FSx ONTAP für MLOps

NetApp artificial intelligence solutions

NetApp
February 12, 2026

Inhalt

FSx ONTAP für MLOps

Amazon FSx for NetApp ONTAP (FSx ONTAP) für MLOps

Dieser Abschnitt befasst sich mit der praktischen Anwendung der Entwicklung von KI-Infrastrukturen und bietet eine umfassende Anleitung zum Erstellen einer MLOps-Pipeline mit FSx ONTAP. Es umfasst drei umfassende Beispiele und führt Sie durch die Erfüllung Ihrer MLOps-Anforderungen mithilfe dieser leistungsstarken Datenverwaltungsplattform.

Diese Artikel konzentrieren sich auf:

1. "Teil 1 – Integration von Amazon FSx for NetApp ONTAP (FSx ONTAP) als privater S3-Bucket in AWS SageMaker"
2. "Teil 2 – Nutzung von Amazon FSx for NetApp ONTAP (FSx ONTAP) als Datenquelle für das Modelltraining in SageMaker"
3. "Teil 3 – Erstellen einer vereinfachten MLOps-Pipeline (CI/CT/CD)"

Am Ende dieses Abschnitts haben Sie ein solides Verständnis dafür erlangt, wie Sie FSx ONTAP zur Optimierung von MLOps-Prozessen verwenden können.

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

Dieser Abschnitt enthält eine Anleitung zum Konfigurieren von FSx ONTAP als privater S3-Bucket mit AWS SageMaker.

Einführung

Am Beispiel von SageMaker bietet diese Seite eine Anleitung zum Konfigurieren von FSx ONTAP als privatem S3-Bucket.

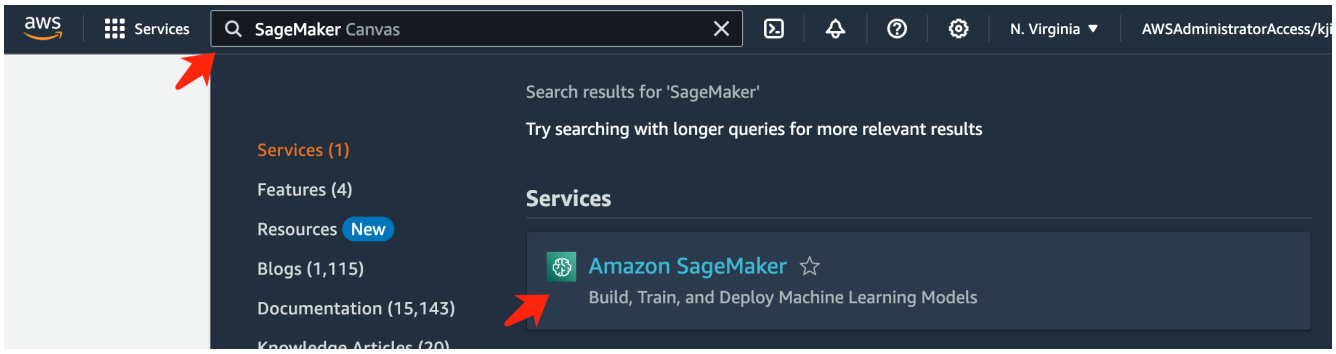
Weitere Informationen zu FSx ONTAP finden Sie in dieser Präsentation ("Videolink")

User Guide

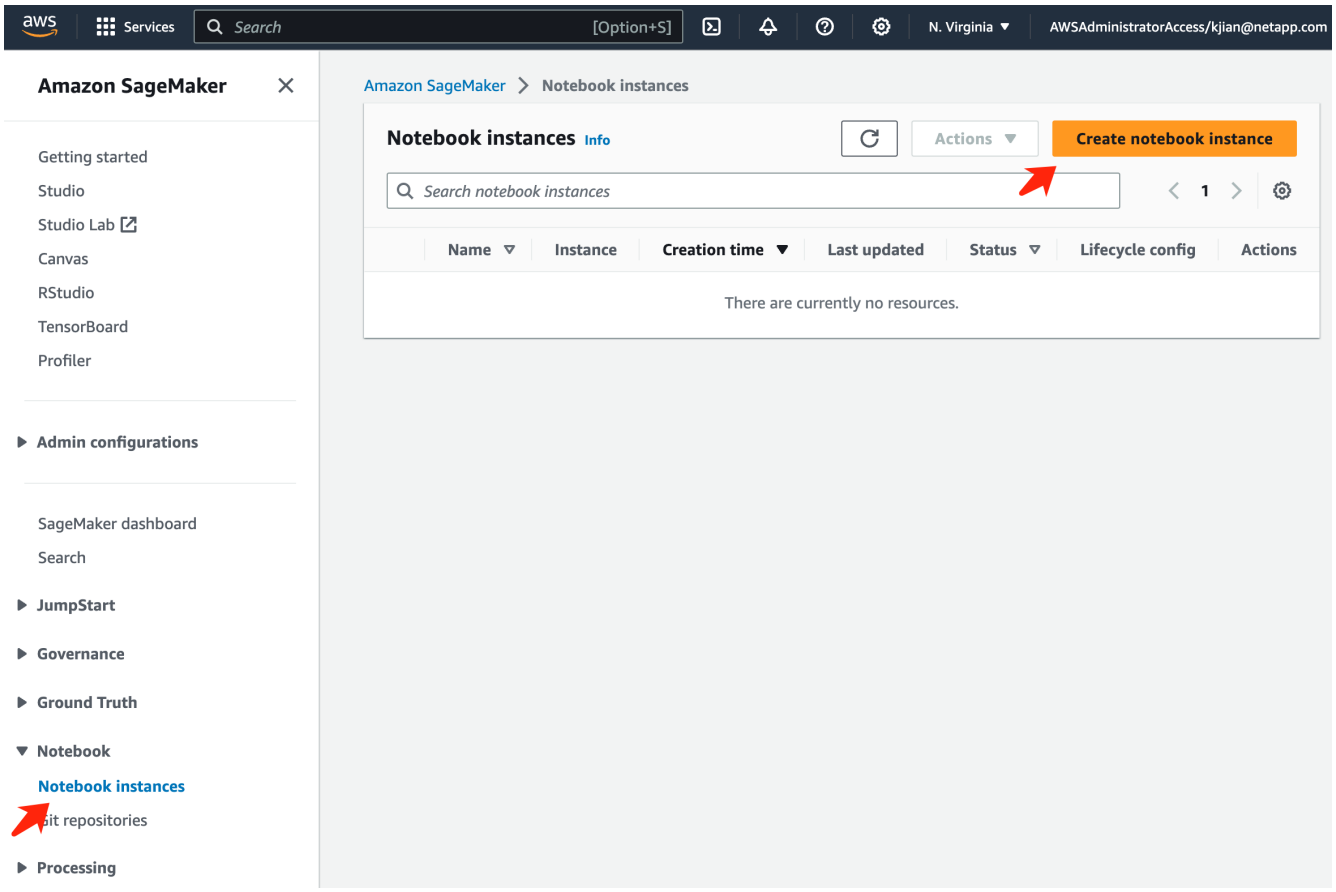
Servererstellung

Erstellen einer SageMaker-Notebook-Instanz

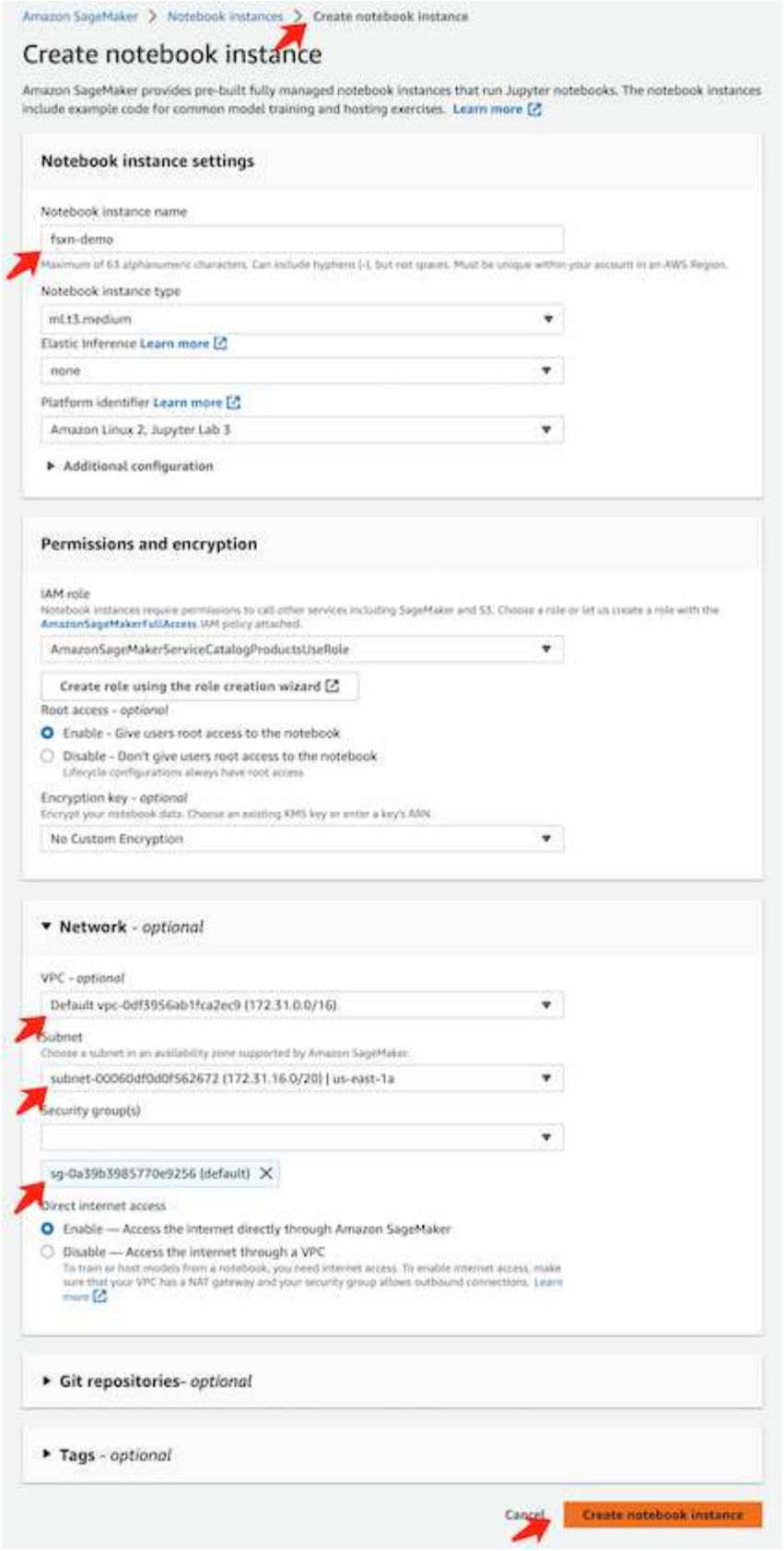
1. Öffnen Sie die AWS-Konsole. Suchen Sie im Suchfeld nach SageMaker und klicken Sie auf den Dienst **Amazon SageMaker**.



2. Öffnen Sie die **Notebook-Instanzen** unter der Registerkarte „Notebook“ und klicken Sie auf die orangefarbene Schaltfläche „Notebook-Instanz erstellen“*.

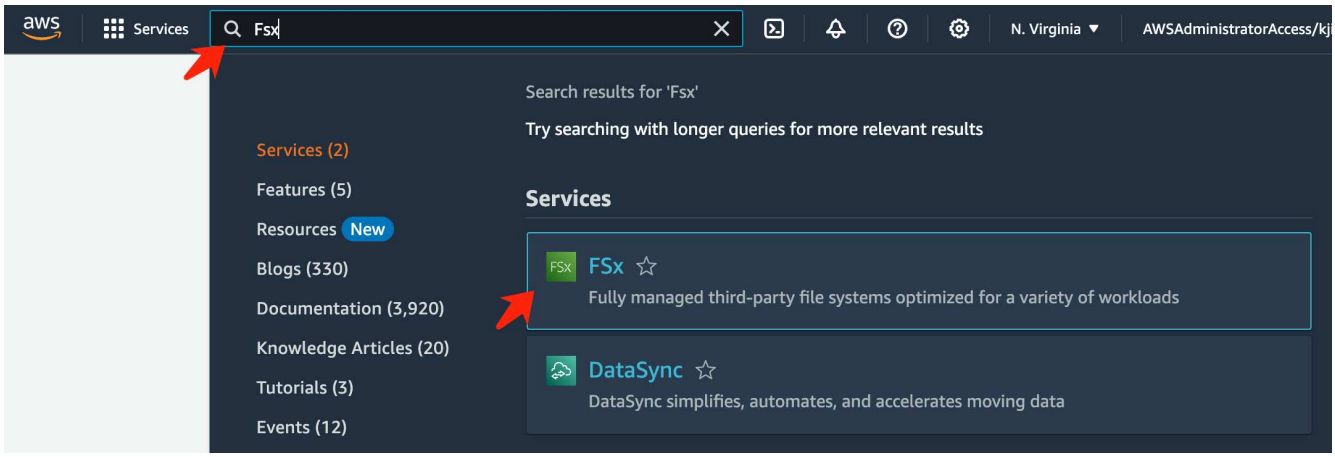


3. Geben Sie auf der Erstellungsseite den **Namen der Notebook-Instanz** ein. Erweitern Sie das **Netzwerk** -Bedienfeld. Belassen Sie die anderen Einträge auf den Standardwerten und wählen Sie eine **VPC**, ein **Subnetz** und **Sicherheitsgruppe(n)** aus. (Diese **VPC** und dieses **Subnetz** werden später zum Erstellen des FSx ONTAP Dateisystems verwendet.) Klicken Sie unten rechts auf die orangefarbene Schaltfläche **Notebook-Instanz erstellen**.

