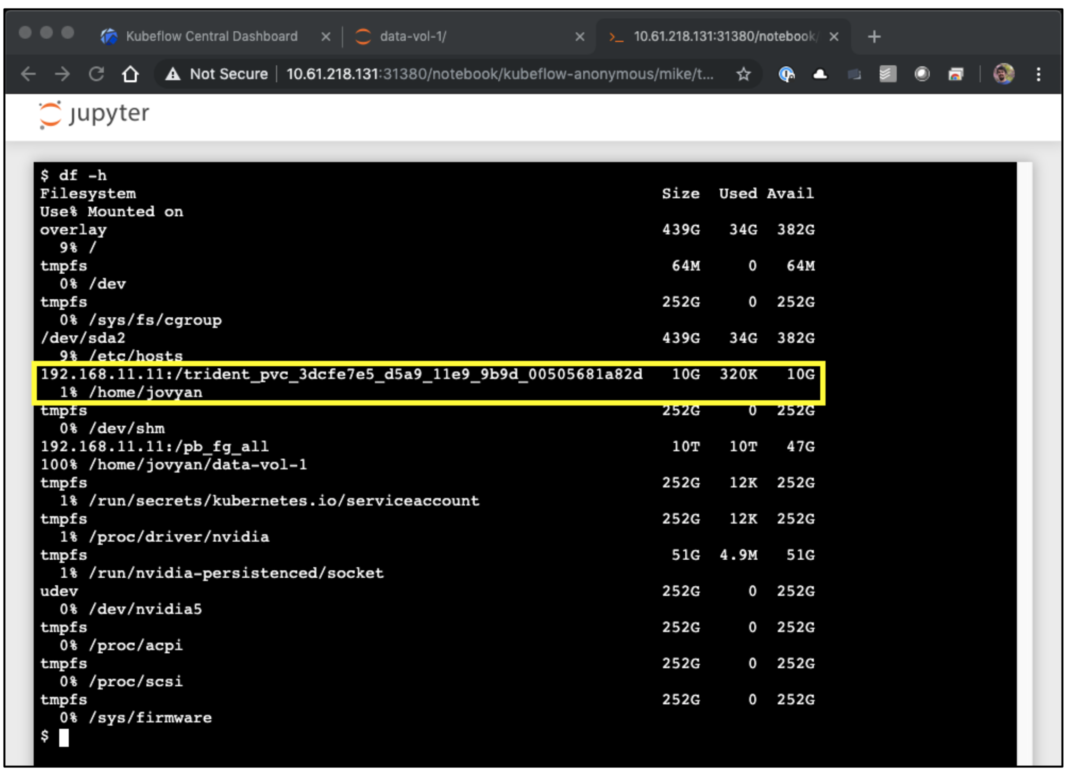




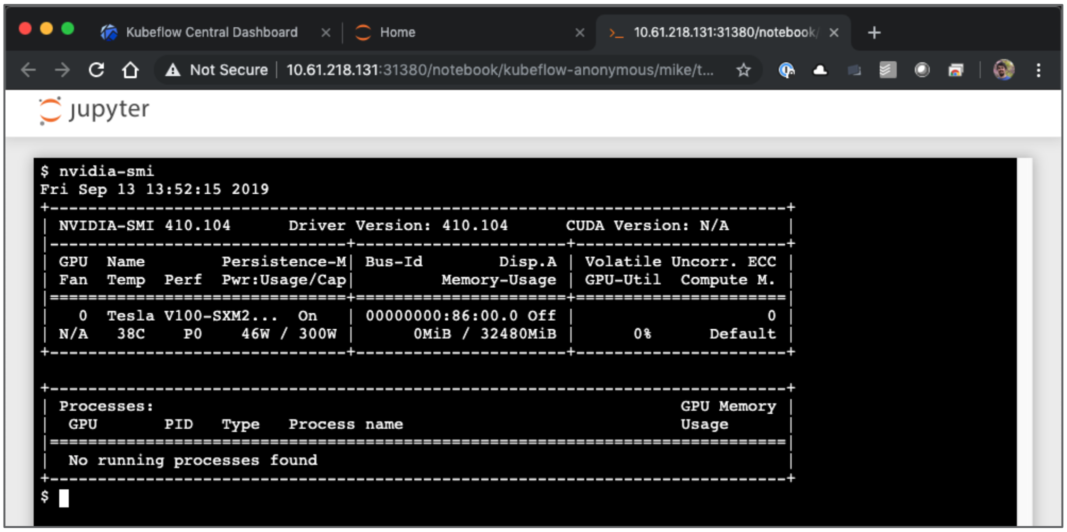
12. Open a terminal and, assuming that a new volume was requested in step 5, execute `df -h` to confirm that a new Trident-provisioned persistent volume is mounted as the default workspace.

    Note: The default workspace directory is the base directory that you are presented with when you first 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.





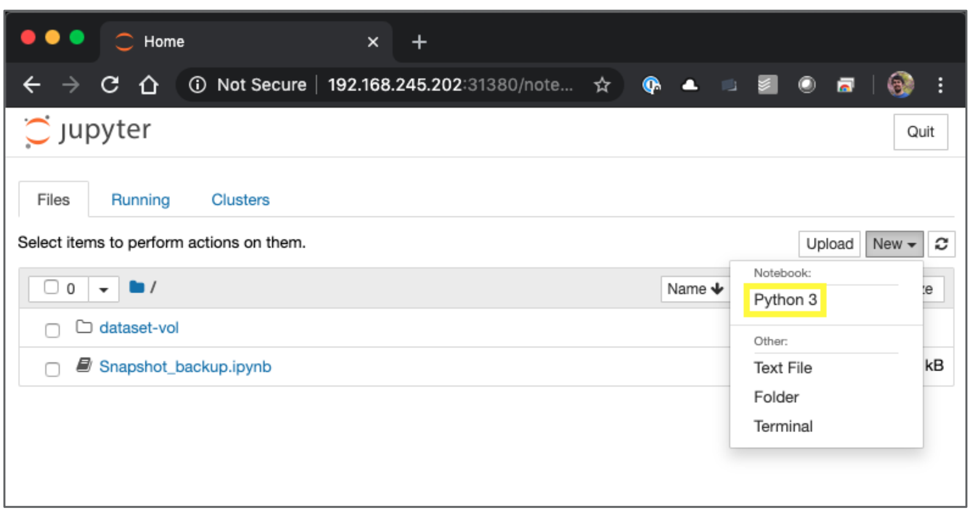
13. Using the terminal, run `nvidia-smi` to confirm that the correct number of GPUs were allocated to the notebook server. In the following example, one GPU has been allocated to the notebook server as requested in step 7.



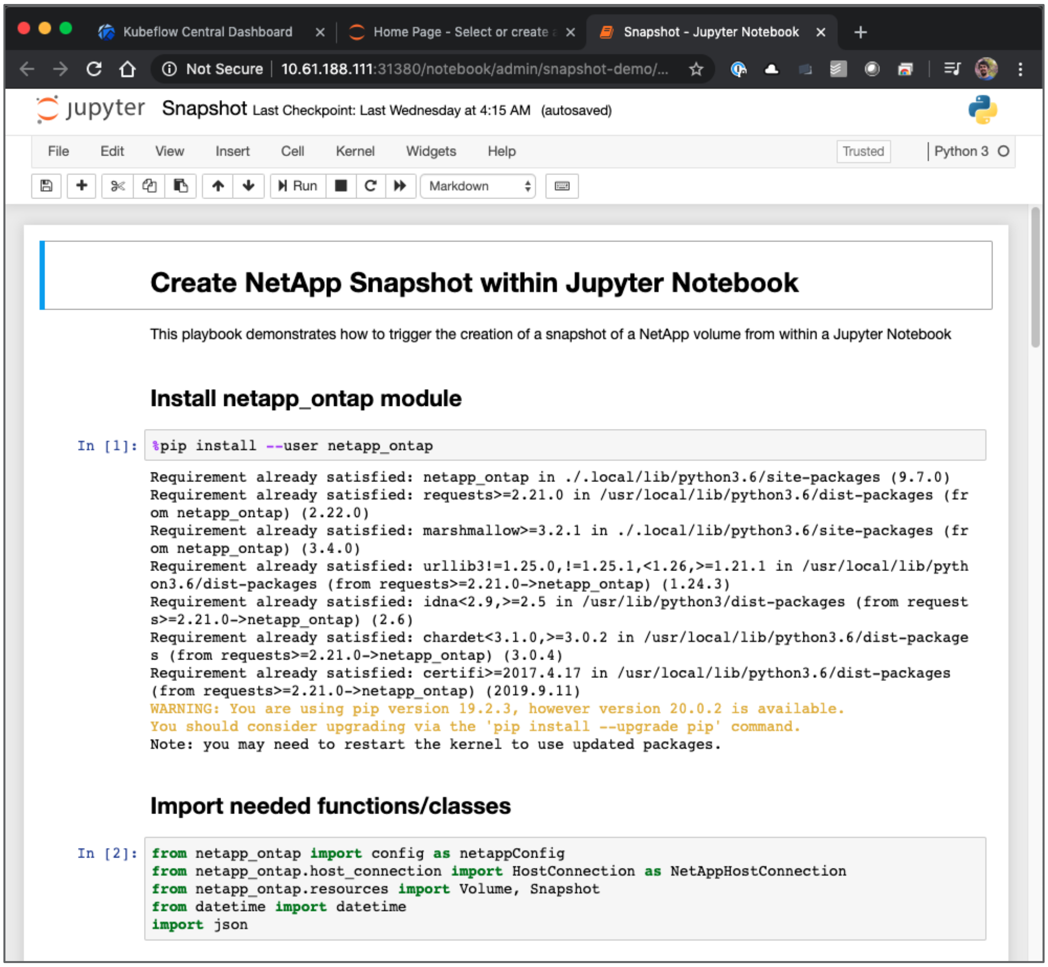
Create a Snapshot of an ONTAP Volume from Within a Jupyter Notebook

To trigger the creation of a snapshot, from within a Jupyter Notebook, of a NetApp ONTAP volume that is mounted in the Jupyter Notebook Server's workspace, perform the following tasks. This operation takes advantage of the NetApp ONTAP REST APIs and the NetApp ONTAP Python module. For more information about the REST APIs and the Python module, see the NetApp support site. Note that tasks in this section only work for volumes that reside on ONTAP storage systems or software-defined instances.

1. Connect to a Jupyter Notebook server's web interface. See the section "Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use" for instructions on how to provision a Jupyter Notebook Server.
2. Open an existing Python 3 notebook or create a new Python 3 notebook. The following example shows the creation of a new Python 3 notebook.



3. Add the following content to the Notebook, update variable values as stated in the comments, and then run all cells. Alternatively, an example Jupyter Notebook containing this content can be downloaded from NetApp's Kubeflow and Jupyter Examples GitHub repository.



Create NetApp Snapshot within Jupyter Notebook

This playbook demonstrates how to trigger the creation of a snapshot of a NetApp volume from within a Jupyter Notebook

Install netapp_ontap module

In [1]:

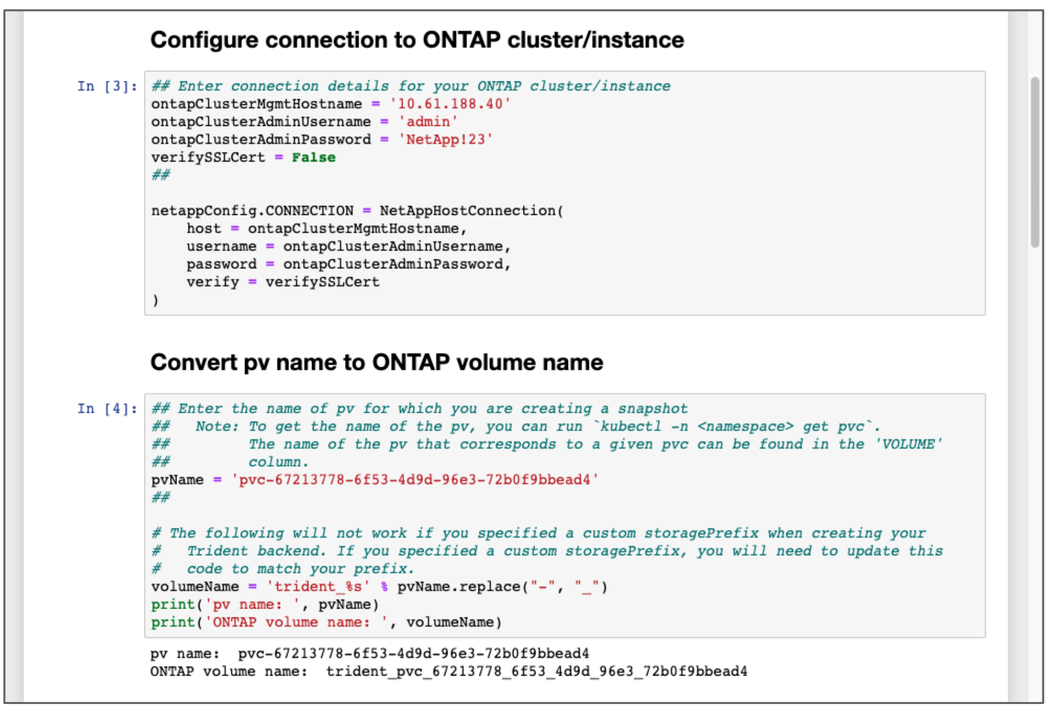
```
%pip install --user netapp_ontap
```

```
Requirement already satisfied: netapp_ontap in ./.local/lib/python3.6/site-packages (9.7.0)
Requirement already satisfied: requests>=2.21.0 in /usr/local/lib/python3.6/dist-packages (fr
om netapp_ontap) (2.22.0)
Requirement already satisfied: marshmallow>=3.2.1 in ./.local/lib/python3.6/site-packages (fr
om netapp_ontap) (3.4.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/pyth
on3.6/dist-packages (from requests>=2.21.0->netapp_ontap) (1.24.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/lib/python3/dist-packages (from request
s>=2.21.0->netapp_ontap) (2.6)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-package
s (from requests>=2.21.0->netapp_ontap) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages
(from requests>=2.21.0->netapp_ontap) (2019.9.11)
WARNING: You are using pip version 19.2.3, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

Import needed functions/classes

In [2]:

```
from netapp_ontap import config as netappConfig
from netapp_ontap.host_connection import HostConnection as NetAppHostConnection
from netapp_ontap.resources import Volume, Snapshot
from datetime import datetime
import json
```



Configure connection to ONTAP cluster/instance

In [3]:

```
## Enter connection details for your ONTAP cluster/instance
ontapClusterMgmtHostname = '10.61.188.40'
ontapClusterAdminUsername = 'admin'
ontapClusterAdminPassword = 'NetApp!23'
verifySSLCert = False
##

netappConfig.CONNECTION = NetAppHostConnection(
    host = ontapClusterMgmtHostname,
    username = ontapClusterAdminUsername,
    password = ontapClusterAdminPassword,
    verify = verifySSLCert
)
```

Convert pv name to ONTAP volume name

In [4]:

```
## Enter the name of pv for which you are creating a snapshot
##   Note: To get the name of the pv, you can run `kubectl -n <namespace> get pvc`.
##         The name of the pv that corresponds to a given pvc can be found in the 'VOLUME'
##         column.
pvName = 'pvc-67213778-6f53-4d9d-96e3-72b0f9bbead4'
##

# The following will not work if you specified a custom storagePrefix when creating your
#   Trident backend. If you specified a custom storagePrefix, you will need to update this
#   code to match your prefix.
volumeName = 'trident_%s' % pvName.replace("-", "_")
print('pv name: ', pvName)
print('ONTAP volume name: ', volumeName)
```

```
pv name:  pvc-67213778-6f53-4d9d-96e3-72b0f9bbead4
ONTAP volume name:  trident_pvc_67213778_6f53_4d9d_96e3_72b0f9bbead4
```

Create snapshot

In [5]:
```
volume = Volume.find(name = volumeName)
timestamp = datetime.today().strftime("%Y%m%d_%H%M%S")
snapshot = Snapshot.from_dict({
    'name': 'jupyter_%s' % timestamp,
    'comment': 'Snapshot created from within a Jupyter Notebook',
    'volume': volume.to_dict()
})
response = snapshot.post()
print("API Response:")
print(response.http_response.text)
```


```
API Response:
{
  "uuid": "ea754776-49ba-11ea-8196-d039ea06490a",
  "description": "POST /api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/?n
ame=jupyter_20200207_200323",
  "state": "success",
  "message": "success",
  "code": 0,
  "start_time": "2020-02-07T15:02:53+00:00",
  "end_time": "2020-02-07T15:02:53+00:00",
  "_links": {
    "self": {
      "href": "/api/cluster/jobs/ea754776-49ba-11ea-8196-d039ea06490a"
    }
  }
}
```


Optional: Retrieve details for newly created snapshot

In [6]:
```
snapshot.get()
print(json.dumps(snapshot.to_dict(), indent=2))
```


```
{
  "svm": {
    "uuid": "e6121682-3224-11ea-8196-d039ea06490a",
    "name": "ai221_data",
    "_links": {
      "self": {
        "href": "/api/svm/svms/e6121682-3224-11ea-8196-d039ea06490a"
      }
    }
  },
  "uuid": "899f336d-166a-426a-a1a7-aa349764b6cc",
  "volume": {
    "uuid": "14a07f2d-468e-11ea-808d-d039ea06439f",
    "name": "trident_pvc_67213778_6f53_4d9d_96e3_72b0f9bbead4",
    "_links": {
      "self": {
        "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f"
      }
    }
  },
  "_links": {
    "self": {
      "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/899f336d-1
66a-426a-a1a7-aa349764b6cc"
    }
  },
  "create_time": "2020-02-07T15:02:53+00:00",
  "comment": "Snapshot created from within a Jupyter Notebook",
  "name": "jupyter_20200207_200323"
}
```


Optional: Retrieve a list of all snapshots that exist for the volume

In [7]:

```python
numVolumeSnapshots = 0
for volumeSnapshot in Snapshot.get_collection(volume.uuid, max_records = 256) :
    numVolumeSnapshots+=1
    volumeSnapshot.get()
    print("Snapshot #%s:" % numVolumeSnapshots)
    print(json.dumps(volumeSnapshot.to_dict(), indent=2), "\n")

if numVolumeSnapshots >= 256 :
    print("256 snapshots retrieved. More snapshots may exist.")
else :
    print("Total Snapshots: %s" % numVolumeSnapshots)
```


```
Snapshot #1:
{
  "svm": {
    "uuid": "e6121682-3224-11ea-8196-d039ea06490a",
    "name": "ai221_data",
    "_links": {
      "self": {
        "href": "/api/svm/svms/e6121682-3224-11ea-8196-d039ea06490a"
      }
    }
  },
  "uuid": "87698310-8688-4746-8bca-50f87d79e034",
  "volume": {
    "uuid": "14a07f2d-468e-11ea-808d-d039ea06439f",
    "name": "trident_pvc_67213778_6f53_4d9d_96e3_72b0f9bbead4",
    "_links": {
      "self": {
        "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f"
      }
    }
  },
  "_links": {
    "self": {
      "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/87698310-8
688-4746-8bca-50f87d79e034"
    }
  },
  "create_time": "2020-02-04T16:54:04+00:00",
  "name": "clone_kfp_clone_202002.1"
}
```

```
Snapshot #2:
{
  "svm": {
    "uuid": "e6121682-3224-11ea-8196-d039ea06490a",
    "name": "ai221_data",
    "_links": {
      "self": {
        "href": "/api/svm/svms/e6121682-3224-11ea-8196-d039ea06490a"
      }
    }
  },
  "uuid": "5e754480-73a9-4314-9f07-5f879b1f214f",
  "volume": {
    "uuid": "14a07f2d-468e-11ea-808d-d039ea06439f",
    "name": "trident_pvc_67213778_6f53_4d9d_96e3_72b0f9bbead4",
    "_links": {
      "self": {
        "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f"
      }
    }
  },
  "_links": {
    "self": {
      "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/5e754480-7
3a9-4314-9f07-5f879b1f214f"
    }
  },
  "create_time": "2020-02-05T09:12:35+00:00",
  "comment": "Snapshot created from within a Jupyter Notebook",
  "name": "jupyter_20200205_141259"
}
```



```
Snapshot #3:
{
  "svm": {
    "uuid": "e6121682-3224-11ea-8196-d039ea06490a",
    "name": "ai221_data",
    "_links": {
      "self": {
        "href": "/api/svm/svms/e6121682-3224-11ea-8196-d039ea06490a"
      }
    }
  },
  "uuid": "899f336d-166a-426a-a1a7-aa349764b6cc",
  "volume": {
    "uuid": "14a07f2d-468e-11ea-808d-d039ea06439f",
    "name": "trident_pvc_67213778_6f53_4d9d_96e3_72b0f9bbead4",
    "_links": {
      "self": {
        "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f"
      }
    }
  },
  "_links": {
    "self": {
      "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/899f336d-1
66a-426a-a1a7-aa349764b6cc"
    }
  },
  "create_time": "2020-02-07T15:02:53+00:00",
  "comment": "Snapshot created from within a Jupyter Notebook",
  "name": "jupyter_20200207_200323"
}

Total Snapshots: 3
```


Trigger a Cloud Sync Replication Update from Within a Jupyter Notebook

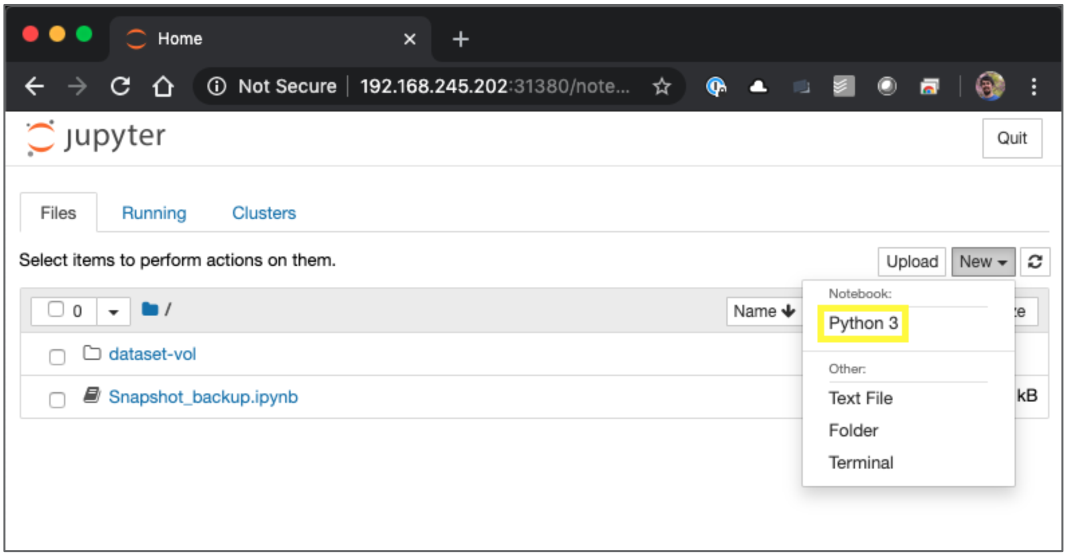
From directly within a Jupyter Notebook, you can trigger the replication of data to and from a variety of file and object storage platforms by using NetApp Cloud Sync replication technology. Potential use cases include:

- Replicating newly acquired sensor data gathered at the edge back to the core data center or to the cloud to be used for AI/ML model training or retraining.
- Replicating a newly trained or newly-updated model from the core data center to the edge or to the cloud to be deployed as part of an inferencing application.
- Copying data from an S3 data lake to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Copying data from a Hadoop data lake (through Hadoop NFS Gateway) to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Saving a new version of a trained model to an S3 or Hadoop data lake for permanent storage.
- Copying NFS-accessible data from a legacy or non-NetApp system of record to a high-performance AI/ML training environment for use in the training of an AI/ML model.

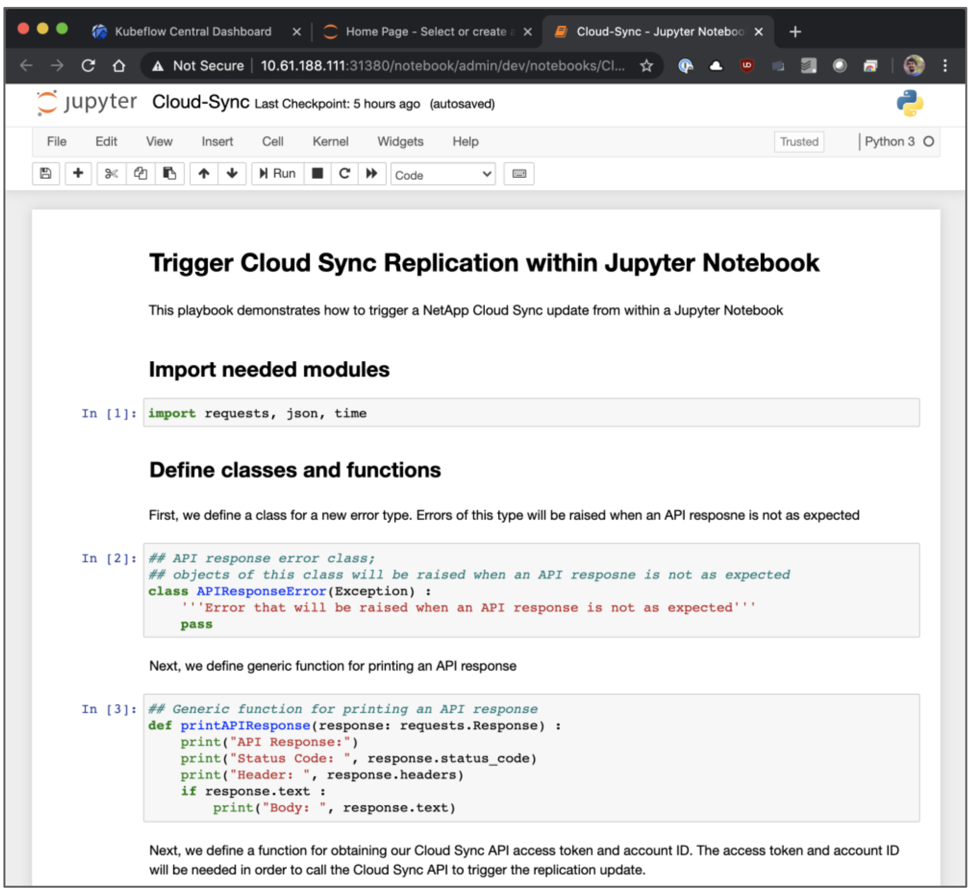
To trigger a Cloud Sync replication update from within a Jupyter Notebook, perform the following tasks:

Note: Before you perform the exercises that are outlined in this section, we assume that you have already initiated the Cloud Sync relationship that you wish to trigger an update for. To initiate a relationship, visit cloudsync.netapp.com.

1. Connect to a Jupyter Notebook server's web interface. For instructions on how to provision a Jupyter Notebook server, see the section "Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use.".
2. Open an existing Python 3 notebook or create a new Python 3 notebook. The following example shows the creation of a new Python 3 notebook.



3. Add the following content to the Notebook, update variable values as stated in the instructions, and then run all cells. Alternatively, an example Jupyter Notebook containing this content can be downloaded from [NetApp's Kubeflow and Jupyter Examples GitHub repository](https://github.com/NetApp/kubeflow_jupyter_pipeline).




