AWS FSX pour NetApp ONTAP (FSxN) pour MLOps

NetApp Solutions

NetApp
April 26, 2024

Sommaire

AWS FSX pour NetApp ONTAP (FSxN) pour MLOps

Auteur(s):
Jian Jian (Ken), scientifique senior en données et applications, NetApp

Cette section examine l'application pratique du développement d'infrastructures d'IA et fournit une description de bout en bout de la construction d'un pipeline MLOps à l'aide de FSxN. Il comprend trois exemples complets et vous guide pour répondre à vos besoins MLOps via cette puissante plateforme de gestion des données.

Ces articles portent sur :

1. "1re partie : intégration d'AWS FSX pour NetApp ONTAP (FSxN) en tant que compartiment S3 privé dans AWS SageMaker"
2. "2e partie - exploitation d'AWS FSX pour NetApp ONTAP (FSxN) en tant que source de données pour l'entraînement des modèles dans SageMaker"
3. "Partie 3 - construire Un pipeline MLO simplifié (ci/CT/CD)"

À la fin de cette section, vous aurez acquis une solide compréhension de la façon d'utiliser FSxN pour rationaliser les processus MLOps.

1re partie : intégration d'AWS FSX pour NetApp ONTAP (FSxN) en tant que compartiment S3 privé dans AWS SageMaker

Auteur(s):
Jian Jian (Ken), scientifique senior en données et applications, NetApp

Introduction

Utilisation de SageMaker comme exemple, cette page fournit des conseils sur la configuration de FSxN en tant que compartiment S3 privé.

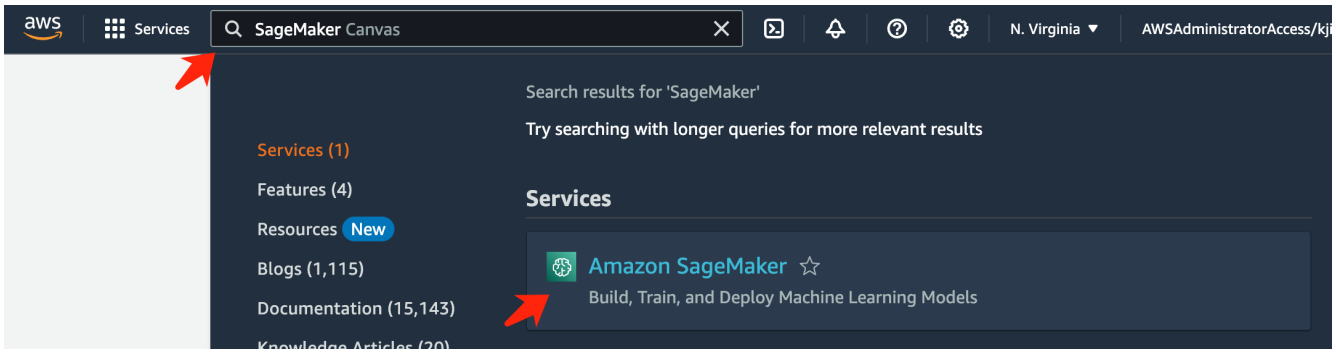
Pour plus d'informations sur FSxN, veuillez consulter cette présentation ("Lien vidéo")

Guide de l'utilisateur

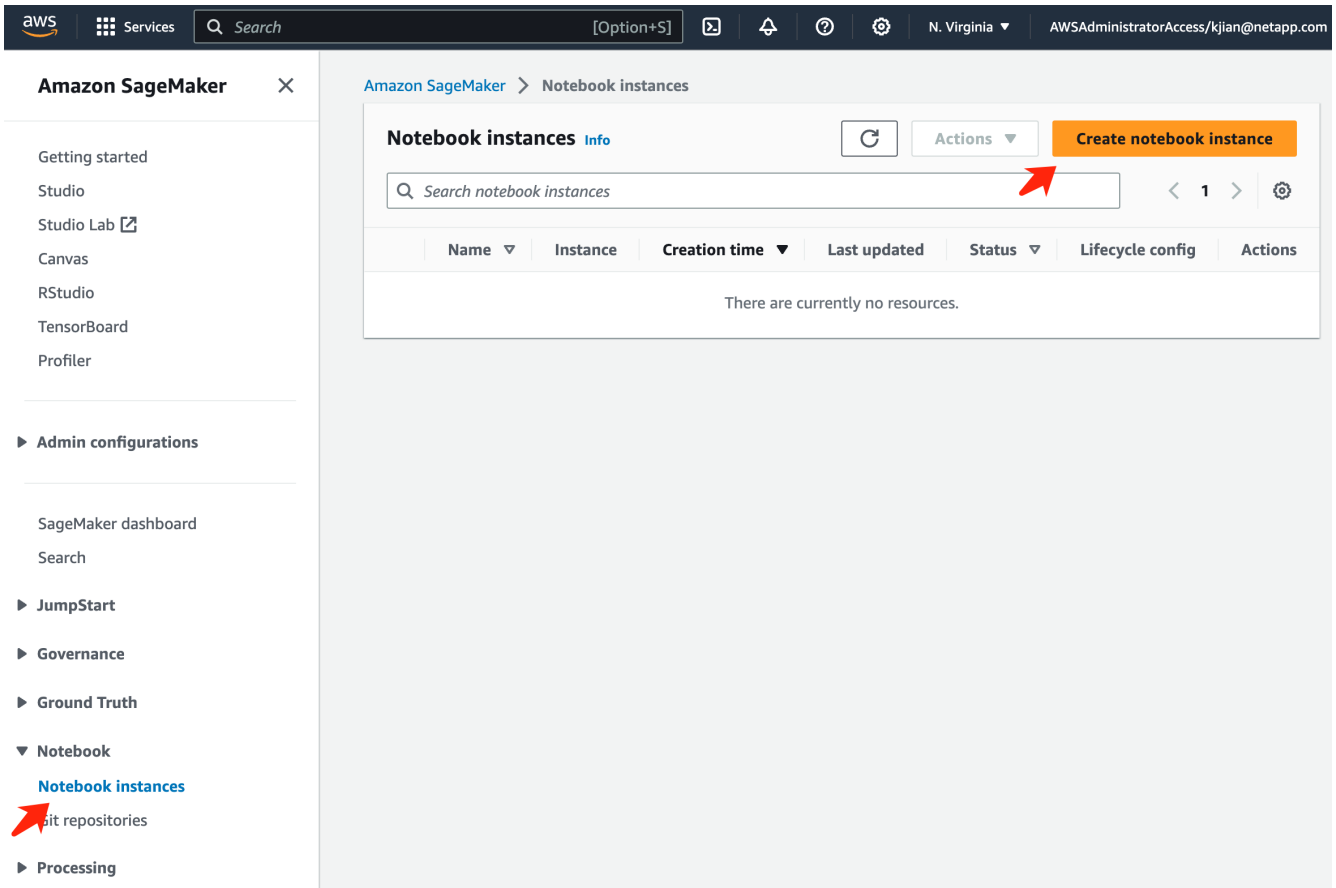
Création du serveur

Créez une instance de bloc-notes SageMaker

1. Ouvrez la console AWS. Dans le panneau de recherche, recherchez SageMaker et cliquez sur le service **Amazon SageMaker**.



2. Ouvrez **Notebook instances** sous l'onglet Notebook, cliquez sur le bouton orange **Créer une instance de bloc-notes**.



3. Dans la page de création,
   Entrez le **Nom de l'instance du bloc-notes**
   Développez le panneau **réseau**
   Laissez les autres entrées par défaut et sélectionnez un **VPC**, **Subnet** et **Groupe(s) de sécurité**. (Ce **VPC** et ce **Subnet** seront utilisés ultérieurement pour créer un système de fichiers FSxN)
   Cliquez sur le bouton orange **Créer une instance de bloc-notes** en bas à droite.

Amazon SageMaker > Notebook instances > Create notebook instance

Create notebook instance

Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. Learn more

Notebook instance settings

Notebook instance name

fsxn-demo

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Notebook instance type

ml.t3.medium

Elastic Inference Learn more

none

Platform identifier Learn more

Amazon Linux 2, Jupyter Lab 3

▶ Additional configuration

Permissions and encryption

IAM role

Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the **AmazonSageMakerFullAccess** IAM policy attached.

AmazonSageMakerServiceCatalogProductsUseRole

Create role using the role creation wizard

Root access - *optional*

- (●) Enable - Give users root access to the notebook
- ( ) Disable - Don't give users root access to the notebook
  Lifecycle configurations always have root access

Encryption key - *optional*

Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.

No Custom Encryption

▼ Network - *optional*

VPC - *optional*

Default vpc-0df3956ab1fca2ec9 (172.31.0.0/16)

Subnet

Choose a subnet in an availability zone supported by Amazon SageMaker.

subnet-00060df0d0f562672 (172.31.16.0/20) | us-east-1a

Security group(s)

sg-0a39b3985770e9256 (default) ✕

Direct internet access

- (●) Enable — Access the internet directly through Amazon SageMaker
- ( ) Disable — Access the internet through a VPC
  To train or host models from a notebook, you need internet access. To enable internet access, make sure that your VPC has a NAT gateway and your security group allows outbound connections. Learn more

