# Table of Contents

Artificial Intelligence  ........................................................................................................ 1
  TR-4834: NetApp and Iguazio for MLRun Pipeline .......................................................... 1
  TR-4841: Hybrid Cloud AI Operating System with Data Caching ................................ 18
  WP-7328: NetApp Conversational AI Using NVIDIA Jarvis ......................................... 31
  TR-4858: NetApp Orchestration Solution with Run:AI .................................................... 48
  NVA-1144: NetApp HCI AI Inferencing at the Edge Data Center with H615c and NVIDIA T4 .... 64
Artificial Intelligence

TR-4834: NetApp and Iguazio for MLRun Pipeline

Rick Huang, David Arnette, NetApp
Marcelo Litovsky, Iguazio

This document covers the details of the MLRun pipeline using NetApp ONTAP AI, NetApp AI Control Plane, NetApp Cloud Volumes software, and the Iguazio Data Science Platform. We used Nuclio serverless function, Kubernetes Persistent Volumes, NetApp Cloud Volumes, NetApp Snapshot copies, Grafana dashboard, and other services on the Iguazio platform to build an end-to-end data pipeline for the simulation of network failure detection. We integrated Iguazio and NetApp technologies to enable fast model deployment, data replication, and production monitoring capabilities on premises as well as in the cloud.

The work of a data scientist should be focused on the training and tuning of machine learning (ML) and artificial intelligence (AI) models. However, according to research by Google, data scientists spend ~80% of their time figuring out how to make their models work with enterprise applications and run at scale, as shown in the following image depicting model development in the AI/ML workflow.

To manage end-to-end AI/ML projects, a wider understanding of enterprise components is needed. Although DevOps have taken over the definition, integration, and deployment these types of components, machine learning operations target a similar flow that includes AI/ML projects. To get an idea of what an end-to-end AI/ML pipeline touches in the enterprise, see the following list of required components:

- Storage
- Networking
- Databases
- File systems
- Containers
- Continuous integration and continuous deployment (CI/CD) pipeline
- Development integrated development environment (IDE)
- Security
- Data access policies
- Hardware
- Cloud
- Virtualization
- Data science toolsets and libraries

In this paper, we demonstrate how the partnership between NetApp and Iguazio drastically simplifies the development of an end-to-end AI/ML pipeline. This simplification accelerates the time to market for all of your AI/ML applications.

Target Audience

The world of data science touches multiple disciplines in information technology and business.
• The data scientist needs the flexibility to use their tools and libraries of choice.
• The data engineer needs to know how the data flows and where it resides.
• A DevOps engineer needs the tools to integrate new AI/ML applications into their CI/CD pipelines.
• Business users want to have access to AI/ML applications. We describe how NetApp and Iguazio help each of these roles bring value to business with our platforms.

Solution Overview

This solution follows the lifecycle of an AI/ML application. We start with the work of data scientists to define the different steps needed to prep data and train and deploy models. We follow with the work needed to create a full pipeline with the ability to track artifacts, experiment with execution, and deploy to Kubeflow. To complete the full cycle, we integrate the pipeline with NetApp Cloud Volumes to enable data versioning, as seen in the following image.

[Error: Missing Graphic Image]

Next: Technology Overview

Technology Overview

NetApp Overview

NetApp is the data authority for the hybrid cloud. NetApp provides a full range of hybrid cloud data services that simplify management of applications and data across cloud and on-premises environments to accelerate digital transformation. Together with our partners, NetApp empowers global organizations to unleash the full potential of their data to expand customer touch points, foster greater innovation, and optimize their operations.

NetApp ONTAP AI

NetApp ONTAP AI, powered by NVIDIA DGX systems and NetApp cloud-connected all-flash storage, streamlines the flow of data reliably and speeds up analytics, training, and inference with your data fabric that spans from edge to core to cloud. It gives IT organizations an architecture that provides the following benefits:

• Eliminates design complexities
• Allows 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 offers converged infrastructure stacks incorporating NVIDIA DGX-1, a petaflop-scale AI system, and NVIDIA Mellanox high-performance Ethernet switches to unify AI workloads, simplify deployment, and accelerate ROI. We leveraged ONTAP AI with one DGX-1 and NetApp AFF A800 storage system for this technical report. The following image shows the topology of ONTAP AI with the DGX-1 system used in this validation.

[Error: Missing Graphic Image]

NetApp AI Control Plane

The NetApp AI Control Plane enables you to unleash AI and ML with a solution that offers extreme scalability, streamlined deployment, and nonstop data availability. The AI Control Plane solution integrates Kubernetes and Kubeflow with a data fabric enabled by NetApp. Kubernetes, the industry-standard container orchestration platform for cloud-native deployments, enables workload scalability and portability. Kubeflow is an open-source
machine-learning platform that simplifies management and deployment, enabling developers to do more data
science in less time. A data fabric enabled by NetApp offers uncompromising data availability and portability to
make sure that your data is accessible across the pipeline, from edge to core to cloud. This technical report
uses the NetApp AI Control Plane in an MLRun pipeline. The following image shows Kubernetes cluster
management page where you can have different endpoints for each cluster. We connected NFS Persistent
Volumes to the Kubernetes cluster, and the following images show an Persistent Volume connected to the
cluster, where NetApp Trident offers persistent storage support and data management capabilities.

