Use Cases

NetApp Solutions

NetApp
August 03, 2021

Table of Contents

Use Cases

TR-4896: Distributed training in Azure: Lane detection - Solution design

Muneer Ahmad and Verron Martina, NetApp
Ronen Dar, RUN:AI

Since May 2019, Microsoft delivers an Azure native, first-party portal service for enterprise NFS and SMB file services based on NetApp ONTAP technology. This development is driven by a strategic partnership between Microsoft and NetApp and further extends the reach of world-class ONTAP data services to Azure.

NetApp, a leading cloud data services provider, has teamed up with RUN: AI, a company virtualizing AI infrastructure, to allow faster AI experimentation with full GPU utilization. The partnership enables teams to speed up AI by running many experiments in parallel, with fast access to data, and leveraging limitless compute resources. RUN: AI enables full GPU utilization by automating resource allocation, and the proven architecture of Azure NetApp Files enables every experiment to run at maximum speed by eliminating data pipeline obstructions.

NetApp and RUN: AI have joined forces to offer customers a future-proof platform for their AI journey in Azure. From analytics and high-performance computing (HPC) to autonomous decisions (where customers can optimize their IT investments by only paying for what they need, when they need it), the alliance between NetApp and RUN: AI offers a single unified experience in the Azure Cloud.

Solution overview

In this architecture, the focus is on the most computationally intensive part of the AI or machine learning (ML) distributed training process of lane detection. Lane detection is one of the most important tasks in autonomous driving, which helps to guide vehicles by localization of the lane markings. Static components like lane markings guide the vehicle to drive on the highway interactively and safely.

Convolutional Neural Network (CNN)-based approaches have pushed scene understanding and segmentation to a new level. Although it doesn’t perform well for objects with long structures and regions that could be occluded (for example, poles, shade on the lane, and so on). Spatial Convolutional Neural Network (SCNN) generalizes the CNN to a rich spatial level. It allows information propagation between neurons in the same layer, which makes it best suited for structured objects such as lanes, poles, or truck with occlusions. This compatibility is because the spatial information can be reinforced, and it preserves smoothness and continuity.

Thousands of scene images need to be injected in the system to allow the model learn and distinguish the various components in the dataset. These images include weather, daytime or nighttime, multilane highway roads, and other traffic conditions.

For training, there is a need for good quality and quantity of data. Single GPU or multiple GPUs can take days to weeks to complete the training. Data-distributed training can speed up the process by using multiple and multinode GPUs. Horovod is one such framework that grants distributed training but reading data across clusters of GPUs could act as a hindrance. Azure NetApp Files provides ultrafast, high throughput and sustained low latency to provide scale-out/scale-up capabilities so that GPUs are leveraged to the best of their computational capacity. Our experiments verified that all the GPUs across the cluster are used more than 96% on average for training the lane detection using SCNN.

Target audience

Data science incorporates multiple disciplines in IT and business, therefore multiple personas are part of our targeted audience:

- Data scientists need the flexibility to use the tools and libraries of their choice.
- Data engineers need to know how the data flows and where it resides.
- Autonomous driving use-case experts.
- Cloud administrators and architects to set up and manage cloud (Azure) resources.
- A DevOps engineer needs the tools to integrate new AI/ML applications into their continuous integration and continuous deployment (CI/CD) pipelines.
- Business users want to have access to AI/ML applications.

In this document, we describe how Azure NetApp Files, RUN: AI, and Microsoft Azure help each of these roles bring value to business.

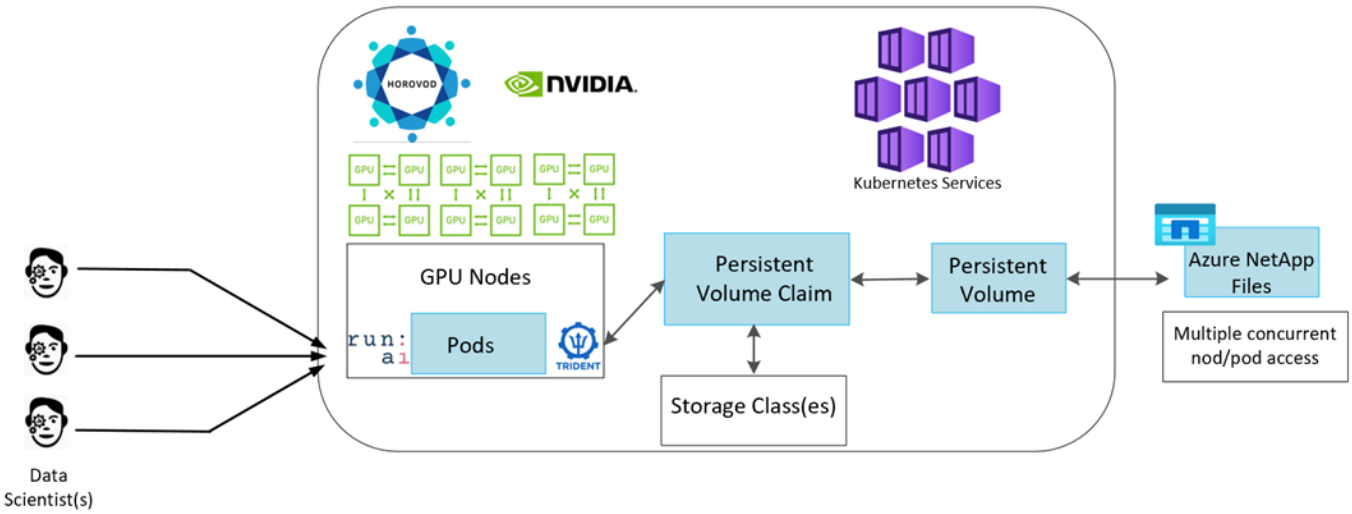
Solution technology

This section covers the technology requirements for the lane detection use case by implementing a distributed training solution at scale that fully runs in the Azure cloud. The figure below provides an overview of the solution architecture.

The elements used in this solution are:

- Azure Kubernetes Service (AKS)
- Azure Compute SKUs with NVIDIA GPUs
- Azure NetApp Files
- RUN: AI
- NetApp Trident

Links to all the elements mentioned here are listed in the Additional information section.



Cloud resources and services requirements

The following table lists the hardware components that are required to implement the solution. The cloud components that are used in any implementation of the solution might vary based on customer requirements.

Cloud	Quantity
AKS	Minimum of three system nodes and three GPU worker nodes
Virtual machine (VM) SKU system nodes	Three Standard_DS2_v2
VM SKU GPU worker nodes	Three Standard_NC6s_v3
Azure NetApp Files	4TB standard tier

Software requirements

The following table lists the software components that are required to implement the solution. The software components that are used in any implementation of the solution might vary based on customer requirements.

Software	Version or other information
AKS - Kubernetes version	1.18.14
RUN:AI CLI	v2.2.25
RUN:AI Orchestration Kubernetes Operator version	1.0.109
Horovod	0.21.2
NetApp Trident	20.01.1
Helm	3.0.0

Lane detection – Distributed training with RUN:AI

This section provides details on setting up the platform for performing lane detection distributed training at scale using the RUN: AI orchestrator. We discuss installation of all the solution elements and running the distributed training job on the said platform. ML versioning is completed by using NetApp SnapshotTM linked with RUN: AI experiments for achieving data and model reproducibility. ML versioning plays a crucial role in tracking models, sharing work between team members, reproducibility of results, rolling new model versions to production, and data provenance. NetApp ML version control (Snapshot) can capture point-in-time versions of the data, trained models, and logs associated with each experiment. It has rich API support making it easy to integrate with the RUN: AI platform; you just have to trigger an event based on the training state. You also have to capture the state of the whole experiment without changing anything in the code or the containers running on top of Kubernetes (K8s).

Finally, this technical report wraps up with performance evaluation on multiple GPU-enabled nodes across AKS.

Distributed training for lane detection use case using the TuSimple dataset

In this technical report, distributed training is performed on the TuSimple dataset for lane detection. Horovod is used in the training code for conducting data distributed training on multiple GPU nodes simultaneously in the Kubernetes cluster through AKS. Code is packaged as container images for TuSimple data download and processing. Processed data is stored on persistent volumes allocated by NetApp Trident plug- in. For the training, one more container image is created, and it uses the data stored on persistent volumes created during

downloading the data.

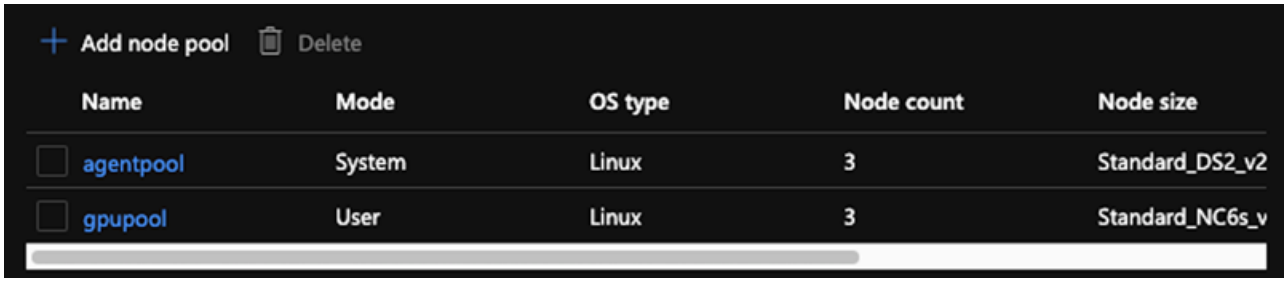
To submit the data and training job, use RUN: AI for orchestrating the resource allocation and management. RUN: AI allows you to perform Message Passing Interface (MPI) operations which are needed for Horovod. This layout allows multiple GPU nodes to communicate with each other for updating the training weights after every training mini batch. It also enables monitoring of training through the UI and CLI, making it easy to monitor the progress of experiments.

NetApp Snapshot is integrated within the training code and captures the state of data and the trained model for every experiment. This capability enables you to track the version of data and code used, and the associated trained model generated.

AKS setup and installation

For setup and installation of the AKS cluster go to Create an AKS Cluster. Then, follow these series of steps:

1. When selecting the type of nodes (whether it be system (CPU) or worker (GPU) nodes), select the following:
   a. Add primary system node named `agentpool` at the `Standard_DS2_v2` size. Use the default three nodes.
   b. Add worker node `gpupool` with `the Standard_NC6s_v3` pool size. Use three nodes minimum for GPU nodes.



Deployment takes 5–10 minutes.

2. After deployment is complete, click Connect to Cluster. To connect to the newly created AKS cluster, install the Kubernetes command-line tool from your local environment (laptop/PC). Visit Install Tools to install it as per your OS.
3. Install Azure CLI on your local environment.
4. To access the AKS cluster from the terminal, first enter `az login` and put in the credentials.
5. Run the following two commands:

```
az account set --subscription xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx
aks get-credentials --resource-group resourcegroup --name aksclustername
```

6. Enter this command in the Azure CLI:

```
kubectl get nodes
```

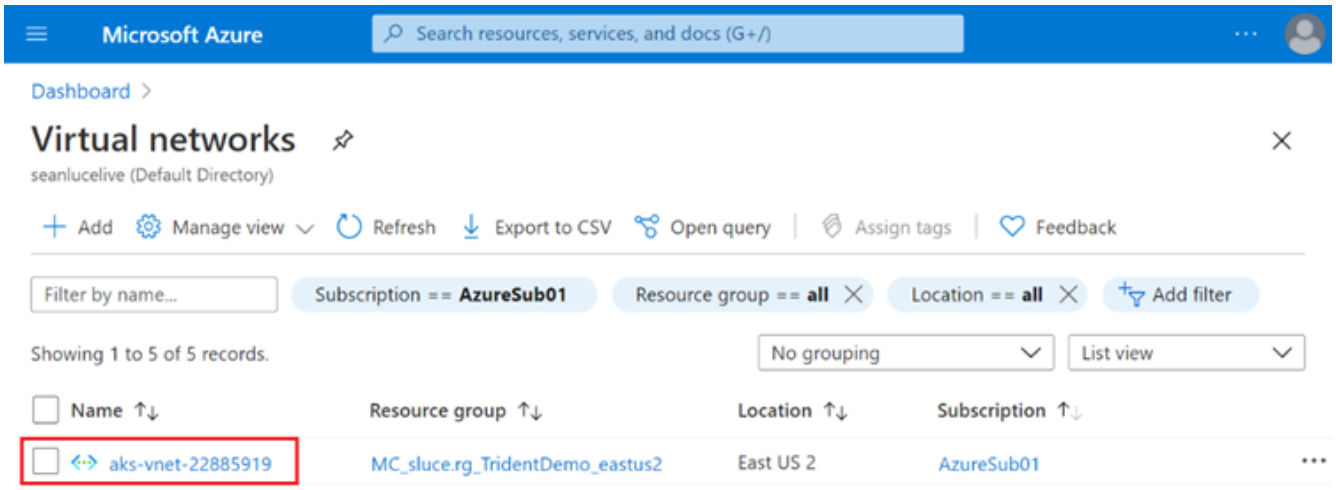
If all six nodes are up and running as seen here, your AKS cluster is ready and connected to your local environment.

```
verronmartina@verron-mac-0 ~ % kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-agentpool-34613062-vmss000000   Ready    agent   22m   v1.18.14
aks-agentpool-34613062-vmss000001   Ready    agent   22m   v1.18.14
aks-agentpool-34613062-vmss000002   Ready    agent   22m   v1.18.14
aks-gpupool-34613062-vmss000000     Ready    agent   20m   v1.18.14
aks-gpupool-34613062-vmss000001     Ready    agent   20m   v1.18.14
aks-gpupool-34613062-vmss000002     Ready    agent   20m   v1.18.14
verronmartina@verron-mac-0 ~ %
```

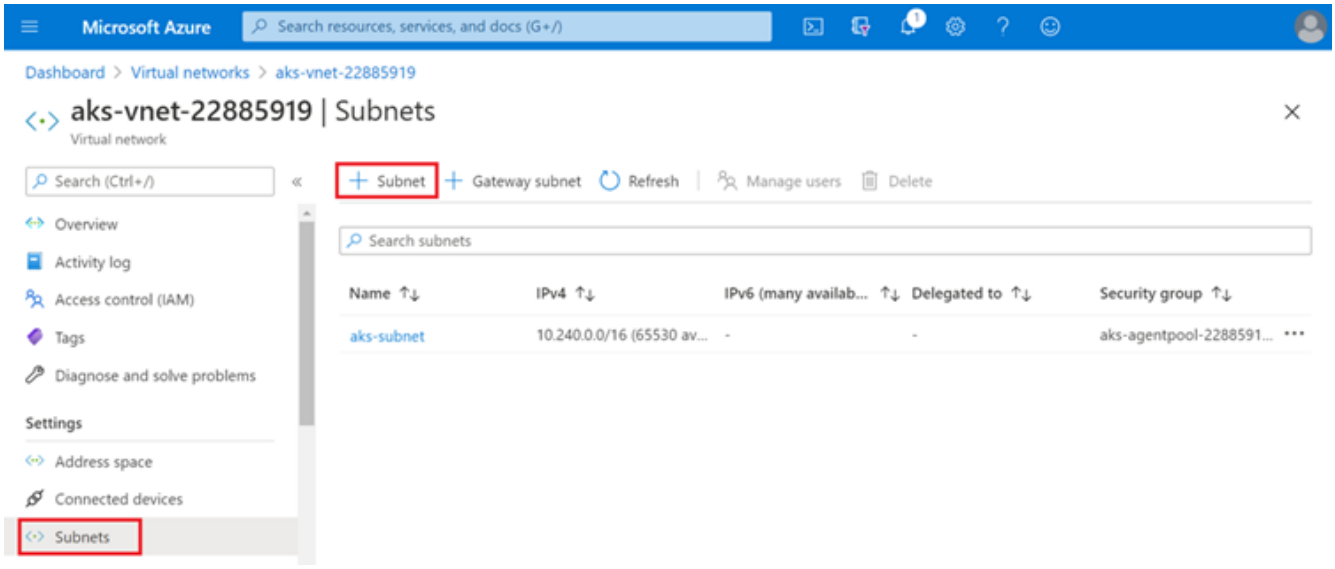
Create a delegated subnet for Azure NetApp Files

To create a delegated subnet for Azure NetApp Files, follow this series of steps:

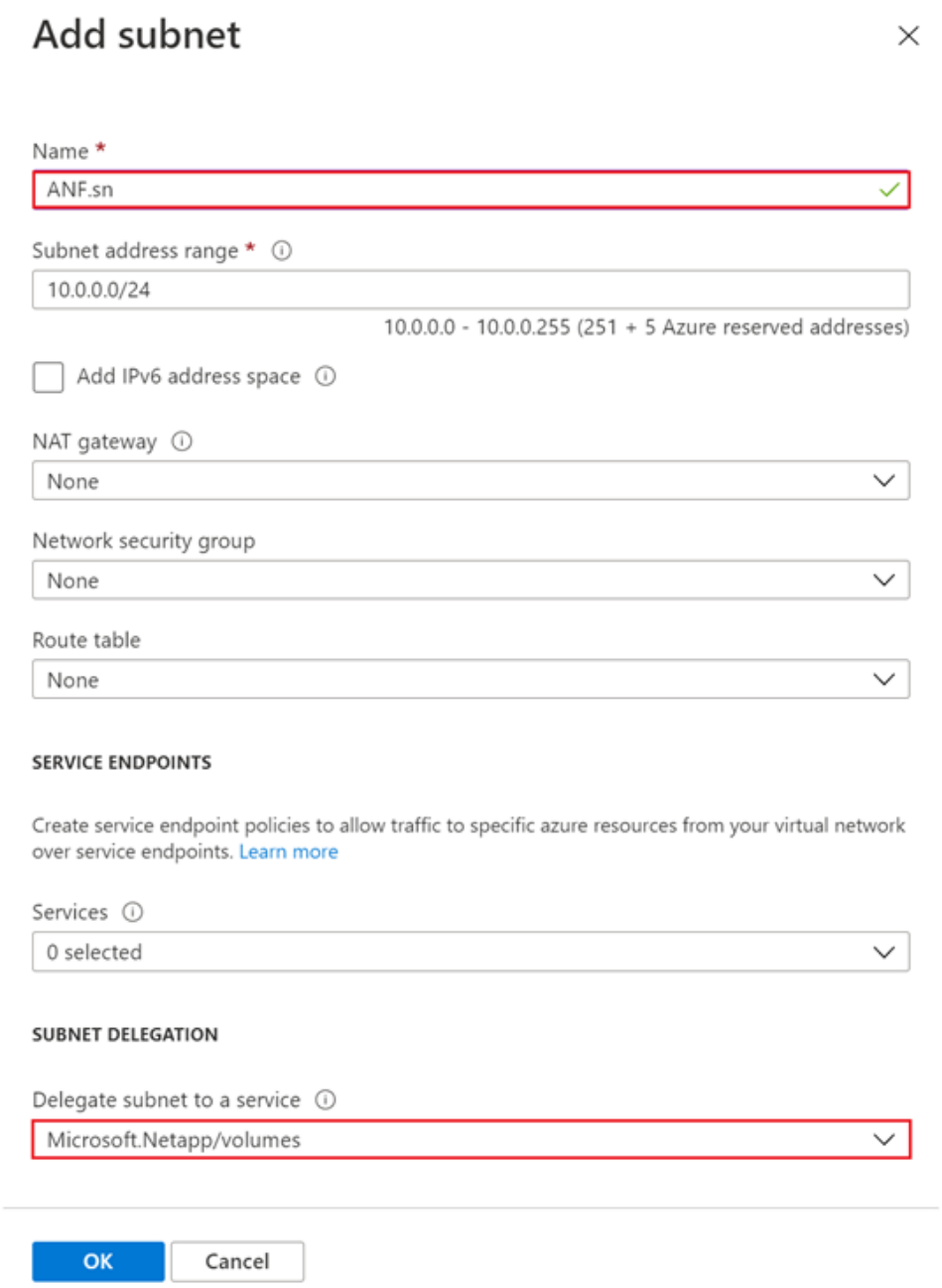
1. Navigate to Virtual networks within the Azure portal. Find your newly created virtual network. It should have a prefix such as aks-vnet, as seen here. Click the name of the virtual network.



2. Click Subnets and select +Subnet from the top toolbar.



3. Provide the subnet with a name such as `ANF.sn` and under the Subnet Delegation heading, select Microsoft.NetApp/volumes. Do not change anything else. Click OK.



Azure NetApp Files volumes are allocated to the application cluster and are consumed as persistent volume claims (PVCs) in Kubernetes. In turn, this allocation provides us the flexibility to map volumes to different services, be it Jupyter notebooks, serverless functions, and so on

Users of services can consume storage from the platform in many ways. The main benefits of Azure NetApp Files are:

- Provides users with the ability to use snapshots.
- Enables users to store large quantities of data on Azure NetApp Files volumes.
- Procure the performance benefits of Azure NetApp Files volumes when running their models on large sets of files.

Azure NetApp Files setup

To complete the setup of Azure NetApp Files, you must first configure it as described in Quickstart: Set up Azure NetApp Files and create an NFS volume.

However, you may omit the steps to create an NFS volume for Azure NetApp Files as you will create volumes through Trident. Before continuing, be sure that you have:

1. Registered for Azure NetApp Files and NetApp Resource Provider (through the Azure Cloud Shell).
2. Created an account in Azure NetApp Files.
3. Set up a capacity pool (minimum 4TiB Standard or Premium depending on your needs).

Peering of AKS virtual network and Azure NetApp Files virtual network

Next, peer the AKS virtual network (VNet) with the Azure NetApp Files VNet by following these steps:

1. In the search box at the top of the Azure portal, type virtual networks.
2. Click VNet aks- vnet-name, then enter Peerings in the search field.
3. Click +Add and enter the information provided in the table below:

Field	Value or description
Peering link name	aks-vnet-name_to_anf
SubscriptionID	Subscription of the Azure NetApp Files VNet to which you're peering
VNet peering partner	Azure NetApp Files VNet

Leave all the nonasterisk sections on default

4. Click ADD or OK to add the peering to the virtual network.

For more information, visit Create, change, or delete a virtual network peering.

Trident

Trident is an open-source project that NetApp maintains for application container persistent storage. Trident has been implemented as an external provisioner controller that runs as a pod itself, monitoring volumes and completely automating the provisioning process.

NetApp Trident enables smooth integration with K8s by creating and attaching persistent volumes for storing training datasets and trained models. This capability makes it easier for data scientists and data engineers to use K8s without the hassle of manually storing and managing datasets. Trident also eliminates the need for data scientists to learn managing new data platforms as it integrates the data management-related tasks through the logical API integration.

Install Trident

To install Trident software, complete the following steps:

1. First install helm.
2. Download and extract the Trident 21.01.1 installer.

```
wget
https://github.com/NetApp/trident/releases/download/v21.01.1/trident-
installer-21.01.1.tar.gz
tar -xf trident-installer-21.01.1.tar.gz
```

3. Change the directory to `trident-installer`.

```
cd trident-installer
```

4. Copy `tridentctl` to a directory in your system `$PATH`.

```
cp ./tridentctl /usr/local/bin
```

5. Install Trident on K8s cluster with Helm:
   a. Change directory to helm directory.

```
cd helm
```

   b. Install Trident.

```
helm install trident trident-operator-21.01.1.tgz --namespace trident
--create-namespace
```

   c. Check the status of Trident pods the usual K8s way:

```
kubectl -n trident get pods
```

   d. If all the pods are up and running, Trident is installed and you are good to move forward.

Set up Azure NetApp Files back-end and storage class

To set up Azure NetApp Files back-end and storage class, complete the following steps:

1. Switch back to the home directory.

```
cd ~
```

2. Clone the project repository `lane-detection-SCNN-horovod`.
3. Go to the `trident-config` directory.

```
cd ./lane-detection-SCNN-horovod/trident-config
```

4. Create an Azure Service Principle (the service principle is how Trident communicates with Azure to access your Azure NetApp Files resources).

```
az ad sp create-for-rbac --name
```

The output should look like the following example:

```
{
  "appId": "xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
   "displayName": "netapptrident",
    "name": "http://netapptrident",
    "password": "xxxxxxxxxxxxxx.xxxxxxxxxxxxx",
    "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx"
 }
```

5. Create the Trident `backend json` file.
6. Using your preferred text editor, complete the following fields from the table below inside the `anf-backend.json` file.

Field	Value
subscriptionID	Your Azure Subscription ID
tenantID	Your Azure Tenant ID (from the output of az ad sp in the previous step)
clientID	Your appID (from the output of az ad sp in the previous step)
clientSecret	Your password (from the output of az ad sp in the previous step)

The file should look like the following example:

```
{
    "version": 1,
    "storageDriverName": "azure-netapp-files",
    "subscriptionID": "fakec765-4774-fake-ae98-a721add4fake",
    "tenantID": "fakef836-edc1-fake-bff9-b2d865eefake",
    "clientID": "fake0f63-bf8e-fake-8076-8de91e57fake",
    "clientSecret": "SECRET",
    "location": "westeurope",
    "serviceLevel": "Standard",
    "virtualNetwork": "anf-vnet",
    "subnet": "default",
    "nfsMountOptions": "vers=3,proto=tcp",
    "limitVolumeSize": "500Gi",
    "defaults": {
    "exportRule": "0.0.0.0/0",
    "size": "200Gi"
}
```

7. Instruct Trident to create the Azure NetApp Files back- end in the `trident` namespace, using `anf-backend.json` as the configuration file as follows:

```
tridentctl create backend -f anf-backend.json -n trident
```

8. Create the storage class:
   a. K8 users provision volumes by using PVCs that specify a storage class by name. Instruct K8s to create a storage class `azurenetappfiles` that will reference the Azure NetApp Files back end created in the previous step using the following:

   ```
   kubectl create -f anf-storage-class.yaml
   ```

   b. Check that storage class is created by using the following command:

   ```
   kubectl get sc azurenetappfiles
   ```

   The output should look like the following example:

```
NAME              PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
azurenetappfiles  csi.trident.netapp.io   Delete          Immediate           false                  98s
```

Deploy and set up volume snapshot components on AKS

If your cluster does not come pre-installed with the correct volume snapshot components, you may manually install these components by running the following steps:

AKS 1.18.14 does not have pre-installed Snapshot Controller.

1. Install Snapshot Beta CRDs by using the following commands:

```
kubectl create -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. Install Snapshot Controller by using the following documents from GitHub:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

3. Set up K8s `volumesnapshotclass`: Before creating a volume snapshot, a volume snapshot class must be set up. Create a volume snapshot class for Azure NetApp Files, and use it to achieve ML versioning by using NetApp Snapshot technology. Create `volumesnapshotclass netapp-csi-snapclass` and set it to default `volumesnapshotclass `as such:

```
kubectl create -f netapp-volume-snapshot-class.yaml
```

The output should look like the following example:

```
volumesnapshotclass.snapshot.storage.k8s.io/netapp-csi-snapclass created
```

4. Check that the volume Snapshot copy class was created by using the following command:

```
kubectl get volumesnapshotclass
```

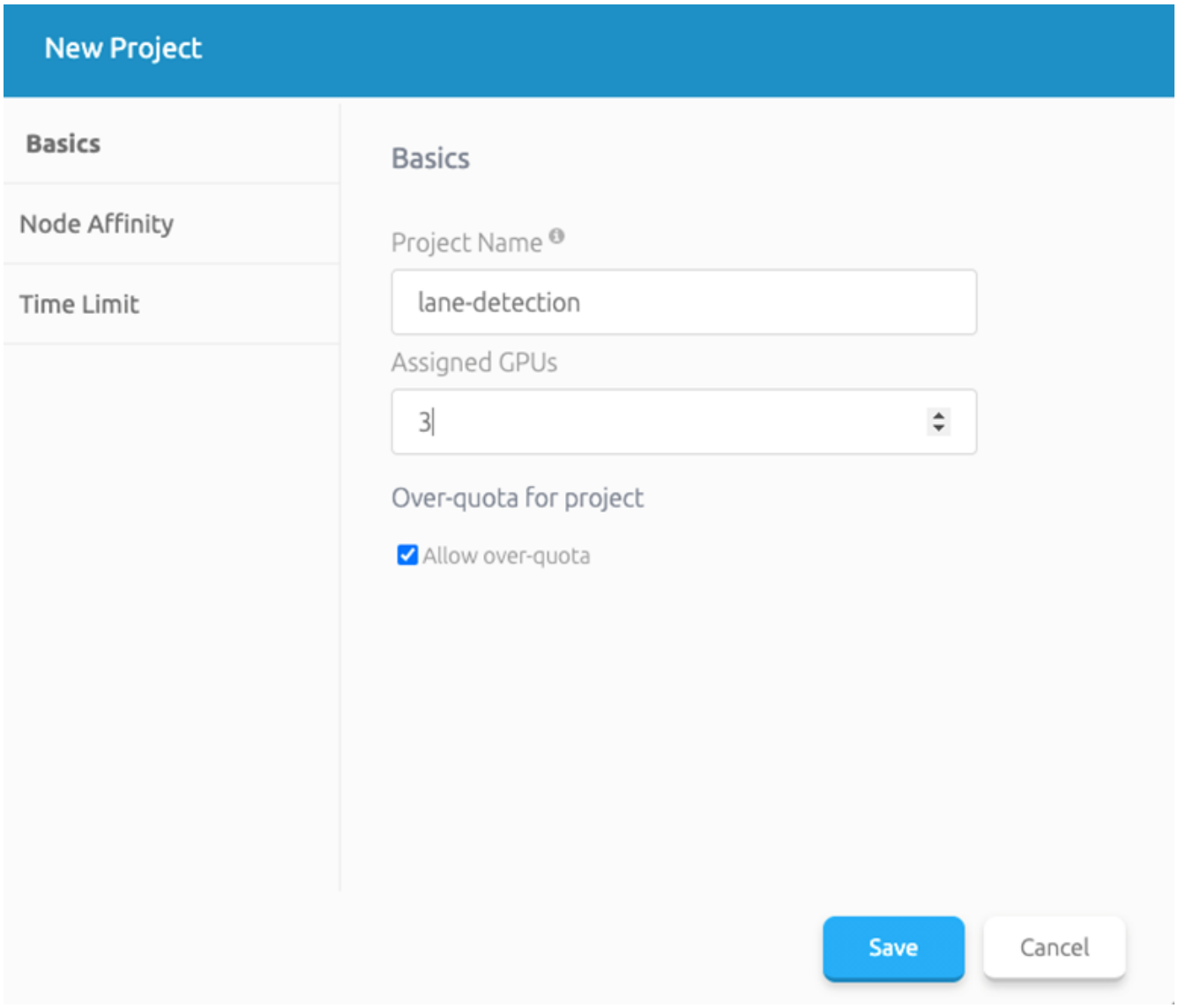
The output should look like the following example:

```
NAME                    DRIVER                  DELETIONPOLICY   AGE
netapp-csi-snapclass    csi.trident.netapp.io   Delete           63s
```

RUN:AI installation

To install RUN:AI, complete the following steps:

1. Install RUN:AI cluster on AKS.
2. Go to app.runai.ai, click create New Project, and name it lane-detection. It will create a namespace on a K8s cluster starting with `runai-` followed by the project name. In this case, the namespace created would be runai-lane-detection.



3. Install RUN:AI CLI.
4. On your terminal, set lane-detection as a default RUN: AI project by using the following command:

```
`runai config project lane-detection`
```

The output should look like the following example:

```
Project lane-detection has been set as default project
```

5. Create ClusterRole and ClusterRoleBinding for the project namespace (for example, `lane-detection`) so the default service account belonging to `runai-lane-detection` namespace has permission to perform `volumesnapshot` operations during job execution:

    a. List namespaces to check that `runai-lane-detection` exists by using this command:

    ```
    kubectl get namespaces
    ```

    The output should appear like the following example:

    ```
    NAME                 STATUS   AGE
    default              Active   130m
    kube-node-lease      Active   130m
    kube-public          Active   130m
    kube-system          Active   130m
    runai                Active   4m44s
    runai-lane-detection Active   13s
    trident              Active   102m
    ```

6. Create ClusterRole `netappsnapshot` and ClusterRoleBinding `netappsnapshot` using the following commands:

    ```
    `kubectl create -f runai-project-snap-role.yaml`
    `kubectl create -f runai-project-snap-role-binding.yaml`
    ```

Download and process the TuSimple dataset as RUN:AI job

The process to download and process the TuSimple dataset as a RUN: AI job is optional. It involves the following steps:

1. Build and push the docker image, or omit this step if you want to use an existing docker image (for example, `muneer7589/download-tusimple:1.0`)

    a. Switch to the home directory:

    ```
    cd ~
    ```

    b. Go to the data directory of the project `lane-detection-SCNN-horovod`:

    ```
    cd ./lane-detection-SCNN-horovod/data
    ```

    c. Modify `build_image.sh` shell script and change docker repository to yours. For example, replace `muneer7589` with your docker repository name. You could also change the docker image name and

TAG (such as `download-tusimple` and `1.0`):

```
#!/bin/bash
#
# A simple script to build the Docker image.
#
# $ build_image.sh
set -ex

IMAGE=muneer7589/download-tusimple
TAG=1.0

# Build image
echo "Building image: "$IMAGE
docker build . -f Dockerfile \
 --tag "${IMAGE}:${TAG}"
echo "Finished building image: "$IMAGE

# Push image
echo "Pushing image: "$IMAGE
docker push "${IMAGE}:${TAG}"
echo "Finished pushing image: "$IMAGE
```

d. Run the script to build the docker image and push it to the docker repository using these commands:

```
chmod +x build_image.sh
./build_image.sh
```

2. Submit the RUN: AI job to download, extract, pre-process, and store the TuSimple lane detection dataset in a `pvc`, which is dynamically created by NetApp Trident:

   a. Use the following commands to submit the RUN: AI job:

```
runai submit
--name download-tusimple-data
--pvc azurenetappfiles:100Gi:/mnt
--image muneer7589/download-tusimple:1.0
```

b. Enter the information from the table below to submit the RUN:AI job:

Field	Value or description
-name	Name of the job
-pvc	PVC of the format [StorageClassName]:Size:ContainerMountPath In the above job submission, you are creating an PVC based on-demand using Trident with storage class azurenetappfiles. Persistent volume capacity here is 100Gi and it's mounted at path /mnt.
-image	Docker image to use when creating the container for this job

The output should look like the following example:

```
The job 'download-tusimple-data' has been submitted successfully
You can run `runai describe job download-tusimple-data -p lane-detection` to check the job status
```

c. List the submitted RUN:AI jobs.

```
runai list jobs
```

```
Showing jobs for project lane-detection
NAME                STATUS            AGE  NODE                          IMAGE                              TYPE   PROJECT         USER          GPUs Allocated (Requested)
PODs Running (Pending)  SERVICE URL(S)
download-tusimple-data  ContainerCreating  1m   aks-agentpool-34613062-vmss00000a  muneer7589/download-tusimple:1.0  Train  lane-detection  verronmartina  0 (0)
1 (0)
```

d. Check the submitted job logs.

```
runai logs download-tusimple-data -t 10
```

```
751150K .......... .......... .......... .......... ..........  6% 16.2M 20m37s
751200K .......... .......... .......... .......... ..........  6% 11.1M 20m37s
751250K .......... .......... .......... .......... ..........  6% 12.5M 20m36s
751300K .......... .......... .......... .......... ..........  6% 11.3M 20m36s
751350K .......... .......... .......... .......... ..........  6% 15.2M 20m36s
751400K .......... .......... .......... .......... ..........  6% 10.5M 20m36s
751450K .......... .......... .......... .......... ..........  6% 15.2M 20m36s
751500K .......... .......... .......... .......... ..........  6% 14.1M 20m36s
751550K .......... .......... .......... .......... ..........  6% 24.3M 20m36s
751600K .......... .......... .......... .......... ..........  6% 26.3M 20m36s
```

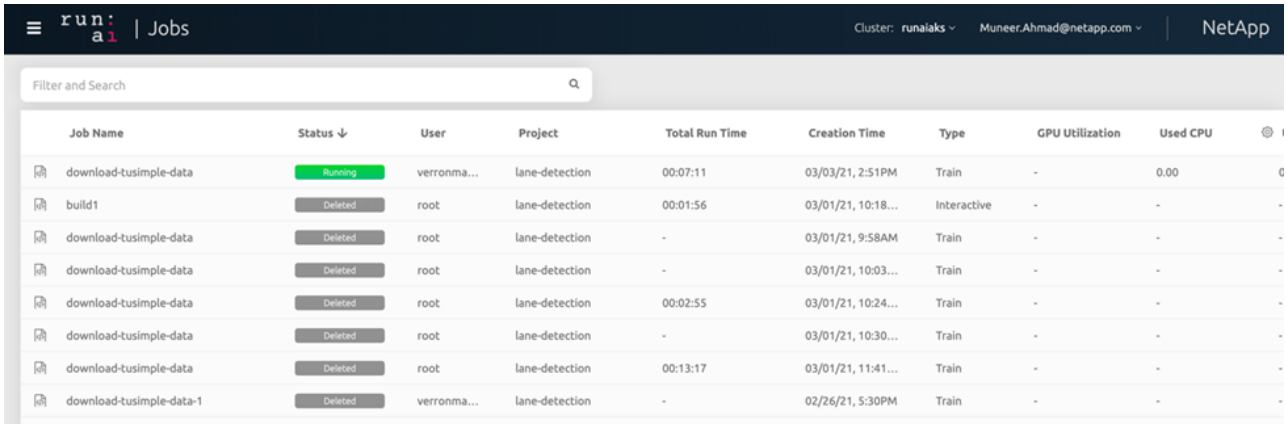
e. List the `pvc` created. Use this `pvc` command for training in the next step.

```
kubectl get pvc | grep download-tusimple-data
```

The output should look like the following example:

```
pvc-download-tusimple-data-0   Bound    pvc-bb03b74d-2c17-40c4-a445-79f3de8d16d5   100Gi      RWO            azurenetappfiles   4m47s
```

f. Check the job in RUN: AI UI (or `app.run.ai`).



Perform distributed lane detection training using Horovod

Performing distributed lane detection training using Horovod is an optional process. However, here are the steps involved:

1. Build and push the docker image, or skip this step if you want to use the existing docker image (for example, `muneer7589/dist-lane-detection:3.1)`:

   a. Switch to home directory.

   ```
   cd ~
   ```

   b. Go to the project directory `lane-detection-SCNN-horovod.`

   ```
   cd ./lane-detection-SCNN-horovod
   ```

   c. Modify the `build_image.sh` shell script and change docker repository to yours (for example, replace `muneer7589` with your docker repository name). You could also change the docker image name and TAG (`dist-lane-detection` and `3.1, for example)`.

```
#!/bin/bash
#
# A simple script to build the distributed Docker image.
#
# $ build_image.sh
set -ex

IMAGE=muneer7589/dist-lane-detection
TAG=3.0

# Build image
echo "Building image: "$IMAGE
docker build . -f Dockerfile \
 --tag "${IMAGE}:${TAG}"
echo "Finished building image: "$IMAGE

# Push image
echo "Pushing image: "$IMAGE
docker push "${IMAGE}:${TAG}"
echo "Finished pushing image: "$IMAGE
```

d. Run the script to build the docker image and push to the docker repository.

```
chmod +x build_image.sh
./build_image.sh
```

2. Submit the RUN: AI job for carrying out distributed training (MPI):

a. Using submit of RUN: AI for automatically creating PVC in the previous step (for downloading data) only allows you to have RWO access, which does not allow multiple pods or nodes to access the same PVC for distributed training. Update the access mode to ReadWriteMany and use the Kubernetes patch to do so.

b. First, get the volume name of the PVC by running the following command:

```
kubectl get pvc | grep download-tusimple-data
```

```
root@ai-w-gpu-2:/mnt/ai_data/anf_runai/lane-detection-SCNN-horovod# kubectl get pvc | grep download-tusimple-data
pvc-download-tusimple-data-0   Bound    pvc-bb03b74d-2c17-40c4-a445-79f3de8d16d5   100Gi      RWX            azurenetappfiles   2d4h
```

c. Patch the volume and update access mode to ReadWriteMany (replace volume name with yours in the following command):

```
kubectl patch pv pvc-bb03b74d-2c17-40c4-a445-79f3de8d16d5 -p
'{"spec":{"accessModes":["ReadWriteMany"]}}'
```

d. Submit the RUN: AI MPI job for executing the distributed training` job using information from the table below:

```
runai submit-mpi
--name dist-lane-detection-training
--large-shm
--processes=3
--gpu 1
--pvc pvc-download-tusimple-data-0:/mnt
--image muneer7589/dist-lane-detection:3.1
-e USE_WORKERS="true"
-e NUM_WORKERS=4
-e BATCH_SIZE=33
-e USE_VAL="false"
-e VAL_BATCH_SIZE=99
-e ENABLE_SNAPSHOT="true"
-e PVC_NAME="pvc-download-tusimple-data-0"
```

Field	**Value or description**
name	Name of the distributed training job
large shm	Mount a large /dev/shm device It is a shared file system mounted on RAM and provides large enough shared memory for multiple CPU workers to process and load batches into CPU RAM.
processes	Number of distributed training processes
gpu	Number of GPUs/processes to allocate for the job In this job, there are three GPU worker processes (--processes=3), each allocated with a single GPU (--gpu 1)
pvc	Use existing persistent volume (pvc-download-tusimple-data-0) created by previous job (download-tusimple-data) and it is mounted at path /mnt
image	Docker image to use when creating the container for this job
Define environment variables to be set in the container	
USE_WORKERS	Setting the argument to true turns on multi-process data loading
NUM_WORKERS	Number of data loader worker processes
BATCH_SIZE	Training batch size

Field	Value or description
USE_VAL	Setting the argument to true allows validation
VAL_BATCH_SIZE	Validation batch size
ENABLE_SNAPSHOT	Setting the argument to true enables taking data and trained model snapshots for ML versioning purposes
PVC_NAME	Name of the pvc to take a snapshot of. In the above job submission, you are taking a snapshot of pvc-download-tusimple-data-0, consisting of dataset and trained models

The output should look like the following example:

```
The job 'dist-lane-detection-training' has been submitted successfully
You can run `runai describe job dist-lane-detection-training -p lane-detection` to check the job status
```

e. List the submitted job.