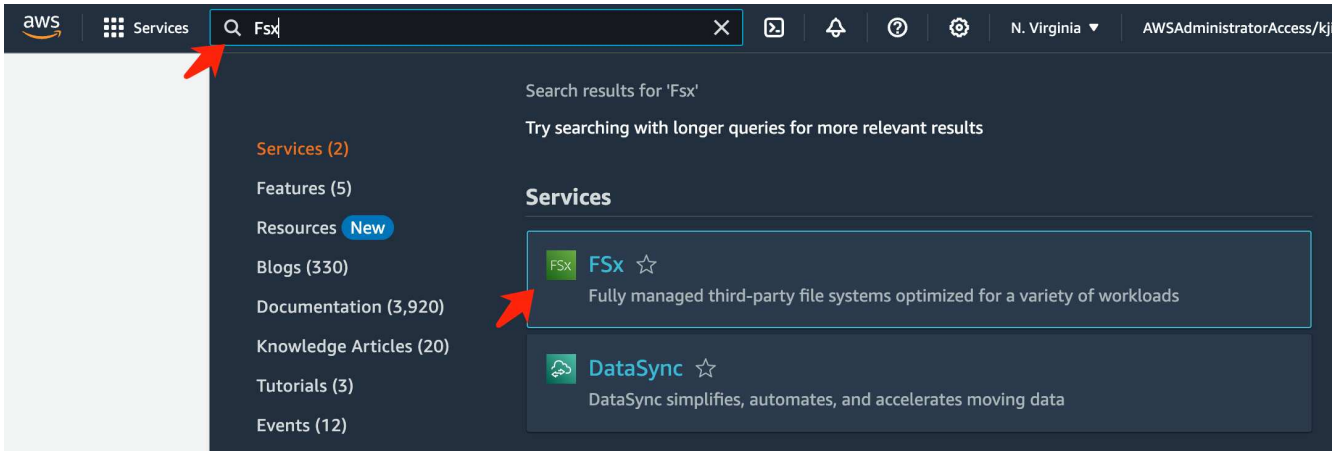
▶ Git repositories- *optional*

▶ Tags - *optional*

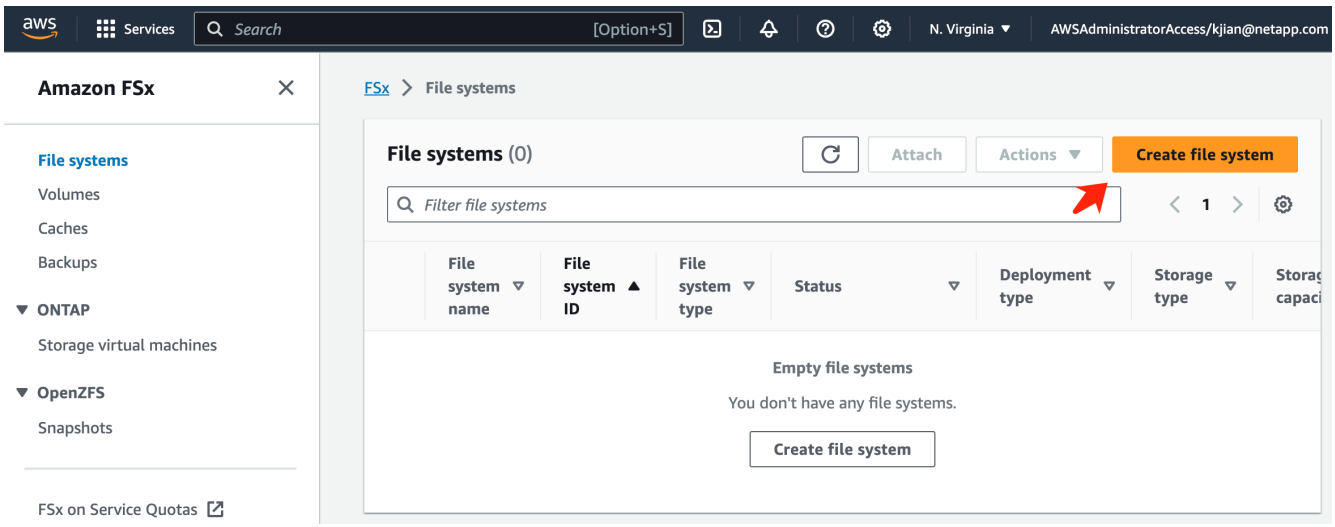
Cancel | Create notebook instance

Créez un système de fichiers FSxN

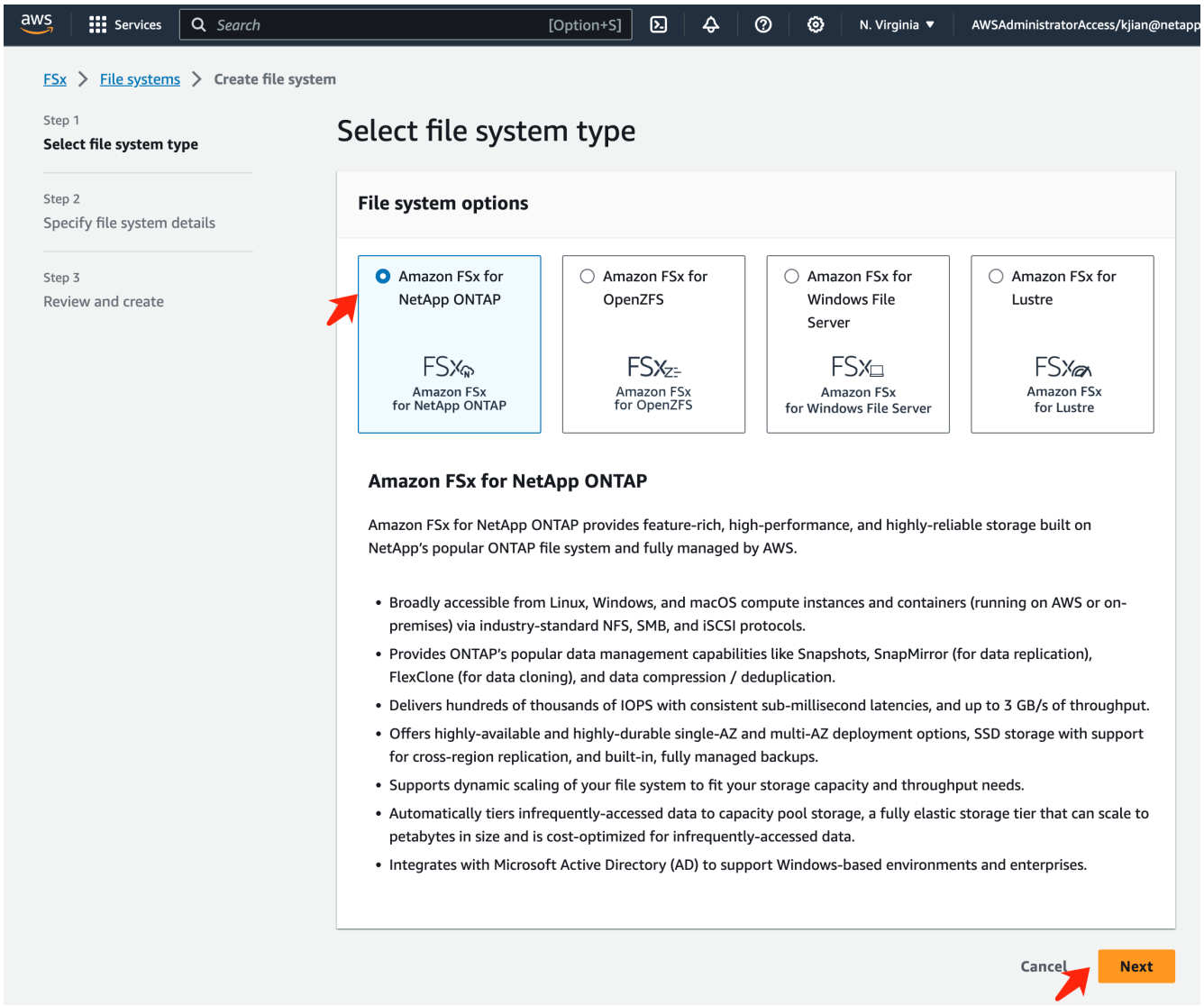
1. Ouvrez la console AWS. Dans le panneau de recherche, recherchez FSX et cliquez sur le service **FSX**.



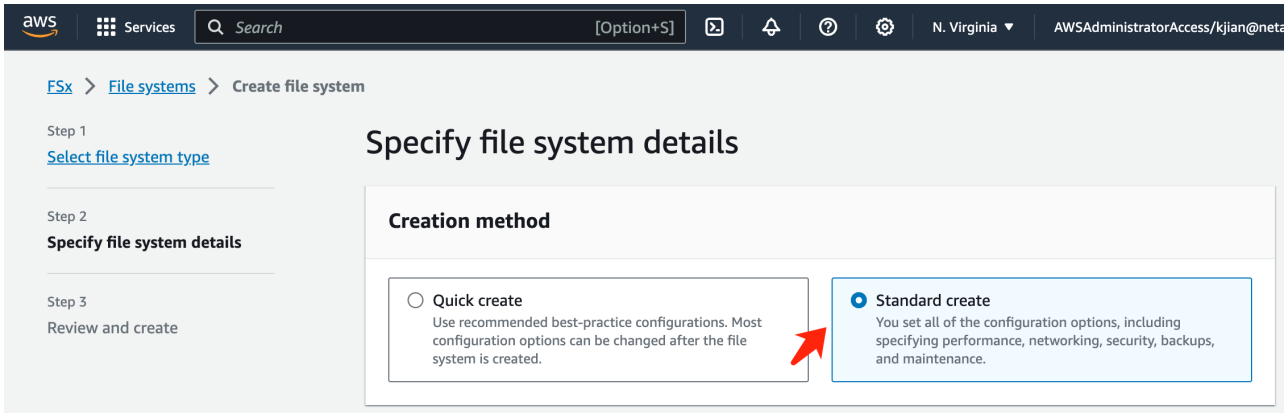
2. Cliquez sur **Créer un système de fichiers**.



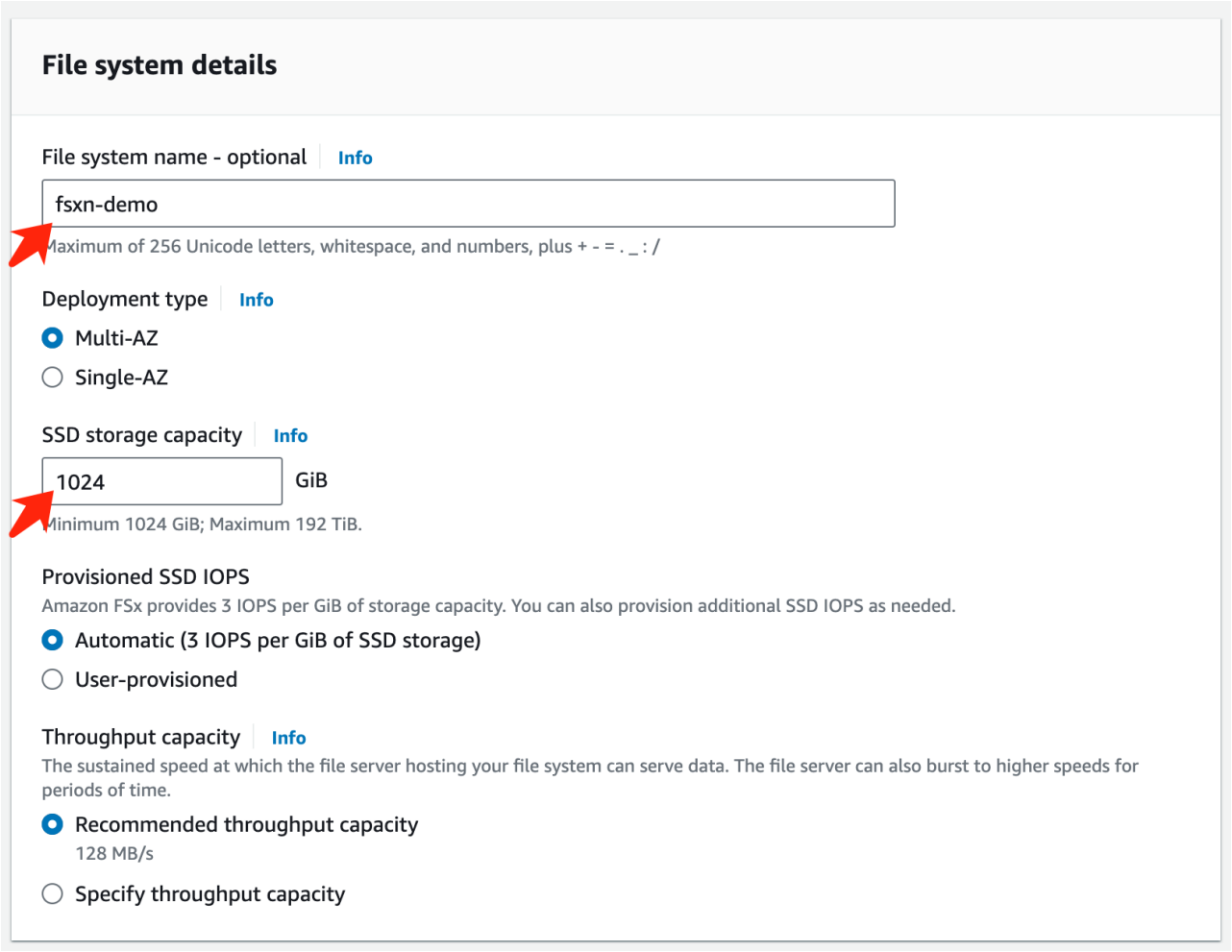
3. Sélectionnez la première carte **FSX pour NetApp ONTAP** et cliquez sur **Suivant**.



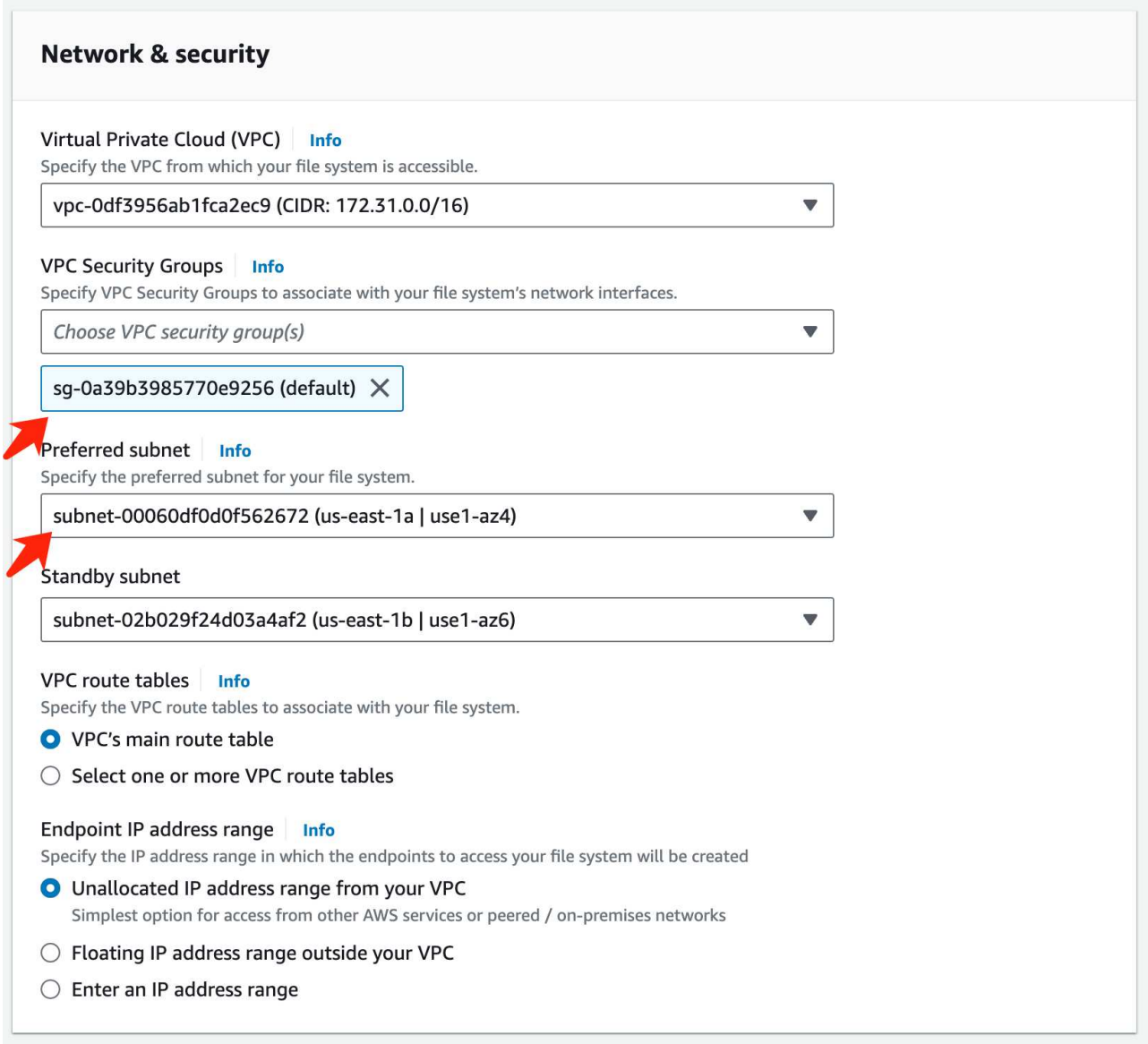
4. Dans la page de configuration détaillée.
   a. Sélectionnez l'option **création standard**.



