Amazon FSX for NetApp ONTAP (FSX ONTAP) für MLOps

NetApp Solutions

NetApp
December 19, 2024

Inhalt

Amazon FSX for NetApp ONTAP (FSX ONTAP) für MLOps

Amazon FSX for NetApp ONTAP (FSX ONTAP) für MLOps

In diesem Abschnitt wird die praktische Anwendung der Entwicklung von KI-Infrastrukturen erläutert, indem der Aufbau einer MLOps-Pipeline mithilfe von FSX ONTAP vollständig demonstriert wird. Sie umfasst drei umfassende Beispiele und hilft Ihnen, Ihre MLOps-Anforderungen über diese leistungsstarke Datenmanagement-Plattform zu erfüllen.

Autor(en):
Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

Die Schwerpunkte dieser Artikel liegen auf:

1. "Teil 1 – Integration von Amazon FSX for NetApp ONTAP (FSX ONTAP) als privaten S3-Bucket in AWS SageMaker"
2. "Teil 2 – Nutzung von Amazon FSX for NetApp ONTAP (FSX ONTAP) als Datenquelle für Modelltraining in SageMaker"
3. "Teil 3 – Aufbau Einer vereinfachten MLOPS-Pipeline (CI/CT/CD)"

Am Ende dieses Abschnitts haben Sie ein fundiertes Verständnis darüber gewonnen, wie Sie FSX ONTAP zur Optimierung von MLOPS-Prozessen nutzen können.

Teil 1 – Integration von Amazon FSX for NetApp ONTAP (FSX ONTAP) als privaten S3-Bucket in AWS SageMaker

Dieser Abschnitt enthält einen Leitfaden zur Konfiguration von FSX ONTAP als privaten S3-Bucket mit AWS SageMaker.

Autor(en):
Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

Einführung

Anhand von SageMaker als Beispiel enthält diese Seite Anleitungen zur Konfiguration von FSX ONTAP als privaten S3-Bucket.

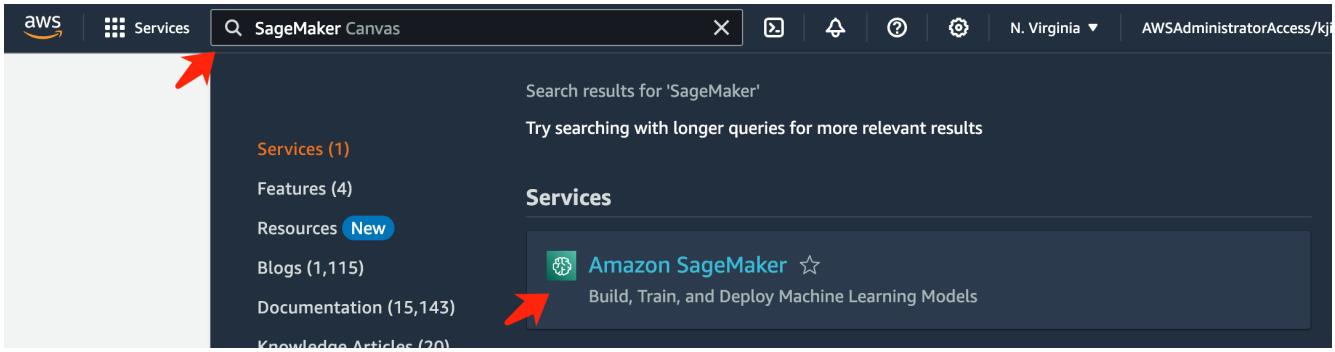
Für weitere Informationen über FSX ONTAP, nehmen Sie bitte einen Blick auf diese Präsentation ("Video-Link")

Benutzerhandbuch

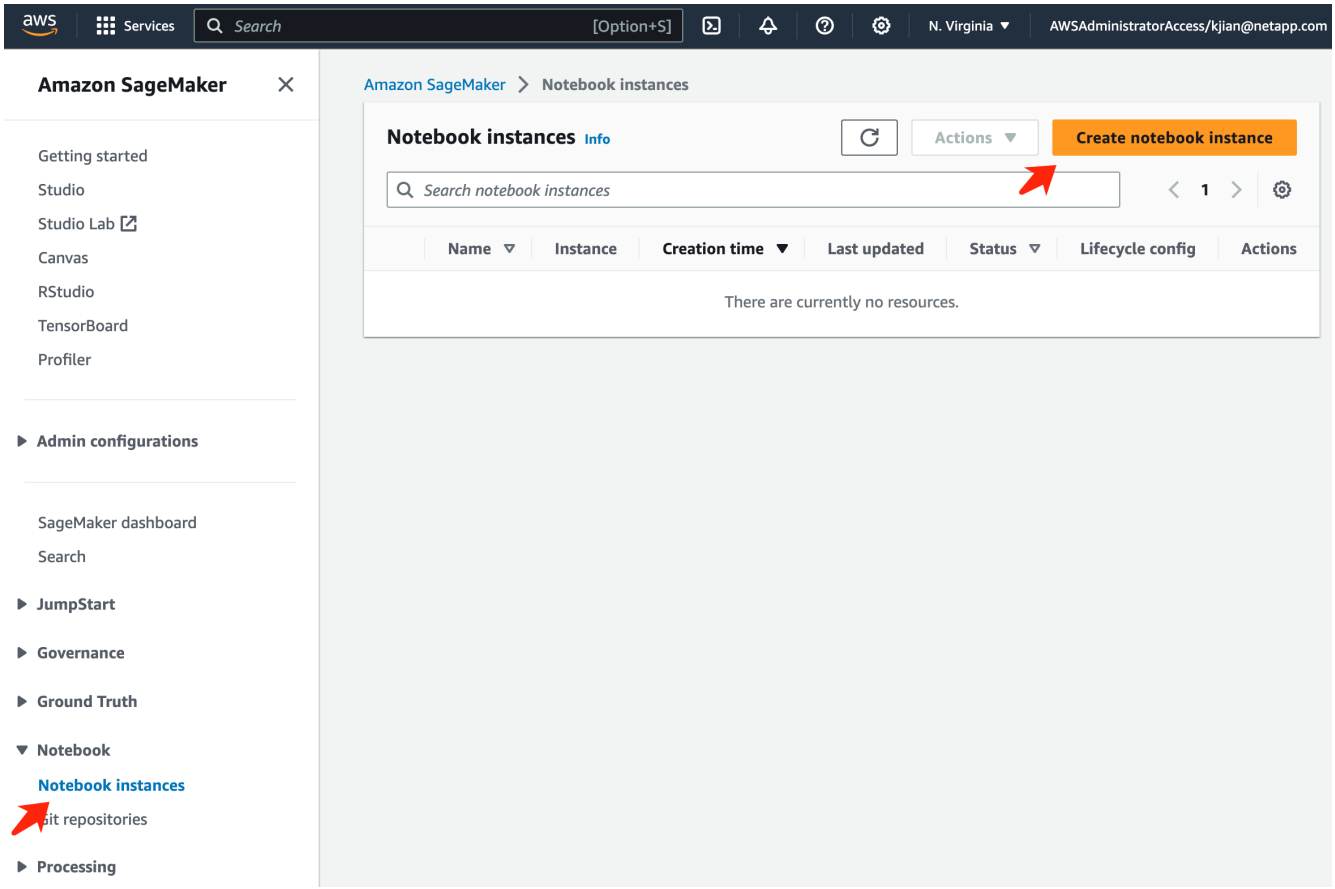
Server-Erstellung

Erstellen Sie eine SageMaker-Notebook-Instanz

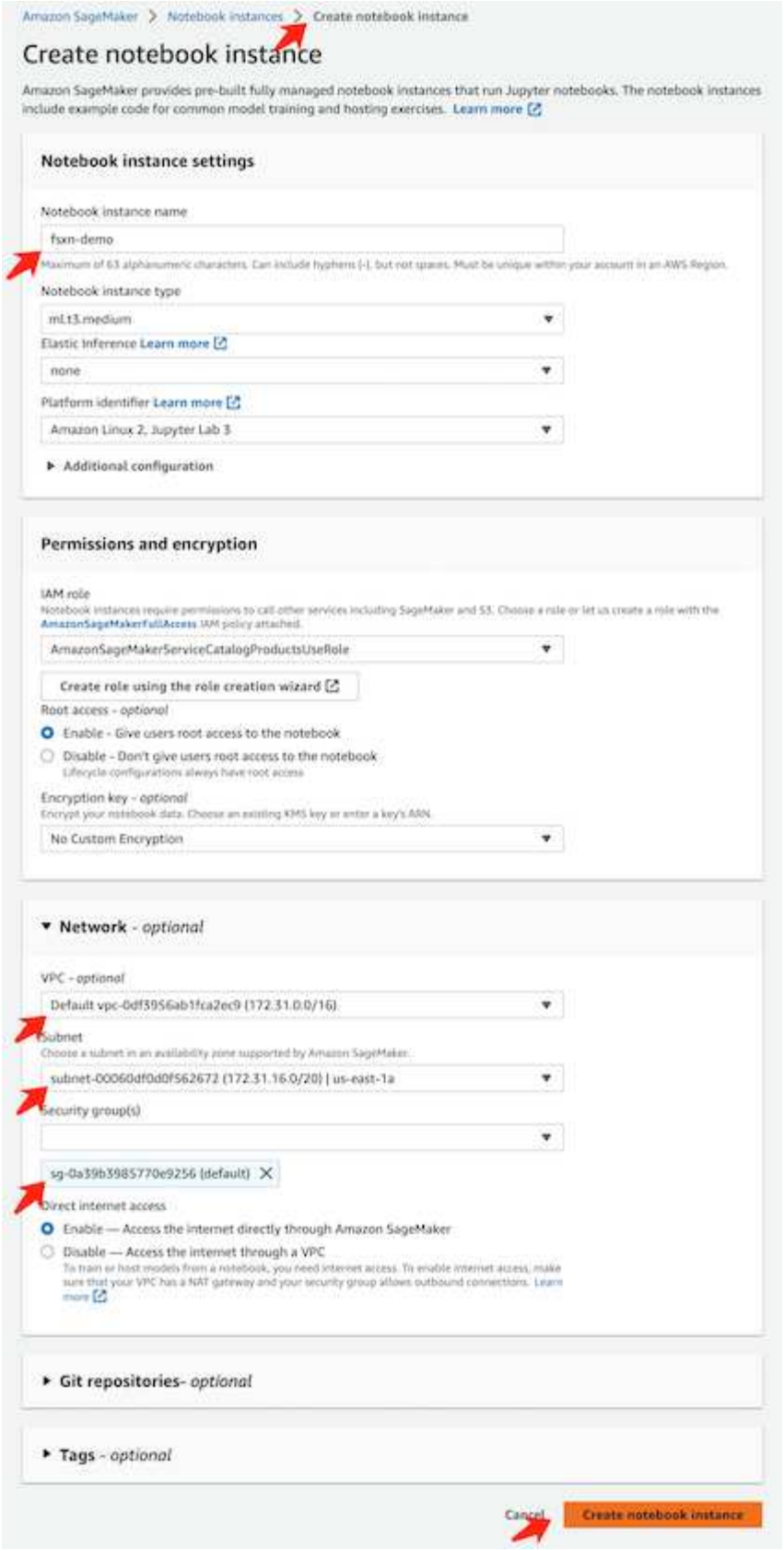
1. Öffnen Sie die AWS-Konsole. Suchen Sie im Suchfeld nach SageMaker und klicken Sie auf den Dienst **Amazon SageMaker**.



