Abstract

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

1 Introduction .......................................................................................................................... 4

2 Concepts and Components ..................................................................................................... 5
   2.1 Artificial Intelligence ........................................................................................................ 5
   2.2 Containers ....................................................................................................................... 6
   2.3 Kubernetes ..................................................................................................................... 6
   2.4 NetApp Trident .............................................................................................................. 6
   2.5 NVIDIA DeepOps .......................................................................................................... 7
   2.6 Kubeflow ....................................................................................................................... 7
   2.7 NetApp ONTAP 9 .......................................................................................................... 8
   2.8 NetApp Snapshot Copies .............................................................................................. 9
   2.9 NetApp FlexClone Technology .................................................................................. 10
   2.10 NetApp SnapMirror Data Replication Technology ................................................... 11
   2.11 NetApp ONTAP FlexGroup Volumes ................................................................. 12

3 Hardware and Software Requirements .................................................................................... 13

4 Kubernetes Deployment ......................................................................................................... 14
   4.1 Prerequisites ............................................................................................................... 14
   4.2 Use NVIDIA DeepOps to Install and Configure Kubernetes ................................. 14

5 NetApp Trident Deployment and Configuration ...................................................................... 15
   5.1 Prerequisites ............................................................................................................... 15
   5.2 Install Trident ............................................................................................................. 15
   5.3 Example Trident Backends for ONTAP AI Deployments ........................................ 15
   5.4 Example Kubernetes StorageClasses for ONTAP AI Deployments ..................... 17

6 Kubeflow Deployment ............................................................................................................. 19
   6.1 Prerequisites ............................................................................................................... 19
   6.2 Set Default Kubernetes StorageClass ...................................................................... 19
   6.3 Use NVIDIA DeepOps to Deploy Kubeflow ............................................................ 20

7 Example Kubeflow Operations and Tasks ............................................................................... 24
   7.1 Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use ........ 24
   7.2 Create a Snapshot of an ONTAP Volume from Within a Jupyter Notebook ............ 33
   7.3 Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning ..................................................... 38
   7.4 Create a Kubeflow Pipeline to Rapidly Clone a Dataset for a Data Scientist Workspace ................................................................. 59
   7.5 Create a Kubeflow Pipeline to Trigger a SnapMirror Volume Replication Update ....... 73
8 Example Trident Operations ........................................................................................................... 76
  8.1 Import an Existing Volume .............................................................................................................. 76
  8.2 Provision a New Volume ................................................................................................................ 77

9 Example High-performance Jobs for ONTAP AI Deployments ................................................... 78
  9.1 Execute a Single-Node AI Workload ............................................................................................... 78
  9.2 Execute a Synchronous Distributed AI Workload ......................................................................... 80

10 Performance Testing ......................................................................................................................... 85

11 Conclusion ......................................................................................................................................... 85

Acknowledgments .................................................................................................................................. 85

Where to Find Additional Information ................................................................................................. 86

Version History ....................................................................................................................................... 87

LIST OF TABLES
Table 1) Validation environment infrastructure details. ........................................................................... 13
Table 2) Validation environment software version details. ....................................................................... 14
Table 3) Performance comparison results. ................................................................................................. 85

LIST OF FIGURES
Figure 1) Solution visualization. ............................................................................................................. 5
Figure 2) Virtual machines versus containers. .......................................................................................... 6
Figure 3) Kubeflow visualization .............................................................................................................. 8
Figure 4) NetApp Snapshot copies ......................................................................................................... 10
Figure 5) NetApp FlexClone technology ................................................................................................ 11
Figure 6) NetApp SnapMirror example. .................................................................................................. 12
Figure 7) NetApp FlexGroup volumes. .................................................................................................... 13
Figure 8) Synchronous distributed AI job. ............................................................................................... 81
1 Introduction

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

This report shows you how to rapidly clone a data namespace just as you would a Git repo. It also shows you how to seamlessly replicate data across sites and regions in order 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.

Because this solution is targeted towards data scientists and data engineers, no NetApp or NetApp ONTAP® expertise is required. With this solution, data management functions can be executed using simple and familiar tools and interfaces. Furthermore, this solution uses 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 minutes.
### 2 Concepts and Components

#### 2.1 Artificial Intelligence

AI is a computer science discipline in which computers are trained to mimic the cognitive functions of the human mind. AI 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.
2.2 Containers

Containers are isolated user-space instances that run on top of a shared host operating system kernel. The adoption of containers is increasing rapidly. Containers offer many of the same application sandboxing benefits that virtual machines (VMs) offer. However, because the hypervisor and guest operating system layers that VMs rely on have been eliminated, containers are far more lightweight. See Figure 2 for a visualization.

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

Figure 2) Virtual machines versus containers.

2.3 Kubernetes

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

2.4 NetApp Trident

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

### 2.5 NVIDIA DeepOps

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

### 2.6 Kubeflow

Kubeflow is an open source AI and ML toolkit for Kubernetes that was originally developed by Google. The Kubeflow project makes deployments of AI and ML workflows on Kubernetes simple, portable, and scalable. Kubeflow abstracts away the intricacies of Kubernetes, allowing data scientists to focus on what they know best—data science. See Figure 3 for a visualization. Kubeflow has been gaining significant traction as enterprise IT departments have increasingly standardized on Kubernetes. For more information, visit the Kubeflow website.

#### Kubeflow Pipelines

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

#### Jupyter Notebook Server

A Jupyter Notebook Server is an open source web application that allows data scientists to create wiki-like documents called Jupyter Notebooks that contain live code as well as descriptive 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.
2.7 **NetApp ONTAP 9**

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

### Simplify Data Management

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

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

### Accelerate and Protect Data

ONTAP delivers superior levels of performance and data protection and extends these capabilities with the following features:
• **High performance and low latency.** ONTAP offers the highest possible throughput at the lowest possible latency.

• **NetApp ONTAP FlexGroup technology.** A FlexGroup volume is a high-performance data container that can scale linearly to up to 20PB and 400 billion files, providing a single namespace that simplifies data management.

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

• **NetApp Volume Encryption.** ONTAP offers native volume-level encryption with both onboard and external key management support.

### Future-Proof Infrastructure

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

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

• **Cloud connection.** ONTAP is one of the most cloud-connected storage management software, with options for software-defined storage (ONTAP Select) and cloud-native instances (NetApp Cloud Volumes Service) in all public clouds.

• **Integration with emerging applications.** By using the same infrastructure that supports existing enterprise apps, ONTAP offers enterprise-grade data services for next-generation platforms and applications such as OpenStack, Hadoop, and MongoDB.

### 2.8 NetApp Snapshot Copies

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

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

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

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

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

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

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

A FlexGroup volume (Figure 7) is a single namespace that comprises multiple constituent member volumes. From a storage administrator viewpoint, a FlexGroup volume is managed and acts like a NetApp FlexVol® volume. Files in a FlexGroup volume are allocated to individual member volumes and are not striped across volumes or nodes. They enable the following capabilities:

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

All procedures outlined in this document were validated on the NetApp ONTAP AI converged infrastructure solution described in NVA-1121. This verified architecture pairs a NetApp AFF A800 all-flash storage system with the NVIDIA DGX-1 Deep Learning System using Cisco Nexus networking. For this validation exercise, two bare-metal NVIDIA DGX-1 systems, each featuring eight NVIDIA V100 GPUs, were used as Kubernetes worker nodes. A NetApp AFF A800 all-flash storage system provided a single persistent storage namespace across nodes, and two Cisco Nexus 3232C switches were used to provide network connectivity. Three virtual machines (VMs) that ran on a separate physical server outside of the ONTAP AI pod were used as Kubernetes master nodes. See Table 1 for validation environment infrastructure details. See Table 2 for validation environment software version details.

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

Table 1) Validation environment infrastructure details.

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Details</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment jump host</td>
<td>1</td>
<td>VM</td>
<td>Ubuntu 18.04.3 LTS</td>
</tr>
<tr>
<td>Kubernetes master nodes</td>
<td>3</td>
<td>VM</td>
<td>Ubuntu 18.04.3 LTS</td>
</tr>
<tr>
<td>Kubernetes worker nodes</td>
<td>2</td>
<td>NVIDIA DGX-1 (bare-metal)</td>
<td>NVIDIA DGX OS 4.0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(based on Ubuntu 18.04.2 LTS)</td>
</tr>
<tr>
<td>Storage</td>
<td>1 HA Pair</td>
<td>NetApp AFF A800</td>
<td>NetApp ONTAP 9.5 P1</td>
</tr>
<tr>
<td>Network connectivity</td>
<td>2</td>
<td>Cisco Nexus 3232C</td>
<td>Cisco NX-OS 7.0(3)I6(1)</td>
</tr>
</tbody>
</table>
4 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](https://kubeflow.org). For a list of Kubernetes versions that are supported by Trident, see the [Trident documentation](https://netapp.com/ontap/trident).

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.

For cloud-based Kubernetes deployments, NetApp recommends using the NetApp Kubernetes Service (NKS). This document does not cover the deployment of a Kubernetes cluster using NKS. If you wish to deploy a cloud-based Kubernetes cluster using NKS, visit the [NKS homepage](https://netapp.com/nks).

