FSx ONTAP per MLOps

NetApp artificial intelligence solutions

NetApp
February 12, 2026

Sommario

FSx ONTAP per MLOps . . . 1
Amazon FSx for NetApp ONTAP (FSx ONTAP) per MLOps . . . 1
Parte 1 - Integrazione di Amazon FSx for NetApp ONTAP (FSx ONTAP) come bucket S3 privato in AWS SageMaker . . . 1
Introduzione . . . 1
Guida per l’utente . . . 1
Lista di controllo utile per il debug . . . 14
FAQ (aggiornato al 27 settembre 2023) . . . 15
Parte 2 - Utilizzo di AWS Amazon FSx for NetApp ONTAP (FSx ONTAP) come origine dati per l’addestramento del modello in SageMaker . . . 15
Introduzione . . . 15
Che cos’è FSx ONTAP . . . 15
Prerequisito . . . 16
Panoramica sull’integrazione . . . 16
Integrazione passo dopo passo . . . 17
Parte 3 - Creazione di una pipeline MLOps semplificata (CI/CT/CD) . . . 24
Introduzione . . . 24
Manifesto . . . 24
Prerequisito . . . 25
Architettura . . . 25
Configurazione passo passo . . . 25

FSx ONTAP per MLOps

Amazon FSx for NetApp ONTAP (FSx ONTAP) per MLOps

Questa sezione approfondisce l'applicazione pratica dello sviluppo dell'infrastruttura AI, fornendo una guida completa alla costruzione di una pipeline MLOps utilizzando FSx ONTAP. Composto da tre esempi esaustivi, ti guida a soddisfare le tue esigenze MLOps tramite questa potente piattaforma di gestione dei dati.

Questi articoli si concentrano su:

1. "Parte 1 - Integrazione di Amazon FSx for NetApp ONTAP (FSx ONTAP) come bucket S3 privato in AWS SageMaker"
2. "Parte 2 - Utilizzo di Amazon FSx for NetApp ONTAP (FSx ONTAP) come origine dati per l'addestramento del modello in SageMaker"
3. "Parte 3 - Creazione di una pipeline MLOps semplificata (CI/CT/CD)"

Al termine di questa sezione, avrai acquisito una solida comprensione di come utilizzare FSx ONTAP per semplificare i processi MLOps.

Parte 1 - Integrazione di Amazon FSx for NetApp ONTAP (FSx ONTAP) come bucket S3 privato in AWS SageMaker

Questa sezione fornisce una guida sulla configurazione di FSx ONTAP come bucket S3 privato utilizzando AWS SageMaker.

Introduzione

Utilizzando SageMaker come esempio, questa pagina fornisce indicazioni sulla configurazione di FSx ONTAP come bucket S3 privato.

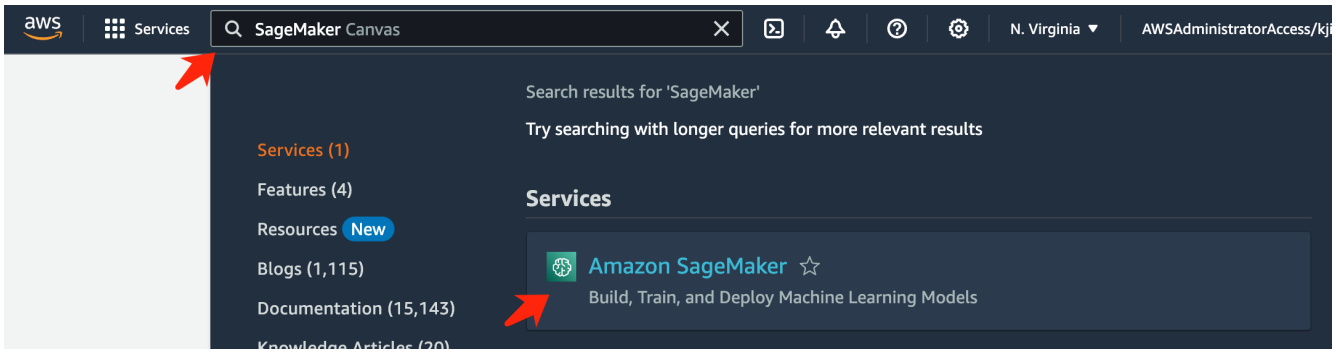
Per maggiori informazioni su FSx ONTAP, dai un'occhiata a questa presentazione ("Collegamento video")