```
In [4]: ## Function for obtaining access token and account ID for calling Cloud Sync API
def netappCloudSyncAuth(refreshToken: str) :
    ## Step 1: Obtain limited time access token using refresh token

    # Define parameters for API call
    url = "https://netapp-cloud-account.auth0.com/oauth/token"
    headers = {
        "Content-Type": "application/json"
    }
    data = {
        "grant_type": "refresh_token",
        "refresh_token": refreshToken,
        "client_id": "Mu0V1ywgYteI6w1MbD15fKfVIUrNXGWC"
    }

    # Call API to optain access token
    response = requests.post(url = url, headers = headers, data = json.dumps(data))

    # Parse response to retrieve access token
    try :
        responseBody = json.loads(response.text)
        accessToken = responseBody["access_token"]
    except :
        errorMessage = "Error obtaining access token from Cloud Sync API"
        raise APIResponseError(errorMessage, response)

    ## Step 2: Obtain account ID

    # Define parameters for API call
    url = "https://cloudsync.netapp.com/api/accounts"
    headers = {
        "Content-Type": "application/json",
        "Authorization": "Bearer " + accessToken
    }

    # Call API to obtain account ID
    response = requests.get(url = url, headers = headers)

    # Parse response to retrieve account ID
    try :
        responseBody = json.loads(response.text)
        accountId = responseBody[0]["accountId"]
    except :
        errorMessage = "Error obtaining account ID from Cloud Sync API"
        raise APIResponseError(errorMessage, response)

    # Return access token and account ID
    return accessToken, accountId
```

Next, we define a function for actually triggering the Cloud Sync update

```
In [5]: ## Function for triggering an update for a specific Cloud Sync relationship
def netappCloudSyncUpdate(refreshToken: str, relationshipId: str, printResponse: bool = True) :
    # Step 1: Obtain access token and account ID for accessing Cloud Sync API
    try :
        accessToken, accountId = netappCloudSyncAuth(refreshToken = refreshToken)
    except APIResponseError as err:
        if printResponse :
            errorMessage = err.args[0]
            response = err.args[1]
            print(errorMessage)
            printAPIResponse(response)
        raise

    # Step 2: Trigger Cloud Sync update

    # Define parameters for API call
    url = "https://cloudsync.netapp.com/api/relationships/%s/sync" % (relationshipId)
    headers = {
        "Content-Type": "application/json",
        "Accept": "application/json",
        "x-account-id": accountId,
        "Authorization": "Bearer " + accessToken
    }

    # Call API to trigger update
    response = requests.put(url = url, headers = headers)

    # Check for API response status code of 202; if not 202, raise error
    if response.status_code != 202 :
        errorMessage = "Error calling Cloud Sync API to trigger update."
        if printResponse :
            print(errorMessage)
            printAPIResponse(response)
        raise APIResponseError(errorMessage, response)

    # Print API response
    if printResponse :
        print("Note: Status Code 202 denotes that update was successfully triggered.")
        printAPIResponse(response)
```


Lastly, we define a function for monitoring the progress of the latest replication update

In [6]:
```
## Function for monitoring the progress of the latest update for a specific Cloud Sync
## relationship
def netappCloudSyncMonitor(refreshToken: str, relationshipId: str,
                           keepCheckingUntilComplete: bool = True, printProgress: bool = True,
                           printResponses: bool = False) :
    # Step 1: Obtain access token and account ID for accessing Cloud Sync API
    try :
        accessToken, accountId = netappCloudSyncAuth(refreshToken = refreshToken)
    except APIResponseError as err:
        if printResponse :
            errorMessage = err.args[0]
            response = err.args[1]
            print(errorMessage)
            printAPIResponse(response)
        raise

    # Step 2: Obtain status of the latest update;
    # optionally, keep checking until the latest update has completed

    while True :
        # Define parameters for API call
        url = "https://cloudsync.netapp.com/api/relationships-v2/%s" % (relationshipId)
        headers = {
            "Accept": "application/json",
            "x-account-id": accountId,
            "Authorization": "Bearer " + accessToken
        }

        # Call API to obtain status of latest update
        response = requests.get(url = url, headers = headers)

        # Print API response
        if printResponses :
            printAPIResponse(response)

        # Parse response to retrieve status of latest update
        try :
            responseBody = json.loads(response.text)
            latestActivityType = responseBody["activity"]["type"]
            latestActivityStatus = responseBody["activity"]["status"]
        except :
            errorMessage = "Error status of latest update from Cloud Sync API"
            raise APIResponseError(errorMessage, response)
```

```
        # End execution if the latest update is complete
        if latestActivityType == "Sync" and latestActivityStatus == "DONE" :
            if printProgress :
                print("Success: Cloud Sync update is complete.")
            break

        # Print message re: progress
        if printProgress :
            print("Cloud Sync update is not yet complete.")

        # End execution if calling program doesn't want to monitor until the latest update
        # has completed
        if not keepCheckingUntilComplete :
            break

        # Sleep for 60 seconds before checking progress again
        print("Checking again in 60 seconds...")
        time.sleep(60)
```

Set Cloud Sync refresh token

A refresh token is needed in order to obtain an access token. If you do not yet have a refresh token, you can create one here: https://services.cloud.netapp.com/refresh-token.

In [7]:
```
refreshToken = "<enter your refresh token>"
```

Optional: obtain Cloud Sync relationship ID

If you do not already know the relationship ID for the specific Cloud Sync relationship that you wish to trigger an update for, then you must obtain it. In order to do this, we define a function for obtaining a list of all Cloud Sync relationships that are tied to our account.

If you already know the relationship id for the specific relationship that you wish to trigger an update for, then you can skip this section


```
In [8]: def netappCloudSyncGetRelationships(refreshToken: str, printResponse: bool = True) :
            # Step 1: Obtain access token and account ID for accessing Cloud Sync API
            try :
                accessToken, accountId = netappCloudSyncAuth(refreshToken = refreshToken)
            except APIResponseError as err:
                if printResponse :
                    errorMessage = err.args[0]
                    response = err.args[1]
                    print(errorMessage)
                    printAPIResponse(response)
                raise

            # Step 2: Retrieve list of relationships

            # Define parameters for API call
            url = "https://cloudsync.netapp.com/api/relationships-v2"
            headers = {
                "Accept": "application/json",
                "x-account-id": accountId,
                "Authorization": "Bearer " + accessToken
            }

            # Call API to retrieve list of relationships
            response = requests.get(url = url, headers = headers)

            # Check for API response status code of 200; if not 200, raise error
            if response.status_code != 200 :
                errorMessage = "Error calling Cloud Sync API to retrieve list of relationships."
                if printResponse :
                    print(errorMessage)
                    printAPIResponse(response)
                raise APIResponseError(errorMessage, response)

            # Print API response
            if printResponse :
                print("API Response:")
                print("Note: Status Code 200 denotes success.")
                print("Status Code: ", response.status_code)
                print("Header: ", response.headers)
                print("Body: ", response.text)

            # Return json object containing response body
            responseBody = json.loads(response.text)
            return responseBody
```

Now, we wil call the function that we just defined. If you receive an error, try restarting the kernel and running again.

```
In [9]: relationships = netappCloudSyncGetRelationships(refreshToken = refreshToken)
```


```
API Response:
Note: Status Code 200 denotes success.
Status Code:  200
Header:  {'Date': 'Fri, 29 May 2020 19:23:15 GMT', 'Content-Type': 'application/json; charset
=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'X-DNS-Prefetch-Contro
l': 'off', 'Strict-Transport-Security': 'max-age=15552000; includeSubDomains', 'X-Download-Op
tions': 'noopen', 'X-Content-Type-Options': 'nosniff', 'X-XSS-Protection': '1; mode=block',
'X-DFIO-Req-Id': '16b50864-666f-4e73-a74e-9d6e525137f5', 'Access-Control-Allow-Origin': '*',
'ETag': 'W/"bdf-w7B1rN0+gXeBim4ZLbHJoM2pyiA"', 'Vary': 'Accept-Encoding', 'Content-Encoding':
'gzip'}
Body:  [{"isQstack":false,"isCvo":false,"isCm":false,"phase":"Sync","source":{"protocol":"nf
s","nfs":{"host":"192.168.200.41","export":"/trident_pvc_1b3d8a1c_b3d5_4a3a_a767_a936dfe5287
1","path":"","version":"3","provider":"nfs"}},"target":{"protocol":"nfs","nfs":{"host":"192.1
68.200.41","export":"/trident_pvc_0361a52b_9f65_4adc_9092_aafeb602a809","path":"","versio
n":"3","provider":"nfs"}},"settings":{"gracePeriod":30,"deleteOnSource":false,"deleteOnTarge
t":false,"objectTagging":true,"retries":3,"copyAcl":false,"files":{"excludeExtensions":[],"ma
xSize":9007199254740991,"minSize":0,"minDate":"1970-01-01","maxDate":null},"fileTypes":{"file
s":true,"directories":true,"symlinks":true},"schedule":{"syncInDays":1,"syncInHours":0,"syncI
nMinutes":0,"isEnabled":false,"syncWhenCreated":true,"nextTime":"2020-05-29T18:00:00.000
Z"}},"dataBroker":{"lastPing":{"wasabi":1590780193457},"type":"ONPREM","name":"ailab01","grou
pId":"5ea35f9fac30c3972da9f533","createdAt":1587765162950,"transferRate":1049.21968787515,"up
dateNewVersion":true,"id":"5ea35f9f94465a000a7c9fc3","placement":{"hostname":"ubuntu1804","pl
atform":"linux","privateIp":"10.61.188.114","version":"1.3.0.17118-85c1cf2-production","o
s":"Linux","release":"4.15.0-96-generic","osTotalMem":"16819924992","node":"14.0.0","cpu
s":"4","processMaxMem":"78729216"},"status":"COMPLETE","fileLink":"https://cf.cloudsync.netap
p.com/5ea35f9f94465a000a7c9fc3_installer"},"group":{"dataBrokers":[{"lastPing":{"wasabi":1590
780193457},"type":"ONPREM","name":"ailab01","groupId":"5ea35f9fac30c3972da9f533","createdAt":
1587765162950,"transferRate":1049.21968787515,"updateNewVersion":true,"id":"5ea35f9f94465a000
a7c9fc3","placement":{"hostname":"ubuntu1804","platform":"linux","privateIp":"10.61.188.11
4","version":"1.3.0.17118-85c1cf2-production","os":"Linux","release":"4.15.0-96-generic","osT
otalMem":"16819924992","node":"14.0.0","cpus":"4","processMaxMem":"78729216"},"status":"COMPL
ETE","fileLink":"https://cf.cloudsync.netapp.com/5ea35f9f94465a000a7c9fc3_installer"}],"nam
e":"ailab01","createdAt":"2020-04-24T21:52:31.258Z","id":"5ea35f9fac30c3972da9f533"},"startTi
me":"2020-05-29T19:17:10.966Z","createdAt":1590692247835,"endTime":"2020-05-29T19:19:22.084
Z","id":"5ed00996ca85650009a83db2","relationshipId":"5ed00996ca85650009a83db2","activity":{"t
ype":"Sync","status":"DONE","failureMessage":"","executionTime":131118,"startTime":"2020-05-2
9T19:17:10.966Z","endTime":"2020-05-29T19:19:22.084Z","bytesMarkedForCopy":0,"filesMarkedForC
opy":0,"dirsMarkedForCopy":0,"filesCopied":0,"bytesCopied":0,"dirsCopied":0,"filesFailed":
0,"bytesFailed":0,"dirsFailed":0,"filesMarkedforRemove":0,"bytesMarkedForRemove":0,"dirsMarke
dForRemove":0,"filesRemoved":0,"bytesRemoved":0,"dirsRemoved":0,"bytesRemovedFailed":0,"files
RemovedFailed":0,"filesMarkedForGrace":0,"bytesMarkedForGrace":0,"dirsMarkedForGrace":0,"file
sMarkedForIgnore":0,"dirsScanned":2,"filesScanned":1,"dirsFailedToScan":0,"bytesScanned":0,"p
rogress":100,"lastMessageTime":"2020-05-29T19:19:22.107Z","topFiveMostCommonRelationshipError
s":[]}}]
```

Now, we will print out the list of relationships. Identify the specific relationship in this list that you wish to trigger an update for, and note the relationship id. You will need to enter this relationships id in the next section. If you have multiple relationship set up for the same source and destination, then you will want to change 'printFullDetails' to True.

```
In [10]: printFullDetails = False

numRelationships = 0
for relationship in relationships :
    numRelationships+=1
    print("-- Relationship #", numRelationships, "--\n")
    if printFullDetails:
        print(json.dumps(relationship, indent=2), "\n")
    else :
        print("id: ", relationship["id"])
        print("source: ", json.dumps(relationship["source"], indent=2))
        print("target: ", json.dumps(relationship["target"], indent=2), "\n")
```

```
-- Relationship # 1 --

id:  5ed00996ca85650009a83db2
source:  {
  "protocol": "nfs",
  "nfs": {
    "host": "192.168.200.41",
    "export": "/trident_pvc_1b3d8a1c_b3d5_4a3a_a767_a936dfe52871",
    "path": "",
    "version": "3",
    "provider": "nfs"
  }
}
target:  {
  "protocol": "nfs",
  "nfs": {
    "host": "192.168.200.41",
    "export": "/trident_pvc_0361a52b_9f65_4adc_9092_aafeb602a809",
    "path": "",
    "version": "3",
    "provider": "nfs"
  }
}
```

Set Cloud Sync relationship id

Note: this is the same relationship id that we just retrieved in the previous section.

```
In [11]: relationshipId = "5ed00996ca85650009a83db2"
```

Trigger Cloud Sync update

Lastly, we will call the function that we defined above to trigger an update for our specified Cloud Sync relationship. If you receive an error, try restarting the kernel and running again.

```
In [12]: netappCloudSyncUpdate(refreshToken = refreshToken, relationshipId = relationshipId)
```

```
Note: Status Code 202 denotes that update was successfully triggered.
API Response:
Status Code:  202
Header:  {'Date': 'Fri, 29 May 2020 19:23:17 GMT', 'Content-Type': 'application/json; charset
=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'X-DNS-Prefetch-Contro
l': 'off', 'Strict-Transport-Security': 'max-age=15552000; includeSubDomains', 'X-Download-Op
tions': 'noopen', 'X-Content-Type-Options': 'nosniff', 'X-XSS-Protection': '1; mode=block',
'X-DFIO-Req-Id': 'fcf19b4e-9a11-42f0-8dbc-181eb242ee73', 'Access-Control-Allow-Origin': '*',
'Vary': 'Accept-Encoding'}
```

Check Cloud Sync progress

```
In [13]: netappCloudSyncMonitor(refreshToken = refreshToken, relationshipId = relationshipId,
                                keepCheckingUntilComplete = True)
```

```
Cloud Sync update is not yet complete.
Checking again in 60 seconds...
Cloud Sync update is not yet complete.
Checking again in 60 seconds...
Success: Cloud Sync update is complete.
```

Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning

To define and execute a new Kubeflow Pipeline that takes advantage of NetApp Snapshot technology in order to integrate rapid and efficient dataset and model versioning and traceability into an end-to-end AI/ML model training workflow, perform the following tasks. For more information about Kubeflow pipelines, see the [official Kubeflow documentation](). Note that the example pipeline that is shown in this section only works with volumes that reside on ONTAP storage systems or software-defined instances.

1. Create a Kubernetes secret containing the username and password of the cluster admin account for the ONTAP cluster on which your volumes reside. This secret must be created in the `kubeflow` namespace because this is the namespace that pipelines are executed in. Note that you must replace `username` and `password` with your username and password when executing these commands, and you must use the output of the base64 commands (see highlighted text) in your secret definition accordingly.

```
$ echo -n 'username' | base64
dXNlcm5hbWU=
$ echo -n 'password' | base64
cGFzc3dvcmQ=
$ cat << EOF > ./secret-ontap-cluster-mgmt-account.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ontap-cluster-mgmt-account
  namespace: kubeflow
data:
  username: dXNlcm5hbWU=
  password: cGFzc3dvcmQ=
EOF
$ kubectl create -f ./secret-ontap-cluster-mgmt-account.yaml
secret/ontap-cluster-mgmt-account created
```

2. If the volume containing the data that you plan to use to train your model is not tied to a PVC in the `kubeflow` namespace, then you must import this volume into that namespace. Use the Trident volume import functionality to import this volume. The volume must be imported into the `kubeflow` namespace because this is the namespace that pipelines are executed in.

   If your dataset volume is already tied to a PVC in the `kubeflow` namespace, then you can skip this step. If you do not yet have a dataset volume, then you must provision one and then transfer your data to it. See the section "Provision a New Volume" for an example showing how to provision a new volume with Trident.

   The example commands that follow show the importing of an existing FlexVol volume, named `dataset_vol`, into the `kubeflow` namespace. For more information about PVCs, see the [official Kubernetes documentation](). For more information about the volume import functionality, see the [Trident documentation](). For a detailed example showing the importing of a volume using Trident, see the section "Import an Existing Volume."

```
$ cat << EOF > ./pvc-import-dataset-vol-kubeflow.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: dataset-vol
  namespace: kubeflow
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ontap-ai-flexvols-retain
EOF
$ tridentctl import volume ontap-ai-flexvols dataset_vol -f ./pvc-import-dataset-vol-
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
dataset-vol                                 Bound    pvc-43b12235-f32e-4dc4-a7b8-88e90d935a12
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
```

3. If the volume on which you wish to store your trained model is not tied to a PVC in the `kubeflow` namespace, then you must import this volume into that namespace. Use the Trident volume import functionality to import this volume. The volume must be imported into the `kubeflow` namespace because this is the namespace that pipelines are executed in.

   If your trained model volume is already tied to a PVC in the `kubeflow` namespace, then you can skip this step. If you do not yet have a trained model volume, then you must provision one. See the section “Provision a New Volume” for an example showing how to provision a new volume with Trident.

   The example commands that follow show the importing of an existing FlexVol volume, named `kfp_model_vol`, into the `kubeflow` namespace. For more information about PVCs, see the [official Kubernetes documentation]. For more information about the volume import functionality, see the [Trident documentation]. For a detailed example showing the importing of a volume using Trident, see the section “Import an Existing Volume.”

```
$ cat << EOF > ./pvc-import-dataset-vol-kubeflow.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: kfp-model-vol
  namespace: kubeflow
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ontap-ai-flexvols-retain
EOF
$ tridentctl import volume ontap-ai-flexvols kfp_model_vol -f ./pvc-import-kfp-model-vol-
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
kfp-model-vol                               Bound    pvc-236e893b-63b4-40d3-963b-e709b9b2816b
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
```

4. If you have not already done so, you must install the Kubeflow Pipelines SDK. See the official Kubeflow documentation for installation instructions.
5. Define your Kubeflow Pipeline in Python using the Kubeflow Pipelines SDK. The example commands that follow show the creation of a pipeline definition script for a pipeline that accepts the following parameters at run-time and then executes the following steps. Modify the pipeline definition script as needed depending on your specific process.

Run-time parameters:

- `ontap_cluster_mgmt_hostname`: The host name or IP address of the ONTAP cluster on which your dataset and model volumes are stored.
- `ontap_cluster_admin_acct_k8s_secret`: the name of the Kubernetes secret that was created in step 1.
- `ontap_verify_ssl_cert`: Denotes whether to verify your cluster's SSL certificate when communicating with the ONTAP API (true/false).
- `dataset_volume_pvc_existing`: The name of the Kubernetes PersistentVolumeClaim (PVC) in the `kubeflow` namespace that is tied to the volume that contains the data that you want to use to train your model.
- `dataset_volume_pv_existing`: the name of the Kubernetes PersistentVolume (PV) object that corresponds to the dataset volume PVC. To get the name of the PV, you can run `kubectl -n kubeflow get pvc`. The name of the PV that corresponds to a given PVC can be found in the `VOLUME` column.
- `trained_model_volume_pvc_existing`: The name of the Kubernetes PersistentVolumeClaim (PVC) in the `kubeflow` namespace that is tied to the volume on which you want to store your trained model.
- `trained_model_volume_pv_existing`: The name of the Kubernetes PersistentVolume (PV) object that corresponds to the trained model volume PVC. To get the name of the PV, you can run `kubectl -n kubeflow get pvc`. The name of the PV that corresponds to a given PVC can be found in the `VOLUME` column.
- `execute_data_prep_step__yes_or_no`: Denotes whether you wish to execute a data prep step as part of this particular pipeline execution (yes/no).
- `data_prep_step_container_image`: The container image in which you wish to execute your data prep step.
- `data_prep_step_command`: The command that you want to execute as your data prep step.
- `data_prep_step_dataset_volume_mountpoint`: The mountpoint at which you want to mount your dataset volume for your data prep step.
- `train_step_container_image`: The container image in which you wish to execute your training step.
- `train_step_command`: The command that you want to execute as your training step.
- `train_step_dataset_volume_mountpoint`: The mountpoint at which you want to mount your dataset volume for your training step.
- `train_step_model_volume_mountpoint`: The mountpoint at which you want to mount your model volume for your training step.
- `validation_step_container_image`: The container image in which you wish to execute your validation step.
- `validation_step_command`: The command that you want to execute as your validation step.

- `validation_step_dataset_volume_mountpoint`: the mountpoint at which you want to mount your dataset volume for your validation step.
- `validation_step_model_volume_mountpoint`: The mountpoint at which you want to mount your model volume for your validation step.

Pipeline steps:

a. Optional: Execute a data prep step.

b. Trigger the creation of a Snapshot copy, using NetApp Snapshot technology, of your dataset volume.

Note: This Snapshot copy is created for traceability purposes. Each time that this pipeline workflow is executed, a Snapshot copy is created. Therefore, as long as the Snapshot copy is not deleted, it is always possible to trace a specific training run back to the exact training dataset that was used for that run.

c. Execute a training step.

d. Trigger the creation of a Snapshot copy, using NetApp Snapshot technology, of your trained model volume.

Note: This Snapshot copy is created for versioning purposes. Each time that this pipeline workflow is executed, a Snapshot copy is created. Therefore, for each individual training run, a read-only versioned copy of the resulting trained model is automatically saved.

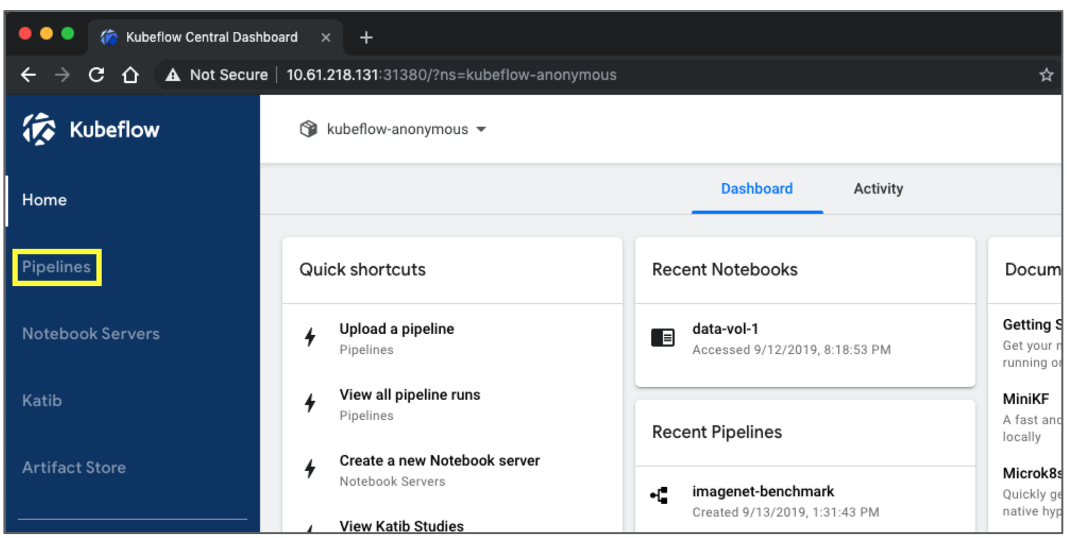
e. Execute a validation step.