4.1 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](https://github.com/NVIDIA/deepops).

4.2 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](https://github.com/NVIDIA/deepops) on the NVIDIA DeepOps GitHub site.
2. Deploy Kubernetes in your cluster by following the instructions on the Kubernetes Deployment Guide page on the NVIDIA DeepOps GitHub site.

   **Note:** For the DeepOps Kubernetes deployment to work, the same user must exist on all Kubernetes master and worker nodes.

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

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

5 **NetApp Trident Deployment and Configuration**

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

### 5.1 Prerequisites

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

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

### 5.2 Install Trident

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

1. Deploy Trident for Kubernetes in your cluster by following the deployment instructions in the Trident documentation. Be sure to create at least one Trident Backend and at least one Kubernetes StorageClass. For more information about Backends and StorageClasses, see the Trident documentation.

   **Note:** If you are deploying the NetApp AI Control Plane solution on an ONTAP AI pod, see section 5.3 for some examples of different Trident Backends that you might want to create and section 5.4 for some examples of different Kubernetes StorageClasses that you might want to create.

### 5.3 Example Trident Backends for ONTAP AI Deployments

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

1. NetApp recommends creating a FlexGroup-enabled Trident Backend for each data LIF (logical network interface that provides data access) that you want to use on your NetApp AFF system. Due to NFS protocol limitations, a single NFS mount can provide only 1.5GBps to 2GBps of bandwidth. If
you need more bandwidth for a job, Trident enables you to add multiple NFS mounts (mounting the same NFS volume multiple times) quickly and easily when you create a Kubernetes pod. For maximum performance, these multiple mounts should be distributed across different data LIFs. You must create a Trident Backend for each data LIF that you want to use for these mounts.

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

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

```
$ cat << EOF > ./trident-backend-ontap-ai-flexgroups-iface1.json
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "backendName": "ontap-ai-flexgroups-iface1",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.11.11",
    "svm": "ontap_ai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF

$ tridentctl create backend -f ./trident-backend-ontap-ai-flexgroups-iface1.json -n trident

| VOLUMES | | STORAGEX | | STORAGEX |
|---------|-----------------|-----------------|
| VOLUMES | | STORAGEX | | STORAGEX |
| VOLUMES | | STORAGEX | | STORAGEX |
| VOLUMES | | STORAGEX | | STORAGEX |

$ 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": "ontap_ai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF

$ tridentctl create backend -f ./trident-backend-ontap-ai-flexgroups-iface2.json -n trident

| VOLUMES | | STORAGEX | | STORAGEX |
|---------|-----------------|-----------------|
| VOLUMES | | STORAGEX | | STORAGEX |
| VOLUMES | | STORAGEX | | STORAGEX |

$ tridentctl get backend -n trident

```
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.

```bash
$ 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
```

<table>
<thead>
<tr>
<th>VOLUMES</th>
<th>NAME</th>
<th>STORAGE DRIVER</th>
<th>UUID</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>------</td>
<td>----------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>online</td>
<td>ontap-ai-flexvols</td>
<td>ontap-nas</td>
<td>52b9b3b1-13a5-4513-a9c1-52a69657fabe</td>
<td>online</td>
</tr>
</tbody>
</table>

5.4 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 section 5.3, 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 section 5.3, step 1. The highlighted text shows where the Trident Backend is specified in the StorageClass definition file. For more information about StorageClasses, see the Trident documentation.

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

```bash
$ 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 section 5.3, step 2. The example commands that follow show the creation of a single StorageClass for FlexVol volumes.

**Note:** In the following example, a particular Backend is not specified in the StorageClass definition file because only one FlexVol-enabled Trident Backend was created in section 5.2, step 2. When you use Kubernetes to administer volumes that use this StorageClass, Trident attempts to use any available Backend that uses the ontap-nas driver.

```bash
$ 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-flexvols-retain-iface1 netapp.io/trident 1m
ontap-ai-flexvols-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.

```bash
$ 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           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
```

### 6. Kubeflow Deployment

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

#### 6.1 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 Section 5.

#### 6.2 Set Default Kubernetes StorageClass

Before you deploy Kubeflow, you must designate a default StorageClass within your Kubernetes cluster. The Kubeflow deployment attempts to provision new persistent volumes using the default StorageClass. If no StorageClass is designated as the default StorageClass, then the deployment fails. To designate a default StorageClass within your cluster, perform the following task from the deployment jump host. If you have already designated a default StorageClass within your cluster, then you can skip this step.

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

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

```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
```
6.3 Use NVIDIA DeepOps to Deploy Kubeflow

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

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

1. **Deploy Kubeflow in your cluster by following the Kubeflow deployment instructions** on the NVIDIA DeepOps GitHub site.

2. **Note down the Kubeflow Dashboard URL** that the DeepOps Kubeflow deployment tool outputs.