Guida per l'utente

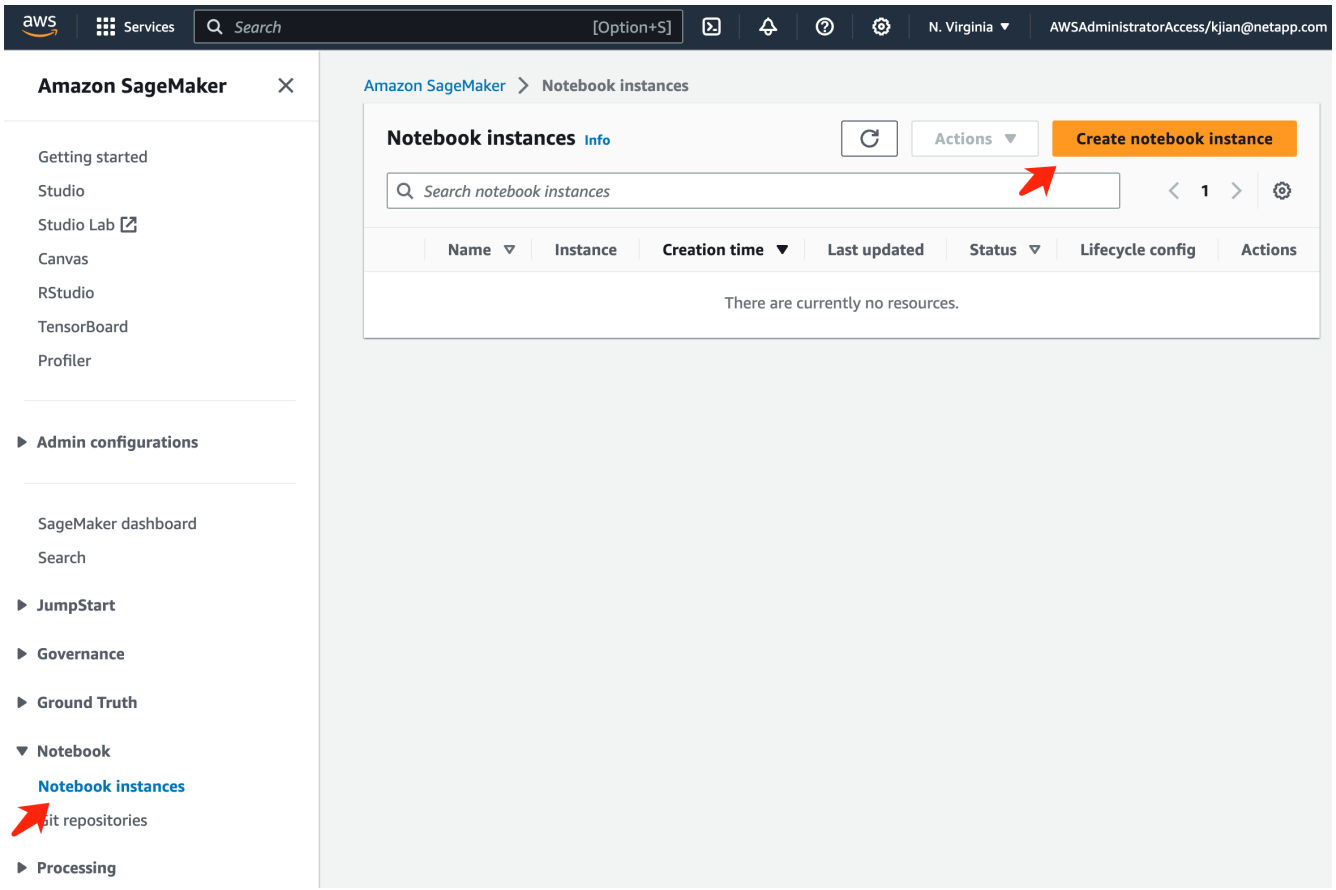
Creazione del server

Creare un'istanza di SageMaker Notebook

