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 solution demonstrates how you can address these challenges by pairing NetApp data management capabilities with popular open-source tools and frameworks.

This solution is intended to demonstrate several different open-source tools and frameworks that can be incorporated into an MLOps workflow. These different tools and frameworks can be used together or by themselves depending on the requirements and use case.

The following tools/frameworks are covered in this solution:

- Apache Airflow
- Kubeflow

Technology Overview

This section focuses on the technology overview for OpenSource MLOps with NetApp.
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](https://www.docker.com).

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. For more information, visit the Kubernetes website.

**NetApp Astra Trident**

Astra Trident enables consumption and management of storage resources across all popular NetApp storage platforms, in the public cloud or on premises, including ONTAP (AFF, FAS, Select, Cloud, Amazon FSx for NetApp ONTAP), Element software (NetApp HCI, SolidFire), Azure NetApp Files service, and Cloud Volumes Service on Google Cloud. Astra Trident is a Container Storage Interface (CSI) compliant dynamic storage orchestrator that natively integrates with Kubernetes.

**NetApp DataOps Toolkit**

The NetApp DataOps Toolkit is a Python-based tool that simplifies the management of development/training workspaces and inference servers that are backed by high-performance, scale-out NetApp storage. Key capabilities include:

- Rapidly provision new high-capacity workspaces that are backed by high-performance, scale-out NetApp storage.
- Near-instaneously clone high-capacity workspaces in order to enable experimentation or rapid iteration.
- Near-instaneously save snapshots of high-capacity workspaces for backup and/or traceability/baselining.
- Near-instaneously provision, clone, and snapshot high-capacity, high-performance data volumes.

**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 is a good open-source option for organizations that prefer an all-in-one MLOps platform. 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 text. 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.

**Katib**

Katib is a Kubernetes-native project for automated machine learning (AutoML). Katib supports hyperparameter tuning, early stopping and neural architecture search (NAS). Katib is the project which is agnostic to machine learning (ML) frameworks. It can tune hyperparameters of applications written in any language of the users’ choice and natively supports many ML frameworks, such as TensorFlow, MXNet, PyTorch, XGBoost, and
Katib supports a lot of various AutoML algorithms, such as Bayesian optimization, Tree of Parzen Estimators, Random Search, Covariance Matrix Adaptation Evolution Strategy, Hyperband, Efficient Neural Architecture Search, Differentiable Architecture Search and many more. 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

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

Simplify data management

Data management is crucial to enterprise IT operations and data scientists so that appropriate resources are used for AI applications and training AI/ML datasets. The following additional information about NetApp technologies is out of scope for this validation but might be relevant depending on your deployment.

ONTAP data management software includes the following features to streamline and simplify 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. This applies to data stored locally and data tiered to the cloud.
- Minimum, maximum, and adaptive quality of service (AQoS). Granular quality of service (QoS) controls help maintain performance levels for critical applications in highly shared environments.
- NetApp FabricPool. Provides automatic tiering of cold data to public and private cloud storage options, including Amazon Web Services (AWS), Azure, and NetApp StorageGRID storage solution. For more information about FabricPool, see TR-4598: FabricPool best practices.

Accelerate and protect data

ONTAP delivers superior levels of performance and data protection and extends these capabilities in the following ways:
• Performance and lower latency. ONTAP offers the highest possible throughput at the lowest possible latency.

• Data protection. ONTAP provides built-in data protection capabilities with common management across all platforms.

• NetApp Volume Encryption (NVE). ONTAP offers native volume-level encryption with both onboard and External Key Management support.

• Multitenancy and multifactor authentication. ONTAP enables sharing of infrastructure resources with the highest levels of security.

Future-proof infrastructure

ONTAP helps meet demanding and constantly changing business needs with the following features:

• Seamless scaling and nondisruptive operations. ONTAP supports the nondisruptive addition of capacity to existing controllers and to scale-out clusters. Customers can upgrade to the latest technologies, such as NVMe and 32Gb FC, without costly data migrations or outages.

• Cloud connection. ONTAP is the most cloud-connected storage management software, with options for software-defined storage and cloud-native instances in all public clouds.