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.**

```
$ ./scripts/k8s_deploy_kubeflow.sh

INFO[0007] Applied the configuration Successfully!   

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

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/ml-pipeline-5875b9db95-g8t2k</td>
<td>1/1 Running 0</td>
<td>91s</td>
<td></td>
</tr>
<tr>
<td>pod/ml-pipeline-persistenceagent-b969dd46-bt9r9</td>
<td>1/1 Running 0</td>
<td>90s</td>
<td></td>
</tr>
<tr>
<td>pod/ml-pipeline-scheduledworkflow-7b8d756c76-7x56s</td>
<td>1/1 Running 0</td>
<td>90s</td>
<td></td>
</tr>
<tr>
<td>pod/ml-pipeline-ui-79ff9d076-fcwqpd</td>
<td>1/1 Running 0</td>
<td>90s</td>
<td></td>
</tr>
<tr>
<td>pod/ml-pipeline-viewer-controller-deployment-5fdcb58-b2t9r</td>
<td>1/1 Running 0</td>
<td>90s</td>
<td></td>
</tr>
<tr>
<td>pod/mysql-657f87857d-15k9z</td>
<td>1/1 Running 0</td>
<td>91s</td>
<td></td>
</tr>
<tr>
<td>pod/notebook-controller-deployment-56b4f59b9f-8bvnrr</td>
<td>1/1 Running 0</td>
<td>92s</td>
<td></td>
</tr>
<tr>
<td>pod/profiles-deployment-6bc745947-mrdkh</td>
<td>2/2 Running 0</td>
<td>90s</td>
<td></td>
</tr>
<tr>
<td>pod/pytorch-operator-77c9f4879-hmrlyv</td>
<td>1/1 Running 0</td>
<td>92s</td>
<td></td>
</tr>
<tr>
<td>pod/seldon-operator-controller-manager-0</td>
<td>1/1 Running 1</td>
<td>91s</td>
<td></td>
</tr>
<tr>
<td>pod/spartakus-volunteer-5fdff7b779q-7qkm</td>
<td>1/1 Running 0</td>
<td>92s</td>
<td></td>
</tr>
<tr>
<td>pod/tensorboard-654478d94-nh8b2</td>
<td>1/1 Running 0</td>
<td>92s</td>
<td></td>
</tr>
<tr>
<td>pod/tf-job-dashboard-56f79c59dd-6w59t</td>
<td>1/1 Running 0</td>
<td>92s</td>
<td></td>
</tr>
<tr>
<td>pod/tf-job-operator-79cbfd6dbc-rrb58c</td>
<td>1/1 Running 0</td>
<td>91s</td>
<td></td>
</tr>
<tr>
<td>pod/workflow-controller-db644d554-cwrbnc</td>
<td>1/1 Running 0</td>
<td>91s</td>
<td></td>
</tr>
</tbody>
</table>

NAME | PORT(S) | TYPE | CLUSTER-IP | EXTERNAL-IP |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>service/admission-webhook-service</td>
<td>443/TCP</td>
<td>97s</td>
<td>ClusterIP</td>
<td>10.233.51.169</td>
</tr>
<tr>
<td>service/application-controller-service</td>
<td>443/TCP</td>
<td>98s</td>
<td>ClusterIP</td>
<td>10.233.4.54</td>
</tr>
<tr>
<td>service/argo-ui</td>
<td>80:3199/TCP</td>
<td>97s</td>
<td>NodePort</td>
<td>10.233.47.191</td>
</tr>
<tr>
<td>service/centraldashboard</td>
<td>80/TCP</td>
<td>97a</td>
<td>ClusterIP</td>
<td>10.233.8.36</td>
</tr>
<tr>
<td>service/jupyter-web-app-service</td>
<td>80/TCP</td>
<td>97a</td>
<td>ClusterIP</td>
<td>10.233.1.42</td>
</tr>
<tr>
<td>service/katib-controller</td>
<td>443/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.25.226</td>
</tr>
<tr>
<td>service/katib-db</td>
<td>3306/TCP</td>
<td>97a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/katib-manager</td>
<td>6789/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.46.239</td>
</tr>
<tr>
<td>service/katib-manager-rest</td>
<td>80/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.55.32</td>
</tr>
<tr>
<td>service/katib-suggestion-bayesianoptimization</td>
<td>6789/TCP</td>
<td>95a</td>
<td>ClusterIP</td>
<td>10.233.49.191</td>
</tr>
<tr>
<td>service/katib-suggestion-grid</td>
<td>6789/TCP</td>
<td>95a</td>
<td>ClusterIP</td>
<td>10.233.46.239</td>
</tr>
<tr>
<td>service/katib-suggestion-hyperband</td>
<td>6789/TCP</td>
<td>95a</td>
<td>ClusterIP</td>
<td>10.233.55.32</td>
</tr>
<tr>
<td>service/katib-suggestion-naa</td>
<td>6789/TCP</td>
<td>95a</td>
<td>ClusterIP</td>
<td>10.233.9.105</td>
</tr>
<tr>
<td>service/katib-suggestion-random</td>
<td>6789/TCP</td>
<td>95a</td>
<td>ClusterIP</td>
<td>10.233.6.116</td>
</tr>
<tr>
<td>service/katib-ui</td>
<td>80/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/metadata-db</td>
<td>3306/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/metadata-service</td>
<td>8080/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/metadata-ui</td>
<td>80/TCP</td>
<td>96a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/minio-service</td>
<td>9000/TCP</td>
<td>94s</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/ml-pipeline</td>
<td>8888/TCP</td>
<td>94s</td>
<td>ClusterIP</td>
<td>10.233.41.201</td>
</tr>
<tr>
<td>service/ml-pipeline-tensorboard-ui</td>
<td>80/TCP</td>
<td>93a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/ml-pipeline-ui</td>
<td>80/TCP</td>
<td>93a</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/mysql</td>
<td>3306/TCP</td>
<td>94s</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/notebook-controller-service</td>
<td>443/TCP</td>
<td>95a</td>
<td>ClusterIP</td>
<td>10.233.41.201</td>
</tr>
<tr>
<td>service/profiles-kfam</td>
<td>8081/TCP</td>
<td>92s</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>service/pytorch-operator</td>
<td>8443/TCP</td>
<td>95s</td>
<td>ClusterIP</td>
<td>10.233.33.151</td>
</tr>
<tr>
<td>NAME</td>
<td>READY</td>
<td>UP-TO-DATE</td>
<td>AVAILABLE</td>
<td>AGE</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------</td>
<td>------------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>deployment.apps/admission-webhook-deployment</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/centraldashboard</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/jupyter-web-app-deployment</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/katib-controller</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-db</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/katib-manager</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-manager-rest</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-bayesianoptimization</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-grid</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-hyperband</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-nasrl</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-random</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>deployment.apps/katib-ui</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/metadata-db</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/metadata-deployment</td>
<td>3/3</td>
<td>3</td>
<td>3</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/metadata-ui</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/argo-ui</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/argo-ui-5dcf5d8b4f</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/centraldashboard-cf4874ddc</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/jupyter-web-app-deployment-685b455447</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/katib-controller-88c97d85c</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-db-8598468fd8</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>97s</td>
</tr>
<tr>
<td>deployment.apps/katib-manager-574c8c6f9</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-manager-rest-778857c98</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-bayesianoptimization</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>96s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-grid-56bf69f597</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>deployment.apps/katib-suggestion-hyperband-7777b76cb9</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>95s</td>
</tr>
<tr>
<td>STORAGECLASS</td>
<td>NAME</td>
<td>STATUS</td>
<td>VOLUME</td>
<td>CAPACITY</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------</td>
<td>--------</td>
<td>-------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>katib-mysql Bound</td>
<td>pvc-b07f293e-d028-11e9-9b9d-00505681a82d</td>
<td></td>
<td>10Gi</td>
<td>RWO</td>
</tr>
<tr>
<td>ontap-ai-flexvols-retain</td>
<td>pvc-b0f3f032-d028-11e9-9b9d-00505681a82d</td>
<td></td>
<td>10Gi</td>
<td>RWO</td>
</tr>
<tr>
<td>metadata-mysql Bound</td>
<td>pvc-b22727ee-d028-11e9-9b9d-00505681a82d</td>
<td></td>
<td>20Gi</td>
<td>RWO</td>
</tr>
<tr>
<td>mysql-pv-claim Bound</td>
<td>pvc-b2429af-d028-11e9-9b9d-00505681a82d</td>
<td></td>
<td>20Gi</td>
<td>RWO</td>
</tr>
<tr>
<td>ontap-ai-flexvols-retain</td>
<td>pvc-b2429af-d028-11e9-9b9d-00505681a82d</td>
<td></td>
<td>20Gi</td>
<td>RWO</td>
</tr>
</tbody>
</table>

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

    Note: The default username is `admin@kubeflow.org`, and the default password is `12341234`. To create additional users, follow the instructions in the [official Kubeflow documentation](https://kubeflow.org/docs).
7 Example Kubeflow Operations and Tasks

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

7.1 Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use

Kubeflow is capable of rapidly provisioning new Jupyter Notebook servers to act as data scientist workspaces. To provision a new Jupyter Notebook server with Kubeflow, perform the following tasks. For more information about Jupyter Notebooks within the Kubeflow context, see the official Kubeflow documentation.

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

The example commands that follow show the importing of an existing volume named `pb_fg_all` into the `kubeflow-anonymous` namespace. These commands create a PVC in the `kubeflow-anonymous` namespace that is tied to the volume on the NetApp storage system. For more
information about PVCs, see the official Kubernetes documentation. For more information about the volume import functionality, see the Trident documentation. For a detailed example showing the importing of a volume using Trident, see Section 8.1.

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

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

$ tridentctl import volume ontap-ai-flexgroups-iface1 pb_fg_all -f ./pvc-import_pb_fg_all-kubeflow-anonymous.yaml -n trident
```

<table>
<thead>
<tr>
<th>BACKEND UUID</th>
<th>NAME</th>
<th>SIZE</th>
<th>STORAGE CLASS</th>
<th>PROTOCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvc-1ed071be-d5a6-11e9-8278-00505681feb6</td>
<td>10 TiB</td>
<td>ontap-ai-flexgroups-retain</td>
<td>file</td>
<td></td>
</tr>
</tbody>
</table>

```
$ kubectl get pvc -n kubeflow-anonymous
NAME        STATUS   VOLUME CAPACITY   ACCESS MODES STORAGECLASS   AGE
pb-fg-all    Bound   pvc-1ed071be-d5a6-11e9-8278-00505681feb6 10Ti   ROX ontap-ai-flexgroups-retain 14s
```

2. From the Kubeflow central dashboard, click Notebook Servers in the main menu to navigate to the Jupyter Notebook server administration page.
3. Click New Server to provision a new Jupyter Notebook server.
4. Give your new server a name, choose the Docker image that you want your server to be based on, and specify the amount of CPU and RAM to be reserved by your server. If the Namespace field is blank, use the Select Namespace menu in the page header to choose a namespace. The Namespace field is then auto-populated with the chosen namespace.

In the following example, the kubeflow-anonymous namespace is chosen. In addition, the default values for Docker image, CPU, and RAM are accepted.
5. Specify the workspace volume details. If you choose to create a new volume, then that volume or PVC is provisioned using the default StorageClass. Because a StorageClass utilizing Trident was designated as the default StorageClass in section 6.2, the volume or PVC is provisioned with Trident. This volume is automatically mounted as the default workspace within the Jupyter Notebook Server container. Any notebooks that a user creates on the server that are not saved to a separate data volume are automatically saved to this workspace volume. Therefore, the notebooks are persistent across reboots.

6. Add data volumes. The following example specifies the existing volume that was imported by the example commands in step 1 and accepts the default mount point.
7. **Optional**: Request that the desired number of GPUs be allocated to your notebook server. In the following example, one GPU is requested.

8. Click Launch to provision your new notebook server.

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

11. Confirm that the dataset volume that was specified in step 6 is mounted on the server. Note that this volume is mounted within the default workspace by default. From the perspective of the user, this is just another folder within the workspace. The user, who is likely a data scientist and not an infrastructure expert, does not need to possess any storage expertise in order to use this volume.
12. Open a Terminal and, assuming that a new volume was requested in step 5, execute `df -h` to confirm that a new Trident-provisioned persistent volume is mounted as the default workspace.

**Note:** The default workspace directory is the base directory that you are presented with when you first access the server’s web interface. Therefore, any artifacts that the user creates using the web interface are stored on this Trident-provisioned persistent volume.
13. Using the terminal, run `nvidia-smi` to confirm that the correct number of GPUs were allocated to the notebook server. In the following example, one GPU has been allocated to the notebook server as requested in step 7.
7.2 Create a Snapshot of an ONTAP Volume from Within a Jupyter Notebook

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

1. Connect to a Jupyter Notebook server’s web interface. See section 7.1 for instructions on how to provision a Jupyter Notebook Server.

2. Open an existing Python 3 notebook or create a new Python 3 notebook. The following example shows the creation of a new Python 3 notebook.

