NetApp AI Control Plane

NetApp Solutions

NetApp
April 26, 2024

Table of Contents

NetApp AI Control Plane

TR-4798: NetApp AI Control Plane

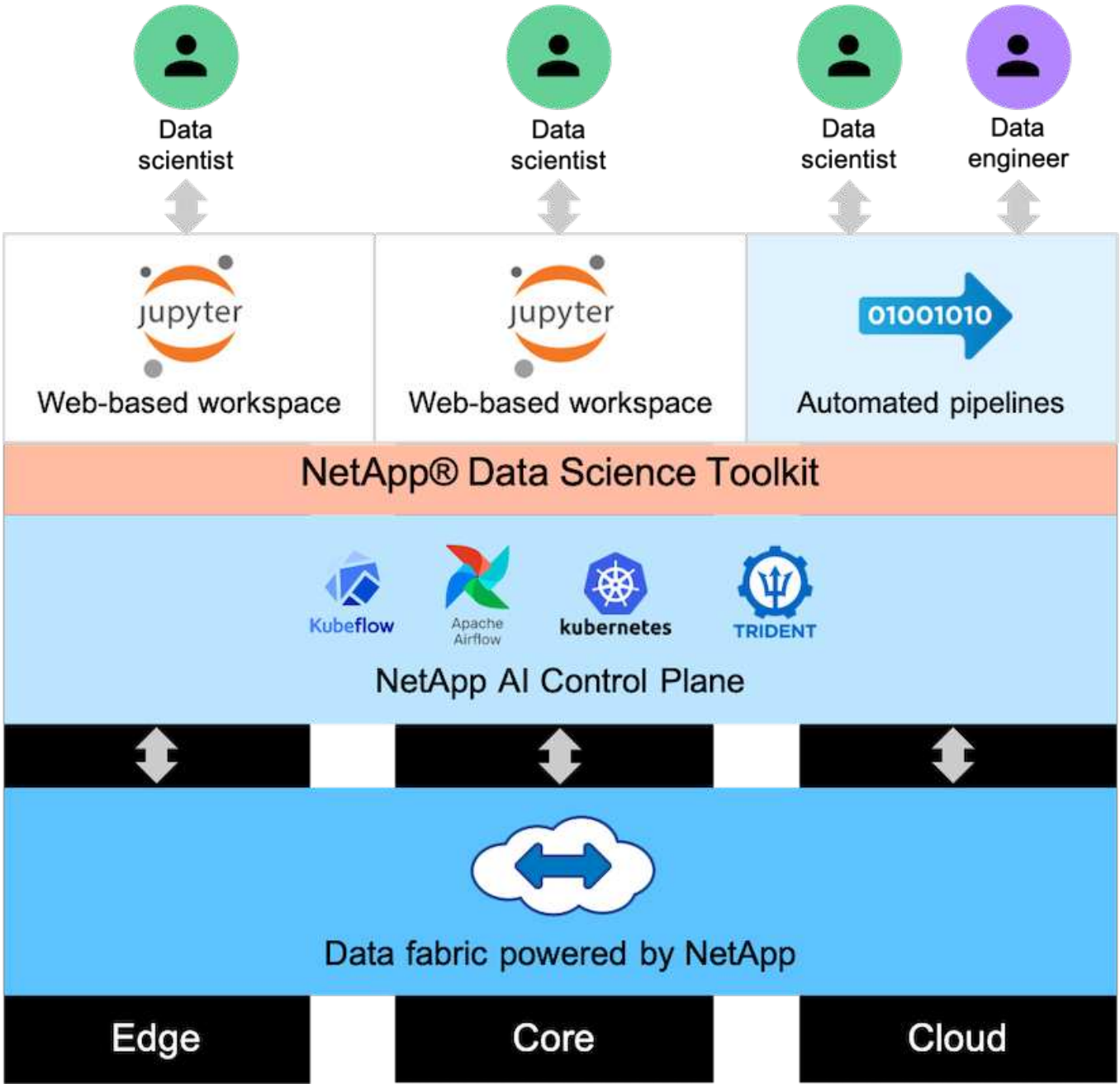
Mike Oglesby, NetApp

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. 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.

Note: For HPC style distributed training at scale involving a large number of GPU servers that require shared access to the same dataset, or if you require/prefer a parallel file system, check out TR-4890. This technical report describes how to include NetApp's fully supported parallel file system solution BeeGFS as part of the NetApp AI Control Plane. This solution is designed to scale from a handful of NVIDIA DGX A100 systems, up to a full blown 140 node SuperPOD.

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. The following figure provides a visualization of the solution.



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

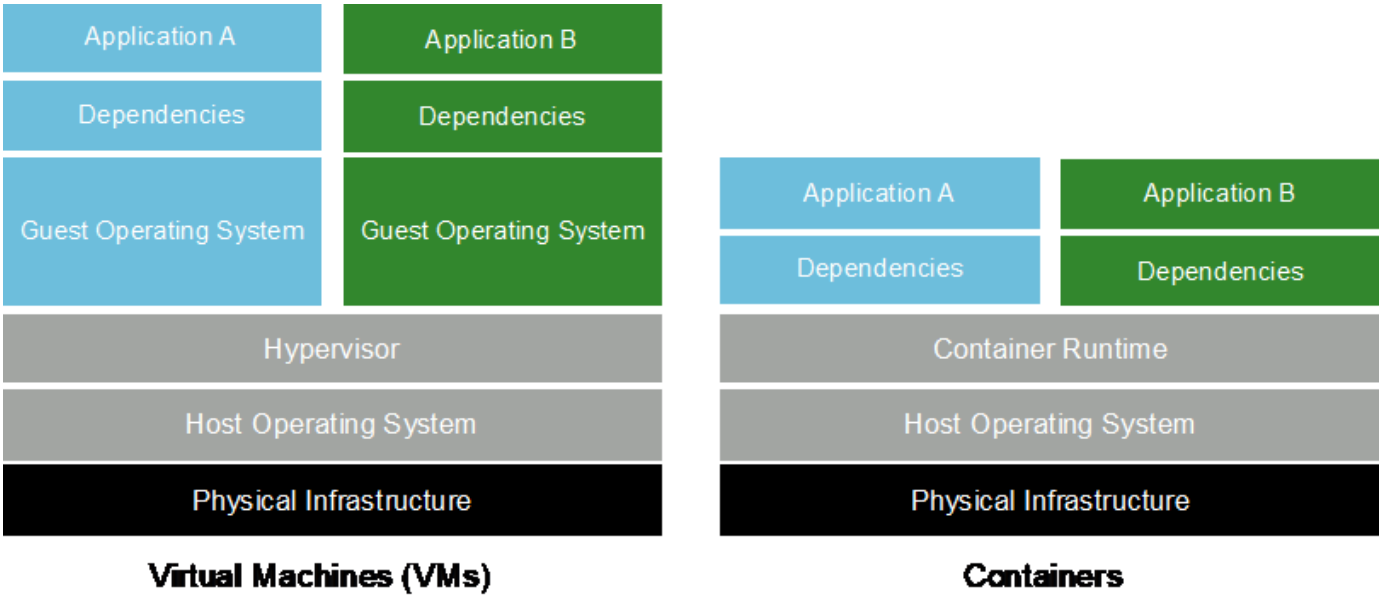
Modern AI training and inference workloads require massively parallel computing capabilities. Therefore, GPUs

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. The following figure depicts a visualization of virtual machines versus containers.

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.



