MLOps 向け FSx ONTAP

NetApp artificial intelligence solutions

NetApp
February 12, 2026

目次

MLOps 向け FSx ONTAP

MLOps 向けAmazon FSx for NetApp ONTAP (FSx ONTAP)

このセクションでは、AI インフラストラクチャ開発の実際のアプリケーションについて詳しく説明し、FSx ONTAPを使用して MLOps パイプラインを構築するエンドツーエンドのチュートリアルを提供します。 3 つの包括的な例で構成されており、この強力なデータ管理プラットフォームを介して MLOps のニーズを満たす方法をガイドします。

これらの記事は以下に焦点を当てています。

1. "パート 1 - Amazon FSx for NetApp ONTAP (FSx ONTAP) をプライベート S3 バケットとして AWS SageMaker に統合する"
2. "パート 2 - SageMaker でのモデルトレーニング用のデータソースとしてAmazon FSx for NetApp ONTAP (FSx ONTAP) を活用する"
3. "パート 3 - 簡素化された MLOps パイプラインの構築 (CI/CT/CD)"

このセクションの終わりまでに、FSx ONTAPを使用して MLOps プロセスを効率化する方法についてしっかりと理解できるようになります。

パート 1 - Amazon FSx for NetApp ONTAP (FSx ONTAP) をプライベート S3 バケットとして AWS SageMaker に統合する

このセクションでは、AWS SageMaker を使用して FSx ONTAP をプライベート S3 バケットとして設定する方法について説明します。

はじめに

このページでは、SageMaker を例に、FSx ONTAP をプライベート S3 バケットとして設定する方法について説明します。

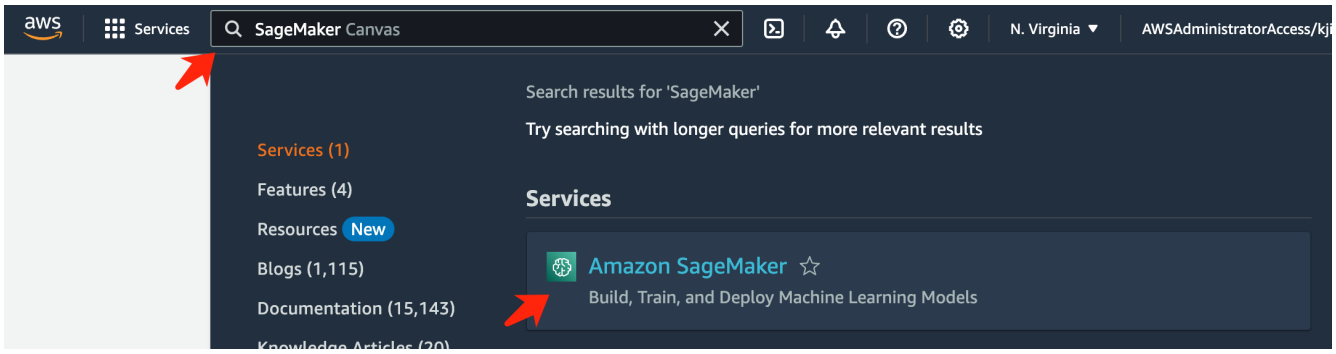
FSx ONTAPの詳細については、このプレゼンテーションをご覧ください ("ビデオリンク")

ユーザーガイド

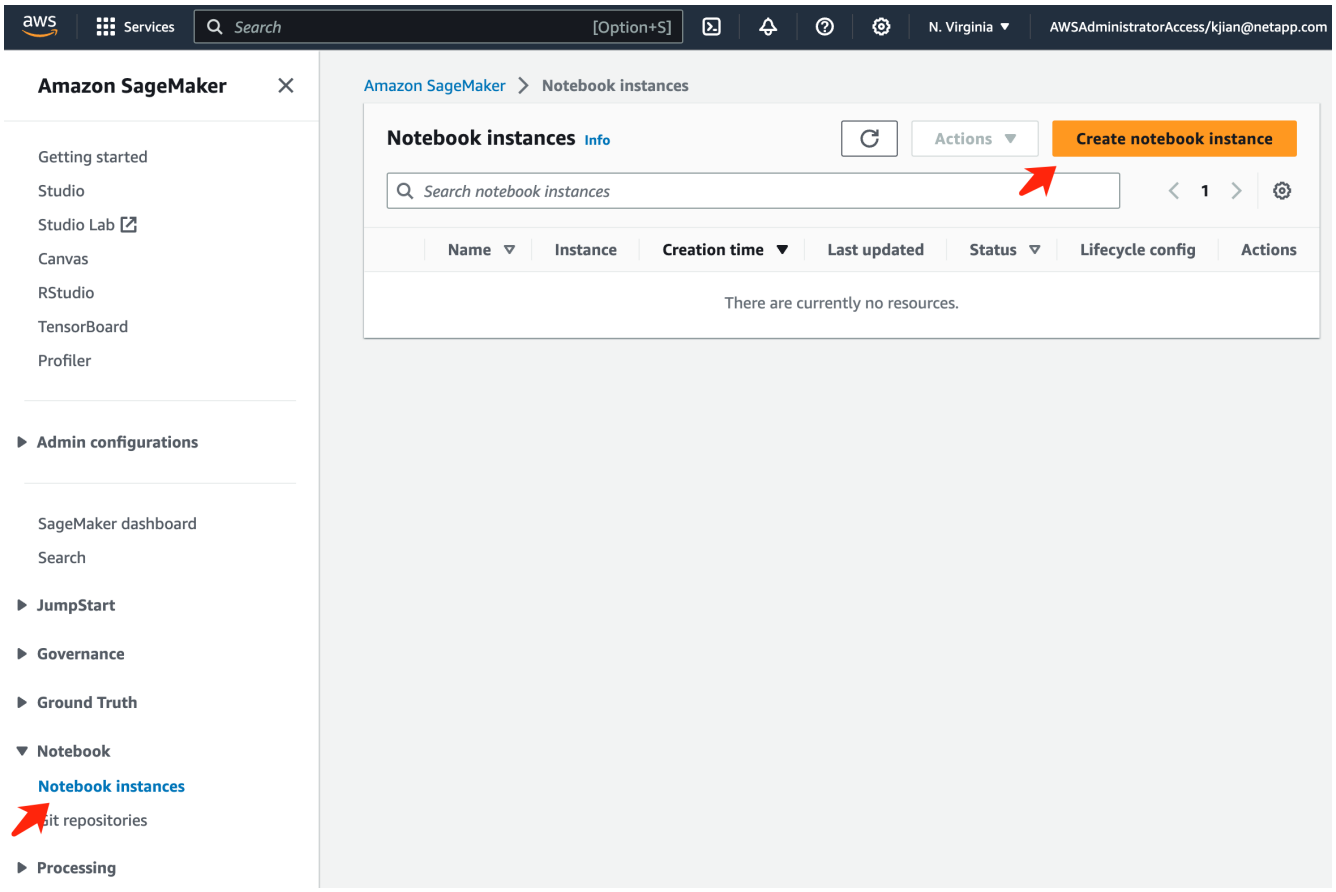
サーバーの作成

SageMakerノートブックインスタンスを作成する

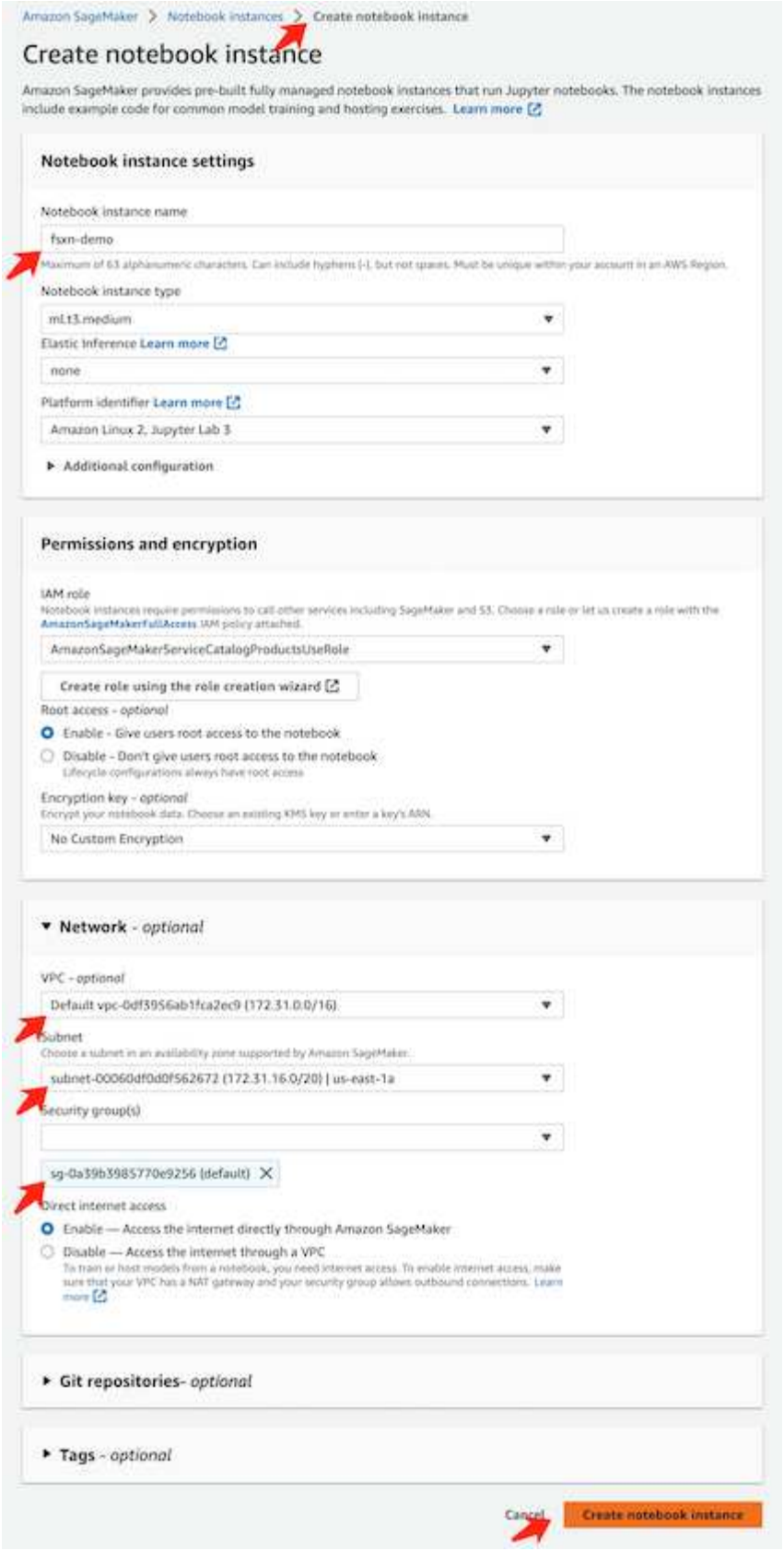
1. AWS コンソールを開きます。検索パネルで SageMaker を検索し、サービス **Amazon SageMaker** をクリックします。



2. [ノートブック] タブの [ノートブック インスタンス] を開き、オレンジ色のボタン [ノートブック インスタンスの作成] をクリックします。

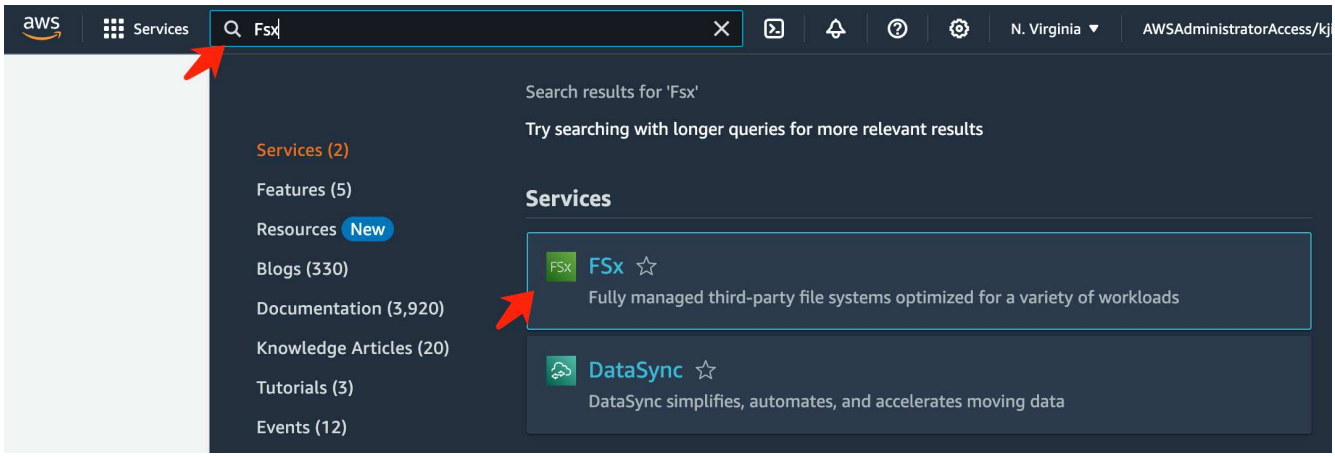


3. 作成ページで、ノートブックインスタンス名*を入力します。*ネットワーク*パネルを展開します。他のエントリはデフォルトのままにして、***VPC**、サブネット、および*セキュリティグループ*を選択します。 (この **VPC** と サブネット は、後で FSx ONTAPファイル システムを作成するために使用されます) 右下にあるオレンジ色のボタン ノートブック インスタンスの作成 をクリックします。