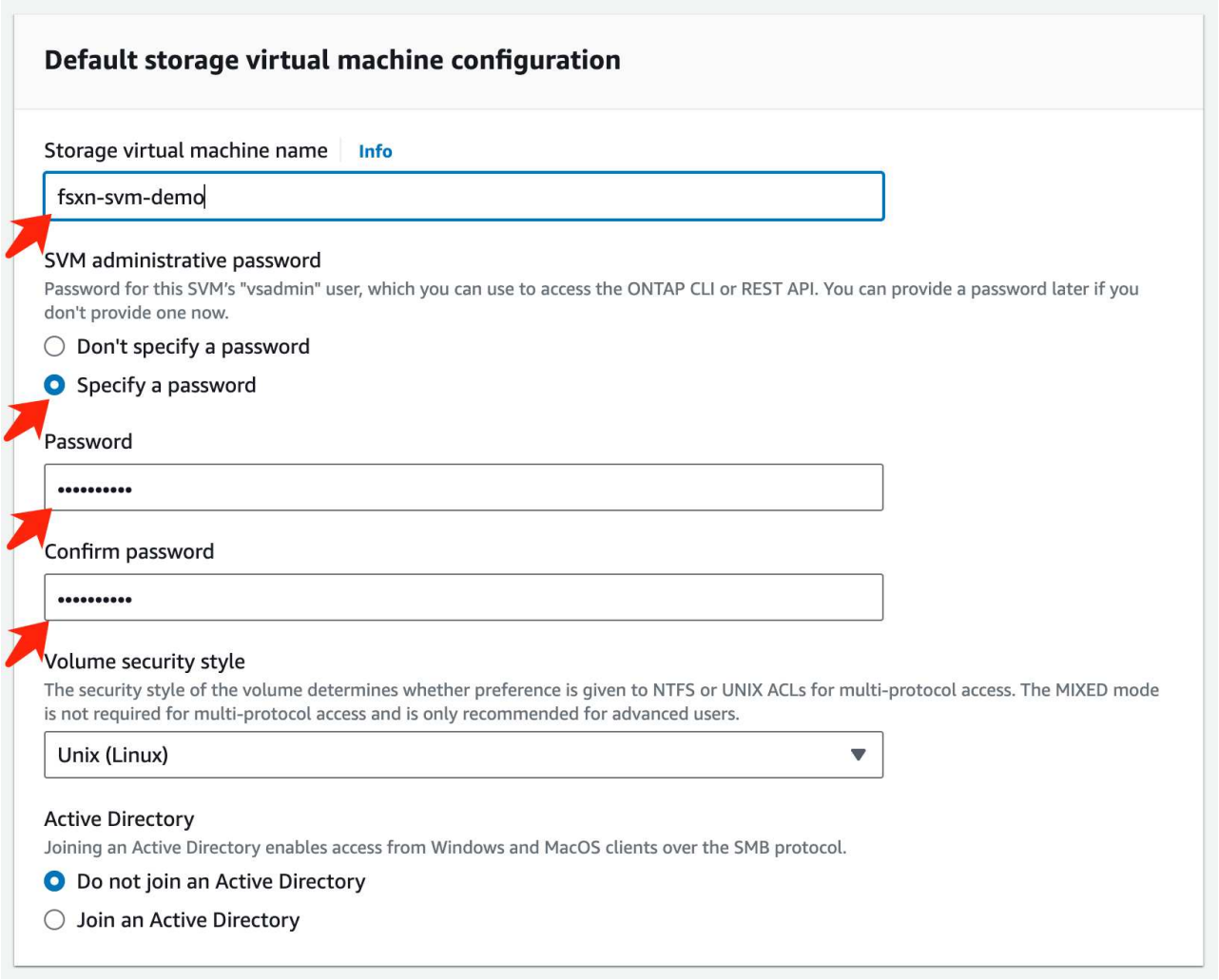
   b. Entrez le **Nom du système de fichiers** et la **capacité de stockage SSD**.



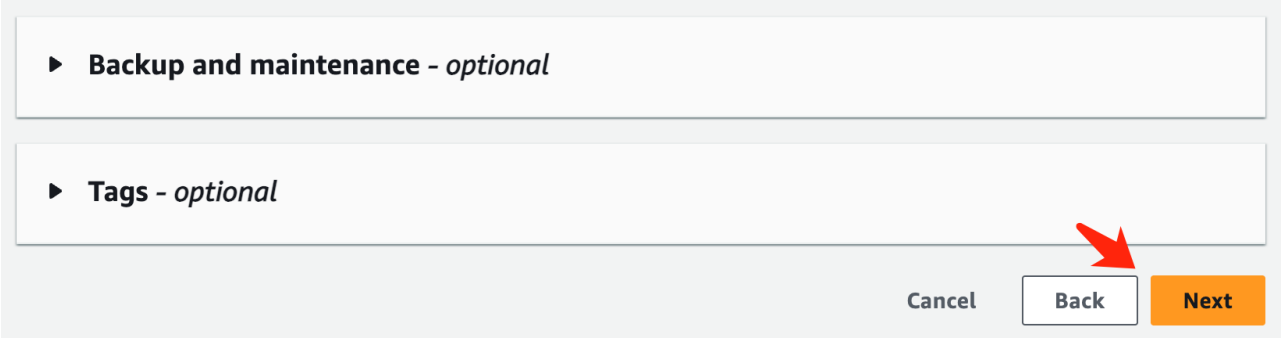
c. Assurez-vous d'utiliser le **VPC** et le **subnet** de la même manière que l'instance **SageMaker Notebook**.



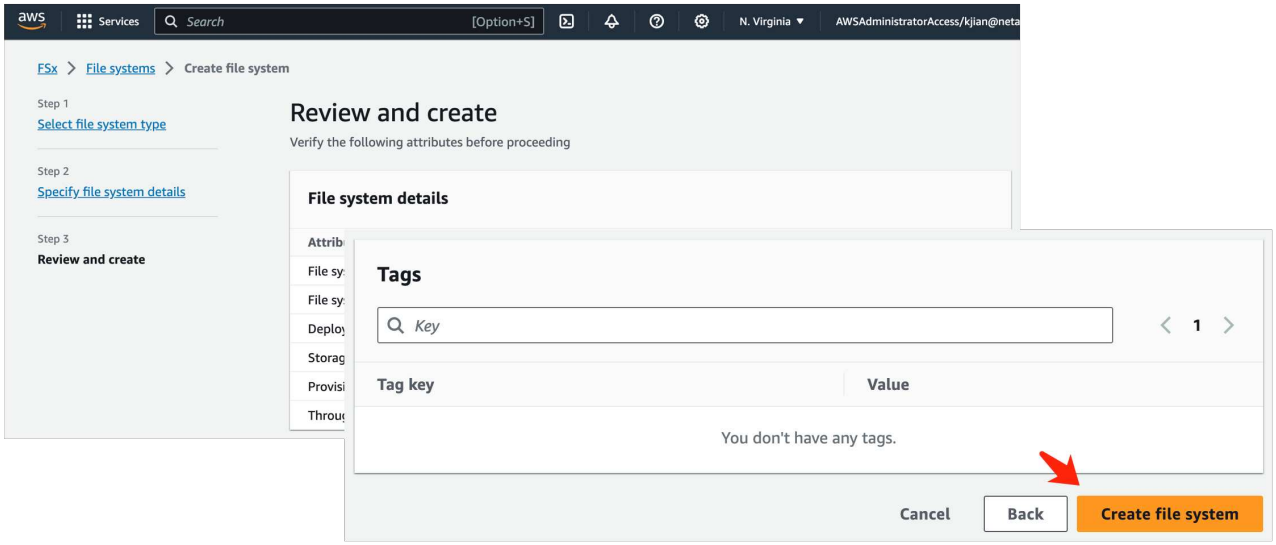
d. Entrer le nom **Storage Virtual machine** et **spécifier un mot de passe** pour votre SVM (Storage Virtual machine).



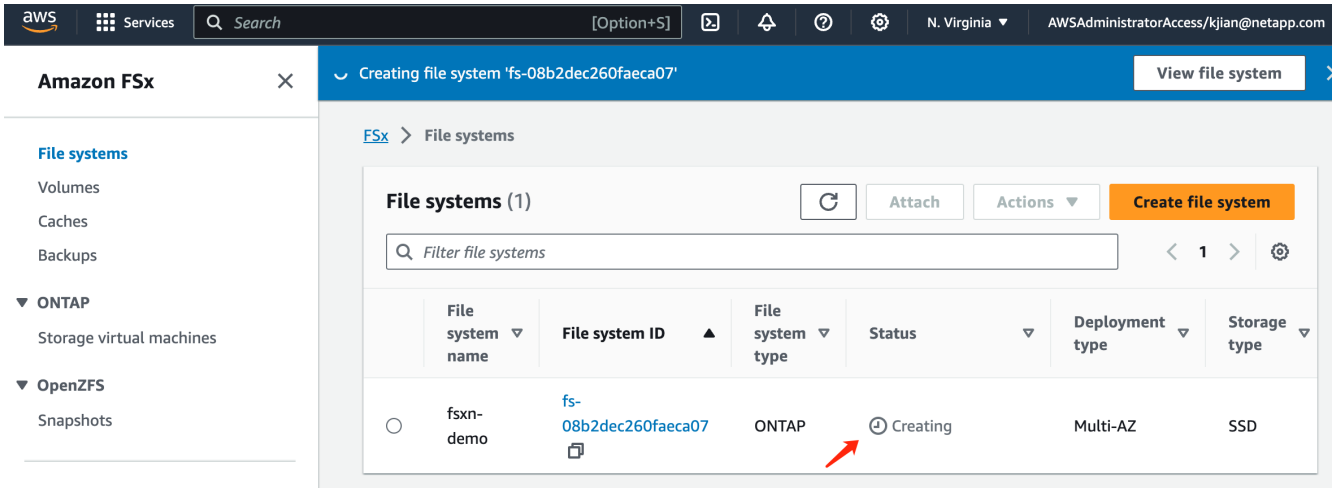
e. Laissez les autres entrées par défaut et cliquez sur le bouton orange **Suivant** en bas à droite.



f. Cliquez sur le bouton orange **Créer un système de fichiers** en bas à droite de la page de revue.



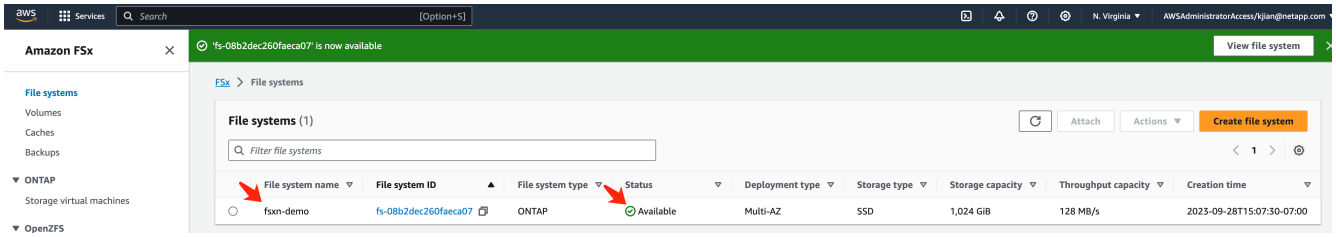
5. Le démarrage du système de fichiers FSX peut prendre environ **20-40 minutes**.



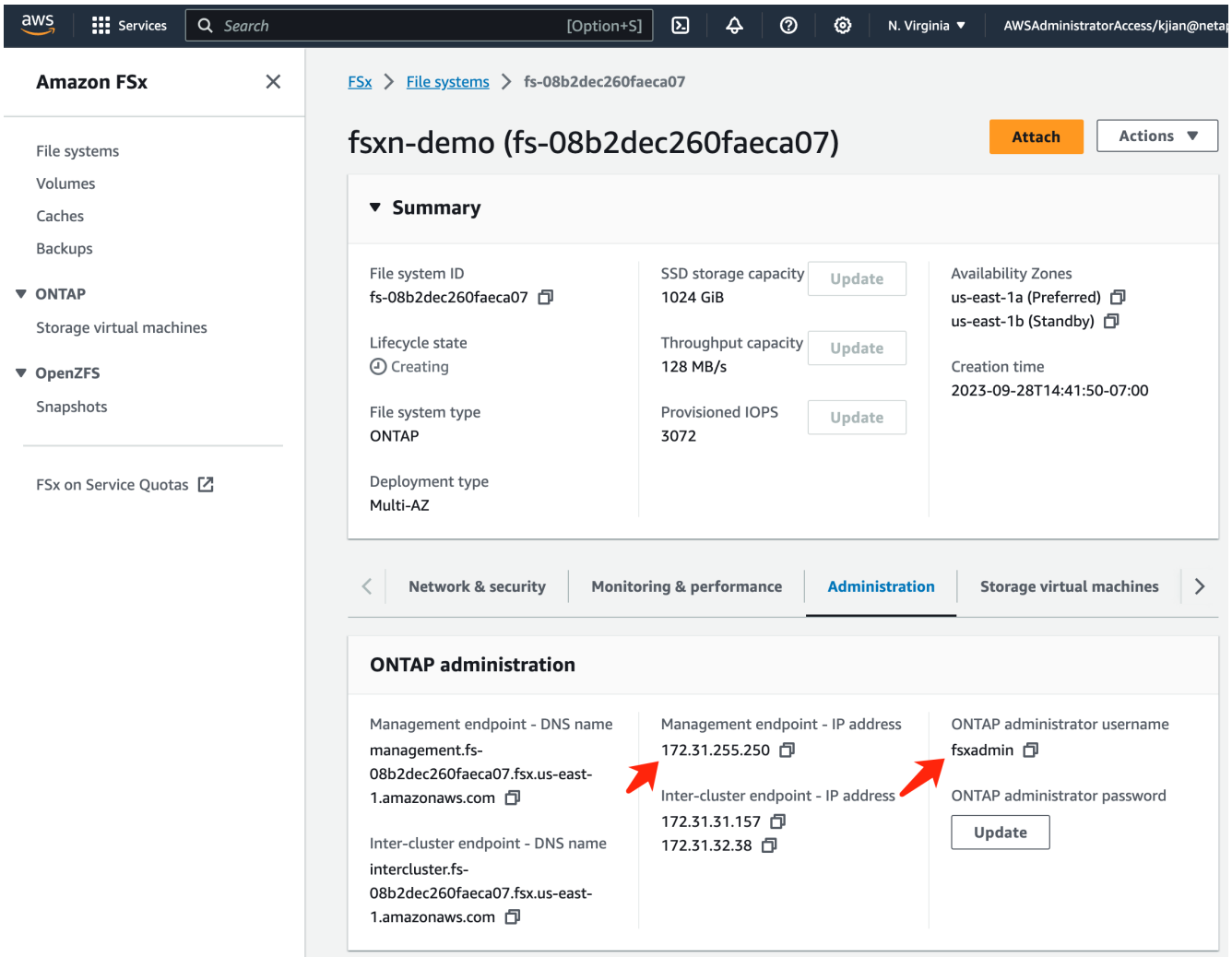
Configuration du serveur

Configuration ONTAP

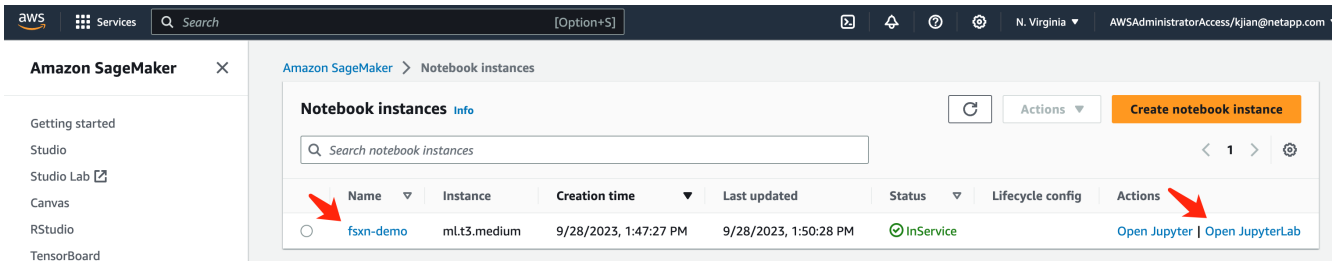
1. Ouvrez le système de fichiers FSX créé. Veuillez vous assurer que l'état est **disponible**.