2. Öffnen Sie die **Notizbuchinstanzen** unter der Registerkarte Notizbuch, klicken Sie auf die orangefarbene Schaltfläche **Notizbuchinstanz erstellen**.

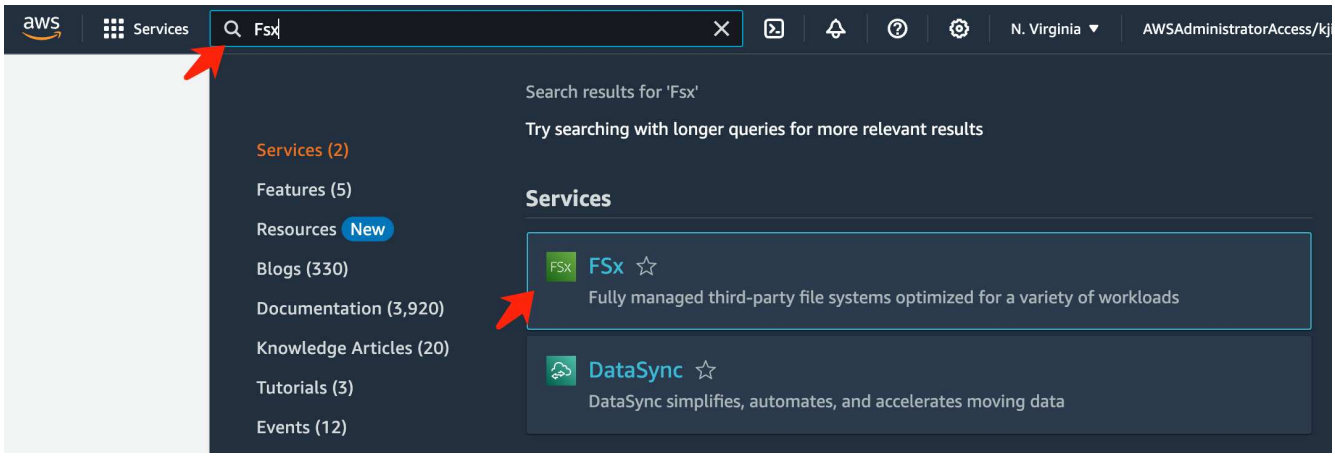


3. Geben Sie auf der Erstellungsseite den Namen der **Notebook-Instanz** ein erweitern Sie das Feld **Netzwerk** andere Einträge standardmäßig lassen und wählen Sie eine **VPC**, **Subnetz** und **Sicherheitsgruppe(n)** aus. (Diese **VPC** und **Subnetz** werden später verwendet, um FSX ONTAP Dateisystem zu erstellen) Klicken Sie auf die orangefarbene Schaltfläche **Notizbuchinstanz erstellen** unten rechts.

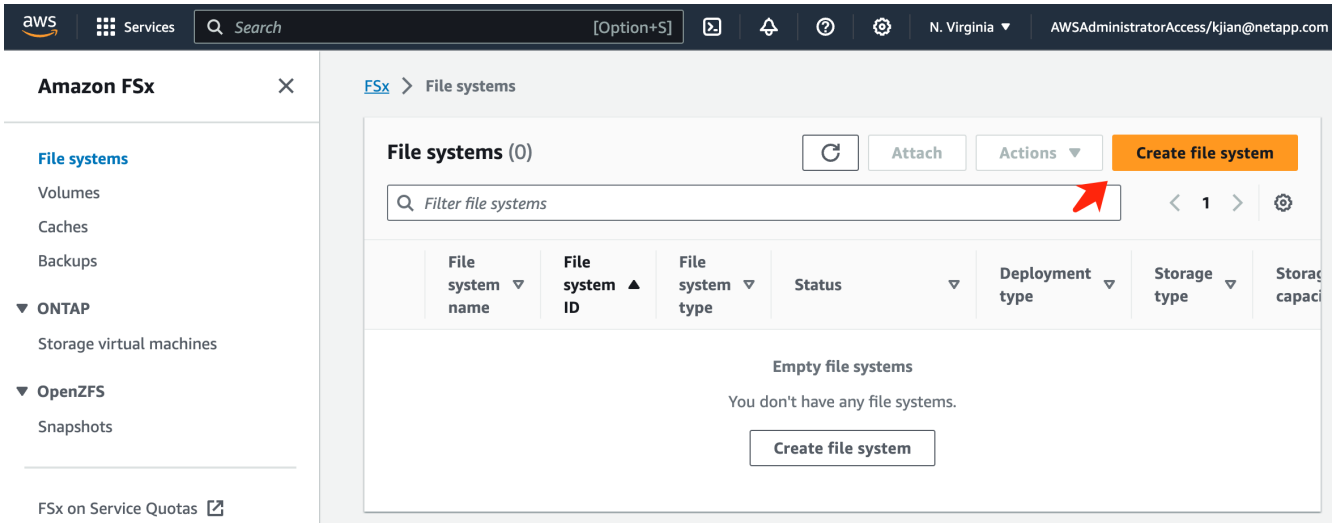


Erstellen Sie ein FSX ONTAP Dateisystem

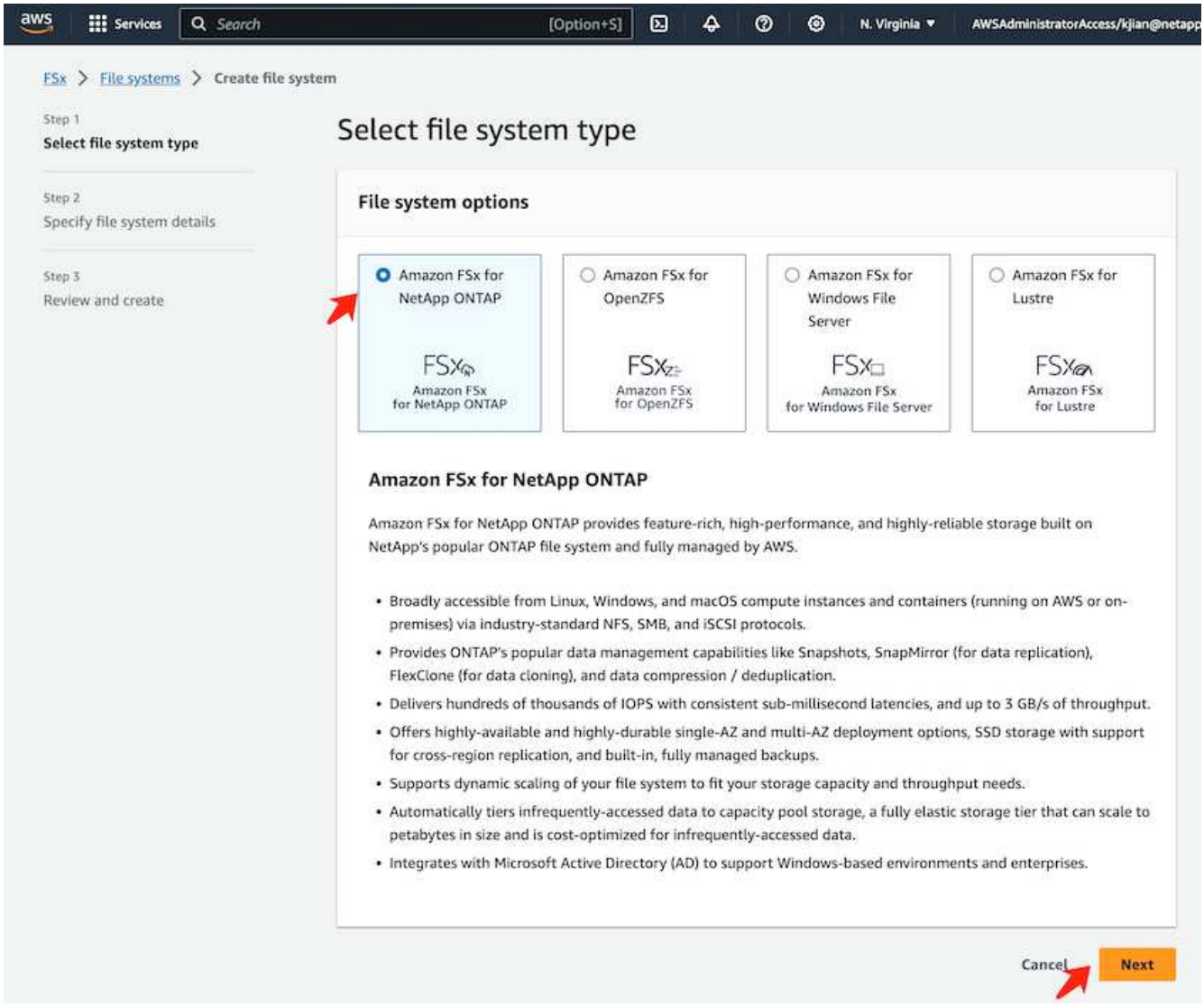
1. Öffnen Sie die AWS-Konsole. Suchen Sie im Suchfeld FSX und klicken Sie auf den Dienst **FSX**.



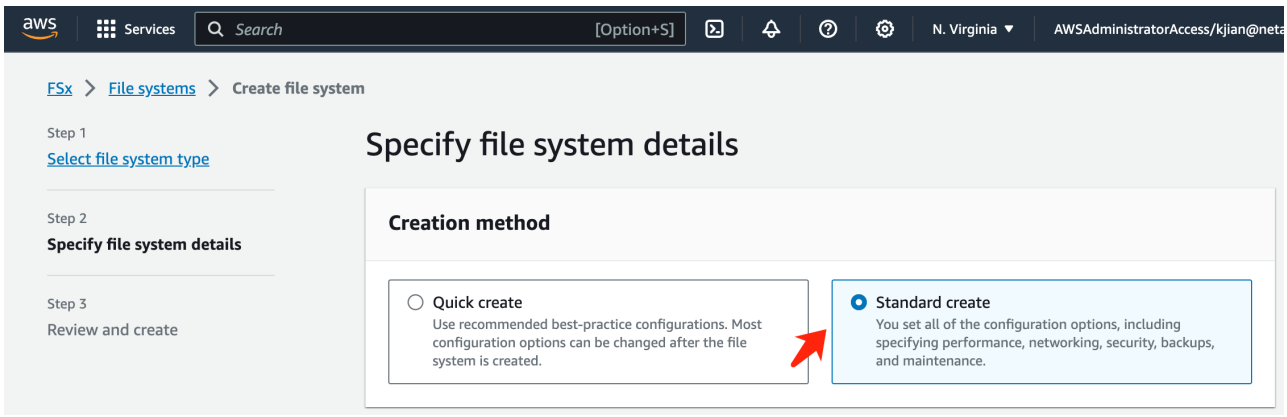
2. Klicken Sie auf **Dateisystem erstellen**.