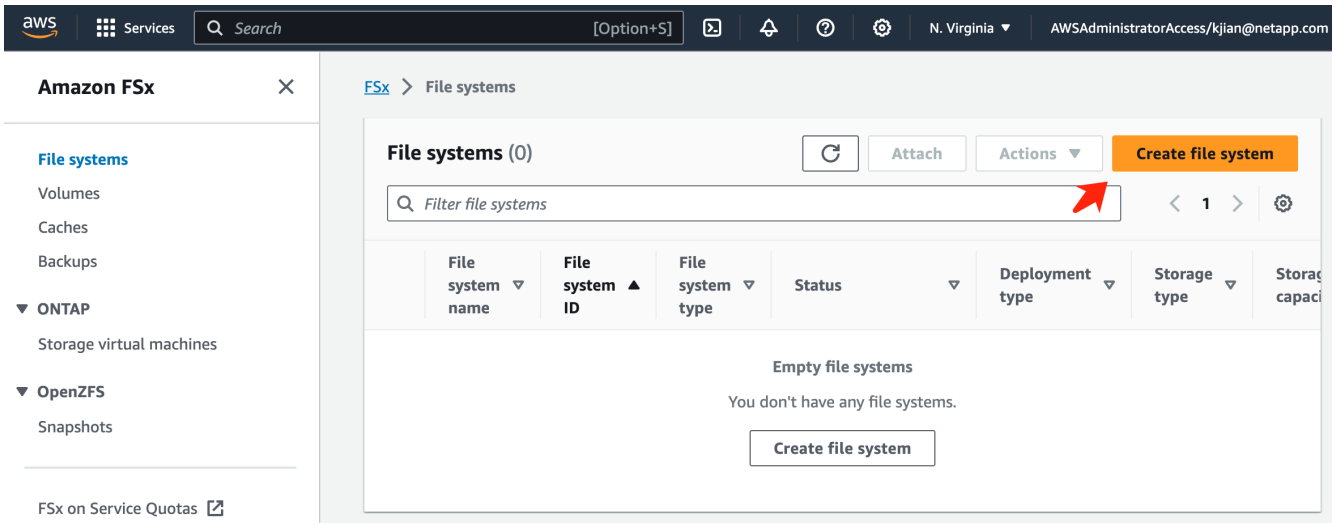


FSx ONTAPファイルシステムを作成する

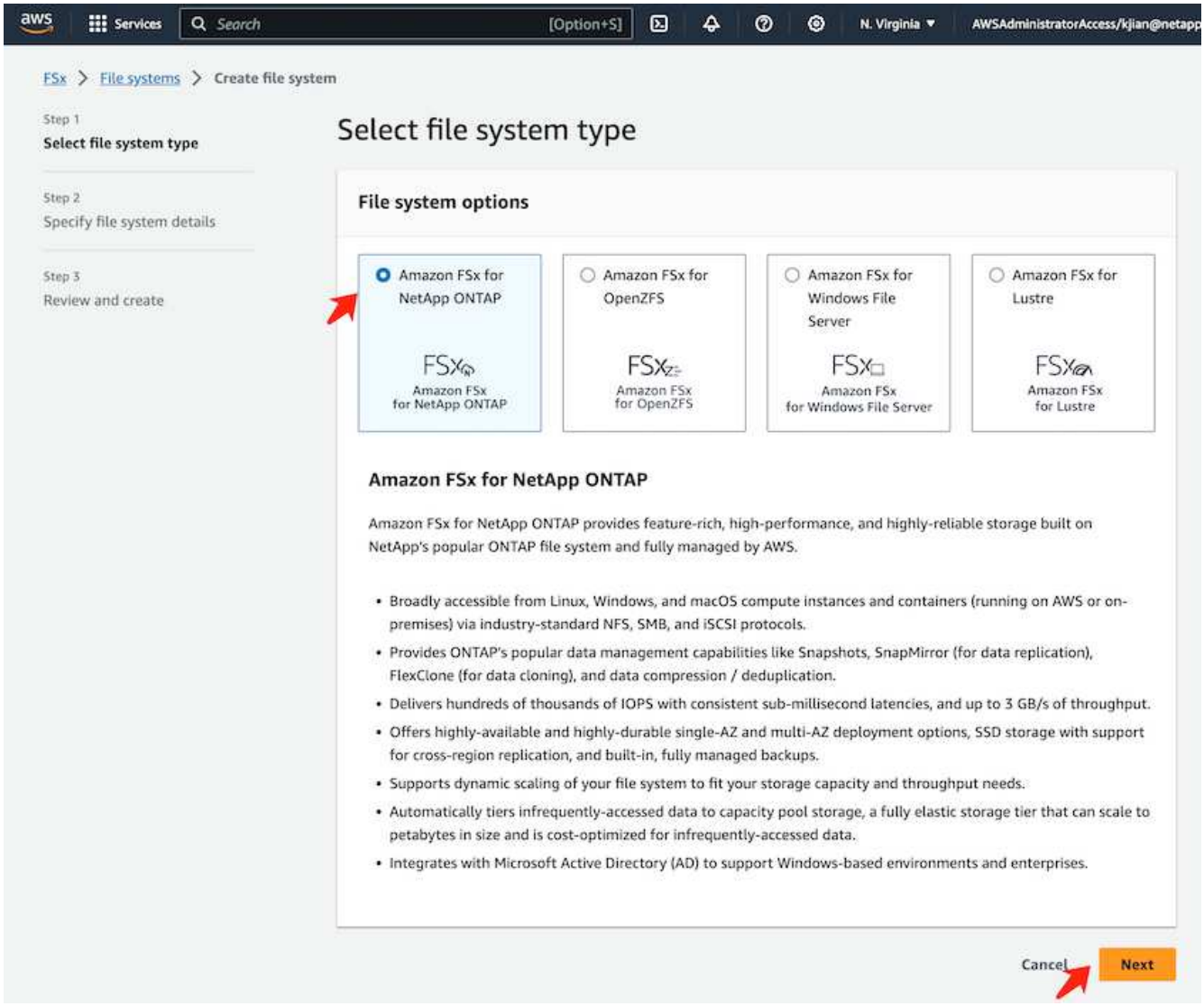
1. AWS コンソールを開きます。検索パネルで「Fsx」を検索し、サービス **FSx** をクリックします。



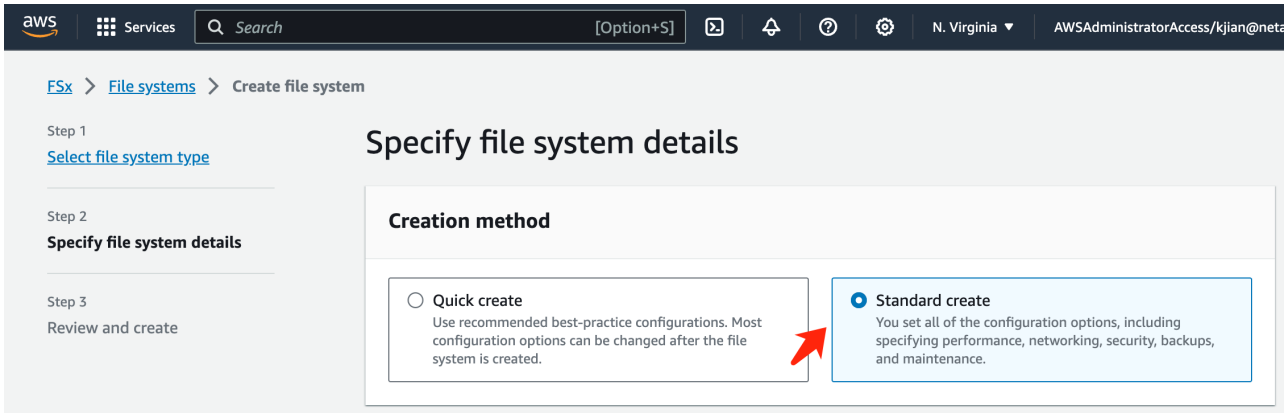
2. *ファイルシステムの作成*をクリックします。



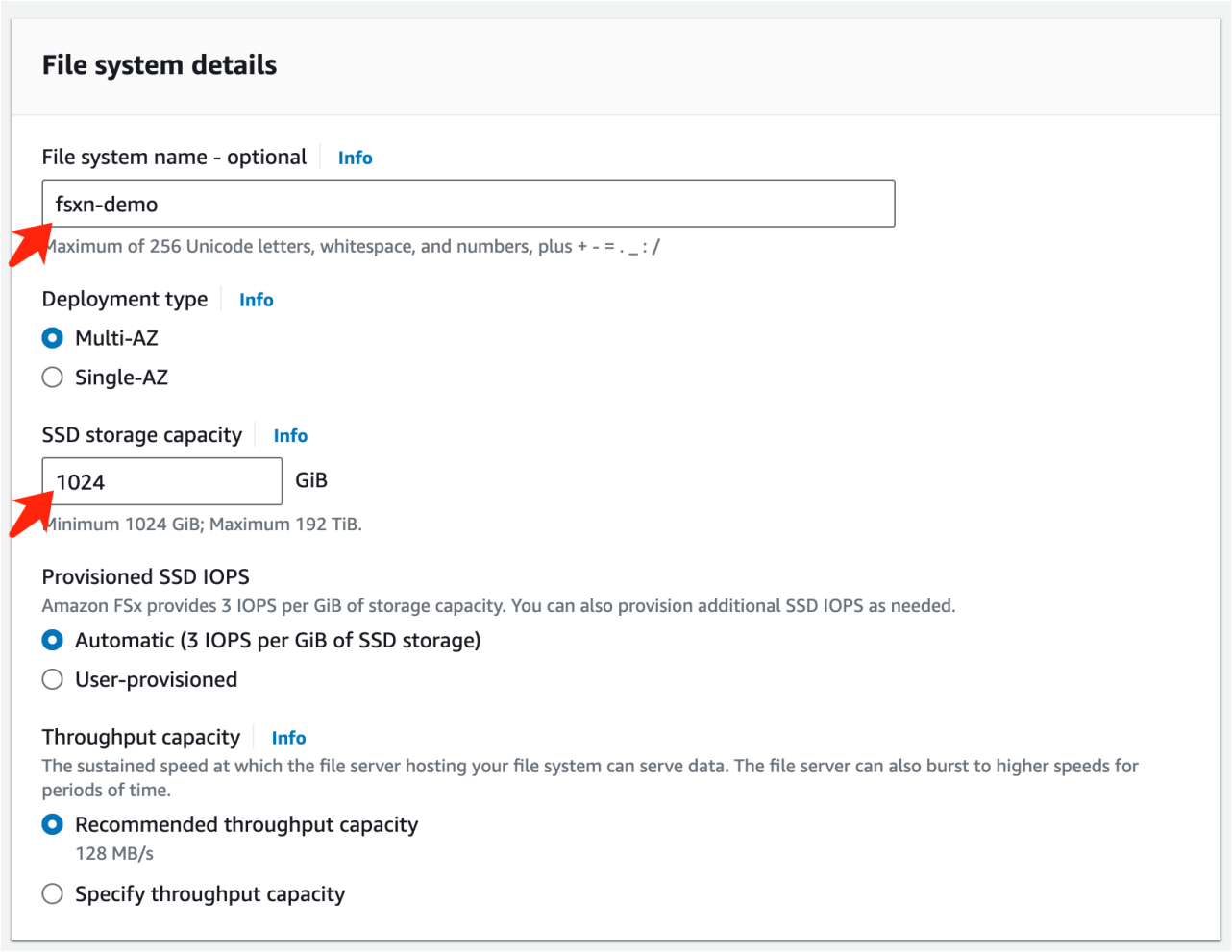
3. 最初のカード **FSx ONTAP** を選択し、次へ をクリックします。