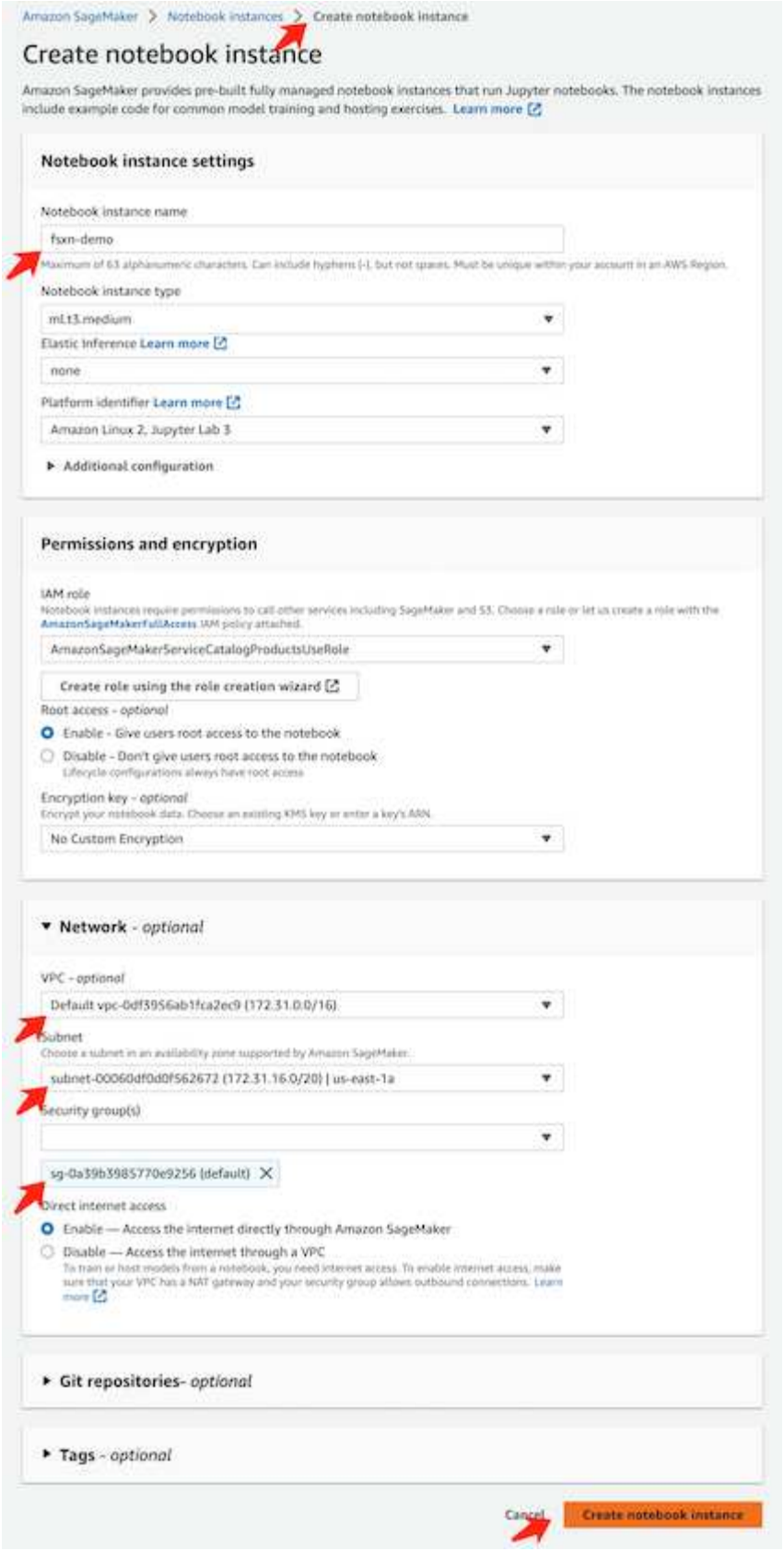
1. Apri la console AWS. Nel pannello di ricerca, cerca SageMaker e clicca sul servizio **Amazon SageMaker**.