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

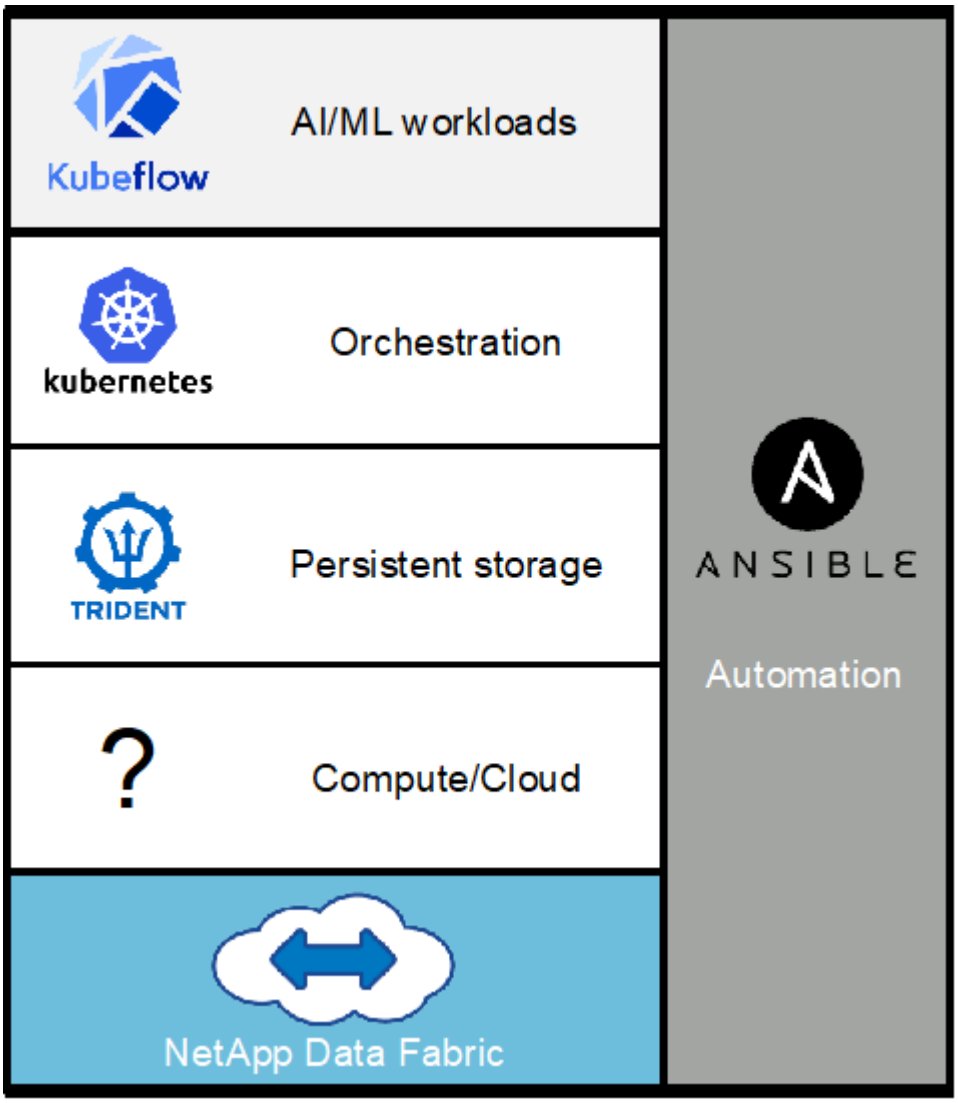
simple to consume. For more information, visit the Trident website.

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 the following figure 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.

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:

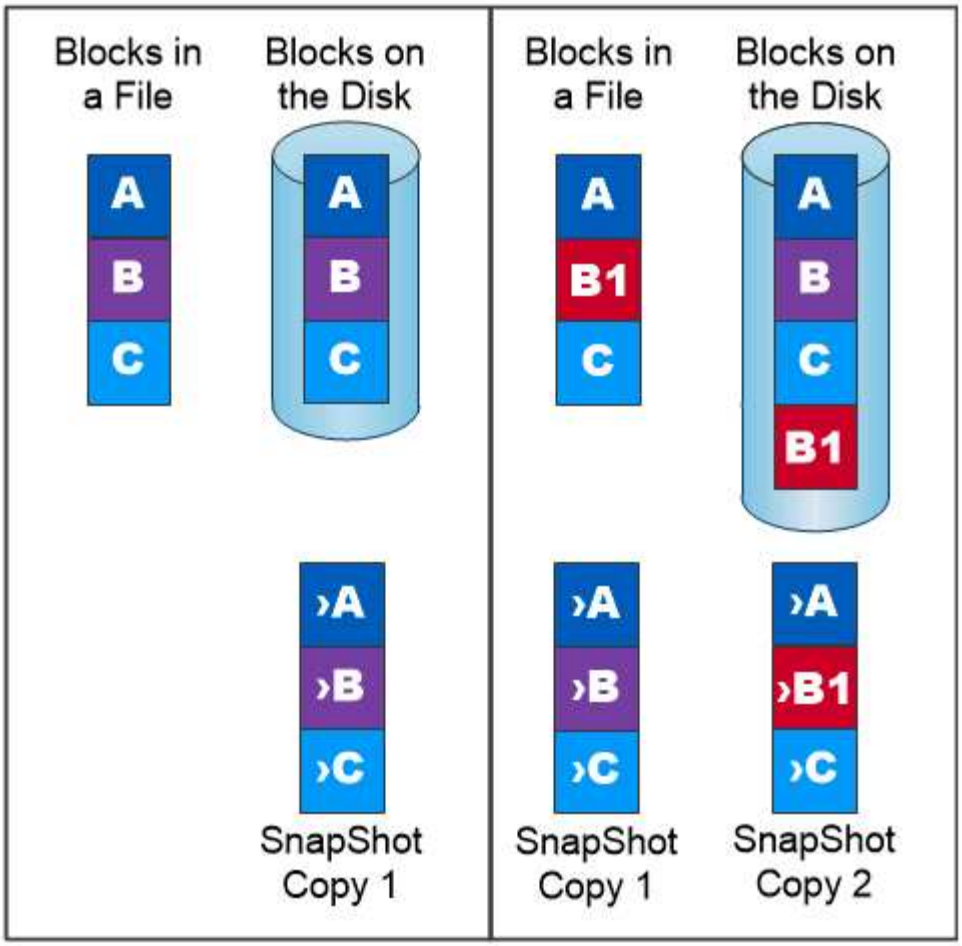
- **Seamless scaling and nondisruptive operations.** ONTAP supports the nondisruptive addition of capacity to existing controllers and to scale-out clusters. You can upgrade to the latest technologies, such as NVMe and 32Gb FC, without costly data migrations or outages.
- **Cloud connection.** ONTAP 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, as depicted in the following figure.

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.

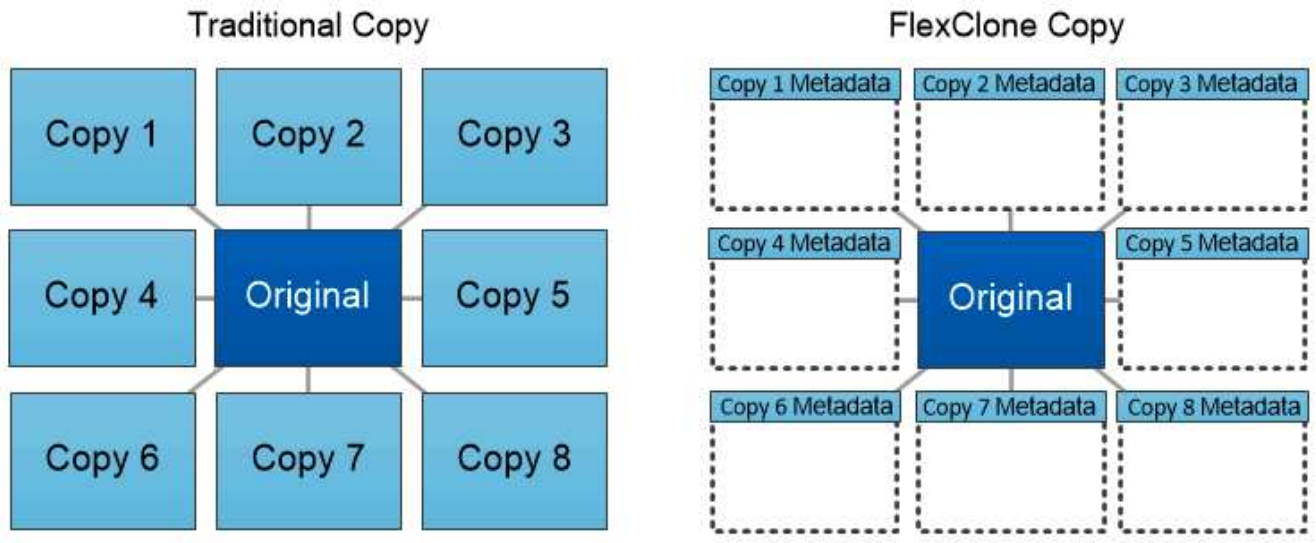
You can use a Snapshot copy to recover individual files or LUNs or to restore the entire contents of a volume. ONTAP compares pointer information in the Snapshot copy with data on disk to reconstruct the missing or damaged object, without downtime or a significant performance cost.



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, as depicted in the following figure. 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).