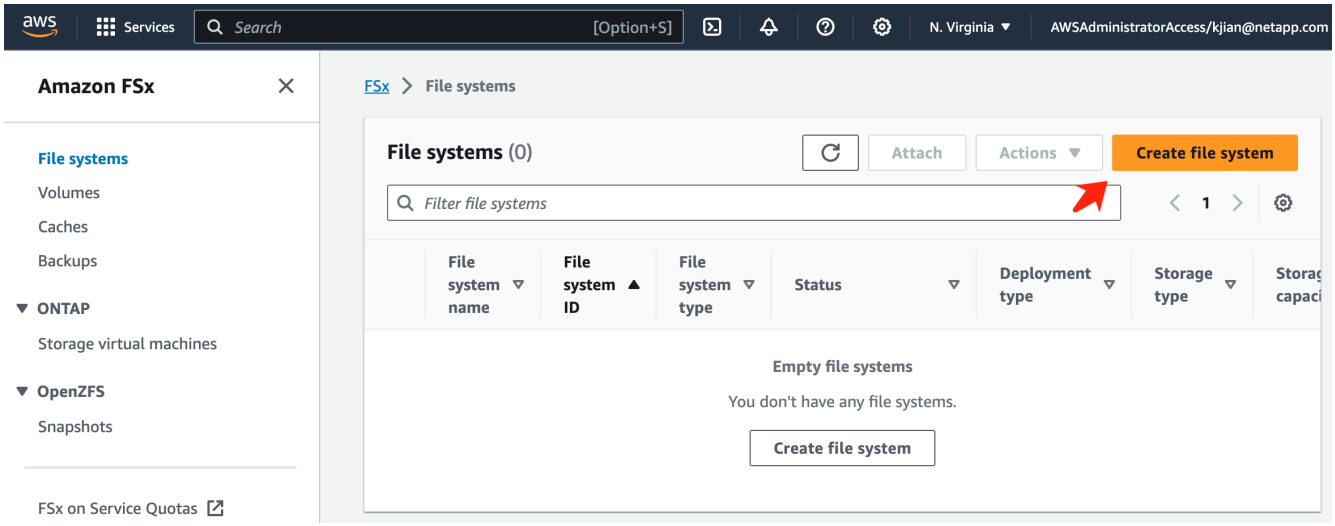


Erstellen Sie ein FSx ONTAP Dateisystem

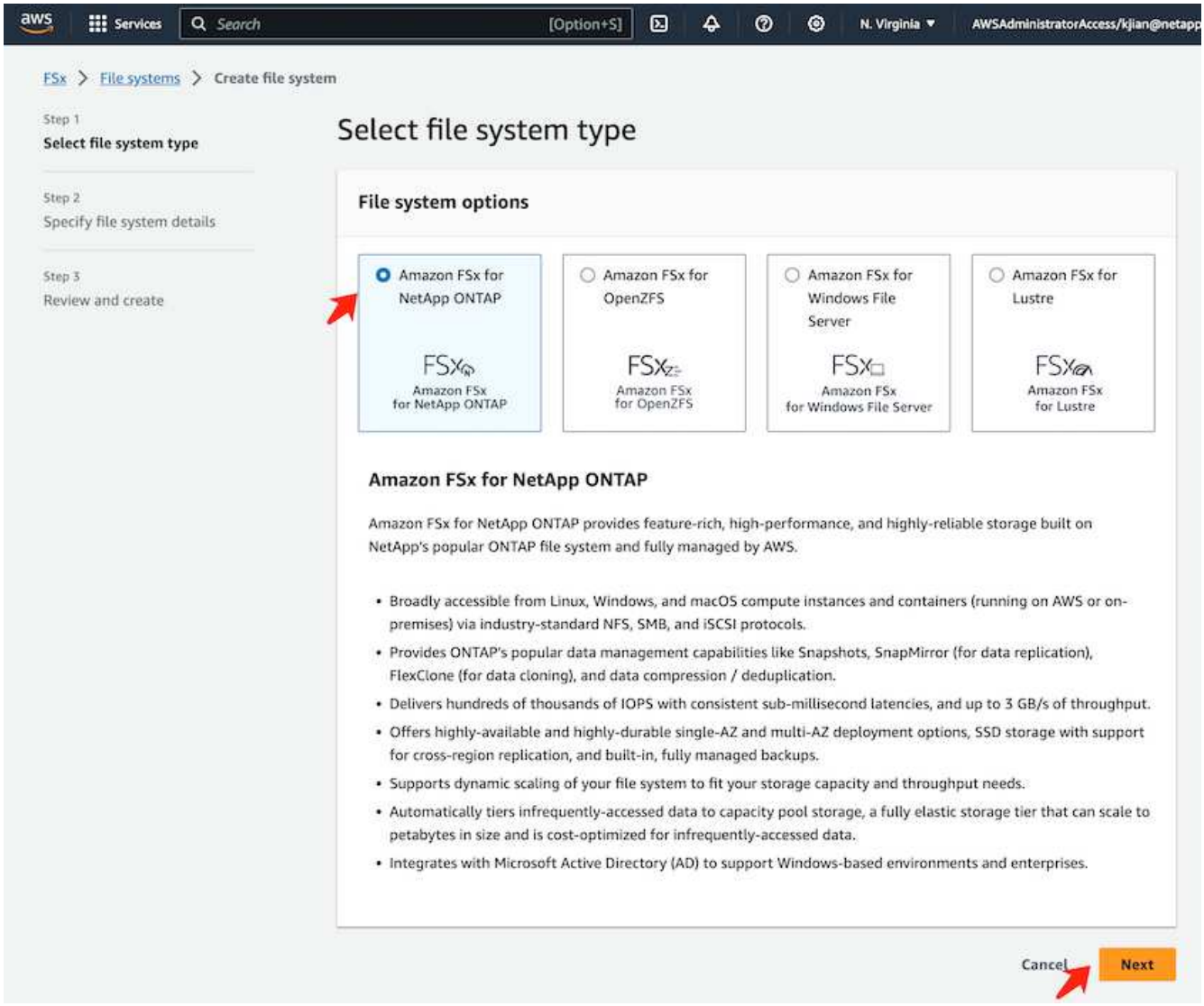
1. Öffnen Sie die AWS-Konsole. Suchen Sie im Suchfeld nach Fsx und klicken Sie auf den Dienst **FSx**.



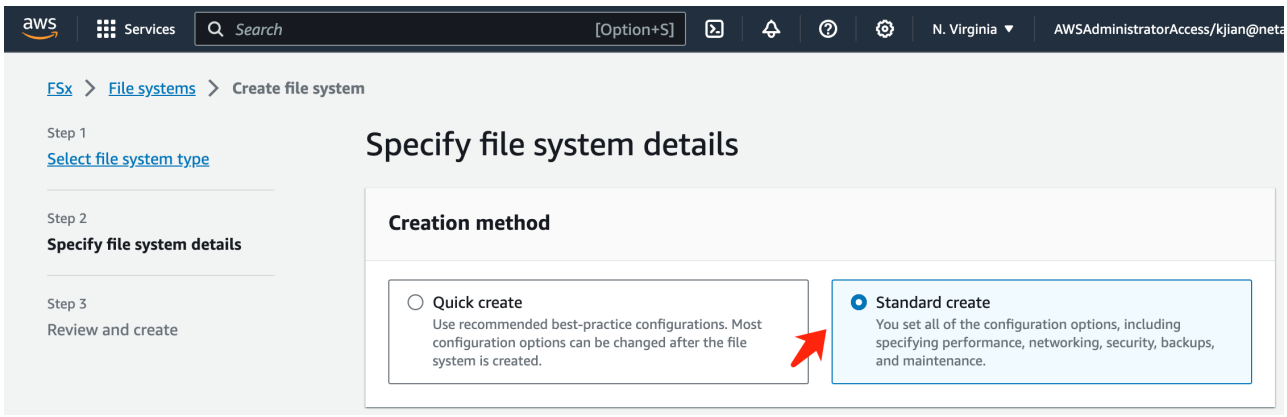
2. Klicken Sie auf **Dateisystem erstellen**.



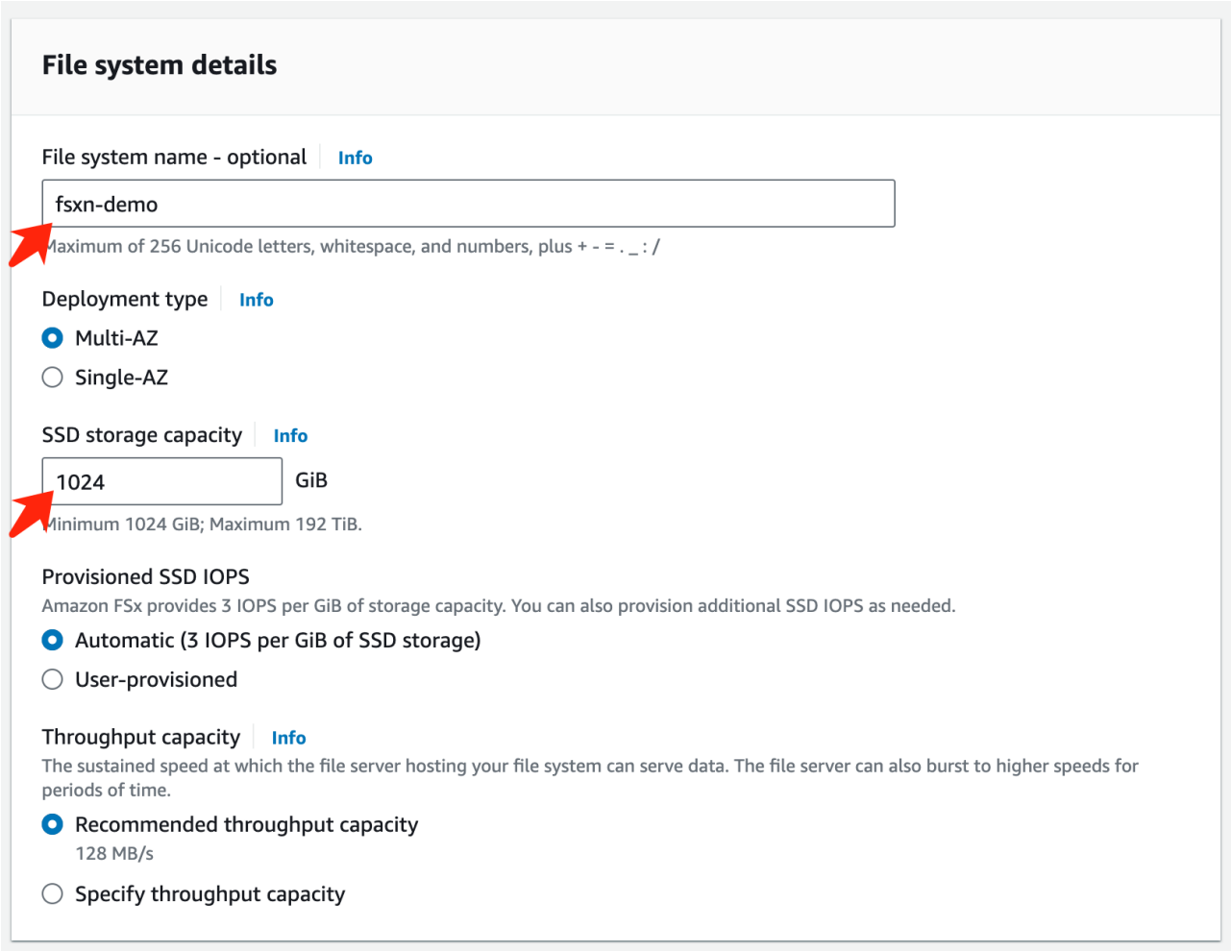
3. Wählen Sie die erste Karte **FSx ONTAP** aus und klicken Sie auf **Weiter**.



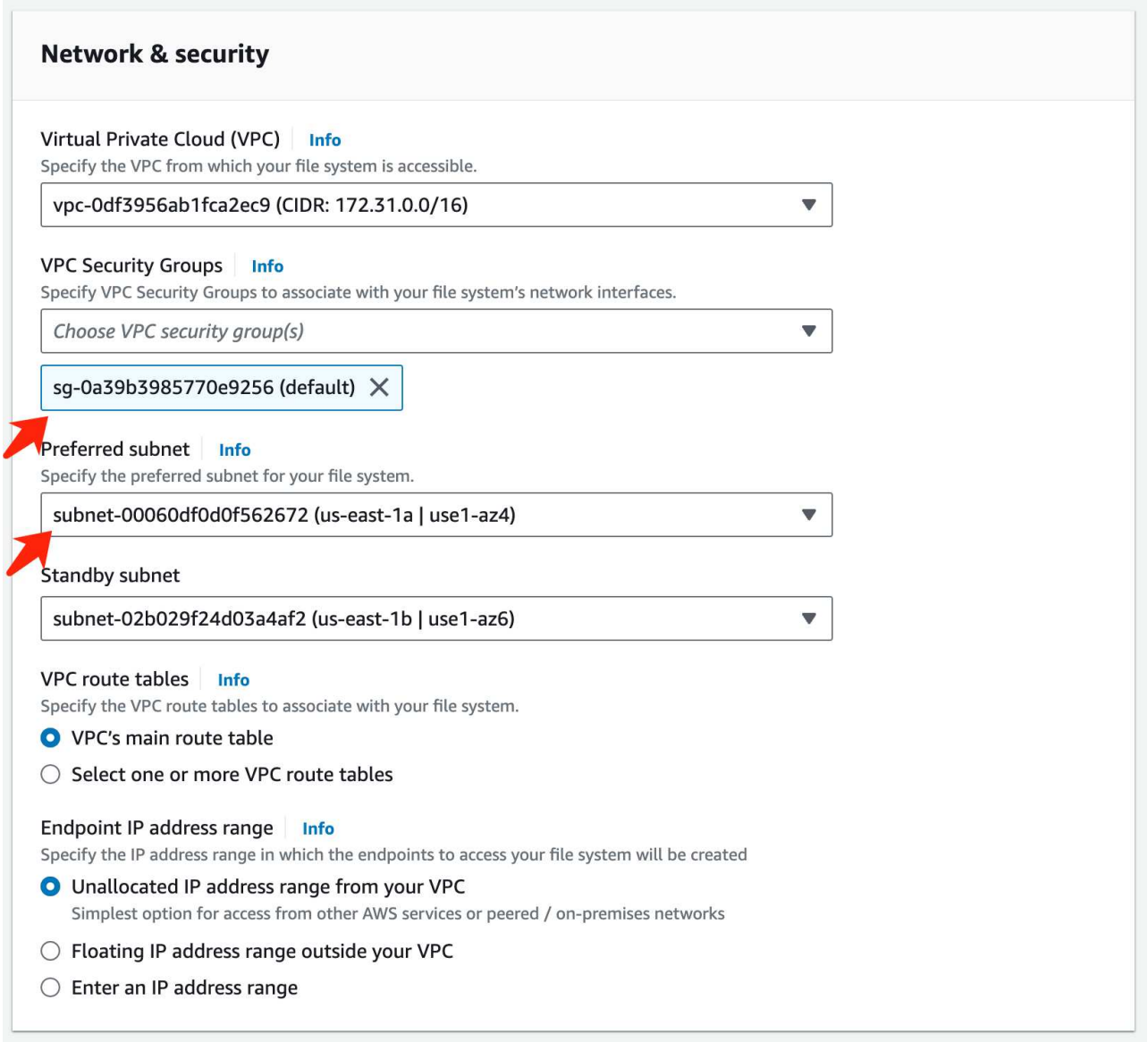
4. Auf der Detailkonfigurationsseite.
   a. Wählen Sie die Option **Standard erstellen**.