```
$ git clone https://github.com/NetApp/kubeflow_jupyter_pipeline.git
$ cd kubeflow_jupyter_pipeline/Pipelines/
$ vi ai-training-run.py
```

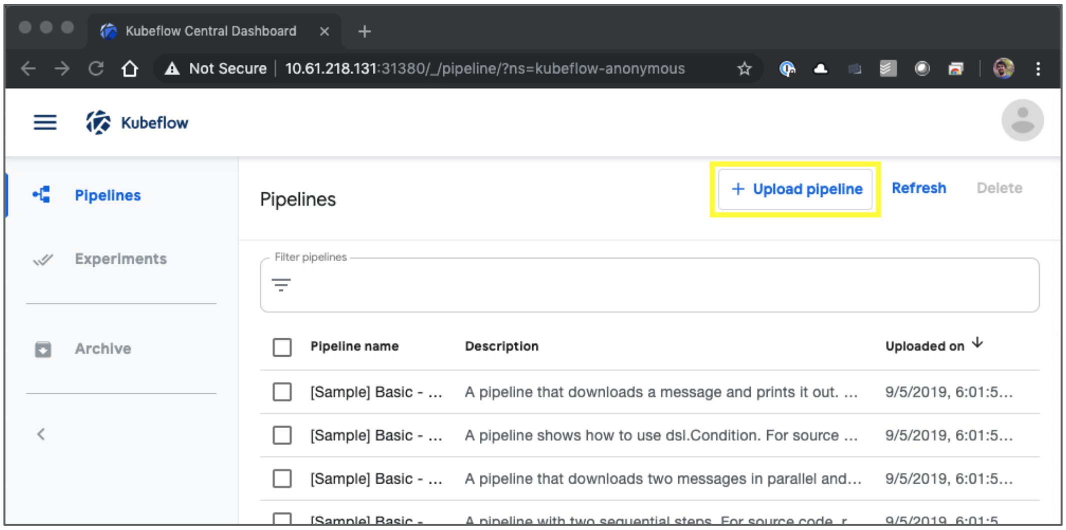
6. Execute the pipeline definition script that you created in step 5 to create a `.yaml` manifest for your pipeline.