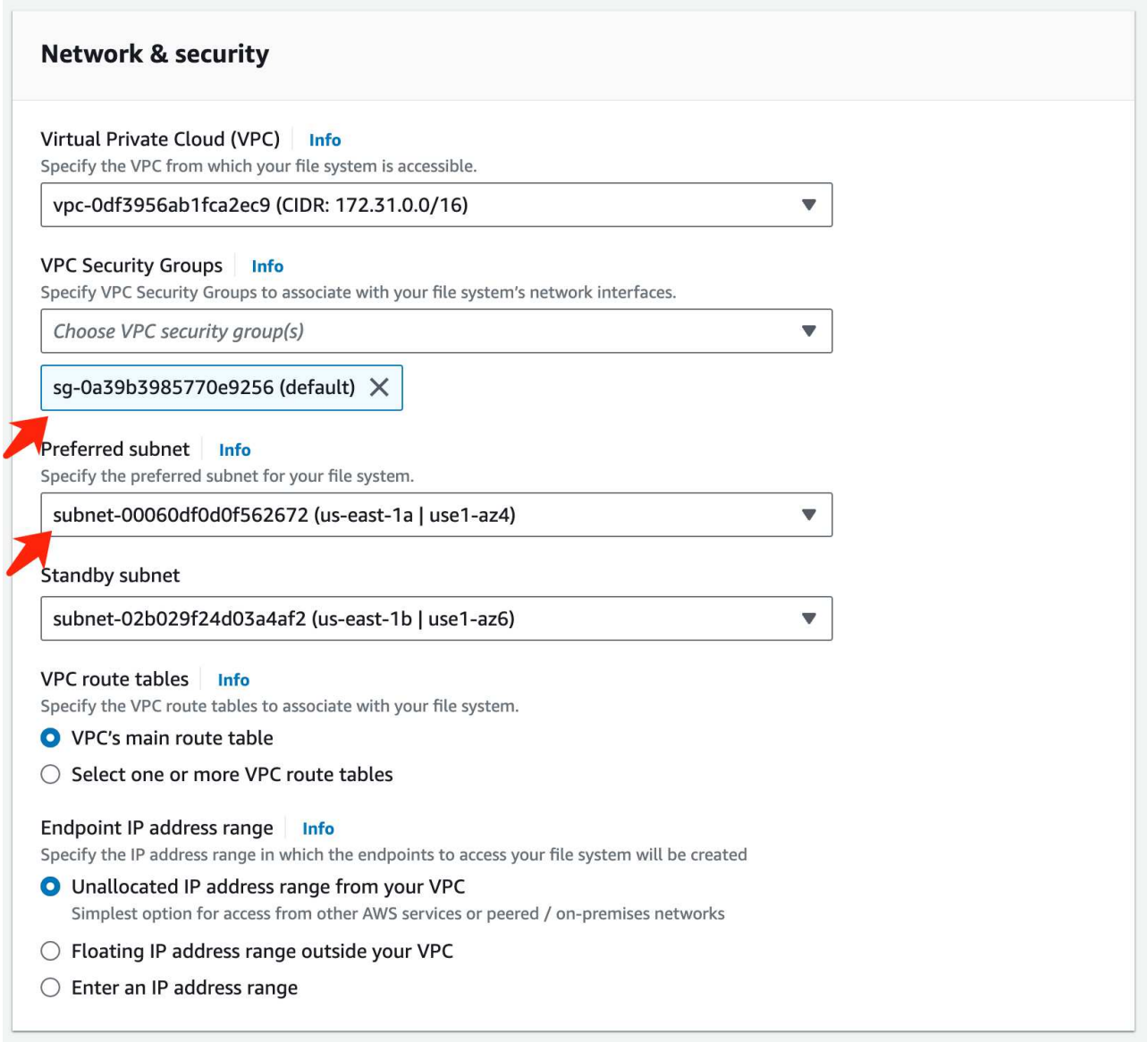
4. 詳細設定ページで。
   a. *標準作成*オプションを選択します。



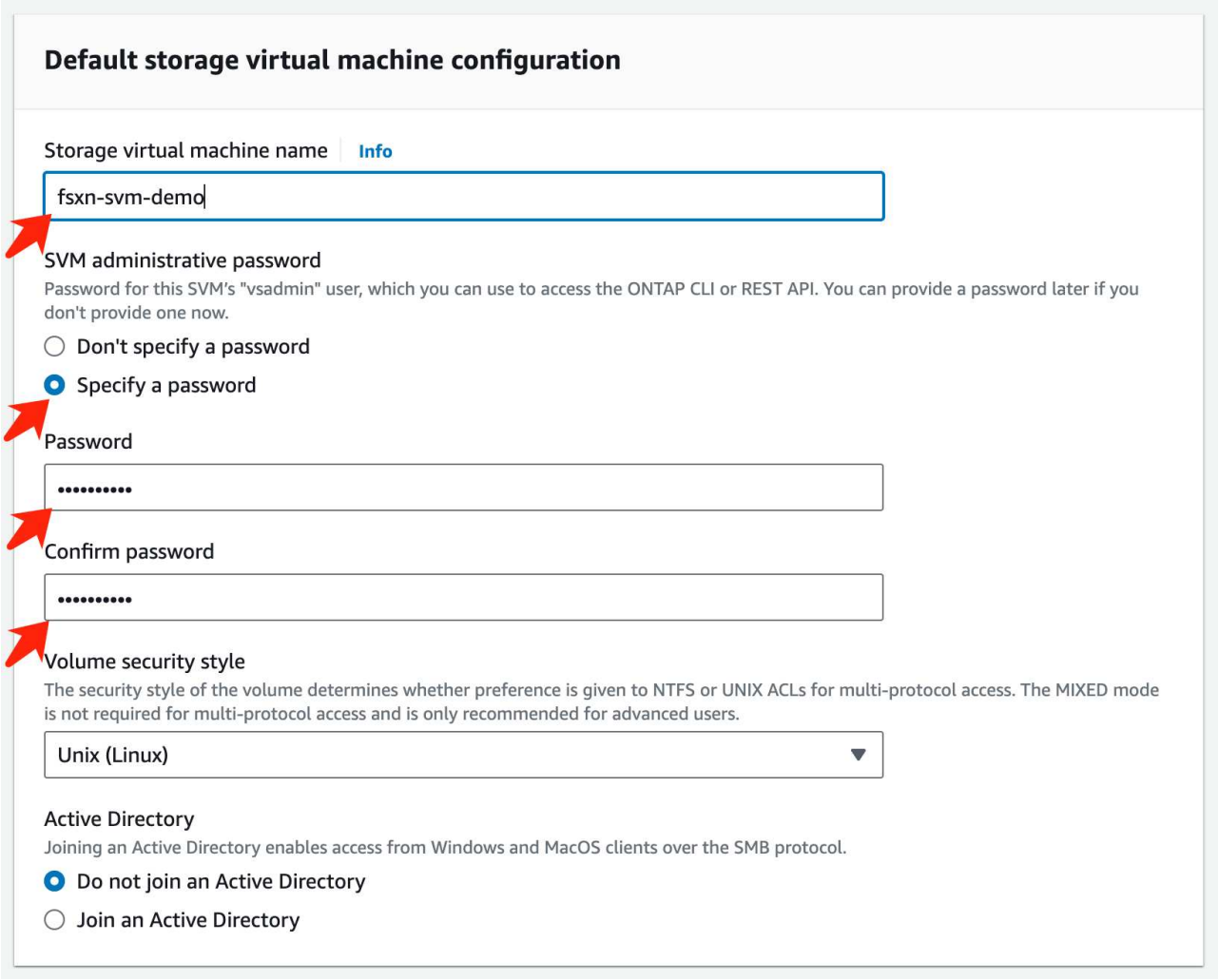
   b. *ファイルシステム名*と*SSDストレージ容量*を入力します。



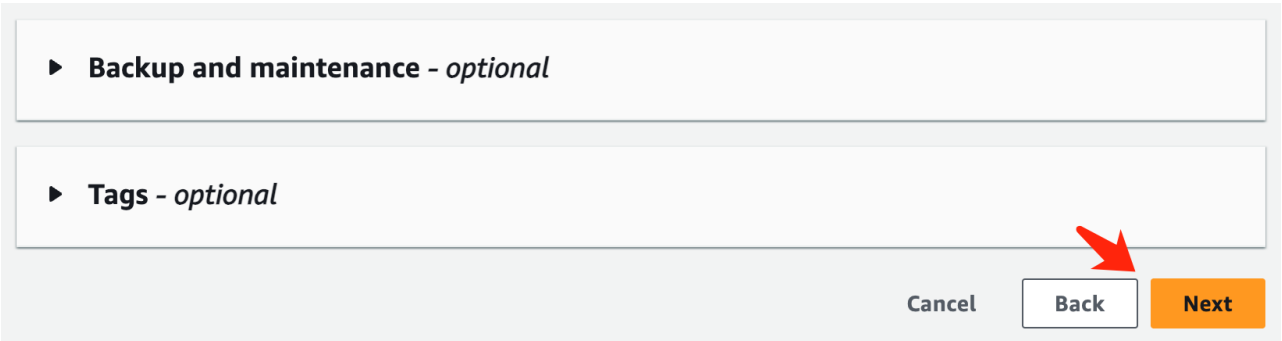
c. **SageMaker Notebook** インスタンスと同じ **VPC** と サブネット を使用するようにしてください。



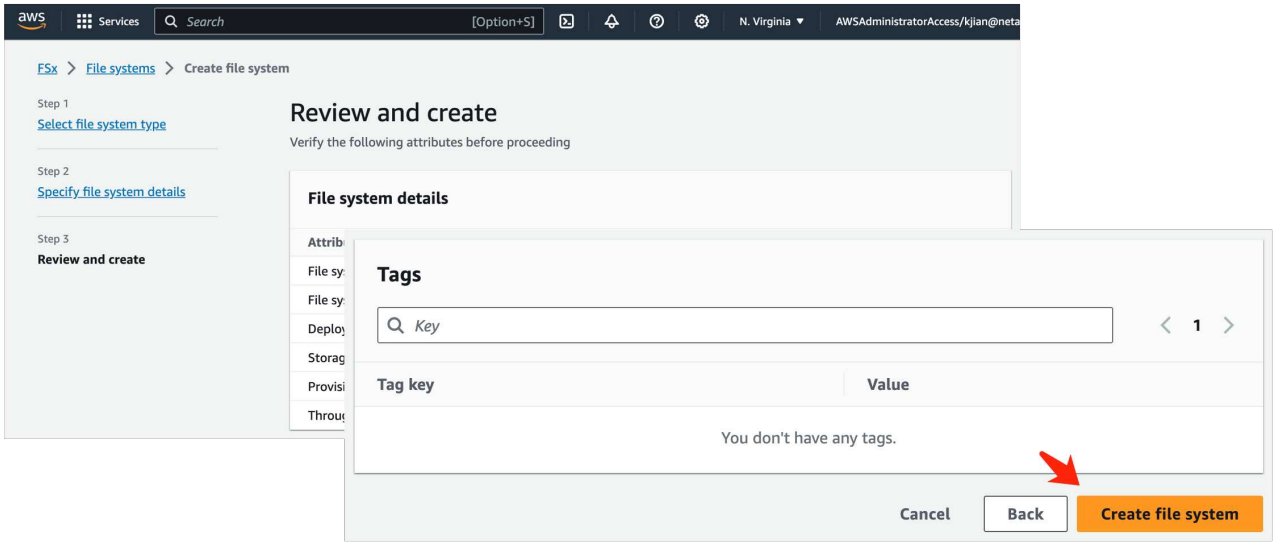
d. ストレージ仮想マシン 名を入力し、SVM (ストレージ仮想マシン) の パスワードを指定 します。



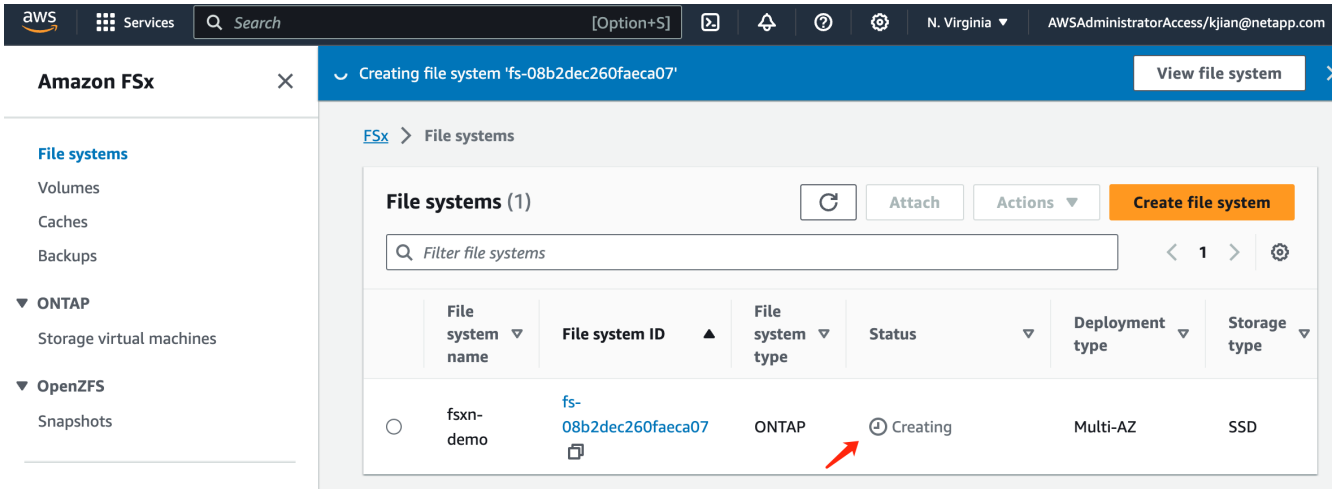
e. 他のエントリはデフォルトのままにして、右下にあるオレンジ色のボタン「次へ」をクリックします。



f. レビュー ページの右下にあるオレンジ色のボタン * ファイル システムの作成 * をクリックします。



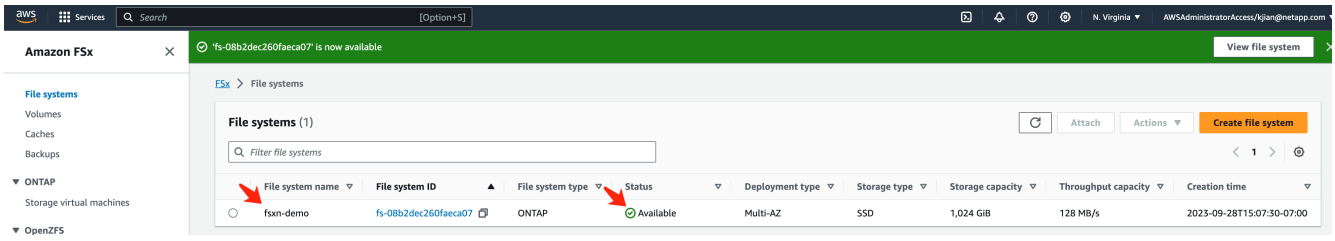
5. FSx ファイル システムの起動には約 **20** ～ **40** 分 かかる場合があります。



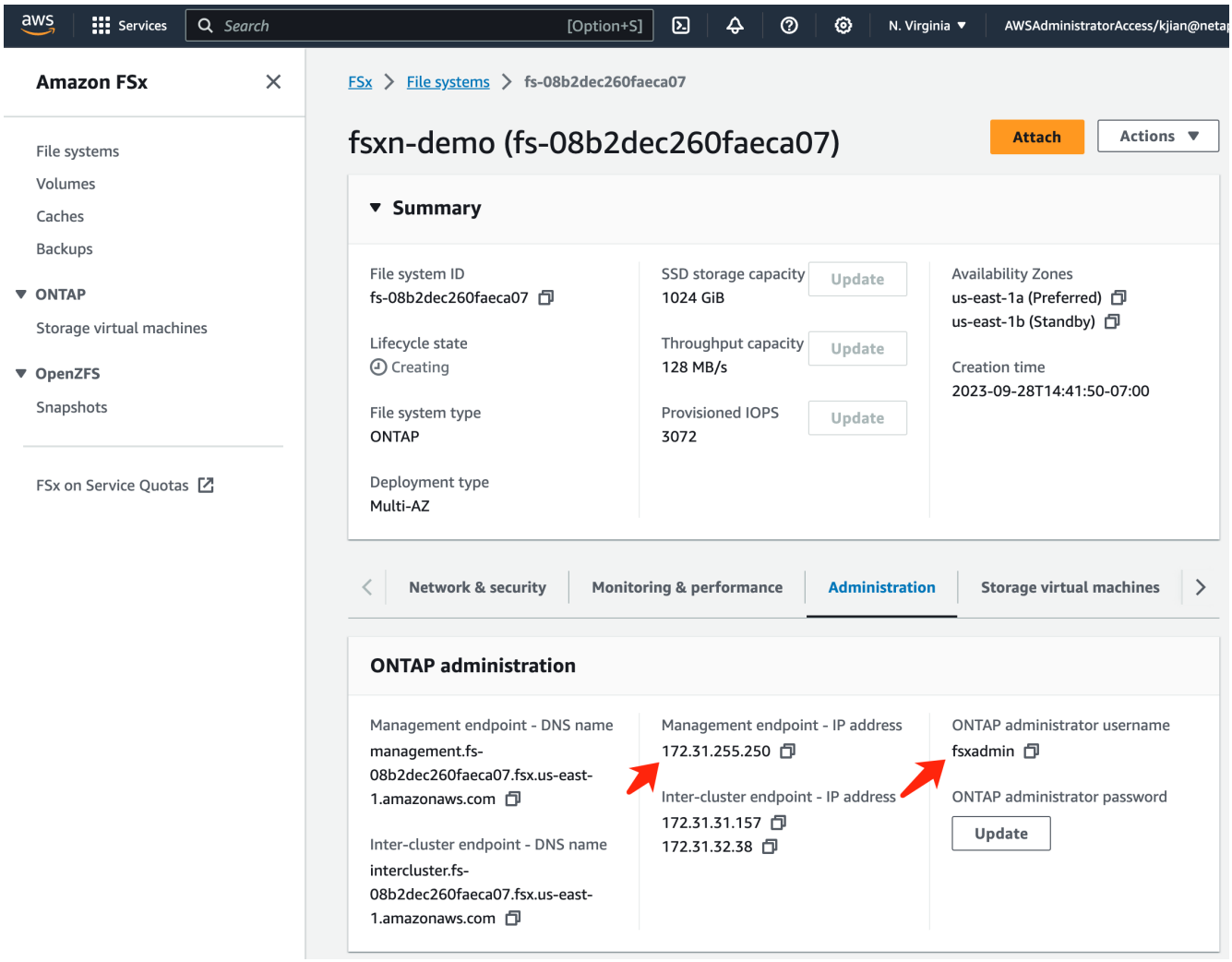
サーバー構成

ONTAP構成

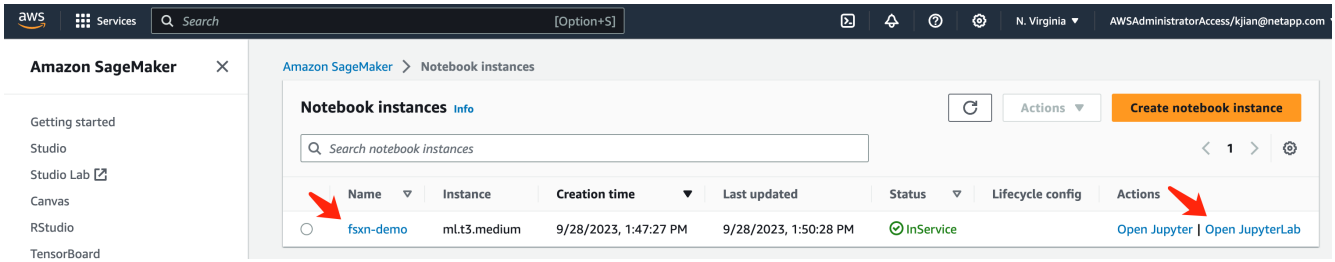
1. 作成された FSx ファイル システムを開きます。ステータスが「利用可能」であることを確認してください。