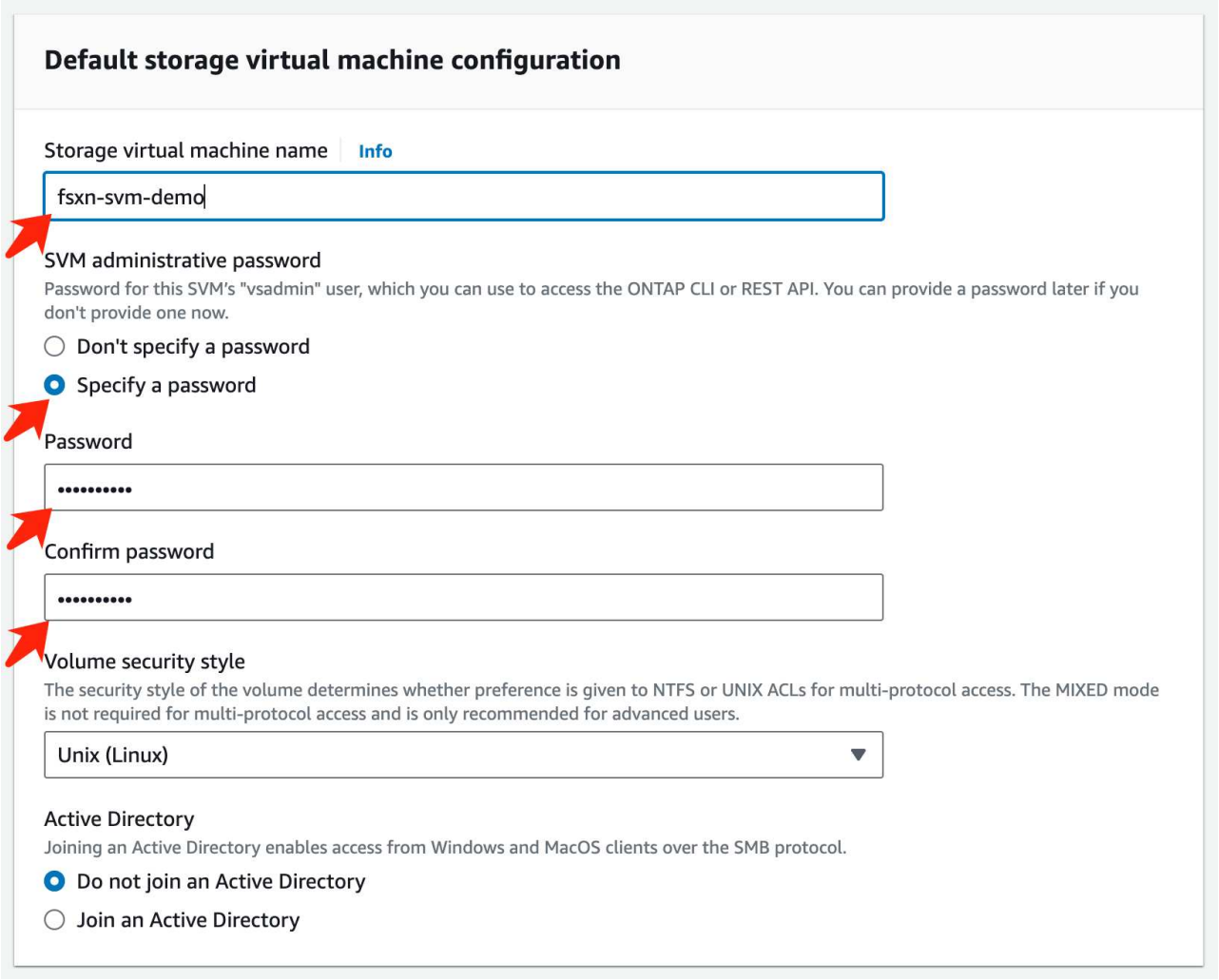
   b. Geben Sie den **Dateisystemnamen** und die **SSD-Speicherkapazität** ein.



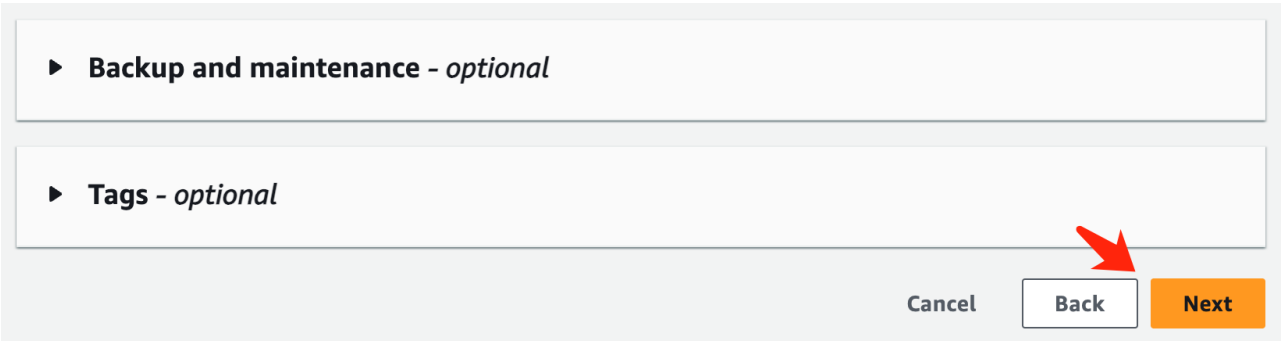
c. Stellen Sie sicher, dass Sie dasselbe **VPC** und **Subnetz** wie für die **SageMaker Notebook**-Instanz verwenden.



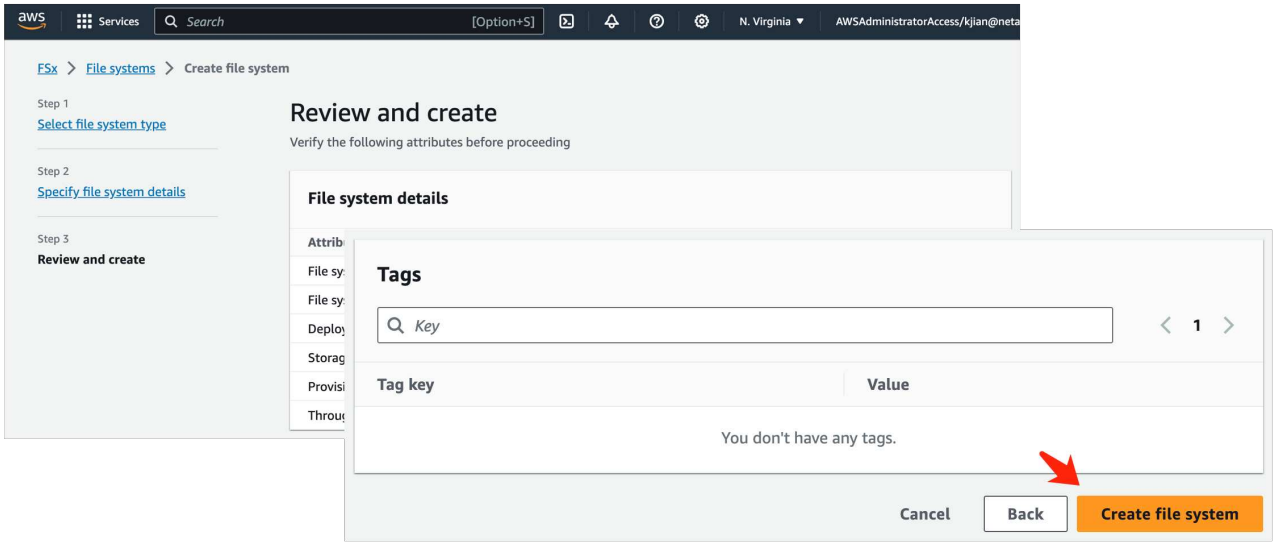
d. Geben Sie den Namen der **Storage Virtual Machine** ein und **Geben Sie ein Kennwort an** für Ihre SVM (Storage Virtual Machine).



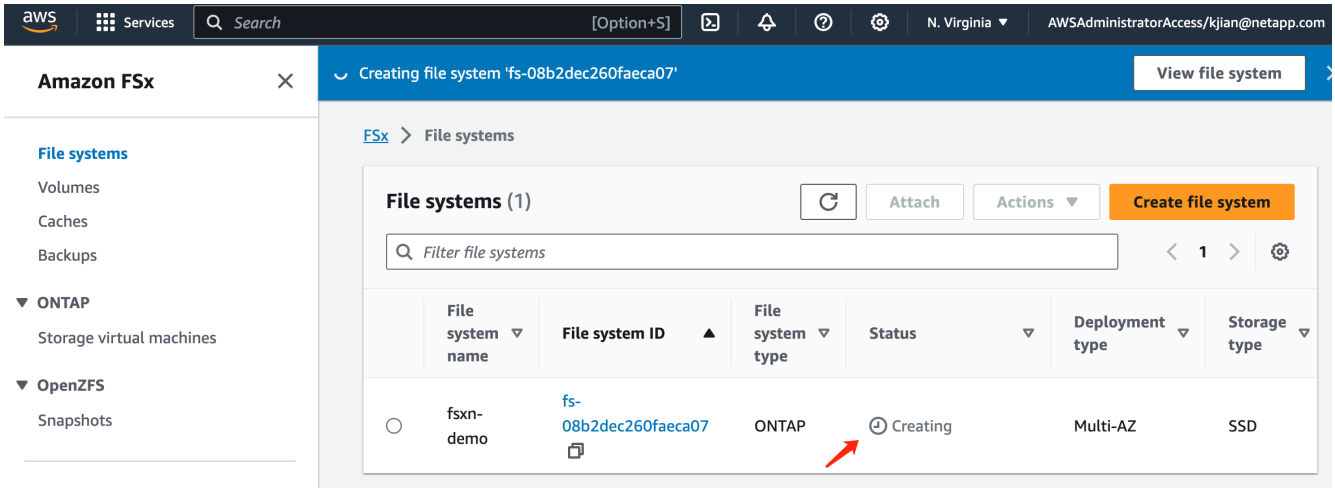
e. Belassen Sie die anderen Einträge auf Standard und klicken Sie unten rechts auf die orangefarbene Schaltfläche **Weiter**.



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



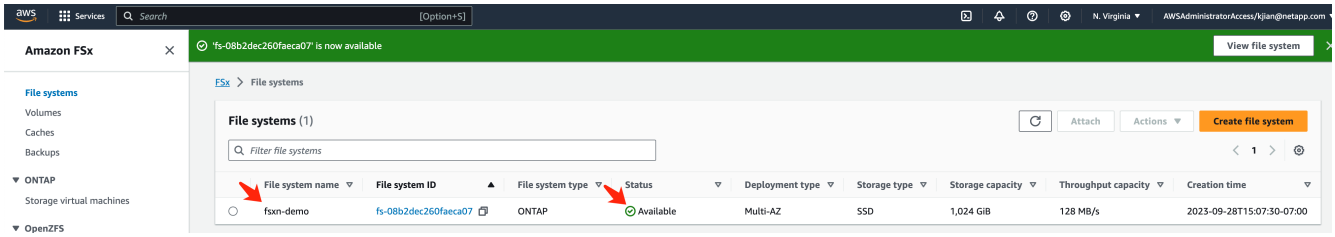
5. Das Hochfahren des FSx-Dateisystems kann etwa **20–40 Minuten** dauern.



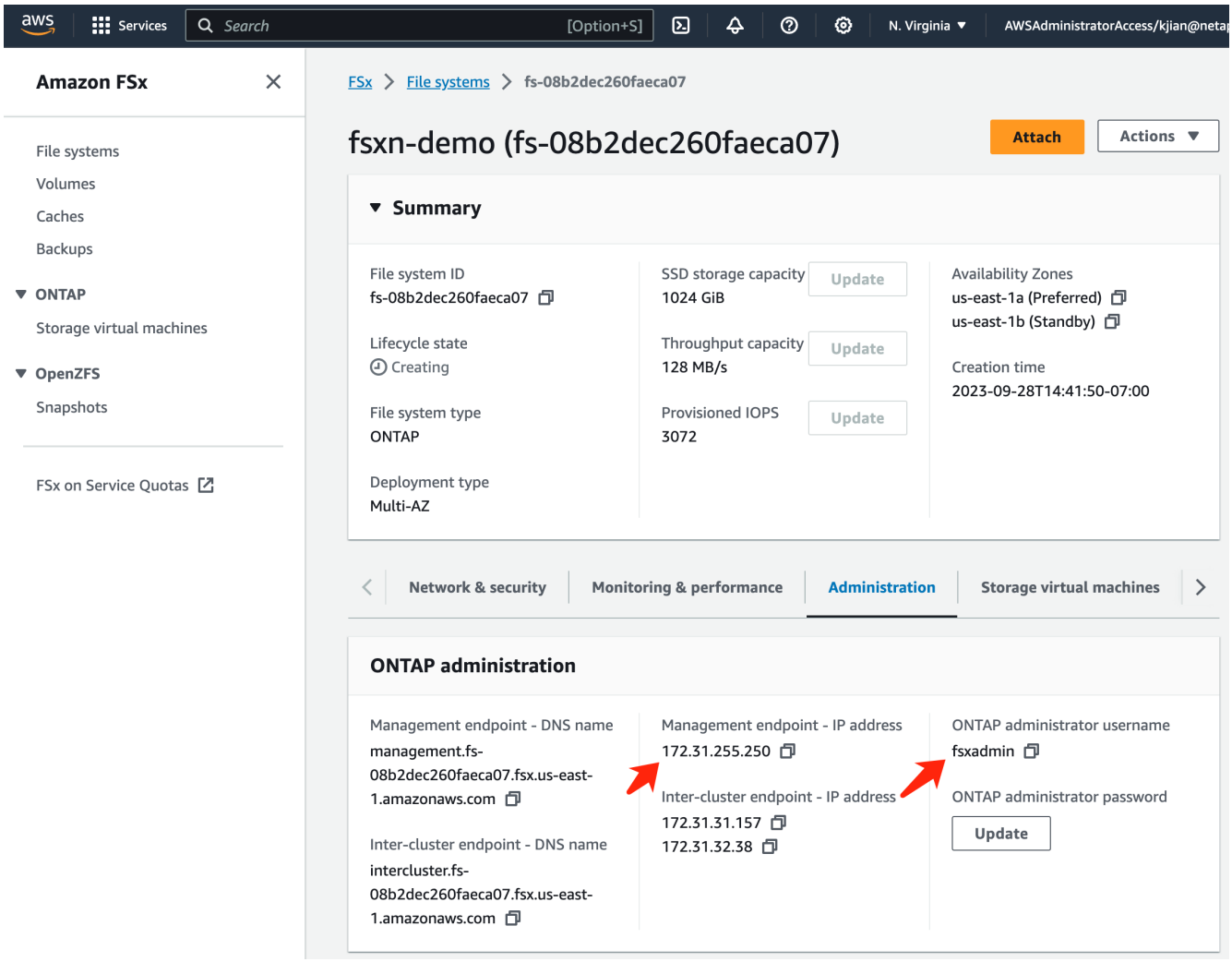
Serverkonfiguration

ONTAP -Konfiguration

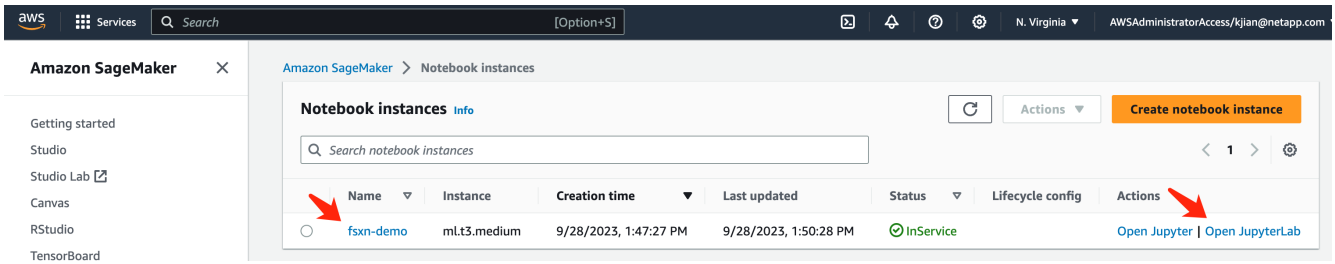
1. Öffnen Sie das erstellte FSx-Dateisystem. Bitte stellen Sie sicher, dass der Status **Verfügbar** ist.