```
runai list jobs
```

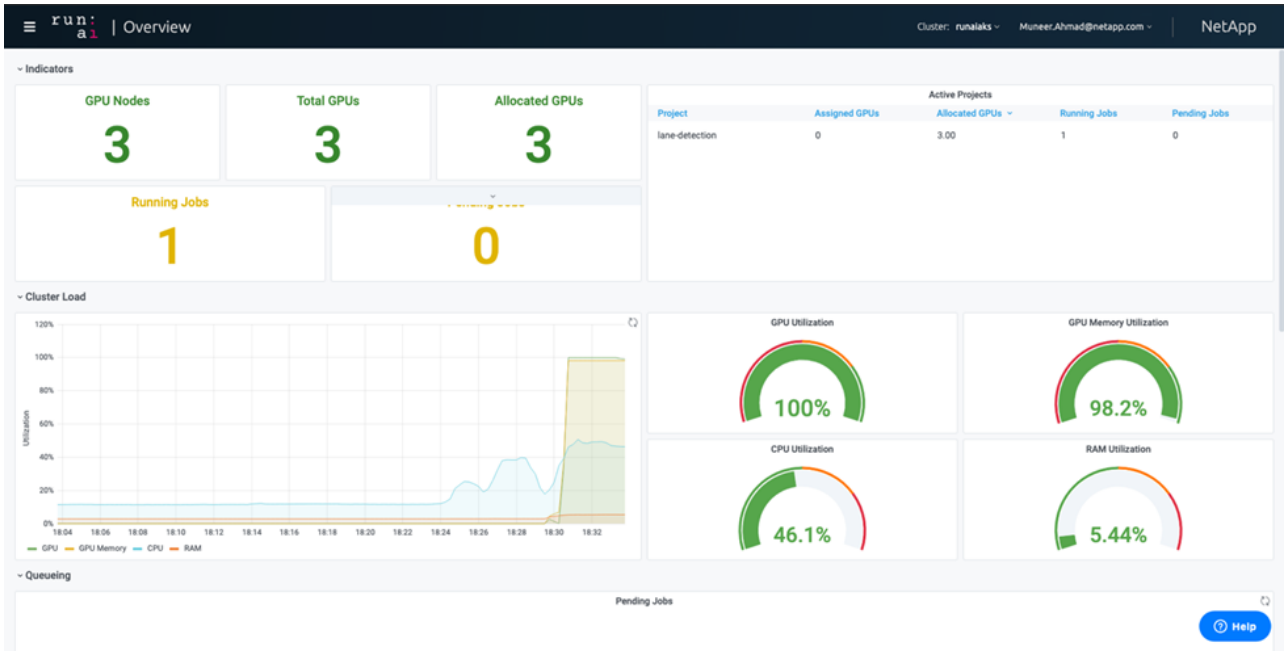
```
NAME                          STATUS     AGE  NODE        IMAGE                                 TYPE   PROJECT         USER          GPUs Allocated (Requested)  PODs
 SERVICE URL(S)
download-tusimple-data        Succeeded  1d                muneer7589/download-tusimple:1.0      Train  lane-detection  verronmartina  - (0)                       0 (0)

dist-lane-detection-training  Init:0/1   2m   <multiple>  muneer7589/dist-lane-detection:3.1    Train  lane-detection  root           3 (3)                       4 (0)
```

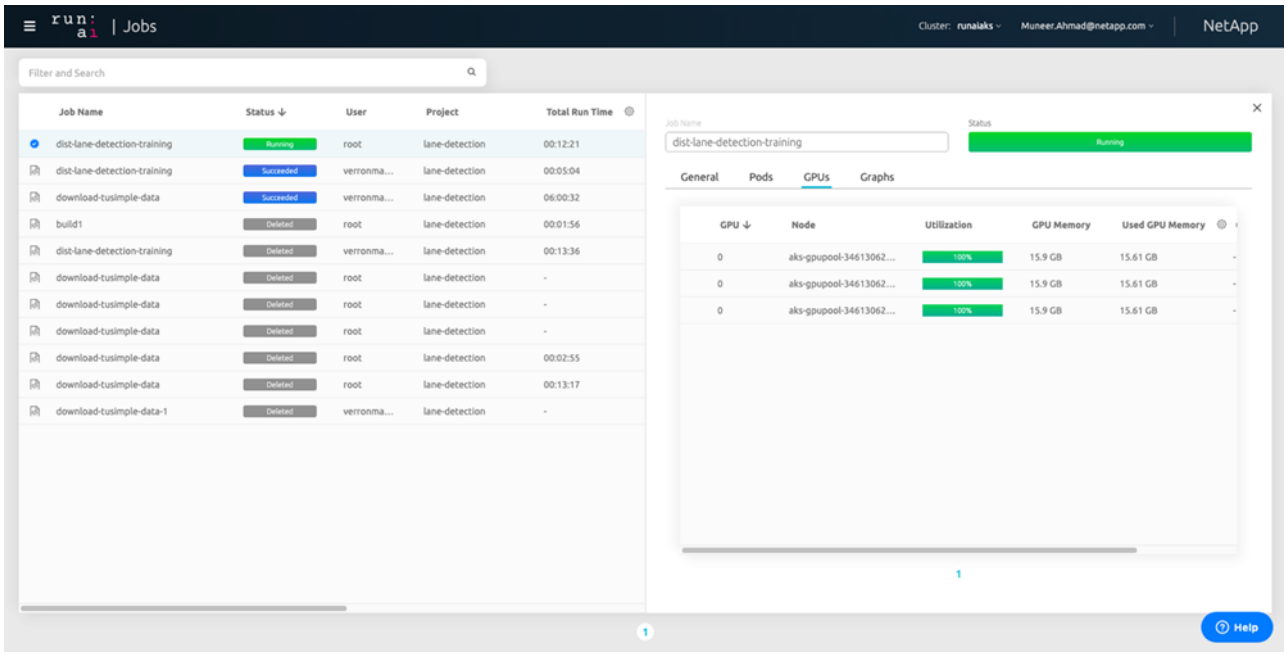
f. Submitted job logs:

```
runai logs dist-lane-detection-training
```

```
root@ai-w-gpu-2:~/runai# runai logs dist-lane-detection-training
Running with 3 workers
2021-03-04 17:29:23.158449: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudart.so.10.1
+ POD_NAME=dist-lane-detection-training-worker-0
+ [ d = - ]
+ shift
+ /opt/kube/kubectl cp /opt/kube/hosts dist-lane-detection-training-worker-0:/etc/hosts_of_nodes
+ POD_NAME=dist-lane-detection-training-worker-2
+ [ d = - ]
+ shift
+ /opt/kube/kubectl cp /opt/kube/hosts dist-lane-detection-training-worker-2:/etc/hosts_of_nodes
+ POD_NAME=dist-lane-detection-training-worker-1
```

g. Check training job in RUN: AI GUI (or app.runai.ai): RUN: AI Dashboard, as seen in the figures below. The first figure details three GPUs allocated for the distributed training job spread across three nodes on AKS, and the second RUN:AI jobs:





h. After the training is finished, check the NetApp Snapshot copy that was created and linked with RUN:AI job.

```
runai logs dist-lane-detection-training --tail 1
```

```
[1,0]<stdout>:Snapshot snap-pvc-download-tusimple-data-0-dist-lane-detection-training-launcher-2021-03-05-16-23-42 created in namespace runai-lane-detection
```

```
kubectl get volumesnapshots | grep download-tusimple-data-0
```

Restore data from the NetApp Snapshot copy

To restore data from the NetApp Snapshot copy, complete the following steps:

1. Switch to home directory.

```
cd ~
```

2. Go to the project directory `lane-detection-SCNN-horovod`.

```
cd ./lane-detection-SCNN-horovod
```

3. Modify `restore-snaphot-pvc.yaml` and update `dataSource name` field to the Snapshot copy from which you want to restore data. You could also change PVC name where the data will be restored to, in this example its `restored-tusimple`.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restored-tusimple
spec:
  storageClassName: azurenetappfiles
  dataSource:
    name: snap-pvc-download-tusimple-data-0-dist-lane-detection-training-launcher-2021-03-05-16-23-42
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
```

4. Create a new PVC by using `restore-snapshot-pvc.yaml`.

```
kubectl create -f restore-snapshot-pvc.yaml
```

The output should look like the following example:

```
persistentvolumeclaim/restored-tusimple created
```

5. If you want to use the just restored data for training, job submission remains the same as before; only replace the `PVC_NAME` with the restored `PVC_NAME` when submitting the training job, as seen in the following commands:

```
runai submit-mpi
--name dist-lane-detection-training
--large-shm
--processes=3
--gpu 1
--pvc restored-tusimple:/mnt
--image muneer7589/dist-lane-detection:3.1
-e USE_WORKERS="true"
-e NUM_WORKERS=4
-e BATCH_SIZE=33
-e USE_VAL="false"
-e VAL_BATCH_SIZE=99
-e ENABLE_SNAPSHOT="true"
-e PVC_NAME="restored-tusimple"
```

Performance evaluation

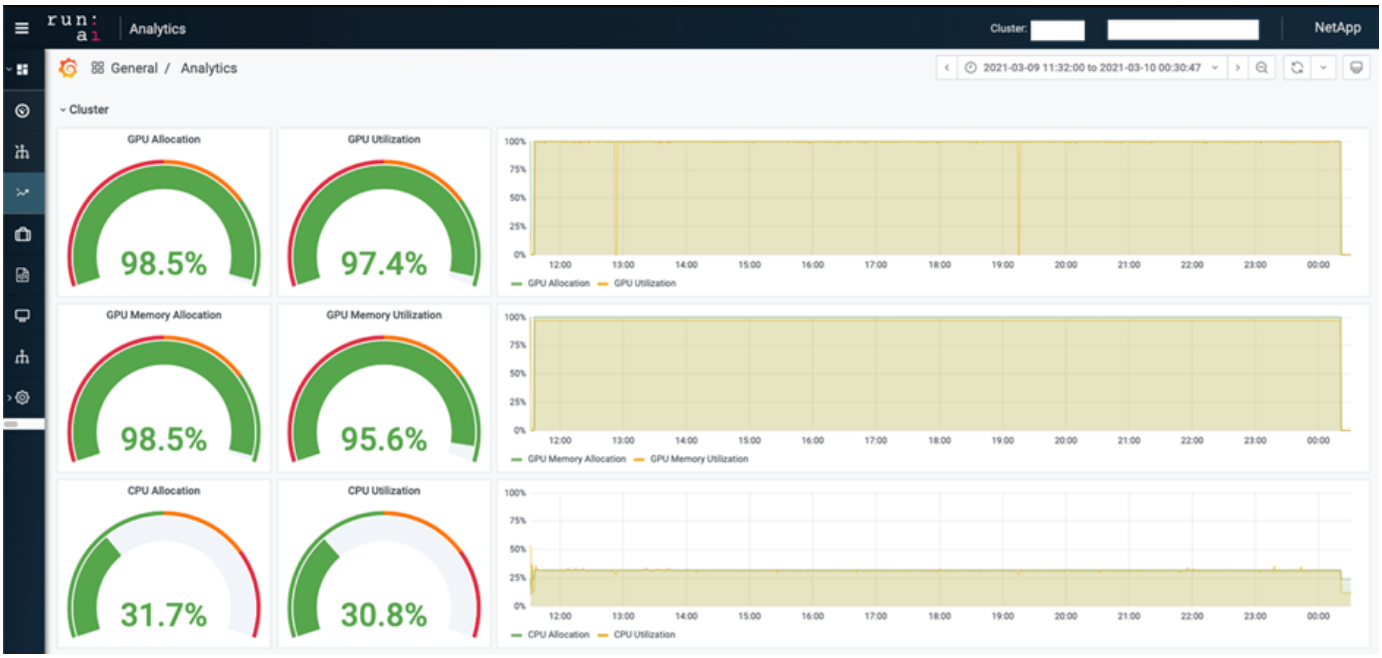
To show the linear scalability of the solution, performance tests have been done for two scenarios: one GPU and three GPUs. GPU allocation, GPU and memory utilization, different single- and three- node metrics have been captured during the training on the TuSimple lane detection dataset. Data is increased five- fold just for the sake of analyzing resource utilization during the training processes.

The solution enables customers to start with a small dataset and a few GPUs. When the amount of data and the demand of GPUs increase, customers can dynamically scale out the terabytes in the Standard Tier and quickly scale up to the Premium Tier to get four times the throughput per terabyte without moving any data. This process is further explained in the section, Azure NetApp Files service levels.

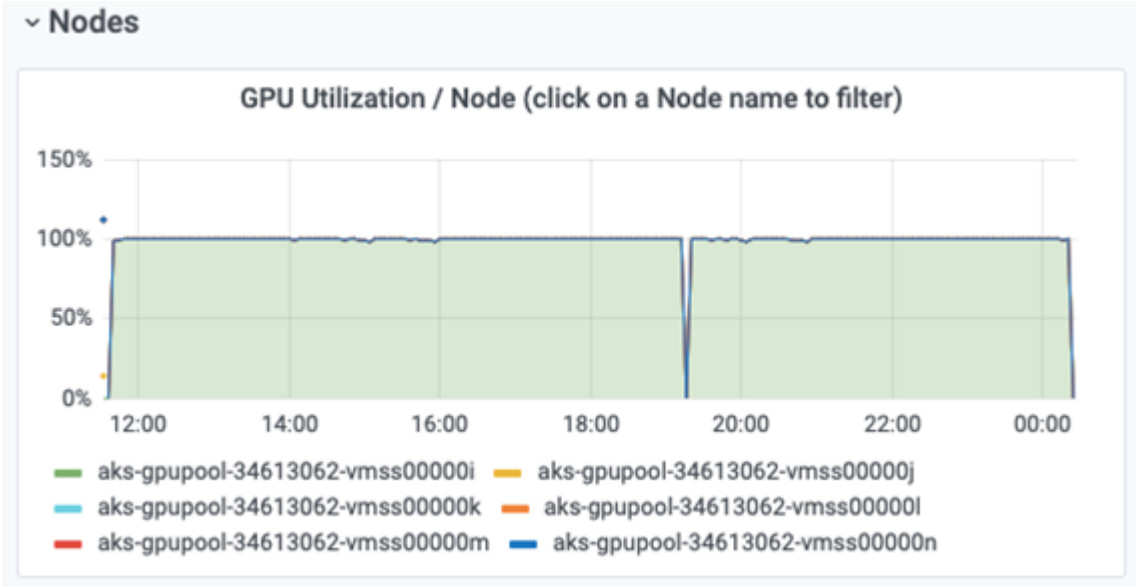
Processing time on one GPU was 12 hours and 45 minutes. Processing time on three GPUs across three nodes was approximately 4 hours and 30 minutes.

The figures shown throughout the remainder of this document illustrate examples of performance and scalability based on individual business needs.

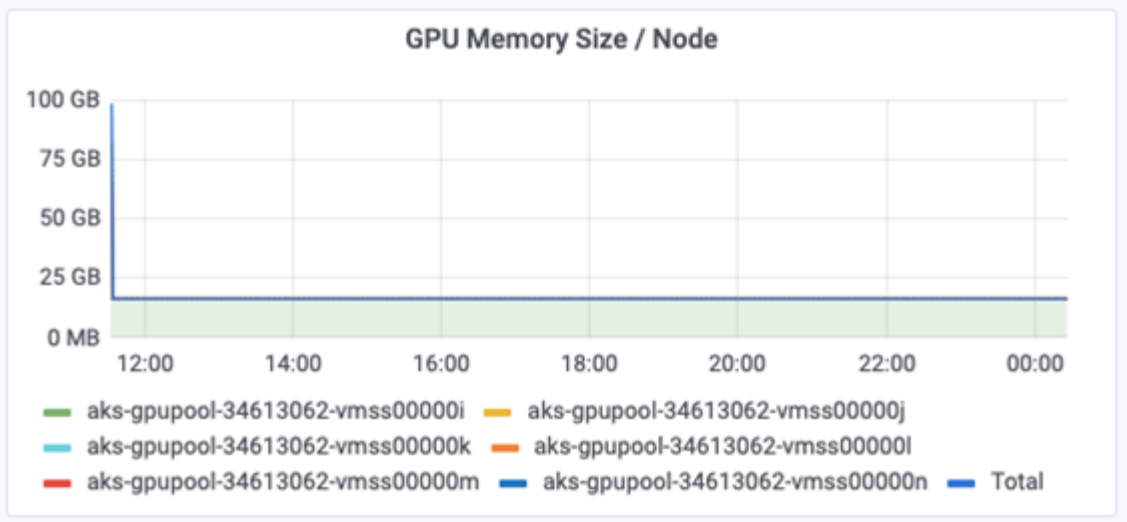
The figure below illustrates 1 GPU allocation and memory utilization.



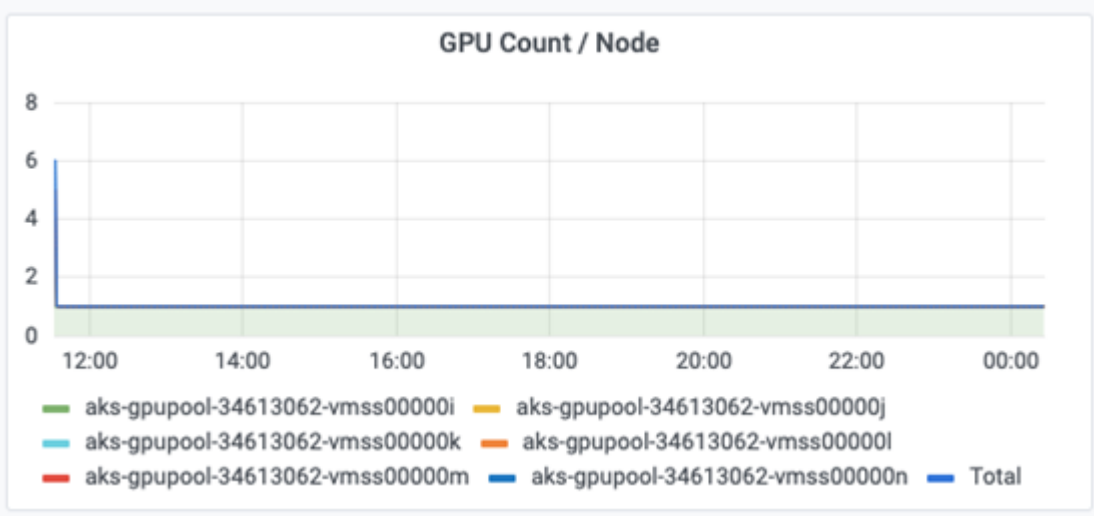
The figure below illustrates single node GPU utilization.



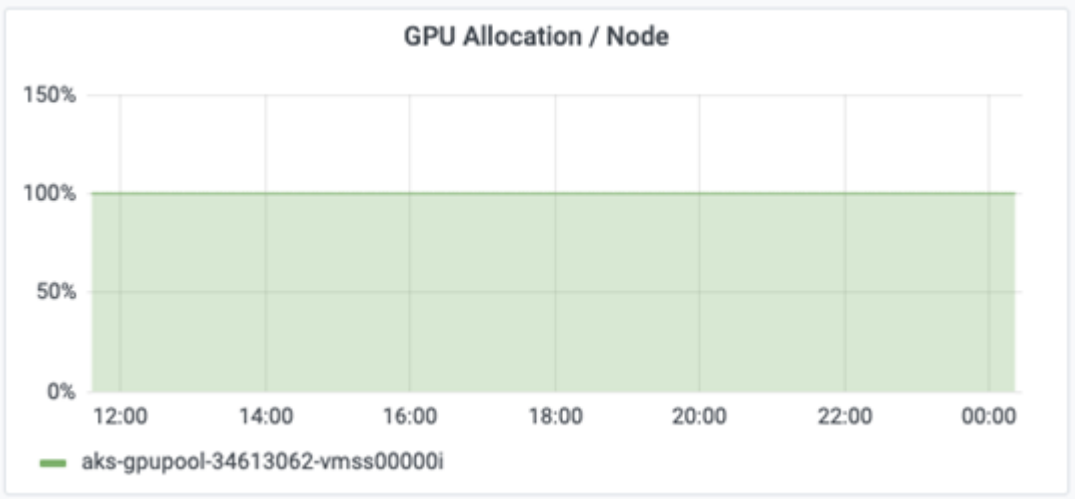
The figure below illustrates single node memory size (16GB).



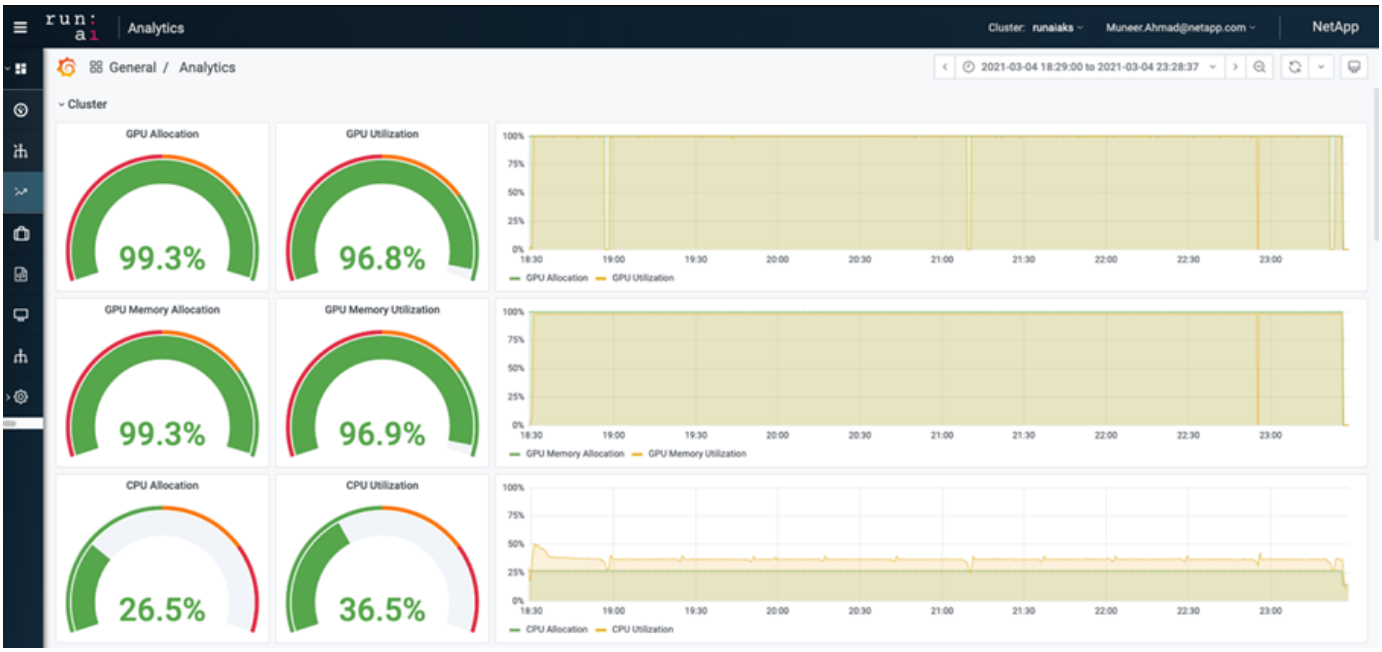
The figure below illustrates single node GPU count (1).



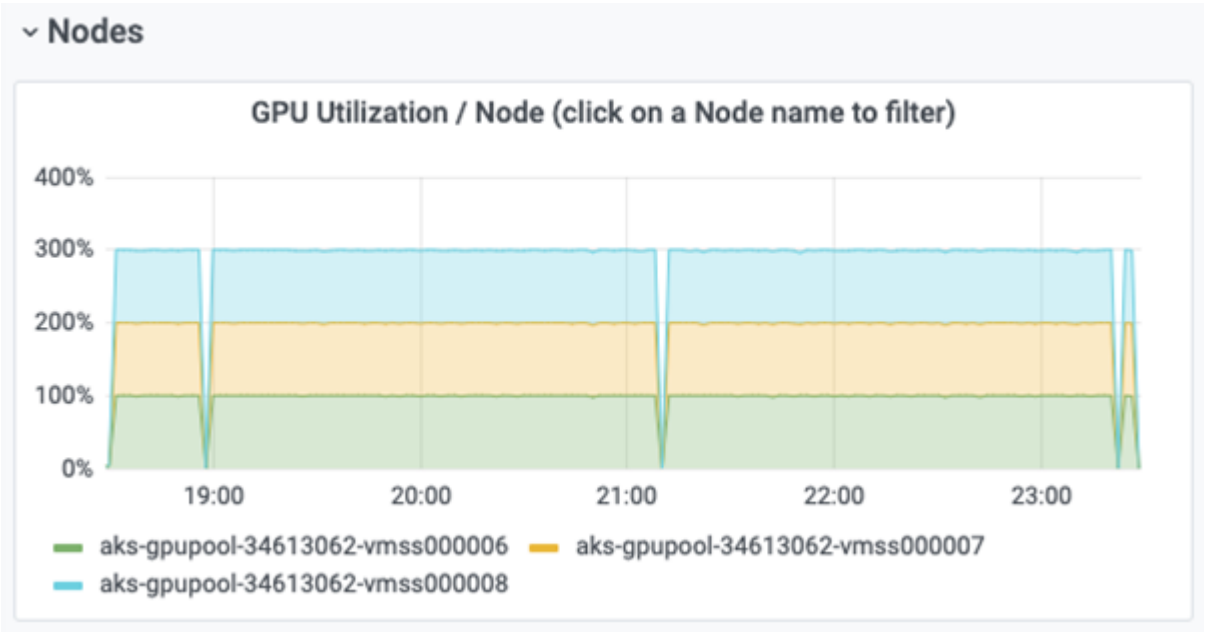
The figure below illustrates single node GPU allocation (%).



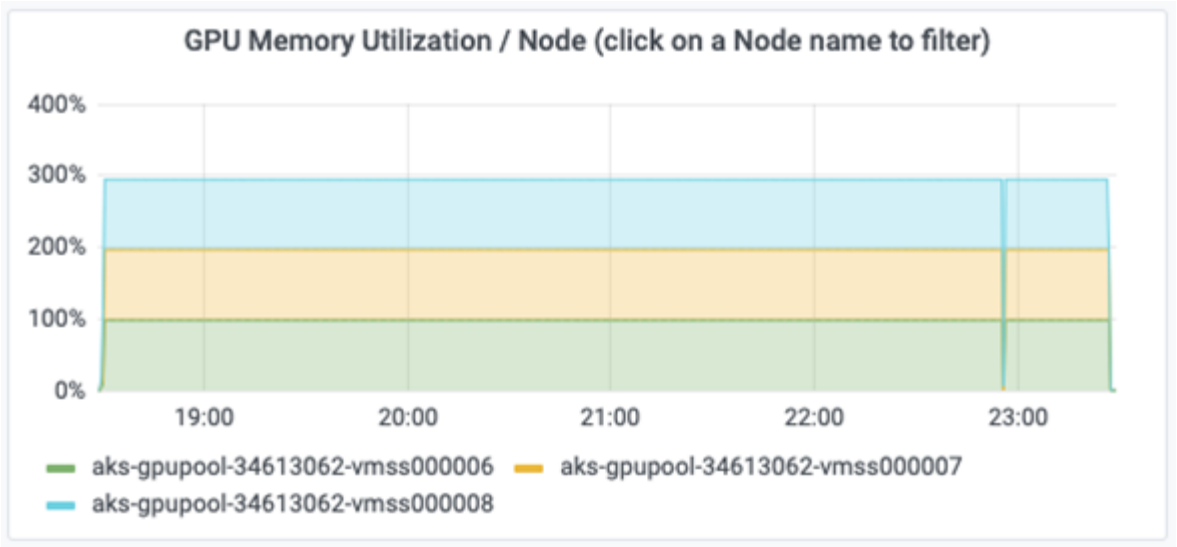
The figure below illustrates three GPUs across three nodes – GPUs allocation and memory.



The figure below illustrates three GPUs across three nodes utilization (%).



The figure below illustrates three GPUs across three nodes memory utilization (%).



Azure NetApp Files service levels

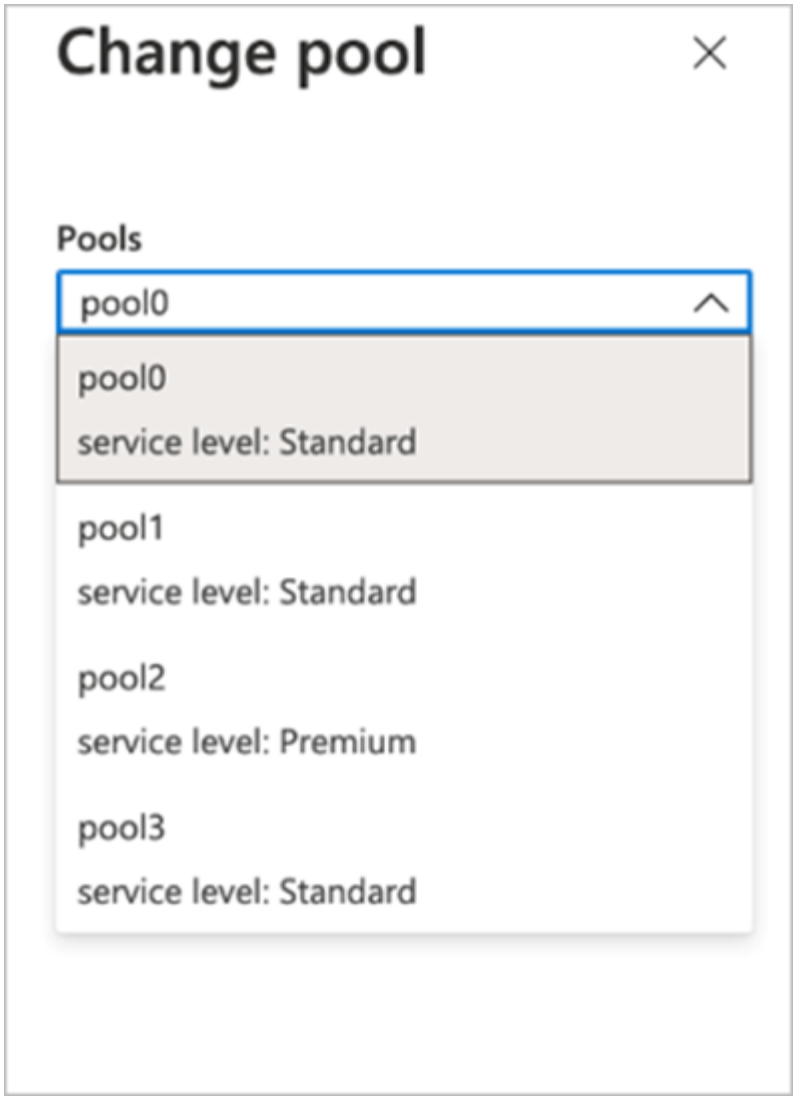
You can change the service level of an existing volume by moving the volume to another capacity pool that uses the service level you want for the volume. This existing service-level change for the volume does not require that you migrate data. It also does not affect access to the volume.

Dynamically change the service level of a volume

To change the service level of a volume, use the following steps:

1. On the Volumes page, right-click the volume whose service level you want to change. Select Change Pool.

2. In the Change Pool window, select the capacity pool you want to move the volume to. Then, click OK.



Automate service level change

Dynamic Service Level change is currently still in Public Preview, but it is not enabled by default. To enable this feature on the Azure subscription, follow these steps provided in the document " Dynamically change the service level of a volume."

- You can also use the following commands for Azure: CLI. For more information about changing the pool size of Azure NetApp Files, visit az netappfiles volume: Manage Azure NetApp Files (ANF) volume resources.

```
az netappfiles volume pool-change -g mygroup
--account-name myaccname
-pool-name mypoolname
--name myvolname
--new-pool-resource-id mynewresourceid
```

- The `set- aznetappfilesvolumepool` cmdlet shown here can change the pool of an Azure NetApp Files volume. More information about changing volume pool size and Azure PowerShell can be found by visiting Change pool for an Azure NetApp Files volume.

```
Set-AzNetAppFilesVolumePool
-ResourceGroupName "MyRG"
-AccountName "MyAnfAccount"
-PoolName "MyAnfPool"
-Name "MyAnfVolume"
-NewPoolResourceId 7d6e4069-6c78-6c61-7bf6-c60968e45fbf
```

Conclusion

NetApp and RUN: AI have partnered in the creation of this technical report to demonstrate the unique capabilities of the Azure NetApp Files together with the RUN: AI platform for simplifying orchestration of AI workloads. This technical report provides a reference architecture for streamlining the process of both data pipelines and workload orchestration for distributed lane detection training.

In conclusion, with regard to distributed training at scale (especially in a public cloud environment), the resource orchestration and storage component is a critical part of the solution. Making sure that data managing never hinders multiple GPU processing, therefore results in the optimal utilization of GPU cycles. Thus, making the system as cost effective as possible for large- scale distributed training purposes.

Data fabric delivered by NetApp overcomes the challenge by enabling data scientists and data engineers to connect together on-premises and in the cloud to have synchronous data, without performing any manual intervention. In other words, data fabric smooths the process of managing AI workflow spread across multiple locations. It also facilitates on demand-based data availability by bringing data close to compute and performing analysis, training, and validation wherever and whenever needed. This capability not only enables data integration but also protection and security of the entire data pipeline.

Additional information

To learn more about the information that is described in this document, review the following documents and/or websites:

- Dataset: TuSimple

  https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection

- Deep Learning Network Architecture: Spatial Convolutional Neural Network

  https://arxiv.org/abs/1712.06080

- Distributed deep learning training framework: Horovod

  https://horovod.ai/

- RUN: AI container orchestration solution: RUN: AI product introduction

  https://docs.run.ai/home/components/

- RUN: AI installation documentation

  https://docs.run.ai/Administrator/Cluster-Setup/cluster-install/#step-3-install-runai
  https://docs.run.ai/Administrator/Researcher-Setup/cli-install/#runai-cli-installation

- Submitting jobs in RUN: AI CLI

  https://docs.run.ai/Researcher/cli-reference/runai-submit/

  https://docs.run.ai/Researcher/cli-reference/runai-submit-mpi/

- Azure Cloud resources: Azure NetApp Files

  https://docs.microsoft.com/azure/azure-netapp-files/

- Azure Kubernetes Service

  https://azure.microsoft.com/services/kubernetes-service/-features

- Azure VM SKUs

  https://azure.microsoft.com/services/virtual-machines/

- Azure VM with GPU SKUs

  https://docs.microsoft.com/azure/virtual-machines/sizes-gpu

- NetApp Trident

  https://github.com/NetApp/trident/releases

- Data Fabric powered by NetApp

  https://www.netapp.com/data-fabric/what-is-data-fabric/

- NetApp Product Documentation

  https://www.netapp.com/support-and-training/documentation/

TR-4841: Hybrid Cloud AI Operating System with Data Caching

Rick Huang, David Arnette, NetApp
Yochay Ettun, cnvrg.io

The explosive growth of data and the exponential growth of ML and AI have converged to create a zettabyte economy with unique development and implementation challenges.

Although it is a widely known that ML models are data-hungry and require high-performance data storage proximal to compute resources, in practice, it is not so straight forward to implement this model, especially with hybrid cloud and elastic compute instances. Massive quantities of data are usually stored in low-cost data lakes, where high-performance AI compute resources such as GPUs cannot efficiently access it. This problem is aggravated in a hybrid-cloud infrastructure where some workloads operate in the cloud and some are located on-premises or in a different HPC environment entirely.

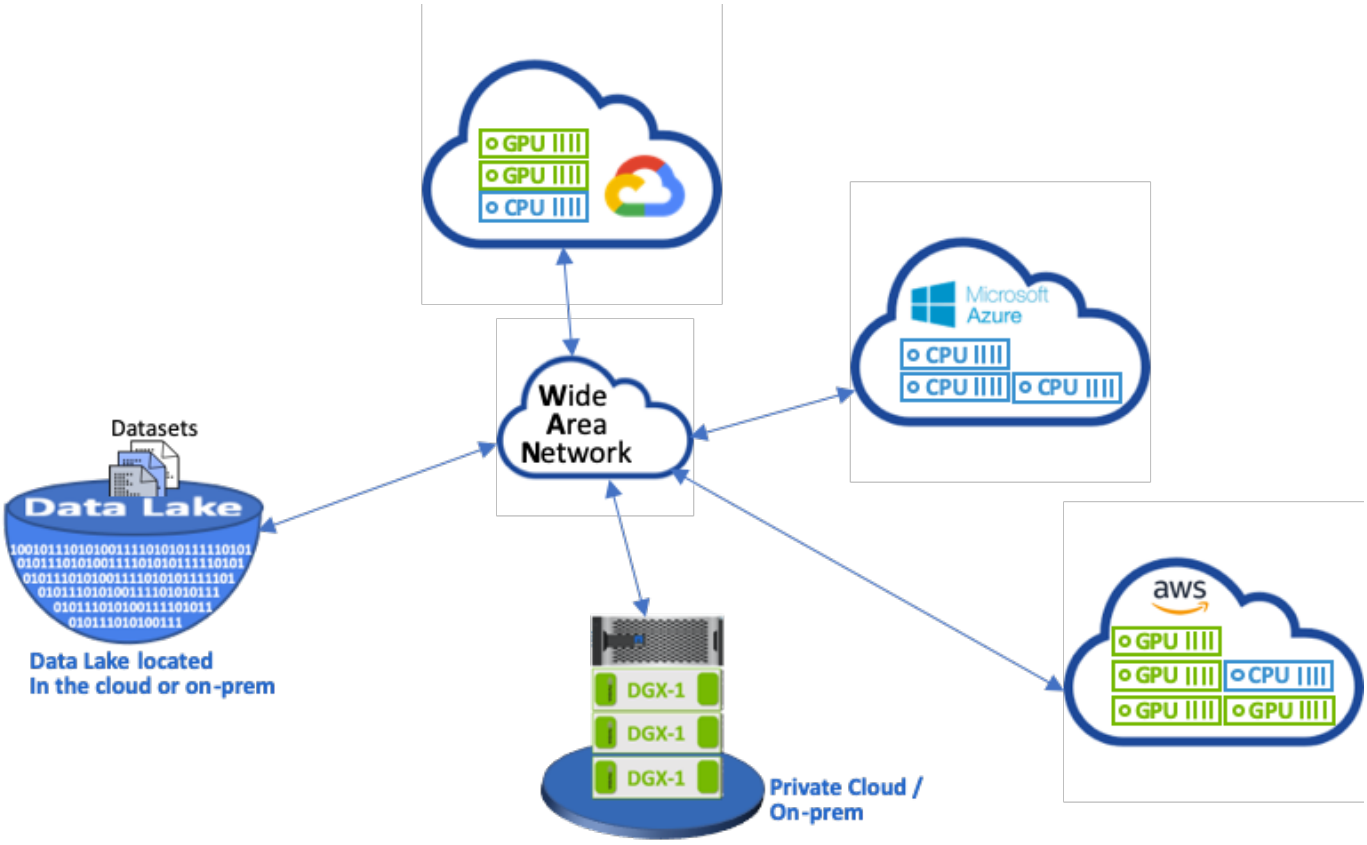
In this document, we present a novel solution that allows IT professionals and data engineers to create a truly hybrid cloud AI platform with a topology-aware data hub that enables data scientists to instantly and automatically create a cache of their datasets in proximity to their compute resources, wherever they are located. As a result, not only can high-performance model training be accomplished, but additional benefits are

created, including the collaboration of multiple AI practitioners, who have immediate access to dataset caches, versions, and lineages within a dataset version hub.

Next: Use Case Overview and Problem Statement

Use Case Overview and Problem Statement

Datasets and dataset versions are typically located in a data lake, such as NetApp StorageGrid object-based storage, which offers reduced cost and other operational advantages. Data scientists pull these datasets and engineer them in multiple steps to prepare them for training with a specific model, often creating multiple versions along the way. As the next step, the data scientist must pick optimized compute resources (GPUs, high-end CPU instances, an on-premises cluster, and so on) to run the model. The following figure depicts the lack of dataset proximity in an ML compute environment.



However, multiple training experiments must run in parallel in different compute environments, each of which require a download of the dataset from the data lake, which is an expensive and time-consuming process. Proximity of the dataset to the compute environment (especially for a hybrid cloud) is not guaranteed. In addition, other team members that run their own experiments with the same dataset must go through the same arduous process. Beyond the obvious slow data access, challenges include difficulties tracking dataset versions, dataset sharing, collaboration, and reproducibility.

Customer Requirements

Customer requirements can vary in order to achieve high- performance ML runs while efficiently using resources; for example, customers might require the following:

- Fast access to datasets from each compute instance executing the training model without incurring expensive downloads and data access complexities
- The use any compute instance (GPU or CPU) in the cloud or on-premises without concern for the location of the datasets
- Increased efficiency and productivity by running multiple training experiments in parallel with different compute resources on the same dataset without unnecessary delays and data latency
- Minimized compute instance costs
- Improved reproducibility with tools to keep records of the datasets, their lineage, versions, and other metadata details
- Enhanced sharing and collaboration so that any authorized member of the team can access the datasets and run experiments

To implement dataset caching with NetApp ONTAP data management software, customers must perform the following tasks:

- Configure and set the NFS storage that is closest to the compute resources.
- Determine which dataset and version to cache.
- Monitor the total memory committed to cached datasets and how much NFS storage is available for additional cache commits (for example, cache management).
- Age out of datasets in the cache if they have not been used in certain time. The default is one day; other configuration options are available.

Next: Solution Overview

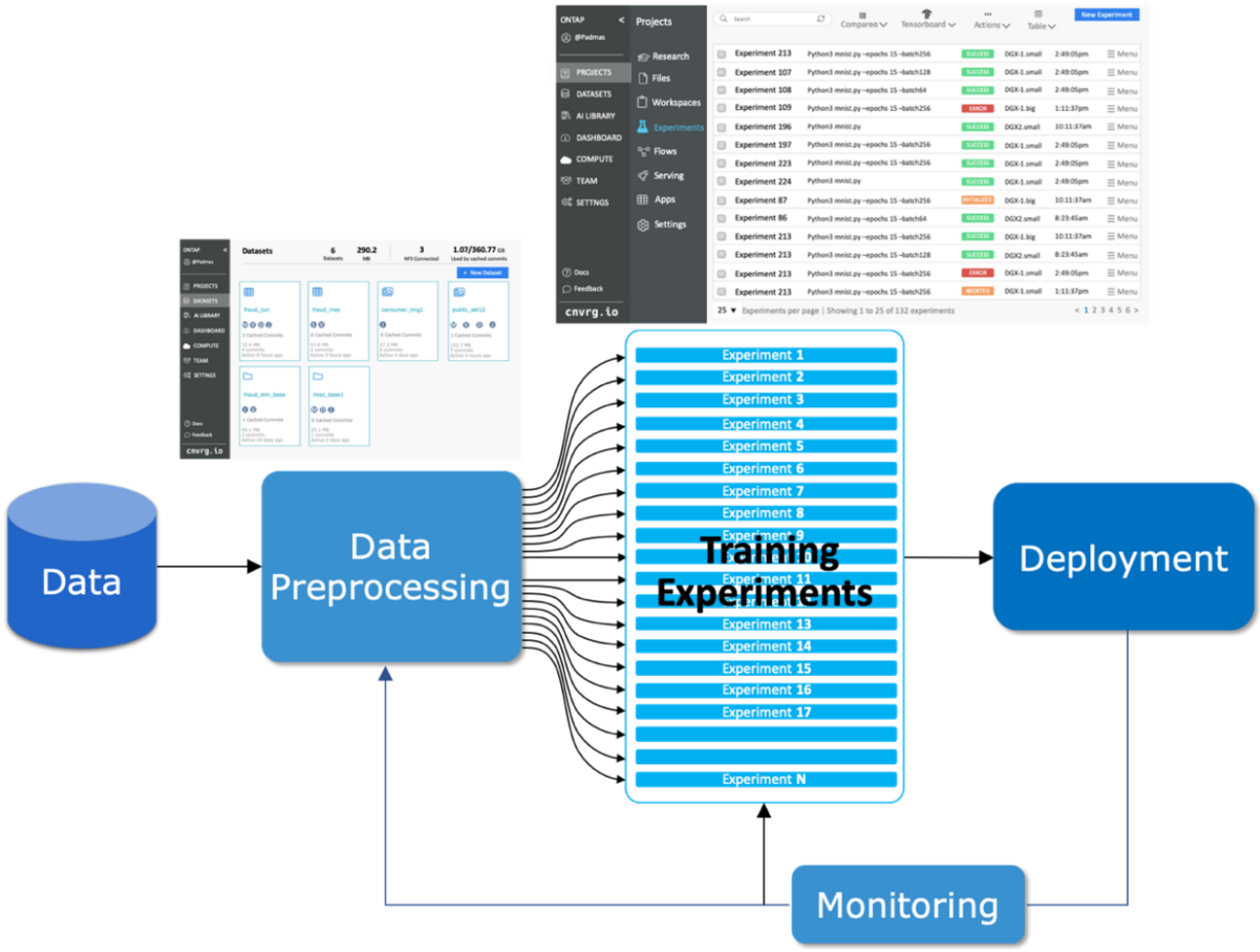
Solution Overview

This section reviews a conventional data science pipeline and its drawbacks. It also presents the architecture of the proposed dataset caching solution.

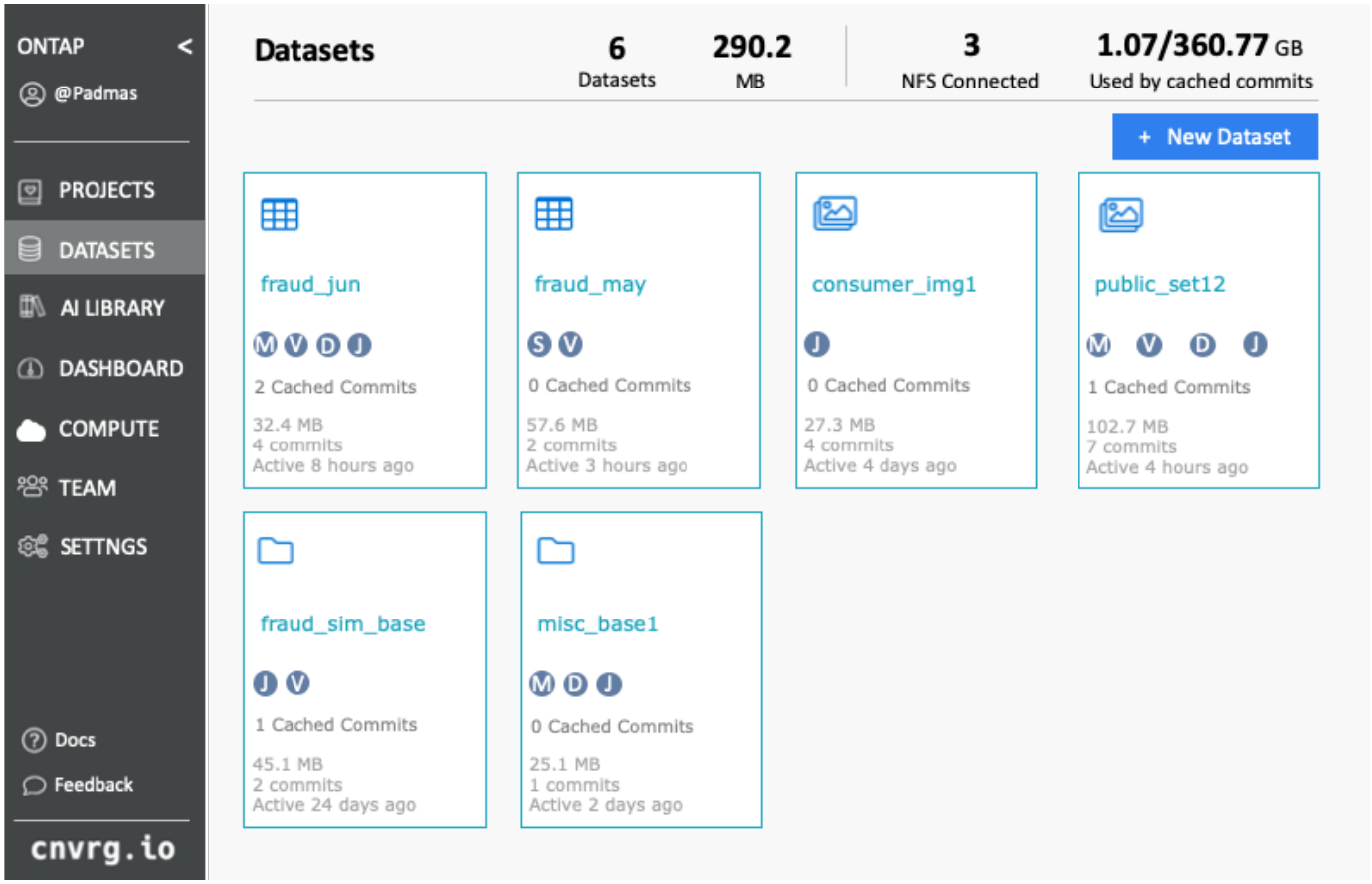
Conventional Data Science Pipeline and Drawbacks

A typical sequence of ML model development and deployment involves iterative steps that include the following:

- Ingesting data
- Data preprocessing (creating multiple versions of the datasets)
- Running multiple experiments involving hyperparameter optimization, different models, and so on
- Deployment
- Monitoringcnvrg.io has developed a comprehensive platform to automate all tasks from research to deployment. A small sample of dashboard screenshots pertaining to the pipeline is shown in the following figure.