```
$ python3 ai-training-run.py
$ ls ai-training-run.py.yaml
ai-training-run.py.yaml
```

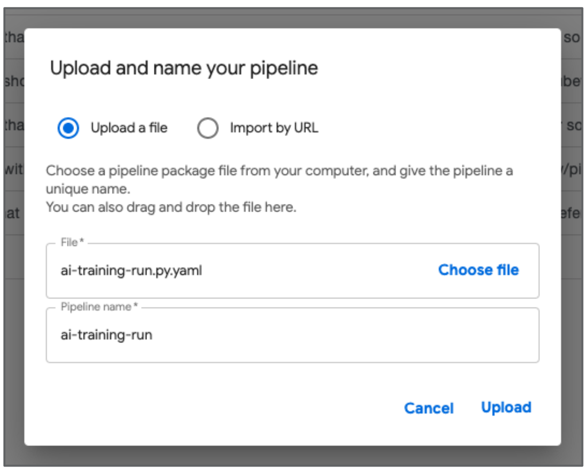
7. From the Kubeflow central dashboard, click Pipelines in the main menu to navigate to the Kubeflow Pipelines administration page.



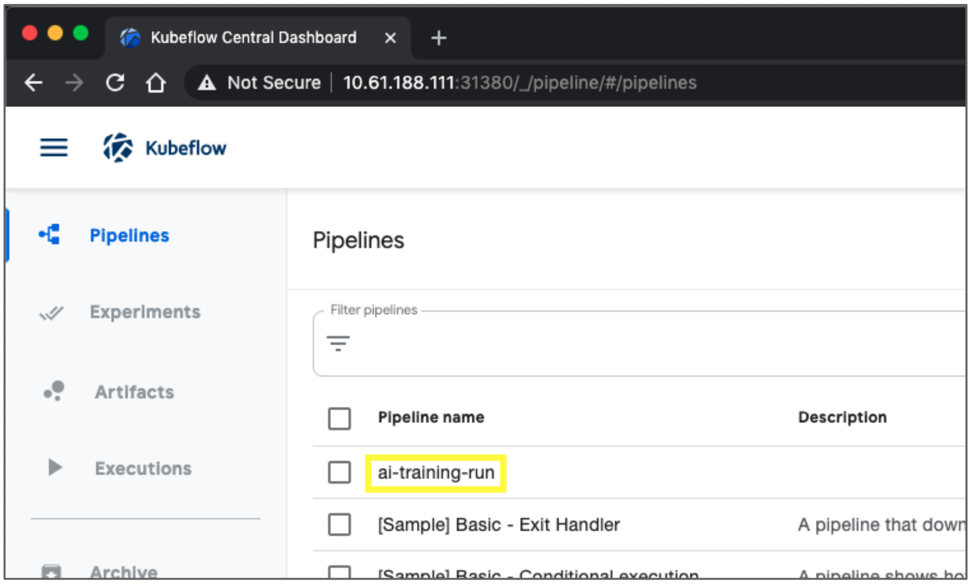
8. Click Upload Pipeline to upload your pipeline definition.



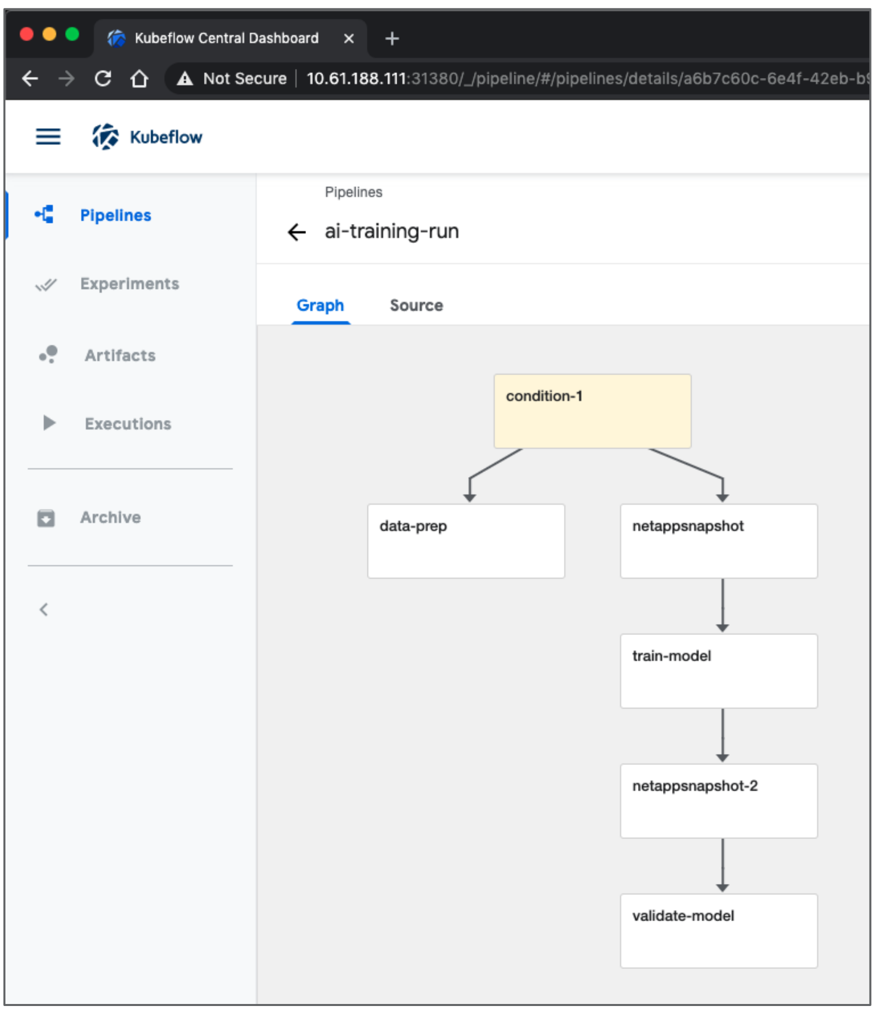
9. Choose the `.yaml` manifest for your pipeline that you created in step 6, give your pipeline a name, and click Upload.



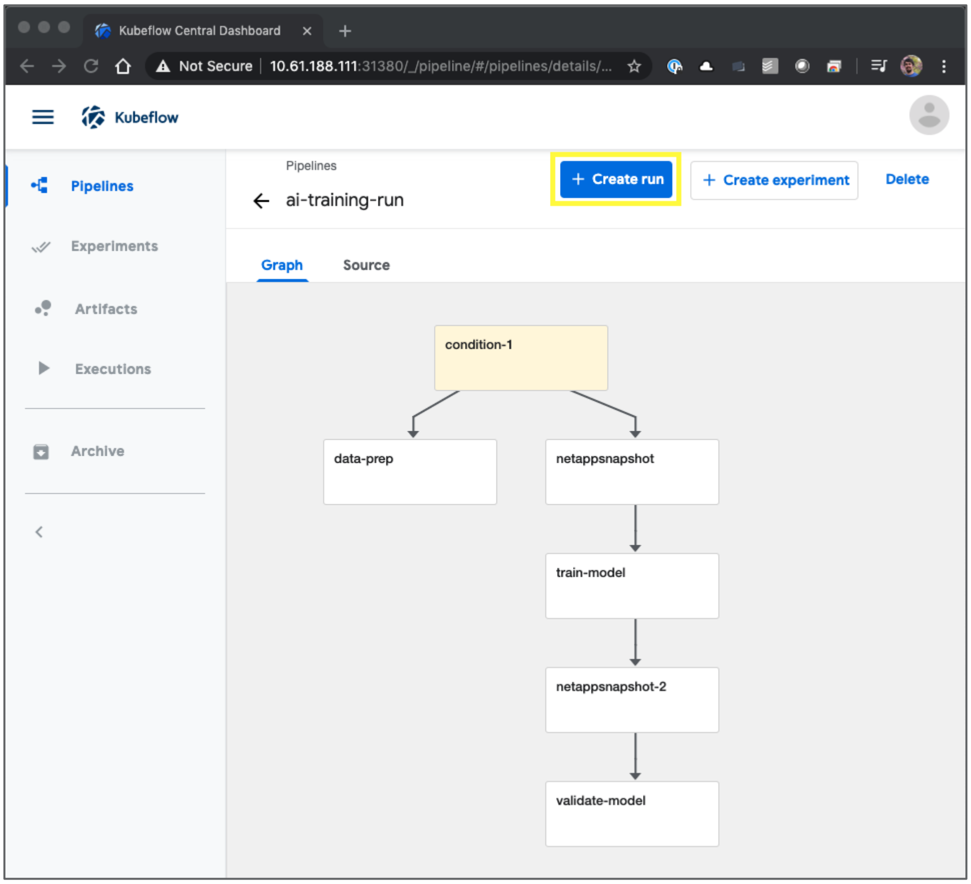
10. You should now see your new pipeline in the list of pipelines on the pipeline administration page. Click your pipeline's name to view it.



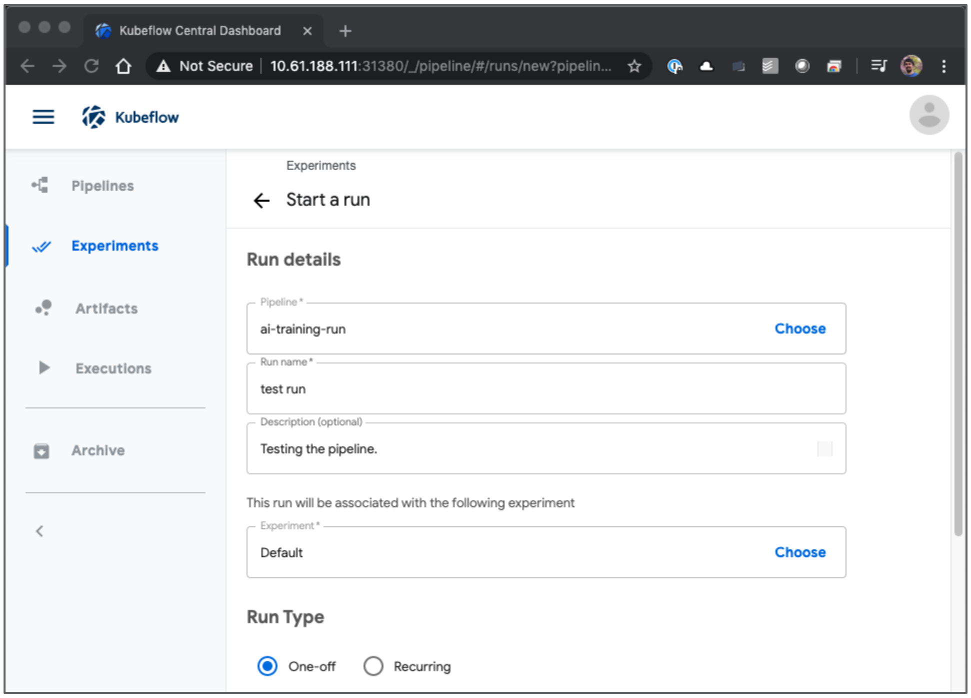
11. Review your pipeline to confirm that it looks correct.



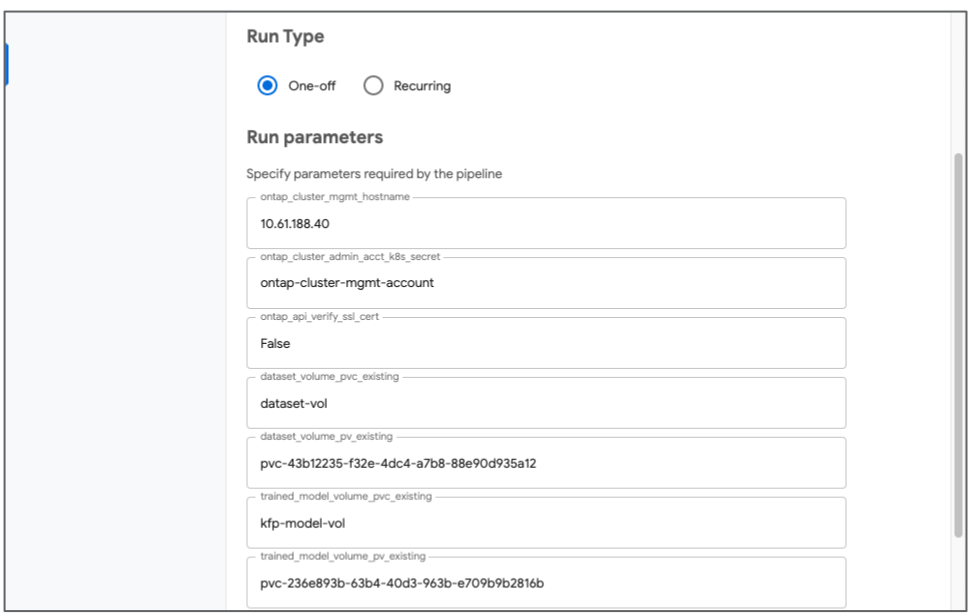
12. Click Create run to run your pipeline.

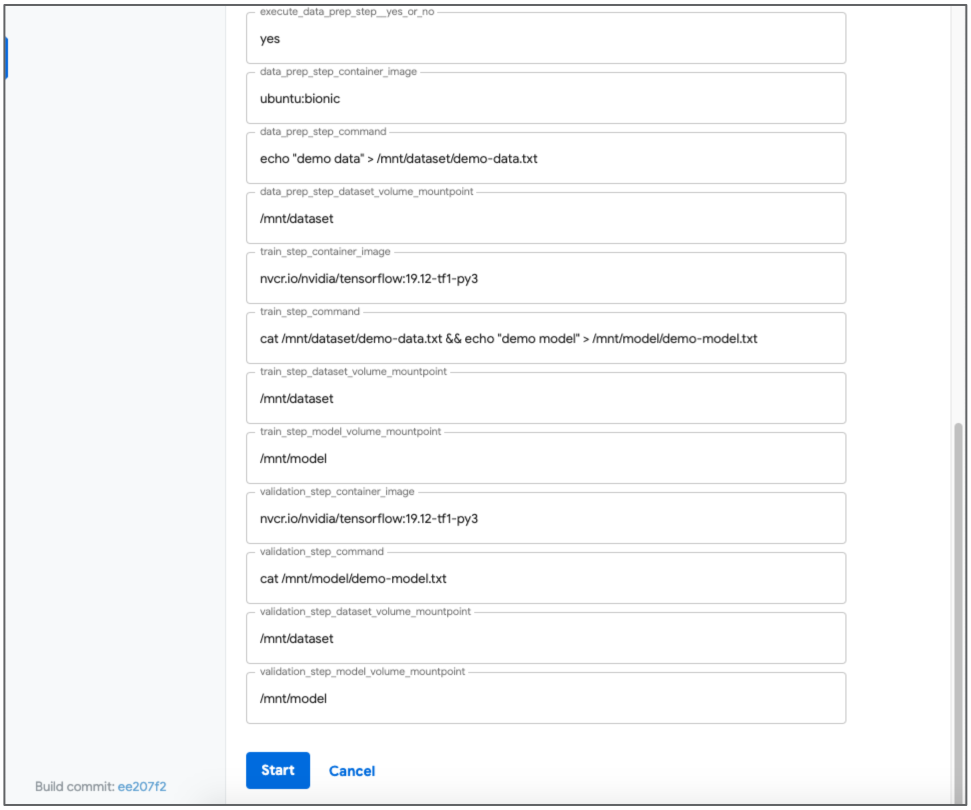


13. You are now presented with a screen from which you can start a pipeline run. Create a name for the run, enter a description, choose an experiment to file the run under, and choose whether you want to initiate a one-off run or schedule a recurring run.

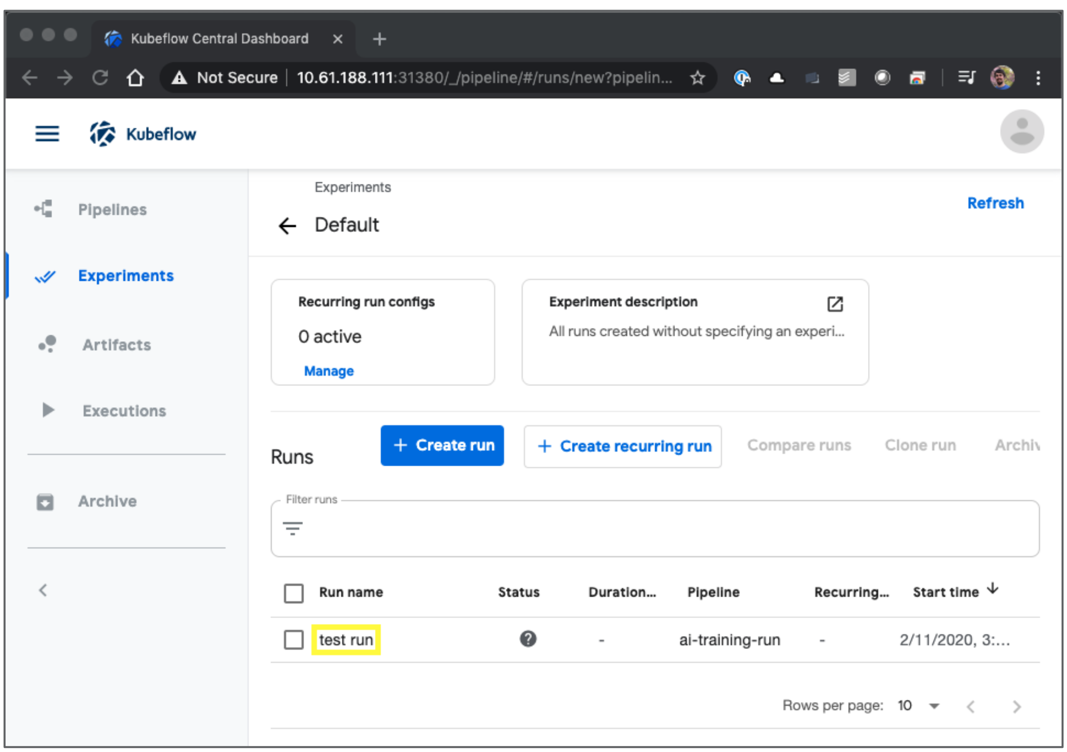


14. Define parameters for the run, and then click Start. In the following example, the default values are accepted for most parameters. Details for the volume that was imported into the `kubeflow` namespace in step 2 are entered for `dataset_volume_pvc_existing` and `dataset_volume_pv_existing`. Details for the volume that was imported into the `kubeflow` namespace in step 3 are entered for `trained_model_volume_pvc_existing` and `trained_model_volume_pv_existing`. Non-AI-related commands are entered for the `data_prep_step_command`, `train_step_command`, and `validation_step_command` parameters in order to plainly demonstrate the functionality of the pipeline. Note that you defined the default values for the parameters within your pipeline definition (see step 5).

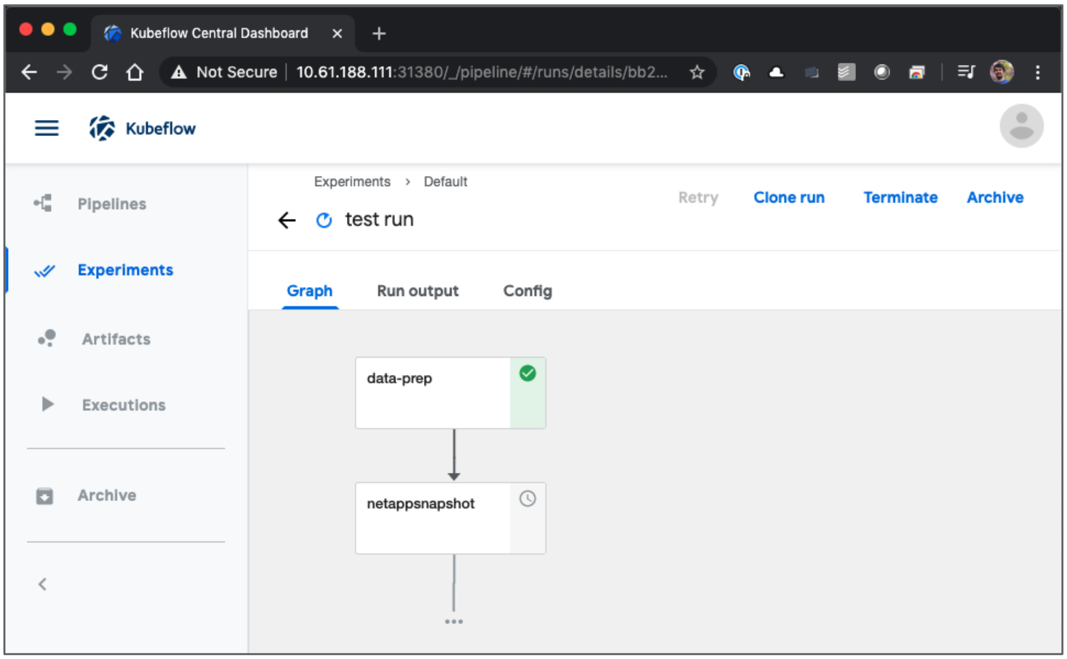




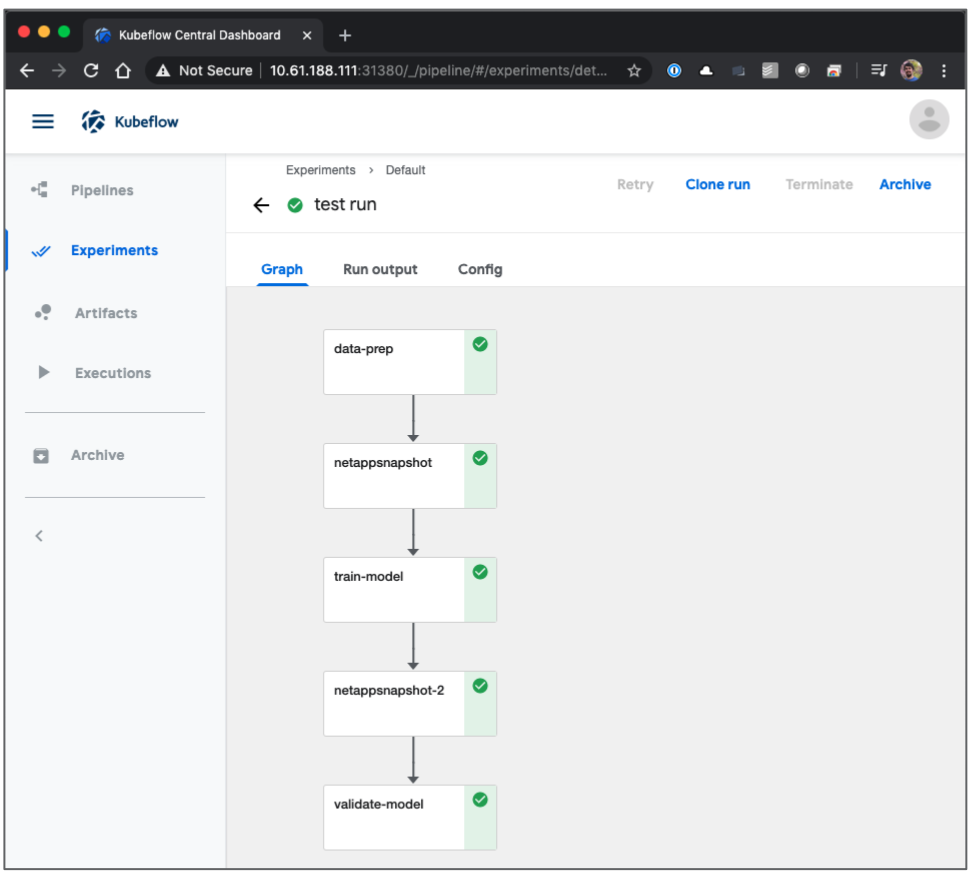
15. You are now presented with a screen listing all runs that fall under the specific experiment. Click the name of the run that you just started to view it.



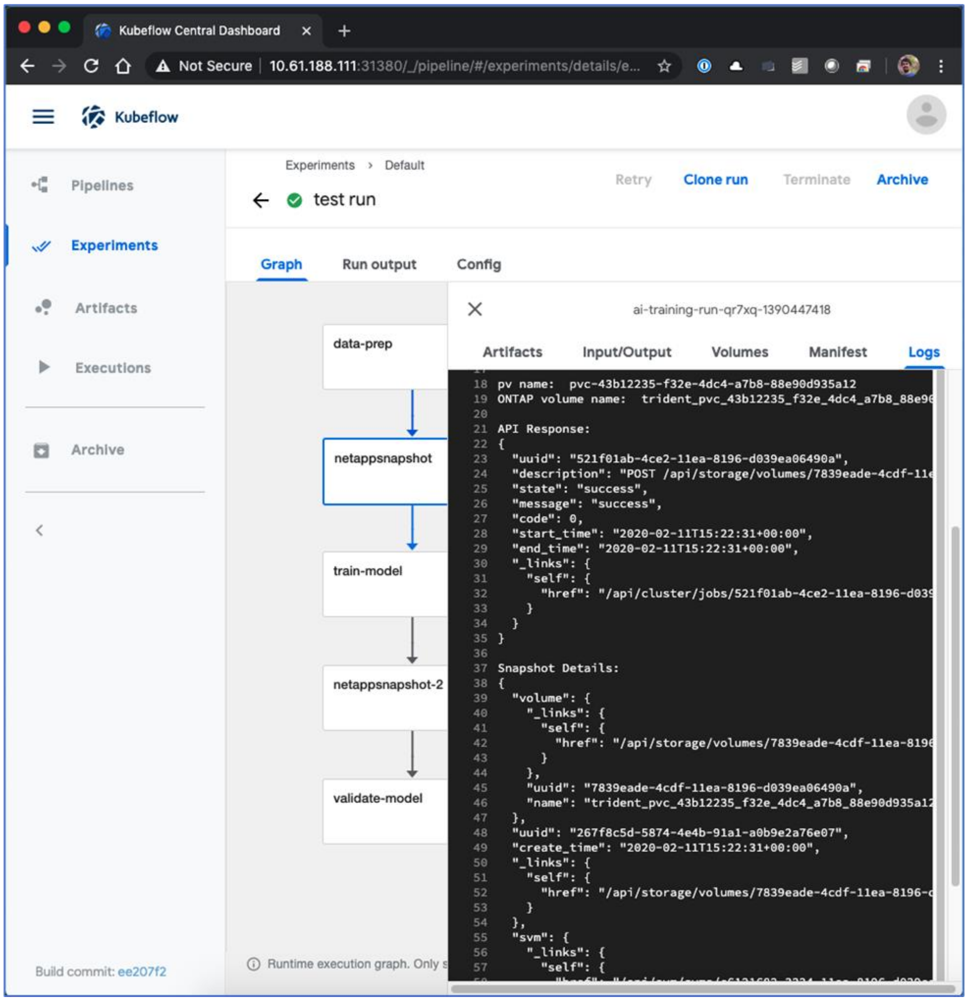
16. At this point, the run is likely still in progress.

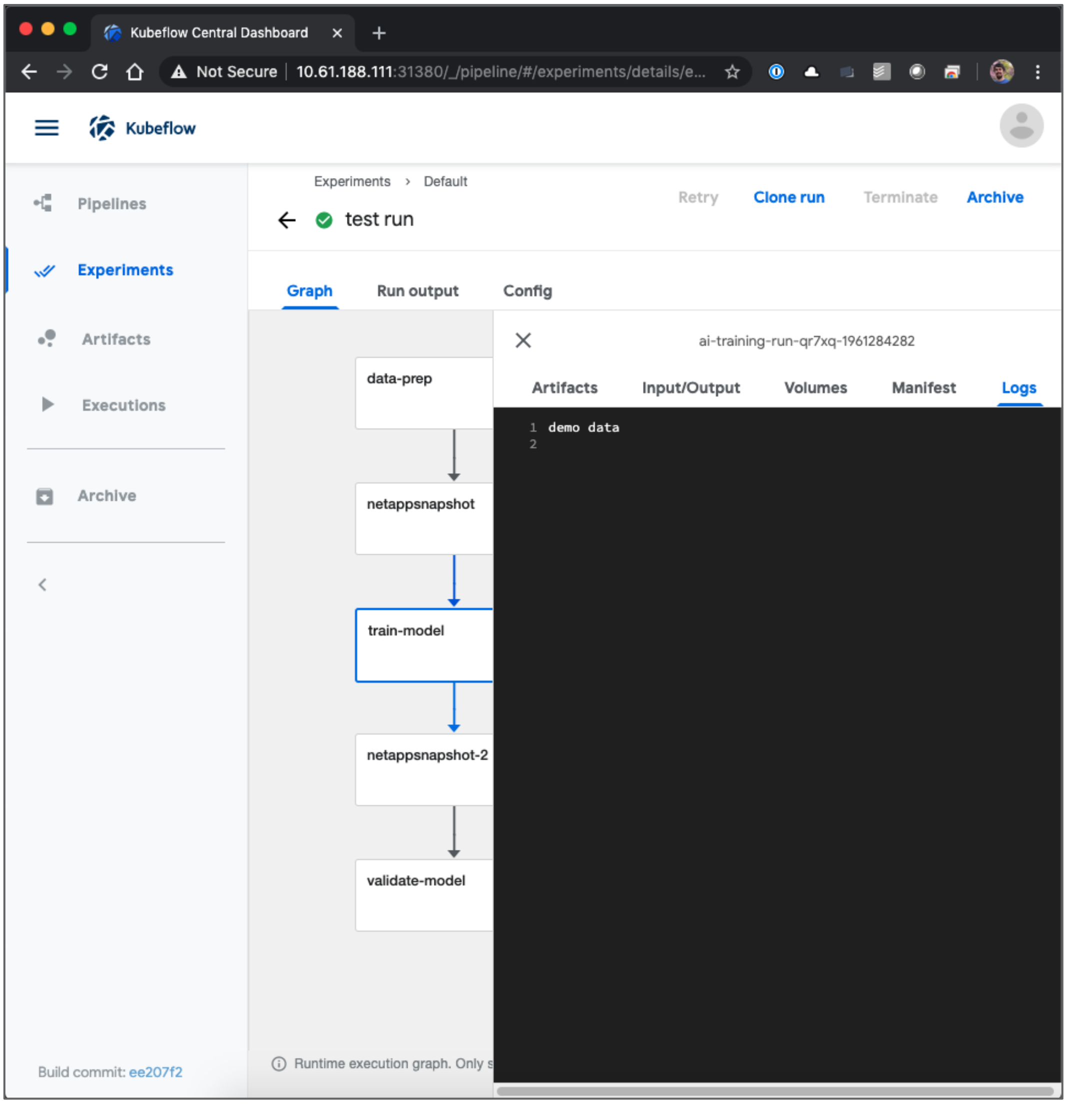


17. Confirm that the run completed successfully. When the run is complete, every stage of the pipeline shows a green check-mark icon.

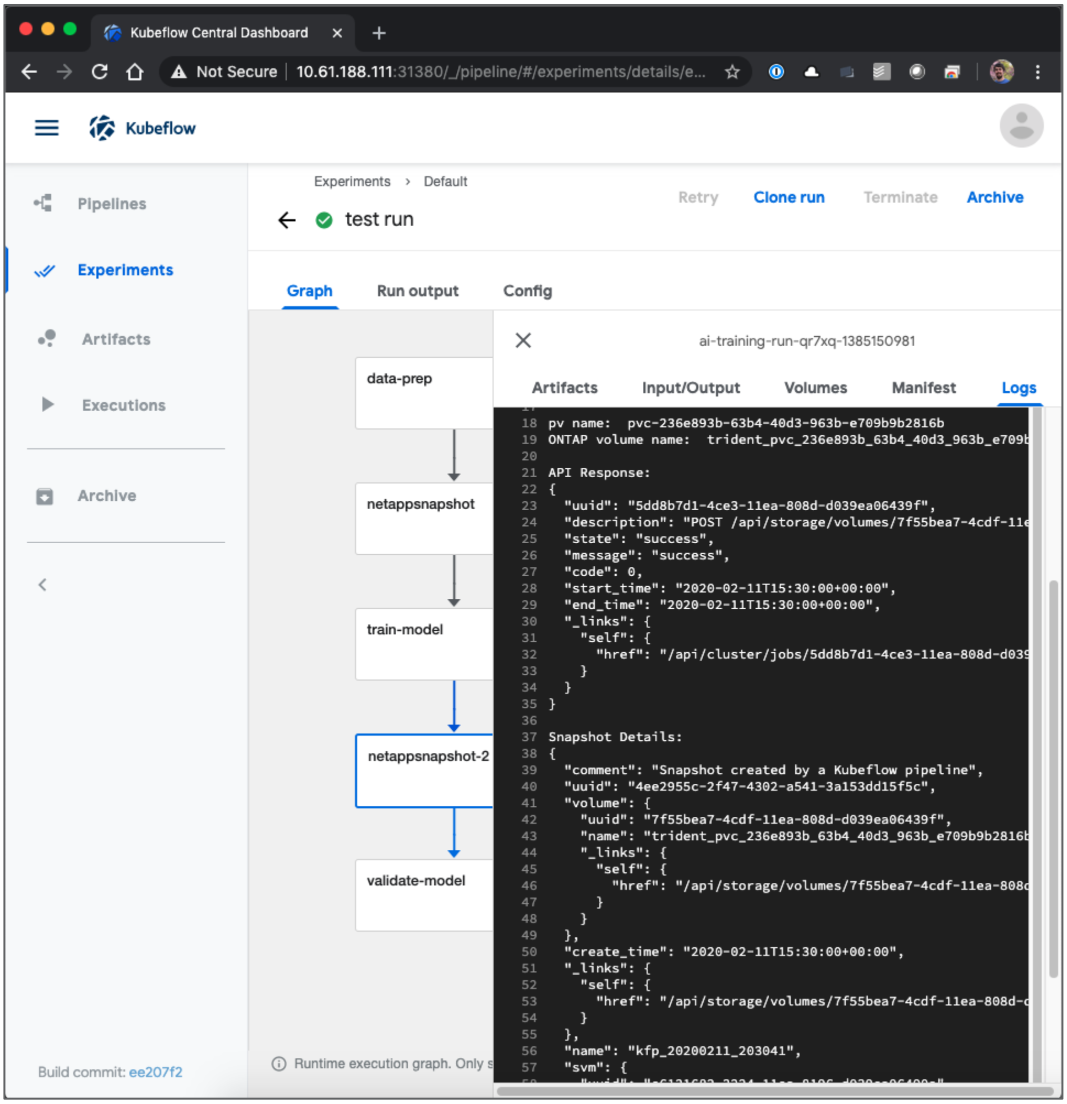


18. Click a specific stage, and then click Logs to view output for that stage.

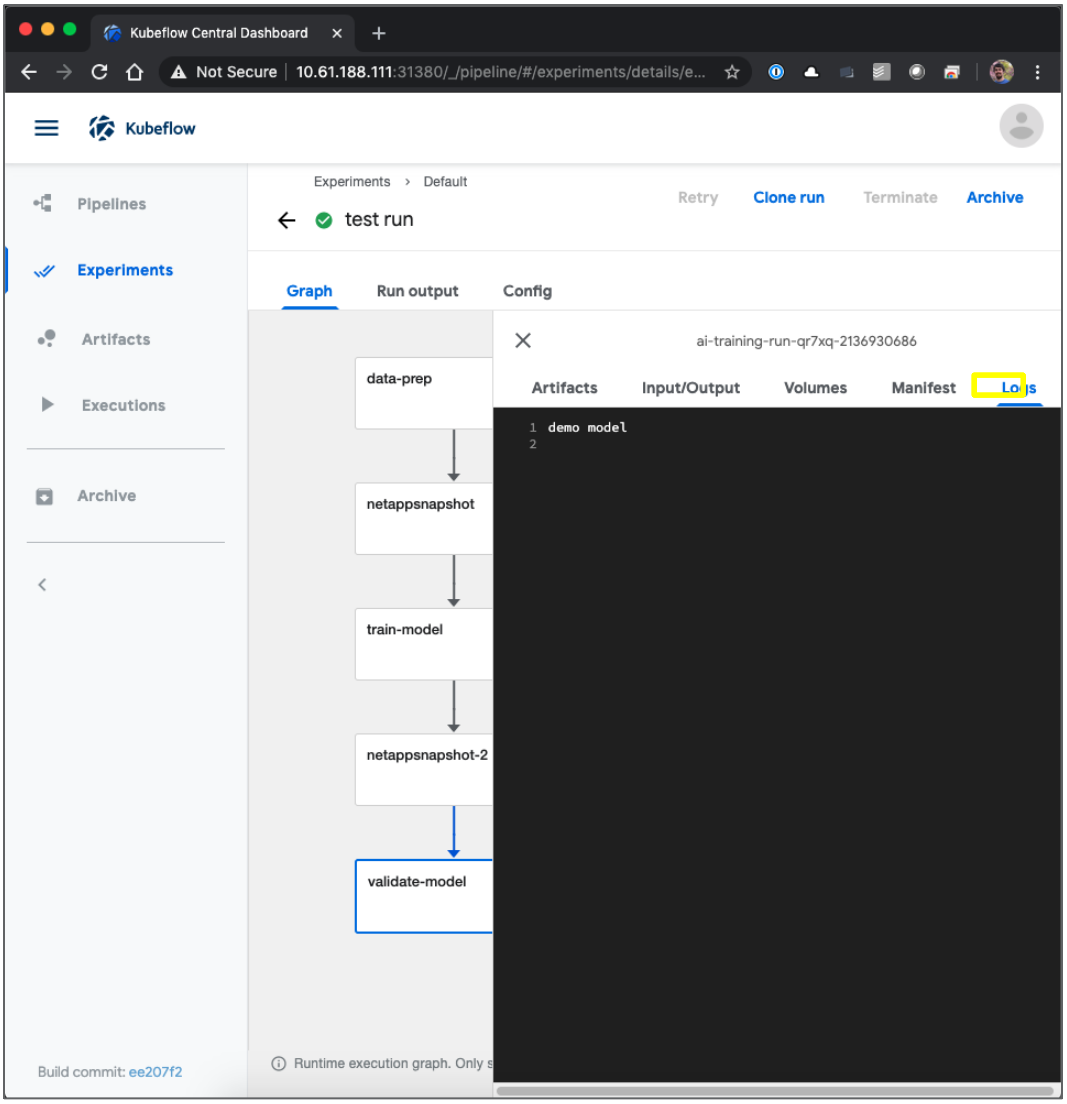




Kubeflow Central Dashboard
Not Secure
10.61.188.111:31380/_/pipeline/#/experiments/details/e...
Kubeflow
Pipelines
Experiments
Artifacts
Executions
Archive
Build commit: ee207f2
Experiments › Default
test run
Retry
Clone run
Terminate
Archive
Graph
Run output
Config
data-prep
netappsnapshot
train-model
netappsnapshot-2
validate-model
Runtime execution graph. Only s
ai-training-run-qr7xq-1961284282
Artifacts
Input/Output
Volumes
Manifest
Logs
1 demo data
2



Kubeflow Central Dashboard
Not Secure | 10.61.188.111:31380/_/pipeline/#/experiments/details/e...
Kubeflow
Pipelines
Experiments
Artifacts
Executions
Archive
Build commit: ee207f2
Experiments › Default
test run
Retry
Clone run
Terminate
Archive
Graph
Run output
Config
data-prep
netappsnapshot
train-model
netappsnapshot-2
validate-model
Runtime execution graph. Only s
ai-training-run-qr7xq-1385150981
Artifacts
Input/Output
Volumes
Manifest
Logs
pv name: pvc-236e893b-63b4-40d3-963b-e709b9b2816b
ONTAP volume name: trident_pvc_236e893b_63b4_40d3_963b_e709b
API Response:
{
"uuid": "5dd8b7d1-4ce3-11ea-808d-d039ea06439f",
"description": "POST /api/storage/volumes/7f55bea7-4cdf-11e
"state": "success",
"message": "success",
"code": 0,
"start_time": "2020-02-11T15:30:00+00:00",
"end_time": "2020-02-11T15:30:00+00:00",
"_links": {
"self": {
"href": "/api/cluster/jobs/5dd8b7d1-4ce3-11ea-808d-d03
}
}
}
Snapshot Details:
{
"comment": "Snapshot created by a Kubeflow pipeline",
"uuid": "4ee2955c-2f47-4302-a541-3a153dd15f5c",
"volume": {
"uuid": "7f55bea7-4cdf-11ea-808d-d039ea06439f",
"name": "trident_pvc_236e893b_63b4_40d3_963b_e709b9b2816b
"_links": {
"self": {
"href": "/api/storage/volumes/7f55bea7-4cdf-11ea-808d
}
}
},
"create_time": "2020-02-11T15:30:00+00:00",
"_links": {
"self": {
"href": "/api/storage/volumes/7f55bea7-4cdf-11ea-808d-d
}
},
"name": "kfp_20200211_203041",
"svm": {



Create a Kubeflow Pipeline to Rapidly Clone a Dataset for a Data Scientist Workspace

Perform the following tasks to define and execute a new Kubeflow Pipeline that takes advantage of NetApp FlexClone technology to clone a dataset volume rapidly and efficiently and create a data scientist or developer workspace. For more information about Kubeflow Pipelines, see the official Kubeflow documentation.

Note: The example Kubeflow Pipeline that is detailed in this section is not compatible with FlexGroup volumes. At the time of this writing, FlexGroup volumes must be cloned by using ONTAP System Manager, the ONTAP CLI, or the ONTAP API, and then imported into the Kubernetes cluster. For details about importing a volume using Trident, see the section “Import an Existing Volume.”

1. If you have not already done so, you must install the Kubeflow Pipelines SDK. For installation instructions, see the official Kubeflow documentation.
2. Define your Kubeflow Pipeline in Python using the Kubeflow Pipelines SDK. The example commands that follow show the creation of a pipeline definition script for a pipeline that accepts the following parameters at run-time and then executes the following steps. Modify the pipeline definition script as needed depending on your specific process.

Run-time parameters:

– `workspace_name`: The name that you want to give to your new workspace.
– `dataset_volume_pvc_existing`: The name of the Kubernetes PersistentVolumeClaim (PVC) that corresponds to the dataset volume that you wish to clone.
– `dataset_volume_pvc_existing_size`: The size of the dataset volume that you wish to clone; for example, 10Gi, 100Gi, or 2Ti.
– `trident_storage_class`: The Kubernetes StorageClass that the dataset volume you wish to clone is associated with.
– `jupyter_namespace`: The namespace in which you intend to create a Jupyter Notebook workspace. For details about creating a Jupyter Notebook workspace, see the section “Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use.” The dataset clone that this pipeline creates is mountable in the Jupyter Notebook workspace.

Note: The existing dataset volume PVC that you wish to clone from (the value of the `dataset_volume_pvc_existing` parameter) must be in this same namespace.

Pipeline steps:

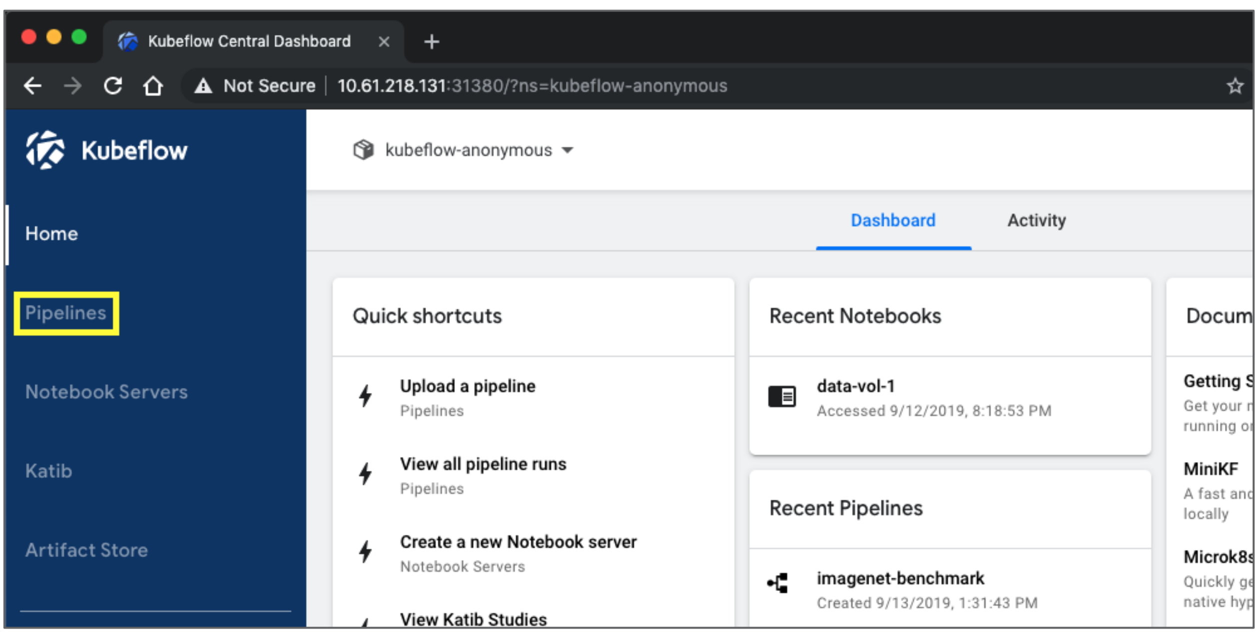
a. Trigger the creation of a clone, using NetApp FlexClone technology, of your dataset volume.
b. Print instructions for deploying an interactive Jupyter Notebook workspace that has access to the dataset clone.

```
$ git clone https://github.com/NetApp/kubeflow_jupyter_pipeline.git
$ cd kubeflow_jupyter_pipeline/Pipelines/
$ vi create-data-scientist-workspace.py
```

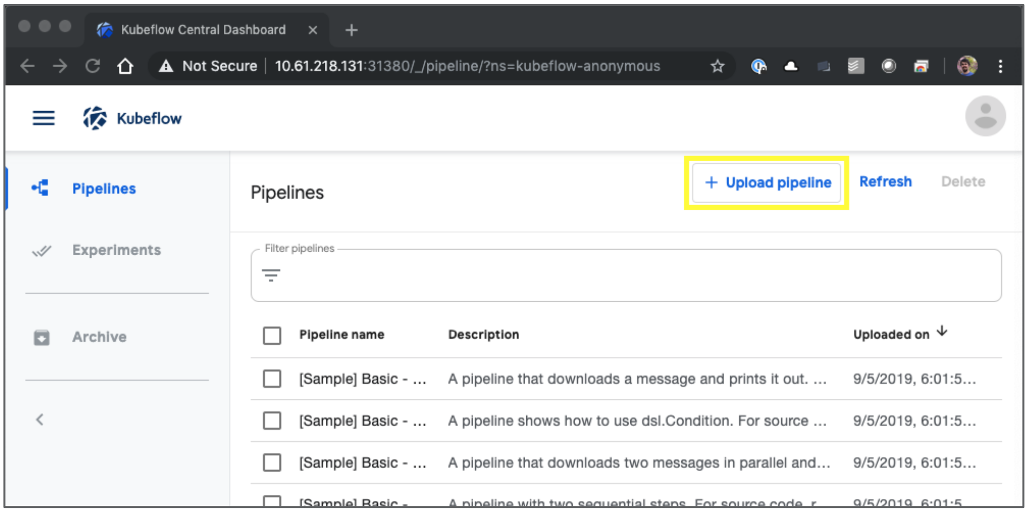
3. Execute the pipeline definition script that you created in step 2 to create a `.yaml` manifest for your pipeline.