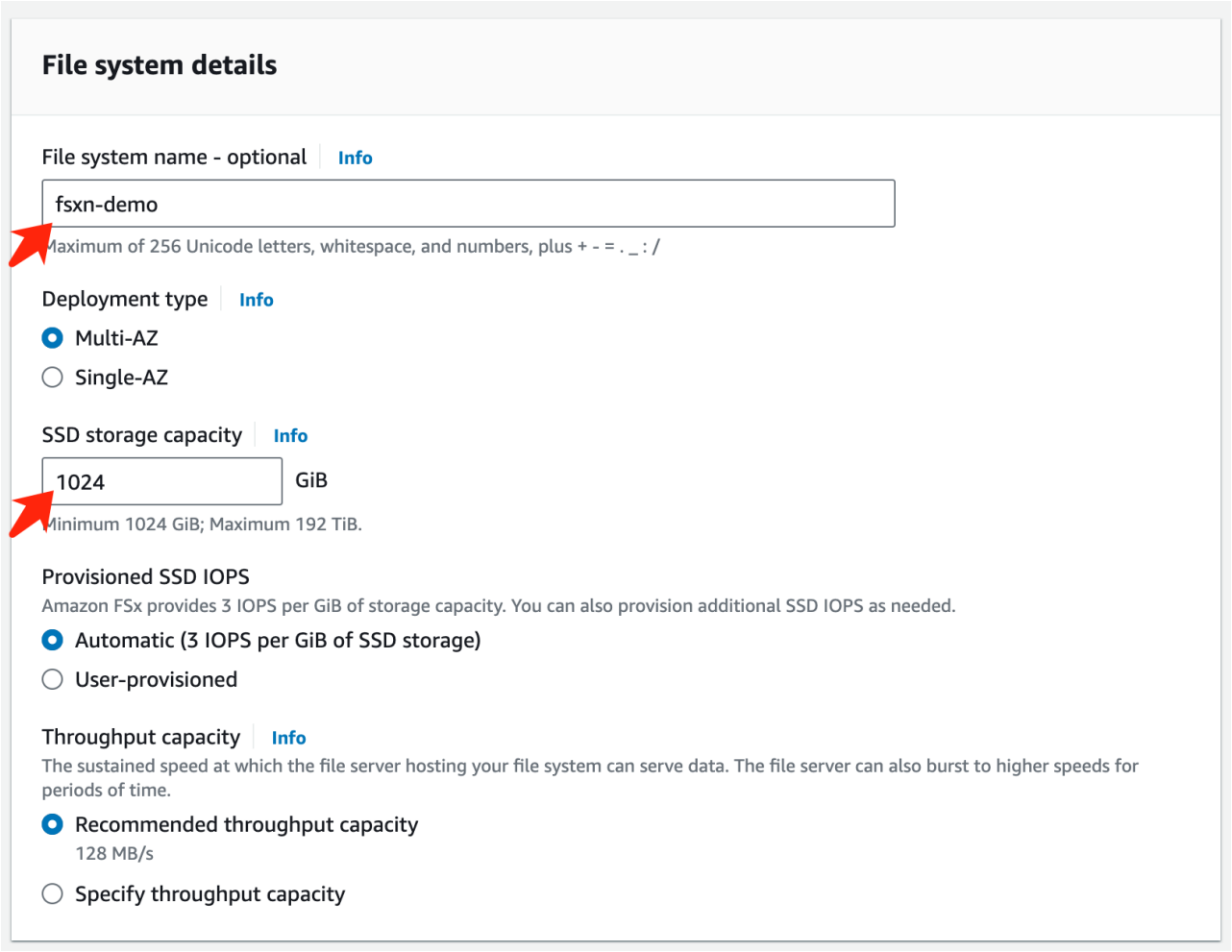
3. Wählen Sie die erste Karte **FSX ONTAP** und klicken Sie auf **Weiter**.



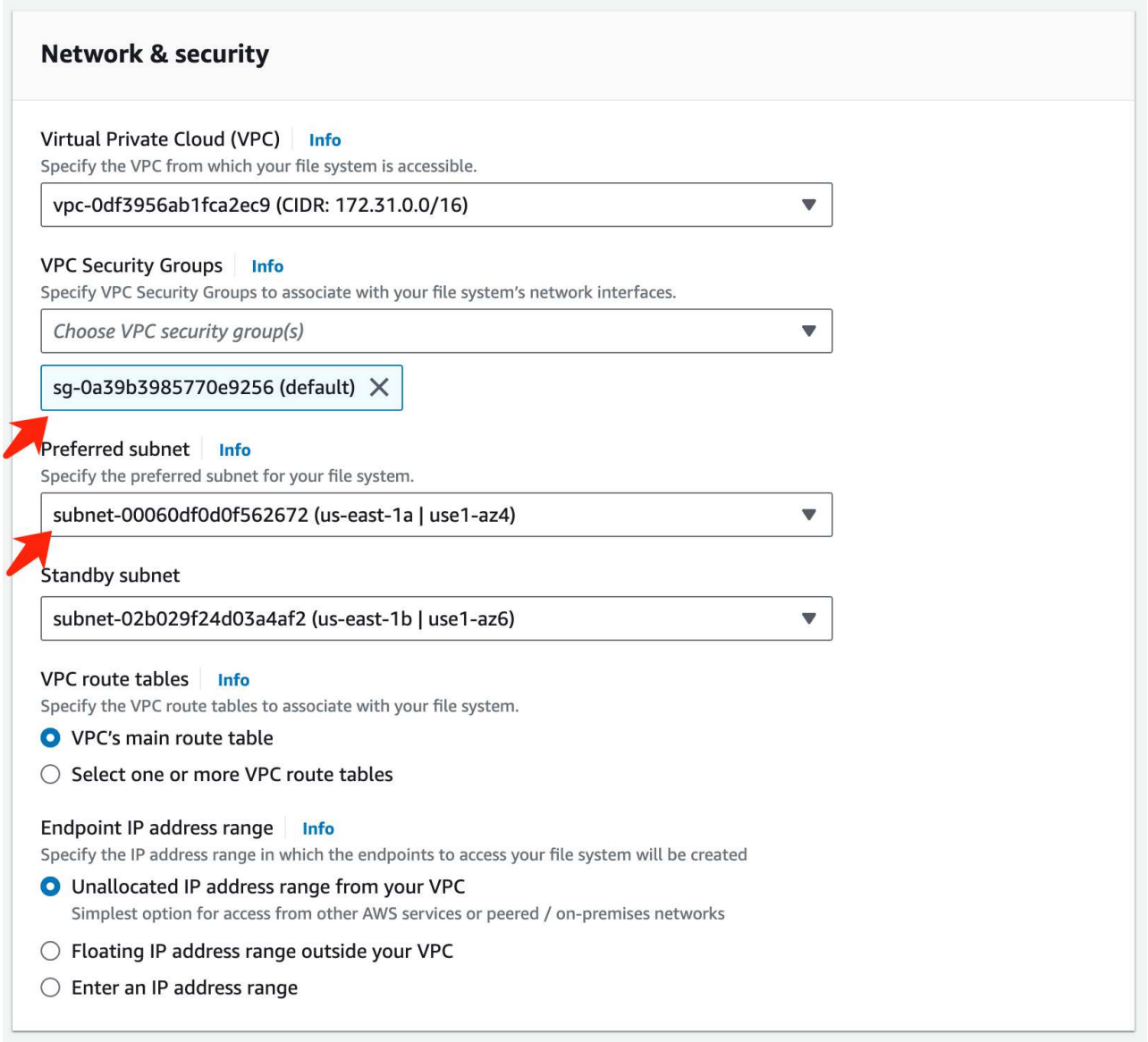
4. Auf der Konfigurationsseite für Details.
   a. Wählen Sie die Option **Standard create**.



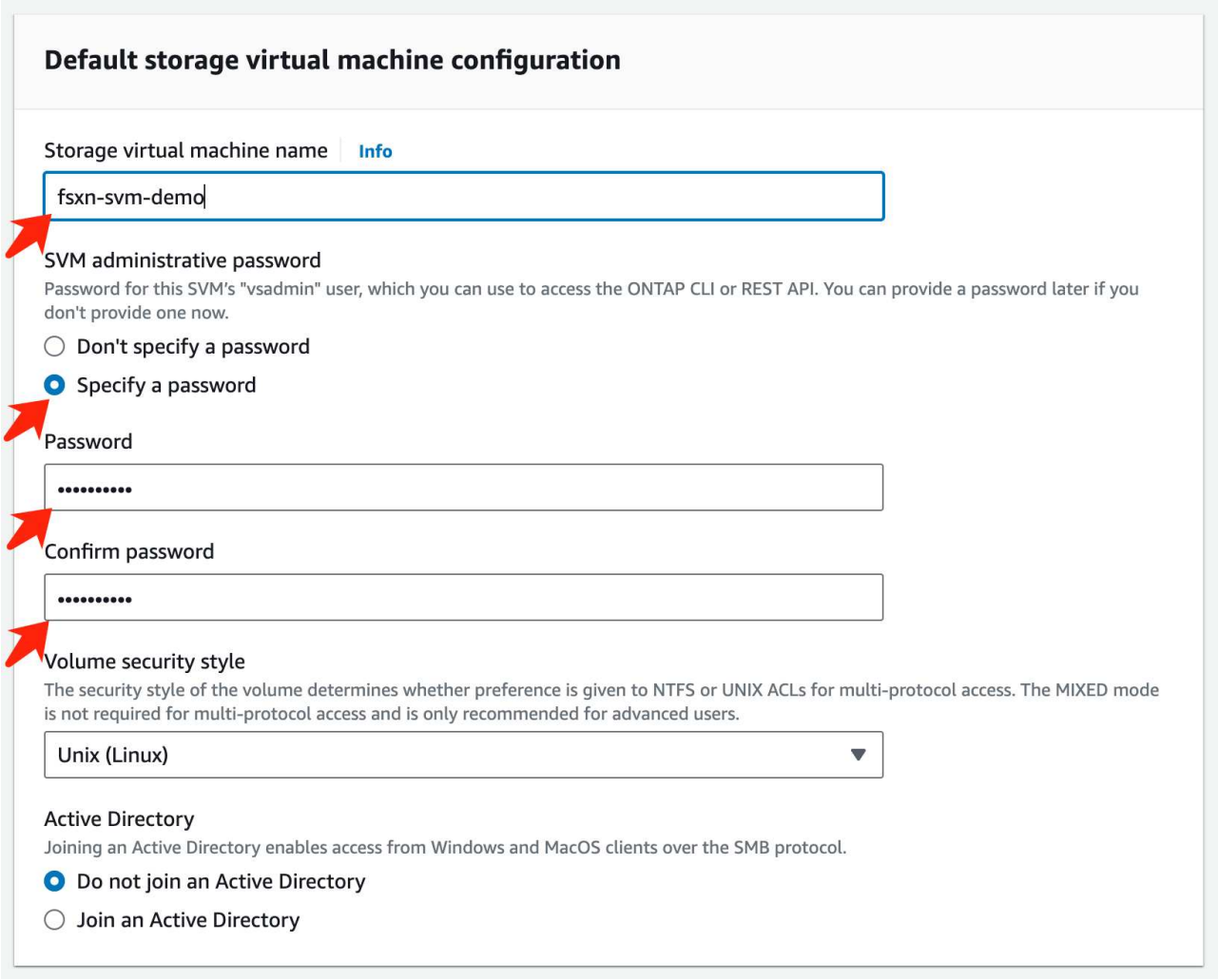
   b. Geben Sie den Namen des **Dateisystems** und die Kapazität des **SSD-Speichers** ein.