Iguazio Overview

The Iguazio Data Science Platform is a fully integrated and secure data-science platform as a service (PaaS)
that simplifies development, accelerates performance, facilitates collaboration, and addresses operational
challenges. This platform incorporates the following components, and the Iguazio Data Science Platform is
presented in the following image:

- A data-science workbench that includes Jupyter Notebooks, integrated analytics engines, and Python
  packages
- Model management with experiments tracking and automated pipeline capabilities
- Managed data and ML services over a scalable Kubernetes cluster
- Nuclio, a real-time serverless functions framework
- An extremely fast and secure data layer that supports SQL, NoSQL, time-series databases, files (simple
  objects), and streaming
- Integration with third-party data sources such as NetApp, Amazon S3, HDFS, SQL databases, and
  streaming or messaging protocols
- Real-time dashboards based on Grafana

Software and Hardware Requirements

Network Configuration

The following is the network configuration requirement for setting up in the cloud:

- The Iguazio cluster and NetApp Cloud Volumes must be in the same virtual private cloud.
- The cloud manager must have access to port 6443 on the Iguazio app nodes.
- We used Amazon Web Services in this technical report. However, users have the option of deploying the
  solution in any Cloud provider. For on-premises testing in ONTAP AI with NVIDIA DGX-1, we used the
  Iguazio hosted DNS service for convenience.

Clients must be able to access dynamically created DNS domains. Customers can use their own DNS if
desired.
Hardware Requirements

You can install Iguazio on-premises in your own cluster. We have verified the solution in NetApp ONTAP AI with an NVIDIA DGX-1 system. The following table lists the hardware used to test this solution.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGX-1 systems</td>
<td>1</td>
</tr>
<tr>
<td>NetApp AFF A800 system</td>
<td>1 high-availability (HA) pair, includes 2 controllers and 48 NVMe SSDs (3.8TB or above)</td>
</tr>
<tr>
<td>Cisco Nexus 3232C network switches</td>
<td>2</td>
</tr>
</tbody>
</table>

The following table lists the software components required for on-premise testing:

<table>
<thead>
<tr>
<th>Software</th>
<th>Version or Other Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetApp ONTAP data management software</td>
<td>9.7</td>
</tr>
<tr>
<td>Cisco NX-OS switch firmware</td>
<td>7.0(3)I6(1)</td>
</tr>
<tr>
<td>NVIDIA DGX OS</td>
<td>4.4 - Ubuntu 18.04 LTS</td>
</tr>
<tr>
<td>Docker container platform</td>
<td>19.03.5</td>
</tr>
<tr>
<td>Container version</td>
<td>20.01-tf1-py2</td>
</tr>
<tr>
<td>Machine learning framework</td>
<td>TensorFlow 1.15.0</td>
</tr>
<tr>
<td>Iguazio</td>
<td>Version 2.8+</td>
</tr>
<tr>
<td>ESX Server</td>
<td>6.5</td>
</tr>
</tbody>
</table>

This solution was fully tested with Iguazio version 2.5 and NetApp Cloud Volumes ONTAP for AWS. The Iguazio cluster and NetApp software are both running on AWS.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version or Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iguazio</td>
<td>Version 2.8+</td>
</tr>
<tr>
<td>App node</td>
<td>M5.4xlarge</td>
</tr>
<tr>
<td>Data node</td>
<td>I3.4xlarge</td>
</tr>
</tbody>
</table>

Next: Network Device Failure Prediction Use Case Summary

Network Device Failure Prediction Use Case Summary

This use case is based on an Iguazio customer in the telecommunications space in Asia. With 100K enterprise customers and 125k network outage events per year, there was a critical need to predict and take proactive action to prevent network failures from affecting customers. This solution provided them with the following benefits:

- Predictive analytics for network failures
- Integration with a ticketing system
- Taking proactive action to prevent network failures

As a result of this implementation of Iguazio, 60% of failures were proactively prevented.
Setup Overview

Iguazio Installation

Iguazio can be installed on-premises or on a cloud provider. Provisioning can be done as a service and managed by Iguazio or by the customer. In both cases, Iguazio provides a deployment application (Provazio) to deploy and manage clusters.

For on-premises installation, please refer to NVA-1121 for compute, network, and storage setup. On-premises deployment of Iguazio is provided by Iguazio without additional cost to the customer. See this page for DNS and SMTP server configurations. The Provazio installation page is shown as follows.

Next: Configuring Kubernetes Cluster

Configuring Kubernetes Cluster

This section is divided into two parts for cloud and on-premises deployment respectively.

Cloud Deployment Kubernetes Configuration

Through NetApp Cloud Manager, you can define the connection to the Iguazio Kubernetes cluster. Trident requires access to multiple resources in the cluster to make the volume available.

1. To enable access, obtain the Kubernetes config file from one the Iguazio nodes. The file is located under /home/Iguazio/.kube/config. Download this file to your desktop.

2. Go to Discover Cluster to configure.

3. Upload the Kubernetes config file. See the following image.

4. Deploy Trident and associate a volume with the cluster. See the following image on defining and assigning a Persistent Volume to the Iguazio cluster. This process creates a Persistent Volume (PV) in Iguazio's Kubernetes cluster. Before you can use it, you must define a Persistent Volume Claim (PVC).

On-Premises Deployment Kubernetes Configuration

For on-premises installation of NetApp Trident, see TR-4798 for details. After configuring your Kubernetes cluster and installing NetApp Trident, you can connect Trident to the Iguazio cluster to enable NetApp data management capabilities, such as taking Snapshot copies of your data and model.