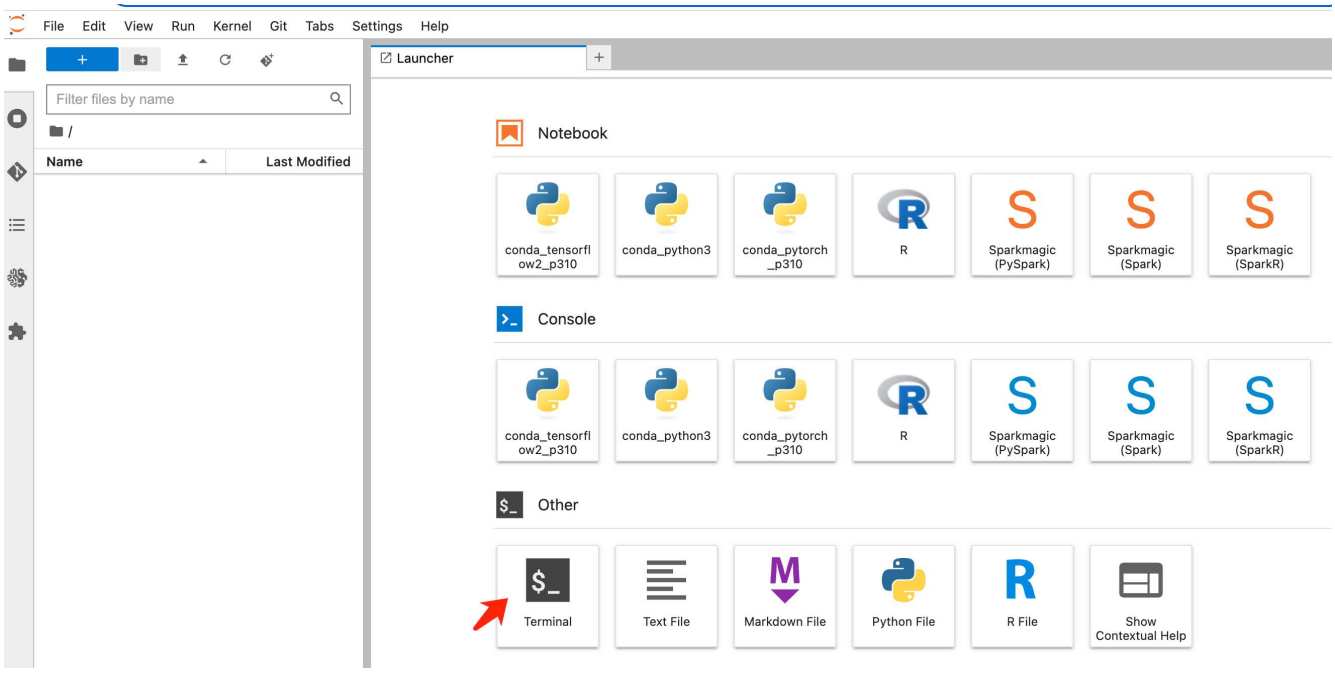
2. 管理*タブを選択し、*管理エンドポイント **- IP** アドレス*と ONTAP管理者ユーザー名*を保持します。



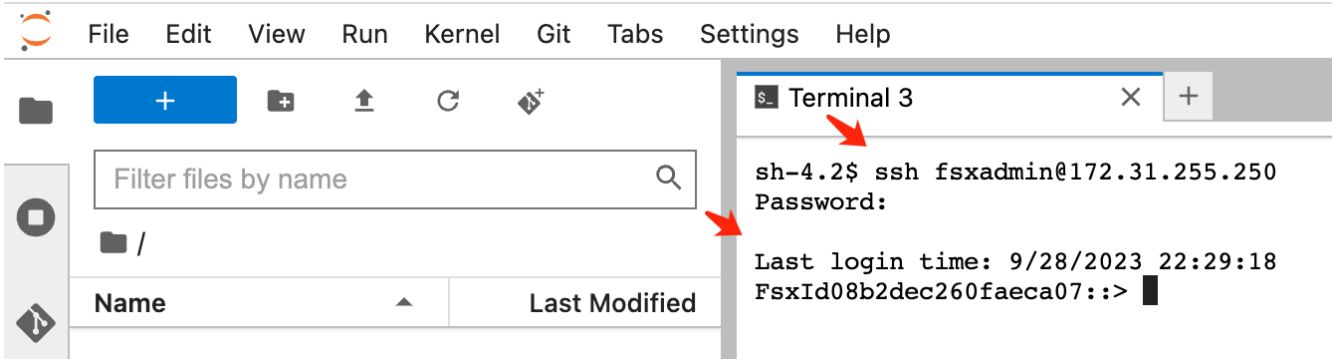
3. 作成された*SageMaker Notebookインスタンス*を開き、*JupyterLabを開く*をクリックします。



4. Jupyter Lab ページで、新しい ターミナル を開きます。



5. FSx ONTAPファイル システムにログインするには、ssh コマンド ssh <admin user name>@< ONTAP server IP> を入力します。 (ユーザー名とIPアドレスは手順2で取得します) *ストレージ仮想マシン*作成時に使用したパスワードを使用してください。



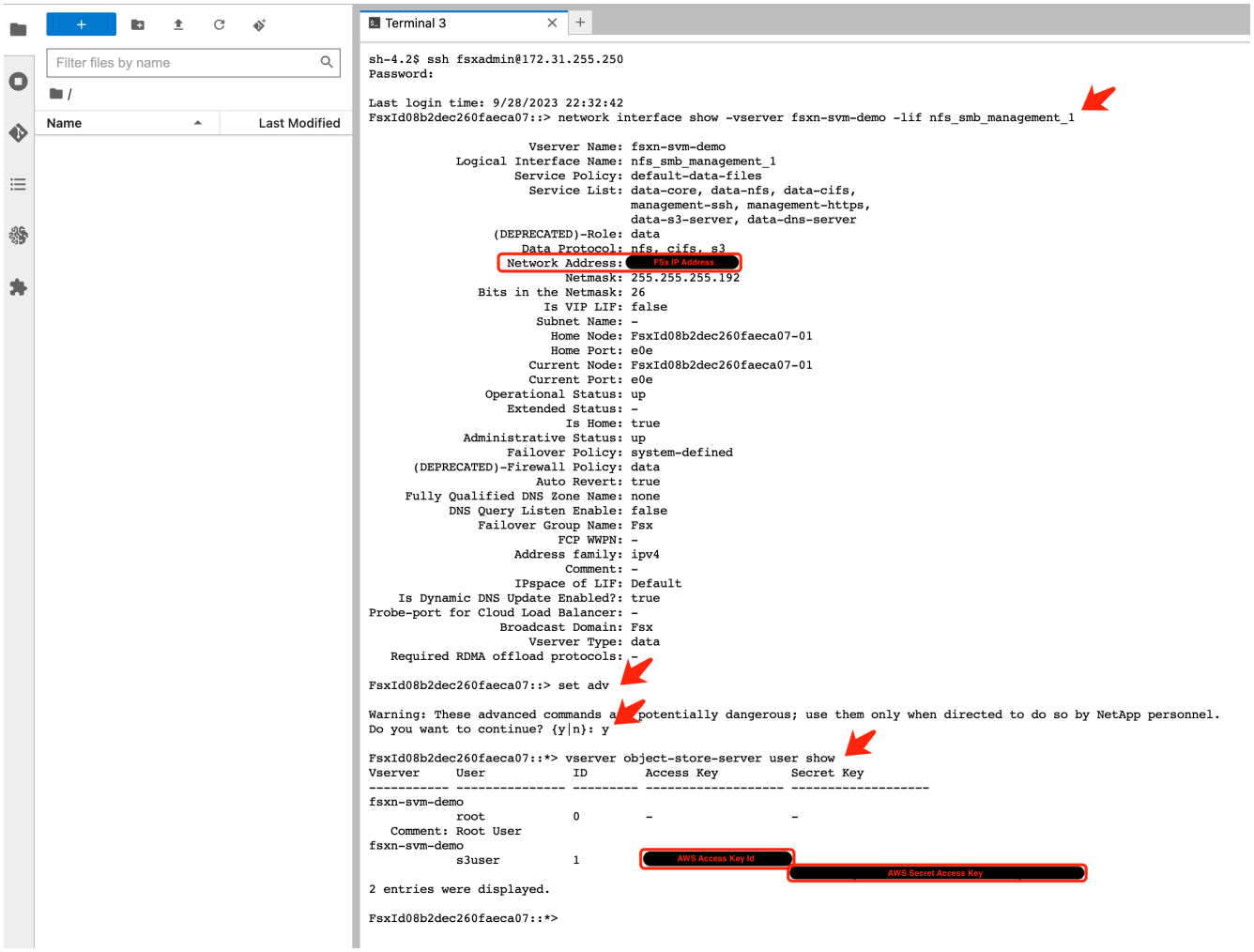
6. 次の順序でコマンドを実行します。 **FSx ONTAP**プライベート **S3** バケット名 の名前として **fsxn-ontap** を使用します。 **-vserver** 引数には ストレージ仮想マシン名 を使用してください。

```
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. 以下のコマンドを実行して、FSx ONTAPプライベート S3 のエンドポイント IP と資格情報を取得します。

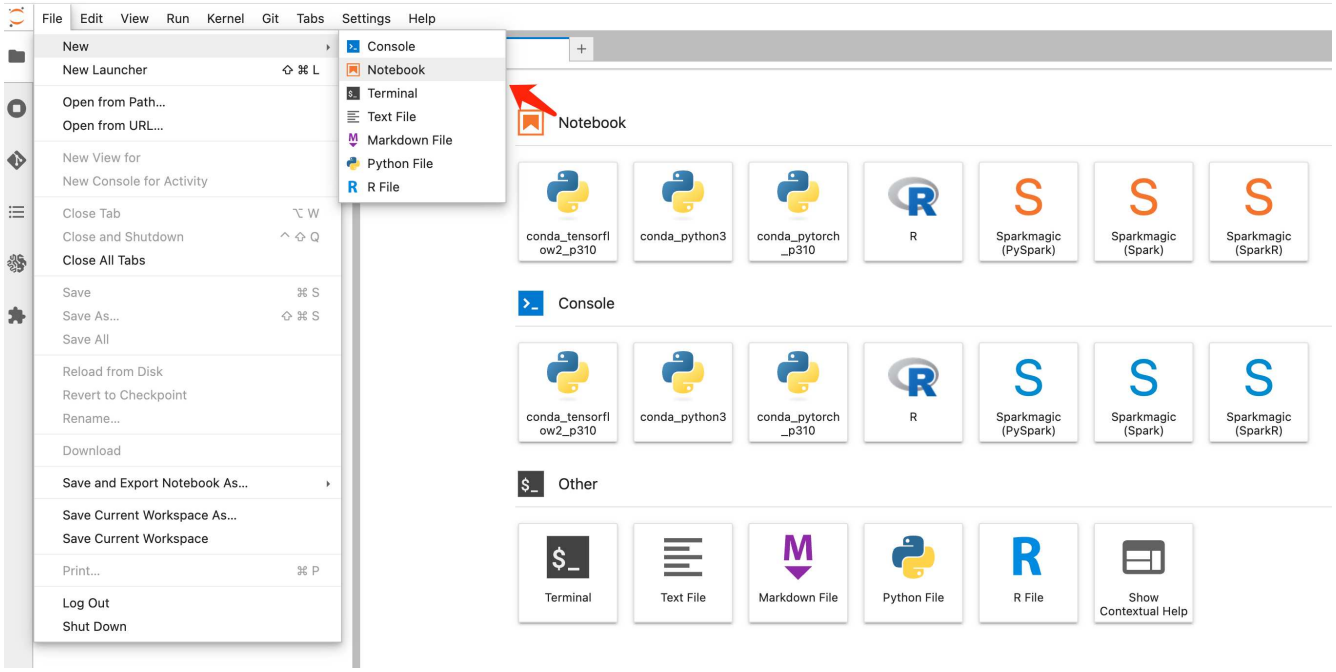
```
network interface show -vserver fsxn-svm-demo -lif nfs_smb_management_1

set adv

vserver object-store-server user show
```

8. 将来使用するためにエンドポイントの IP と資格情報を保持します。



クライアント構成

1. SageMaker Notebook インスタンスで、新しい Jupyter ノートブックを作成します。



2. 以下のコードを回避策として使用して、FSx ONTAPプライベート S3 バケットにファイルをアップロードします。包括的なコード例については、このノートブックを参照してください。"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']
```

これで、FSx ONTAPと SageMaker インスタンスの統合は完了です。

便利なデバッグチェックリスト

- SageMaker Notebook インスタンスと FSx ONTAPファイルシステムが同じ VPC にあることを確認します。

- 権限レベルを **dev** に設定するには、 ONTAPで **set dev** コマンドを実行することを忘れないでください。

よくある質問（**2023**年**9**月**27**日現在）

Q: FSx ONTAPにファイルをアップロードするときに、「**CreateMultipartUpload** 操作の呼び出し時にエラーが発生しました **(NotImplemented):** 要求した **s3** コマンドは実装されていません」というエラーが表示されるのはなぜですか?

A: プライベート S3 バケットとして、FSx ONTAP は最大 100 MB のファイルのアップロードをサポートします。 S3 プロトコルを使用する場合、100 MB を超えるファイルは 100 MB のチャンクに分割され、「CreateMultipartUpload」関数が呼び出されます。ただし、FSx ONTAPプライベート S3 の現在の実装では、この機能はサポートされていません。

Q: FSx ONTAPにファイルをアップロードするときに、「**PutObject** 操作の呼び出し時にエラーが発生しました **(AccessDenied)**。アクセスが拒否されました」というエラーが表示されるのはなぜですか?

A: SageMaker Notebook インスタンスから FSx ONTAPプライベート S3 バケットにアクセスするには、AWS 認証情報を FSx ONTAP認証情報に切り替えます。ただし、インスタンスに書き込み権限を付与するには、バケットをマウントし、「chmod」シェル コマンドを実行して権限を変更するという回避策が必要です。

Q: FSx ONTAPプライベート S3 バケットを他の SageMaker ML サービスと統合するにはどうすればよいですか?