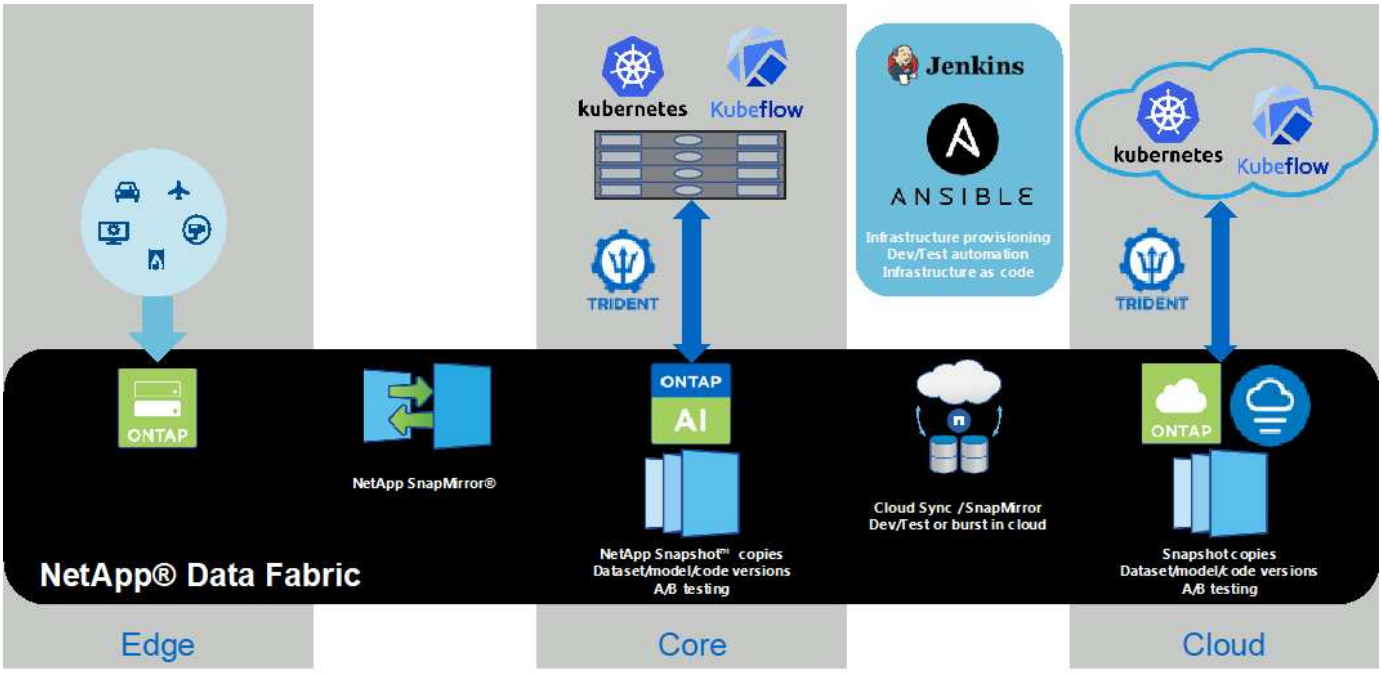


FlexClone copies share data blocks with their parents, consuming no storage except what is required for metadata.

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 the following figure 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%.



NetApp BlueXP Copy and Sync

BlueXP Copy and 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, BlueXP Copy and 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. BlueXP Copy and Sync can sync data on-demand when an update is triggered or continuously sync data based on a predefined schedule. Regardless, BlueXP Copy and Sync only moves the deltas, so time and money spent on data replication is minimized.

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

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.

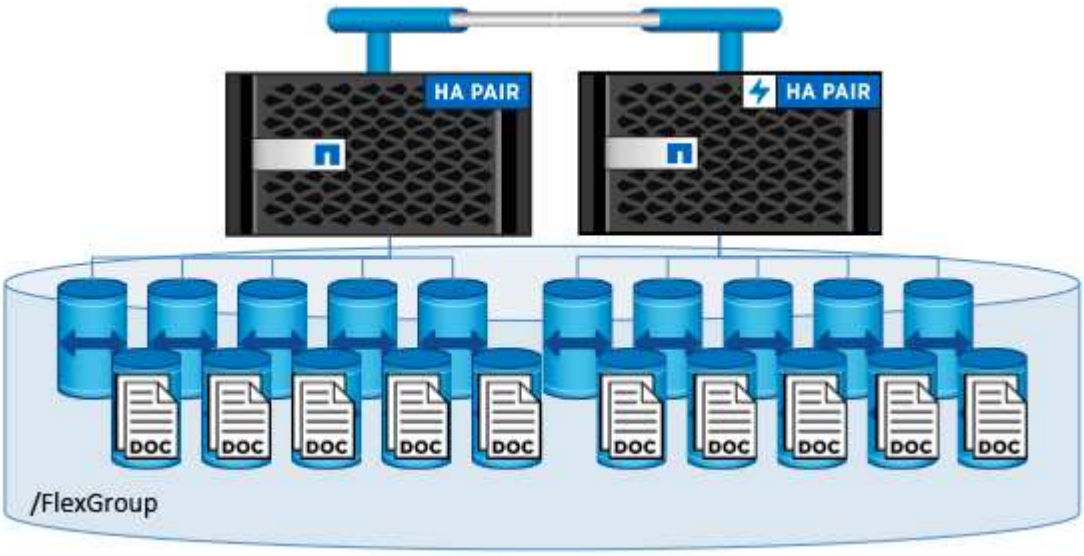
NetApp ONTAP FlexGroup Volumes

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 is a single namespace that comprises multiple constituent member volumes, as shown in the following figure. 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.



Hardware and Software Requirements

The NetApp AI Control Plane solution is not dependent on this specific hardware. The solution is compatible with any NetApp physical storage appliance, software-defined instance, or cloud service, that is supported by Trident. 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. See the following tables for details on the environment that was used to validate the solution.

Infrastructure Component	Quantity	Details	Operating System
Deployment jump host	1	VM	Ubuntu 20.04.2 LTS

Infrastructure Component	Quantity	Details	Operating System
Kubernetes master nodes	1	VM	Ubuntu 20.04.2 LTS
Kubernetes worker nodes	2	VM	Ubuntu 20.04.2 LTS
Kubernetes GPU 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 A220	NetApp ONTAP 9.7 P6

Software Component	Version
Apache Airflow	2.0.1
Apache Airflow Helm Chart	8.0.8
Docker	19.03.12
Kubeflow	1.2
Kubernetes	1.18.9
NetApp Trident	21.01.2
NVIDIA DeepOps	Trident deployment functionality from master branch as of commit 61898cdfda; All other functionality from version 21.03

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.

NetApp Trident Deployment and Configuration

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 is supported by Trident.

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:
    - 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.
    - 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 how to configure Backends and StorageClasses see the linked subsections at NetApp Docs.

If you are deploying the NetApp AI Control Plane solution on an ONTAP AI pod, see Example Trident Backends for ONTAP AI Deployments for some examples of different Trident Backends that you might want to create and 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. This will allow you to balance volume mounts across LIFs

   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. For more information about StorageClasses, see the Trident documentation.

   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.

   In the following example, a particular Backend is not specified in the StorageClass definition file because only one FlexVol-enabled Trident backend was created. 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 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.

> The `ontap-nas-flexgroup` Trident Backend type has a minimum PVC size that is fairly large. By default, Kubeflow attempts to provision PVCs that are only a few GBs in size. 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.

> 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 -x
…
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. It may take several minutes for all pods to start.

```
$ 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
```