```
$ python3 create-data-scientist-workspace.py
$ ls create-data-scientist-workspace.py.yaml
create-data-scientist-workspace.py.yaml
```

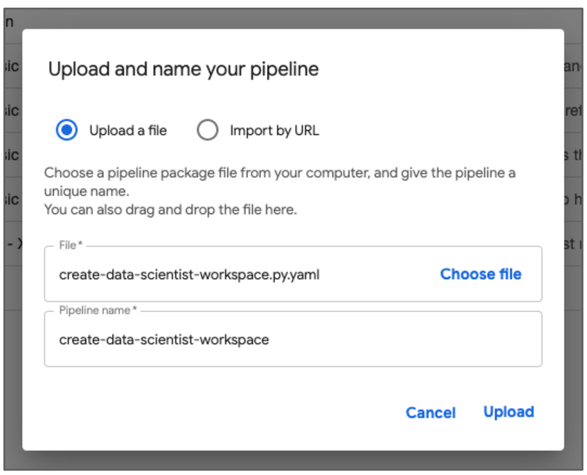
4. From the Kubeflow central dashboard, click Pipelines in the main menu to navigate to the Kubeflow Pipelines administration page.



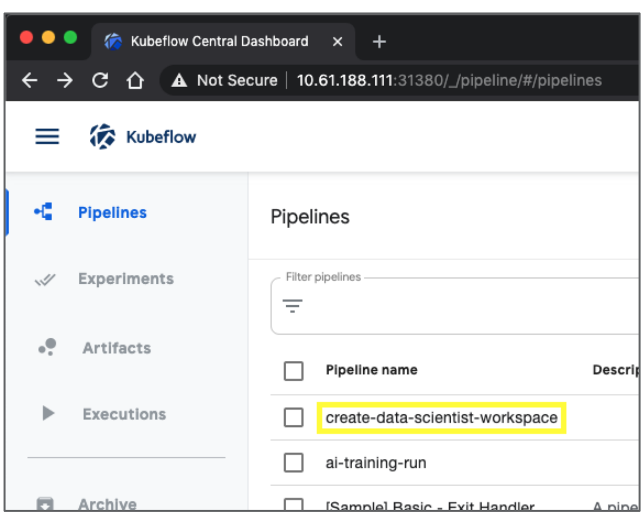
5. Click Upload Pipeline to upload your pipeline definition.



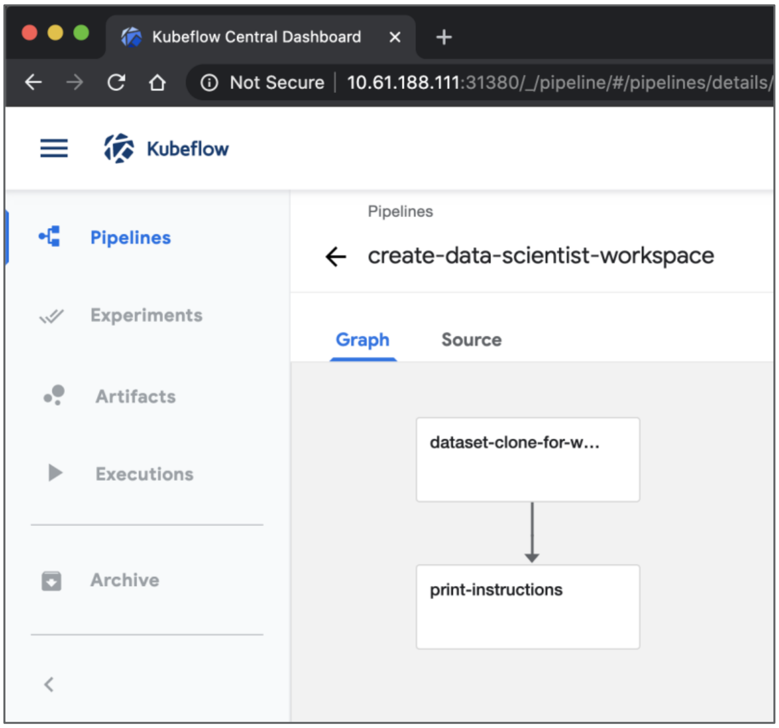
6. Choose the `.yaml` file containing your pipeline definition that you created in step 3, give your pipeline a name, and click Upload.



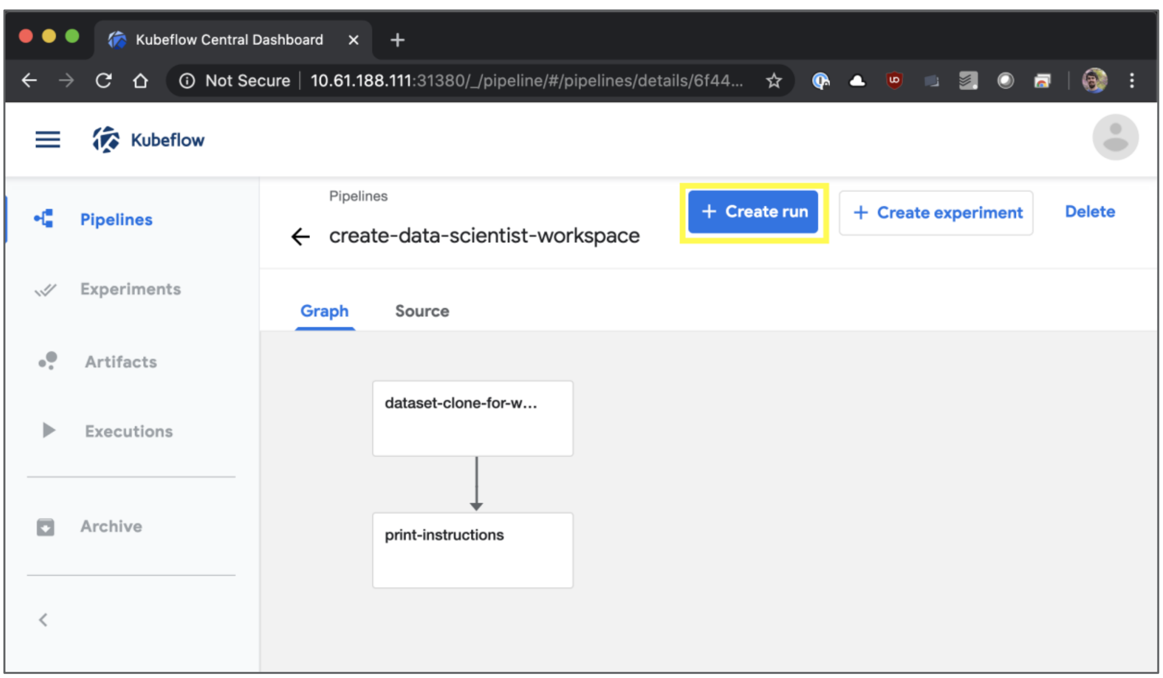
7. You should now see your new pipeline in the list of pipelines on the pipeline administration page. Click your pipeline's name to view it.



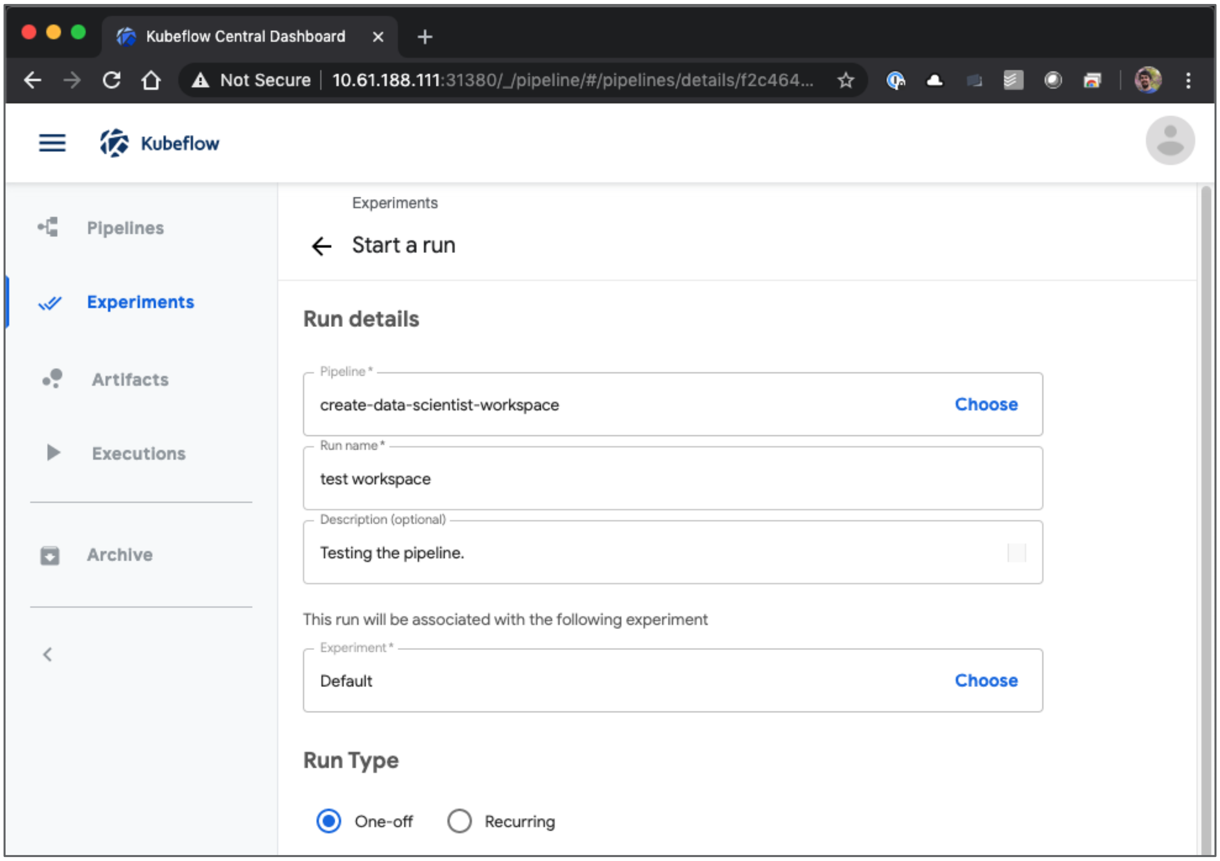
8. Review your pipeline to confirm that it looks correct.



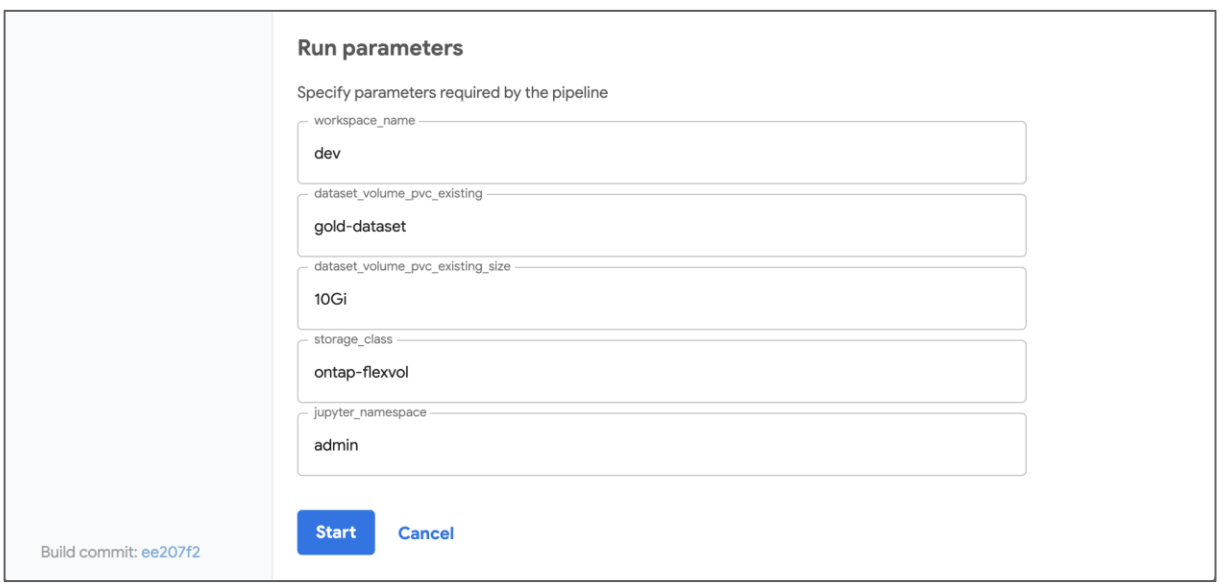
9. Click Create run to run your pipeline.



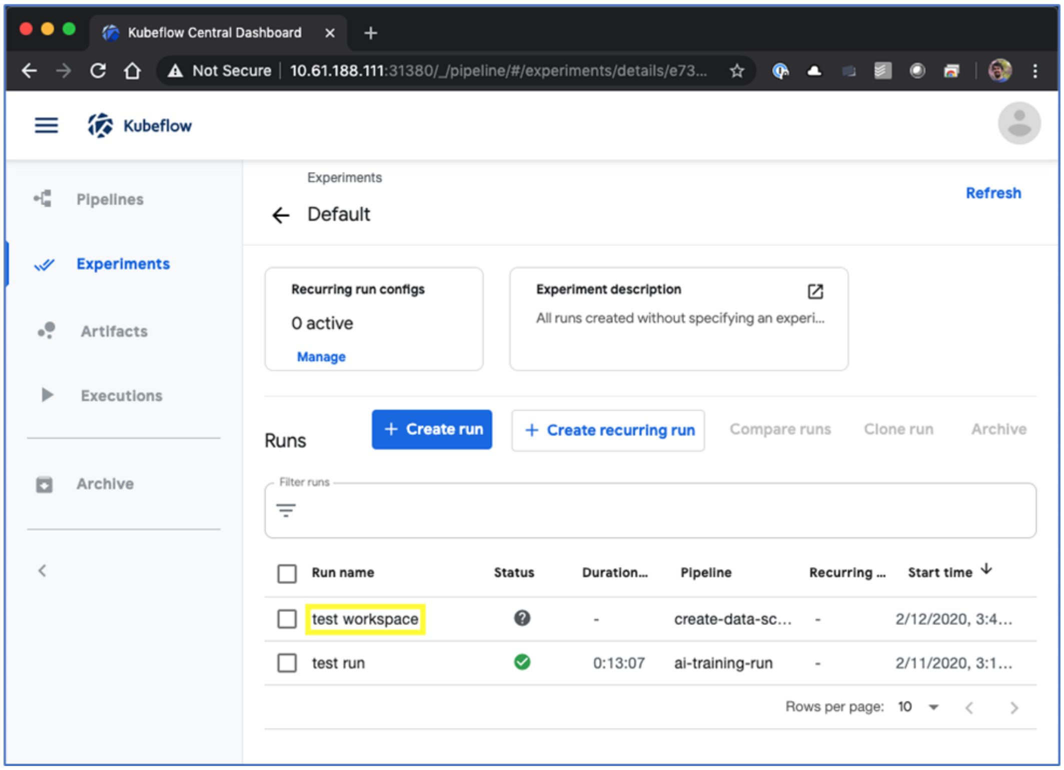
10. You are now presented with a screen from which you can start a pipeline run. Create a name for the run, enter a description, select an experiment to file the run under, and select whether you want to initiate a one-off run or schedule a recurring run.



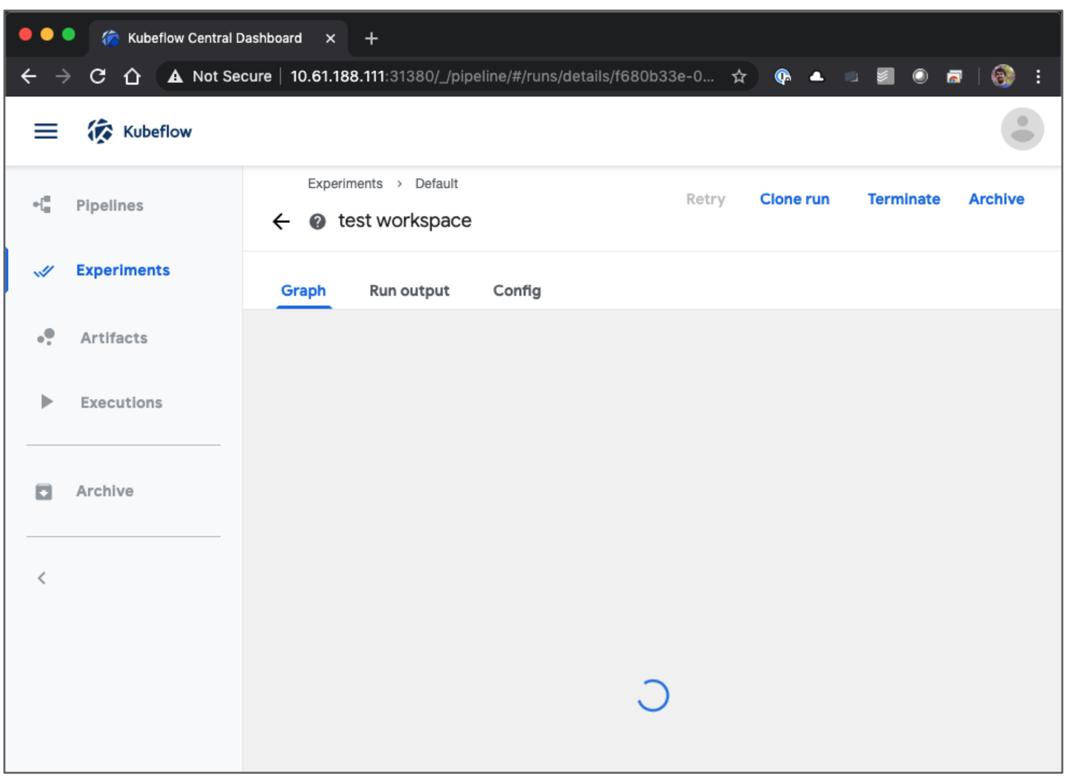
11. Define parameters for the run, and then click Start. Reference step 2 for details on the individual parameters.



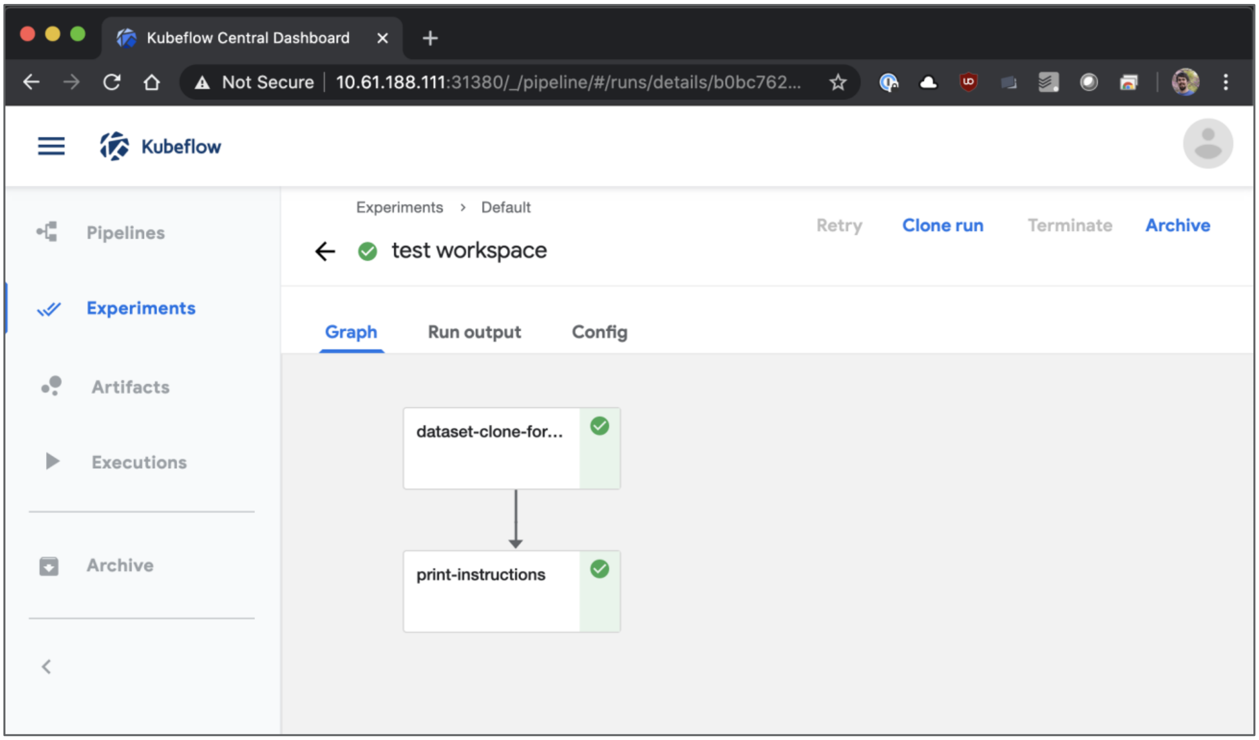
12. You are now presented with a screen listing all runs that fall under the specific experiment. Click the name of the run that you just started to view it.



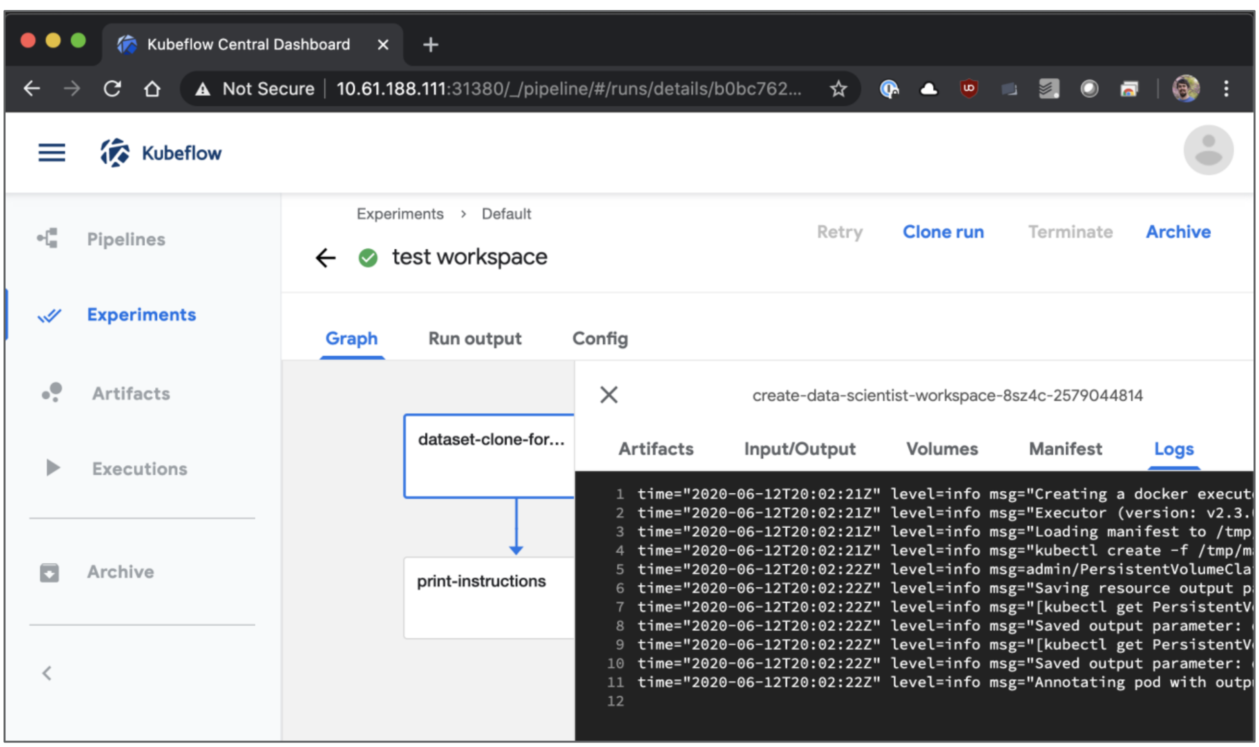
13. At this point, the run is likely still in progress.



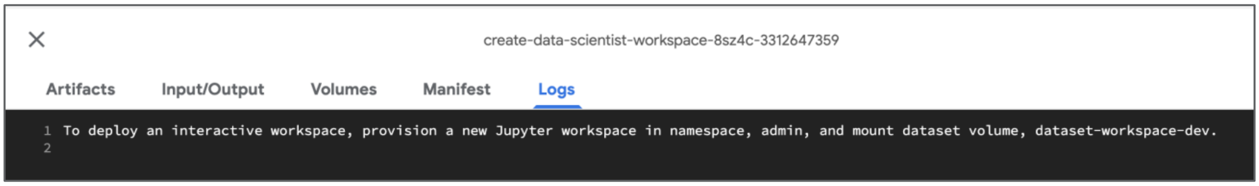
14. Confirm that the run completed successfully. When the run is complete, every stage of the pipeline shows a green check-mark icon.



15. Click the `dataset-clone-for-workspace` stage, and then click Logs to view output for that stage.



16. Click the `print-instructions` stage, and then click Logs to view the outputted instructions. See the section “Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use” for details on creating a Jupyter Notebook workspace.



Create a Kubeflow Pipeline to Trigger a SnapMirror Volume Replication Update

You can define and execute a new Kubeflow pipeline that takes advantage of NetApp SnapMirror data replication technology to replicate the contents of a volume between different ONTAP clusters.

This pipeline can be used to replicate data of any type between ONTAP clusters that might or might not be located at different sites or in different regions. Potential use cases include:

- Replicating newly acquired sensor data gathered at the edge back to the core data center or to the cloud to be used for AI/ML model training or retraining.
- Replicating a newly trained or newly-updated model from the core data center to the edge or to the cloud to be deployed as part of an inferencing application.

For more information about Kubeflow pipelines, see the official Kubeflow documentation. Note that the example pipeline that is shown in this section only works with volumes that reside on ONTAP storage systems or software-defined instances.

To create a new Kubeflow pipeline to trigger a SnapMirror volume replication update, perform the following steps:

Note: Before you perform the exercises that are outlined in this section, we assume that you have already initiated an asynchronous SnapMirror relationship between the source and the destination volume according to standard configuration instructions. For details, refer to official NetApp documentation.

1. If you have not already done so, create a Kubernetes secret containing the username and password of the cluster admin account for the ONTAP cluster on which your destination volume resides
2. This secret must be created in the `kubeflow` namespace because this is the namespace that pipelines are executed in. Replace `username` and `password` with your username and password when executing these commands and use the output of the base64 commands (see highlighted text) in your secret definition accordingly.