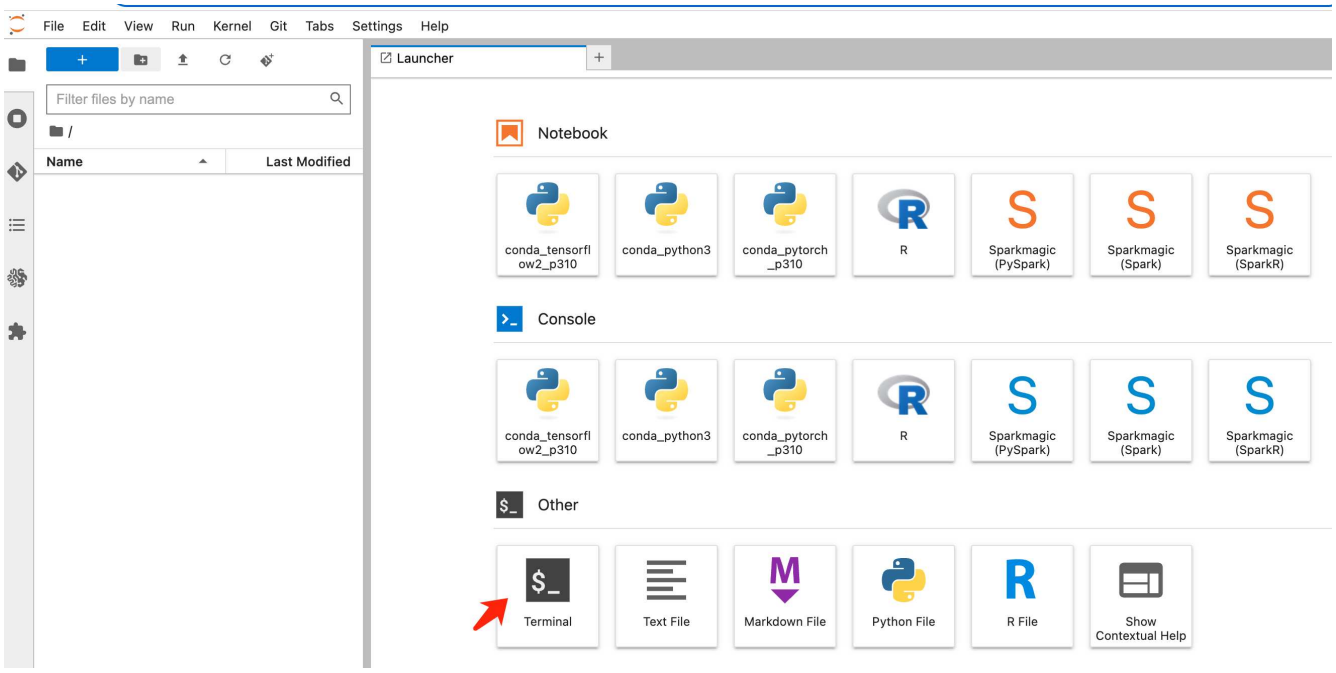
2. Wählen Sie die Registerkarte **Administration** und behalten Sie die **IP-Adresse des Verwaltungsendpunkts** und den * ONTAP -Administratorbenutzernamen* bei.



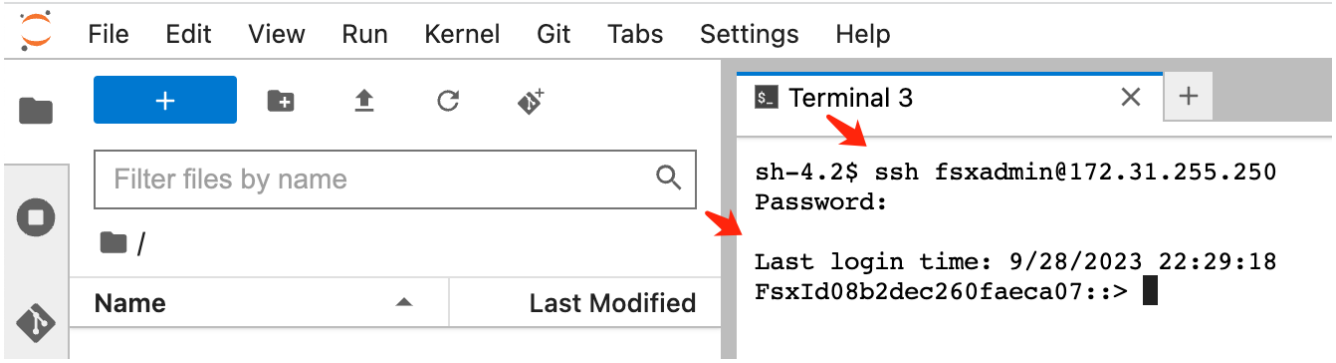
3. Öffnen Sie die erstellte **SageMaker Notebook-Instanz** und klicken Sie auf **JupyterLab öffnen**.



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



5. Geben Sie den SSH-Befehl ssh <Administrator-Benutzername>@ <ONTAP> ein, um sich beim FSx ONTAP Dateisystem anzumelden. (Benutzername und IP-Adresse werden aus Schritt 2 abgerufen) Bitte verwenden Sie das Kennwort, das Sie beim Erstellen der **Storage-virtuellen Maschine** verwendet haben.



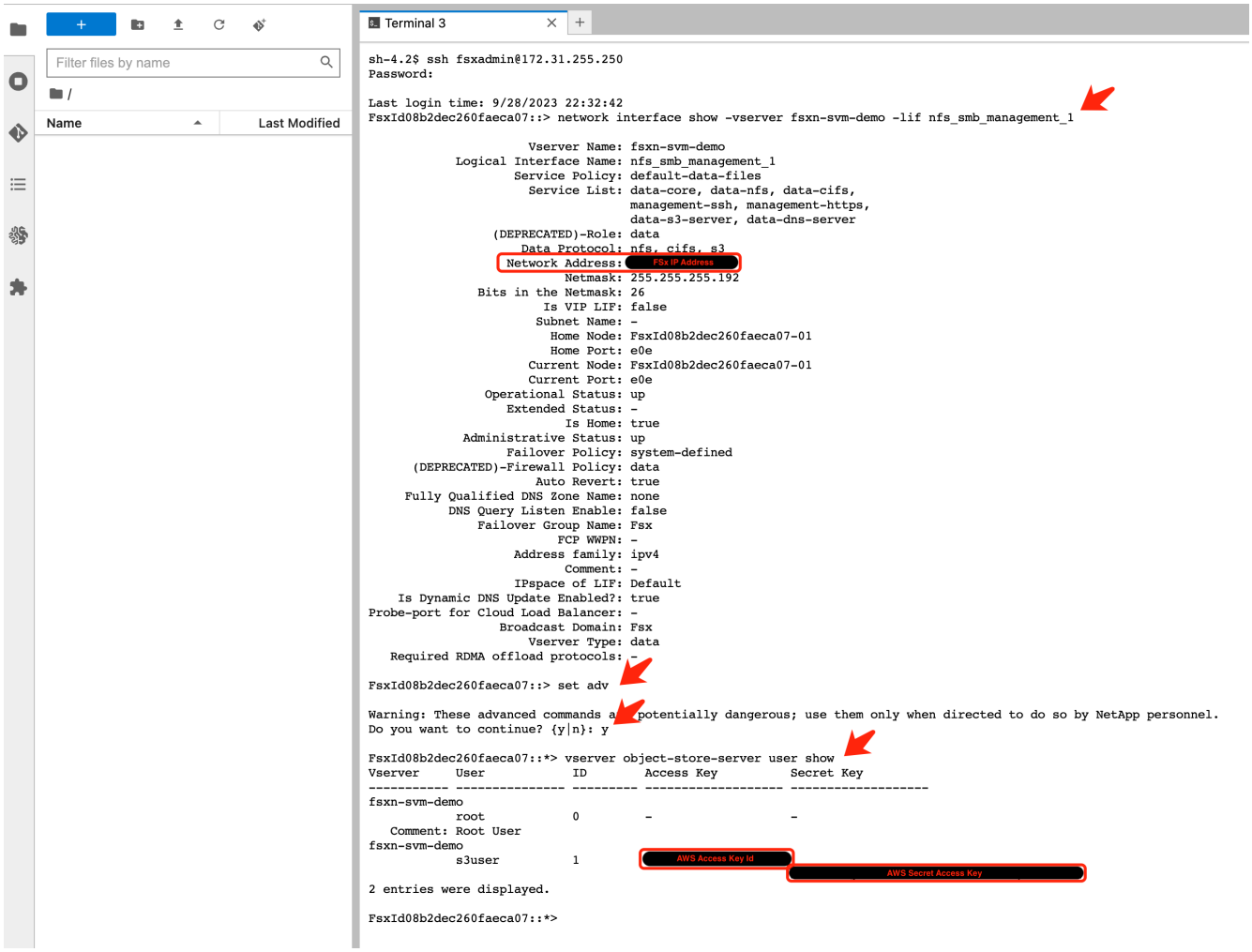
6. Führen Sie die Befehle in der folgenden Reihenfolge aus. Wir verwenden **fsxn-ontap** als Namen für den **Namen des privaten S3-Buckets von FSx ONTAP *. Bitte verwenden Sie den *Namen der virtuellen Speichermaschine** für das Argument **-vserver**.

```
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 Anmeldeinformationen für FSx ONTAP private S3 abzurufen.

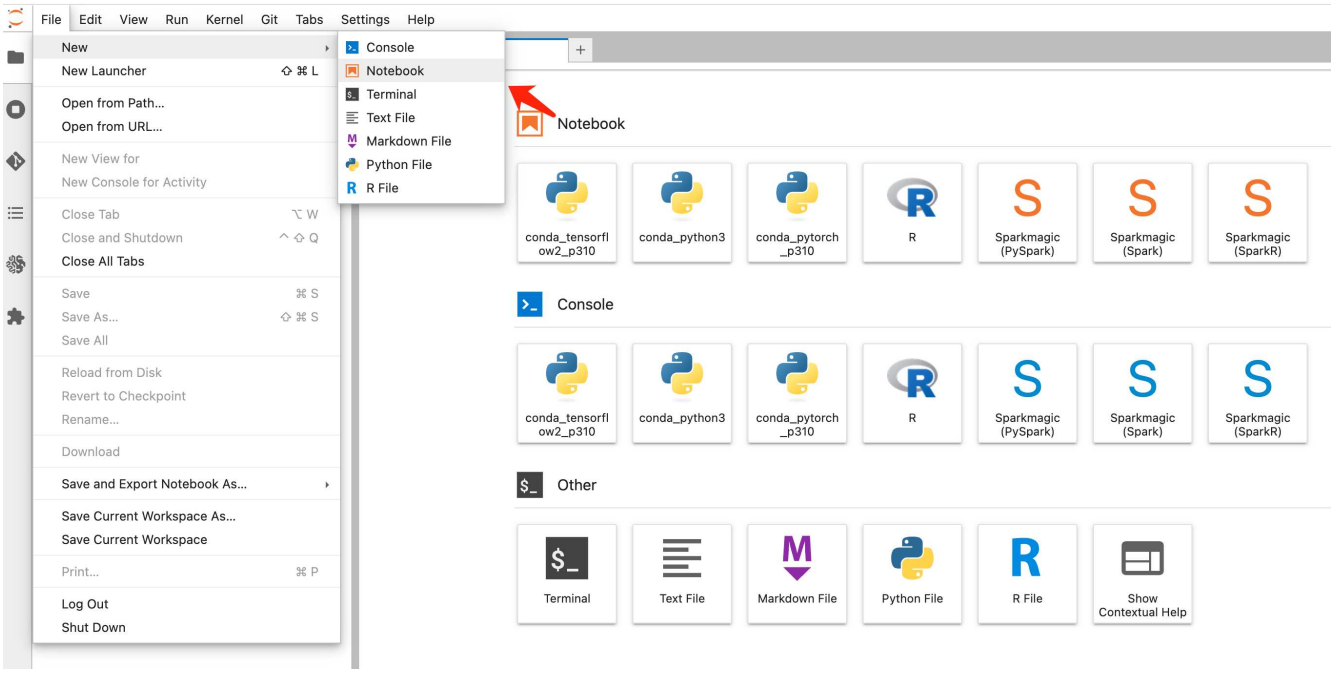
```
network interface show -vserver fsxn-svm-demo -lif nfs_smb_management_1