A: 残念ながら、SageMaker サービス SDK では、プライベート S3 バケットのエンドポイントを指定する方法は提供されていません。その結果、FSx ONTAP S3 は、Sagemaker Data Wrangler、Sagemaker Clarify 、Sagemaker Glue、Sagemaker Athena、Sagemaker AutoML などの SageMaker サービスと互換性がありません。

パート **2 - SageMaker** でのモデルトレーニングのデータソースとして **AWS Amazon FSx for NetApp ONTAP (FSx ONTAP)** を活用する

この記事は、Amazon FSx for NetApp ONTAP (FSx ONTAP) を使用して SageMaker で PyTorch モデルをトレーニングする方法について、具体的にはタイヤ品質分類プロジェクト向けのチュートリアルです。

はじめに

このチュートリアルでは、コンピューター ビジョン分類プロジェクトの実用的な例を示し、SageMaker 環境内で FSx ONTAP をデータ ソースとして利用する ML モデルの構築に関する実践的な体験を提供します。このプロジェクトは、ディープラーニング フレームワークである PyTorch を使用して、タイヤ画像に基づいてタイヤの品質を分類することに重点を置いています。 Amazon SageMaker のデータソースとして FSx ONTAPを使用した機械学習モデルの開発に重点を置いています。

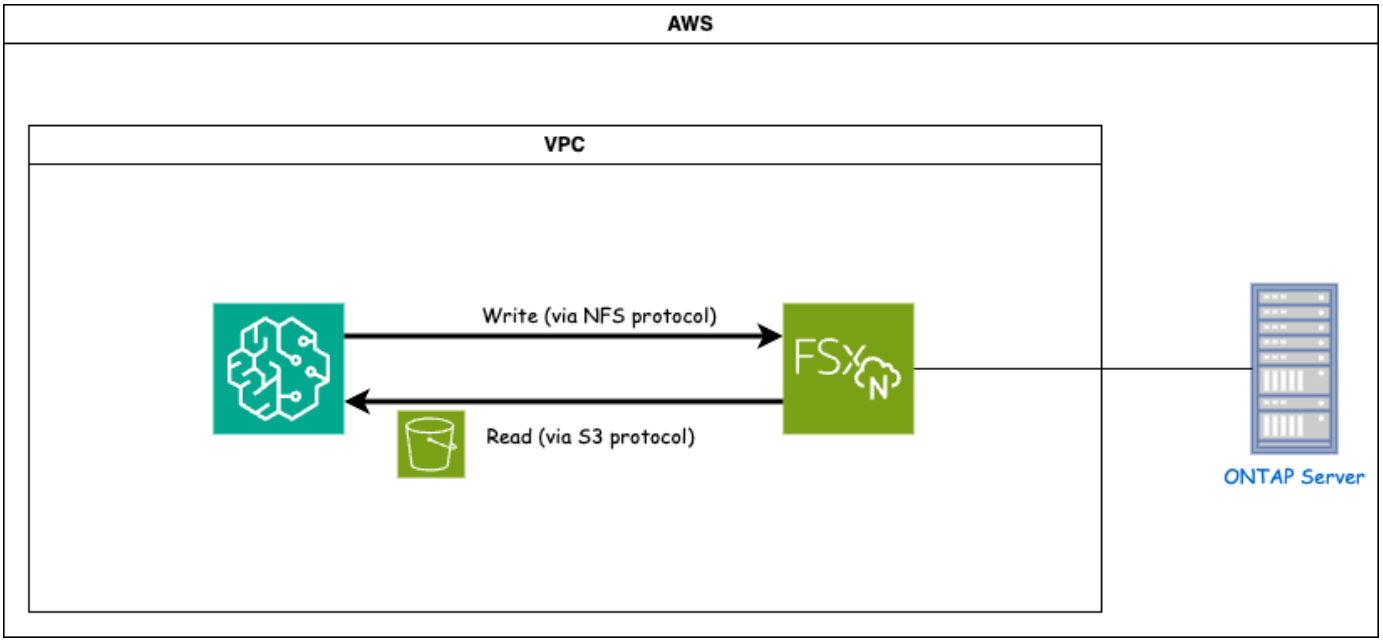
FSx ONTAPとは

Amazon FSx ONTAP は、AWS が提供する完全に管理されたストレージソリューションです。 NetApp のONTAPファイル システムを活用して、信頼性が高く高性能なストレージを提供します。 NFS、SMB 、iSCSI などのプロトコルをサポートしているため、さまざまなコンピューティング インスタンスやコンテナーからのシームレスなアクセスが可能になります。このサービスは、優れたパフォーマンスを提供し、高速

かつ効率的なデータ操作を保証するように設計されています。また、高い可用性と耐久性も提供し、データのアクセスと保護が維持されます。さらに、 Amazon FSx ONTAPのストレージ容量はスケーラブルなので、ニーズに応じて簡単に調整できます。

前提条件

ネットワーク環境



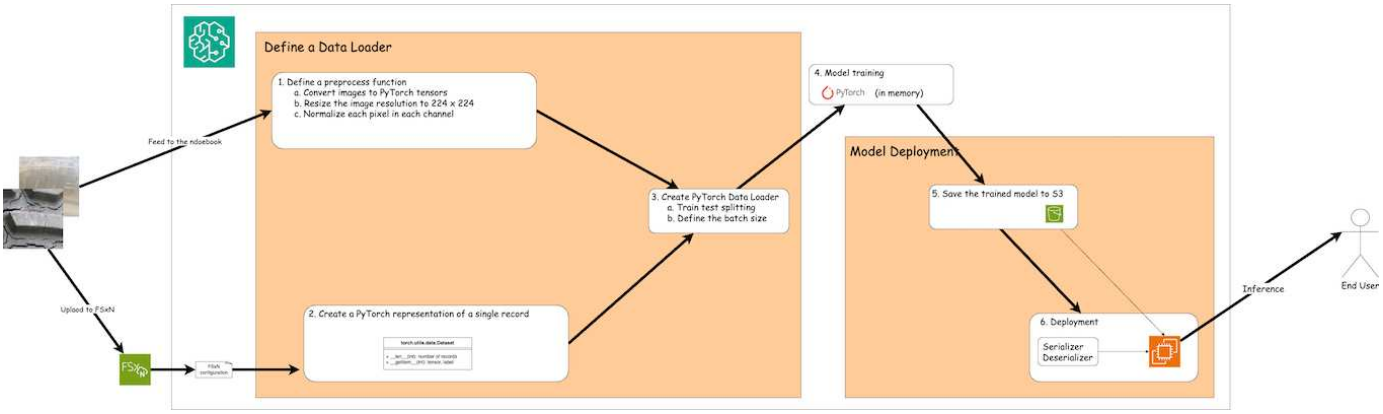
FSx ONTAP (Amazon FSx ONTAP) は AWS ストレージサービスです。これには、 NetApp ONTAPシステム上で実行されるファイルシステムと、それに接続する AWS 管理のシステム仮想マシン (SVM) が含まれます。提供された図では、AWS によって管理されるNetApp ONTAPサーバーは VPC の外部に配置されています。SVM は SageMaker とNetApp ONTAPシステム間の仲介役として機能し、SageMaker から操作要求を受信して、基盤となるストレージに転送します。 FSx ONTAPにアクセスするには、SageMaker を FSx ONTAPデプロイメントと同じ VPC 内に配置する必要があります。この構成により、SageMaker と FSx ONTAP間の通信とデータ アクセスが保証されます。

データ アクセス

実際のシナリオでは、データ サイエンティストは通常、FSx ONTAPに保存されている既存のデータを活用して機械学習モデルを構築します。ただし、デモンストレーションの目的では、FSx ONTAPファイル システムは作成後最初は空であるため、トレーニング データを手動でアップロードする必要があります。これは、FSx ONTAP をボリュームとして SageMaker にマウントすることで実現できます。ファイルシステムが正常にマウントされると、マウントされた場所にデータセットをアップロードして、SageMaker 環境内でモデルをトレーニングするためにアクセスできるようになります。このアプローチにより、モデルの開発とトレーニングに SageMaker を使用しながら、FSx ONTAPのストレージ容量と機能を活用できます。

データ読み取りプロセスでは、FSx ONTAP をプライベート S3 バケットとして構成する必要があります。詳細な設定手順については、以下を参照してください。"パート 1 - Amazon FSx for NetApp ONTAP (FSx ONTAP) をプライベート S3 バケットとして AWS SageMaker に統合する"

統合の概要



FSx ONTAPのトレーニング データを使用して SageMaker でディープ ラーニング モデルを構築するワークフローは、データ ローダーの定義、モデルのトレーニング、およびデプロイメントという 3 つの主なステップに要約できます。大まかに言えば、これらのステップは MLOps パイプラインの基盤を形成します。ただし、包括的な実装のために、各ステップにはいくつかの詳細なサブステップが含まれます。これらのサブステップには、データの前処理、データセットの分割、モデルの構成、ハイパーパラメータの調整、モデルの評価、モデルの展開などのさまざまなタスクが含まれます。これらの手順により、SageMaker 環境内で FSx ONTAPのトレーニング データを使用してディープ ラーニング モデルを構築および展開するための徹底的かつ効果的なプロセスが保証されます。

ステップバイステップの統合

データ**Loader**

PyTorch ディープラーニング ネットワークをデータでトレーニングするために、データのフィードを容易にするデータ ローダーが作成されます。データ ローダーはバッチ サイズを定義するだけでなく、バッチ内の各レコードを読み取って前処理する手順も決定します。データ ローダーを構成することで、データの処理をバッチで処理し、ディープ ラーニング ネットワークのトレーニングが可能になります。

データ ローダーは 3 つの部分で構成されます。

前処理関数

```
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]
    )
])
```

上記のコード スニペットは、**torchvision.transforms** モジュールを使用した画像前処理変換の定義を示しています。このチュートリアルでは、一連の変換を適用するための前処理オブジェクトが作成されます。まず、**ToTensor()** 変換により、画像をテンソル表現に変換します。その後、**Resize 224,224** 変換により、画像のサイズが 224x224 ピクセルの固定サイズに変更されます。最後に、**Normalize()** 変換は、各チャネルに

沿って平均を減算し、標準偏差で割ることでテンソル値を正規化します。正規化に使用される平均値と標準偏差値は、事前トレーニング済みのニューラル ネットワーク モデルでよく使用されます。全体として、このコードは、画像データをテンソルに変換し、サイズを変更し、ピクセル値を正規化することで、画像データをさらに処理したり、事前トレーニング済みモデルに入力したりできるように準備します。

PyTorch データセットクラス

```
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
```

このクラスは、データセット内のレコードの合計数を取得する機能を提供し、各レコードのデータを読み取るメソッドを定義します。 ***getitem*** 関数内で、コードは boto3 S3 バケット オブジェクトを使用して、FSx ONTAPからバイナリ データを取得します。 FSx ONTAPからデータにアクセスするためのコード スタイルは、Amazon S3 からデータを読み取る場合と似ています。以降の説明では、プライベート S3 オブジェクト **bucket** の作成プロセスについて詳しく説明します。

プライベート**S3**リポジトリとしての**FSx ONTAP**

```
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 ---
```

SageMaker で FSx ONTAPからデータを読み取るために、S3 プロトコルを使用して FSx ONTAPストレージを指すハンドラーが作成されます。これにより、FSx ONTAP をプライベート S3 バケットとして扱うことができます。ハンドラーの設定には、FSx ONTAP SVM の IP アドレス、バケット名、および必要な資格情報の指定が含まれます。これらの構成項目の取得に関する詳細な説明については、次の文書を参照してください。"パート 1 - Amazon FSx for NetApp ONTAP (FSx ONTAP) をプライベート S3 バケットとして AWS SageMaker に統合する" 。

上記の例では、バケット オブジェクトを使用して PyTorch データセット オブジェクトをインスタンス化しています。データセット オブジェクトについては、後続のセクションでさらに詳しく説明します。

PyTorchデータ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)
```

提供されている例では、バッチ サイズ 64 が指定されており、各バッチに 64 個のレコードが含まれることを示しています。 PyTorch **Dataset** クラス、前処理関数、およびトレーニング バッチ サイズを組み合わせることで、トレーニング用のデータ ローダーを取得します。このデータ ローダーは、トレーニング フェーズ中にデータセットをバッチで反復処理するプロセスを容易にします。

モデルトレーニング

```
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')
```

このコードは標準の PyTorch トレーニング プロセスを実装します。これは、畳み込み層と線形層を使用してタイヤの品質を分類する **TyreQualityClassifier** と呼ばれるニューラル ネットワーク モデルを定義します。トレーニング ループはデータ バッチを反復処理し、損失を計算し、バックプロパゲーションと最適化を使用してモデルのパラメータを更新します。さらに、監視の目的で現在の時刻、エポック、バッチ、損失を出力します。

モデルの展開

導入

```
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())
```

SageMaker ではモデルをデプロイするために S3 に保存する必要があるため、コードは PyTorch モデルを **Amazon S3** に保存します。モデルを **Amazon S3** にアップロードすると、SageMaker からアクセスできるようになり、デプロイされたモデルのデプロイと推論が可能になります。

```
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
)
```

このコードは、SageMaker への PyTorch モデルのデプロイを容易にします。これは、入力データを PyTorch テンソルとして前処理してシリアル化するカスタム シリアライザー **TyreQualitySerializer** を定義します。**TyreQualityPredictor** クラスは、定義されたシリアライザーと **JSONDeserializer** を利用するカスタム予測子です。このコードは、モデルの S3 の場所、IAM ロール、フレームワークのバージョン、推論のエントリ ポイントを指定するための **PyTorchModel** オブジェクトも作成します。コードはタイムスタンプを生成し、モ

デルとタイムスタンプに基づいてエンドポイント名を構築します。最後に、インスタンス数、インスタンスタイプ、生成されたエンドポイント名を指定して、deploy メソッドを使用してモデルがデプロイされます。これにより、PyTorch モデルをデプロイし、SageMaker で推論にアクセスできるようになります。

推論