It is very common to have multiple datasets in play from public repositories and private data. In addition, each dataset is likely to have multiple versions resulting from dataset cleanup or feature engineering. A dashboard that provides a dataset hub and a version hub is needed to make sure collaboration and consistency tools are available to the team, as can be seen in the following figure.



The next step in the pipeline is training, which requires multiple parallel instances of training models, each associated with a dataset and a certain compute instance. The binding of a dataset to a certain experiment with a certain compute instance is a challenge because it is possible that some experiments are performed by GPU instances from Amazon Web Services (AWS), while other experiments are performed by DGX-1 or DGX-2 instances on- premises. Other experiments might be executed in CPU servers in GCP, while the dataset location is not in reasonable proximity to the compute resources performing the training. A reasonable proximity would have full 10GbE or more low-latency connectivity from the dataset storage to the compute instance.

It is a common practice for data scientists to download the dataset to the compute instance performing the training and execute the experiment. However, there are several potential problems with this approach:

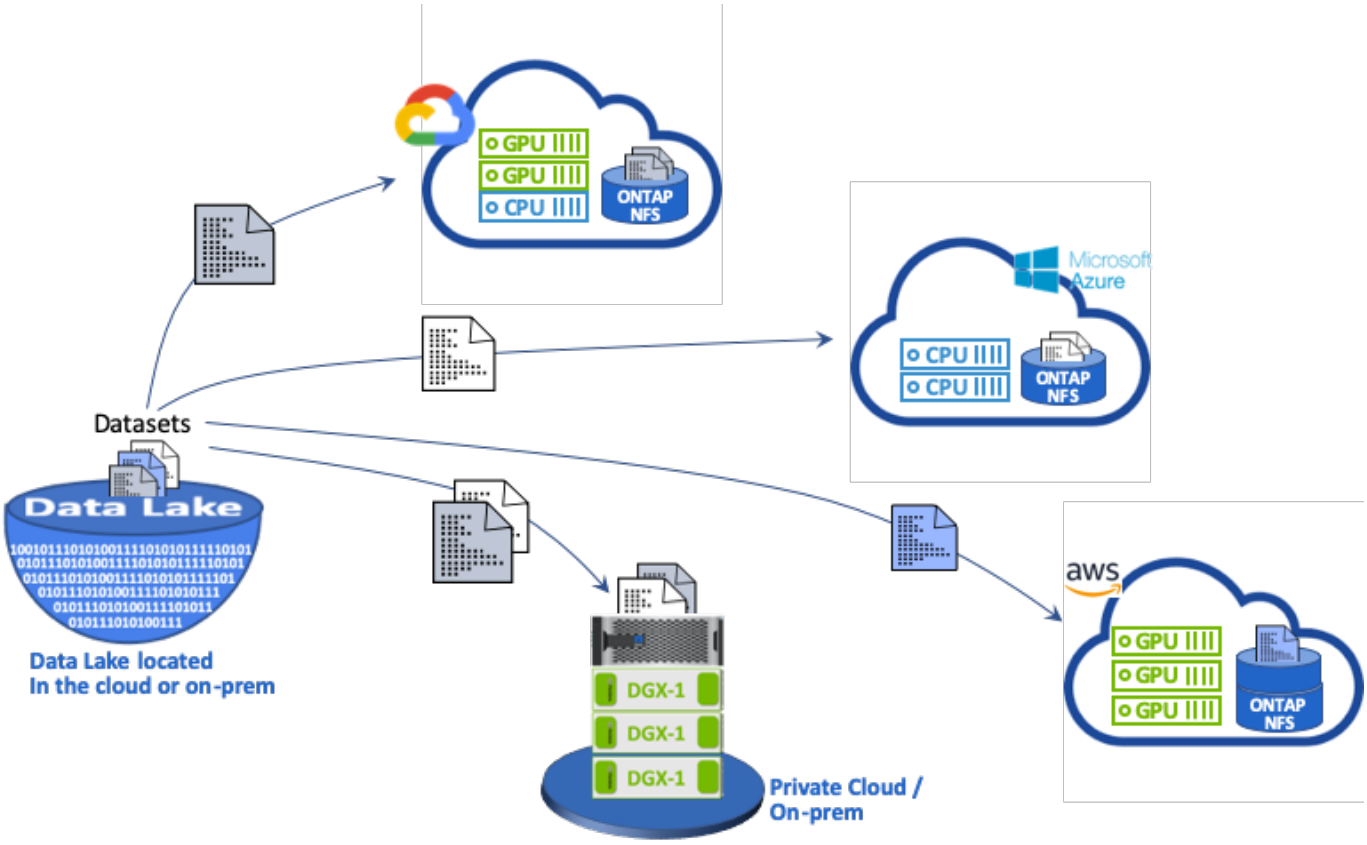
- When the data scientist downloads the dataset to a compute instance, there are no guarantees that the integrated compute storage is high performance (an example of a high-performance system would be the ONTAP AFF A800 NVMe solution).
- When the downloaded dataset resides in one compute node, storage can become a bottleneck when distributed models are executed over multiple nodes (unlike with NetApp ONTAP high-performance distributed storage).
- The next iteration of the training experiment might be performed in a different compute instance due to queue conflicts or priorities, again creating significant network distance from the dataset to the compute location.
- Other team members executing training experiments on the same compute cluster cannot share this dataset; each performs the (expensive) download of the dataset from an arbitrary location.
- If other datasets or versions of the same dataset are needed for the subsequent training jobs, the data scientists must again perform the (expensive) download of the dataset to the compute instance performing the training.NetApp and cnvrg.io have created a new dataset caching solution that eliminates these

hurdles. The solution creates accelerated execution of the ML pipeline by caching hot datasets on the ONTAP high- performance storage system. With ONTAP NFS, the datasets are cached once (and only once) in a data fabric powered by NetApp (such as AFF A800), which is collocated with the compute. As the NetApp ONTAP NFS high-speed storage can serve multiple ML compute nodes, the performance of the training models is optimized, bringing cost savings, productivity, and operational efficiency to the organization.

Solution Architecture

This solution from NetApp and cnvrg.io provides dataset caching, as shown in the following figure. Dataset caching allows data scientists to pick a desired dataset or dataset version and move it to the ONTAP NFS cache, which lies in proximity to the ML compute cluster. The data scientist can now run multiple experiments without incurring delays or downloads. In addition, all collaborating engineers can use the same dataset with the attached compute cluster (with the freedom to pick any node) without additional downloads from the data lake. The data scientists are offered a dashboard that tracks and monitors all datasets and versions and provides a view of which datasets were cached.

The cnvrg.io platform auto-detects aged datasets that have not been used for a certain time and evicts them from the cache, which maintains free NFS cache space for more frequently used datasets. It is important to note that dataset caching with ONTAP works in the cloud and on-premises, thus providing maximum flexibility.



Next: Concepts and Components

Concepts and Components

This section covers concepts and components associated with data caching in an ML workflow.

Machine Learning

ML is rapidly becoming essential to many businesses and organizations around the world. Therefore, IT and DevOps teams are now facing the challenge of standardizing ML workloads and provisioning cloud, on-premises, and hybrid compute resources that support the dynamic and intensive workflows that ML jobs and pipelines require.

Container-Based Machine Learning and Kubernetes

Containers are isolated user-space instances that run on top of a shared host operating system kernel. The adoption of containers is rapidly increasing. 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.

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

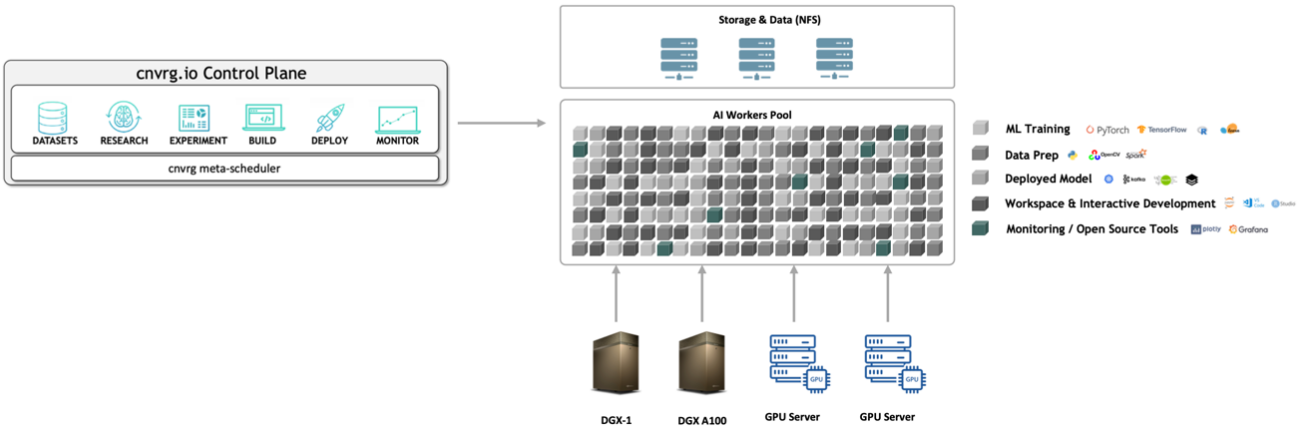
Kubernetes, the popular container orchestrator, allows data scientists to launch flexible, container-based jobs and pipelines. It also enables infrastructure teams to manage and monitor ML workloads in a single managed and cloud-native environment. For more information, visit the Kubernetes website.

cnvrg.io

cnvrg.io is an AI operating system that transforms the way enterprises manage, scale, and accelerate AI and data science development from research to production. The code-first platform is built by data scientists for data scientists and offers flexibility to run on-premises or in the cloud. With model management, MLOps, and continual ML solutions, cnvrg.io brings top- of- the- line technology to data science teams so they can spend less time on DevOps and focus on the real magic—algorithms. Since using cnvrg.io, teams across industries have gotten more models to production resulting in increased business value.

cnvrg.io Meta-Scheduler

cnvrg. io has a unique architecture that allows IT and engineers to attach different compute resources to the same control plane and have cnvrg.io manage ML jobs across all resources. This means that IT can attach multiple on-premises Kubernetes clusters, VM servers, and cloud accounts and run ML workloads on all resources, as shown in the following figure.



cnvrg.io Data Caching

cnvrg.io allows data scientists to define hot and cold dataset versions with its data-caching technology. By default, datasets are stored in a centralized object storage database. Then, data scientists can cache a specific data version on the selected compute resource to save time on download and therefor increase ML development and productivity. Datasets that are cached and are not in use for a few days are automatically cleared from the selected NFS. Caching and clearing the cache can be performed with a single click; no coding, IT, or DevOps work is required.

cnvrg.io Flows and ML Pipelines

cnvrg.io Flows is a tool for building production ML pipelines. Each component in a flow is a script/code running on a selected compute with a base docker image. This design enables data scientists and engineers to build a single pipeline that can run both on-premises and in the cloud. cnvrg.io makes sure data, parameters, and artifacts are moving between the different components. In addition, each flow is monitored and tracked for 100% reproducible data science.

cnvrg.io CORE

cnvrg.io CORE is a free platform for the data science community to help data scientists focus more on data science and less on DevOps. CORE's flexible infrastructure gives data scientists the control to use any language, AI framework, or compute environment whether on- premises or in the cloud so they can do what they do best, build algorithms. cnvrg.io CORE can be easily installed with a single command on any Kubernetes cluster.

NetApp ONTAP AI

ONTAP AI is a data center reference architecture for ML and deep learning (DL) workloads that uses NetApp AFF storage systems and NVIDIA DGX systems with Tesla V100 GPUs. ONTAP AI is based on the industry-standard NFS file protocol over 100Gb Ethernet, providing customers with a high-performance ML/DL infrastructure that uses standard data center technologies to reduce implementation and administration overhead. Using standardized network and protocols enables ONTAP AI to integrate into hybrid cloud environments while maintaining operational consistency and simplicity. As a prevalidated infrastructure solution, ONTAP AI reduces deployment time and risk and reduces administration overhead significantly, allowing customers to realize faster time to value.

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.

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—it 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.

NetApp StorageGRID

NetApp StorageGRID is a software-defined object storage platform designed to meet these needs by providing simple, cloud-like storage that users can access using the S3 protocol. StorageGRID is a scale-out system designed to support multiple nodes across internet-connected sites, regardless of distance. With the intelligent policy engine of StorageGRID, users can choose erasure-coding objects across sites for geo-resiliency or object replication between remote sites to minimize WAN access latency. StorageGrid provides an excellent private-cloud primary object storage data lake in this solution.

NetApp Cloud Volumes ONTAP

NetApp Cloud Volumes ONTAP data management software delivers control, protection, and efficiency to user data with the flexibility of public cloud providers including AWS, Google Cloud Platform, and Microsoft Azure. Cloud Volumes ONTAP is cloud-native data management software built on the NetApp ONTAP storage software, providing users with a superior universal storage platform that addresses their cloud data needs. Having the same storage software in the cloud and on- premises provides users with the value of a data fabric without having to train IT staff in all-new methods to manage data.

For customers that are interested in hybrid cloud deployment models, Cloud Volumes ONTAP can provide the same capabilities and class-leading performance in most public clouds to provide a consistent and seamless user experience in any environment.

Next: Hardware and Software Requirements

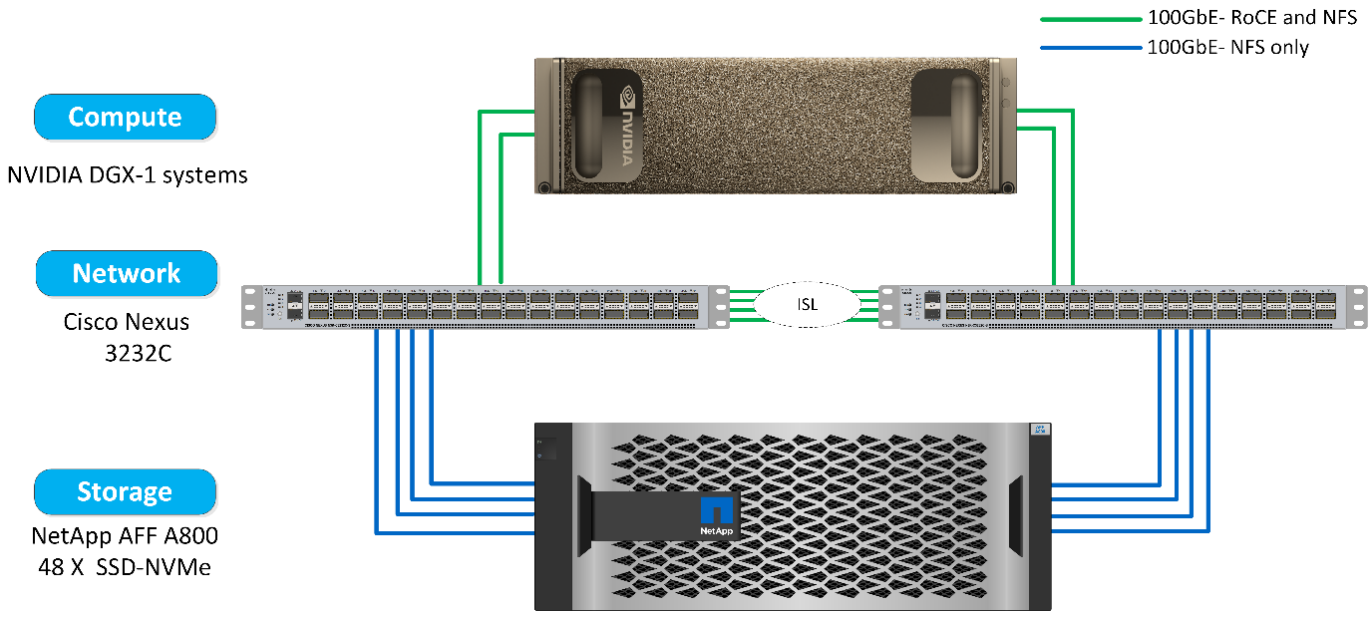
Hardware and Software Requirements

This section covers the technology requirements for the ONTAP AI solution.

Hardware Requirements

Although hardware requirements depend on specific customer workloads, ONTAP AI can be deployed at any scale for data engineering, model training, and production inferencing from a single GPU up to rack-scale configurations for large-scale ML/DL operations. For more information about ONTAP AI, see the ONTAP AI website.

This solution was validated using a DGX-1 system for compute, a NetApp AFF A800 storage system, and Cisco Nexus 3232C for network connectivity. The AFF A800 used in this validation can support as many as 10 DGX-1 systems for most ML/DL workloads. The following figure shows the ONTAP AI topology used for model training in this validation.



To extend this solution to a public cloud, Cloud Volumes ONTAP can be deployed alongside cloud GPU compute resources and integrated into a hybrid cloud data fabric that enables customers to use whatever resources are appropriate for any given workload.

Software Requirements

The following table shows the specific software versions used in this solution validation.

Component	Version
Ubuntu	18.04.4 LTS
NVIDIA DGX OS	4.4.0
NVIDIA DeepOps	20.02.1
Kubernetes	1.15
Helm	3.1.0
cnvrg.io	3.0.0
NetApp ONTAP	9.6P4

For this solution validation, Kubernetes was deployed as a single-node cluster on the DGX-1 system. For large-scale deployments, independent Kubernetes master nodes should be deployed to provide high availability of management services as well as reserve valuable DGX resources for ML and DL workloads.

Next: Solution Deployment and Validation Details

Solution Deployment and Validation Details

The following sections discuss the details of solution deployment and validation.

Next: ONTAP AI Deployment

ONTAP AI Deployment

Deployment of ONTAP AI requires the installation and configuration of networking, compute, and storage hardware. Specific instructions for deployment of the ONTAP AI infrastructure are beyond the scope of this document. For detailed deployment information, see NVA-1121-DEPLOY: NetApp ONTAP AI, Powered by NVIDIA.

For this solution validation, a single volume was created and mounted to the DGX-1 system. That mount point was then mounted to the containers to make data accessible for training. For large-scale deployments, NetApp Trident automates the creation and mounting of volumes to eliminate administrative overhead and enable end-user management of resources.

Next: Kubernetes Deployment

Kubernetes Deployment

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

1. Download NVIDIA DeepOps by following the instructions on the Getting Started page on the NVIDIA DeepOps GitHub site.
2. Deploy Kubernetes in your cluster by following the instructions on the Kubernetes Deployment Guide on the NVIDIA DeepOps GitHub site.

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 running the `which kubectl` command directly on that node.

Next: Cnvrg.io Deployment

cnvrg.io Deployment

Deploy cnvrg CORE Using Helm

Helm is the easiest way to quickly deploy cnvrg using any cluster, on-premises, Minikube, or on any cloud cluster (such as AKS, EKS, and GKE). This section describes how cnvrg was installed on an on-premises (DGX-1) instance with Kubernetes installed.

Prerequisites

Before you can complete the installation, you must install and prepare the following dependencies on your

local machine:

- Kubectl
- Helm 3.x
- Kubernetes cluster 1.15+

Deploy Using Helm

1. To download the most updated cnvrg helm charts, run the following command:

```
helm repo add cnvrg https://helm.cnvrg.io
helm repo update
```

2. Before you deploy cnvrg, you need the external IP address of the cluster and the name of the node on which you will deploy cnvrg. To deploy cnvrg on an on-premises Kubernetes cluster, run the following command:

```
helm install cnvrg cnvrg/cnvrg --timeout 1500s  --wait \ --set
global.external_ip=<ip_of_cluster> \ --set global.node=<name_of_node>
```

3. Run the `helm install` command. All the services and systems automatically install on your cluster. The process can take up to 15 minutes.
4. The `helm install` command can take up to 10 minutes. When the deployment completes, go to the URL of your newly deployed cnvrg or add the new cluster as a resource inside your organization. The `helm` command informs you of the correct URL.

```
Thank you for installing cnvrg.io!
Your installation of cnvrg.io is now available, and can be reached via:
Talk to our team via email at
```

5. When the status of all the containers is running or complete, cnvrg has been successfully deployed. It should look similar to the following example output:

```
NAME                                READY   STATUS      RESTARTS   AGE
cnvrg-app-69fbb9df98-6xrgf                  1/1     Running     0          2m
cnvrg-sidekiq-b9d54d889-5x4fc               1/1     Running     0          2m
controller-65895b47d4-s96v6                 1/1     Running     0          2m
init-app-vs-config-wv9c4                    0/1     Completed   0          9m
init-gateway-vs-config-2zbpp                0/1     Completed   0          9m
init-minio-vs-config-cd2rg                  0/1     Completed   0          9m
minio-0                                     1/1     Running     0          2m
postgres-0                                  1/1     Running     0          2m
redis-695c49c986-kcbt9                      1/1     Running     0          2m
seeder-wh655                                0/1     Completed   0          2m
speaker-5sghr                               1/1     Running     0          2m
```

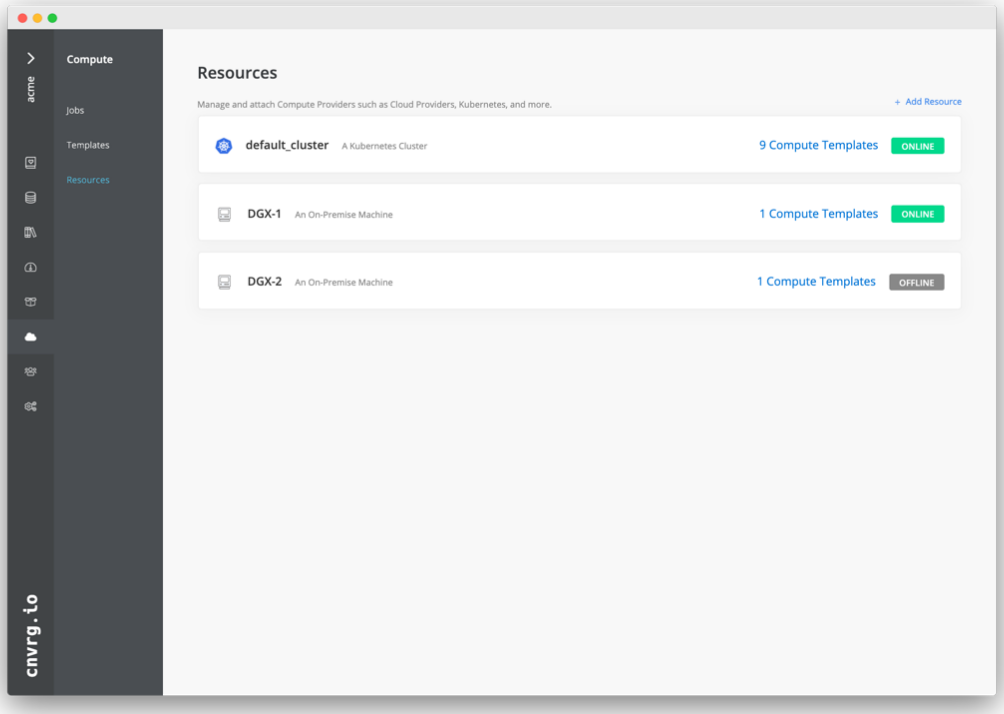
Computer Vision Model Training with ResNet50 and the Chest X-ray Dataset

cnvrg.io AI OS was deployed on a Kubernetes setup on a NetApp ONTAP AI architecture powered by the NVIDIA DGX system. For validation, we used the NIH Chest X-ray dataset consisting of de-identified images of chest x-rays. The images were in the PNG format. The data was provided by the NIH Clinical Center and is available through the NIH download site. We used a 250GB sample of the data with 627, 615 images across 15 classes.

The dataset was uploaded to the cnvrg platform and was cached on an NFS export from the NetApp AFF A800 storage system.

Set up the Compute Resources

The cnvrg architecture and meta-scheduling capability allow engineers and IT professionals to attach different compute resources to a single platform. In our setup, we used the same cluster cnvrg that was deployed for running the deep-learning workloads. If you need to attach additional clusters, use the GUI, as shown in the following screenshot.

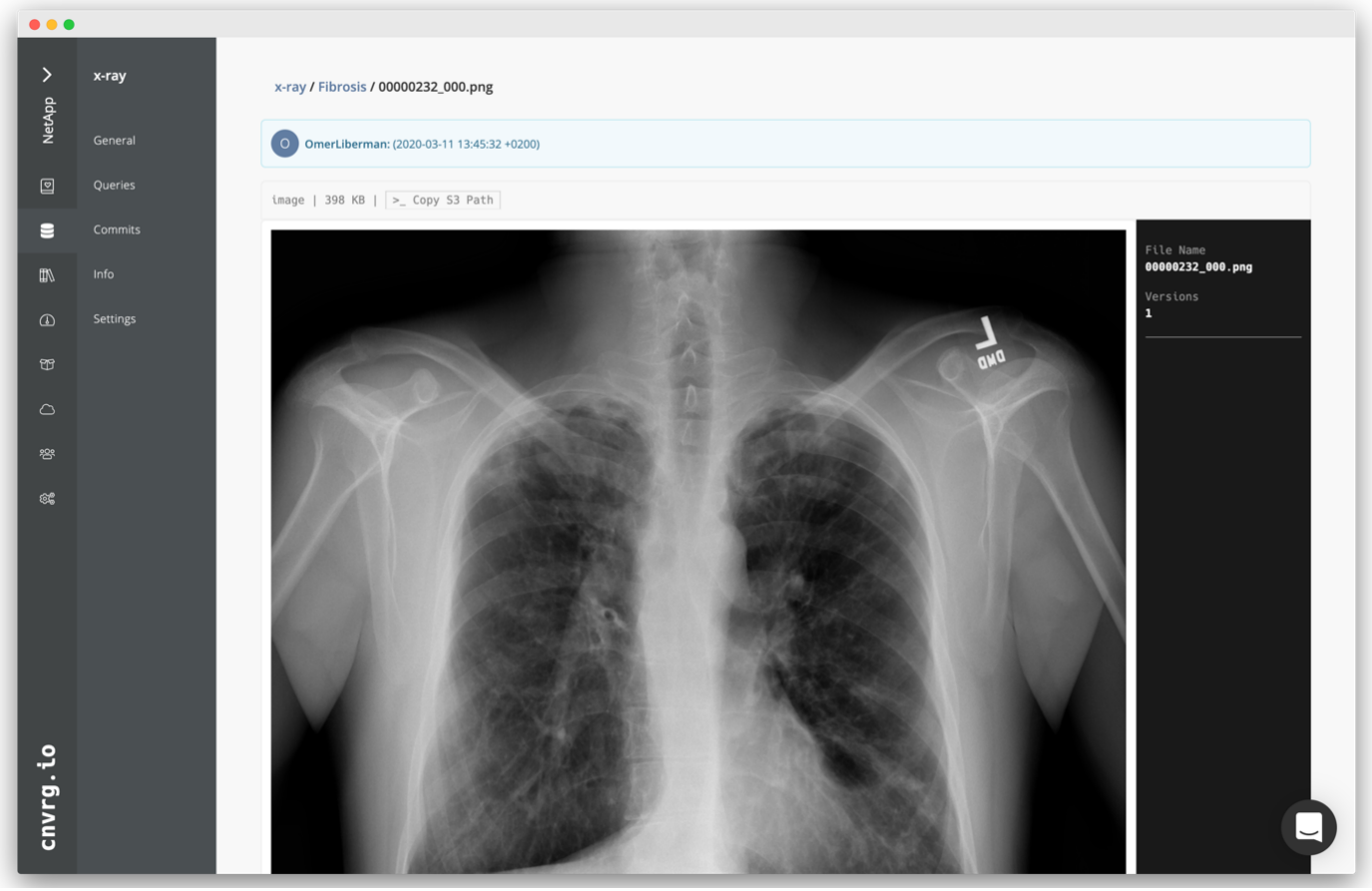


Load Data

To upload data to the cnvrg platform, you can use the GUI or the cnvrg CLI. For large datasets, NetApp recommends using the CLI because it is a strong, scalable, and reliable tool that can handle a large number of files.

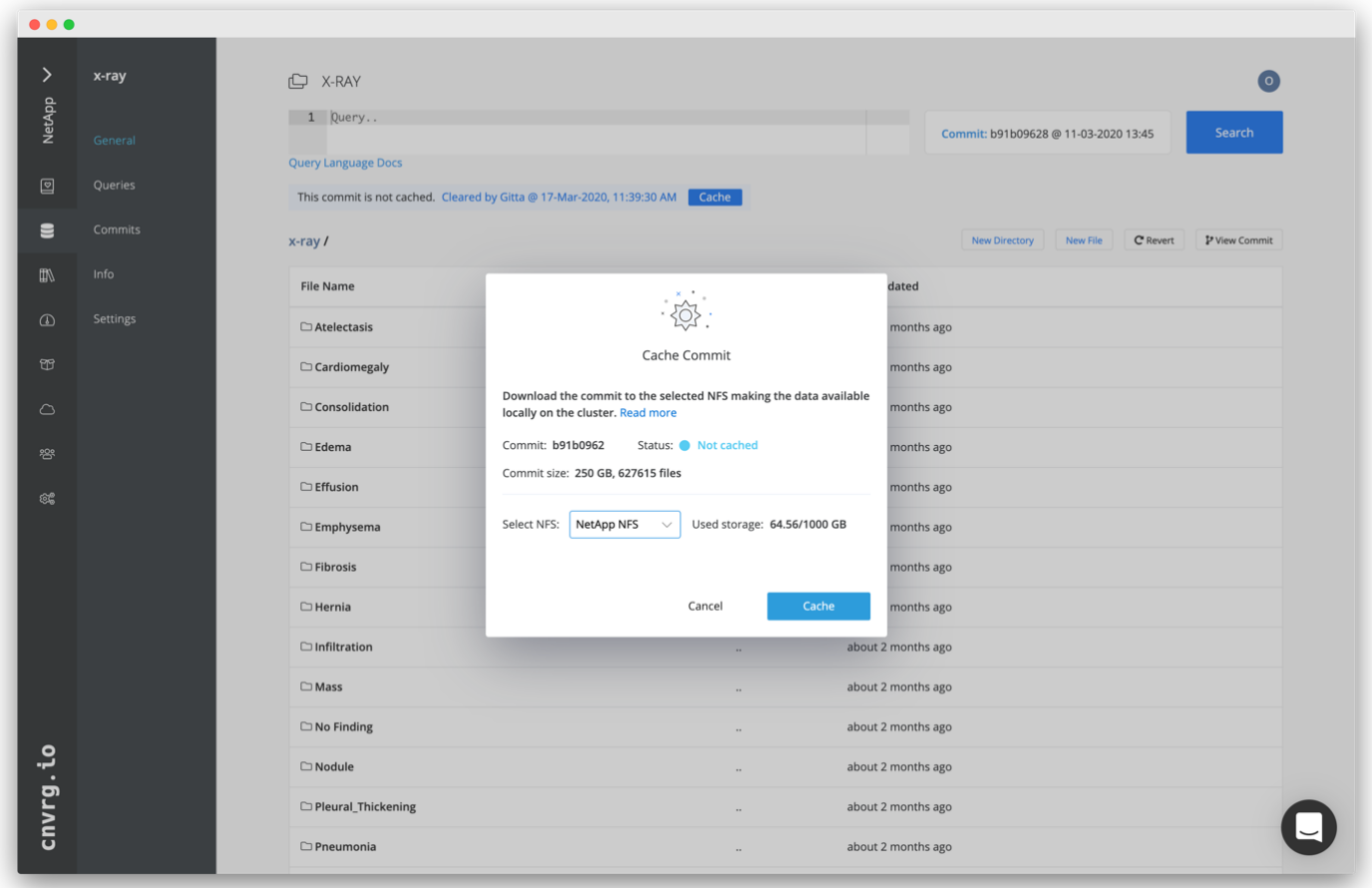
To upload data, complete the following steps:

1. Download the cnvrg CLI.
2. navigate to the x-ray directory.
3. Initialize the dataset in the platform with the `cnvrg data init` command.
4. Upload all contents of the directory to the central data lake with the `cnvrg data sync` command.After the data is uploaded to the central object store (StorageGRID, S3, or others), you can browse with the GUI. The following figure shows a loaded chest X-ray fibrosis image PNG file. In addition, cnvrg versions the data so that any model you build can be reproduced down to the data version.



Cach Data

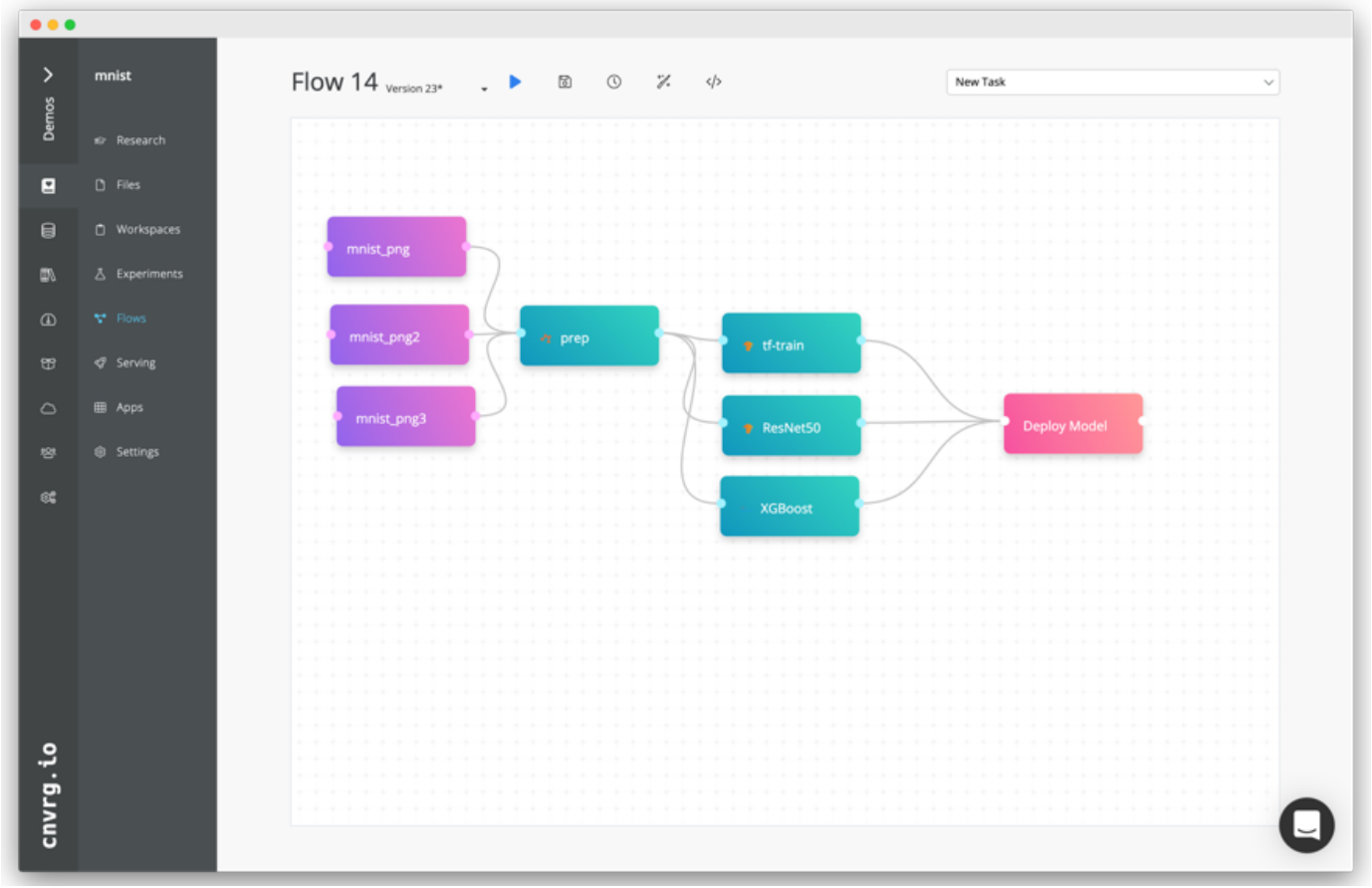
To make training faster and avoid downloading 600k+ files for each model training and experiment, we used the data-caching feature after data was initially uploaded to the central data-lake object store.



After users click Cache, cnvrg downloads the data in its specific commit from the remote object store and caches it on the ONTAP NFS volume. After it completes, the data is available for instant training. In addition, if the data is not used for a few days (for model training or exploration, for example), cnvrg automatically clears the cache.

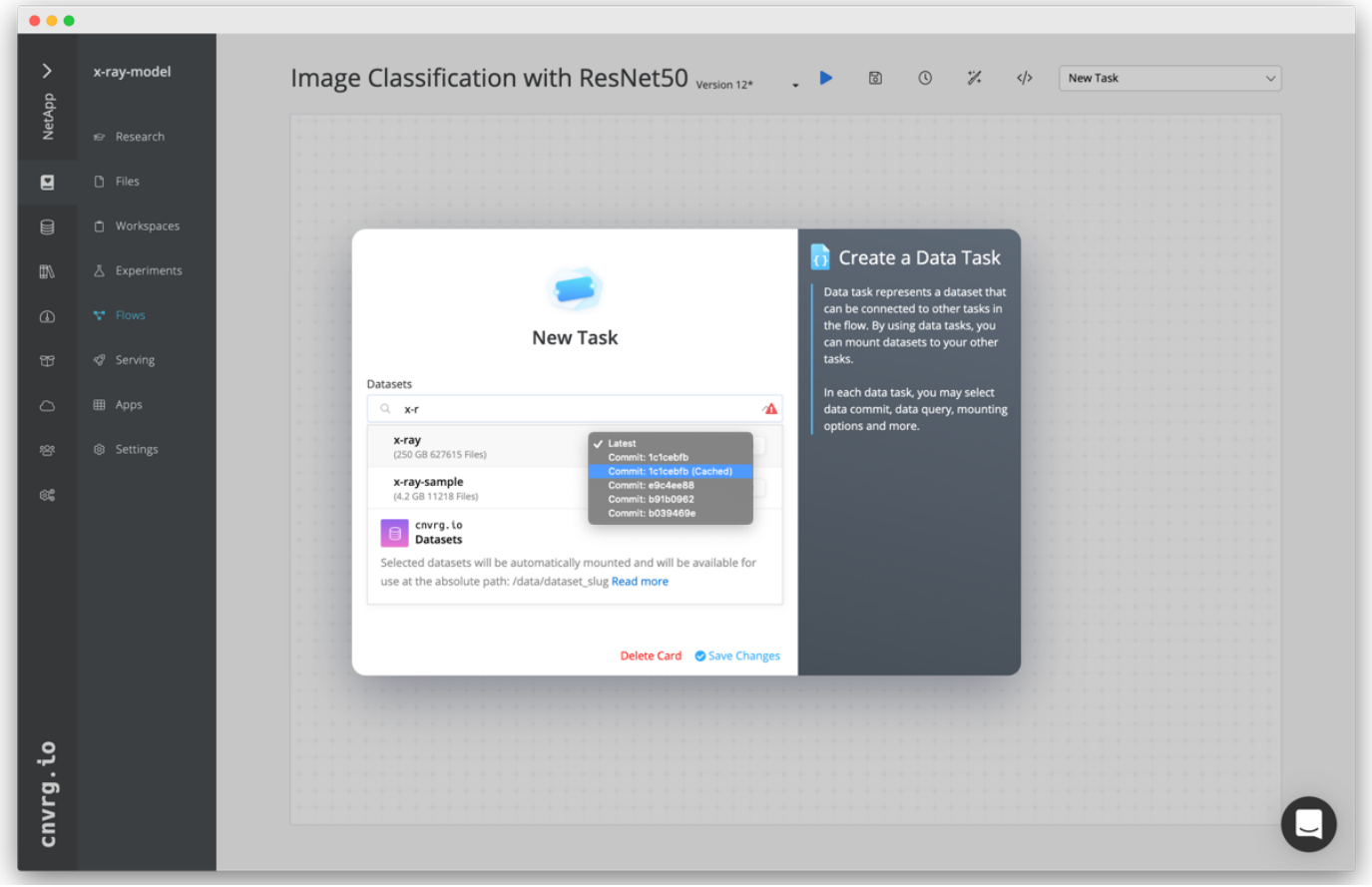
Build an ML Pipeline with Cached Data

cnvrg flows allows you to easily build production ML pipelines. Flows are flexible, can work for any kind of ML use case, and can be created through the GUI or code. Each component in a flow can run on a different compute resource with a different Docker image, which makes it possible to build hybrid cloud and optimized ML pipelines.



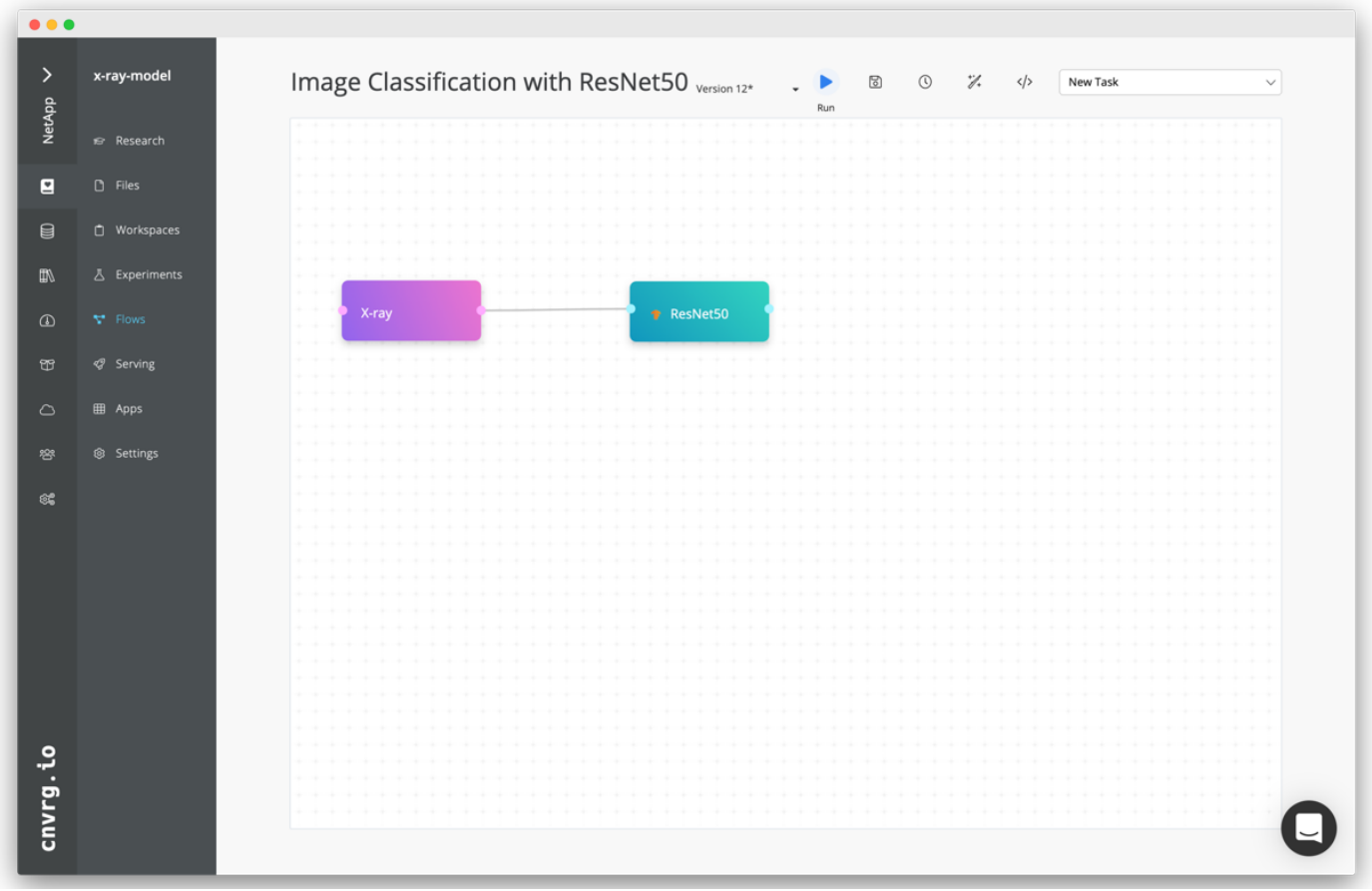
Building the Chest X-ray Flow: Setting Data

We added our dataset to a newly created flow. When adding the dataset, you can select the specific version (commit) and indicate whether you want the cached version. In this example, we selected the cached commit.