set adv

vserver object-store-server user show
```

8. Bewahren Sie die Endpunkt-IP und die Anmeldeinformationen für die zukünftige Verwendung auf.



Client-Konfiguration

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



2. Verwenden Sie den folgenden Code als Workaround-Lösung, um Dateien in den privaten S3-Bucket von FSx ONTAP hochzuladen. Ein umfassendes 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 Checkliste zum Debuggen

- Stellen Sie sicher, dass sich die SageMaker Notebook-Instance und das FSx ONTAP Dateisystem im selben VPC befinden.
- Denken Sie daran, den Befehl **set dev** auf ONTAP auszuführen, um die Berechtigungsstufe auf **dev**

festzulegen.

FAQ (Stand: 27.09.2023)

F: Warum erhalte ich beim Hochladen von Dateien auf FSx ONTAP die Fehlermeldung „**Beim Aufrufen des Vorgangs „CreateMultipartUpload“ ist ein Fehler aufgetreten (NotImplemented): Der von Ihnen angeforderte S3-Befehl ist nicht implementiert**“?

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

F: Warum erhalte ich beim Hochladen von Dateien auf FSx ONTAP die Fehlermeldung „**Beim Aufrufen der PutObject-Operationen ist ein Fehler aufgetreten (Zugriff verweigert): Zugriff verweigert**“?

A: Um von einer SageMaker Notebook-Instanz auf den privaten S3-Bucket von FSx ONTAP zuzugreifen, ändern Sie die AWS-Anmeldeinformationen in die FSx ONTAP Anmeldeinformationen. Um der Instanz Schreibberechtigungen zu erteilen, ist jedoch eine Problemumgehungslösung erforderlich, bei der der Bucket gemountet und der Shell-Befehl „chmod“ ausgeführt wird, um die Berechtigungen zu ändern.

F: Wie kann ich den privaten S3-Bucket von FSx ONTAP in andere SageMaker ML-Dienste integrieren?

A: Leider bietet das SageMaker Services SDK keine Möglichkeit, den Endpunkt für den privaten S3-Bucket anzugeben. Daher ist FSx ONTAP S3 nicht mit SageMaker-Diensten wie Sagemaker Data Wrangler, Sagemaker Clarify, Sagemaker Glue, Sagemaker Athena, Sagemaker AutoML und anderen kompatibel.

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) zum Trainieren von PyTorch-Modellen in SageMaker, insbesondere für ein Projekt zur Klassifizierung der Reifenqualität.

Einführung

Dieses Tutorial bietet ein praktisches Beispiel für ein Computer Vision-Klassifizierungsprojekt und vermittelt praktische Erfahrung beim Erstellen von ML-Modellen, die FSx ONTAP als Datenquelle innerhalb der SageMaker-Umgebung verwenden. Das Projekt konzentriert sich auf die Verwendung von PyTorch, einem Deep-Learning-Framework, um die Reifenqualität anhand von Reifenbildern zu klassifizieren. Der Schwerpunkt liegt auf der Entwicklung von Modellen für maschinelles Lernen unter Verwendung von FSx ONTAP als Datenquelle in Amazon SageMaker.

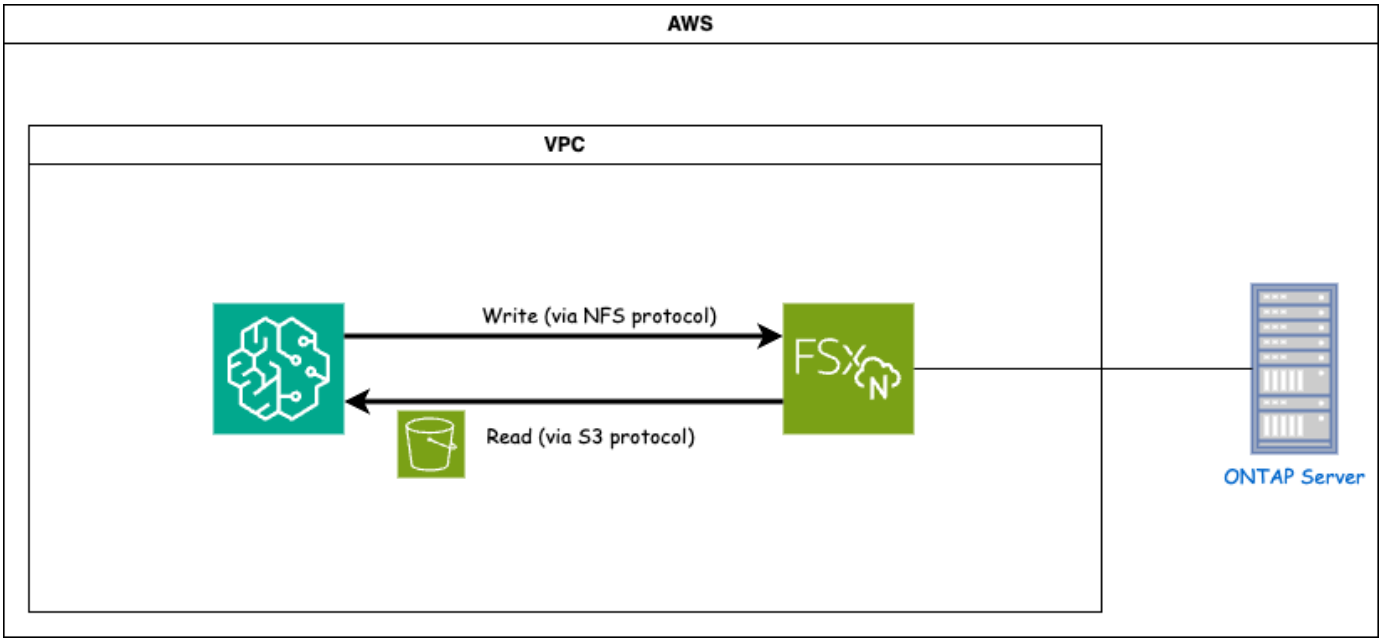
Was ist FSx ONTAP

Amazon FSx ONTAP ist tatsächlich eine vollständig verwaltete Speicherlösung, die von AWS angeboten wird. Es nutzt das ONTAP Dateisystem von NetApp, um zuverlässigen und leistungsstarken Speicher bereitzustellen. Mit Unterstützung für Protokolle wie NFS, SMB und iSCSI ermöglicht es nahtlosen Zugriff von verschiedenen Compute-Instanzen und Containern. Der Dienst ist auf außergewöhnliche Leistung ausgelegt und gewährleistet schnelle und effiziente Datenvorgänge. Darüber hinaus bietet es eine hohe Verfügbarkeit und Haltbarkeit und stellt sicher, dass Ihre Daten zugänglich und geschützt bleiben. Darüber hinaus ist die

Speicherkapazität von Amazon FSx ONTAP skalierbar, sodass Sie sie problemlos an Ihre Bedürfnisse anpassen können.

Voraussetzung

Netzwerkumgebung



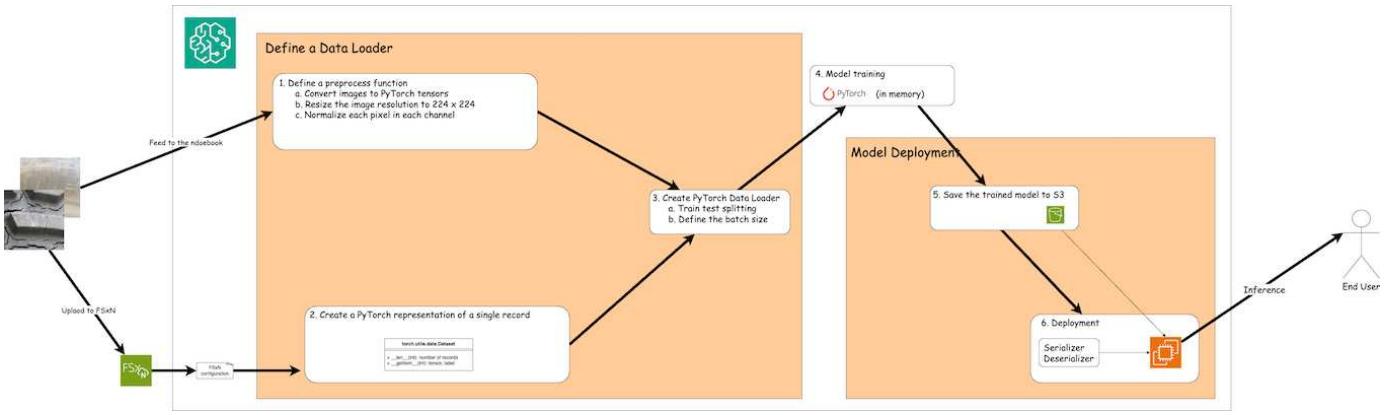
FSx ONTAP (Amazon FSx ONTAP) ist ein AWS-Speicherdienst. Es umfasst ein Dateisystem, das auf dem NetApp ONTAP -System ausgeführt wird, und eine von AWS verwaltete System-Virtual-Machine (SVM), die eine Verbindung damit herstellt. Im bereitgestellten Diagramm befindet sich der von AWS verwaltete NetApp ONTAP -Server außerhalb des VPC. Der SVM dient als Vermittler zwischen SageMaker und dem NetApp ONTAP -System, empfängt Betriebsanforderungen von SageMaker und leitet sie an den zugrunde liegenden Speicher weiter. Um auf FSx ONTAP zuzugreifen, muss SageMaker im selben VPC wie die FSx ONTAP Bereitstellung platziert werden. Diese Konfiguration gewährleistet die Kommunikation und den Datenzugriff zwischen SageMaker und FSx ONTAP.

Datenzugriff

In realen Szenarien verwenden Datenwissenschaftler normalerweise die vorhandenen, in FSx ONTAP gespeicherten Daten, um ihre Modelle für maschinelles Lernen zu erstellen. Da das FSx ONTAP Dateisystem nach der Erstellung jedoch zunächst leer ist, ist es zu Demonstrationszwecken erforderlich, die Trainingsdaten manuell hochzuladen. Dies kann erreicht werden, indem FSx ONTAP als Volume in SageMaker eingebunden wird. Sobald das Dateisystem erfolgreich gemountet ist, können Sie Ihren Datensatz an den gemounteten Speicherort hochladen und ihn so für das Training Ihrer Modelle innerhalb der SageMaker-Umgebung zugänglich machen. Mit diesem Ansatz können Sie die Speicherkapazität und Funktionen von FSx ONTAP nutzen, während Sie mit SageMaker zur Modellentwicklung und -schulung arbeiten.

Der Datenlesevorgang umfasst die Konfiguration von FSx ONTAP als privater S3-Bucket. Um die detaillierten Konfigurationsanweisungen zu erfahren, lesen Sie bitte"Teil 1 – Integration von Amazon FSx for NetApp ONTAP (FSx ONTAP) als privater S3-Bucket in AWS SageMaker"

Integrationsübersicht



Der Workflow zur Verwendung von Trainingsdaten in FSx ONTAP zum Erstellen eines Deep-Learning-Modells in SageMaker lässt sich in drei Hauptschritte zusammenfassen: Definition des Datenladers, Modelltraining und Bereitstellung. Auf hoher Ebene bilden diese Schritte die Grundlage einer MLOps-Pipeline. Für eine umfassende Umsetzung sind jedoch zu jedem Schritt mehrere detaillierte Unterschritte erforderlich. Diese Unterschritte umfassen verschiedene Aufgaben wie Datenvorverarbeitung, Datensatzaufteilung, Modellkonfiguration, Hyperparameter-Tuning, Modellbewertung und Modellbereitstellung. Diese Schritte gewährleisten einen gründlichen und effektiven Prozess zum Erstellen und Bereitstellen von Deep-Learning-Modellen mithilfe von Trainingsdaten von FSx ONTAP innerhalb der SageMaker-Umgebung.

Schrittweise Integration

Loader

Um ein PyTorch-Deep-Learning-Netzwerk mit Daten zu trainieren, wird ein Datenlader erstellt, der die Dateneingabe erleichtert. Der Datenlader definiert nicht nur die Batchgröße, sondern bestimmt auch das Verfahren zum Lesen und Vorverarbeiten jedes Datensatzes innerhalb des Batches. Durch die Konfiguration des Datenladers können wir die Verarbeitung der Daten in Stapeln durchführen 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 obige Codeausschnitt demonstriert die Definition von Bildvorverarbeitungstransformationen mithilfe des Moduls **torchvision.transforms**. In diesem Tutorial wird das Vorverarbeitungsobjekt erstellt, um eine Reihe von Transformationen anzuwenden. Zunächst wandelt die **ToTensor()**-Transformation das Bild in eine Tensordarstellung um. Anschließend wird die Bildgröße durch die Transformation **Resize 224,224** auf eine

feste Größe von 224 x 224 Pixeln geändert. Schließlich normalisiert die **Normalize()**-Transformation die Tensorwerte, indem sie den Mittelwert subtrahiert und durch die Standardabweichung entlang jedes Kanals dividiert. Die zur Normalisierung verwendeten Mittelwert- und Standardabweichungswerte werden üblicherweise in vortrainierten neuronalen Netzwerkmodellen verwendet. Insgesamt bereitet dieser Code die Bilddaten für die weitere Verarbeitung oder Eingabe in ein vortrainiertes Modell vor, indem er sie in einen Tensor umwandelt, ihre Größe ändert 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 der Daten für jeden Datensatz. Innerhalb der Funktion ***getitem*** verwendet der Code das S3-Bucket-Objekt boto3, um die Binärdaten von FSx ONTAP abzurufen. Der Codestil für den Zugriff auf Daten von FSx ONTAP ähnelt dem Lesen von Daten von Amazon S3. Die folgende Erklärung befasst sich eingehend mit dem Erstellungsprozess des privaten S3-Objekts **Bucket**.

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

Im oben genannten Beispiel wird das Bucket-Objekt verwendet, um das PyTorch-Dataset-Objekt zu instanziieren. Das Dataset-Objekt wird im folgenden 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 bereitgestellten Beispiel wird eine Batchgröße von 64 angegeben, was bedeutet, dass jeder Batch 64 Datensätze enthält. Durch die Kombination der PyTorch-Klasse **Dataset**, der Vorverarbeitungsfunktion und der Trainings-Batchgröße erhalten wir den Datenlader für das Training. Dieser Datenlader erleichtert den Prozess der stapelweisen Iteration durch den Datensatz während der Trainingsphase.

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 Netzwerkmodell namens **TyreQualityClassifier**, das Faltungsschichten und eine lineare Schicht zur Klassifizierung der Reifenqualität verwendet. Die Trainingsschleife iteriert über Datenstapel, berechnet den Verlust und aktualisiert die Parameter des Modells mithilfe von Backpropagation und Optimierung. Darüber hinaus werden zu Überwachungszwecken die aktuelle Zeit, Epoche, Charge und der Verlust 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 für die Bereitstellung die Speicherung des Modells in S3 erfordert. Durch das Hochladen des Modells auf **Amazon S3** wird es für SageMaker zugänglich, was die Bereitstellung und Inferenz des bereitgestellten Modells 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 Serialisierer, **TyreQualitySerializer**, der Eingabedaten als PyTorch-Tensor vorverarbeitet und serialisiert. Die Klasse **TyreQualityPredictor** ist ein benutzerdefinierter Prädiktor, der den definierten Serialisierer und einen **JSONDeserializer** verwendet. Der Code erstellt außerdem ein **PyTorchModel**-Objekt, um den S3-Speicherort, die IAM-Rolle, die Framework-Version und den Einstiegspunkt für die Inferenz des

Modells anzugeben. 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 generierte Endpunktname angegeben werden. Dadurch kann das PyTorch-Modell bereitgestellt und für Inferenzen auf SageMaker zugänglich gemacht werden.

Schlussfolgerung

```
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 ein Beispiel für die Verwendung des bereitgestellten Endpunkts zur Durchführung der Inferenz.

Teil 3 – Erstellen einer vereinfachten MLOps-Pipeline (CI/CT/CD)

Dieser Artikel bietet eine Anleitung zum Erstellen einer MLOps-Pipeline mit AWS-Diensten und konzentriert sich dabei auf die automatische Neuschulung, Bereitstellung und Kostenoptimierung von Modellen.

Einführung

In diesem Tutorial erfahren Sie, wie Sie verschiedene AWS-Dienste nutzen, um eine einfache MLOps-Pipeline zu erstellen, die Continuous Integration (CI), Continuous Training (CT) und Continuous Deployment (CD) umfasst. Im Gegensatz zu herkömmlichen DevOps-Pipelines erfordert MLOps zusätzliche Überlegungen, um den Betriebszyklus abzuschließen. In diesem Tutorial erhalten Sie Einblicke in die Integration von CT in die MLOps-Schleife, wodurch ein kontinuierliches Training Ihrer Modelle und eine nahtlose Bereitstellung für die Inferenz ermöglicht wird. Das Tutorial führt Sie durch den Prozess der Nutzung von AWS-Diensten zum Einrichten dieser End-to-End-MLOps-Pipeline.

