Data Pipelines, Data Lakes and Management

NetApp Solutions

NetApp
March 12, 2024

This PDF was generated from https://docs.netapp.com/us-en/netapp-solutions/ai/mlops_fsxn_s3_integration.html on March 12, 2024. Always check docs.netapp.com for the latest.

Table of Contents

Data Pipelines, Data Lakes and Management

AWS FSx for NetApp ONTAP (FSxN) for MLOps

Author(s):

Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

This section delves into the practical application of AI infrastructure development, providing an end-to-end walkthrough of constructing an MLOps pipeline using FSxN. Comprising three comprehensive examples, it guides you to meet your MLOps needs via this powerful data management platform.

These articles focus on:

1. Part 1 - Integrating AWS FSx for NetApp ONTAP (FSxN) as a private S3 bucket into AWS SageMaker
2. Part 2 - Leveraging AWS FSx for NetApp ONTAP (FSxN) as a Data Source for Model Training in SageMaker
3. Part 3 - Building A Simplified MLOps Pipeline (CI/CT/CD)

By the end of this section, you will have gained a solid understanding of how to use FSxN to streamline MLOps processes.

Part 1 - Integrating AWS FSx for NetApp ONTAP (FSxN) as a private S3 bucket into AWS SageMaker

Author(s):

Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

Introduction

Using SageMaker as an example, this page provides guidance on configuring FSxN as a private S3 bucket.

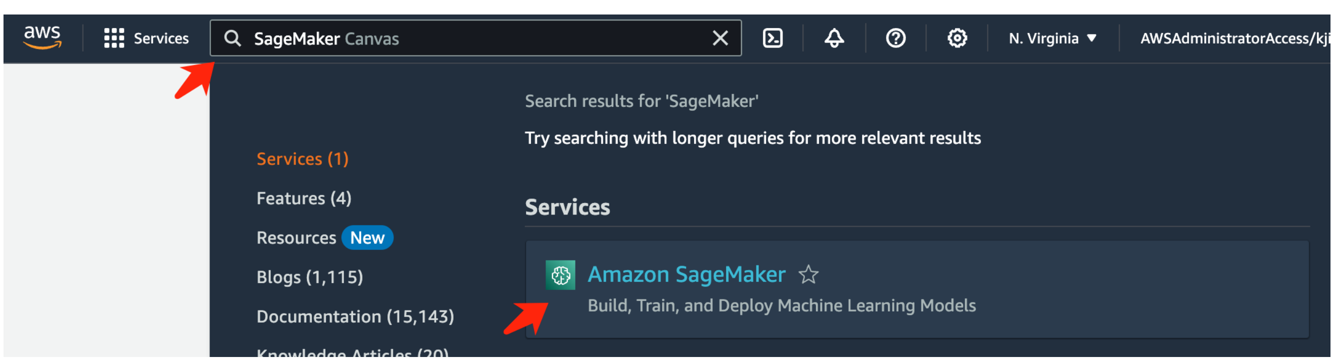
For more information about FSxN, please take a look at this presentation (Video Link)

User Guide

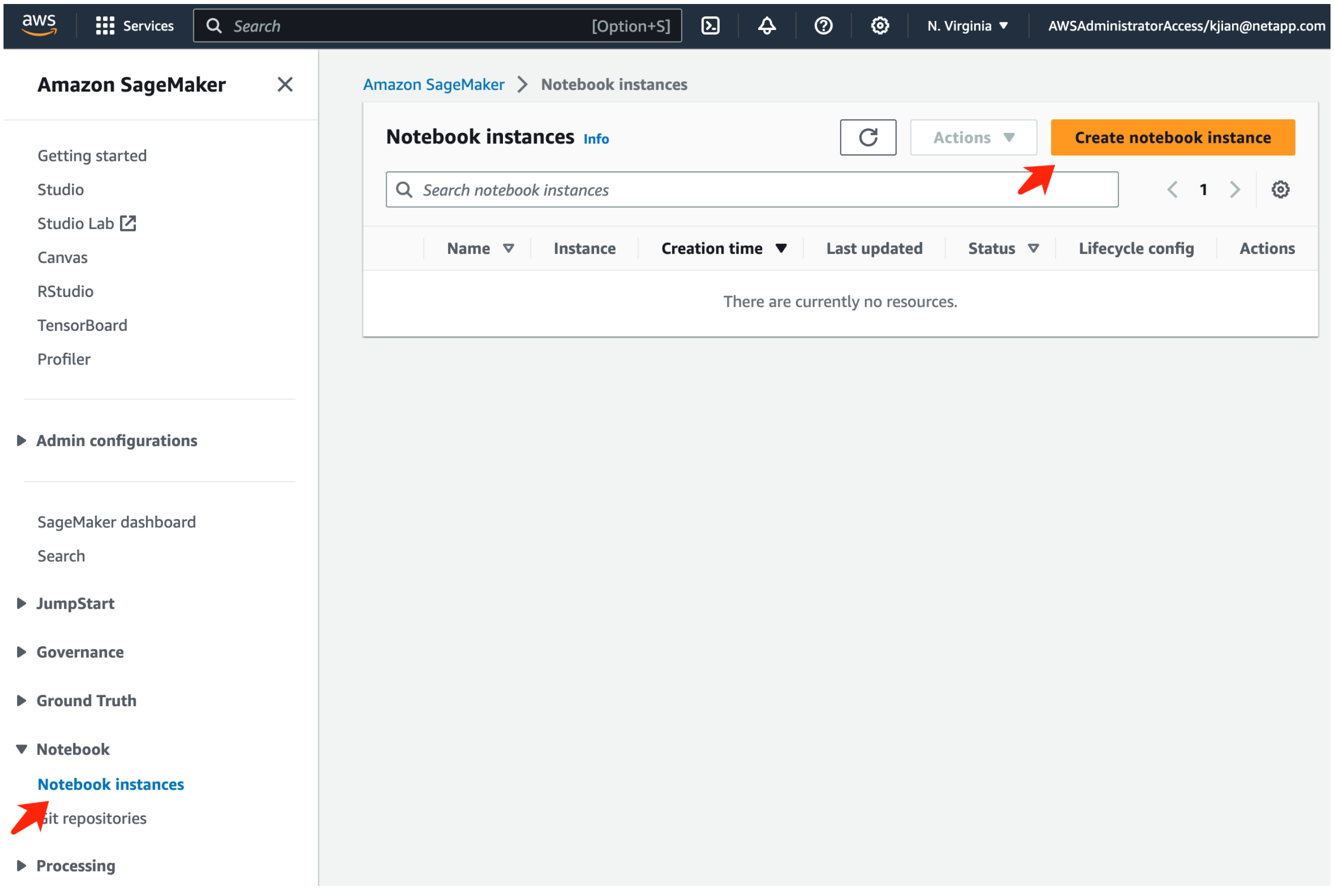
Server creation

Create a SageMaker Notebook Instance

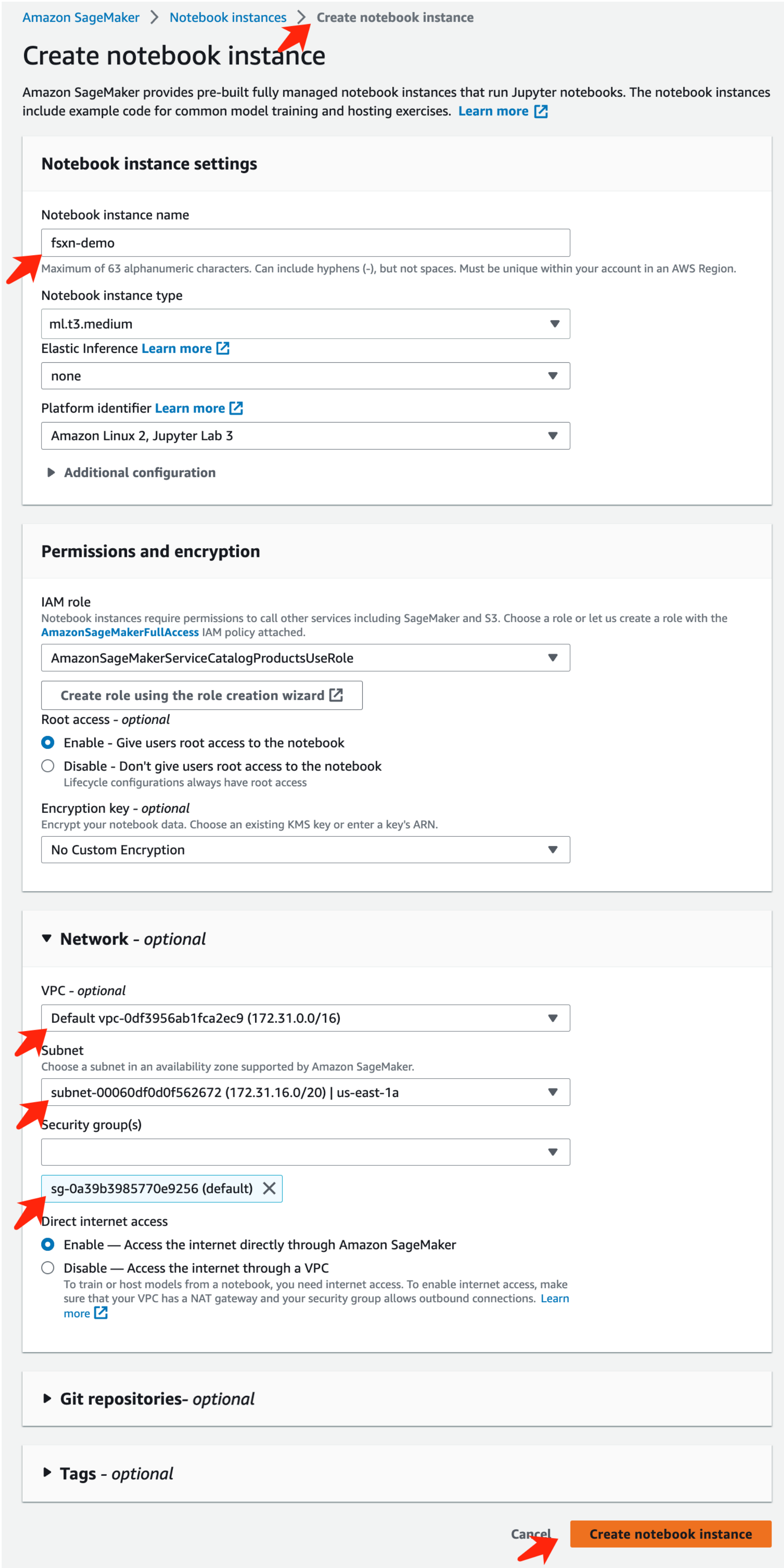
1. Open AWS console. In the search panel, search SageMaker and click the service **Amazon SageMaker**.



2. Open the **Notebook instances** under Notebook tab, click the orange button **Create notebook instance**.

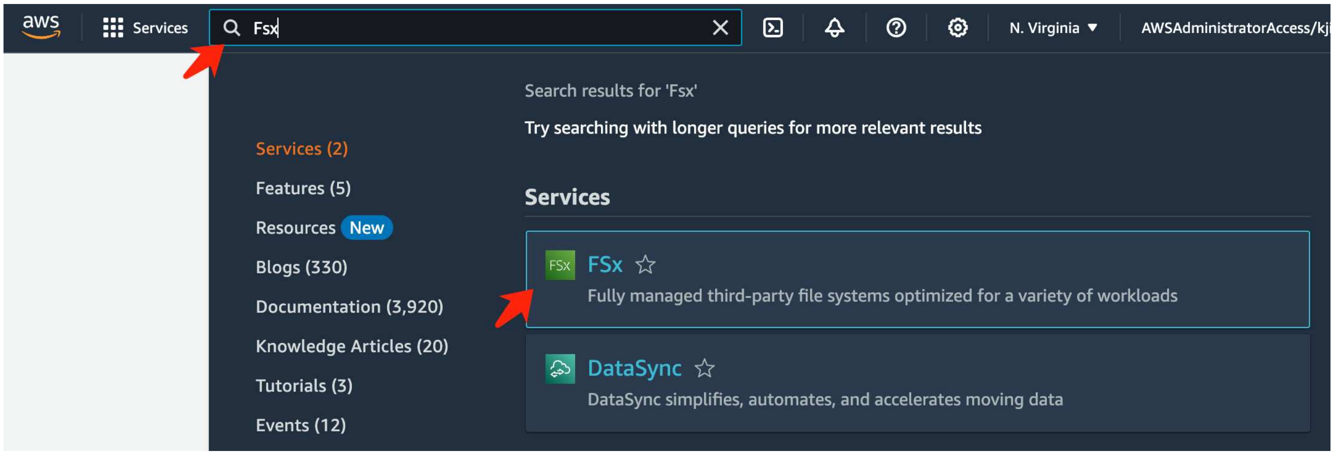


3. In the creation page,
   Enter the **Notebook instance name**
   Expand the **Network** panel
   Leave other entries default and select a **VPC**, **Subnet**, and **Security group(s)**. (This **VPC** and **Subnet** will be used to create FSxN file system later)
   Click the orange button **Create notebook instance** at the bottom right.

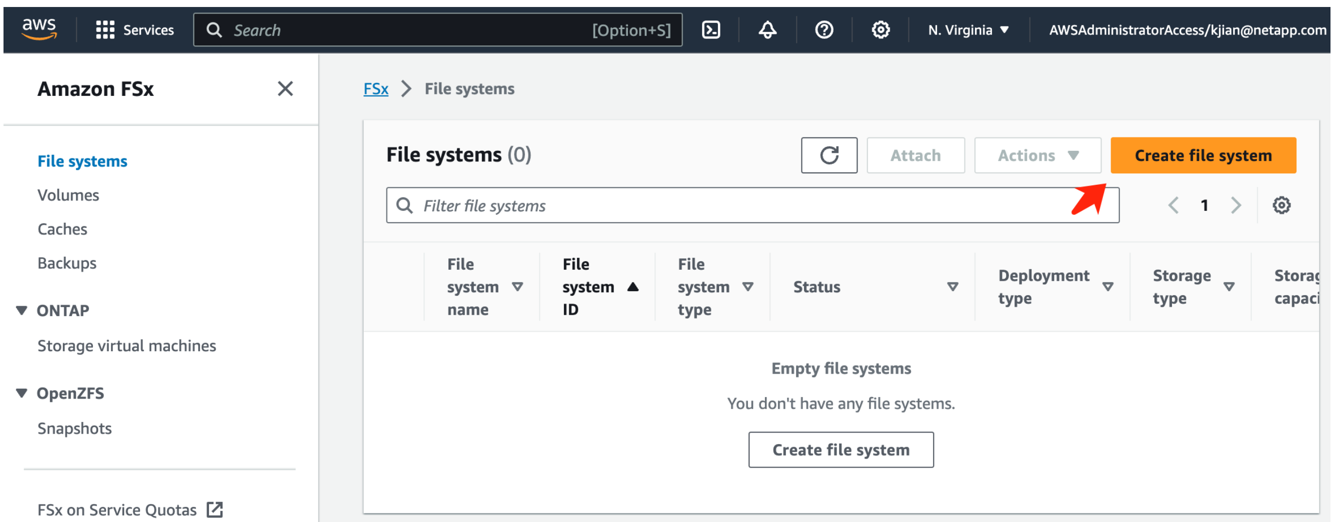

Amazon SageMaker > Notebook instances > Create notebook instance
Create notebook instance
Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. Learn more
Notebook instance settings
Notebook instance name
fsxn-demo
Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.
Notebook instance type
ml.t3.medium
Elastic Inference Learn more
none
Platform identifier Learn more
Amazon Linux 2, Jupyter Lab 3
Additional configuration
Permissions and encryption
IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the AmazonSageMakerFullAccess IAM policy attached.
AmazonSageMakerServiceCatalogProductsUseRole
Create role using the role creation wizard
Root access - optional
Enable - Give users root access to the notebook
Disable - Don't give users root access to the notebook
Lifecycle configurations always have root access
Encryption key - optional
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.
No Custom Encryption
Network - optional
VPC - optional
Default vpc-0df3956ab1fca2ec9 (172.31.0.0/16)
Subnet
Choose a subnet in an availability zone supported by Amazon SageMaker.
subnet-00060df0d0f562672 (172.31.16.0/20) | us-east-1a
Security group(s)
sg-0a39b3985770e9256 (default)
Direct internet access
Enable — Access the internet directly through Amazon SageMaker
Disable — Access the internet through a VPC
To train or host models from a notebook, you need internet access. To enable internet access, make sure that your VPC has a NAT gateway and your security group allows outbound connections. Learn more
Git repositories- optional
Tags - optional
Cancel
Create notebook instance

Create an FSxN File System

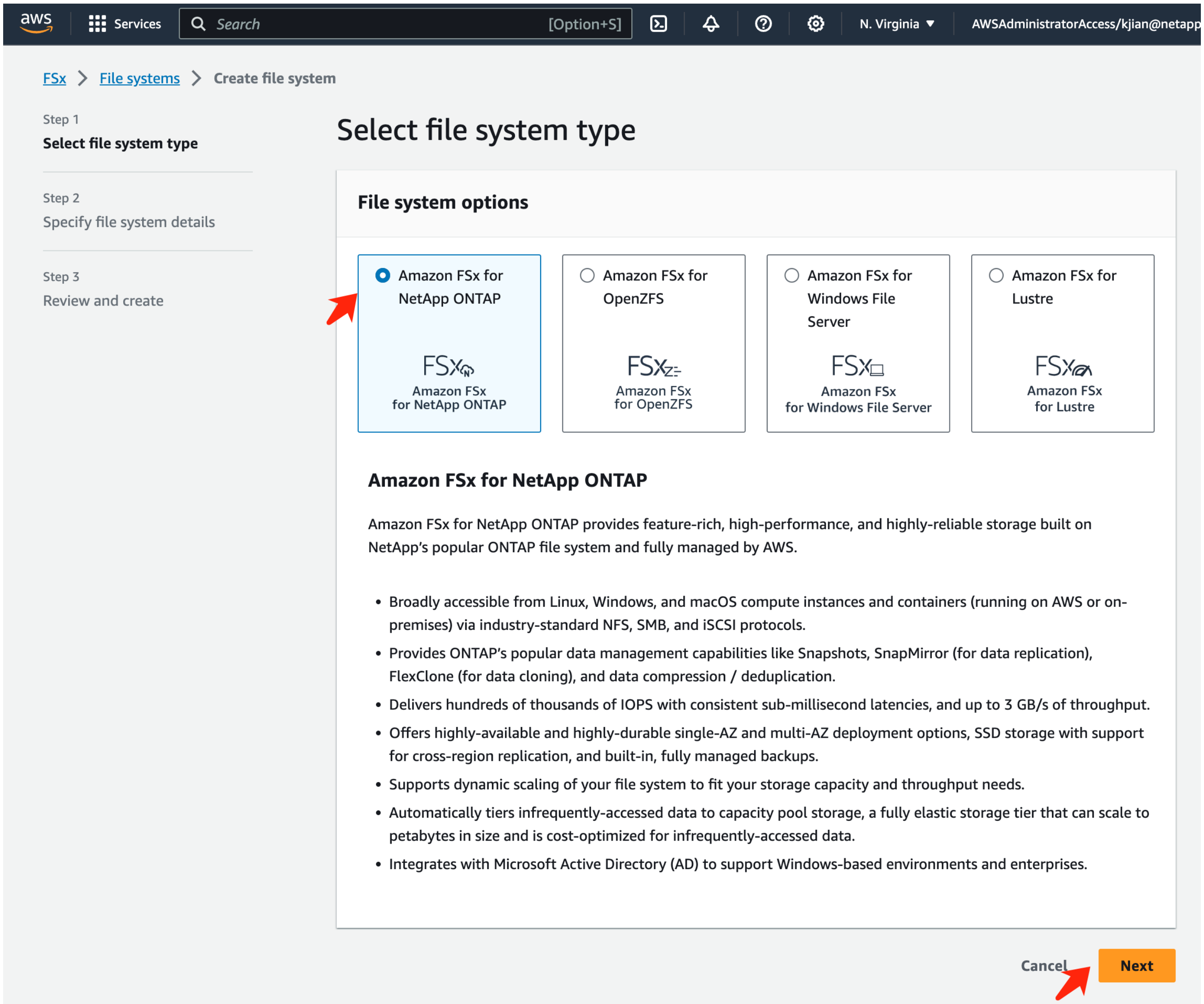
1. Open AWS console. In the search panel, search Fsx and click the service **FSx**.



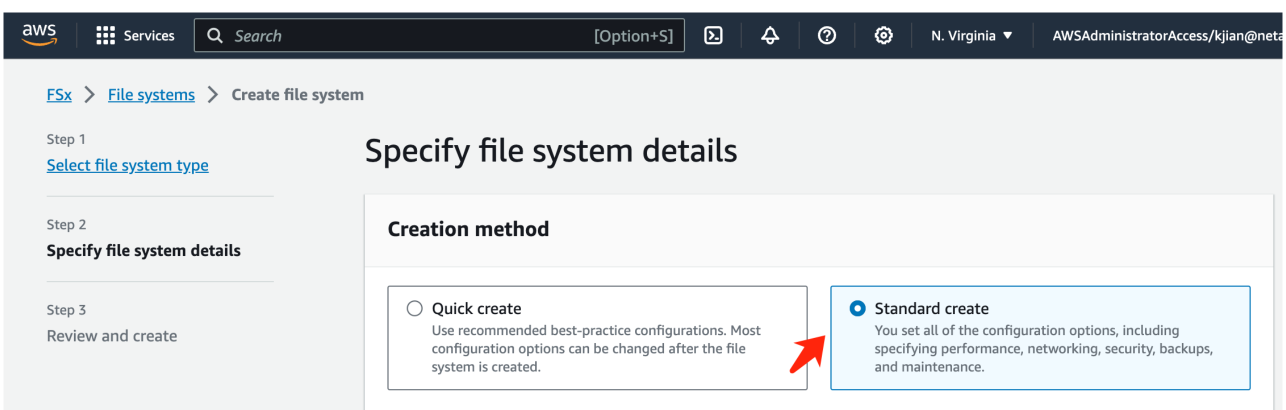
2. Click **Create file system**.



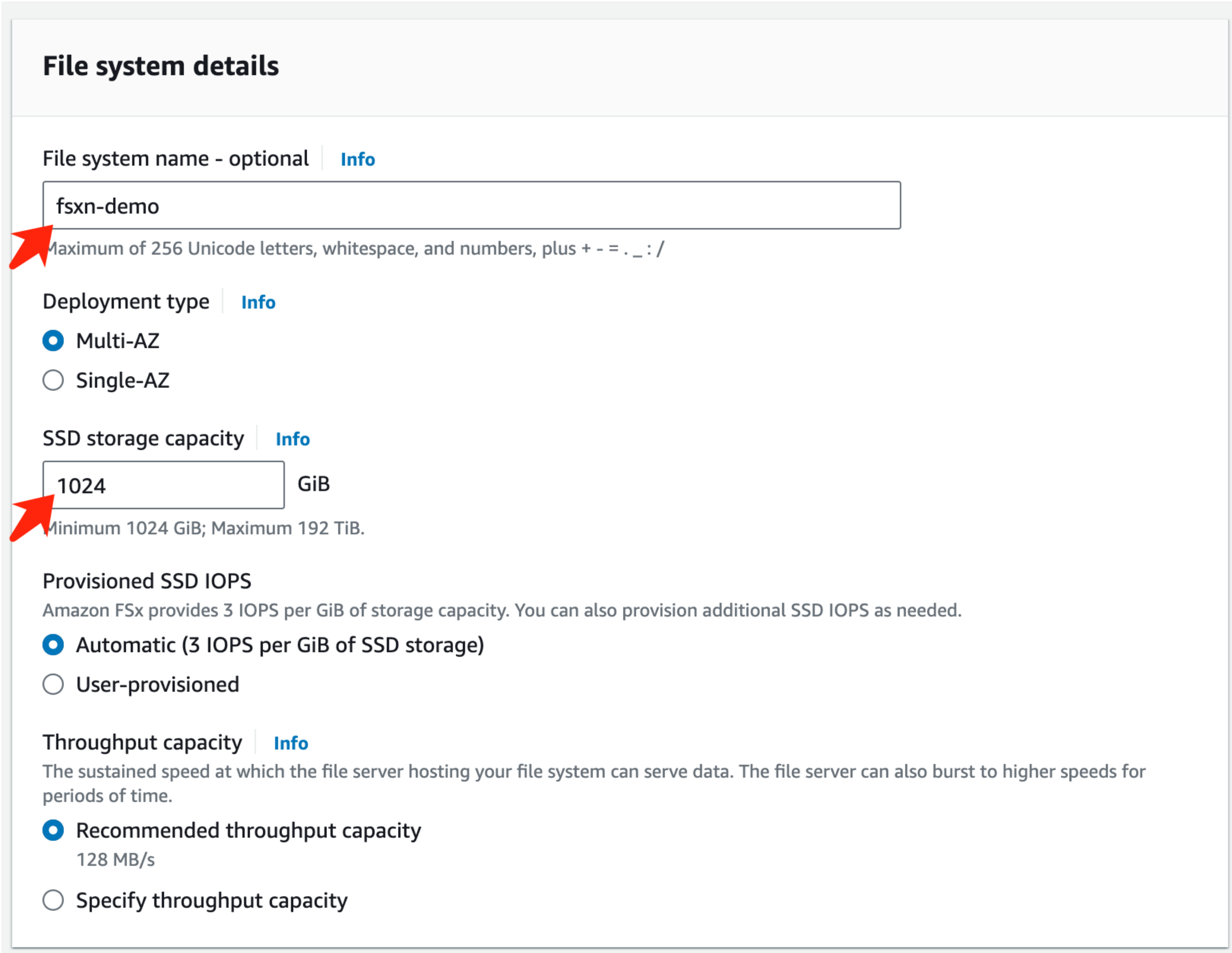
3. Select the first card **FSx for NetApp ONTAP** and click **Next**.



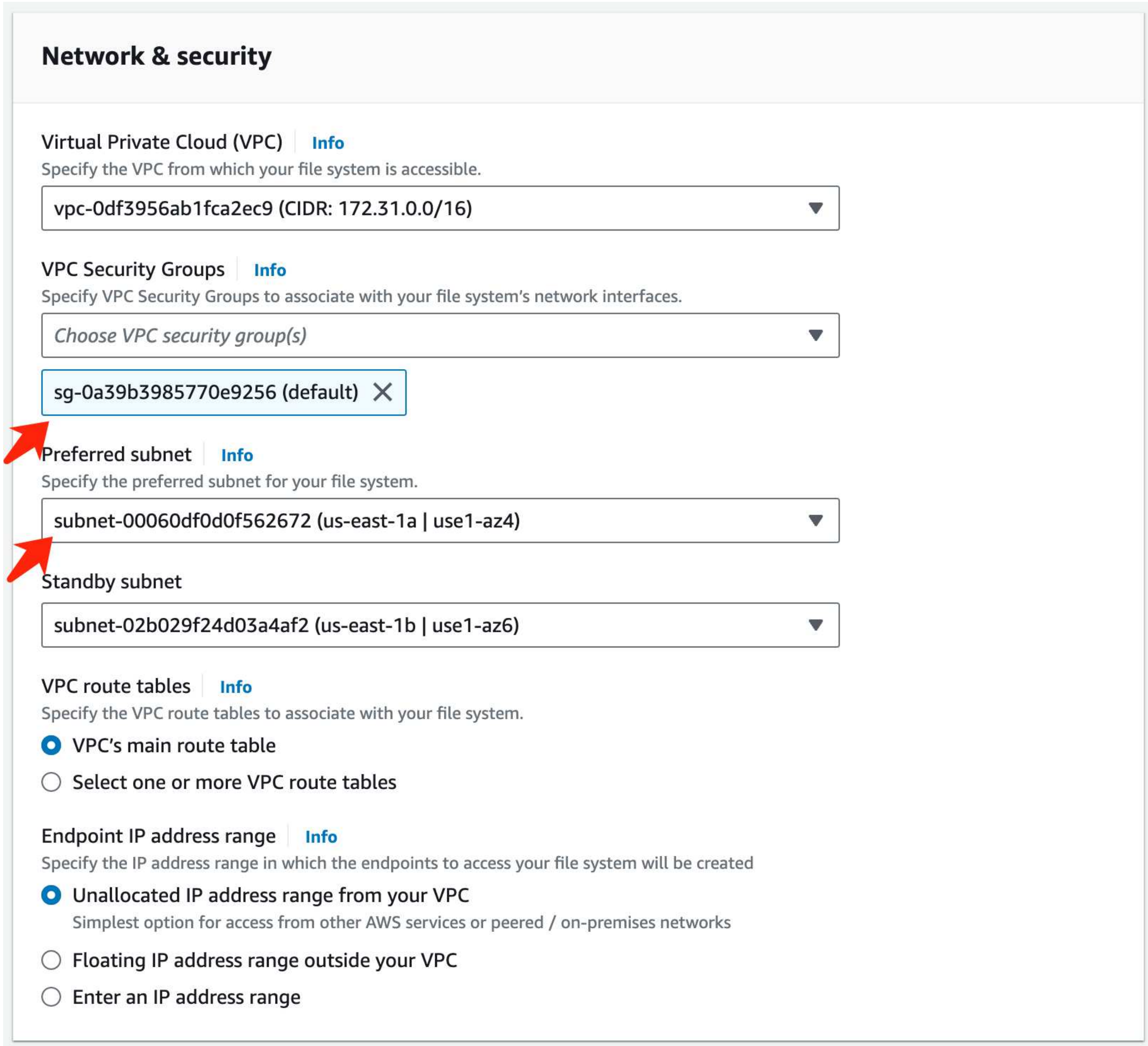
4. In the details configuration page.
    a. Select the **Standard create** option.



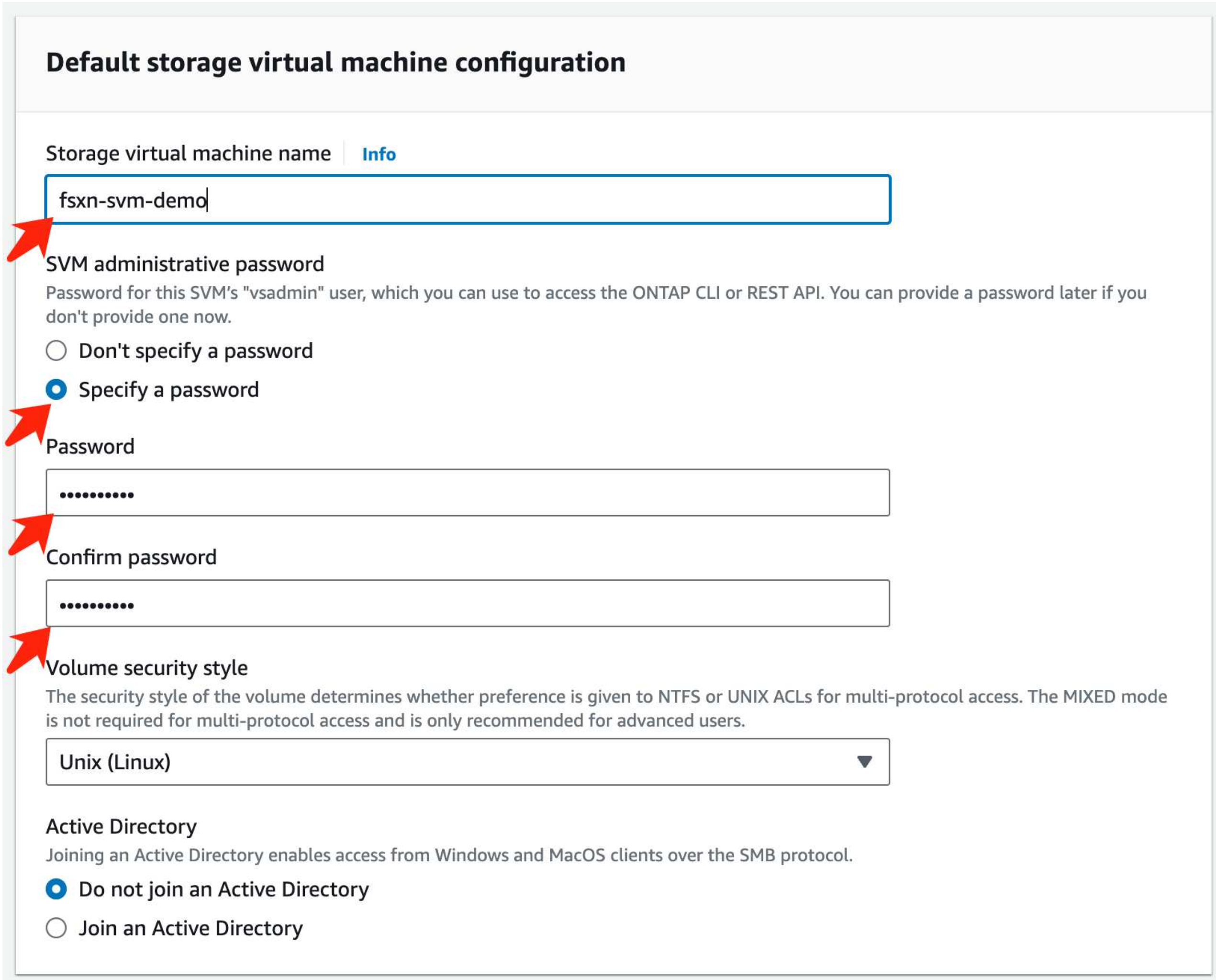
    b. Enter the **File system name** and the **SSD storage capacity**.



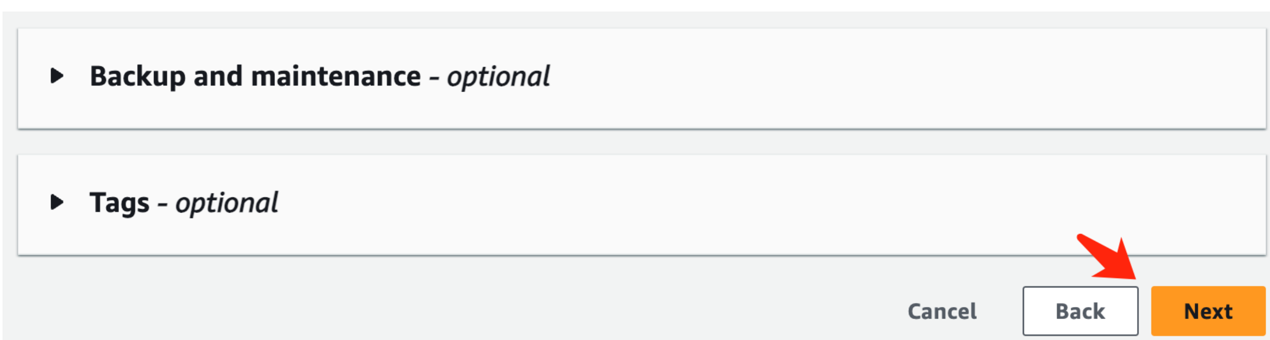
c. Make sure to use the **VPC** and **subnet** same to the **SageMaker Notebook** instance.



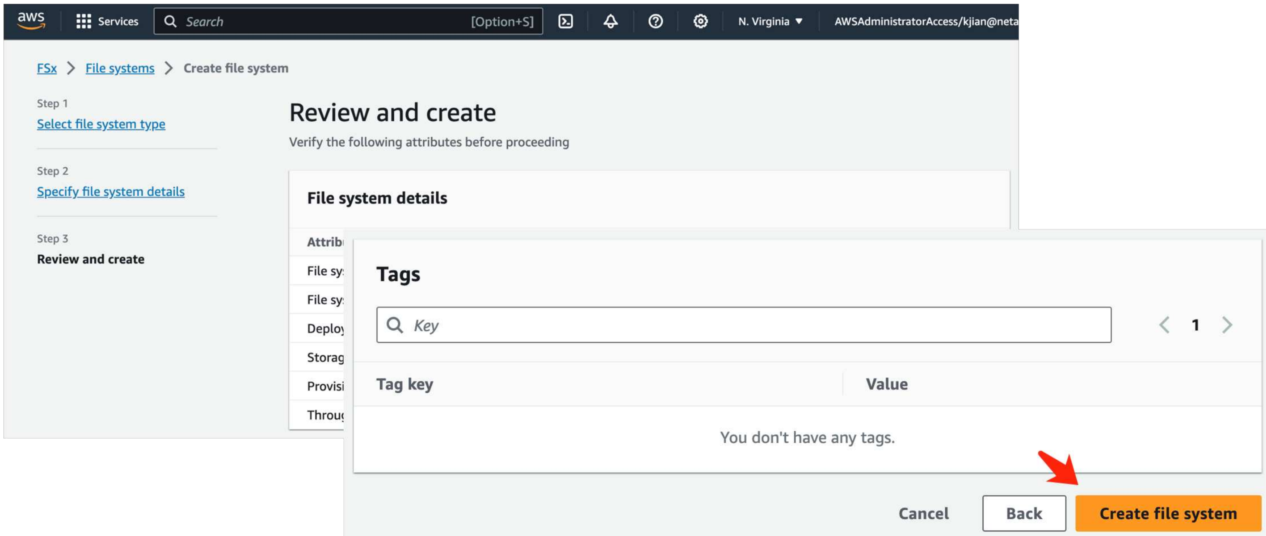
d. Enter the **Storage virtual machine** name and **Specify a password** for your SVM (storage virtual machine).



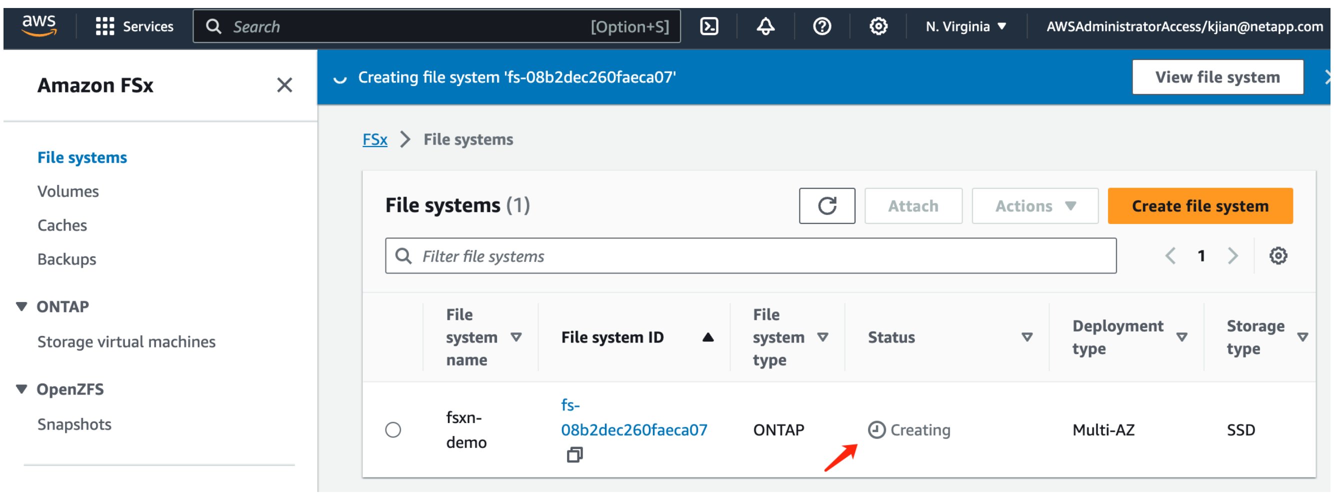
e. Leave other entries default and click the orange button **Next** at the bottom right.



f. Click the orange button **Create file system** at the bottom right of the review page.



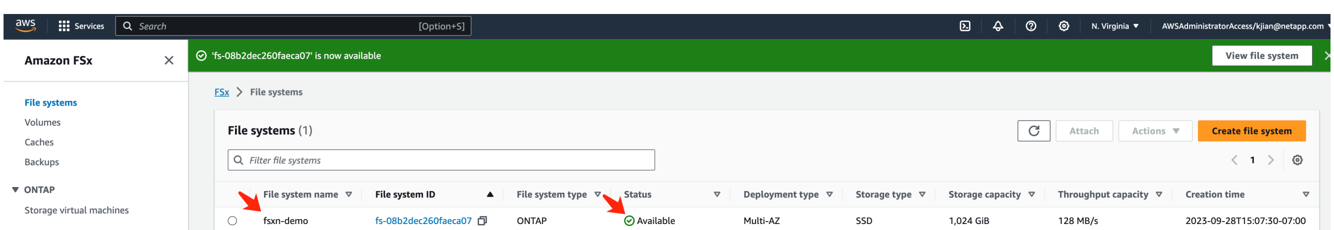
5. It may takes about **20-40 minutes** to spin up the FSx file system.



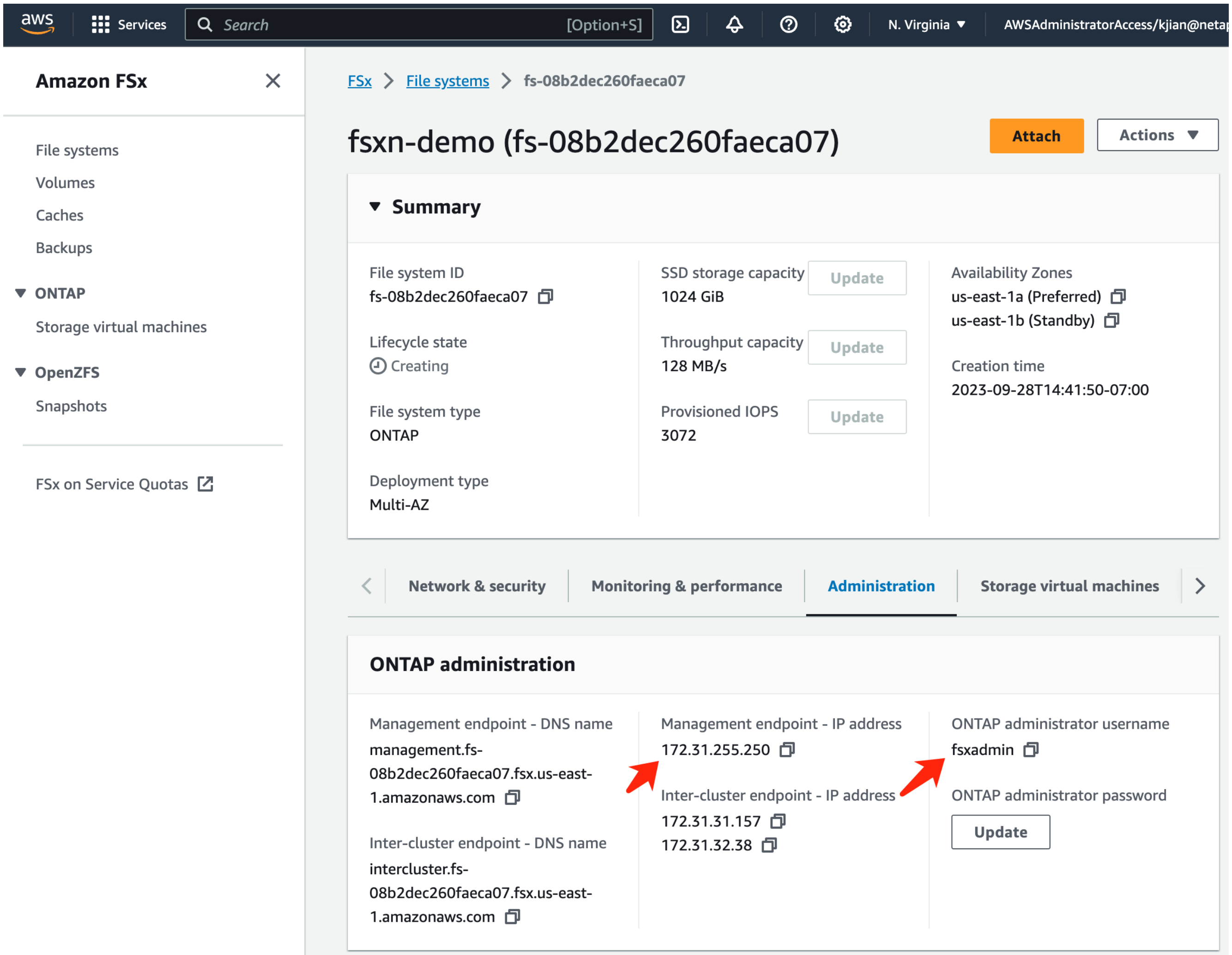
Server Configuration

ONTAP Configuration

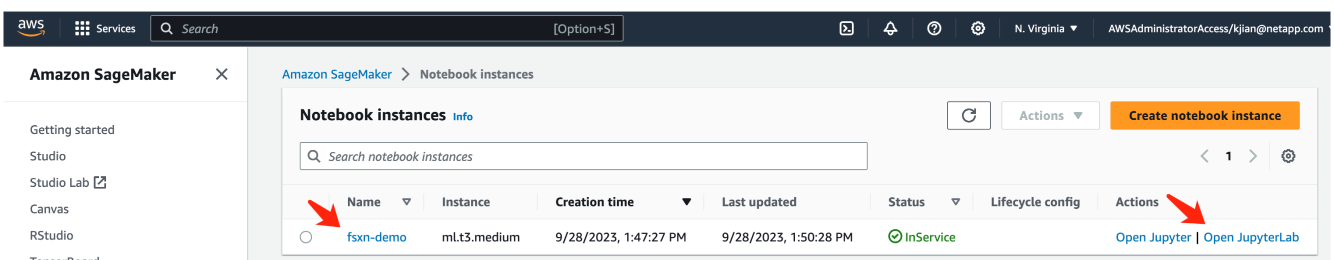
1. Open the created FSx file system. Please make sure the status is **Available**.



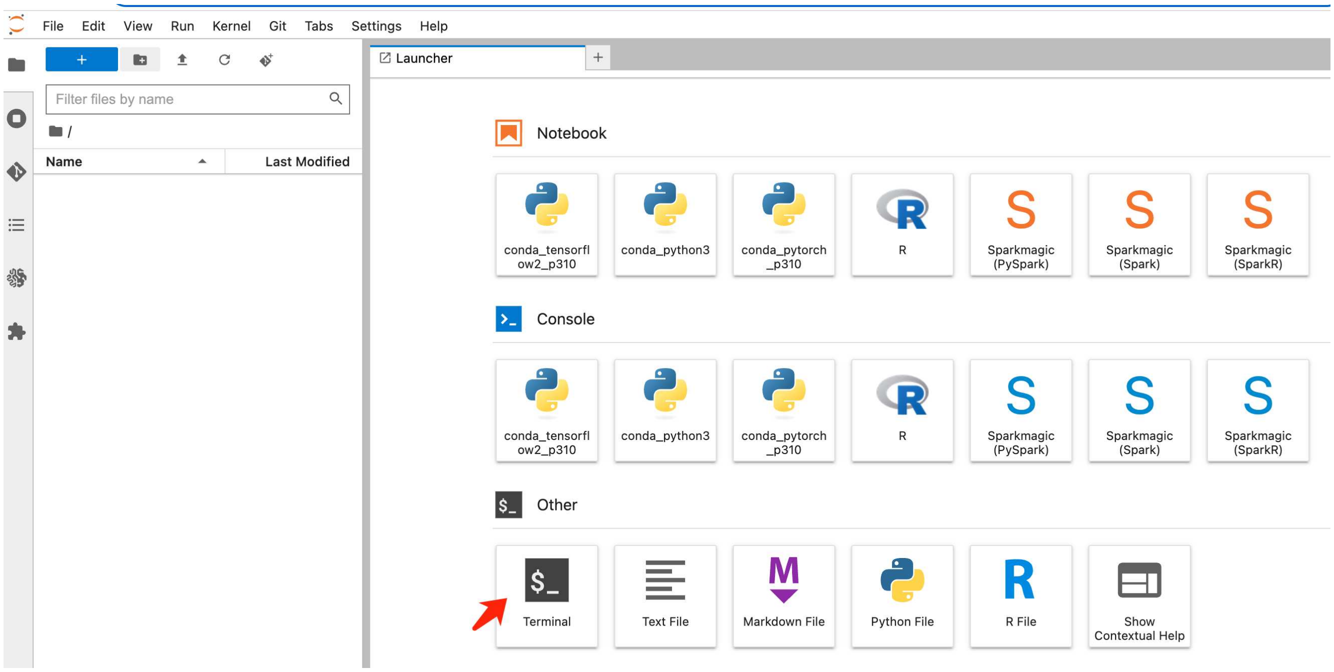
2. Select the **Administration** tab and keep the **Management endpoint - IP address** and **ONTAP administrator username**.



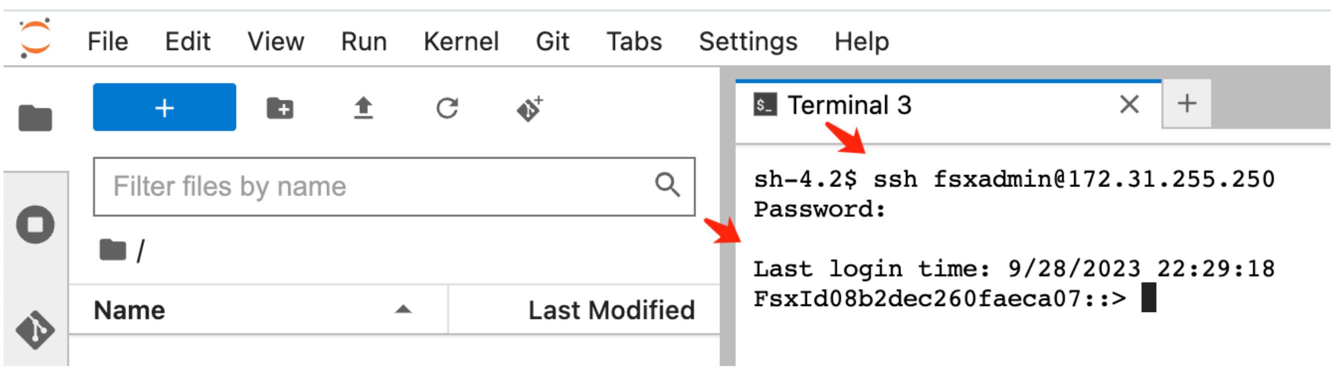
3. Open the created **SageMaker Notebook instance** and click **Open JupyterLab**.