```
$ echo -n 'username' | base64
dXNlcm5hbWU=
$ echo -n 'password' | base64
cGFzc3dvcmQ=
$ cat << EOF > ./secret-ontap-cluster-mgmt-account.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ontap-cluster-mgmt-account
  namespace: kubeflow
data:
  username: dXNlcm5hbWU=
  password: cGFzc3dvcmQ=
EOF
$ kubectl create -f ./secret-ontap-cluster-mgmt-account.yaml
secret/ontap-cluster-mgmt-account created
```

3. If you have not already done so, install the Kubeflow Pipelines SDK. See the official Kubeflow documentation for installation instructions.

4. Define your Kubeflow Pipeline in Python using the Kubeflow Pipelines SDK. The example commands that follow show the creation of a pipeline definition script for a pipeline that accepts the following parameters at run-time and then executes the following steps. Modify the pipeline definition script as needed depending on your specific process.

   Pipeline steps:

   a. Trigger a replication update for the specified asynchronous SnapMirror relationship.

   Run-time parameters:

   – `ontap_cluster_mgmt_hostname`: The host name or IP address of the ONTAP cluster on which the destination volume resides.

   – `ontap_cluster_admin_acct_k8s_secret`: The name of the Kubernetes secret that was created in step 1.

   – `ontap_api_verify_ssl_cert`: Denotes whether to verify your cluster's SSL certificate when communicating with the ONTAP API (yes/no).

   – `source_svm`: The name of the SVM on which the source volume resides.

   – `source_volume`: The name of the source volume (the volume that you are replicating from) on the source cluster.

   – `destination_svm`: The name of the SVM on which the destination volume resides.

   – `destination_volume`: The name of the destination volume (the volume that you are replicating to) on the destination cluster.

```
$ git clone https://github.com/NetApp/kubeflow_jupyter_pipeline.git
$ cd kubeflow_jupyter_pipeline/Pipelines/
$ vi replicate-data-snapmirror.py
```

5. Execute the pipeline definition script that you created in step 4 to create a `.yaml` manifest for your pipeline.

```
$ python3 replicate-data-snapmirror.py
$ ls replicate-data-snapmirror.py.yaml
replicate-data-snapmirror.py.yaml
```

6. Follow steps 7 through 18 from the section "Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning."

   Be sure to use the `.yaml` manifest that was created in the previous step (step 5) of this section instead of the manifest that was created in the section "Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning."

Create a Kubeflow Pipeline to Trigger a Cloud Sync Replication Update

You can define and execute a new Kubeflow pipeline that takes advantage of NetApp Cloud Sync replication technology to replicate data to and from a variety of file and object storage platforms. Potential use cases include:

- Replicating newly-acquired sensor data gathered at the edge back to the core data center or to the cloud to be used for AI/ML model training or retraining.
- Replicating a newly-trained or newly-updated model from the core data center to the edge or to the cloud to be deployed as part of an inferencing application.
- Copying data from an S3 data lake to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Copying data from a Hadoop data lake (through Hadoop NFS Gateway) to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Saving a new version of a trained model to an S3 or Hadoop data lake for permanent storage.

- Copying NFS-accessible data from a legacy or non-NetApp system of record to a high-performance AI/ML training environment for use in the training of an AI/ML model.

For more information about Kubeflow pipelines, see the official Kubeflow documentation.

Note: The example pipeline that is shown in this section only works with volumes that reside on ONTAP storage systems or software-defined instances.

To create a new Kubeflow pipeline to trigger a Cloud Sync replication update, perform the following steps:

Note: Before you perform the exercises that are outlined in this section, we assume that you have already initiated the Cloud Sync relationship that you wish to trigger an update for. To initiate a relationship, visit cloudsync.netapp.com.

1. If you do not yet have a Cloud Sync API refresh token, access the following URL using your web browser to create one: https://services.cloud.netapp.com/refresh-token.
2. If you have not already done so, create a Kubernetes secret containing your Cloud Sync API refresh token. This secret must be created in the `kubeflow` namespace because this is the namespace that pipelines are executed in. Replace `<your refresh token>` with your refresh token when executing these commands and use the output of the base64 command (see highlighted text) in your secret definition accordingly.

```
$ echo -n '<your refresh token>' | base64
PHlvdXIgcmVmcmVzaCB0b2tlbj4=
$ cat << EOF > ./secret-cloud-sync-refresh-token.yaml
apiVersion: v1
kind: Secret
metadata:
  name: cloud-sync-refresh-token
  namespace: kubeflow
data:
  refreshToken: PHlvdXIgcmVmcmVzaCB0b2tlbj4=
EOF
$ kubectl create -f ./secret-cloud-sync-refresh-token.yaml
secret/ secret-cloud-sync-refresh-token created
```

3. If you have not already done so, install the Kubeflow Pipelines SDK. For installation instructions, see the official Kubeflow documentation.
4. Define your Kubeflow Pipeline in Python using the Kubeflow Pipelines SDK. The example commands that follow show the creation of a pipeline definition script for a pipeline that accepts the following parameters at run-time and then executes the following steps. Modify the pipeline definition script as needed depending on your specific process.

   Pipeline steps:

   a. Trigger a replication update for the specified Cloud Sync relationship.

   Run-time parameters:

   – `cloud_sync_relationship_id`: The relationship ID of the Cloud Sync relationship for which you want to trigger an update. If you do not know the relationship ID, you can retrieve it by using the Jupyter Notebook that is included in the section “Trigger a Cloud Sync Replication Update from Within a Jupyter Notebook” or by directly calling the Relationships-v2 API.

   – `cloud_sync_refresh_token_k8s_secret`: The name of the Kubernetes secret that was created in step 2.

```
$ git clone https://github.com/NetApp/kubeflow_jupyter_pipeline.git
$ cd kubeflow_jupyter_pipeline/Pipelines/
$ vi replicate-data-cloud-sync.py
```

5. Execute the pipeline definition script that you created in step 4 to create a `.yaml` manifest for your pipeline

```
$ python3 replicate-data-cloud-sync.py
$ ls replicate-data-cloud-sync.py.yaml
```



```
replicate-data-cloud-sync.py.yaml
```

6. Follow steps 7 through 18 from the section “Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning.”

   Be sure to use the `.yaml` manifest that was created in the previous step (step 5) of this section instead of the manifest that was created in the section “Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning.”

Apache Airflow Deployment

NetApp recommends running Apache Airflow on top of Kubernetes. This section describes the tasks that you must complete to deploy Airflow in your Kubernetes cluster.

Note: It is possible to deploy Airflow on platforms other than Kubernetes. Deploying Airflow on platforms other than Kubernetes is outside of the scope of this document.

Prerequisites

Before you perform the deployment exercise that is outlined in this section, we assume that you have already performed the following tasks:

1. You already have a working Kubernetes cluster.
2. You have already installed and configured NetApp Trident in your Kubernetes cluster as outlined in the section “NetApp Trident Deployment and Configuration.”

Install Helm

Airflow is deployed using Helm, a popular package manager for Kubernetes. Before you deploy Airflow, you must install Helm on the deployment jump host. To install Helm on the deployment jump host, follow the installation instructions in the official Helm documentation.

Set Default Kubernetes StorageClass

Before you deploy Airflow, you must designate a default StorageClass within your Kubernetes cluster. The Airflow deployment process attempts to provision new persistent volumes 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, follow the instructions outlined in the section “Set Default Kubernetes StorageClass.” If you have already designated a default StorageClass within your cluster, then you can skip this step.

Use Helm to Deploy Airflow

To deploy Airflow in your Kubernetes cluster using Helm, perform the following tasks from the deployment jump host:

1. Deploy Airflow using Helm by following the deployment instructions for the official Airflow chart on the Helm Hub. The example commands that follow show the deployment of Airflow using Helm. Modify, add, and/or remove values in the `custom-values.yaml` file as needed depending on your environment and desired configuration.

```
$ cat << EOF > custom-values.yaml
###################################
# Airflow - Common Configs
###################################
airflow:
  ## the airflow executor type to use
  ##
  executor: "KubernetesExecutor"
```



```
  ## environment variables for the web/scheduler/worker Pods (for airflow configs)
  ##
  config:
    AIRFLOW__KUBERNETES__DELETE_WORKER_PODS: "False"
    AIRFLOW__KUBERNETES__GIT_REPO: "git@github.com:mboglesby/airflow-dev.git"
    AIRFLOW__KUBERNETES__GIT_BRANCH: master
    AIRFLOW__KUBERNETES__GIT_DAGS_FOLDER_MOUNT_POINT: "/opt/airflow/dags"
    AIRFLOW__KUBERNETES__DAGS_VOLUME_SUBPATH: "repo/"
    AIRFLOW__KUBERNETES__GIT_SSH_KEY_SECRET_NAME: "airflow-git-key"
    AIRFLOW__KUBERNETES__WORKER_CONTAINER_REPOSITORY: "apache/airflow"
    AIRFLOW__KUBERNETES__WORKER_CONTAINER_TAG: "1.10.12"
    AIRFLOW__KUBERNETES__RUN_AS_USER: "50000"
    AIRFLOW__KUBERNETES__LOGS_VOLUME_CLAIM: "airflow-k8s-exec-logs"

workers:
  enabled: false # Celery workers

###################################
# Airflow - WebUI Configs
###################################
web:
  ## configs for the Service of the web Pods
  ##
  service:
    type: NodePort

###################################
# Airflow - Logs Configs
###################################
logs:
  persistence:
    enabled: true

###################################
# Airflow - DAGs Configs
###################################
dags:
  ## configs for the DAG git repository & sync container
  ##
  git:
    ## url of the git repository
    ##
    url: "git@github.com:mboglesby/airflow-dev.git"

    ## the branch/tag/sha1 which we clone
    ##
    ref: master

    ## the name of a pre-created secret containing files for ~/.ssh/
    ##
    ## NOTE:
    ## - this is ONLY RELEVANT for SSH git repos
    ## - the secret commonly includes files: id_rsa, id_rsa.pub, known_hosts
    ## - known_hosts is NOT NEEDED if `git.sshKeyscan` is true
    ##
    secret: "airflow-git-key-files"
    sshKeyscan: true

    ## the name of the private key file in your `git.secret`
    ##
    ## NOTE:
    ## - this is ONLY RELEVANT for PRIVATE SSH git repos
    ##
    privateKeyName: id_rsa

    ## the host name of the git repo
    ##
    ## NOTE:
    ## - this is ONLY REQUIRED for SSH git repos
    ##
    ## EXAMPLE:
```



```
    ##   repoHost: "github.com"
    ##
    repoHost: "github.com"

    ## the port of the git repo
    ##
    ## NOTE:
    ## - this is ONLY REQUIRED for SSH git repos
    ##
    repoPort: 22

    ## configs for the git-sync container
    ##
    gitSync:
      ## enable the git-sync sidecar container
      ##
      enabled: true

      ## the git sync interval in seconds
      ##
      refreshTime: 60
EOF
$ helm install "airflow" stable/airflow --version "7.10.1" --namespace "airflow" --values
./custom-values.yaml
NAME: airflow
LAST DEPLOYED: Mon Oct  5 18:32:11 2020
NAMESPACE: airflow
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Congratulations. You have just deployed Apache Airflow!

1. Get the Airflow Service URL by running these commands:
   export NODE_PORT=$(kubectl get --namespace airflow -o jsonpath="{.spec.ports[0].nodePort}"
services airflow-web)
   export NODE_IP=$(kubectl get nodes --namespace airflow -o
jsonpath="{.items[0].status.addresses[0].address}")
   echo http://$NODE_IP:$NODE_PORT/

2. Open Airflow in your web browser
```

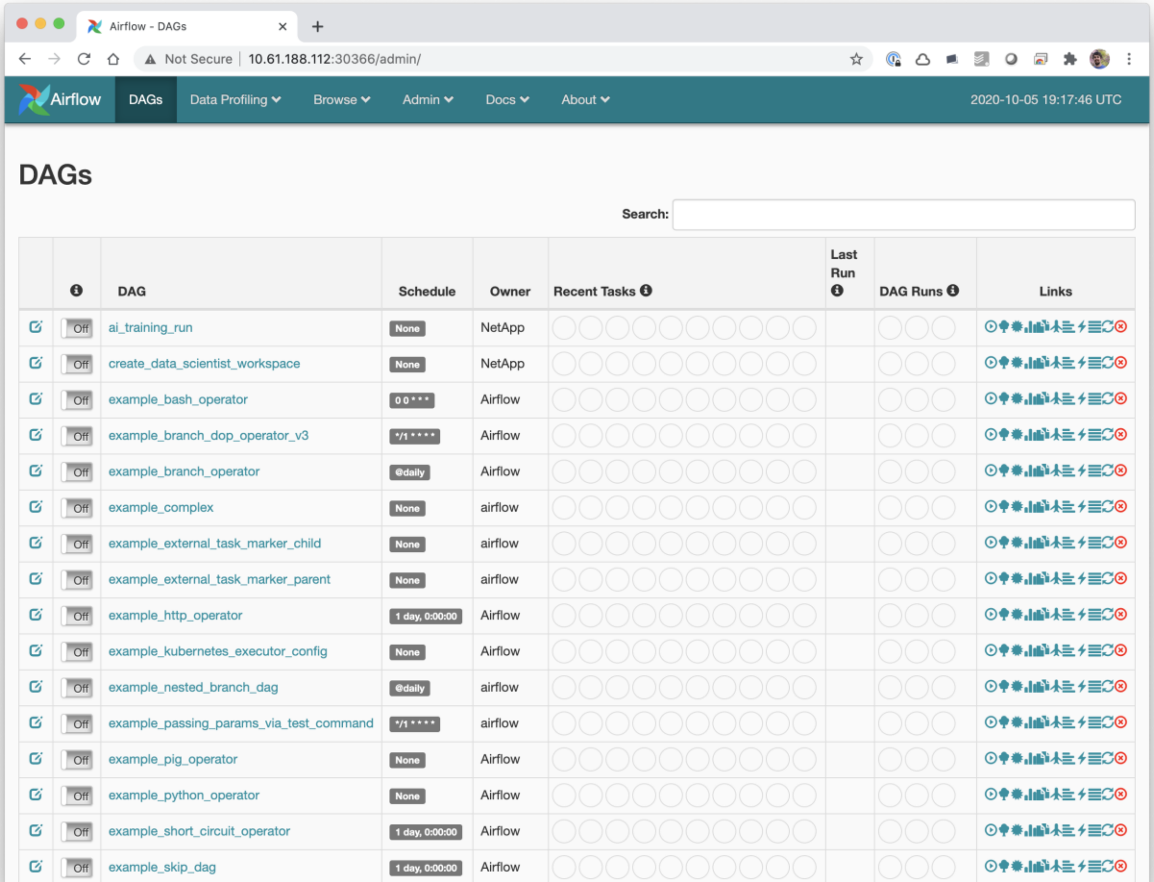
2. Confirm that all Airflow pods are up and running.

```
$ kubectl -n airflow get pod
NAME                                 READY   STATUS    RESTARTS   AGE
airflow-postgresql-0                 1/1     Running   0          38m
airflow-redis-master-0               1/1     Running   0          38m
airflow-scheduler-7fb4bf56cc-g88z4   2/2     Running   2          38m
airflow-web-8f4bdf5fb-hhxr7          2/2     Running   1          38m
airflow-worker-0                     2/2     Running   0          38m
```

3. Obtain the Airflow web service URL by following the instructions that were printed to the console when you deployed Airflow using Helm in step 1.

```
$ export NODE_PORT=$(kubectl get --namespace airflow -o jsonpath="{.spec.ports[0].nodePort}"
services airflow-web)
$ export NODE_IP=$(kubectl get nodes --namespace airflow -o
jsonpath="{.items[0].status.addresses[0].address}")
$ echo http://$NODE_IP:$NODE_PORT/
http://10.61.188.112:30366/
```

4. Confirm that you can access the Airflow web service.



Example Apache Airflow Workflows

This section includes example Apache Airflow DAGs that highlight various NetApp data management capabilities and demonstrate how they can be implemented as part of an Airflow workflow. For more information on DAGs and for detailed instructions regarding how to define and execute them, refer to the official Airflow documentation.

Implement an End-to-End AI Training Workflow with Built-in Traceability and Versioning

The example DAG outlined in this section implements a workflow that takes advantage of NetApp Snapshot technology to integrate rapid and efficient dataset and model versioning and traceability into an end-to-end AI/ML model training workflow.

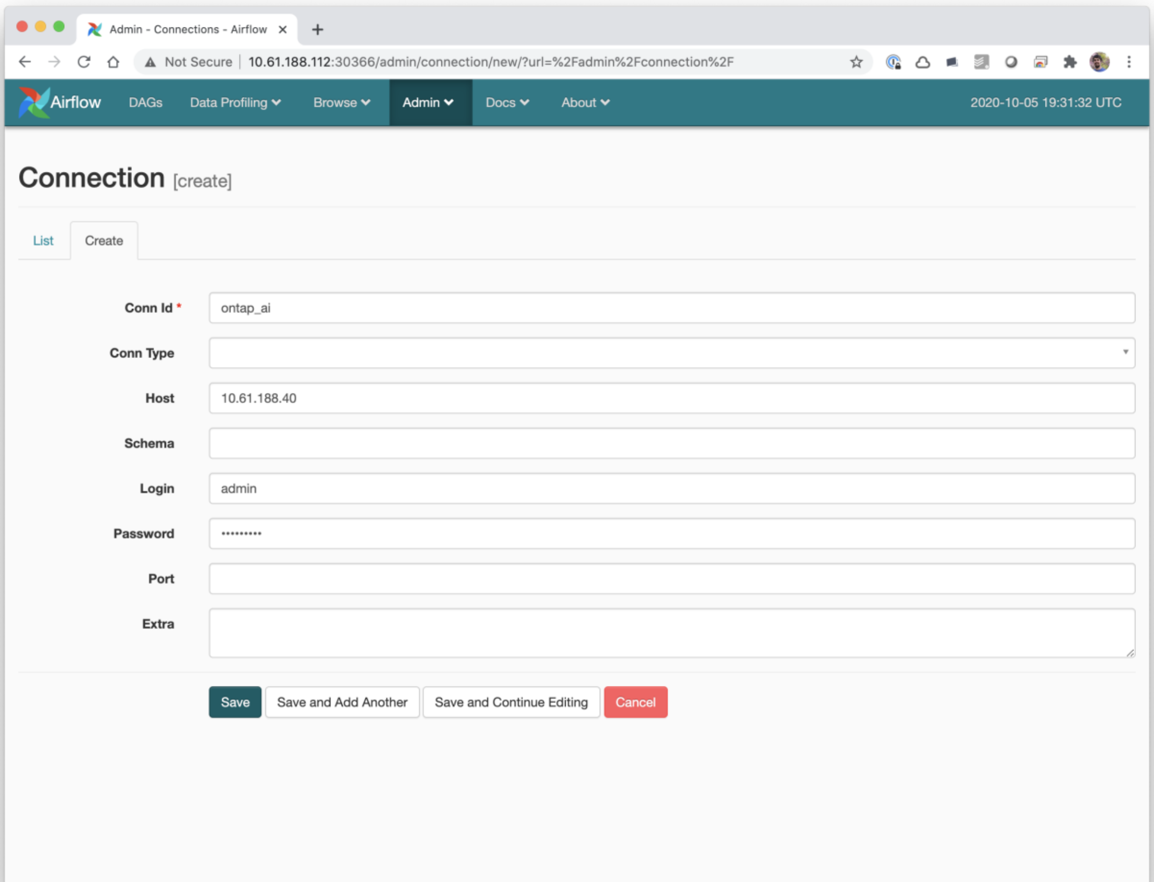
Prerequisites

For this DAG to function correctly, you must complete the following prerequisites:

1. You must have created a connection in Airflow for your ONTAP system.

To manage connections in Airflow, navigate to Admin > Connections in the Airflow web service UI. The example screenshot that follows shows the creation of a connection for a specific ONTAP system. The following values are required:

- **Conn ID.** Unique name for the connection.
- **Host.** The host name or IP address of the ONTAP cluster on which your dataset and model volumes are stored.
- **Login.** Username of the cluster admin account for the ONTAP cluster on which your volumes reside.
- **Password.** Password of the cluster admin account for the ONTAP cluster on which your volumes reside.



2. There must be an existing PersistentVolumeClaim (PVC) in the `airflow` namespace that is tied to the volume that contains the data that you want to use to train your model.
3. There must be an existing PersistentVolumeClaim (PVC) in the `airflow` namespace that is tied to the volume on which you want to store your trained model.

DAG Definition

The Python code excerpt that follows contains the definition for the example DAG. Before executing this example DAG in your environment, you must modify the parameter values in the `DEFINE PARAMETERS` section to match your environment.