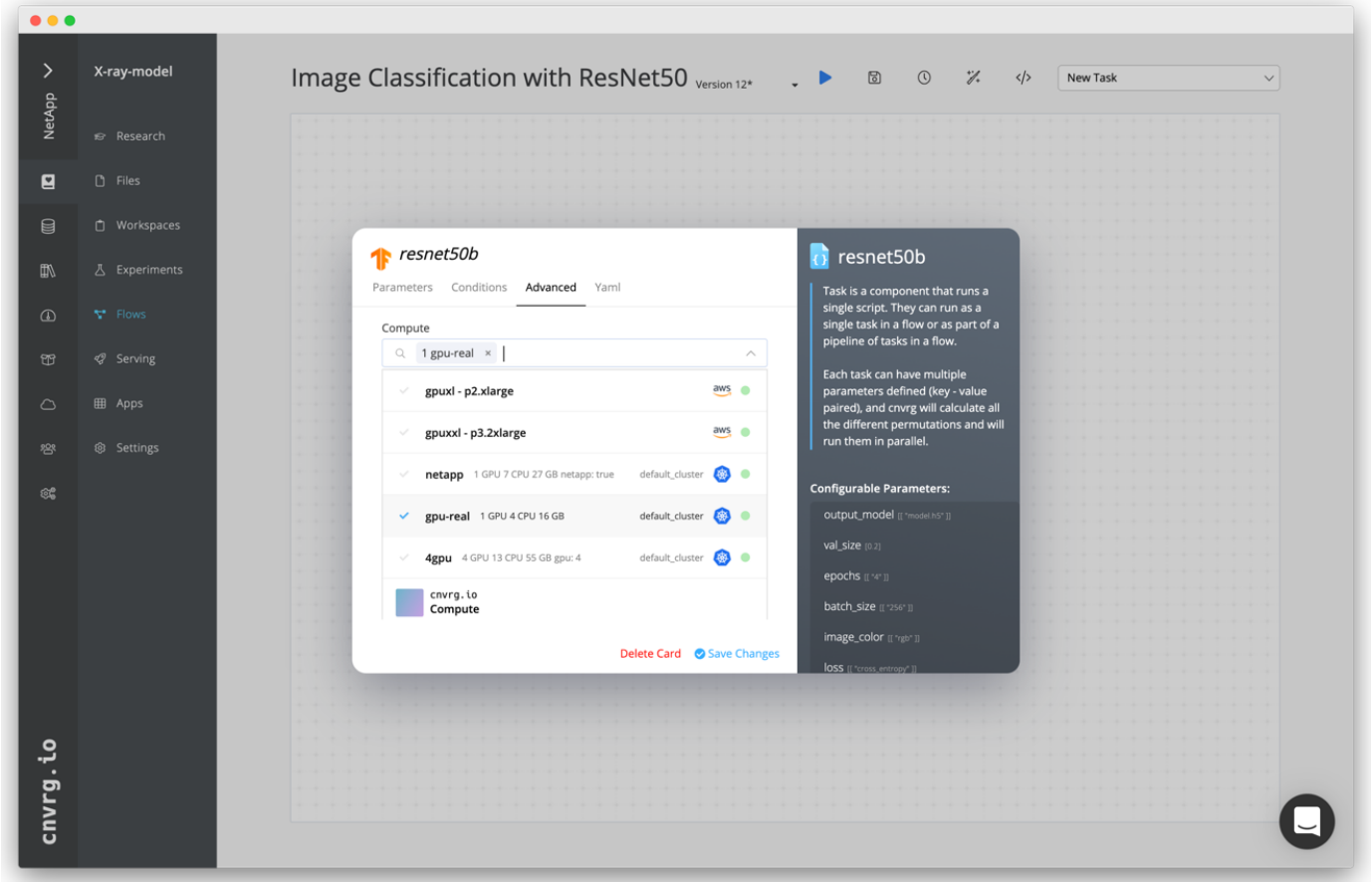
Building the Chest X-ray Flow: Setting Training Model: ResNet50

In the pipeline, you can add any kind of custom code you want. In cnvrg, there is also the AI library, a reusable ML components collection. In the AI library, there are algorithms, scripts, data sources, and other solutions that can be used in any ML or deep learning flow. In this example, we selected the prebuilt ResNet50 module. We used default parameters such as batch_size:128, epochs:10, and more. These parameters can be viewed in the AI Library docs. The following screenshot shows the new flow with the X-ray dataset connected to ResNet50.



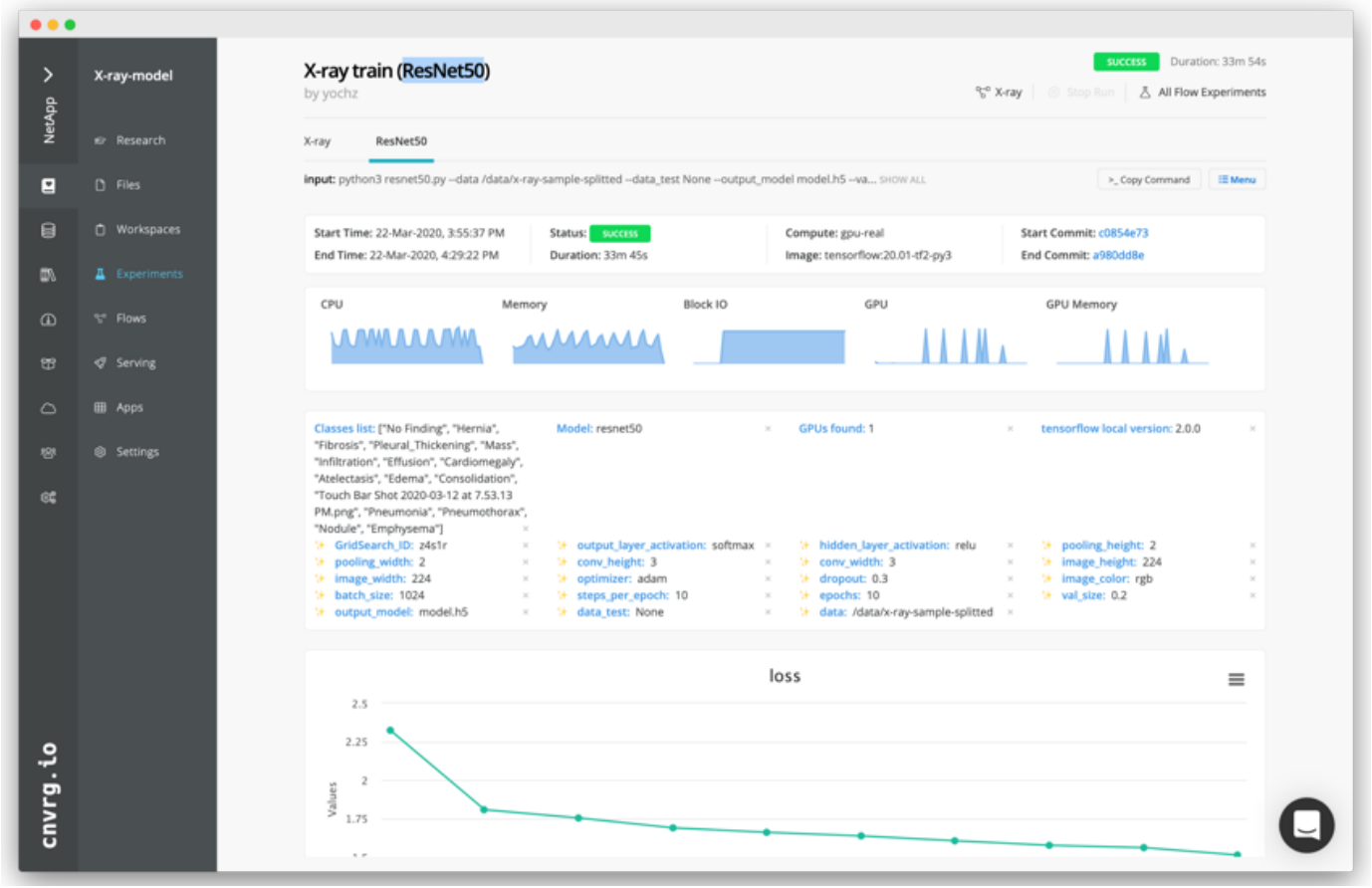
Define the Compute Resource for ResNet50

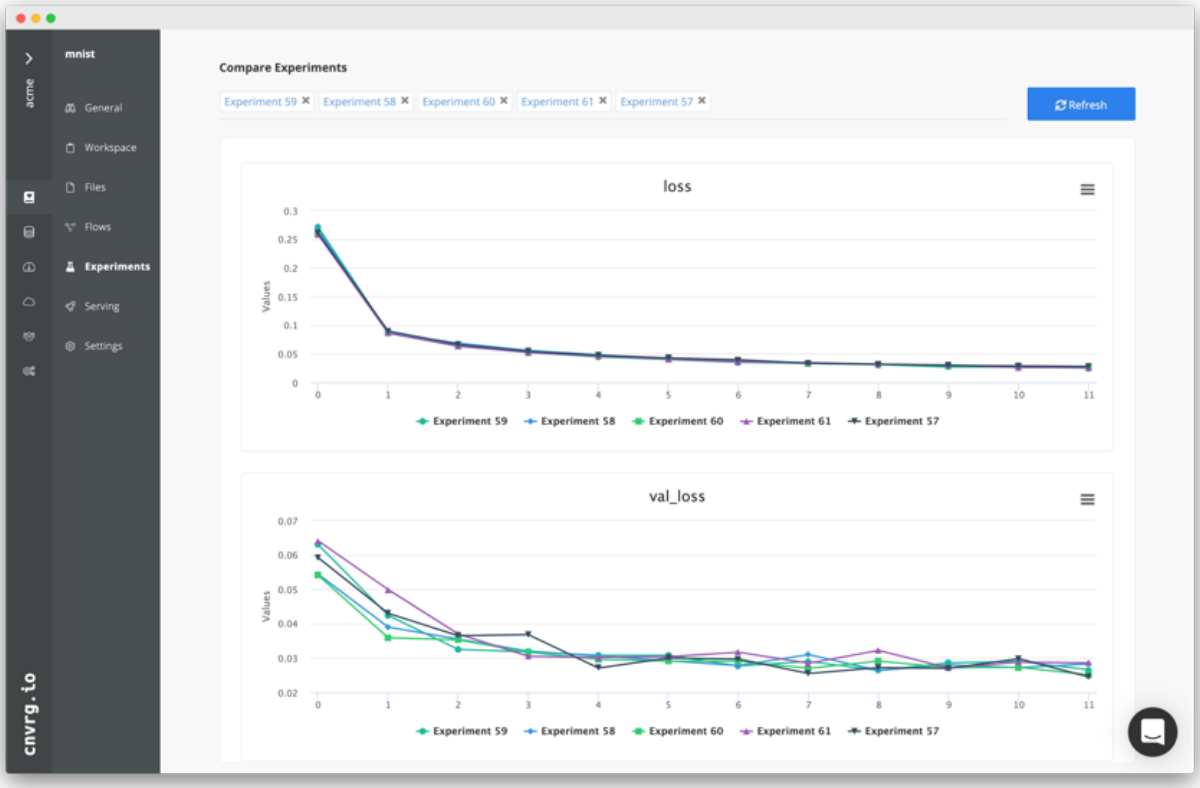
Each algorithm or component in cnvrg flows can run on a different compute instance, with a different Docker image. In our setup, we wanted to run the training algorithm on the NVIDIA DGX systems with the NetApp ONTAP AI architecture. In The following figure, we selected `gpu-real`, which is a compute template and specification for our on-premises cluster. We also created a queue of templates and selected multiple templates. In this way, if the `gpu-real` resource cannot be allocated (if, for example, other data scientists are using it), then you can enable automatic cloud-bursting by adding a cloud provider template. The following screenshot shows the use of gpu-real as a compute node for ResNet50.



Tracking and Monitoring Results

After a flow is executed, cnvrg triggers the tracking and monitoring engine. Each run of a flow is automatically documented and updated in real time. Hyperparameters, metrics, resource usage (GPU utilization, and more), code version, artifacts, logs, and so on are automatically available in the Experiments section, as shown in the following two screenshots.





Next: Conclusion

Conclusion

NetApp and cnvrg.io have partnered to offer customers a complete data management solution for ML and DL software development. ONTAP AI provides high-performance compute and storage for any scale of operation, and cnvrg.io software streamlines data science workflows and improves resource utilization.

Next: Acknowledgments

Acknowledgments

- Mike Oglesby, Technical Marketing Engineer, NetApp
- Santosh Rao, Senior Technical Director, NetApp

Next: Where to Find Additional Information

Where to Find Additional Information

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

- Cnvrg.io (https://cnvrg.io):
  - Cnvrg CORE (free ML platform)

    https://cnvrg.io/platform/core
  - Cnvrg docs

    https://app.cnvrg.io/docs
- NVIDIA DGX-1 servers:
  - NVIDIA DGX-1 servers

    https://www.nvidia.com/en-us/data-center/dgx-1/
  - NVIDIA Tesla V100 Tensor Core GPU

    https://www.nvidia.com/en-us/data-center/tesla-v100/
  - NVIDIA GPU Cloud (NGC)

    https://www.nvidia.com/en-us/gpu-cloud/
- NetApp AFF systems:
  - AFF datasheet

    https://www.netapp.com/us/media/d-3582.pdf
  - NetApp FlashAdvantage for AFF

    https://www.netapp.com/us/media/ds-3733.pdf
  - ONTAP 9.x documentation

http://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62286

- NetApp FlexGroup technical report

  https://www.netapp.com/us/media/tr-4557.pdf

- NetApp persistent storage for containers:
  - NetApp Trident

    https://netapp.io/persistent-storage-provisioner-for-kubernetes/

- NetApp Interoperability Matrix:
  - NetApp Interoperability Matrix Tool

    http://support.netapp.com/matrix

- ONTAP AI networking:
  - Cisco Nexus 3232C Switches

    https://www.cisco.com/c/en/us/products/switches/nexus-3232c-switch/index.html

  - Mellanox Spectrum 2000 series switches

    http://www.mellanox.com/page/products_dyn?product_family=251&mtag=sn2000

- ML framework and tools:
  - DALI

    https://github.com/NVIDIA/DALI

  - TensorFlow: An Open-Source Machine Learning Framework for Everyone

    https://www.tensorflow.org/

  - Horovod: Uber's Open-Source Distributed Deep Learning Framework for TensorFlow

    https://eng.uber.com/horovod/

  - Enabling GPUs in the Container Runtime Ecosystem

    https://devblogs.nvidia.com/gpu-containers-runtime/

  - Docker

    https://docs.docker.com

  - Kubernetes

    https://kubernetes.io/docs/home/

  - NVIDIA DeepOps

    https://github.com/NVIDIA/deepops

- Kubeflow

  http://www.kubeflow.org/

- Jupyter Notebook Server

  http://www.jupyter.org/

- Dataset and benchmarks:
  - NIH chest X-ray dataset

    https://nihcc.app.box.com/v/ChestXray-NIHCC

  - Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, Ronald Summers, ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases, IEEE CVPR, pp. 3462-3471, 2017TR-4841-0620

NVA-1144: NetApp HCI AI Inferencing at the Edge Data Center with H615c and NVIDIA T4

Arvind Ramakrishnan, NetApp

This document describes how NetApp HCI can be designed to host artificial intelligence (AI) inferencing workloads at edge data center locations. The design is based on NVIDIA T4 GPU-powered NetApp HCI compute nodes, an NVIDIA Triton Inference Server, and a Kubernetes infrastructure built using NVIDIA DeepOps. The design also establishes the data pipeline between the core and edge data centers and illustrates implementation to complete the data lifecycle path.

Modern applications that are driven by AI and machine learning (ML) have pushed the limits of the internet. End users and devices demand access to applications, data, and services at any place and any time, with minimal latency. To meet these demands, data centers are moving closer to their users to boost performance, reduce back-and-forth data transfer, and provide cost-effective ways to meet user requirements.

In the context of AI, the core data center is a platform that provides centralized services, such as machine learning and analytics, and the edge data centers are where the real-time production data is subject to inferencing. These edge data centers are usually connected to a core data center. They provide end-user services and serve as a staging layer for data generated by IoT devices that need additional processing and that is too time sensitive to be transmitted back to a centralized core.
This document describes a reference architecture for AI inferencing that uses NetApp HCI as the base platform.

Customer Value

NetApp HCI offers differentiation in the hyperconverged market for this inferencing solution, including the following advantages:

- A disaggregated architecture allows independent scaling of compute and storage and lowers the virtualization licensing costs and performance tax on independent NetApp HCI storage nodes.
- NetApp Element storage provides quality of service (QoS) for each storage volume, which provides guaranteed storage performance for workloads on NetApp HCI. Therefore, adjacent workloads do not negatively affect inferencing performance.
- A data fabric powered by NetApp allows data to be replicated from core to edge to cloud data centers,

which moves data closer to where application needs it.

- With a data fabric powered by NetApp and NetApp FlexCache software, AI deep learning models trained on NetApp ONTAP AI can be accessed from NetApp HCI without having to export the model.
- NetApp HCI can host inference servers on the same infrastructure concurrently with multiple workloads, either virtual-machine (VM) or container-based, without performance degradation.
- NetApp HCI is certified as NVIDIA GPU Cloud (NGC) ready for NVIDIA AI containerized applications.
- NGC-ready means that the stack is validated by NVIDIA, is purpose built for AI, and enterprise support is available through NGC Support Services.
- With its extensive AI portfolio, NetApp can support the entire spectrum of AI use cases from edge to core to cloud, including ONTAP AI for training and inferencing, Cloud Volumes Service and Azure NetApp Files for training in the cloud, and inferencing on the edge with NetApp HCI.

Next: Use Cases

Use Cases

Although all applications today are not AI driven, they are evolving capabilities that allow them to access the immense benefits of AI. To support the adoption of AI, applications need an infrastructure that provides them with the resources needed to function at an optimum level and support their continuing evolution.

For AI-driven applications, edge locations act as a major source of data. Available data can be used for training when collected from multiple edge locations over a period of time to form a training dataset. The trained model can then be deployed back to the edge locations where the data was collected, enabling faster inferencing without the need to repeatedly transfer production data to a dedicated inferencing platform.

The NetApp HCI AI inferencing solution, powered by NetApp H615c compute nodes with NVIDIA T4 GPUs and NetApp cloud-connected storage systems, was developed and verified by NetApp and NVIDIA. NetApp HCI simplifies the deployment of AI inferencing solutions at edge data centers by addressing areas of ambiguity, eliminating complexities in the design and ending guesswork.
This solution gives IT organizations a prescriptive architecture that:

- Enables AI inferencing at edge data centers
- Optimizes consumption of GPU resources
- Provides a Kubernetes-based inferencing platform for flexibility and scalability
- Eliminates design complexities

Edge data centers manage and process data at locations that are very near to the generation point. This proximity increases the efficiency and reduces the latency involved in handling data. Many vertical markets have realized the benefits of an edge data center and are heavily adopting this distributed approach to data processing.

The following table lists the edge verticals and applications.

Vertical	Applications
Medical	Computer-aided diagnostics assist medical staff in early disease detection

Vertical	Applications
Oil and gas	Autonomous inspection of remote production facilities, video, and image analytics
Aviation	Air traffic control assistance and real-time video feed analytics
Media and entertainment	Audio/video content filtering to deliver family-friendly content
Business analytics	Brand recognition to analyze brand appearance in live-streamed televised events
E-Commerce	Smart bundling of supplier offers to find ideal merchant and warehouse combinations
Retail	Automated checkout to recognize items a customer placed in cart and facilitate digital payment
Smart city	Improve traffic flow, optimize parking, and enhance pedestrian and cyclist safety
Manufacturing	Quality control, assembly-line monitoring, and defect identification
Customer service	Customer service automation to analyze and triage inquiries (phone, email, and social media)
Agriculture	Intelligent farm operation and activity planning, to optimize fertilizer and herbicide application

Target Audience

The target audience for the solution includes the following groups:

- Data scientists
- IT architects
- Field consultants
- Professional services
- IT managers
- Anyone else who needs an infrastructure that delivers IT innovation and robust data and application services at edge locations

Next: Architecture

Architecture

Solution Technology

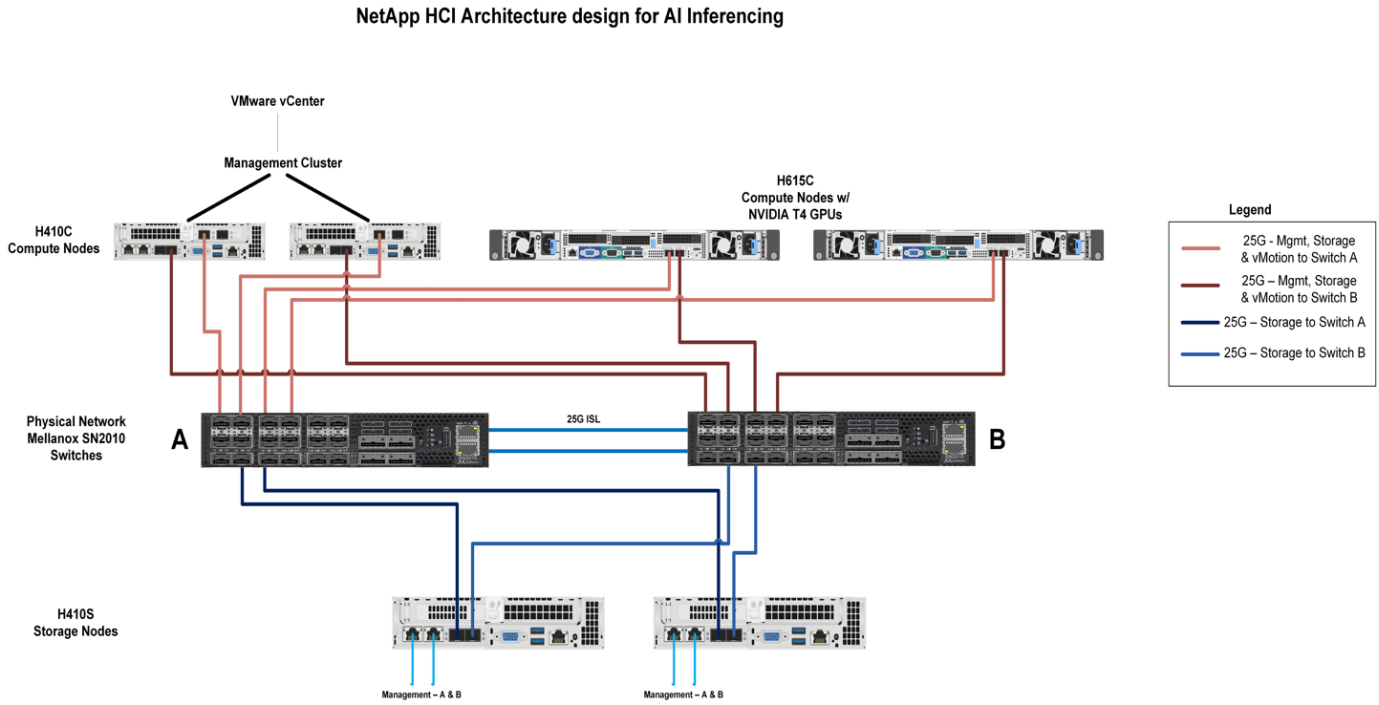
This solution is designed with a NetApp HCI system that contains the following components:

- Two H615c compute nodes with NVIDIA T4 GPUs
- Two H410c compute nodes

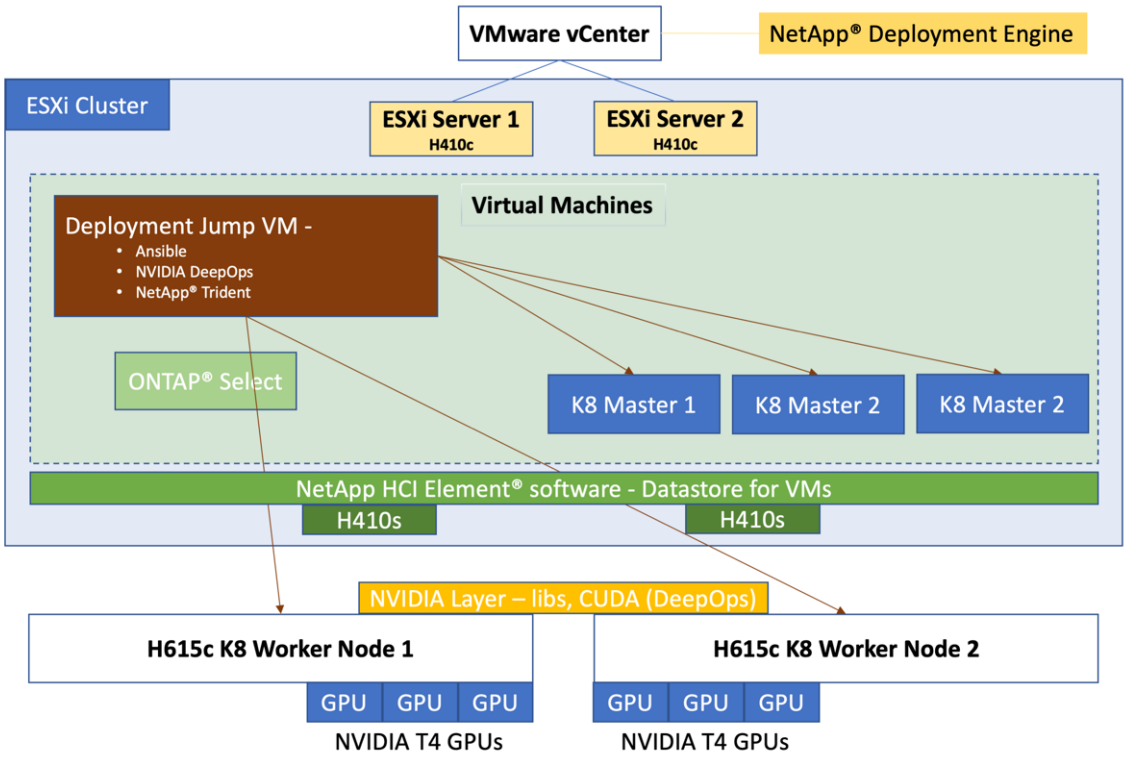
- Two H410s storage nodes
- Two Mellanox SN2010 10GbE/25GbE switches

Architectural Diagram

The following diagram illustrates the solution architecture for the NetApp HCI AI inferencing solution.



The following diagram illustrates the virtual and physical elements of this solution.



A VMware infrastructure is used to host the management services required by this inferencing solution. These services do not need to be deployed on a dedicated infrastructure; they can coexist with any existing workloads. The NetApp Deployment Engine (NDE) uses the H410c and H410s nodes to deploy the VMware infrastructure.

After NDE has completed the configuration, the following components are deployed as VMs in the virtual infrastructure:

- **Deployment Jump VM.** Used to automate the deployment of NVIDIA DeepOps. See NVIDIA DeepOps and storage management using NetApp Trident.
- **ONTAP Select.** An instance of ONTAP Select is deployed to provide NFS file services and persistent storage to the AI workload running on Kubernetes.
- **Kubernetes Masters.** During deployment, three VMs are installed and configured with a supported Linux distribution and configured as Kubernetes master nodes. After the management services have been set up, two H615c compute nodes with NVIDIA T4 GPUs are installed with a supported Linux distribution. These two nodes function as the Kubernetes worker nodes and provide the infrastructure for the inferencing platform.

Hardware Requirements

The following table lists the hardware components that are required to implement the solution. The hardware components that are used in any particular implementation of the solution might vary based on customer requirements.

Layer	Product Family	Quantity	Details
Compute	H615c	2	3 NVIDIA Tesla T4 GPUs per node
	H410c	2	Compute nodes for management infrastructure
Storage	H410s	2	Storage for OS and workload
Network	Mellanox SN2010	2	10G/25G switches

Software Requirements

The following table lists the software components that are required to implement the solution. The software components that are used in any particular implementation of the solution might vary based on customer requirements.

Layer	Software	Version
Storage	NetApp Element software	12.0.0.333
	ONTAP Select	9.7
	NetApp Trident	20.07
NetApp HCI engine	NDE	1.8
Hypervisor	Hypervisor	VMware vSphere ESXi 6.7U1
	Hypervisor Management System	VMware vCenter Server 6.7U1

Layer	Software	Version
Inferencing Platform	NVIDIA DeepOps	20.08
	NVIDIA GPU Operator	1.1.7
	Ansible	2.9.5
	Kubernetes	1.17.9
	Docker	Docker CE 18.09.7
	CUDA Version	10.2
	GPU Device Plugin	0.6.0
	Helm	3.1.2
	NVIDIA Tesla Driver	440.64.00
	NVIDIA Triton Inference Server	2.1.0 – NGC Container v20.07
K8 Master VMs	Linux	Any supported distribution across NetApp IMT, NVIDIA DeepOps, and GPUOperator Ubuntu 18.04.4 LTS was used in this solution Kernel version: 4.15
Host OS/ K8 Worker Nodes	Linux	Any supported distribution across NetApp IMT, NVIDIA DeepOps, and GPUOperator Ubuntu 18.04.4 LTS was used in this solution Kernel version: 4.15

Next: Design Considerations

Design Considerations

Network Design

The switches used to handle the NetApp HCI traffic require a specific configuration for successful deployment.

Consult the NetApp HCI Network Setup Guide for the physical cabling and switch details. This solution uses a two-cable design for compute nodes. Optionally, compute nodes can be configured in a six-node cable design affording options for deployment of compute nodes.

The diagram under Architecture depicts the network topology of this NetApp HCI solution with a two-cable design for the compute nodes.

Compute Design

The NetApp HCI compute nodes are available in two form factors, half-width and full-width, and in two rack unit sizes, 1 RU and 2 RU. The 410c nodes used in this solution are half-width and 1 RU and are housed in a chassis that can hold a maximum of four such nodes. The other compute node that is used in this solution is the H615c, which is a full-width node, 1 RU in size. The H410c nodes are based on Intel Skylake processors,

and the H615c nodes are based on the second-generation Intel Cascade Lake processors. NVIDIA GPUs can be added to the H615c nodes, and each node can host a maximum of three NVIDIA Tesla T4 16GB GPUs.

The H615c nodes are the latest series of compute nodes for NetApp HCI and the second series that can support GPUs. The first model to support GPUs is the H610c node (full width, 2RU), which can support two NVIDIA Tesla M10 GPUs.

In this solution, H615c nodes are preferred over H610c nodes because of the following advantages:

- Reduced data center footprint, critical for edge deployments
- Support for a newer generation of GPUs designed for faster inferencing
- Reduced power consumption
- Reduced heat dissipation

NVIDIA T4 GPUs

The resource requirements of inferencing are nowhere close to those of training workloads. In fact, most modern hand-held devices are capable of handling small amounts of inferencing without powerful resources like GPUs. However, for mission-critical applications and data centers that are dealing with a wide variety of applications that demand very low inferencing latencies while subject to extreme parallelization and massive input batch sizes, the GPUs play a key role in reducing inference time and help to boost application performance.

The NVIDIA Tesla T4 is an x16 PCIe Gen3 single-slot low-profile GPU based on the Turing architecture. The T4 GPUs deliver universal inference acceleration that spans applications such as image classification and tagging, video analytics, natural language processing, automatic speech recognition, and intelligent search. The breadth of the Tesla T4's inferencing capabilities enables it to be used in enterprise solutions and edge devices.

These GPUs are ideal for deployment in edge infrastructures due to their low power consumption and small PCIe form factor. The size of the T4 GPUs enables the installation of two T4 GPUs in the same space as a double-slot full-sized GPU. Although they are small, with 16GB memory, the T4s can support large ML models or run inference on multiple smaller models simultaneously.

The Turing- based T4 GPUs include an enhanced version of Tensor Cores and support a full range of precisions for inferencing FP32, FP16, INT8, and INT4. The GPU includes 2,560 CUDA cores and 320 Tensor Cores, delivering up to 130 tera operations per second (TOPS) of INT8 and up to 260 TOPS of INT4 inferencing performance. When compared to CPU-based inferencing, the Tesla T4, powered by the new Turing Tensor Cores, delivers up to 40 times higher inference performance.

The Turing Tensor Cores accelerate the matrix-matrix multiplication at the heart of neural network training and inferencing functions. They particularly excel at inference computations in which useful and relevant information can be inferred and delivered by a trained deep neural network based on a given input.

The Turing GPU architecture inherits the enhanced Multi-Process Service (MPS) feature that was introduced in the Volta architecture. Compared to Pascal-based Tesla GPUs, MPS on Tesla T4 improves inference performance for small batch sizes, reduces launch latency, improves QoS, and enables the servicing of higher numbers of concurrent client requests.

The NVIDIA T4 GPU is a part of the NVIDIA AI Inference Platform that supports all AI frameworks and provides comprehensive tooling and integrations to drastically simplify the development and deployment of advanced AI.

Storage Design: Element Software

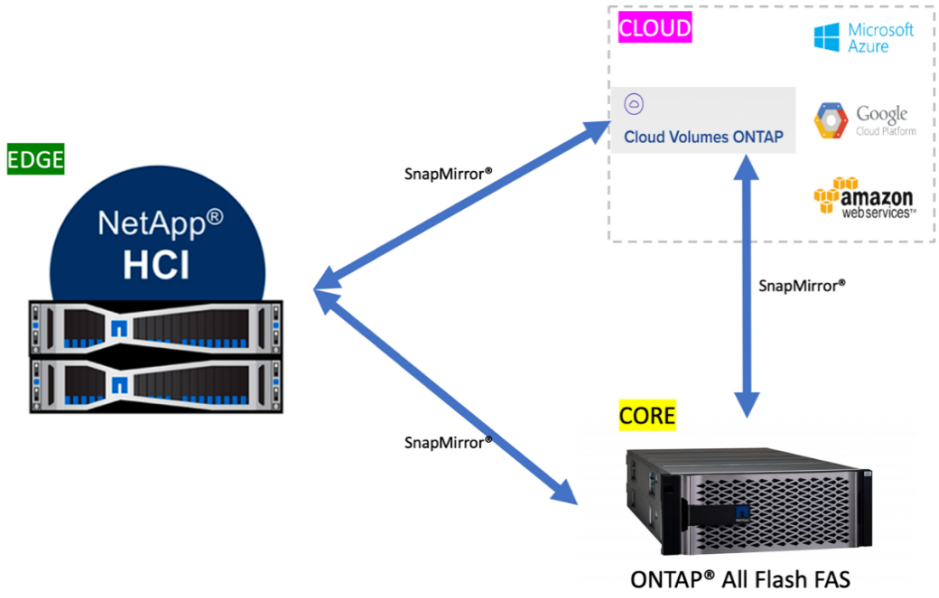
NetApp Element software powers the storage of the NetApp HCI systems. It delivers agile automation through scale-out flexibility and guaranteed application performance to accelerate new services.

Storage nodes can be added to the system non-disruptively in increments of one, and the storage resources are made available to the applications instantly. Every new node added to the system delivers a precise amount of additional performance and capacity to a usable pool. The data is automatically load balanced in the background across all nodes in the cluster, maintaining even utilization as the system grows.

Element software supports the NetApp HCI system to comfortably host multiple workloads by guaranteeing QoS to each workload. By providing fine-grained performance control with minimum, maximum, and burst settings for each workload, the software allows well-planned consolidations while protecting application performance. It decouples performance from capacity and allows each volume to be allocated with a specific amount of capacity and performance. These specifications can be modified dynamically without any interruption to data access.

As illustrated in the following figure, Element software integrates with NetApp ONTAP to enable data mobility between NetApp storage systems that are running different storage operating systems. Data can be moved from the Element software to ONTAP or vice versa by using NetApp SnapMirror technology. Element uses the same technology to provide cloud connectivity by integrating with NetApp Cloud Volumes ONTAP, which enables data mobility from the edge to the core and to multiple public cloud service providers.

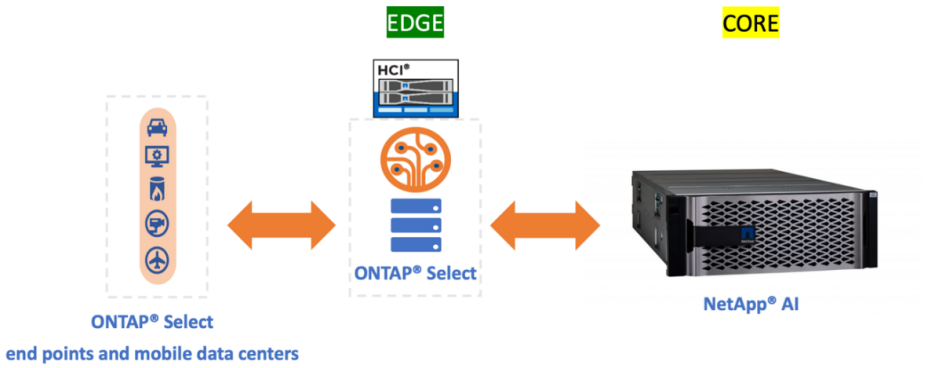
In this solution, the Element-backed storage provides the storage services that are required to run the workloads and applications on the NetApp HCI system.



Storage Design: ONTAP Select

NetApp ONTAP Select introduces a software-defined data storage service model on top of NetApp HCI. It builds on NetApp HCI capabilities, adding a rich set of file and data services to the HCI platform while extending the data fabric.

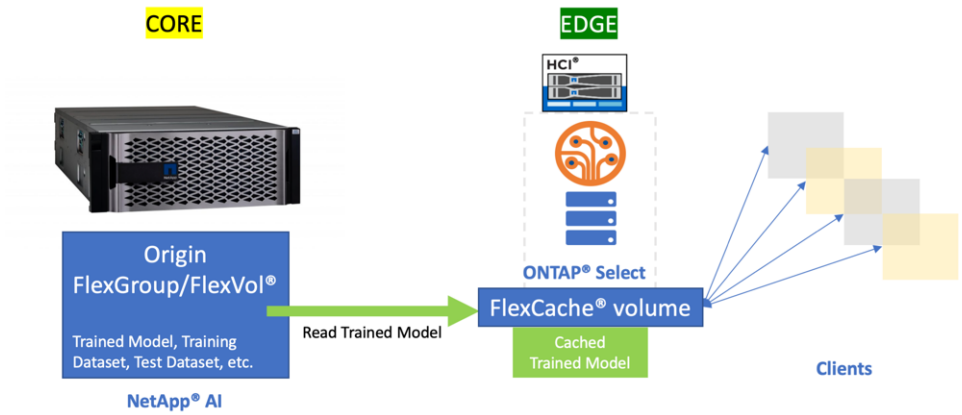
Although ONTAP Select is an optional component for implementing this solution, it does provide a host of benefits, including data gathering, protection, mobility, and so on, that are extremely useful in the context of the overall AI data lifecycle. It helps to simplify several day-to-day challenges for data handling, including ingestion, collection, training, deployment, and tiering.



ONTAP Select can run as a VM on VMware and still bring in most of the ONTAP capabilities that are available when it is running on a dedicated FAS platform, such as the following:

- Support for NFS and CIFS
- NetApp FlexClone technology
- NetApp FlexCache technology
- NetApp ONTAP FlexGroup volumes
- NetApp SnapMirror software

ONTAP Select can be used to leverage the FlexCache feature, which helps to reduce data-read latencies by caching frequently read data from a back-end origin volume, as is shown in the following figure. In the case of high-end inferencing applications with a lot of parallelization, multiple instances of the same model are deployed across the inferencing platform, leading to multiple reads of the same model. Newer versions of the trained model can be seamlessly introduced to the inferencing platform by verifying that the desired model is available in the origin or source volume.



NetApp Trident

NetApp Trident is an open-source dynamic storage orchestrator that allows you to manage storage resources across all major NetApp storage platforms. It integrates with Kubernetes natively so that persistent volumes (PVs) can be provisioned on demand with native Kubernetes interfaces and constructs. Trident enables microservices and containerized applications to use enterprise-class storage services such as QoS, storage efficiencies, and cloning to meet the persistent storage demands of applications.

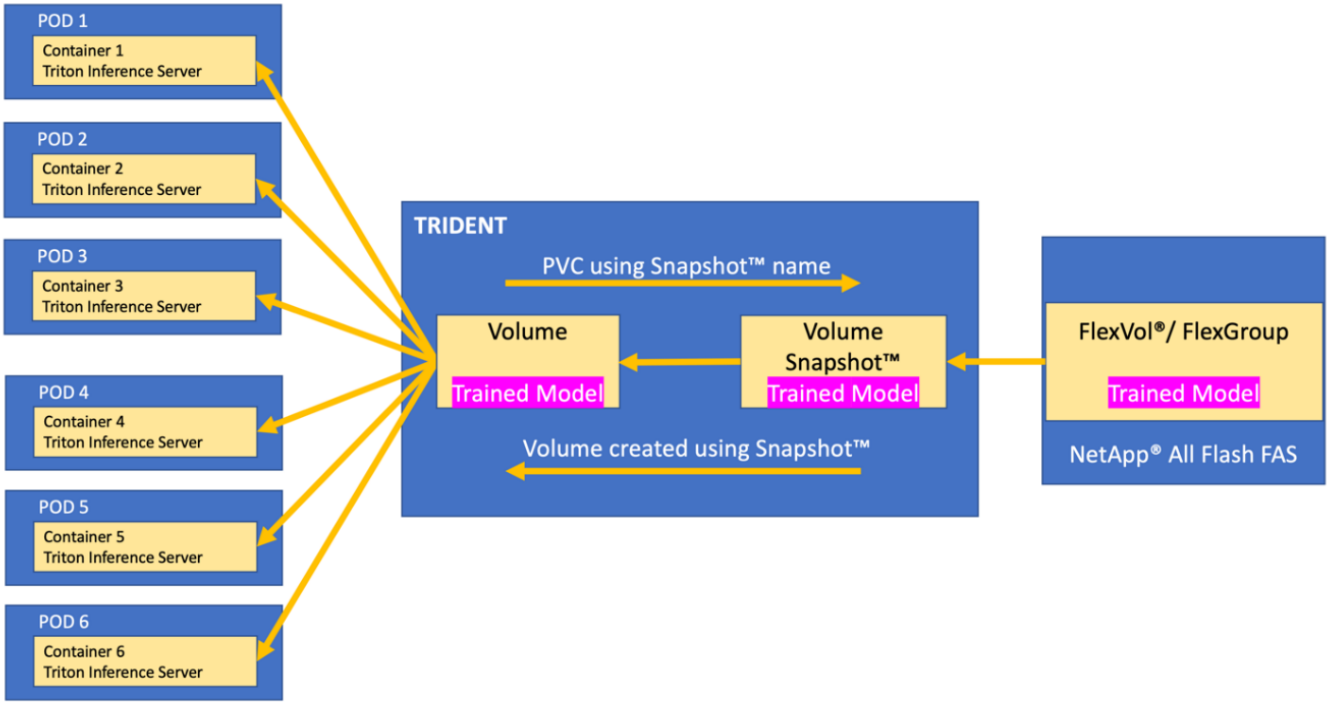
Containers are among the most popular methods of packaging and deploying applications, and Kubernetes is one of the most popular platforms for hosting containerized applications. In this solution, the inferencing platform is built on top of a Kubernetes infrastructure.

Trident currently supports storage orchestration across the following platforms:

- ONTAP: NetApp AFF, FAS, and Select
- Element software: NetApp HCI and NetApp SolidFire all-flash storage
- NetApp SANtricity software: E-Series and EF-series
- Cloud Volumes ONTAP
- Azure NetApp Files
- NetApp Cloud Volumes Service: AWS and Google Cloud

Trident is a simple but powerful tool to enable storage orchestration not just across multiple storage platforms, but also across the entire spectrum of the AI data lifecycle, ranging from the edge to the core to the cloud.

Trident can be used to provision a PV from a NetApp Snapshot copy that makes up the trained model. The following figure illustrates the Trident workflow in which a persistent volume claim (PVC) is created by referring to an existing Snapshot copy. Following this, Trident creates a volume by using the Snapshot copy.



This method of introducing trained models from a Snapshot copy supports robust model versioning. It simplifies the process of introducing newer versions of models to applications and switching inferencing between different versions of the model.

NVIDIA DeepOps

NVIDIA DeepOps is a modular collection of Ansible scripts that can be used to automate the deployment of a Kubernetes infrastructure. There are multiple deployment tools available that can automate the deployment of a Kubernetes cluster. In this solution, DeepOps is the preferred choice because it does not just deploy a Kubernetes infrastructure, it also installs the necessary GPU drivers, NVIDIA Container Runtime for Docker (nvidia-docker2), and various other dependencies for GPU-accelerated work. It encapsulates the best practices for NVIDIA GPUs and can be customized or run as individual components as needed.