4. In the Jupyter Lab page, open a new **Terminal**.

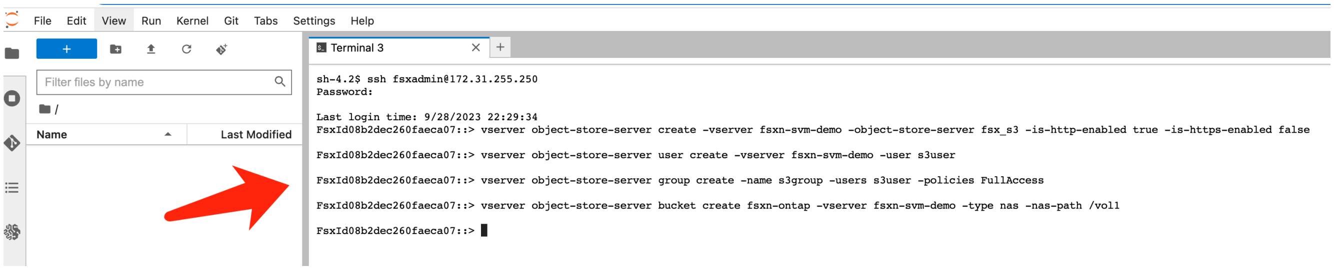


5. Enter the ssh command ssh <admin user name>@<ONTAP server IP> to login to the FSxN ONTAP file system. (The user name and IP address are retrieved from the step 2)
Please use the password used when creating the **Storage virtual machine**.



6. Execute the commands in the following order.
We use **fsxn-ontap** as the name for the **FSxN private S3 bucket name**.
Please use the **storage virtual machine name** for the **-vserver** argument.

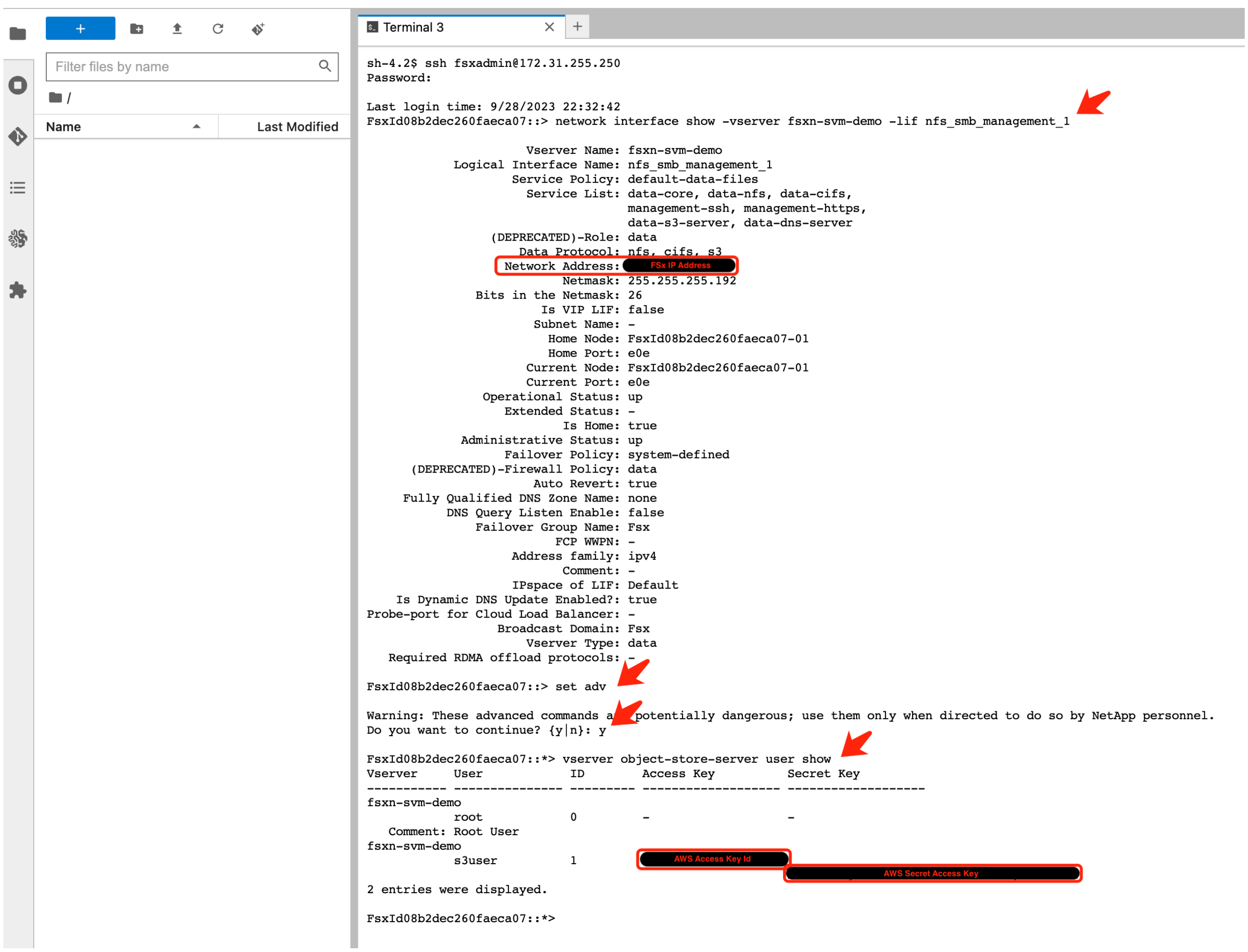
```
vserver object-store-server create -vserver fsxn-svm-demo -object-store
-server fsx_s3 -is-http-enabled true -is-https-enabled false

vserver object-store-server user create -vserver fsxn-svm-demo -user
s3user

vserver object-store-server group create -name s3group -users s3user
-policies FullAccess

vserver object-store-server bucket create fsxn-ontap -vserver fsxn-svm-
demo -type nas -nas-path /vol1
```



7. Execute the below commands to retrieve the endpoint IP and credentials for FSxN private S3.

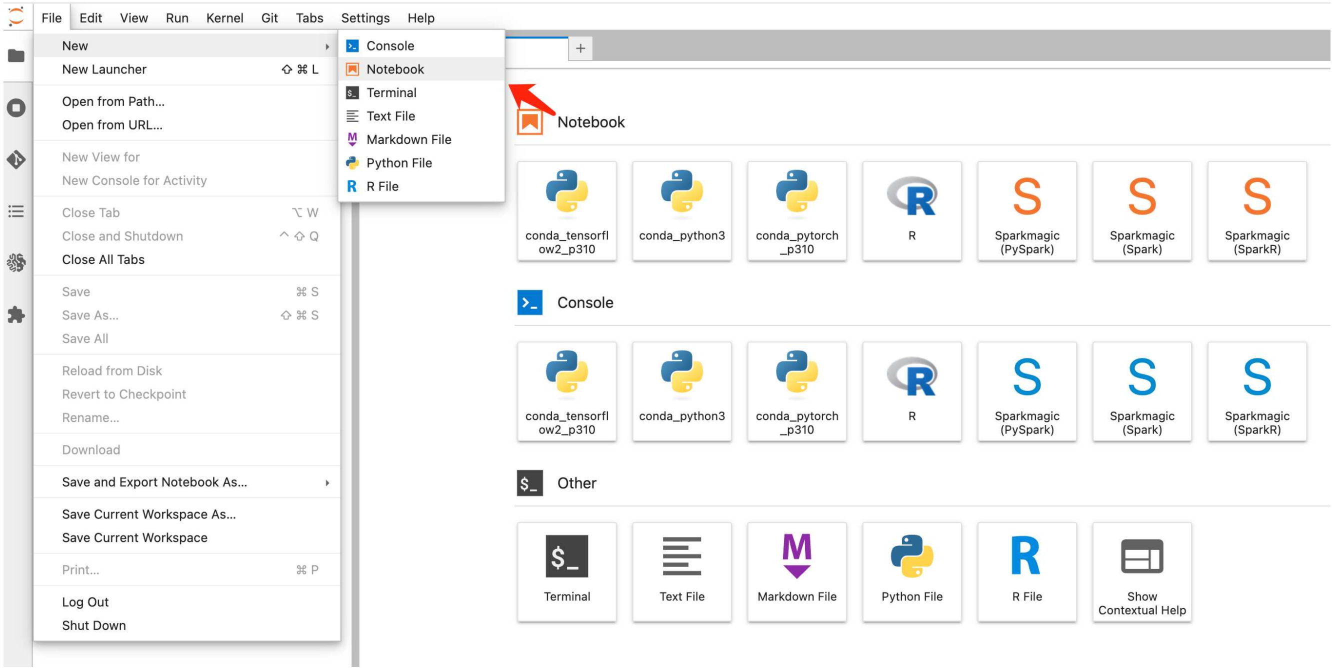
```
network interface show -vserver fsxn-svm-demo -lif nfs_smb_management_1

set adv

vserver object-store-server user show
```

8. Keep the endpoint IP and credential for future use.



Client Configuration

1. In SageMaker Notebook instance, create a new Jupyter notebook.



2. Use the below code as a work around solution to upload files to FSxN private S3 bucket. For a comprehensive code example please refer to this notebook. fsxn_demo.ipynb

```
# Setup configurations
# -------- Manual configurations --------
seed: int = 77                                              # Random seed
bucket_name: str = 'fsxn-ontap'                             # The bucket name in ONTAP
aws_access_key_id = '<Your ONTAP bucket key id>'            # Please get this credential from ONTAP
aws_secret_access_key = '<Your ONTAP bucket access key>'    # Please get this credential from ONTAP
fsx_endpoint_ip: str = '<Your FSxN IP address>'             # Please get this IP address from FSXN
# -------- Manual configurations --------

# Workaround
## Permission patch
!mkdir -p vol1
!sudo mount -t nfs $fsx_endpoint_ip:/vol1 /home/ec2-user/SageMaker/vol1
!sudo chmod 777 /home/ec2-user/SageMaker/vol1

## Authentication for FSxN as a Private S3 Bucket
!aws configure set aws_access_key_id $aws_access_key_id
```

```
!aws configure set aws_secret_access_key $aws_secret_access_key

## Upload file to the FSxN Private S3 Bucket
%%capture
local_file_path: str = <Your local file path>

!aws s3 cp --endpoint-url http://$fsx_endpoint_ip /home/ec2-user
/SageMaker/$local_file_path  s3://$bucket_name/$local_file_path

# Read data from FSxN Private S3 bucket
## Initialize a s3 resource client
import boto3

# Get session info
region_name = boto3.session.Session().region_name

# Initialize Fsxn S3 bucket object
# --- Start integrating SageMaker with FSXN ---
# This is the only code change we need to incorporate SageMaker with
FSXN
s3_client: boto3.client = boto3.resource(
    's3',
    region_name=region_name,
    aws_access_key_id=aws_access_key_id,
    aws_secret_access_key=aws_secret_access_key,
    use_ssl=False,
    endpoint_url=f'http://{fsx_endpoint_ip}',
    config=boto3.session.Config(
        signature_version='s3v4',
        s3={'addressing_style': 'path'}
    )
)
# --- End integrating SageMaker with FSXN ---

## Read file byte content
bucket = s3_client.Bucket(bucket_name)

binary_data = bucket.Object(data.filename).get()['Body']
```

This concludes the integration between FSxN and the SageMaker instance.

Useful debugging checklist

- Ensure that the SageMaker Notebook instance and FSxN file system are in the same VPC.
- Remember to run the **set dev** command on ONTAP to set the privilege level to **dev**.

FAQ (As of Sep 27, 2023)

Q: Why am I getting the error "**An error occurred (NotImplemented) when calling the CreateMultipartUpload operation: The s3 command you requested is not implemented**" when uploading files to FSxN?

A: As a private S3 bucket, FSxN supports uploading files up to 100MB. When using the S3 protocol, files larger than 100MB are divided into 100MB chunks, and the 'CreateMultipartUpload' function is called. However, the current implementation of FSxN private S3 does not support this function.

Q: Why am I getting the error "**An error occurred (AccessDenied) when calling the PutObject operations: Access Denied**" when uploading files to FSxN?

A: To access the FSxN private S3 bucket from a SageMaker Notebook instance, switch the AWS credentials to the FSxN credentials. However, granting write permission to the instance requires a workaround solution that involves mounting the bucket and running the 'chmod' shell command to change the permissions.

Q: How can I integrate the FSxN private S3 bucket with other SageMaker ML services?

A: Unfortunately, the SageMaker services SDK does not provide a way to specify the endpoint for the private S3 bucket. As a result, FSxN S3 is not compatible with SageMaker services such as Sagemaker Data Wrangler, Sagemaker Clarify, Sagemaker Glue, Sagemaker Athena, Sagemaker AutoML, and others.

Part 2 - Leveraging AWS FSx for NetApp ONTAP (FSxN) as a Data Source for Model Training in SageMaker

Author(s):

Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

Introduction

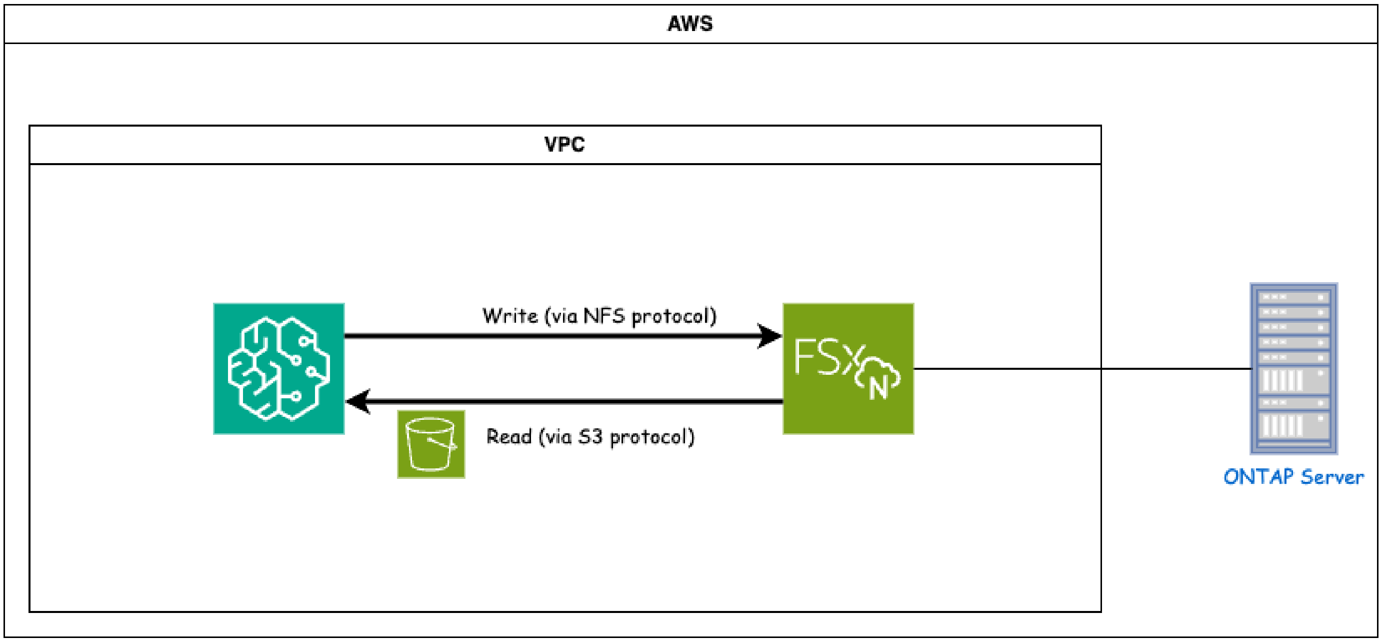
This tutorial offers a practical example of a computer vision classification project, providing hands-on experience in building ML models that utilize FSxN as the data source within the SageMaker environment. The project focuses on using PyTorch, a deep learning framework, to classify tire quality based on tire images. It emphasizes the development of machine learning models using FSxN as the data source in Amazon SageMaker.

What is FSxN

Amazon FSx for NetApp ONTAP is indeed a fully managed storage solution offered by AWS. It leverages NetApp’s ONTAP file system to provide reliable and high-performance storage. With support for protocols like NFS, SMB, and iSCSI, it allows seamless access from different compute instances and containers. The service is designed to deliver exceptional performance, ensuring fast and efficient data operations. It also offers high availability and durability, ensuring that your data remains accessible and protected. Additionally, the storage capacity of Amazon FSx for NetApp ONTAP is scalable, allowing you to easily adjust it according to your needs.

Prerequisite

Network Environment



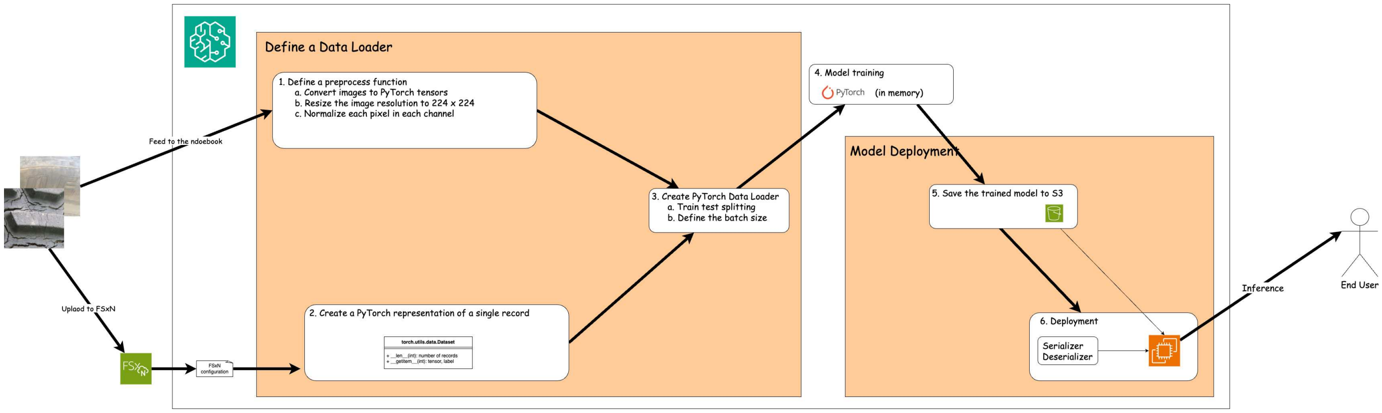
FSxN (Amazon FSx for NetApp ONTAP) is an AWS storage service. It includes a file system running on the NetApp ONTAP system and an AWS-managed system virtual machine (SVM) that connects to it. In the provided diagram, the NetApp ONTAP server managed by AWS is located outside the VPC. The SVM serves as the intermediary between SageMaker and the NetApp ONTAP system, receiving operation requests from SageMaker and forwarding them to the underlying storage. To access FSxN, SageMaker must be placed within the same VPC as the FSxN deployment. This configuration ensures communication and data access between SageMaker and FSxN.

Data Access

In real-world scenarios, data scientists typically utilize the existing data stored in FSxN to build their machine learning models. However, for demonstration purposes, since the FSxN file system is initially empty after creation, it is necessary to manually upload the training data. This can be achieved by mounting FSxN as a volume to SageMaker. Once the file system is successfully mounted, you can upload your dataset to the mounted location, making it accessible for training your models within the SageMaker environment. This approach allows you to leverage the storage capacity and capabilities of FSxN while working with SageMaker for model development and training.

The data reading process involves configuring FSxN as a private S3 bucket. To learn the detailed configuration instructions, please refer to Part 1 - Integrating AWS FSx for NetApp ONTAP (FSxN) as a private S3 bucket into AWS SageMaker

Integration Overview



The workflow of using training data in FSxN to build a deep learning model in SageMaker can be summarized into three main steps: data loader definition, model training, and deployment. At a high level, these steps form the foundation of an MLOps pipeline. However, each step involves several detailed sub-steps for a comprehensive implementation. These sub-steps encompass various tasks such as data preprocessing, dataset splitting, model configuration, hyperparameter tuning, model evaluation, and model deployment. These steps ensure a thorough and effective process for building and deploying deep learning models using training data from FSxN within the SageMaker environment.

Step-by-Step Integration

Data Loader

In order to train a PyTorch deep learning network with data, a data loader is created to facilitate the feeding of data. The data loader not only defines the batch size but also determines the procedure for reading and preprocessing each record within the batch. By configuring the data loader, we can handle the processing of data in batches, enabling training of the deep learning network.

The data loader consists of 3 parts.

Preprocessing Function

```
from torchvision import transforms

preprocess = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((224,224)),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

The above code snippet demonstrates the definition of image preprocessing transformations using the **torchvision.transforms** module. In this turtorial, the preprocess object is created to apply a series of transformations. Firstly, the **ToTensor()** transformation converts the image into a tensor representation. Subsequently, the **Resize 224,224** transformation resizes the image to a fixed size of 224x224 pixels. Finally, the **Normalize()** transformation normalizes the tensor values by subtracting the mean and dividing by the standard deviation along each channel. The mean and standard deviation values used for normalization are commonly employed in pre-trained neural network models. Overall, this code prepares the image data for further processing or input into a pre-trained model by converting it to a tensor, resizing it, and normalizing the pixel values.

The PyTorch Dataset Class

```
import torch
from io import BytesIO
from PIL import Image


class FSxNImageDataset(torch.utils.data.Dataset):
    def __init__(self, bucket, prefix='', preprocess=None):
        self.image_keys = [
            s3_obj.key
            for s3_obj in list(bucket.objects.filter(Prefix=prefix).all())
        ]
        self.preprocess = preprocess

    def __len__(self):
        return len(self.image_keys)

    def __getitem__(self, index):
        key = self.image_keys[index]
        response = bucket.Object(key)

        label = 1 if key[13:].startswith('defective') else 0

        image_bytes = response.get()['Body'].read()
        image = Image.open(BytesIO(image_bytes))
        if image.mode == 'L':
            image = image.convert('RGB')

        if self.preprocess is not None:
            image = self.preprocess(image)
        return image, label
```

This class provides functionality to obtain the total number of records in the dataset and defines the method for reading data for each record. Within the ***getitem*** function, the code utilizes the boto3 S3 bucket object to retrieve the binary data from FSxN. The code style for accessing data from FSxN is similar to reading data from Amazon S3. The subsequent explanation delves into the creation process of the private S3 object **bucket**.

FSxN as a private S3 repository

```
seed = 77                                                       # Random seed
bucket_name = '<Your ONTAP bucket name>'                        # The bucket
name in ONTAP
aws_access_key_id = '<Your ONTAP bucket key id>'                # Please get
this credential from ONTAP
aws_secret_access_key = '<Your ONTAP bucket access key>'        # Please get
this credential from ONTAP
fsx_endpoint_ip = '<Your FSxN IP address>'                      # Please get
this IP address from FSXN
```

```
import boto3

# Get session info
region_name = boto3.session.Session().region_name

# Initialize Fsxn S3 bucket object
# --- Start integrating SageMaker with FSXN ---
# This is the only code change we need to incorporate SageMaker with FSXN
s3_client: boto3.client = boto3.resource(
    's3',
    region_name=region_name,
    aws_access_key_id=aws_access_key_id,
    aws_secret_access_key=aws_secret_access_key,
    use_ssl=False,
    endpoint_url=f'http://{fsx_endpoint_ip}',
    config=boto3.session.Config(
        signature_version='s3v4',
        s3={'addressing_style': 'path'}
    )
)
# s3_client = boto3.resource('s3')
bucket = s3_client.Bucket(bucket_name)
# --- End integrating SageMaker with FSXN ---
```

To read data from FSxN in SageMaker, a handler is created that points to the FSxN storage using the S3 protocol. This allows FSxN to be treated as a private S3 bucket. The handler configuration includes specifying the IP address of the FSxN SVM, the bucket name, and the necessary credentials. For a comprehensive explanation on obtaining these configuration items, please refer to the document at Part 1 - Integrating AWS FSx for NetApp ONTAP (FSxN) as a private S3 bucket into AWS SageMaker.

In the example mentioned above, the bucket object is used to instantiate the PyTorch dataset object. The dataset object will be further explained in the subsequent section.

The PyTorch Data Loader

```
from torch.utils.data import DataLoader
torch.manual_seed(seed)

# 1. Hyperparameters
batch_size = 64

# 2. Preparing for the dataset
dataset = FSxNImageDataset(bucket, 'dataset/tyre', preprocess=preprocess)

train, test = torch.utils.data.random_split(dataset, [1500, 356])

data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

In the example provided, a batch size of 64 is specified, indicating that each batch will contain 64 records. By combining the PyTorch **Dataset** class, the preprocessing function, and the training batch size, we obtain the data loader for training. This data loader facilitates the process of iterating through the dataset in batches during the training phase.

Model Training

```
from torch import nn


class TyreQualityClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3,32,(3,3)),
            nn.ReLU(),
            nn.Conv2d(32,32,(3,3)),
            nn.ReLU(),
            nn.Conv2d(32,64,(3,3)),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(64*(224-6)*(224-6),2)
        )
    def forward(self, x):
        return self.model(x)
```

```
import datetime

num_epochs = 2
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = TyreQualityClassifier()
fn_loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)


model.to(device)
for epoch in range(num_epochs):
    for idx, (X, y) in enumerate(data_loader):
        X = X.to(device)
        y = y.to(device)

        y_hat = model(X)

        loss = fn_loss(y_hat, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        current_time = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
        print(f"Current Time: {current_time} - Epoch [{epoch+1}/
{num_epochs}]- Batch [{idx + 1}] - Loss: {loss}", end='\r')
```

This code implements a standard PyTorch training process. It defines a neural network model called **TyreQualityClassifier** using convolutional layers and a linear layer to classify tire quality. The training loop iterates over data batches, computes the loss, and updates the model's parameters using backpropagation and optimization. Additionally, it prints the current time, epoch, batch, and loss for monitoring purposes.

Model Deployment

Deployment

```python
import io
import os
import tarfile
import sagemaker

# 1. Save the PyTorch model to memory
buffer_model = io.BytesIO()
traced_model = torch.jit.script(model)
torch.jit.save(traced_model, buffer_model)

# 2. Upload to AWS S3
sagemaker_session = sagemaker.Session()
bucket_name_default = sagemaker_session.default_bucket()
model_name = f'tyre_quality_classifier.pth'

# 2.1. Zip PyTorch model into tar.gz file
buffer_zip = io.BytesIO()
with tarfile.open(fileobj=buffer_zip, mode="w:gz") as tar:
    # Add PyTorch pt file
    file_name = os.path.basename(model_name)
    file_name_with_extension = os.path.split(file_name)[-1]
    tarinfo = tarfile.TarInfo(file_name_with_extension)
    tarinfo.size = len(buffer_model.getbuffer())
    buffer_model.seek(0)
    tar.addfile(tarinfo, buffer_model)

# 2.2. Upload the tar.gz file to S3 bucket
buffer_zip.seek(0)
boto3.resource('s3') \
    .Bucket(bucket_name_default) \
    .Object(f'pytorch/{model_name}.tar.gz') \
    .put(Body=buffer_zip.getvalue())
```

The code saves the PyTorch model to **Amazon S3** because SageMaker requires the model to be stored in S3 for deployment. By uploading the model to **Amazon S3**, it becomes accessible to SageMaker, allowing for the deployment and inference on the deployed model.

```python
import time
from sagemaker.pytorch import PyTorchModel
from sagemaker.predictor import Predictor
from sagemaker.serializers import IdentitySerializer
from sagemaker.deserializers import JSONDeserializer


class TyreQualitySerializer(IdentitySerializer):
```

```
    CONTENT_TYPE = 'application/x-torch'

    def serialize(self, data):
        transformed_image = preprocess(data)
        tensor_image = torch.Tensor(transformed_image)

        serialized_data = io.BytesIO()
        torch.save(tensor_image, serialized_data)
        serialized_data.seek(0)
        serialized_data = serialized_data.read()

        return serialized_data


class TyreQualityPredictor(Predictor):
    def __init__(self, endpoint_name, sagemaker_session):
        super().__init__(
            endpoint_name,
            sagemaker_session=sagemaker_session,
            serializer=TyreQualitySerializer(),
            deserializer=JSONDeserializer(),
        )

sagemaker_model = PyTorchModel(
    model_data=f's3://{bucket_name_default}/pytorch/{model_name}.tar.gz',
    role=sagemaker.get_execution_role(),
    framework_version='2.0.1',
    py_version='py310',
    predictor_cls=TyreQualityPredictor,
    entry_point='inference.py',
    source_dir='code',
)

timestamp = int(time.time())
pytorch_endpoint_name = '{}-{}-{}'.format('tyre-quality-classifier', 'pt',
timestamp)
sagemaker_predictor = sagemaker_model.deploy(
    initial_instance_count=1,
    instance_type='ml.p3.2xlarge',
    endpoint_name=pytorch_endpoint_name
)
```

This code facilitates the deployment of a PyTorch model on SageMaker. It defines a custom serializer, **TyreQualitySerializer**, which preprocesses and serializes input data as a PyTorch tensor. The **TyreQualityPredictor** class is a custom predictor that utilizes the defined serializer and a **JSONDeserializer**. The code also creates a **PyTorchModel** object to specify the model's S3 location, IAM role, framework version, and entry point for inference. The code generates a timestamp and constructs an endpoint name based on the model and timestamp. Finally, the model is deployed using the deploy method, specifying the instance count, instance type, and generated endpoint name. This enables the PyTorch model to be deployed and accessible for inference on SageMaker.

Inference

```
image_object = list(bucket.objects.filter('dataset/tyre'))[0].get()
image_bytes = image_object['Body'].read()

with Image.open(with Image.open(BytesIO(image_bytes)) as image:
    predicted_classes = sagemaker_predictor.predict(image)

    print(predicted_classes)
```

This is the example of using the deployed endpoint to do the inference.

Part 3 - Building A Simplified MLOps Pipeline (CI/CT/CD)

Author(s):

Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

Introduction

In this tutorial, you will learn how to leverage various AWS services to construct a simple MLOps pipeline that encompasses Continuous Integration (CI), Continuous Training (CT), and Continuous Deployment (CD). Unlike traditional DevOps pipelines, MLOps requires additional considerations to complete the operational cycle. By following this tutorial, you will gain insights into incorporating CT into the MLOps loop, enabling continuous training of your models and seamless deployment for inference. The tutorial will guide you through the process of utilizing AWS services to establish this end-to-end MLOps pipeline.

Manifest

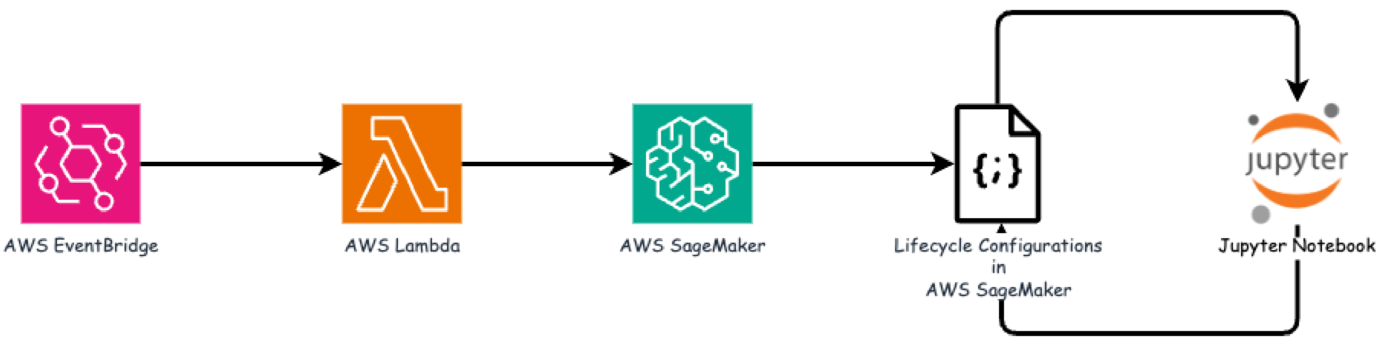
Functionality	Name	Comment
Data storage	AWS FSxN	Refer to Part 1 - Integrating AWS FSx for NetApp ONTAP (FSxN) as a private S3 bucket into AWS SageMaker.
Data science IDE	AWS SageMaker	This tutorial is based on the Jupyter notebook presented in Part 2 - Leveraging AWS FSx for NetApp ONTAP (FSxN) as a Data Source for Model Training in SageMaker.
Function to trigger the MLOps pipeline	AWS Lambda function	-
Cron job trigger	AWS EventBridge	-
Deep learning framework	PyTorch	-
AWS Python SDK	boto3	-

Functionality	Name	Comment
Programming Language	Python	v3.10

Prerequisite

- An pre-configured FSxN file system. This tutorial utilizes data stored in FSxN for the training process.
- A **SageMaker Notebook instance** that is configured to share the same VPC as the FSxN file system mentioned above.
- Before triggering the **AWS Lambda function**, ensure that the **SageMaker Notebook instance** is in **stopped** status.
- The **ml.g4dn.xlarge** instance type is required to leverage the GPU acceleration necessary for the computations of deep neural networks.

Architecture



This MLOps pipeline is a practical implementation that utilizes a cron job to trigger a serverless function, which in turn executes an AWS service registered with a lifecycle callback function. The **AWS EventBridge** acts as the cron job. It periodically invokes an **AWS Lambda function** responsible for retraining and redeploying the model. This process involves spinning up the **AWS SageMaker Notebook** instance to perform the necessary tasks.

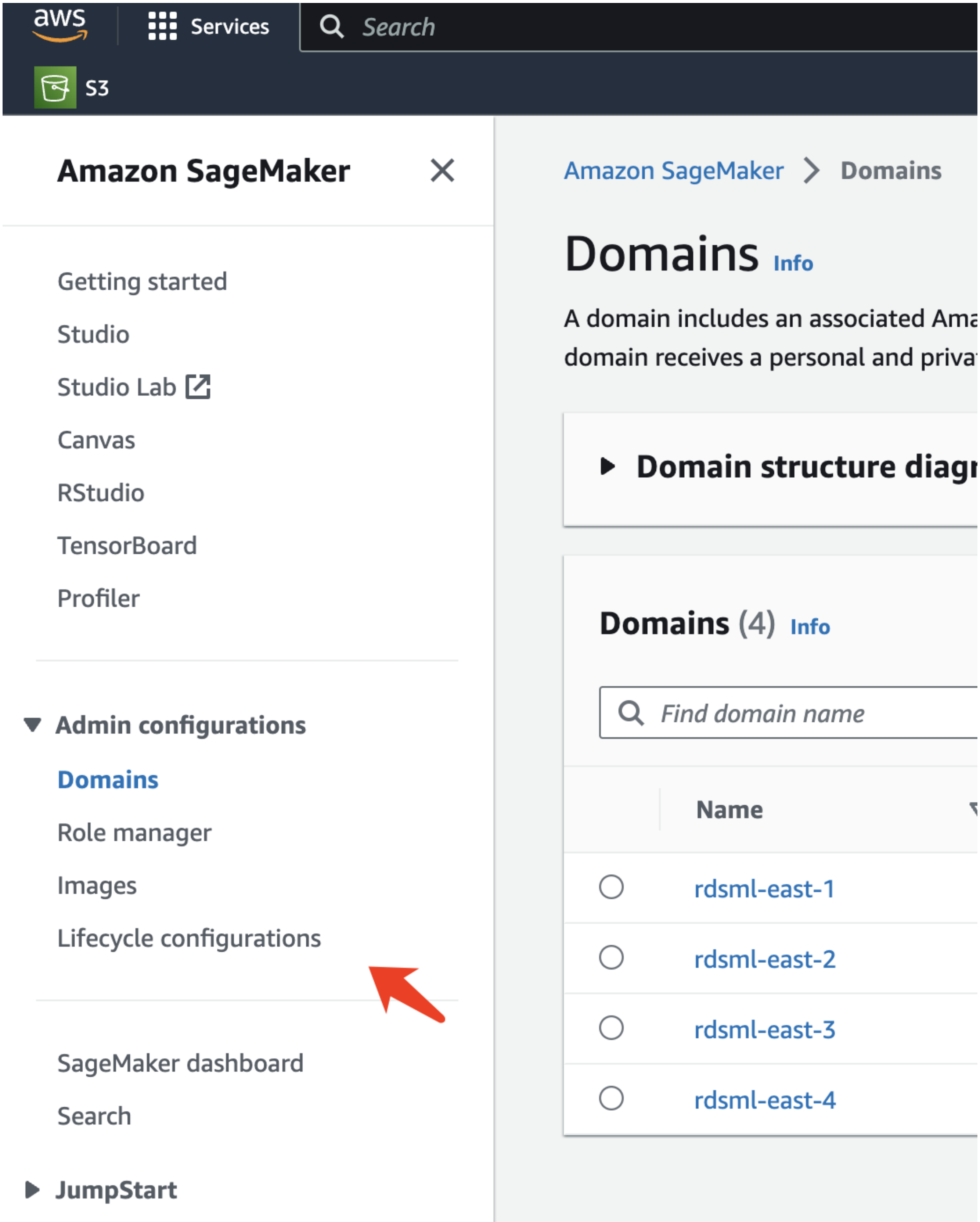
Step-by-Step Configuration

Lifecycle configurations

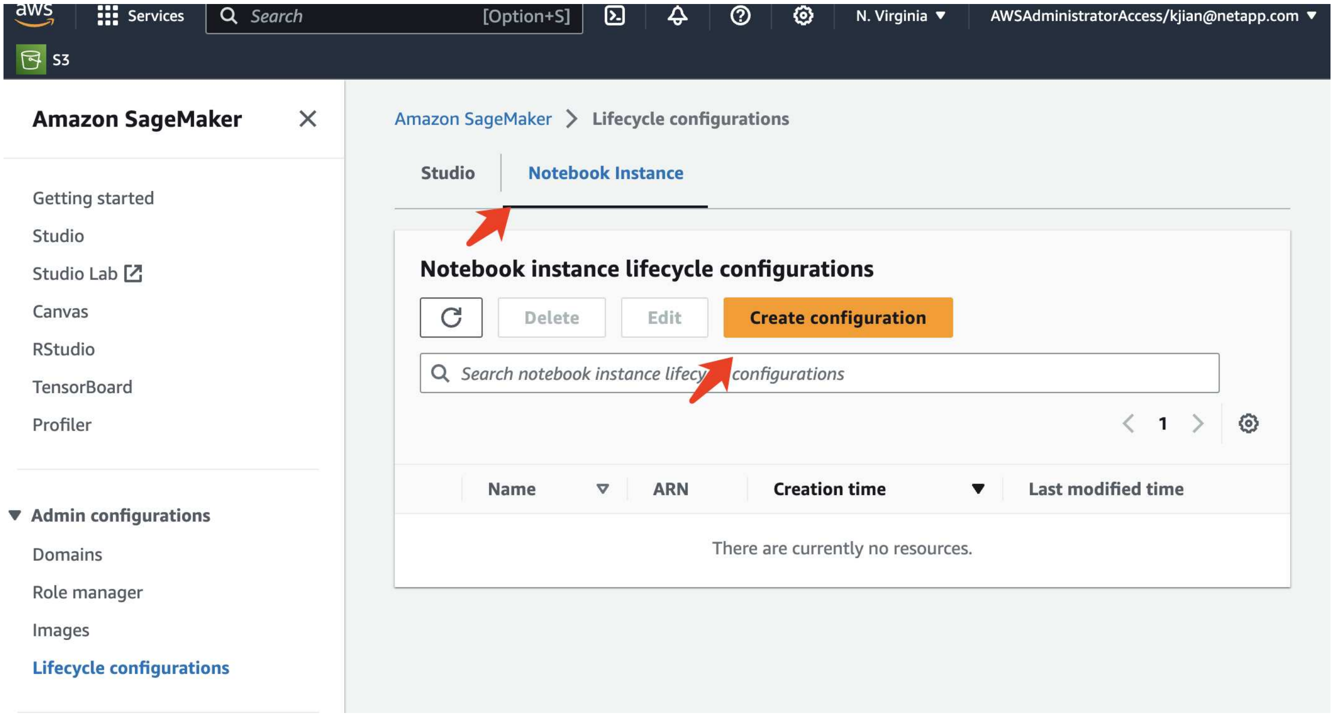
To configure the lifecycle callback function for the AWS SageMaker Notebook instance, you would utilize **Lifecycle configurations**. This service allow you to define the necessary actions to be performed during when spinning up the notebook instance. Specifically, a shell script can be implemented within the **Lifecycle configurations** to automatically shut down the notebook instance once the training and deployment processes are completed. This is a required configuration as the cost is one of the major consideration in MLOps.

It's important to note that the configuration for **Lifecycle configurations** needs to be set up in advance. Therefore, it is recommended to prioritize configuring this aspect before proceeding with the other MLOps pipeline setup.

1. To set up a Lifecycle configurations, open the **Sagemaker** panel and navigate to **Lifecycle configurations** under the section **Admin configurations**.



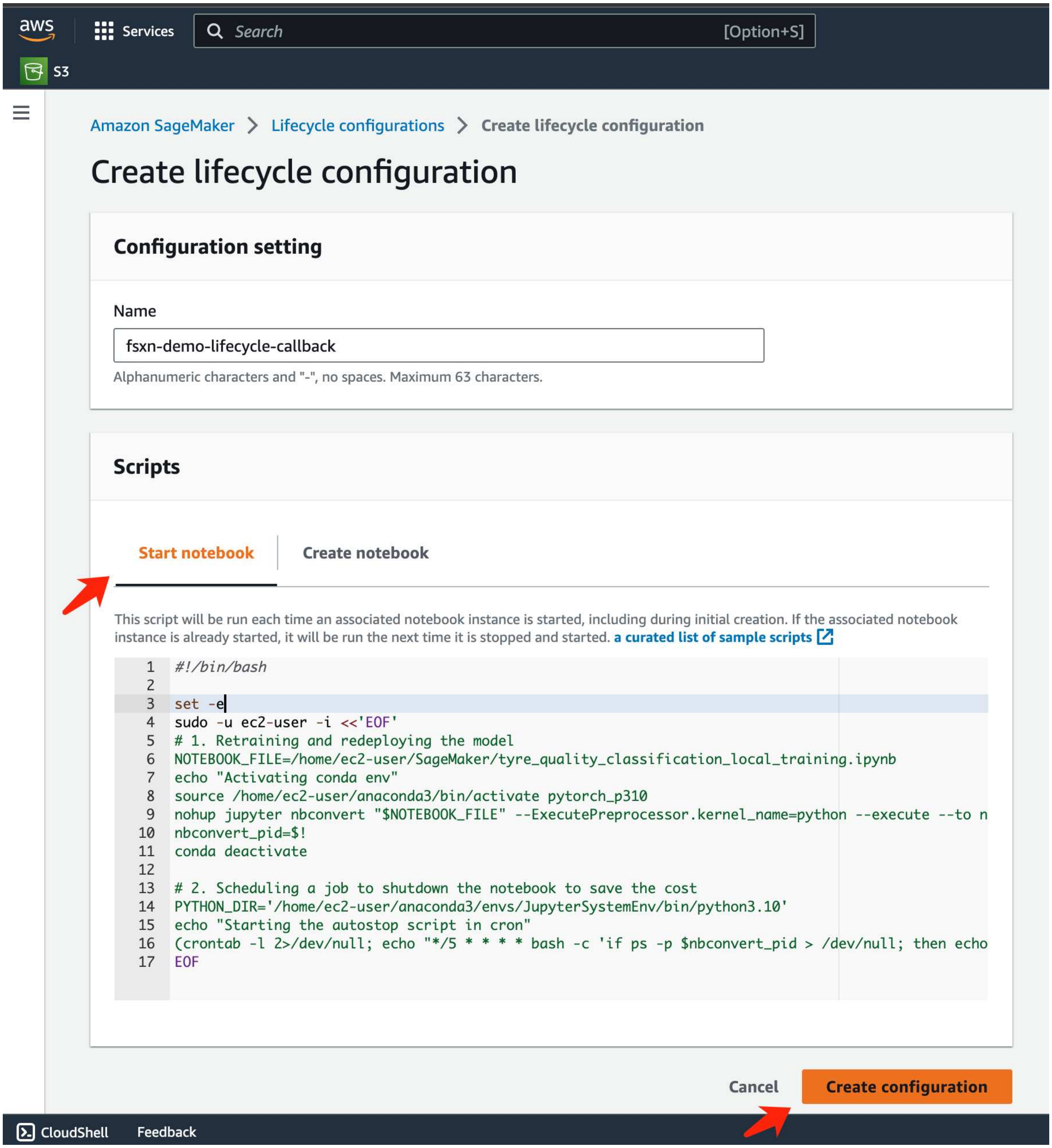
2. Select the **Notebook Instance** tab and click the **Create configuration** button



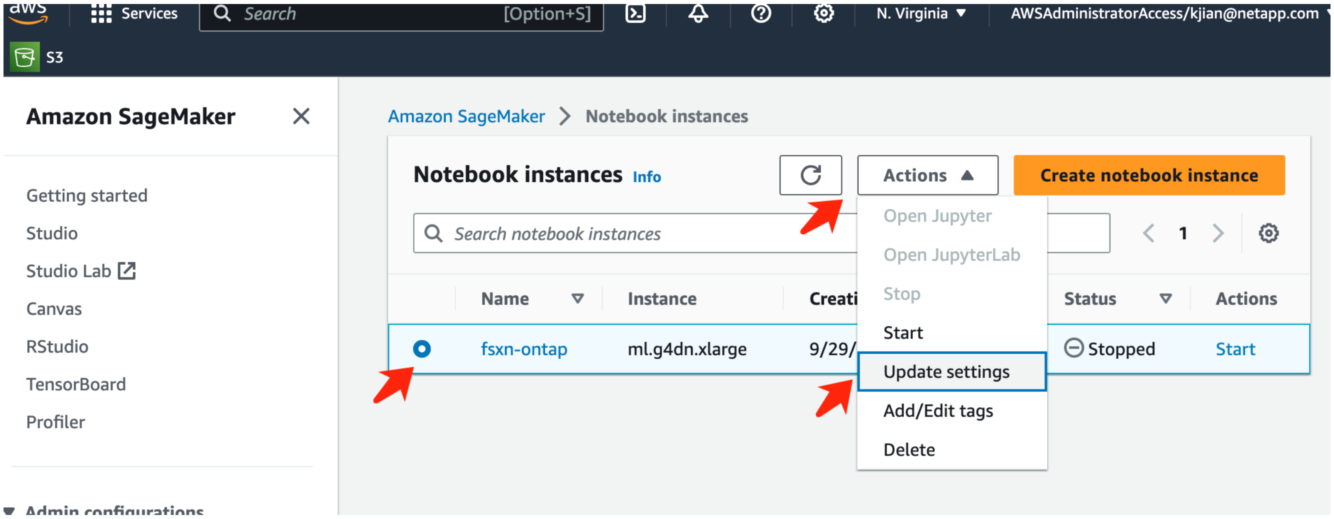
3. Paste the below code to the entry area.

```
#!/bin/bash

set -e
sudo -u ec2-user -i <<'EOF'
# 1. Retraining and redeploying the model
NOTEBOOK_FILE=/home/ec2-
user/SageMaker/tyre_quality_classification_local_training.ipynb
echo "Activating conda env"
source /home/ec2-user/anaconda3/bin/activate pytorch_p310
nohup jupyter nbconvert "$NOTEBOOK_FILE"
--ExecutePreprocessor.kernel_name=python --execute --to notebook &
nbconvert_pid=$!
conda deactivate

# 2. Scheduling a job to shutdown the notebook to save the cost
PYTHON_DIR='/home/ec2-
user/anaconda3/envs/JupyterSystemEnv/bin/python3.10'
echo "Starting the autostop script in cron"
(crontab -l 2>/dev/null; echo "*/5 * * * * bash -c 'if ps -p
$nbconvert_pid > /dev/null; then echo \"Notebook is still running.\" >>
/var/log/jupyter.log; else echo \"Notebook execution completed.\" >>
/var/log/jupyter.log; $PYTHON_DIR -c \"import boto3;boto3.client(
\'sagemaker\').stop_notebook_instance(NotebookInstanceName=get_notebook_
name())\" >> /var/log/jupyter.log; fi'") | crontab -
EOF
```

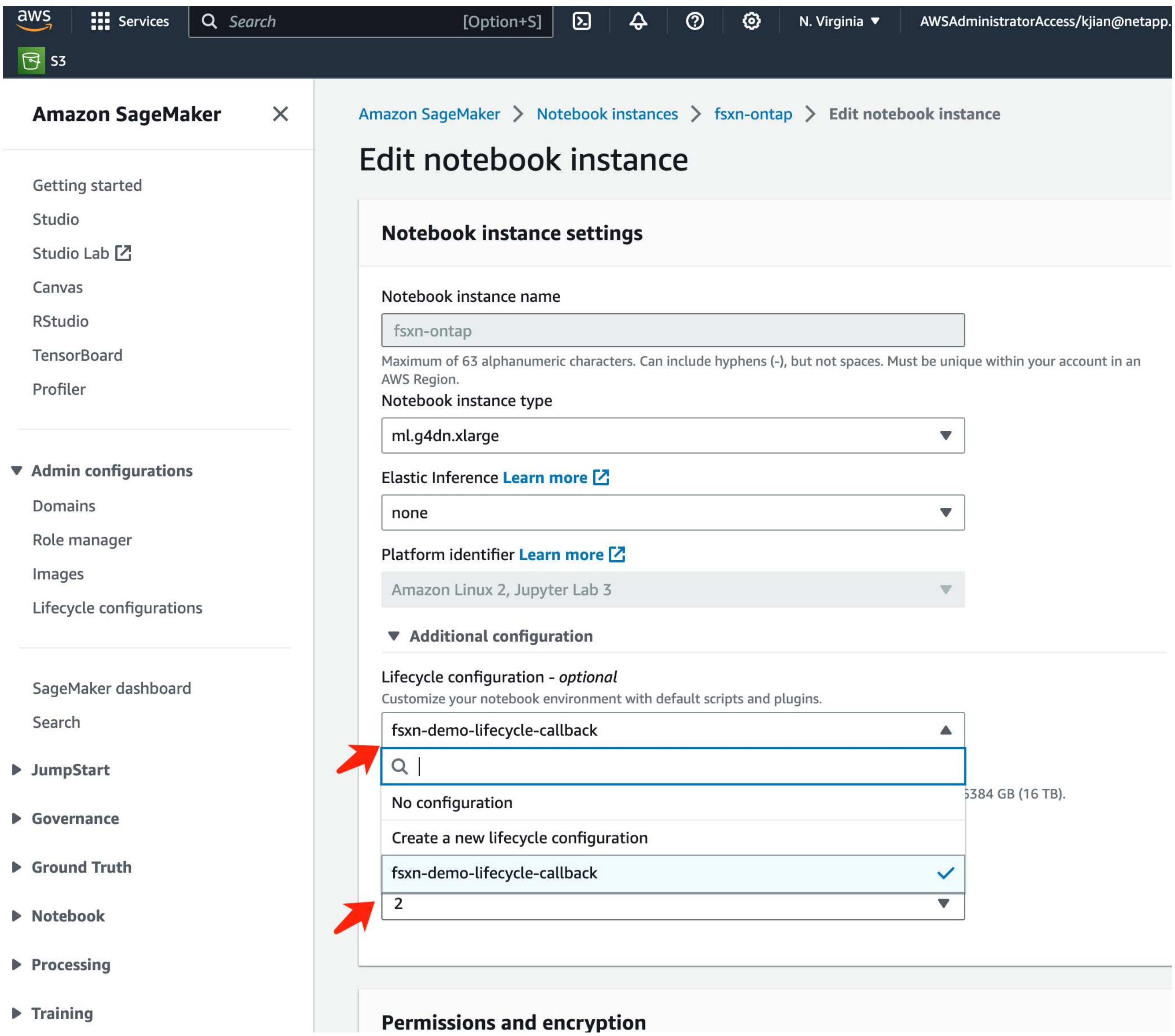
4. This script executes the Jupyter Notebook, which handles the retraining and redeployment of the model for inference. After the execution is complete, the notebook will automatically shut down within 5 minutes. To learn more about the problem statement and the code implementation, please refer to Part 2 - Leveraging AWS FSx for NetApp ONTAP (FSxN) as a Data Source for Model Training in SageMaker.