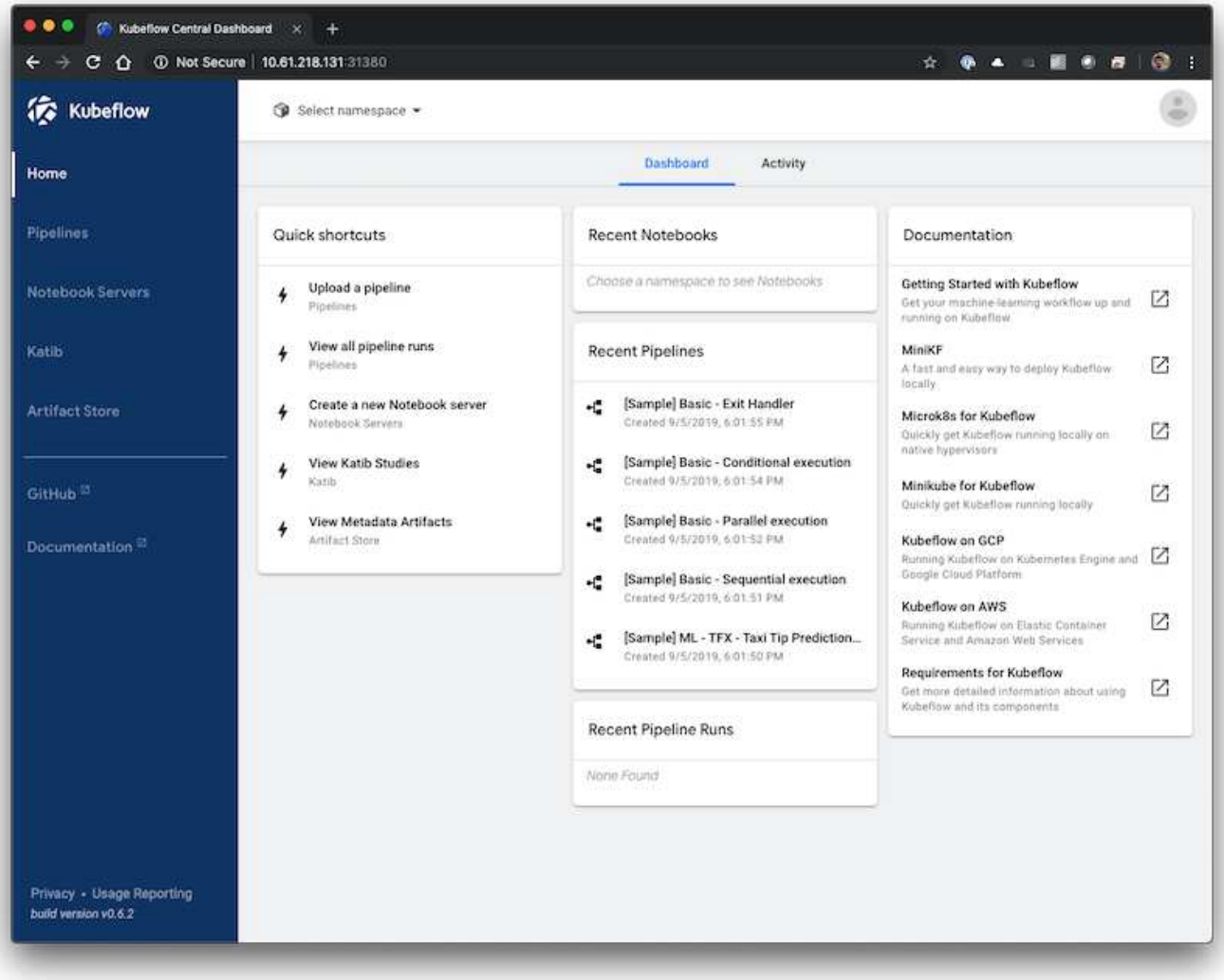
```
deployment.apps/tensorboard                                 1/1     1
1           94s
deployment.apps/tf-job-dashboard                            1/1     1
1           94s
deployment.apps/tf-job-operator                             1/1     1
1           94s
deployment.apps/workflow-controller                         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.

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.

Example Kubeflow Operations and Tasks

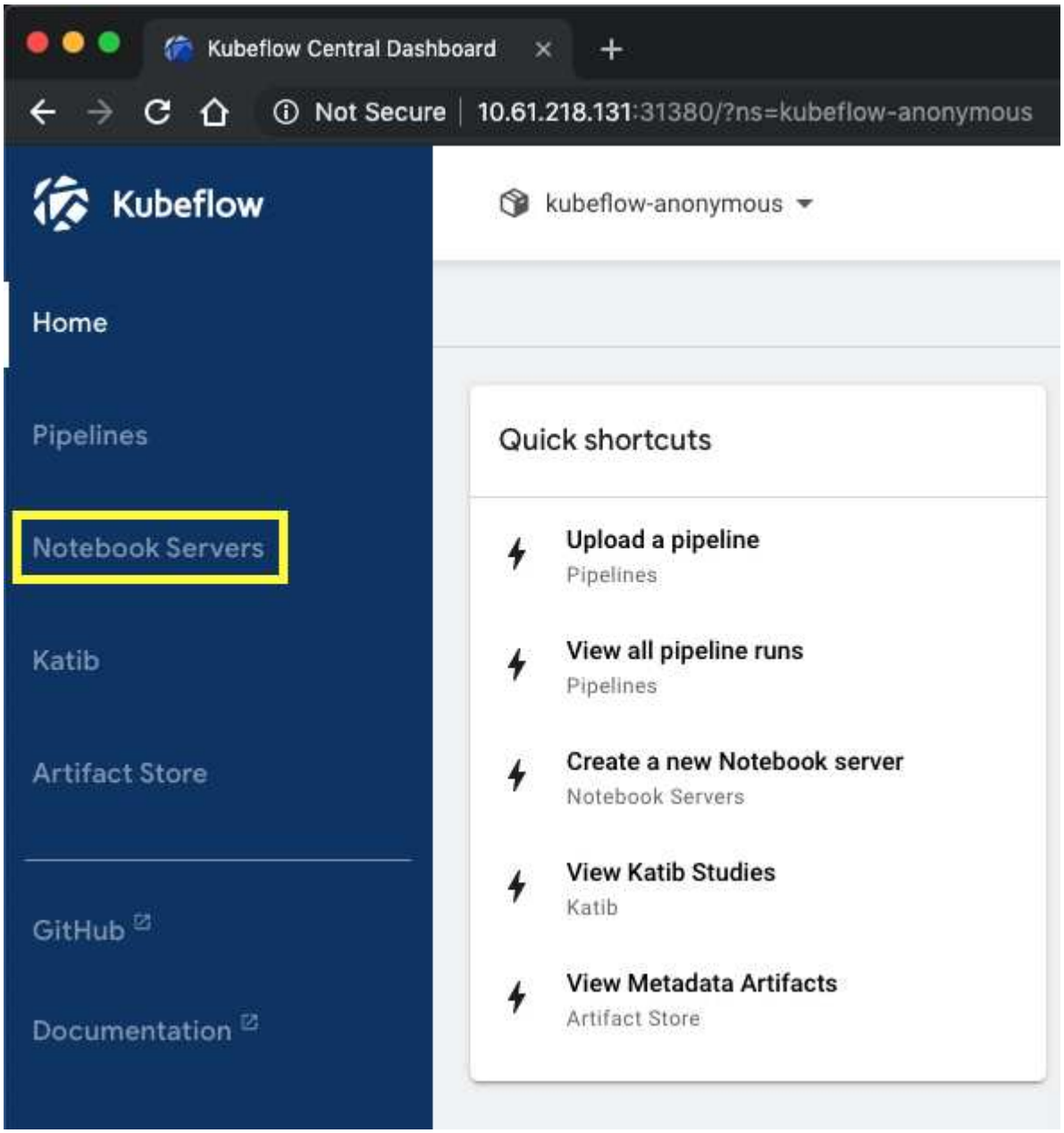
This section includes examples of various operations and tasks that you may want to perform using Kubeflow.

Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use

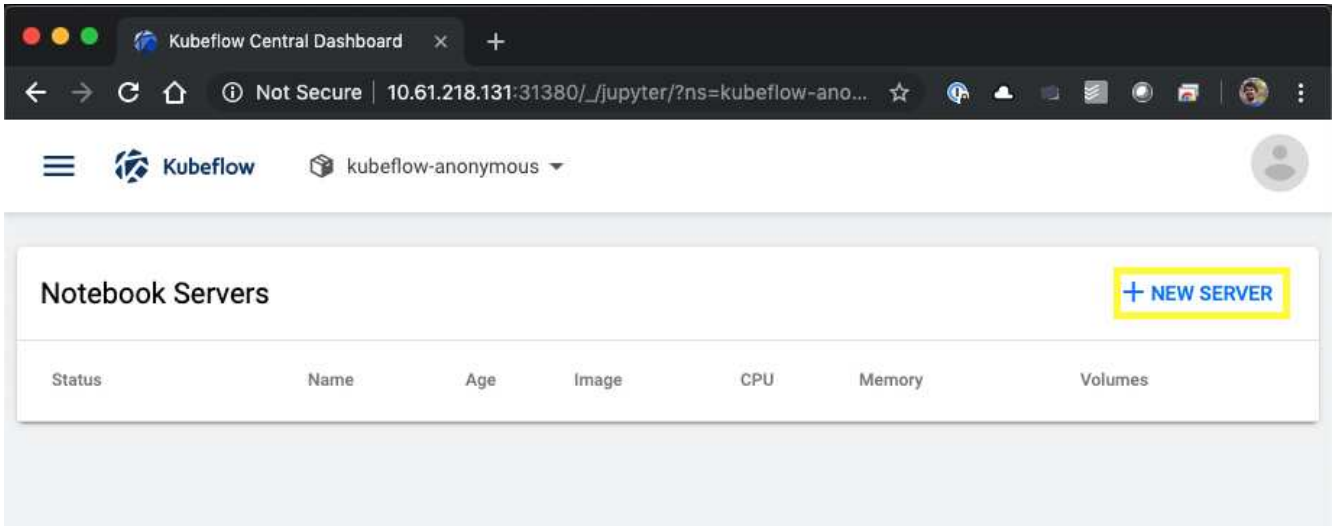
Kubeflow is capable of rapidly provisioning new Jupyter Notebook servers to act as data

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

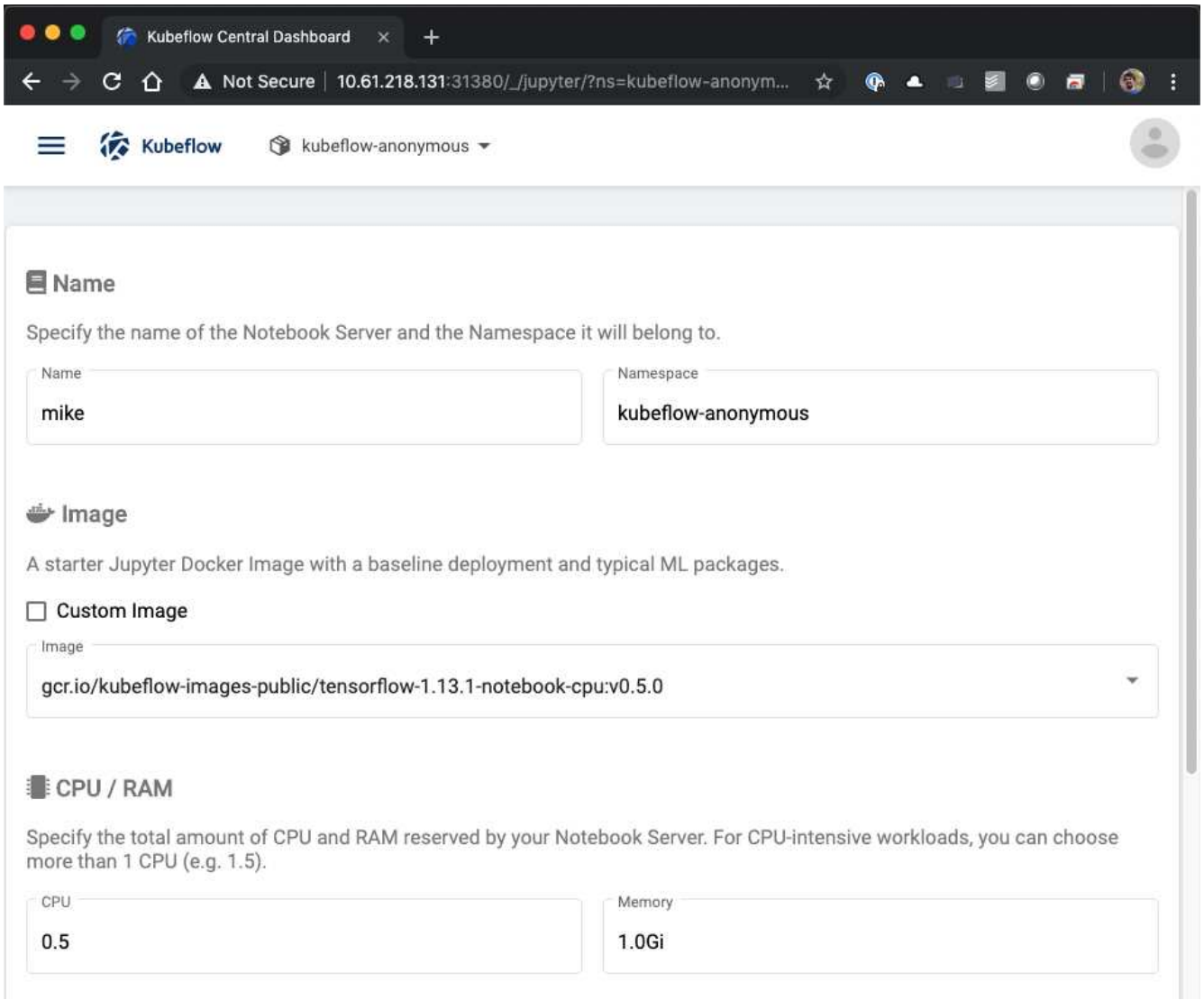


2. Click New Server to provision a new Jupyter Notebook server.

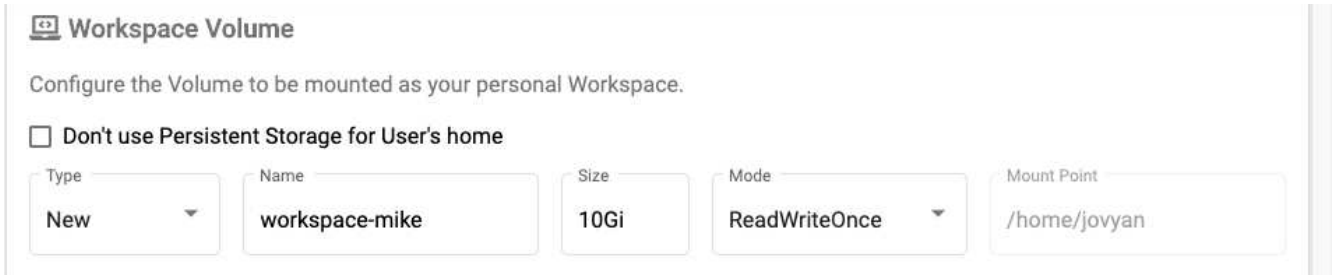


3. Give your new server a name, choose the Docker image that you want your server to be based on, and specify the amount of CPU and RAM to be reserved by your server. If the Namespace field is blank, use the Select Namespace menu in the page header to choose a namespace. The Namespace field is then auto-populated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

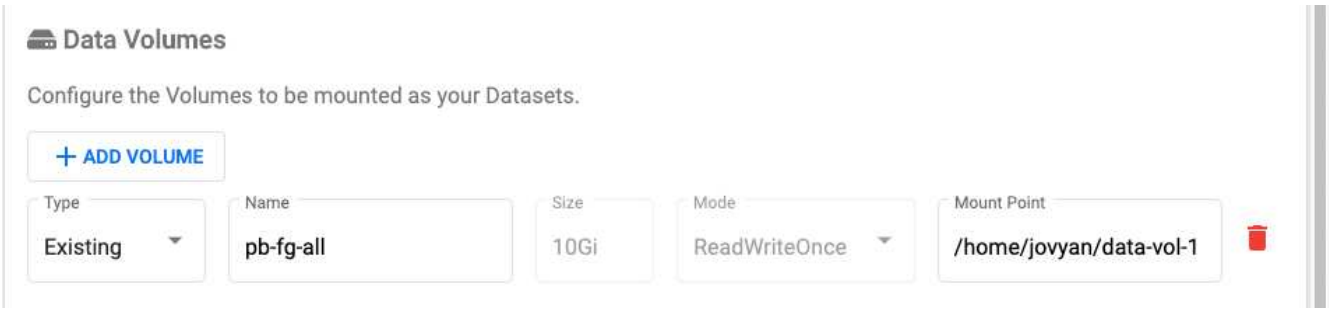
   In the following example, the `kubeflow-anonymous` namespace is chosen. In addition, the default values for Docker image, CPU, and RAM are accepted.



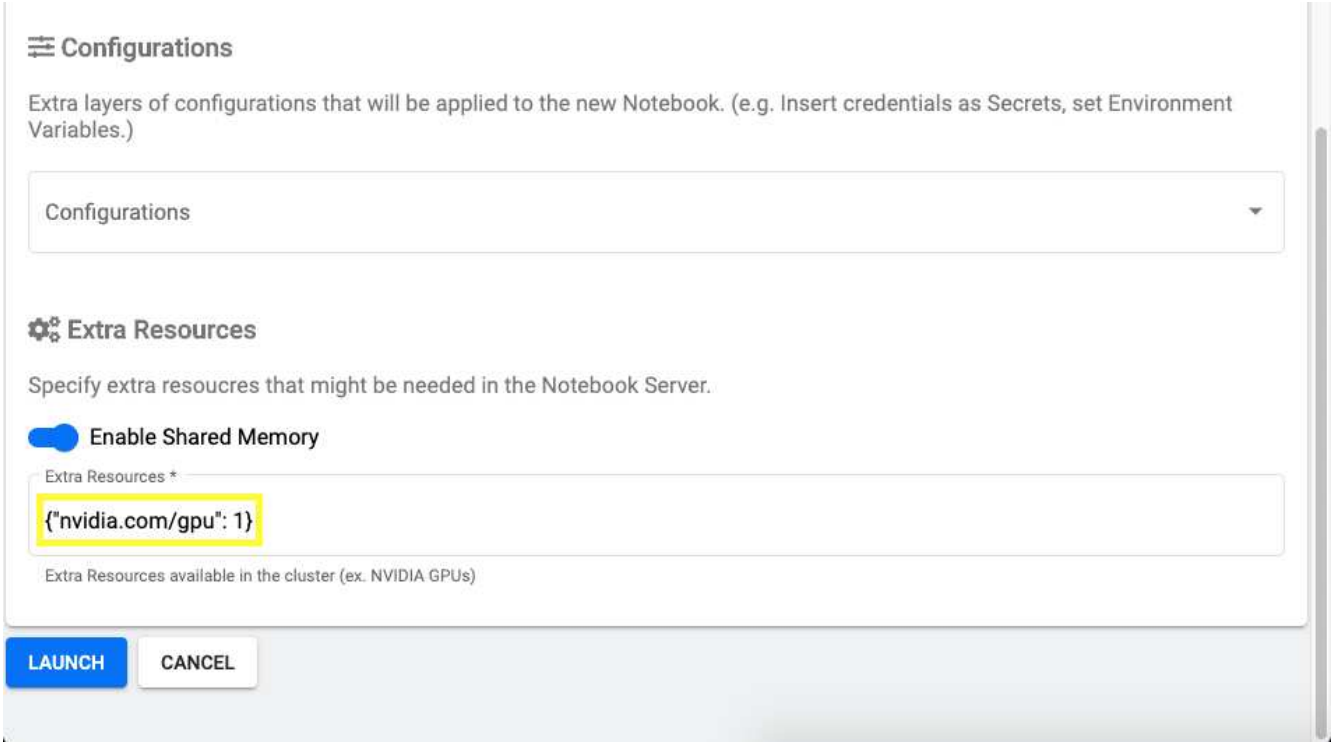
4. Specify the workspace volume details. If you choose to create a new volume, then that volume or PVC is provisioned using the default StorageClass. Because a StorageClass utilizing Trident was designated as the default StorageClass in the section Kubeflow Deployment, the volume or PVC is provisioned 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.