3. Add the following content to the Notebook, update variable values as stated in the comments, and then run all cells.
Create NetApp Snapshot within Jupyter Notebook

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

Install netapp_ontap module

```
In [1]: pip install --user netapp_ontap
```

```
Requirement already satisfied: netapp_ontap in /usr/local/lib/python3.6/site-packages (9.7.0)
Requirement already satisfied: requests>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from netapp_ontap) (2.22.0)
Requirement already satisfied: marshmallow>=3.2.1 in /usr/local/lib/python3.6/site-packages (from netapp_ontap) (3.4.0)
Requirement already satisfied: urllib3>=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests>=2.21.0->netapp_ontap) (1.24.3)
Requirement already satisfied: idna<=2.9,>2.5 in /usr/local/lib/python3.6/dist-packages (from requests>=2.21.0->netapp_ontap) (2.6)
Requirement already satisfied: Pygments>=2.1 in /usr/local/lib/python3.6/dist-packages (from requests>=2.21.0->netapp_ontap) (2.4)
WARNING: You are using pip version 19.2.3, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Import needed functions/classes

```
In [2]: from netapp_ontap import config as netappConfig
from netapp_ontap.host_connection import HostConnection as NetAppHostConnection
from netapp_ontap.resources import Volume, Snapshot
from datetime import import datetime
import json
```
Configure connection to ONTAP cluster/instance

```python
In [3]:
    # Enter connection details for your ONTAP cluster/instance
    ontapClusterMgmtHostname = '10.61.188.40'
    ontapClusterAdminUsername = 'admin'
    ontapClusterAdminPassword = 'NetApp!23'
    verifySSLCert = False
    
    netappConfig.CONNECTION = NetAppHostConnection(
        host = ontapClusterMgmtHostname,
        username = ontapClusterAdminUsername,
        password = ontapClusterAdminPassword,
        verify = verifySSLCert)
```

Convert pv name to ONTAP volume name

```python
In [4]:
    # Enter the name of pv for which you are creating a snapshot
    # Note: To get the name of the pv, you can run `kubectl -n <namespace> get pvc`.
    # The name of the pv that corresponds to a given pvc can be found in the 'VOLUME' column.
    pvName = 'pvc-67213778-6f53-4d9d-96e3-72b0f9bbead4'
    
    # The following will not work if you specified a custom storagePrefix when creating your
    # Trident backend. If you specified a custom storagePrefix, you will need to update this
    # code to match your prefix.
    volumeName = trident_pvc % pvName.replace('=', '_')
    print('pv name: %s, pvName: %s, volumeName: %s' % (pvName, pvName, volumeName))
```

Create snapshot

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

API Response:

```
{  
    "uuid": "ea754776-49ba-11ea-8196-d039ea06490a",
    "description": "POST /api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f.snapshots/?name=jupyter_20200207_200323",
    "state": "success",
    "message": "success",
    "code": 0,
    "start_time": "2020-02-07T15:02:53+00:00",
    "end_time": "2020-02-07T15:02:53+00:00",
    "links": {
        "self": {
            "href": "/api/cluster/jobs/ea754776-49ba-11ea-8196-d039ea06490a"
        }
    }
}```
Optional: Retrieve details for newly created snapshot

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

```json
{
  "svms": {
    "uuid": "e6121692-3224-11ea-8196-d039ea06490a",
    "name": "ai221_data",
    "_links": {
      "self": {
        "href": "/api/svm/svms/e6121692-3224-11ea-8196-d039ea06490a"
      }
    },
    "uuid": "899f336d-166a-426a-aa7-a349764b6cc",
    "volume": {
      "uuid": "14a07f2d-468e-11ea-808d-d039ea06439f",
      "name": "trident_pvc_67213778_6f53_4d9d_96e3_72b0f9bbead4",
      "_links": {
        "self": {
          "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f"
        }
      },
      "_links": {
        "self": {
          "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/899f336d-166a-426a-aa7-a349764b6cc"
        }
      },
      "create_time": "2020-02-07T15:02:53+00:00",
      "comment": "Snapshot created from within a Jupyter Notebook",
      "name": "jupyter_20200207_200323"
    }
  }
}
```
Optional: Retrieve a list of all snapshots that exist for the volume

```python
In [7]:
numVolumeSnapshots = 0
for volumeSnapshot in snapshot.get_collection(volume.uuid, max_records = 256) :
    numVolumeSnapshots += 1
    volumeSnapshot.get()
    print("Snapshot #{}: \n".format(numVolumeSnapshots))
    print(json.dumps(volumeSnapshot.to_dict(), indent=2), \n"
)

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

Snapshot #1:
{
    "svm": {
        "uuid": "e6121682-3224-11ea-8196-d039ea06490a",
        "name": "ai221_data",
        "links": {
            "self": {
                "href": "/api/svm/svms/e6121682-3224-11ea-8196-d039ea06490a"
            }
        },
        "uuid": "87698310-8488-4746-8bca-50f87d79e034",
        "volume": {
            "uuid": "14a07f2d-468e-11ea-808d-d039ea06439f",
            "name": "trident_pvc_67213778_6f53_4d9d_96e3_72b0f9b0ead4",
            "links": {
                "self": {
                    "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f"
                }
            },
            "links": {
                "self": {
                    "href": "/api/storage/volumes/14a07f2d-468e-11ea-808d-d039ea06439f/snapshots/87698310-8488-4746-8bca-50f87d79e034"
                }
            },
            "create_time": "2020-02-04T16:34:00+00:00",
            "name": "clone_kfp_clone_202002.1"
        }
    }
}
```
7.3 Create a Kubeflow Pipeline to Execute an End-to-End AI Training Workflow with Built-in Traceability and Versioning

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

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

```
$ echo -n 'username' | base64
dXNlc3NhBGU=
$ echo -n 'password' | base64
cGFzc3dvcmQ=
$ cat << EOF > ./secret/ontap
apiVersion: v1
kind: Secret
metadata:
  name: ontap-cluster-mgmt-account
namespace: kubeflow
data:
  username: dXNlc3NhBGU=
  password: cGFzc3dvcmQ=
EOF
$ kubectl create -f ./secret/ontap-cluster-mgmt-account.yaml
secret/ontap-cluster-mgmt-account created
```

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

If your dataset volume is already tied to a PVC in the kubeflow namespace, then you can skip this step. If you do not yet have a dataset volume, then you must provision one and then transfer your data to it. See Section 8.2 for an example showing how to provision a new volume with Trident.

The example commands that follow show the importing of an existing FlexVol volume, named dataset_vol, into the kubeflow namespace. For more information about PVCs, see the official Kubernetes documentation. For more information about the volume import functionality, see the Trident documentation. For a detailed example showing the importing of a volume using Trident, see Section 8.1.

```
$ cat << EOF > ./pvc-import-dataset-vol-kubeflow.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: dataset-vol
  namespace: kubeflow
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ontap-ai-flexvols-retain
EOF
$ tridentctl import volume ontap-ai-flexvols dataset_vol -f ./pvc-import-dataset-vol-kubeflow.yaml -n trident
bash:./pvc-import-dataset-vol-kubeflow.yaml: No such file or directory
```

<table>
<thead>
<tr>
<th>BACKEND UID</th>
<th>NAME</th>
<th>STATE</th>
<th>Managed</th>
<th>STORAGE CLASS</th>
<th>PROTOCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvc-3c70ad14-d88f-11e9-b5e2-00505681f3d9</td>
<td>10 TiB</td>
<td>ontap-ai-flexvols-retain</td>
<td>file</td>
<td>2942d386-afcf-462e-bf89-1d2aa3376a7b</td>
<td>online</td>
</tr>
</tbody>
</table>

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

If your trained model volume is already tied to a PVC in the `kubeflow` namespace, then you can skip this step. If you do not yet have a trained model volume, then you must provision one. See Section 8.2 for an example showing how to provision a new volume with Trident.

The example commands that follow show the importing of an existing FlexVol volume, named `kfp_model_vol`, into the `kubeflow` namespace. For more information about PVCs, see the official Kubernetes documentation. For more information about the volume import functionality, see the Trident documentation. For a detailed example showing the importing of a volume using Trident, see Section 8.1.

```
$ cat << EOF > ./pvc-import-dataset-vol-kubeflow.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: kfp-model-vol
namespace: kubeflow
spec:
  accessModes: 
    - ReadWriteMany
  storageClassName: ontap-ai-flexvols-retain
EOF

$ tridentctl import volume ontap-ai-flexvols kfp_model_vol -f ./pvc-import-kfp-model-vol-kubeflow.yaml -n trident
```

<table>
<thead>
<tr>
<th>BACKEND UUID</th>
<th>NAME</th>
<th>STATE</th>
<th>MANAGED</th>
<th>SIZE</th>
<th>STORAGE CLASS</th>
<th>PROTOCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvc-sc70ad14-d88f-11e9-b5e2-00505681f3d9</td>
<td></td>
<td>Bound</td>
<td></td>
<td>1 TiB</td>
<td>ontap-ai-flexvols-retain</td>
<td>file</td>
</tr>
<tr>
<td>2942d386-afcf-462e-bf89-1d2aa3376a7b</td>
<td></td>
<td>Bound</td>
<td></td>
<td>10 GiB</td>
<td>ontap-ai-flexvols-retain</td>
<td>file</td>
</tr>
</tbody>
</table>

```
$ kubectl get pvc -n kubeflow
NAME                      CAPACITY ACCESS MODES STORAGECLASS STATUS VOLUME
imagenet-benchmark-job-gbiqk-kf presuits Bound pvc-a4e32212-d65c-11e9-a043-00505681a82d 1Gi
kfp-model-vol Bound pvc-b07f293e-d028-11e9-9b9d-00505681a82d 10Gi
metadata-mysql Bound pvc-b0f3f032-d028-11e9-9b9d-00505681a82d 10Gi
minio-pv-claim Bound pvc-b2272ee-d028-11e9-9b9d-00505681a82d 20Gi
mysql-pv-claim Bound pvc-b2429afdf-d028-11e9-9b9d-00505681a82d 20Gi
```

---

© 2020 NetApp, Inc. All Rights Reserved.
4. If you have not already done so, you must install the Kubeflow Pipelines SDK. See the official Kubeflow documentation for installation instructions.