5. After the creation, navigate to Notebook instances, select the target instance, and click **Update settings** under Actions dropdown.



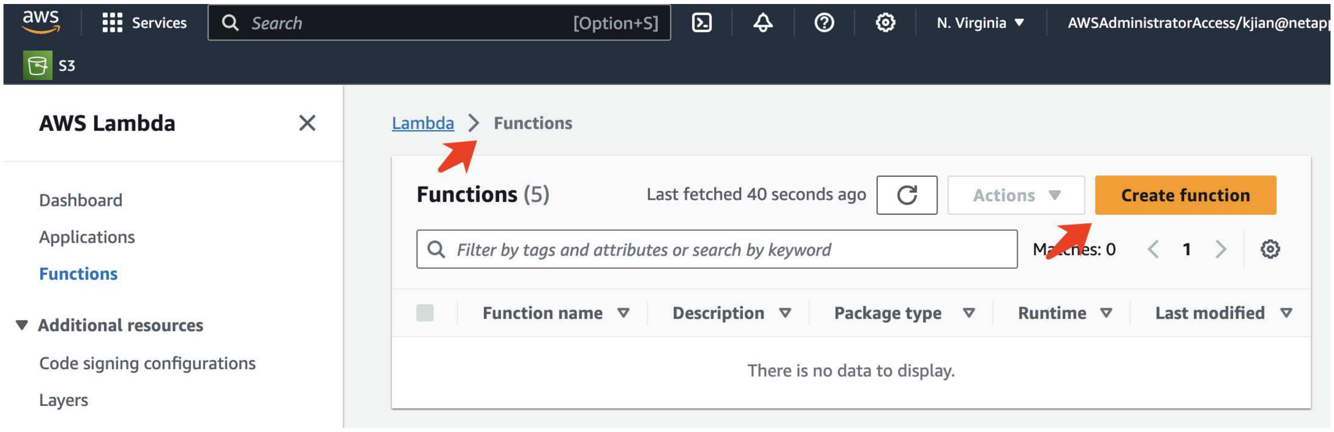
6. Select the created **Lifecycle configuration** and click **Update notebook instance**.



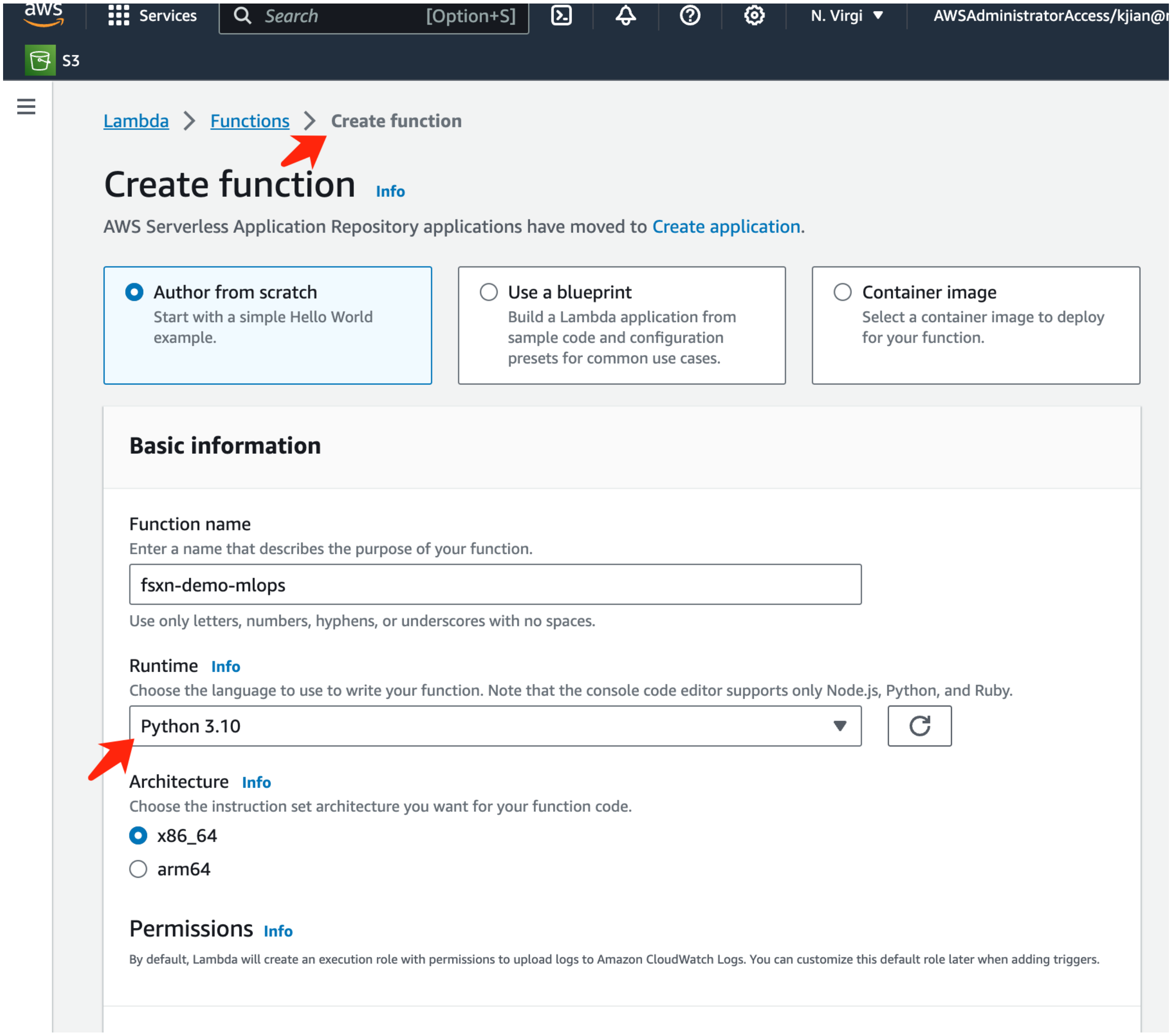
AWS Lambda serverless function

As mentioned earlier, the **AWS Lambda function** is responsible for spinning up the **AWS SageMaker Notebook instance**.

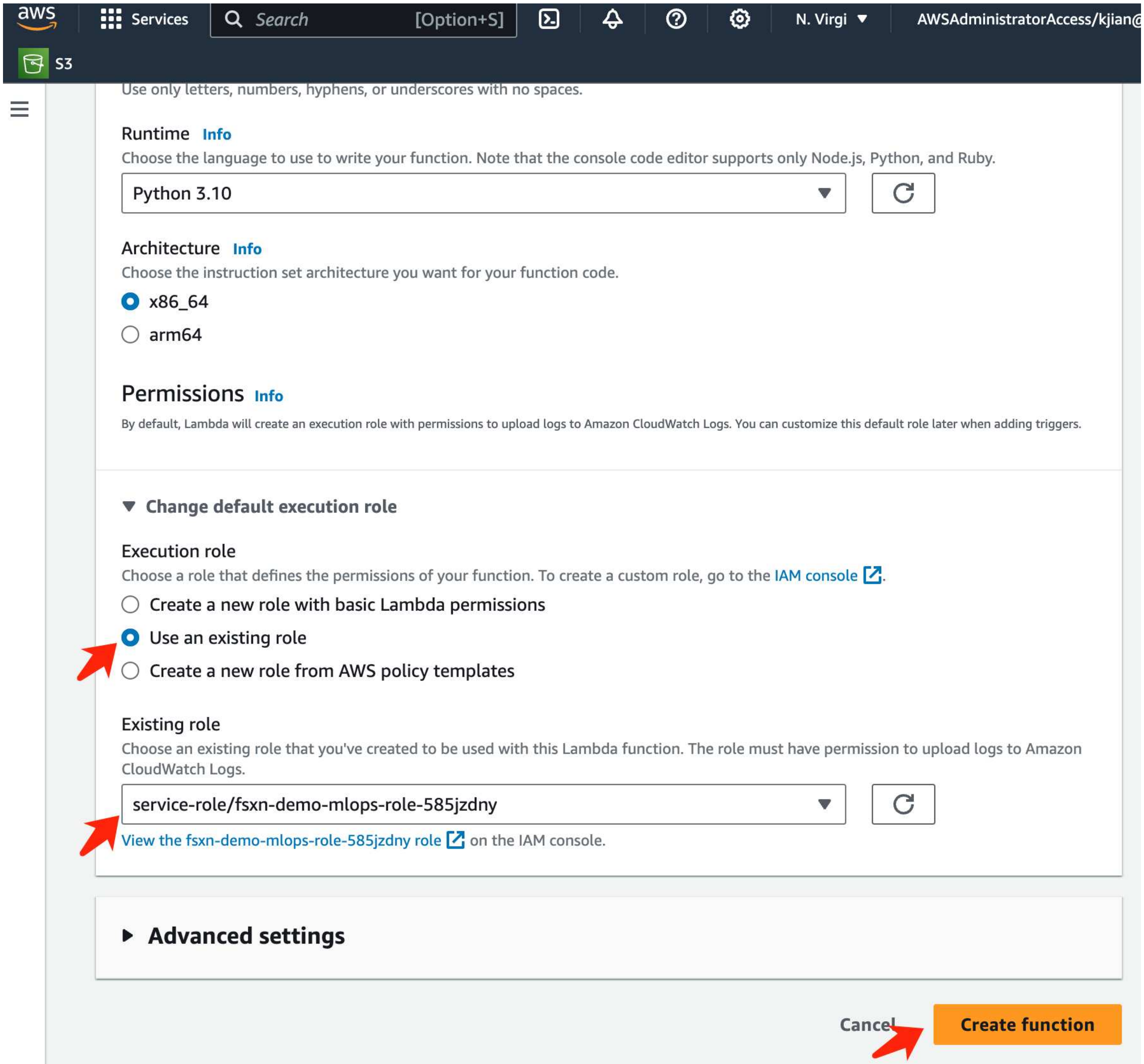
1. To create an **AWS Lambda function**, navigate to the respective panel, switch to the **Functions** tab, and click on **Create Function**.



2. Please file all required entries on the page and remember to switch the Runtime to **Python 3.10**.



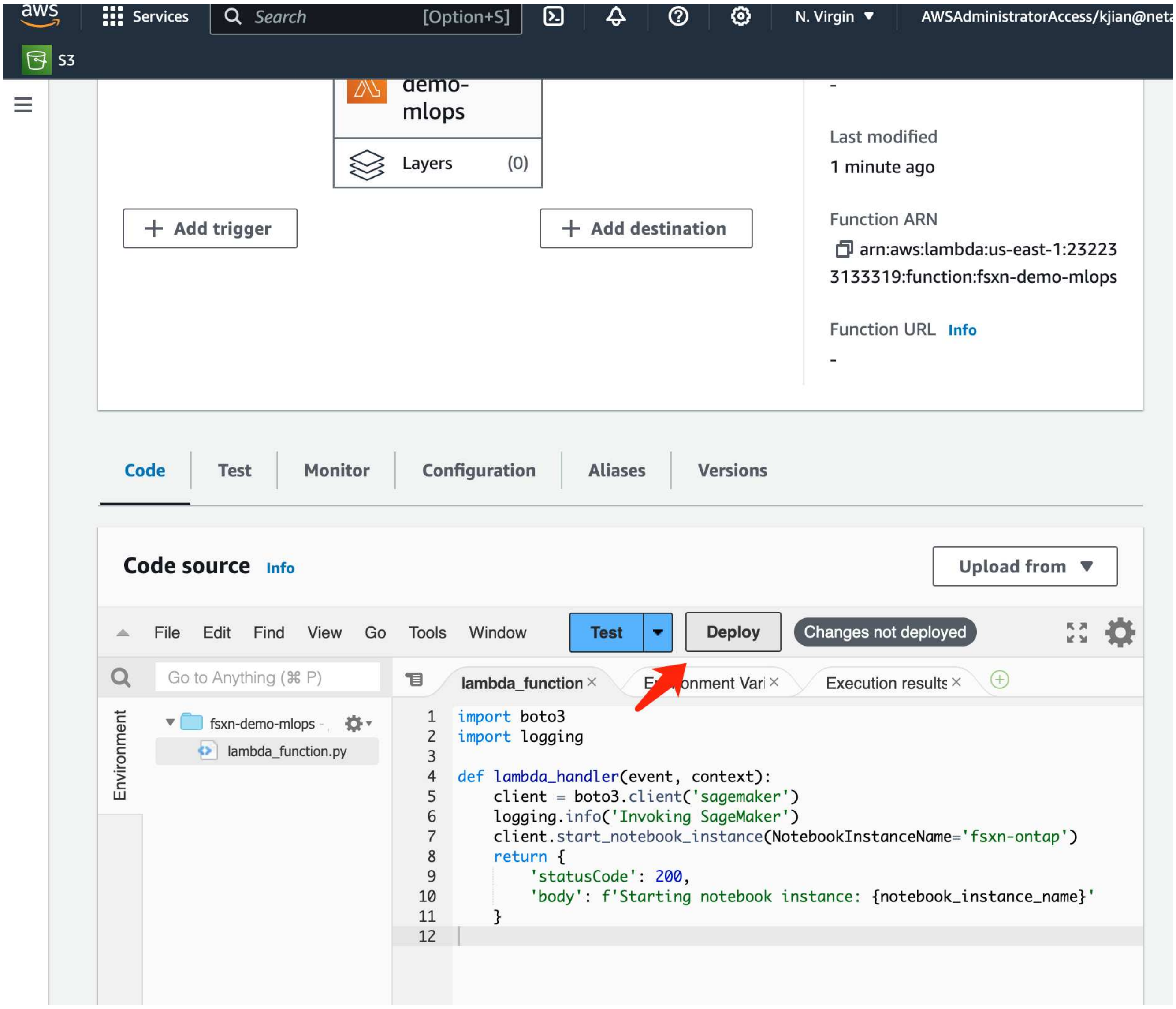
3. Please verify that the designated role has the required permission **AmazonSageMakerFullAccess** and click on the **Create function** button.



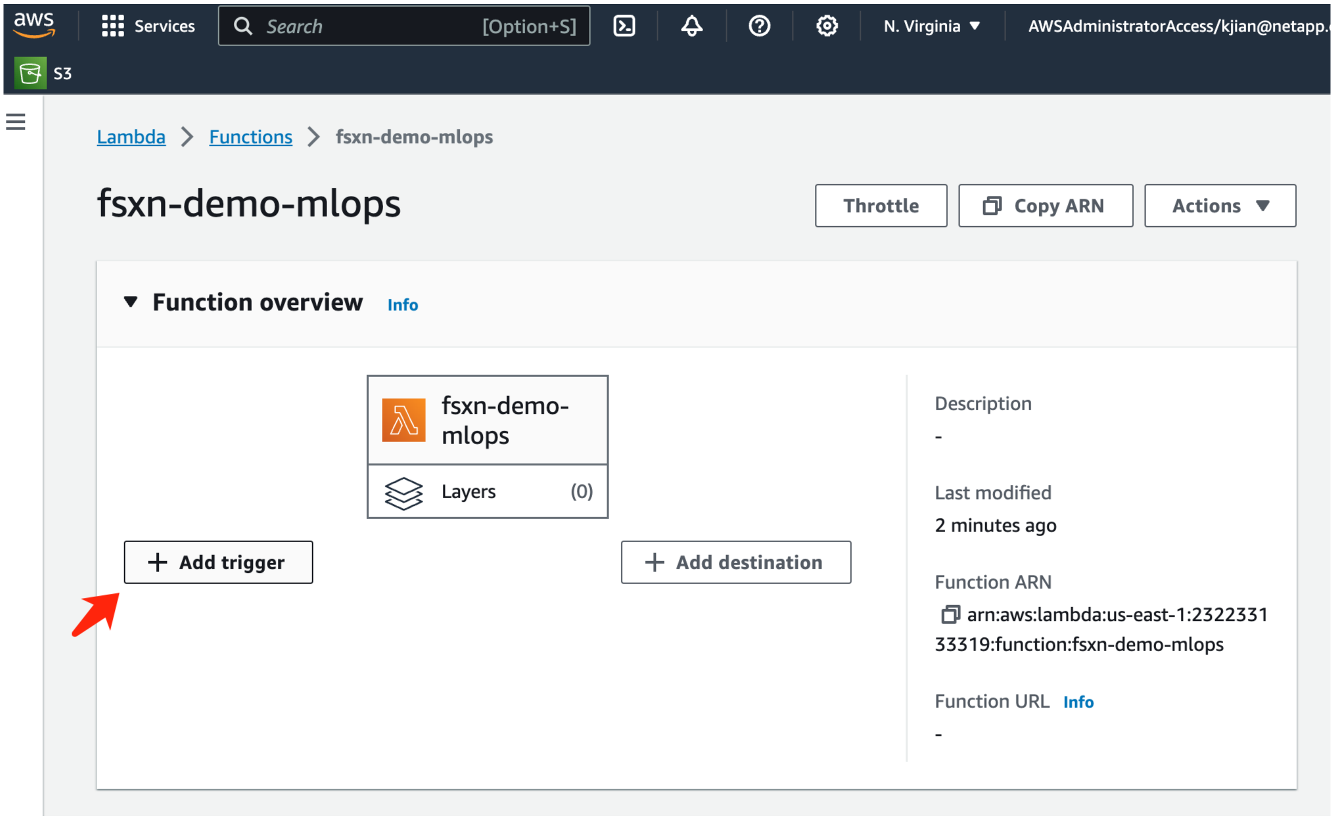
4. Select the created Lambda function. In the code tab, copy and paste the following code into the text area. This code starts the notebook instance named **fsxn-ontap**.

```
import boto3
import logging

def lambda_handler(event, context):
    client = boto3.client('sagemaker')
    logging.info('Invoking SageMaker')
    client.start_notebook_instance(NotebookInstanceName='fsxn-ontap')
    return {
        'statusCode': 200,
        'body': f'Starting notebook instance: {notebook_instance_name}'
    }
```

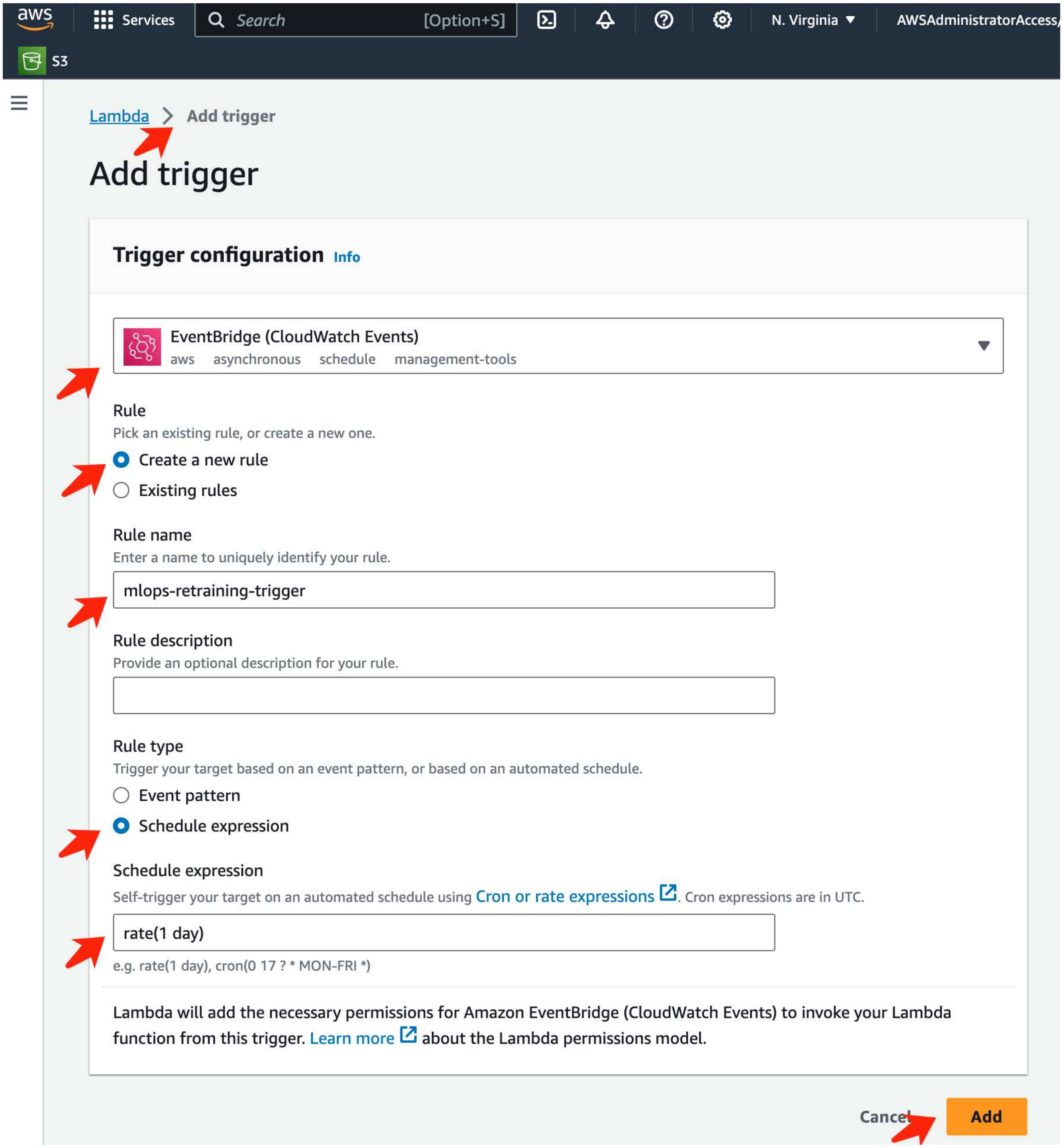
5. Click the **Deploy** button to apply this code change.



6. To specify how to trigger this AWS Lambda function, click on the Add Trigger button.



7. Select EventBridge from the dropdown menu, then click on the radio button labeled Create a new rule. In the schedule expression field, enter `rate(1 day)`, and click on the Add button to create and apply this new cron job rule to the AWS Lambda function.



After completing the two-step configuration, on a daily basis, the **AWS Lambda function** will initiate the **SageMaker Notebook**, perform model retraining using the data from the **FSxN** repository, redeploy the updated model to the production environment, and automatically shut down the **SageMaker Notebook instance** to optimize cost. This ensures that the model remains up to date.

This concludes the tutorial for developing an MLOps pipeline.

Hybrid Multicloud MLOps with Domino Data Lab and NetApp

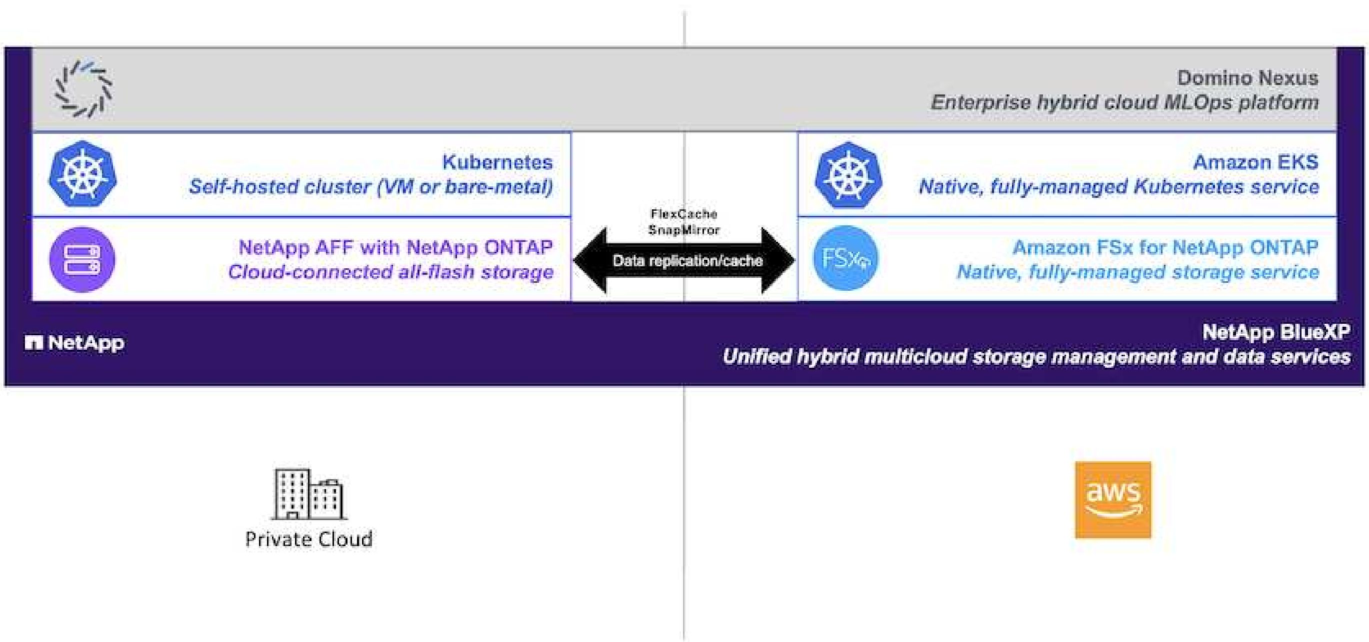
Hybrid Multicloud MLOps with Domino Data Lab and NetApp

Mike Oglesby, NetApp

Organizations all over the world are currently adopting AI to transform their businesses and processes. Because of this, AI-ready compute infrastructure is often in short supply. Enterprises are adopting hybrid multicloud MLOps architectures in order to take advantage of available compute environments across different regions, data centers, and clouds - balancing cost, availability, and performance.

Domino Nexus, from Domino Data Lab, is a unified MLOps control plane that lets you run data science and machine learning workloads across any compute cluster — in any cloud, region, or on-premises. It unifies data science silos across the enterprise, so you have one place to build, deploy, and monitor models. Likewise, NetApp's hybrid cloud data management capabilities enable you to bring your data to your jobs and workspaces, no matter where they are running. When you pair Domino Nexus with NetApp, you have the flexibility to schedule workloads across environments without having to worry about data availability. In other words, you have the ability to send your workloads and your data to the appropriate compute environment, enabling you to accelerate your AI deployments while navigating regulations around data privacy and sovereignty.

This solution demonstrates the deployment of a unified MLOps control plane incorporating an on-premises Kubernetes cluster and an Elastic Kubernetes Service (EKS) cluster running in Amazon Web Services (AWS).



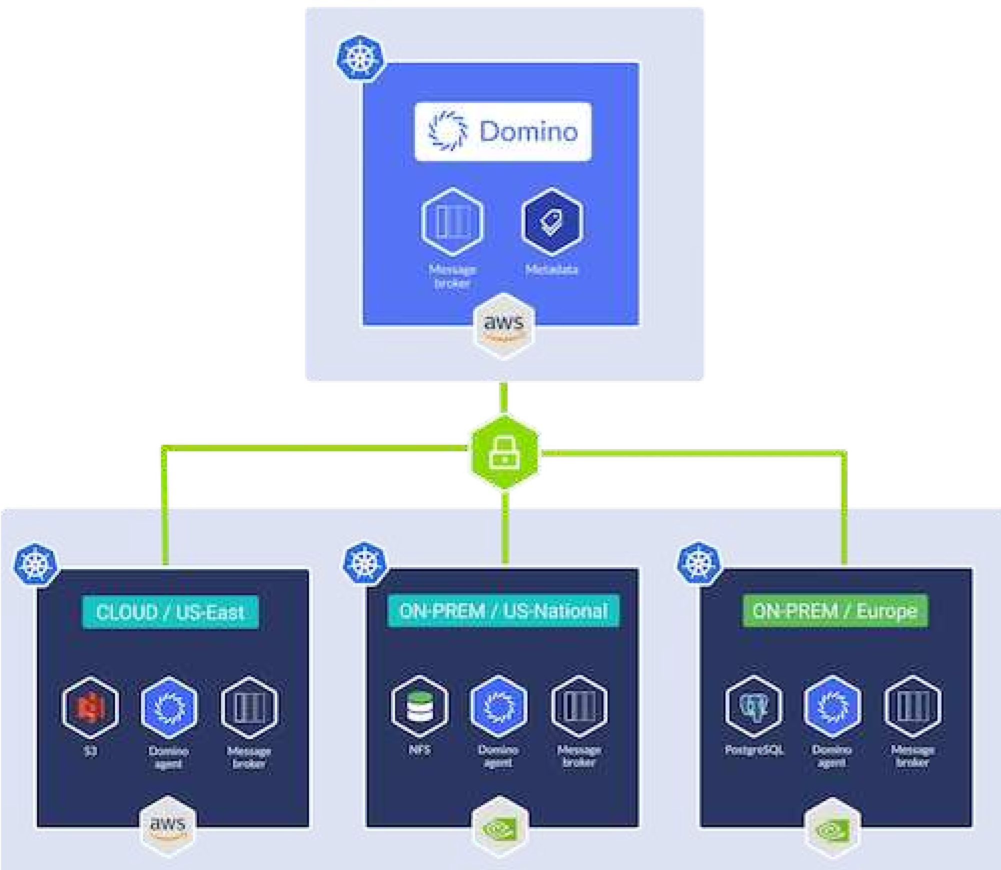
Technology Overview

Domino Data Lab

Domino Data Lab powers model-driven businesses with its leading Enterprise AI platform trusted by over 20% of the Fortune 100. Domino accelerates the development and deployment of data science work while increasing collaboration and governance. With Domino, enterprises worldwide can develop better medicines, grow more productive crops, build better cars, and much more. Founded in 2013, Domino is backed by Coatue Management, Great Hill Partners, Highland Capital, Sequoia Capital and other leading investors.

Domino lets enterprises and their data scientists build, deploy and manage AI on a unified, end-to-end platform — fast, responsibly and cost-effectively. Teams can access all of the data, tools, compute, models, and projects they need across any environment, so they can collaborate, reuse past work, track models in production to improve accuracy, standardize with best practices, and make AI responsible and governed.

- **Open and Flexible:** Access the broadest ecosystem of open source and commercial tools, and infrastructure, for the best innovations and no vendor lock-in.
- **System of Record:** Central hub for AI operations and knowledge across the enterprise, enabling best practices, cross-functional collaboration, faster innovation, and efficiency.
- **Integrated:** Integrated workflows and automation — built for enterprise processes, controls, and governance — satisfy your compliance and regulatory needs.
- **Hybrid Multicloud:** Run AI workloads close to your data anywhere — on-premises, hybrid, any cloud or multi-cloud — for lower cost, optimal performance and compliance.



Domino Nexus

Domino Nexus is a single pane of glass that lets you run data science and machine learning workloads across any compute cluster — in any cloud, region, or on-premises. It unifies data science silos across the enterprise, so you have one place to build, deploy, and monitor models.

NetApp BlueXP

NetApp BlueXP unifies all of NetApp's storage and data services into a single tool that lets you build, protect, and govern your hybrid multicloud data estate. It delivers a unified experience for storage and data services across on-premises and cloud environments, and enables operational simplicity through the power of AIOps, with the flexible consumption parameters and integrated protection required for today's cloud-led world.

NetApp ONTAP

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

Simplify data management

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

ONTAP data management software includes the following features to streamline and simplify operations and reduce your total cost of operation:

- Inline data compaction and expanded deduplication. Data compaction reduces wasted space inside storage blocks, and deduplication significantly increases effective capacity. This applies to data stored locally and data tiered to the cloud.
- Minimum, maximum, and adaptive quality of service (AQoS). Granular quality of service (QoS) controls help maintain performance levels for critical applications in highly shared environments.
- NetApp FabricPool. Provides automatic tiering of cold data to public and private cloud storage options, including Amazon Web Services (AWS), Azure, and NetApp StorageGRID storage solution. For more information about FabricPool, see TR-4598: FabricPool best practices.

Accelerate and protect data

ONTAP delivers superior levels of performance and data protection and extends these capabilities in the following ways:

- Performance and lower latency. ONTAP offers the highest possible throughput at the lowest possible latency.
- Data protection. ONTAP provides built-in data protection capabilities with common management across all platforms.
- NetApp Volume Encryption (NVE). ONTAP offers native volume-level encryption with both onboard and External Key Management support.
- Multitenancy and multifactor authentication. ONTAP enables sharing of infrastructure resources with the highest levels of security.

Future-proof infrastructure

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

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

- Cloud connection. ONTAP is the most cloud-connected storage management software, with options for software-defined storage and cloud-native instances in all public clouds.
- Integration with emerging applications. ONTAP offers enterprise-grade data services for next generation platforms and applications, such as autonomous vehicles, smart cities, and Industry 4.0, by using the same infrastructure that supports existing enterprise apps.

Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP is a first-party, fully managed AWS service that provides highly reliable, scalable, high-performing, and feature-rich file storage built on NetApp's popular ONTAP file system. FSx for ONTAP combines the familiar features, performance, capabilities, and API operations of NetApp file systems with the agility, scalability, and simplicity of a fully managed AWS service.

NetApp Astra Trident

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

Kubernetes

Kubernetes is an open source, distributed, container orchestration platform that was originally designed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes enables the automation of deployment, management, and scaling functions for containerized applications, and is the dominant container orchestration platform in enterprise environments.

Amazon Elastic Kubernetes Service (EKS)

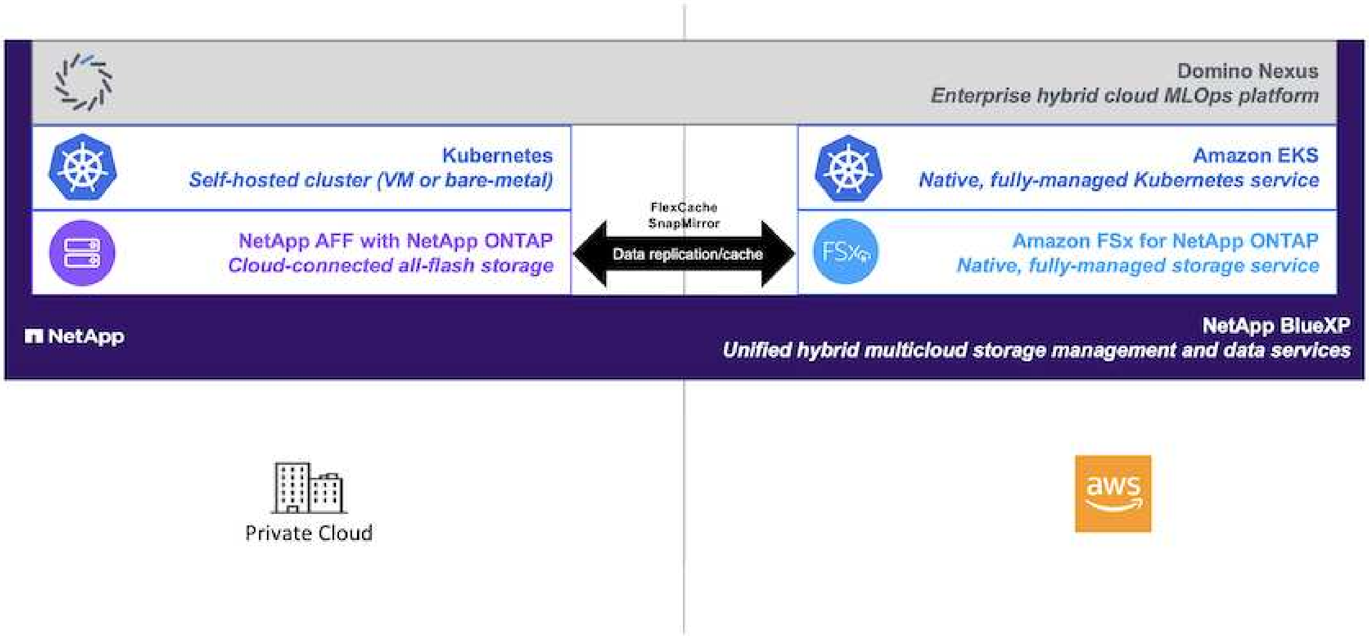
Amazon Elastic Kubernetes Service (Amazon EKS) is a managed Kubernetes service in the AWS cloud. Amazon EKS automatically manages the availability and scalability of the Kubernetes control plane nodes responsible for scheduling containers, managing application availability, storing cluster data, and other key tasks. With Amazon EKS, you can take advantage of all the performance, scale, reliability, and availability of AWS infrastructure, as well as integrations with AWS networking and security services.

Architecture

This solution combines Domino Nexus' hybrid multicloud workload scheduling capabilities with NetApp data services to create a unified hybrid cloud MLOps platform. See the following table for details.

Component	Name	Environment
MLOps Control Plane	Domino Enterprise AI Platform with Domino Nexus	AWS
MLOps Platform Compute Environments	Domino Nexus Data Planes	AWS, On-premises data center

Component	Name	Environment
On-premises Compute Platform	Kubernetes with NetApp Astra Trident	On-premises data center
Cloud Compute Platform	Amazon Elastic Kubernetes Service (EKS) with NetApp Astra Trident	AWS
On-premises Data Platform	NetApp storage appliance powered by NetApp ONTAP	On-premises data center
Cloud Data Platform	Amazon FSx for NetApp ONTAP	AWS



Initial Setup

This section describes the initial setup tasks that need to be performed in order to utilize Domino Nexus with NetApp data services in a hybrid environment incorporating an on-premises data center and AWS.

Prerequisites

Before you perform the steps that are outlined in this section, we assume that you have already performed the following tasks:

- You have already deployed and configured your on-premises NetApp ONTAP storage platform. For more information, refer to the NetApp product documentation.
- You have already provisioned an Amazon FSx for NetApp ONTAP instance in AWS. For more information, refer to the Amazon FSx for NetApp ONTAP product page.
- You have already provisioned a Kubernetes cluster in your on-premises data center. For more information, refer to the Domino admin guide.
- You have already provisioned an Amazon EKS cluster in AWS. For more information, refer to the Domino admin guide.
- You have installed NetApp Astra Trident in your on-premises Kubernetes cluster. Additionally, you have

configured this Trident instance to use your on-premises NetApp ONTAP storage platform when provisioning and managing storage resources. For more information, refer to the NetApp Astra Trident documentation.

- You have installed NetApp Astra Trident in your Amazon EKS cluster. Additionally, you have configured this Trident instance to use your Amazon FSx for NetApp ONTAP instance when provisioning and managing storage resources. For more information, refer to the NetApp Astra Trident documentation.
- You must have bi-directional network connectivity between your on-premises data center and your Virtual Private Cloud (VPC) in AWS. For more details on the various options for implementing this, refer to the Amazon Virtual Private Network (VPN) documentation.

Install the Domino Enterprise AI Platform in AWS

To install the Domino Enterprise MLOps Platform in AWS, follow the instructions outlined in Domino admin guide. You must deploy Domino in the same Amazon EKS cluster that you previously provisioned. Additionally, NetApp Astra Trident must already be installed and configured in this EKS cluster, and you must specify a Trident-managed storage class as the shared storage class in your domino.yml install configuration file.

> Refer to the Domino install configuration reference guide for details on how to specify a shared storage class in your domino.yml install configuration file.

> Technical Report TR-4952 walks through the deployment of Domino in AWS with Amazon FSx for NetApp ONTAP and may be a useful reference for troubleshooting any issues that arise.

Enable Domino Nexus

Next, you must enable Domino Nexus. Refer to the Domino admin guide for details.

Deploy a Domino Data Plane in your On-premises Data Center

Next, you must deploy a Domino Data Plane in your on-premises data center. You must deploy this data plane in the on-premises Kubernetes cluster that you previously provisioned. Additionally, NetApp Astra Trident must already be installed and configured in this Kubernetes cluster. Refer to the Domino admin guide for details.

Expose Existing NetApp Volumes to Domino

This section describes the tasks that need to be performed in order to expose existing NetApp ONTAP NFS volumes to the Domino MLOps platform. These same steps apply both on-premises and in AWS.

Why Expose NetApp ONTAP Volumes to Domino?

Using NetApp volumes in conjunction with Domino provides the following benefits:

- You can execute workloads against extremely large datasets by taking advantage of NetApp ONTAP's scale-out capabilities.
- You can execute workloads across multiple compute nodes without having to copy your data to the individual nodes.
- You can take advantage of NetApp's hybrid multicloud data movement and sync capabilities in order to access your data across multiple data centers and/or clouds.
- You want to be able to quickly and easily create a cache of your data in a different data center or cloud.

Expose Existing NFS Volumes that were not Provisioned by Astra Trident

If your existing NetApp ONTAP NFS volume was not provisioned by Astra Trident, follow the steps outlined in this sub-section.

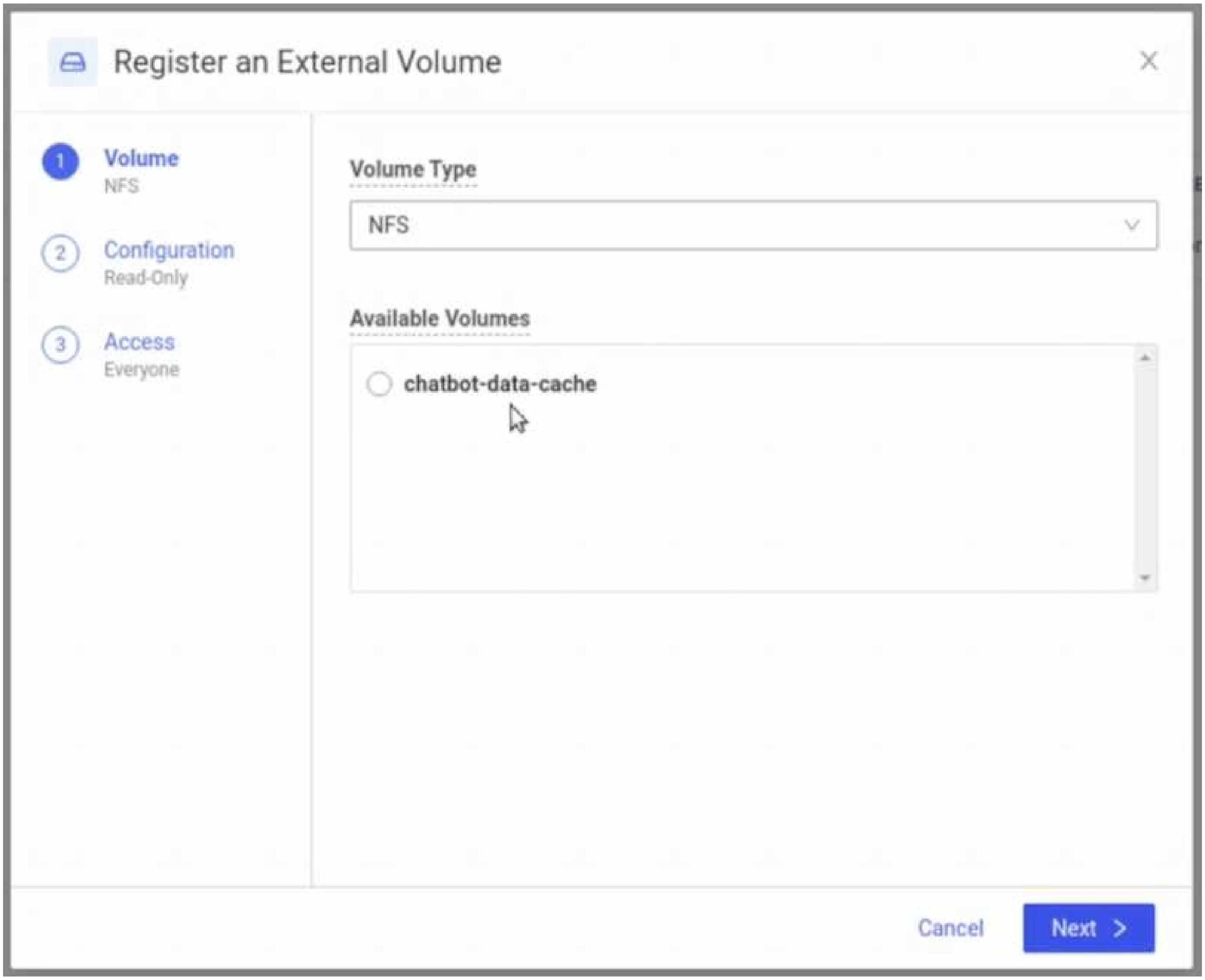
Create PV and PVC in Kubernetes

> For on-premises volumes, create the PV and PVC in your on-premises Kubernetes cluster. For Amazon FSx for NetApp ONTAP volumes, create the PV and PVC in Amazon EKS.

First, you must create a persistent volume (PV) and persistent volume claim (PVC) in your Kubernetes cluster. To create the PV and PVC, use the NFS PV/PVC example from the Domino admin guide and update the values to reflect to your environment. Be sure to specify the correct values for the `namespace`, `nfs.path`, and `nfs.server` fields. Additionally, we recommend giving your PV and PVC unique names that represent that nature of the data that is stored on the corresponding ONTAP NFS volume. For example, if the volume contains images of manufacturing defects, you might name the PV, `pv-mfg-defect-images`, and the PVC, `pvc-mfg-defect-images`.

Register External Data Volume in Domino

Next, you must register an external data volume in Domino. To register an external data volume, refer to the instructions in the Domino admin guide. When registering the volume, be sure to select "NFS" from the 'Volume Type' drop-down menu. After selecting "NFS", you should see your PVC in the 'Available Volumes' list.



Expose Existing Volumes that were Provisioned by Astra Trident

If your existing volume was provisioned by Astra Trident, follow the steps outlined in this sub-section.

Edit Existing PVC

If your volume was provisioned by Astra Trident, then you already have a persistent volume claim (PVC) corresponding to your volume. In order to expose this volume to Domino, you must edit the PVC and add the following label to the list of labels in the `metadata.labels` field:

```
"dominodatalab.com/external-data-volume": "Generic"
```

Register External Data Volume in Domino

Next, you must register an external data volume in Domino. To register an external data volume, refer to the instructions in the Domino admin guide. When registering the volume, be sure to select "Generic" from the 'Volume Type' drop-down menu. After selecting "Generic", you should see your PVC in the 'Available Volumes' list.

Access the same Data Across Different Environments

This section describes the tasks that need to be performed in order to access the same data across different compute environments. In the Domino MLOps platform, compute environments are referred to "data planes." Follow the tasks outlined in this section if your data resides on a NetApp volume in one data plane, but you need to access it in another data plane. This type of scenario is often referred to as "bursting" or, when the destination environment is the cloud, "cloud bursting." This capability is often needed when dealing with constrained or over-subscribed compute resources. For example, if your on-premises compute cluster is over-subscribed, you may want to schedule workloads to the cloud where they can be started immediately.

There are two recommended options for accessing a NetApp volume that resides in a different data plane. These options are outlined in the sub-sections below. Choose one of these options depending on your specific requirements. The benefits and drawbacks of the two options are described in the following table.

Option	Benefits	Drawbacks
Option 1 - Cache	- Simpler workflow - Ability to cache a subset of data based on needs - Ability to write data back to source - No remote copy to manage	- Increased latency on initial data access as cache is hydrated.
Option 2 - Mirror	- Full copy of source volume - No increased latency due to cache hydration (after mirror operation is complete)	- Must wait for mirror operation to complete before accessing data - Must manage a remote copy - No ability to write back to source

Option 1 - Create a Cache of a Volume that Resides in a Different Data Plane

With NetApp FlexCache technology, you can create a cache of a NetApp volume that resides in a different data plane. For example, if you have a NetApp volume in your on-premises data plane, and you need to access that volume in your AWS data plane, you can create a cache of the volume in AWS. This section outlines the tasks that need to be performed in order to create a cache of a NetApp volume that resides in a different data plane.