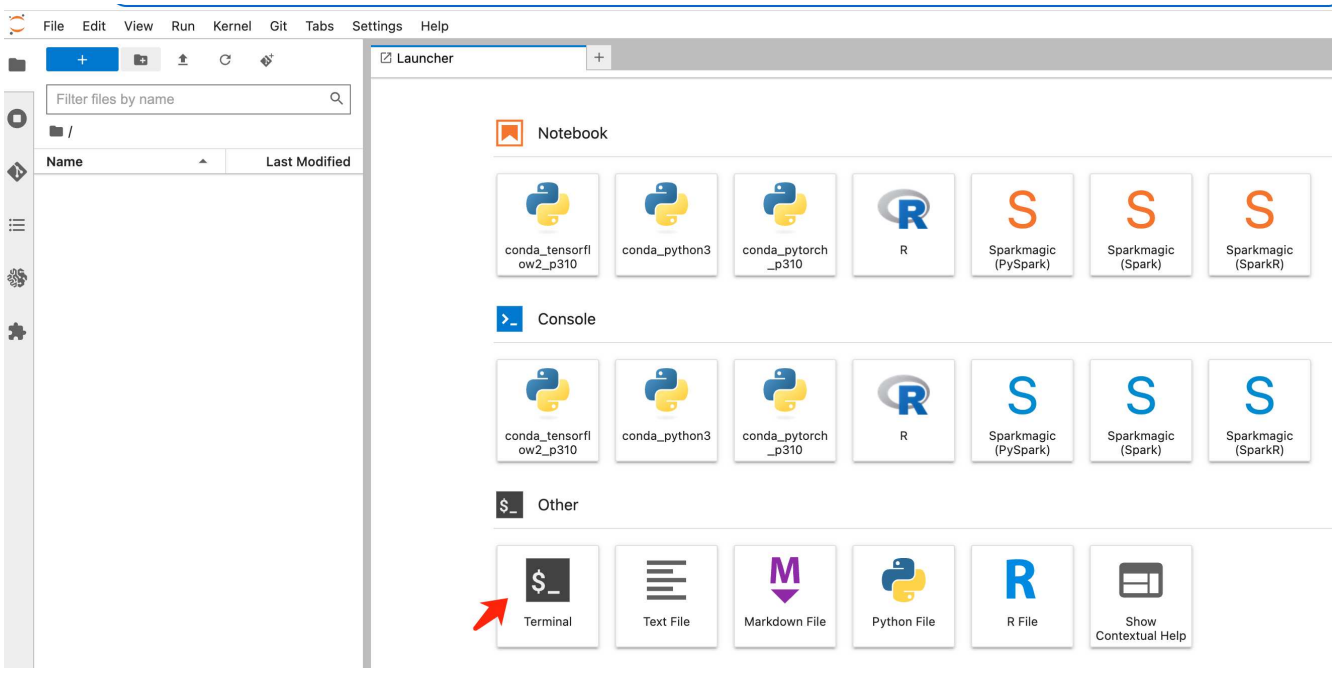
2. Sélectionnez l'onglet **Administration** et conservez le **noeud final de gestion - adresse IP** et le **nom d'utilisateur de l'administrateur ONTAP**.



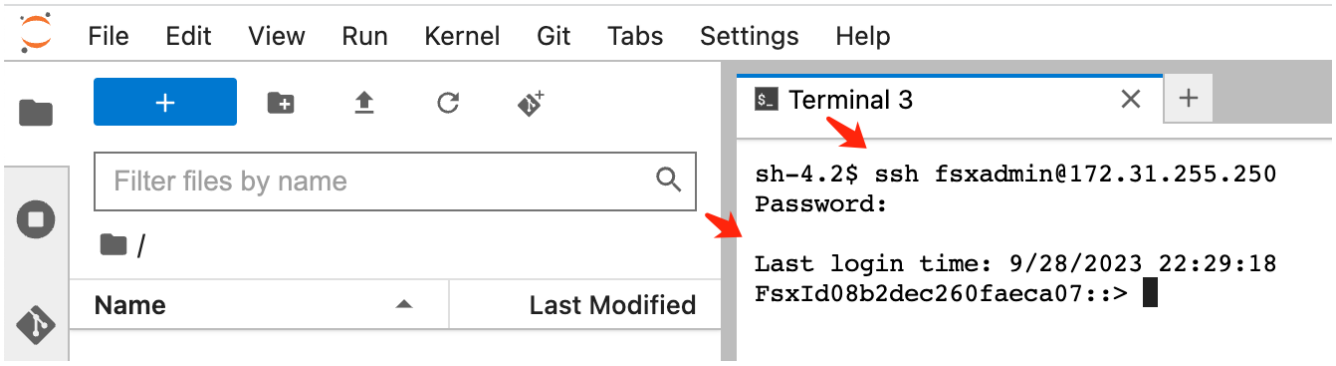
3. Ouvrez l'instance **SageMaker Notebook** créée et cliquez sur **Ouvrir JupyterLab**.



4. Dans la page Jupyter Lab, ouvrez un nouveau **terminal**.

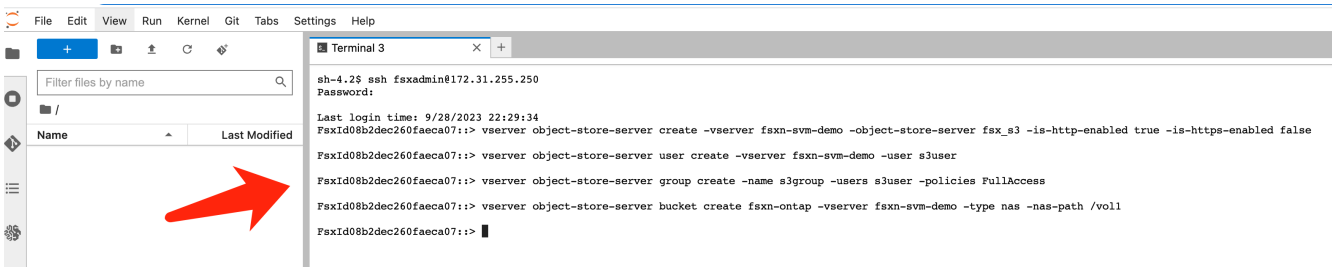


5. Entrez la commande ssh ssh <nom d'utilisateur admin>@<adresse IP du serveur ONTAP> pour vous connecter au système de fichiers ONTAP FSxN. (Le nom d'utilisateur et l'adresse IP sont extraits de l'étape 2)
   Veuillez utiliser le mot de passe utilisé lors de la création de la **machine virtuelle de stockage**.



6. Exécutez les commandes dans l'ordre suivant.
   Nous utilisons **fsxn-ontap** comme nom du compartiment privé **FSxN S3**.
   Veuillez utiliser le **nom de la machine virtuelle de stockage** pour l'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. Exécutez les commandes ci-dessous pour récupérer l'adresse IP et les informations d'identification du terminal pour FSxN privé S3.

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

set adv

vserver object-store-server user show
```

8. Conservez l'adresse IP et les informations d'identification du point de terminaison pour une utilisation ultérieure.

```
sh-4.2$ ssh fsxadmin@172.31.255.250
Password:

Last login time: 9/28/2023 22:32:42
FsxId08b2dec260faeca07::> network interface show -vserver fsxn-svm-demo -lif nfs_smb_management_1

                    Vserver Name: fsxn-svm-demo
            Logical Interface Name: nfs_smb_management_1
                   Service Policy: default-data-files
                     Service List: data-core, data-nfs, data-cifs,
                                   management-ssh, management-https,
                                   data-s3-server, data-dns-server
               (DEPRECATED)-Role: data
                  Data Protocol: nfs, cifs, s3
                 Network Address: [redacted]
                         Netmask: 255.255.255.192
             Bits in the Netmask: 26
                      Is VIP LIF: false
                     Subnet Name: -
                       Home Node: FsxId08b2dec260faeca07-01
                       Home Port: e0e
                    Current Node: FsxId08b2dec260faeca07-01
                    Current Port: e0e
                Operational Status: up
                   Extended Status: -
                         Is Home: true
             Administrative Status: up
                   Failover Policy: system-defined
      (DEPRECATED)-Firewall Policy: data
                       Auto Revert: true
     Fully Qualified DNS Zone Name: none
           DNS Query Listen Enable: false
               Failover Group Name: Fsx
                          FCP WWPN: -
                    Address family: ipv4
                           Comment: -
                   IPspace of LIF: Default
    Is Dynamic DNS Update Enabled?: true
Probe-port for Cloud Load Balancer: -
                  Broadcast Domain: Fsx
                       Vserver Type: data
   Required RDMA offload protocols: -

FsxId08b2dec260faeca07::> set adv

Warning: These advanced commands are potentially dangerous; use them only when directed to do so by NetApp personnel.
Do you want to continue? {y|n}: y

FsxId08b2dec260faeca07::*> vserver object-store-server user show
Vserver     User            ID        Access Key           Secret Key
----------- --------------- --------- -------------------- --------------------
fsxn-svm-demo
            root            0         -                    -
   Comment: Root User
fsxn-svm-demo
            s3user          1         [redacted]           [redacted]

2 entries were displayed.