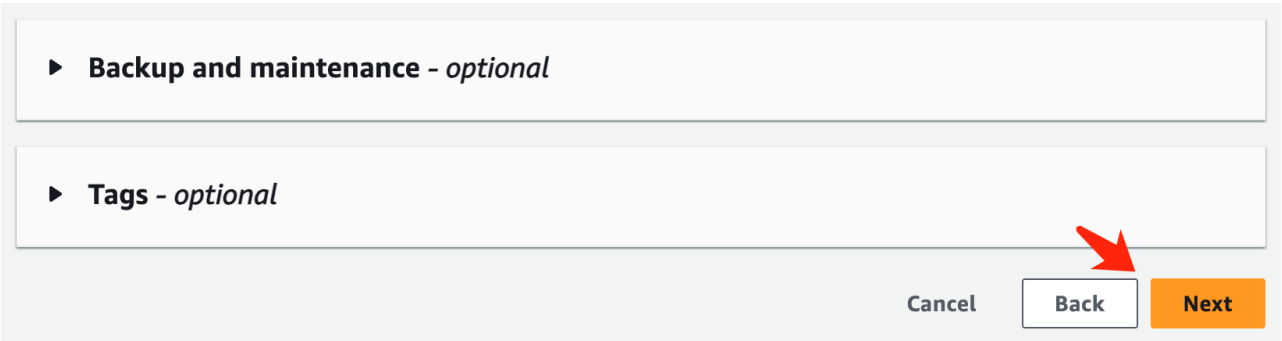
c. Stellen Sie sicher, dass Sie die **VPC** und **Subnetz** mit der **SageMaker Notebook**-Instanz identisch verwenden.



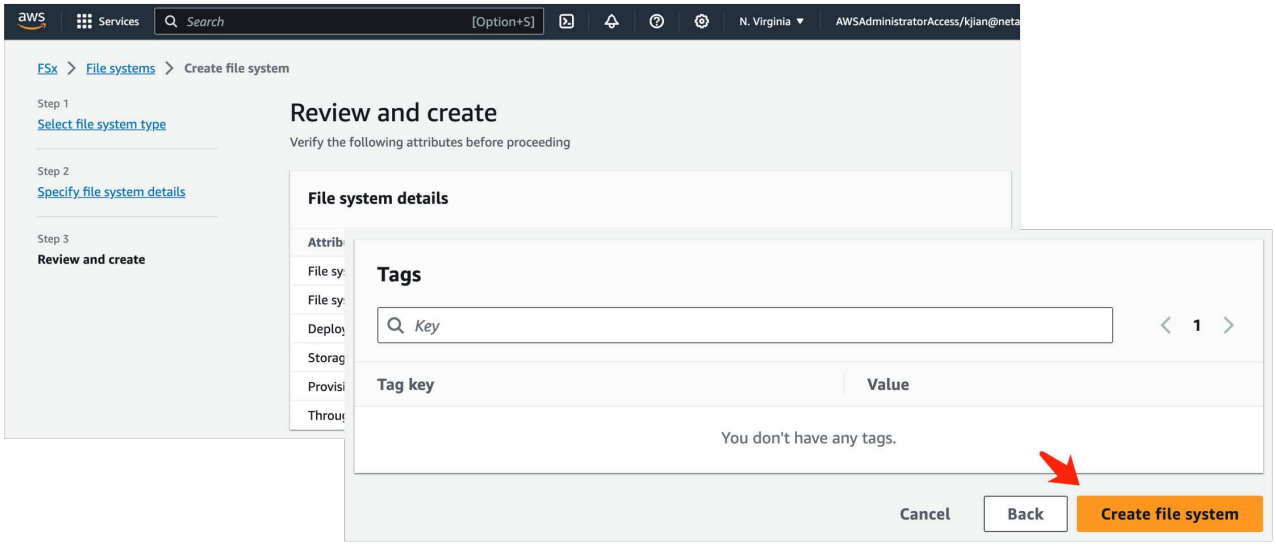
d. Geben Sie den Namen der **Storage Virtual Machine** und **Geben Sie ein Passwort** für Ihre SVM (Storage Virtual Machine) ein.



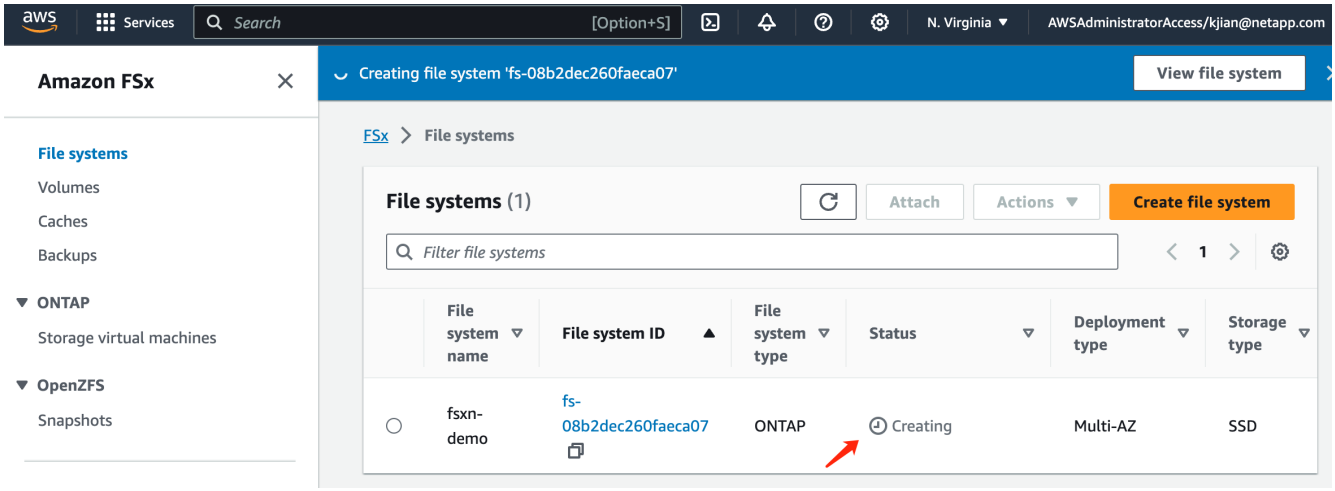
e. Lassen Sie andere Einträge standardmäßig und klicken Sie auf die orangefarbene Schaltfläche **Weiter** unten rechts.



f. Klicken Sie auf die orangefarbene Schaltfläche **Dateisystem erstellen** unten rechts auf der Überprüfungsseite.



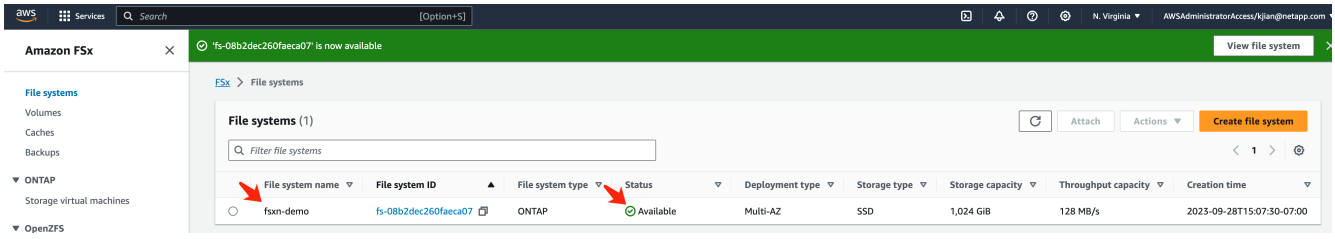
5. Es kann ungefähr 20-40 Minuten* dauern, um das FSX Dateisystem hochzudrehen.