• Integration with emerging applications. ONTAP offers enterprise-grade data services for next generation platforms and applications, such as autonomous vehicles, smart cities, and Industry 4.0, by using the same infrastructure that supports existing enterprise apps.

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).
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.

Architecture

This solution is not dependent on 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, Amazon FSx for NetApp ONTAP, Azure NetApp Files, 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 Astra 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.

<table>
<thead>
<tr>
<th>Software Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Airflow</td>
<td>2.0.1</td>
</tr>
<tr>
<td>Apache Airflow Helm Chart</td>
<td>8.0.8</td>
</tr>
<tr>
<td>Kubeflow</td>
<td>1.7, deployed via deployKF 0.1.1</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>1.26</td>
</tr>
<tr>
<td>NetApp Astra Trident</td>
<td>23.07</td>
</tr>
</tbody>
</table>

**Support**

NetApp does not offer enterprise support for Apache Airflow, Kubeflow, or Kubernetes. If you are interested in a fully supported MLOps platform, contact NetApp about fully supported MLOps solutions that NetApp offers jointly with partners.

**NetApp Astra Trident Configuration**

**Example Astra Trident Backends for NetApp AlPod Deployments**

Before you can use Astra 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 components of this solution on a NetApp AlPod. For more information about Backends, see the Astra Trident documentation.

1. NetApp recommends creating a FlexGroup-enabled Trident Backend for your AlPod.

   The example commands that follow show the creation of a FlexGroup-enabled Trident Backend for an AlPod storage virtual machine (SVM). This Backend uses 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.
2. NetApp also recommends creating a FlexVol-enabled Trident Backend. You may want to use FlexVol volumes for hosting persistent applications, 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.
$ cat << EOF > ./trident-backend-aipod-flexvols.json
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "aipod-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-aipod-flexvols.json -n trident

+-------------------------+---------------------+-------------------------+-------------------------+---------------+------------+---------+
|                       | NAME                 | STORAGE DRIVER         | UUID                    | STATE        | VOLUMES   |         |
+-------------------------+---------------------+-------------------------+-------------------------+---------------+------------+---------+
| aipod-flexvols          | ontpap-nas           | 52bdb3b1-13a5-4513-a9c1-52a69657fabe | online | 0 |       0 |
+-------------------------+---------------------+-------------------------+-------------------------+---------------+------------+---------+

$ tridentctl get backend -n trident

+-------------------------+---------------------+-------------------------+-------------------------+---------------+------------+---------+
|                       | NAME                 | STORAGE DRIVER         | UUID                    | STATE        | VOLUMES   |         |
+-------------------------+---------------------+-------------------------+-------------------------+---------------+------------+---------+
| aipod-flexvols          | ontpap-nas           | 52bdb3b1-13a5-4513-a9c1-52a69657fabe | online | 0 |       0 |
| aipod-flexgroups-iface1 | ontpap-nas-flexgroup | b74cbddd-e0b8-40b7-b263-b6da66dec0b0d | online | 0 |         |
+-------------------------+---------------------+-------------------------+-------------------------+---------------+------------+---------+

**Example Kubernetes StorageClasses for NetApp AlPod Deployments**

Before you can use Astra 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 components of this solution on a NetApp AlPod. For more information about StorageClasses, see the Astra Trident documentation.
1. NetApp recommends creating a StorageClass for the FlexGroup-enabled Trident Backend that you created in the section Example Astra Trident Backends for NetApp AIPod Deployments, step 1. The example commands that follow show the creation of multiple StorageClasses that corresponds to the two example Backend that was created in the section Example Astra Trident Backends for NetApp AIPod Deployments, step 1 - one that utilizes NFS over RDMA and one that does not.

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.

Note: The following example StorageClasses use a maximum transfer size of 262144. To use this maximum transfer size, you must configure the maximum transfer size on your ONTAP system accordingly. Refer to the ONTAP documentation for details.

Note: To use NFS over RDMA, you must configure NFS over RDMA on your ONTAP system. Refer to the link https://docs.netapp.com/us-en/ontap/nfs-rdma/[ONTAP documentation] for details.