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

**Run-time parameters:**

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

a. **Optional:** Executes a data prep step.

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

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

c. Executes a training step.

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

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

e. Executes a validation step.

```
$ cat << EOF > ./ai-training-run.py
# Kubeflow Pipeline Definition: AI Training Run

import kfp.dsl as dsl
import kfp.onprem as onprem
import kfp.components as comp
from kubernetes import client as k8s_client

# Define function that triggers the creation of a NetApp snapshot
def netappSnapshot(
    ontapClusterMgmtHostname: str,
    pvName: str,
    verifySSLCert: bool = True
) -> str:
    # Install netapp_ontap package
    import sys, subprocess;
    subprocess.run([sys.executable, '-m', 'pip', 'install', 'netapp_ontap'])

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

    # Retrieve ONTAP cluster admin account details from mounted K8s secrets
    usernameSecret = open('/mnt/secret/username', 'r')
    ontapClusterAdminUsername = usernameSecret.read().strip()

    passwordSecret = open('/mnt/secret/password', 'r')
    ontapClusterAdminPassword = passwordSecret.read().strip()

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

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

    print('Net app volume name: ', volumeName)

EOF
```
# Create snapshot; print API response
volume = Volume.find(name = volumeName)
timestamp = datetime.today().strftime("%Y%m%d_%H%M%S")
snapshot = Snapshot.from_dict(
    'name': 'kfp_%s' % timestamp,
    'comment': 'Snapshot created by a Kubeflow pipeline',
    'volume': volume.to_dict()
)
response = snapshot.post()
print("\nAPI Response:\n")
print(response.http_response.text)

# Retrieve snapshot details
snapshot.get()

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

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

# Convert netappSnapshot function to Kubeflow Pipeline ContainerOp named 'NetappSnapshotOp'
NetappSnapshotOp = comp.func_to_container_op(netappSnapshot, base_image='python:3')

# Define Kubeflow Pipeline
@dsl.pipeline(
    name="AI Training Run",
    description="Template for executing an AI training run with built-in training dataset traceability and trained model versioning")
def ai_training_run(
    # Define variables that the user can set in the pipelines UI; set default values
    ontap_cluster_mgmt_hostname: str = "10.61.188.40",
    ontap_cluster_admin_acct_k8s_secret: str = "ontap-cluster-mgmt-account",
    ontap_api_verify_ssl_cert: bool = True,
    dataset_volume_pvc_existing: str = "dataset-vol",
    dataset_volume_pv_existing: str = "pvc-43b12235-332e-4dc4-a7b8-8ae90d935a12",
    trained_model_volume_pvc_existing: str = "kfp-model-vol",
    trained_model_volume_pv_existing: str = "pvc-236e893b-63b4-40d3-963b-e709b92816b",
    execute_data_prep_step__yes_or_no: str = "yes",
    data_prep_step_container_image: str = "ubuntu:bionic",
    data_prep_step_command: str = "<insert command here>",
    data_prep_step_dataset_volume_mountpoint: str = "/mnt/dataset",
    train_step_container_image: str = "nvcr.io/nvidia/tensorflow:19.12-tf1-py3",
    train_step_command: str = "<insert command here>",
    train_step_dataset_volume_mountpoint: str = "/mnt/dataset",
    train_step_model_volume_mountpoint: str = "pvc-236e893b-63b4-40d3-963b-e709b92816b",
    validation_step_container_image: str = "nvcr.io/nvidia/tensorflow:19.12-tf1-py3",
    validation_step_command: str = "<insert command here>",
    validation_step_dataset_volume_mountpoint: str = "/mnt/dataset",
    validation_step_model_volume_mountpoint: str = "pvc-236e893b-63b4-40d3-963b-e709b92816b"
):
    # Set GPU limits; Due to SDK limitations, this must be hardcoded
    train_step_num_gpu = 0
    validation_step_num_gpu = 0

    # Pipeline Steps:

    # Execute data prep step
    with dsl.Condition(execute_data_prep_step__yes_or_no == "yes") :
        data_prep = dsl.ContainerOp(
            name="data-prep",
            image=data_prep_step_container_image,
            command=["sh", "-c"],
            arguments=[data_prep_step_command]
        )

    # Mount dataset volume/pvc
```python
data_prep.apply(
onprem.mount_pvc(dataset_volume_pvc_existing, 'dataset',
data_prep_step_dataset_volume_mountpoint)
)

# Create a snapshot of the dataset volume/pvc for traceability
dataset_snapshot = NetappSnapshotOp(
    ontap_cluster_mgmt_hostname,
dataset_volume_pv_existing,
ontap_api_verify_ssl_cert
)

# Mount k8s secret containing ONTAP cluster admin account details
dataset_snapshot.add_pvolumes({
    '/mnt/secret': k8s_client.V1Volume(
        name='ontap-cluster-admin',
        secret=k8s_client.V1SecretVolumeSource(
            secret_name=ontap_cluster_admin_acct_k8s_secret
        )
    )
})

# State that snapshot should be created after the data prep job completes
dataset_snapshot.after(data_prep)

# Execute training step
train = dsl.ContainerOp(
    name="train-model",
    image=train_step_container_image,
    command=['sh', '-c'],
aruments=[train_step_command]
)

# Mount dataset volume/pvc
train.apply(
onprem.mount_pvc(dataset_volume_pvc_existing, 'datavol',
train_step_dataset_volume_mountpoint)
)

# Mount model volume/pvc
train.apply(
onprem.mount_pvc(trained_model_volume_pvc_existing, 'modelvol',
train_step_model_volume_mountpoint)
)

# Request that GPUs be allocated to training job pod
if train_step_num_gpu > 0:
    train.set_gpu_limit(train_step_num_gpu, 'nvidia')

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

# Create a snapshot of the model volume/pvc for model versioning
model_snapshot = NetappSnapshotOp(
    ontap_cluster_mgmt_hostname,
    trained_model_volume_pv_existing,
ontap_api_verify_ssl_cert
)

# Mount k8s secret containing ONTAP cluster admin account details
model_snapshot.add_pvolumes({
    '/mnt/secret': k8s_client.V1Volume(
        name='ontap-cluster-admin',
        secret=k8s_client.V1SecretVolumeSource(
            secret_name=ontap_cluster_admin_acct_k8s_secret
        )
    )
})

# State that snapshot should be created after the training job completes
model_snapshot.after(train)

# Execute inference validation job
inference_validation = dsl.ContainerOp(
    name="validate-model",
    image=validation_step_container_image,
    command=['sh', '-c'],
aruments=[validation_step_command]
)
```

# Mount dataset volume/pvc
inference_validation.apply(onprem.mount_pvc(dataset_volume_pvc_existing, 'datavol',
validation_step_dataset_volume_mountpoint))

# Mount model volume/pvc
inference_validation.apply(onprem.mount_pvc(trained_model_volume_pvc_existing, 'modelvol',
validation_step_model_volume_mountpoint))

# Request that GPUs be allocated to pod
if validation_step_num_gpu > 0:
inference_validation.set_gpu_limit(validation_step_num_gpu, 'nvidia')

# State that inference validation job should be executed after model volume snapshot is taken
inference_validation.after(model_snapshot)

if __name__ == "__main__":
    import kfp.compiler as compiler
    compiler.Compiler().compile(ai_training_run, __file__ + "_.yaml")

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

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

7. Click Upload Pipeline to upload your pipeline definition.
8. Choose the `.yaml` archive containing your pipeline definition that you created in step 5, give your pipeline a name, and click Upload.