2. Aprire le **Istanze del notebook** nella scheda Notebook, fare clic sul pulsante arancione **Crea istanza del notebook**.

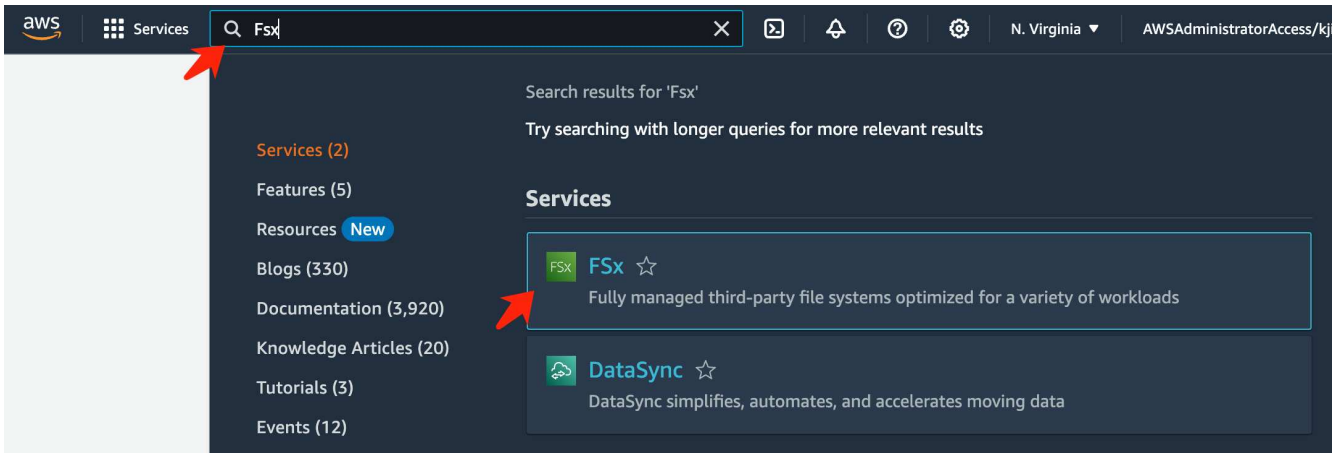


3. Nella pagina di creazione, inserisci il **Nome dell'istanza del notebook**. Espandi il pannello **Rete**. Lascia le altre voci predefinite e seleziona una **VPC**, una **Subnet** e un **Gruppo/i di sicurezza**. (Questa **VPC** e **Subnet** verranno utilizzate in seguito per creare il file system FSx ONTAP) Fare clic sul pulsante arancione **Crea istanza notebook** in basso a destra.