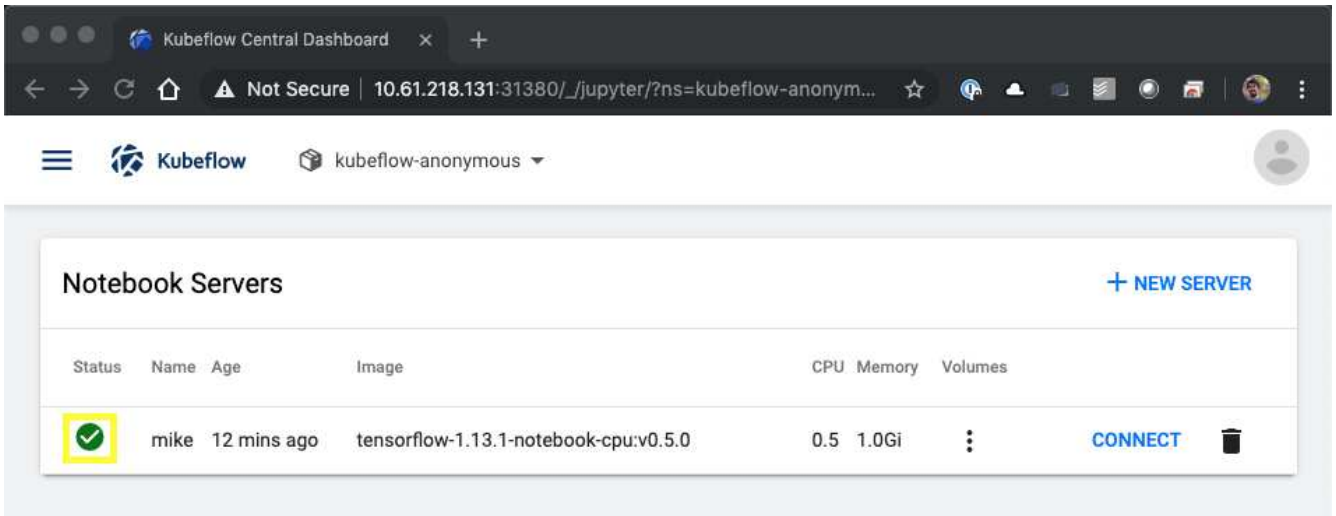
5. Add data volumes. The following example specifies an existing PVC named 'pb-fg-all' and accepts the default mount point.



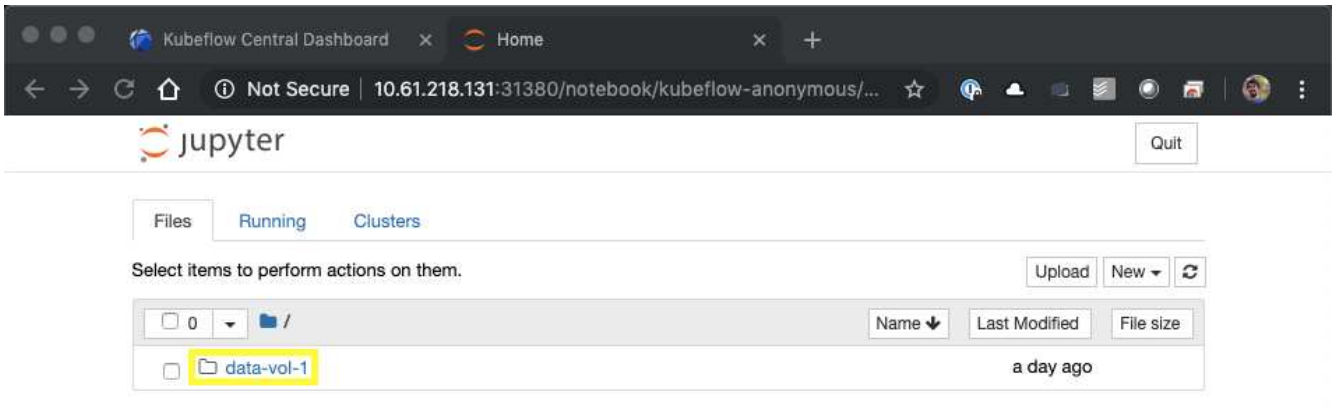
6. **Optional:** Request that the desired number of GPUs be allocated to your notebook server. In the following example, one GPU is requested.

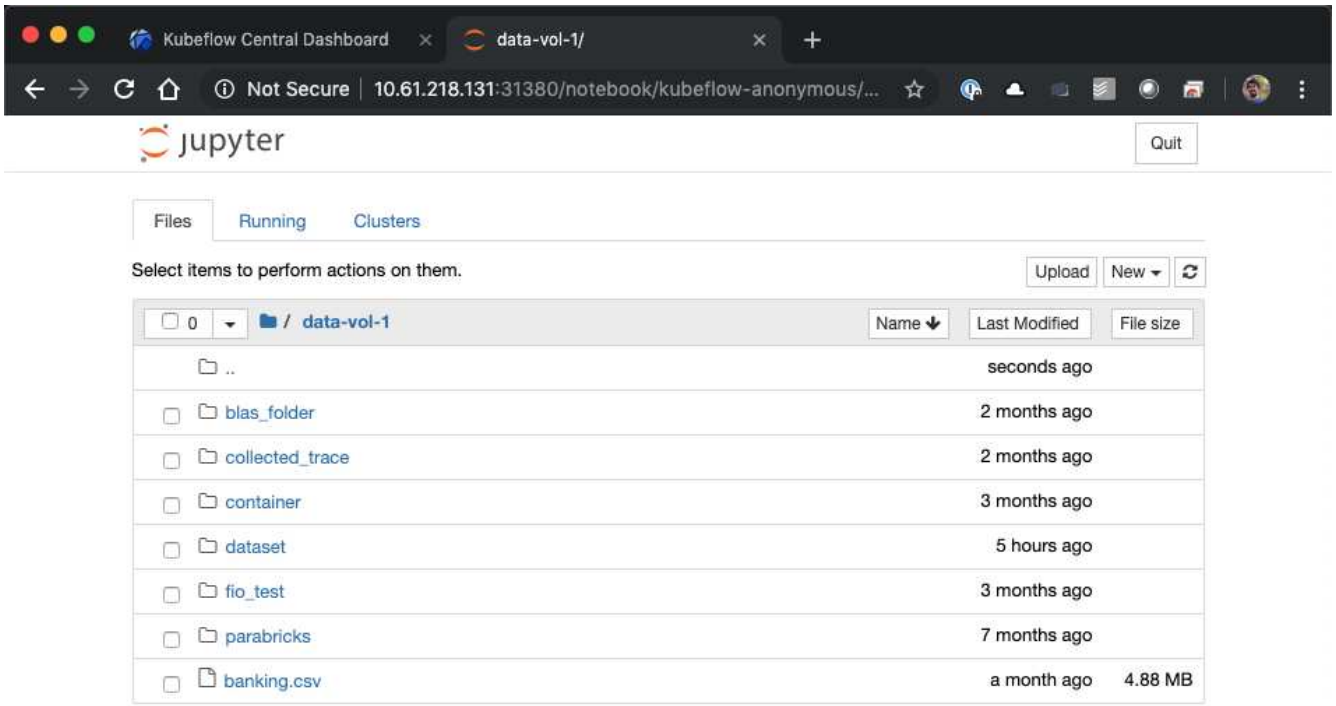


7. Click Launch to provision your new notebook server.
8. Wait for your notebook server to be fully provisioned. This can take several minutes if you have never provisioned a server using the Docker image that you specified because the image needs to be downloaded. When your server has been fully provisioned,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



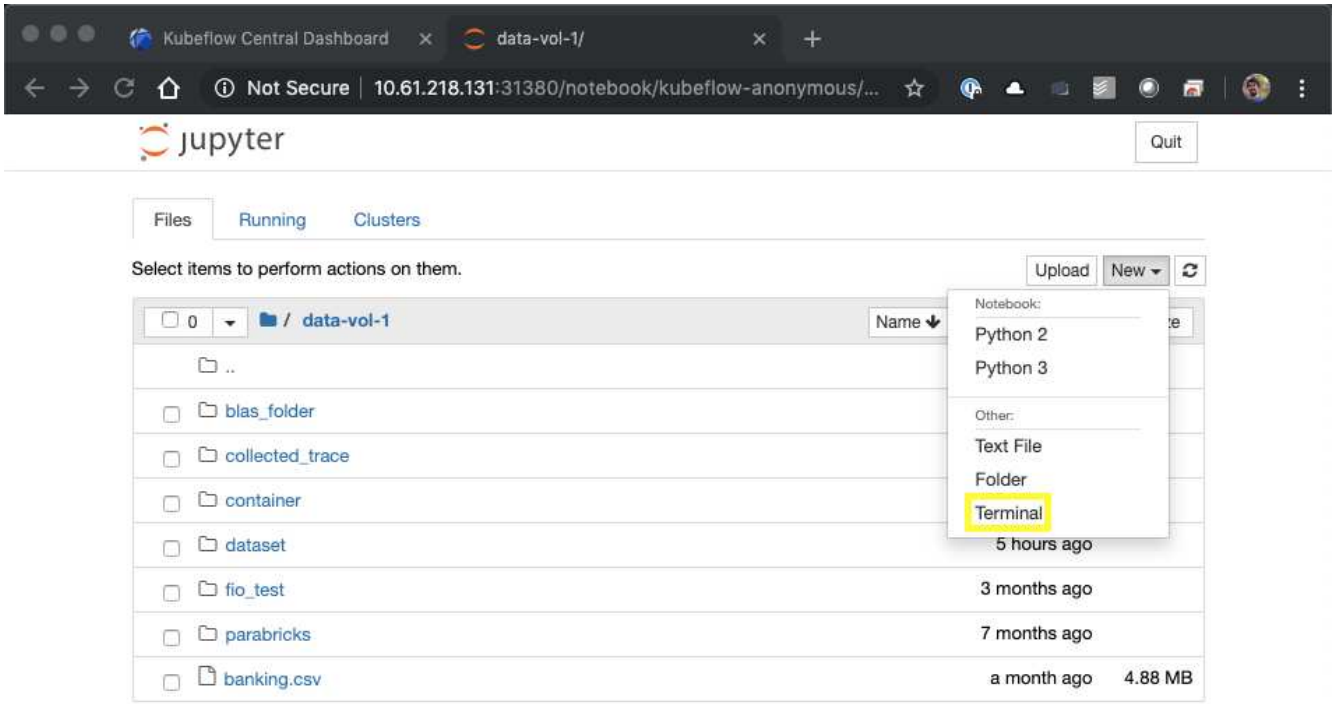
9. Click Connect to connect to your new server web interface.
10. Confirm that the dataset volume that was specified in step 6 is mounted on the server. Note that this volume is mounted within the default workspace by default. From the perspective of the user, this is just another folder within the workspace.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

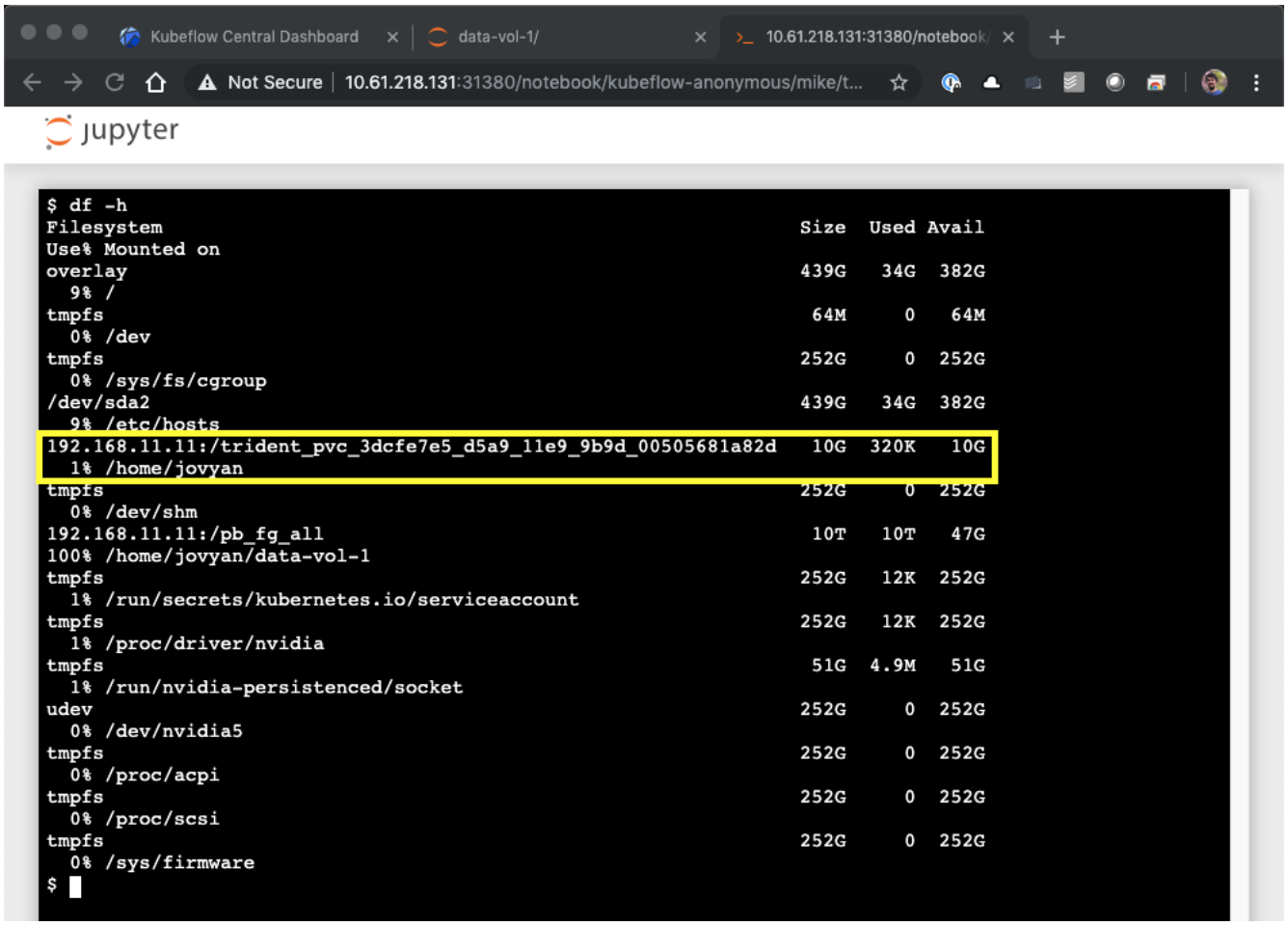




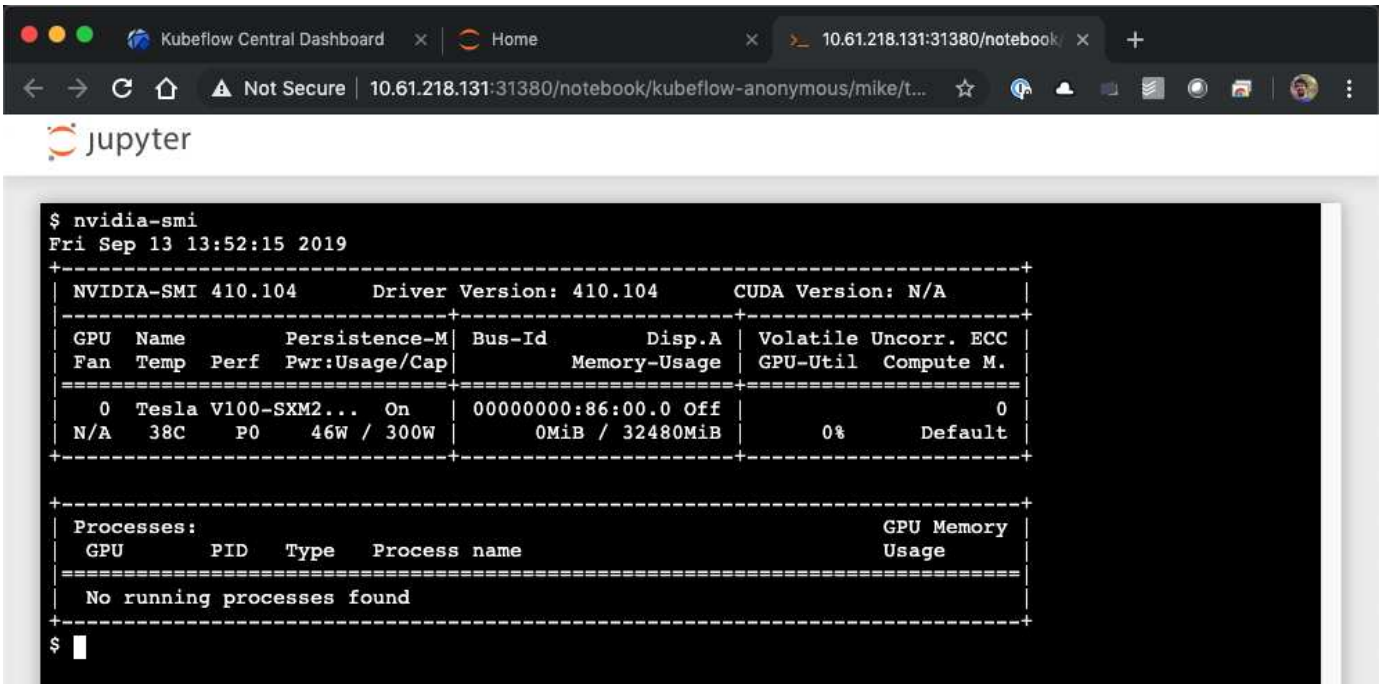
11. Open a Terminal and, assuming that a new volume was requested in step 5, execute `df -h` to confirm that a new Trident-provisioned persistent volume is mounted as the default workspace.

    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 you create by using the web interface are stored on this Trident-provisioned persistent volume.





12. 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.