9. You should now see your new pipeline in the list of pipelines on the pipeline administration page. Click your pipeline’s name to view it.
10. Review your pipeline to confirm that it looks correct.
11. Click Create run to run your pipeline.
12. You are now presented with a screen from which you can start a pipeline run. Create a name for the run, enter a description, choose an experiment to file the run under, and choose whether you want to initiate a one-off run or schedule a recurring run.
13. Define parameters for the run, and then click Start. In the following example, the default values are accepted for most parameters. Details for the volume that was imported into the kubeflow namespace in step 2 are entered for `dataset_volume_pvc_existing` and `dataset_volume_pv_existing`. Details for the volume that was imported into the kubeflow namespace in step 3 are entered for `trained_model_volume_pvc_existing` and `trained_model_volume_pv_existing`. Non-AI-related commands are entered for the `data_prep_step_command`, `train_step_command`, and `validation_step_command` parameters in order to plainly demonstrate the functionality of the pipeline. Note that you defined the default values for the parameters within your pipeline definition (see step 5).
### Run Type
- [ ] One-off
- [ ] Recurring

### Run parameters

Specify parameters required by the pipeline

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ontap_cluster_mgmt_hostname</td>
<td>10.61.198.40</td>
</tr>
<tr>
<td>ontap_cluster_admin_acct_kib_secret</td>
<td></td>
</tr>
<tr>
<td>ontap_cluster_mgmt_account</td>
<td></td>
</tr>
<tr>
<td>ontap_api_verify_ssl_cert</td>
<td>False</td>
</tr>
<tr>
<td>dataset_volume_pvc_existing</td>
<td>dataset-vol</td>
</tr>
<tr>
<td>dataset_volume_pv_existing</td>
<td>pvc-43b12235-f32e-4d64-a7b8-88e90d935a12</td>
</tr>
<tr>
<td>trained_model_volume_pvc_existing</td>
<td>kfp-model-vol</td>
</tr>
<tr>
<td>trained_model_volume_pv_existing</td>
<td>pvc-236e893b-63b4-40f3-963b-e709f9b2816b</td>
</tr>
</tbody>
</table>
14. You are now presented with a screen listing all runs that fall under the specific experiment. Click the name of the run that you just started to view it.
15. At this point, the run is likely still in progress.
16. Confirm that the run completed successfully. When the run is complete, every stage of the pipeline shows a green check-mark icon.
17. Click a specific stage, and then click Logs to view output for that stage.
Build commit: ee207f2

Runtime execution graph. Only some active components are shown.
7.4 Create a Kubeflow Pipeline to Rapidly Clone a Dataset for a Data Scientist Workspace

To define and execute a new Kubeflow Pipeline that takes advantage of NetApp FlexClone technology in order to rapidly and efficiently clone a dataset volume in order to create a data scientist or developer workspace, perform the following tasks. For more information about Kubeflow Pipelines, see the official Kubeflow documentation. Note that the example pipeline that is shown in this section only works with volumes that reside on ONTAP storage systems or software-defined instances.

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

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

2. If you have not already done so, you must install the Kubeflow Pipelines SDK. Refer to the official Kubeflow documentation for installation instructions.

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

**Run-time Parameters:**
- `ontap_cluster_mgmt_hostname`: Hostname/IP address of the ONTAP cluster on which your dataset and model volumes are stored.
- `ontap_cluster_admin_acct_k8s_secret`: Name of the Kubernetes secret that was created in step 1.
- `ontap_verify_ssl_cert`: Denotes whether or not to verify your cluster’s SSL certificate when communicating with the ONTAP API (True/False).
- `workspace_name`: Name that you want to give to your new workspace.
- `jupyter_namespace`: Namespace in which you intend to create a Jupyter Notebook workspace. See Section 7.1 for details on creating a Jupyter Notebook workspace. The dataset clone that this pipeline creates will be mountable in the Jupyter Notebook workspace.
- `dataset_volume_pv_existing`: Name of the Kubernetes PersistentVolume (PV) object that corresponds to the dataset volume PVC that is tied to the volume that you wish to clone. To get the name of the PV, you can run `kubectl -n <namespace> get pvc`. The name of the PV that corresponds to a given PVC can be found in the `VOLUME` column. Note that this could be the volume that was imported in Section 7.1, step 1, or it could be a different volume.
- `trident_storage_class`: Kubernetes StorageClass that you wish to use to create this clone. This would generally be a StorageClass that you created in Section 5.
- `trident_namespace`: Namespace that Trident is installed in. Note that, by default, Trident is installed in the `trident` namespace.
- `trident_backend`: Trident Backend that you wish to use to create this clone. This would generally be a Backend that you created in Section 5.

**Pipeline Steps:**
- Triggers the creation of a clone, using NetApp FlexClone technology, of your dataset volume.
- Prints instructions for deploying an interactive Jupyter Notebook workspace that has access to the dataset clone.

```
$ cat << EOF > ./create-data-scientist-workspace.py
# Kubeflow Pipeline Definition: Create Data Scientist Workspace
```
import kfp.dsl as dsl
import kfp.components as comp
from kubernetes import client as k8s_client

# Define function that triggers the creation of a NetApp snapshot
def netappClone(
    ontapClusterMgmtHostname: str,
    sourcePvName: str,
    verifySSLCert: bool = True
) -> str:
    # Install netapp_ontap package
    import sys, subprocess
    subprocess.run([sys.executable, '-m', 'pip', 'install', 'netapp_ontap'])

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

    # Retrieve ONTAP cluster admin account details from mounted K8s secrets
    usernameSecret = open('/mnt/secret/username', 'r')
    ontapClusterAdminUsername = usernameSecret.read().strip()
    passwordSecret = open('/mnt/secret/password', 'r')
    ontapClusterAdminPassword = passwordSecret.read().strip()

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

    # Convert pv name to ONTAP volume name
    sourceVolumeName = 'trident_%s' % sourcePvName.replace('-', '_')

    # The following will not work if you specified a custom storagePrefix when creating your
    # Trident backend. If you specified a custom storagePrefix, you will need to update this
    # code to match your prefix.
    sourceVolumeName = 'kfp_clone_%s' % timestamp
    cloneVolumeName = 'kfp_clone_%s' % timestamp
    cloneVolume = Volume.from_dict(
        name=cloneVolumeName,
        svm=sourceVolume.to_dict()['svm'],
        clone={
            'is_flexclone': 'true',
            'parent_volume': sourceVolume.to_dict()
        },
        nas={
            'path': '/%s' % cloneVolumeName
        }
    )
    response = cloneVolume.post()
    print('
API Response:')
    print(response.http_response.text)

    # Retrieve clone volume details
    cloneVolume.get()

    # Convert clone volume details to JSON string
    cloneVolumeDetails = cloneVolume.to_dict()
    print('
Clone Volume Details:
')
    print(json.dumps(cloneVolumeDetails, indent=2))
# Return name of new clone volume
return cloneVolumeDetails['name']

# Convert netappClone function to Kubeflow Pipeline ContainerOp named 'NetappCloneOp'
NetappCloneOp = comp.func_to_container_op(netappClone, base_image='python:3')

# Define Kubeflow Pipeline
@dsl.pipeline(name="Create Data Scientist Workspace",
description="Template for cloning dataset volume in order to create data scientist/developer workspace")
def create_data_scientist_workspace(
    # Define variables that the user can set in the pipelines UI; set default values
    ontap_cluster_mgmt_hostname: str = "10.61.188.40",
    ontap_cluster_admin_acct_k8s_secret: str = "ontap-cluster-mgmt-account",
    ontap_api_verify_ssl_cert: bool = True,
    workspace_name: str = "dev",
    jupyter_namespace: str = "admin",
    dataset_volume_pv_existing: str = "pvc-db963a53-abf2-4ffa-9c07-8815ce78d506",
    trident_storage_class: str = "ontap-ai-flexvols-retain",
    trident_namespace: str = "trident",
    trident_backend: str = "ontap-ai"
):
    # Pipeline Steps:
    # Create a clone of the source dataset volume
    dataset_clone = NetappCloneOp(
        ontap_cluster_mgmt_hostname,
        dataset_volume_pv_existing,
        ontap_api_verify_ssl_cert
    )
    # Mount k8s secret containing ONTAP cluster admin account details
    dataset_clone.add_pvolumes({
        '/mnt/secret': k8s_client.V1Volume(
            name='ontap-cluster-admin',
            secret=k8s_client.V1SecretVolumeSource(
                secret_name=ontap_cluster_admin_acct_k8s_secret
            )
        )
    })
    # Retrieve clone volume name from op output
    clone_volume_Name = dataset_clone.output
    # Convert clone volume name to allowed pvc name (for user instructions)
    workspace_pvc_name = 'dataset-workspace' + str(workspace_name)
    # Define user instructions
    user_instructions = '''
1) Execute the following commands against your Kubernetes cluster:

    cat << EOF > import-pvc-pipeline-clone.yaml
    kind: PersistentVolumeClaim
    apiVersion: v1
    metadata:
      name: %s
      namespace: %s
    spec:
      accessModes:
      - ReadWriteMany
      storageClassName: %s
    EOF
    tridentctl -n %s import volume %s %s -f ./import-pvc-pipeline-clone.yaml

2) From Kubeflow "Notebook Servers" dashboard, provision a new Jupyter workspace in namespace, "%s", and mount dataset pvc, "%s", """ % (workspace_pvc_name, jupyter_namespace, trident_storage_class, trident_namespace, trident_backend, clone_volume_Name, jupyter_namespace, workspace_pvc_name)
# Print instructions for deploying an interactive workspace
print_instructions = dsl.ContainerOp(
    name="print-instructions",
    image="ubuntu:bionic",
    command=["sh", "-c"],
    arguments=["echo '%s' % user_instructions"]
)
# State that instructions should be printed after clone is created
print_instructions.after(dataset_clone)

if __name__ == '__main__':
    import kfp.compiler as compiler
    compiler.Compiler().compile(create_data_scientist_workspace, __file__ + '.yaml')

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

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

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

7. You should now see your new pipeline in the list of pipelines on the pipeline administration page. Click your pipeline’s name to view it.
8. Review your pipeline to confirm that it looks correct.