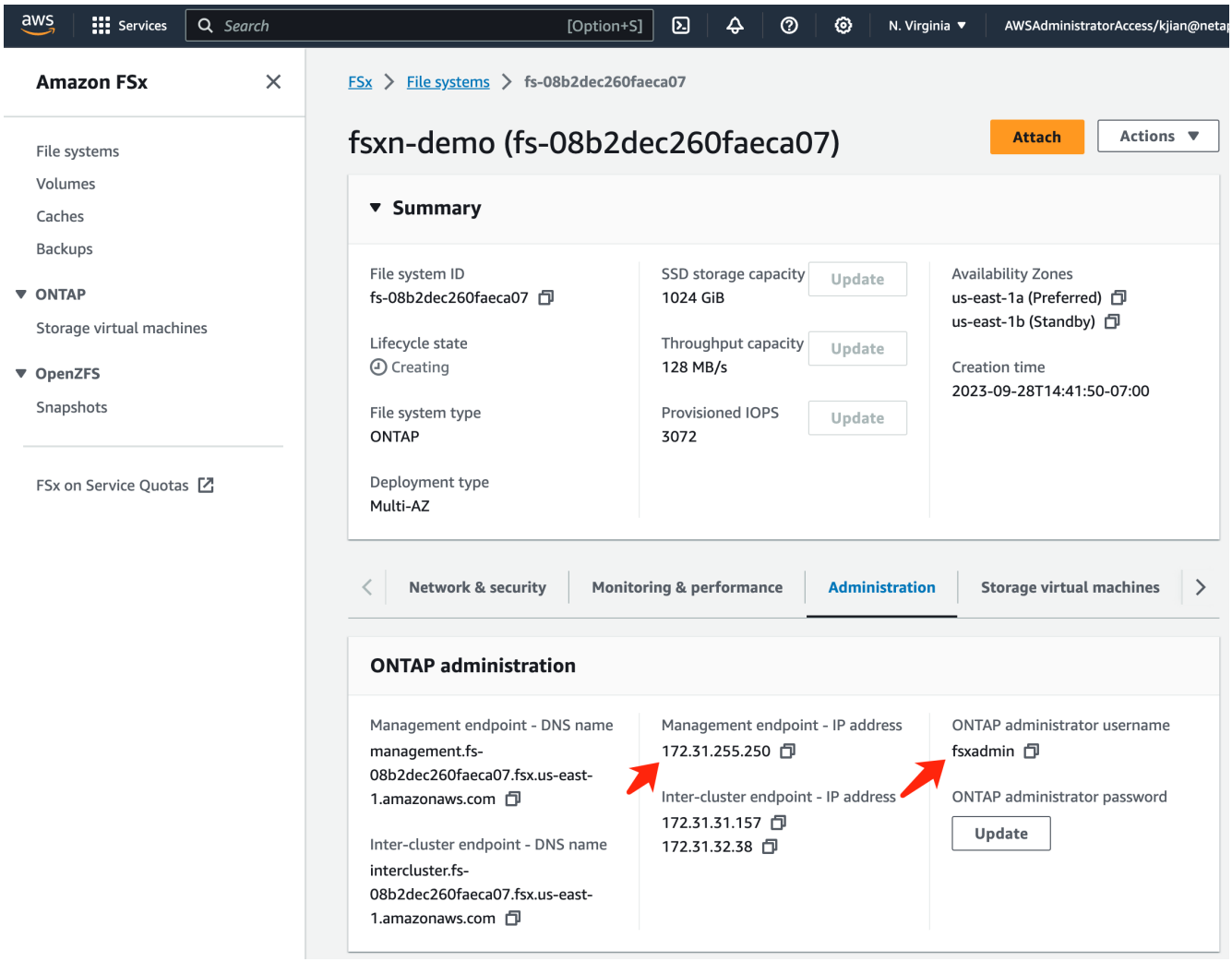
Serverkonfiguration

ONTAP-Konfiguration

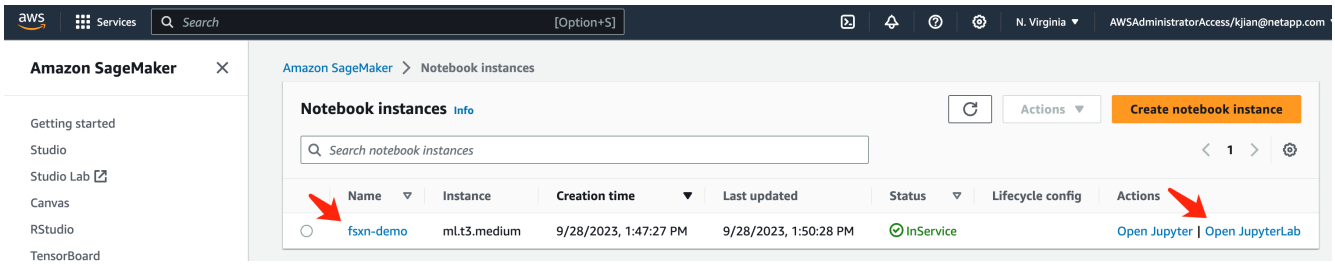
1. Öffnen Sie das erstellte FSX Dateisystem. Bitte stellen Sie sicher, dass der Status **verfügbar** ist.



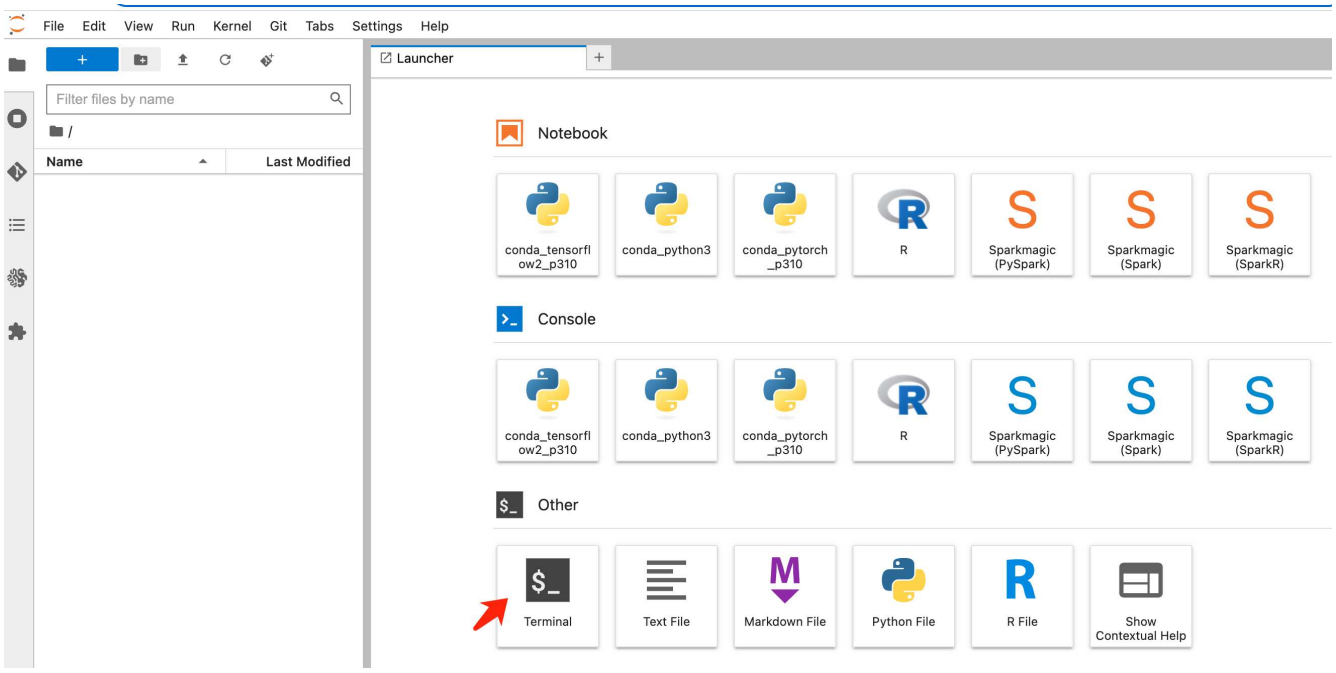
2. Wählen Sie die Registerkarte **Verwaltung** aus und behalten Sie die Optionen **Verwaltungsendpunkt - IP-Adresse** und **ONTAP-Administratorbenutzername** bei.



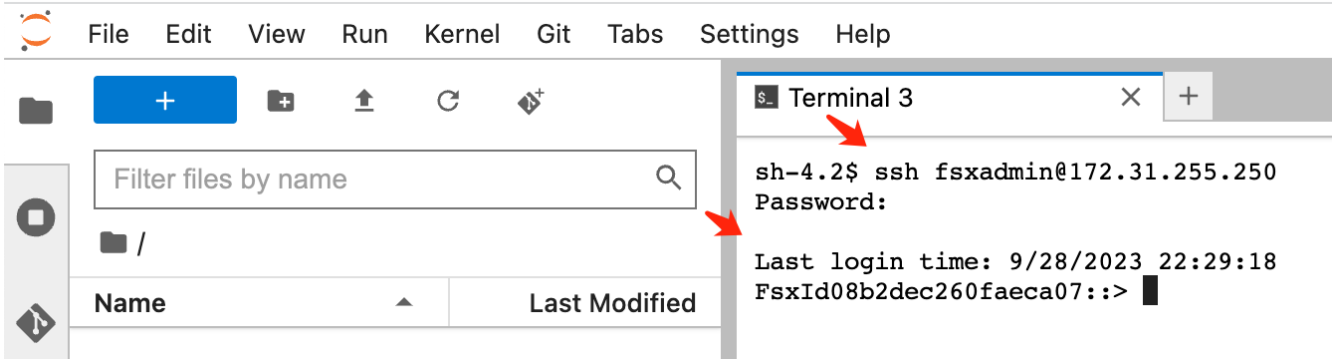
3. Öffnen Sie die erstellte **SageMaker Notebook-Instanz** und klicken Sie auf **JupyterLab öffnen**.



4. Öffnen Sie auf der Seite Jupyter Lab ein neues **Terminal**.



5. Geben Sie den ssh-Befehl ssh <admin user Name>@<ONTAP Server IP> ein, um sich beim FSX ONTAP-Dateisystem anzumelden. (Der Benutzername und die IP-Adresse werden aus Schritt 2 abgerufen) Bitte verwenden Sie das Passwort, das beim Erstellen der **Storage Virtual Machine** verwendet wird.