FsxId08b2dec260faeca07::*>
```

Configuration du client

1. Dans l'instance de SageMaker Notebook, créez un nouveau bloc-notes Jupyter.

2. Le code ci-dessous vous permettra de télécharger des fichiers vers un compartiment S3 privé FSxN. Pour obtenir un exemple de code complet, reportez-vous à cet ordinateur portable. "fsxn_demo.ipynb"

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

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

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

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

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

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

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

# Initialize Fsxn S3 bucket object
# --- Start integrating SageMaker with FSXN ---
# This is the only code change we need to incorporate SageMaker with
FSXN
```

```
s3_client: boto3.client = boto3.resource(
    's3',
    region_name=region_name,
    aws_access_key_id=aws_access_key_id,
    aws_secret_access_key=aws_secret_access_key,
    use_ssl=False,
    endpoint_url=f'http://{fsx_endpoint_ip}',
    config=boto3.session.Config(
        signature_version='s3v4',
        s3={'addressing_style': 'path'}
    )
)
# --- End integrating SageMaker with FSXN ---

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

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

Ceci conclut l'intégration entre FSxN et l'instance SageMaker.

Liste de contrôle de débogage utile

- Assurez-vous que l'instance de l'ordinateur portable SageMaker et le système de fichiers FSxN se trouvent dans le même VPC.
- N'oubliez pas d'exécuter la commande **set dev** sur ONTAP pour définir le niveau de privilège sur **dev**.

FAQ (au 27 septembre 2023)

Q: Pourquoi reçois-je l'erreur "**une erreur s'est produite (NotImplemented) lors de l'appel de l'opération CreateMultipartUpload : la commande s3 demandée n'est pas implémentée**" lors du téléchargement de fichiers vers FSxN ?

R : en tant que compartiment S3 privé, FSxN prend en charge le téléchargement de fichiers jusqu'à 100 Mo. Lors de l'utilisation du protocole S3, les fichiers de plus de 100 Mo sont divisés en blocs de 100 Mo et la fonction 'CreateMultipartUpload' est appelée. Toutefois, la mise en œuvre actuelle de FSxN Private S3 ne prend pas en charge cette fonction.

Q: Pourquoi reçois-je l'erreur "**une erreur s'est produite (AccessDenied) lors de l'appel des opérations PutObject: Access denied**" lors du téléchargement de fichiers vers FSxN?

R : pour accéder au compartiment S3 privé FSxN à partir d'une instance d'ordinateur portable SageMaker, basculez les informations d'identification AWS sur les informations d'identification FSxN. Cependant, l'octroi d'une autorisation d'écriture à l'instance nécessite une solution de contournement qui implique le montage du compartiment et l'exécution de la commande shell 'chmod' pour modifier les autorisations.

Q : Comment puis-je intégrer le compartiment S3 privé FSxN avec d'autres services SageMaker ML ?

R: Malheureusement, le SDK des services SageMaker ne permet pas de spécifier le noeud final pour le compartiment S3 privé. Par conséquent, FSxN S3 n'est pas compatible avec les services SageMaker tels que

Sagemaker Data Wrangler, Sagemaker Clarify, Sagemaker Glue, Sagemaker Athena, Sagemaker AutoML, et autres.

2e partie - exploitation d'AWS FSX pour NetApp ONTAP (FSxN) en tant que source de données pour l'entraînement des modèles dans SageMaker

Auteur(s):

Jian Jian (Ken), scientifique senior en données et applications, NetApp

Introduction

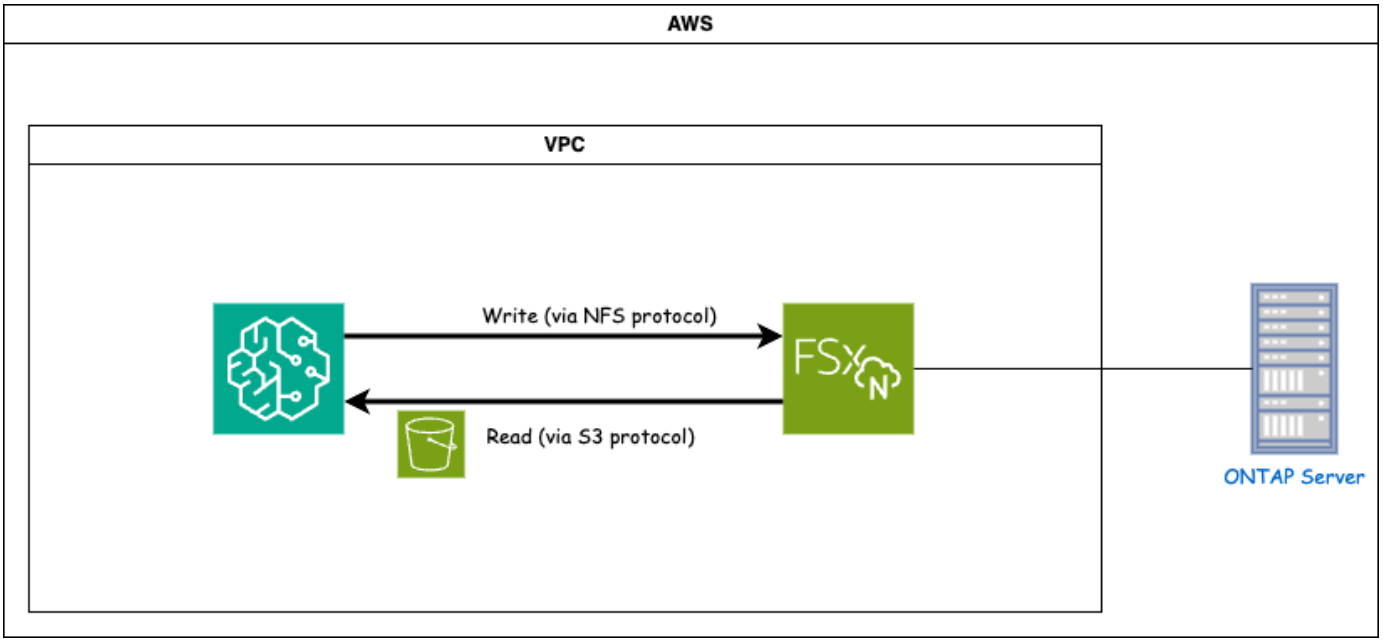
Ce didacticiel présente un exemple pratique de projet de classification de la vision par ordinateur. Il fournit une expérience pratique de la création de modèles de ML utilisant FSxN comme source de données dans l'environnement SageMaker. Le projet se concentre sur l'utilisation de PyTorch, un framework de deep learning, pour classer la qualité des pneus en fonction des images. Il insiste sur le développement de modèles de machine learning utilisant FSxN comme source de données dans Amazon SageMaker.

Qu'est-ce que FSxN

Amazon FSX pour NetApp ONTAP est une solution de stockage entièrement gérée proposée par AWS. Il exploite le système de fichiers ONTAP de NetApp pour fournir un stockage fiable et haute performance. Grâce à la prise en charge de protocoles comme NFS, SMB et iSCSI, il permet un accès transparent à partir de plusieurs instances de calcul et conteneurs. Ce service est conçu pour fournir des performances exceptionnelles garantissant des opérations de données rapides et efficaces. En outre, il offre une haute disponibilité et une durabilité élevées, assurant l'accessibilité et la protection de vos données. De plus, la capacité de stockage d'Amazon FSX for NetApp ONTAP est évolutive et vous permet de l'ajuster facilement en fonction de vos besoins.

Condition préalable

Environnement réseau



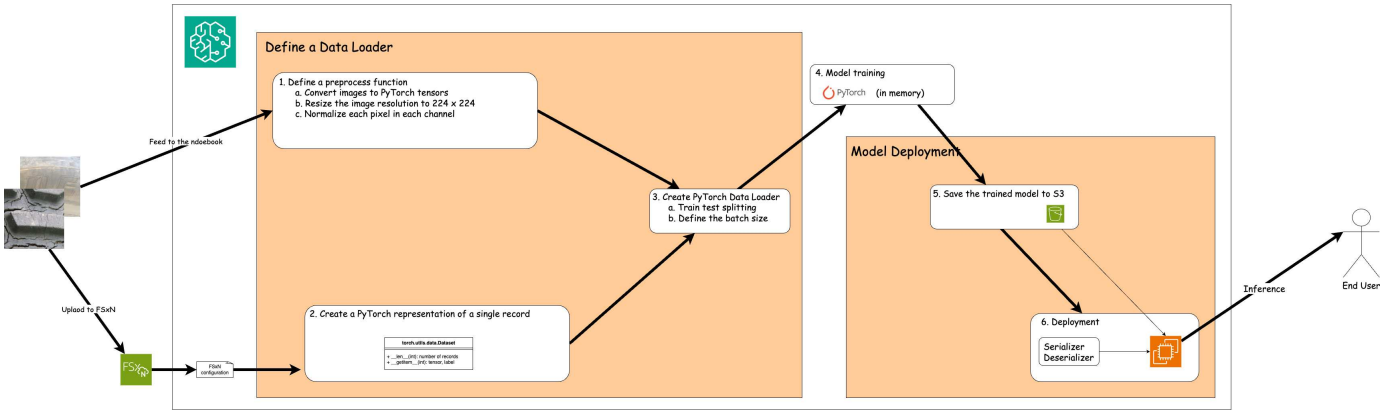
FSxN (Amazon FSX pour NetApp ONTAP) est un service de stockage AWS. Elle comprend un système de fichiers s'exécutant sur le système NetApp ONTAP et une machine virtuelle (SVM) gérée par AWS qui se connecte à celui-ci. Dans le diagramme fourni, le serveur NetApp ONTAP géré par AWS se trouve en dehors du VPC. Le SVM sert d'intermédiaire entre SageMaker et le système NetApp ONTAP, en recevant les demandes d'opération de SageMaker et en les transférant vers le stockage sous-jacent. Pour accéder à FSxN, SageMaker doit être placé dans le même VPC que le déploiement FSxN. Cette configuration assure la communication et l'accès aux données entre SageMaker et FSxN.

Accès aux données

Dans des scénarios réels, les data Scientists utilisent généralement les données stockées dans FSxN pour créer leurs modèles de machine learning. Toutefois, à des fins de démonstration, puisque le système de fichiers FSxN est initialement vide après la création, il est nécessaire de télécharger manuellement les données de formation. Ceci peut être réalisé en montant FSxN en tant que volume sur SageMaker. Une fois le système de fichiers correctement monté, vous pouvez télécharger votre dataset à l'emplacement monté, le rendant accessible pour l'entraînement de vos modèles dans l'environnement SageMaker. Cette approche vous permet de tirer parti de la capacité de stockage et des fonctionnalités de FSxN tout en travaillant avec SageMaker pour le développement de modèles et la formation.

Le processus de lecture des données implique la configuration de FSxN en tant que compartiment S3 privé. Pour connaître les instructions de configuration détaillées, reportez-vous à la section "1re partie : intégration d'AWS FSX pour NetApp ONTAP (FSxN) en tant que compartiment S3 privé dans AWS SageMaker"

Présentation de l'intégration



Le workflow d'utilisation des données d'entraînement dans FSxN pour créer un modèle de deep learning dans SageMaker peut être résumé en trois étapes principales : la définition du chargeur de données, l'entraînement du modèle et le déploiement. À un niveau élevé, ces étapes constituent la base d'un pipeline MLOps. Cependant, chaque étape implique plusieurs sous-étapes détaillées pour une mise en œuvre complète. Ces sous-étapes englobent diverses tâches, telles que le prétraitement des données, la division des datasets, la configuration des modèles, le réglage des hyperparamètres, l'évaluation des modèles, et le déploiement des modèles. Ces étapes permettent de mettre en place un processus complet et efficace pour créer et déployer des modèles de deep learning à l'aide des données d'entraînement de FSxN dans l'environnement SageMaker.

Intégration pas à pas

Chargeur de données

Afin d'entraîner un réseau de deep learning PyTorch avec des données, un chargeur de données est créé pour faciliter l'alimentation en données. Le chargeur de données définit non seulement la taille du lot, mais

détermine également la procédure de lecture et de prétraitement de chaque enregistrement dans le lot. Grâce à la configuration du chargeur de données, nous pouvons traiter les données par lots, ce qui permet l'entraînement du réseau de deep learning.

Le chargeur de données se compose de 3 parties.

Fonction de prétraitement

```
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]
    )
])
```

L'extrait de code ci-dessus illustre la définition des transformations de prétraitement d'image à l'aide du module **torchvision.Transforms**. Dans ce schéma, l'objet de prétraitement est créé pour appliquer une série de transformations. Tout d'abord, la transformation **ToTensor()** convertit l'image en représentation de tenseur. Par la suite, la transformation **Resize 224,224** redimensionne l'image à une taille fixe de 224x224 pixels. Enfin, la transformation **Normalize()** normalise les valeurs de tenseurs en soustrayant la moyenne et en divisant par l'écart-type le long de chaque canal. Les valeurs d'écart moyen et standard utilisées pour la normalisation sont généralement utilisées dans les modèles de réseaux neuronaux pré-entraînés. Dans l'ensemble, ce code prépare les données d'image pour un traitement ou une entrée ultérieurs dans un modèle pré-entraîné en les convertissant en tenseur, en les redimensionnant et en normalisant les valeurs de pixels.

Classe de jeu de données PyTorch

```
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
```

Cette classe offre des fonctionnalités permettant d'obtenir le nombre total d'enregistrements dans le jeu de données et définit la méthode de lecture des données pour chaque enregistrement. Dans la fonction ***getitem***, le code utilise l'objet de compartiment S3 boto3 pour extraire les données binaires de FSxN. Le style de code pour l'accès aux données à partir de FSxN est similaire à celui pour la lecture des données à partir d'Amazon S3. L'explication suivante est intégrée au processus de création de l'objet privé S3 **bucket**.

FSxN en tant que référentiel S3 privé

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

```
import boto3

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

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

Pour lire les données de FSxN dans SageMaker, un gestionnaire est créé et pointe vers le stockage FSxN à l'aide du protocole S3. Ainsi, FSxN peut être traité comme un compartiment S3 privé. La configuration du gestionnaire inclut la spécification de l'adresse IP du SVM FSxN, du nom du compartiment et des informations d'identification nécessaires. Pour obtenir une explication complète sur l'obtention de ces éléments de configuration, reportez-vous au document à l'adresse "1re partie : intégration d'AWS FSX pour NetApp ONTAP (FSxN) en tant que compartiment S3 privé dans AWS SageMaker".

Dans l'exemple mentionné ci-dessus, l'objet de compartiment est utilisé pour instancier l'objet de jeu de données PyTorch. L'objet Dataset sera expliqué plus en détail dans la section suivante.

Le chargeur de données PyTorch

```
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)
```

Dans l'exemple fourni, une taille de lot de 64 est spécifiée, indiquant que chaque lot contiendra 64 enregistrements. En combinant la classe PyTorch **Dataset**, la fonction de prétraitement et la taille du lot d'entraînement, nous obtenons le chargeur de données pour l'entraînement. Ce chargeur de données facilite le processus d'itération dans l'ensemble de données en lots pendant la phase d'entraînement.

Entraînement du modèle

```
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')
```

Ce code met en œuvre un processus de formation PyTorch standard. Il définit un modèle de réseau neuronal appelé **TireQualityClassifier** utilisant des couches convolutionnelles et une couche linéaire pour classer la qualité des pneus. La boucle d'entraînement effectue une itération sur les lots de données, calcule la perte et met à jour les paramètres du modèle à l'aide de la rétropropagation et de l'optimisation. En outre, il imprime l'heure, l'époque, le lot et la perte actuels à des fins de surveillance.

Déploiement du modèle

Déploiement

```
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())
```

Le code enregistre le modèle PyTorch dans **Amazon S3** car SageMaker requiert que le modèle soit stocké dans S3 pour le déploiement. En téléchargeant le modèle vers **Amazon S3**, il devient accessible à SageMaker, ce qui permet le déploiement et l'inférence sur le modèle déployé.

```
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
)
```

Ce code facilite le déploiement d'un modèle PyTorch sur SageMaker. Il définit un sérialiseur personnalisé, **TireQualitySerializer**, qui prétraite et sérialise les données d'entrée en tant que tenseur PyTorch. La classe **TireQualityPredictor** est un prédicteur personnalisé qui utilise le sérialiseur défini et un **JSONDeserializer**. Le code crée également un objet **PyTorchModel** pour spécifier l'emplacement S3 du modèle, le rôle IAM, la version du framework et le point d'entrée pour l'inférence. Le code génère un horodatage et construit un nom

de point final basé sur le modèle et l'horodatage. Enfin, le modèle est déployé à l'aide de la méthode deploy, en spécifiant le nombre d'instances, le type d'instance et le nom du noeud final généré. Cela permet de déployer le modèle PyTorch et d'y accéder pour l'inférence sur SageMaker.

Inférence

```
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)
```

Voici un exemple d'utilisation du terminal déployé pour effectuer l'inférence.

Partie 3 - construire Un pipeline MLO simplifié (ci/CT/CD)

Auteur(s):

Jian Jian (Ken), scientifique senior en données et applications, NetApp

Introduction

Dans ce tutoriel, vous apprendrez à utiliser différents services AWS pour construire un pipeline MLOps simple qui englobe l'intégration continue (ci), la formation continue (CT) et le déploiement continu (CD). Contrairement aux pipelines DevOps classiques, le MLOps nécessite des considérations supplémentaires pour mener à bien le cycle opérationnel. En suivant ce tutoriel, vous allez apprendre à intégrer la tomodensitométrie dans la boucle MLOps, ce qui permet d'entraîner en continu vos modèles et de procéder à un déploiement transparent pour l'inférence. Ce tutoriel vous guidera tout au long du processus d'utilisation des services AWS pour établir ce pipeline MLOps de bout en bout.

Manifeste

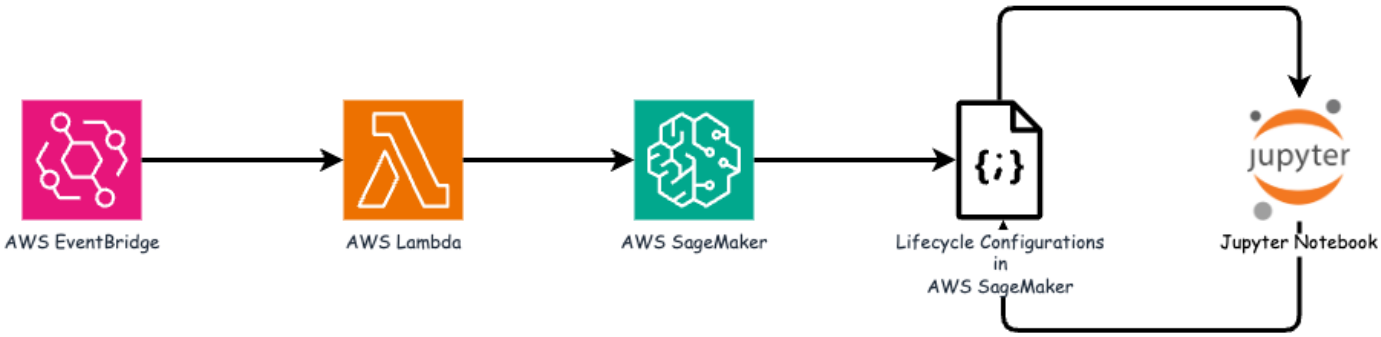
Fonctionnalité	Nom	Commentaire
Stockage des données	FSxN AWS	Reportez-vous à la section "1re partie : intégration d'AWS FSX pour NetApp ONTAP (FSxN) en tant que compartiment S3 privé dans AWS SageMaker".
IDE data science	AWS SageMaker	Ce tutoriel est basé sur le portable Jupyter présenté dans "2e partie - exploitation d'AWS FSX pour NetApp ONTAP (FSxN) en tant que source de données pour l'entraînement des modèles dans SageMaker".
Fonction permettant de déclencher le pipeline MLOps	Fonction Lambda d'AWS	-

Fonctionnalité	Nom	Commentaire
Déclencheur de tâche cron	AWS EventBridge	-
Structure de deep learning	PyTorch	-
Kit de développement logiciel AWS Python	boto3	-
Langage de programmation	Python	v3.10

Condition préalable

- Un système de fichiers FSxN préconfiguré. Ce tutoriel utilise les données stockées dans FSxN pour le processus de formation.
- Instance **SageMaker Notebook** configurée pour partager le même VPC que le système de fichiers FSxN mentionné ci-dessus.
- Avant de déclencher la fonction **AWS Lambda**, assurez-vous que l'instance **SageMaker Notebook** est à l'état **Arrêté**.
- Le type d'instance **ml.g4dn.xlarge** est requis pour exploiter l'accélération GPU nécessaire au calcul des réseaux neuronaux profonds.