DeepOps internally uses Kubespray to deploy Kubernetes, and it is included as a submodule in DeepOps. Therefore, common Kubernetes cluster management operations such as adding nodes, removing nodes, and cluster upgrades should be performed using Kubespray.

A software based L2 LoadBalancer using MetalLb and an Ingress Controller based on NGINX are also deployed as part of this solution by using the scripts that are available with DeepOps.

In this solution, three Kubernetes master nodes are deployed as VMs, and the two H615c compute nodes with NVIDIA Tesla T4 GPUs are set up as Kubernetes worker nodes.

NVIDIA GPU Operator

The GPU operator deploys the NVIDIA k8s-device-plugin for GPU support and runs the NVIDIA drivers as containers. It is based on the Kubernetes operator framework, which helps to automate the management of all NVIDIA software components that are needed to provision GPUs. The components include NVIDIA drivers, Kubernetes device plug-in for GPUs, the NVIDIA container runtime, and automatic node labeling, which is used in tandem with Kubernetes Node Feature Discovery.

The GPU operator is an important component of the NVIDIA EGX software-defined platform that is designed to make large-scale hybrid-cloud and edge operations possible and efficient. It is specifically useful when the Kubernetes cluster needs to scale quickly—for example, when provisioning additional GPU-based worker nodes and managing the lifecycle of the underlying software components. Because the GPU operator runs everything as containers, including NVIDIA drivers, administrators can easily swap various components by simply starting or stopping containers.

NVIDIA Triton Inference Server

NVIDIA Triton Inference Server (Triton Server) simplifies the deployment of AI inferencing solutions in production data centers. This microservice is specifically designed for inferencing in production data centers. It maximizes GPU utilization and integrates seamlessly into DevOps deployments with Docker and Kubernetes.

Triton Server provides a common solution for AI inferencing. Therefore, researchers can focus on creating high-quality trained models, DevOps engineers can focus on deployment, and developers can focus on applications without the need to redesign the platform for each AI-powered application.

Here are some of the key features of Triton Server:

- **Support for multiple frameworks.** Triton Server can handle a mix of models, and the number of models is limited only by system disk and memory resources. It can support the TensorRT, TensorFlow GraphDef, TensorFlow SavedModel, ONNX, PyTorch, and Caffe2 NetDef model formats.
- *Concurrent model execution. *Multiple models or multiple instances of the same model can be run simultaneously on a GPU.
- **Multi-GPU support.** Triton Server can maximize GPU utilization by enabling inference for multiple models on one or more GPUs.
- **Support for batching.** Triton Server can accept requests for a batch of inputs and respond with the corresponding batch of outputs. The inference server supports multiple scheduling and batching algorithms that combine individual inference requests together to improve inference throughput. Batching algorithms are available for both stateless and stateful applications and need to be used appropriately. These scheduling and batching decisions are transparent to the client that is requesting inference.
- **Ensemble support.** An ensemble is a pipeline with multiple models with connections of input and output tensors between those models. An inference request can be made to an ensemble, which results in the execution of the complete pipeline.
- **Metrics.** Metrics are details about GPU utilization, server throughput, server latency, and health for auto scaling and load balancing.

NetApp HCI is a hybrid multi-cloud infrastructure that can host multiple workloads and applications, and the Triton Inference Server is well equipped to support the inferencing requirements of multiple applications.

In this solution, Triton Server is deployed on the Kubernetes cluster using a deployment file. With this method, the default configuration of Triton Server can be overridden and customized as required. Triton Server also provides an inference service using an HTTP or GRPC endpoint, allowing remote clients to request inferencing for any model that is being managed by the server.

A Persistent Volume is presented via NetApp Trident to the container that runs the Triton Inference Server and this persistent volume is configured as the model repository for the Inference server.

The Triton Inference Server is deployed with varying sets of resources using Kubernetes deployment files, and each server instance is presented with a LoadBalancer front end for seamless scalability. This approach also illustrates the flexibility and simplicity with which resources can be allocated to the inferencing workloads.

Next: Deploying NetApp HCI – AI Inferencing at the Edge

Overview

This section describes the steps required to deploy the AI inferencing platform using NetApp HCI. The following list provides the high-level tasks involved in the setup:

1. Configure network switches
2. Deploy the VMware virtual infrastructure on NetApp HCI using NDE
3. Configure the H615c compute nodes to be used as K8 worker nodes
4. Set up the deployment jump VM and K8 master VMs
5. Deploy a Kubernetes cluster with NVIDIA DeepOps
6. Deploy ONTAP Select within the virtual infrastructure
7. Deploy NetApp Trident
8. Deploy NVIDIA Triton inference Server
9. Deploy the client for the Triton inference server
10. Collect inference metrics from the Triton inference server

Configure Network Switches (Automated Deployment)

Prepare Required VLAN IDs

The following table lists the necessary VLANs for deployment, as outlined in this solution validation. You should configure these VLANs on the network switches prior to executing NDE.

Network Segment	Details	VLAN ID
Out-of-band management network	Network for HCI terminal user interface (TUI)	16
In-band management network	Network for accessing management interfaces of nodes, hosts, and guests	3488
VMware vMotion	Network for live migration of VMs	3489
iSCSI SAN storage	Network for iSCSI storage traffic	3490
Application	Network for Application traffic	3487
NFS	Network for NFS storage traffic	3491
IPL*	Interpeer link between Mellanox switches	4000
Native	Native VLAN	2

*Only for Mellanox switches

Switch Configuration

This solution uses Mellanox SN2010 switches running Onyx. The Mellanox switches are configured using an Ansible playbook. Prior to running the Ansible playbook, you should perform the initial configuration of the switches manually:

1. Install and cable the switches to the uplink switch, compute, and storage nodes.
2. Power on the switches and configure them with the following details:
   a. Host name
   b. Management IP and gateway
   c. NTP
3. Log into the Mellanox switches and run the following commands:

```
configuration write to pre-ansible
configuration write to post-ansible
```

   The `pre-ansible` configuration file created can be used to restore the switch's configuration to the state before the Ansible playbook execution.

   The switch configuration for this solution is stored in the `post-ansible` configuration file.

4. The configuration playbook for Mellanox switches that follows best practices and requirements for NetApp HCI can be downloaded from the NetApp HCI Toolkit.

   The HCI Toolkit also provides a playbook to setup Cisco Nexus switches with similar best practices and requirements for NetApp HCI.

   Additional guidance on populating the variables and executing the playbook is available in the respective switch README.md file.

5. Fill out the credentials to access the switches and variables needed for the environment. The following text is a sample of the variable file for this solution.

```
# vars file for nar_hci_mellanox_deploy
#These set of variables will setup the Mellanox switches for NetApp HCI
that uses a 2-cable compute connectivity option.
#Ansible connection variables for mellanox
ansible_connection: network_cli
ansible_network_os: onyx
#--------------------
# Primary Variables
#--------------------
#Necessary VLANs for Standard NetApp HCI Deployment [native, Management,
iSCSI_Storage, vMotion, VM_Network, IPL]
#Any additional VLANs can be added to this in the prescribed format
below
netapp_hci_vlans:
- {vlan_id: 2 , vlan_name: "Native" }
- {vlan_id: 3488 , vlan_name: "IB-Management" }
- {vlan_id: 3490 , vlan_name: "iSCSI_Storage" }
- {vlan_id: 3489 , vlan_name: "vMotion" }
```

```
- {vlan_id: 3491 , vlan_name: "NFS " }
- {vlan_id: 3487 , vlan_name: "App_Network" }
- {vlan_id: 4000 , vlan_name: "IPL" }#Modify the VLAN IDs to suit your
environment
#Spanning-tree protocol type for uplink connections.
#The valid options are 'network' and 'normal'; selection depends on the
uplink switch model.
uplink_stp_type: network
#----------------------
# IPL variables
#----------------------
#Inter-Peer Link Portchannel
#ipl_portchannel to be defined in the format - Po100
ipl_portchannel: Po100
#Inter-Peer Link Addresses
#The IPL IP address should not be part of the management network. This
is typically a private network
ipl_ipaddr_a: 10.0.0.1
ipl_ipaddr_b: 10.0.0.2
#Define the subnet mask in CIDR number format. Eg: For subnet /22, use
ipl_ip_subnet: 22
ipl_ip_subnet: 24
#Inter-Peer Link Interfaces
#members to be defined with Eth in the format. Eg: Eth1/1
peer_link_interfaces:
  members: ['Eth1/20', 'Eth1/22']
  description: "peer link interfaces"
#MLAG VIP IP address should be in the same subnet as that of the
switches' mgmt0 interface subnet
#mlag_vip_ip to be defined in the format - <vip_ip>/<subnet_mask>. Eg:
x.x.x.x/y
mlag_vip_ip: <<mlag_vip_ip>>
#MLAG VIP Domain Name
#The mlag domain must be unique name for each mlag domain.
#In case you have more than one pair of MLAG switches on the same
network, each domain (consist of two switches) should be configured with
different name.
mlag_domain_name: MLAG-VIP-DOM
#---------------------
# Interface Details
#---------------------
#Storage Bond10G Interface details
#members to be defined with Eth in the format. Eg: Eth1/1
#Only numerical digits between 100 to 1000 allowed for mlag_id
#Operational link speed [variable 'speed' below] to be defined in terms
of bytes.
```

```
#For 10 Gigabyte operational speed, define 10G. [Possible values - 10G
and 25G]
#Interface descriptions append storage node data port numbers assuming
all Storage Nodes' Port C -> Mellanox Switch A and all Storage Nodes'
Port D -> Mellanox Switch B
#List the storage Bond10G interfaces, their description, speed and MLAG
IDs in list of dictionaries format
storage_interfaces:
- {members: "Eth1/1", description: "HCI_Storage_Node_01", mlag_id: 101,
speed: 25G}
- {members: "Eth1/2", description: "HCI_Storage_Node_02", mlag_id: 102,
speed: 25G}
#In case of additional storage nodes, add them here
#Storage Bond1G Interface
#Mention whether or not these Mellanox switches will also be used for
Storage Node Mgmt connections
#Possible inputs for storage_mgmt are 'yes' and 'no'
storage_mgmt: <<yes or no>>
#Storage Bond1G (Mgmt) interface details. Only if 'storage_mgmt' is set
to 'yes'
#Members to be defined with Eth in the format. Eg: Eth1/1
#Interface descriptions append storage node management port numbers
assuming all Storage Nodes' Port A -> Mellanox Switch A and all Storage
Nodes' Port B -> Mellanox Switch B
#List the storage Bond1G interfaces and their description in list of
dictionaries format
storage_mgmt_interfaces:
- {members: "Ethx/y", description: "HCI_Storage_Node_01"}
- {members: "Ethx/y", description: "HCI_Storage_Node_02"}
#In case of additional storage nodes, add them here
#LACP load balancing algorithm for IP hash method
#Possible options are: 'destination-mac', 'destination-ip',
'destination-port', 'source-mac', 'source-ip', 'source-port', 'source-
destination-mac', 'source-destination-ip', 'source-destination-port'
#This variable takes multiple options in a single go
#For eg: if you want to configure load to be distributed in the port-
channel based on the traffic source and destination IP address and port
number, use 'source-destination-ip source-destination-port'
#By default, Mellanox sets it to source-destination-mac. Enter the
values below only if you intend to configure any other load balancing
algorithm
#Make sure the load balancing algorithm that is set here is also
replicated on the host side
#Recommended algorithm is source-destination-ip source-destination-port
#Fill the lacp_load_balance variable only if you are using configuring
interfaces on compute nodes in bond or LAG with LACP
```

```
lacp_load_balance: "source-destination-ip source-destination-port"
#Compute Interface details
#Members to be defined with Eth in the format. Eg: Eth1/1
#Fill the mlag_id field only if you intend to configure interfaces of
compute nodes into bond or LAG with LACP
#In case you do not intend to configure LACP on interfaces of compute
nodes, either leave the mlag_id field unfilled or comment it or enter NA
in the mlag_id field
#In case you have a mixed architecture where some compute nodes require
LACP and some don't,
#1. Fill the mlag_id field with appropriate MLAG ID for interfaces that
connect to compute nodes requiring LACP
#2. Either fill NA or leave the mlag_id field blank or comment it for
interfaces connecting to compute nodes that do not require LACP
#Only numerical digits between 100 to 1000 allowed for mlag_id.
#Operational link speed [variable 'speed' below] to be defined in terms
of bytes.
#For 10 Gigabyte operational speed, define 10G. [Possible values - 10G
and 25G]
#Interface descriptions append compute node port numbers assuming all
Compute Nodes' Port D -> Mellanox Switch A and all Compute Nodes' Port E
-> Mellanox Switch B
#List the compute interfaces, their speed, MLAG IDs and their
description in list of dictionaries format
compute_interfaces:
- members: "Eth1/7"#Compute Node for ESXi, setup by NDE
  description: "HCI_Compute_Node_01"
  mlag_id: #Fill the mlag_id only if you wish to use LACP on interfaces
towards compute nodes
  speed: 25G
- members: "Eth1/8"#Compute Node for ESXi, setup by NDE
  description: "HCI_Compute_Node_02"
  mlag_id: #Fill the mlag_id only if you wish to use LACP on interfaces
towards compute nodes
  speed: 25G
#In case of additional compute nodes, add them here in the same format
as above- members: "Eth1/9"#Compute Node for Kubernetes Worker node
  description: "HCI_Compute_Node_01"
  mlag_id: 109 #Fill the mlag_id only if you wish to use LACP on
interfaces towards compute nodes
  speed: 10G
- members: "Eth1/10"#Compute Node for Kubernetes Worker node
  description: "HCI_Compute_Node_02"
  mlag_id: 110 #Fill the mlag_id only if you wish to use LACP on
interfaces towards compute nodes
  speed: 10G
```

```
#Uplink Switch LACP support
#Possible options are 'yes' and 'no' - Set to 'yes' only if your uplink
switch supports LACP
uplink_switch_lacp: <<yes or no>>
#Uplink Interface details
#Members to be defined with Eth in the format. Eg: Eth1/1
#Only numerical digits between 100 to 1000 allowed for mlag_id.
#Operational link speed [variable 'speed' below] to be defined in terms
of bytes.
#For 10 Gigabyte operational speed, define 10G. [Possible values in
Mellanox are 1G, 10G and 25G]
#List the uplink interfaces, their description, MLAG IDs and their speed
in list of dictionaries format
uplink_interfaces:
- members: "Eth1/18"
  description_switch_a: "SwitchA:Ethx/y -> Uplink_Switch:Ethx/y"
  description_switch_b: "SwitchB:Ethx/y -> Uplink_Switch:Ethx/y"
  mlag_id: 118  #Fill the mlag_id only if 'uplink_switch_lacp' is set to
'yes'
  speed: 10G
  mtu: 1500
```

The fingerprint for the switch's key must match with that present in the host machine from where the playbook is being executed. To ensure this, add the key to `/root/.ssh/known_host` or any other appropriate location.

Rollback the Switch Configuration

1. In case of any timeout failures or partial configuration, run the following command to roll back the switch to the initial state.

```
configuration switch-to pre-ansible
```

This operation requires a reboot of the switch.

2. Switch the configuration to the state before running the Ansible playbook.

```
configuration delete post-ansible
```

3. Delete the post-ansible file that had the configuration from the Ansible playbook.

```
configuration write to post-ansible
```

4. Create a new file with the same name post-ansible, write the pre-ansible configuration to it, and switch to the new configuration to restart configuration.

IP Address Requirements

The deployment of the NetApp HCI inferencing platform with VMware and Kubernetes requires multiple IP addresses to be allocated. The following table lists the number of IP addresses required. Unless otherwise indicated, addresses are assigned automatically by NDE.

IP Address Quantity	Details	VLAN ID	IP Address
One per storage and compute node*	HCI terminal user interface (TUI) addresses	16	
One per vCenter Server (VM)	vCenter Server management address	3488	
One per management node (VM)	Management node IP address		
One per ESXi host	ESXi compute management addresses		
One per storage/witness node	NetApp HCI storage node management addresses		
One per storage cluster	Storage cluster management address		
One per ESXi host	VMware vMotion address	3489	
Two per ESXi host	ESXi host initiator address for iSCSI storage traffic	3490	
Two per storage node	Storage node target address for iSCSI storage traffic		
Two per storage cluster	Storage cluster target address for iSCSI storage traffic		
Two for mNode	mNode iSCSI storage access		

The following IPs are assigned manually when the respective components are configured.

IP Address Quantity	Details	VLAN ID	IP Address
One for Deployment Jump Management network	Deployment Jump VM to execute Ansible playbooks and configure other parts of the system – management connectivity	3488	
One per Kubernetes master node – management network	Kubernetes master node VMs (three nodes)	3488	

IP Address Quantity	Details	VLAN ID	IP Address
One per Kubernetes worker node – management network	Kubernetes worker nodes (two nodes)	3488	
One per Kubernetes worker node – NFS network	Kubernetes worker nodes (two nodes)	3491	
One per Kubernetes worker node – application network	Kubernetes worker nodes (two nodes)	3487	
Three for ONTAP Select – management network	ONTAP Select VM	3488	
One for ONTAP Select – NFS network	ONTAP Select VM – NFS data traffic	3491	
At least two for Triton Inference Server Load Balancer – application network	Load balancer IP range for Kubernetes load balancer service	3487	

*This validation requires the initial setup of the first storage node TUI address. NDE automatically assigns the TUI address for subsequent nodes.

DNS and Timekeeping Requirement

Depending on your deployment, you might need to prepare DNS records for your NetApp HCI system. NetApp HCI requires a valid NTP server for timekeeping; you can use a publicly available time server if you do not have one in your environment.

This validation involves deploying NetApp HCI with a new VMware vCenter Server instance using a fully qualified domain name (FQDN). Before deployment, you must have one Pointer (PTR) record and one Address (A) record created on the DNS server.

Next: Virtual Infrastructure with Automated Deployment

Deploy VMware Virtual Infrastructure on NetApp HCI with NDE (Automated Deployment)

NDE Deployment Prerequisites

Consult the NetApp HCI Prerequisites Checklist to see the requirements and recommendations for NetApp HCI before you begin deployment.

1. Network and switch requirements and configuration
2. Prepare required VLAN IDs
3. Switch configuration
4. IP Address Requirements for NetApp HCI and VMware
5. DNS and time-keeping requirements
6. Final preparations

NDE Execution

Before you execute the NDE, you must complete the rack and stack of all components, configuration of the network switches, and verification of all prerequisites. You can execute NDE by connecting to the management address of a single storage node if you plan to allow NDE to automatically configure all addresses.

NDE performs the following tasks to bring an HCI system online:

1. Installs the storage node (NetApp Element software) on a minimum of two storage nodes.
2. Installs the VMware hypervisor on a minimum of two compute nodes.
3. Installs VMware vCenter to manage the entire NetApp HCI stack.
4. Installs and configures the NetApp storage management node (mNode) and NetApp Monitoring Agent.

This validation uses NDE to automatically configure all addresses. You can also set up DHCP in your environment or manually assign IP addresses for each storage node and compute node. These steps are not covered in this guide.

   As mentioned previously, this validation uses a two-cable configuration for compute nodes.

   Detailed steps for the NDE are not covered in this document.

   For step-by-step guidance on completing the deployment of the base NetApp HCI platform, see the Deployment guide.

5. After NDE has finished, login to the vCenter and create a Distributed Port Group `NetApp HCI VDS 01-NFS_Network` for the NFS network to be used by ONTAP Select and the application.

Next: Configure NetApp H615c (Manual Deployment)

Configure NetApp H615c (Manual Deployment)

In this solution, the NetApp H615c compute nodes are configured as Kubernetes worker nodes. The Inferencing workload is hosted on these nodes.

Deploying the compute nodes involves the following tasks:

- Install Ubuntu 18.04.4 LTS.
- Configure networking for data and management access.
- Prepare the Ubuntu instances for Kubernetes deployment.

Install Ubuntu 18.04.4 LTS

The following high-level steps are required to install the operating system on the H615c compute nodes:

1. Download Ubuntu 18.04.4 LTS from Ubuntu releases.
2. Using a browser, connect to the IPMI of the H615c node and launch Remote Control.
3. Map the Ubuntu ISO using the Virtual Media Wizard and start the installation.
4. Select one of the two physical interfaces as the `Primary network interface` when prompted.

   An IP from a DHCP source is allocated when available, or you can switch to a manual IP configuration later. The network configuration is modified to a bond-based setup after the OS has been installed.

5. Provide a hostname followed by a domain name.
6. Create a user and provide a password.
7. Partition the disks according to your requirements.
8. Under Software Selection, select `OpenSSH server` and click Continue.
9. Reboot the node.

Configure Networking for Data and Management Access

The two physical network interfaces of the Kubernetes worker nodes are set up as a bond and VLAN interfaces for management and application, and NFS data traffic is created on top of it.

The inferencing applications and associated containers use the application network for connectivity.

1. Connect to the console of the Ubuntu instance as a user with root privileges and launch a terminal session.
2. Navigate to `/etc/netplan` and open the `01-netcfg.yaml` file.
3. Update the netplan file based on the network details for the management, application, and NFS traffic in your environment.

   The following template of the netplan file was used in this solution:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp59s0f0: #Physical Interface 1
      match:
        macaddress: <<mac_address Physical Interface 1>>
      set-name: enp59s0f0
      mtu: 9000
    enp59s0f1: # Physical Interface 2
      match:
        macaddress: <<mac_address Physical Interface 2>>
      set-name: enp59s0f1
      mtu: 9000
  bonds:
      bond0:
        mtu: 9000
        dhcp4: false
        dhcp6: false
        interfaces: [ enp59s0f0, enp59s0f1 ]
        parameters:
          mode: 802.3ad
          mii-monitor-interval: 100
```

```
  vlans:
    vlan.3488: #Management VLAN
      id: 3488
      xref:{relative_path}bond0
      dhcp4: false
      addresses: [ipv4_address/subnet]
      routes:
      - to: 0.0.0.0/0
        via: 172.21.232.111
        metric: 100
        table: 3488
      - to: x.x.x.x/x # Additional routes if any
        via: y.y.y.y
        metric: <<metric>>
        table: <<table #>>
      routing-policy:
      - from: 0.0.0.0/0
        priority: 32768#Higher Priority than table 3487
        table: 3488
      nameservers:
        addresses: [nameserver_ip]
        search: [ search_domain ]
      mtu: 1500
    vlan.3487:
      id: 3487
      xref:{relative_path}bond0
      dhcp4: false
      addresses: [ipv4_address/subnet]
      routes:
      - to: 0.0.0.0/0
        via: 172.21.231.111
        metric: 101
        table: 3487
      - to: x.x.x.x/x
        via: y.y.y.y
        metric: <<metric>>
        table: <<table #>>
      routing-policy:
      - from: 0.0.0.0/0
        priority: 32769#Lower Priority
        table: 3487
      nameservers:
        addresses: [nameserver_ip]
        search: [ search_domain ]
      mtu: 1500    vlan.3491:
      id: 3491
```

```
      xref:{relative_path}bond0
      dhcp4: false
      addresses: [ipv4_address/subnet]
      mtu: 9000
```

4. Confirm that the priorities for the routing policies are lower than the priorities for the main and default tables.
5. Apply the netplan.

```
sudo netplan --debug apply
```

6. Make sure that there are no errors.
7. If Network Manager is running, stop and disable it.

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

8. Add a host record for the server in DNS.
9. Open a VI editor to `/etc/iproute2/rt_tables` and add the two entries.

```
#
# reserved values
#
255     local
254     main
253     default
0       unspec
#
# local
#
#1      inr.ruhep
101     3488
102     3487
```

10. Match the table number to what you used in the netplan.
11. Open a VI editor to `/etc/sysctl.conf` and set the value of the following parameters.

```
net.ipv4.conf.default.rp_filter=0
net.ipv4.conf.all.rp_filter=0net.ipv4.ip_forward=1
```

12. Update the system.

```
sudo apt-get update && sudo apt-get upgrade
```

13. Reboot the system
14. Repeat steps 1 through 13 for the other Ubuntu instance.

Next: Set Up the Deployment Jump and the Kubernetes Master Node VMs (Manual Deployment)

Set Up the Deployment Jump VM and the Kubernetes Master Node VMs (Manual Deployment)

A Deployment Jump VM running a Linux distribution is used for the following purposes:

- Deploying ONTAP Select using an Ansible playbook
- Deploying the Kubernetes infrastructure with NVIDIA DeepOps and GPU Operator
- Installing and configuring NetApp Trident

Three more VMs running Linux are set up; these VMs are configured as Kubernetes Master Nodes in this solution.

Ubuntu 18.04.4 LTS was used in this solution deployment.

1. Deploy the Ubuntu 18.04.4 LTS VM with VMware tools

   You can refer to the high-level steps described in section Install Ubuntu 18.04.4 LTS.

2. Configure the in-band management network for the VM. See the following sample netplan template:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    ens160:
      dhcp4: false
      addresses: [ipv4_address/subnet]
      routes:
      - to: 0.0.0.0/0
        via: 172.21.232.111
        metric: 100
        table: 3488
      routing-policy:
      - from: 0.0.0.0/0
        priority: 32768
        table: 3488
      nameservers:
        addresses: [nameserver_ip]
        search: [ search_domain ]
      mtu: 1500
```

This template is not the only way to setup the network. You can use any other approach that you prefer.

3. Apply the netplan.

```
sudo netplan --debug apply
```

4. Stop and disable Network Manager if it is running.

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

5. Open a VI editor to `/etc/iproute2/rt_tables` and add a table entry.

```
#
# reserved values
#
255     local
254     main
253     default
0       unspec
#
# local
#
#1      inr.ruhep
101     3488
```

6. Add a host record for the VM in DNS.
7. Verify outbound internet access.
8. Update the system.

```
sudo apt-get update && sudo apt-get upgrade
```

9. Reboot the system.
10. Repeat steps 1 through 9 to set up the other three VMs.

Next: Deploy a Kubernetes Cluster with NVIDIA DeepOps (Automated Deployment)

Deploy a Kubernetes Cluster with NVIDIA DeepOps Automated Deployment

To deploy and configure the Kubernetes Cluster with NVIDIA DeepOps, complete the following steps:

1. Make sure that the same user account is present on all the Kubernetes master and worker nodes.
2. Clone the DeepOps repository.

```
git clone https://github.com/NVIDIA/deepops.git
```

3. Check out a recent release tag.

```
cd deepops
git checkout tags/20.08
```

   If this step is skipped, the latest development code is used, not an official release.

4. Prepare the Deployment Jump by installing the necessary prerequisites.

```
./scripts/setup.sh
```

5. Create and edit the Ansible inventory by opening a VI editor to `deepops/config/inventory`.
   a. List all the master and worker nodes under [all].
   b. List all the master nodes under [kube-master]
   c. List all the master nodes under [etcd]
   d. List all the worker nodes under [kube-node]

```
######
# ALL NODES
# NOTE: Use existing hostnames here, DeepOps will conf
######
[all]
hci-ai-k8-master-01     ansible_host=172.21.232.114
hci-ai-k8-master-02     ansible_host=172.21.232.115
hci-ai-k8-master-03     ansible_host=172.21.232.116
hci-ai-k8-worker-01     ansible_host=172.21.232.109
hci-ai-k8-worker-02     ansible_host=172.21.232.110

######
# KUBERNETES
######
[kube-master]
hci-ai-k8-master-01
hci-ai-k8-master-02
hci-ai-k8-master-03

# Odd number of nodes required
[etcd]
hci-ai-k8-master-01
hci-ai-k8-master-02
hci-ai-k8-master-03

# Also add mgmt/master nodes here if they will run non
[kube-node]
hci-ai-k8-worker-01
hci-ai-k8-worker-02

[k8s-cluster:children]
kube-master
kube-node
```

6. Enable GPUOperator by opening a VI editor to `deepops/config/group_vars/k8s-cluster.yml`.

```
# Provide option to use GPU Operator instead of setting up NVIDIA driver and
# Docker configuration.
deepops_gpu_operator_enabled: true
```

7. Set the value of `deepops_gpu_operator_enabled` to true.
8. Verify the permissions and network configuration.

```
ansible all -m raw -a "hostname" -k -K
```

   - If SSH to the remote hosts requires a password, use -k.
   - If sudo on the remote hosts requires a password, use -K.

9. If the previous step passed without any issues, proceed with the setup of Kubernetes.

```
ansible-playbook --limit k8s-cluster playbooks/k8s-cluster.yml -k -K
```

10. To verify the status of the Kubernetes nodes and the pods, run the following commands:

```
kubectl get nodes
```

```
rarvind@deployment-jump:~/deepops$ kubectl get nodes
NAME                 STATUS   ROLES    AGE     VERSION
hci-ai-k8-master-01   Ready    master   2d19h   v1.17.6
hci-ai-k8-master-02   Ready    master   2d19h   v1.17.6
hci-ai-k8-master-03   Ready    master   2d19h   v1.17.6
hci-ai-k8-worker-01   Ready    <none>   2d19h   v1.17.6
hci-ai-k8-worker-02   Ready    <none>   2d19h   v1.17.6
```

```
kubectl get pods -A
```

It can take a few minutes for all the pods to run.

```
rarvind@deployment-jump:~/deepops$ kubectl get pods -A
NAMESPACE                NAME                                                              READY   STATUS
default                  gpu-operator-74c97448d9-ppdlc                                     1/1     Running
default                  nvidia-gpu-operator-node-feature-discovery-master-ffccb57dx9wtl   1/1     Running
default                  nvidia-gpu-operator-node-feature-discovery-worker-2lr9t           1/1     Running
default                  nvidia-gpu-operator-node-feature-discovery-worker-616x7           1/1     Running
default                  nvidia-gpu-operator-node-feature-discovery-worker-jf696           1/1     Running
default                  nvidia-gpu-operator-node-feature-discovery-worker-tmtwv           1/1     Running
default                  nvidia-gpu-operator-node-feature-discovery-worker-z4nlh           1/1     Running
gpu-operator-resources   nvidia-container-toolkit-daemonset-7jbl4                          1/1     Running
gpu-operator-resources   nvidia-container-toolkit-daemonset-x5ktb                          1/1     Running
gpu-operator-resources   nvidia-dcgm-exporter-5x94p                                        1/1     Running
gpu-operator-resources   nvidia-dcgm-exporter-7cbrl                                        1/1     Running
gpu-operator-resources   nvidia-device-plugin-daemonset-n8vrk                              1/1     Running
gpu-operator-resources   nvidia-device-plugin-daemonset-z7j6s                              1/1     Running
gpu-operator-resources   nvidia-device-plugin-validation                                   0/1     Completed
gpu-operator-resources   nvidia-driver-daemonset-7h752                                     1/1     Running
gpu-operator-resources   nvidia-driver-daemonset-v4rbj                                     1/1     Running
gpu-operator-resources   nvidia-driver-validation                                          0/1     Completed
kube-system              calico-kube-controllers-777478f4ff-jknxg                          1/1     Running
kube-system              calico-node-2j9mr                                                 1/1     Running
kube-system              calico-node-czk76                                                 1/1     Running
kube-system              calico-node-jpdxn                                                 1/1     Running
kube-system              calico-node-nwnvn                                                 1/1     Running
kube-system              calico-node-ssjrx                                                 1/1     Running
kube-system              coredns-76798d84dd-5pvgf                                          1/1     Running
kube-system              coredns-76798d84dd-w7l2j                                          1/1     Running
kube-system              dns-autoscaler-85f898cd5c-qqrbp                                   1/1     Running
kube-system              kube-apiserver-hci-ai-k8-master-01                                1/1     Running
kube-system              kube-apiserver-hci-ai-k8-master-02                                1/1     Running
kube-system              kube-apiserver-hci-ai-k8-master-03                                1/1     Running
kube-system              kube-controller-manager-hci-ai-k8-master-01                       1/1     Running
kube-system              kube-controller-manager-hci-ai-k8-master-02                       1/1     Running
kube-system              kube-controller-manager-hci-ai-k8-master-03                       1/1     Running
kube-system              kube-proxy-5znxk                                                  1/1     Running
kube-system              kube-proxy-fk6h6                                                  1/1     Running
kube-system              kube-proxy-hphfb                                                  1/1     Running
kube-system              kube-proxy-qzxhr                                                  1/1     Running
kube-system              kube-proxy-rkjds                                                  1/1     Running
kube-system              kube-scheduler-hci-ai-k8-master-01                                1/1     Running
kube-system              kube-scheduler-hci-ai-k8-master-02                                1/1     Running
kube-system              kube-scheduler-hci-ai-k8-master-03                                1/1     Running
kube-system              kubernetes-dashboard-5fcff756f-dmswt                              1/1     Running
kube-system              kubernetes-metrics-scraper-747b4fd5cd-4q4p2                       1/1     Running
kube-system              nginx-proxy-hci-ai-k8-worker-01                                   1/1     Running
kube-system              nginx-proxy-hci-ai-k8-worker-02                                   1/1     Running
kube-system              nodelocaldns-2dmjr                                                1/1     Running
kube-system              nodelocaldns-b7xrw                                                1/1     Running
kube-system              nodelocaldns-jrhs2                                                1/1     Running
kube-system              nodelocaldns-jztzs                                                1/1     Running
kube-system              nodelocaldns-wgx84                                                1/1     Running
```

11. Verify that the Kubernetes setup can access and use the GPUs.

```
./scripts/k8s_verify_gpu.sh
```

Expected sample output:

```
rarvind@deployment-jump:~/deepops$ ./scripts/k8s_verify_gpu.sh
job_name=cluster-gpu-tests
Node found with 3 GPUs
Node found with 3 GPUs
total_gpus=6
Creating/Deleting sandbox Namespace
updating test yml
downloading containers ...
```

```
job.batch/cluster-gpu-tests condition met
executing ...
Mon Aug 17 16:02:45 2020
+-----------------------------------------------------------------------------
------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version:
10.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|===============================+======================+================
======|
|   0  Tesla T4            On   | 00000000:18:00.0 Off |
0 |
| N/A   38C    P8    10W /  70W |      0MiB / 15109MiB |      0%
Default |
+-------------------------------+----------------------
+----------------------+
+-----------------------------------------------------------------------------
------+
| Processes:                                                       GPU
Memory |
|  GPU       PID   Type   Process name                             Usage
|
|=============================================================================
======|
|  No running processes found
|
+-----------------------------------------------------------------------------
------+
Mon Aug 17 16:02:45 2020
+-----------------------------------------------------------------------------
------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version:
10.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|===============================+======================+================
======|
```

```
|   0  Tesla T4            On   | 00000000:18:00.0 Off |
0 |
| N/A   38C    P8    10W /  70W |      0MiB / 15109MiB |      0%
Default |
+-------------------------------+----------------------
+----------------------+
+-----------------------------------------------------------------------------
------+
| Processes:                                                                  GPU
Memory |
|  GPU       PID   Type   Process name                                  Usage
|
|=============================================================================
======|
|  No running processes found
|
+-----------------------------------------------------------------------------
------+
Mon Aug 17 16:02:45 2020
+-----------------------------------------------------------------------------
------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version:
10.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|===============================+======================+================
======|
|   0  Tesla T4            On   | 00000000:18:00.0 Off |
0 |
| N/A   38C    P8    10W /  70W |      0MiB / 15109MiB |      0%
Default |
+-------------------------------+----------------------
+----------------------+
+-----------------------------------------------------------------------------
------+
| Processes:                                                                  GPU
Memory |
|  GPU       PID   Type   Process name                                  Usage
|
|=============================================================================
======|
|  No running processes found
```

```
|
+-----------------------------------------------------------------------------
------+
Mon Aug 17 16:02:45 2020
+-----------------------------------------------------------------------------
------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version:
10.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|===============================+======================+===============
======|
|   0  Tesla T4            On   | 00000000:18:00.0 Off |
0 |
| N/A   38C    P8    10W /  70W |      0MiB / 15109MiB |      0%
Default |
+-------------------------------+----------------------
+----------------------+
+-----------------------------------------------------------------------------
------+
| Processes:                                                       GPU
Memory |
|  GPU       PID   Type   Process name                             Usage
|
|=============================================================================
======|
|  No running processes found
|
+-----------------------------------------------------------------------------
------+
Mon Aug 17 16:02:45 2020
+-----------------------------------------------------------------------------
------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version:
10.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|===============================+======================+===============
```

```
======|
|   0  Tesla T4            On   | 00000000:18:00.0 Off |
0 |
| N/A   38C    P8    10W /  70W |      0MiB / 15109MiB |      0%
Default |
+-------------------------------+----------------------
+----------------------+
+-----------------------------------------------------------------------
------+
| Processes:                                                       GPU
Memory |
|  GPU       PID   Type   Process name                             Usage
|
|=======================================================================
======|
|  No running processes found
|
+-----------------------------------------------------------------------
------+
Mon Aug 17 16:02:45 2020
+-----------------------------------------------------------------------
------+
| NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version:
10.2     |
|-------------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util
Compute M. |
|===============================+======================+================
======|
|   0  Tesla T4            On   | 00000000:18:00.0 Off |
0 |
| N/A   38C    P8    10W /  70W |      0MiB / 15109MiB |      0%
Default |
+-------------------------------+----------------------
+----------------------+
+-----------------------------------------------------------------------
------+
| Processes:                                                       GPU
Memory |
|  GPU       PID   Type   Process name                             Usage
|
|=======================================================================
======|
```

```
|  No running processes found
|
+-----------------------------------------------------------------------------
------+
Number of Nodes: 2
Number of GPUs: 6
6 / 6 GPU Jobs COMPLETED
job.batch "cluster-gpu-tests" deleted
namespace "cluster-gpu-verify" deleted
```

12. Install Helm on the Deployment Jump.

```
./scripts/install_helm.sh
```

13. Remove the taints on the master nodes.

```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

This step is required to run the LoadBalancer pods.

14. Deploy LoadBalancer.
15. Edit the `config/helm/metallb.yml` file and provide a range of IP ddresses in the `Application Network` to be used as LoadBalancer.

```
---
# Default address range matches private network for the virtual cluster
# defined in virtual/.
# You should set this address range based on your site's infrastructure.
configInline:
  address-pools:
  - name: default
    protocol: layer2
    addresses:
    - 172.21.231.130-172.21.231.140#Application Network
controller:
  nodeSelector:
    node-role.kubernetes.io/master: ""
```

16. Run a script to deploy LoadBalancer.

```
./scripts/k8s_deploy_loadbalancer.sh
```

17. Deploy an Ingress Controller.

```
./scripts/k8s_deploy_ingress.sh
```

Next: Deploy and Configure ONTAP Select in the VMware Virtual Infrastructure (Automated Deployment)

Deploy and Configure ONTAP Select in the VMware Virtual Infrastructure (Automated Deployment)

To deploy and configure an ONTAP Select instance within the VMware Virtual Infrastructure, complete the following steps:

1. From the Deployment Jump VM, login to the NetApp Support Site and download the ONTAP Select OVA for ESXi.
2. Create a directory OTS and obtain the Ansible roles for deploying ONTAP Select.

```
mkdir OTS
cd OTS
git clone https://github.com/NetApp/ansible.git
cd ansible
```

3. Install the prerequisite libraries.