Note: In the following example, a specific Backend is not specified in the storagePool field in StorageClass definition file.
2. NetApp also recommends creating a StorageClass that corresponds to the FlexVol-enabled Trident Backend that you created in the section Example Astra Trident Backends for AIPod Deployments, step 2. The example commands that follow show the creation of a single StorageClass for FlexVol volumes.

   Note: In the following example, a particular Backend is not specified in the storagePool field in StorageClass definition file. When you use Kubernetes to administer volumes using this StorageClass, Trident attempts to use any available backend that uses the ontap-nas driver.
$ cat << EOF > ./storage-class-aipod-flexvols-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aipod-flexvols-retain
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
reclaimPolicy: Retain
EOF

$ kubectl create -f ./storage-class-aipod-flexvols-retain.yaml
storageclass.storage.k8s.io/aipod-flexvols-retain created

$ kubectl get storageclass

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROVISIONER</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>aipod-flexgroups-retain</td>
<td>csi.trident.netapp.io</td>
<td>0m</td>
</tr>
<tr>
<td>aipod-flexgroups-retain-rdma</td>
<td>csi.trident.netapp.io</td>
<td>0m</td>
</tr>
<tr>
<td>aipod-flexvols-retain</td>
<td>csi.trident.netapp.io</td>
<td>0m</td>
</tr>
</tbody>
</table>

**Kubeflow**

**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 the Kubeflow version that you intend to deploy. For a list of supported Kubernetes versions, refer to the dependencies for your Kubeflow version in the official Kubeflow documentation.

2. You have already installed and configured NetApp Astra Trident in your Kubernetes cluster. For more details on Astra Trident, refer to the Astra Trident documentation.

**Set Default Kubernetes StorageClass**

Before you deploy Kubeflow, we recommend designating a default StorageClass within your Kubernetes cluster. The Kubeflow deployment process may attempt to provision new persistent volumes using the default StorageClass. If no StorageClass is designated as the default StorageClass, then the deployment may fail. 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 command to get storage classes](image)

```bash
$ 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
```

**Kubeflow Deployment Options**

There are many different options for deploying Kubeflow. Refer to the official Kubeflow documentation for a list of deployment options, and choose the option that is the best fit for your needs.

For validation purposes, we deployed Kubeflow 1.7 using `deployKF` 0.1.1.

**Example Kubeflow Operations and Tasks**

**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. For more information about Jupyter Notebooks within the Kubeflow context, see the official Kubeflow documentation.
Use the NetApp DataOps Toolkit with Kubeflow

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, such as creating snapshots and clones, directly from within a Jupyter Notebook.
- Advanced NetApp data management operations, such as creating snapshots and clones, 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.

Example Workflow - Train an Image Recognition Model Using Kubeflow and the NetApp DataOps Toolkit

This section describes the steps involved in training and deploying a Neural Network for Image Recognition using Kubeflow and the NetApp DataOps Toolkit. This is intended to serve as an example to show a training job that incorporates NetApp storage.

Prerequisites

Create a Dockerfile with the required configurations to use for the train and test steps within the Kubeflow pipeline.

Here is an example of a Dockerfile -
FROM pytorch/pytorch:latest
RUN pip install torchvision numpy scikit-learn matplotlib tensorboard
WORKDIR /app
COPY . /app
COPY train_mnist.py /app/train_mnist.py
CMD ["python", "train_mnist.py"]

Depending on your requirements, install all required libraries and packages needed to run the program. Before you train the Machine Learning model, it is assumed that you already have a working Kubeflow deployment.

Train a Small NN on MNIST Data Using PyTorch and Kubeflow Pipelines

We use the example of a small Neural Network trained on MNIST data. The MNIST dataset consists of handwritten images of digits from 0-9. The images are 28x28 pixels in size. The dataset is divided into 60,000 train images and 10,000 validation images. The Neural Network used for this experiment is a 2-layer feedforward network. Training is executed using Kubeflow Pipelines. Refer to the documentation here for more information. Our Kubeflow pipeline incorporates the docker image from the Prerequisites section.