Architecture



Ce pipeline MLOps est une implémentation pratique qui utilise un travail cron pour déclencher une fonction sans serveur, qui à son tour exécute un service AWS enregistré avec une fonction de rappel de cycle de vie. **AWS EventBridge** agit comme travail cron. Il invoque périodiquement une fonction **AWS Lambda** responsable du recyclage et du redéploiement du modèle. Ce processus implique l'exécution de l'instance **AWS SageMaker Notebook** pour effectuer les tâches nécessaires.

Configuration pas à pas

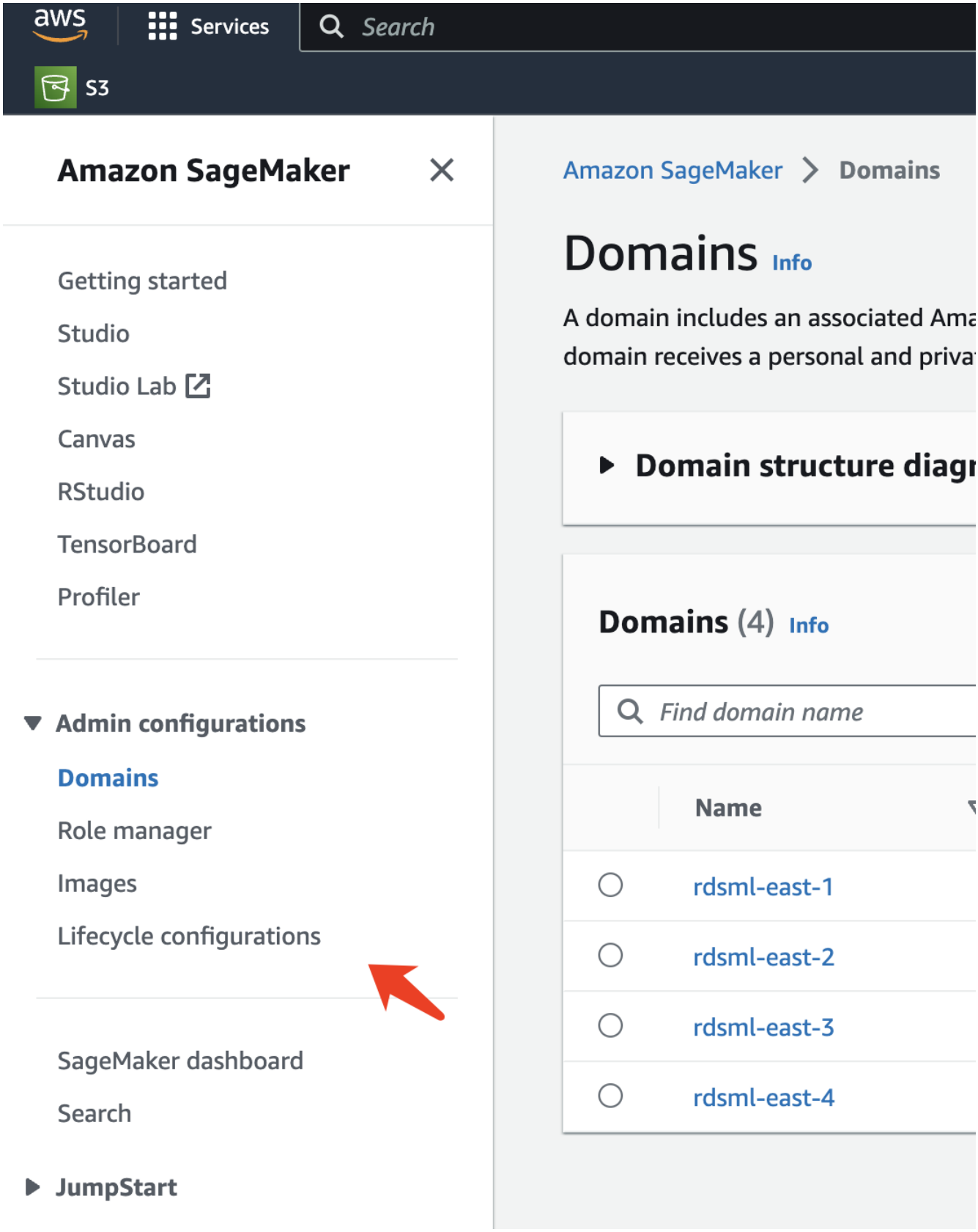
Configurations de cycle de vie

Pour configurer la fonction de rappel de cycle de vie pour l'instance d'ordinateur portable AWS SageMaker, vous devez utiliser **Lifecycle configurations**. Ce service vous permet de définir les actions à effectuer lors de l'activation de l'instance de bloc-notes. Plus précisément, un script shell peut être implémenté dans les configurations **Lifecycle** pour arrêter automatiquement l'instance de bloc-notes une fois les processus de formation et de déploiement terminés. Il s'agit d'une configuration requise, car le coût est l'un des principaux éléments à prendre en compte dans MLOps.

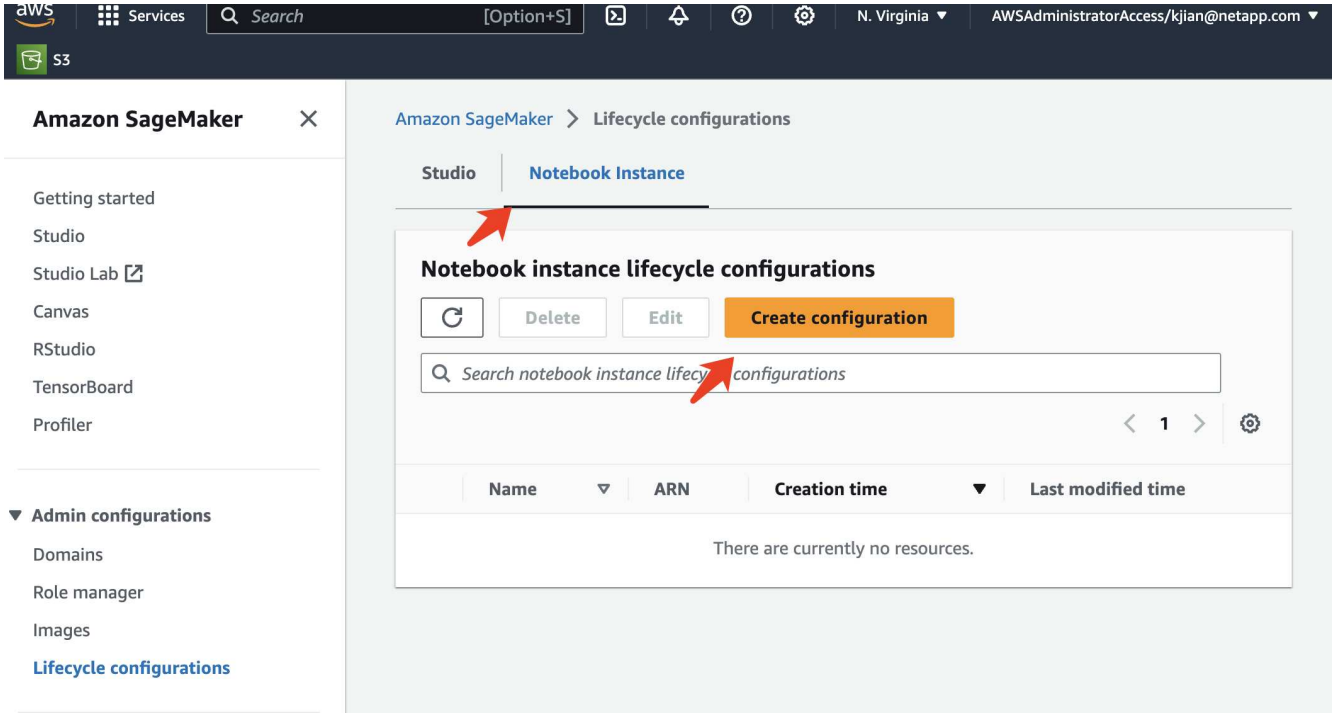
Il est important de noter que la configuration des **configurations Lifecycle** doit être configurée à l'avance. Par

conséquent, il est recommandé de hiérarchiser la configuration de cet aspect avant de procéder à la configuration des autres pipelines MLOps.

1. Pour configurer une configuration Lifecycle, ouvrez le panneau **Sagemaker** et naviguez jusqu'à **Lifecycle configurations** sous la section **Admin configurations**.



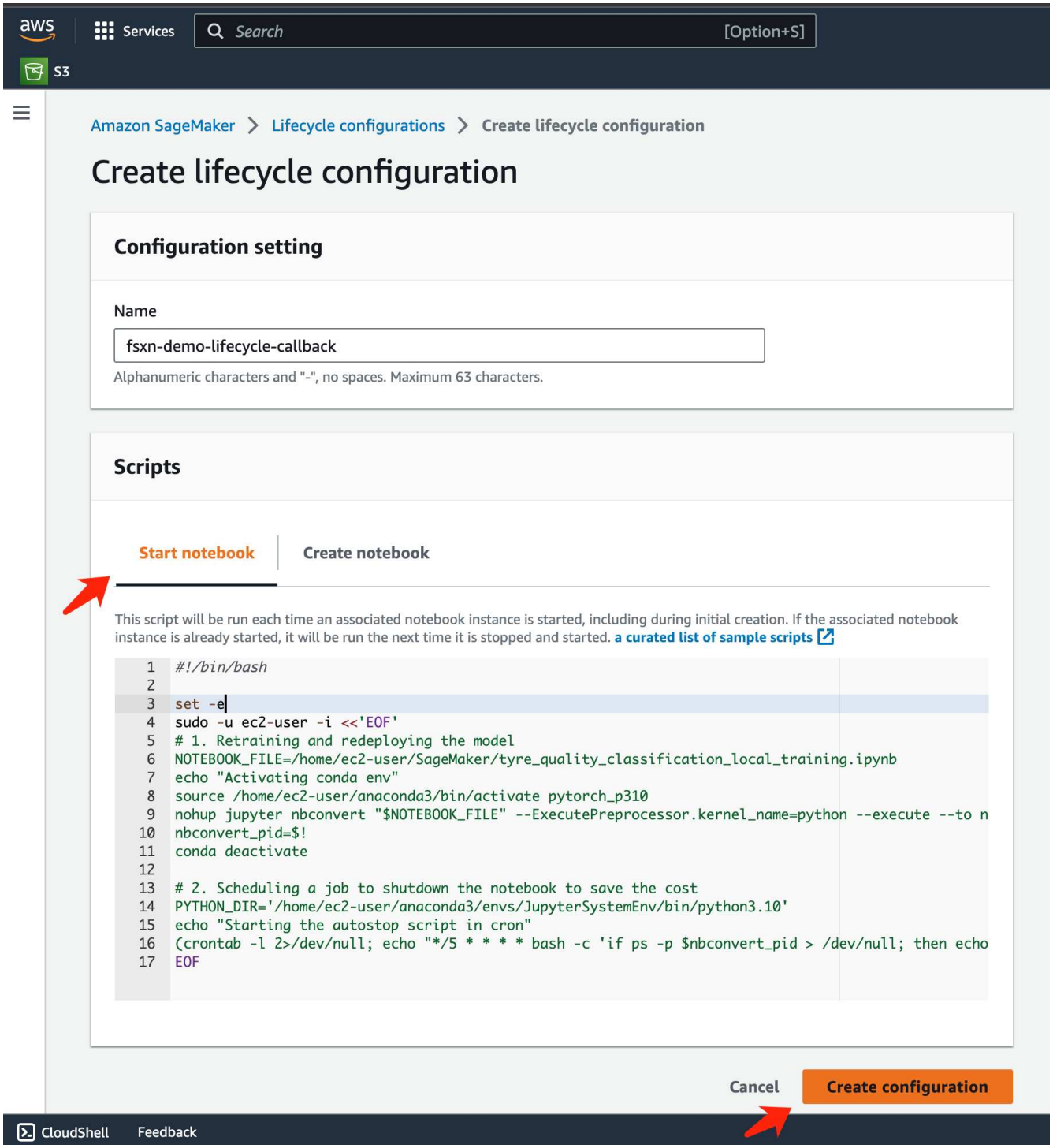
2. Sélectionnez l'onglet **Notebook instance** et cliquez sur le bouton **Créer une configuration**