Create FlexCache Volume in Destination Environment

> ⓘ If the destination environment is your on-premises data center, you will create the FlexCache volume on your on-premises ONTAP system. If the destination environment is AWS, you will create the FlexCache volume on your Amazon FSx for NetApp ONTAP instance.

First, you must create a FlexCache volume in the destination environment.

We recommend using BlueXP to create the FlexCache volume. To create a FlexCache volume with BlueXP, follow the instructions outlined in the BlueXP volume caching documentation.

If you prefer not to use BlueXP, you can use ONTAP System Manager or the ONTAP CLI to create the FlexCache volume. To create a FlexCache volume with System Manager, refer to the instructions outlined in the ONTAP documentation. To create a FlexCache volume with the ONTAP CLI, refer to the instructions outlined in the ONTAP documentation.

If you wish to automate this process, you can use the BlueXP API, the ONTAP REST API, or the ONTAP Ansible collection.

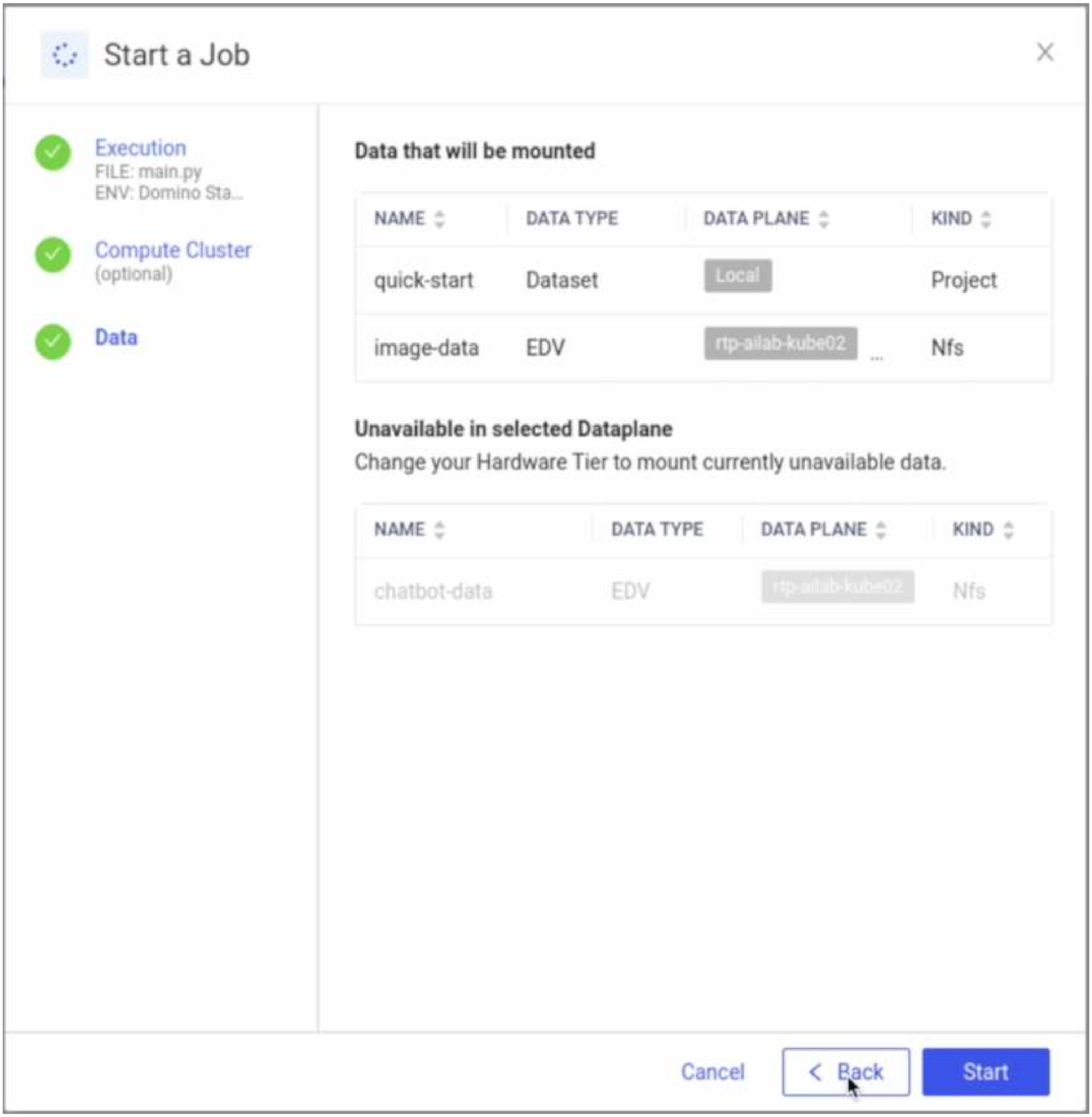
> System Manager is not available in Amazon FSx for NetApp ONTAP.

Expose FlexCache Volume to Domino

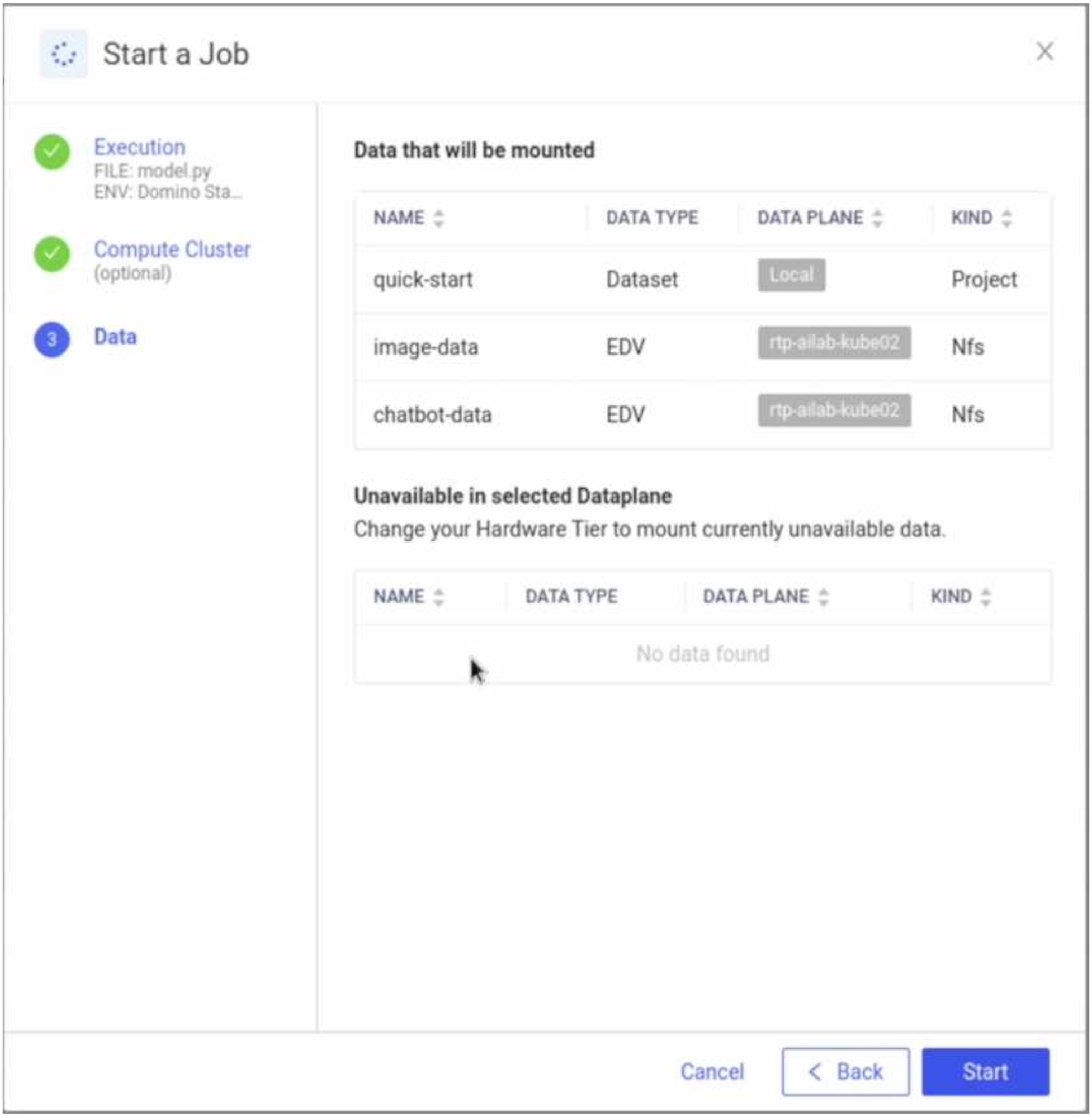
Next, you must expose the FlexCache volume to the Domino MLOps platform. To expose the FlexCache volume to Domino, follow the instructions outlined in the 'Expose Existing NFS Volumes that were not Provisioned by Astra Trident' sub-section of the 'Expose Existing NetApp Volumes to Domino' section of this solution.

Now, you will be able to mount the FlexCache volume when launching jobs and workspaces in the destination data plane as shown in the following screenshots.

Before Creating FlexCache Volume



After Exposing FlexCache Volume to Domino



Option 2 - Replicate a Volume that Resides in a Different Data Plane

With NetApp SnapMirror data replication technology, you can create a copy of a NetApp volume that resides in a different data plane. For example, if you have a NetApp volume in your on-premises data plane, and you need to access that volume in your AWS data plane, you can create a copy of the volume in AWS. This section outlines the tasks that need to be performed in order to create a copy of a NetApp volume that resides in a different data plane.

Create SnapMirror Relationship

First, you must create a SnapMirror relationship between your source volume and a new destination volume in the destination environment. Note that the destination volume will be created as part of the process of creating the SnapMirror relationship.

We recommend using BlueXP to create the SnapMirror relationship. To create a SnapMirror relationship with BlueXP, follow the instructions outlined in the BlueXP replication documentation.

If you prefer not to use BlueXP, you can use ONTAP System Manager or the ONTAP CLI to create the SnapMirror relationship. To create a SnapMirror relationship with System Manager, refer to the instructions outlined in the ONTAP documentation. To create a SnapMirror relationship with the ONTAP CLI, refer to the instructions outlined in the ONTAP documentation.

If you wish to automate this process, you can use the BlueXP API, the ONTAP REST API, or the ONTAP Ansible collection.

> System Manager is not available in Amazon FSx for NetApp ONTAP.

Break SnapMirror Relationship

Next, you must break the SnapMirror relationship in order to activate the destination volume for data access. Wait until the initial replication is complete before performing this step.

> You can determine whether or not the replication is complete by checking the mirror state in BlueXP, ONTAP System Manager, or the ONTAP CLI. When the replication is complete, the mirror state will be "snapmirrored".

We recommend using BlueXP to break the SnapMirror relationship. To break a SnapMirror relationship with BlueXP, follow the instructions outlined in the BlueXP replication documentation.

If you prefer not to use BlueXP, you can use ONTAP System Manager or the ONTAP CLI to break the SnapMirror relationship. To break a SnapMirror relationship with System Manager, refer to the instructions outlined in the ONTAP documentation. To break a SnapMirror relationship with the ONTAP CLI, refer to the instructions outlined in the ONTAP documentation.

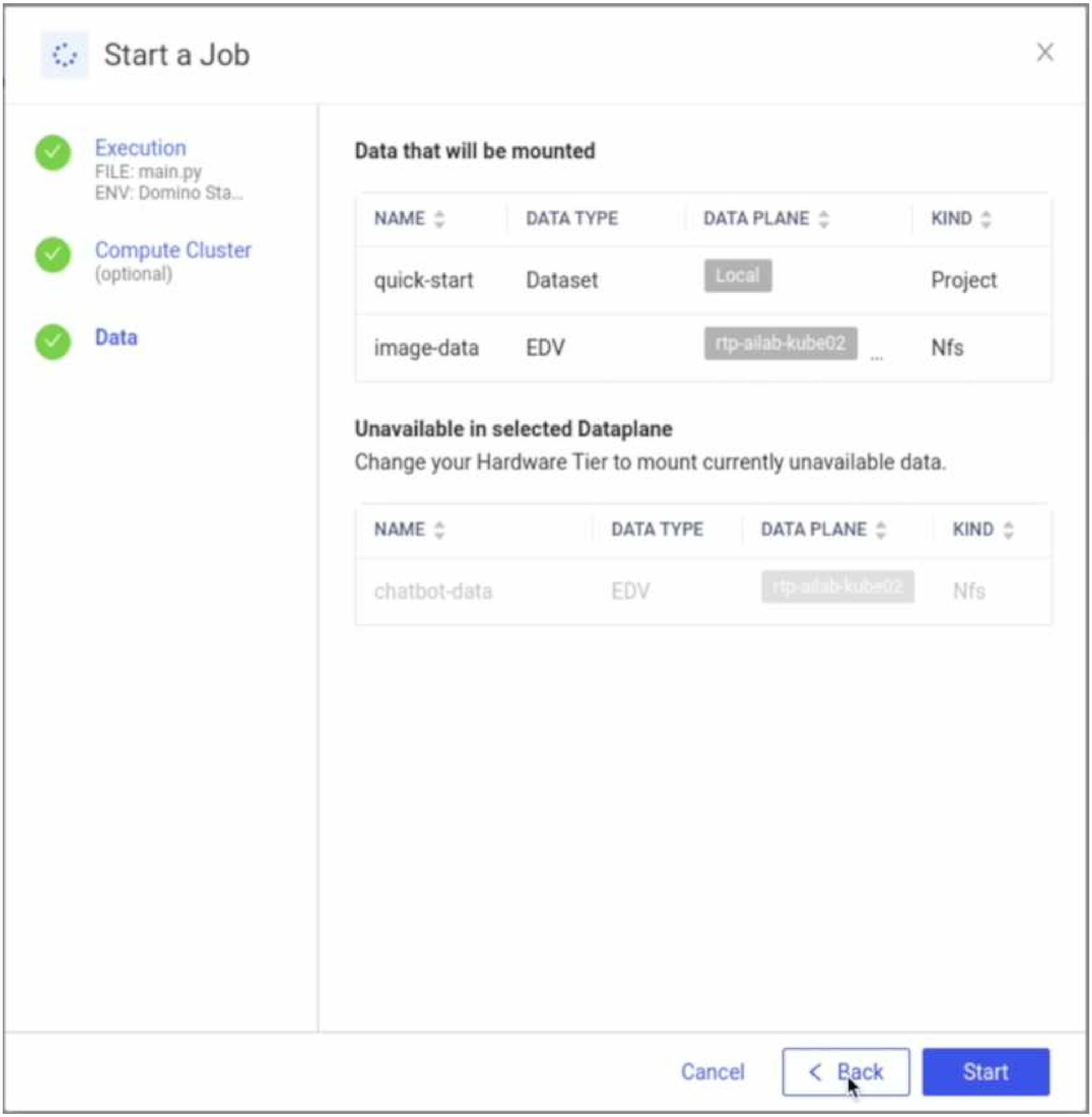
If you wish to automate this process, you can use the BlueXP API, the ONTAP REST API, or the ONTAP Ansible collection.

Expose Destination Volume to Domino

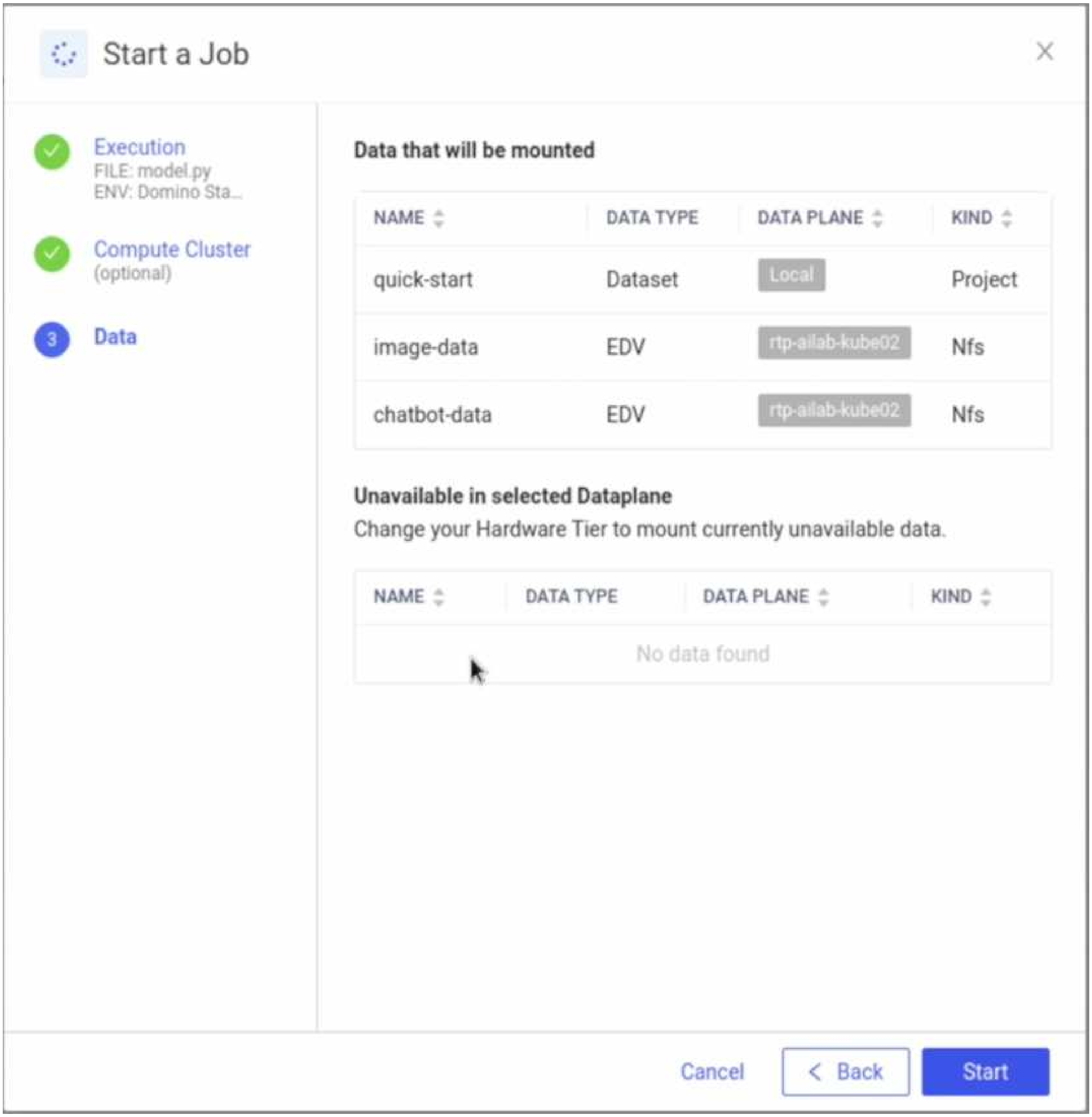
Next, you must expose the destination volume to the Domino MLOps platform. To expose the destination volume to Domino, follow the instructions outlined in the 'Expose Existing NFS Volumes that were not Provisioned by Astra Trident' sub-section of the 'Expose Existing NetApp Volumes to Domino' section of this solution.

Now, you will be able to mount the destination volume when launching jobs and workspaces in the destination data plane as shown in the following screenshots.

Before Creating SnapMirror Relationship



After Exposing Destination Volume to Domino



Where to Find Additional Information

To learn more about the information described in this document, refer to the following documents and/or websites:

- Domino Data Lab

  https://domino.ai

- Domino Nexus

  https://domino.ai/platform/nexus

* NetApp BlueXP

  https://bluexp.netapp.com

* NetApp ONTAP data management software

  https://www.netapp.com/data-management/ontap-data-management-software/

* NetApp AI Solutions

  https://www.netapp.com/artificial-intelligence/

Acknowledgments

* Josh Mineroff, Director of SA for Tech Alliances, Domino Data Lab
* Nicholas Jablonski, Field CTO, Domino Data Lab
* Prabu Arjunan, Solution Architect, NetApp
* Brian Young, Global Alliance Director, Technology Alliance Partners, NetApp

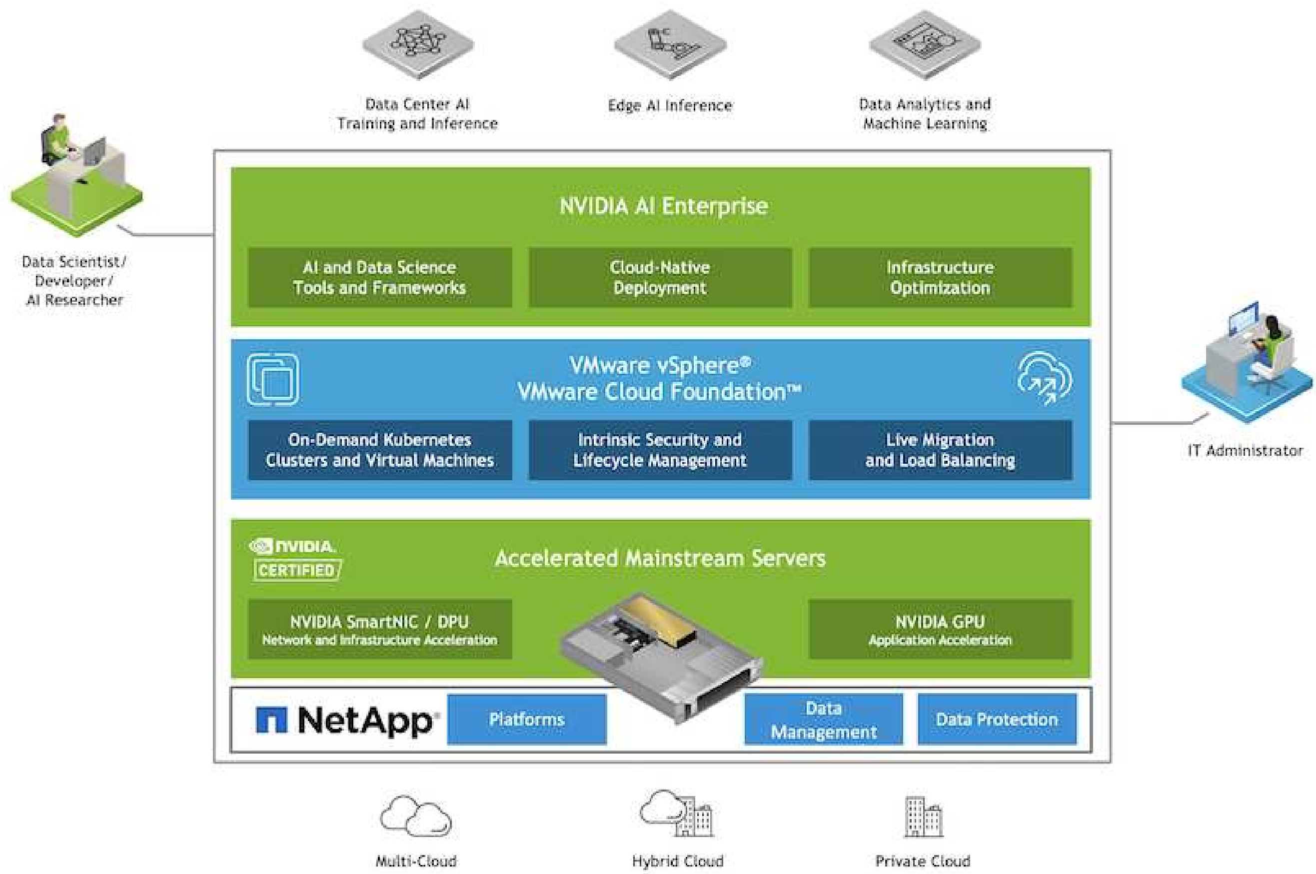
NVIDIA AI Enterprise with NetApp and VMware

NVIDIA AI Enterprise with NetApp and VMware

Mike Oglesby, NetApp

For IT architects and admins, AI tooling can be complicated and unfamiliar. Additionally, many AI platforms are not enterprise-ready. NVIDIA AI Enterprise, powered by NetApp and VMware, was created to deliver a streamlined, enterprise-class AI architecture.

NVIDIA AI Enterprise is an end-to-end, cloud-native suite of AI and data analytics software that is optimized, certified, and supported by NVIDIA to run on VMware vSphere with NVIDIA-Certified Systems. This software facilitates the simple and rapid deployment, management, and scaling of AI workloads in the modern hybrid cloud environment. NVIDIA AI Enterprise, powered by NetApp and VMware, delivers enterprise-class AI workload and data management in a simplified, familiar package.



Technology Overview

NVIDIA AI Enterprise

NVIDIA AI Enterprise is an end-to-end, cloud-native suite of AI and data analytics software that is optimized, certified, and supported by NVIDIA to run on VMware vSphere with NVIDIA-Certified Systems. This software facilitates the simple and rapid deployment, management, and scaling of AI workloads in the modern hybrid cloud environment.

NVIDIA GPU Cloud (NGC)

NVIDIA NGC hosts a catalog of GPU-optimized software for AI practitioners to develop their AI solutions. It also provides access to various AI services including NVIDIA Base Command for model training, NVIDIA Fleet Command to deploy and monitor models, and the NGC Private Registry for securely accessing and managing proprietary AI software. Also, NVIDIA AI Enterprise customers can request support through the NGC portal.

VMware vSphere

VMware vSphere is VMware’s virtualization platform, which transforms data centers into aggregated computing infrastructures that include CPU, storage, and networking resources. vSphere manages these infrastructures as a unified operating environment, and provides administrators with the tools to manage the data centers that participate in that environment.

The two core components of vSphere are ESXi and vCenter Server. ESXi is the virtualization platform where administrators create and run virtual machines and virtual appliances. vCenter Server is the service through which administrators manage multiple hosts connected in a network and pool host resources.

NetApp ONTAP

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

Simplify data management

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

ONTAP data management software includes the following features to streamline and simplify operations and reduce your total cost of operation:

- Inline data compaction and expanded deduplication. Data compaction reduces wasted space inside storage blocks, and deduplication significantly increases effective capacity. This applies to data stored locally and data tiered to the cloud.
- Minimum, maximum, and adaptive quality of service (AQoS). Granular quality of service (QoS) controls help maintain performance levels for critical applications in highly shared environments.
- NetApp FabricPool. Provides automatic tiering of cold data to public and private cloud storage options, including Amazon Web Services (AWS), Azure, and NetApp StorageGRID storage solution. For more information about FabricPool, see TR-4598: FabricPool best practices.

Accelerate and protect data

ONTAP delivers superior levels of performance and data protection and extends these capabilities in the following ways:

- Performance and lower latency. ONTAP offers the highest possible throughput at the lowest possible latency.
- Data protection. ONTAP provides built-in data protection capabilities with common management across all platforms.
- NetApp Volume Encryption (NVE). ONTAP offers native volume-level encryption with both onboard and External Key Management support.
- Multitenancy and multifactor authentication. ONTAP enables sharing of infrastructure resources with the highest levels of security.

Future-proof infrastructure

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

- Seamless scaling and nondisruptive operations. ONTAP supports the nondisruptive addition of capacity to existing controllers and to scale-out clusters. Customers can upgrade to the latest technologies, such as NVMe and 32Gb FC, without costly data migrations or outages.
- Cloud connection. ONTAP is the most cloud-connected storage management software, with options for software-defined storage (ONTAP Select) and cloud-native instances (NetApp Cloud Volumes Service) in all public clouds.
- Integration with emerging applications. ONTAP offers enterprise-grade data services for next generation

platforms and applications, such as autonomous vehicles, smart cities, and Industry 4.0, by using the same infrastructure that supports existing enterprise apps.

NetApp DataOps Toolkit

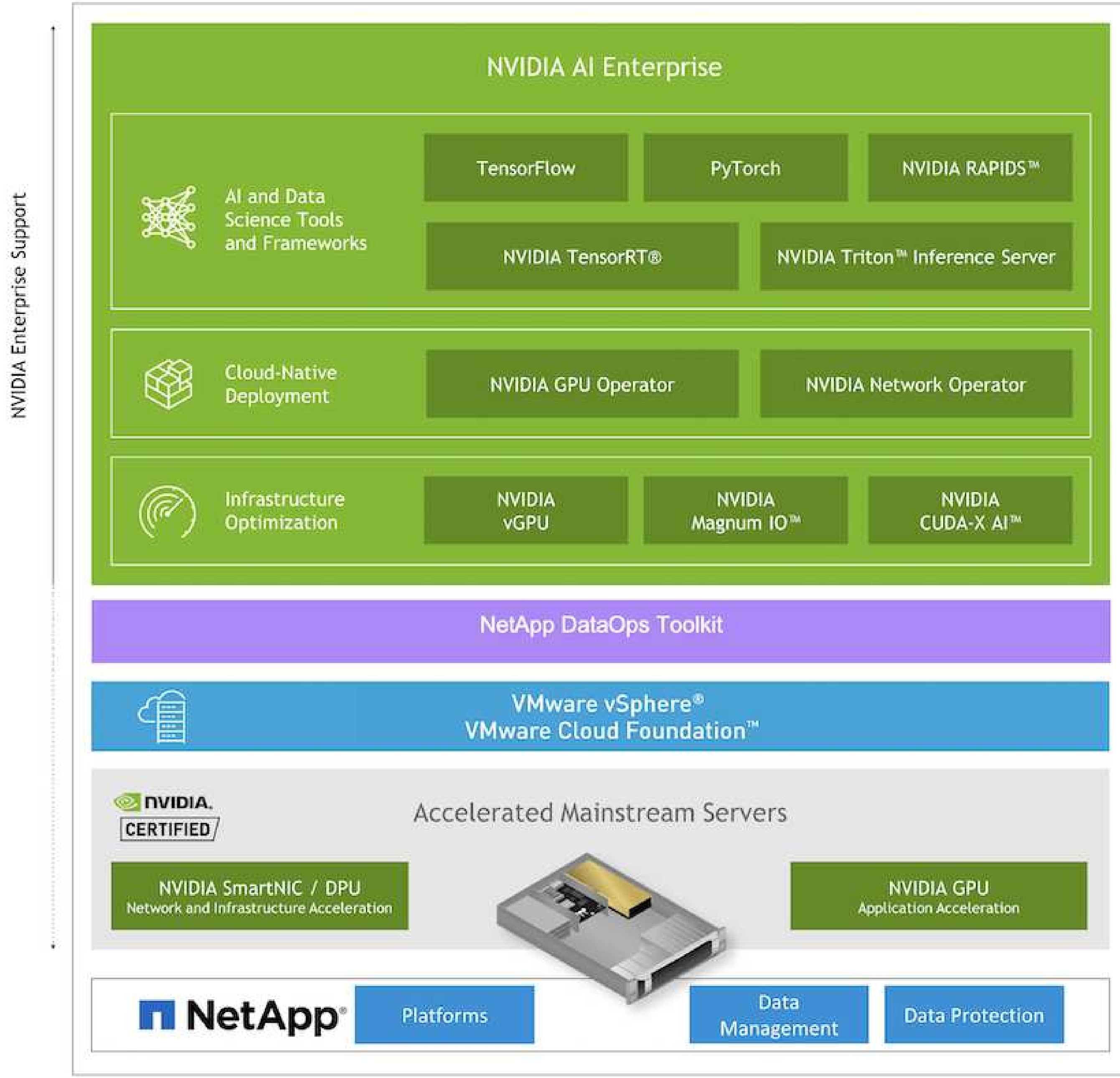
The NetApp DataOps Toolkit is a Python-based tool that simplifies the management of development/training workspaces and inference servers that are backed by high-performance, scale-out NetApp storage. Key capabilities include:

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

Architecture

This solution builds upon a proven and familiar architecture featuring NetApp, VMware, and NVIDIA-Certified Systems. See the following table for details.

Component	Details
AI and Data Analytics Software	NVIDIA AI Enterprise for VMware
Virtualization Platform	VMware vSphere
Compute Platform	NVIDIA-Certified Systems
Data Management Platform	NetApp ONTAP



Initial Setup

This section describes the initial setup tasks that need to be performed in order to utilize NVIDIA AI Enterprise with NetApp and VMware.

Prerequisites

Before you perform the steps that are outlined in this section, we assume that you have already deployed VMware vSphere and NetApp ONTAP. Refer to the NVIDIA AI Enterprise Product Support Matrix for details on supported vSphere versions. Refer to the NetApp and VMware solution documentation for details on deploying VMware vSphere with NetApp ONTAP.

Install NVIDIA AI Enterprise Host Software

To install the NVIDIA AI Entrprise host software, follow the instructions outlined in sections 1-4 in the NVIDIA AI Enterprise Quick Start Guide.

Utilize NVIDIA NGC Software

This section describes the tasks that need to be performed in order to utilize NVIDIA NGC enterprise software within an NVIDIA AI Enterprise environment.

Setup

This section describes the initial setup tasks that need to be performed in order to utilize NVIDIA NGC enterprise software within an NVIDIA AI Enterprise environment.

Prerequisites

Before you perform the steps that are outlined in this section, we assume that you have already deployed the NVIDIA AI Entrprise host software by following the instructions outlined on the Initial Setup page.

Create an Ubuntu Guest VM with vGPU

First, you must create an Ubuntu 20.04 guest VM with vGPU. To create an Ubuntu 20.04 guest VM with vGPU, follow the instructions outline in the NVIDIA AI Enterprise Deployment Guide.

Download and Install NVIDIA Guest Software

Next, you must install the required NVIDIA guest software within the guest VM that you created in the previous step. To download and install the required NVIDIA guest software within the guest VM, follow the instructions outlined in sections 5.1-5.4 in the NVIDIA AI Enterprise Quick Start Guide.

> When performing the verification tasks outlined in section 5.4, you may need to use a different CUDA container image version tag as the CUDA container image has been updated since the writing of the guide. In our validation, we used 'nvidia/cuda:11.0.3-base-ubuntu20.04'.

Download AI/Analytics Framework Container(s)

Next, you must download needed AI or analytics framework container images from NVIDIA NGC so that they will be available within your guest VM. To download framework containers within the guest VM, follow the instructions outlined in the NVIDIA AI Enterprise Deployment Guide.

Install and Configure the NetApp DataOps Toolkit

Next, you must install the NetApp DataOps Toolkit for Traditional Environemnts within the guest VM. The NetApp DataOps Toolkit can be used to manage scale-out data volumes on your ONTAP system directly from the terminal within the guest VM. To install the NetApp DataOps Toolkit within the guest VM, perform the following tasks.

1. Install pip.

```
$ sudo apt update
$ sudo apt install python3-pip
$ python3 -m pip install netapp-dataops-traditional
```

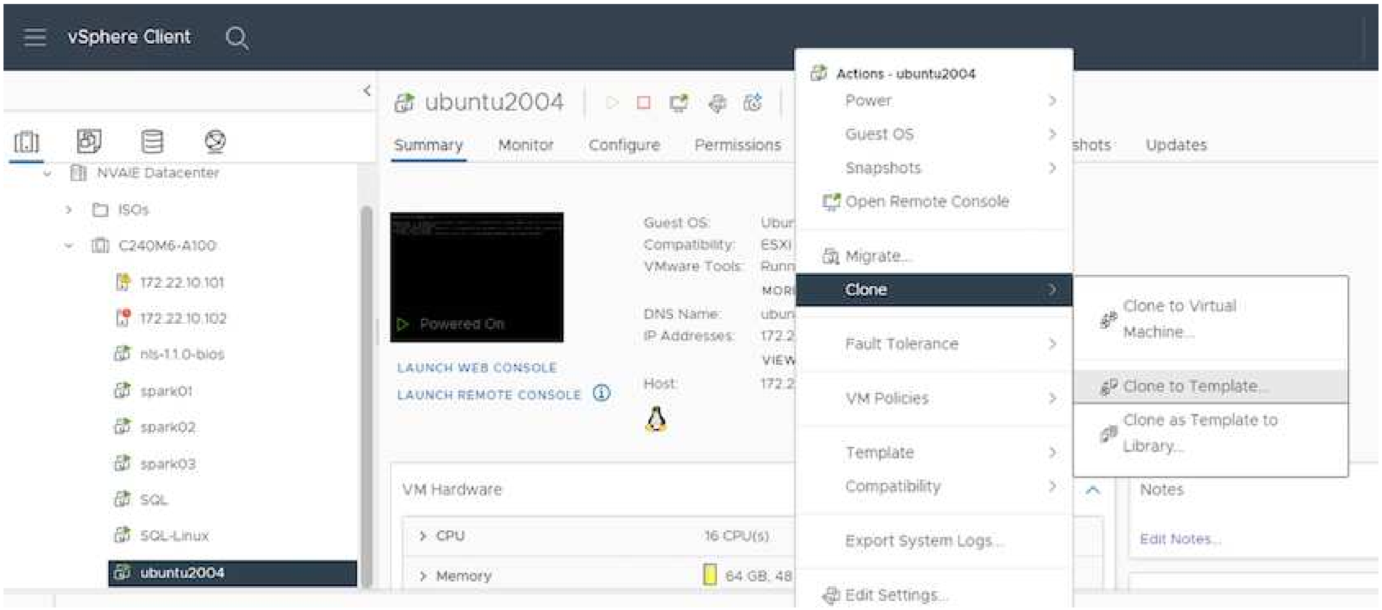
2. Log out of the guest VM terminal and then log back in.
3. Configure the NetApp DataOps Toolkit. In order to complete this step, you will need API access details for your ONTAP system. You may need to obtain these from your storage admin.

```
$ netapp_dataops_cli.py config

Enter ONTAP management LIF hostname or IP address (Recommendation: Use
SVM management interface): 172.22.10.10
Enter SVM (Storage VM) name: NVAIE-client
Enter SVM NFS data LIF hostname or IP address: 172.22.13.151
Enter default volume type to use when creating new volumes
(flexgroup/flexvol) [flexgroup]:
Enter export policy to use by default when creating new volumes
[default]:
Enter snapshot policy to use by default when creating new volumes
[none]:
Enter unix filesystem user id (uid) to apply by default when creating
new volumes (ex. '0' for root user) [0]:
Enter unix filesystem group id (gid) to apply by default when creating
new volumes (ex. '0' for root group) [0]:
Enter unix filesystem permissions to apply by default when creating new
volumes (ex. '0777' for full read/write permissions for all users and
groups) [0777]:
Enter aggregate to use by default when creating new FlexVol volumes:
aff_a400_01_NVME_SSD_1
Enter ONTAP API username (Recommendation: Use SVM account): admin
Enter ONTAP API password (Recommendation: Use SVM account):
Verify SSL certificate when calling ONTAP API (true/false): false
Do you intend to use this toolkit to trigger BlueXP Copy and Sync
operations? (yes/no): no
Do you intend to use this toolkit to push/pull from S3? (yes/no): no
Created config file: '/home/user/.netapp_dataops/config.json'.
```

Create a Guest VM template

Lastly, you must create a VM template based on your guest VM. You will be able to use this template to quickly create guest VMs for utilizing NVIDIA NGC software.

To create a VM template based on your guest VM, log into VMware vSphere, righ-click on the guest VM name, choose 'Clone', choose 'Clone to Template…', and then follow the wizard.



Example Use Case - TensorFlow Training Job

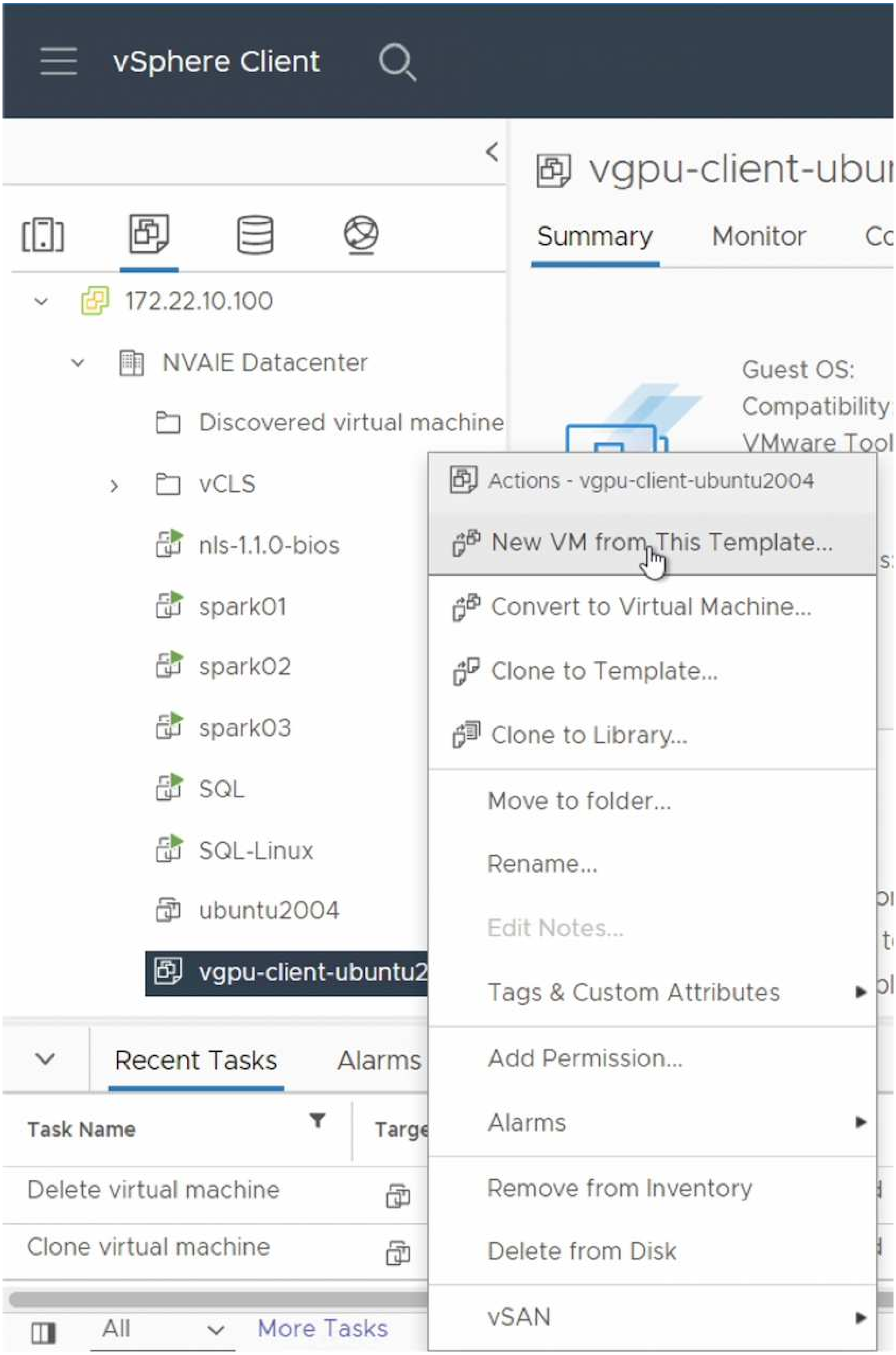
This section describes the tasks that need to be performed in order to execute a TensorFlow training job within an NVIDIA AI Enterprise environment.

Prerequisites

Before you perform the steps that are outlined in this section, we assume that you have already created a guest VM template by following the instructions outlined on the Setup page.

Create Guest VM from Template

First, you must create a new guest VM from the template that you created in the previous section. To create a new guest VM from your template, log into VMware vSphere, righ-click on the template name, choose 'New VM from This Template…', and then follow the wizard.


vSphere Client
vgpu-client-ubun
Summary
Monitor
172.22.10.100
NVAIE Datacenter
Discovered virtual machine
vCLS
nls-1.1.0-bios
spark01
spark02
spark03
SQL
SQL-Linux
ubuntu2004
vgpu-client-ubuntu2
Guest OS:
Actions - vgpu-client-ubuntu2004
New VM from This Template...
Convert to Virtual Machine...
Clone to Template...
Clone to Library...
Move to folder...
Rename...
Edit Notes...
Tags & Custom Attributes
Add Permission...
Alarms
Remove from Inventory
Delete from Disk
vSAN
Recent Tasks
Alarms
Task Name
Delete virtual machine
Clone virtual machine
All
More Tasks

Create and Mount Data Volume

Next, you must create a new data volume on which to store your training dataset. You can quickly create a new data volume using the NetApp DataOps Toolkit. The example command that follows shows the creation of a volume named 'imagenet' with a capacity of 2 TB.

```
$ netapp_dataops_cli.py create vol -n imagenet -s 2TB
```

Before you can populate your data volume with data, you must mount it within the guest VM. You can quickly mount a data volume using the NetApp DataOps Toolkit. The example command that follows shows the mouting of the volume that was created in the previous step.

```
$ sudo -E netapp_dataops_cli.py mount vol -n imagenet -m ~/imagenet
```

Populate Data Volume

After the new volume has been provisioned and mounted, the training dataset can be retrieved from the source location and placed on the new volume. This typically will involve pulling the data from an S3 or Hadoop data lake and sometimes will involve help from a data engineer.

Execute TensorFlow Training Job

Now, you are ready to execute your TensorFlow training job. To execute your TensorFlow training job, perform the following tasks.

1. Pull the NVIDIA NGC enterprise TensorFlow container image.

   ```
   $ sudo docker pull nvcr.io/nvaie/tensorflow-2-1:22.05-tf1-nvaie-2.1-py3
   ```

2. Launch an instance of the NVIDIA NGC enterprise TensorFlow container. Use the '-v' option to attach your data volume to the container.

   ```
   $ sudo docker run --gpus all -v ~/imagenet:/imagenet -it --rm
   nvcr.io/nvaie/tensorflow-2-1:22.05-tf1-nvaie-2.1-py3
   ```

3. Execute your TensorFlow training program within the container. The example command that follows shows the execution of an example ResNet-50 training program that is included in the container image.

   ```
   $ python ./nvidia-examples/cnn/resnet.py --layers 50 -b 64 -i 200 -u
   batch --precision fp16 --data_dir /imagenet/data
   ```

Where to Find Additional Information

To learn more about the information described in this document, refer to the following documents and/or websites:

- NetApp ONTAP data management software — ONTAP information library

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

- NetApp DataOps Toolkit

  https://github.com/NetApp/netapp-dataops-toolkit

- NVIDIA AI Enterprise with VMware

  https://www.nvidia.com/en-us/data-center/products/ai-enterprise/vmware/^]

Acknowledgments

- Bobby Oommen, Sr. Manager, NetApp
- Ramesh Isaac, Systems Administrator, NetApp
- Roney Daniel, Technical Marketing Engineer, NetApp

TR-4851: NetApp StorageGRID data lake for autonomous driving workloads - Solution design

David Arnette, NetApp

TR-4851 demonstrates the use of NetApp StorageGRID object storage as a data repository and management system for machine learning (ML) and deep learning (DL) software development. This paper describes the data flow and requirements in autonomous vehicle software development and the StorageGRID features that streamline the data lifecycle. This solution applies to any multistage data pipeline workflow that is typical in ML and DL development processes.

TR-4851: NetApp StorageGRID data lake for autonomous driving workloads - Solution design

NetApp AI Control Plane

TR-4798: NetApp AI Control Plane

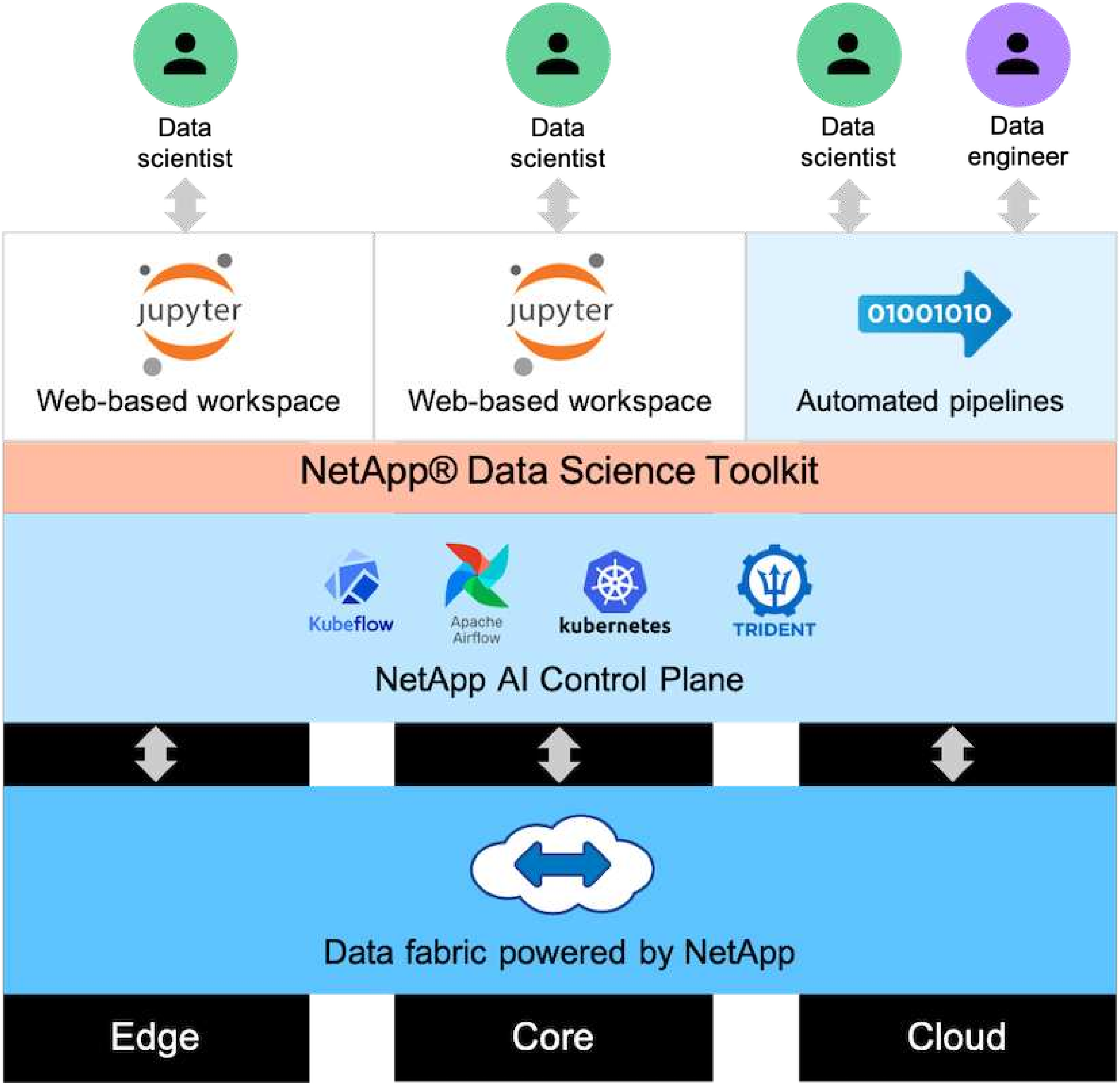
Mike Oglesby, NetApp

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

This report shows you how to rapidly clone a data namespace. It also shows you how to seamlessly replicate data across sites and regions to create a cohesive and unified AI/ML/DL data pipeline. Additionally, it walks you through the defining and implementing of AI, ML, and DL training workflows that incorporate the near-instant creation of data and model baselines for traceability and versioning. With this solution, you can trace every model training run back to the exact dataset that was used to train and/or validate the model. Lastly, this document shows you how to swiftly provision Jupyter Notebook workspaces with access to massive datasets.