9. Click Create run to run your pipeline.
10. You are now presented with a screen from which you can start a pipeline run. Create a name for the run, enter a description, choose an experiment to file the run under, and choose whether you want to initiate a one-off run or schedule a recurring run.
11. Define parameters for the run, and then click Start. In the following example, the default values are accepted for most parameters. The name of an already-existing PV is entered for `dataset_volume_pv_existing`. The value, `admin`, is entered for `jupyter_namespace` as this is the namespace that we intend to provision a new Jupyter Notebook workspace in. Note that you defined the default values for the parameters within your pipeline definition (see step 3).
12. You are now presented with a screen listing all runs that fall under the specific experiment. Click the name of the run that you just started to view it.
13. At this point, the run is likely still in progress.
14. Confirm that the run completed successfully. When the run is complete, every stage of the pipeline shows a green check-mark icon.
15. Click the netappclone stage, and then click Logs to view output for that stage.
16. Click the **print-instructions** stage, and then click Logs to view the outputted instructions. See Section 7.1 for details on creating a Jupyter Notebook workspace.
Create a Kubeflow Pipeline to Trigger a SnapMirror Volume Replication Update

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

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

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

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

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

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

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

```bash
$ echo -n 'username' | base64
dXNlc3NhbWU=
$ echo -n 'password' | base64
cGFzc3dvcmQ=
$ cat << EOF > ./secret
apiVersion: v1
kind: Secret
metadata:
  name: ontap-cluster-mgmt-account
namespace: kubeflow
data:
  username: dXNlc3NhbWU=
EOF
```
2. If you have not already done so, install the Kubeflow Pipelines SDK. See the official Kubeflow documentation for installation instructions.

3. Define your Kubeflow pipeline in Python using the Kubeflow Pipelines SDK.

   **Pipeline Steps:**
   a. Trigger a replication update for the specified asynchronous SnapMirror relationship.

   The example commands below show the creation of a pipeline definition for a pipeline that accepts the following parameters at run-time. Modify the pipeline definition as needed depending on your specific process.

   **Run-time Parameters:**
   - `ontap_cluster_mgmt_hostname`: Hostname or IP address of the ONTAP cluster on which the destination volume resides.
   - `ontap_cluster_admin_acct_k8s_secret`: Name of the Kubernetes secret that was created in step 1.
   - `ontap_api_verify_ssl_cert`: Denotes whether to verify your cluster’s SSL certificate when communicating with the ONTAP API (yes/no).
   - `source_svm`: Name of the SVM on which the source volume resides.
   - `source_volume`: Name of the source volume (the volume that you are replicating from) on the source cluster.
   - `destination_svm`: Name of the SVM on which the destination volume resides.
   - `destination_volume`: Name of the destination volume (the volume that you are replicating to) on the destination cluster.

```bash
$ cat << EOF > ./replicate-data-snapmirror.py
# Kubeflow Pipeline Definition: Replicate data - SnapMirror

import kfp.dsl as dsl
import kfp.components as comp
from kubernetes import client as k8s_client

# Define function that triggers the creation of a NetApp snapshot
def netappSnapMirrorUpdate(
    ontapClusterMgmtHostname: str,
    sourceSvm: str,
    sourceVolume: str,
    destinationSvm: str,
    destinationVolume: str,
    verifySSLCert: str = 'no'
) -> int:
    # Install ansible package
    import sys, subprocess
    print("Installing required Python modules:\n")
    subprocess.run([sys.executable, '-m', 'pip', 'install', 'ansible', 'netapp-lib'])

    # Retrieve ONTAP cluster admin account details from mounted K8s secrets
    usernameSecret = open('/mnt/secret/username', 'r')
    ontapClusterAdminUsername = usernameSecret.read().strip()
    passwordSecret = open('/mnt/secret/password', 'r')
    ontapClusterAdminPassword = passwordSecret.read().strip()

    # Define Ansible playbook for triggering SnapMirror update
    snapMirrorPlaybookContent = ""
    ---
    - name: "Trigger SnapMirror Update"
      hosts: localhost
      tasks:
```

```bash
kubectl create -f ./secret-ontap-cluster-mgmt-account.yaml
secret/ontap-cluster-mgmt-account created
```
- name: update snapmirror
  na_ontap_snapmirror:
    state: present
    source_path: '%s:%s'
    destination_path: '%s:%s'
    hostname: '%s'
    username: '%s'
    password: '%s'
    https: 'yes'
    validate_certs: '%s'
  
print("Creating Ansible playbook:")
print(snapMirrorPlaybookContent, ")

snapMirrorPlaybookFile = open("/root/snapmirror-update.yaml", "w")
snapMirrorPlaybookFile.write(snapMirrorPlaybookContent)
snapMirrorPlaybookFile.close()

# Trigger SnapMirror update
print("Executing Ansible playbook to trigger SnapMirror update:
")
try:
    subprocess.run(['ansible-playbook', '/root/snapmirror-update.yaml'])
except Exception as e:
    print(str(e).strip())
    raise

# Return success code
return 0

# Convert netappSnapMirrorUpdate function to Kubeflow Pipeline ContainerOp named 'NetappSnapMirrorUpdateOp'
NetappSnapMirrorUpdateOp = comp.func_to_container_op(netappSnapMirrorUpdate, base_image='python:3')

# Define Kubeflow Pipeline
@dsl.pipeline(
    name="Replicate Data",
    description="Template for triggering a NetApp SnapMirror update in order to replicate data across environments"
)
def replicate_data(
    # Define variables that the user can set in the pipelines UI; set default values
    ontap_cluster_mgmt_hostname: str = "10.61.188.40",
    ontap_cluster_admin_acct_k8s_secret: str = "ontap-cluster-mgmt-account",
    ontap_api_verify_ssl_cert: str = "yes",
    source_svm: str = "ailab",
    destination_svm: str = "ai221_data",
    destination_volume: str = "sm_dest"
):
    # Pipeline Steps:
    # Trigger SnapMirror replication
    replicate = NetappSnapMirrorUpdateOp(
        ontap_cluster_mgmt_hostname,
        source_svm,
        destination_svm,
        destination_volume,
        ontap_api_verify_ssl_cert
    )
    # Mount k8s secret containing ONTAP cluster admin account details
    replicate.add_pvolumes({
        '/mnt/secret': k8s_client.V1Volume(
            name='ontap-cluster-admin',
            secret=k8s_client.V1SecretVolumeSource(
                secret_name='ontap_cluster_admin_acct_k8s_secret
            )
        )
    })
if __name__ == '__main__' :
    import kfp.compiler as compiler
    compiler.Compiler().compile(replicate_data, __file__ + '.yaml')
EOF

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

4. Follow steps 6 through 17 from section 7.3 in this document.
   Be sure to use the pipeline definition that was created in the previous step (step 3) of this section instead of the pipeline definition that was created in section 7.3.

8 Example Trident Operations

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

8.1 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 section 5.3, 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 section 5.3, step 1. For more information about PVCs, see the official Kubernetes documentation. For more information about the volume import functionality, see the Trident documentation.

Note: An accessModes value of ReadOnlyMany is specified in the example PVC spec files. This value means that multiple pods can mount these volumes at the same time and that access will be read-only. For more information about the accessMode field, see the official Kubernetes documentation.

Note: The Backend names that are specified in the following example import commands are highlighted for reference. These names correspond to the Backends that were created in the example in section 5.3, step 1.

Note: The StorageClass names that are specified in the following example PVC definition files are highlighted for reference. These names correspond to the StorageClasses that were created in the example in section 5.4, step 1.

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

$ tridentctl import volume ontap-ai-flexgroups-ifacel pb_fg_all -f ./pvc-import-pb_fg_all-ifacel.yaml -n trident
+------------------------------------+-------------------+---------------------------------+-------------------+
<table>
<thead>
<tr>
<th>BACKEND UUID</th>
<th>NAME</th>
<th>SIZE</th>
<th>STORAGE CLASS</th>
<th>PROTOCOL</th>
</tr>
</thead>
</table>
+------------------------------------+-------------------+---------------------------------+-------------------+-------------------+

8.2 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 section 5.4, step 2.

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

```
$ cat << EOF > ./pvc-tensorflow-results.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tensorflow-results
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
```

9 Example High-performance Jobs for ONTAP AI Deployments

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

9.1 Execute a Single-Node AI Workload

To execute a single-node AI and ML job in your Kubernetes cluster, perform the following tasks from the deployment jump host. With Trident, you can quickly and easily make a data volume, potentially containing petabytes of data, accessible to a Kubernetes workload. To make such a data volume accessible from within a Kubernetes pod, simply specify a PVC, such as one of the PVCs that was created in the example in section 8.1, in the pod definition. This step is a Kubernetes-native operation; no NetApp expertise is required.