Example Notebooks and Pipelines

The NetApp Data Science Toolkit for Kubernetes can be used in conjunction with Kubeflow. Using the NetApp Data Science Toolkit with Kubeflow provides the following benefits:

- Data scientists can perform advanced NetApp data management operations directly from within a Jupyter Notebook.
- Advanced NetApp data management operations can be incorporated into automated workflows using the Kubeflow Pipelines framework.

Refer to the Kubeflow Examples section within the NetApp Data Science Toolkit GitHub repository for details on using the toolkit with Kubeflow.

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.

> 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 solution.

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 Kubeflow Deployment. 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 Artifact 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: "CeleryExecutor"
  ## environment variables for the web/scheduler/worker Pods (for
airflow configs)
  ##
  #
####################################
# 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
  ##
  gitSync:
    enabled: true
    ## url of the git repository
    ##
    repo: "git@github.com:mboglesby/airflow-dev.git"
    ## the branch/tag/sha1 which we clone
    ##
    branch: master
    revision: HEAD
```

```
    ## 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
    ##
    sshSecret: "airflow-ssh-git-secret"
    ## the name of the private key file in your `git.secret`
    ##
    ## NOTE:
    ## - this is ONLY RELEVANT for PRIVATE SSH git repos
    ##
    sshSecretKey: id_rsa
    ## the git sync interval in seconds
    ##
    syncWait: 60
EOF
$ helm install airflow airflow-stable/airflow -n airflow --version 8.0.8
--values ./custom-values.yaml
...
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
```

