



Distributed Azure training - Lane detection

NetApp Solutions

NetApp
March 09, 2022

Table of Contents

TR-4896: Distributed training in Azure: Lane detection - Solution design 1

 Solution overview 1

 Lane detection – Distributed training with RUN:AI 3

 Conclusion 28

 Additional information 28

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 Snapshot™ 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.



Name	Mode	OS type	Node count	Node size
<input type="checkbox"/> agentpool	System	Linux	3	Standard_DS2_v2
<input type="checkbox"/> gpupool	User	Linux	3	Standard_NC6s_v



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 xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx
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.

Add subnet ✕

Name * ✓

Subnet address range * ⓘ
10.0.0.0 - 10.0.0.255 (251 + 5 Azure reserved addresses)

☐ Add IPv6 address space ⓘ

NAT gateway ⓘ ▼

Network security group ▼

Route table ▼

SERVICE ENDPOINTS

Create service endpoint policies to allow traffic to specific azure resources from your virtual network over service endpoints. [Learn more](#)

Services ⓘ ▼

SUBNET DELEGATION

Delegate subnet to a service ⓘ ▼

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-xxxxxxxxxxxxx",
  "displayName": "netapptrident",
  "name": "http://netapptrident",
  "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx"
}
```

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 <code>az ad sp</code> in the previous step)
clientID	Your <code>appId</code> (from the output of <code>az ad sp</code> in the previous step)
clientSecret	Your password (from the output of <code>az ad sp</code> 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 `azurenetaappfiles` 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 azurenetaappfiles
```

The output should look like the following example:

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
azurenetaappfiles	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`.

New Project

Basics

Project Name ⓘ
lane-detection

Assigned GPUs
3

Over-quota for project
☒ Allow over-quota

Save **Cancel**

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 azurenetaappfiles. 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-34613862-vmss00000a  muneer7589/download-tusimple:1.0  Train  lane-detection  veronmartina  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:

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



The screenshot shows the 'Jobs' page in the RUN: AI interface. The table lists various jobs, including 'download-tusimple-data' and 'build1'. The first job is in a 'Running' state, while the others are 'Deleted'.

Job Name	Status	User	Project	Total Run Time	Creation Time	Type	GPU Utilization	Used CPU
download-tusimple-data	Running	verronma...	lane-detection	00:07:11	03/03/21, 2:51PM	Train	-	0.00
build1	Deleted	root	lane-detection	00:01:56	03/01/21, 10:18...	Interactive	-	-
download-tusimple-data	Deleted	root	lane-detection	-	03/01/21, 9:58AM	Train	-	-
download-tusimple-data	Deleted	root	lane-detection	-	03/01/21, 10:03...	Train	-	-
download-tusimple-data	Deleted	root	lane-detection	00:02:55	03/01/21, 10:24...	Train	-	-
download-tusimple-data	Deleted	root	lane-detection	-	03/01/21, 10:30...	Train	-	-
download-tusimple-data	Deleted	root	lane-detection	00:13:17	03/01/21, 11:41...	Train	-	-
download-tusimple-data-1	Deleted	verronma...	lane-detection	-	02/26/21, 5:30PM	Train	-	-

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   azurenetaappfiles   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	SERVICE URL(S)	STATUS	AGE	NODE	IMAGE	TYPE	PROJECT	USER	GPUs Allocated (Requested)	PODs
download-tusimple-data		Succeeded	1d		muneer7589/download-tusimple:1.0	Train	lane-detection	verronmartina	0 (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/kubect1 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/kubect1 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-snapshot-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.

NFSv3	10.28.254.4:/norootfor	Standard	pool0	...
NFSv4.1	NAS-735a.docs.lab:/fo	Premium		...
NFSv4.1	NAS-735a.docs.lab:/krt	Premium		...
NFSv3	10.28.254.4:/moveme0	Premium		...
NFSv3	10.28.254.4:/placeholder	Premium		...

Resize
 Edit
 Change pool
 Delete

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 myacname
-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/>

Copyright Information

Copyright © 2022 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

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

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

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

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

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

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