Manifest

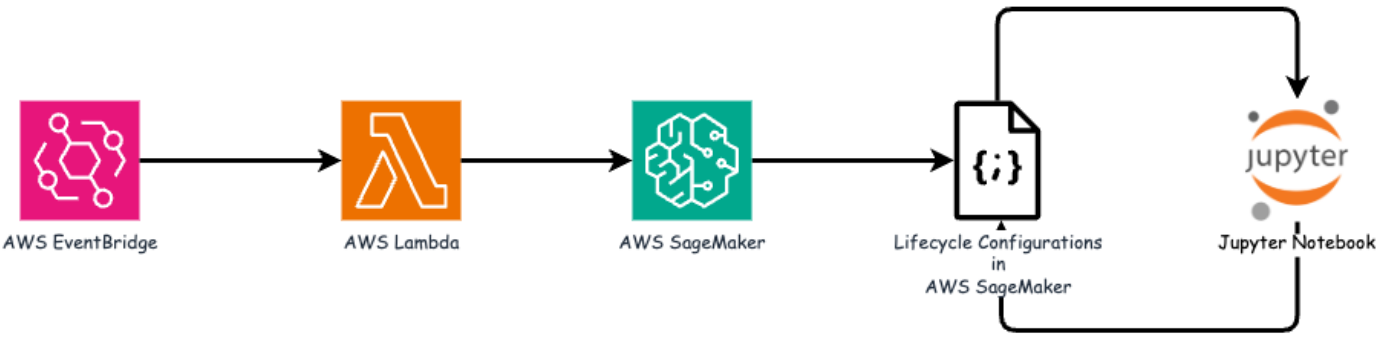
Funktionalität	Name	Kommentar
Datenspeicherung	AWS FSx ONTAP	Weitere Informationen finden Sie unter "Teil 1 – Integration von Amazon FSx for NetApp ONTAP (FSx ONTAP) als privater S3-Bucket in AWS SageMaker" .

Funktionalität	Name	Kommentar
Data Science-IDE	AWS SageMaker	Dieses Tutorial basiert auf dem Jupyter-Notebook, das in"Teil 2 – Nutzung von Amazon FSx for NetApp ONTAP (FSx ONTAP) als Datenquelle für das Modelltraining in SageMaker" .
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. Dieses Tutorial verwendet für den Trainingsprozess in FSx ONTAP gespeicherte Daten.
- Eine **SageMaker Notebook-Instanz**, die so konfiguriert ist, dass sie dieselbe VPC wie das oben erwähnte FSx ONTAP Dateisystem gemeinsam nutzt.
- Stellen Sie vor dem Auslösen der **AWS Lambda-Funktion** sicher, dass sich die **SageMaker Notebook-Instanz** im Status **gestoppt** befindet.
- Der Instanztyp **ml.g4dn.xlarge** ist erforderlich, um die für die Berechnungen tiefer neuronaler Netzwerke erforderliche GPU-Beschleunigung zu nutzen.

Architektur



Diese MLOps-Pipeline ist eine praktische Implementierung, die einen Cron-Job verwendet, um eine serverlose Funktion auszulösen, die wiederum einen AWS-Dienst ausführt, der mit einer Lebenszyklus-Callback-Funktion registriert ist. Als Cronjob fungiert die **AWS EventBridge**. Es ruft regelmäßig eine **AWS Lambda-Funktion** auf, die für die Neuschulung und erneute Bereitstellung des Modells verantwortlich ist. Bei diesem Vorgang wird die **AWS SageMaker Notebook**-Instanz hochgefahren, um die erforderlichen Aufgaben auszuführen.

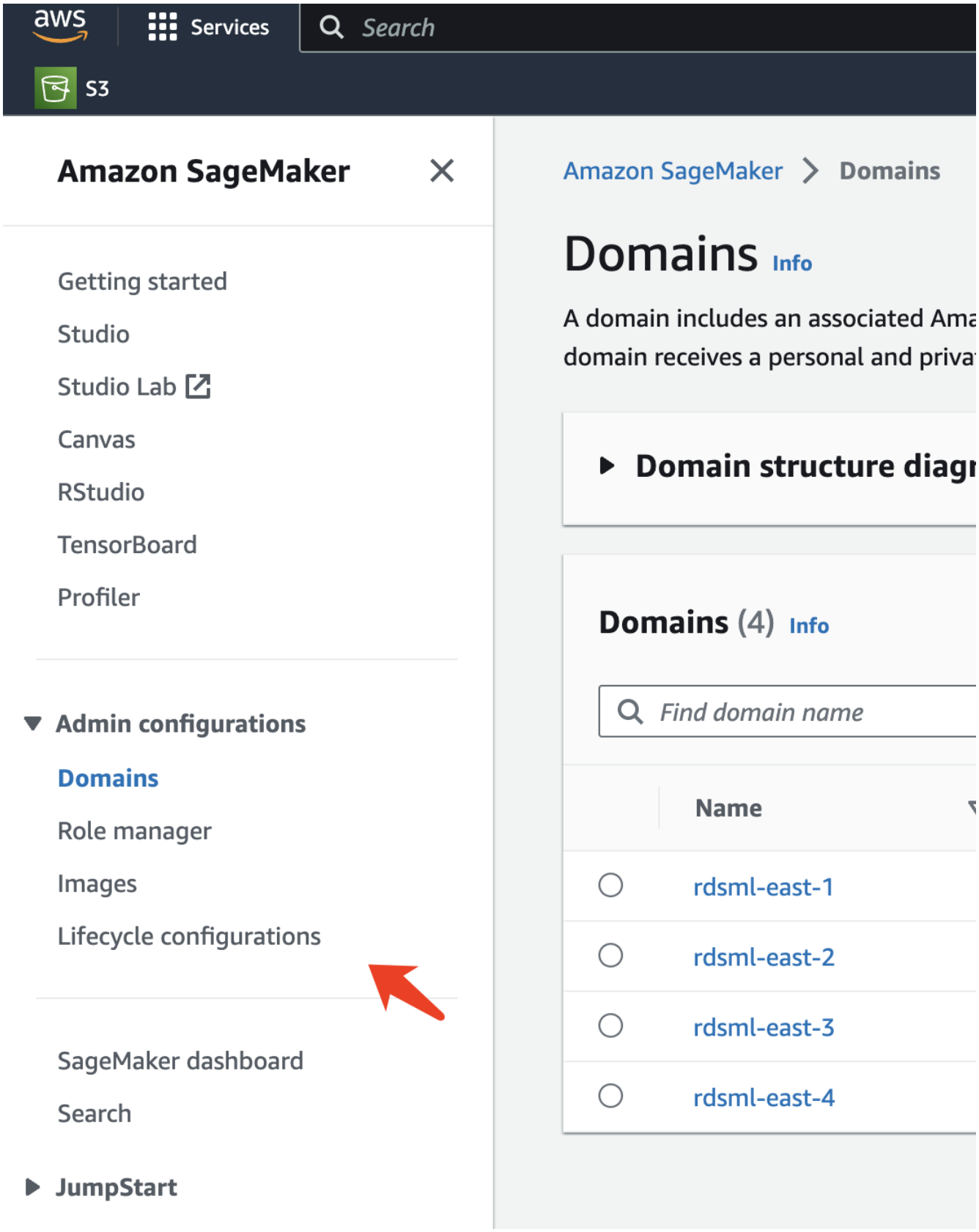
Schritt-für-Schritt-Konfiguration

Lebenszykluskonfigurationen

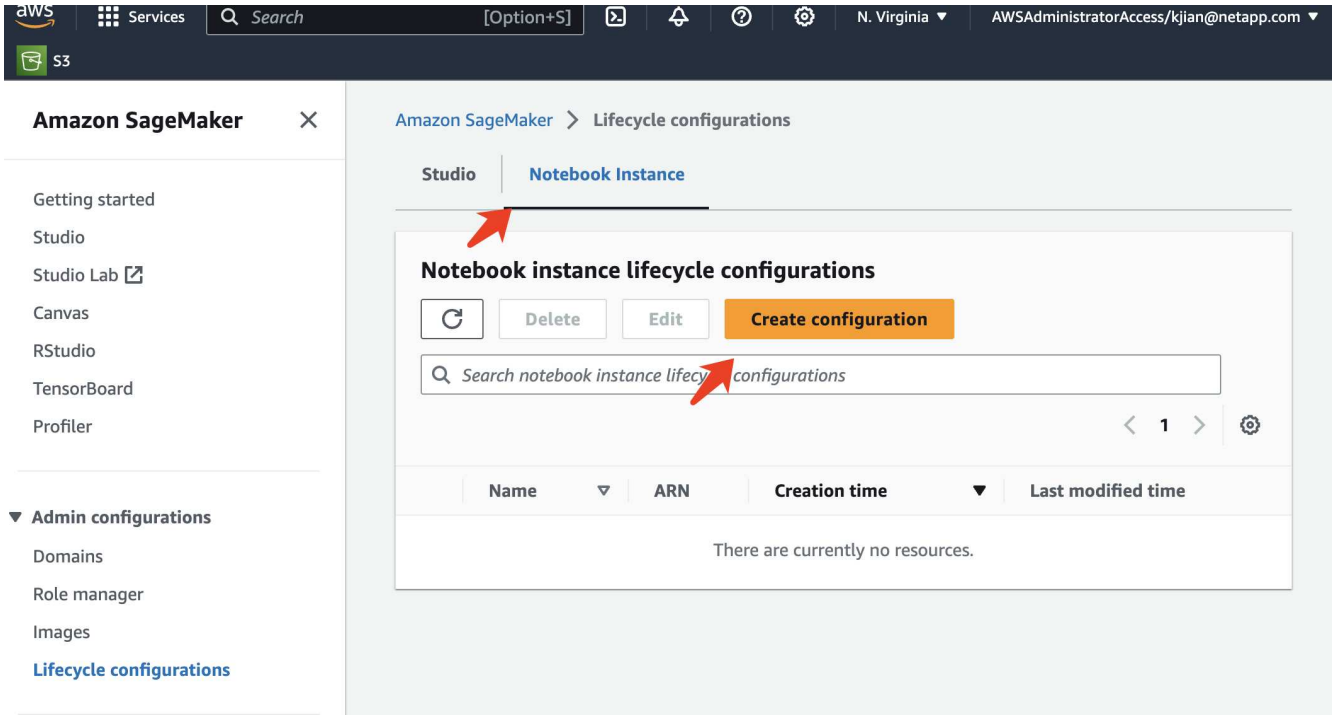
Um die Lebenszyklus-Rückruffunktion für die AWS SageMaker Notebook-Instanz zu konfigurieren, verwenden Sie **Lebenszykluskonfigurationen**. Mit diesem Dienst können Sie die erforderlichen Aktionen definieren, die beim Hochfahren der Notebook-Instanz ausgeführt werden sollen. Insbesondere kann innerhalb der **Lebenszykluskonfigurationen** ein Shell-Skript implementiert werden, um die Notebook-Instanz automatisch herunterzufahren, sobald die Trainings- und Bereitstellungsprozesse abgeschlossen sind. Dies ist eine erforderliche Konfiguration, da die Kosten einer der wichtigsten Aspekte bei MLOps sind.

Es ist wichtig zu beachten, dass die Konfiguration für **Lebenszykluskonfigurationen** im Voraus eingerichtet werden muss. Daher wird empfohlen, die Konfiguration dieses Aspekts zu priorisieren, bevor Sie mit der Einrichtung der anderen MLOps-Pipeline fortfahren.

1. Um eine Lebenszykluskonfiguration einzurichten, öffnen Sie das **Sagemaker**-Bedienfeld und navigieren Sie zu **Lebenszykluskonfigurationen** im Abschnitt **Admin-Konfigurationen**.



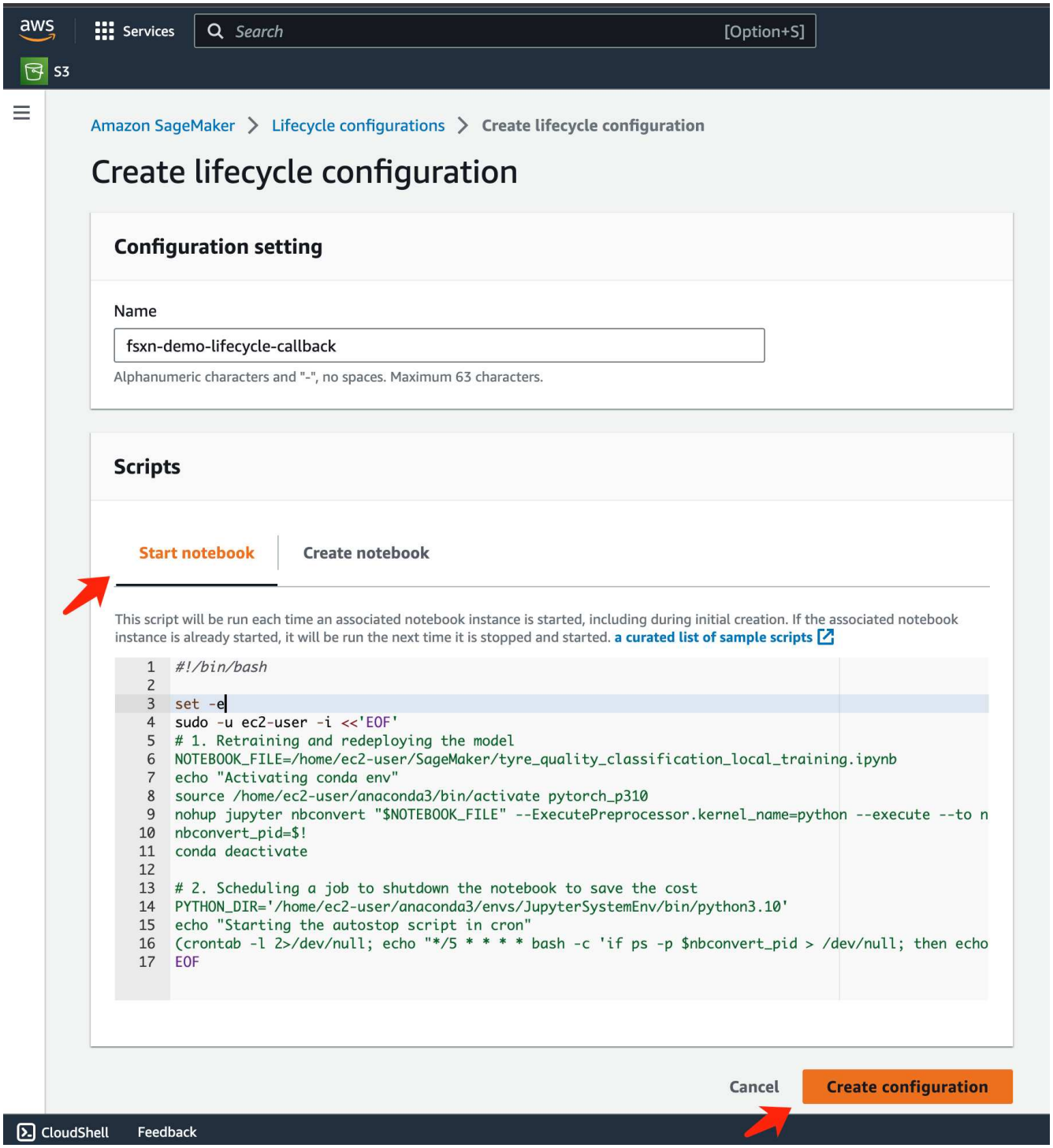
2. Wählen Sie die Registerkarte **Notebook-Instanz** und klicken Sie auf die Schaltfläche **Konfiguration erstellen**