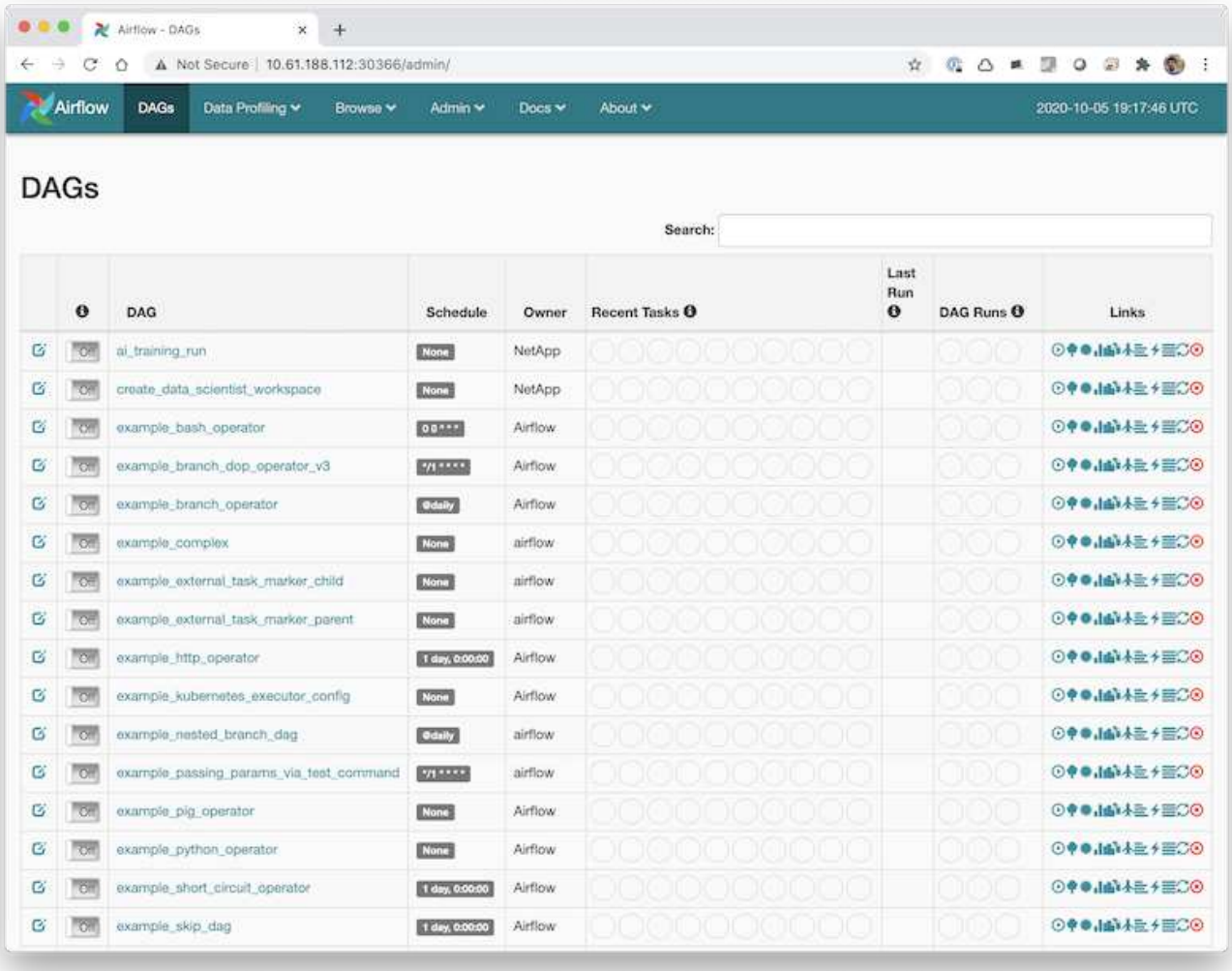
2. Confirm that all Airflow pods are up and running. It may take a few minutes for all pods to start.

```
$ kubectl -n airflow get pod
NAME                                READY   STATUS    RESTARTS   AGE
airflow-flower-b5656d44f-h8qjk      1/1     Running   0          2h
airflow-postgresql-0                1/1     Running   0          2h
airflow-redis-master-0              1/1     Running   0          2h
airflow-scheduler-9d95fcdf9-clf4b   2/2     Running   2          2h
airflow-web-59c94db9c5-z7rg4        1/1     Running   0          2h
airflow-worker-0                    2/2     Running   2          2h
```