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

1. The following example commands show the creation of a Kubernetes job for a TensorFlow benchmark workload that uses the ImageNet dataset. For more information about the ImageNet dataset, see the ImageNet website.

   ```
   $ kubectl create -f ./pvc-tensorflow-results.yaml
   persistentvolumeclaim/tensorflow-results created
   $ kubectl get pvc
   
   NAME                              STATUS    VO Access Modes StorageClass
   $pb-fg-all-iface1  Bound      default-pb-fg-all-iface1-7d9f1  ontap-ai-flexvols-retain
   $pb-fg-all-iface2  Bound      default-pb-fg-all-iface2-85ae  ontap-ai-flexvols-retain
   tensorflow-results  Bound      default-tensorflow-results-2fd60  ontap-ai-flexvols-retain
   RWX  Bound      default-tensorflow-results-2fd60  ontap-ai-flexvols-retain
   ``

   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.

© 2020 NetApp, Inc. All Rights Reserved.
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
metadata:
  name: 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: tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command:
          
```
```
--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 pods
```
NAME                                                   READY   STATUS      RESTARTS   AGE
netapp-tensorflow-single-imagenet                      1/1     Running     0          3m
```

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
netapp-tensorflow-single-imagenet-m7x92 1/1     Running     0          3m
```

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                      1/1     Running     0          3m
```

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

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

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

9.2 **Execute a Synchronous Distributed AI Workload**

To execute a synchronous multinode AI and ML job in your Kubernetes cluster, perform the following tasks on the deployment jump host. This process enables you to take advantage of data that is stored on a NetApp volume and to use more GPUs than a single worker node can provide. See Figure 8 for a visualization.

**Note:** Synchronous distributed jobs can help increase performance and training accuracy compared with asynchronous distributed jobs. A discussion of the pros and cons of synchronous jobs versus asynchronous jobs is outside the scope of this document.
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 section 9.1. 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-ifacel
EOF
```
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>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>60s</td>
</tr>
<tr>
<td>10.61.218.154</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 section 9.1.

This example master job requests eight GPUs and thus can run on a single GPU worker node that features eight or more GPUs. If your GPU worker nodes feature more than eight GPUs, to maximize performance, you might want to increase this number to be equal to the number of GPUs that your worker nodes feature.

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

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-master.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-multi-imagenet-master
spec:
  backoffLimit: 5
  template:
    spec:
      hostNetwork: true
EOF
```
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
netapp-tensorflow-multi- imagenet-master-ppwwj 1/1     Running   0          45s
10.61.218.152 10.61.218.152 10.61.218.154
netapp-tensorflow-multi- imagenet-worker-654fc7f486-v6725 1/1     Running   0          26s
10.61.218.154

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
rm: cannot remove ‘/lib’: Is a directory
[10.61.218.154:00033] PMIX ERROR: NO_PERMISSIONS in file gda_dstore.c at line 702
[10.61.218.154:00033] PMIX ERROR: NO_PERMISSIONS in file gda_dstore.c at line 711
6. **Delete the worker deployment when you no longer need it.** The following example commands show the deletion of the worker deployment object that was created in step 1.

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

```
$ kubectl get deployments
NAME                                      DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
netapp-tensorflow-multi-imagenet-worker 1 1 1 1 43m

$ kubectl get pods
NAME                                      READY STATUS RESTARTS AGE
netapp-tensorflow-multi-imagenet-master-ppwwj 0/1 Completed 0 17m
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725 1/1 Running 0 43m
```

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

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

```
$ kubectl get pods
NAME                                      READY STATUS RESTARTS AGE
netapp-tensorflow-multi-imagenet-master-ppwwj 0/1 Completed 0 19m
```

---

© 2020 NetApp, Inc. All Rights Reserved.

---

[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 pmi ob1 -mca btl "openib -mca btl_tcp_if_include enp1s0f0 -x NCCL_IB_HCA=mlx5 -x NCCL_NET_GDR_READ=1 -x NCCL_IB_SHM几-3 -x NCCL_IB_GID_INDEX=3 -x NCCL_SOCKET_IFNAME=eth5,eth6,eth7,eth8,eth9,eth10,eth11,eth12,eth13,eth14,eth15,eth16,eth17,eth18,eth19,eth20,eth21,eth22 -mca orte_base_help_aggregate 0 -mca plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" python /netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=resnet50 --batch_size=256 --device=gpu --force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48 --variable_update=horovod --batch_group_size=20 --num_batches=500 --nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True --use_tf_layers=False --data_name=imagenet --use_datasets=True --data_dir=/mnt/mount_0/dataset/imagenet --datasets_parallel_interleave_cycle_length=10 --datasets_sloppy_parallel_interleave=False --num_mounts=2 --mount_prefix=/mnt/mount_0 --datasets_prefetch_buffer_size=2000 --datasets_use_prefetch=True --datasets_num_private_threads=4 --horovod_device=gpu > /tmp/20190814_161609_tensorflow_horovod_rdma_resnet50_gpu_16_256_b500_imagenet_nodistort_fp16_r10_m2_nockpt.txt 2>&1

---

NetApp AI Control Plane

© 2020 NetApp, Inc. All Rights Reserved.

---
10 Performance Testing

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

Table 3) Performance comparison results.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Docker Run (images/sec)</th>
<th>Kubernetes (images/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-node TensorFlow</td>
<td>Synthetic data</td>
<td>6,667.2475</td>
<td>6,661.93125</td>
</tr>
<tr>
<td>Single-node TensorFlow</td>
<td>ImageNet</td>
<td>6,570.2025</td>
<td>6,530.59125</td>
</tr>
<tr>
<td>Synchronous distributed two-node TensorFlow</td>
<td>Synthetic data</td>
<td>13,213.70625</td>
<td>13,218.288125</td>
</tr>
<tr>
<td>Synchronous distributed two-node TensorFlow</td>
<td>ImageNet</td>
<td>12,941.69125</td>
<td>12,881.33875</td>
</tr>
</tbody>
</table>

11 Conclusion

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

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

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

Acknowledgments

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

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

- **NVIDIA DGX-1 servers:**
  - NVIDIA DGX-1 servers
  - NVIDIA Tesla V100 Tensor Core GPU
  - NVIDIA GPU Cloud (NGC)

- **NetApp AFF systems:**
  - AFF datasheet
  - NetApp FlashAdvantage for AFF
  - ONTAP 9.x documentation
  - NetApp FlexGroup technical report

- **NetApp persistent storage for containers:**
  - NetApp Trident
    https://netapp.io/persistent-storage-provisioner-for-kubernetes/

- **NetApp Interoperability Matrix:**
  - NetApp Interoperability Matrix Tool
    http://support.netapp.com/matrix

- **ONTAP AI networking:**
  - Cisco Nexus 3232C Switches
  - Mellanox Spectrum 2000 series switches

- **ML framework and tools:**
  - DALI
    https://github.com/NVIDIA/DALI
    https://www.tensorflow.org/
    https://eng.uber.com/horovod/
  - Enabling GPUs in the Container Runtime Ecosystem
    https://devblogs.nvidia.com/gpu-containers-runtime/
  - Docker
    https://docs.docker.com
  - Kubernetes
    https://kubernetes.io/docs/home/
  - NVIDIA DeepOps
    https://github.com/NVIDIA/deepops
  - Kubeflow
    http://www.kubeflow.org/
- Jupyter Notebook Server

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

### Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Document Version History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1.0</td>
<td>September 2019</td>
<td>Initial release.</td>
</tr>
<tr>
<td>Version 2.0</td>
<td>September 2019</td>
<td>Added sections on triggering Snapshot copies/FlexClone volumes using kubectl commands (removed from document in version 3.0); added section on Kubeflow (sections 2.6 and 6.); added Figure 8; and updated DeepOps troubleshooting instructions (section 4.2).</td>
</tr>
<tr>
<td>Version 3.0</td>
<td>March 2020</td>
<td>Added section on creating a Snapshot from within a Jupyter Notebook (section 7.2); added example Kubeflow pipelines (sections 7.3 and 7.4); added NetApp Snapshot copies and NetApp FlexClone technology descriptions to the Concepts and Components section (section 2); and reordered sections within document; and removed sections on triggering Snapshot copies/FlexClone volumes using kubectl commands (due to Kubernetes API changes).</td>
</tr>
<tr>
<td>Version 4.0</td>
<td>May 2020</td>
<td>Added example Kubeflow pipeline (section 7.5); added NetApp SnapMirror technology description (section 2.10); and updated Abstract and Introduction.</td>
</tr>
</tbody>
</table>
Refer to the **Interoperability Matrix Tool (IMT)** on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer’s installation in accordance with published specifications.

**Copyright Information**

Copyright © 2019 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

**THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Data contained herein pertains to a commercial item (as defined in FAR 2.101) and is proprietary to NetApp, Inc. The U.S. Government has a non-exclusive, non-transferable, non-sublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b).

**Trademark Information**

NETAPP, the NETAPP logo, and the marks listed at [http://www.netapp.com/TM](http://www.netapp.com/TM) are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.

TR-4798 -0919