```
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)
```

これは、デプロイされたエンドポイントを使用して推論を行う例です。

パート 3 - 簡素化された MLOps パイプラインの構築 (CI/CT/CD)

この記事では、自動化されたモデルの再トレーニング、デプロイ、コストの最適化に焦点を当て、AWS サービスを使用して MLOps パイプラインを構築するためのガイドを提供します。

はじめに

このチュートリアルでは、さまざまな AWS サービスを活用して、継続的インテグレーション (CI)、継続的トレーニング (CT)、継続的デプロイメント (CD) を含むシンプルな MLOps パイプラインを構築する方法を学習します。従来の DevOps パイプラインとは異なり、MLOps では運用サイクルを完了するために追加の考慮が必要です。このチュートリアルに従うことで、CT を MLOps ループに組み込み、モデルの継続的なトレーニングと推論のシームレスな展開を可能にする方法について理解を深めることができます。このチュートリアルでは、AWS サービスを利用してこのエンドツーエンドの MLOps パイプラインを確立するプロセスについて説明します。

マニフェスト

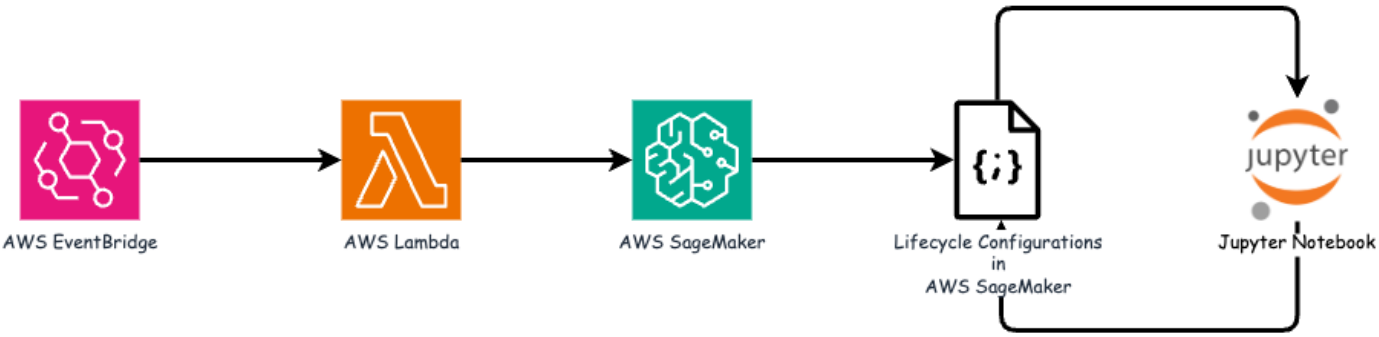
機能	Name	コメント
データストレージ	AWS FSx ONTAP	。"パート 1 - Amazon FSx for NetApp ONTAP (FSx ONTAP) をプライベート S3 バケットとして AWS SageMaker に統合する" 。
データサイエンスIDE	AWS セージメーカー	このチュートリアルは、"パート 2 - SageMaker でのモデルトレーニング用のデータソースとしてAmazon FSx for NetApp ONTAP (FSx ONTAP) を活用する" 。

機能	Name	コメント
MLOpsパイプラインをトリガーする関数	AWS Lambda関数	-
Cronジョブトリガー	AWS イベントブリッジ	-
ディープラーニングフレームワーク	パイトーチ	-
AWS Python SDK	ボト3	-
プログラミング言語	Python	バージョン3.10

前提条件

- 事前設定された FSx ONTAPファイル システム。このチュートリアルでは、トレーニング プロセスに FSx ONTAPに保存されているデータを利用します。
- 上記の FSx ONTAPファイルシステムと同じ VPC を共有するように設定された **SageMaker Notebook** インスタンス。
- **AWS Lambda** 関数 をトリガーする前に、**SageMaker Notebook** インスタンス が 停止 状態であることを確認してください。
- ディープ ニューラル ネットワークの計算に必要な GPU アクセラレーションを活用するには、**ml.g4dn.xlarge** インスタンス タイプが必要です。

アーキテクチャ



この MLOps パイプラインは、cron ジョブを利用してサーバーレス関数をトリガーし、ライフサイクルコールバック関数に登録された AWS サービスを実行する実用的な実装です。 **AWS EventBridge** は cron ジョブとして機能します。モデルの再トレーニングと再デプロイを担当する **AWS Lambda** 関数 を定期的に呼び出します。このプロセスでは、必要なタスクを実行するために **AWS SageMaker Notebook** インスタンスを起動する必要があります。

ステップバイステップの設定

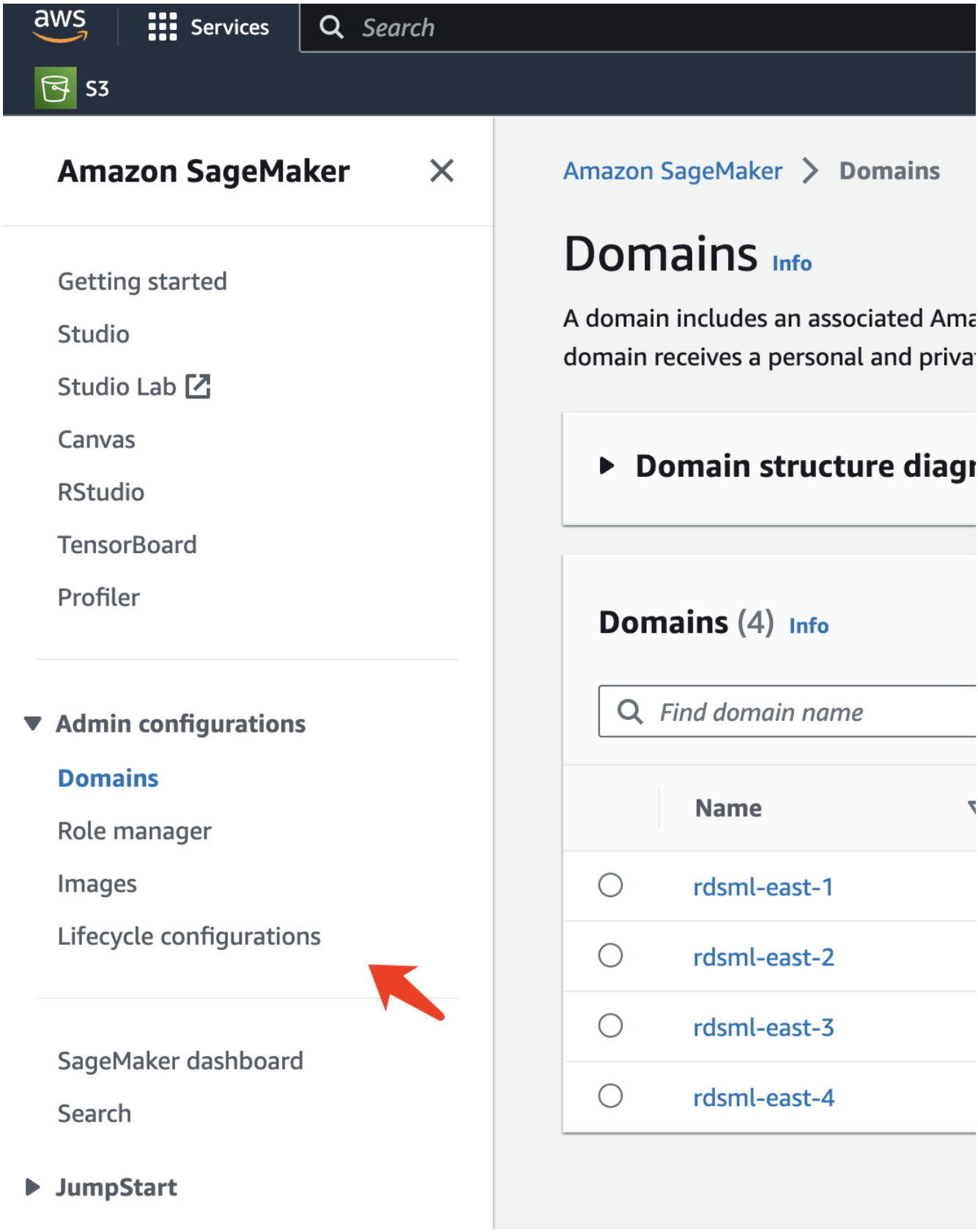
ライフサイクル構成

AWS SageMaker Notebook インスタンスのライフサイクルコールバック関数を設定するには、*ライフサイクル設定*を利用します。このサービスを使用すると、ノートブック インスタンスを起動するときに実行する必要なアクションを定義できます。具体的には、*ライフサイクル構成*内にシェル スクリプトを実装して、トレーニングとデプロイメントのプロセスが完了するとノートブック インスタンスを自動的にシャットダウンす

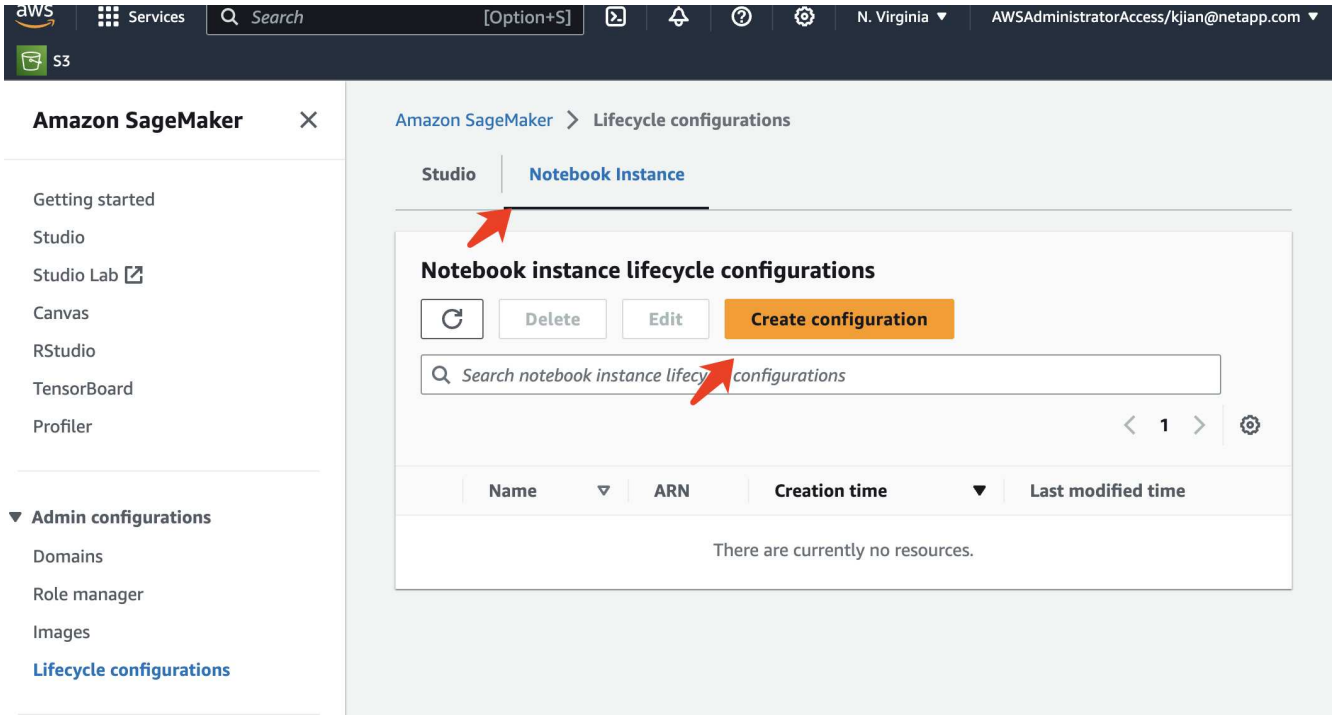
ることができます。 MLOps ではコストが主要な考慮事項の 1 つであるため、これは必須の構成です。

*ライフサイクル構成*の構成を事前に設定しておく必要があることに注意することが重要です。したがって、他の MLOps パイプラインのセットアップに進む前に、この側面の構成を優先することをお勧めします。

1. ライフサイクル構成を設定するには、**Sagemaker** パネルを開き、管理構成 セクションの ライフサイクル構成 に移動します。



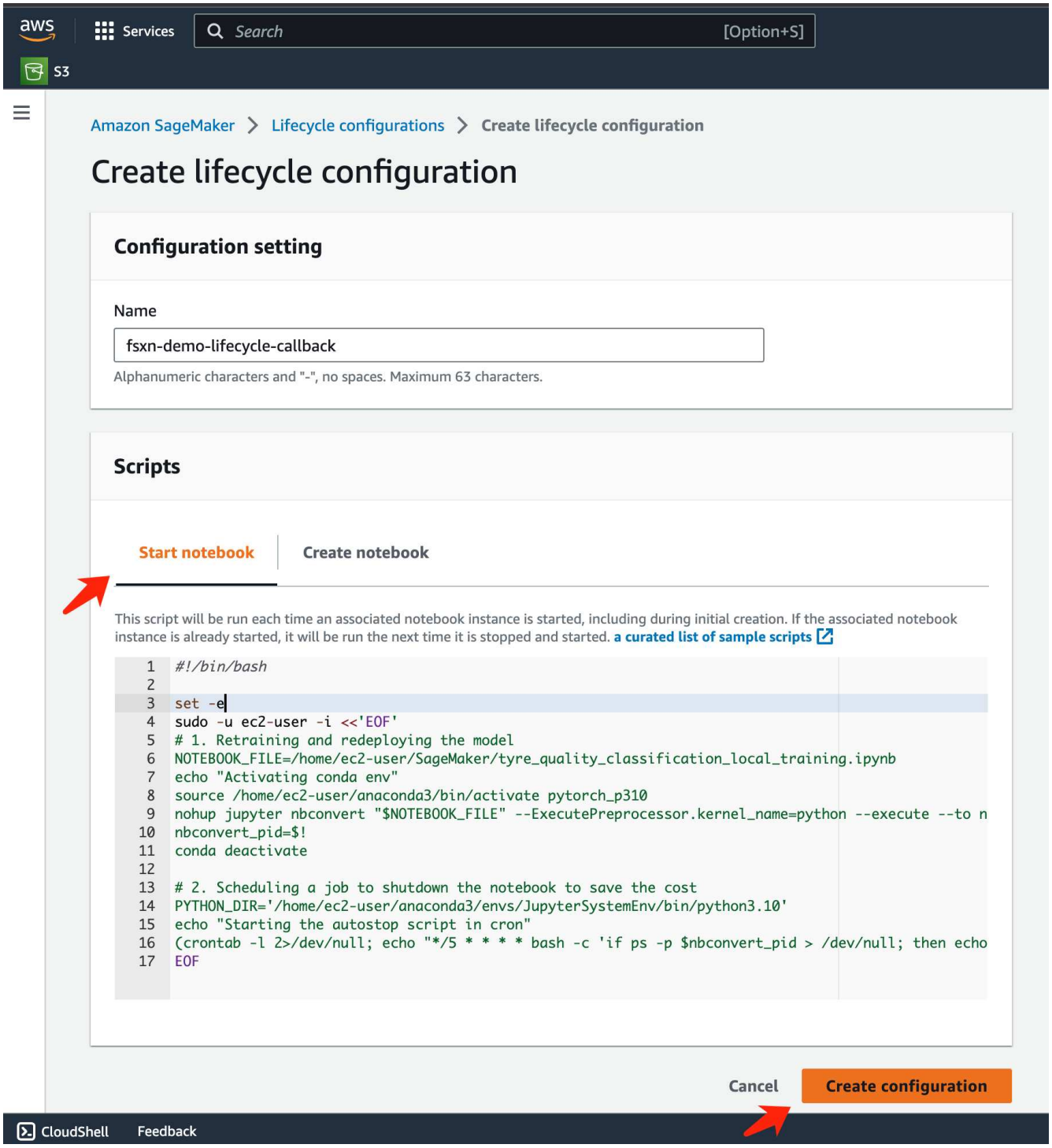
2. *ノートブックインスタンス*タブを選択し、*構成の作成*ボタンをクリックします。



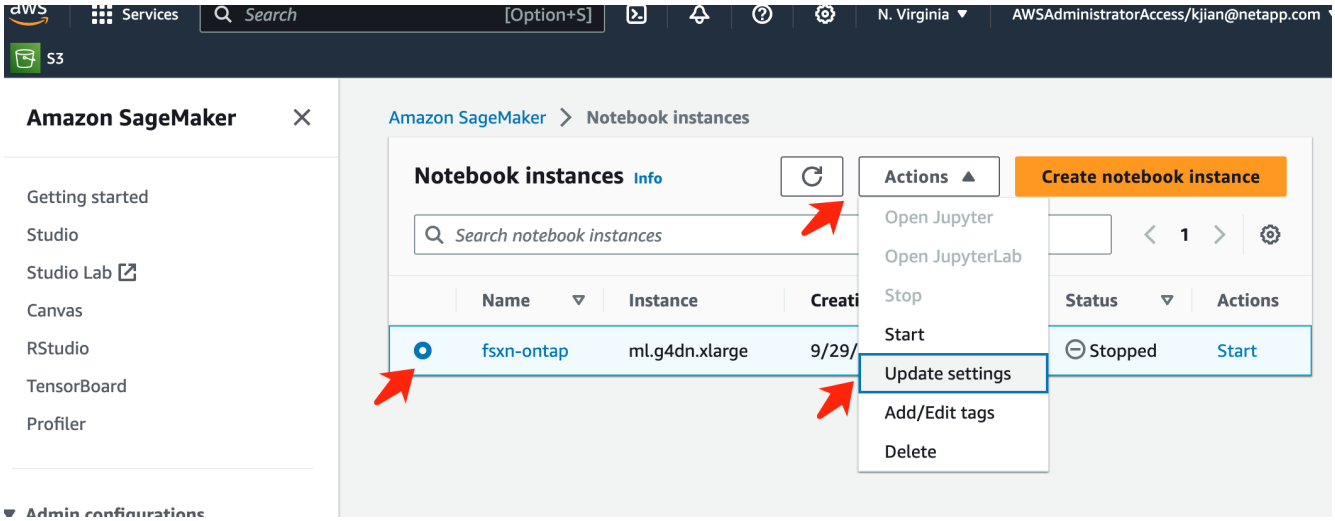
3. 以下のコードを入力エリアに貼り付けます。