Next: Define Persistent Volume Claim

Define Persistent Volume Claim

1. Save the following YAML to a file to create a PVC of type Basic.
2. Apply the YAML file to your Iguazio Kubernetes cluster.

```
Kubectl -n default-tenant apply -f <your yaml file>
```

Attach NetApp Volume to the Jupyter Notebook

Iguazio offers several managed services to provide data scientists with a full end-to-end stack for development and deployment of AI/ML applications. You can read more about these components at the Iguazio Overview of Application Services and Tools.

One of the managed services is Jupyter Notebook. Each developer gets its own deployment of a notebook container with the resources they need for development. To give them access to the NetApp Cloud Volume, you can assign the volume to their container and resource allocation, running user, and environment variable settings for Persistent Volume Claims is presented in the following image.

For an on-premises configuration, you can refer to TR-4798 on the Trident setup to enable NetApp ONTAP data management capabilities, such as taking Snapshot copies of your data or model for versioning control. Add the following line in your Trident back-end config file to make Snapshot directories visible:

```
{
  ...
  "defaults": {
    "snapshotDir": "true"
  }
}
```

You must create a Trident back-end config file in JSON format, and then run the following Trident command to reference it:

```
tridentctl create backend -f <backend-file>
```

[Error: Missing Graphic Image]
Deploying the Application

The following sections describe how to install and deploy the application.

Next: Get Code from GitHub.

Get Code from GitHub

Now that the NetApp Cloud Volume or NetApp Trident volume is available to the Iguazio cluster and the developer environment, you can start reviewing the application.

Users have their own workspace (directory). On every notebook, the path to the user directory is `/User`. The Iguazio platform manages the directory. If you follow the instructions above, the NetApp Cloud volume is available in the `/netapp` directory.

Get the code from GitHub using a Jupyter terminal.

[Error: Missing Graphic Image]

At the Jupyter terminal prompt, clone the project.

```bash
cd /User
git clone .
```

You should now see the `netops- netapp` folder on the file tree in Jupyter workspace.

Next: Configure Working Environment

Configure Working Environment

Copy the `Notebook set_env-Example.ipynb` as `set_env.ipynb`. Open and edit `set_env.ipynb`. This notebook sets variables for credentials, file locations, and execution drivers.

If you follow the instructions above, the following steps are the only changes to make:

1. Obtain this value from the Iguazio services dashboard: `docker_registry`
   
   Example: `docker-registry.default-tenant.app.clusterq.iguaziodev.com:80`

2. Change `admin` to your Iguazio username:
   
   `IGZ_CONTAINER_PATH = '/users/admin'`

   The following are the ONTAP system connection details. Include the volume name that was generated when Trident was installed. The following setting is for an on-premises ONTAP cluster:
ontapClusterMgmtHostname = '0.0.0.0'
ontapClusterAdminUsername = 'USER'
ontapClusterAdminPassword = 'PASSWORD'
sourceVolumeName = 'SOURCE VOLUME'

The following setting is for Cloud Volumes ONTAP:

MANAGER=ontapClusterMgmtHostname
svm='svm'
email='email'
password=ontapClusterAdminPassword
weid="weid"
volume=sourceVolumeName

Create Base Docker Images

Everything you need to build an ML pipeline is included in the Iguazio platform. The developer can define the specifications of the Docker images required to run the pipeline and execute the image creation from Jupyter Notebook. Open the notebook create-images.ipynb and Run All Cells.

This notebook creates two images that we use in the pipeline.

- **iguazio/netapp.** Used to handle ML tasks.

- **netapp/pipeline.** Contains utilities to handle NetApp Snapshot copies.

Review Individual Jupyter Notebooks

The following table lists the libraries and frameworks we used to build this task. All these components have been fully integrated with Iguazio’s role-based access and security controls.

<table>
<thead>
<tr>
<th>Libraries/Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLRun</td>
<td>An managed by Iguazio to enable the assembly, execution, and monitoring of an ML/AI pipeline.</td>
</tr>
<tr>
<td>Nuclio</td>
<td>A serverless functions framework integrated with Iguazio. Also available as an open-source project managed by Iguazio.</td>
</tr>
<tr>
<td>Kubeflow</td>
<td>A Kubernetes-based framework to deploy the pipeline. This is also an open-source project to which Iguazio contributes. It is integrated with Iguazio for added security and integration with the rest of the infrastructure.</td>
</tr>
<tr>
<td>Libraries/Framework</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Docker</td>
<td>A Docker registry run as a service in the Iguazio platform. You can also change this to connect to your registry.</td>
</tr>
<tr>
<td>NetApp Cloud Volumes</td>
<td>Cloud Volumes running on AWS give us access to large amounts of data and the ability to take Snapshot copies to version the datasets used for training.</td>
</tr>
<tr>
<td>Trident</td>
<td>Trident is an open-source project managed by NetApp. It facilitates the integration with storage and compute resources in Kubernetes.</td>
</tr>
</tbody>
</table>

We used several notebooks to construct the ML pipeline. Each notebook can be tested individually before being brought together in the pipeline. We cover each notebook individually following the deployment flow of this demonstration application.

The desired result is a pipeline that trains a model based on a Snapshot copy of the data and deploys the model for inference. A block diagram of a completed MLRun pipeline is shown in the following image.