```
pip install requests
pip install pyvmomi
Open a VI Editor and create a playbook ``ots_setup.yaml`' with the below
content to deploy the ONTAP Select OVA and initialize the ONTAP cluster.
---
- name: Create ONTAP Select Deploy VM from OVA (ESXi)
  hosts: localhost
  gather_facts: false
  connection: 'local'
  vars_files:
    - ots_deploy_vars.yaml
  roles:
    - na_ots_deploy
- name: Wait for 1 minute before starting cluster setup
  hosts: localhost
  gather_facts: false
  tasks:
  - pause:
      minutes: 1
- name: Create ONTAP Select cluster (ESXi)
  hosts: localhost
  gather_facts: false
  vars_files:
  - ots_cluster_vars.yaml
  roles:
    - na_ots_cluster
```

4. Open a VI editor, create a variable file `ots_deploy_vars.yaml`, and fill in hte following parameters:

```
target_vcenter_or_esxi_host: "10.xxx.xx.xx"# vCenter IP
host_login: "yourlogin@yourlab.local"  # vCenter Username
ovf_path: "/run/deploy/ovapath/ONTAPdeploy.ova"# Path to OVA on
Deployment Jump VM
datacenter_name: "your-Lab"# Datacenter name in vCenter
esx_cluster_name: "your Cluster"# Cluster name in vCenter
datastore_name: "your-select-dt"# Datastore name in vCenter
mgt_network: "your-mgmt-network"# Management Network to be used by OVA
deploy_name: "test-deploy-vm"# Name of the ONTAP Select VM
deploy_ipAddress: "10.xxx.xx.xx"# Management IP Address of ONTAP Select
VM
deploy_gateway: "10.xxx.xx.1"# Default Gateway
deploy_proxy_url: ""# Proxy URL (Optional and if used)
deploy_netMask: "255.255.255.0"# Netmask
deploy_product_company: "NetApp"# Name of Organization
deploy_primaryDNS: "10.xxx.xx.xx"# Primary DNS IP
deploy_secondaryDNS: ""# Secondary DNS (Optional)
deploy_searchDomains: "your.search.domain.com"# Search Domain Name
```

Update the variables to match your environment.

5. Open a VI editor, create a variable file `ots_cluster_vars.yaml`, and fill it out with the following parameters:

```
node_count: 1#Number of nodes in the ONTAP Cluster
monitor_job: truemonitor_deploy_job: true
deploy_api_url: #Use the IP of the ONTAP Select VM
deploy_login: "admin"
vcenter_login: "administrator@vsphere.local"
vcenter_name: "172.21.232.100"
esxi_hosts:
  - host_name: 172.21.232.102
  - host_name: 172.21.232.103
cluster_name: "hci-ai-ots"# Name of ONTAP Cluster
cluster_ip: "172.21.232.118"# Cluster Management IP
cluster_netmask: "255.255.255.0"
cluster_gateway: "172.21.232.1"
cluster_ontap_image: "9.7"
cluster_ntp:
  - "10.61.186.231"
cluster_dns_ips:
  - "10.61.186.231"
cluster_dns_domains:
  - "sddc.netapp.com"
mgt_network: "NetApp HCI VDS 01-Management_Network"# Name of VM Port
Group for Mgmt Network
data_network: "NetApp HCI VDS 01-NFS_Network"# Name of VM Port Group for
NFS Network
internal_network: ""# Not needed for Single Node Cluster
instance_type: "small"
cluster_nodes:
  - node_name: "{{ cluster_name }}-01"
    ipAddress: 172.21.232.119# Node Management IP
    storage_pool: NetApp-HCI-Datastore-02 # Name of Datastore in vCenter
to use
    capacityTB: 1# Usable capacity will be ~700GB
    host_name: 172.21.232.102# IP Address of an ESXi host to deploy node
```

Update the variables to match your environment.

6. Start ONTAP Select setup.

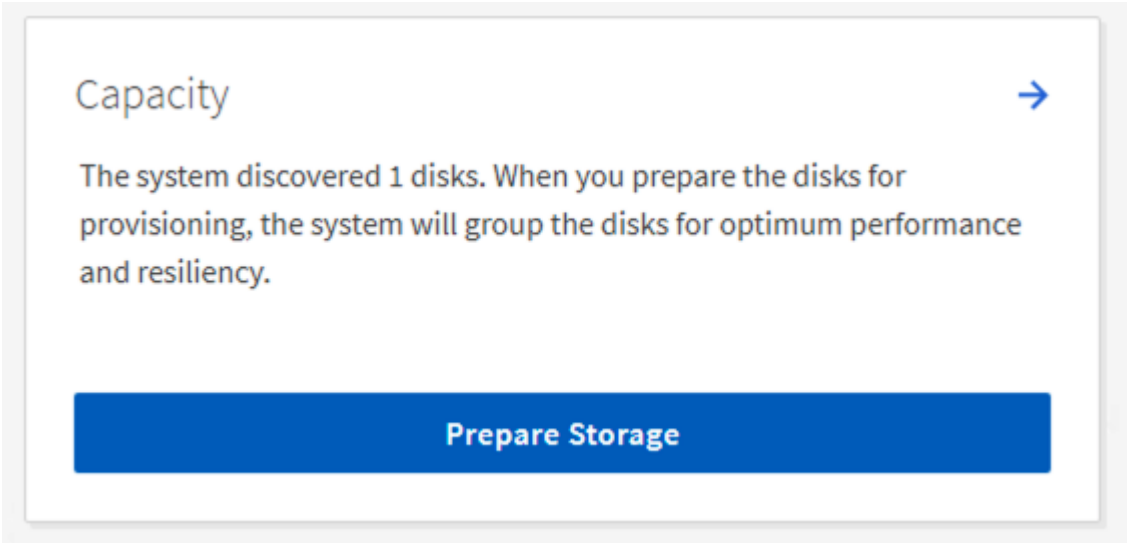
```
ansible-playbook ots_setup.yaml --extra-vars deploy_pwd=$'"P@ssw0rd"'
--extra-vars vcenter_password=$'"P@ssw0rd"' --extra-vars
ontap_pwd=$'"P@ssw0rd"' --extra-vars host_esx_password=$'"P@ssw0rd"'
--extra-vars host_password=$'"P@ssw0rd"' --extra-vars
deploy_password=$'"P@ssw0rd"'
```

7. Update the command with `deploy_pwd` `(ONTAP Select VM instance), `vcenter_password`(vCenter), `ontap_pwd` (ONTAP login password), `host_esx_password` (VMware ESXi), `host_password` (vCenter), and `deploy_password` (ONTAP Select VM instance).

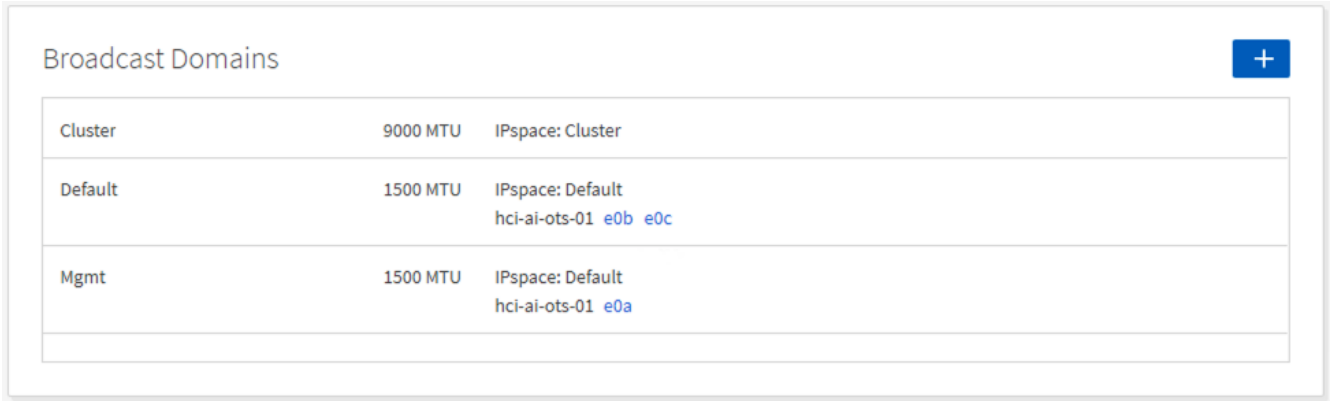
Configure the ONTAP Select Cluster – Manual Deployment

To configure the ONTAP Select cluster, complete the following steps:

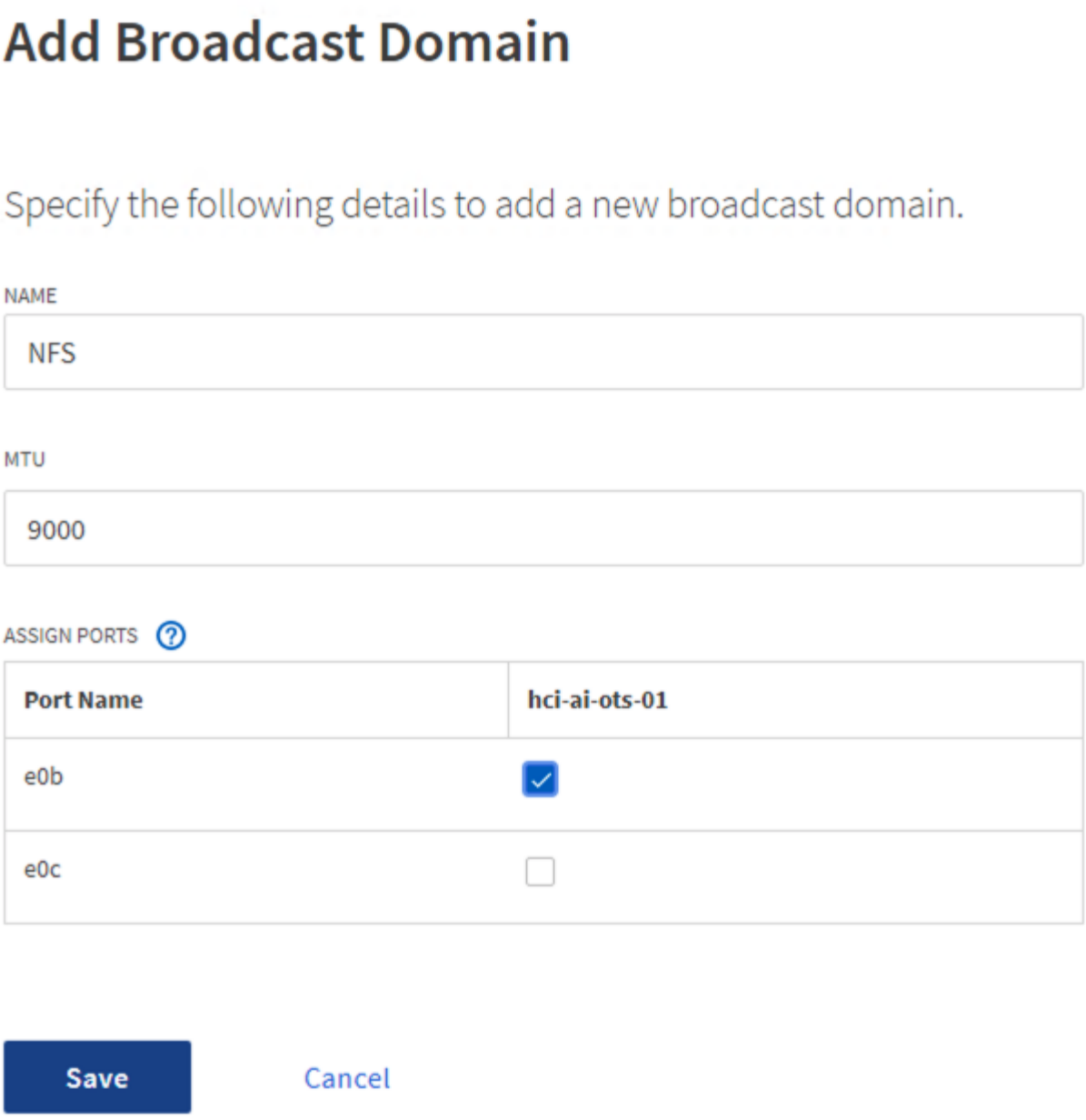
1. Open a browser and log into the ONTAP cluster's System Manager using its cluster management IP.
2. On the DASHBOARD page, click Prepare Storage under Capacity.



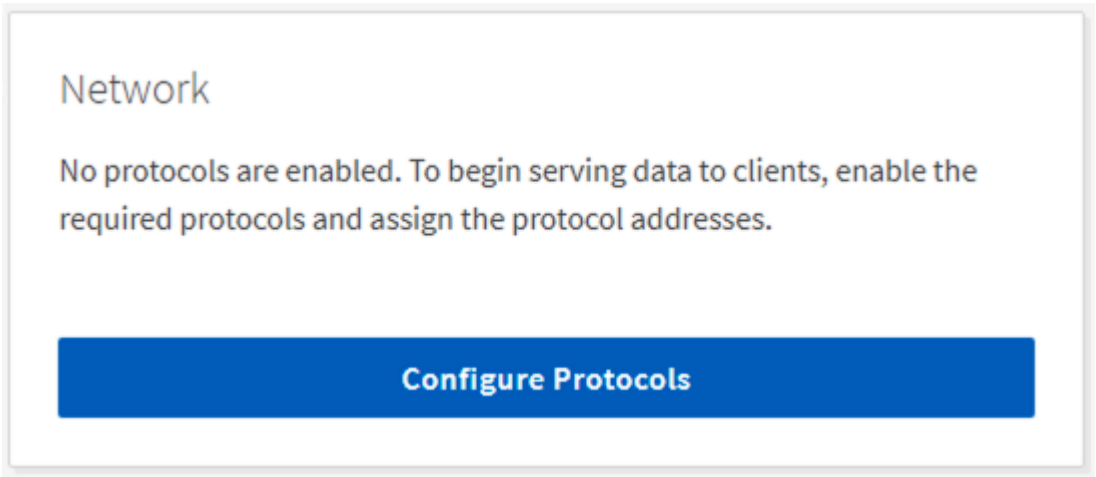
3. Select the radio button to continue without onboard key manager, and click Prepare Storage.
4. On the NETWORK page, click the + sign in the Broadcast Domains window.



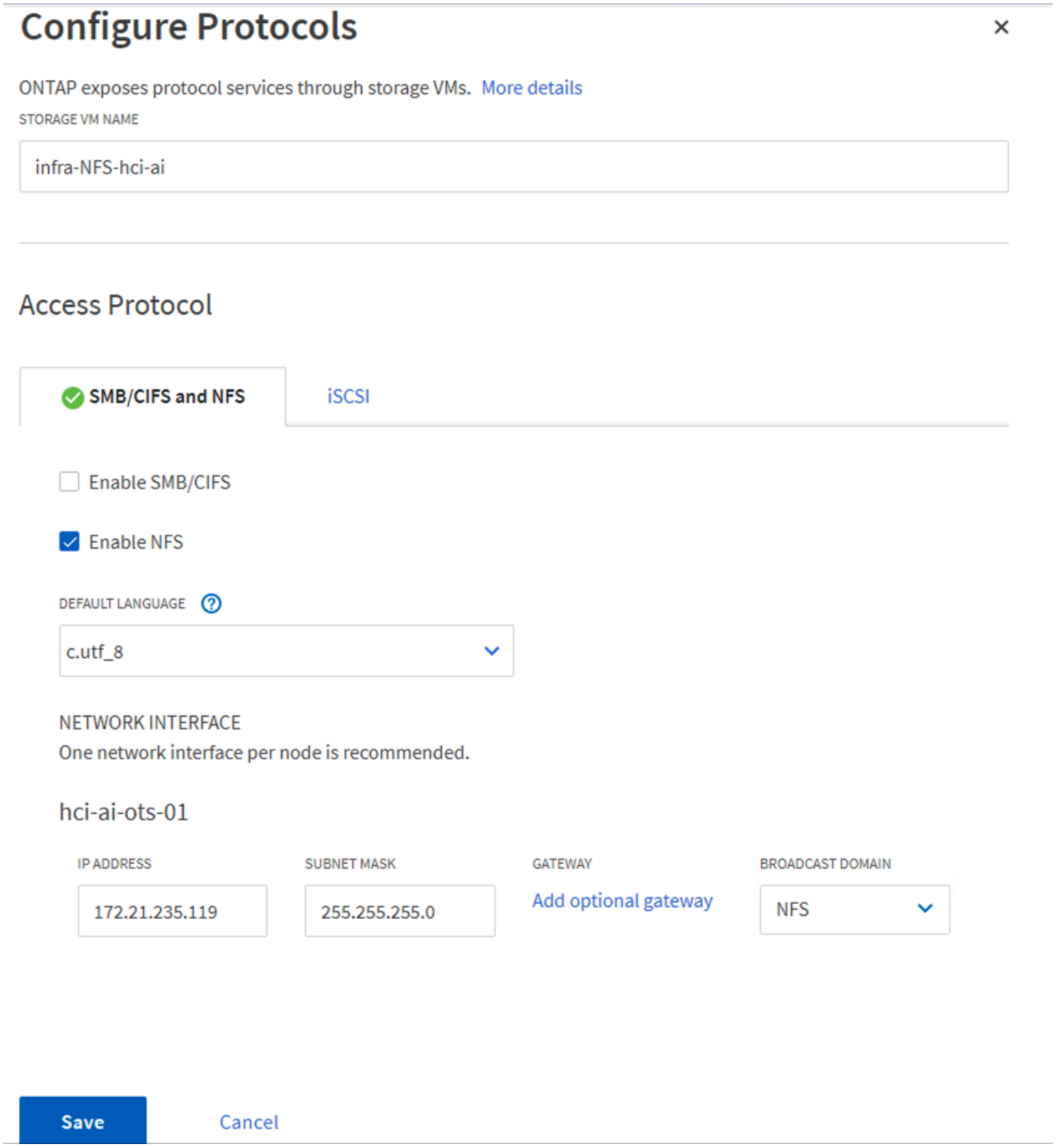
5. Enter the Name as `NFS`, set the MTU to `9000`, and select the port `e0b`. Click Save.



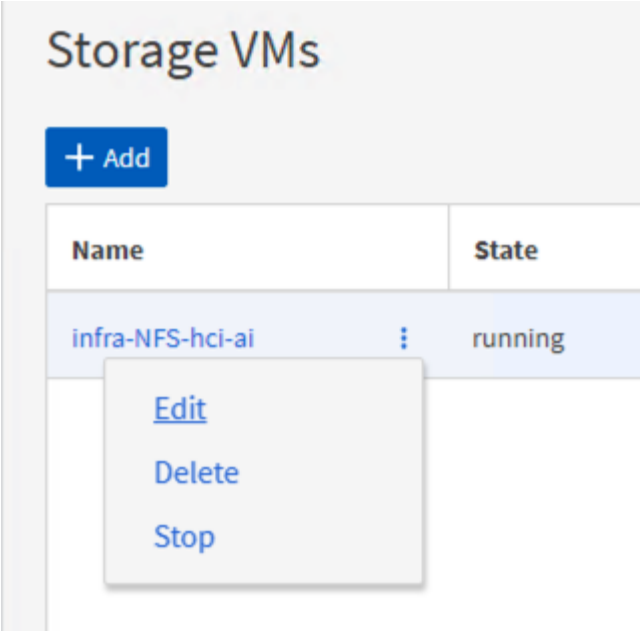
6. On the DASHBOARD page, click `Configure Protocols` under Network.



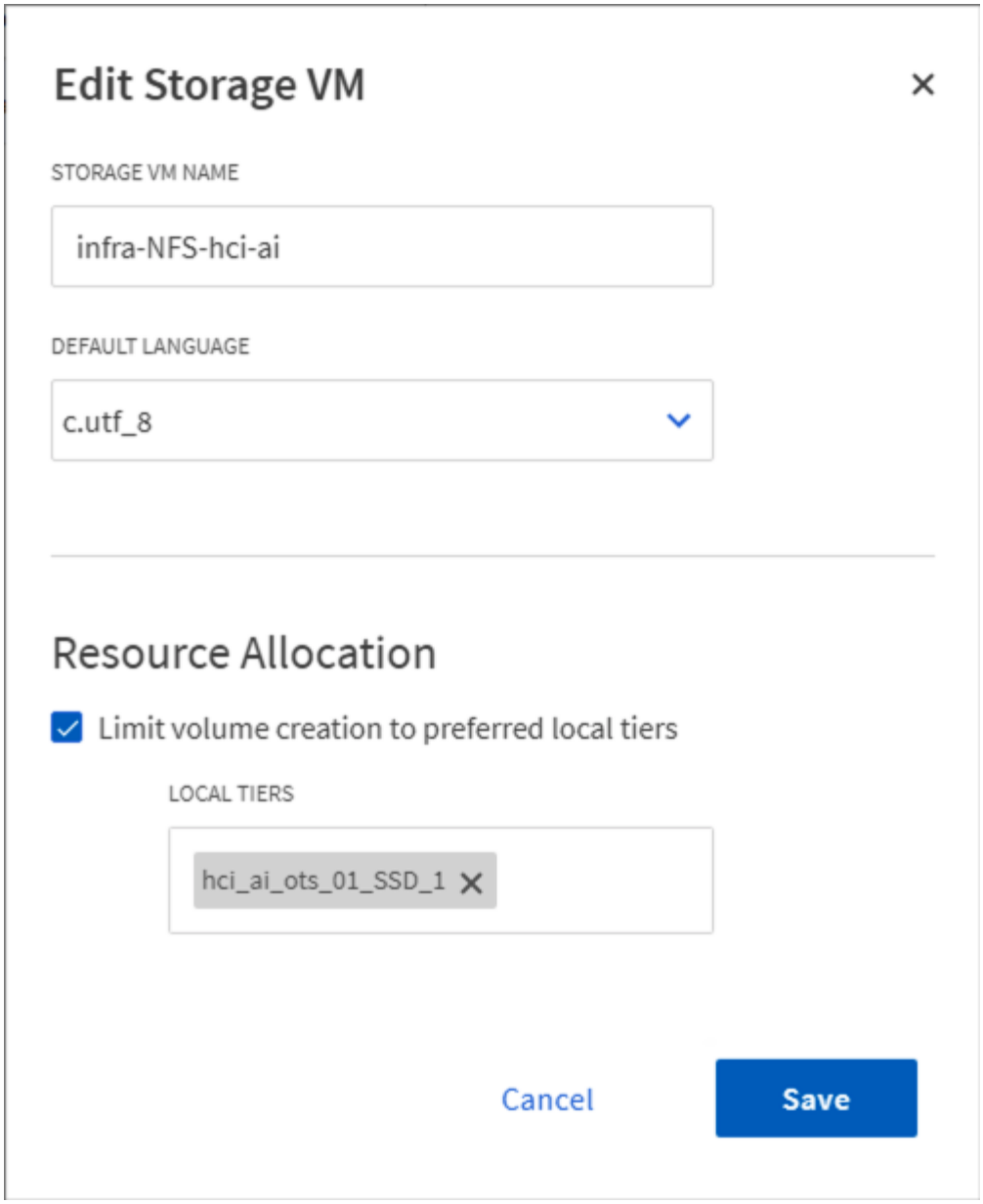
7. Enter a name for the SVM, select Enable NFS, provide an IP and subnet mask for the NFS LIF, set the Broadcast Domain to NFS, and click Save.



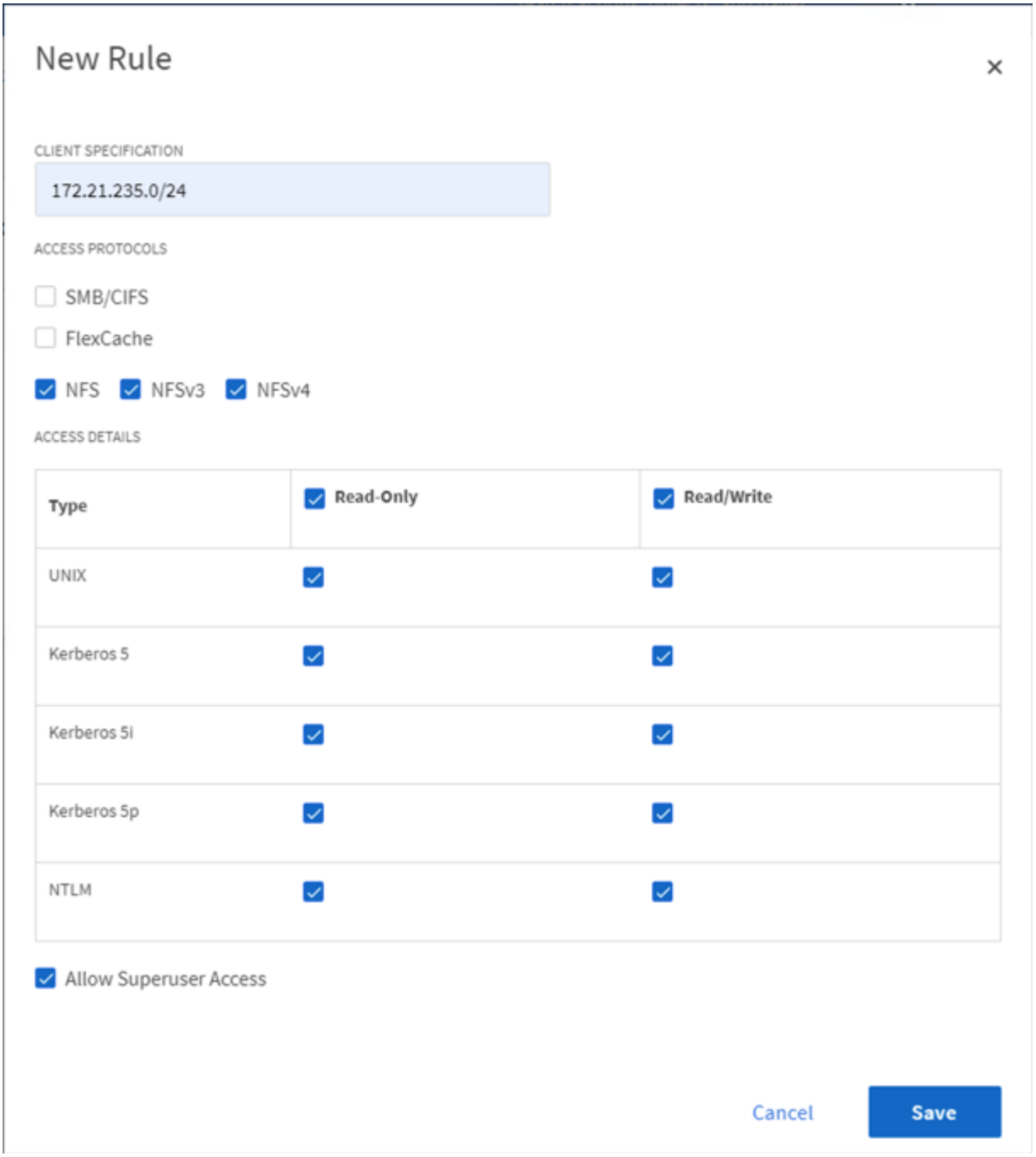
8. Click STORAGE in the left pane, and from the dropdown select Storage VMs
   a. Edit the SVM.



b. Select the checkbox under Resource Allocation, make sure that the local tier is listed, and click Save.



9. Click the SVM name, and on the right panel scroll down to Policies.
10. Click the arrow within the Export Policies tile, and click the default policy.
11. If there is a rule already defined, you can edit it; if no rule exists, then create a new one.
    a. Select NFS Network Clients as the Client Specification.
    b. Select the Read-Only and Read/Write checkboxes.
    c. Select the checkbox to Allow Superuser Access.



Next: Deploy NetApp Trident (Automated Deployment)

Deploy NetApp Trident (Automated Deployment)

NetApp Trident is deployed by using an Ansible playbook that is available with NVIDIA DeepOps. Follow these steps to set up NetApp Trident:

1. From the Deployment Jump VM, navigate to the DeepOps directory and open a VI editor to `config/group_vars/netapp-trident.yml`. The file from DeepOps lists two backends and two storage classes. In this solution only one backend and storage class are used.

   Use the following template to update the file and its parameters (highlighted in yellow) to match your environment.

```
---
# vars file for netapp-trident playbook
# URL of the Trident installer package that you wish to download and use
trident_version: "20.07.0"# Version of Trident desired
trident_installer_url:
"https://github.com/NetApp/trident/releases/download/v{{ trident_version
}}/trident-installer-{{ trident_version }}.tar.gz"
# Kubernetes version
# Note: Do not include patch version, e.g. provide value of 1.16, not
1.16.7.
# Note: Versions 1.14 and above are supported when deploying Trident
with DeepOps.
#   If you are using an earlier version, you must deploy Trident
manually.
k8s_version: 1.17.9# Version of Kubernetes running
# Denotes whether or not to create new backends after deploying trident
# For more info, refer to: https://netapp-
trident.readthedocs.io/en/stable-v20.04/kubernetes/operator-
install.html#creating-a-trident-backend
create_backends: true
# List of backends to create
# For more info on parameter values, refer to: https://netapp-
trident.readthedocs.io/en/stable-
v20.04/kubernetes/operations/tasks/backends/ontap.html
# Note: Parameters other than those listed below are not avaible when
creating a backend via DeepOps
#   If you wish to use other parameter values, you must create your
backend manually.
backends_to_create:
  - backendName: ontap-flexvol
    storageDriverName: ontap-nas # only 'ontap-nas' and 'ontap-nas-
flexgroup' are supported when creating a backend via DeepOps
    managementLIF: 172.21.232.118# Cluster Management IP or SVM Mgmt LIF
IP
    dataLIF: 172.21.235.119# NFS LIF IP
    svm: infra-NFS-hci-ai# Name of SVM
    username: admin# Username to connect to the ONTAP cluster
    password:  P@ssw0rd# Password to login
    storagePrefix: trident
    limitAggregateUsage: ""
    limitVolumeSize: ""
    nfsMountOptions: ""
    defaults:
      spaceReserve: none
      snapshotPolicy: none
      snapshotReserve: 0
```

```
      splitOnClone: false
      encryption: false
      unixPermissions: 777
      snapshotDir: false
      exportPolicy: default
      securityStyle: unix
      tieringPolicy: none
# Add additional backends as needed
# Denotes whether or not to create new StorageClasses for your NetApp
storage
# For more info, refer to: https://netapp-
trident.readthedocs.io/en/stable-v20.04/kubernetes/operator-
install.html#creating-a-storage-class
create_StorageClasses: true
# List of StorageClasses to create
# Note: Each item in the list should be an actual K8s StorageClass
definition in yaml format
# For more info on StorageClass definitions, refer to https://netapp-
trident.readthedocs.io/en/stable-
v20.04/kubernetes/concepts/objects.html#kubernetes-storageclass-objects.
storageClasses_to_create:
  - apiVersion: storage.k8s.io/v1
    kind: StorageClass
    metadata:
      name: ontap-flexvol
      annotations:
        storageclass.kubernetes.io/is-default-class: "true"
    provisioner: csi.trident.netapp.io
    parameters:
      backendType: "ontap-nas"
# Add additional StorageClasses as needed
# Denotes whether or not to copy tridenctl binary to localhost
copy_tridentctl_to_localhost: true
# Directory that tridentctl will be copied to on localhost
tridentctl_copy_to_directory: ../ # will be copied to 'deepops/'
directory
```

2. Setup NetApp Trident by using the Ansible playbook.

```
ansible-playbook -l k8s-cluster playbooks/netapp-trident.yml
```

3. Verify that Trident is running.

```
./tridentctl -n trident version
```

The expected output is as follows:

```
rarvind@deployment-jump:~/deepops$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 20.07.0        | 20.07.0        |
+----------------+----------------+
```

Next: Deploy NVIDIA Triton Inference Server (Automated Deployment)

Deploy NVIDIA Triton Inference Server (Automated Deployment)

To set up automated deployment for the Triton Inference Server, complete the following steps:

1. Open a VI editor and create a PVC yaml file `vi pvc-triton-model- repo.yaml`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: triton-pvc  namespace: triton
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: ontap-flexvol
```

2. Create the PVC.

```
kubectl create -f pvc-triton-model-repo.yaml
```

3. Open a VI editor, create a deployment for the Triton Inference Server, and call the file `triton_deployment.yaml`.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: triton-3gpu
  name: triton-3gpu
  namespace: triton
```

```
spec:
  ports:
  - name: grpc-trtis-serving
    port: 8001
    targetPort: 8001
  - name: http-trtis-serving
    port: 8000
    targetPort: 8000
  - name: prometheus-metrics
    port: 8002
    targetPort: 8002
  selector:
    app: triton-3gpu
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: triton-1gpu
  name: triton-1gpu
  namespace: triton
spec:
  ports:
  - name: grpc-trtis-serving
    port: 8001
    targetPort: 8001
  - name: http-trtis-serving
    port: 8000
    targetPort: 8000
  - name: prometheus-metrics
    port: 8002
    targetPort: 8002
  selector:
    app: triton-1gpu
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-3gpu
  name: triton-3gpu
  namespace: triton
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: triton-3gpu      version: v1
  template:
    metadata:
      labels:
        app: triton-3gpu
        version: v1
    spec:
      containers:
      - image: nvcr.io/nvidia/tritonserver:20.07-v1-py3
        command: ["/bin/sh", "-c"]
        args: ["trtserver --model-store=/mnt/model-repo"]
        imagePullPolicy: IfNotPresent
        name: triton-3gpu
        ports:
        - containerPort: 8000
        - containerPort: 8001
        - containerPort: 8002
        resources:
          limits:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 3
          requests:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 3
        volumeMounts:
        - name: triton-model-repo
          mountPath: /mnt/model-repo      nodeSelector:
        gpu-count: “3”
      volumes:
      - name: triton-model-repo
        persistentVolumeClaim:
          claimName: triton-pvc---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-1gpu
  name: triton-1gpu
  namespace: triton
spec:
  replicas: 3
  selector:
```

```
    matchLabels:
      app: triton-1gpu
      version: v1
  template:
    metadata:
      labels:
        app: triton-1gpu
        version: v1
    spec:
      containers:
      - image: nvcr.io/nvidia/tritonserver:20.07-v1-py3
        command: ["/bin/sh", "-c", "sleep 1000"]
        args: ["trtserver --model-store=/mnt/model-repo"]
        imagePullPolicy: IfNotPresent
        name: triton-1gpu
        ports:
        - containerPort: 8000
        - containerPort: 8001
        - containerPort: 8002
        resources:
          limits:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 1
          requests:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 1
        volumeMounts:
        - name: triton-model-repo
          mountPath: /mnt/model-repo      nodeSelector:
        gpu-count: "1"
      volumes:
      - name: triton-model-repo
        persistentVolumeClaim:
          claimName: triton-pvc
```

Two deployments are created here as an example. The first deployment spins up a pod that uses three GPUs and has replicas set to 1. The other deployment spins up three pods each using one GPU while the replica is set to 3. Depending on your requirements, you can change the GPU allocation and replica counts.

Both of the deployments use the PVC created earlier and this persistent storage is provided to the Triton inference servers as the model repository.

For each deployment, a service of type LoadBalancer is created. The Triton Inference Server can be accessed by using the LoadBalancer IP which is in the application network.

A nodeSelector is used to ensure that both deployments get the required number of GPUs without any issues.

4. Label the K8 worker nodes.

```
kubectl label nodes hci-ai-k8-worker-01 gpu-count=3
kubectl label nodes hci-ai-k8-worker-02 gpu-count=1
```

5. Create the deployment.

```
kubectl apply -f triton_deployment.yaml
```

6. Make a note of the LoadBalancer service external LPS.

```
kubectl get services -n triton
```

The expected sample output is as follows:

```
rarvind@deployment-jump:~/triton-inference-server$ kubectl get services -n triton
NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)                                        AGE
triton-1gpu-v20-07-v1   LoadBalancer   10.233.21.185   172.21.231.133   8001:31238/TCP,8000:30171/TCP,8002:32348/TCP   10h
triton-3gpu-v20-07-v1   LoadBalancer   10.233.13.17    172.21.231.132   8001:31549/TCP,8000:30220/TCP,8002:31517/TCP   10h
```

7. Connect to any one of the pods that were created from the deployment.

```
kubectl exec -n triton --stdin --tty triton-1gpu-86c4c8dd64-545lx --
/bin/bash
```

8. Set up the model repository by using the example model repository.

```
git clone
cd triton-inference-server
git checkout r20.07
```

9. Fetch any missing model definition files.

```
cd docs/examples
./fetch_models.sh
```

10. Copy all the models to the model repository location or just a specific model that you wish to use.

```
cp -r model_repository/resnet50_netdef/ /mnt/model-repo/
```

In this solution, only the resnet50_netdef model is copied over to the model repository as an example.

11. Check the status of the Triton Inference Server.

```
curl -v <<LoadBalancer_IP_recorded earlier>>:8000/api/status
```

The expected sample output is as follows:

```
curl -v 172.21.231.132:8000/api/status
*   Trying 172.21.231.132...
* TCP_NODELAY set
* Connected to 172.21.231.132 (172.21.231.132) port 8000 (#0)
> GET /api/status HTTP/1.1
> Host: 172.21.231.132:8000
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< NV-Status: code: SUCCESS server_id: "inference:0" request_id: 9
< Content-Length: 1124
< Content-Type: text/plain
<
id: "inference:0"
version: "1.15.0"
uptime_ns: 377890294368
model_status {
  key: "resnet50_netdef"
  value {
    config {
      name: "resnet50_netdef"
      platform: "caffe2_netdef"
      version_policy {
        latest {
          num_versions: 1
        }
      }
      max_batch_size: 128
      input {
        name: "gpu_0/data"
        data_type: TYPE_FP32
        format: FORMAT_NCHW
        dims: 3
        dims: 224
        dims: 224
      }
```

```
      output {
        name: "gpu_0/softmax"
        data_type: TYPE_FP32
        dims: 1000
        label_filename: "resnet50_labels.txt"
      }
      instance_group {
        name: "resnet50_netdef"
        count: 1
        gpus: 0
        gpus: 1
        gpus: 2
        kind: KIND_GPU
      }
      default_model_filename: "model.netdef"
      optimization {
        input_pinned_memory {
          enable: true
        }
        output_pinned_memory {
          enable: true
        }
      }
    }
    version_status {
      key: 1
      value {
        ready_state: MODEL_READY
        ready_state_reason {
        }
      }
    }
  }
}
ready_state: SERVER_READY
* Connection #0 to host 172.21.231.132 left intact
```

Next: Deploy the Client for Triton Inference Server (Automated Deployment)

Deploy the Client for Triton Inference Server (Automated Deployment)

To deploy the client for the Triton Inference Server, complete the following steps:

1. Open a VI editor, create a deployment for the Triton client, and call the file `triton_client.yaml`.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-client
  name: triton-client
  namespace: triton
spec:
  replicas: 1
  selector:
    matchLabels:
      app: triton-client
      version: v1
  template:
    metadata:
      labels:
        app: triton-client
        version: v1
    spec:
      containers:
      - image: nvcr.io/nvidia/tritonserver:20.07- v1- py3-clientsdk
        imagePullPolicy: IfNotPresent
        name: triton-client
        resources:
          limits:
            cpu: "2"
            memory: 4Gi
          requests:
            cpu: "2"
            memory: 4Gi
```

2. Deploy the client.

```
kubectl apply -f triton_client.yaml
```

Next: Collect Inference Metrics from Triton Inference Server

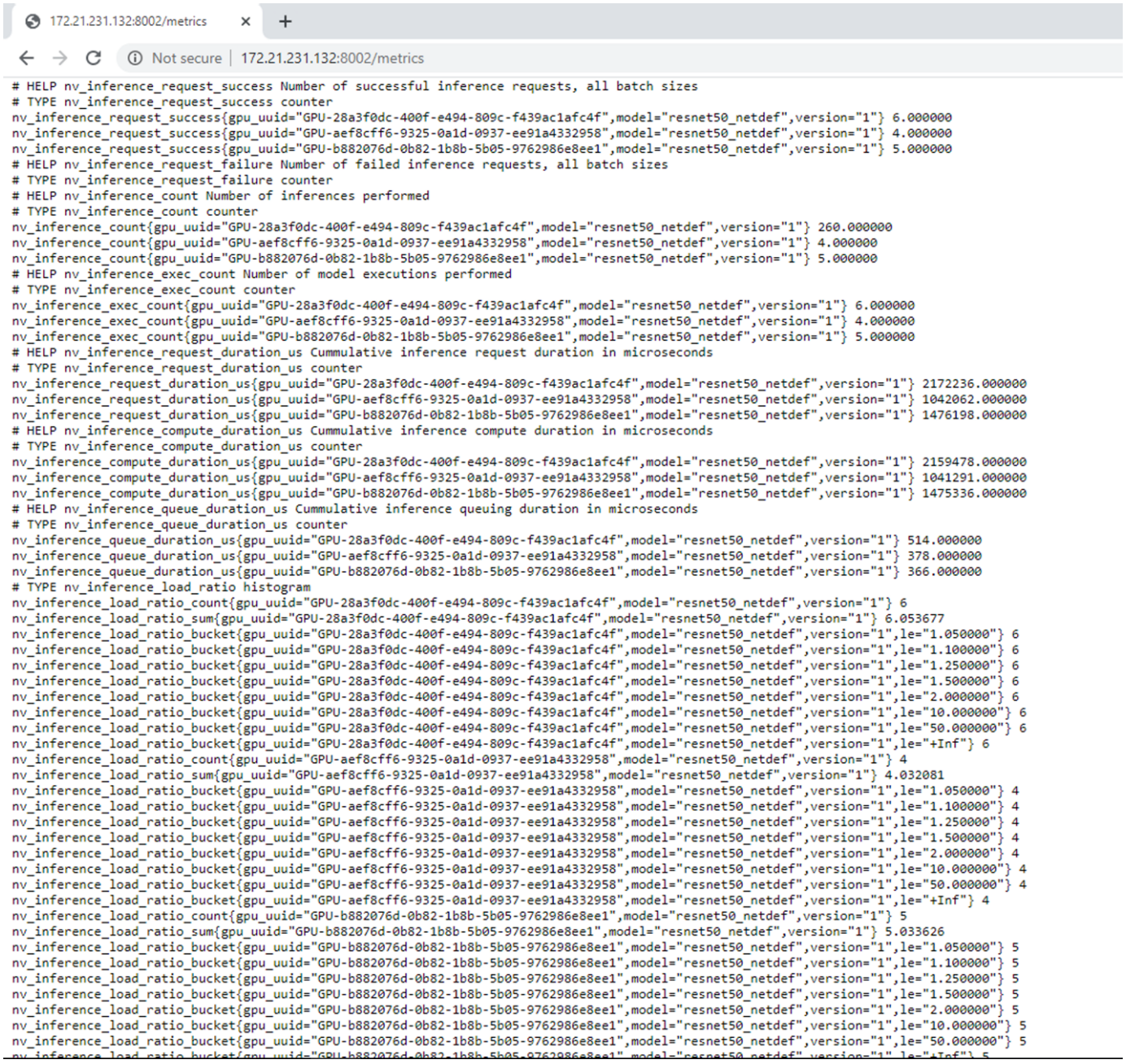
Collect Inference Metrics from Triton Inference Server

The Triton Inference Server provides Prometheus metrics indicating GPU and request statistics.

By default, these metrics are available at "http://<triton_inference_server_IP>:8002/metrics".

The Triton Inference Server IP is the LoadBalancer IP that was recorded earlier.

The metrics are only available by accessing the endpoint and are not pushed or published to any remote server.




```
# HELP nv_inference_request_success Number of successful inference requests, all batch sizes
# TYPE nv_inference_request_success counter
nv_inference_request_success{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6.000000
nv_inference_request_success{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.000000
nv_inference_request_success{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.000000
# HELP nv_inference_request_failure Number of failed inference requests, all batch sizes
# TYPE nv_inference_request_failure counter
# HELP nv_inference_count Number of inferences performed
# TYPE nv_inference_count counter
nv_inference_count{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 260.000000
nv_inference_count{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.000000
nv_inference_count{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.000000
# HELP nv_inference_exec_count Number of model executions performed
# TYPE nv_inference_exec_count counter
nv_inference_exec_count{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6.000000
nv_inference_exec_count{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.000000
nv_inference_exec_count{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.000000
# HELP nv_inference_request_duration_us Cummulative inference request duration in microseconds
# TYPE nv_inference_request_duration_us counter
nv_inference_request_duration_us{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 2172236.000000
nv_inference_request_duration_us{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 1042062.000000
nv_inference_request_duration_us{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 1476198.000000
# HELP nv_inference_compute_duration_us Cummulative inference compute duration in microseconds
# TYPE nv_inference_compute_duration_us counter
nv_inference_compute_duration_us{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 2159478.000000
nv_inference_compute_duration_us{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 1041291.000000
nv_inference_compute_duration_us{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 1475336.000000
# HELP nv_inference_queue_duration_us Cummulative inference queuing duration in microseconds
# TYPE nv_inference_queue_duration_us counter
nv_inference_queue_duration_us{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 514.000000
nv_inference_queue_duration_us{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 378.000000
nv_inference_queue_duration_us{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 366.000000
# TYPE nv_inference_load_ratio histogram
nv_inference_load_ratio_count{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6
nv_inference_load_ratio_sum{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6.053677
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.050000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.100000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.250000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.500000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="2.000000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="10.000000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="50.000000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="+Inf"} 6
nv_inference_load_ratio_count{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4
nv_inference_load_ratio_sum{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.032081
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.050000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.100000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.250000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.500000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="2.000000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="10.000000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="50.000000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="+Inf"} 4
nv_inference_load_ratio_count{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5
nv_inference_load_ratio_sum{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.033626
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.050000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.100000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.250000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.500000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="2.000000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="10.000000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="50.000000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="+Inf"} 5
```

```
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="+Inf"} 5
# HELP nv_gpu_utilization GPU utilization rate [0.0 - 1.0)
# TYPE nv_gpu_utilization gauge
nv_gpu_utilization{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 0.000000
nv_gpu_utilization{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 0.000000
nv_gpu_utilization{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 0.000000
# HELP nv_gpu_memory_total_bytes GPU total memory, in bytes
# TYPE nv_gpu_memory_total_bytes gauge
nv_gpu_memory_total_bytes{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 15843721216.000000
nv_gpu_memory_total_bytes{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 15843721216.000000
nv_gpu_memory_total_bytes{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 15843721216.000000
# HELP nv_gpu_memory_used_bytes GPU used memory, in bytes
# TYPE nv_gpu_memory_used_bytes gauge
nv_gpu_memory_used_bytes{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 1466236928.000000
nv_gpu_memory_used_bytes{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 13004767232.000000
nv_gpu_memory_used_bytes{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 1466236928.000000
# HELP nv_gpu_power_usage GPU power usage in watts
# TYPE nv_gpu_power_usage gauge
nv_gpu_power_usage{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 27.999000
nv_gpu_power_usage{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 28.428000
nv_gpu_power_usage{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 27.632000
# HELP nv_gpu_power_limit GPU power management limit in watts
# TYPE nv_gpu_power_limit gauge
nv_gpu_power_limit{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 70.000000
nv_gpu_power_limit{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 70.000000
nv_gpu_power_limit{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 70.000000
# HELP nv_energy_consumption GPU energy consumption in joules since the Triton Server started
# TYPE nv_energy_consumption counter
nv_energy_consumption{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 9796.449000
nv_energy_consumption{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 9997.538000
nv_energy_consumption{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 9669.536000
```

Next: Validation Results

Validation Results

To run a sample inference request, complete the following steps:

1. Get a shell to the client container/pod.

   ```
   kubectl exec --stdin --tty <<client_pod_name>> -- /bin/bash
   ```

2. Run a sample inference request.

   ```
   image_client -m resnet50_netdef -s INCEPTION -u
   <<LoadBalancer_IP_recorded earlier>>:8000 -c 3 images/mug.jpg
   ```

   ```
   root@triton-client-v20-07-v1-5566895bc-zqz6w:/workspace# image_client -m resnet50_netdef -s INCEPTION -u 172.21.231.133:8000 -c 3 images/mug.jpg
   Request 0, batch size 1
   Image 'images/mug.jpg':
       504 (COFFEE MUG) = 0.723991
       968 (CUP) = 0.270953
       967 (ESPRESSO) = 0.00115996
   ```

   This inferencing request calls the `resnet50_netdef` model that is used for image recognition. Other clients can also send inferencing requests concurrently by following a similar approach and calling out the appropriate model.

Next: Where to Find Additional Information

Additional Information

To learn more about the information that is described in this document, review the following documents and/or websites:

- NetApp HCI Theory of Operations

  https://www.netapp.com/us/media/wp-7261.pdf

- NetApp Product Documentation

  docs.netapp.com

- NetApp HCI Solution Catalog Documentation

  https://docs.netapp.com/us-en/hci/solutions/index.html

- HCI Resources page

  https://mysupport.netapp.com/info/web/ECMLP2831412.html

- ONTAP Select

  https://www.netapp.com/us/products/data-management-software/ontap-select-sds.aspx

- NetApp Trident

  https://netapp-trident.readthedocs.io/en/stable-v20.01/

- NVIDIA DeepOps

  https://github.com/NVIDIA/deepops

- NVIDIA Triton Inference Server

  https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-master-branch-guide/docs/index.html

WP-7328: NetApp Conversational AI Using NVIDIA Jarvis

Rick Huang, Sung-Han Lin, NetApp
Davide Onofrio, NVIDIA

The NVIDIA DGX family of systems is made up of the world's first integrated artificial intelligence (AI)-based systems that are purpose-built for enterprise AI. NetApp AFF storage systems deliver extreme performance and industry-leading hybrid cloud data-management capabilities. NetApp and NVIDIA have partnered to create the NetApp ONTAP AI reference architecture, a turnkey solution for AI and machine learning (ML) workloads that provides enterprise-class performance, reliability, and support.

This white paper gives directional guidance to customers building conversational AI systems in support of different use cases in various industry verticals. It includes information about the deployment of the system using NVIDIA Jarvis. The tests were performed using an NVIDIA DGX Station and a NetApp AFF A220 storage system.