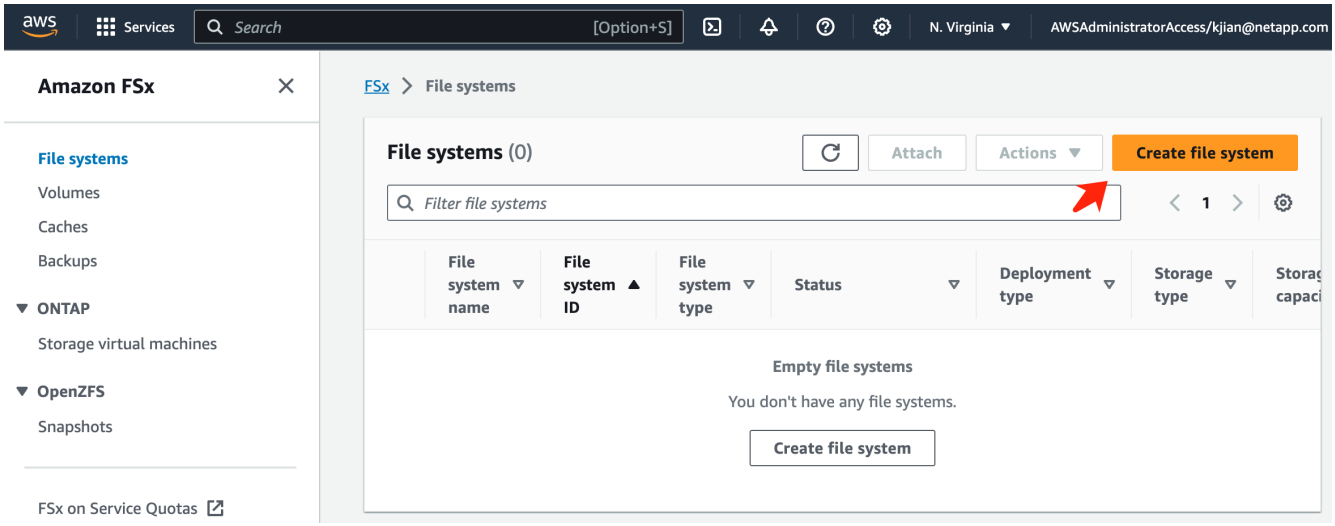


Creare un file system FSx ONTAP

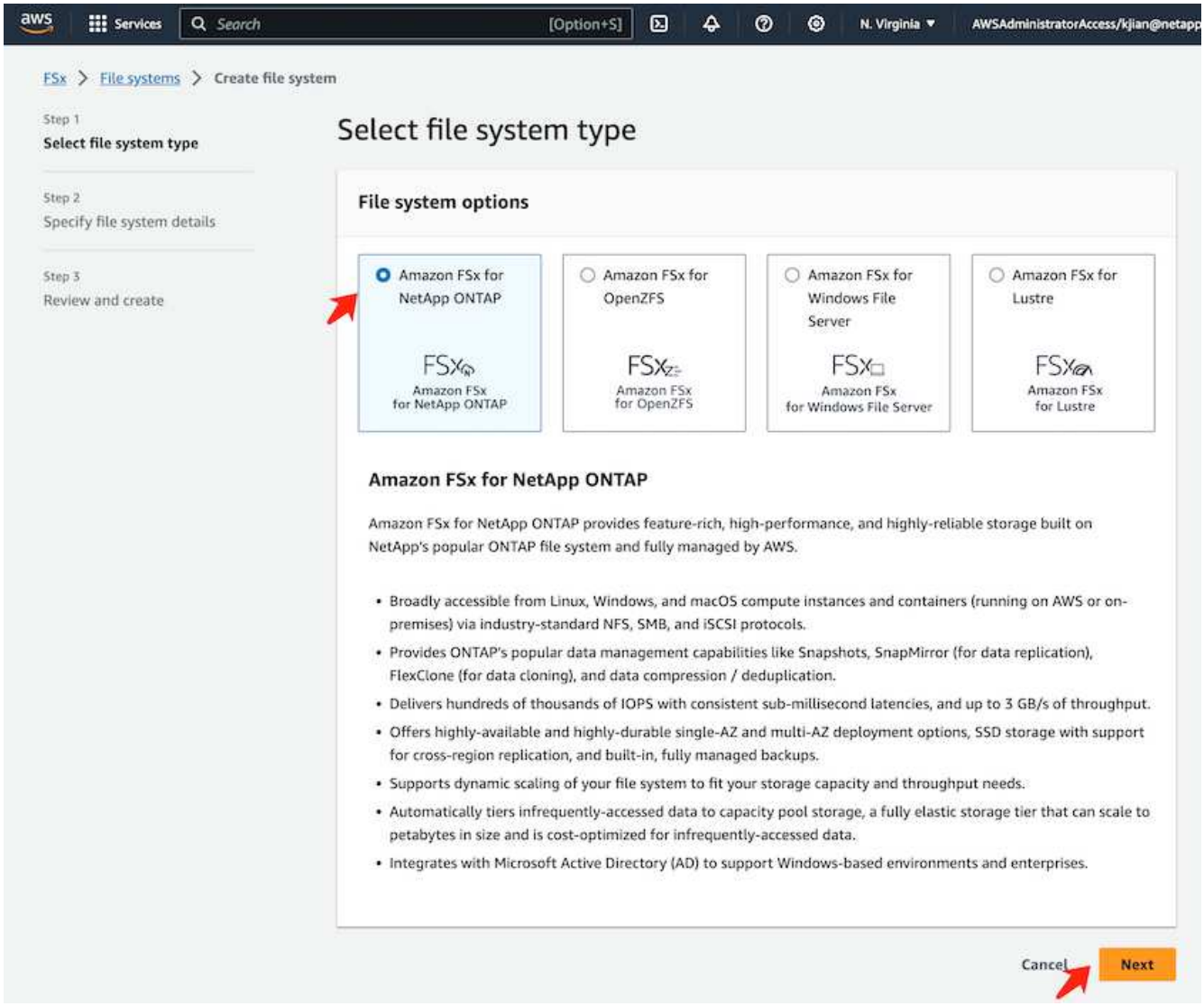
1. Apri la console AWS. Nel