Note: For HPC style distributed training at scale involving a large number of GPU servers that require shared access to the same dataset, or if you require/prefer a parallel file system, check out TR-4890. This technical report describes how to include NetApp's fully supported parallel file system solution BeeGFS as part of the NetApp AI Control Plane. This solution is designed to scale from a handful of NVIDIA DGX A100 systems, up to a full blown 140 node SuperPOD.

The NetApp AI Control Plane is targeted towards data scientists and data engineers, and, thus, minimal NetApp or NetApp ONTAP® expertise is required. With this solution, data management functions can be executed using simple and familiar tools and interfaces. If you already have NetApp storage in your environment, you can test drive the NetApp AI Control plane today. If you want to test drive the solution but you do not have already have NetApp storage, visit cloud.netapp.com, and you can be up and running with a cloud-based NetApp storage solution in minutes. The following figure provides a visualization of the solution.



Concepts and Components

Artificial Intelligence

AI is a computer science discipline in which computers are trained to mimic the cognitive functions of the human mind. AI developers train computers to learn and to solve problems in a manner that is similar to, or even superior to, humans. Deep learning and machine learning are subfields of AI. Organizations are increasingly adopting AI, ML, and DL to support their critical business needs. Some examples are as follows:

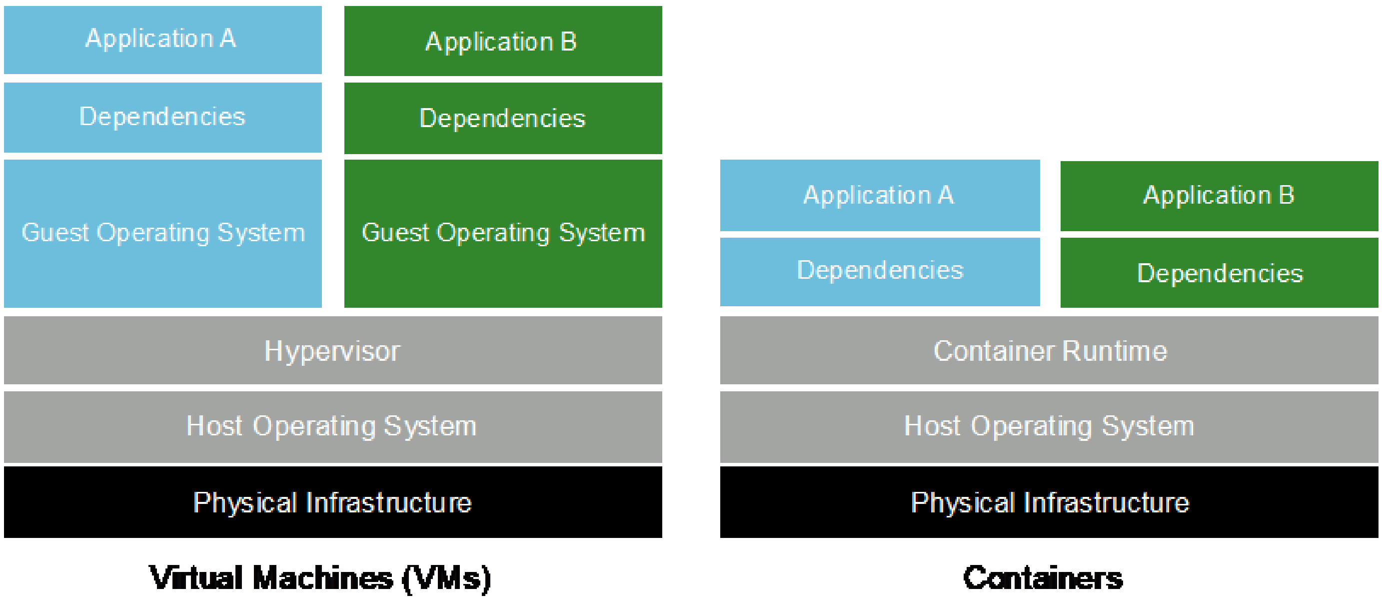
- Analyzing large amounts of data to unearth previously unknown business insights
- Interacting directly with customers by using natural language processing
- Automating various business processes and functions

Modern AI training and inference workloads require massively parallel computing capabilities. Therefore, GPUs are increasingly being used to execute AI operations because the parallel processing capabilities of GPUs are vastly superior to those of general-purpose CPUs.

Containers

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

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



Kubernetes

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

NetApp Trident

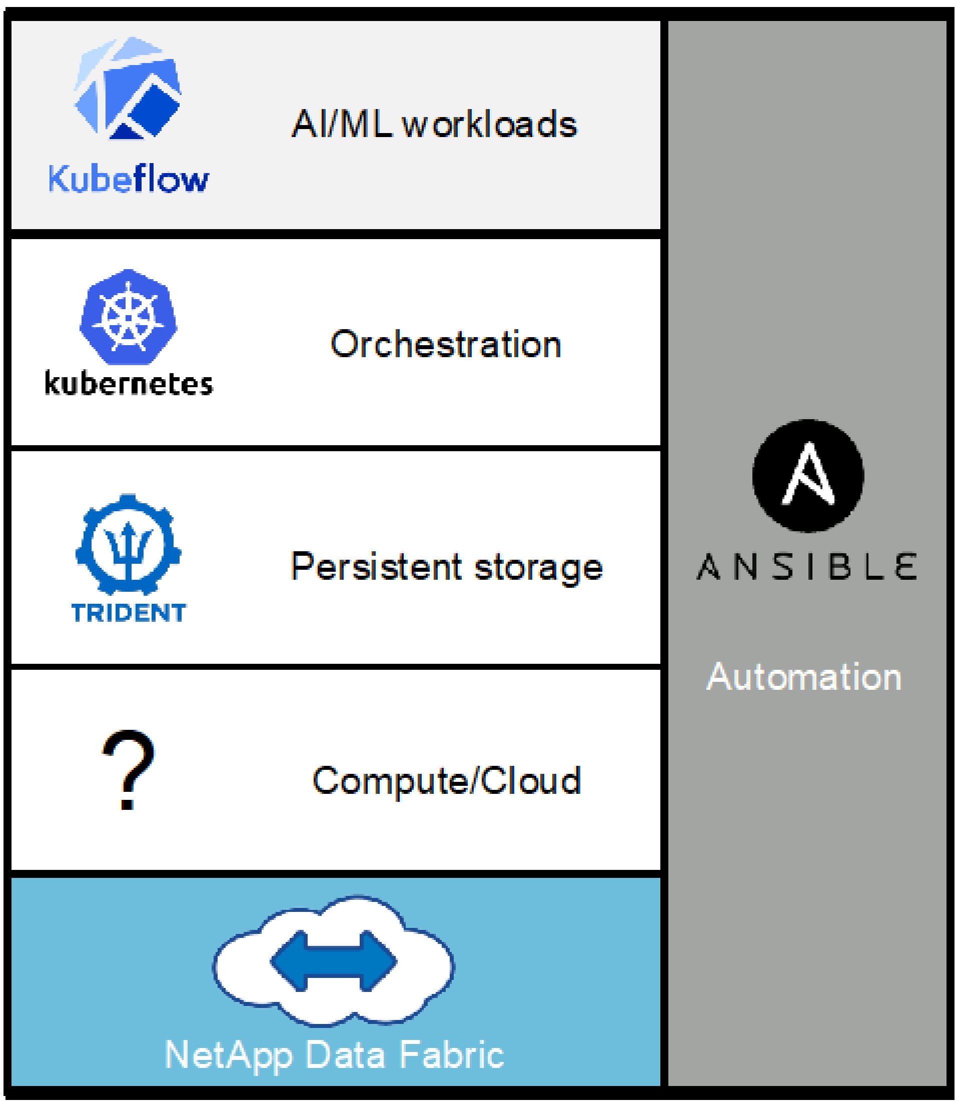
Trident is an open source storage orchestrator developed and maintained by NetApp that greatly simplifies the creation, management, and consumption of persistent storage for Kubernetes workloads. Trident, itself a Kubernetes-native application, runs directly within a Kubernetes cluster. With Trident,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

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.

Kubeflow

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



Kubeflow Pipelines

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

Jupyter Notebook Server

A Jupyter Notebook Server is an open source web application that allows data scientists to create wiki-like documents called Jupyter Notebooks that contain live code as well as descriptive test. Jupyter Notebooks are widely used in the AI and ML community as a means of documenting, storing, and sharing AI and ML projects. Kubeflow simplifies the provisioning and deployment of Jupyter Notebook Servers on Kubernetes. For more information on Jupyter Notebooks, visit the Jupyter website. For more information about Jupyter Notebooks within the context of Kubeflow, see the official Kubeflow documentation.

Apache Airflow

Apache Airflow is an open-source workflow management platform that enables programmatic authoring, scheduling, and monitoring for complex enterprise workflows. It is often used to automate ETL and data pipeline workflows, but it is not limited to these types of workflows. The Airflow project was started by Airbnb but has since become very popular in the industry and now falls under the auspices of The Apache Software Foundation. Airflow is written in Python, Airflow workflows are created via Python scripts, and Airflow is designed under the principle of "configuration as code." Many enterprise Airflow users now run Airflow on top of Kubernetes.

Directed Acyclic Graphs (DAGs)

In Airflow, workflows are called Directed Acyclic Graphs (DAGs). DAGs are made up of tasks that are executed in sequence, in parallel, or a combination of the two, depending on the DAG definition. The Airflow scheduler executes individual tasks on an array of workers, adhering to the task-level dependencies that are specified in the DAG definition. DAGs are defined and created via Python scripts.

NetApp ONTAP 9

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

Simplify Data Management

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

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

Accelerate and Protect Data

ONTAP delivers superior levels of performance and data protection and extends these capabilities with the following features:

- **High performance and low latency.** ONTAP offers the highest possible throughput at the lowest possible latency.

- **NetApp ONTAP FlexGroup technology.** A FlexGroup volume is a high-performance data container that can scale linearly to up to 20PB and 400 billion files, providing a single namespace that simplifies data management.
- **Data protection.** ONTAP provides built-in data protection capabilities with common management across all platforms.
- **NetApp Volume Encryption.** ONTAP offers native volume-level encryption with both onboard and external key management support.

Future-Proof Infrastructure

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

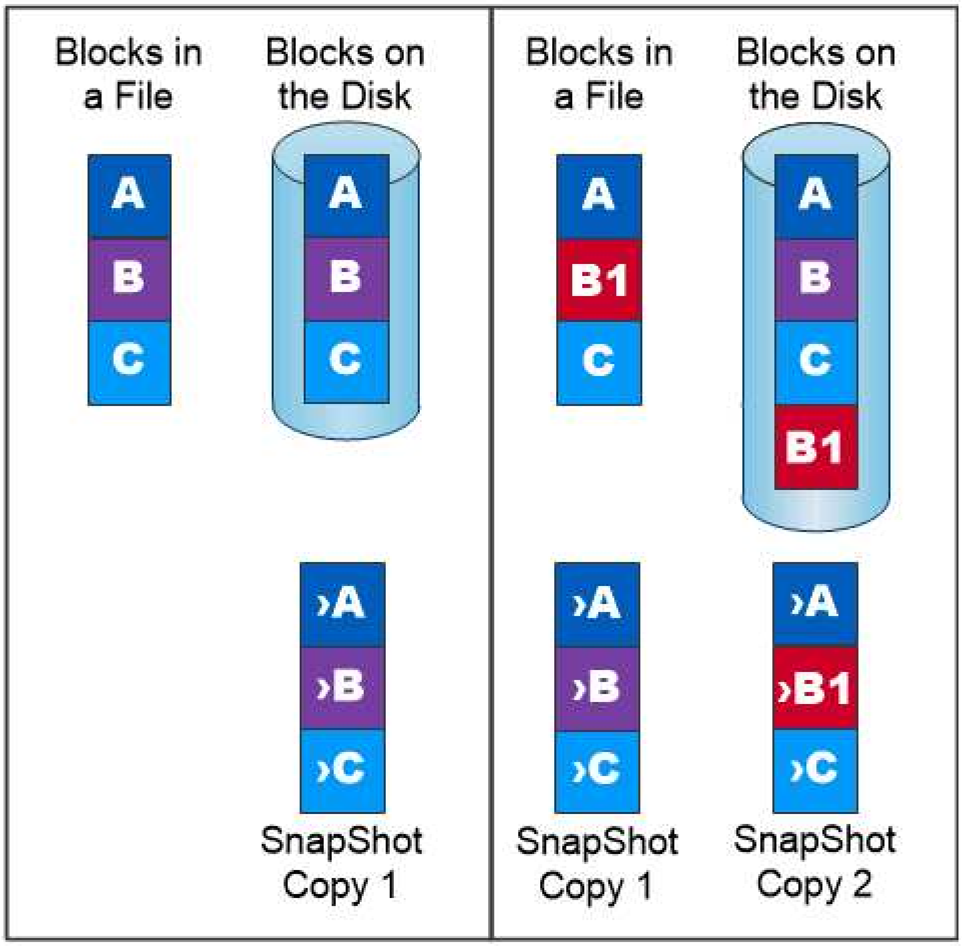
- **Seamless scaling and nondisruptive operations.** ONTAP supports the nondisruptive addition of capacity to existing controllers and to scale-out clusters. You can upgrade to the latest technologies, such as NVMe and 32Gb FC, without costly data migrations or outages.
- **Cloud connection.** ONTAP is one of the most cloud-connected storage management software, with options for software-defined storage (ONTAP Select) and cloud-native instances (NetApp Cloud Volumes Service) in all public clouds.
- **Integration with emerging applications.** By using the same infrastructure that supports existing enterprise apps, ONTAP offers enterprise-grade data services for next-generation platforms and applications such as OpenStack, Hadoop, and MongoDB.

NetApp Snapshot Copies

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

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

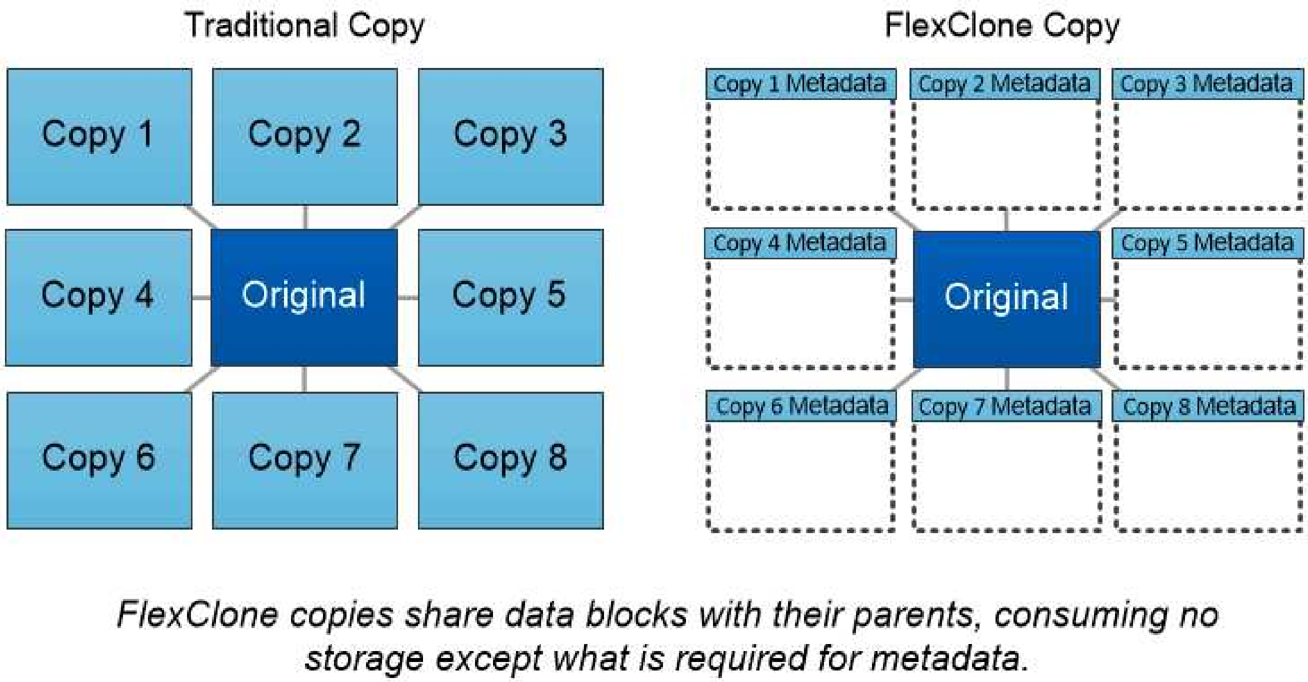
You can use a Snapshot copy to recover individual files or LUNs or to restore the entire contents of a volume. ONTAP compares pointer information in the Snapshot copy with data on disk to reconstruct the missing or damaged object, without downtime or a significant performance cost.



A Snapshot copy records only changes to the active file system since the last Snapshot copy.

NetApp FlexClone Technology

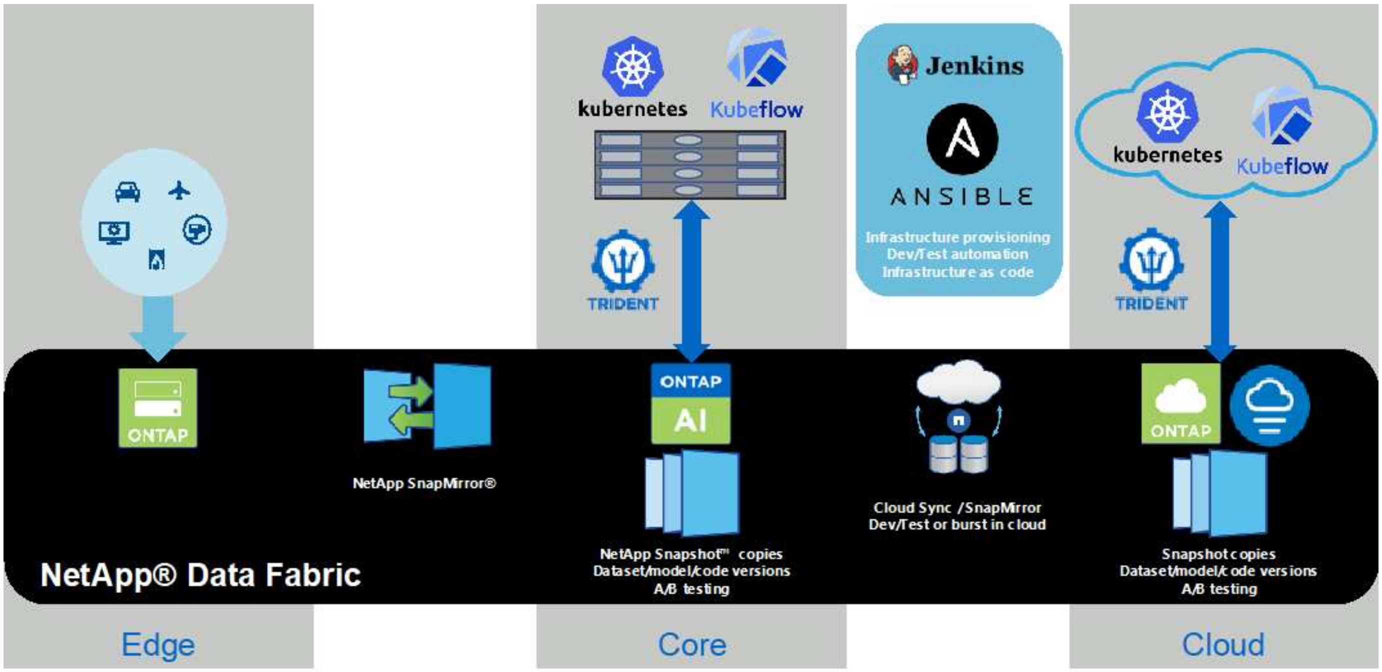
NetApp FlexClone technology references Snapshot metadata to create writable, point-in-time copies of a volume. Copies share data blocks with their parents, consuming no storage except what is required for metadata until changes are written to the copy, as depicted in the following figure. Where traditional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



NetApp SnapMirror Data Replication Technology

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

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



NetApp BlueXP Copy and Sync

BlueXP Copy and Sync is a NetApp service for rapid and secure data synchronization. Whether you need to transfer files between on-premises NFS or SMB file shares, NetApp StorageGRID, NetApp ONTAP S3, NetApp Cloud Volumes Service, Azure NetApp Files, AWS S3, AWS EFS, Azure Blob, Google Cloud Storage, or IBM Cloud Object Storage, BlueXP Copy and Sync moves the files where you need them quickly and securely.

After your data is transferred, it is fully available for use on both source and target. BlueXP Copy and Sync can sync data on-demand when an update is triggered or continuously sync data based on a predefined schedule. Regardless, BlueXP Copy and Sync only moves the deltas, so time and money spent on data replication is minimized.

BlueXP Copy and Sync is a software as a service (SaaS) tool that is extremely simple to set up and use. Data transfers that are triggered by BlueXP Copy and Sync are carried out by data brokers. BlueXP Copy and Sync data brokers can be deployed in AWS, Azure, Google Cloud Platform, or on-premises.

NetApp XCP

NetApp XCP is client-based software for any-to-NetApp and NetApp-to-NetApp data migrations and file system insights. XCP is designed to scale and achieve maximum performance by utilizing all available system resources to handle high-volume datasets and high-performance migrations. XCP helps you to gain complete visibility into the file system with the option to generate reports.

NetApp XCP is available in a single package that supports NFS and SMB protocols. XCP includes a Linux binary for NFS data sets and a windows executable for SMB data sets.

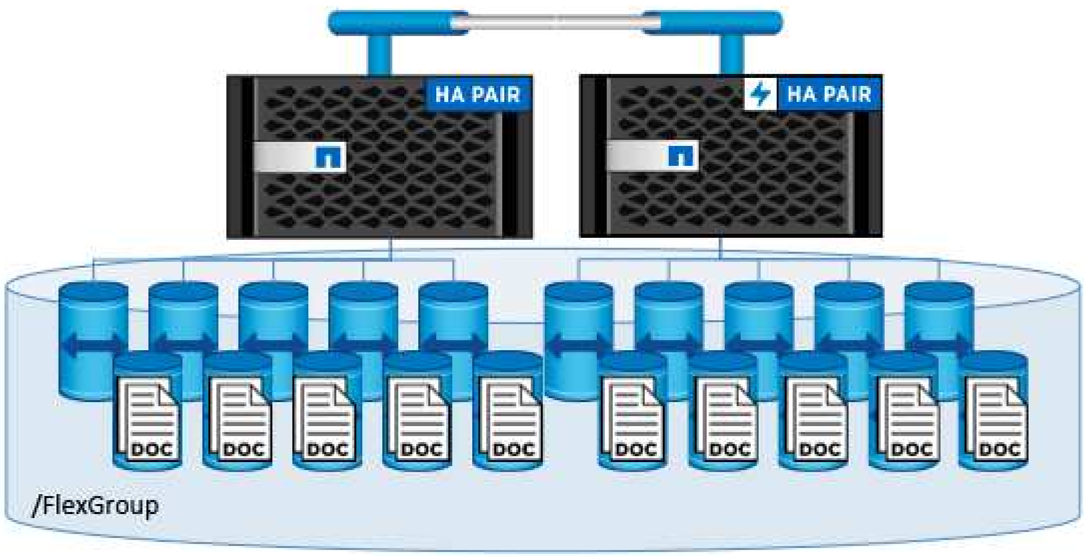
NetApp XCP File Analytics is host-based software that detects file shares, runs scans on the file system, and provides a dashboard for file analytics. XCP File Analytics is compatible with both NetApp and non-NetApp systems and runs on Linux or Windows hosts to provide analytics for NFS and SMB-exported file systems.

NetApp ONTAP FlexGroup Volumes

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

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

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



Hardware and Software Requirements

The NetApp AI Control Plane solution is not dependent on this specific hardware. The solution is compatible with any NetApp physical storage appliance, software-defined instance, or cloud service, that is supported by Trident. Examples include a NetApp AFF storage system, Azure NetApp Files, NetApp Cloud Volumes Service, a NetApp ONTAP Select software-defined storage instance, or a NetApp Cloud Volumes ONTAP instance. Additionally, the solution can be implemented on any Kubernetes cluster as long as the Kubernetes version used is supported by Kubeflow and NetApp Trident. For a list of Kubernetes versions that are supported by Kubeflow, see the see the official Kubeflow documentation. For a list of Kubernetes versions that are supported by Trident, see the Trident documentation. See the following tables for details on the environment that was used to validate the solution.

Infrastructure Component	Quantity	Details	Operating System
Deployment jump host	1	VM	Ubuntu 20.04.2 LTS

Infrastructure Component	Quantity	Details	Operating System
Kubernetes master nodes	1	VM	Ubuntu 20.04.2 LTS
Kubernetes worker nodes	2	VM	Ubuntu 20.04.2 LTS
Kubernetes GPU worker nodes	2	NVIDIA DGX-1 (bare-metal)	NVIDIA DGX OS 4.0.5 (based on Ubuntu 18.04.2 LTS)
Storage	1 HA Pair	NetApp AFF A220	NetApp ONTAP 9.7 P6

Software Component	Version
Apache Airflow	2.0.1
Apache Airflow Helm Chart	8.0.8
Docker	19.03.12
Kubeflow	1.2
Kubernetes	1.18.9
NetApp Trident	21.01.2
NVIDIA DeepOps	Trident deployment functionality from master branch as of commit 61898cdfda; All other functionality from version 21.03

Support

NetApp does not offer enterprise support for Apache Airflow, Docker, Kubeflow, Kubernetes, or NVIDIA DeepOps. If you are interested in a fully supported solution with capabilities similar to the NetApp AI Control Plane solution, contact NetApp about fully supported AI/ML solutions that NetApp offers jointly with partners.

Kubernetes Deployment

This section describes the tasks that you must complete to deploy a Kubernetes cluster in which to implement the NetApp AI Control Plane solution. If you already have a Kubernetes cluster, then you can skip this section as long as you are running a version of Kubernetes that is supported by Kubeflow and NetApp Trident. For a list of Kubernetes versions that are supported by Kubeflow, see the see the official Kubeflow documentation. For a list of Kubernetes versions that are supported by Trident, see the Trident documentation.

For on-premises Kubernetes deployments that incorporate bare-metal nodes featuring NVIDIA GPU(s), NetApp recommends using NVIDIA's DeepOps Kubernetes deployment tool. This section outlines the deployment of a Kubernetes cluster using DeepOps.

Prerequisites

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

1. You have already configured any bare-metal Kubernetes nodes (for example, an NVIDIA DGX system that is part of an ONTAP AI pod) according to standard configuration instructions.
2. You have installed a supported operating system on all Kubernetes master and worker nodes and on a deployment jump host. For a list of operating systems that are supported by DeepOps, see the DeepOps GitHub site.

Use NVIDIA DeepOps to Install and Configure Kubernetes

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

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

NetApp Trident Deployment and Configuration

NetApp Trident Deployment and Configuration

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

Prerequisites

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

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

Install Trident

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

1. Deploy Trident using one of the following methods:
   - If you used NVIDIA DeepOps to deploy your Kubernetes cluster, you can also use NVIDIA DeepOps to deploy Trident in your Kubernetes cluster. To deploy Trident with DeepOps, follow the Trident deployment instructions on the NVIDIA DeepOps GitHub site.
   - If you did not use NVIDIA DeepOps to deploy your Kubernetes cluster or if you simply prefer to deploy Trident manually, you can deploy Trident by following the deployment instructions in the Trident documentation. Be sure to create at least one Trident Backend and at least one Kubernetes StorageClass, for more information about how to configure Backends and StorageClasses see the linked subsections at NetApp Docs.

> If you are deploying the NetApp AI Control Plane solution on an ONTAP AI pod, see Example Trident Backends for ONTAP AI Deployments for some examples of different Trident Backends that you might want to create and Example Kubernetes Storageclasses for ONTAP AI Deployments for some examples of different Kubernetes StorageClasses that you might want to create.

Example Trident Backends for ONTAP AI Deployments

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

1. NetApp recommends creating a FlexGroup-enabled Trident Backend for each data LIF (logical network interface that provides data access) that you want to use on your NetApp AFF system. This will allow you to balance volume mounts across LIFs

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

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

```
$ cat << EOF > ./trident-backend-ontap-ai-flexgroups-iface1.json
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "backendName": "ontap-ai-flexgroups-iface1",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.11.11",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-ontap-ai-flexgroups-
iface1.json -n trident
+----------------------------+---------------------
+--------------------------------------+---------+---------+
|            NAME            |   STORAGE DRIVER    |
UUID                     | STATE   | VOLUMES |
```

```
+------------------------------+---------------------
+--------------------------------------+--------+---------+
| ontap-ai-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-
b263-b6da6dec0bdd | online |       0 |
+------------------------------+---------------------
+--------------------------------------+--------+---------+
$ cat << EOF > ./trident-backend-ontap-ai-flexgroups-iface2.json
{
    "version": 1,
    "storageDriverName": "ontap-nas-flexgroup",
    "backendName": "ontap-ai-flexgroups-iface2",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.12.12",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-ontap-ai-flexgroups-
iface2.json -n trident
+------------------------------+---------------------
+--------------------------------------+--------+---------+
|            NAME            |   STORAGE DRIVER    |
UUID                 | STATE  | VOLUMES |
+------------------------------+---------------------
+--------------------------------------+--------+---------+
| ontap-ai-flexgroups-iface2 | ontap-nas-flexgroup | 61814d48-c770-436b-
9cb4-cf7ee661274d | online |       0 |
+------------------------------+---------------------
+--------------------------------------+--------+---------+
$ tridentctl get backend -n trident
+------------------------------+---------------------
+--------------------------------------+--------+---------+
|            NAME            |   STORAGE DRIVER    |
UUID                 | STATE  | VOLUMES |
+------------------------------+---------------------
+--------------------------------------+--------+---------+
| ontap-ai-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-
b263-b6da6dec0bdd | online |       0 |
| ontap-ai-flexgroups-iface2 | ontap-nas-flexgroup | 61814d48-c770-436b-
9cb4-cf7ee661274d | online |       0 |
+------------------------------+---------------------
+--------------------------------------+--------+---------+
```

2. NetApp also recommends creating one or more FlexVol- enabled Trident Backends. If you use FlexGroup volumes for training dataset storage, you might want to use FlexVol volumes for storing results, output,

debug information, and so on. If you want to use FlexVol volumes, you must create one or more FlexVol-enabled Trident Backends. The example commands that follow show the creation of a single FlexVol-enabled Trident Backend that uses a single data LIF.

```
$ cat << EOF > ./trident-backend-ontap-ai-flexvols.json
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "ontap-ai-flexvols",
    "managementLIF": "10.61.218.100",
    "dataLIF": "192.168.11.11",
    "svm": "ontapai_nfs",
    "username": "admin",
    "password": "ontapai"
}
EOF
$ tridentctl create backend -f ./trident-backend-ontap-ai-flexvols.json -n
trident
+-----------------------------+---------------------
+--------------------------------------+--------+---------+
|            NAME             |   STORAGE DRIVER    |                 UUID
| STATE  | VOLUMES |
+-----------------------------+---------------------
+--------------------------------------+--------+---------+
| ontap-ai-flexvols           | ontap-nas           | 52bdb3b1-13a5-4513-
a9c1-52a69657fabe | online |       0 |
+-----------------------------+---------------------
+--------------------------------------+--------+---------+
$ tridentctl get backend -n trident
+-----------------------------+---------------------
+--------------------------------------+--------+---------+
|            NAME             |   STORAGE DRIVER    |                 UUID
| STATE  | VOLUMES |
+-----------------------------+---------------------
+--------------------------------------+--------+---------+
| ontap-ai-flexvols           | ontap-nas           | 52bdb3b1-13a5-4513-
a9c1-52a69657fabe | online |       0 |
| ontap-ai-flexgroups-iface1 | ontap-nas-flexgroup | b74cbddb-e0b8-40b7-
b263-b6da6dec0bdd | online |       0 |
| ontap-ai-flexgroups-iface2 | ontap-nas-flexgroup | 61814d48-c770-436b-
9cb4-cf7ee661274d | online |       0 |
+-----------------------------+---------------------
+--------------------------------------+--------+---------+
```

Example Kubernetes StorageClasses for ONTAP AI Deployments

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

1. NetApp recommends creating a separate StorageClass for each FlexGroup-enabled Trident Backend that you created in the section Example Trident Backends for ONTAP AI Deployments, step 1. These granular StorageClasses enable you to add NFS mounts that correspond to specific LIFs (the LIFs that you specified when you created the Trident Backends) as a particular Backend that is specified in the StorageClass spec file. The example commands that follow show the creation of two StorageClasses that correspond to the two example Backends that were created in the section Example Trident Backends for ONTAP AI Deployments, step 1. For more information about StorageClasses, see the Trident documentation.

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

```
$ cat << EOF > ./storage-class-ontap-ai-flexgroups-retain-iface1.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexgroups-retain-iface1
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas-flexgroup"
  storagePools: "ontap-ai-flexgroups-iface1:.*"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexgroups-retain-
iface1.yaml
storageclass.storage.k8s.io/ontap-ai-flexgroups-retain-iface1 created
$ cat << EOF > ./storage-class-ontap-ai-flexgroups-retain-iface2.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexgroups-retain-iface2
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas-flexgroup"
  storagePools: "ontap-ai-flexgroups-iface2:.*"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexgroups-retain-
iface2.yaml
storageclass.storage.k8s.io/ontap-ai-flexgroups-retain-iface2 created
$ kubectl get storageclass
NAME                                PROVISIONER         AGE
ontap-ai-flexgroups-retain-iface1   netapp.io/trident   0m
ontap-ai-flexgroups-retain-iface2   netapp.io/trident   0m
```

2. NetApp also recommends creating a StorageClass that corresponds to the FlexVol-enabled Trident Backend that you created in the section Example Trident Backends for ONTAP AI Deployments, step 2. The example commands that follow show the creation of a single StorageClass for FlexVol volumes.

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

```
$ cat << EOF > ./storage-class-ontap-ai-flexvols-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexvols-retain
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexvols-retain.yaml
storageclass.storage.k8s.io/ontap-ai-flexvols-retain created
$ kubectl get storageclass
NAME                                PROVISIONER         AGE
ontap-ai-flexgroups-retain-iface1   netapp.io/trident   1m
ontap-ai-flexgroups-retain-iface2   netapp.io/trident   1m
ontap-ai-flexvols-retain            netapp.io/trident   0m
```

3. NetApp also recommends creating a generic StorageClass for FlexGroup volumes. The following example commands show the creation of a single generic StorageClass for FlexGroup volumes.

   Note that a particular backend is not specified in the StorageClass definition file. Therefore, when you use Kubernetes to administer volumes that use this StorageClass, Trident attempts to use any available backend that uses the `ontap-nas-flexgroup` driver.

```
$ cat << EOF > ./storage-class-ontap-ai-flexgroups-retain.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-ai-flexgroups-retain
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas-flexgroup"
reclaimPolicy: Retain
EOF
$ kubectl create -f ./storage-class-ontap-ai-flexgroups-retain.yaml
storageclass.storage.k8s.io/ontap-ai-flexgroups-retain created
$ kubectl get storageclass
NAME                                PROVISIONER         AGE
ontap-ai-flexgroups-retain          netapp.io/trident   0m
ontap-ai-flexgroups-retain-iface1   netapp.io/trident   2m
ontap-ai-flexgroups-retain-iface2   netapp.io/trident   2m
ontap-ai-flexvols-retain            netapp.io/trident   1m
```

Kubeflow Deployment

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

Prerequisites

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

1. You already have a working Kubernetes cluster, and you are running a version of Kubernetes that is supported by Kubeflow. For a list of supported versions, see the official Kubeflow documentation.
2. You have already installed and configured NetApp Trident in your Kubernetes cluster as outlined in Trident Deployment and Configuration.

Set Default Kubernetes StorageClass

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

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

> The `ontap-nas-flexgroup` Trident Backend type has a minimum PVC size that is fairly large. By default, Kubeflow attempts to provision PVCs that are only a few GBs in size. Therefore, you should not designate a StorageClass that utilizes the `ontap-nas-flexgroup` Backend type as the default StorageClass for the purposes of Kubeflow deployment.

```
$ kubectl get sc
NAME                                  PROVISIONER               AGE
ontap-ai-flexgroups-retain            csi.trident.netapp.io     25h
ontap-ai-flexgroups-retain-iface1     csi.trident.netapp.io     25h
ontap-ai-flexgroups-retain-iface2     csi.trident.netapp.io     25h
ontap-ai-flexvols-retain              csi.trident.netapp.io     3s
$ kubectl patch storageclass ontap-ai-flexvols-retain -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
storageclass.storage.k8s.io/ontap-ai-flexvols-retain patched
$ kubectl get sc
NAME                                   PROVISIONER               AGE
ontap-ai-flexgroups-retain             csi.trident.netapp.io     25h
ontap-ai-flexgroups-retain-iface1      csi.trident.netapp.io     25h
ontap-ai-flexgroups-retain-iface2      csi.trident.netapp.io     25h
ontap-ai-flexvols-retain (default)     csi.trident.netapp.io     54s
```

Use NVIDIA DeepOps to Deploy Kubeflow

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

> Alternatively, you can deploy Kubeflow manually by following the installation instructions in the official Kubeflow documentation

1. Deploy Kubeflow in your cluster by following the Kubeflow deployment instructions on the NVIDIA DeepOps GitHub site.
2. Note down the Kubeflow Dashboard URL that the DeepOps Kubeflow deployment tool outputs.

```
$ ./scripts/k8s/deploy_kubeflow.sh -x
…
INFO[0007] Applied the configuration Successfully!
filename="cmd/apply.go:72"
Kubeflow app installed to: /home/ai/kubeflow
It may take several minutes for all services to start. Run 'kubectl get
pods -n kubeflow' to verify
To remove (excluding CRDs, istio, auth, and cert-manager), run:
./scripts/k8s_deploy_kubeflow.sh -d
To perform a full uninstall : ./scripts/k8s_deploy_kubeflow.sh -D
Kubeflow Dashboard (HTTP NodePort): http://10.61.188.111:31380
```

3. Confirm that all pods deployed within the Kubeflow namespace show a `STATUS` of `Running` and confirm that no components deployed within the namespace are in an error state. It may take several minutes for all pods to start.

```
$ kubectl get all -n kubeflow
NAME                                                            READY
STATUS    RESTARTS   AGE
pod/admission-webhook-bootstrap-stateful-set-0                  1/1
Running   0          95s
pod/admission-webhook-deployment-6b89c84c98-vrtbh               1/1
Running   0          91s
pod/application-controller-stateful-set-0                       1/1
Running   0          98s
pod/argo-ui-5dcf5d8b4f-m2wn4                                    1/1
Running   0          97s
pod/centraldashboard-cf4874ddc-7hcr8                            1/1
Running   0          97s
pod/jupyter-web-app-deployment-685b455447-gjhh7                 1/1
Running   0          96s
pod/katib-controller-88c97d85c-kgq66                            1/1
Running   1          95s
```

```
pod/katib-db-8598468fd8-5jw2c                                     1/1
Running   0          95s
pod/katib-manager-574c8c67f9-wtrf5                                1/1
Running   1          95s
pod/katib-manager-rest-778857c989-fjbzn                           1/1
Running   0          95s
pod/katib-suggestion-bayesianoptimization-65df4d7455-qthmw        1/1
Running   0          94s
pod/katib-suggestion-grid-56bf69f597-98vwn                        1/1
Running   0          94s
pod/katib-suggestion-hyperband-7777b76cb9-9v6dq                   1/1
Running   0          93s
pod/katib-suggestion-nasrl-77f6f9458c-2qzxq                       1/1
Running   0          93s
pod/katib-suggestion-random-77b88b5c79-l64j9                      1/1
Running   0          93s
pod/katib-ui-7587c5b967-nd629                                     1/1
Running   0          95s
pod/metacontroller-0                                              1/1
Running   0          96s
pod/metadata-db-5dd459cc-swzkm                                    1/1
Running   0          94s
pod/metadata-deployment-6cf77db994-69fk7                          1/1
Running   3          93s
pod/metadata-deployment-6cf77db994-mpbjt                          1/1
Running   3          93s
pod/metadata-deployment-6cf77db994-xg7tz                          1/1
Running   3          94s
pod/metadata-ui-78f5b59b56-qb6kr                                  1/1
Running   0          94s
pod/minio-758b769d67-llvdr                                        1/1
Running   0          91s
pod/ml-pipeline-5875b9db95-g8t2k                                  1/1
Running   0          91s
pod/ml-pipeline-persistenceagent-9b69ddd46-bt9r9                  1/1
Running   0          90s
pod/ml-pipeline-scheduledworkflow-7b8d756c76-7x56s                1/1
Running   0          90s
pod/ml-pipeline-ui-79ffd9c76-fcwpd                                1/1
Running   0          90s
pod/ml-pipeline-viewer-controller-deployment-5fdc87f58-b2t9r      1/1
Running   0          90s
pod/mysql-657f87857d-l5k9z                                        1/1
Running   0          91s
pod/notebook-controller-deployment-56b4f59bbf-8bvnr               1/1
Running   0          92s
```

```
pod/profiles-deployment-6bc745947-mrdkh                        2/2
Running   0          90s
pod/pytorch-operator-77c97f4879-hmlrv                          1/1
Running   0          92s
pod/seldon-operator-controller-manager-0                       1/1
Running   1          91s
pod/spartakus-volunteer-5fdfddb779-l7qkm                       1/1
Running   0          92s
pod/tensorboard-6544748d94-nh8b2                               1/1
Running   0          92s
pod/tf-job-dashboard-56f79c59dd-6w59t                          1/1
Running   0          92s
pod/tf-job-operator-79cbfd6dbc-rb58c                           1/1
Running   0          91s
pod/workflow-controller-db644d554-cwrnb                        1/1
Running   0          97s
NAME                                                 TYPE
CLUSTER-IP      EXTERNAL-IP   PORT(S)              AGE
service/admission-webhook-service                    ClusterIP
10.233.51.169   <none>        443/TCP              97s
service/application-controller-service               ClusterIP
10.233.4.54     <none>        443/TCP              98s
service/argo-ui                                      NodePort
10.233.47.191   <none>        80:31799/TCP         97s
service/centraldashboard                             ClusterIP
10.233.8.36     <none>        80/TCP               97s
service/jupyter-web-app-service                      ClusterIP
10.233.1.42     <none>        80/TCP               97s
service/katib-controller                             ClusterIP
10.233.25.226   <none>        443/TCP              96s
service/katib-db                                     ClusterIP
10.233.33.151   <none>        3306/TCP             97s
service/katib-manager                                ClusterIP
10.233.46.239   <none>        6789/TCP             96s
service/katib-manager-rest                           ClusterIP
10.233.55.32    <none>        80/TCP               96s
service/katib-suggestion-bayesianoptimization        ClusterIP
10.233.49.191   <none>        6789/TCP             95s
service/katib-suggestion-grid                        ClusterIP
10.233.9.105    <none>        6789/TCP             95s
service/katib-suggestion-hyperband                   ClusterIP
10.233.22.2     <none>        6789/TCP             95s
service/katib-suggestion-nasrl                       ClusterIP
10.233.63.73    <none>        6789/TCP             95s
service/katib-suggestion-random                      ClusterIP
10.233.57.210   <none>        6789/TCP             95s
```

```
service/katib-ui                                          ClusterIP
10.233.6.116    <none>        80/TCP              96s
service/metadata-db                                       ClusterIP
10.233.31.2     <none>        3306/TCP            96s
service/metadata-service                                  ClusterIP
10.233.27.104   <none>        8080/TCP            96s
service/metadata-ui                                       ClusterIP
10.233.57.177   <none>        80/TCP              96s
service/minio-service                                     ClusterIP
10.233.44.90    <none>        9000/TCP            94s
service/ml-pipeline                                       ClusterIP
10.233.41.201   <none>        8888/TCP,8887/TCP   94s
service/ml-pipeline-tensorboard-ui                        ClusterIP
10.233.36.207   <none>        80/TCP              93s
service/ml-pipeline-ui                                    ClusterIP
10.233.61.150   <none>        80/TCP              93s
service/mysql                                             ClusterIP
10.233.55.117   <none>        3306/TCP            94s
service/notebook-controller-service                       ClusterIP
10.233.10.166   <none>        443/TCP             95s
service/profiles-kfam                                     ClusterIP
10.233.33.79    <none>        8081/TCP            92s
service/pytorch-operator                                  ClusterIP
10.233.37.112   <none>        8443/TCP            95s
service/seldon-operator-controller-manager-service   ClusterIP
10.233.30.178   <none>        443/TCP             92s
service/tensorboard                                       ClusterIP
10.233.58.151   <none>        9000/TCP            94s
service/tf-job-dashboard                                  ClusterIP
10.233.4.17     <none>        80/TCP              94s
service/tf-job-operator                                   ClusterIP
10.233.60.32    <none>        8443/TCP            94s
service/webhook-server-service                            ClusterIP
10.233.32.167   <none>        443/TCP             87s
NAME                                                            READY   UP-
TO-DATE   AVAILABLE   AGE
deployment.apps/admission-webhook-deployment                    1/1     1
1           97s
deployment.apps/argo-ui                                         1/1     1
1           97s
deployment.apps/centraldashboard                                1/1     1
1           97s
deployment.apps/jupyter-web-app-deployment                      1/1     1
1           97s
deployment.apps/katib-controller                                1/1     1
1           96s
```

```
deployment.apps/katib-db                                     1/1     1
1           97s
deployment.apps/katib-manager                                1/1     1
1           96s
deployment.apps/katib-manager-rest                           1/1     1
1           96s
deployment.apps/katib-suggestion-bayesianoptimization        1/1     1
1           95s
deployment.apps/katib-suggestion-grid                        1/1     1
1           95s
deployment.apps/katib-suggestion-hyperband                   1/1     1
1           95s
deployment.apps/katib-suggestion-nasrl                       1/1     1
1           95s
deployment.apps/katib-suggestion-random                      1/1     1
1           95s
deployment.apps/katib-ui                                     1/1     1
1           96s
deployment.apps/metadata-db                                  1/1     1
1           96s
deployment.apps/metadata-deployment                          3/3     3
3           96s
deployment.apps/metadata-ui                                  1/1     1
1           96s
deployment.apps/minio                                        1/1     1
1           94s
deployment.apps/ml-pipeline                                  1/1     1
1           94s
deployment.apps/ml-pipeline-persistenceagent                 1/1     1
1           93s
deployment.apps/ml-pipeline-scheduledworkflow                1/1     1
1           93s
deployment.apps/ml-pipeline-ui                               1/1     1
1           93s
deployment.apps/ml-pipeline-viewer-controller-deployment     1/1     1
1           93s
deployment.apps/mysql                                        1/1     1
1           94s
deployment.apps/notebook-controller-deployment               1/1     1
1           95s
deployment.apps/profiles-deployment                          1/1     1
1           92s
deployment.apps/pytorch-operator                             1/1     1
1           95s
deployment.apps/spartakus-volunteer                          1/1     1
1           94s
```

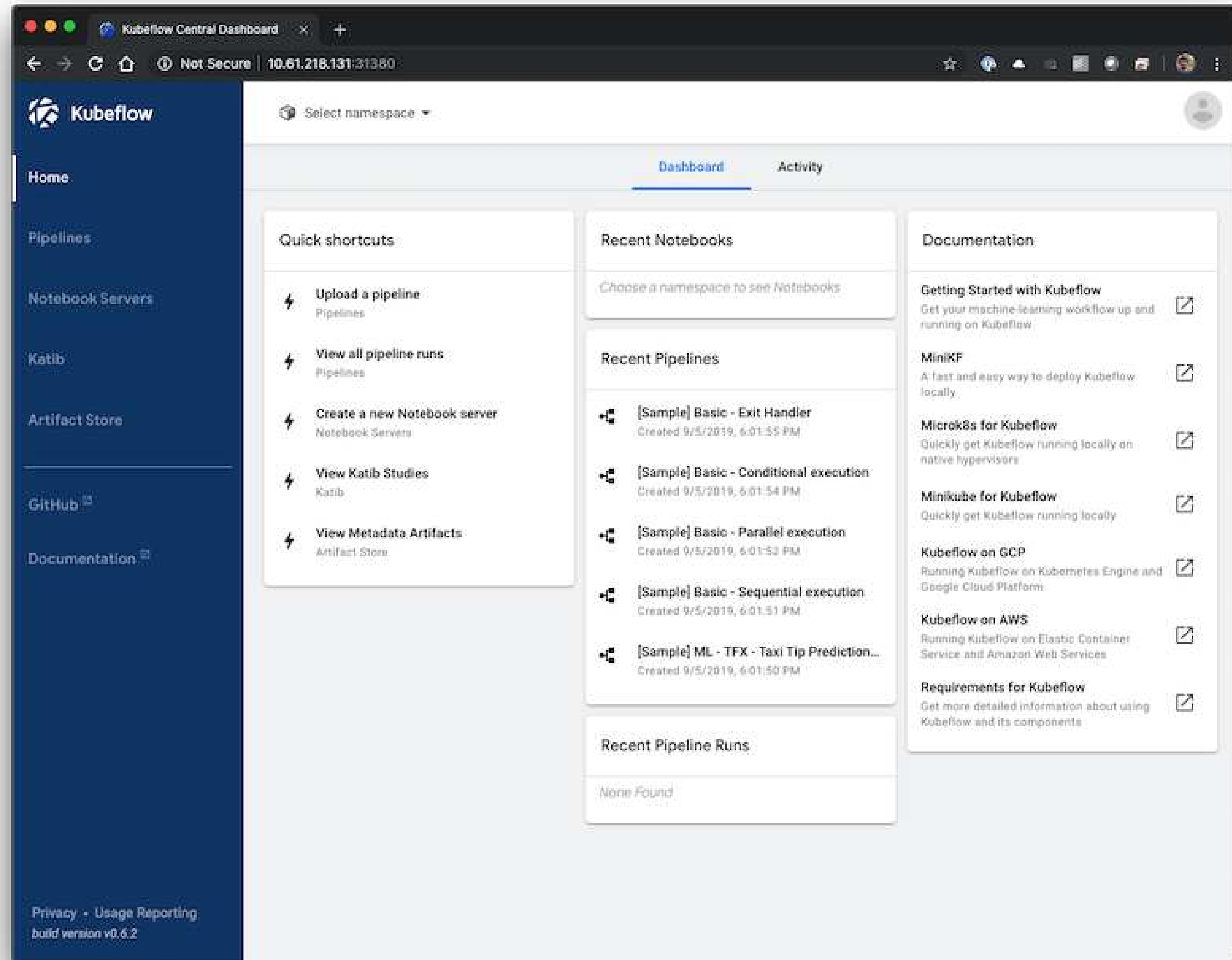
```
deployment.apps/tensorboard                                 1/1     1
1           94s
deployment.apps/tf-job-dashboard                            1/1     1
1           94s
deployment.apps/tf-job-operator                             1/1     1
1           94s
deployment.apps/workflow-controller                         1/1     1
1           97s
NAME
DESIRED   CURRENT   READY   AGE
replicaset.apps/admission-webhook-deployment-6b89c84c98               1
1         1       97s
replicaset.apps/argo-ui-5dcf5d8b4f                                    1
1         1       97s
replicaset.apps/centraldashboard-cf4874ddc                            1
1         1       97s
replicaset.apps/jupyter-web-app-deployment-685b455447                 1
1         1       97s
replicaset.apps/katib-controller-88c97d85c                            1
1         1       96s
replicaset.apps/katib-db-8598468fd8                                   1
1         1       97s
replicaset.apps/katib-manager-574c8c67f9                              1
1         1       96s
replicaset.apps/katib-manager-rest-778857c989                         1
1         1       96s
replicaset.apps/katib-suggestion-bayesianoptimization-65df4d7455     1
1         1       95s
replicaset.apps/katib-suggestion-grid-56bf69f597                      1
1         1       95s
replicaset.apps/katib-suggestion-hyperband-7777b76cb9                 1
1         1       95s
replicaset.apps/katib-suggestion-nasrl-77f6f9458c                     1
1         1       95s
replicaset.apps/katib-suggestion-random-77b88b5c79                    1
1         1       95s
replicaset.apps/katib-ui-7587c5b967                                   1
1         1       96s
replicaset.apps/metadata-db-5dd459cc                                  1
1         1       96s
replicaset.apps/metadata-deployment-6cf77db994                        3
3         3       96s
replicaset.apps/metadata-ui-78f5b59b56                                1
1         1       96s
replicaset.apps/minio-758b769d67                                      1
1         1       93s
```

```
replicaset.apps/ml-pipeline-5875b9db95                                  1
1         1       93s
replicaset.apps/ml-pipeline-persistenceagent-9b69ddd46                  1
1         1       92s
replicaset.apps/ml-pipeline-scheduledworkflow-7b8d756c76                1
1         1       91s
replicaset.apps/ml-pipeline-ui-79ffd9c76                                1
1         1       91s
replicaset.apps/ml-pipeline-viewer-controller-deployment-5fdc87f58      1
1         1       91s
replicaset.apps/mysql-657f87857d                                        1
1         1       92s
replicaset.apps/notebook-controller-deployment-56b4f59bbf               1
1         1       94s
replicaset.apps/profiles-deployment-6bc745947                           1
1         1       91s
replicaset.apps/pytorch-operator-77c97f4879                             1
1         1       94s
replicaset.apps/spartakus-volunteer-5fdfddb779                          1
1         1       94s
replicaset.apps/tensorboard-6544748d94                                  1
1         1       93s
replicaset.apps/tf-job-dashboard-56f79c59dd                             1
1         1       93s
replicaset.apps/tf-job-operator-79cbfd6dbc                              1
1         1       93s
replicaset.apps/workflow-controller-db644d554                           1
1         1       97s
NAME                                                        READY   AGE
statefulset.apps/admission-webhook-bootstrap-stateful-set   1/1     97s
statefulset.apps/application-controller-stateful-set        1/1     98s
statefulset.apps/metacontroller                             1/1     98s
statefulset.apps/seldon-operator-controller-manager         1/1     92s
$ kubectl get pvc -n kubeflow
NAME             STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS               AGE
katib-mysql      Bound    pvc-b07f293e-d028-11e9-9b9d-00505681a82d
10Gi       RWO            ontap-ai-flexvols-retain   27m
metadata-mysql   Bound    pvc-b0f3f032-d028-11e9-9b9d-00505681a82d
10Gi       RWO            ontap-ai-flexvols-retain   27m
minio-pv-claim   Bound    pvc-b22727ee-d028-11e9-9b9d-00505681a82d
20Gi       RWO            ontap-ai-flexvols-retain   27m
mysql-pv-claim   Bound    pvc-b2429afd-d028-11e9-9b9d-00505681a82d
20Gi       RWO            ontap-ai-flexvols-retain   27m
```

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

down in step 2.