The target audience for the solution includes the following groups:

- Enterprise architects who design solutions for the development of AI models and software for conversational AI use cases such as a virtual retail assistant
- Data scientists looking for efficient ways to achieve language modeling development goals

- Data engineers in charge of maintaining and processing text data such as customer questions and dialogue transcripts
- Executive and IT decision makers and business leaders interested in transforming the conversational AI experience and achieving the fastest time to market from AI initiatives

Next: Solution Overview

Solution Overview

NetApp ONTAP AI and Cloud Sync

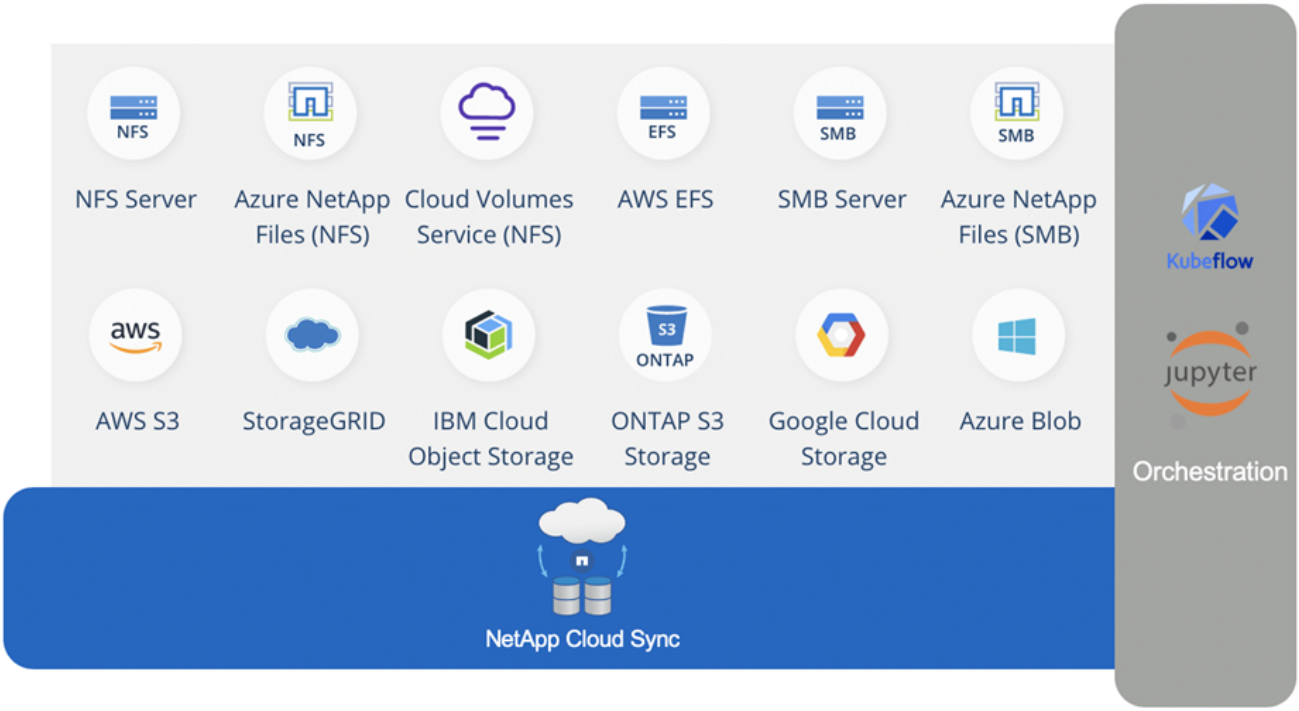
The NetApp ONTAP AI architecture, powered by NVIDIA DGX systems and NetApp cloud-connected storage systems, was developed and verified by NetApp and NVIDIA. This reference architecture gives IT organizations the following advantages:

- Eliminates design complexities
- Enables independent scaling of compute and storage
- Enables customers to start small and scale seamlessly
- Offers a range of storage options for various performance and cost pointsNetApp ONTAP AI tightly integrates DGX systems and NetApp AFF A220 storage systems with state-of-the-art networking. NetApp ONTAP AI and DGX systems simplify AI deployments by eliminating design complexity and guesswork. Customers can start small and grow their systems in an uninterrupted manner while intelligently managing data from the edge to the core to the cloud and back.

NetApp Cloud Sync enables you to move data easily over various protocols, whether it's between two NFS shares, two CIFS shares, or one file share and Amazon S3, Amazon Elastic File System (EFS), or Azure Blob storage. Active-active operation means that you can continue to work with both source and target at the same time, incrementally synchronizing data changes when required. By enabling you to move and incrementally synchronize data between any source and destination system, whether on-premises or cloud-based, Cloud Sync opens up a wide variety of new ways in which you can use data. Migrating data between on-premises systems, cloud on-boarding and cloud migration, or collaboration and data analytics all become easily achievable. The figure below shows available sources and destinations.

In conversational AI systems, developers can leverage Cloud Sync to archive conversation history from the cloud to data centers to enable offline training of natural language processing (NLP) models. By training models to recognize more intents, the conversational AI system will be better equipped to manage more complex questions from end-users.

NVIDIA Jarvis Multimodal Framework



NVIDIA Jarvis is an end-to-end framework for building conversational AI services. It includes the following GPU-optimized services:

- Automatic speech recognition (ASR)
- Natural language understanding (NLU)
- Integration with domain-specific fulfillment services
- Text-to-speech (TTS)
- Computer vision (CV)Jarvis-based services use state-of-the-art deep learning models to address the complex and challenging task of real-time conversational AI. To enable real-time, natural interaction with an end user, the models need to complete computation in under 300 milliseconds. Natural interactions are challenging, requiring multimodal sensory integration. Model pipelines are also complex and require coordination across the above services.

Jarvis is a fully accelerated, application framework for building multimodal conversational AI services that use an end-to-end deep learning pipeline. The Jarvis framework includes pretrained conversational AI models, tools, and optimized end-to-end services for speech, vision, and NLU tasks. In addition to AI services, Jarvis enables you to fuse vision, audio, and other sensor inputs simultaneously to deliver capabilities such as multi-user, multi-context conversations in applications such as virtual assistants, multi-user diarization, and call center assistants.

NVIDIA NeMo

NVIDIA NeMo is an open-source Python toolkit for building, training, and fine-tuning GPU-accelerated state-of-the-art conversational AI models using easy-to-use application programming interfaces (APIs). NeMo runs mixed precision compute using Tensor Cores in NVIDIA GPUs and can scale up to multiple GPUs easily to deliver the highest training performance possible. NeMo is used to build models for real-time ASR, NLP, and TTS applications such as video call transcriptions, intelligent video assistants, and automated call center support across different industry verticals, including healthcare, finance, retail, and telecommunications.

We used NeMo to train models that recognize complex intents from user questions in archived conversation history. This training extends the capabilities of the retail virtual assistant beyond what Jarvis supports as

delivered.

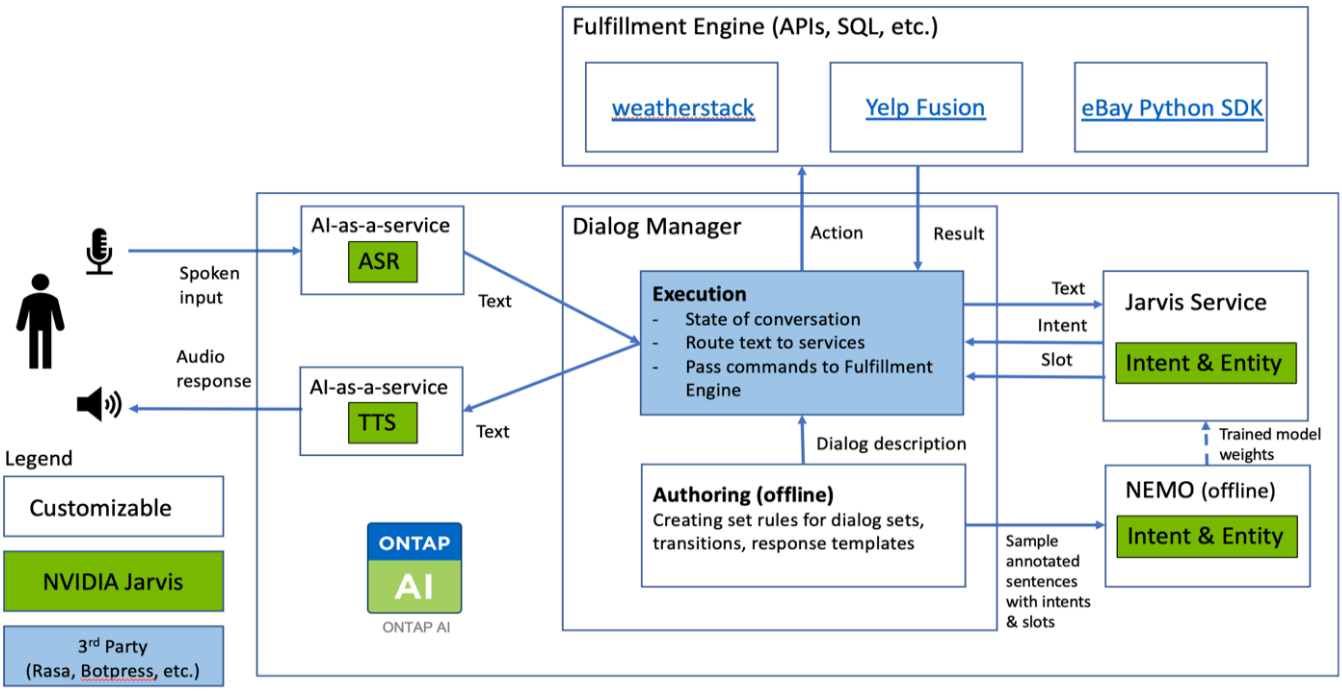
Retail Use Case Summary

Using NVIDIA Jarvis, we built a virtual retail assistant that accepts speech or text input and answers questions regarding weather, points-of-interest, and inventory pricing. The conversational AI system is able to remember conversation flow, for example, ask a follow-up question if the user does not specify location for weather or points-of-interest. The system also recognizes complex entities such as “Thai food” or “laptop memory.” It understands natural language questions like “will it rain next week in Los Angeles?” A demonstration of the retail virtual assistant can be found in Customize States and Flows for Retail Use Case.

Next: Solution Technology

Solution Technology

The following figure illustrates the proposed conversational AI system architecture. You can interact with the system with either speech signal or text input. If spoken input is detected, Jarvis AI-as-service (AIaaS) performs ASR to produce text for Dialog Manager. Dialog Manager remembers states of conversation, routes text to corresponding services, and passes commands to Fulfillment Engine. Jarvis NLP Service takes in text, recognizes intents and entities, and outputs those intents and entity slots back to Dialog Manager, which then sends Action to Fulfillment Engine. Fulfillment Engine consists of third-party APIs or SQL databases that answer user queries. After receiving Result from Fulfillment Engine, Dialog Manager routes text to Jarvis TTS AIaaS to produce an audio response for the end-user. We can archive conversation history, annotate sentences with intents and slots for NeMo training such that NLP Service improves as more users interact with the system.



Hardware Requirements

This solution was validated using one DGX Station and one AFF A220 storage system. Jarvis requires either a T4 or V100 GPU to perform deep neural network computations.

The following table lists the hardware components that are required to implement the solution as tested.

Hardware	Quantity
T4 or V100 GPU	1
NVIDIA DGX Station	1

Software Requirements

The following table lists the software components that are required to implement the solution as tested.

Software	Version or Other Information
NetApp ONTAP data management software	9.6
Cisco NX-OS switch firmware	7.0(3)I6(1)
NVIDIA DGX OS	4.0.4 - Ubuntu 18.04 LTS
NVIDIA Jarvis Framework	EA v0.2
NVIDIA NeMo	nvcr.io/nvidia/nemo:v0.10
Docker container platform	18.06.1-ce [e68fc7a]

Next: Build a Virtual Assistant Using Jarvis, Cloud Sync, and NeMo Overview

Overview

This section provides detail on the implementation of the virtual retail assistant.

Next: Jarvis Deployment

Jarvis Deployment

You can sign up for Jarvis Early Access program to gain access to Jarvis containers on NVIDIA GPU Cloud (NGC). After receiving credentials from NVIDIA, you can deploy Jarvis using the following steps:

1. Sign-on to NGC.
2. Set your organization on NGC: `ea-2-jarvis`.
3. Locate Jarvis EA v0.2 assets: Jarvis containers are in `Private Registry` > `Organization Containers`.
4. Select Jarvis: navigate to `Model Scripts` and click `Jarvis Quick Start`
5. Verify that all assets are working properly.
6. Find the documentation to build your own applications: PDFs can be found in `Model Scripts` > `Jarvis Documentation` > `File Browser`.

Next: Customize States and Flows for Retail Use Case

Customize States and Flows for Retail Use Case

You can customize States and Flows of Dialog Manager for your specific use cases. In

our retail example, we have the following four yaml files to direct the conversation according to different intents.

Se the following list of file names and description of each file:

- `main_flow.yml`: Defines the main conversation flows and states and directs the flow to the other three yaml files when necessary.
- `retail_flow.yml`: Contains states related to retail or points-of-interest questions. The system either provides the information of the nearest store, or the price of a given item.
- `weather_flow.yml`: Contains states related to weather questions. If the location cannot be determined, the system asks a follow up question to clarify.
- `error_flow.yml`: Handles cases where user intents do not fall into the above three yaml files. After displaying an error message, the system re-routes back to accepting user questions.The following sections contain the detailed definitions for these yaml files.

main_flow.yml

```
name: JarvisRetail
intent_transitions:
  jarvis_error: error
  price_check: retail_price_check
  inventory_check: retail_inventory_check
  store_location: retail_store_location
  weather.weather: weather
  weather.temperature: temperature
  weather.sunny: sunny
  weather.cloudy: cloudy
  weather.snow: snow
  weather.rainfall: rain
  weather.snow_yes_no: snowfall
  weather.rainfall_yes_no: rainfall
  weather.temperature_yes_no: tempyesno
  weather.humidity: humidity
  weather.humidity_yes_no: humidity
  navigation.startnavigationpoi: retail # Transitions should be context
and slot based. Redirecting for now.
  navigation.geteta: retail
  navigation.showdirection: retail
  navigation.showmappoi: idk_what_you_talkin_about
  nomatch.none: idk_what_you_talkin_about
states:
  init:
    type: message_text
    properties:
      text: "Hi, welcome to NARA retail and weather service. How can I
help you?"
```

```
  input_intent:
    type: input_context
    properties:
      nlp_type: jarvis
      entities:
        intent: dontcare
# This state is executed if the intent was not understood
  dont_get_the intent:
    type: message_text_random
    properties:
      responses:
        - "Sorry I didn't get that! Please come again."
        - "I beg your pardon! Say that again?"
        - "Are we talking about weather? What would you like to know?"
        - "Sorry I know only about the weather"
        - "You can ask me about the weather, the rainfall, the
temperature, I don't know much more"
      delay: 0
    transitions:
      next_state: input_intent
  idk_what_you_talkin_about:
    type: message_text_random
    properties:
      responses:
        - "Sorry I didn't get that! Please come again."
        - "I beg your pardon! Say that again?"
        - "Are we talking about retail or weather? What would you like to
know?"
        - "Sorry I know only about retail and the weather"
        - "You can ask me about retail information or the weather, the
rainfall, the temperature. I don't know much more."
      delay: 0
    transitions:
      next_state: input_intent
  error:
    type: change_context
    properties:
        update_keys:
           intent: 'error'
    transitions:
        flow: error_flow
  retail_inventory_check:
    type: change_context
    properties:
        update_keys:
           intent: 'retail_inventory_check'
```

```
  transitions:
      flow: retail_flow
retail_price_check:
  type: change_context
  properties:
      update_keys:
         intent: 'check_item_price'
  transitions:
      flow: retail_flow
retail_store_location:
  type: change_context
  properties:
      update_keys:
         intent: 'find_the_store'
  transitions:
      flow: retail_flow
weather:
  type: change_context
  properties:
      update_keys:
         intent: 'weather'
  transitions:
      flow: weather_flow
temperature:
  type: change_context
  properties:
      update_keys:
         intent: 'temperature'
  transitions:
      flow: weather_flow
rainfall:
  type: change_context
  properties:
      update_keys:
         intent: 'rainfall'
  transitions:
      flow: weather_flow
sunny:
  type: change_context
  properties:
      update_keys:
         intent: 'sunny'
  transitions:
      flow: weather_flow
cloudy:
  type: change_context
```

```
    properties:
        update_keys:
           intent: 'cloudy'
    transitions:
        flow: weather_flow
  snow:
    type: change_context
    properties:
        update_keys:
           intent: 'snow'
    transitions:
        flow: weather_flow
  rain:
    type: change_context
    properties:
        update_keys:
           intent: 'rain'
    transitions:
        flow: weather_flow
  snowfall:
      type: change_context
      properties:
          update_keys:
             intent: 'snowfall'
      transitions:
          flow: weather_flow
  tempyesno:
      type: change_context
      properties:
          update_keys:
             intent: 'tempyesno'
      transitions:
          flow: weather_flow
  humidity:
      type: change_context
      properties:
          update_keys:
             intent: 'humidity'
      transitions:
          flow: weather_flow
  end_state:
    type: reset
    transitions:
      next_state: init
```

retail_flow.yml

```
name: retail_flow
states:
  store_location:
    type: conditional_exists
    properties:
      key: '{{location}}'
    transitions:
      exists: retail_state
      notexists: ask_retail_location
  retail_state:
    type: Retail
    properties:
    transitions:
      next_state: output_retail
  output_retail:
      type: message_text
      properties:
        text: '{{retail_status}}'
      transitions:
        next_state: input_intent
  ask_retail_location:
    type: message_text
    properties:
      text: "For which location? I can find the closest store near you."
    transitions:
      next_state: input_retail_location
  input_retail_location:
    type: input_user
    properties:
      nlp_type: jarvis
      entities:
        slot: location
      require_match: true
    transitions:
      match: retail_state
      notmatch: check_retail_jarvis_error
  output_retail_acknowledge:
    type: message_text_random
    properties:
      responses:
        - 'ok in {{location}}'
        - 'the store in {{location}}'
        - 'I always wanted to shop in {{location}}'
      delay: 0
```

```
    transitions:
      next_state: retail_state
  output_retail_notlocation:
    type: message_text
    properties:
      text: "I did not understand the location. Can you please repeat?"
    transitions:
      next_state: input_intent
  check_rerail_jarvis_error:
    type: conditional_exists
    properties:
      key: '{{jarvis_error}}'
    transitions:
      exists: show_retail_jarvis_api_error
      notexists: output_retail_notlocation
  show_retail_jarvis_api_error:
    type: message_text
    properties:
      text: "I am having troubled understanding right now. Come again on
that?"
    transitions:
      next_state: input_intent
```

weather_flow.yml

```
name: weather_flow
states:
  check_weather_location:
    type: conditional_exists
    properties:
      key: '{{location}}'
    transitions:
      exists: weather_state
      notexists: ask_weather_location
  weather_state:
    type: Weather
    properties:
    transitions:
      next_state: output_weather
  output_weather:
      type: message_text
      properties:
        text: '{{weather_status}}'
      transitions:
        next_state: input_intent
```

```
  ask_weather_location:
    type: message_text
    properties:
      text: "For which location?"
    transitions:
      next_state: input_weather_location
  input_weather_location:
    type: input_user
    properties:
      nlp_type: jarvis
      entities:
        slot: location
      require_match: true
    transitions:
      match: weather_state
      notmatch: check_jarvis_error
  output_weather_acknowledge:
    type: message_text_random
    properties:
      responses:
        - 'ok in {{location}}'
        - 'the weather in {{location}}'
        - 'I always wanted to go in {{location}}'
      delay: 0
    transitions:
      next_state: weather_state
  output_weather_notlocation:
    type: message_text
    properties:
      text: "I did not understand the location, can you please repeat?"
    transitions:
      next_state: input_intent
  check_jarvis_error:
    type: conditional_exists
    properties:
      key: '{{jarvis_error}}'
    transitions:
      exists: show_jarvis_api_error
      notexists: output_weather_notlocation
  show_jarvis_api_error:
    type: message_text
    properties:
      text: "I am having troubled understanding right now. Come again on
that, else check jarvis services?"
    transitions:
      next_state: input_intent
```

error_flow.yml

```
name: error_flow
states:
  error_state:
    type: message_text_random
    properties:
      responses:
        - "Sorry I didn't get that!"
        - "Are we talking about retail or weather? What would you like to
know?"
        - "Sorry I know only about retail information or the weather"
        - "You can ask me about retail information or the weather, the
rainfall, the temperature. I don't know much more"
        - "Let's talk about retail or the weather!"
      delay: 0
    transitions:
      next_state: input_intent
```

Next: Connect to Third-Party APIs as Fulfillment Engine

Connect to Third-Party APIs as Fulfillment Engine

We connected the following third-party APIs as a Fulfillment Engine to answer questions:

- WeatherStack API: returns weather, temperature, rainfall, and snow in a given location.
- Yelp Fusion API: returns the nearest store information in a given location.
- eBay Python SDK: returns the price of a given item.

Next: NetApp Retail Assistant Demonstration

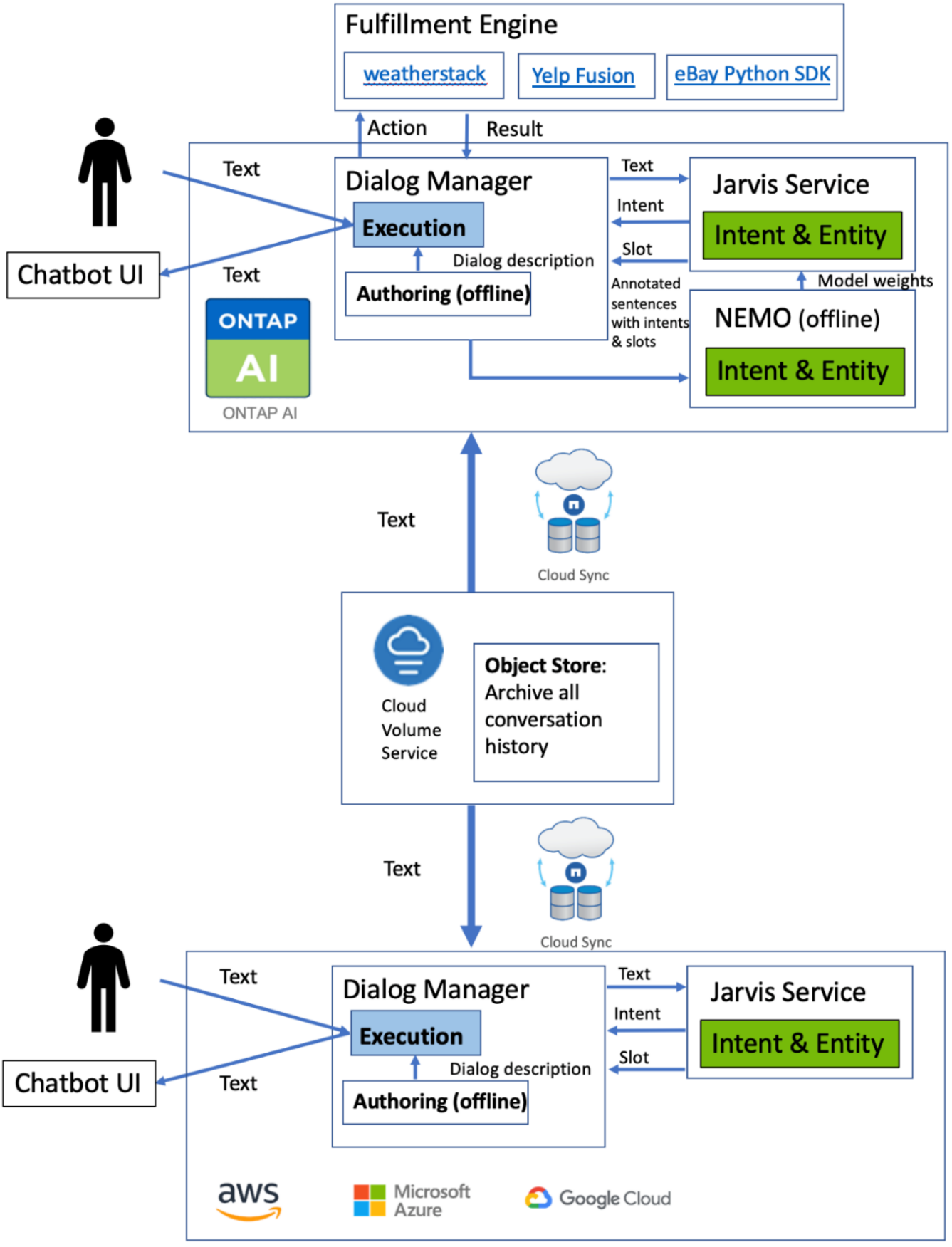
NetApp Retail Assistant Demonstration

We recorded a demonstration video of NetApp Retail Assistant (NARA). Click this link to open the following figure and play the video demonstration.

Next: Use NetApp Cloud Sync to Archive Conversation History

Use NetApp Cloud Sync to Archive Conversation History

By dumping conversation history into a CSV file once a day, we can then leverage Cloud Sync to download the log files into local storage. The following figure shows the architecture of having Jarvis deployed on-premises and in public clouds, while using Cloud Sync to send conversation history for NeMo training. Details of NeMo training can be found in the section Expand Intent Models Using NeMo Training.



Next: Expand Intent Models Using NeMo Training

Expand Intent Models Using NeMo Training

NVIDIA NeMo is a toolkit built by NVIDIA for creating conversational AI applications. This toolkit includes collections of pre-trained modules for ASR, NLP, and TTS, enabling researchers and data scientists to easily compose complex neural network architectures and put more focus on designing their own applications.

As shown in the previous example, NARA can only handle a limited type of question. This is because the pre-trained NLP model only trains on these types of questions. If we want to enable NARA to handle a broader range of questions, we need to retrain it with our own datasets. Thus, here, we demonstrate how we can use NeMo to extend the NLP model to satisfy the requirements. We start by converting the log collected from NARA into the format for NeMo, and then train with the dataset to enhance the NLP model.

Model

Our goal is to enable NARA to sort the items based on user preferences. For instance, we might ask NARA to suggest the highest-rated sushi restaurant or might want NARA to look up the jeans with the lowest price. To this end, we use the intent detection and slot filling model provided in NeMo as our training model. This model allows NARA to understand the intent of searching preference.

Data Preparation

To train the model, we collect the dataset for this type of question, and convert it to the NeMo format. Here, we listed the files we use to train the model.

dict.intents.csv

This file lists all the intents we want the NeMo to understand. Here, we have two primary intents and one intent only used to categorize the questions that do not fit into any of the primary intents.

```
price_check
find_the_store
unknown
```

dict.slots.csv

This file lists all the slots we can label on our training questions.

```
B-store.type
B-store.name
B-store.status
B-store.hour.start
B-store.hour.end
B-store.hour.day
B-item.type
B-item.name
B-item.color
B-item.size
B-item.quantity
B-location
B-cost.high
```

```
B-cost.average
B-cost.low
B-time.period_of_time
B-rating.high
B-rating.average
B-rating.low
B-interrogative.location
B-interrogative.manner
B-interrogative.time
B-interrogative.personal
B-interrogative
B-verb
B-article
I-store.type
I-store.name
I-store.status
I-store.hour.start
I-store.hour.end
I-store.hour.day
I-item.type
I-item.name
I-item.color
I-item.size
I-item.quantity
I-location
I-cost.high
I-cost.average
I-cost.low
I-time.period_of_time
I-rating.high
I-rating.average
I-rating.low
I-interrogative.location
I-interrogative.manner
I-interrogative.time
I-interrogative.personal
I-interrogative
I-verb
I-article
O
```

train.tsv

This is the main training dataset. Each line starts with the question following the intent category listing in the file dict.intent.csv. The label is enumerated starting from zero.

train_slots.tsv

```
20 46 24 25 6 32 6
52 52 24 6
23 52 14 40 52 25 6 32 6
…
```

Train the Model

```
docker pull nvcr.io/nvidia/nemo:v0.10
```

We then use the following command to launch the container. In this command, we limit the container to use a single GPU (GPU ID = 1) since this is a lightweight training exercise. We also map our local workspace /workspace/nemo/ to the folder inside container /nemo.

```
NV_GPU='1' docker run --runtime=nvidia -it --shm-size=16g \
                        --network=host --ulimit memlock=-1 --ulimit
stack=67108864 \
                        -v /workspace/nemo:/nemo\
                        --rm nvcr.io/nvidia/nemo:v0.10
```

Inside the container, if we want to start from the original pre-trained BERT model, we can use the following command to start the training procedure. data_dir is the argument to set up the path of the training data. work_dir allows you to configure where you want to store the checkpoint files.

```
cd examples/nlp/intent_detection_slot_tagging/
python joint_intent_slot_with_bert.py \
    --data_dir /nemo/training_data\
    --work_dir /nemo/log
```

If we have new training datasets and want to improve the previous model, we can use the following command to continue from the point we stopped. checkpoint_dir takes the path to the previous checkpoints folder.

```
cd examples/nlp/intent_detection_slot_tagging/
python joint_intent_slot_infer.py \
    --data_dir /nemo/training_data \
    --checkpoint_dir /nemo/log/2020-05-04_18-34-20/checkpoints/ \
    --eval_file_prefix test
```

Inference the Model

We need to validate the performance of the trained model after a certain number of epochs. The following command allows us to test the query one-by-one. For instance, in this command, we want to check if our

model can properly identify the intention of the query `where can I get the best pasta`.

```
cd examples/nlp/intent_detection_slot_tagging/
python joint_intent_slot_infer_b1.py \
--checkpoint_dir /nemo/log/2020-05-29_23-50-58/checkpoints/ \
--query "where can i get the best pasta" \
--data_dir /nemo/training_data/ \
--num_epochs=50
```

Then, the following is the output from the inference. In the output, we can see that our trained model can properly predict the intention find_the_store, and return the keywords we are interested in. With these keywords, we enable the NARA to search for what users want and do a more precise search.

```
[NeMo I 2020-05-30 00:06:54 actions:728] Evaluating batch 0 out of 1
[NeMo I 2020-05-30 00:06:55 inference_utils:34] Query: where can i get the
best pasta
[NeMo I 2020-05-30 00:06:55 inference_utils:36] Predicted intent:       1
find_the_store
[NeMo I 2020-05-30 00:06:55 inference_utils:50] where   B-
interrogative.location
[NeMo I 2020-05-30 00:06:55 inference_utils:50] can     O
[NeMo I 2020-05-30 00:06:55 inference_utils:50] i       O
[NeMo I 2020-05-30 00:06:55 inference_utils:50] get     B-verb
[NeMo I 2020-05-30 00:06:55 inference_utils:50] the     B-article
[NeMo I 2020-05-30 00:06:55 inference_utils:50] best    B-rating.high
[NeMo I 2020-05-30 00:06:55 inference_utils:50] pasta   B-item.type
```

Next: Conclusion

Conclusion

A true conversational AI system engages in human-like dialogue, understands context, and provides intelligent responses. Such AI models are often huge and highly complex. With NVIDIA GPUs and NetApp storage, massive, state-of-the-art language models can be trained and optimized to run inference rapidly. This is a major stride towards ending the trade- off between an AI model that is fast versus one that is large and complex. GPU-optimized language understanding models can be integrated into AI applications for industries such as healthcare, retail, and financial services, powering advanced digital voice assistants in smart speakers and customer service lines. These high-quality conversational AI systems allow businesses across verticals to provide previously unattainable personalized services when engaging with customers.

Jarvis enables the deployment of use cases such as virtual assistants, digital avatars, multimodal sensor fusion (CV fused with ASR/NLP/TTS), or any ASR/NLP/TTS/CV stand-alone use case, such as transcription. We built a virtual retail assistant that can answer questions regarding weather, points-of-interest, and inventory pricing. We also demonstrated how to improve the natural language understanding capabilities of the conversational AI system by archiving conversation history using Cloud Sync and training NeMo models on new data.

Next: Acknowledgments

Acknowledgments

The authors gratefully acknowledge the contributions that were made to this white paper by our esteemed colleagues from NVIDIA: Davide Onofrio, Alex Qi, Sicong Ji, Marty Jain, and Robert Sohigian. The authors would also like to acknowledge the contributions of key NetApp team members: Santosh Rao, David Arnette, Michael Oglesby, Brent Davis, Andy Sayare, Erik Mulder, and Mike McNamara.

Our sincere appreciation and thanks go to all these individuals, who provided insight and expertise that greatly assisted in the creation of this paper.

Next: Where to Find Additional Information

Where to Find Additional Information

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

- NVIDIA DGX Station, V100 GPU, GPU Cloud
  - NVIDIA DGX Station
    https://www.nvidia.com/en-us/data-center/dgx-station/
  - NVIDIA V100 Tensor Core GPU
    https://www.nvidia.com/en-us/data-center/tesla-v100/
  - NVIDIA NGC
    https://www.nvidia.com/en-us/gpu-cloud/
- NVIDIA Jarvis Multimodal Framework
  - NVIDIA Jarvis
    https://developer.nvidia.com/nvidia-jarvis
  - NVIDIA Jarvis Early Access
    https://developer.nvidia.com/nvidia-jarvis-early-access
- NVIDIA NeMo
  - NVIDIA NeMo
    https://developer.nvidia.com/nvidia-nemo
  - Developer Guide
    https://nvidia.github.io/NeMo/
- NetApp AFF systems
  - NetApp AFF A-Series Datasheet
    https://www.netapp.com/us/media/ds-3582.pdf
  - NetApp Flash Advantage for All Flash FAS
    https://www.netapp.com/us/media/ds-3733.pdf
  - ONTAP 9 Information Library
    http://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62286
  - NetApp ONTAP FlexGroup Volumes technical report
    https://www.netapp.com/us/media/tr-4557.pdf

- NetApp ONTAP AI
  - ONTAP AI with DGX-1 and Cisco Networking Design Guide
    https://www.netapp.com/us/media/nva-1121-design.pdf
  - ONTAP AI with DGX-1 and Cisco Networking Deployment Guide
    https://www.netapp.com/us/media/nva-1121-deploy.pdf
  - ONTAP AI with DGX-1 and Mellanox Networking Design Guide
    http://www.netapp.com/us/media/nva-1138-design.pdf
  - ONTAP AI with DGX-2 Design Guide
    https://www.netapp.com/us/media/nva-1135-design.pdf

TR-4858: NetApp Orchestration Solution with Run:AI

Rick Huang, David Arnette, Sung-Han Lin, NetApp
Yaron Goldberg, Run:AI

NetApp AFF storage systems deliver extreme performance and industry-leading hybrid cloud data-management capabilities. NetApp and Run:AI have partnered to demonstrate the unique capabilities of the NetApp ONTAP AI solution for artificial intelligence (AI) and machine learning (ML) workloads that provides enterprise-class performance, reliability, and support. Run:AI orchestration of AI workloads adds a Kubernetes-based scheduling and resource utilization platform to help researchers manage and optimize GPU utilization. Together with the NVIDIA DGX systems, the combined solution from NetApp, NVIDIA, and Run:AI provide an infrastructure stack that is purpose-built for enterprise AI workloads. This technical report gives directional guidance to customers building conversational AI systems in support of various use cases and industry verticals. It includes information about the deployment of Run:AI and a NetApp AFF A800 storage system and serves as a reference architecture for the simplest way to achieve fast, successful deployment of AI initiatives.

The target audience for the solution includes the following groups:

- Enterprise architects who design solutions for the development of AI models and software for Kubernetes-based use cases such as containerized microservices
- Data scientists looking for efficient ways to achieve efficient model development goals in a cluster environment with multiple teams and projects
- Data engineers in charge of maintaining and running production models
- Executive and IT decision makers and business leaders who would like to create the optimal Kubernetes cluster resource utilization experience and achieve the fastest time to market from AI initiatives

Next: Solution Overview

Solution Overview

NetApp ONTAP AI and AI Control Plane

The NetApp ONTAP AI architecture, developed and verified by NetApp and NVIDIA, is powered by NVIDIA DGX systems and NetApp cloud-connected storage systems. This reference architecture gives IT organizations the following advantages:

- Eliminates design complexities
- Enables independent scaling of compute and storage
- Enables customers to start small and scale seamlessly

- Offers a range of storage options for various performance and cost points

NetApp ONTAP AI tightly integrates DGX systems and NetApp AFF A800 storage systems with state-of-the-art networking. NetApp ONTAP AI and DGX systems simplify AI deployments by eliminating design complexity and guesswork. Customers can start small and grow their systems in an uninterrupted manner while intelligently managing data from the edge to the core to the cloud and back.

NetApp AI Control Plane is a full stack AI, ML, and deep learning (DL) data and experiment management solution for data scientists and data engineers. As organizations increase their use of AI, they face many challenges, including workload scalability and data availability. NetApp AI Control Plane addresses these challenges through functionalities, such as rapidly cloning a data namespace just as you would a Git repo, and defining and implementing AI training workflows that incorporate the near-instant creation of data and model baselines for traceability and versioning. With NetApp AI Control Plane, you can seamlessly replicate data across sites and regions and swiftly provision Jupyter Notebook workspaces with access to massive datasets.

Run:AI Platform for AI Workload Orchestration

Run:AI has built the world's first orchestration and virtualization platform for AI infrastructure. By abstracting workloads from the underlying hardware, Run:AI creates a shared pool of GPU resources that can be dynamically provisioned, enabling efficient orchestration of AI workloads and optimized use of GPUs. Data scientists can seamlessly consume massive amounts of GPU power to improve and accelerate their research while IT teams retain centralized, cross-site control and real-time visibility over resource provisioning, queuing, and utilization. The Run:AI platform is built on top of Kubernetes, enabling simple integration with existing IT and data science workflows.

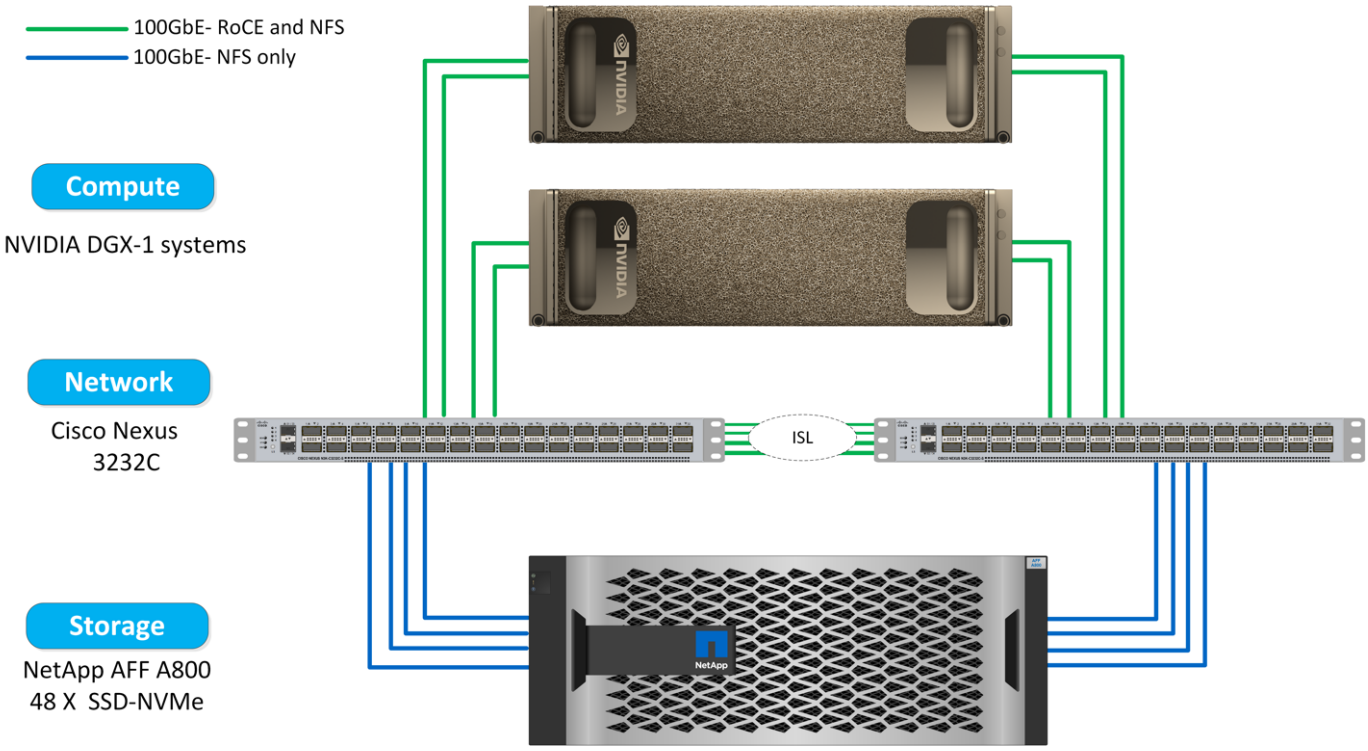
The Run:AI platform provides the following benefits:

- **Faster time to innovation.** By using Run:AI resource pooling, queueing, and prioritization mechanisms together with a NetApp storage system, researchers are removed from infrastructure management hassles and can focus exclusively on data science. Run:AI and NetApp customers increase productivity by running as many workloads as they need without compute or data pipeline bottlenecks.
- **Increased team productivity.** Run:AI fairness algorithms guarantee that all users and teams get their fair share of resources. Policies around priority projects can be preset, and the platform enables dynamic allocation of resources from one user or team to another, helping users to get timely access to coveted GPU resources.
- **Improved GPU utilization.** The Run:AI Scheduler enables users to easily make use of fractional GPUs, integer GPUs, and multiple nodes of GPUs for distributed training on Kubernetes. In this way, AI workloads run based on your needs, not capacity. Data science teams are able to run more AI experiments on the same infrastructure.

Next: Solution Technology

Solution Technology

This solution was implemented with one NetApp AFF A800 system, two DGX-1 servers, and two Cisco Nexus 3232C 100GbE-switches. Each DGX-1 server is connected to the Nexus switches with four 100GbE connections that are used for inter-GPU communications by using remote direct memory access (RDMA) over Converged Ethernet (RoCE). Traditional IP communications for NFS storage access also occur on these links. Each storage controller is connected to the network switches by using four 100GbE-links. The following figure shows the ONTAP AI solution architecture used in this technical report for all testing scenarios.



Hardware Used in This Solution

This solution was validated using the ONTAP AI reference architecture two DGX-1 nodes and one AFF A800 storage system. See NVA-1121 for more details about the infrastructure used in this validation.

The following table lists the hardware components that are required to implement the solution as tested.

Hardware	Quantity
DGX-1 systems	2
AFF A800	1
Nexus 3232C switches	2

Software Requirements

This solution was validated using a basic Kubernetes deployment with the Run:AI operator installed. Kubernetes was deployed using the NVIDIA DeepOps deployment engine, which deploys all required components for a production-ready environment. DeepOps automatically deployed NetApp Trident for persistent storage integration with the k8s environment, and default storage classes were created so containers leverage storage from the AFF A800 storage system. For more information on Trident with Kubernetes on ONTAP AI, see TR-4798.

The following table lists the software components that are required to implement the solution as tested.

Software	Version or Other Information
NetApp ONTAP data management software	9.6p4
Cisco NX-OS switch firmware	7.0(3)I6(1)
NVIDIA DGX OS	4.0.4 - Ubuntu 18.04 LTS

Software	Version or Other Information
Kubernetes version	1.17
Trident version	20.04.0
Run:AI CLI	v2.1.13
Run:AI Orchestration Kubernetes Operator version	1.0.39
Docker container platform	18.06.1-ce [e68fc7a]

Additional software requirements for Run:AI can be found at Run:AI GPU cluster prerequisites.

Next: Optimal Cluster and GPU Utilization with Run AI

Optimal Cluster and GPU Utilization with Run:AI

The following sections provide details on the Run:AI installation, test scenarios, and results performed in this validation.

We validated the operation and performance of this system by using industry standard benchmark tools, including TensorFlow benchmarks. The ImageNet dataset was used to train ResNet-50, which is a famous Convolutional Neural Network (CNN) DL model for image classification. ResNet-50 delivers an accurate training result with a faster processing time, which enabled us to drive a sufficient demand on the storage.

Next: Run AI Installation.

Run:AI Installation

To install Run:AI, complete the following steps:

1. Install the Kubernetes cluster using DeepOps and configure the NetApp default storage class.
2. Prepare GPU nodes:
   a. Verify that NVIDIA drivers are installed on GPU nodes.
   b. Verify that `nvidia-docker` is installed and configured as the default docker runtime.
3. Install Run:AI:
   a. Log into the Run:AI Admin UI to create the cluster.
   b. Download the created `runai-operator-<clustername>.yaml` file.
   c. Apply the operator configuration to the Kubernetes cluster.