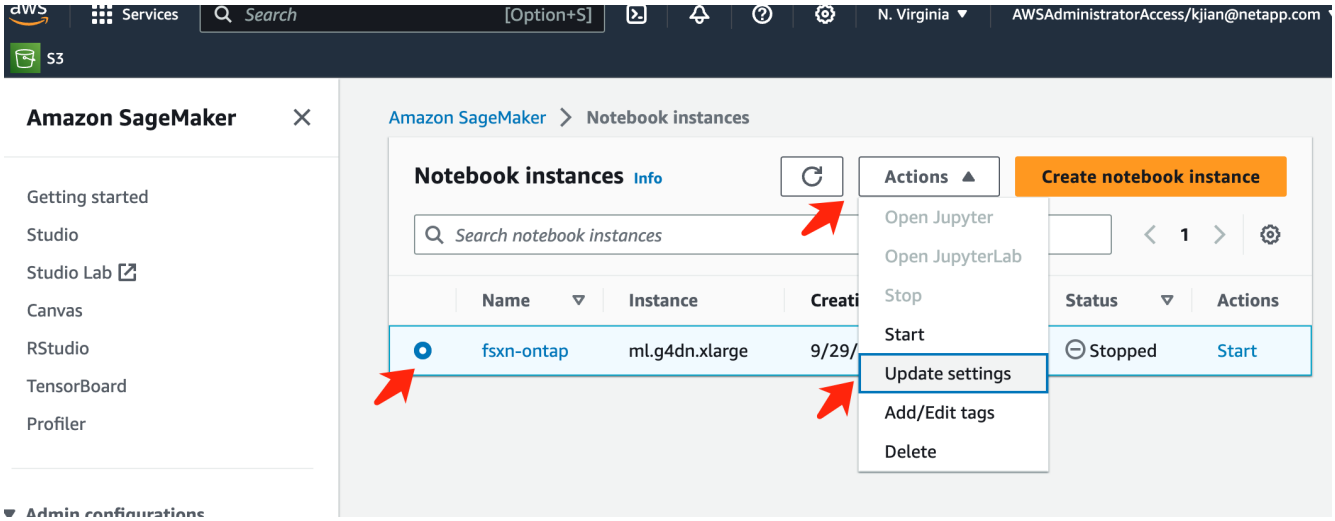
3. Collez le code ci-dessous dans la zone de saisie.

```
#!/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
```

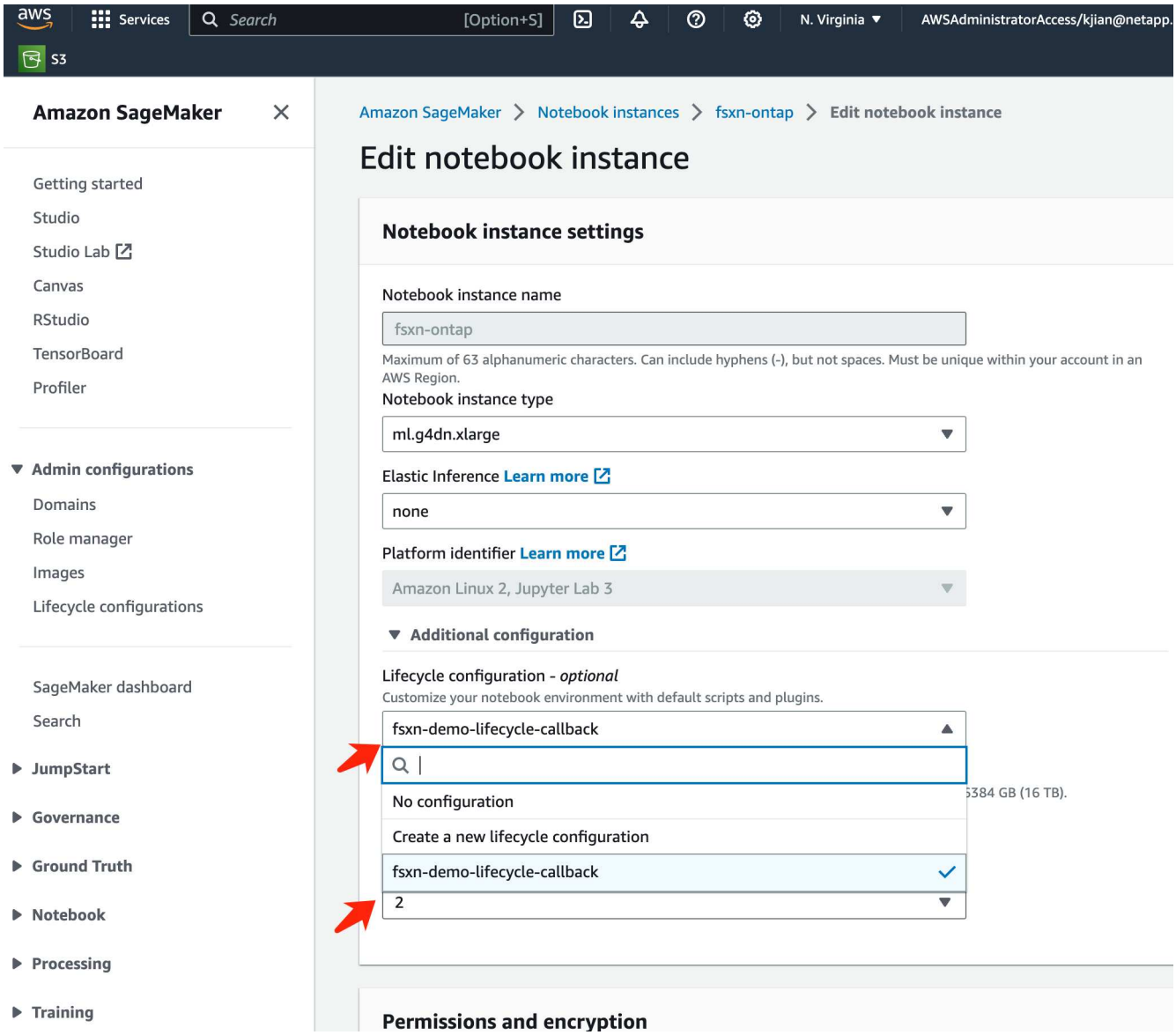
4. Ce script exécute le Jupyter Notebook, qui gère le recyclage et le redéploiement du modèle pour l'inférence. Une fois l'exécution terminée, l'ordinateur s'arrête automatiquement dans les 5 minutes. Pour en savoir plus sur l'énoncé du problème et l'implémentation du code, reportez-vous à la section "2e partie - exploitation d'AWS FSX pour NetApp ONTAP (FSxN) en tant que source de données pour l'entraînement des modèles dans SageMaker".



5. Après la création, naviguez jusqu'à instances de bloc-notes, sélectionnez l'instance cible, puis cliquez sur **mettre à jour les paramètres** dans la liste déroulante actions.



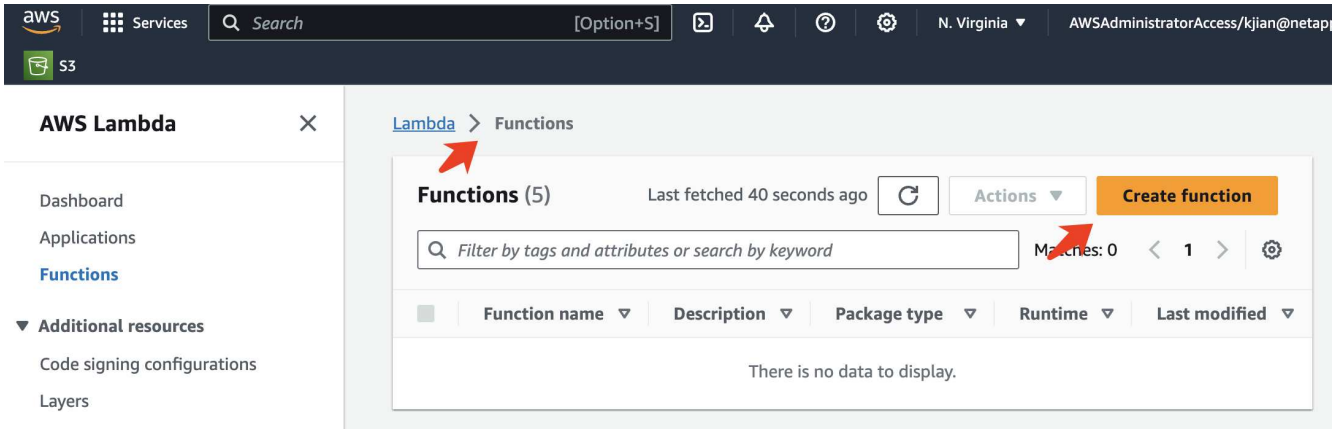
6. Sélectionnez la configuration **Lifecycle** créée et cliquez sur **mettre à jour l'instance de bloc-notes**.



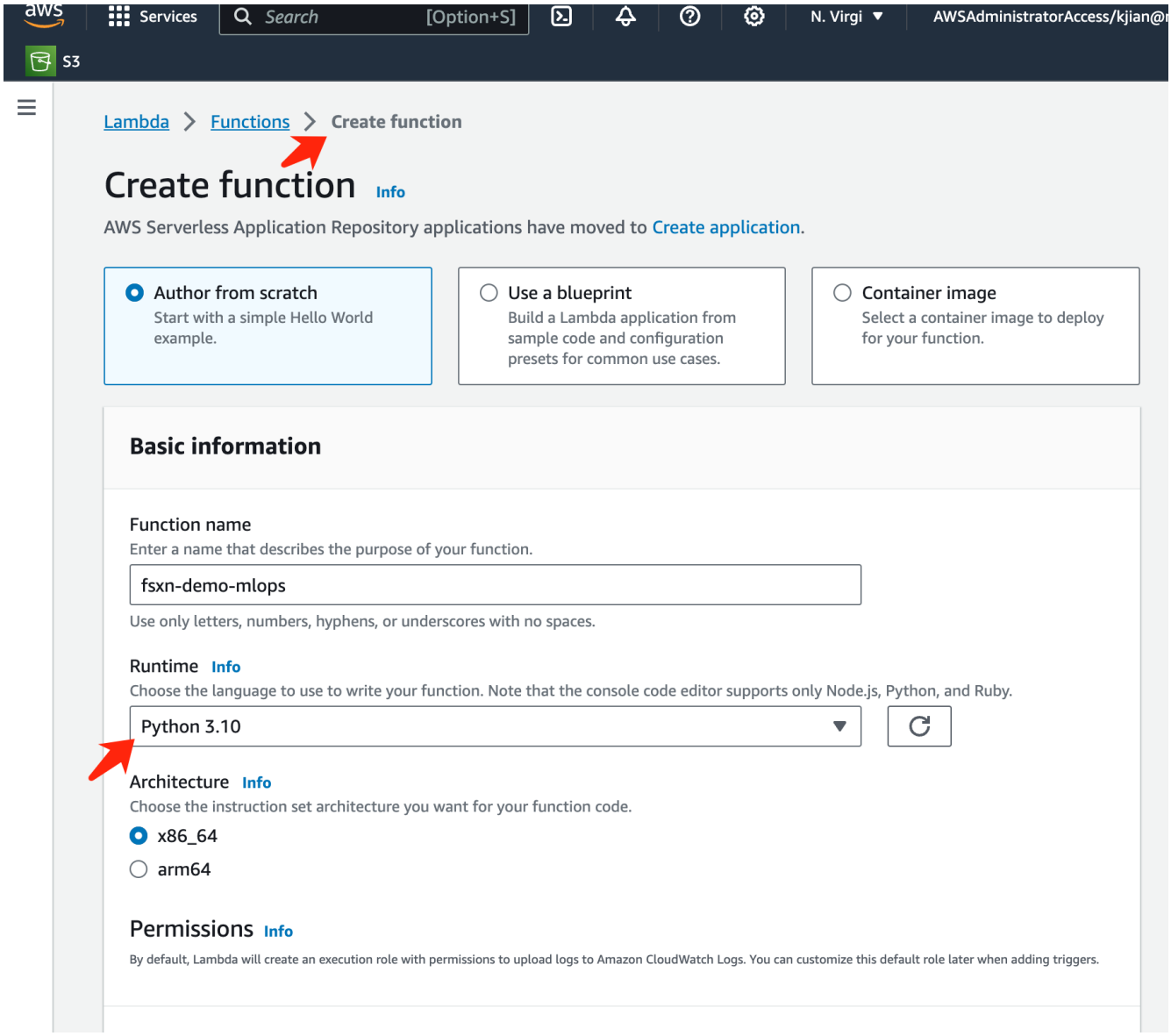
Fonction sans serveur AWS Lambda

Comme mentionné précédemment, la fonction **AWS Lambda** est responsable de l'activation de l'instance **AWS SageMaker Notebook**.

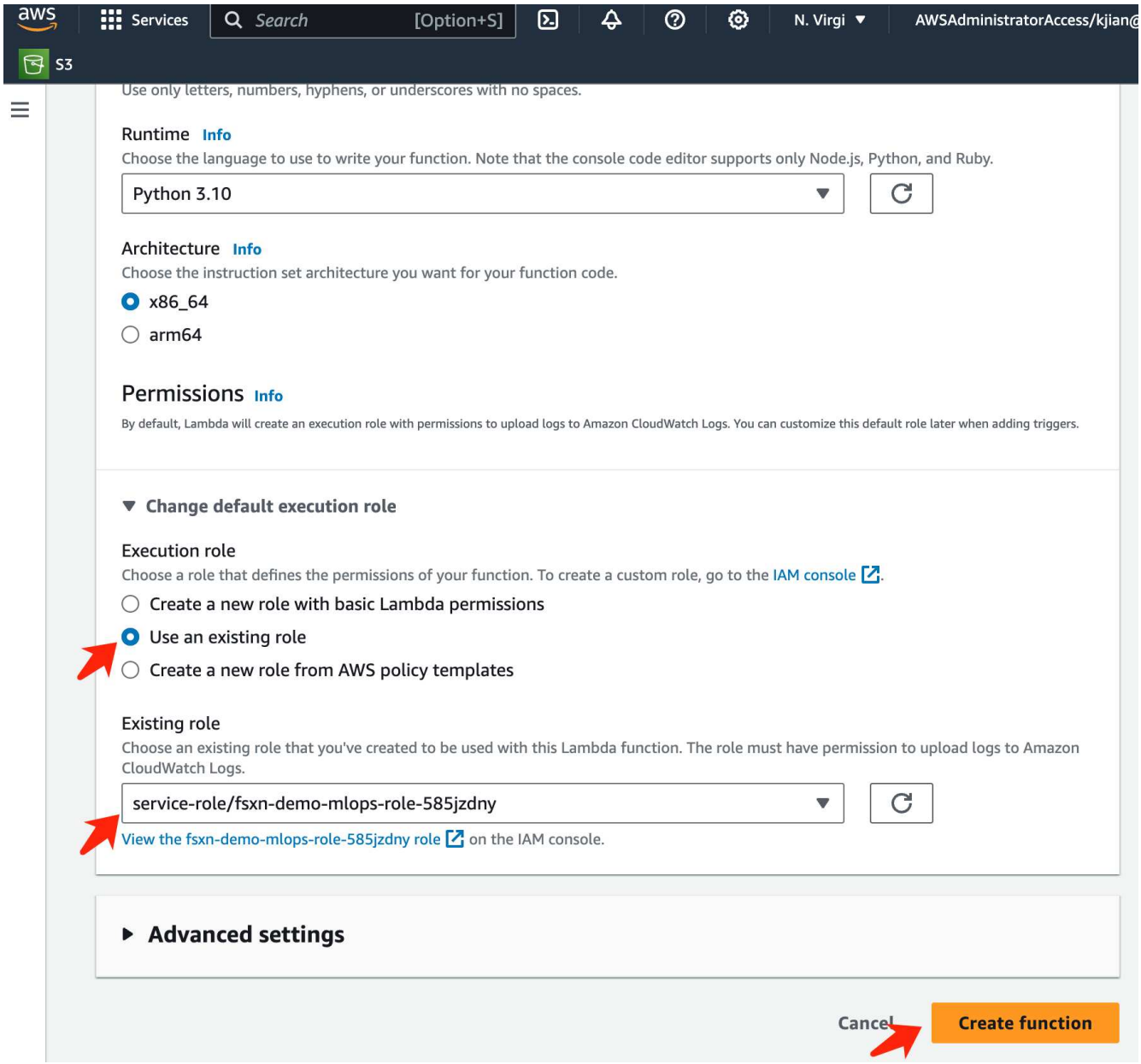
1. Pour créer une fonction **AWS Lambda**, accédez au panneau correspondant, passez à l'onglet **Functions** et cliquez sur **Create Function**.