The default username is `admin@kubeflow.org`, and the default password is `12341234`. To create additional users, follow the instructions in the official Kubeflow documentation.



Example Kubeflow Operations and Tasks

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

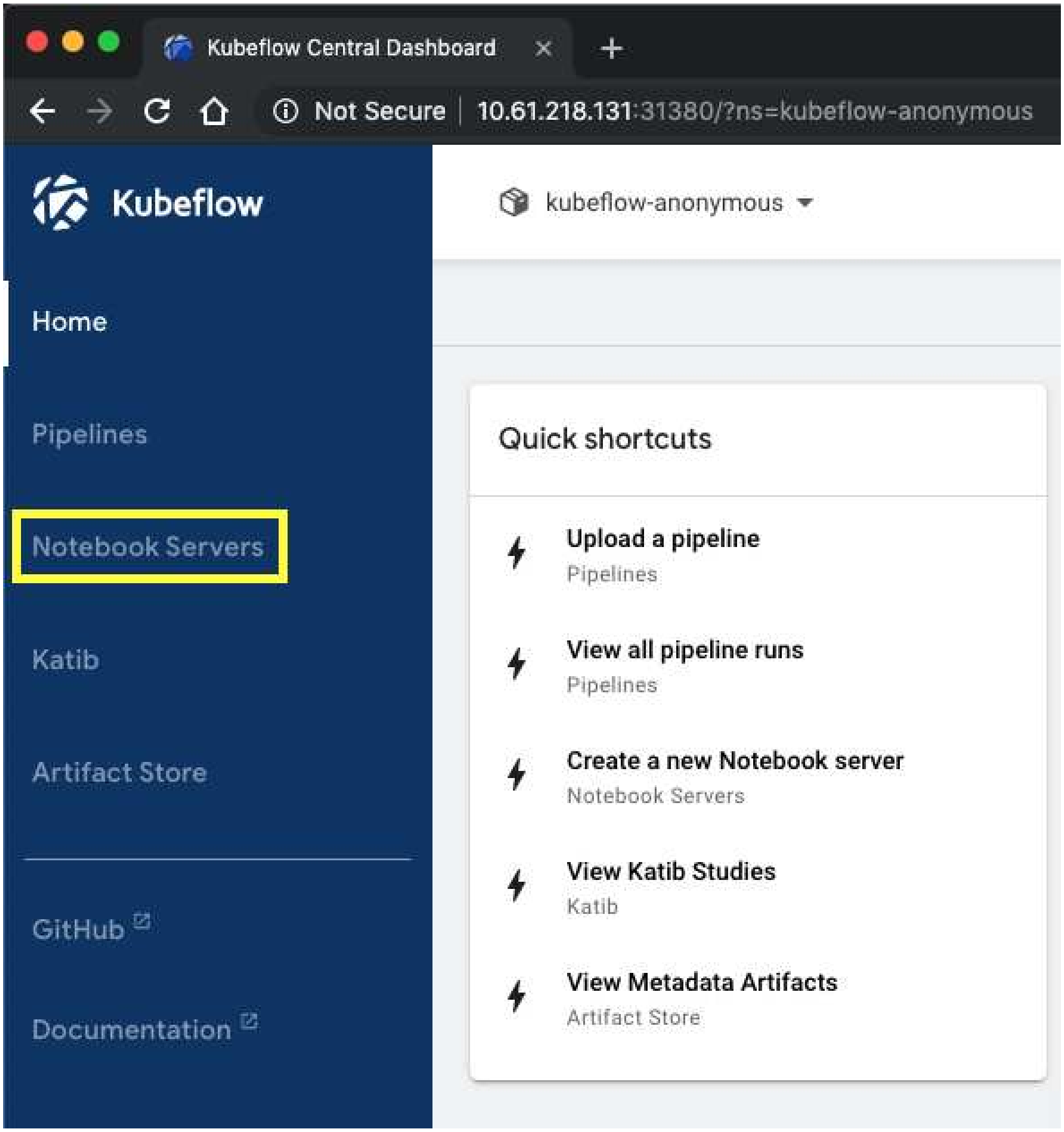
Example Kubeflow Operations and Tasks

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

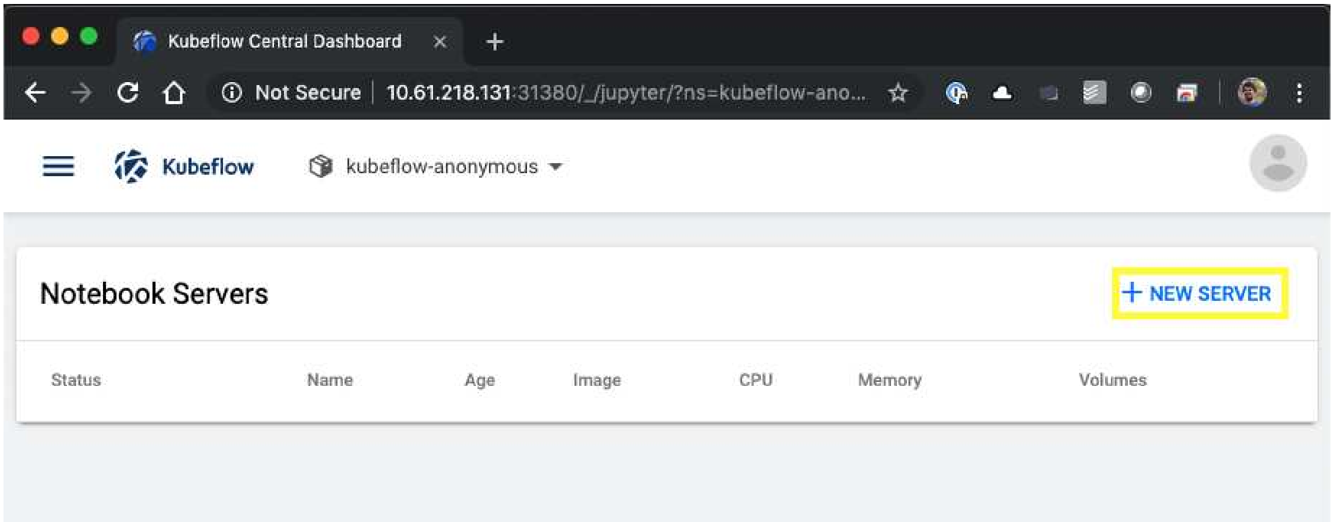
Provision a Jupyter Notebook Workspace for Data Scientist or Developer Use

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

1. From the Kubeflow central dashboard, click Notebook Servers in the main menu to navigate to the Jupyter Notebook server administration page.

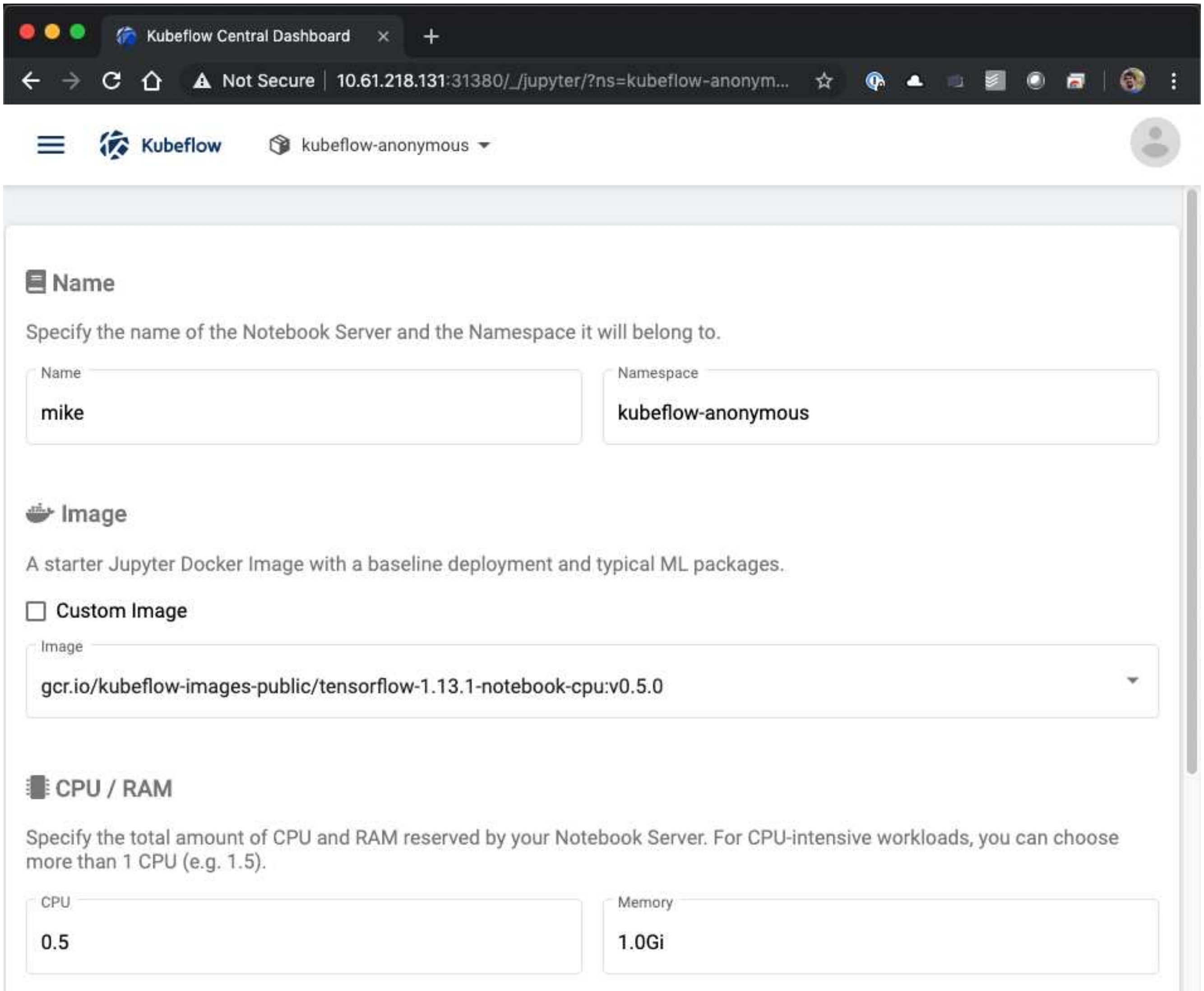


2. Click New Server to provision a new Jupyter Notebook server.

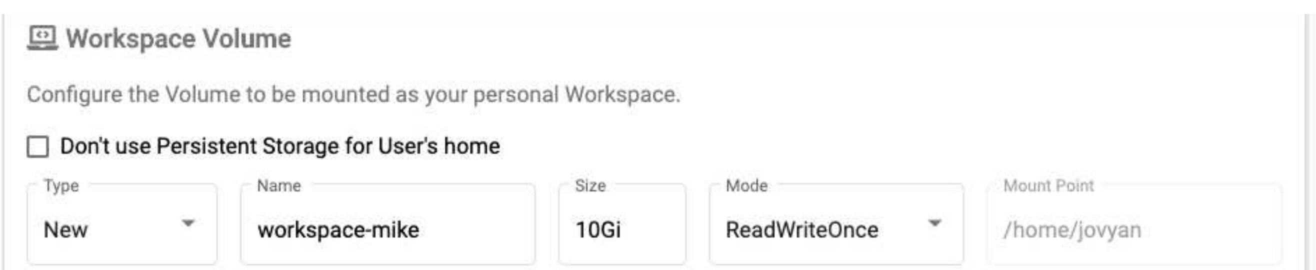


3. Give your new server a name, choose the Docker image that you want your server to be based on, and specify the amount of CPU and RAM to be reserved by your server. If the Namespace field is blank, use the Select Namespace menu in the page header to choose a namespace. The Namespace field is then auto-populated with the chosen namespace.

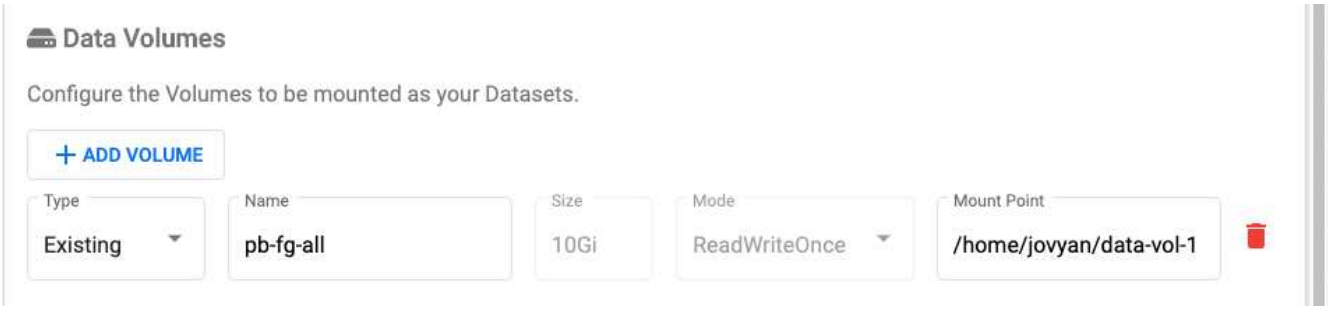
   In the following example, the `kubeflow-anonymous` namespace is chosen. In addition, the default values for Docker image, CPU, and RAM are accepted.



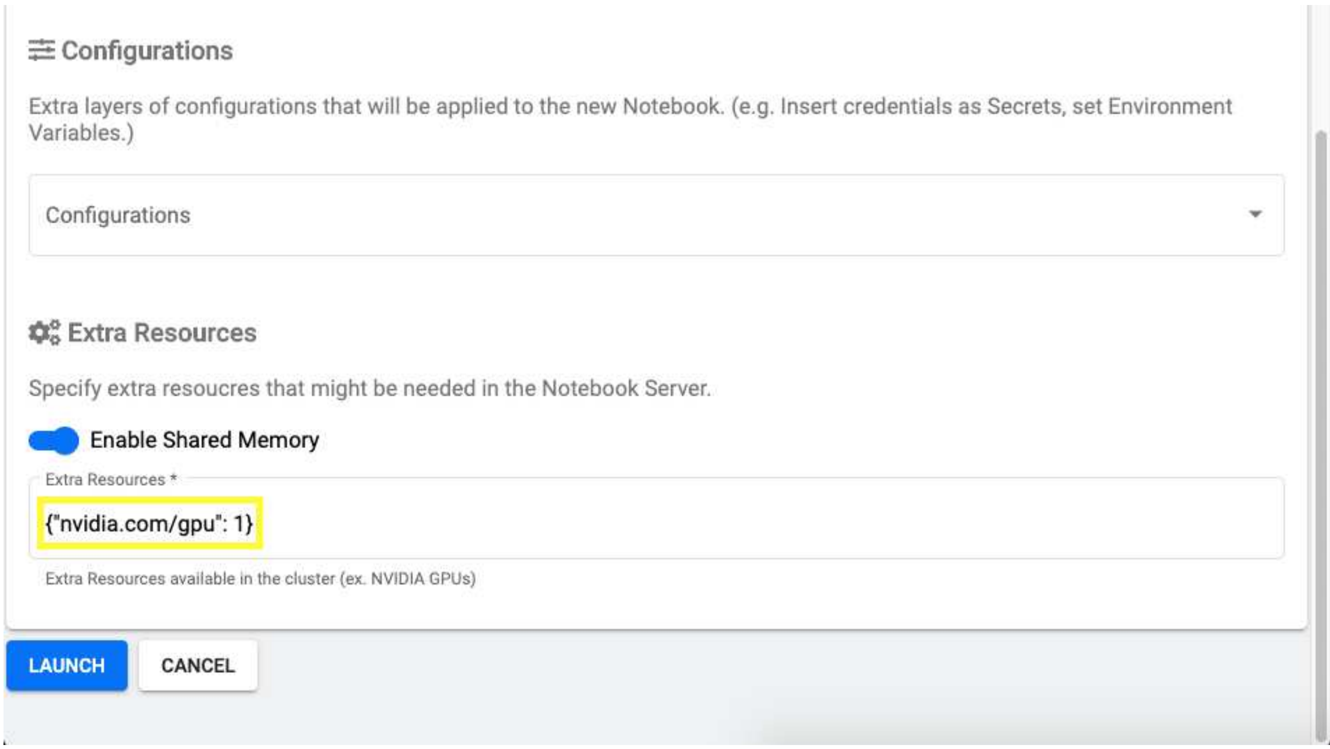
4. Specify the workspace volume details. If you choose to create a new volume, then that volume or PVC is provisioned using the default StorageClass. Because a StorageClass utilizing Trident was designated as the default StorageClass in the section Kubeflow Deployment, the volume or PVC is provisioned with Trident. This volume is automatically mounted as the default workspace within the Jupyter Notebook Server container. Any notebooks that a user creates on the server that are not saved to a separate data volume are automatically saved to this workspace volume. Therefore, the notebooks are persistent across reboots.



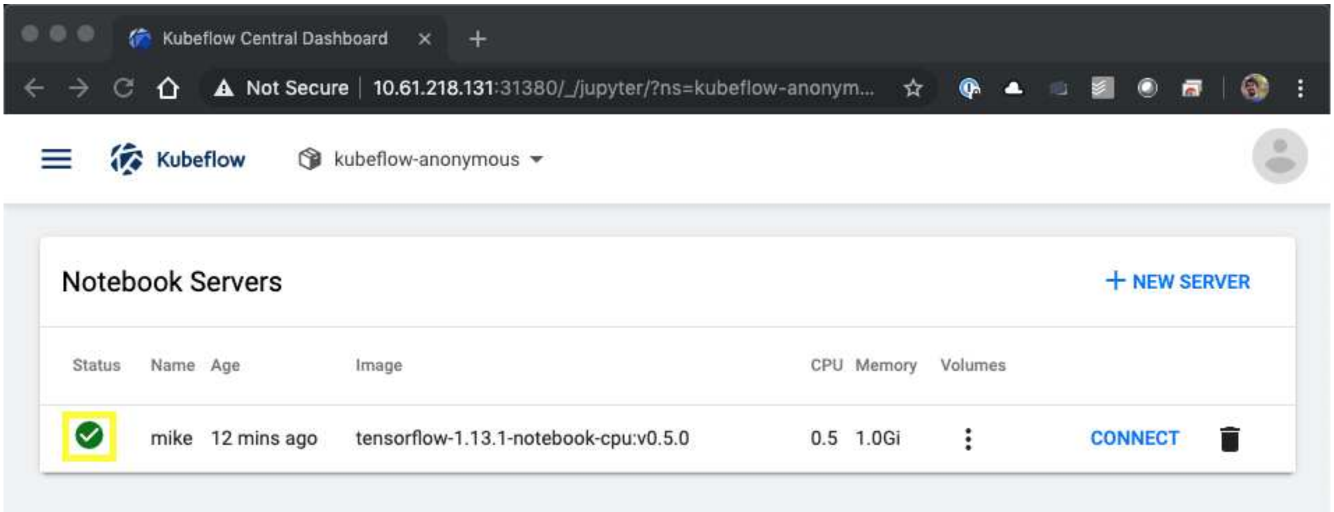
5. Add data volumes. The following example specifies an existing PVC named 'pb-fg-all' and accepts the default mount point.



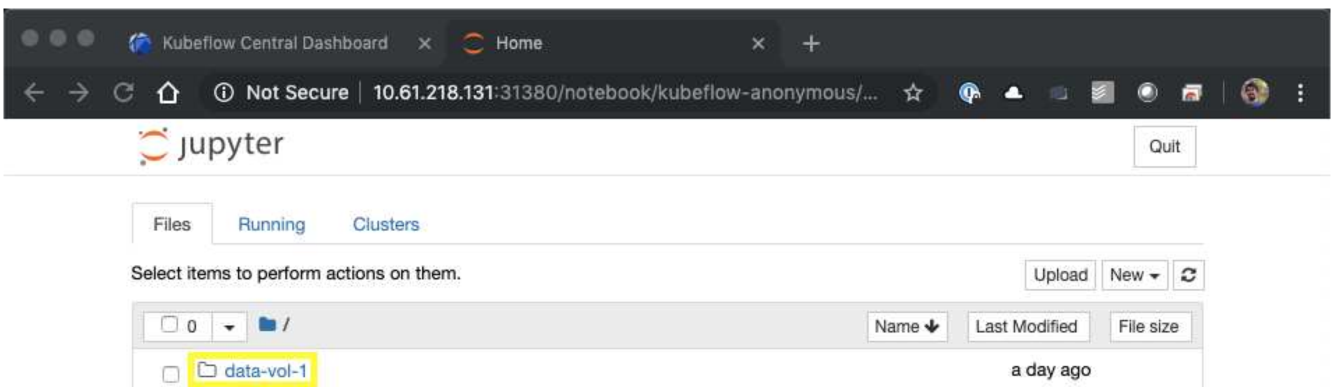
6. **Optional:** Request that the desired number of GPUs be allocated to your notebook server. In the following example, one GPU is requested.

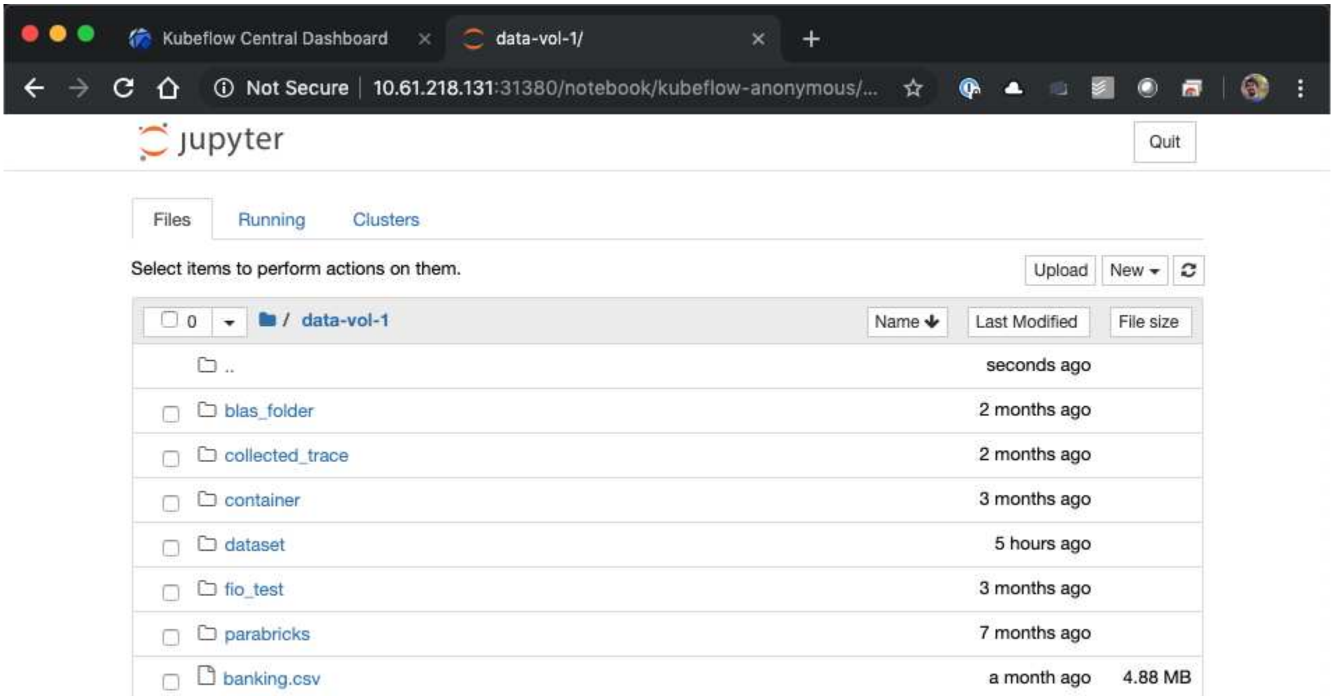


7. Click Launch to provision your new notebook server.
8. Wait for your notebook server to be fully provisioned. This can take several minutes if you have never provisioned a server using the Docker image that you specified because the image needs to be downloaded. When your server has been fully provisioned, you see a green check mark in the Status column on the Jupyter Notebook server administration page.



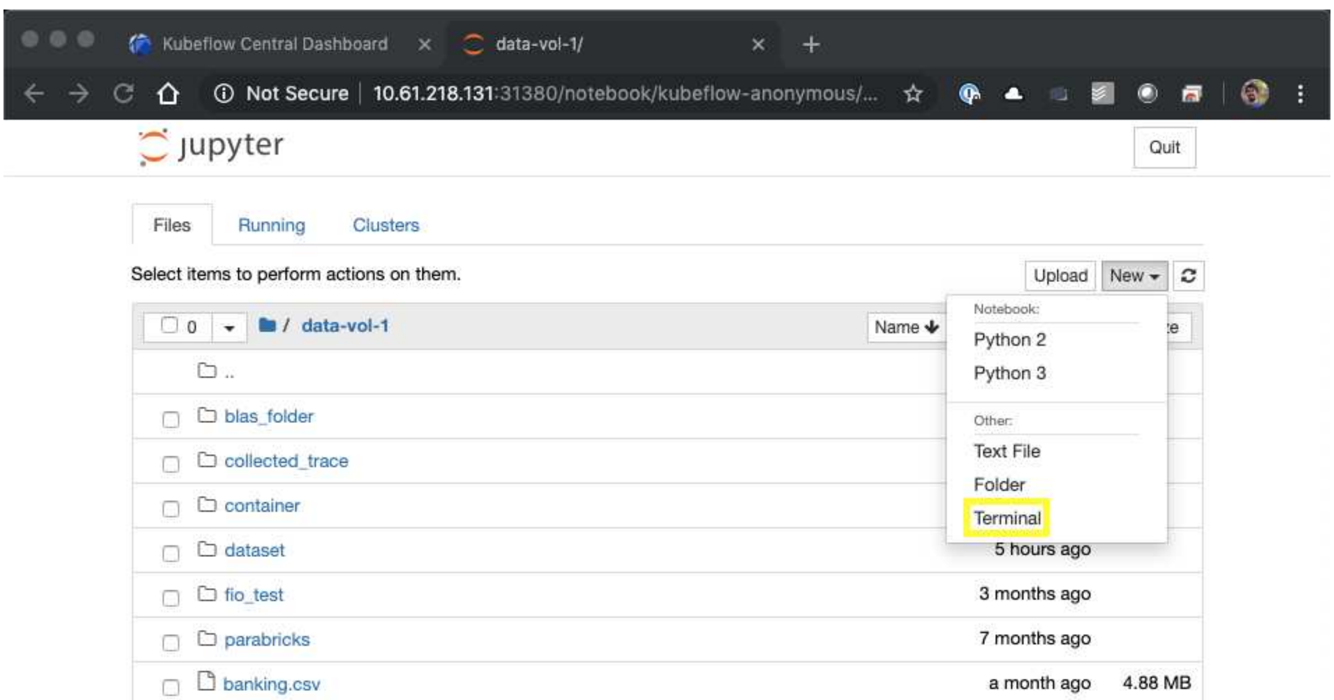
9. Click Connect to connect to your new server web interface.
10. Confirm that the dataset volume that was specified in step 6 is mounted on the server. Note that this volume is mounted within the default workspace by default. From the perspective of the user, this is just another folder within the workspace. The user, who is likely a data scientist and not an infrastructure expert, does not need to possess any storage expertise in order to use this volume.

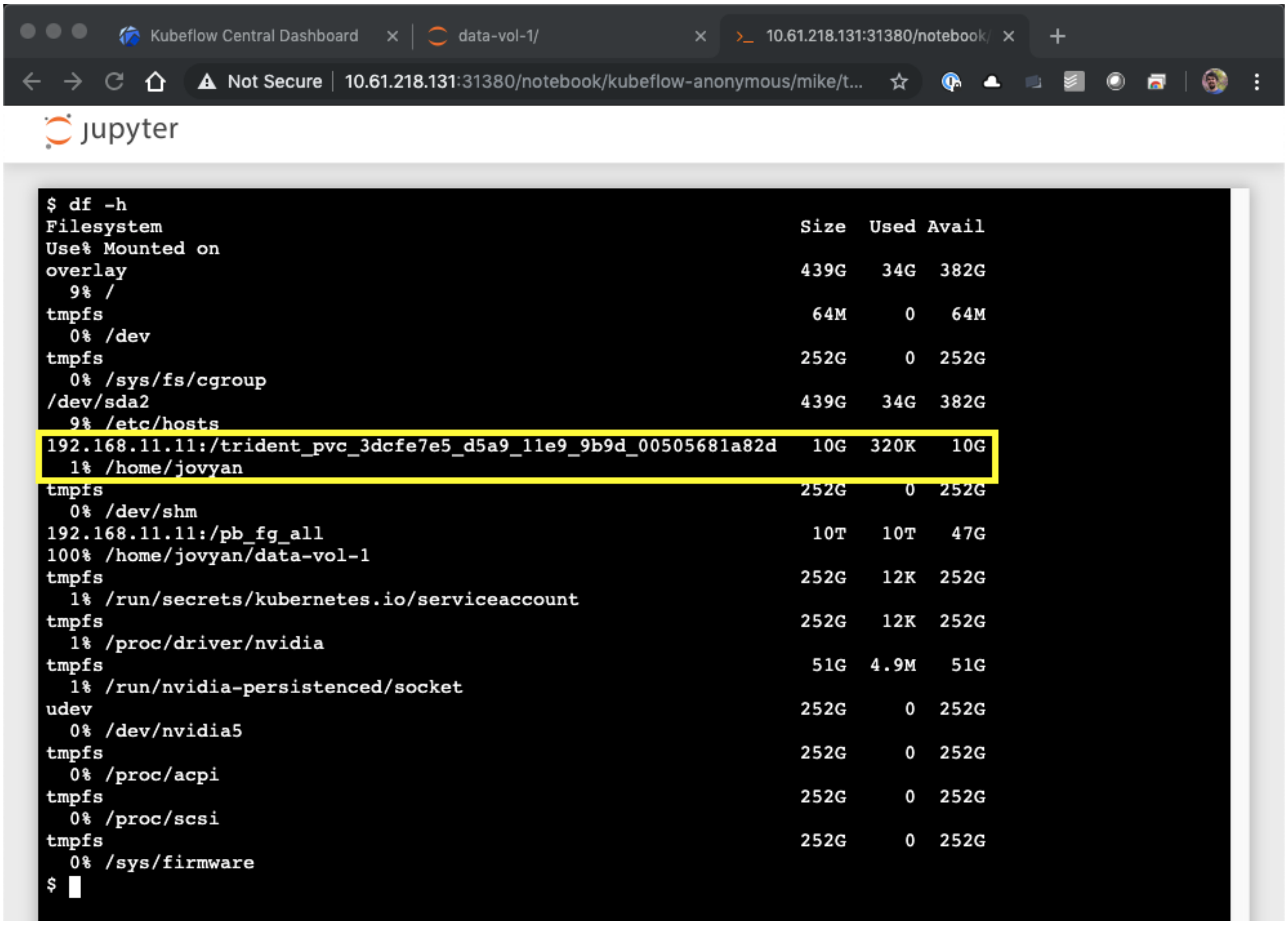




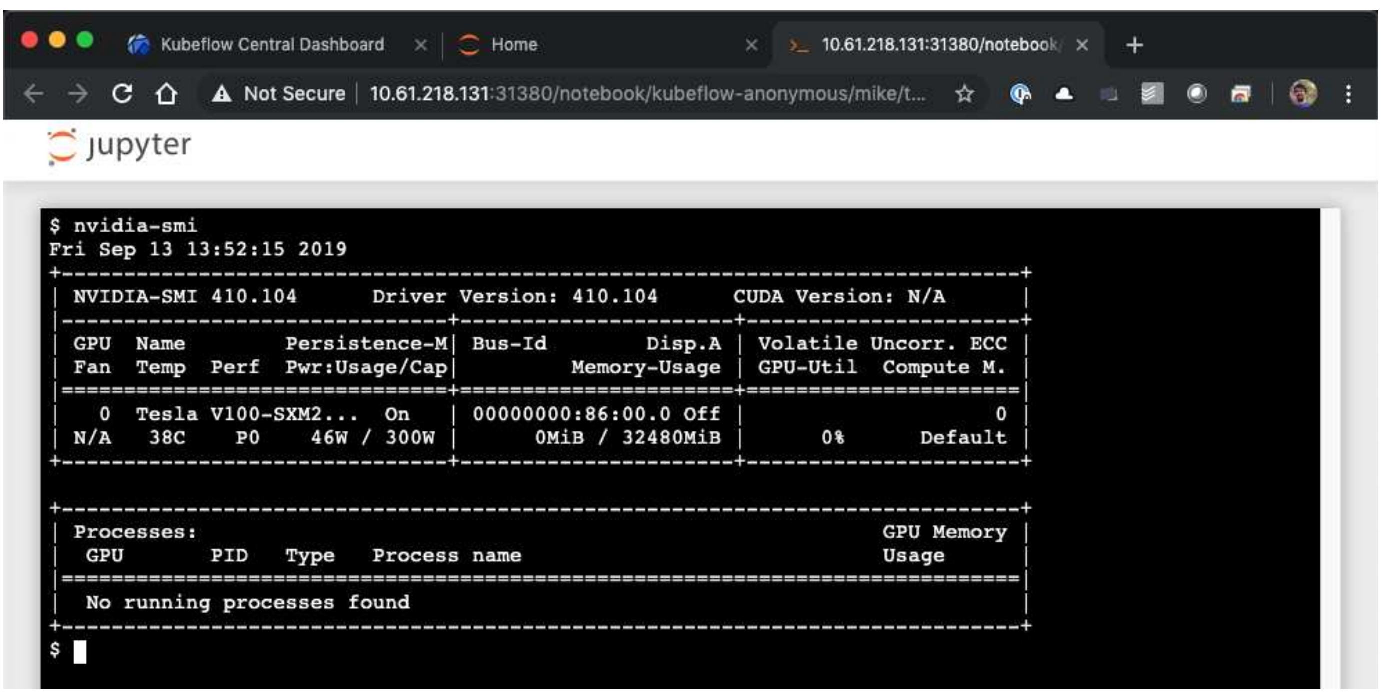
11. Open a Terminal and, assuming that a new volume was requested in step 5, execute `df -h` to confirm that a new Trident-provisioned persistent volume is mounted as the default workspace.

    The default workspace directory is the base directory that you are presented with when you first access the server's web interface. Therefore, any artifacts that you create by using the web interface are stored on this Trident-provisioned persistent volume.





12. Using the terminal, run `nvidia-smi` to confirm that the correct number of GPUs were allocated to the notebook server. In the following example, one GPU has been allocated to the notebook server as requested in step 7.



Example Notebooks and Pipelines

The NetApp Data Science Toolkit for Kubernetes can be used in conjunction with Kubeflow. Using the NetApp Data Science Toolkit with Kubeflow provides the following benefits:

- Data scientists can perform advanced NetApp data management operations directly from within a Jupyter Notebook.
- Advanced NetApp data management operations can be incorporated into automated workflows using the Kubeflow Pipelines framework.

Refer to the Kubeflow Examples section within the NetApp Data Science Toolkit GitHub repository for details on using the toolkit with Kubeflow.

Apache Airflow Deployment

NetApp recommends running Apache Airflow on top of Kubernetes. This section describes the tasks that you must complete to deploy Airflow in your Kubernetes cluster.

> It is possible to deploy Airflow on platforms other than Kubernetes. Deploying Airflow on platforms other than Kubernetes is outside of the scope of this solution.

Prerequisites

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

1. You already have a working Kubernetes cluster.
2. You have already installed and configured NetApp Trident in your Kubernetes cluster as outlined in the section “NetApp Trident Deployment and Configuration.”

Install Helm

Airflow is deployed using Helm, a popular package manager for Kubernetes. Before you deploy Airflow, you must install Helm on the deployment jump host. To install Helm on the deployment jump host, follow the installation instructions in the official Helm documentation.

Set Default Kubernetes StorageClass

Before you deploy Airflow, you must designate a default StorageClass within your Kubernetes cluster. The Airflow deployment process attempts to provision new persistent volumes using the default StorageClass. If no StorageClass is designated as the default StorageClass, then the deployment fails. To designate a default StorageClass within your cluster, follow the instructions outlined in the section Kubeflow Deployment. If you have already designated a default StorageClass within your cluster, then you can skip this step.

Use Helm to Deploy Airflow

To deploy Airflow in your Kubernetes cluster using Helm, perform the following tasks from the deployment jump host:

1. Deploy Airflow using Helm by following the deployment instructions for the official Airflow chart on the Artifact Hub. The example commands that follow show the deployment of Airflow using Helm. Modify, add,

and/or remove values in the `custom- values.yaml` file as needed depending on your environment and desired configuration.

```
$ cat << EOF > custom-values.yaml
###################################
# Airflow - Common Configs
###################################
airflow:
  ## the airflow executor type to use
  ##
  executor: "CeleryExecutor"
  ## environment variables for the web/scheduler/worker Pods (for
airflow configs)
  ##
  #
###################################
# Airflow - WebUI Configs
###################################
web:
  ## configs for the Service of the web Pods
  ##
  service:
    type: NodePort
###################################
# Airflow - Logs Configs
###################################
logs:
  persistence:
    enabled: true
###################################
# Airflow - DAGs Configs
###################################
dags:
  ## configs for the DAG git repository & sync container
  ##
  gitSync:
    enabled: true
    ## url of the git repository
    ##
    repo: "git@github.com:mboglesby/airflow-dev.git"
    ## the branch/tag/sha1 which we clone
    ##
    branch: master
    revision: HEAD
    ## the name of a pre-created secret containing files for ~/.ssh/
    ##
```

```
    ## NOTE:
    ## - this is ONLY RELEVANT for SSH git repos
    ## - the secret commonly includes files: id_rsa, id_rsa.pub,
known_hosts
    ## - known_hosts is NOT NEEDED if `git.sshKeyscan` is true
    ##
    sshSecret: "airflow-ssh-git-secret"
    ## the name of the private key file in your `git.secret`
    ##
    ## NOTE:
    ## - this is ONLY RELEVANT for PRIVATE SSH git repos
    ##
    sshSecretKey: id_rsa
    ## the git sync interval in seconds
    ##
    syncWait: 60
EOF
$ helm install airflow airflow-stable/airflow -n airflow --version 8.0.8
--values ./custom-values.yaml
...
Congratulations. You have just deployed Apache Airflow!
1. Get the Airflow Service URL by running these commands:
   export NODE_PORT=$(kubectl get --namespace airflow -o
jsonpath="{.spec.ports[0].nodePort}" services airflow-web)
   export NODE_IP=$(kubectl get nodes --namespace airflow -o
jsonpath="{.items[0].status.addresses[0].address}")
   echo http://$NODE_IP:$NODE_PORT/
2. Open Airflow in your web browser
```

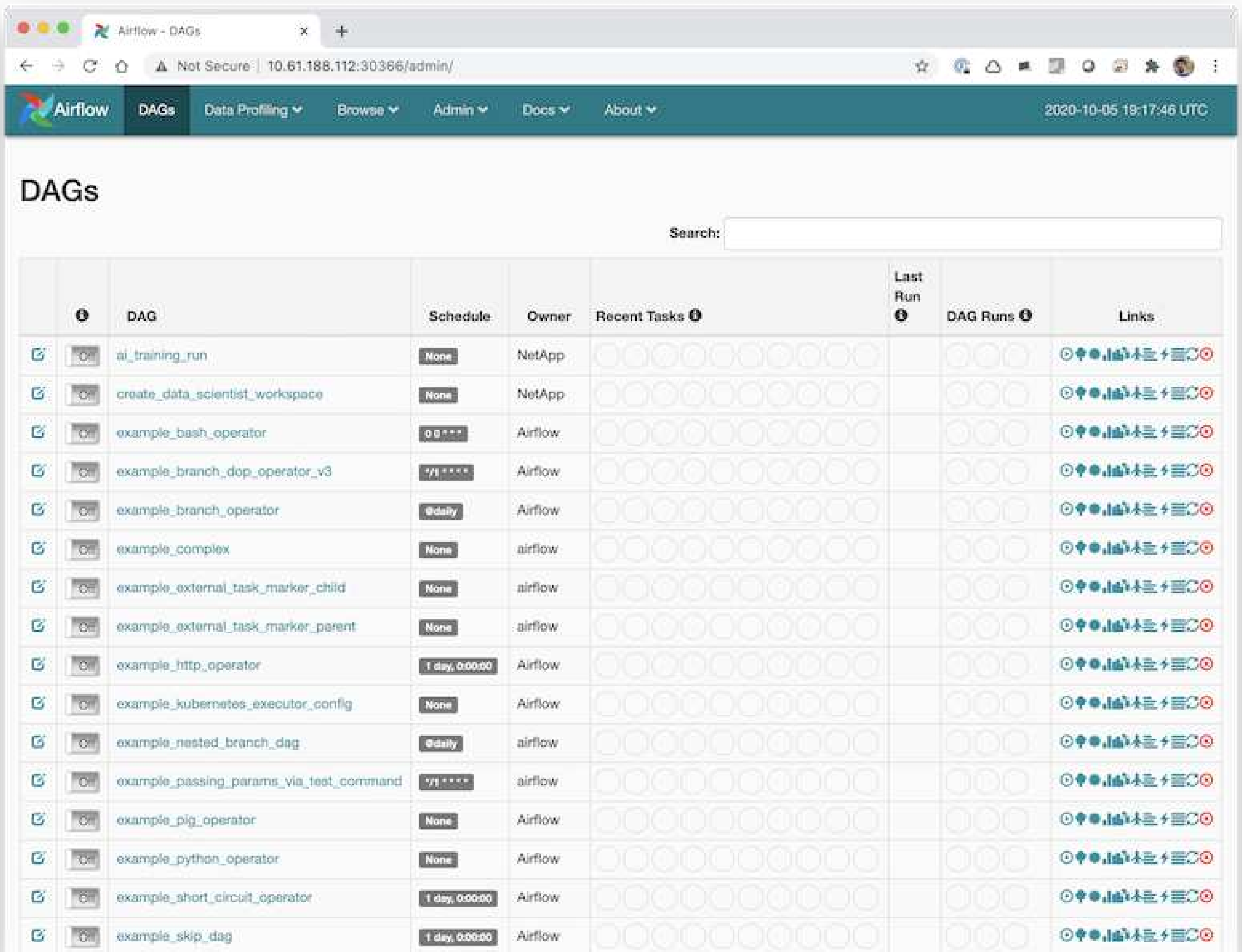
2. Confirm that all Airflow pods are up and running. It may take a few minutes for all pods to start.

```
$ kubectl -n airflow get pod
NAME                                 READY   STATUS    RESTARTS   AGE
airflow-flower-b5656d44f-h8qjk       1/1     Running   0          2h
airflow-postgresql-0                 1/1     Running   0          2h
airflow-redis-master-0               1/1     Running   0          2h
airflow-scheduler-9d95fcdf9-clf4b    2/2     Running   2          2h
airflow-web-59c94db9c5-z7rg4         1/1     Running   0          2h
airflow-worker-0                     2/2     Running   2          2h
```

3. Obtain the Airflow web service URL by following the instructions that were printed to the console when you deployed Airflow using Helm in step 1.

```
$ export NODE_PORT=$(kubectl get --namespace airflow -o
jsonpath="{.spec.ports[0].nodePort}" services airflow-web)
$ export NODE_IP=$(kubectl get nodes --namespace airflow -o
jsonpath="{.items[0].status.addresses[0].address}")
$ echo http://$NODE_IP:$NODE_PORT/
```

4. Confirm that you can access the Airflow web service.



Example Apache Airflow Workflows

The NetApp Data Science Toolkit for Kubernetes can be used in conjunction with Airflow. Using the NetApp Data Science Toolkit with Airflow enables you to incorporate NetApp data management operations into automated workflows that are orchestrated by Airflow.