Visualize Results Using Tensorboard

Once the model is trained, we can visualize the results using Tensorboard. Tensorboard is available as a feature on the Kubeflow Dashboard. You can create a custom tensorboard for your job. An example below shows the plot of training accuracy vs. number of epochs and training loss vs. number of epochs.
Experiment with Hyperparameters Using Katib

Katib is a tool within Kubeflow that can be used to experiment with the model hyperparameters. To create an experiment, define a desired metric/goal first. This is usually the test accuracy. Once the metric is defined, choose hyperparameters that you would like to play around with (optimizer/learning_rate/number of layers). Katib does a hyperparameter sweep with the user-defined values to find the best combination of parameters that satisfy the desired metric. You can define these parameters in each section in the UI. Alternatively, you could define a YAML file with the necessary specifications. Below is an illustration of a Katib experiment -
Use NetApp Snapshots to Save Data for Traceability

During the model training, we may want to save a snapshot of the training dataset for traceability. To do this, we can add a snapshot step to the pipeline as shown below. To create the snapshot, we can use the NetApp DataOps Toolkit for Kubernetes.

```python
@ml.pipeline()
def mlst_pipeline():
    mlst_train_task = mlst_train_mp()
    mlst_train_task.apply(
        k8s.operators.mpr('mlst-data', 'mlst-data-vm', '/mnt/data/'))
    mlst_test_task = mlst_test_mp()
    mlst_test_task.apply(
        k8s.operators.mpr('mlst-data', 'mlst-data-vm', '/mnt/data/'))
    volume_snapshot_name = 'mlst-pytorch-snapshot'
dataset_snapshot = mlst_container(mp
        name='dataset-container',
        image='python:3.9',
        command=['/bin/bash', '-c', 'pipenv --install netapp-dataops k8s ALL',
            'echo " volume snapshot name " + " /dataset/snapshot_name.txt " >>
            "dataset.snapshots"'],
        mlst dintputset 'mlst-data' + ' --snapshot-name' + ' "volume snapshot name" + " -namespace=(workflow namespace)"',
        mlst dintputset 'volume snapshot name' + ' "volume snapshot name" + " -ist"
    )
    mlst_test_task.after(mlst_train_task)
dataset_snapshot.offer(mlst_train_task)
```

Refer to the NetApp DataOps Toolkit example for Kubeflow for more information.

Apache Airflow

Apache Airflow Deployment

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 Astra Trident in your Kubernetes cluster. For more details on Astra Trident, refer to the Astra Trident documentation.

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 Kubeflow Deployment section. 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>airflow-flower-b5656d44f-h8qjk</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2h</td>
</tr>
<tr>
<td>airflow-postgresql-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2h</td>
</tr>
<tr>
<td>airflow-redis-master-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2h</td>
</tr>
<tr>
<td>airflow-scheduler-9d95fcdf9-clf4b</td>
<td>2/2</td>
<td>Running</td>
<td>2</td>
<td>2h</td>
</tr>
<tr>
<td>airflow-web-59c94db9c5-z7rg4</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2h</td>
</tr>
<tr>
<td>airflow-worker-0</td>
<td>2/2</td>
<td>Running</td>
<td>2</td>
<td>2h</td>
</tr>
</tbody>
</table>

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.
Use the NetApp DataOps Toolkit with Airflow

The NetApp DataOps Toolkit for Kubernetes can be used in conjunction with Airflow. Using the NetApp DataOps Toolkit with Airflow enables you to incorporate NetApp data management operations, such as creating snapshots and clones, into automated workflows that are orchestrated by Airflow.

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

Example Astra Trident Operations

This section includes examples of various operations that you may want to perform with Astra 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 a volume named `pb_fg_all`. 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 `accessModes` field, see the official Kubernetes documentation.

```bash
$ 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 |
+--------------------------------+--------+---------+

$ 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 |
+----------------------------------+---------+
Provision a New Volume

You can use Trident to provision a new volume on your NetApp storage system or platform.

Provision a New Volume Using kubectl

The following example commands show the provisioning of a new FlexVol volume using kubectl.

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.