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/
```

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



Example Apache Airflow Workflows

The NetApp Data Science Toolkit for Kubernetes can be used in conjunction with Airflow. Using the NetApp Data Science Toolkit with Airflow enables you to incorporate NetApp data management operations into automated workflows that are orchestrated by Airflow.

Refer to the Airflow Examples section within the NetApp Data Science Toolkit GitHub repository for details on using the toolkit with Airflow.

Example Trident Operations

This section includes examples of various operations that you may want to perform with Trident.

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.

An `accessModes` value of `ReadOnlyMany` is specified in the example PVC spec files. For more information about the `accessMode` field, see the official Kubernetes documentation.

> The Backend names that are specified in the following example import commands correspond to the Backends that were created in the example in the section Example Trident Backends for ONTAP AI Deployments, step 1. The StorageClass names that are specified in the following example PVC definition files 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.

An `accessModes` value of `ReadWriteMany` is specified in the following example PVC definition file. 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 Kubernetes is deployed on an ONTAP AI pod.

Example High-performance Jobs for ONTAP AI Deployments

This section includes examples of various high-performance jobs that can be executed when Kubernetes 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 in the pod definition. This step is a Kubernetes-native operation; no

NetApp expertise is required.

> 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, in order to maximize storage bandwidth, the volume that contains the needed training data is mounted twice within the pod that this job creates. Another volume is also mounted in the pod. This second volume will be used to store results and metrics. 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_be
nchmarks.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_ima
genet_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.

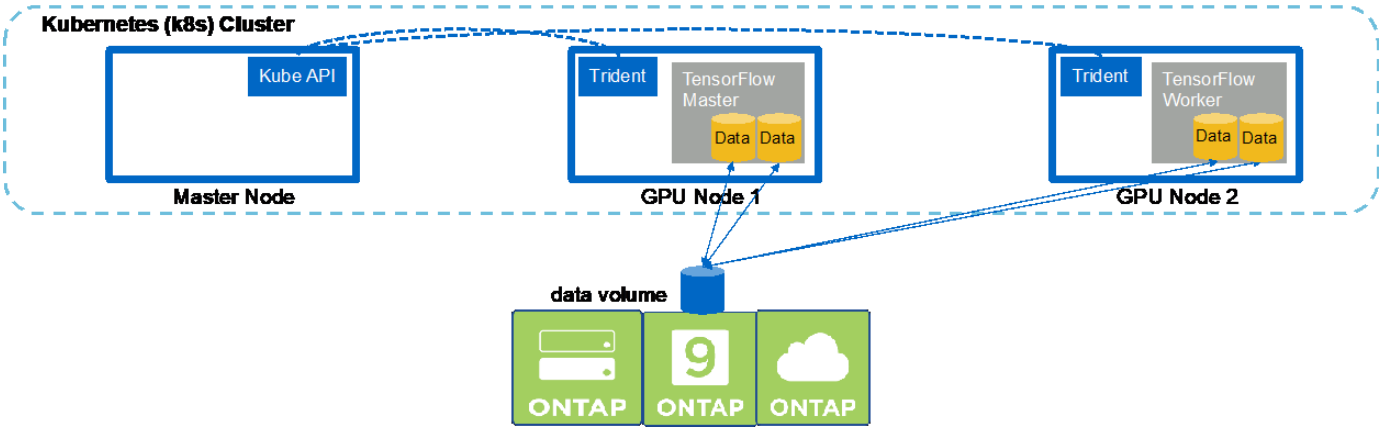
   When you delete the job object, Kubernetes automatically deletes any associated pods.

```
$ 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 the following figure for a depiction of a synchronous distributed AI job.

> 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.



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.

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_be
nchmarks.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_im
agenet_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.

   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.

   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 AI 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 AI training jobs does not adversely affect performance. See the following table for the results of our performance comparison.

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 solution.

This solution enables you to rapidly clone a data namespace. 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, minimal 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.