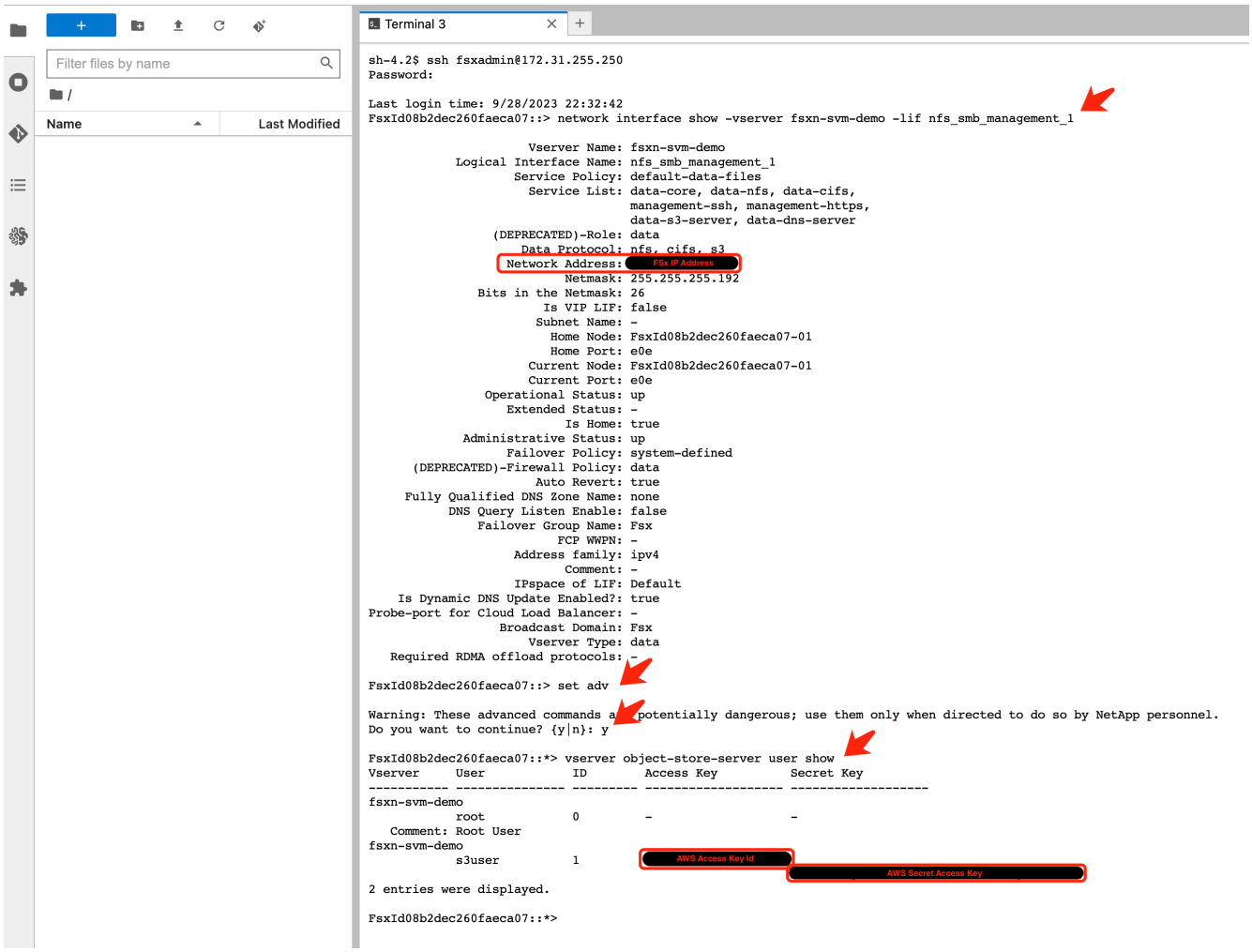
6. Führen Sie die Befehle in der folgenden Reihenfolge aus. Wir verwenden **fsxn-ONTAP** als Namen für den **FSX ONTAP privaten S3-Bucket-Namen**. Bitte verwenden Sie für das Argument **-vserver** den Namen der virtuellen Speichermaschine*.

```
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. Führen Sie die folgenden Befehle aus, um die Endpunkt-IP und die Zugangsdaten für FSX ONTAP Private S3 abzurufen.

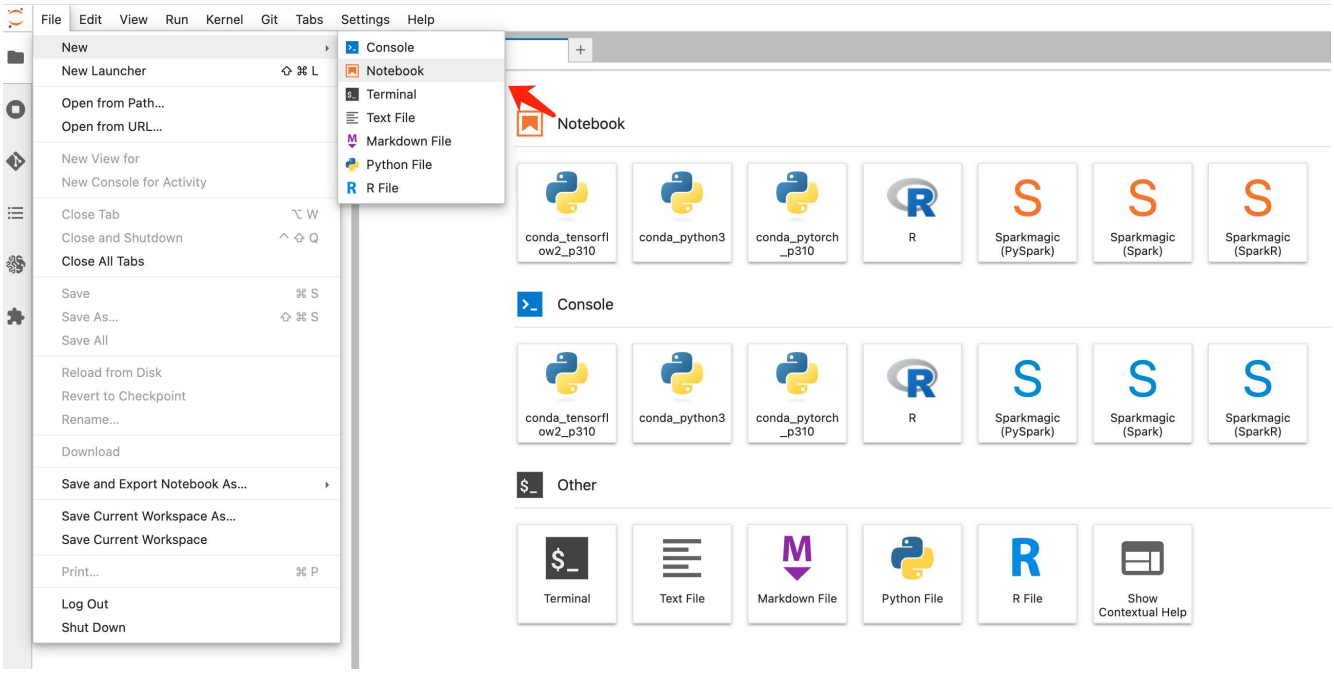
```
network interface show -vserver fsxn-svm-demo -lif nfs_smb_management_1

set adv

vserver object-store-server user show
```

8. Die IP-Adresse des Endpunkts und die Anmeldeinformationen bleiben für die zukünftige Verwendung erhalten.



Client-Konfiguration

1. Erstellen Sie in der SageMaker-Notebook-Instanz ein neues Jupyter-Notebook.



2. Verwenden Sie den unten stehenden Code als Umgehung, um Dateien in FSX ONTAP privaten S3 Bucket hochzuladen. Ein umfangreiches Codebeispiel finden Sie in diesem Notizbuch. "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 FSx ONTAP IP address>'            # Please get
this IP address from FSx ONTAP
# -------- 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 FSx ONTAP 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 FSx ONTAP 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 FSx ONTAP 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']
```

Damit ist die Integration zwischen FSX ONTAP und der SageMaker-Instanz abgeschlossen.

Nützliche Debugging-Checkliste

- Stellen Sie sicher, dass sich die SageMaker-Notebook-Instanz und das FSX ONTAP-Dateisystem in derselben VPC befinden.
- Denken Sie daran, den Befehl **set dev** auf ONTAP auszuführen, um die Berechtigungsebene auf **dev** zu setzen.

FAQ (Stand 27. September 2023)

F: Warum erhalte ich den Fehler "**ein Fehler ist aufgetreten (NotImplementiert) beim Aufruf der CreateMultipartUpload Operation: Der von Ihnen angeforderte s3 Befehl ist nicht implementiert**" beim Hochladen von Dateien auf FSX ONTAP?

A: Als privater S3-Bucket unterstützt FSX ONTAP das Hochladen von Dateien mit bis zu 100 MB. Bei Verwendung des S3-Protokolls werden Dateien mit einer Größe von mehr als 100 MB in 100-MB-Blöcke unterteilt, und die Funktion „CreateMultipartUpload“ wird aufgerufen. Die aktuelle Implementierung von FSX ONTAP private S3 unterstützt diese Funktion jedoch nicht.

F: Warum erhalte ich den Fehler "**ein Fehler ist aufgetreten (AccessDenied) beim Aufruf der PutObject-Operationen: Zugriff verweigert**" beim Hochladen von Dateien auf FSX ONTAP?

A: Um von einer SageMaker-Notebook-Instanz auf den privaten S3-Bucket von FSX ONTAP zuzugreifen, wechseln Sie die AWS-Anmeldeinformationen zu den FSX ONTAP-Anmeldeinformationen. Die Gewährung von Schreibberechtigungen für die Instanz erfordert jedoch eine Problemumgehungslösung, bei der der Bucket gemountet und der Shell-Befehl 'chmod' ausgeführt wird, um die Berechtigungen zu ändern.

F: Wie kann ich den FSX ONTAP privaten S3-Eimer mit anderen SageMaker ML-Diensten integrieren?

A: Leider bietet das SageMaker Services SDK keine Möglichkeit, den Endpunkt für den privaten S3-Bucket anzugeben. Daher ist FSX ONTAP S3 nicht kompatibel mit SageMaker-Diensten wie SageMaker-Datenwächter, SageMaker-Klarstellung, SageMaker-Kleber, SageMaker-Athena, SageMaker-AutoML und anderen.

Teil 2 – Nutzung von AWS Amazon FSX for NetApp ONTAP (FSX ONTAP) als Datenquelle für das Modelltraining in SageMaker

Dieser Artikel ist ein Tutorial zur Verwendung von Amazon FSX for NetApp ONTAP (FSX ONTAP) für das Training von PyTorch-Modellen in SageMaker, speziell für ein Reifenqualitätsklassifizierungsprojekt.

Autor(en):
Jian Jian (Ken), Senior Data & Applied Scientist, NetApp

Einführung

Dieses Tutorial bietet ein praktisches Beispiel für ein Computer-Vision-Klassifizierungsprojekt und bietet praktische Erfahrung beim Aufbau von ML-Modellen, die FSX ONTAP als Datenquelle innerhalb der SageMaker-Umgebung verwenden. Der Schwerpunkt des Projekts liegt auf der Verwendung von PyTorch, einem Deep-Learning-Framework, um die Reifenqualität anhand von Reifenbildern zu klassifizieren. Er betont die Entwicklung von Machine-Learning-Modellen, die FSX ONTAP als Datenquelle in Amazon SageMaker verwenden.

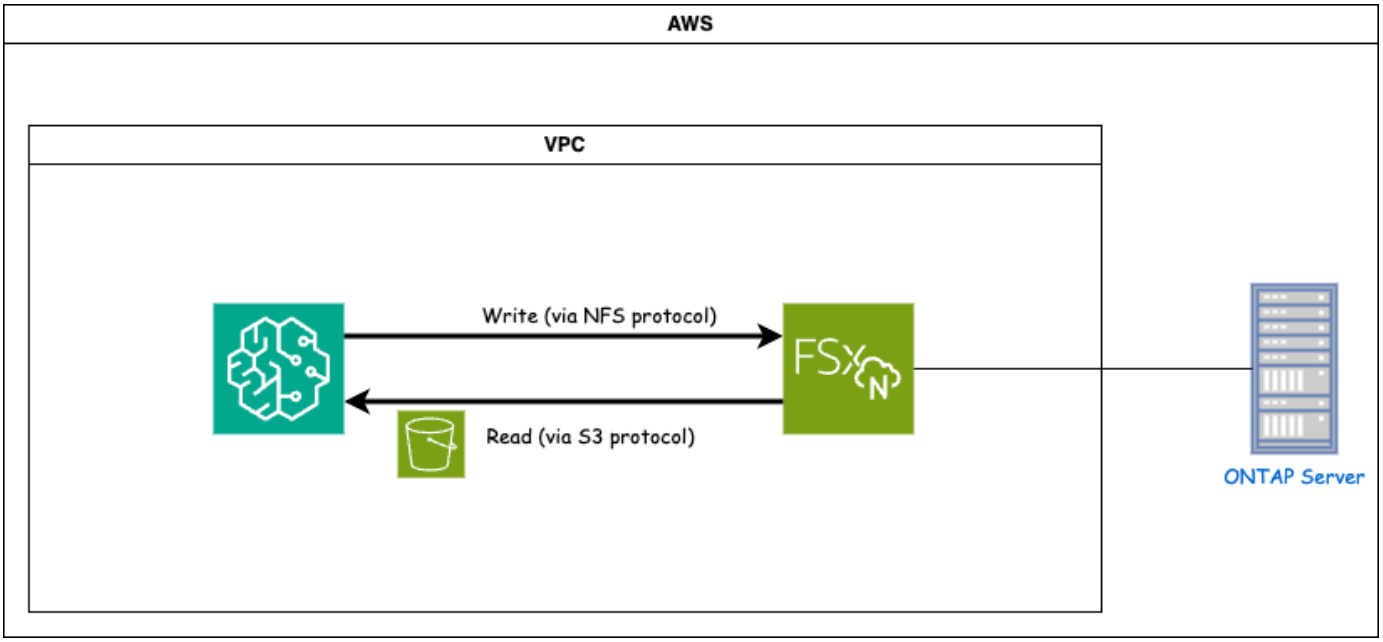
Was ist FSX ONTAP

Amazon FSX ONTAP ist tatsächlich eine von AWS angebotene vollständig gemanagte Storage-Lösung. Es nutzt das ONTAP Filesystem von NetApp, um zuverlässigen und hochperformanten Storage bereitzustellen. Durch die Unterstützung von Protokollen wie NFS, SMB und iSCSI ermöglicht sie einen nahtlosen Zugriff von verschiedenen Computing-Instanzen und Containern. Der Service bietet eine außergewöhnliche Performance,

die schnelle und effiziente Datenvorgänge ermöglicht. Es bietet zudem eine hohe Verfügbarkeit und Langlebigkeit, wodurch sichergestellt wird, dass die Daten zugänglich und geschützt bleiben. Darüber hinaus ist die Speicherkapazität von Amazon FSX ONTAP skalierbar, so dass Sie sie einfach an Ihre Anforderungen anpassen können.