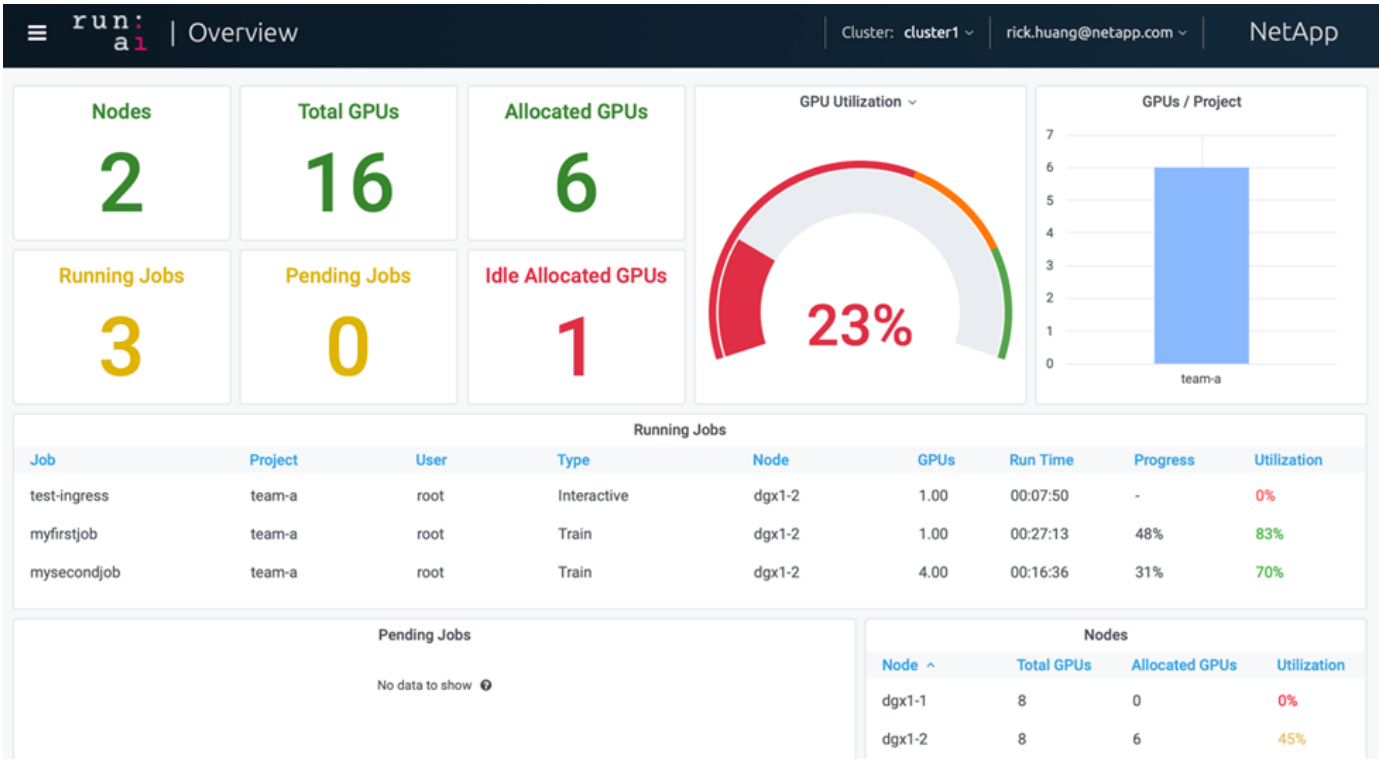
```
kubectl apply -f runai-operator-<clustername>.yaml
```

4. Verify the installation:
   a. Go to https://app.run.ai/.
   b. Go to the Overview dashboard.
   c. Verify that the number of GPUs on the top right reflects the expected number of GPUs and the GPU nodes are all in the list of servers.For more information about Run:AI deployment, see installing Run:AI on an on-premise Kubernetes cluster and installing the Run:AI CLI.

Next: Run AI Dashboards and Views

Run:AI Dashboards and Views

After installing Run:AI on your Kubernetes cluster and configuring the containers correctly, you see the following dashboards and views on https://app.run.ai in your browser, as shown in the following figure.



There are 16 total GPUs in the cluster provided by two DGX-1 nodes. You can see the number of nodes, the total available GPUs, the allocated GPUs that are assigned with workloads, the total number of running jobs, pending jobs, and idle allocated GPUs. On the right side, the bar diagram shows GPUs per Project, which summarizes how different teams are using the cluster resource. In the middle is the list of currently running jobs with job details, including job name, project, user, job type, the node each job is running on, the number of GPU(s) allocated for that job, the current run time of the job, job progress in percentage, and the GPU utilization for that job. Note that the cluster is under-utilized (GPU utilization at 23%) because there are only three running jobs submitted by a single team (`team-a`).

In the following section, we show how to create multiple teams in the Projects tab and allocate GPUs for each team to maximize cluster usage and manage resources when there are many users per cluster. The test scenarios mimic enterprise environments in which memory and GPU resources are shared among training, inferencing, and interactive workloads.

Next: Creating Projects for Data Science Teams and Allocating GPUs

Creating Projects for Data Science Teams and Allocating GPUs

Researchers can submit workloads through the Run:AI CLI, Kubeflow, or similar processes. To streamline resource allocation and create prioritization, Run:AI introduces the concept of Projects. Projects are quota entities that associate a project name with GPU allocation and preferences. It is a simple and convenient way to manage multiple data science teams.

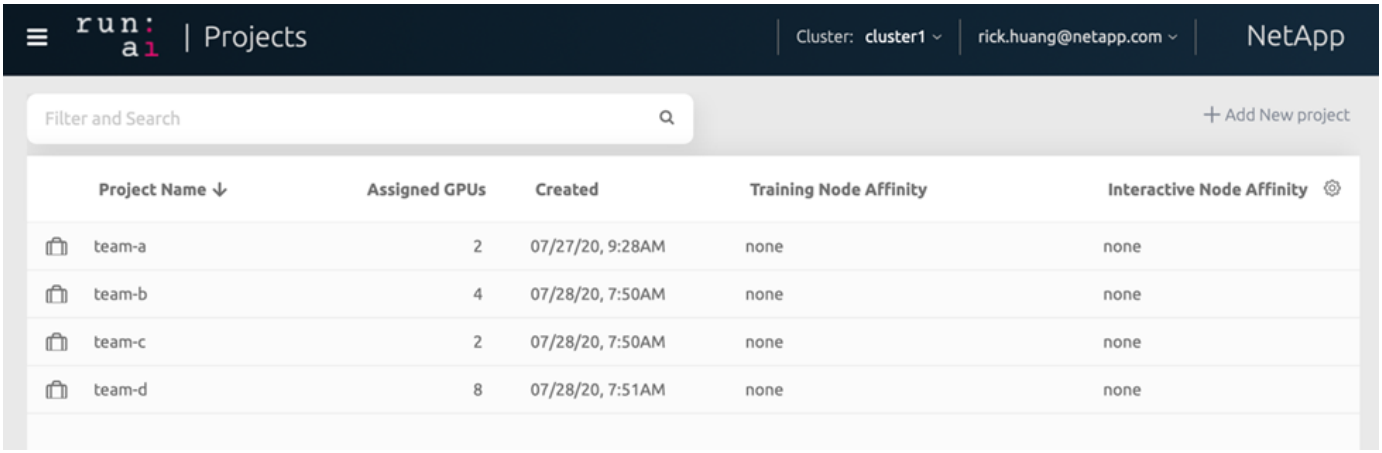
A researcher submitting a workload must associate a project with a workload request. The Run:AI scheduler

compares the request against the current allocations and the project and determines whether the workload can be allocated resources or whether it should remain in a pending state.

As a system administrator, you can set the following parameters in the Run:AI Projects tab:

- **Model projects.** Set a project per user, set a project per team of users, and set a project per a real organizational project.
- **Project quotas.** Each project is associated with a quota of GPUs that can be allocated for this project at the same time. This is a guaranteed quota in the sense that researchers using this project are guaranteed to get this number of GPUs no matter what the status in the cluster is. As a rule, the sum of the project allocation should be equal to the number of GPUs in the cluster. Beyond that, a user of this project can receive an over-quota. As long as GPUs are unused, a researcher using this project can get more GPUs. We demonstrate over-quota testing scenarios and fairness considerations in Achieving High Cluster Utilization with Over-Quota GPU Allocation, Basic Resource Allocation Fairness, and Over-Quota Fairness.
- Create a new project, update an existing project, and delete an existing project.
- **Limit jobs to run on specific node groups**. You can assign specific projects to run only on specific nodes. This is useful when the project team needs specialized hardware, for example, with enough memory. Alternatively, a project team might be the owner of specific hardware that was acquired with a specialized budget, or when you might need to direct build or interactive workloads to work on weaker hardware and direct longer training or unattended workloads to faster nodes. For commands to group nodes and set affinity for a specific project, see the Run:AI Documentation.
- **Limit the duration of interactive jobs**. Researchers frequently forget to close interactive jobs. This might lead to a waste of resources. Some organizations prefer to limit the duration of interactive jobs and close them automatically.

The following figure shows the Projects view with four teams created. Each team is assigned a different number of GPUs to account for different workloads, with the total number of GPUs equal to that of the total available GPUs in a cluster consisting of two DGX-1s.



Next: Submitting Jobs in Run AI CLI

Submitting Jobs in Run:AI CLI

This section provides the detail on basic Run:AI commands that you can use to run any Kubernetes job. It is divided into three parts according to workload type. AI/ML/DL workloads can be divided into two generic types:

- **Unattended training sessions**. With these types of workloads, the data scientist prepares a self-running workload and sends it for execution. During the execution, the customer can examine the results. This type of workload is often used in production or when model development is at a stage where no human

intervention is required.

- **Interactive build sessions**. With these types of workloads, the data scientist opens an interactive session with Bash, Jupyter Notebook, remote PyCharm, or similar IDEs and accesses GPU resources directly. We include a third scenario for running interactive workloads with connected ports to reveal an internal port to the container user..

Unattended Training Workloads

After setting up projects and allocating GPU(s), you can run any Kubernetes workload using the following command at the command line:

```
$ runai project set team-a runai submit hyper1 -i gcr.io/run-ai-
demo/quickstart -g 1
```

This command starts an unattended training job for team-a with an allocation of a single GPU. The job is based on a sample docker image, `gcr.io/run-ai-demo/quickstart`. We named the job `hyper1`. You can then monitor the job's progress by running the following command:

```
$ runai list
```

The following figure shows the result of the `runai list` command. Typical statuses you might see include the following:

- `ContainerCreating`. The docker container is being downloaded from the cloud repository.
- `Pending`. The job is waiting to be scheduled.
- `Running`. The job is running.

To get an additional status on your job, run the following command:

```
$ runai get hyper1
```

To view the logs of the job, run the `runai logs <job-name>` command:

```
$ runai logs hyper1
```

In this example, you should see the log of a running DL session, including the current training epoch, ETA, loss function value, accuracy, and time elapsed for each step.

You can view the cluster status on the Run:AI UI at https://app.run.ai/. Under Dashboards > Overview, you can monitor GPU utilization.

To stop this workload, run the following command:

```
$ runai delte hyper1
```

This command stops the training workload. You can verify this action by running `runai list` again. For more detail, see launching unattended training workloads.

Interactive Build Workloads

After setting up projects and allocating GPU(s) you can run an interactive build workload using the following command at the command line:

```
$ runai submit build1 -i python -g 1 --interactive --command sleep --args
infinity
```

The job is based on a sample docker image python. We named the job build1.

The `-- interactive` flag means that the job does not have a start or end. It is the researcher's responsibility to close the job. The administrator can define a time limit for interactive jobs after which they are terminated by the system.

The `--g 1` flag allocates a single GPU to this job. The command and argument provided is `--command sleep—args infinity`. You must provide a command, or the container starts and then exits immediately.

The following commands work similarly to the commands described in Unattended Training Workloads:

- `runai list`: Shows the name, status, age, node, image, project, user, and GPUs for jobs.
- `runai get build1`: Displays additional status on the job build1.
- `runai delete build1`: Stops the interactive workload build1.To get a bash shell to the container, the following command:

```
$ runai bash build1
```

This provides a direct shell into the computer. Data scientists can then develop or finetune their models within the container.

You can view the cluster status on the Run:AI UI at https://app.run.ai. For more detail, see starting and using interactive build workloads.

Interactive Workloads with Connected Ports

As an extension of interactive build workloads, you can reveal internal ports to the container user when starting a container with the Run:AI CLI. This is useful for cloud environments, working with Jupyter Notebooks, or connecting to other microservices. Ingress allows access to Kubernetes services from outside the Kubernetes cluster. You can configure access by creating a collection of rules that define which inbound connections reach which services.

For better management of external access to the services in a cluster, we suggest that cluster administrators install Ingress and configure LoadBalancer.

To use Ingress as a service type, run the following command to set the method type and the ports when submitting your workload:

```
$ runai submit test-ingress -i jupyter/base-notebook -g 1 \
  --interactive --service-type=ingress --port 8888 \
  --args="--NotebookApp.base_url=test-ingress" --command=start-notebook.sh
```

After the container starts successfully, execute `runai list` to see the `SERVICE URL(S)` with which to access the Jupyter Notebook. The URL is composed of the ingress endpoint, the job name, and the port. For example, see https://10.255.174.13/test-ingress-8888.

For more details, see launching an interactive build workload with connected ports.

Next: Achieving High Cluster Utilization

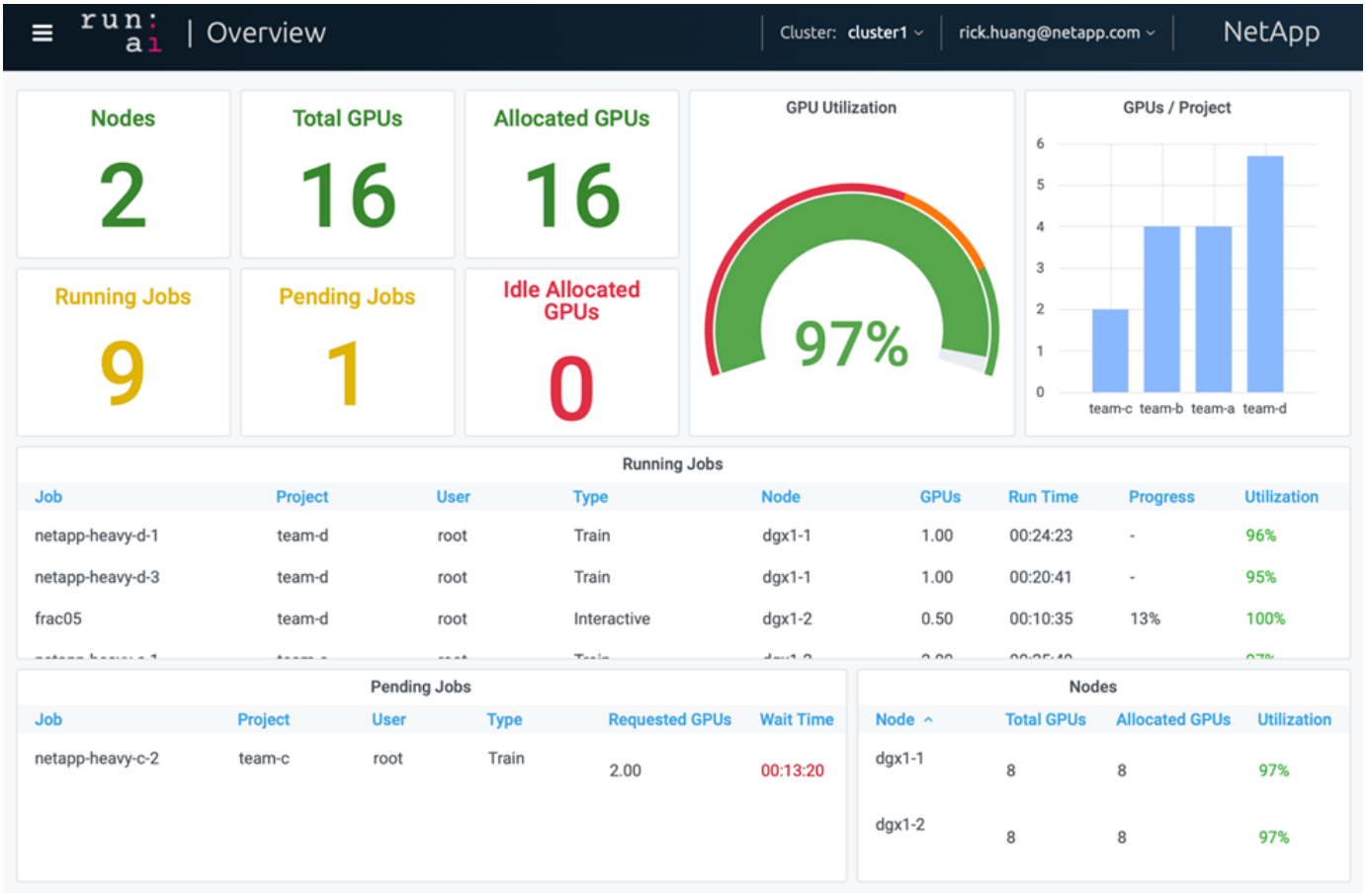
Achieving High Cluster Utilization

In this section, we emulate a realistic scenario in which four data science teams each submit their own workloads to demonstrate the Run:AI orchestration solution that achieves high cluster utilization while maintaining prioritization and balancing GPU resources. We start by using the ResNet-50 benchmark described in the section ResNet-50 with ImageNet Dataset Benchmark Summary:

```
$ runai submit netapp1 -i netapp/tensorflow-tf1-py3:20.01.0 --local-image
--large-shm  -v /mnt:/mnt -v /tmp:/tmp --command python --args
"/netapp/scripts/run.py" --args "--
dataset_dir=/mnt/mount_0/dataset/imagenet/imagenet_original/" --args "--
num_mounts=2"  --args "--dgx_version=dgx1" --args "--num_devices=1" -g 1
```

We ran the same ResNet-50 benchmark as in NVA-1121. We used the flag `--local-image` for containers not residing in the public docker repository. We mounted the directories `/mnt` and `/tmp` on the host DGX-1 node to `/mnt` and `/tmp` to the container, respectively. The dataset is at NetApp AFFA800 with the `dataset_dir` argument pointing to the directory. Both `--num_devices=1` and `-g 1` mean that we allocate one GPU for this job. The former is an argument for the `run.py` script, while the latter is a flag for the `runai submit` command.

The following figure shows a system overview dashboard with 97% GPU utilization and all sixteen available GPUs allocated. You can easily see how many GPUs are allocated for each team in the GPUs/Project bar chart. The Running Jobs pane shows the current running job names, project, user, type, node, GPUs consumed, run time, progress, and utilization details. A list of workloads in queue with their wait time is shown in Pending Jobs. Finally, the Nodes box offers GPU numbers and utilization for individual DGX-1 nodes in the cluster.



Next: Fractional GPU Allocation for Less Demanding or Interactive Workloads

Fractional GPU Allocation for Less Demanding or Interactive Workloads

When researchers and developers are working on their models, whether in the development, hyperparameter tuning, or debugging stages, such workloads usually require fewer computational resources. It is therefore more efficient to provision fractional GPU and memory such that the same GPU can simultaneously be allocated to other workloads. Run:AI's orchestration solution provides a fractional GPU sharing system for containerized workloads on Kubernetes. The system supports workloads running CUDA programs and is especially suited for lightweight AI tasks such as inference and model building. The fractional GPU system transparently gives data science and AI engineering teams the ability to run multiple workloads simultaneously on a single GPU. This enables companies to run more workloads, such as computer vision, voice recognition, and natural language processing on the same hardware, thus lowering costs.

Run:AI's fractional GPU system effectively creates virtualized logical GPUs with their own memory and computing space that containers can use and access as if they were self-contained processors. This enables several workloads to run in containers side-by-side on the same GPU without interfering with each other. The solution is transparent, simple, and portable and it requires no changes to the containers themselves.

A typical usecase could see two to eight jobs running on the same GPU, meaning that you could do eight times the work with the same hardware.

For the job `frac05` belonging to project `team-d` in the following figure, we can see that the number of GPUs allocated was 0.50. This is further verified by the `nvidia-smi` command, which shows that the GPU memory available to the container was 16,255MB: half of the 32GB per V100 GPU in the DGX-1 node.

```
root@run-deploy:~# runai bash frac05 -p team-d
root@frac05-0:/workload# nvidia-smi
Tue Jul 28 15:17:03 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.05    Driver Version: 450.51.05    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   57C    P0   240W / 300W |  15525MiB / 16255MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       156      C   python3                         15525MiB |
+-----------------------------------------------------------------------------+
```

Next: Achieving High Cluster Utilization with Over-Quota GPU Allocation

Achieving High Cluster Utilization with Over-Quota GPU Allocation

In this section and in the sections Basic Resource Allocation Fairness, and Over-Quota Fairness, we have devised advanced testing scenarios to demonstrate the Run:AI orchestration capabilities for complex workload management, automatic preemptive scheduling, and over-quota GPU provisioning. We did this to achieve high cluster-resource usage and optimize enterprise-level data science team productivity in an ONTAP AI environment.

For these three sections, set the following projects and quotas:

Project	Quota
team-a	4
team-b	2
team-c	2
team-d	8

In addition, we use the following containers for these three sections:

- Jupyter Notebook: `jupyter/base-notebook`
- Run:AI quickstart: `gcr.io/run-ai-demo/quickstart`

We set the following goals for this test scenario:

- Show the simplicity of resource provisioning and how resources are abstracted from users
- Show how users can easily provision fractions of a GPU and integer number of GPUs
- Show how the system eliminates compute bottlenecks by allowing teams or users to go over their resource quota if there are free GPUs in the cluster
- Show how data pipeline bottlenecks are eliminated by using the NetApp solution when running compute-intensive jobs, such as the NetApp container
- Show how multiple types of containers are running using the system
  - Jupyter Notebook
  - Run:AI container
- Show high utilization when the cluster is full

For details on the actual command sequence executed during the testing, see Testing Details for Section 4.8.

When all 13 workloads are submitted, you can see a list of container names and GPUs allocated, as shown in the following figure. We have seven training and six interactive jobs, simulating four data science teams, each with their own models running or in development. For interactive jobs, individual developers are using Jupyter Notebooks to write or debug their code. Thus, it is suitable to provision GPU fractions without using too many cluster resources.

```
root@run-deploy:~# runai list -A
NAME           STATUS   AGE  NODE    IMAGE                          TYPE         PROJECT  USER  GPUs  CREATED BY CLI  SERVICE URL(S)
b-4-gg         Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train        team-b   root  2     true
c-5-g          Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train        team-c   root  1     true
c-4-gg         Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train        team-c   root  2     true
b-3-g          Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train        team-b   root  1     true
c-3-g02        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive  team-c   root  0.2   true
d-1-gggg       Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train        team-d   root  4     true
c-2-g03        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive  team-c   root  0.3   true
c-1-g05        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive  team-c   root  0.5   true
a-2-gg         Running  3m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train        team-a   root  2     true
b-2-g04        Running  3m   dgx1-2  gcr.io/run-ai-demo/quickstart  Interactive  team-b   root  0.4   true
a-1-g          Running  3m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train        team-a   root  1     true
b-1-g06        Running  3m   dgx1-2  gcr.io/run-ai-demo/quickstart  Interactive  team-b   root  0.6   true
a-1-1-jupyter  Running  3m   dgx1-1  jupyter/base-notebook          Interactive  team-a   root  1     true            http://10.61.218.134/a-1-1-jupyter,
https://10.61.218.134/a-1-1-jupyter
```

The results of this testing scenario show the following:

- The cluster should be full: 16/16 GPUs are used.
- High cluster utilization.
- More experiments than GPUs due to fractional allocation.
- `team-d` is not using all their quota; therefore, `team-b` and `team-c` can use additional GPUs for their experiments, leading to faster time to innovation.

Next: Basic Resource Allocation Fairness

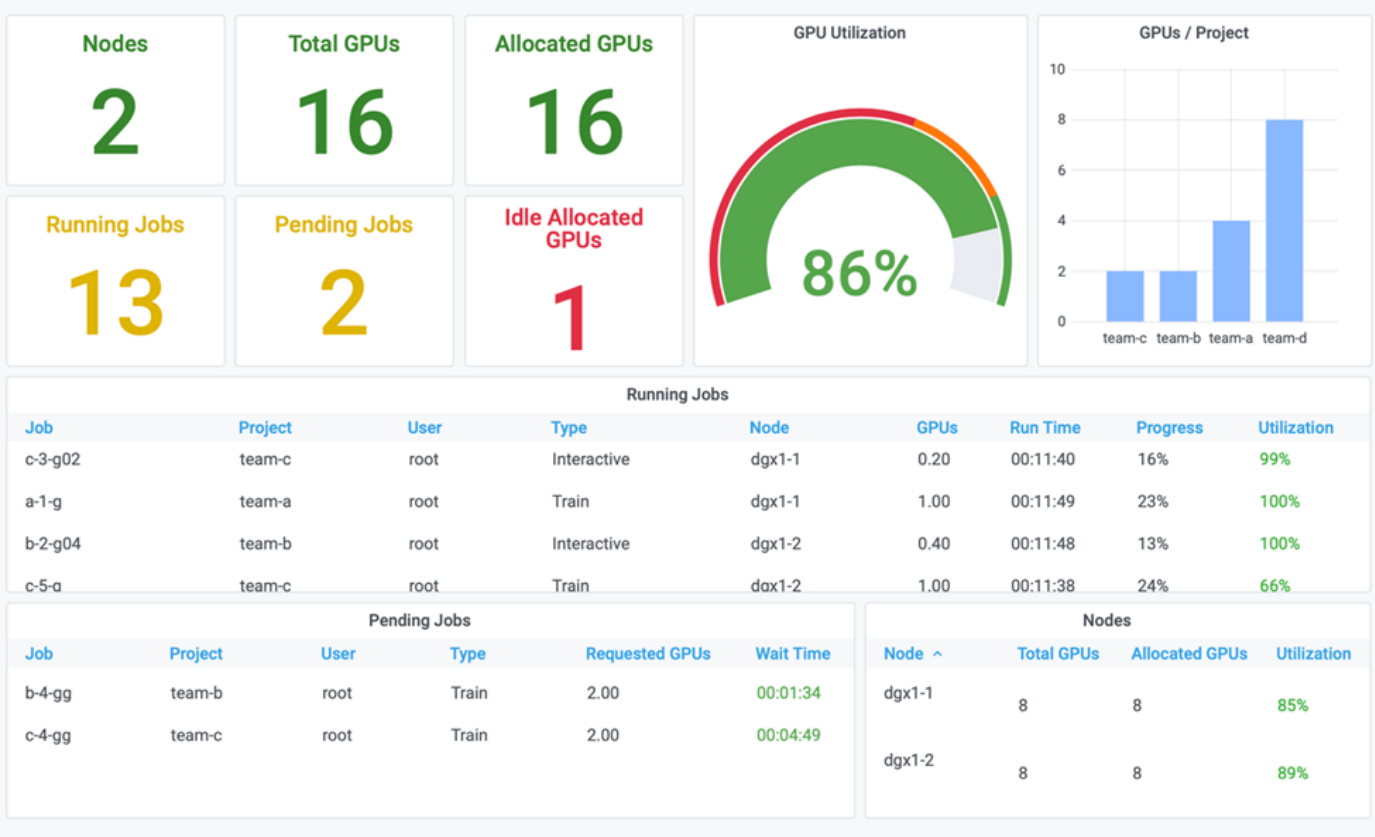
Basic Resource Allocation Fairness

In this section, we show that, when `team-d` asks for more GPUs (they are under their quota), the system pauses the workloads of `team-b` and `team-c` and moves them into a pending state in a fair-share manner.

For details including job submissions, container images used, and command sequences executed, see the section Testing Details for Section 4.9.

The following figure shows the resulting cluster utilization, GPUs allocated per team, and pending jobs due to automatic load balancing and preemptive scheduling. We can observe that when the total number of GPUs

requested by all team workloads exceeds the total available GPUs in the cluster, Run:AI's internal fairness algorithm pauses one job each for `team-b` and `team-c` because they have met their project quota. This provides overall high cluster utilization while data science teams still work under resource constraints set by an administrator.



The results of this testing scenario demonstrate the following:

- **Automatic load balancing.** The system automatically balances the quota of the GPUs, such that each team is now using their quota. The workloads that were paused belong to teams that were over their quota.
- **Fair share pause.** The system chooses to stop the workload of one team that was over their quota and then stop the workload of the other team. Run:AI has internal fairness algorithms.

Next: Over-Quota Fairness

Over-Quota Fairness

In this section, we expand the scenario in which multiple teams submit workloads and exceed their quota. In this way, we demonstrate how Run:AI's fairness algorithm allocates cluster resources according to the ratio of preset quotas.

Goals for this test scenario:

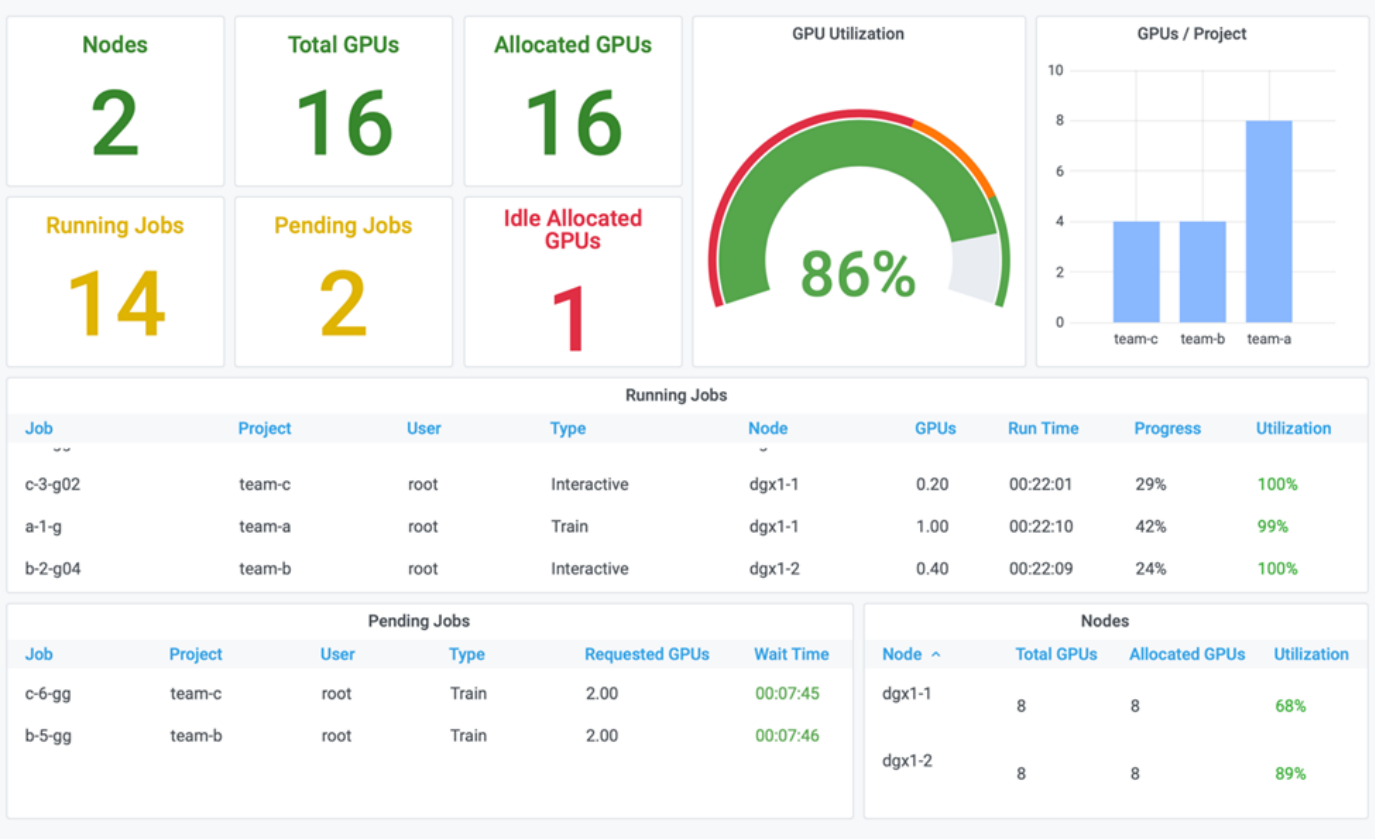
- Show queuing mechanism when multiple teams are requesting GPUs over their quota.
- Show how the system distributes a fair share of the cluster between multiple teams that are over their quota according to the ratio between their quotas, so that the team with the larger quota gets a larger share of the spare capacity.

At the end of Basic Resource Allocation Fairness, there are two workloads queued: one for `team-b` and one

for `team-c`. In this section, we queue additional workloads.

For details including job submissions, container images used, and command sequences executed, see Testing Details for section 4.10.

When all jobs are submitted according to the section Testing Details for section 4.10, the system dashboard shows that `team-a`, `team-b`, and `team-c` all have more GPUs than their preset quota. `team-a` occupies four more GPUs than its preset soft quota (four), whereas `team-b` and `team-c` each occupy two more GPUs than their soft quota (two). The ratio of over-quota GPUs allocated is equal to that of their preset quota. This is because the system used the preset quota as a reference of priority and provisioned accordingly when multiple teams request more GPUs, exceeding their quota. Such automatic load balancing provides fairness and prioritization when enterprise data science teams are actively engaged in AI model development and production.



The results of this testing scenario show the following:

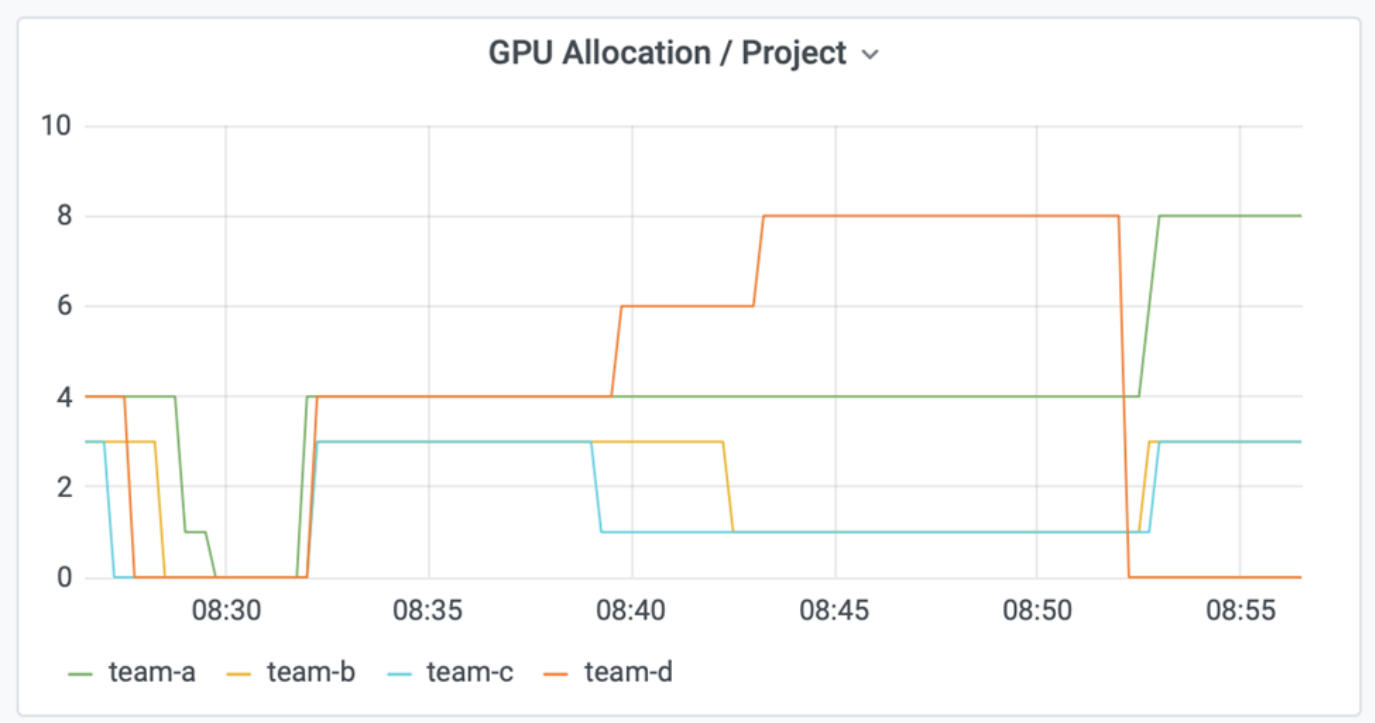
- The system starts to de-queue the workloads of other teams.
- The order of the dequeuing is decided according to fairness algorithms, such that `team-b` and `team-c` get the same amount of over-quota GPUs (since they have a similar quota), and `team-a` gets a double amount of GPUs since their quota is two times higher than the quota of `team-b` and `team-c`.
- All the allocation is done automatically.

Therefore, the system should stabilize on the following states:

Project	GPUs allocated	Comment
team-a	8/4	Four GPUs over the quota. Empty queue.

Project	GPUs allocated	Comment
team-b	4/2	Two GPUs over the quota. One workload queued.
team-c	4/2	Two GPUs over the quota. One workload queued.
team-d	0/8	Not using GPUs at all, no queued workloads.

The following figure shows the GPU allocation per project over time in the Run:AI Analytics dashboard for the sections Achieving High Cluster Utilization with Over-Quota GPU Allocation, Basic Resource Allocation Fairness, and Over-Quota Fairness. Each line in the figure indicates the number of GPUs provisioned for a given data science team at any time. We can see that the system dynamically allocates GPUs according to workloads submitted. This allows teams to go over quota when there are available GPUs in the cluster, and then preempt jobs according to fairness, before finally reaching a stable state for all four teams.



Next: Saving Data to a Trident-Provisioned PersistentVolume

Saving Data to a Trident-Provisioned PersistentVolume

NetApp Trident is a fully supported open source project designed to help you meet the sophisticated persistence demands of your containerized applications. You can read and write data to a Trident-provisioned Kubernetes PersistentVolume (PV) with the added benefit of data tiering, encryption, NetApp Snapshot technology, compliance, and high performance offered by NetApp ONTAP data management software.

Reusing PVCs in an Existing Namespace

For larger AI projects, it might be more efficient for different containers to read and write data to the same Kubernetes PV. To reuse a Kubernetes Persistent Volume Claim (PVC), the user must have already created a PVC. See the NetApp Trident documentation for details on creating a PVC. Here is an example of reusing an existing PVC:

```
$ runai submit pvc-test -p team-a --pvc test:/tmp/pvc1mount -i gcr.io/run-ai-demo/quickstart -g 1
```

Run the following command to see the status of job `pvc-test` for project `team-a`:

```
$ runai get pvc-test -p team-a
```

You should see the PV /tmp/pvc1mount mounted to `team-a` job `pvc-test`. In this way, multiple containers can read from the same volume, which is useful when there are multiple competing models in development or in production. Data scientists can build an ensemble of models and then combine prediction results by majority voting or other techniques.

Use the following to access the container shell:

```
$ runai bash pvc-test -p team-a
```

You can then check the mounted volume and access your data within the container.

This capability of reusing PVCs works with NetApp FlexVol volumes and NetApp ONTAP FlexGroup volumes, enabling data engineers more flexible and robust data management options to leverage your data fabric powered by NetApp.

Next: Conclusion

Conclusion

NetApp and Run:AI have partnered in this technical report to demonstrate the unique capabilities of the NetApp ONTAP AI solution together with the Run:AI Platform for simplifying orchestration of AI workloads. The preceding steps provide a reference architecture to streamline the process of data pipelines and workload orchestration for deep learning. Customers looking to implement these solutions are encouraged to reach out to NetApp and Run:AI for more information.

Next: Testing Details for Section 4.8

Testing Details for Section 4.8

This section contains the testing details for the section Achieving High Cluster Utilization with Over-Quota GPU Allocation.

Submit jobs in the following order:

Project	Image	# GPUs	Total	Comment
team-a	Jupyter	1	1/4	–
team-a	NetApp	1	2/4	–
team-a	Run:AI	2	4/4	Using all their quota
team-b	Run:AI	0.6	0.6/2	Fractional GPU

Project	Image	# GPUs	Total	Comment
team-b	Run:AI	0.4	1/2	Fractional GPU
team-b	NetApp	1	2/2	–
team-b	NetApp	2	4/2	Two over quota
team-c	Run:AI	0.5	0.5/2	Fractional GPU
team-c	Run:AI	0.3	0.8/2	Fractional GPU
team-c	Run:AI	0.2	1/2	Fractional GPU
team-c	NetApp	2	3/2	One over quota
team-c	NetApp	1	4/2	Two over quota
team-d	NetApp	4	4/8	Using half of their quota

Command structure:

```
$ runai submit <job-name> -p <project-name> -g <#GPUs> -i <image-name>
```

Actual command sequence used in testing:

```
$ runai submit a-1-1-jupyter -i jupyter/base-notebook -g 1 \
  --interactive --service-type=ingress --port 8888 \
  --args="--NotebookApp.base_url=team-a-test-ingress" --command=start
-notebook.sh -p team-a
$ runai submit a-1-g -i gcr.io/run-ai-demo/quickstart -g 1 -p team-a
$ runai submit a-2-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-a
$ runai submit b-1-g06 -i gcr.io/run-ai-demo/quickstart -g 0.6
--interactive -p team-b
$ runai submit b-2-g04 -i gcr.io/run-ai-demo/quickstart -g 0.4
--interactive -p team-b
$ runai submit b-3-g -i gcr.io/run-ai-demo/quickstart -g 1 -p team-b
$ runai submit b-4-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-b
$ runai submit c-1-g05 -i gcr.io/run-ai-demo/quickstart -g 0.5
--interactive -p team-c
$ runai submit c-2-g03 -i gcr.io/run-ai-demo/quickstart -g 0.3
--interactive -p team-c
$ runai submit c-3-g02 -i gcr.io/run-ai-demo/quickstart -g 0.2
--interactive -p team-c
$ runai submit c-4-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-c
$ runai submit c-5-g -i gcr.io/run-ai-demo/quickstart -g 1 -p team-c
$ runai submit d-1-gggg -i gcr.io/run-ai-demo/quickstart -g 4 -p team-d
```

At this point, you should have the following states:

Project	GPUs Allocated	Workloads Queued
team-a	4/4 (soft quota/actual allocation)	None
team-b	4/2	None
team-c	4/2	None
team-d	4/8	None

See the section Achieving High Cluster Utilization with Over-uota GPU Allocation for discussions on the proceeding testing scenario.

Next: Testing Details for Section 4.9

Testing Details for Section 4.9

This section contains testing details for the section Basic Resource Allocation Fairness.

Submit jobs in the following order:

Project	# GPUs	Total	Comment
team-d	2	6/8	Team-b/c workload pauses and moves to `pending`.
team-d	2	8/8	Other team (b/c) workloads pause and move to `pending`.

See the following executed command sequence:

```
$ runai submit d-2-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-d$
runai submit d-3-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-d
```

At this point, you should have the following states:

Project	GPUs Allocated	Workloads Queued
team-a	4/4	None
team-b	2/2	None
team-c	2/2	None
team-d	8/8	None

See the section Basic Resource Allocation Fairness for a discussion on the proceeding testing scenario.

Next: Testing Details for Section 4.10

Testing Details for Section 4.10

This section contains testing details for the section Over-Quota Fairness.

Submit jobs in the following order for `team-a`, `team-b`, and `team-c`:

Project	# GPUs	Total	Comment
team-a	2	4/4	1 workload queued
team-a	2	4/4	2 workloads queued
team-b	2	2/2	2 workloads queued
team-c	2	2/2	2 workloads queued

See the following executed command sequence:

```
$ runai submit a-3-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-a$
runai submit a-4-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-a$ runai
submit b-5-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-b$ runai
submit c-6-gg -i gcr.io/run-ai-demo/quickstart -g 2 -p team-c
```

At this point, you should have the following states:

Project	GPUs Allocated	Workloads Queued
team-a	4/4	Two workloads asking for GPUs two each
team-b	2/2	Two workloads asking for two GPUs each
team-c	2/2	Two workloads asking for two GPUs each
team-d	8/8	None

Next, delete all the workloads for `team-d`:

```
$ runai delete -p team-d d-1-gggg d-2-gg d-3-gg
```

See the section Over-Quota Fairness, for discussions on the proceeding testing scenario.

Next: Where to Find Additional Information

Where to Find Additional Information

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

- NVIDIA DGX Systems
  - NVIDIA DGX-1 System
    https://www.nvidia.com/en-us/data-center/dgx-1/
  - NVIDIA V100 Tensor Core GPU
    https://www.nvidia.com/en-us/data-center/tesla-v100/

- NVIDIA NGC
  https://www.nvidia.com/en-us/gpu-cloud/

- Run:AI container orchestration solution
  - Run:AI product introduction
    https://docs.run.ai/home/components/
  - Run:AI installation documentation
    https://docs.run.ai/Administrator/Cluster-Setup/Installing-Run-AI-on-an-on-premise-Kubernetes-Cluster/
    https://docs.run.ai/Administrator/Researcher-Setup/Installing-the-Run-AI-Command-Line-Interface/
  - Submitting jobs in Run:AI CLI
    https://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Launch-Unattended-Training-Workloads-/
    https://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Start-and-Use-Interactive-Build-Workloads-/
  - Allocating GPU fractions in Run:AI CLI
    https://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Using-GPU-Fractions/
- NetApp AI Control Plane
  - Technical report
    https://www.netapp.com/us/media/tr-4798.pdf
  - Short-form demo
    https://youtu.be/gfr_sO27Rvo
  - GitHub repository
    https://github.com/NetApp/kubeflow_jupyter_pipeline
- NetApp AFF systems
  - NetApp AFF A-Series Datasheet
    https://www.netapp.com/us/media/ds-3582.pdf
  - NetApp Flash Advantage for All Flash FAS
    https://www.netapp.com/us/media/ds-3733.pdf
  - ONTAP 9 Information Library
    http://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62286
  - NetApp ONTAP FlexGroup Volumes technical report
    https://www.netapp.com/us/media/tr-4557.pdf
- NetApp ONTAP AI
  - ONTAP AI with DGX-1 and Cisco Networking Design Guide
    https://www.netapp.com/us/media/nva-1121-design.pdf
  - ONTAP AI with DGX-1 and Cisco Networking Deployment Guide
    https://www.netapp.com/us/media/nva-1121-deploy.pdf
  - ONTAP AI with DGX-1 and Mellanox Networking Design Guide
    http://www.netapp.com/us/media/nva-1138-design.pdf
  - ONTAP AI with DGX-2 Design Guide
    https://www.netapp.com/us/media/nva-1135-design.pdf