```
# Airflow DAG Definition: AI Training Run
#
# Steps:
#   1. Data prep job
#   2. Dataset snapshot (for traceability)
#   3. Training job
#   4. Model snapshot (for versioning/baselining)
#   5. Inference validation job


from airflow.utils.dates import days_ago
from airflow.secrets import get_connections
from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.contrib.operators.kubernetes_pod_operator import KubernetesPodOperator
from airflow.contrib.kubernetes.pod import Resources
from airflow.contrib.kubernetes.volume import Volume
from airflow.contrib.kubernetes.volume_mount import VolumeMount


##### DEFINE PARAMETERS: Modify parameter values in this section to match your environment #####

## Define default args for DAG
ai_training_run_dag_default_args = {
    'owner': 'NetApp'
}

## Define DAG details
ai_training_run_dag = DAG(
    dag_id='ai_training_run',
    default_args=ai_training_run_dag_default_args,
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['training']
)

## Define volume details (change values as necessary to match your environment)

# ONTAP system details
airflowConnectionName = 'ontap_ai'  # Name of the Airflow connection that contains connection
details for your ONTAP system's cluster admin account
verifySSLCert = False   # Denotes whether or not to verify the SSL cert when calling the ONTAP
API

# Dataset volume
dataset_volume_mount = VolumeMount(
    'dataset-volume',
    mount_path='/mnt/dataset',
    sub_path=None,
    read_only=False
)
dataset_volume_config= {
    'persistentVolumeClaim': {
        'claimName': 'dataset-vol'
    }
}
dataset_volume = Volume(name='dataset-volume', configs=dataset_volume_config)
dataset_volume_pv_name = 'pvc-79e0855a-30a1-4f63-b34c-1029b1df49f6'

# Model volume
model_volume_mount = VolumeMount(
    'model-volume',
    mount_path='/mnt/model',
    sub_path=None,
    read_only=False
)
model_volume_config= {
    'persistentVolumeClaim': {
        'claimName': 'airflow-model-vol'
    }
}
```



```
model_volume = Volume(name='model-volume', configs=model_volume_config)
model_volume_pv_name = 'pvc-b3e7cb62-2694-45a3-a56d-9fad6b1262e4'

## Define job details (change values as needed)

# Data prep step
data_prep_step_container_image = "ubuntu:bionic"
data_prep_step_command = ["echo", "'No data prep command entered'"] # Replace this echo command
with the data prep command that you wish to execute
data_prep_step_resources = {} # Hint: To request that 1 GPU be allocated to job pod, change to:
{'limit_gpu': 1}

# Training step
train_step_container_image = "nvcr.io/nvidia/tensorflow:20.07-tf1-py3"
train_step_command = ["echo", "'No training command entered'"] # Replace this echo command with
the training command that you wish to execute
train_step_resources = {} # Hint: To request that 1 GPU be allocated to job pod, change to:
{'limit_gpu': 1}

# Inference validation step
validate_step_container_image = "nvcr.io/nvidia/tensorflow:20.07-tf1-py3"
validate_step_command = ["echo", "'No inference validation command entered'"] # Replace this echo
command with the inference validation command that you wish to execute
validate_step_resources = {} # Hint: To request that 1 GPU be allocated to job pod, change to:
{'limit_gpu': 1}

################################################################################################


# Define function that triggers the creation of a NetApp snapshot
def netappSnapshot(**kwargs) -> str :
    # Parse args
    for key, value in kwargs.items() :
        if key == 'pvName' :
            pvName = value
        elif key == 'verifySSLCert' :
            verifySSLCert = value
        elif key == 'airflowConnectionName' :
            airflowConnectionName = value

    # Install netapp_ontap package
    import sys, subprocess
    result = subprocess.check_output([sys.executable, '-m', 'pip', 'install', '--user', 'netapp-
ontap'])
    print(str(result).replace('\\n', '\n'))

    # Import needed functions/classes
    from netapp_ontap import config as netappConfig
    from netapp_ontap.host_connection import HostConnection as NetAppHostConnection
    from netapp_ontap.resources import Volume, Snapshot
    from datetime import datetime
    import json

    # Retrieve ONTAP cluster admin account details from Airflow connection
    connections = get_connections(conn_id = airflowConnectionName)
    ontapConnection = connections[0]    # Assumes that you only have one connection with the
specified conn_id configured in Airflow
    ontapClusterAdminUsername = ontapConnection.login
    ontapClusterAdminPassword = ontapConnection.password
    ontapClusterMgmtHostname = ontapConnection.host

    # Configure connection to ONTAP cluster/instance
    netappConfig.CONNECTION = NetAppHostConnection(
        host = ontapClusterMgmtHostname,
        username = ontapClusterAdminUsername,
        password = ontapClusterAdminPassword,
        verify = verifySSLCert
    )

    # Convert pv name to ONTAP volume name
    # The following will not work if you specified a custom storagePrefix when creating your
```



```
    #   Trident backend. If you specified a custom storagePrefix, you will need to update this
    #   code to match your prefix.
    volumeName = 'trident_%s' % pvName.replace("-", "_")
    print('\npv name: ', pvName)
    print('ONTAP volume name: ', volumeName)

    # Create snapshot; print API response
    volume = Volume.find(name = volumeName)
    timestamp = datetime.today().strftime("%Y%m%d_%H%M%S")
    snapshot = Snapshot.from_dict({
        'name': 'airflow_%s' % timestamp,
        'comment': 'Snapshot created by a Apache Airflow DAG',
        'volume': volume.to_dict()
    })
    response = snapshot.post()
    print("\nAPI Response:")
    print(response.http_response.text)

    # Retrieve snapshot details
    snapshot.get()

    # Convert snapshot details to JSON string and print
    snapshotDetails = snapshot.to_dict()
    print("\nSnapshot Details:")
    print(json.dumps(snapshotDetails, indent=2))

    # Return name of newly created snapshot
    return snapshotDetails['name']


# Define DAG steps/workflow
with ai_training_run_dag as dag :

    # Define data prep step using Kubernetes Pod operator
(https://airflow.apache.org/docs/stable/kubernetes.html#kubernetespodoperator)
    data_prep = KubernetesPodOperator(
        namespace='airflow',
        image=data_prep_step_container_image,
        cmds=data_prep_step_command,
        resources = data_prep_step_resources,
        volumes=[dataset_volume, model_volume],
        volume_mounts=[dataset_volume_mount, model_volume_mount],
        name="ai-training-run-data-prep",
        task_id="data-prep",
        is_delete_operator_pod=True,
        hostnetwork=False
    )

    # Define step to take a snapshot of the dataset volume for traceability
    dataset_snapshot = PythonOperator(
        task_id='dataset-snapshot',
        python_callable=netappSnapshot,
        op_kwargs={
            'airflowConnectionName': airflowConnectionName,
            'pvName': dataset_volume_pv_name,
            'verifySSLCert': verifySSLCert
        },
        dag=dag
    )

    # State that the dataset snapshot should be created after the data prep job completes
    data_prep >> dataset_snapshot

    # Define training step using Kubernetes Pod operator
(https://airflow.apache.org/docs/stable/kubernetes.html#kubernetespodoperator)
    train = KubernetesPodOperator(
        namespace='airflow',
        image=train_step_container_image,
        cmds=train_step_command,
        resources = train_step_resources,
        volumes=[dataset_volume, model_volume],
```



```
        volume_mounts=[dataset_volume_mount, model_volume_mount],
        name="ai-training-run-train",
        task_id="train",
        is_delete_operator_pod=True,
        hostnetwork=False
    )

    # State that training job should be executed after dataset volume snapshot is taken
    dataset_snapshot >> train

    # Define step to take a snapshot of the model volume for versioning/baselining
    model_snapshot = PythonOperator(
        task_id='model-snapshot',
        python_callable=netappSnapshot,
        op_kwargs={
            'airflowConnectionName': airflowConnectionName,
            'pvName': model_volume_pv_name,
            'verifySSLCert': verifySSLCert
        },
        dag=dag
    )

    # State that the model snapshot should be created after the training job completes
    train >> model_snapshot

    # Define inference validation step using Kubernetes Pod operator
(https://airflow.apache.org/docs/stable/kubernetes.html#kubernetespodoperator)
    validate = KubernetesPodOperator(
        namespace='airflow',
        image=validate_step_container_image,
        cmds=validate_step_command,
        resources = validate_step_resources,
        volumes=[dataset_volume, model_volume],
        volume_mounts=[dataset_volume_mount, model_volume_mount],
        name="ai-training-run-validate",
        task_id="validate",
        is_delete_operator_pod=True,
        hostnetwork=False
    )

    # State that inference validation job should be executed after model volume snapshot is taken
    model_snapshot >> validate
```

Rapidly Clone a Dataset to create a Data Scientist Workspace

The example DAG outlined in this section implements a workflow that takes advantage of NetApp FlexClone technology to clone a dataset volume rapidly and efficiently and create a data scientist or developer workspace.

Prerequisites

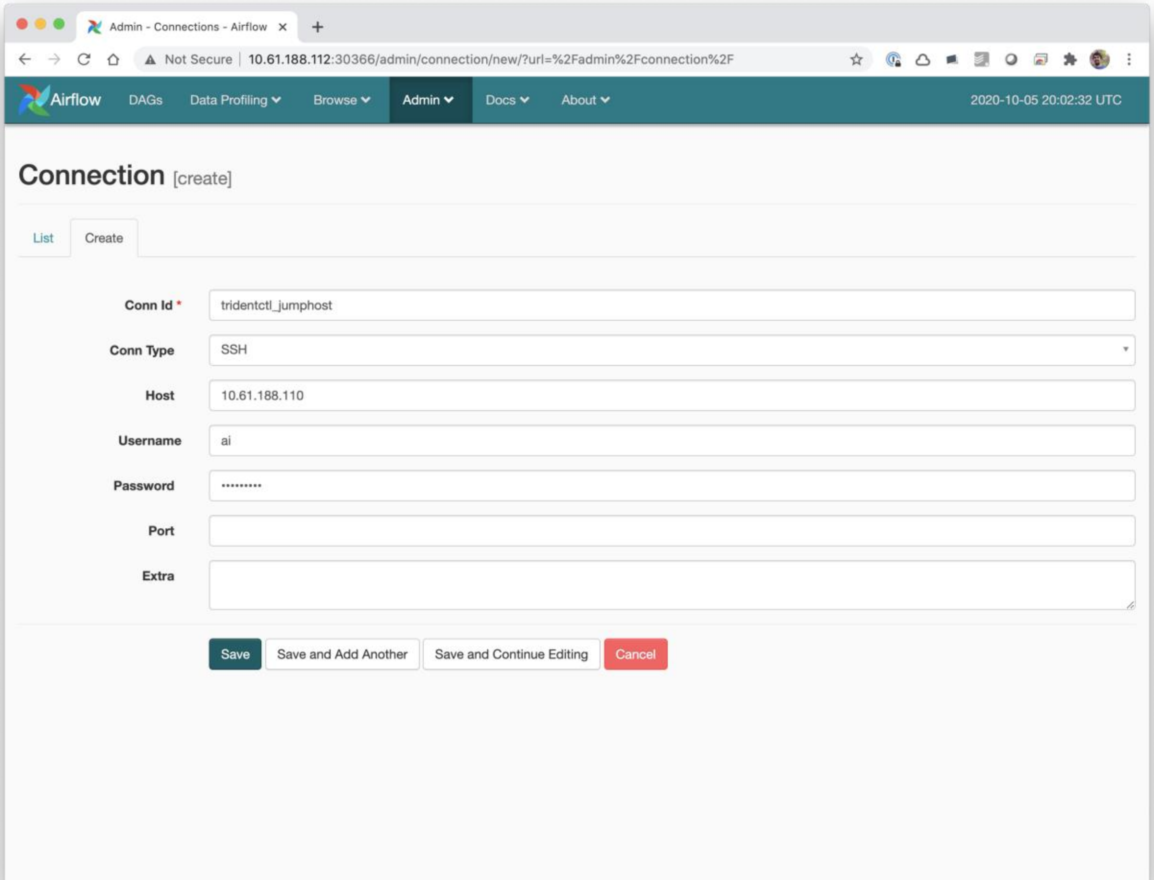
For this DAG to function correctly, you must complete the following prerequisites:

1. You must have created a connection in Airflow for your ONTAP system as outlined in Prerequisite #1 in the section "Implement an End-to-End AI Training Workflow with Built-in Traceability and Versioning."
2. You must have created a connection in Airflow for a host that is accessible via SSH and on which `tridentctl`, the NetApp Trident management utility, is installed and configured to point to your Kubernetes cluster.

   To manage connections in Airflow, navigate to Admin > Connections in the Airflow web service UI. The example screenshot that follows shows the creation of a connection for a specific host on which `tridentctl` is installed and configured. The following values are required:

   - **Conn ID.** Unique name for the connection.
   - **Conn Type.** Must be set to 'SSH'.

- **Host.** The host name or IP address of the host.
- **Login.** Username to use when accessing the host via SSH.
- **Password.** Password to use when accessing the host via SSH.



3. There must be an existing PersistentVolumeClaim (PVC) within your Kubernetes cluster that is tied to the volume that contains the dataset that you wish to clone.

DAG Definition

The Python code excerpt that follows contains the definition for the example DAG. Before executing this example DAG in your environment, you must modify the parameter values in the `DEFINE PARAMETERS` section to match your environment.

```
# Airflow DAG Definition: Create Data Scientist Workspace
#
# Steps:
#   1. Clone source volume
#   2. Import clone into Kubernetes using Trident


from airflow.utils.dates import days_ago
from airflow.secrets import get_connections
from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.contrib.operators.ssh_operator import SSHOperator
```



```
from datetime import datetime


##### DEFINE PARAMETERS: Modify parameter values in this section to match your environment #####

## Define default args for DAG
create_data_scientist_workspace_dag_default_args = {
    'owner': 'NetApp'
}

## Define DAG details
create_data_scientist_workspace_dag = DAG(
    dag_id='create_data_scientist_workspace',
    default_args=create_data_scientist_workspace_dag_default_args,
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['dev-workspace']
)

## Define volume details (change values as necessary to match your environment)

# ONTAP system details
ontapAirflowConnectionName = 'ontap_ai'  # Name of the Airflow connection that contains
connection details for your ONTAP system's cluster admin account
verifySSLCert = False   # Denotes whether or not to verify the SSL cert when calling the ONTAP
API

# Source volume details
sourcePvName = 'pvc-79e0855a-30a1-4f63-b34c-1029b1df49f6' # Name of Kubernetes PV corresponding
to source volume

# Clone volume details (details for the new clone that you will be creating)
timestampForVolumeName = datetime.today().strftime("%Y%m%d_%H%M%S")
cloneVolumeName = 'airflow_clone_%s' % timestampForVolumeName
clonePvcNamespace = 'airflow'   # Kubernetes namespace that you want the new clone volume to be
imported into

## Define tridentctl jumphost details (change values as necessary to match your environment)
tridentctlAirflowConnectionName = 'tridentctl_jumphost' # Name of the Airflow connection of type
'ssh' that contains connection details for a jumphost on which tridentctl is installed

## Define Trident details
tridentStorageClass = 'ontap-flexvol'   # Kubernetes StorageClass that you want to use when
importing the new clone volume
tridentNamespace = 'trident'    # Namespace that Trident is installed in
tridentBackend = 'ontap-flexvol'    # Trident backend that you want to use when importing the new
clone volume

################################################################################################


# Define function that clones a NetApp volume
def netappClone(task_instance, **kwargs) -> str :
    # Parse args
    for key, value in kwargs.items() :
        if key == 'sourcePvName' :
            sourcePvName = value
        elif key == 'verifySSLCert' :
            verifySSLCert = value
        elif key == 'airflowConnectionName' :
            airflowConnectionName = value
        elif key == 'cloneVolumeName' :
            cloneVolumeName = value

    # Install netapp_ontap package
    import sys, subprocess
    result = subprocess.check_output([sys.executable, '-m', 'pip', 'install', '--user', 'netapp-
ontap'])
    print(str(result).replace('\\n', '\n'))

    # Import needed functions/classes
```



```
    from netapp_ontap import config as netappConfig
    from netapp_ontap.host_connection import HostConnection as NetAppHostConnection
    from netapp_ontap.resources import Volume, Snapshot
    from datetime import datetime
    import json

    # Retrieve ONTAP cluster admin account details from Airflow connection
    connections = get_connections(conn_id = airflowConnectionName)
    ontapConnection = connections[0]    # Assumes that you only have one connection with the 
specified conn_id configured in Airflow
    ontapClusterAdminUsername = ontapConnection.login
    ontapClusterAdminPassword = ontapConnection.password
    ontapClusterMgmtHostname = ontapConnection.host

    # Configure connection to ONTAP cluster/instance
    netappConfig.CONNECTION = NetAppHostConnection(
        host = ontapClusterMgmtHostname,
        username = ontapClusterAdminUsername,
        password = ontapClusterAdminPassword,
        verify = verifySSLCert
    )

    # Convert pv name to ONTAP volume name
    # The following will not work if you specified a custom storagePrefix when creating your
    #   Trident backend. If you specified a custom storagePrefix, you will need to update this
    #   code to match your prefix.
    sourceVolumeName = 'trident_%s' % sourcePvName.replace("-", "_")
    print('\nSource pv name: ', sourcePvName)
    print('Source ONTAP volume name: ', sourceVolumeName)

    # Create clone
    sourceVolume = Volume.find(name = sourceVolumeName)
    cloneVolume = Volume.from_dict({
        'name': cloneVolumeName,
        'svm': sourceVolume.to_dict()['svm'],
        'clone': {
            'is_flexclone':'true',
            'parent_volume': sourceVolume.to_dict()
        },
        'nas': {
            'path': '/%s' % cloneVolumeName
        }
    })
    response = cloneVolume.post()
    print("\nAPI Response:")
    print(response.http_response.text)

    # Retrieve clone volume details
    cloneVolume.get()

    # Convert clone volume details to JSON string
    cloneVolumeDetails = cloneVolume.to_dict()
    print("\nClone Volume Details:")
    print(json.dumps(cloneVolumeDetails, indent=2))

    # Create PVC name that resembles volume name and push as XCom for future use
    task_instance.xcom_push(key = 'clone_pvc_name', value = 
cloneVolumeDetails['name'].replace('_', '-'))

    # Return name of new clone volume
    return cloneVolumeDetails['name']


# Define DAG steps/workflow
with create_data_scientist_workspace_dag as dag :

    # Define step to clone source volume
    clone_source = PythonOperator(
        task_id='clone-source',
        provide_context=True,
        python_callable=netappClone,
```



```
        op_kwargs={
            'airflowConnectionName': ontapAirflowConnectionName,
            'sourcePvName': sourcePvName,
            'verifySSLCert': verifySSLCert,
            'cloneVolumeName': cloneVolumeName
        },
        dag=dag
    )

    # Define step to import clone into Kubernetes using Trident
    cloneVolumeName = "{{ task_instance.xcom_pull(task_ids='clone-source', key='return_value')
}}"
    clonePvcName = "{{ task_instance.xcom_pull(task_ids='clone-source', key='clone_pvc_name') }}"
    import_command = '''cat << EOD > import-pvc-%s.yaml && tridentctl -n %s import volume %s %s -
f ./import-pvc-%s.yaml && rm -f import-pvc-%s.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: %s
  namespace: %s
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: %s
EOD''' % (clonePvcName, tridentNamespace, tridentBackend, cloneVolumeName, clonePvcName,
clonePvcName, clonePvcName, clonePvcNamespace, tridentStorageClass)
    import_clone = SSHOperator(
        task_id="import-clone",
        command=import_command,
        ssh_conn_id=tridentctlAirflowConnectionName
    )

    # State that the import step should be executed after the initial clone step completes
    clone_source >> import_clone
```

Trigger a SnapMirror Volume Replication Update

The example DAG outlined in this section implements a workflow that takes advantage of NetApp SnapMirror data replication technology to replicate the contents of a volume between different ONTAP clusters.

This pipeline can be used to replicate data of any type between ONTAP clusters that might or might not be located at different sites or in different regions. Potential use cases include the following:

- Replicating newly acquired sensor data gathered at the edge back to the core data center or to the cloud to be used for AI/ML model training or retraining.
- Replicating a newly trained or newly updated model from the core data center to the edge or to the cloud to be deployed as part of an inferencing application.

Prerequisites

For this DAG to function correctly, you must complete the following prerequisites.

- You must have created a connection in Airflow for your ONTAP system as outlined in Prerequisite #1 in the section “Implement an End-to-End AI Training Workflow with Built-in Traceability and Versioning.”
- You must have already initiated an asynchronous SnapMirror relationship between the source and the destination volume according to standard configuration instructions. For details, refer to official NetApp documentation.

DAG Definition

The Python code excerpt that follows contains the definition for the example DAG. Before executing this example DAG in your environment, you must modify the parameter values in the `DEFINE PARAMETERS` section to match your environment.

```
# Airflow DAG Definition: Replicate Data - SnapMirror
#
# Steps:
#   1. Trigger NetApp SnapMirror update


from airflow.utils.dates import days_ago
from airflow.secrets import get_connections
from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator


##### DEFINE PARAMETERS: Modify parameter values in this section to match your environment #####

## Define default args for DAG
replicate_data_snapmirror_dag_default_args = {
    'owner': 'NetApp'
}

## Define DAG details
replicate_data_snapmirror_dag = DAG(
    dag_id='replicate_data_snapmirror',
    default_args=replicate_data_snapmirror_dag_default_args,
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['data-movement']
)

## Define SnapMirror details (change values as necessary to match your environment)

# Destination ONTAP system details
airflowConnectionName = 'ontap_ai'  # Name of the Airflow connection that contains connection
details for the destination ONTAP system's cluster admin account
verifySSLCert = False   # Denotes whether or not to verify the SSL cert when calling the ONTAP
API

# SnapMirror relationship details (existing SnapMirroer relationship for which you want to
trigger an update)
sourceSvm = "ailab"
sourceVolume = "sm"
destinationSvm = "ai221_data"
destinationVolume = "sm_dest"

################################################################################################


# Define function that triggers a NetApp SnapMirror update
def netappSnapMirrorUpdate(**kwargs) -> int :
    # Parse args
    for key, value in kwargs.items() :
        if key == 'sourceSvm' :
            sourceSvm = value
        elif key == 'sourceVolume' :
            sourceVolume = value
        elif key == 'destinationSvm' :
            destinationSvm = value
        elif key == 'destinationVolume' :
            destinationVolume = value
        elif key == 'verifySSLCert' :
            verifySSLCert = value
        elif key == 'airflowConnectionName' :
            airflowConnectionName = value

    # Install ansible package
```



```
    import sys, subprocess, os
    print("Installing required Python modules:\n")
    result = subprocess.check_output([sys.executable, '-m', 'pip', 'install', '--user',
'ansible', 'netapp-lib'])
    print(str(result).replace('\\n', '\n'))

    # Retrieve ONTAP cluster admin account details from Airflow connection
    connections = get_connections(conn_id = airflowConnectionName)
    ontapConnection = connections[0]    # Assumes that you only have one connection with the
specified conn_id configured in Airflow
    ontapClusterAdminUsername = ontapConnection.login
    ontapClusterAdminPassword = ontapConnection.password
    ontapClusterMgmtHostname = ontapConnection.host

    # Define temporary Ansible playbook for triggering SnapMirror update
    snapMirrorPlaybookContent = """
---
- name: "Trigger SnapMirror Update"
  hosts: localhost
  tasks:
  - name: update snapmirror
    na_ontap_snapmirror:
      state: present
      source_path: '%s:%s'
      destination_path: '%s:%s'
      hostname: '%s'
      username: '%s'
      password: '%s'
      https: 'yes'
      validate_certs: '%s'""" % (sourceSvm, sourceVolume, destinationSvm, destinationVolume,
ontapClusterMgmtHostname,
        ontapClusterAdminUsername, ontapClusterAdminPassword, str(verifySSLCert))
    print("Creating temporary Ansible playbook.\n")
    snapMirrorPlaybookFilepath = "/home/airflow/snapmirror-update.yaml"
    snapMirrorPlaybookFile = open(snapMirrorPlaybookFilepath, "w")
    snapMirrorPlaybookFile.write(snapMirrorPlaybookContent)
    snapMirrorPlaybookFile.close()

    # Trigger SnapMirror update
    print("Executing Ansible playbook to trigger SnapMirror update:\n")
    try :
        result = subprocess.check_output(['ansible-playbook', snapMirrorPlaybookFilepath])
        print(str(result).replace('\\n', '\n'))
    except Exception as e :
        print("Exception:", str(e).strip())
        print("Removing temporary Ansible playbook.")
        os.remove(snapMirrorPlaybookFilepath) # Remove temporary Ansible playbook before exiting
        raise

    # Remove temporary Ansible playbook before exiting
    print("Removing temporary Ansible playbook.\n")
    os.remove(snapMirrorPlaybookFilepath)

    # Return success code
    return 0


# Define DAG steps/workflow
with replicate_data_snapmirror_dag as dag :

    # Define step to trigger a NetApp SnapMirror update
    trigger_snapmirror = PythonOperator(
        task_id='trigger-snapmirror',
        python_callable=netappSnapMirrorUpdate,
        op_kwargs={
            'airflowConnectionName': airflowConnectionName,
            'verifySSLCert': verifySSLCert,
            'sourceSvm': sourceSvm,
            'sourceVolume': sourceVolume,
            'destinationSvm': destinationSvm,
            'destinationVolume': destinationVolume
```



```
        },
        dag=dag
    )
```

Trigger a Cloud Sync Replication Update

The example DAG outlined in this section implements a workflow that takes advantage of NetApp Cloud Sync replication technology to replicate data to and from a variety of file and object storage platforms. Potential use cases include the following:

- Replicating newly acquired sensor data gathered at the edge back to the core data center or to the cloud to be used for AI/ML model training or retraining.
- Replicating a newly trained or newly updated model from the core data center to the edge or to the cloud to be deployed as part of an inferencing application.
- Copying data from an S3 data lake to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Copying data from a Hadoop data lake (through Hadoop NFS Gateway) to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Saving a new version of a trained model to an S3 or Hadoop data lake for permanent storage.
- Copying NFS-accessible data from a legacy or non-NetApp system of record to a high-performance AI/ML training environment for use in the training of an AI/ML model.

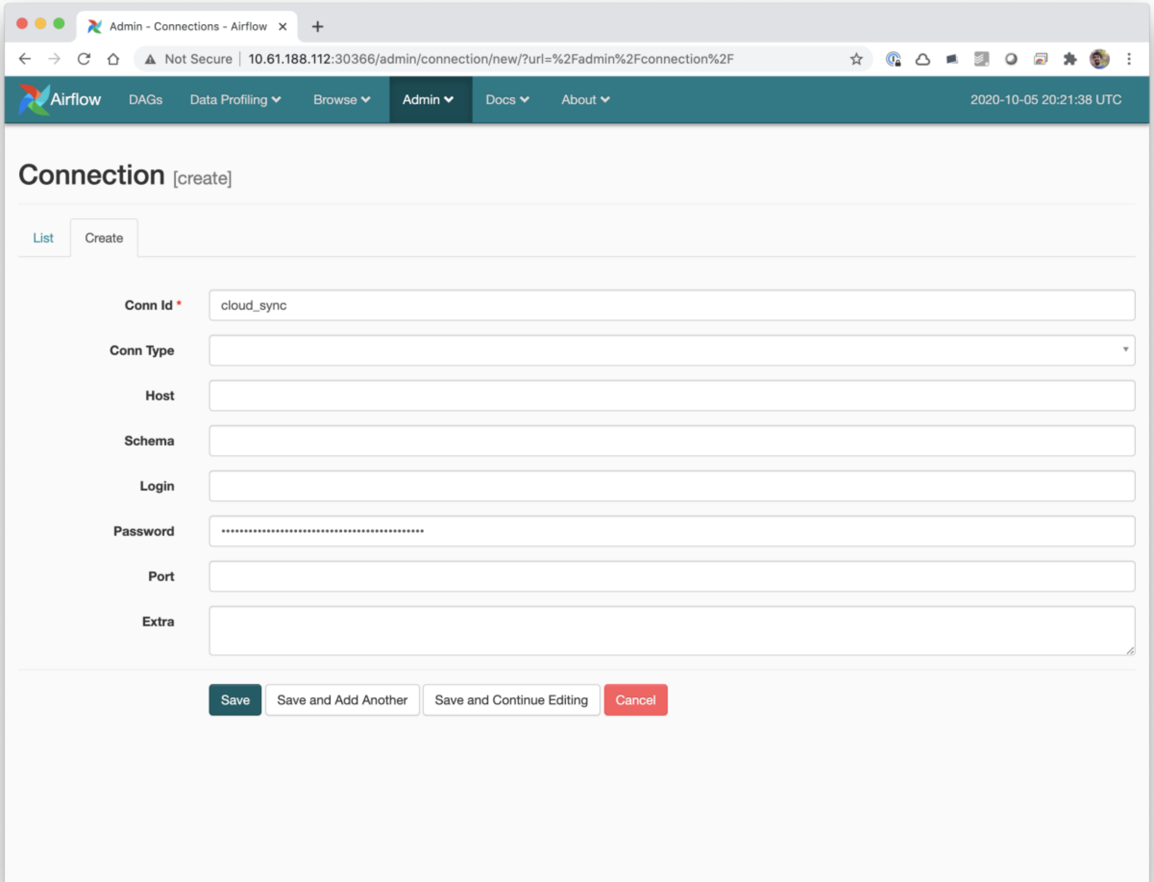
Prerequisites

For this DAG to function correctly, you must complete the following prerequisites.

1. You must have created a connection in Airflow for the NetApp Cloud Sync API.

   To manage connections in Airflow, navigate to Admin > Connections in the Airflow web service UI. The example screenshot that follows shows the creation of a connection for the Cloud Sync API. The following values are required:

   - **Conn ID.** Unique name for the connection.
   - **Password.** Your Cloud Sync API refresh token.



2. You must have already initiated the Cloud Sync relationship that you wish to trigger an update for. To initiate a relationship, visit cloudsync.netapp.com.

DAG Definition

The Python code excerpt that follows contains the definition for the example DAG. Before executing this example DAG in your environment, you must modify the parameter values in the `DEFINE PARAMETERS` section to match your environment.