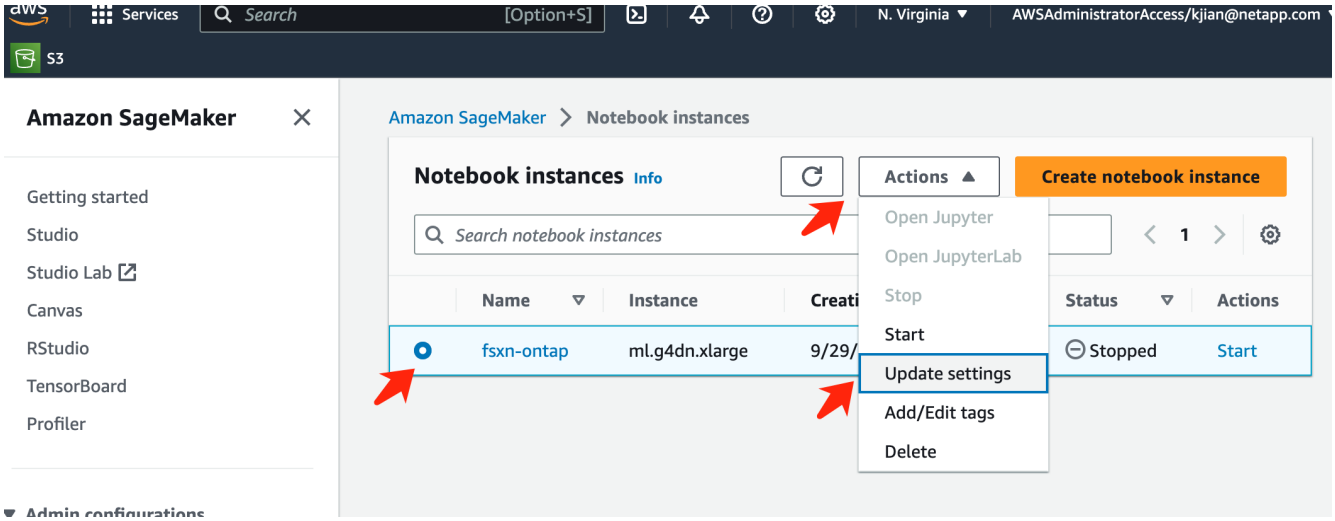
3. Fügen Sie den folgenden Code in den Eingabebereich ein.

```
#!/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
```

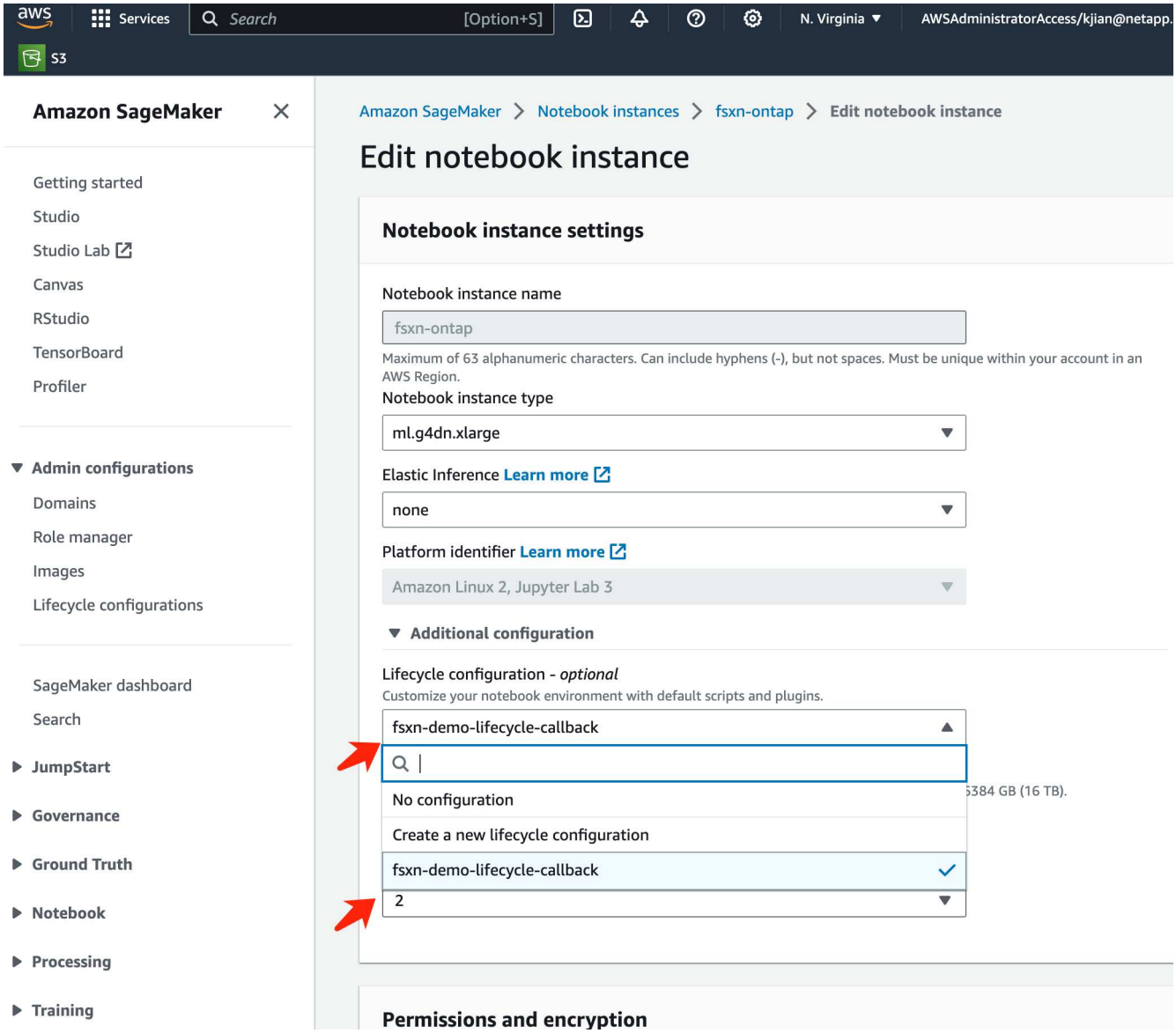
4. Dieses Skript führt das Jupyter-Notebook aus, das das erneute Trainieren und die erneute Bereitstellung des Modells für die Inferenz übernimmt. Nach Abschluss der Ausführung wird das Notebook innerhalb von 5 Minuten automatisch heruntergefahren. Weitere Informationen zur Problemstellung und zur Codeimplementierung finden Sie unter"Teil 2 – Nutzung von Amazon FSx for NetApp ONTAP (FSx ONTAP) als Datenquelle für das Modelltraining in SageMaker" .



5. Navigieren Sie nach der Erstellung zu den Notebook-Instanzen, wählen Sie die Zielinstanz aus und klicken Sie im Dropdown-Menü „Aktionen“ auf „Einstellungen aktualisieren“.



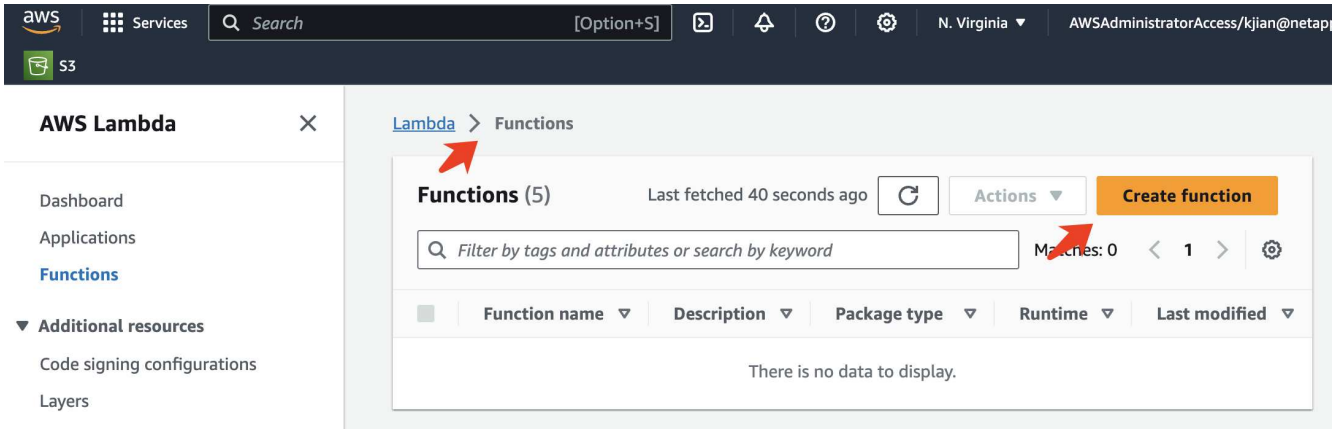
6. Wählen Sie die erstellte **Lebenszykluskonfiguration** aus und klicken Sie auf **Notebook-Instanz aktualisieren**.



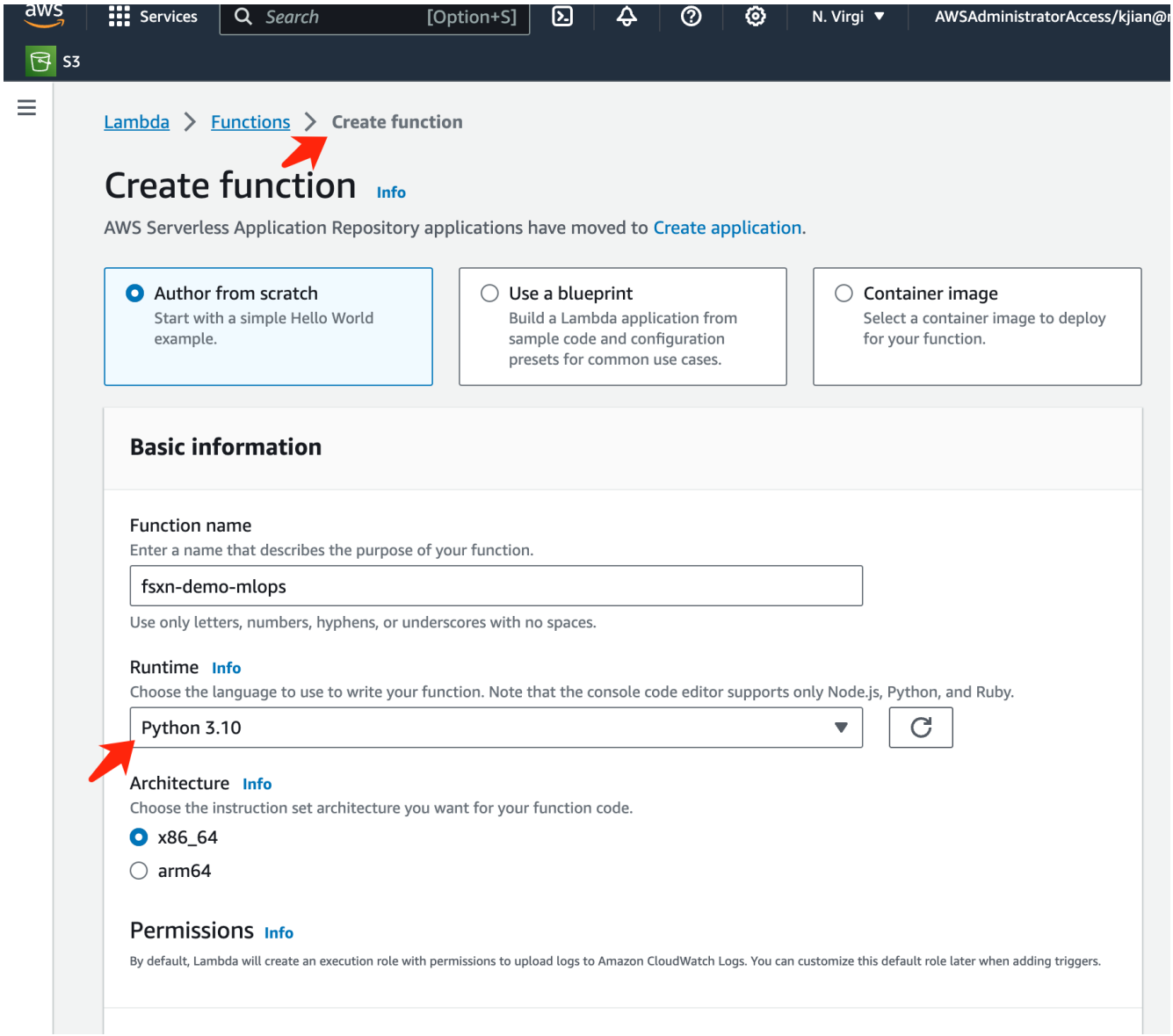
Serverlose AWS Lambda-Funktion

Wie bereits erwähnt, ist die **AWS Lambda-Funktion** für das Hochfahren der **AWS SageMaker Notebook-Instanz** verantwortlich.