[Error: Missing Graphic Image]

**Deploy Data Generation Function**

This section describes how we used Nuclio serverless functions to generate network device data. The use case is adapted from an Iguazio client that deployed the pipeline and used Iguazio services to monitor and predict network device failures.

We simulated data coming from network devices. Executing the Jupyter notebook `data-generator.ipynb` creates a serverless function that runs every 10 minutes and generates a Parquet file with new data. To deploy the function, run all the cells in this notebook. See the Nuclio website to review any unfamiliar components in this notebook.

A cell with the following comment is ignored when generating the function. Every cell in the notebook is assumed to be part of the function. Import the Nuclio module to enable `%nuclio magic`.

```python
# nuclio: ignore
import nuclio
```

In the spec for the function, we defined the environment in which the function executes, how it is triggered, and the resources it consumes.

```python
spec = nuclio.ConfigSpec(config={"spec.triggers.inference.kind":"cron",
      "spec.triggers.inference.attributes.interval" :"10m",
      "spec.readinessTimeoutSeconds" : 60,
      "spec.minReplicas" : 1},......
```

The `init_context` function is invoked by the Nuclio framework upon initialization of the function.
```python
def init_context(context):
    ...
```

Any code not in a function is invoked when the function initializes. When you invoke it, a handler function is executed. You can change the name of the handler and specify it in the function spec.

```python
def handler(context, event):
    ...
```

You can test the function from the notebook prior to deployment.

```py
%%time
# nuclio: ignore
init_context(context)
event = nuclio.Event(body='')
output = handler(context, event)
output
```

The function can be deployed from the notebook or it can be deployed from a CI/CD pipeline (adapting this code).

```py
addr = nuclio.deploy_file(name='generator', project='netops', spec=spec, tag='v1.1')
```

**Pipeline Notebooks**

These notebooks are not meant to be executed individually for this setup. This is just a review of each notebook. We invoked them as part of the pipeline. To execute them individually, review the MLRun documentation to execute them as Kubernetes jobs.

**snap_cv.ipynb**

This notebook handles the Cloud Volume Snapshot copies at the beginning of the pipeline. It passes the name of the volume to the pipeline context. This notebook invokes a shell script to handle the Snapshot copy. While running in the pipeline, the execution context contains variables to help locate all files needed for execution. While writing this code, the developer does not have to worry about the file location in the container that executes it. As described later, this application is deployed with all its dependencies, and it is the definition of the pipeline parameters that provides the execution context.

```py
command = os.path.join(context.get_param('APP_DIR'),"snap_cv.sh")
```

The created Snapshot copy location is placed in the MLRun context to be consumed by steps in the pipeline.
The next three notebooks are run in parallel.

**data-prep.ipynb**

Raw metrics must be turned into features to enable model training. This notebook reads the raw metrics from the Snapshot directory and writes the features for model training to the NetApp volume.

When running in the context of the pipeline, the input `DATA_DIR` contains the Snapshot copy location.

```python
metrics_table = os.path.join(str(mlruncontext.get_input('DATA_DIR', os.getenv('DATA_DIR','/netpp'))),
                           mlruncontext.get_param('metrics_table',
                           os.getenv('metrics_table','netops_metrics_parquet')))
```

**describe.ipynb**

To visualize the incoming metrics, we deploy a pipeline step that provides plots and graphs that are available through the Kubeflow and MLRun UIs. Each execution has its own version of this visualization tool.

```python
ax.set_title("features correlation")
plt.savefig(os.path.join(base_path, "plots/corr.png"))
context.log_artifact(PlotArtifact("correlation", body=plt.gcf()), local_path="plots/corr.html")
```

**deploy-feature-function.ipynb**

We continuously monitor the metrics looking for anomalies. This notebook creates a serverless function that generates the features need to run prediction on incoming metrics. This notebook invokes the creation of the function. The function code is in the notebook `data-prep.ipynb`. Notice that we use the same notebook as a step in the pipeline for this purpose.

**training.ipynb**

After we create the features, we trigger the model training. The output of this step is the model to be used for inferencing. We also collect statistics to keep track of each execution (experiment).

For example, the following command enters the accuracy score into the context for that experiment. This value is visible in Kubeflow and MLRun.

```python
context.log_result('accuracy', score)
```
deploy-inference-function.ipynb

The last step in the pipeline is to deploy the model as a serverless function for continuous inferencing. This notebook invokes the creation of the serverless function defined in nuclio-inference-function.ipynb.

Review and Build Pipeline

The combination of running all the notebooks in a pipeline enables the continuous run of experiments to reassess the accuracy of the model against new metrics. First, open the pipeline.ipynb notebook. We take you through details that show how NetApp and Iguazio simplify the deployment of this ML pipeline.