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

Example Trident Operations

This section includes examples of various operations that you may want to perform with Trident.

Import an Existing Volume

If there are existing volumes on your NetApp storage system/platform that you want to mount on containers within your Kubernetes cluster, but that are not tied to PVCs in the cluster, then you must import these volumes. You can use the Trident volume import functionality to import these volumes.

The example commands that follow show the importing of the same volume, named `pb_fg_all`, twice, once for each Trident Backend that was created in the example in the section Example Trident Backends for ONTAP AI Deployments, step 1. Importing the same volume twice in this manner enables you to mount the volume (an existing FlexGroup volume) multiple times across different LIFs, as described in the section Example Trident Backends for ONTAP AI Deployments, step 1. For more information about PVCs, see the official Kubernetes documentation. For more information about the volume import functionality, see the Trident documentation.

An `accessModes` value of `ReadOnlyMany` is specified in the example PVC spec files. For more information about the `accessMode` field, see the official Kubernetes documentation.

> The Backend names that are specified in the following example import commands correspond to the Backends that were created in the example in the section Example Trident Backends for ONTAP AI Deployments, step 1. The StorageClass names that are specified in the following example PVC definition files correspond to the StorageClasses that were created in the example in the section Example Kubernetes StorageClasses for ONTAP AI Deployments, step 1.

```
$ cat << EOF > ./pvc-import-pb_fg_all-iface1.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pb-fg-all-iface1
  namespace: default
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexgroups-retain-iface1
EOF
$ tridentctl import volume ontap-ai-flexgroups-iface1 pb_fg_all -f ./pvc-
import-pb_fg_all-iface1.yaml -n trident
+--------------------------------+--------
+------------------------------------+----------
+------------------------------------------+--------+---------+
|          NAME                    |  SIZE   |       STORAGE CLASS
| PROTOCOL |             BACKEND UUID                         | STATE   |
MANAGED |
+--------------------------------+--------
+------------------------------------+----------
+------------------------------------------+--------+---------+
```

```
| default-pb-fg-all-iface1-7d9f1 | 10 TiB | ontap-ai-flexgroups-retain-
iface1 | file     | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd | online | true
|
+--------------------------------+--------
+------------------------------------+----------
+--------------------------------------------+--------+---------+
$ cat << EOF > ./pvc-import-pb_fg_all-iface2.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pb-fg-all-iface2
  namespace: default
spec:
  accessModes:
    - ReadOnlyMany
  storageClassName: ontap-ai-flexgroups-retain-iface2
EOF
$ tridentctl import volume ontap-ai-flexgroups-iface2 pb_fg_all -f ./pvc-
import-pb_fg_all-iface2.yaml -n trident
+--------------------------------+--------
+------------------------------------+----------
+--------------------------------------------+--------+---------+
|          NAME                  |  SIZE  |       STORAGE CLASS
| PROTOCOL |             BACKEND UUID                         | STATE  |
MANAGED |
+--------------------------------+--------
+------------------------------------+----------
+------------------------------------------+--------+---------+
| default-pb-fg-all-iface2-85aee | 10 TiB | ontap-ai-flexgroups-retain-
iface2 | file     | 61814d48-c770-436b-9cb4-cf7ee661274d | online | true
|
+--------------------------------+--------
+------------------------------------+----------
+--------------------------------------------+--------+---------+
$ tridentctl get volume -n trident
+----------------------------------+---------
+------------------------------------+----------
+--------------------------------------+--------+---------+
|               NAME               |  SIZE   |           STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+----------------------------------+---------
+------------------------------------+----------
+--------------------------------------+--------+---------+
| default-pb-fg-all-iface1-7d9f1   | 10 TiB  | ontap-ai-flexgroups-retain-
iface1 | file     | b74cbddb-e0b8-40b7-b263-b6da6dec0bdd | online | true
|
```

```
| default-pb-fg-all-iface2-85aee   | 10 TiB  | ontap-ai-flexgroups-retain-
iface2 | file     | 61814d48-c770-436b-9cb4-cf7ee661274d | online | true
|
+-----------------------------------+---------
+------------------------------------+----------
+--------------------------------------+--------+---------+
$ kubectl get pvc
NAME                 STATUS   VOLUME                             CAPACITY
ACCESS MODES   STORAGECLASS                        AGE
pb-fg-all-iface1     Bound    default-pb-fg-all-iface1-7d9f1
10995116277760   ROX            ontap-ai-flexgroups-retain-iface1   25h
pb-fg-all-iface2     Bound    default-pb-fg-all-iface2-85aee
10995116277760   ROX            ontap-ai-flexgroups-retain-iface2   25h
```

Provision a New Volume

You can use Trident to provision a new volume on your NetApp storage system or platform. The following example commands show the provisioning of a new FlexVol volume. In this example, the volume is provisioned using the StorageClass that was created in the example in the section Example Kubernetes StorageClasses for ONTAP AI Deployments, step 2.

An `accessModes` value of `ReadWriteMany` is specified in the following example PVC definition file. For more information about the `accessMode` field, see the official Kubernetes documentation.

```
$ cat << EOF > ./pvc-tensorflow-results.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: tensorflow-results
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-ai-flexvols-retain
EOF
$ kubectl create -f ./pvc-tensorflow-results.yaml
persistentvolumeclaim/tensorflow-results created
$ kubectl get pvc
NAME                              STATUS    VOLUME
CAPACITY         ACCESS MODES   STORAGECLASS                        AGE
pb-fg-all-iface1                  Bound     default-pb-fg-all-iface1-7d9f1
10995116277760   ROX            ontap-ai-flexgroups-retain-iface1   26h
pb-fg-all-iface2                  Bound     default-pb-fg-all-iface2-85aee
10995116277760   ROX            ontap-ai-flexgroups-retain-iface2   26h
tensorflow-results                Bound     default-tensorflow-results-
2fd60   1073741824       RWX            ontap-ai-flexvols-retain
25h
```

Example High-performance Jobs for ONTAP AI Deployments

This section includes examples of various high-performance jobs that can be executed when Kubernetes is deployed on an ONTAP AI pod.

Example High-performance Jobs for ONTAP AI Deployments

This section includes examples of various high-performance jobs that can be executed when Kubernetes is deployed on an ONTAP AI pod.

Execute a Single-Node AI Workload

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

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

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

   This example job requests eight GPUs and therefore can run on a single GPU worker node that features eight or more GPUs. This example job could be submitted in a cluster for which a worker node featuring eight or more GPUs is not present or is currently occupied with another workload. If so, then the job remains in a pending state until such a worker node becomes available.

   Additionally, in order to maximize storage bandwidth, the volume that contains the needed training data is mounted twice within the pod that this job creates. Another volume is also mounted in the pod. This second volume will be used to store results and metrics. These volumes are referenced in the job definition by using the names of the PVCs. For more information about Kubernetes jobs, see the official Kubernetes documentation.

   An `emptyDir` volume with a `medium` value of `Memory` is mounted to `/dev/shm` in the pod that this example job creates. The default size of the `/dev/shm` virtual volume that is automatically created by the Docker container runtime can sometimes be insufficient for TensorFlow's needs. Mounting an `emptyDir` volume as in the following example provides a sufficiently large `/dev/shm` virtual volume. For more information about `emptyDir` volumes, see the official Kubernetes documentation.

   The single container that is specified in this example job definition is given a `securityContext > privileged` value of `true`. This value means that the container effectively has root access on the host. This annotation is used in this case because the specific workload that is being executed requires root access. Specifically, a clear cache operation that the workload performs requires root access. Whether or not this `privileged: true` annotation is necessary depends on the requirements of the specific workload that you are executing.

   ```
   $ cat << EOF > ./netapp-tensorflow-single-imagenet.yaml
   apiVersion: batch/v1
   kind: Job
   metadata:
     name: netapp-tensorflow-single-imagenet
   spec:
     backoffLimit: 5
     template:
       spec:
         volumes:
         - name: dshm
           emptyDir:
             medium: Memory
         - name: testdata-iface1
           persistentVolumeClaim:
             claimName: pb-fg-all-iface1
         - name: testdata-iface2
           persistentVolumeClaim:
             claimName: pb-fg-all-iface2
   ```

```
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
dataset_dir=/mnt/mount_0/dataset/imagenet", "--dgx_version=dgx1", "--
num_devices=8"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-single-imagenet.yaml
job.batch/netapp-tensorflow-single-imagenet created
$ kubectl get jobs
NAME                                       COMPLETIONS   DURATION   AGE
netapp-tensorflow-single-imagenet          0/1           24s        24s
```

2. Confirm that the job that you created in step 1 is running correctly. The following example command confirms that a single pod was created for the job, as specified in the job definition, and that this pod is currently running on one of the GPU worker nodes.

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

3. Confirm that the job that you created in step 1 completes successfully. The following example commands confirm that the job completed successfully.

```
$ kubectl get jobs
NAME                                                  COMPLETIONS   DURATION
AGE
netapp-tensorflow-single-imagenet                     1/1           5m42s
10m
$ kubectl get pods
NAME                                                        READY   STATUS
RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92                     0/1     Completed
0          11m
$ kubectl logs netapp-tensorflow-single-imagenet-m7x92
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-
PERMISSIONS in file gds_dstore.c at line 702
[netapp-tensorflow-single-imagenet-m7x92:00008] PMIX ERROR: NO-
PERMISSIONS in file gds_dstore.c at line 711
Total images/sec = 6530.59125
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 1 -H localhost:1 bash -c 'sync; echo 1 >
/proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 8 -H localhost:8 -bind-to none -map-by
slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_be
nchmarks.py --model=resnet50 --batch_size=256 --device=gpu
--force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48
--variable_update=horovod --batch_group_size=20 --num_batches=500
--nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True
--use_tf_layers=False --data_name=imagenet --use_datasets=True
--data_dir=/mnt/mount_0/dataset/imagenet
--datasets_parallel_interleave_cycle_length=10
--datasets_sloppy_parallel_interleave=False --num_mounts=2
--mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000
--datasets_use_prefetch=True --datasets_num_private_threads=4
--horovod_device=gpu >
/tmp/20190814_105450_tensorflow_horovod_rdma_resnet50_gpu_8_256_b500_ima
genet_nodistort_fp16_r10_m2_nockpt.txt 2>&1
```

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

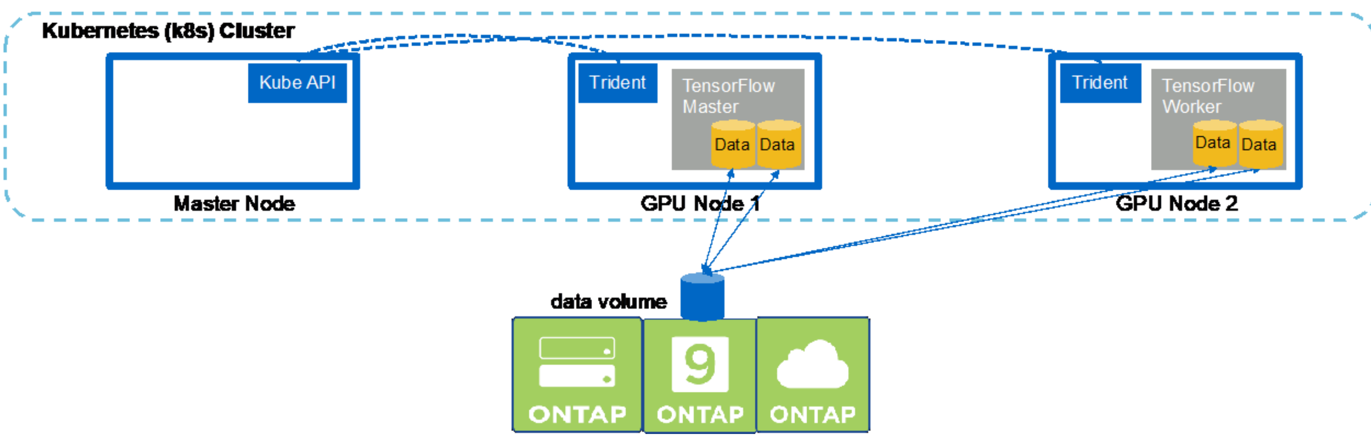
   When you delete the job object, Kubernetes automatically deletes any associated pods.

```
$ kubectl get jobs
NAME                                          COMPLETIONS   DURATION
AGE
netapp-tensorflow-single-imagenet             1/1           5m42s
10m
$ kubectl get pods
NAME                                                READY   STATUS
RESTARTS   AGE
netapp-tensorflow-single-imagenet-m7x92             0/1     Completed
0          11m
$ kubectl delete job netapp-tensorflow-single-imagenet
job.batch "netapp-tensorflow-single-imagenet" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.
```

Execute a Synchronous Distributed AI Workload

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

> Synchronous distributed jobs can help increase performance and training accuracy compared with asynchronous distributed jobs. A discussion of the pros and cons of synchronous jobs versus asynchronous jobs is outside the scope of this document.



1. The following example commands show the creation of one worker that participates in the synchronous distributed execution of the same TensorFlow benchmark job that was executed on a single node in the example in the section Execute a Single-Node AI Workload. In this specific example, only a single worker is deployed because the job is executed across two worker nodes.

This example worker deployment requests eight GPUs and thus can run on a single GPU worker node that features eight or more GPUs. If your GPU worker nodes feature more than eight GPUs, to maximize performance, you might want to increase this number to be equal to the number of GPUs that your worker nodes feature. For more information about Kubernetes deployments, see the official Kubernetes documentation.

A Kubernetes deployment is created in this example because this specific containerized worker would never complete on its own. Therefore, it doesn't make sense to deploy it by using the Kubernetes job construct. If your worker is designed or written to complete on its own, then it might make sense to use the job construct to deploy your worker.

The pod that is specified in this example deployment specification is given a `hostNetwork` value of `true`. This value means that the pod uses the host worker node's networking stack instead of the virtual networking stack that Kubernetes usually creates for each pod. This annotation is used in this case because the specific workload relies on Open MPI, NCCL, and Horovod to execute the workload in a synchronous distributed manner. Therefore, it requires access to the host networking stack. A discussion about Open MPI, NCCL, and Horovod is outside the scope of this document. Whether or not this `hostNetwork: true` annotation is necessary depends on the requirements of the specific workload that you are executing. For more information about the `hostNetwork` field, see the official Kubernetes documentation.

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-worker.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: netapp-tensorflow-multi-imagenet-worker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: netapp-tensorflow-multi-imagenet-worker
  template:
    metadata:
      labels:
        app: netapp-tensorflow-multi-imagenet-worker
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
```

```
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["bash", "/netapp/scripts/start-slave-multi.sh",
"22122"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-worker.yaml
deployment.apps/netapp-tensorflow-multi-imagenet-worker created
$ kubectl get deployments
NAME                                      DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1
1           4s
```

2. Confirm that the worker deployment that you created in step 1 launched successfully. The following example commands confirm that a single worker pod was created for the deployment, as indicated in the deployment definition, and that this pod is currently running on one of the GPU worker nodes.

```
$ kubectl get pods -o wide
NAME                                                       READY
STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running   0          60s   10.61.218.154   10.61.218.154   <none>
$ kubectl logs netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725
22122
```

3. Create a Kubernetes job for a master that kicks off, participates in, and tracks the execution of the synchronous multinode job. The following example commands create one master that kicks off, participates in, and tracks the synchronous distributed execution of the same TensorFlow benchmark job that was executed on a single node in the example in the section Execute a Single-Node AI Workload.

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

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

```
$ cat << EOF > ./netapp-tensorflow-multi-imagenet-master.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: netapp-tensorflow-multi-imagenet-master
spec:
  backoffLimit: 5
  template:
    spec:
      hostNetwork: true
      volumes:
      - name: dshm
        emptyDir:
          medium: Memory
      - name: testdata-iface1
        persistentVolumeClaim:
          claimName: pb-fg-all-iface1
      - name: testdata-iface2
        persistentVolumeClaim:
          claimName: pb-fg-all-iface2
      - name: results
        persistentVolumeClaim:
          claimName: tensorflow-results
      containers:
      - name: netapp-tensorflow-py2
        image: netapp/tensorflow-py2:19.03.0
        command: ["python", "/netapp/scripts/run.py", "--
dataset_dir=/mnt/mount_0/dataset/imagenet", "--port=22122", "--
num_devices=16", "--dgx_version=dgx1", "--
nodes=10.61.218.152,10.61.218.154"]
        resources:
          limits:
            nvidia.com/gpu: 8
        volumeMounts:
        - mountPath: /dev/shm
          name: dshm
        - mountPath: /mnt/mount_0
          name: testdata-iface1
        - mountPath: /mnt/mount_1
```

```
          name: testdata-iface2
        - mountPath: /tmp
          name: results
        securityContext:
          privileged: true
      restartPolicy: Never
EOF
$ kubectl create -f ./netapp-tensorflow-multi-imagenet-master.yaml
job.batch/netapp-tensorflow-multi-imagenet-master created
$ kubectl get jobs
NAME                                      COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   0/1           25s        25s
```

4. Confirm that the master job that you created in step 3 is running correctly. The following example command confirms that a single master pod was created for the job, as indicated in the job definition, and that this pod is currently running on one of the GPU worker nodes. You should also see that the worker pod that you originally saw in step 1 is still running and that the master and worker pods are running on different nodes.

```
$ kubectl get pods -o wide
NAME                                                           READY
STATUS    RESTARTS   AGE
IP              NODE            NOMINATED NODE
netapp-tensorflow-multi-imagenet-master-ppwwj                  1/1
Running   0          45s   10.61.218.152   10.61.218.152   <none>
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running   0          26m   10.61.218.154   10.61.218.154   <none>
```

5. Confirm that the master job that you created in step 3 completes successfully. The following example commands confirm that the job completed successfully.

```
$ kubectl get jobs
NAME                                      COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      9m18s
$ kubectl get pods
NAME                                                           READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj                  0/1
Completed   0          9m38s
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running     0          35m
$ kubectl logs netapp-tensorflow-multi-imagenet-master-ppwwj
[10.61.218.152:00008] WARNING: local probe returned unhandled
shell:unknown assuming bash
rm: cannot remove '/lib': Is a directory
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
```

```
line 702
[10.61.218.154:00033] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 711
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 702
[10.61.218.152:00008] PMIX ERROR: NO-PERMISSIONS in file gds_dstore.c at
line 711
Total images/sec = 12881.33875
================ Clean Cache !!! ==================
mpirun -allow-run-as-root -np 2 -H 10.61.218.152:1,10.61.218.154:1 -mca
pml ob1 -mca btl ^openib -mca btl_tcp_if_include enp1s0f0 -mca
plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" bash -c 'sync; echo 1 >
/proc/sys/vm/drop_caches'
=========================================
mpirun -allow-run-as-root -np 16 -H 10.61.218.152:8,10.61.218.154:8
-bind-to none -map-by slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH
-mca pml ob1 -mca btl ^openib -mca btl_tcp_if_include enp1s0f0 -x
NCCL_IB_HCA=mlx5 -x NCCL_NET_GDR_READ=1 -x NCCL_IB_SL=3 -x
NCCL_IB_GID_INDEX=3 -x
NCCL_SOCKET_IFNAME=enp5s0.3091,enp12s0.3092,enp132s0.3093,enp139s0.3094
-x NCCL_IB_CUDA_SUPPORT=1 -mca orte_base_help_aggregate 0 -mca
plm_rsh_agent ssh -mca plm_rsh_args "-p 22122" python
/netapp/tensorflow/benchmarks_190205/scripts/tf_cnn_benchmarks/tf_cnn_be
nchmarks.py --model=resnet50 --batch_size=256 --device=gpu
--force_gpu_compatible=True --num_intra_threads=1 --num_inter_threads=48
--variable_update=horovod --batch_group_size=20 --num_batches=500
--nodistortions --num_gpus=1 --data_format=NCHW --use_fp16=True
--use_tf_layers=False --data_name=imagenet --use_datasets=True
--data_dir=/mnt/mount_0/dataset/imagenet
--datasets_parallel_interleave_cycle_length=10
--datasets_sloppy_parallel_interleave=False --num_mounts=2
--mount_prefix=/mnt/mount_%d --datasets_prefetch_buffer_size=2000 --
datasets_use_prefetch=True --datasets_num_private_threads=4
--horovod_device=gpu >
/tmp/20190814_161609_tensorflow_horovod_rdma_resnet50_gpu_16_256_b500_im
agenet_nodistort_fp16_r10_m2_nockpt.txt 2>&1
```

6. Delete the worker deployment when you no longer need it. The following example commands show the deletion of the worker deployment object that was created in step 1.

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

```
$ kubectl get deployments
NAME                                       DESIRED   CURRENT   UP-TO-DATE
AVAILABLE   AGE
netapp-tensorflow-multi-imagenet-worker   1         1         1
1           43m
$ kubectl get pods
NAME                                                       READY
STATUS      RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj              0/1
Completed   0          17m
netapp-tensorflow-multi-imagenet-worker-654fc7f486-v6725   1/1
Running     0          43m
$ kubectl delete deployment netapp-tensorflow-multi-imagenet-worker
deployment.extensions "netapp-tensorflow-multi-imagenet-worker" deleted
$ kubectl get deployments
No resources found.
$ kubectl get pods
NAME                                            READY   STATUS
RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj   0/1     Completed   0
18m
```

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

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

```
$ kubectl get jobs
NAME                                      COMPLETIONS   DURATION   AGE
netapp-tensorflow-multi-imagenet-master   1/1           5m50s      19m
$ kubectl get pods
NAME                                            READY   STATUS
RESTARTS   AGE
netapp-tensorflow-multi-imagenet-master-ppwwj   0/1     Completed   0
19m
$ kubectl delete job netapp-tensorflow-multi-imagenet-master
job.batch "netapp-tensorflow-multi-imagenet-master" deleted
$ kubectl get jobs
No resources found.
$ kubectl get pods
No resources found.
```

Performance Testing

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

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

Conclusion

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

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

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

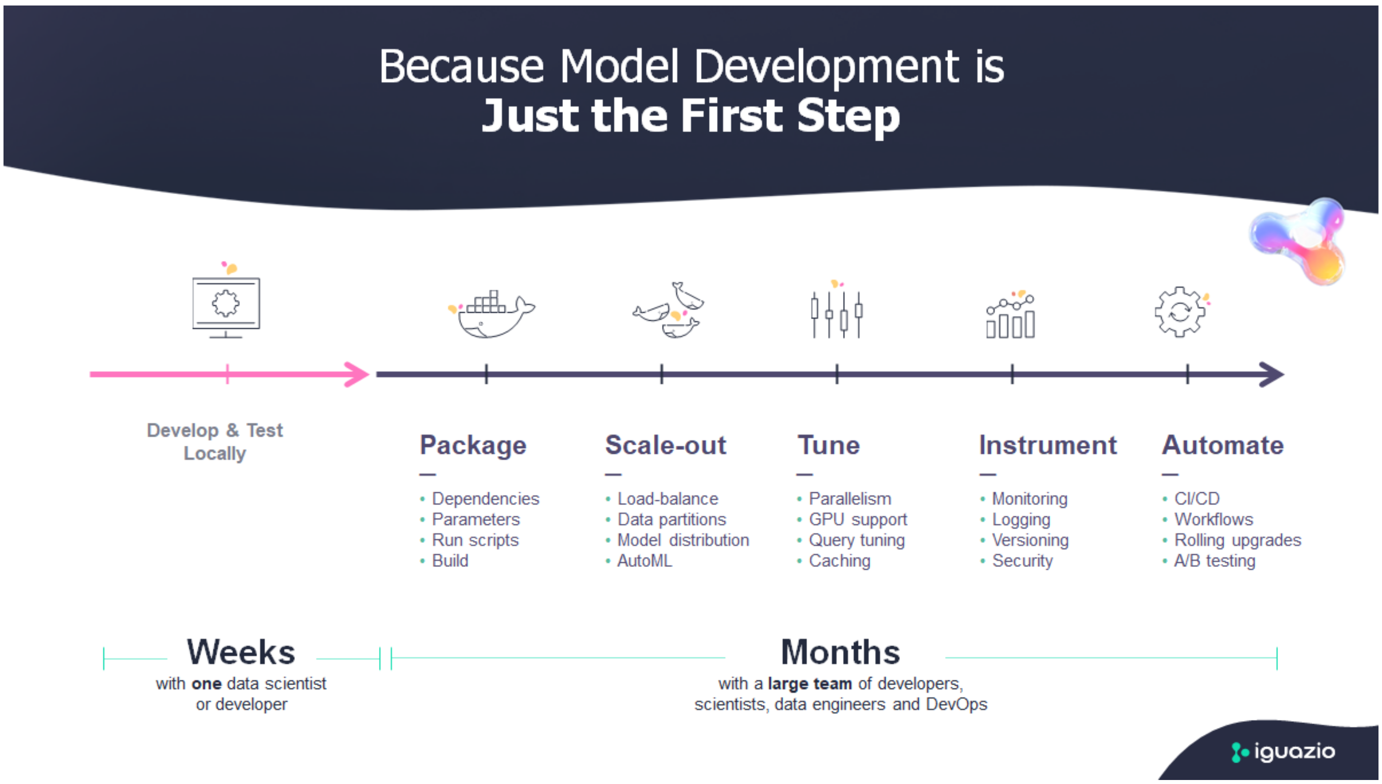
MLRun Pipeline with Iguazio

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.

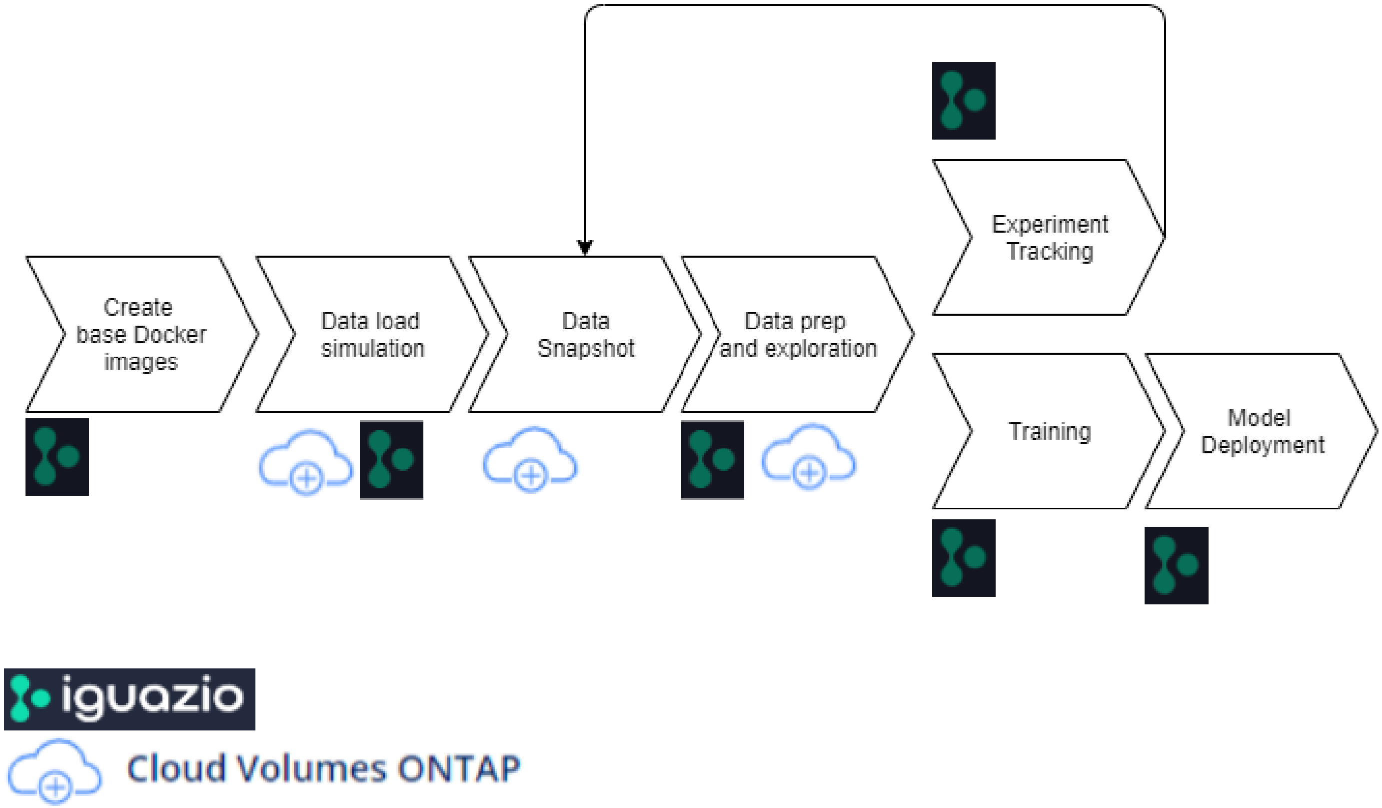
Target Audience

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.



Technology Overview

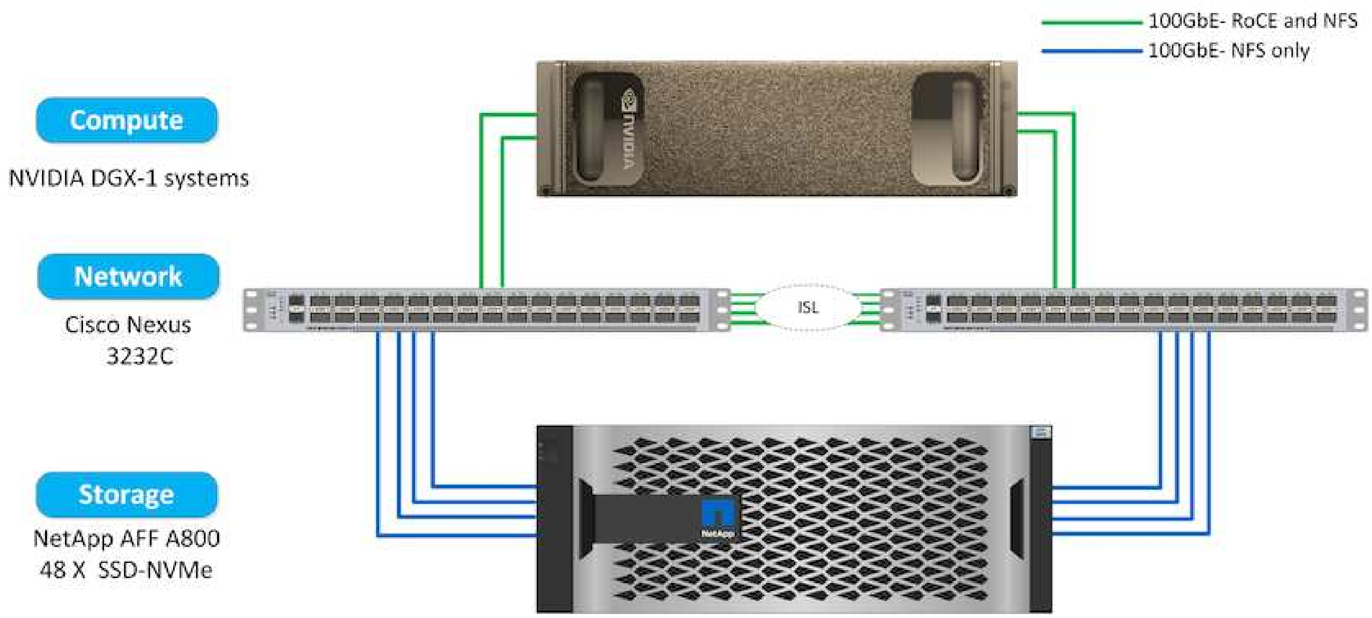
NetApp Overview

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

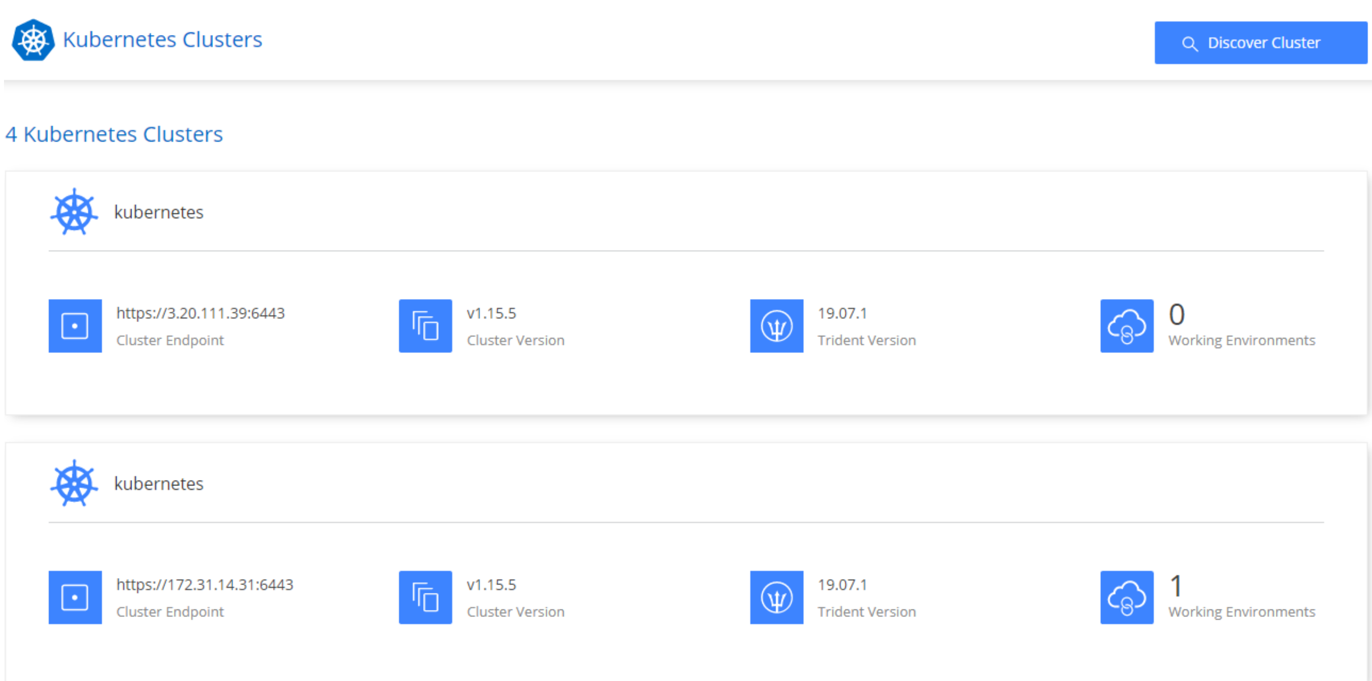
NetApp ONTAP AI, powered by NVIDIA DGX systems and NetApp cloud-connected all-flash storage, streamlines the flow of data reliably and speeds up analytics, training, and inference with your data fabric that spans from edge to core to cloud. It gives IT organizations an architecture that provides the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

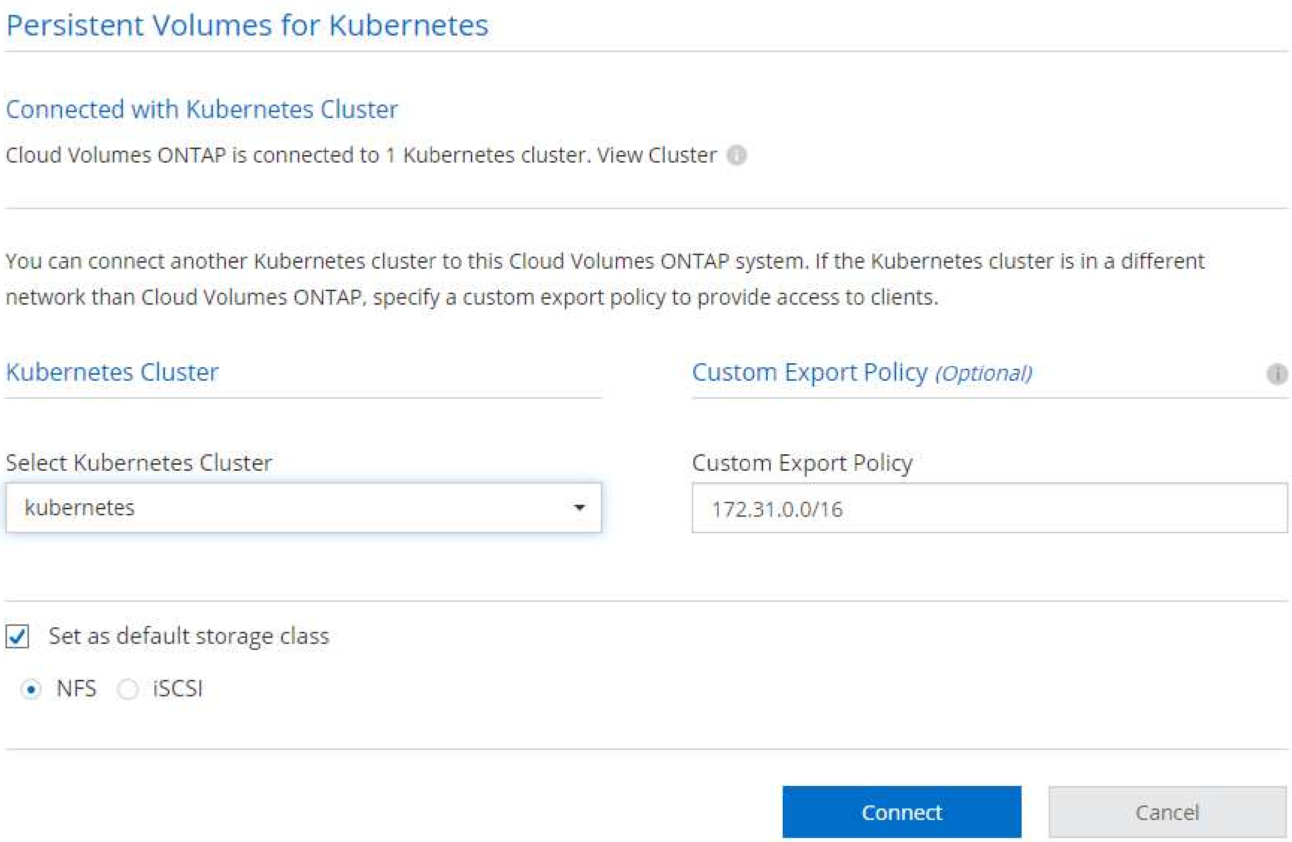
- Eliminates design complexities
- Allows independent scaling of compute and storage
- Enables customers to start small and scale seamlessly
- Offers a range of storage options for various performance and cost pointsNetApp ONTAP AI offers converged infrastructure stacks incorporating NVIDIA DGX-1, a 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.

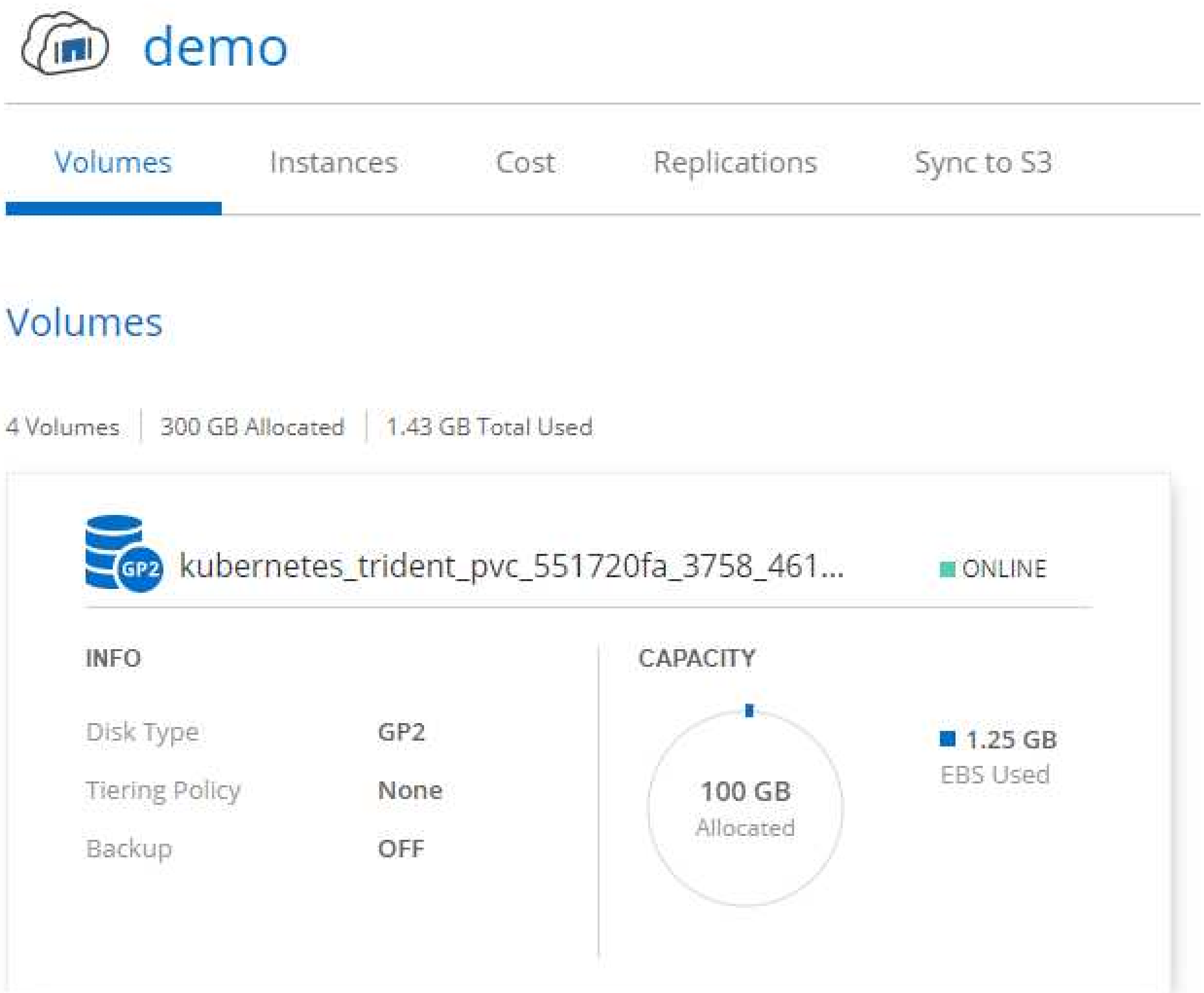


NetApp AI Control Plane

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.



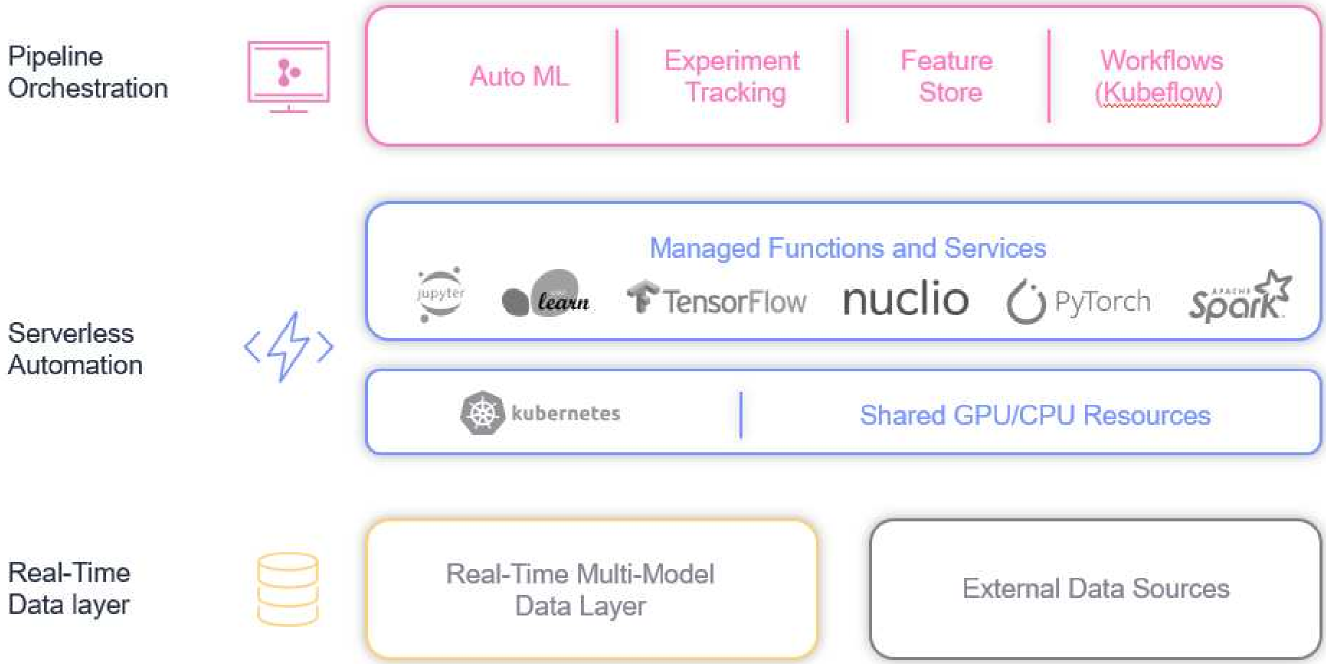

Persistent Volumes for Kubernetes
Connected with Kubernetes Cluster
Cloud Volumes ONTAP is connected to 1 Kubernetes cluster. View Cluster
You can connect another Kubernetes cluster to this Cloud Volumes ONTAP system. If the Kubernetes cluster is in a different network than Cloud Volumes ONTAP, specify a custom export policy to provide access to clients.
Kubernetes Cluster
Custom Export Policy (Optional)
Select Kubernetes Cluster
kubernetes
Custom Export Policy
172.31.0.0/16
Set as default storage class
NFS
iSCSI
Connect
Cancel



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.

Hardware	Quantity
DGX-1 systems	1
NetApp AFF A800 system	1 high-availability (HA) pair, includes 2 controllers and 48 NVMe SSDs (3.8TB or above)
Cisco Nexus 3232C network switches	2

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

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

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.

Software	Version or Type
Iguazio	Version 2.8+
App node	M5.4xlarge
Data node	I3.4xlarge

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 failuresAs a result of this implementation of Iguazio, 60% of failures were proactively prevented.

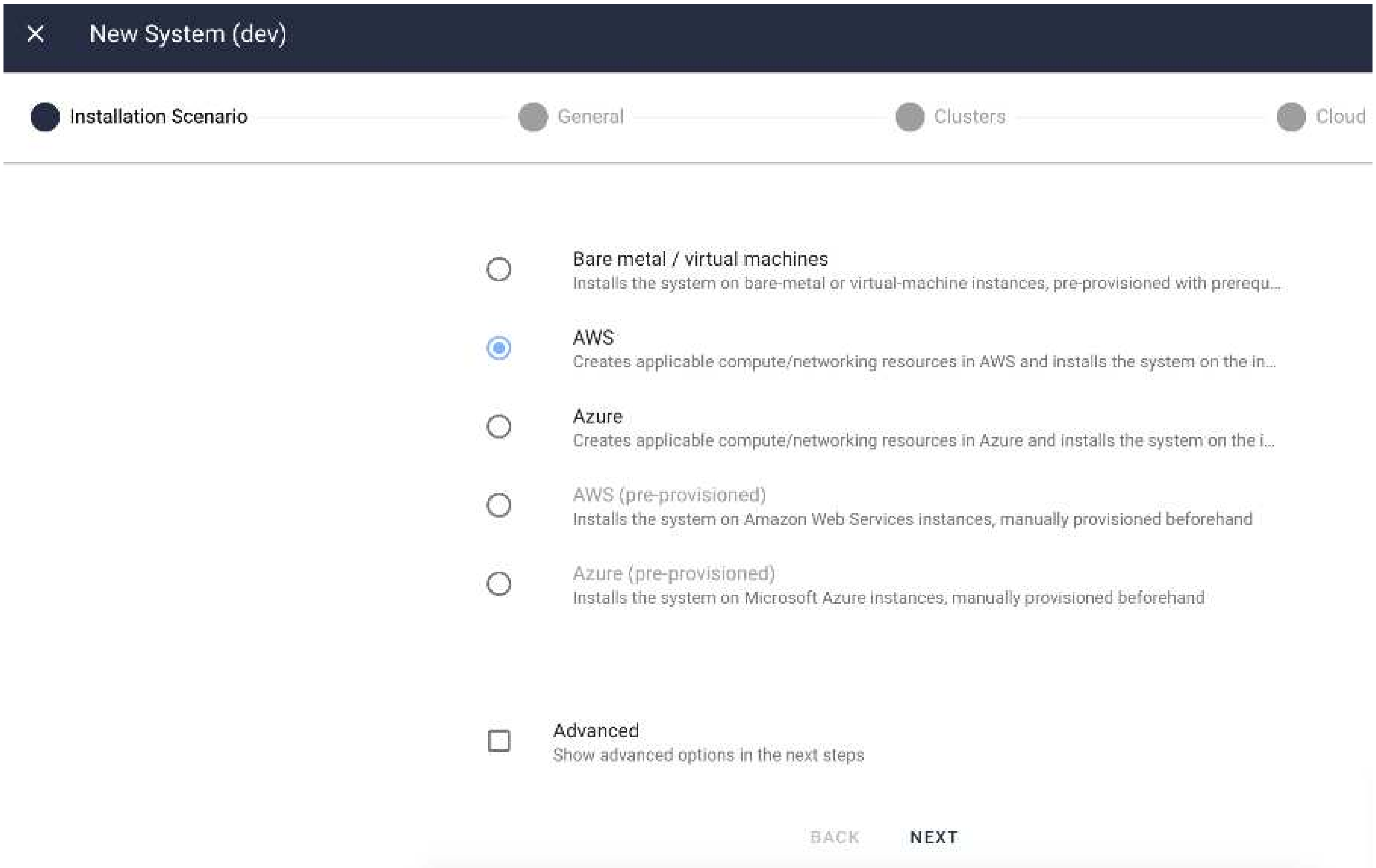
Setup Overview

Iguazio can be installed on-premises or on a cloud provider.

Iguazio Installation

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.



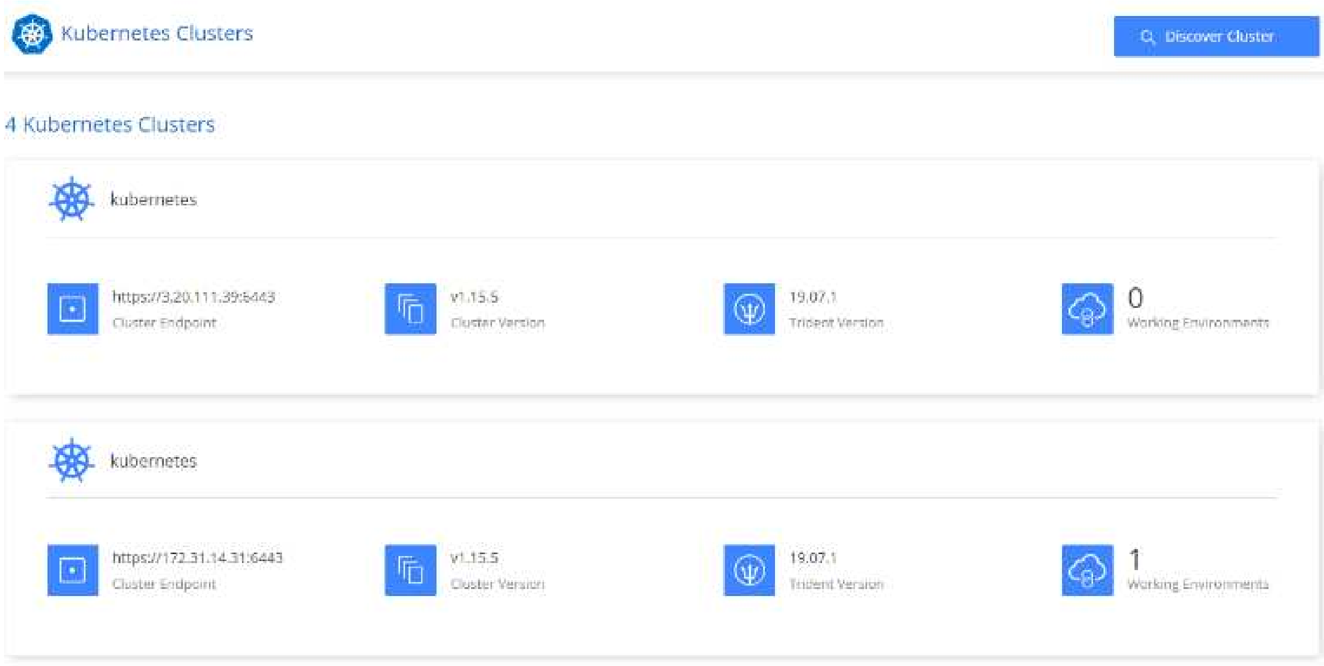
Configuring Kubernetes Cluster

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

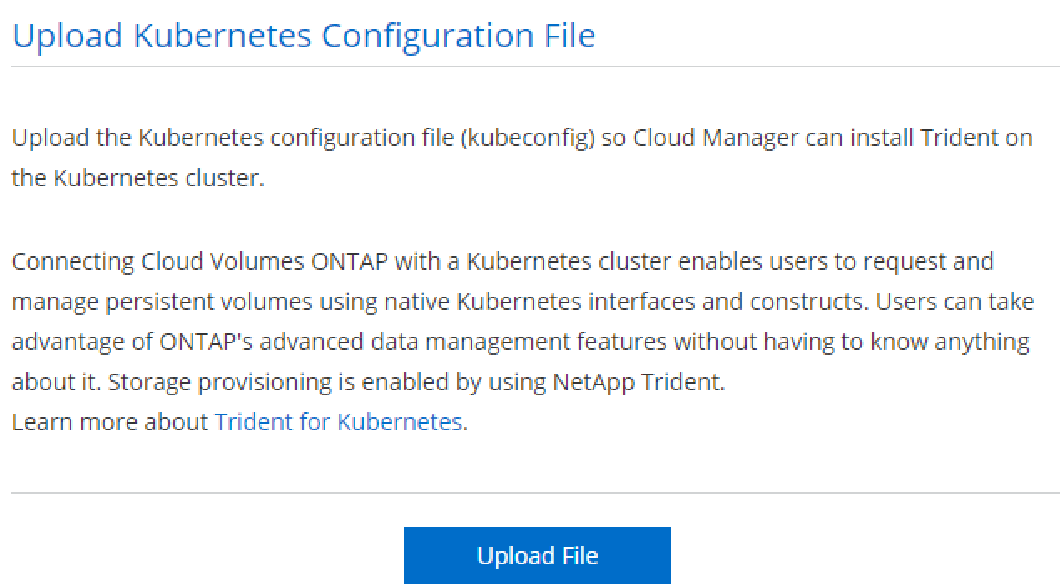
Cloud Deployment Kubernetes Configuration

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.

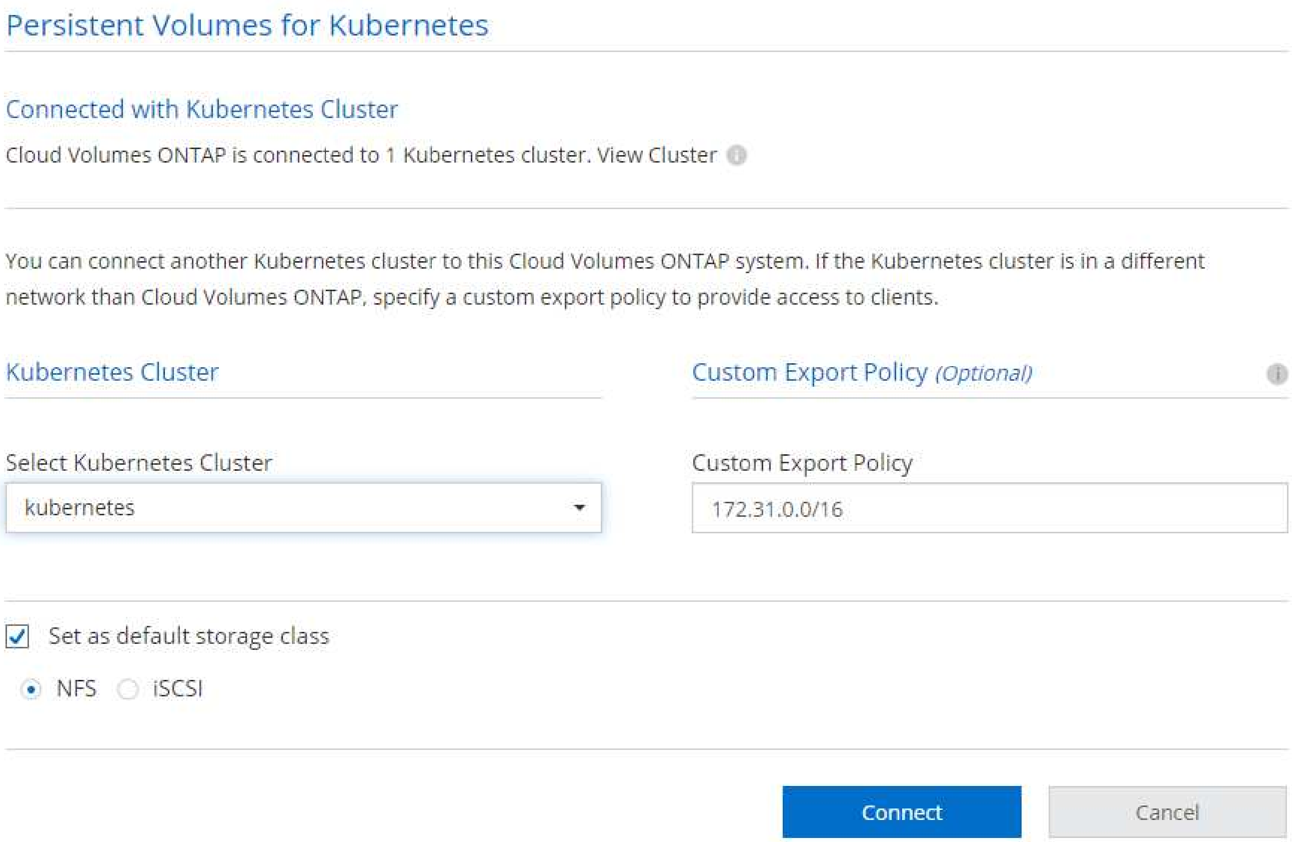
1. To enable access, obtain the Kubernetes config file from one the Iguazio nodes. The file is located under `/home/Iguazio/.kube/config.` Download this file to your desktop.
2. Go to Discover Cluster to configure.



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.

Define Persistent Volume Claim

1. Save the following YAML to a file to create a PVC of type Basic.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: basic
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  storageClassName: netapp-file
```