```
#!/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
```

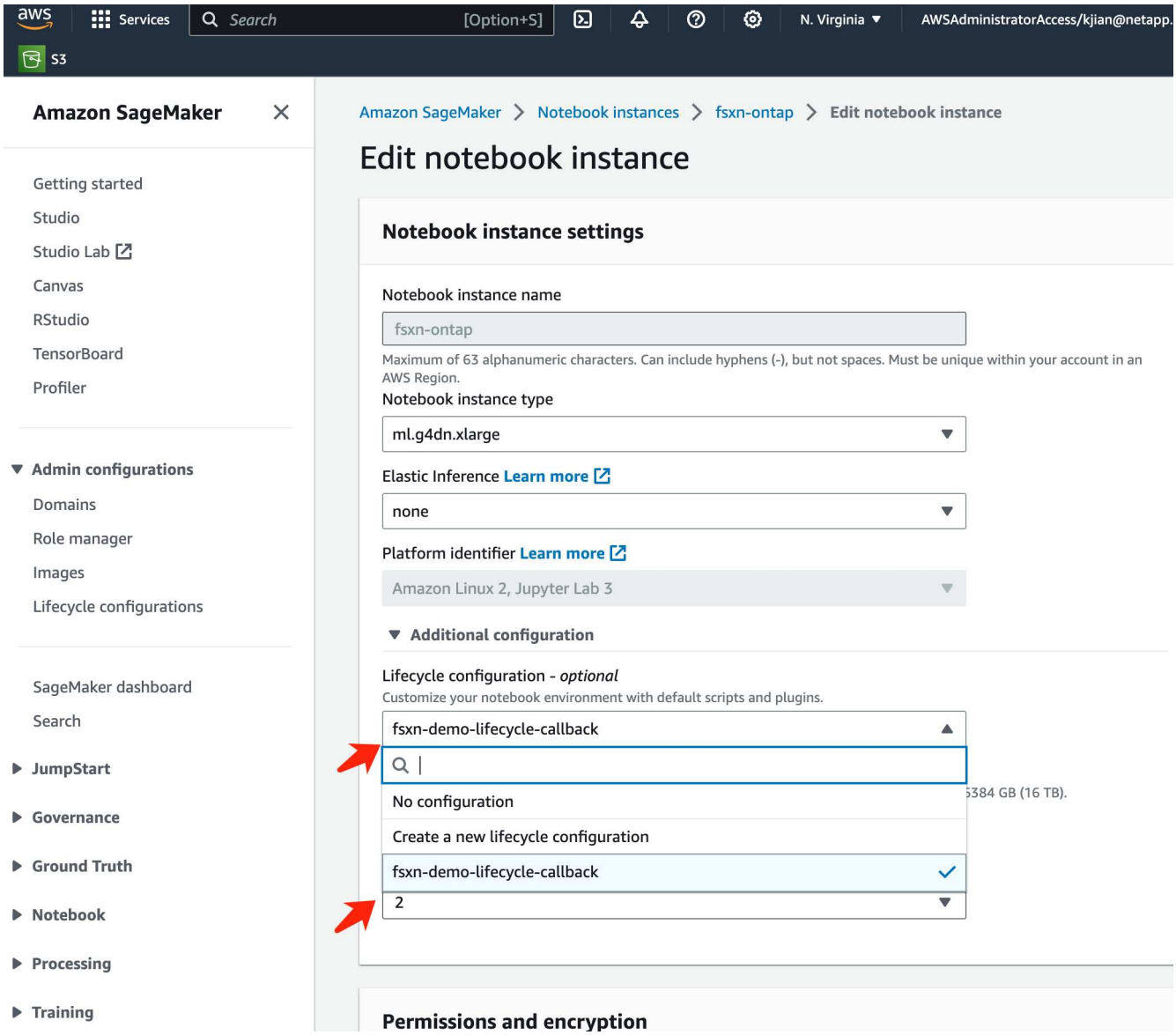
4. このスクリプトは、推論用モデルの再トレーニングと再デプロイメントを処理する Jupyter Notebook を実行します。実行が完了すると、ノートブックは 5 分以内に自動的にシャットダウンします。問題の説明とコードの実装の詳細については、以下を参照してください。"パート 2 - SageMaker でのモデルトレーニング用のデータソースとしてAmazon FSx for NetApp ONTAP (FSx ONTAP) を活用する" 。



5. 作成後、ノートブックインスタンスに移動し、ターゲットインスタンスを選択して、[アクション] ドロップダウンの [設定の更新] をクリックします。



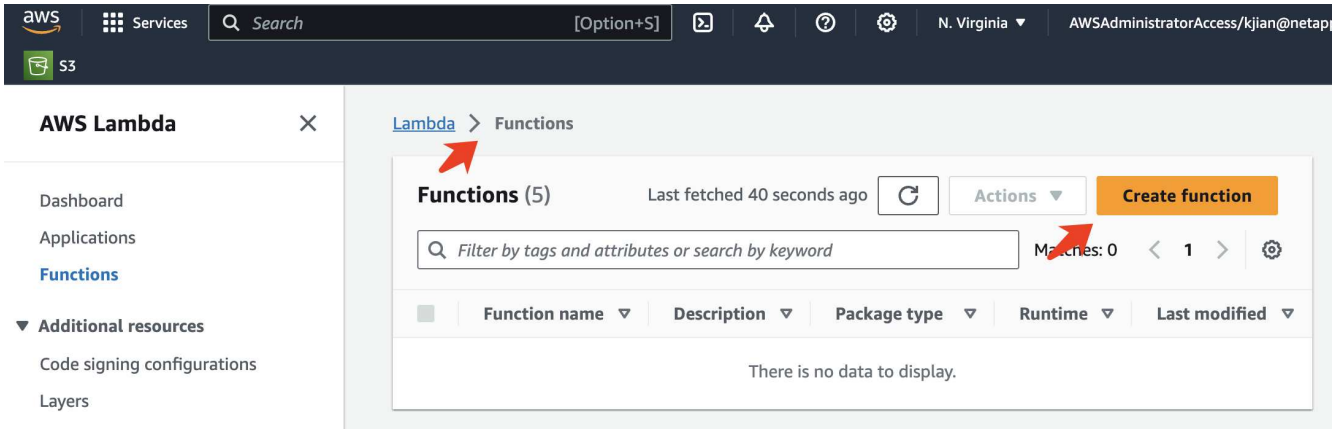
6. 作成した*ライフサイクル構成*を選択し、*ノートブックインスタンスの更新*をクリックします。



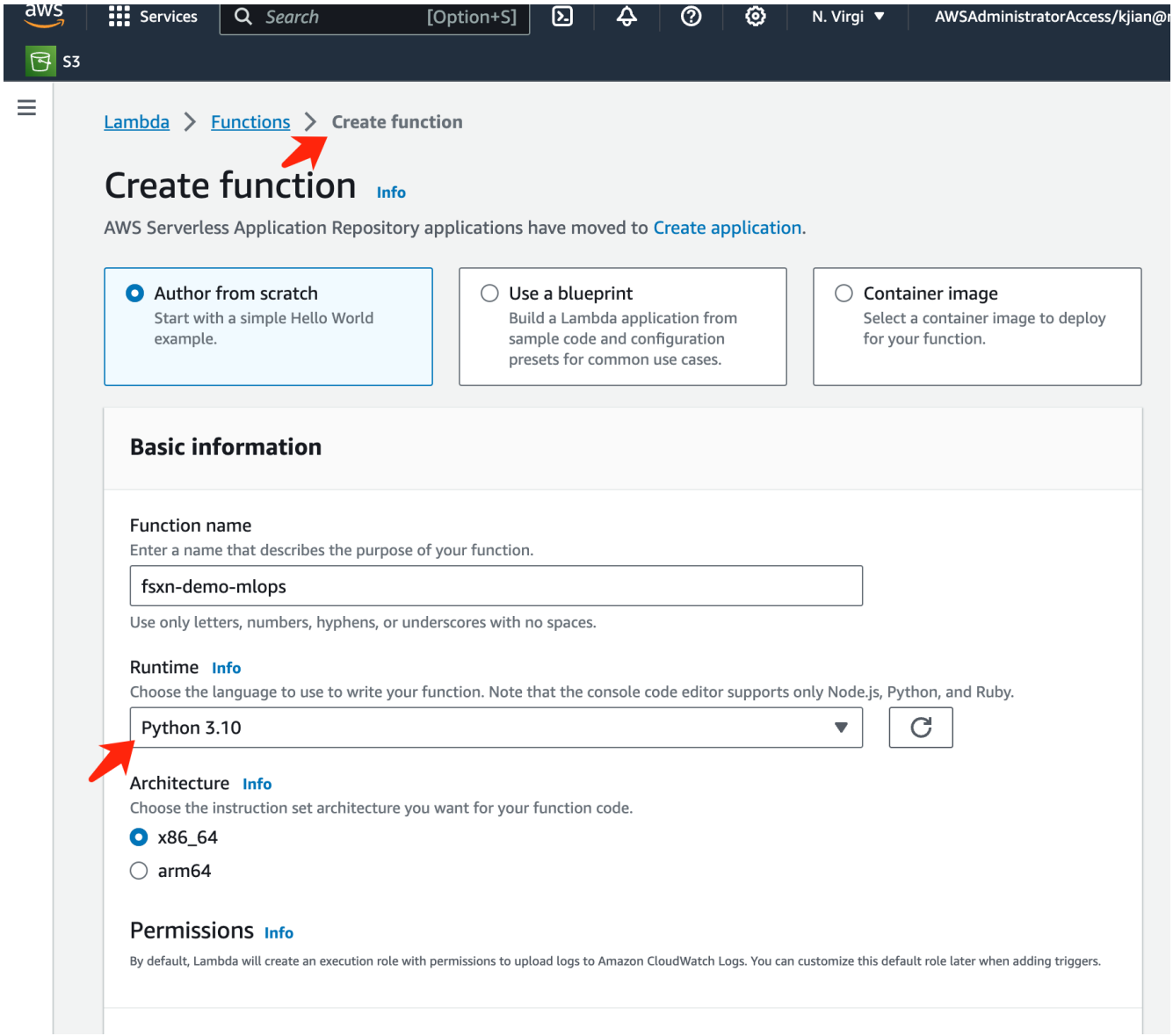
AWS Lambda サーバーレス関数

前述のように、**AWS Lambda** 関数*は ***AWS SageMaker Notebook** インスタンス を起動する役割を担っています。

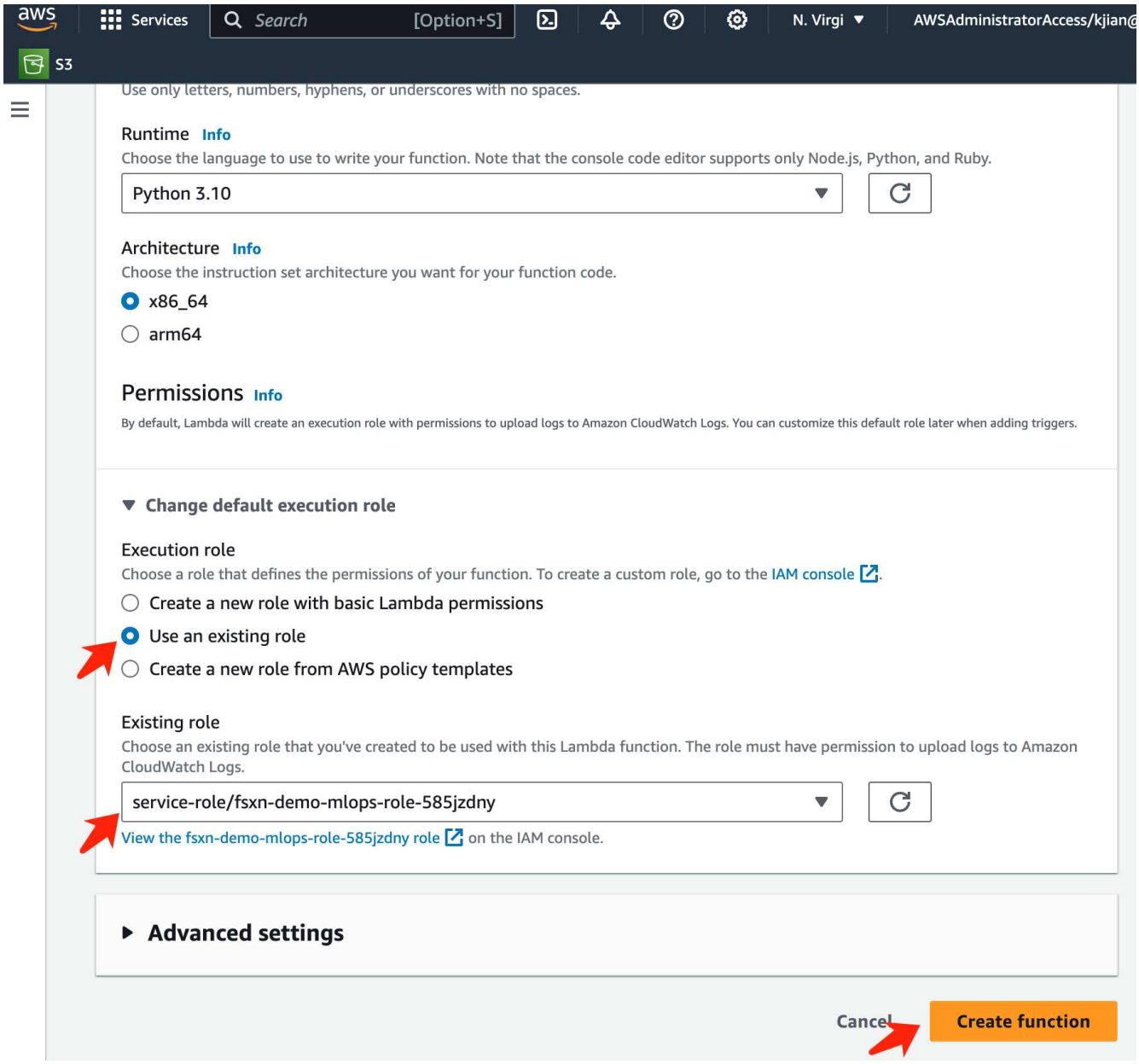
1. **AWS Lambda** 関数 を作成するには、該当するパネルに移動し、関数 タブに切り替えて、関数の作成 をクリックします。



2. ページ上のすべての必須エントリを入力し、ランタイムを **Python 3.10** に切り替えることを忘れないでください。



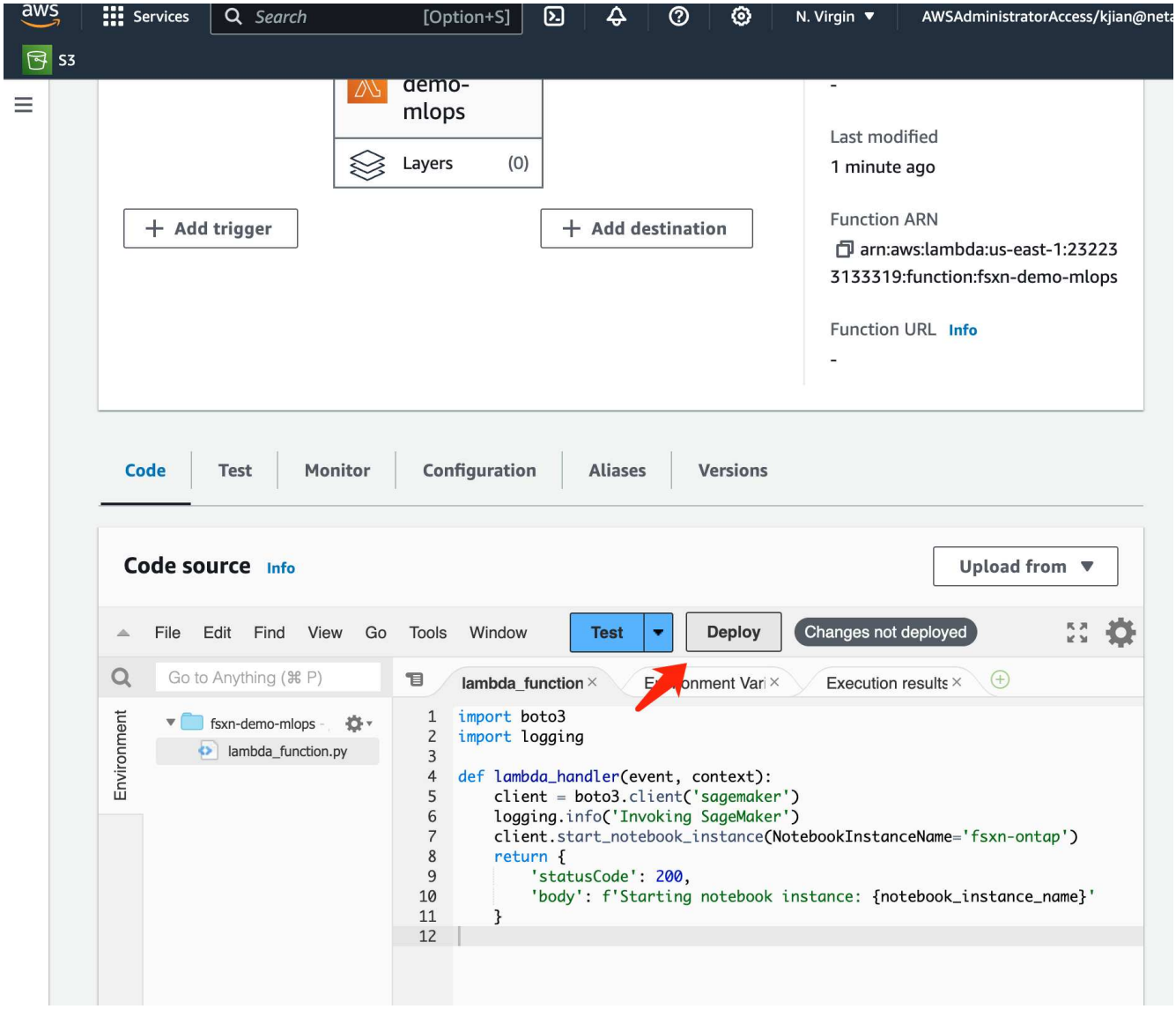
3. 指定されたロールに必要な権限 **AmazonSageMakerFullAccess** があることを確認し、 関数の作成 ボタンをクリックしてください。



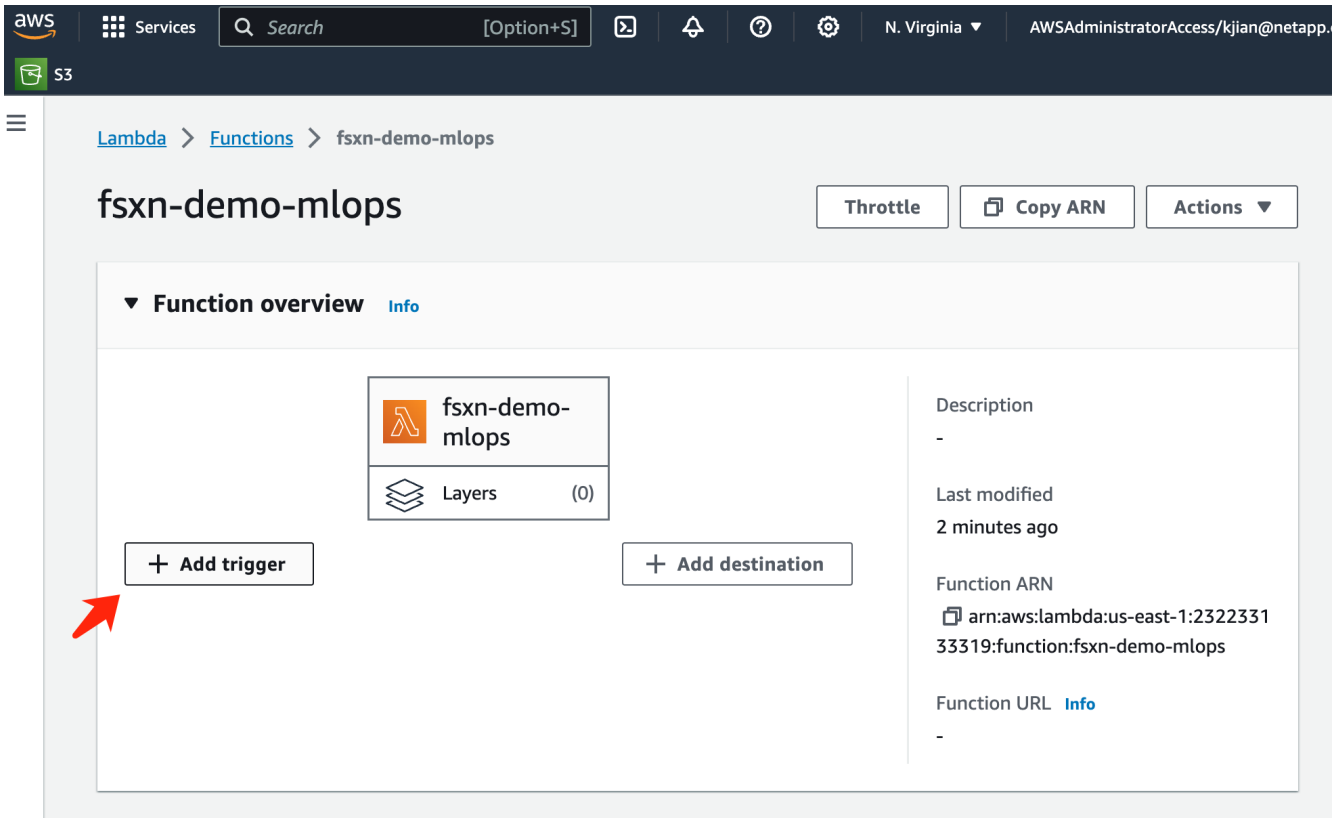
4. 作成したLambda関数を選択します。コードタブで、次のコードをコピーしてテキスト領域に貼り付けます。このコードは、**fsxn-ontap** という名前のノートブック インスタンスを起動します。