We use MLRun to provide context and handle resource allocation to each step of the pipeline. The MLRun API service runs in the Iguazio platform and is the point of interaction with Kubernetes resources. Each developer cannot directly request resources; the API handles the requests and enables access controls.

```python
# MLRun API connection definition
mlconf.dbpath = 'http://mlrun-api:8080'
```

The pipeline can work with NetApp Cloud Volumes and on-premises volumes. We built this demonstration to use Cloud Volumes, but you can see in the code the option to run on-premises.

```python
# Initialize the NetApp snap function once for all functions in a notebook
if [ NETAPP_CLOUD_VOLUME ]:
    snapfn = 
    code_to_function('snap',project='NetApp',kind='job',filename="snap_cv.ipynb").apply(mount_v3io())
    snap_params = {
        "metrics_table" : metrics_table,
        "NETAPP_MOUNT_PATH" : NETAPP_MOUNT_PATH,
        'MANAGER' : MANAGER,
        'svm' : svm,
        'email': email,
        'password': password,
        'weid': weid,
        'volume': volume,
        "APP_DIR" : APP_DIR
    }
else:
    snapfn = 
    code_to_function('snap',project='NetApp',kind='job',filename="snapshot.ipynb").apply(mount_v3io())
...
```
```
snapfn.spec.image = docker_registry + '/netapp/pipeline:latest'
snapfn.spec.volume_mounts =
    [snapfn.spec.volume_mounts[0],netapp_volume_mounts]
snapfn.spec.volumes = [ snapfn.spec.volumes[0],netapp_volumes]
```
The first action needed to turn a Jupyter notebook into a Kubeflow step is to turn the code into a function. A function has all the specifications required to run that notebook. As you scroll down the notebook, you can see that we define a function for every step in the pipeline.

<table>
<thead>
<tr>
<th>Part of the Notebook</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;code_to_function&gt;</td>
<td>Name of the function: Project name. used to organize all project artifacts. This is visible in the MLRun UI. Kind. In this case, a Kubernetes job. This could be Dask, mpi, sparkk8s, and more. See the MLRun documentation for more details. File. The name of the notebook. This can also be a location in Git (HTTP).</td>
</tr>
<tr>
<td>image</td>
<td>The name of the Docker image we are using for this step. We created this earlier with the create-image.ipynb notebook.</td>
</tr>
<tr>
<td>volume_mounts &amp; volumes</td>
<td>Details to mount the NetApp Cloud Volume at run time.</td>
</tr>
</tbody>
</table>

We also define parameters for the steps.

```python
```

After you have the function definition for all steps, you can construct the pipeline. We use the kfp module to make this definition. The difference between using MLRun and building on your own is the simplification and shortening of the coding.

The functions we defined are turned into step components using the as_step function of MLRun.
Snapshot Step Definition

Initiate a Snapshot function, output, and mount v3io as source:

```python
snap = snapfn.as_step(NewTask(handler='handler',params=snap_params),
name='NetApp_Cloud_Volume_Snapshot',outputs=['snapVolumeDetails','training_parquet_file']).apply(mount_v3io())
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewTask</td>
<td>NewTask is the definition of the function run.</td>
</tr>
<tr>
<td>(MLRun module)</td>
<td>Handler. Name of the Python function to invoke. We used the name handler in the notebook, but it is not required. params. The parameters we passed to the execution. Inside our code, we use context.get_param('PARAMETER') to get the values.</td>
</tr>
<tr>
<td>as_step</td>
<td>Name. Name of the Kubeflow pipeline step. outputs. These are the values that the step adds to the dictionary on completion. Take a look at the snap_cv.ipynb notebook. mount_v3io(). This configures the step to mount /User for the user executing the pipeline.</td>
</tr>
</tbody>
</table>

```python
prep = data_prep.as_step(name='data-prep',
handler='handler',params=params,
    inputs = {'DATA_DIR': snap.outputs['snapVolumeDetails']},
    out_path=artifacts_path).apply(mount_v3io()).after(snap)
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs</td>
<td>You can pass to a step the outputs of a previous step. In this case, snap.outputs['snapVolumeDetails'] is the name of the Snapshot copy we created on the snap step.</td>
</tr>
<tr>
<td>out_path</td>
<td>A location to place artifacts generating using the MLRun module log_artifacts.</td>
</tr>
</tbody>
</table>

You can run `pipeline.ipynb` from top to bottom. You can then go to the Pipelines tab from the Iguazio dashboard to monitor progress as seen in the Iguazio dashboard Pipelines tab.

[Error: Missing Graphic Image]

Because we logged the accuracy of training step in every run, we have a record of accuracy for each experiment, as seen in the record of training accuracy.
If you select the Snapshot step, you can see the name of the Snapshot copy that was used to run this experiment.

The described step has visual artifacts to explore the metrics we used. You can expand to view the full plot as seen in the following image.

The MLRun API database also tracks inputs, outputs, and artifacts for each run organized by project. An example of inputs, outputs, and artifacts for each run can be seen in the following image.

For each job, we store additional details.

There is more information about MLRun than we can cover in this document. AI artifacts, including the definition of the steps and functions, can be saved to the API database, versioned, and invoked individually or as a full project. Projects can also be saved and pushed to Git for later use. We encourage you to learn more at the MLRun GitHub site.

**Deploy Grafana Dashboard**

After everything is deployed, we run inferences on new data. The models predict failure on network device equipment. The results of the prediction are stored in an Iguazio TimeSeries table. You can visualize the results with Grafana in the platform integrated with Iguazio’s security and data access policy.

You can deploy the dashboard by importing the provided JSON file into the Grafana interfaces in the cluster.

1. To verify that the Grafana service is running, look under Services.

2. If it is not present, deploy an instance from the Services section:
   a. Click New Service.
   b. Select Grafana from the list.
   c. Accept the defaults.
   d. Click Next Step.
   e. Enter your user ID.
   f. Click Save Service.
   g. Click Apply Changes at the top.