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.

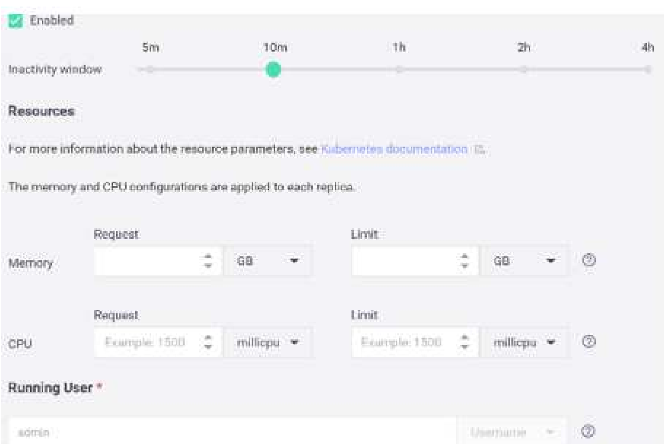
One of the managed services is Jupyter Notebook. Each developer gets its own deployment of a notebook container with the resources they need for development. To give them access to the NetApp Cloud Volume, you can assign the volume to their container and resource allocation, running user, and environment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

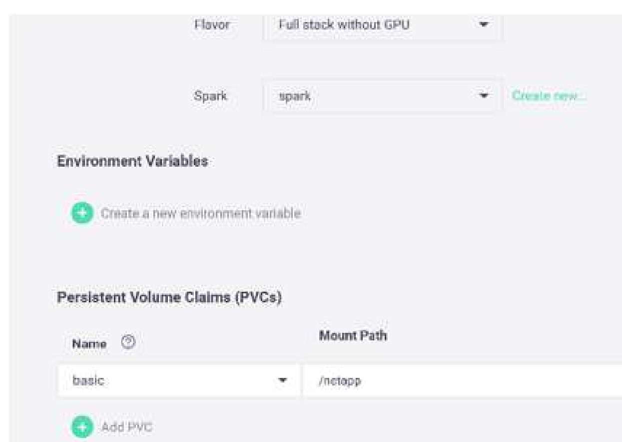
For an on-premises configuration, you can refer to TR-4798 on the Trident setup to enable NetApp ONTAP data management capabilities, such as taking Snapshot copies of your data or model for versioning control. Add the following line in your Trident back- end config file to make Snapshot directories visible:

```
{
    …
    "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>
```





Deploying the Application

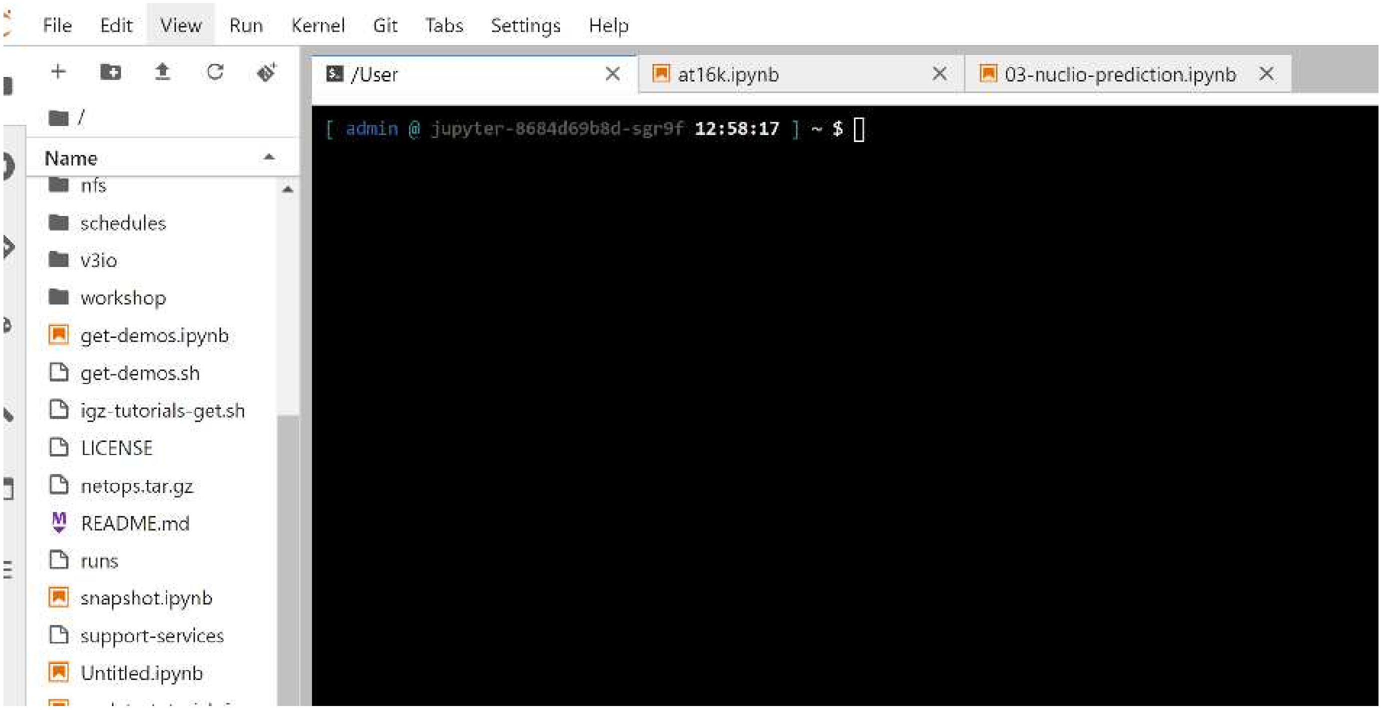
The following sections describe how to install and deploy the application.

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.



At the Jupyter terminal prompt, clone the project.

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

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

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:

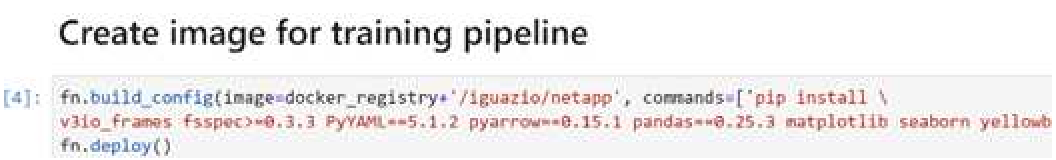
```
MANAGER=ontapClusterMgmtHostname
svm='svm'
email='email'
password=ontapClusterAdminPassword
weid="weid"
volume=sourceVolumeName
```

Create Base Docker Images

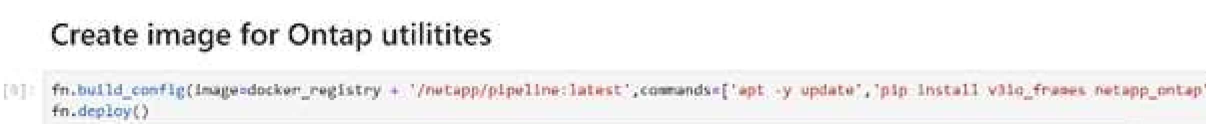
Everything you need to build an ML pipeline is included in the Iguazio platform. The developer can define the specifications of the Docker images required to run the pipeline and execute the image creation from Jupyter Notebook. Open the notebook `create- images.ipynb` and Run All Cells.

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.



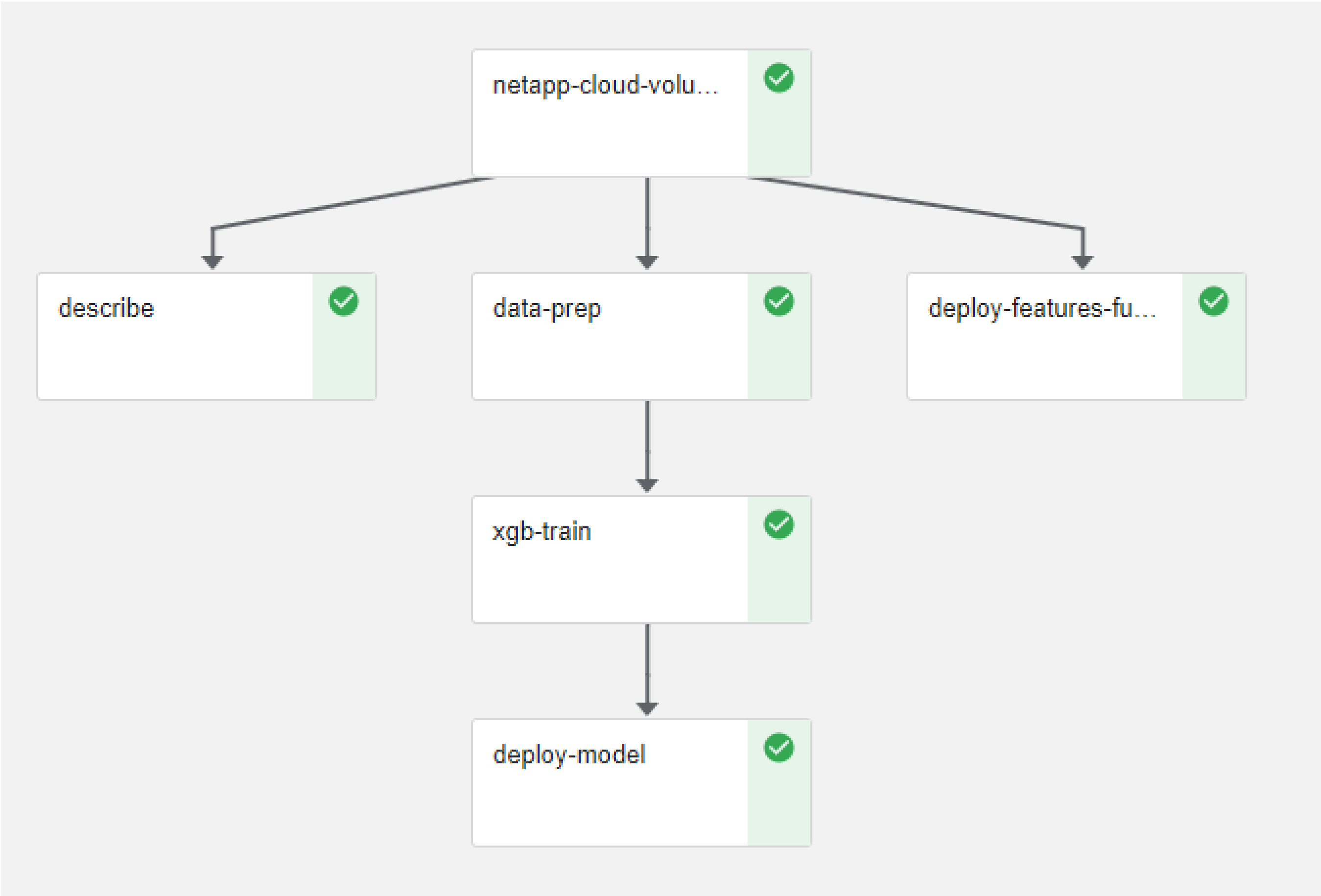
Review Individual Jupyter Notebooks

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.

Libraries/Framework	Description
MLRun	An managed by Iguazio to enable the assembly, execution, and monitoring of an ML/AI pipeline.
Nuclio	A serverless functions framework integrated with Iguazio. Also available as an open-source project managed by Iguazio.
Kubeflow	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.
Docker	A Docker registry run as a service in the Iguazio platform. You can also change this to connect to your registry.
NetApp Cloud Volumes	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.
Trident	Trident is an open-source project managed by NetApp. It facilitates the integration with storage and compute resources in Kubernetes.

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.



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`.

```
# 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.

```
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.

```
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.

```
def handler(context, event):
            …
```

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

```
%%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).

```
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.

```
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.

```
context.log_result('snapVolumeDetails',snap_path)
```

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.

```
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.

```
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.

```
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.

```
# 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.

```
# Initialize the NetApp snap fucntion once for all functions in a notebook
if [ NETAPP_CLOUD_VOLUME ]:
    snapfn =
code_to_function('snap',project='NetApp',kind='job',filename="snap_cv.ipyn
b").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.ipy
nb").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.

Part of the Notebook	Description
<code_to_function> (part of the MLRun module)	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).
image	The name of the Docker image we are using for this step. We created this earlier with the create-image.ipynb notebook.
volume_mounts & volumes	Details to mount the NetApp Cloud Volume at run time.

We also define parameters for the steps.

```
params={   "FEATURES_TABLE":FEATURES_TABLE,
           "SAVE_TO" : SAVE_TO,
           "metrics_table" : metrics_table,
           'FROM_TSDB': 0,
           'PREDICTIONS_TABLE': PREDICTIONS_TABLE,
           'TRAIN_ON_LAST': '1d',
           'TRAIN_SIZE':0.7,
           'NUMBER_OF_SHARDS' : 4,
           'MODEL_FILENAME' : 'netops.v3.model.pickle',
           'APP_DIR' : APP_DIR,
           'FUNCTION_NAME' : 'netops-inference',
           'PROJECT_NAME' : 'netops',
           'NETAPP_SIM' : NETAPP_SIM,
           'NETAPP_MOUNT_PATH': NETAPP_MOUNT_PATH,
           'NETAPP_PVC_CLAIM' : NETAPP_PVC_CLAIM,
           'IGZ_CONTAINER_PATH' : IGZ_CONTAINER_PATH,
           'IGZ_MOUNT_PATH' : IGZ_MOUNT_PATH
            }
```

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:

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

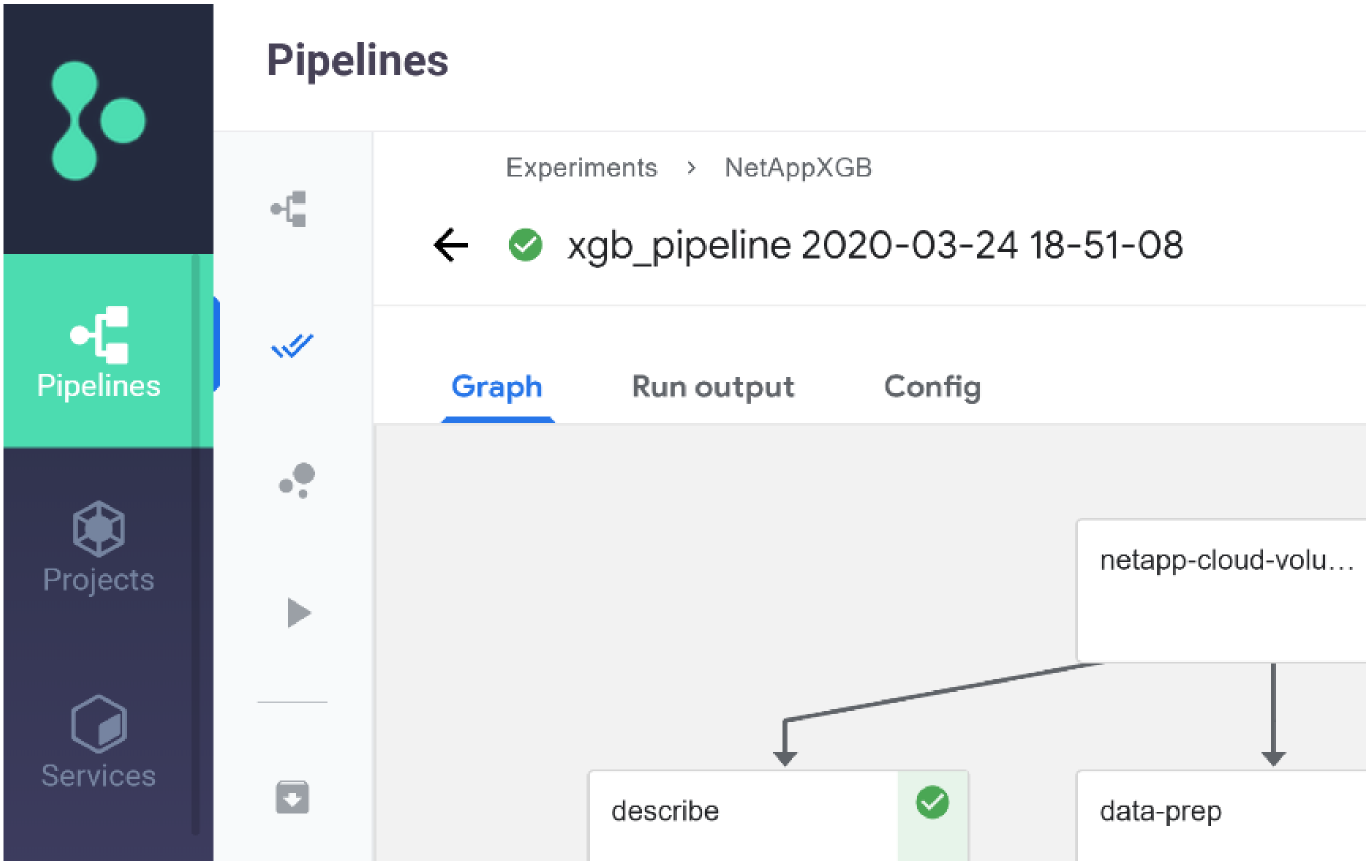
Parameters	Details
NewTask	NewTask is the definition of the function run.
(MLRun module)	Handler. Name of the Python function to invoke. We used the name handler in the notebook, but it is not required. params. The parameters we passed to the execution. Inside our code, we use context.get_param (‘PARAMETER’) to get the values.

Parameters	Details
as_step	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.

```
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)
```

Parameters	Details
inputs	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.
out_path	A location to place artifacts generating using the MLRun module log_artifacts.

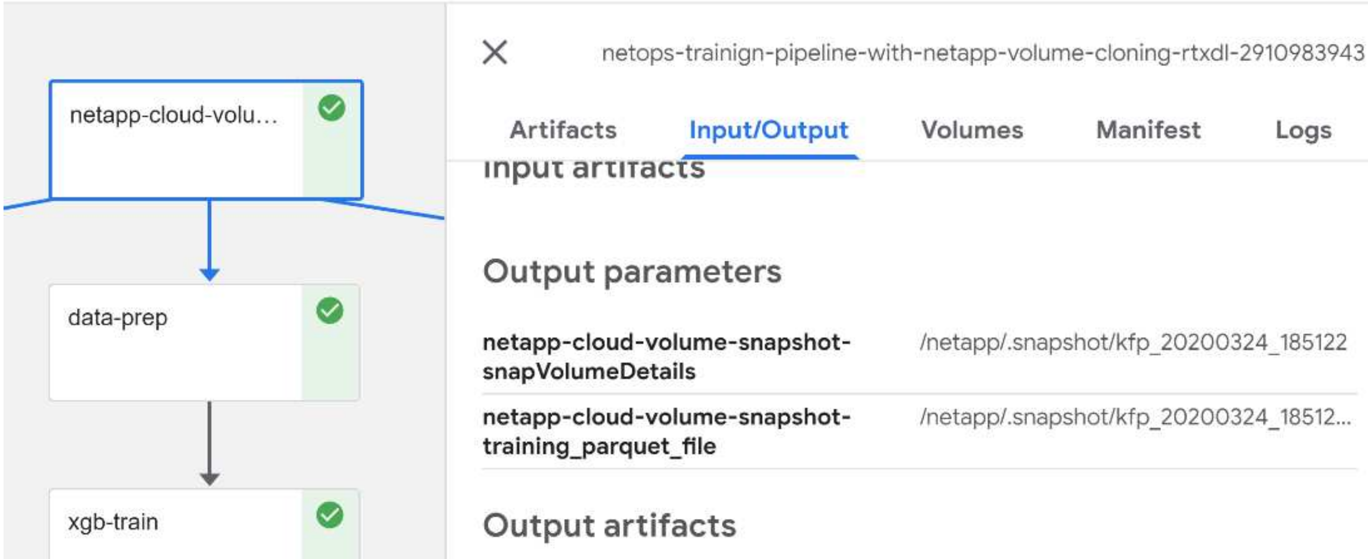
You can run `pipeline.ipynb` from top to bottom. You can then go to the Pipelines tab from the Iguazio dashboard to monitor progress as seen in the Iguazio dashboard Pipelines tab.



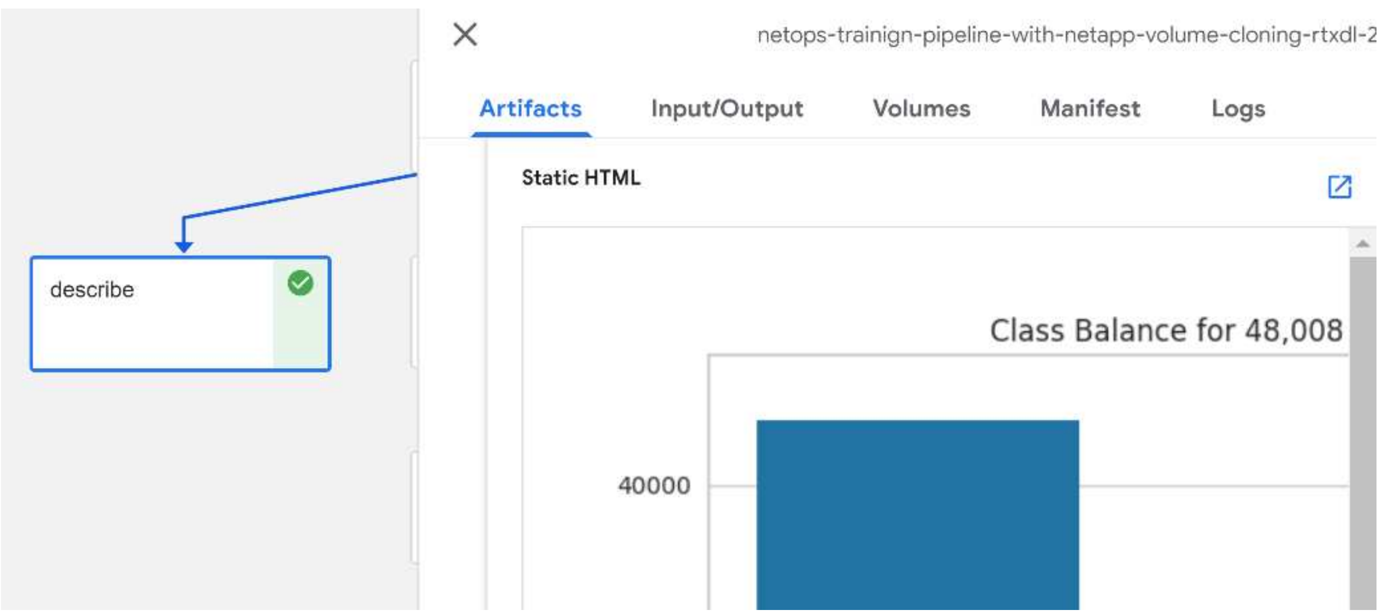
Because we logged the accuracy of training step in every run, we have a record of accuracy for each experiment, as seen in the record of training accuracy.

Run name	Status	Duration	Pipeline Version	Recurring ...	Start time	accuracy
xgb_pipeline 2020-03-24 18-51-...	✓	0:08:43	[View pipeline]	-	3/24/2020, 2:51:09 PM	0.985
xgb_pipeline 2020-03-19 13-31-...	✓	0:08:14	[View pipeline]	-	3/19/2020, 9:31:19 AM	0.980
xgb_pipeline 2020-03-18 12-56-...	✓	0:08:11	[View pipeline]	-	3/18/2020, 8:56:08 AM	0.990
xgb_pipeline 2020-03-17 19-49-...	✓	0:08:03	[View pipeline]	-	3/17/2020, 3:49:31 PM	0.985
xgb_pipeline 2020-03-17 18-34-...	✓	0:05:54	[View pipeline]	-	3/17/2020, 2:34:56 PM	0.980
xgb_pipeline 2020-03-17 17-34-...	✓	0:04:48	[View pipeline]	-	3/17/2020, 1:34:16 PM	0.982
xgb_pipeline 2020-03-17 17-01-...	✓	0:05:25	[View pipeline]	-	3/17/2020, 1:01:58 PM	0.987
xgb_pipeline 2020-03-16 16-47-...	✓	0:06:08	[View pipeline]	-	3/16/2020, 12:47:19 ...	0.983
xgb_pipeline 2020-03-16 13-57-...	✓	0:05:18	[View pipeline]	-	3/16/2020, 9:57:03 AM	0.980

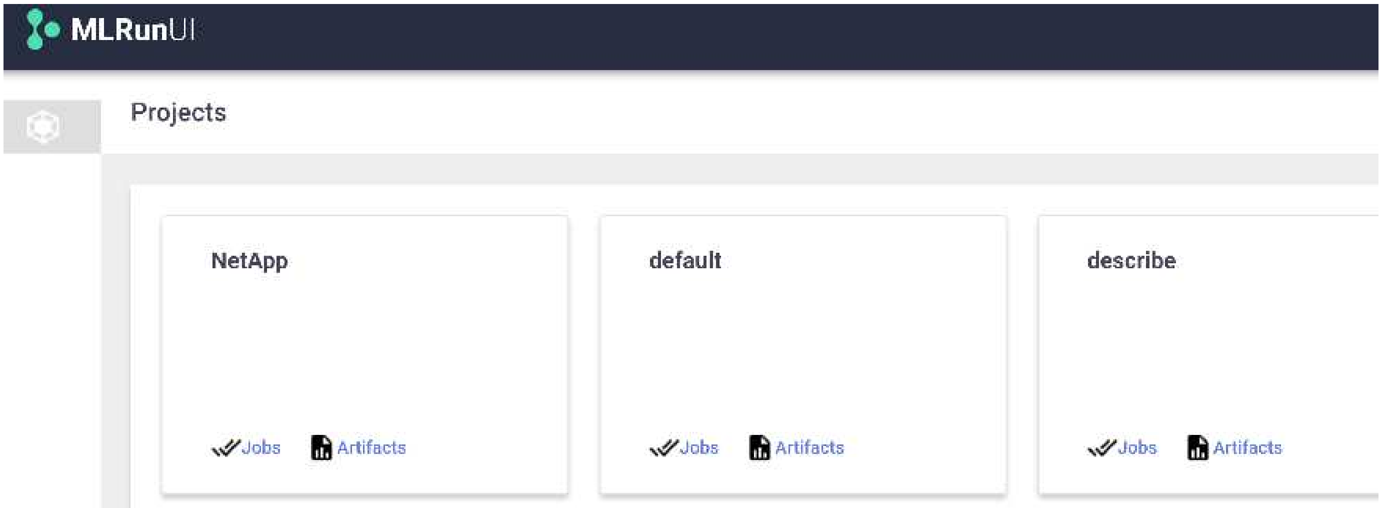
If you select the Snapshot step, you can see the name of the Snapshot copy that was used to run this experiment.



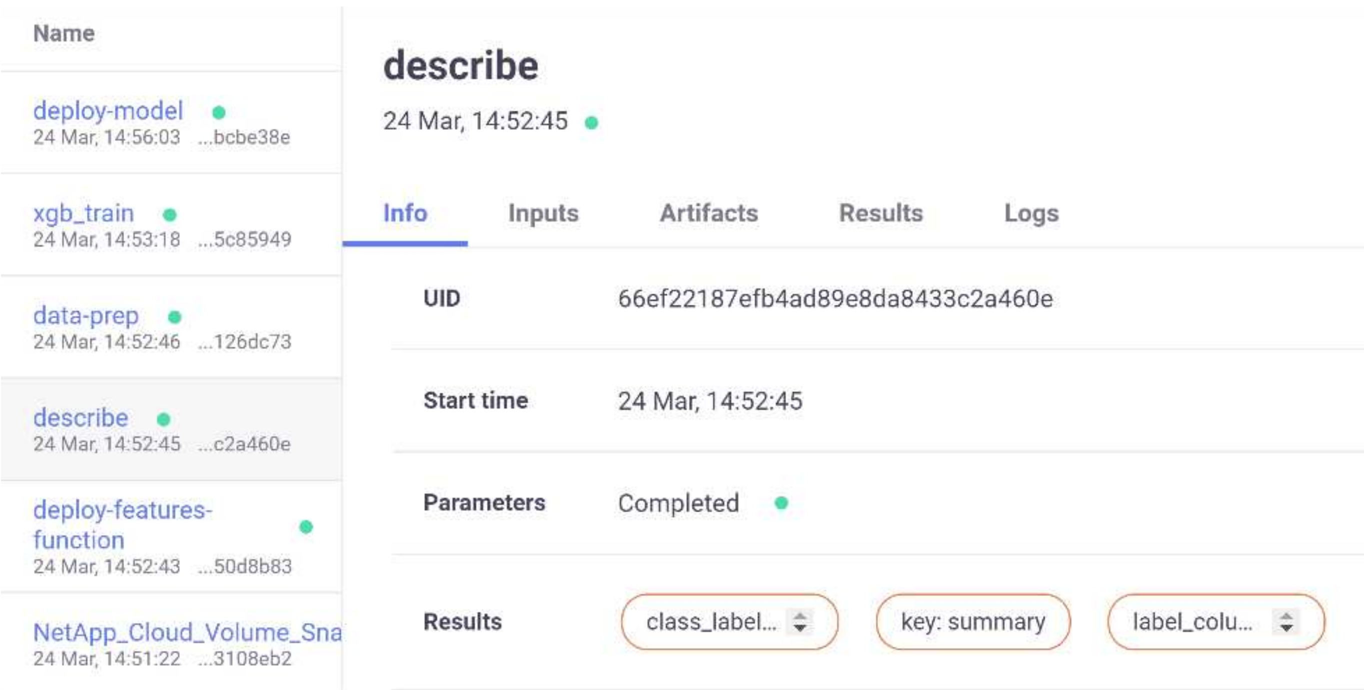
The described step has visual artifacts to explore the metrics we used. You can expand to view the full plot as seen in the following image.



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.



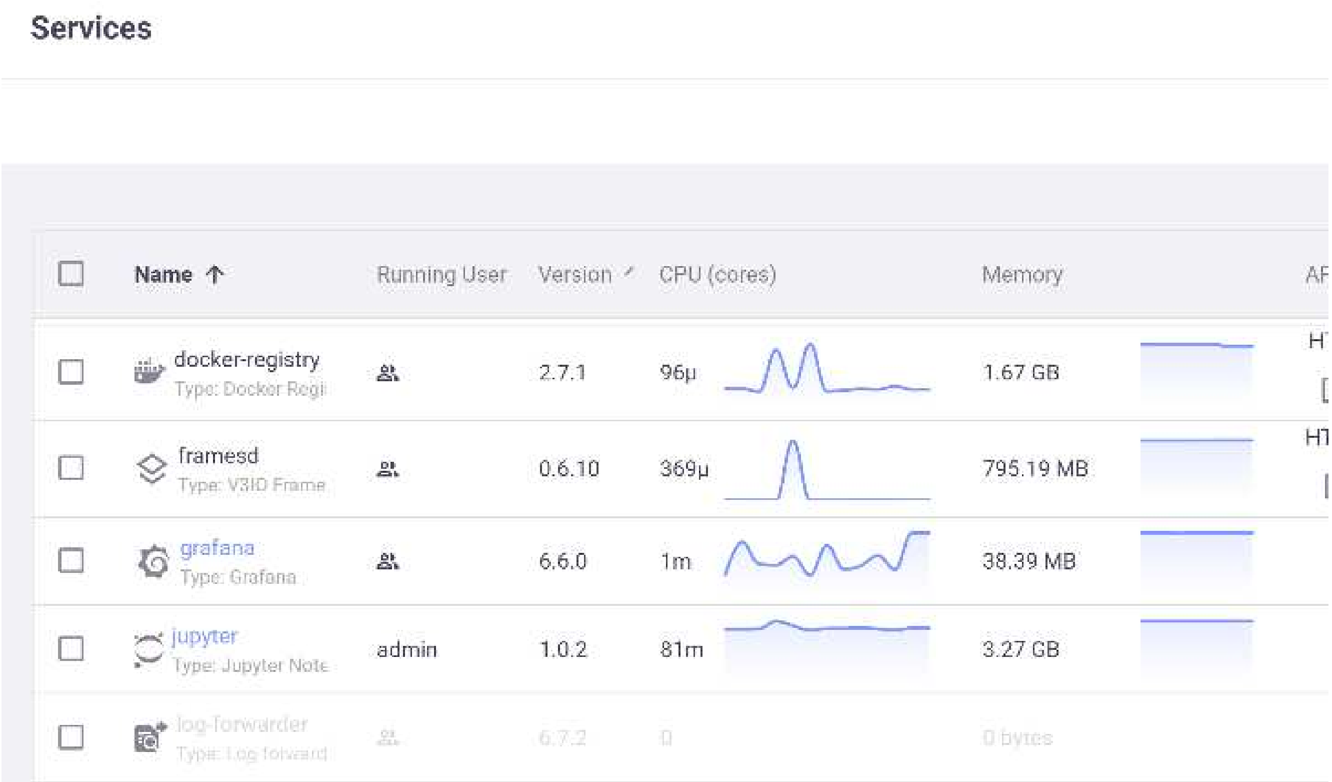
There is more information about MLRun than we can cover in this document. Al artifacts, including the definition of the steps and functions, can be saved to the API database, versioned, and invoked individually or as a full project. Projects can also be saved and pushed to Git for later use. We encourage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

Deploy Grafana Dashboard

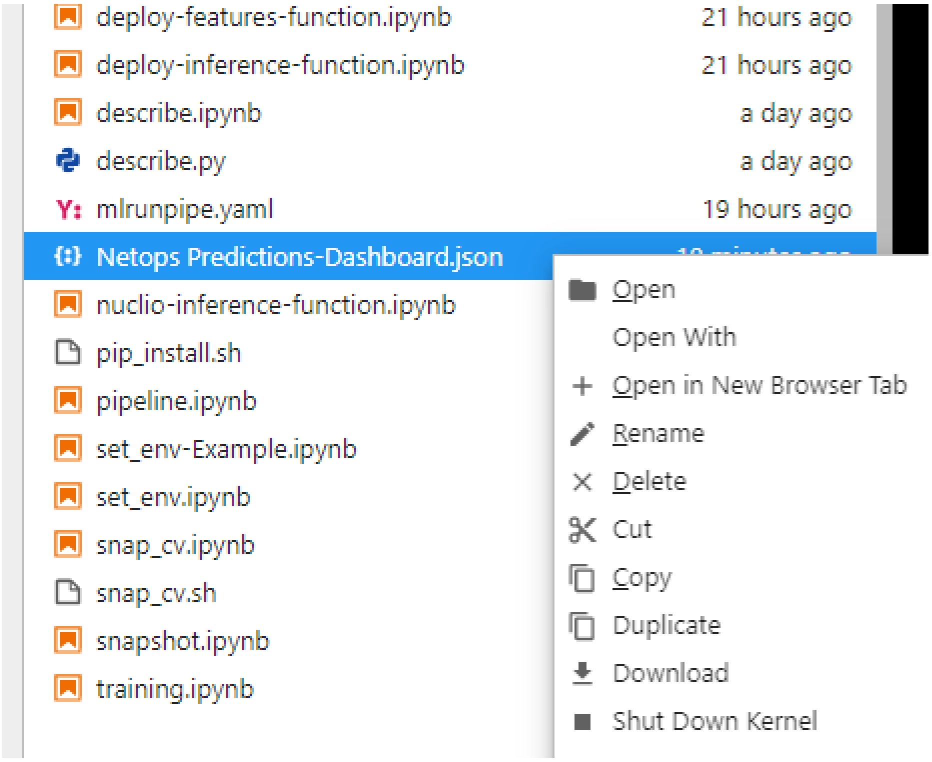
After everything is deployed, we run inferences on new data. The models predict failure on network device equipment. The results of the prediction are stored in an Iguazio TimeSeries table. You can visualize the results with Grafana in the platform integrated with Iguazio’s security and data access policy.

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

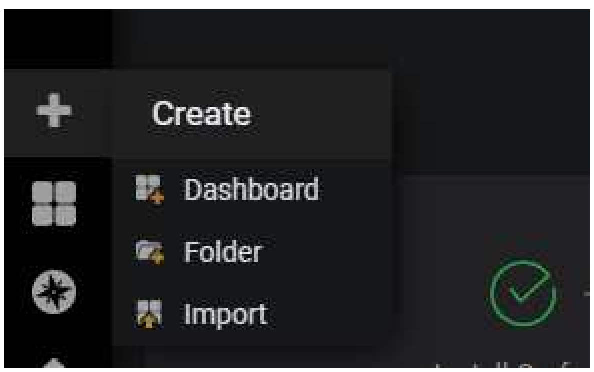
1. To verify that the Grafana service is running, look under Services.



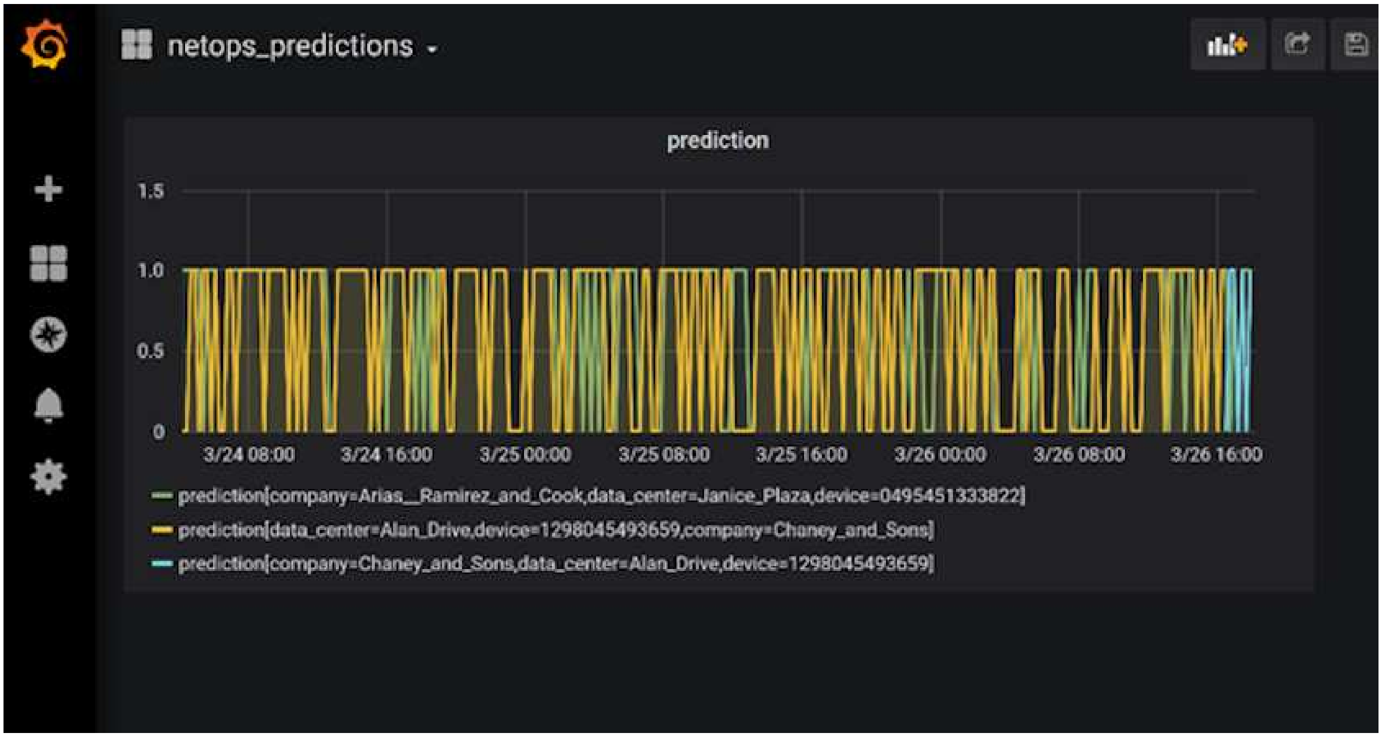
2. If it is not present, deploy an instance from the Services section:
   a. Click New Service.
   b. Select Grafana from the list.
   c. Accept the defaults.
   d. Click Next Step.
   e. Enter your user ID.
   f. Click Save Service.
   g. Click Apply Changes at the top.
3. To deploy the dashboard, download the file `NetopsPredictions-Dashboard.json` through the Jupyter interface.



4. Open Grafana from the Services section and import the dashboard.



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



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.

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.

TR-4915: Data movement with E-Series and BeeGFS for AI and analytics workflows

Cody Harryman and Ryan Rodine, NetApp

TR-4915 describes how to move data from any data repository into a BeeGFS file system backed by NetApp E-Series SAN storage. For artificial intelligence (AI) and machine learning (ML) applications, customers might routinely need to move large data sets exceeding many petabytes of data into their BeeGFS clusters for model development. This document explores how to accomplish this by using NetApp XCP and NetApp BlueXP Copy and Sync tools.

TR-4915: Data movement with E-Series and BeeGFS for AI and analytics workflows

Copyright information

Copyright © 2024 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.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

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.