Voraussetzung

Netzwerkumgebung



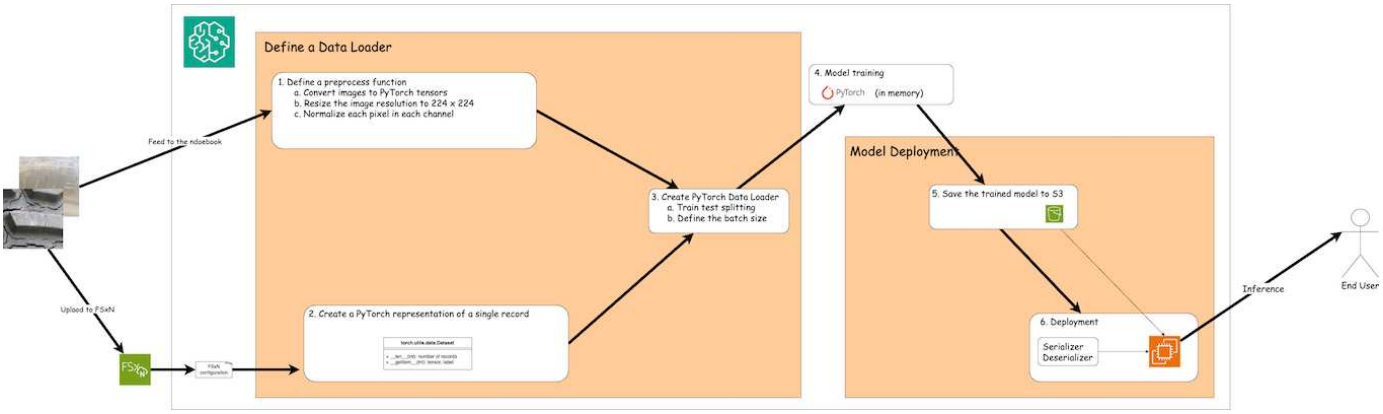
FSX ONTAP (Amazon FSX ONTAP) ist ein AWS-Storage-Service. Sie umfasst ein Filesystem, das auf dem NetApp ONTAP System ausgeführt wird, und eine AWS-gemanagte System Virtual Machine (SVM), die eine Verbindung mit ihr herstellt. Im angegebenen Diagramm befindet sich der von AWS gemanagte NetApp ONTAP-Server außerhalb der VPC. Die SVM dient als Vermittler zwischen SageMaker und dem NetApp ONTAP-System, empfängt Betriebsanforderungen von SageMaker und leitet sie an den zugrunde liegenden Speicher weiter. Für den Zugriff auf FSX ONTAP muss SageMaker in demselben VPC wie die FSX ONTAP-Implementierung platziert werden. Diese Konfiguration gewährleistet die Kommunikation und den Datenzugriff zwischen SageMaker und FSX ONTAP.

Datenzugriff

In realen Szenarien nutzen Data Scientists in der Regel die in FSX ONTAP gespeicherten Daten, um ihre Machine-Learning-Modelle zu erstellen. Da das FSX ONTAP-Dateisystem jedoch nach der Erstellung zunächst leer ist, müssen die Trainingsdaten manuell hochgeladen werden. Dies kann durch die Montage FSX ONTAP als Volumen zu SageMaker erreicht werden. Nachdem das Dateisystem erfolgreich gemountet wurde, können Sie den Datensatz an den gemounteten Speicherort hochladen, sodass er für das Training Ihrer Modelle in der SageMaker-Umgebung verfügbar ist. Dieser Ansatz ermöglicht es Ihnen, die Speicherkapazität und Funktionen von FSX ONTAP zu nutzen, während Sie mit SageMaker für die Modellentwicklung und das Training arbeiten.

Beim Lesen von Daten wird FSX ONTAP als privater S3-Bucket konfiguriert. Weitere Informationen zur Konfiguration finden Sie unter "Teil 1 – Integration von Amazon FSX for NetApp ONTAP (FSX ONTAP) als privaten S3-Bucket in AWS SageMaker"

Integrationsübersicht



Der Workflow, der Trainingsdaten in FSX ONTAP für den Aufbau eines Deep-Learning-Modells in SageMaker verwendet, kann in drei Hauptschritte zusammengefasst werden: Data-Loader-Definition, Modelltraining und Implementierung. Auf hoher Ebene bilden diese Schritte die Grundlage einer MLOPS-Pipeline. Jeder Schritt umfasst jedoch mehrere detaillierte Teilschritte für eine umfassende Implementierung. Diese Teilschritte umfassen verschiedene Aufgaben wie Datenvorverarbeitung, Dataset-Splitting, Modellkonfiguration, Hyperparameter-Tuning, Modellbewertung, und Modellimplementierung. Diese Schritte gewährleisten einen gründlichen und effektiven Prozess für den Aufbau und die Bereitstellung von Deep-Learning-Modellen unter Verwendung von Trainingsdaten von FSX ONTAP in der SageMaker-Umgebung.

Schritt-für-Schritt-Integration

Datenlader

Um ein PyTorch Deep-Learning-Netzwerk mit Daten zu trainieren, wird ein Datenlader erstellt, um die Dateneinspeisung zu erleichtern. Der Data Loader definiert nicht nur die Batch-Größe, sondern bestimmt auch den Ablauf zum Lesen und Vorverarbeiten jedes Datensatzes innerhalb des Stapels. Durch die Konfiguration des Data Loader können wir die Verarbeitung von Daten in Batches übernehmen und so das Training des Deep-Learning-Netzwerks ermöglichen.

Der Datenlader besteht aus 3 Teilen.

Vorverarbeitungsfunktion

```
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]
    )
])
```

Der oben genannte Code-Snippet demonstriert die Definition von Bildvorverarbeitungstransformationen mit dem Modul **torchvision.transforms**. In dieser turktoriellen Funktion wird das vorprozessende Objekt erstellt,

um eine Reihe von Transformationen anzuwenden. Erstens wandelt die **ToTensor()** Transformation das Bild in eine Tensor-Darstellung um. Anschließend passt die Umwandlung **Resize 224,224** das Bild auf eine feste Größe von 24x224 Pixeln an. Schließlich normalisiert die **Normalisieren()** Transformation die Tensor-Werte, indem sie den Mittelwert subtrahiert und durch die Standardabweichung entlang jedes Kanals dividiert. Die für die Normalisierung verwendeten Werte für Mittelwert und Standardabweichung werden häufig in vortrainierten neuronalen Netzmodellen verwendet. Insgesamt bereitet dieser Code die Bilddaten für die weitere Verarbeitung oder Eingabe in ein vortrainiertes Modell vor, indem er sie in einen Tensor konvertiert, die Größe verändert und die Pixelwerte normalisiert.

Die PyTorch-Dataset-Klasse

```
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
```

Diese Klasse bietet Funktionen zum Abrufen der Gesamtzahl der Datensätze im Datensatz und definiert die Methode zum Lesen von Daten für jeden Datensatz. Innerhalb der Funktion ***getitem*** verwendet der Code das Bucket-Objekt boto3 S3, um die Binärdaten aus FSX ONTAP abzurufen. Der Code-Stil für den Zugriff auf Daten aus FSX ONTAP ähnelt dem Lesen von Daten aus Amazon S3. Die nachfolgende Erklärung geht auf den Erstellungsprozess des privaten S3-Objekts **bucket** ein.

FSX ONTAP als privates 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 FSx ONTAP 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 ---
```

Um Daten aus FSX ONTAP in SageMaker zu lesen, wird ein Handler erstellt, der auf den FSX ONTAP-Storage mit dem S3-Protokoll verweist. Dadurch kann FSX ONTAP als privater S3-Bucket behandelt werden. Die Handler-Konfiguration umfasst die Angabe der IP-Adresse der FSX ONTAP SVM, des Bucket-Namens und der erforderlichen Anmeldedaten. Eine umfassende Erklärung zum Erhalt dieser Konfigurationselemente finden Sie im Dokument unter "Teil 1 – Integration von Amazon FSX for NetApp ONTAP (FSX ONTAP) als privaten S3-Bucket in AWS SageMaker".

In dem oben genannten Beispiel wird das Bucket-Objekt verwendet, um das PyTorch-Datensatzobjekt zu instanziieren. Das Datensatzobjekt wird im nachfolgenden Abschnitt näher erläutert.

Der 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)
```

Im angegebenen Beispiel wird eine Batch-Größe von 64 angegeben, was darauf hinweist, dass jeder Batch 64 Datensätze enthält. Durch die Kombination der PyTorch **Datensatz** Klasse, der Vorverarbeitungsfunktion und der Training Batch Größe erhalten wir den Data Loader für das Training. Dieser Daten-Loader erleichtert den Prozess, den Datensatz während der Trainingsphase in Batches zu durchlaufen.

Modelltraining

```
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')
```

Dieser Code implementiert einen standardmäßigen PyTorch-Trainingsprozess. Es definiert ein neuronales Netzmodell mit dem Namen **TireQualityClassifier**, das konvolutionelle Schichten und eine lineare Schicht verwendet, um die Reifenqualität zu klassifizieren. Die Trainingsschleife iteriert Daten-Batches, berechnet den Verlust und aktualisiert die Parameter des Modells mittels Rückverbreitung und Optimierung. Außerdem werden die aktuelle Zeit, die aktuelle Epoche, der Stapel und der Verlust für Überwachungszwecke gedruckt.