3. To deploy the dashboard, download the file `NetopsPredictions-Dashboard.json` through the Jupyter interface.
4. Open Grafana from the Services section and import the dashboard.

[Error: Missing Graphic Image]

5. Click Upload *.json File and select the file that you downloaded earlier (NetopsPredictions-Dashboard.json). The dashboard displays after the upload is completed.

[Error: Missing Graphic Image]

**Deploy Cleanup Function**

When you generate a lot of data, it is important to keep things clean and organized. To do so, deploy the cleanup function with the cleanup.ipynb notebook.

**Benefits**

NetApp and Iguazio speed up and simplify the deployment of AI and ML applications by building in essential frameworks, such as Kubeflow, Apache Spark, and TensorFlow, along with orchestration tools like Docker and Kubernetes. By unifying the end-to-end data pipeline, NetApp and Iguazio reduce the latency and complexity inherent in many advanced computing workloads, effectively bridging the gap between development and operations. Data scientists can run queries on large datasets and securely share data and algorithmic models with authorized users during the training phase. After the containerized models are ready for production, you can easily move them from development environments to operational environments.

Next: Conclusion

**Conclusion**

When building your own AI/ML pipelines, configuring the integration, management, security, and accessibility of the components in an architecture is a challenging task. Giving developers access and control of their environment presents another set of challenges.

The combination of NetApp and Iguazio brings these technologies together as managed services to accelerate technology adoption and improve the time to market for new AI/ML applications.

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:

- NetApp AI Control Plane:
  - NetApp AI Control Plane Technical Report
    

- NetApp persistent storage for containers:
  - NetApp Trident
    
    https://netapp.io/persistent-storage-provisioner-for-kubernetes/
• ML framework and tools:
  ◦ Docker
    https://docs.docker.com
  ◦ Kubernetes
    https://kubernetes.io/docs/home/
  ◦ Kubeflow
    http://www.kubeflow.org/
  ◦ Jupyter Notebook Server
    http://www.jupyter.org/

• Iguazio Data Science Platform
  ◦ Iguazio Data Science Platform Documentation
    https://www.iguazio.com/docs/
  ◦ Nuclio serverless function
    https://nuclio.io/
  ◦ MLRun open source pipeline orchestration framework
    https://www.iguazio.com/open-source/mlrun/

• NVIDIA DGX-1 systems
  ◦ NVIDIA DGX-1 systems
  ◦ NVIDIA Tesla V100 Tensor core GPU
  ◦ NVIDIA GPU Cloud

• NetApp AFF systems
  ◦ AFF datasheet
  ◦ NetApp Flash Advantage for AFF
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 straightforward 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.
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.
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
- Monitoring
cnvr.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.

[Error: Missing Graphic Image]

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.

![Error: Missing Graphic Image]

**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 thereby 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.

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu</td>
<td>18.04.4 LTS</td>
</tr>
<tr>
<td>NVIDIA DGX OS</td>
<td>4.4.0</td>
</tr>
<tr>
<td>NVIDIA DeepOps</td>
<td>20.02.1</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>1.15</td>
</tr>
<tr>
<td>Helm</td>
<td>3.1.0</td>
</tr>
<tr>
<td>cnvrg.io</td>
<td>3.0.0</td>
</tr>
<tr>
<td>NetApp ONTAP</td>
<td>9.6P4</td>
</tr>
</tbody>
</table>

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.

**Solution Deployment and Validation Details**

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

**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.

**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:

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

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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

Cache 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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]
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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

[Error: Missing Graphic Image]

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:

• Cnvrng.io (https://cnvrng.io):
  ◦ Cnvrng CORE (free ML platform)
    https://cnvrng.io/platform/core
  ◦ Cnvrng docs
    https://app.cnvrng.io/docs

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

• NetApp AFF systems:
  ◦ AFF datasheet
  ◦ NetApp FlashAdvantage for AFF
  ◦ ONTAP 9.x documentation
  ◦ NetApp FlexGroup technical report

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

• NetApp Interoperability Matrix:
  ◦ NetApp Interoperability Matrix Tool
    http://support.netapp.com/matrix
• ONTAP AI networking:
  ◦ Cisco Nexus 3232C Switches
  ◦ Mellanox Spectrum 2000 series switches

• ML framework and tools:
  ◦ DALI
    https://github.com/NVIDIA/DALI
    https://www.tensorflow.org/
  ◦ 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
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

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 points

NetApp 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**

[Error: Missing Graphic Image]

**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.
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.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>T4 or V100 GPU</td>
<td>1</td>
</tr>
<tr>
<td>NVIDIA DGX Station</td>
<td>1</td>
</tr>
</tbody>
</table>

Software Requirements

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

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

See 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

```yaml
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
```
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
```yaml
# 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_retail_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
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 trouble understanding right now. Come again on that, else check jarvis services?"
  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:
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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

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.
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
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 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 --network=host --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.

```bash
cd examples/nlp/intent_detection_slot_tagging/
python joint_intent_slot_with_bert.py \
   --data_dir /nemo/training_data\n   --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.