```
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}'
    }
```

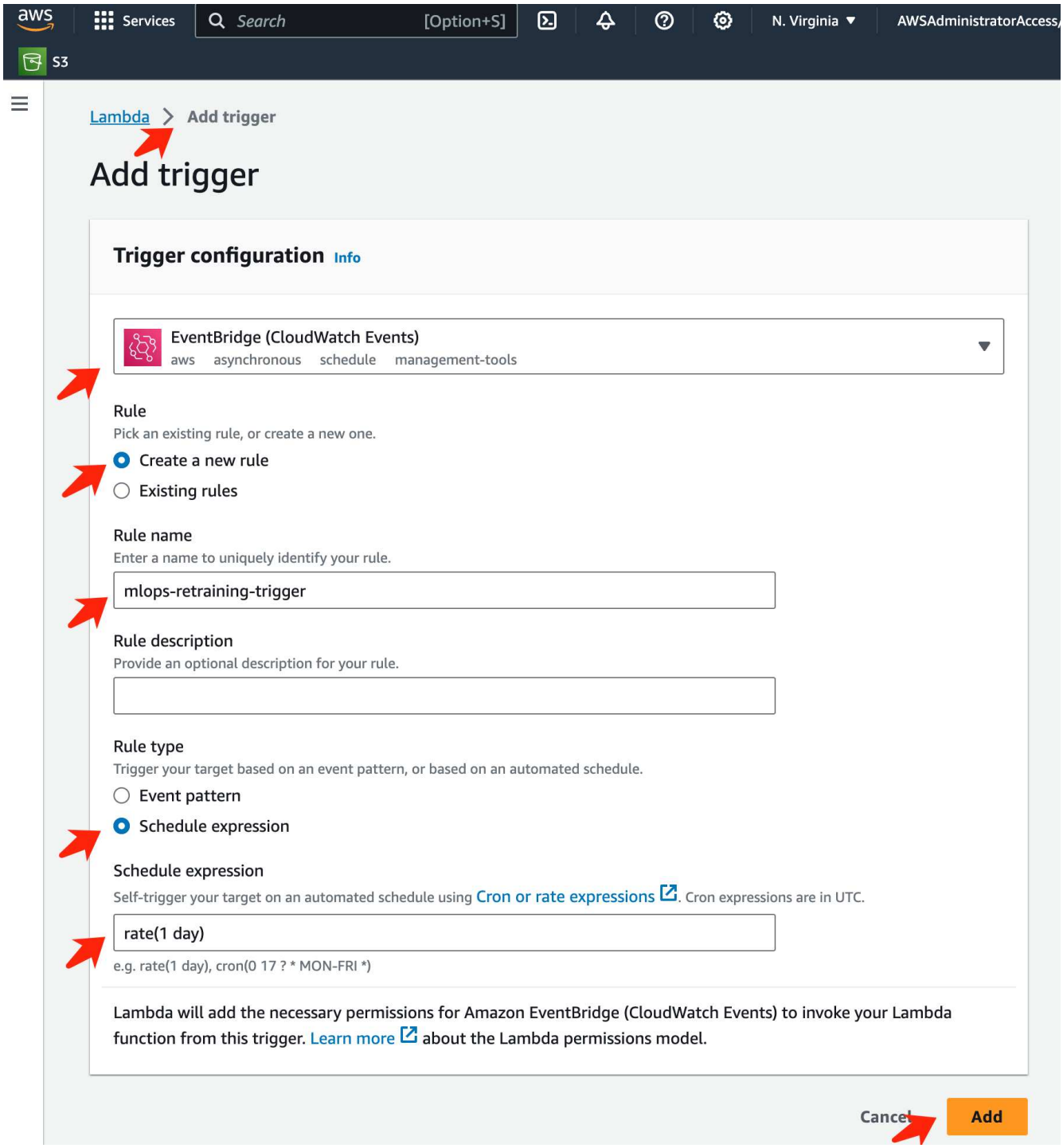
5. このコードの変更を適用するには、[デプロイ] ボタンをクリックします。



6. この AWS Lambda 関数をトリガーする方法を指定するには、「トリガーの追加」ボタンをクリックします。



7. ドロップダウン メニューから EventBridge を選択し、「新しいルールの作成」というラジオ ボタンをクリックします。スケジュール式フィールドに次のように入力します。 `rate(1 day)`をクリックし、[追加] ボタンをクリックして、この新しい cron ジョブ ルールを作成し、AWS Lambda 関数に適用します。



2 段階の設定が完了すると、毎日、**AWS Lambda** 関数 が **SageMaker Notebook** を起動し、**FSx ONTAP** リポジトリのデータを使用してモデルの再トレーニングを実行し、更新されたモデルを本番環境に再デプロイし、**SageMaker Notebook** インスタンス を自動的にシャットダウンしてコストを最適化します。これにより、モデルが最新の状態に保たれます。

これで、MLOps パイプラインの開発に関するチュートリアルは終了です。

著作権に関する情報

商標に関する情報