2. Veuillez classer toutes les entrées requises sur la page et n'oubliez pas de passer à **Python 3.10**.



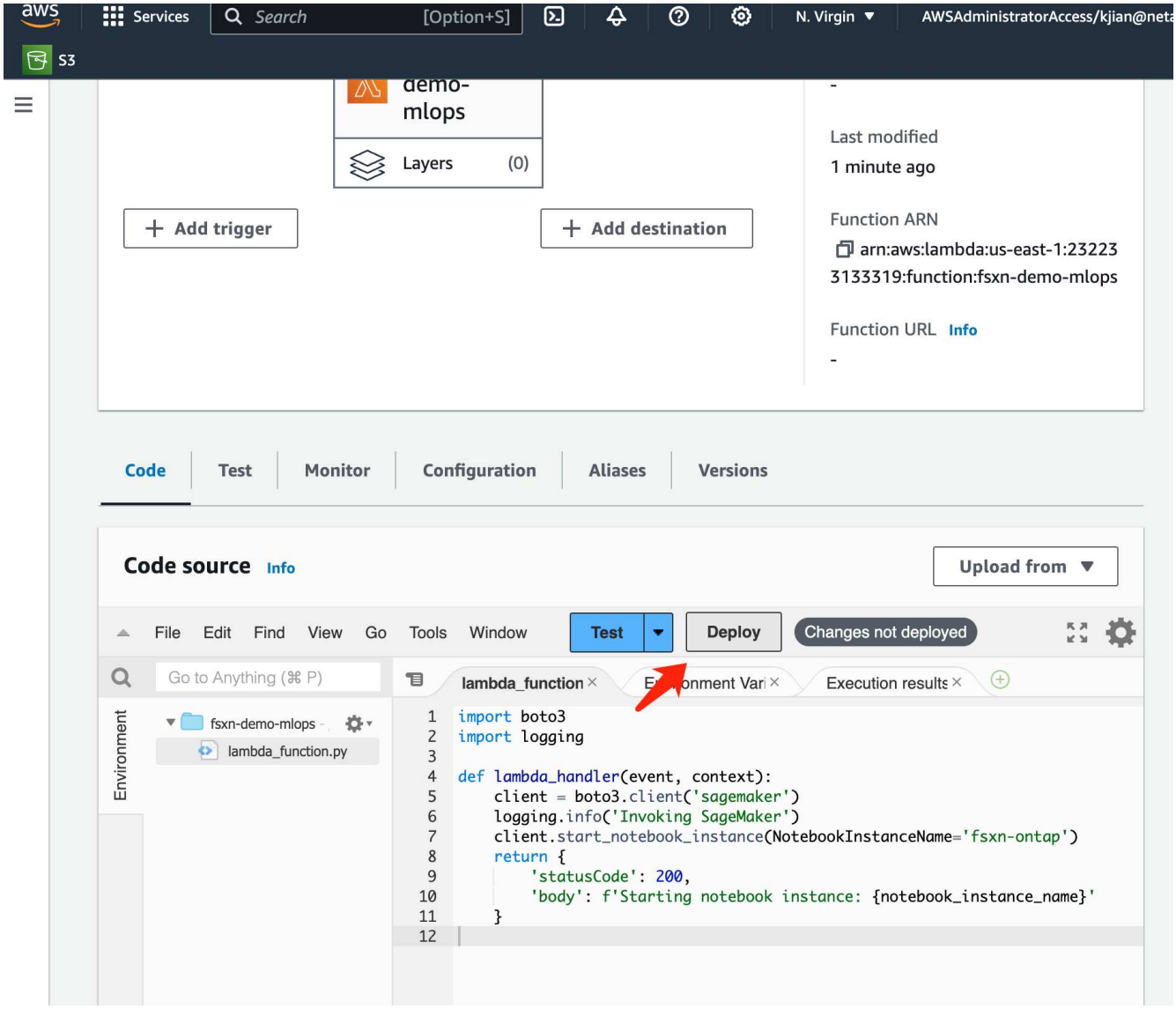
3. Veuillez vérifier que le rôle désigné possède l'autorisation requise **AmazonSageMakerFullAccess** et cliquez sur le bouton **Créer fonction**.



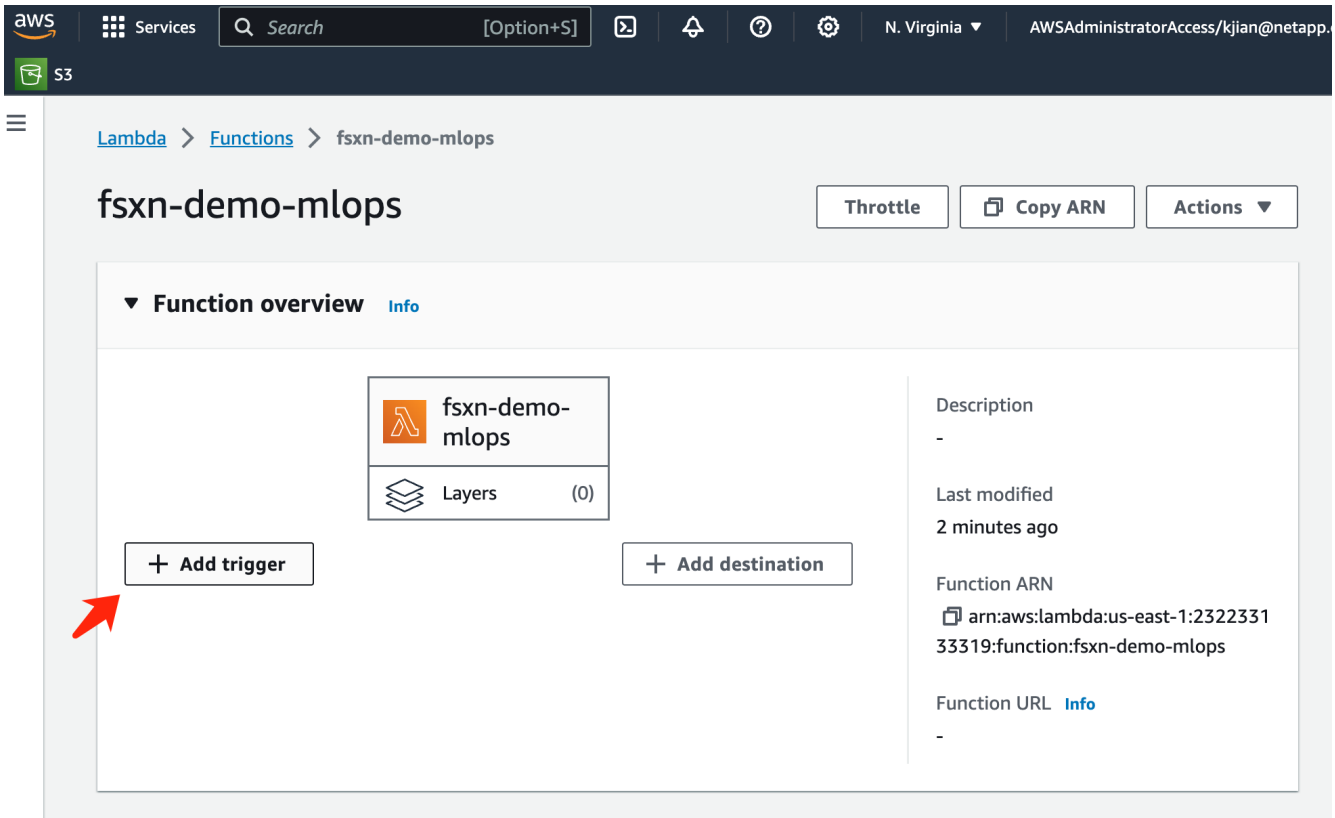
4. Sélectionnez la fonction Lambda créée. Dans l'onglet Code, copiez et collez le code suivant dans la zone de texte. Ce code démarre l'instance d'ordinateur portable nommée **fsxn-ontap**.

```
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}'
    }
```

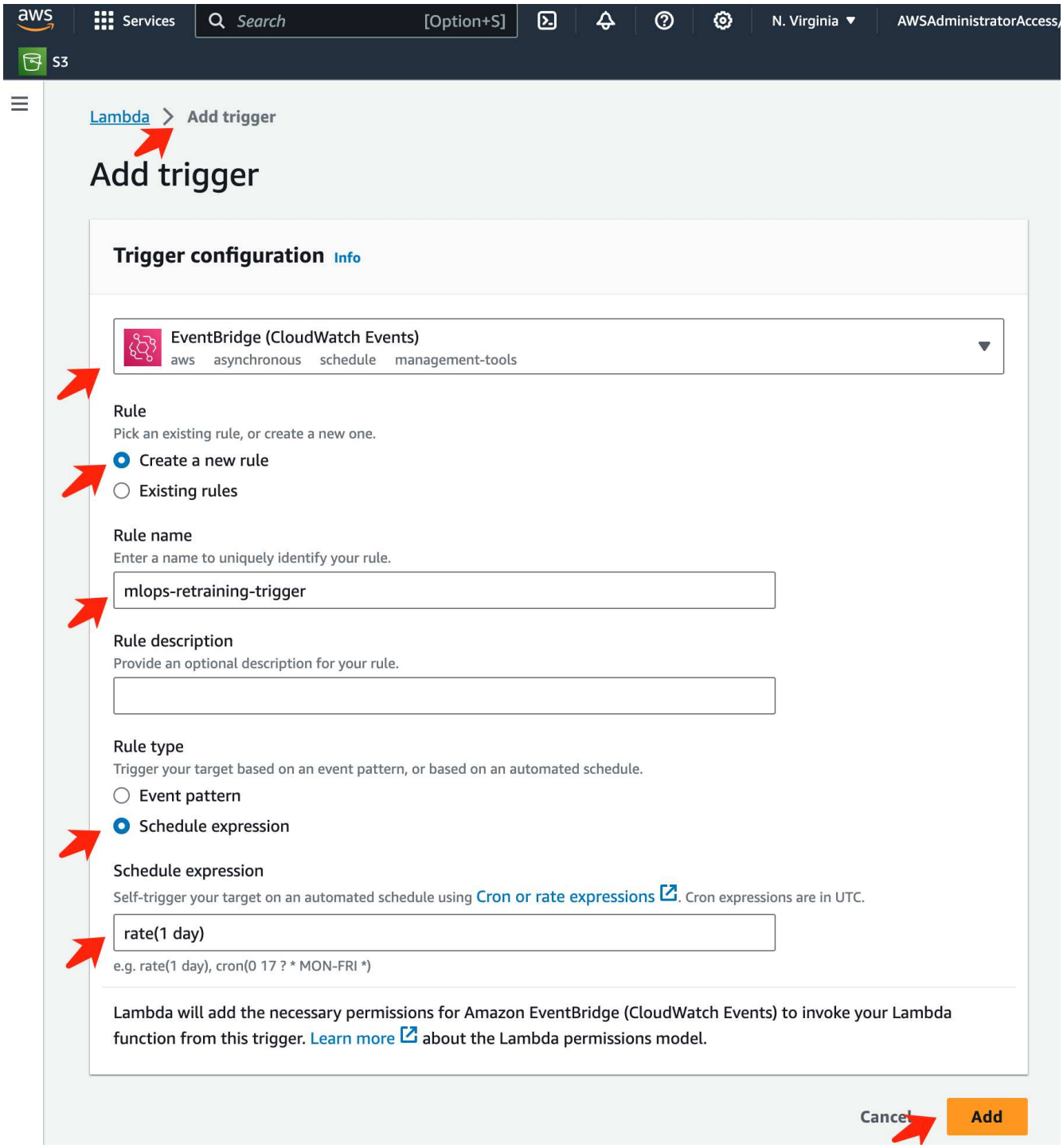
5. Cliquez sur le bouton **déployer** pour appliquer ce changement de code.



6. Pour spécifier comment déclencher cette fonction Lambda d'AWS, cliquez sur le bouton Ajouter un déclencheur.



7. Sélectionnez EventBridge dans le menu déroulant, puis cliquez sur le bouton radio Créer une nouvelle règle. Dans le champ expression du programme, entrez `rate(1 day)`, Puis cliquez sur le bouton Ajouter pour créer et appliquer cette nouvelle règle de travail cron à la fonction Lambda d'AWS.



Après avoir terminé la configuration en deux étapes, chaque jour, la fonction **AWS Lambda** lance le **SageMaker Notebook**, effectue une nouvelle formation du modèle en utilisant les données du référentiel **FSxN**, redéploie le modèle mis à jour dans l'environnement de production et arrête automatiquement l'instance **SageMaker Notebook** pour optimiser les coûts. Cela permet de s'assurer que le modèle reste à jour.

Ceci conclut le tutoriel sur le développement d'un pipeline MLOps.