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

```bash
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/ \n   --query "where can i get the best pasta" \n   --data_dir /nemo/training_data/ \n   --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.
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.

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.
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
  - NVIDIA V100 Tensor Core GPU
  - NVIDIA NGC

- 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
  - NetApp Flash Advantage for All Flash FAS
  - ONTAP 9 Information Library
  - NetApp ONTAP FlexGroup Volumes technical report

- NetApp ONTAP AI
  - ONTAP AI with DGX-1 and Cisco Networking Design Guide
  - ONTAP AI with DGX-1 and Cisco Networking Deployment Guide
  - ONTAP AI with DGX-1 and Mellanox Networking Design Guide
  - ONTAP AI with DGX-2 Design Guide
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.

[Error: Missing Graphic Image]

**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.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGX-1 systems</td>
<td>2</td>
</tr>
<tr>
<td>AFF A800</td>
<td>1</td>
</tr>
</tbody>
</table>
**Hardware**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 3232C switches</td>
<td>2</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version or Other Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetApp ONTAP data management software</td>
<td>9.6p4</td>
</tr>
<tr>
<td>Cisco NX-OS switch firmware</td>
<td>7.0(3)I6(1)</td>
</tr>
<tr>
<td>NVIDIA DGX OS</td>
<td>4.0.4 - Ubuntu 18.04 LTS</td>
</tr>
<tr>
<td>Kubernetes version</td>
<td>1.17</td>
</tr>
<tr>
<td>Trident version</td>
<td>20.04.0</td>
</tr>
<tr>
<td>Run:AI CLI</td>
<td>v2.1.13</td>
</tr>
<tr>
<td>Run:AI Orchestration Kubernetes Operator version</td>
<td>1.0.39</td>
</tr>
<tr>
<td>Docker container platform</td>
<td>18.06.1-ce [e68fc7a]</td>
</tr>
</tbody>
</table>

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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

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:
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.

[Error: Missing Graphic Image]

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:
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:

```bash
$ 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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

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:

<table>
<thead>
<tr>
<th>Project</th>
<th>Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>4</td>
</tr>
<tr>
<td>team-b</td>
<td>2</td>
</tr>
<tr>
<td>team-c</td>
<td>2</td>
</tr>
<tr>
<td>team-d</td>
<td>8</td>
</tr>
</tbody>
</table>

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.

[Error: Missing Graphic Image]

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.

[Error: Missing Graphic Image]

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:

<table>
<thead>
<tr>
<th>Project</th>
<th>GPUs allocated</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>8/4</td>
<td>Four GPUs over the quota. Empty queue.</td>
</tr>
<tr>
<td>team-b</td>
<td>4/2</td>
<td>Two GPUs over the quota. One workload queued.</td>
</tr>
<tr>
<td>team-c</td>
<td>4/2</td>
<td>Two GPUs over the quota. One workload queued.</td>
</tr>
<tr>
<td>team-d</td>
<td>0/8</td>
<td>Not using GPUs at all, no queued workloads.</td>
</tr>
</tbody>
</table>

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.

**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: Conclusion**
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:

<table>
<thead>
<tr>
<th>Project</th>
<th>Image</th>
<th># GPUs</th>
<th>Total</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>Jupyter</td>
<td>1</td>
<td>1/4</td>
<td>–</td>
</tr>
<tr>
<td>team-a</td>
<td>NetApp</td>
<td>1</td>
<td>2/4</td>
<td>–</td>
</tr>
<tr>
<td>team-a</td>
<td>Run:AI</td>
<td>2</td>
<td>4/4</td>
<td>Using all their quota</td>
</tr>
<tr>
<td>team-b</td>
<td>Run:AI</td>
<td>0.6</td>
<td>0.6/2</td>
<td>Fractional GPU</td>
</tr>
<tr>
<td>team-b</td>
<td>Run:AI</td>
<td>0.4</td>
<td>1/2</td>
<td>Fractional GPU</td>
</tr>
<tr>
<td>team-b</td>
<td>NetApp</td>
<td>1</td>
<td>2/2</td>
<td>–</td>
</tr>
<tr>
<td>team-b</td>
<td>NetApp</td>
<td>2</td>
<td>4/2</td>
<td>Two over quota</td>
</tr>
<tr>
<td>team-c</td>
<td>Run:AI</td>
<td>0.5</td>
<td>0.5/2</td>
<td>Fractional GPU</td>
</tr>
<tr>
<td>team-c</td>
<td>Run:AI</td>
<td>0.3</td>
<td>0.8/2</td>
<td>Fractional GPU</td>
</tr>
<tr>
<td>team-c</td>
<td>Run:AI</td>
<td>0.2</td>
<td>1/2</td>
<td>Fractional GPU</td>
</tr>
<tr>
<td>team-c</td>
<td>NetApp</td>
<td>2</td>
<td>3/2</td>
<td>One over quota</td>
</tr>
<tr>
<td>team-c</td>
<td>NetApp</td>
<td>1</td>
<td>4/2</td>
<td>Two over quota</td>
</tr>
<tr>
<td>team-d</td>
<td>NetApp</td>
<td>4</td>
<td>4/8</td>
<td>Using half of their quota</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Project</th>
<th>CPUs Allocated</th>
<th>Workloads Queued</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>4/4 (soft quota/actual allocation)</td>
<td>None</td>
</tr>
<tr>
<td>team-b</td>
<td>4/2</td>
<td>None</td>
</tr>
<tr>
<td>team-c</td>
<td>4/2</td>
<td>None</td>
</tr>
<tr>
<td>team-d</td>
<td>4/8</td>
<td>None</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Project</th>
<th># GPUs</th>
<th>Total</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-d</td>
<td>2</td>
<td>6/8</td>
<td>Team-b/c workload pauses</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Project</th>
<th>GPUs Allocated</th>
<th>Workloads Queued</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>4/4</td>
<td>None</td>
</tr>
<tr>
<td>team-b</td>
<td>2/2</td>
<td>None</td>
</tr>
<tr>
<td>team-c</td>
<td>2/2</td>
<td>None</td>
</tr>
<tr>
<td>team-d</td>
<td>8/8</td>
<td>None</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Project</th>
<th># GPUs</th>
<th>Total</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>2</td>
<td>4/4</td>
<td>1 workload queued</td>
</tr>
<tr>
<td>team-a</td>
<td>2</td>
<td>4/4</td>
<td>2 workloads queued</td>
</tr>
<tr>
<td>team-b</td>
<td>2</td>
<td>2/2</td>
<td>2 workloads queued</td>
</tr>
<tr>
<td>team-c</td>
<td>2</td>
<td>2/2</td>
<td>2 workloads queued</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Project</th>
<th>GPUs Allocated</th>
<th>Workloads Queued</th>
</tr>
</thead>
<tbody>
<tr>
<td>team-a</td>
<td>4/4</td>
<td>Two workloads asking for GPUs two each</td>
</tr>
<tr>
<td>team-b</td>
<td>2/2</td>
<td>Two workloads asking for two GPUs each</td>
</tr>
<tr>
<td>team-c</td>
<td>2/2</td>
<td>Two workloads asking for two GPUs each</td>
</tr>
<tr>
<td>team-d</td>
<td>8/8</td>
<td>None</td>
</tr>
</tbody>
</table>