```yaml
$ 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   25h
tensorflow-results                Bound     default-tensorflow-results-2fd60   1073741824       RWX            ontap-ai-flexvols-retain               26h
EOF
```
Provision a New Volume Using the NetApp DataOps Toolkit

You can also use the NetApp DataOps Toolkit for Kubernetes to provision a new volume on your NetApp storage system or platform. The NetApp DataOps Toolkit for Kubernetes utilizes Trident to provision volumes but simplifies the process for the user. Refer to the documentation for details.

Example High-performance Jobs for AIPod Deployments

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 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.

```bash
$ cat << EOF > ./netapp-tensorflow-single-imagenet.yaml
apiVersion: batch/v1
kind: Job
EOF
```
metadata:
  name: netapp-tensorflow-single-imagenet
spec:
  backoffLimit: 5
  template:
    spec:
      volumes:
        - name: dshm
          emptyDir:
            medium: Memory
        - name: testdata-iface1
          persistentVolumeClaim:
            claimName: pb-fg-all-iface1
        - name: testdata-iface2
          persistentVolumeClaim:
            claimName: pb-fg-all-iface2
        - name: results
          persistentVolumeClaim:
            claimName: tensorflow-results
      containers:
        - name: netapp-tensorflow-py2
          image: netapp/tensorflow-py2:19.03.0
          command: ["python", "/netapp/scripts/run.py", "--dataset_dir=/mnt/mount_0/dataset/imagenet", "--dgx_version=dgx1", "--num_devices=8"]
          resources:
            limits:
              nvidia.com/gpu: 8
          volumeMounts:
            - mountPath: /dev/shm
              name: dshm
            - mountPath: /mnt/mount_0
              name: testdata-iface1
            - mountPath: /mnt/mount_1
              name: testdata-iface2
            - mountPath: /tmp
              name: results
          securityContext:
            privileged: true
          restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-single-imagenet.yaml
job.batch/netapp-tensorflow-single-imagenet created
$ kubectl get jobs
NAME                     COMPLETIONS DURATION  AGE
netapp-tensorflow-single-imagenet  0/1        24s     24s
2. Confirm that the job that you created in step 1 is running correctly. The following example command confirms that a single pod was created for the job, as specified in the job definition, and that this pod is currently running on one of the GPU worker nodes.

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

3. Confirm that the job that you created in step 1 completes successfully. The following example commands confirm that the job completed successfully.
$ kubectl get jobs
NAME                                    COMPLETIONS DURATION
AGE
netapp-tensorflow-single-imagenet       1/1         5m42s
10m

$ kubectl get pods
NAME                                      READY STATUS
RESTARTS AGE
netapp-tensorflow-single-imagenet-m7x92   0/1     Completed
0    11m

$ kubectl logs netapp-tensorflow-single-imagenet-m7x92
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 702
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at line 711
Total images/sec = 6530.59125
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 1 -H localhost:1 bash -c 'sync; echo 1 > /proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 8 -H localhost:8 -bind-to none -map-by slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH python /netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=resnet50 --batch_size=256 --device=gpu
--force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48
--variable_update=horovod --batch_group_size=20 --num_batches=500
--nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True
--use_tf_layers= False --data_name=imagenet --use_datasets=True
--data_dir=/mnt/mount_0/dataset/imagenet
--datasets_parallel_interleave_cycle_length=10
--datasets_sloppy_parallel_interleave=False --num_mounts=2
--mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000
--datasets_use_prefetch=True --datasets_num_private_threads=4
--horovod_device=gpu
/tmp/20190814_105450_tensorflow_horovod_rdma_resnet50_gpu_8_256_b500_imagenet_nodistort_fp16_r10_m2_nockpt.txt 2>&1

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

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

```yaml
$ 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:
EOF
```
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-ifacel
- 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>UP-TO-DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>netapp-tensorflow-multi-imagenet-worker</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
</tr>
</thead>
<tbody>
<tr>
<td>netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725</td>
<td>1/1</td>
</tr>
<tr>
<td>Running 0</td>
<td>60s</td>
</tr>
<tr>
<td>10.61.218.154</td>
<td>10.61.218.154</td>
</tr>
</tbody>
</table>

$ 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
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
EOF
```
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
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-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 delete Pods
$ kubectl get pods
No resources found.

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

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