```
# Airflow DAG Definition: Replicate Data - Cloud Sync
#
# Steps:
#   1. Trigger NetApp Cloud Sync update


from airflow.utils.dates import days_ago
from airflow.secrets import get_connections
from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator


##### DEFINE PARAMETERS: Modify parameter values in this section to match your environment #####

## Define default args for DAG
replicate_data_cloud_sync_dag_default_args = {
    'owner': 'NetApp'
```



```
}

## Define DAG details
replicate_data_cloud_sync_dag = DAG(
    dag_id='replicate_data_cloud_sync',
    default_args=replicate_data_cloud_sync_dag_default_args,
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['data-movement']
)

## Define Cloud Sync details (change values as necessary to match your environment)

# Cloud Sync refresh token details
airflowConnectionName = 'cloud_sync'  # Name of the Airflow connection that contains your Cloud
Sync refresh token

# Cloud Sync relationship details (existing Cloud Sync relationship for which you want to trigger
an update)
relationshipId = '5ed00996ca85650009a83db2'

################################################################################################


## Function for triggering an update for a specific Cloud Sync relationship
def netappCloudSyncUpdate(**kwargs) :
    # Parse args
    printResponse = False # Default value
    keepCheckingUntilComplete = True # Default value
    for key, value in kwargs.items() :
        if key == 'relationshipId' :
            relationshipId = value
        elif key == 'printResponse' :
            printResponse = value
        elif key == 'keepCheckingUntilComplete' :
            keepCheckingUntilComplete = value
        elif key == 'airflowConnectionName' :
            airflowConnectionName = value

    # Install requests module
    import sys, subprocess
    subprocess.run([sys.executable, '-m', 'pip', 'install', 'requests'])

    # Import needed modules
    import requests, json, time


    ## API response error class; objects of this class will be raised when an API resposne is not
as expected
    class APIResponseError(Exception) :
        '''Error that will be raised when an API response is not as expected'''
        pass


    ## Generic function for printing an API response
    def printAPIResponse(response: requests.Response) :
        print("API Response:")
        print("Status Code: ", response.status_code)
        print("Header: ", response.headers)
        if response.text :
            print("Body: ", response.text)


    ## Function for obtaining access token and account ID for calling Cloud Sync API
    def netappCloudSyncAuth(refreshToken: str) :
        ## Step 1: Obtain limited time access token using refresh token

        # Define parameters for API call
        url = "https://netapp-cloud-account.auth0.com/oauth/token"
        headers = {
            "Content-Type": "application/json"
```



```
        }
        data = {
            "grant_type": "refresh_token",
            "refresh_token": refreshToken,
            "client_id": "Mu0V1ywgYteI6w1MbD15fKfVIUrNXGWC"
        }

        # Call API to optain access token
        response = requests.post(url = url, headers = headers, data = json.dumps(data))

        # Parse response to retrieve access token
        try :
            responseBody = json.loads(response.text)
            accessToken = responseBody["access_token"]
        except :
            errorMessage = "Error obtaining access token from Cloud Sync API"
            raise APIResponseError(errorMessage, response)

        ## Step 2: Obtain account ID

        # Define parameters for API call
        url = "https://cloudsync.netapp.com/api/accounts"
        headers = {
            "Content-Type": "application/json",
            "Authorization": "Bearer " + accessToken
        }

        # Call API to obtain account ID
        response = requests.get(url = url, headers = headers)

        # Parse response to retrieve account ID
        try :
            responseBody = json.loads(response.text)
            accountId = responseBody[0]["accountId"]
        except :
            errorMessage = "Error obtaining account ID from Cloud Sync API"
            raise APIResponseError(errorMessage, response)

        # Return access token and account ID
        return accessToken, accountId


    ## Function for monitoring the progress of the latest update for a specific Cloud Sync
relationship
    def netappCloudSyncMonitor(refreshToken: str, relationshipId: str, keepCheckingUntilComplete:
bool = True, printProgress: bool = True, printResponses: bool = False) :
        # Step 1: Obtain access token and account ID for accessing Cloud Sync API
        try :
            accessToken, accountId = netappCloudSyncAuth(refreshToken = refreshToken)
        except APIResponseError as err:
            if printResponse :
                errorMessage = err.args[0]
                response = err.args[1]
                print(errorMessage)
                printAPIResponse(response)
            raise

        # Step 2: Obtain status of the latest update; optionally, keep checking until the latest
update has completed

        while True :
            # Define parameters for API call
            url = "https://cloudsync.netapp.com/api/relationships-v2/%s" % (relationshipId)
            headers = {
                "Accept": "application/json",
                "x-account-id": accountId,
                "Authorization": "Bearer " + accessToken
            }

            # Call API to obtain status of latest update
            response = requests.get(url = url, headers = headers)
```



```
            # Print API response
            if printResponses :
                printAPIResponse(response)

            # Parse response to retrieve status of latest update
            try :
                responseBody = json.loads(response.text)
                latestActivityType = responseBody["activity"]["type"]
                latestActivityStatus = responseBody["activity"]["status"]
            except :
                errorMessage = "Error retrieving status of latest update from Cloud Sync API"
                raise APIResponseError(errorMessage, response)

            # End execution if the latest update is complete
            if latestActivityType == "Sync" and latestActivityStatus == "DONE" :
                if printProgress :
                    print("Success: Cloud Sync update is complete.")
                break

            # Print message re: progress
            if printProgress :
                print("Cloud Sync update is not yet complete.")

            # End execution if calling program doesn't want to monitor until the latest update
has completed
            if not keepCheckingUntilComplete :
                break

            # Sleep for 60 seconds before checking progress again
            print("Checking again in 60 seconds...")
            time.sleep(60)


    # Retrieve Cloud Sync refresh token from Airflow connection
    connections = get_connections(conn_id = airflowConnectionName)
    cloudSyncConnection = connections[0]    # Assumes that you only have one connection with the
specified conn_id configured in Airflow
    refreshToken = cloudSyncConnection.password

    # Step 1: Obtain access token and account ID for accessing Cloud Sync API
    try :
        accessToken, accountId = netappCloudSyncAuth(refreshToken = refreshToken)
    except APIResponseError as err:
        errorMessage = err.args[0]
        response = err.args[1]
        print(errorMessage)
        if printResponse :
            printAPIResponse(response)
        raise

    # Step 2: Trigger Cloud Sync update

    # Define parameters for API call
    url = "https://cloudsync.netapp.com/api/relationships/%s/sync" % (relationshipId)
    headers = {
        "Content-Type": "application/json",
        "Accept": "application/json",
        "x-account-id": accountId,
        "Authorization": "Bearer " + accessToken
    }

    # Call API to trigger update
    print("Triggering Cloud Sync update.")
    response = requests.put(url = url, headers = headers)

    # Check for API response status code of 202; if not 202, raise error
    if response.status_code != 202 :
        errorMessage = "Error calling Cloud Sync API to trigger update."
        if printResponse :
            print(errorMessage)
```



```
            printAPIResponse(response)
        raise APIResponseError(errorMessage, response)

    # Print API response
    if printResponse :
        print("Note: Status Code 202 denotes that update was successfully triggered.")
        printAPIResponse(response)

    print("Checking progress.")
    netappCloudSyncMonitor(refreshToken = refreshToken, relationshipId = relationshipId, 
keepCheckingUntilComplete = keepCheckingUntilComplete, printResponses = printResponse)


# Define DAG steps/workflow
with replicate_data_cloud_sync_dag as dag :

    # Define step to trigger a NetApp Cloud Sync update
    trigger_cloud_sync = PythonOperator(
        task_id='trigger-cloud-sync',
        python_callable=netappCloudSyncUpdate,
        op_kwargs={
            'airflowConnectionName': airflowConnectionName,
            'relationshipId': relationshipId
        },
        dag=dag
    )
```

Trigger an XCP Copy or Sync Operation

The example DAG outlined in this section implements a workflow that invokes NetApp XCP to quickly and reliably replicate data between NFS endpoints. Potential use cases include the following:

- Replicating newly acquired sensor data gathered at the edge back to the core data center or to the cloud to be used for AI/ML model training or retraining.
- Replicating a newly trained or newly updated model from the core data center to the edge or to the cloud to be deployed as part of an inferencing application.
- Copying data from a Hadoop data lake (through Hadoop NFS Gateway) to a high-performance AI/ML training environment for use in the training of an AI/ML model.
- Copying NFS-accessible data from a legacy or non-NetApp system of record to a high-performance AI/ML training environment for use in the training of an AI/ML model.

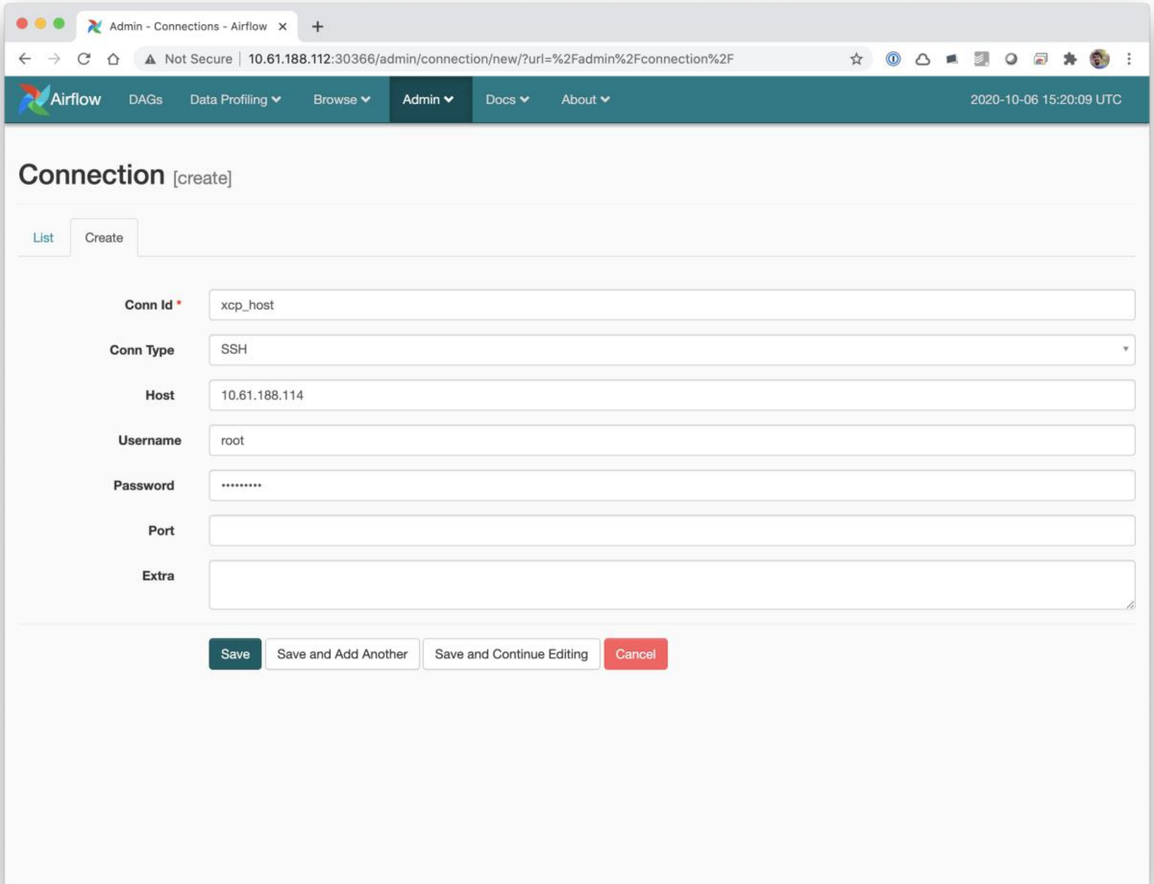
Prerequisites

For this DAG to function correctly, you must complete the following prerequisites.

1. You must have created a connection in Airflow for a host that is accessible via SSH and on which NetApp XCP is installed and configured. For details regarding how to install and configure NetApp XCP, refer to the NetApp XCP homepage and the official NetApp XCP documentation.

   To manage connections in Airflow, navigate to Admin > Connections in the Airflow web service UI. The example screenshot that follows shows the creation of a connection for a specific host on which NetApp XCP is installed and configured. The following values are required:

   – **Conn ID.** Unique name for the connection.
   – **Conn Type.** Must be set to SSH.
   – **Host.** The host name or IP address of the host.
   – **Login.** Username to use when accessing the host via SSH.
   – **Password.** Password to use when accessing the host via SSH.



DAG Definition

The Python code excerpt that follows contains the definition for the example DAG. Before executing this example DAG in your environment, you must modify the parameter values in the `DEFINE PARAMETERS` section to match your environment.

```
# Airflow DAG Definition: Replicate Data - XCP
#
# Steps:
#   1. Invoke NetApp XCP copy or sync operation


from airflow.utils.dates import days_ago
from airflow.secrets import get_connections
from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.contrib.operators.ssh_operator import SSHOperator
from datetime import datetime


##### DEFINE PARAMETERS: Modify parameter values in this section to match your environment #####

## Define default args for DAG
replicate_data_xcp_dag_default_args = {
    'owner': 'NetApp'
}
```



```
## Define DAG details
replicate_data_xcp_dag = DAG(
    dag_id='replicate_data_xcp',
    default_args=replicate_data_xcp_dag_default_args,
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['data-movement']
)

## Define xcp operation details (change values as necessary to match your environment and desired 
operation)

# Define xcp operation to perform
xcpOperation = 'sync' # Must be 'copy' or 'sync'

# Define source and destination for copy operation
xcpCopySource = '192.168.200.41:/trident_pvc_957318e1_9b73_4e16_b857_dca7819dd263'
xcpCopyDestination = '192.168.200.41:/trident_pvc_9e7607c2_29c8_4dbf_9b08_551ba72d0273'

# Define catalog id for sync operation
xcpSyncId = 'autoname_copy_2020-10-06_16.37.44.963391'

## Define xcp host details (change values as necessary to match your environment)
xcpAirflowConnectionName = 'xcp_host' # Name of the Airflow connection of type 'ssh' that 
contains connection details for a host on which xcp is installed, configured, and accessible 
within $PATH

########################################################################################


# Construct xcp command
xcpCommand = 'xcp help'
if xcpOperation == 'copy' :
    xcpCommand = 'xcp copy ' + xcpCopySource + ' ' + xcpCopyDestination
elif xcpOperation == 'sync' :
    xcpCommand = 'xcp sync -id ' + xcpSyncId


# Define DAG steps/workflow
with replicate_data_xcp_dag as dag :

    # Define step to invoke a NetApp XCP copy or sync operation
    invoke_xcp = SSHOperator(
        task_id="invoke-xcp",
        command=xcpCommand,
        ssh_conn_id=xcpAirflowConnectionName
    )
```

Example Basic Trident Operations

This section includes examples of various operations that you may want to perform on your Kubernetes cluster.

Import an Existing Volume

If there are existing volumes on your NetApp storage system/platform that you want to mount on containers within your Kubernetes cluster, but that are not tied to PVCs in the cluster, then you must import these volumes. You can use the Trident volume import functionality to import these volumes.

The example commands that follow show the importing of the same volume, named `pb_fg_all`, twice, once for each Trident backend that was created in the example in the section “Example Trident Backends for ONTAP AI Deployments”, step 1. Importing the same volume twice in this manner enables you to mount the volume (an existing FlexGroup volume) multiple times across different LIFs, as described in the section “Example Trident Backends for ONTAP AI Deployments,” step 1. For more information about

PVCs, 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 volumes at the same time and that access will be read-only. For more information about the `accessMode` field, see the [official Kubernetes documentation]().

Note: The backend names that are specified in the following example import commands are highlighted for reference. These names correspond to the backends that were created in the example in the section "Example Trident Backends for ONTAP AI Deployments," step 1.

Note: The StorageClass names that are specified in the following example PVC definition files are highlighted for reference. These names correspond to the StorageClasses that were created in the example in the section "Example Kubernetes StorageClasses for ONTAP AI Deployments," 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
```

Provision a New Volume

You can use Trident to provision a new volume on your NetApp storage system or platform. The following example commands show the provisioning of a new FlexVol volume. In this example, the volume is provisioned using the StorageClass that was created in the example in the section "Example Kubernetes StorageClasses for ONTAP AI Deployments," step 2.

Note: An `accessModes` value of `ReadWriteMany` is specified in the following example PVC definition file. This value means that multiple containers can mount this PVC at the same time and that access is read-write. For more information about the `accessMode` field, see the official Kubernetes documentation.

```
$ 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
```

Example High-performance Jobs for ONTAP AI Deployments

This section includes examples of various high-performance jobs that can be executed when the NetApp AI Control Plane solution is deployed on an ONTAP AI pod.

Execute a Single-Node AI Workload

To execute a single-node AI and ML job in your Kubernetes cluster, perform the following tasks from the deployment jump host. With Trident, you can quickly and easily make a data volume, potentially containing petabytes of data, accessible 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 was created in the example in the section “Import an Existing Volume,” in the pod definition. This step is a Kubernetes-native operation; no NetApp 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. The following example commands show the creation of a Kubernetes job for a TensorFlow benchmark workload 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. 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.

   Additionally, to provide the required amount of storage bandwidth, the volume that contains the needed training data (the volume that was imported in the example in the section “Import an Existing Volume”) is mounted twice within the pod that this job creates. See the highlighted lines in the following job definition. See the section “Example Trident Backends for ONTAP AI Deployments”, step 1, for details about why you might want 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.

   The volume that was created in the example in the section “Provision a New Volume” is also mounted in the pod. These volumes are referenced in the job definition by using the names of the PVCs. For more information about Kubernetes jobs, see the official Kubernetes documentation.

   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 runtime 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 definition 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 definition, 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.

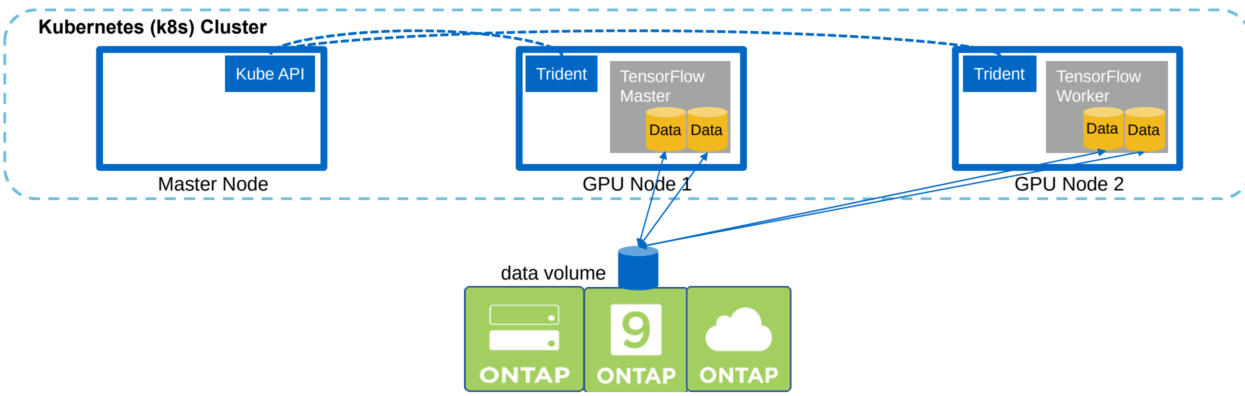

```
$ kubectl get jobs
NAME                                          COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet             1/1           5m42s      10m
$ kubectl get pods
NAME                                                READY   STATUS      RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92             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.
```

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 a NetApp volume and to use more GPUs than a single worker node can provide. See Figure 9 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 9) Synchronous distributed AI job.



1. 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 the example in the section “Execute a Single-Node AI Workload.” 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](#).

```
$ 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 definition, 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 the example in the section “Execute a Single-Node AI Workload.”

   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 definition 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 definition, 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 1 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.
```

Performance Testing

We performed a simple performance comparison as part of the creation of this solution. 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. Table 3 lists 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

Conclusion

Companies and organizations of all sizes and across all industries are turning to artificial intelligence (AI), machine learning (ML), and deep learning (DL) to solve real-world problems, deliver innovative products and services, and to get an edge in an increasingly competitive marketplace. As organizations increase their use of AI, ML, and DL, they face many challenges, including workload scalability and data availability. These challenges can be addressed through the use of the NetApp AI Control Plane, NetApp’s full stack AI data and experiment management solution.

This solution enables you to rapidly clone a data namespace just as you would a Git repo. Additionally, it allows you to define and implement AI, ML, and DL training workflows that incorporate the near-instant creation of data and model baselines for traceability and versioning. With this solution, you can trace every single model training run back to the exact dataset(s) that the model was trained and/or validated with. Lastly, this solution enables you to swiftly provision Jupyter Notebook workspaces with access to massive datasets.

Because this solution is targeted towards data scientists and data engineers, no NetApp or NetApp ONTAP expertise is required. With this solution, data management functions can be executed using simple and familiar tools and interfaces. Furthermore, this solution utilizes fully open-source and free components. Therefore, if you already have NetApp storage in your environment, you can implement this solution today. If you want to test drive this solution but you do not have already have NetApp storage, visit cloud.netapp.com, and you can be up and running with a cloud-based NetApp storage solution in no time.

Acknowledgments

- David Arnette, Technical Marketing Engineer, NetApp
- Sung-Han Lin, Performance Analyst, NetApp
- Steve Guhr, Solutions Engineer, NetApp
- Muneer Ahmad, Solutions Architect, NetApp
- Santosh Rao, Senior Technical Director, NetApp
- Bala Ramesh, Technical Marketing Engineer, NetApp
- George Tehrani, Product Manager, 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 triggering Snapshot copies/FlexClone volumes using kubectl commands (removed from document in version 3.0); added section on Kubeflow ("NVIDIA DeepOps" and "Kubeflow."*); added Figure 9; and updated DeepOps troubleshooting instructions.
Version 3.0	March 2020	Added section on creating a Snapshot from within a Jupyter Notebook ("Create a Snapshot of an ONTAP Volume from Within a Jupyter Notebook"); added example Kubeflow pipelines ("Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and

Version	Date	Document Version History
		Versioning” and “Create a Kubeflow Pipeline to Rapidly Clone a Dataset for a Data Scientist Workspace”); added NetApp Snapshot copies and NetApp FlexClone technology descriptions to the “Concepts and Components” section; and reordered sections within document; and removed sections on triggering Snapshot copies/FlexClone volumes using kubectl commands (due to Kubernetes API changes).
Version 4.0	May 2020	Added example Kubeflow pipeline (“Create a Kubeflow Pipeline to Trigger a SnapMirror Volume Replication Update”); added NetApp SnapMirror technology description (“NetApp SnapMirror Data Replication Technology”); and updated Abstract and Introduction.
Version 5.0	June 2020	Added example Jupyter Notebook (“Trigger a Cloud Sync Replication Update from Within a Jupyter Notebook”); added example Kubeflow pipeline (“Create a Kubeflow Pipeline to Trigger a Cloud Sync Replication Update”); updated example Kubeflow pipeline to use Trident-based annotation cloning method (“Create a Kubeflow Pipeline to Rapidly Clone a Dataset for a Data Scientist Workspace”); added NetApp Cloud Sync technology description (“NetApp Cloud Sync”); added DeepOps option for deploying Trident (“Install Trident”); fixed formatting error in the section “Create a Kubeflow Pipeline to Trigger a SnapMirror Volume Replication Update;” and removed all references to NKS.
Version 6.0	October 2020	Added Apache Airflow sections (sections “Apache Airflow,” “Apache Airflow Deployment,” and “Example Apache Airflow Workflows”); added references to Git repo containing example Kubeflow pipelines and Jupyter Notebooks (“Example Kubeflow Operations and Tasks”); added NetApp XCP to “Concepts and Components;” reworded introduction.

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.

TR-4798-0720

NetApp