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
  - NVIDIA V100 Tensor Core GPU
  - NVIDIA NGC
- 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-Al-on-an-on-premise-Kubernetes-Cluster/
  - Submitting jobs in Run:AI CLI
  - Allocating GPU fractions in Run:AI CLI
    https://docs.run.ai/Researcher/Walkthroughs/Walkthrough-Using-GPU-Fractions/
- NetApp AI Control Plane
  - Technical report
  - Short-form demo
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.

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

**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.

[Error: Missing Graphic Image]

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

[Error: Missing Graphic Image]

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.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Product Family</th>
<th>Quantity</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>H615c</td>
<td>2</td>
<td>3 NVIDIA Tesla T4 GPUs per node</td>
</tr>
<tr>
<td></td>
<td>H410c</td>
<td>2</td>
<td>Compute nodes for management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>infrastructure</td>
</tr>
<tr>
<td>Layer</td>
<td>Product Family</td>
<td>Quantity</td>
<td>Details</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Storage</td>
<td>H410s</td>
<td>2</td>
<td>Storage for OS and workload</td>
</tr>
<tr>
<td>Network</td>
<td>Mellanox SN2010</td>
<td>2</td>
<td>10G/25G switches</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>NetApp Element software</td>
<td>12.0.0.333</td>
</tr>
<tr>
<td></td>
<td>ONTAP Select</td>
<td>9.7</td>
</tr>
<tr>
<td></td>
<td>NetApp Trident</td>
<td>20.07</td>
</tr>
<tr>
<td>NetApp HCI engine</td>
<td>NDE</td>
<td>1.8</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>Hypervisor</td>
<td>VMware vSphere ESXi 6.7U1</td>
</tr>
<tr>
<td></td>
<td>Hypervisor Management System</td>
<td>VMware vCenter Server 6.7U1</td>
</tr>
<tr>
<td>Inferencing Platform</td>
<td>NVIDIA DeepOps</td>
<td>20.08</td>
</tr>
<tr>
<td></td>
<td>NVIDIA GPU Operator</td>
<td>1.1.7</td>
</tr>
<tr>
<td></td>
<td>Ansible</td>
<td>2.9.5</td>
</tr>
<tr>
<td></td>
<td>Kubernetes</td>
<td>1.17.9</td>
</tr>
<tr>
<td></td>
<td>Docker</td>
<td>Docker CE 18.09.7</td>
</tr>
<tr>
<td></td>
<td>CUDA Version</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td>GPU Device Plugin</td>
<td>0.6.0</td>
</tr>
<tr>
<td></td>
<td>Helm</td>
<td>3.1.2</td>
</tr>
<tr>
<td></td>
<td>NVIDIA Tesla Driver</td>
<td>440.64.00</td>
</tr>
<tr>
<td></td>
<td>NVIDIA Triton Inference Server</td>
<td>2.1.0 – NGC Container v20.07</td>
</tr>
<tr>
<td>K8 Master VMs</td>
<td>Linux</td>
<td>Any supported distribution across NetApp IMT, NVIDIA DeepOps, and GPUOperator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ubuntu 18.04.4 LTS was used in this solution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kernel version: 4.15</td>
</tr>
</tbody>
</table>
### 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
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.

![Error: Missing Graphic Image]

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

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

   [Error: Missing Graphic Image]

   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
- NetApp Product Documentation
  docs.netapp.com
- NetApp HCI Solution Catalog Documentation
- HCI Resources page
https://mysupport.netapp.com/info/web/ECMLP2831412.html

• ONTAP Select

• NetApp Trident

• NVIDIA DeepOps
  https://github.com/NVIDIA/deepops

• NVIDIA Triton Inference Server
Copyright Information

Copyright © 2021 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.