Modellbereitstellung

Einsatz

```
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())
```

Der Code speichert das PyTorch-Modell in **Amazon S3**, da SageMaker das Modell für die Bereitstellung in S3 speichern muss. Durch das Hochladen des Modells auf **Amazon S3** wird es für SageMaker zugänglich, was die Bereitstellung und Inferenz auf dem bereitgestellten Modell ermöglicht.

```
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
)
```

Dieser Code erleichtert die Bereitstellung eines PyTorch-Modells auf SageMaker. Es definiert einen benutzerdefinierten Serialisator, **TyreQualitySerializer**, der Eingabedaten als PyTorch-Tensor vorverarbeitet und serialisiert. Die Klasse **TireQualityPredictor** ist ein benutzerdefinierter Prädiktor, der den definierten Serialisator und einen **JSONDeserializer** verwendet. Der Code erstellt außerdem ein **PyTorchModel**-Objekt, um den S3-Standort des Modells, die IAM-Rolle, die Framework-Version und den Eintrittspunkt für die Inferenz

festzulegen. Der Code generiert einen Zeitstempel und erstellt einen Endpunktnamen basierend auf dem Modell und dem Zeitstempel. Schließlich wird das Modell mithilfe der Bereitstellungsmethode bereitgestellt, wobei die Anzahl der Instanzen, der Instanztyp und der Name des generierten Endpunkts angegeben werden. Dadurch kann das PyTorch-Modell auf SageMaker bereitgestellt und für Inferenz zugänglich sein.

Inferenz

```
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)
```

Dies ist das Beispiel für die Verwendung des implementierten Endpunkts zur Inferenz.

Teil 3 – Aufbau Einer vereinfachten MLOPS-Pipeline (CI/CT/CD)

Dieser Artikel enthält einen Leitfaden zum Aufbau einer MLOps-Pipeline mit AWS-Services. Der Schwerpunkt liegt auf automatisiertem Modellumlernen, Implementierung und Kostenoptimierung.

```
*Autor(en):*
Jian Jian (Ken), Senior Data & Applied Scientist, NetApp



== Einführung

In diesem Tutorial erfahren Sie, wie Sie verschiedene AWS-Services für die
Erstellung einer einfachen MLOPS-Pipeline nutzen können, die Continuous
Integration (CI), Continuous Training (CT) und Continuous Deployment (CD)
umfasst. Im Gegensatz zu herkömmlichen DevOps-Pipelines erfordert MLOps
beim Abschluss des Betriebszyklus zusätzliche Überlegungen. Wenn Sie
dieses Tutorial befolgen, erhalten Sie Einblicke in die Integration von CT
in die MLOPS-Schleife, was ein kontinuierliches Training Ihrer Modelle und
eine nahtlose Bereitstellung für die Inferenz ermöglicht. Das Tutorial
führt Sie durch die Nutzung von AWS-Services zur Einrichtung dieser End-
to-End-MLOPS-Pipeline.



== Manifest
```

```
|===
| Funktionalität | Name | Kommentar


| Datenspeicher | AWS FSX ONTAP | Siehe
xref:{relative_path}./mlops_fsxn_s3_integration.html["Teil 1 – Integration
von Amazon FSX for NetApp ONTAP (FSX ONTAP) als privaten S3-Bucket in AWS
SageMaker"].


| Data Science IDE | AWS SageMaker | Dieses Tutorial basiert auf dem
Jupyter
Notebooklink:./mlops_fsxn_sagemaker_integration_training.html["Teil 2 –
Nutzung von Amazon FSX for NetApp ONTAP (FSX ONTAP) als Datenquelle für
Modelltraining in SageMaker"], das in vorgestellt wird.


| Funktion zum Auslösen der MLOPS-Pipeline | AWS Lambda-Funktion | -


| Cron-Job-Trigger | AWS EventBridge | -


| Deep-Learning-Framework | PyTorch | -


| AWS Python SDK | Boto3 | -


| Programmiersprache | Python | v3.10
|===


== Voraussetzung

* Ein vorkonfiguriertes FSX ONTAP-Dateisystem. In diesem Tutorial werden
die in FSX ONTAP gespeicherten Daten für den Trainingsprozess verwendet.
* Eine * SageMaker Notebook-Instanz*, die so konfiguriert ist, dass sie
dieselbe VPC wie das oben erwähnte FSX ONTAP-Dateisystem gemeinsam
verwendet.
* Bevor Sie die *AWS Lambda-Funktion* aktivieren, stellen Sie sicher, dass
die *SageMaker Notebook-Instanz* den Status *angehalten* hat.
* Der Instanztyp *ml.g4dn.xlarge* wird benötigt, um die GPU-Beschleunigung
zu nutzen, die für die Berechnung tiefer neuronaler Netzwerke notwendig
ist.
```

== Der Netapp Architektur Sind

image:mlops_fsxn_cictcd_0.png["Der Netapp Architektur Sind"]

Diese MLOPS-Pipeline ist eine praktische Implementierung, die mithilfe eines Cron-Jobs eine serverlose Funktion auslöst, die wiederum einen AWS-Service ausführt, der mit einer Lifecycle-Callback-Funktion registriert ist. Die *AWS EventBridge* fungiert als Cron-Job. Es ruft regelmäßig eine *AWS Lambda-Funktion* auf, die für die Umschulung und Neuimplementierung des Modells verantwortlich ist. Bei diesem Vorgang wird die *AWS SageMaker Notebook*-Instanz hochgefahren, um die erforderlichen Aufgaben auszuführen.

== Schritt-für-Schritt-Konfiguration

=== Lebenszykluskonfigurationen

Um die Lifecycle-Callback-Funktion für die AWS SageMaker Notebook-Instanz zu konfigurieren, würden Sie *Lifecycle-Konfigurationen* verwenden. Mit diesem Service können Sie die erforderlichen Aktionen definieren, die beim Starten der Notizbuchinstanz ausgeführt werden müssen. Konkret kann ein Shell-Skript innerhalb der *Lifecycle-Konfigurationen* implementiert werden, um die Notebook-Instanz nach Abschluss der Trainings- und Bereitstellungsprozesse automatisch herunterzufahren. Dies ist eine erforderliche Konfiguration, da die Kosten eine der wichtigsten Überlegungen bei MLOPS sind.

Es ist wichtig zu beachten, dass die Konfiguration für *Lifecycle-Konfigurationen* im Voraus eingerichtet werden muss. Daher wird empfohlen, die Konfiguration dieses Aspekts zu priorisieren, bevor mit dem Setup der anderen MLOPS-Pipeline fortgefahren wird.

. Um eine Lifecycle-Konfiguration einzurichten, öffnen Sie das Fenster *SageMaker* und navigieren Sie zu *Lifecycle-Konfigurationen* unter dem Abschnitt *Admin-Konfigurationen*.
+
image:mlops_fsxn_cictcd_1.png["SageMaker-Bedienfeld"]

. Wählen Sie die Registerkarte *Notebook-Instanz* aus und klicken Sie auf die Schaltfläche *Konfiguration erstellen*
+

```
image:mlops_fsxn_cictcd_2.png["Begrüßungsseite zur
Lebenszykluskonfiguration"]

. Fügen Sie den unten stehenden Code in den Eingabebereich ein.
+
[source, bash]
```

#!/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

```
. Dieses Skript führt das Jupyter Notebook aus, das das erneute Training
und die erneute Bereitstellung des Modells für die Inferenz übernimmt.
Nach Abschluss der Ausführung wird das Notebook automatisch innerhalb von
5 Minuten heruntergefahren. Weitere Informationen zur Problemstellung und
zur Codeimplementierung finden Sie unter
xref:{relative_path}./mlops_fsxn_sagemaker_integration_training.html["Teil
2 – Nutzung von Amazon FSX for NetApp ONTAP (FSX ONTAP) als Datenquelle
für Modelltraining in SageMaker"].
+
image:mlops_fsxn_cictcd_3.png["Lebenszyklukonfiguration erstellen"]

. Navigieren Sie nach der Erstellung zu Notizbuchinstanzen, wählen Sie die
Zielinstanz aus, und klicken Sie im Dropdown-Menü Aktionen auf
*Einstellungen aktualisieren*.
+
image:mlops_fsxn_cictcd_4.png["Dropdown-Menü „Einstellungen
```

```
aktualisieren“"]

. Wählen Sie die erstellte *Lifecycle-Konfiguration* aus und klicken Sie
auf *Notebook-Instanz aktualisieren*.
+
image:mlops_fsxn_cictcd_5.png["Lebenszykluskonfiguration des Notebooks
aktualisieren"]




=== AWS Lambda serverlose Funktion

Wie bereits erwähnt, ist die *AWS Lambda-Funktion* für die Einrichtung der
*AWS SageMaker Notebook-Instanz* zuständig.

. Um eine *AWS Lambda-Funktion* zu erstellen, navigieren Sie zum
entsprechenden Panel, wechseln Sie zum Reiter *Funktionen* und klicken Sie
auf *Create Function*.
+
image:mlops_fsxn_cictcd_6.png["Willkommensseite der AWS Lambda-Funktion"]

. Bitte legen Sie alle erforderlichen Einträge auf der Seite ab und denken
Sie daran, die Runtime auf *Python 3.10* umzuschalten.
+
image:mlops_fsxn_cictcd_7.png["Eine AWS Lambda-Funktion erstellen"]

. Bitte überprüfen Sie, ob die vorgesehene Rolle die erforderliche
Berechtigung hat *AmazonSageMakerFullAccess* und klicken Sie auf den
Button *Funktion erstellen*.
+
image:mlops_fsxn_cictcd_8.png["Wählen Sie die Ausführungsrolle aus"]

. Wählen Sie die erstellte Lambda-Funktion aus. Kopieren Sie auf der
Registerkarte Code den folgenden Code, und fügen Sie ihn in den
Textbereich ein. Dieser Code startet die Notebook-Instanz mit dem Namen
*fsxn-ontap*.
+
[source, python]
```

```
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}'
}

```
. Klicken Sie auf die Schaltfläche *deploy*, um diese Codeänderung
anzuwenden.
+
image:mlops_fsxn_cictcd_9.png["Einsatz"]

. Um anzugeben, wie diese AWS Lambda-Funktion ausgelöst werden soll,
klicken Sie auf die Schaltfläche Add Trigger.
+
image:mlops_fsxn_cictcd_10.png["AWS Funktions-Trigger hinzufügen"]

. Wählen Sie EventBridge aus dem Dropdown-Menü aus, und klicken Sie dann
auf das Optionsfeld Neue Regel erstellen. Geben Sie im Feld Ausdruck
Zeitplan ein `rate(1 day)`, Und klicken Sie auf die Schaltfläche
Hinzufügen, um diese neue Cron-Job-Regel auf die AWS Lambda-Funktion zu
erstellen und anzuwenden.
+
image:mlops_fsxn_cictcd_11.png["Auslöser fertig stellen"]



Nach Abschluss der Konfiguration in zwei Schritten wird die *AWS Lambda-
Funktion* täglich das *SageMaker-Notebook* starten, Modellumschulungen mit
den Daten aus dem *FSX ONTAP*-Repository durchführen, das aktualisierte
Modell in der Produktionsumgebung neu bereitstellen und die *SageMaker-
Notebook-Instanz* automatisch herunterfahren, um die Kosten zu optimieren.
Damit bleibt das Modell auf dem neuesten Stand.

Damit ist das Tutorial zur Entwicklung einer MLOPS-Pipeline abgeschlossen.



:leveloffset: -1


:leveloffset: -1

<<<
*Copyright-Informationen*


```