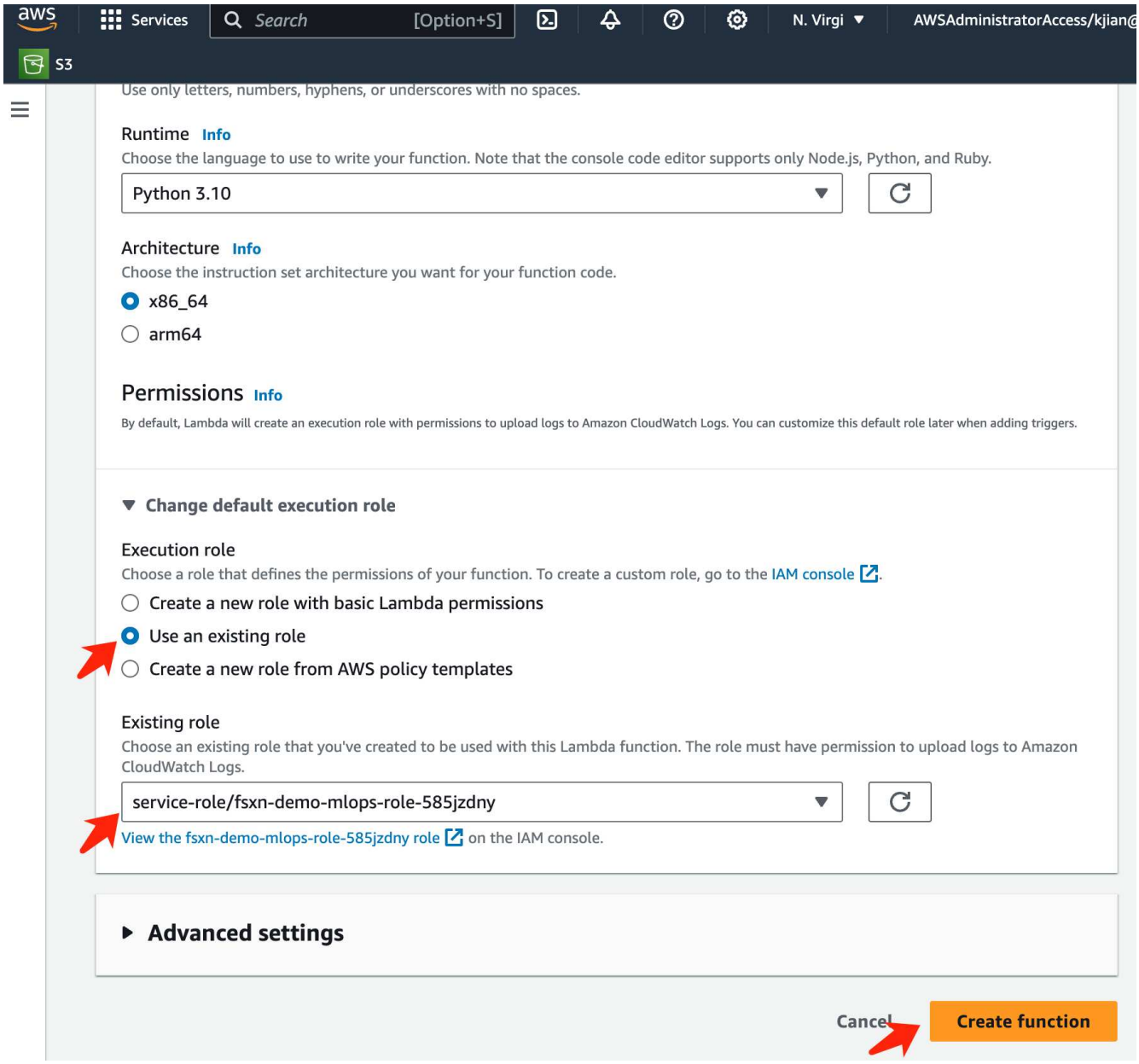
1. Um eine **AWS Lambda-Funktion** zu erstellen, navigieren Sie zum entsprechenden Bereich, wechseln Sie zur Registerkarte **Funktionen** und klicken Sie auf **Funktion erstellen**.



2. Bitte tragen Sie alle erforderlichen Angaben auf der Seite ein und denken Sie daran, die Runtime auf **Python 3.10** umzustellen.



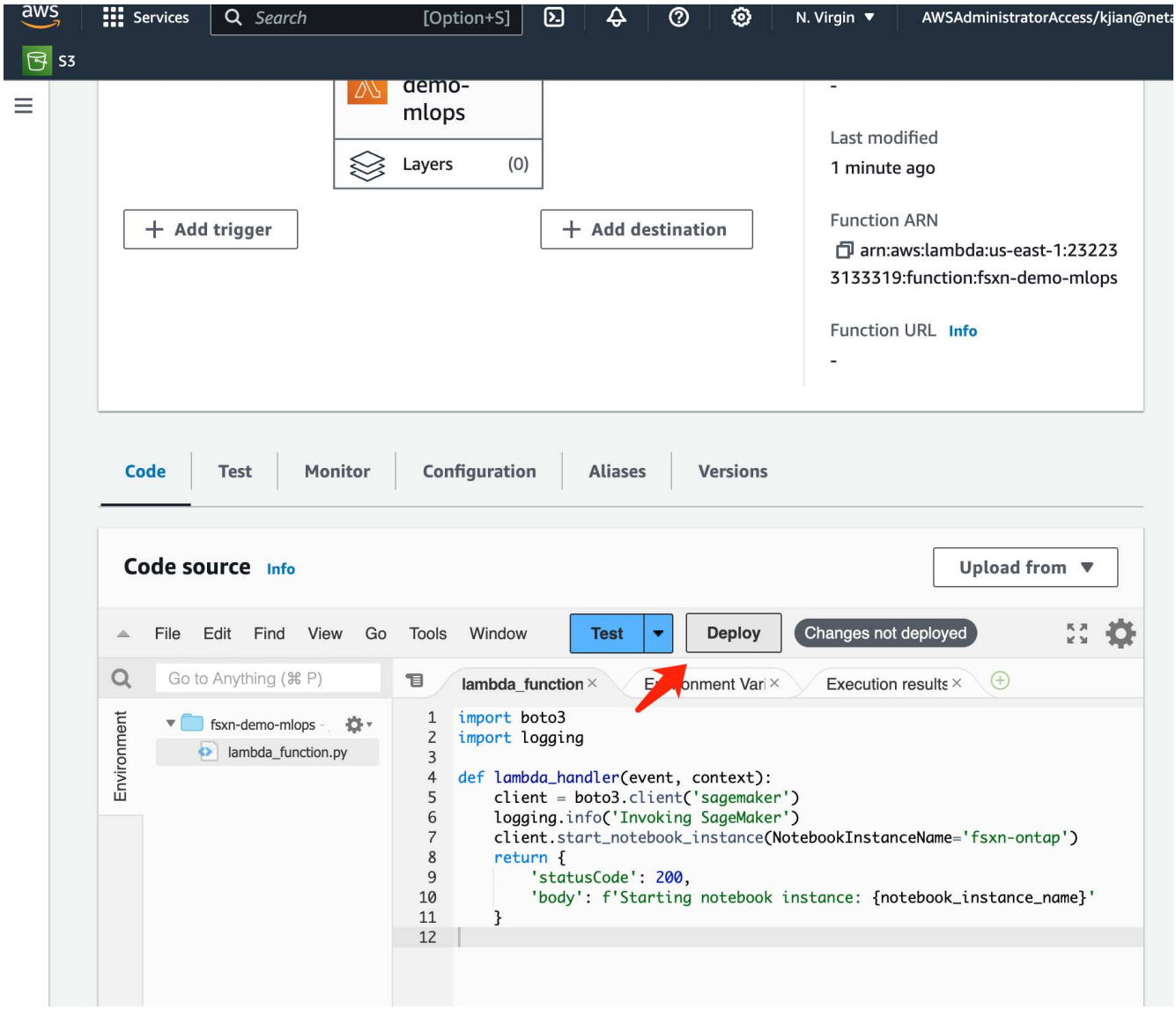
3. Bitte überprüfen Sie, ob die angegebene Rolle über die erforderliche Berechtigung **AmazonSageMakerFullAccess** verfügt, und klicken Sie auf die Schaltfläche **Funktion erstellen**.



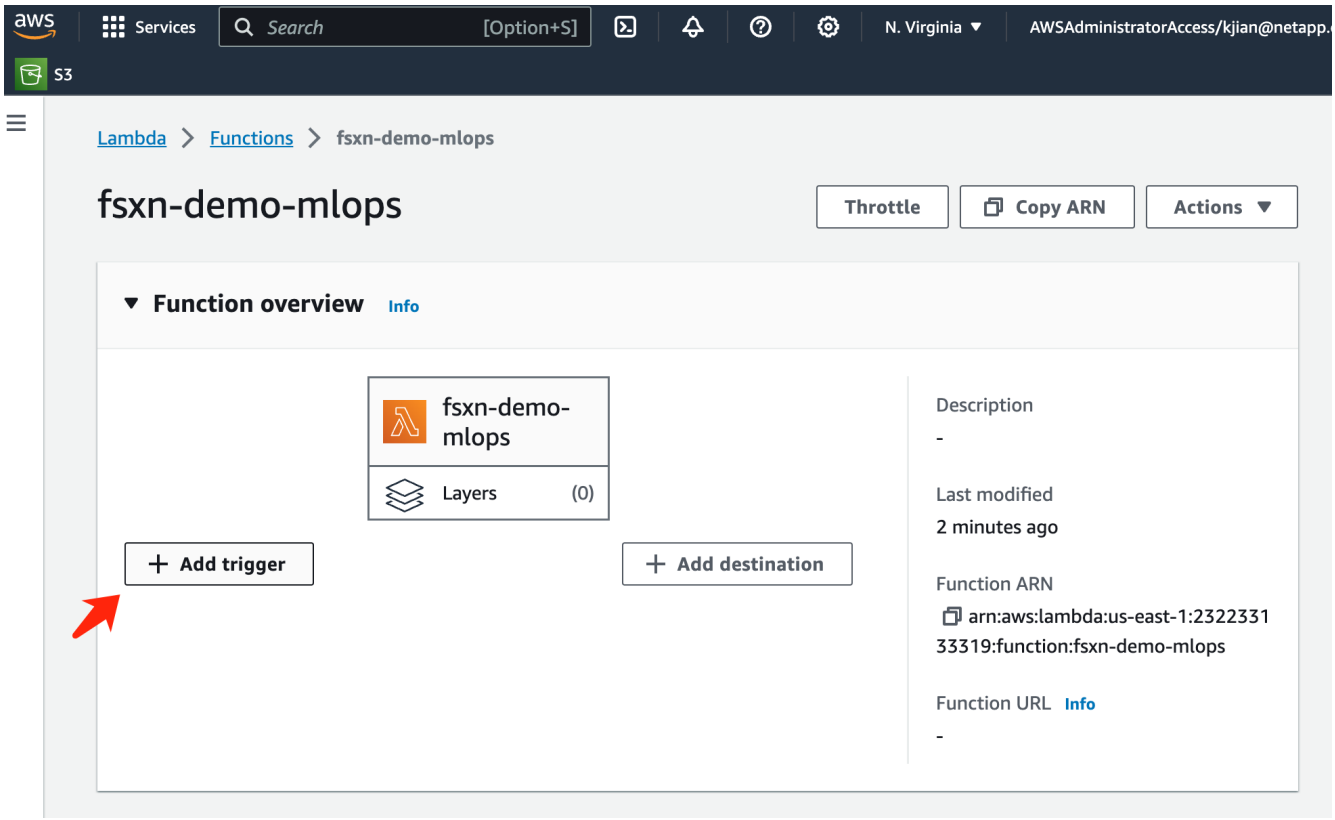
4. Wählen Sie die erstellte Lambda-Funktion aus. Kopieren Sie auf der Registerkarte „Code“ den folgenden Code und fügen Sie ihn in den Textbereich ein. Dieser Code startet die Notebook-Instanz mit dem Namen **fsxn-ontap**.

```
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}'
    }
```

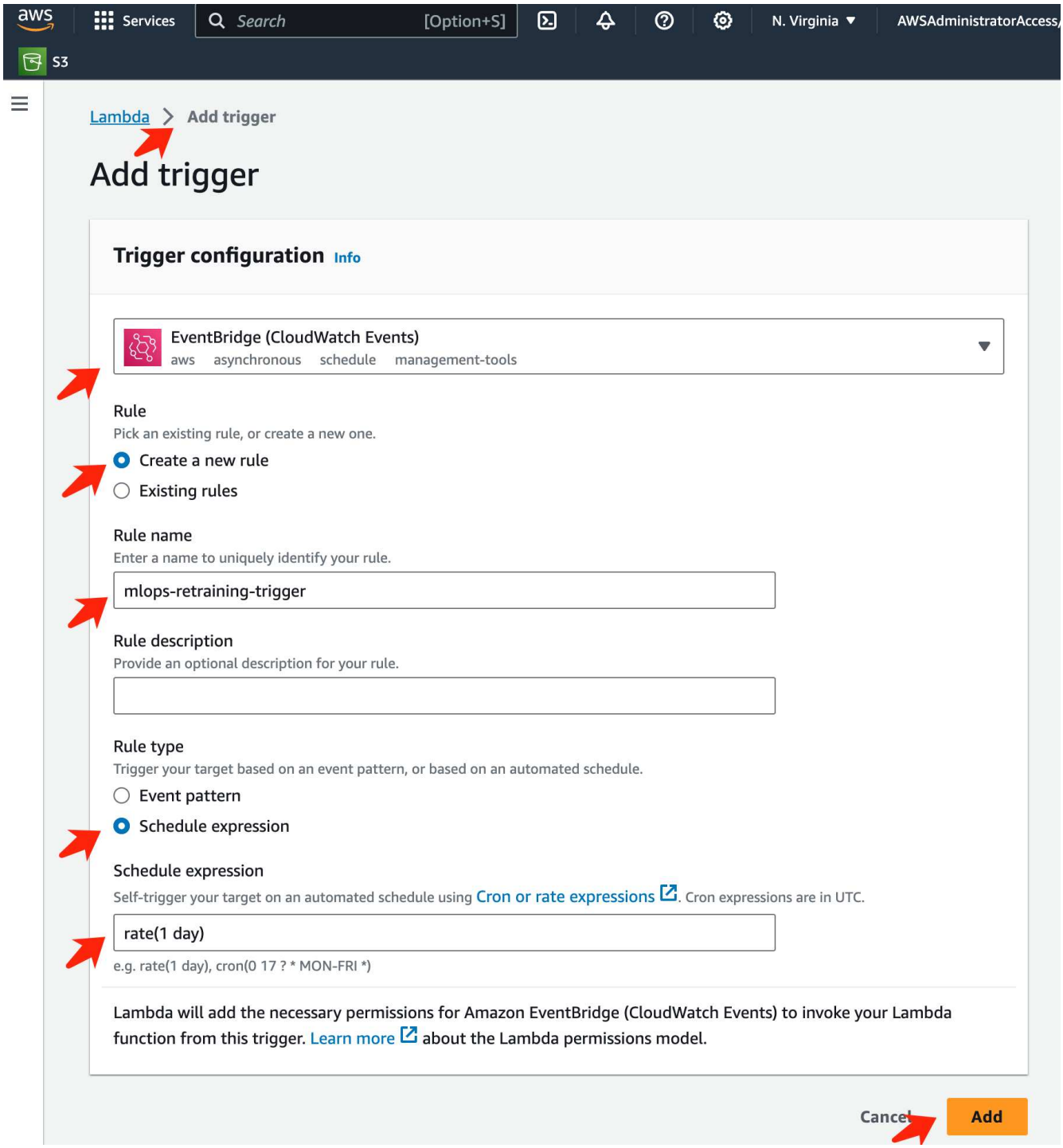
5. Klicken Sie auf die Schaltfläche **Bereitstellen**, um diese Codeänderung anzuwenden.



6. Um anzugeben, wie diese AWS Lambda-Funktion ausgelöst werden soll, klicken Sie auf die Schaltfläche „Trigger hinzufügen“.



7. Wählen Sie EventBridge aus dem Dropdown-Menü aus und klicken Sie dann auf das Optionsfeld „Neue Regel erstellen“. Geben Sie im Feld „Zeitplanausdruck“ Folgendes ein: `rate(1 day)` und klicken Sie auf die Schaltfläche „Hinzufügen“, um diese neue Cronjob-Regel zu erstellen und auf die AWS Lambda-Funktion anzuwenden.



Nach Abschluss der zweistufigen Konfiguration initiiert die **AWS Lambda-Funktion** täglich das **SageMaker-Notebook**, führt eine erneute Modellschulung mithilfe der Daten aus dem **FSx ONTAP**-Repository durch, stellt das aktualisierte Modell erneut in der Produktionsumgebung bereit und fährt die **SageMaker-Notebook-Instanz** automatisch herunter, um die Kosten zu optimieren. Dadurch wird sichergestellt, dass das Modell aktuell bleibt.

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