pannello di ricerca, cerca Fsx e clicca sul servizio **FSx**.



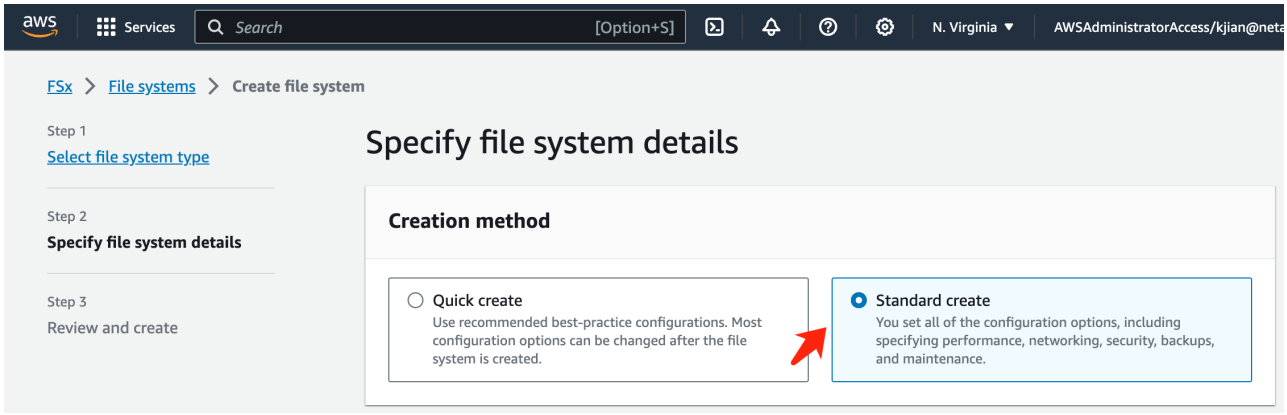
2. Fare clic su **Crea file system**.



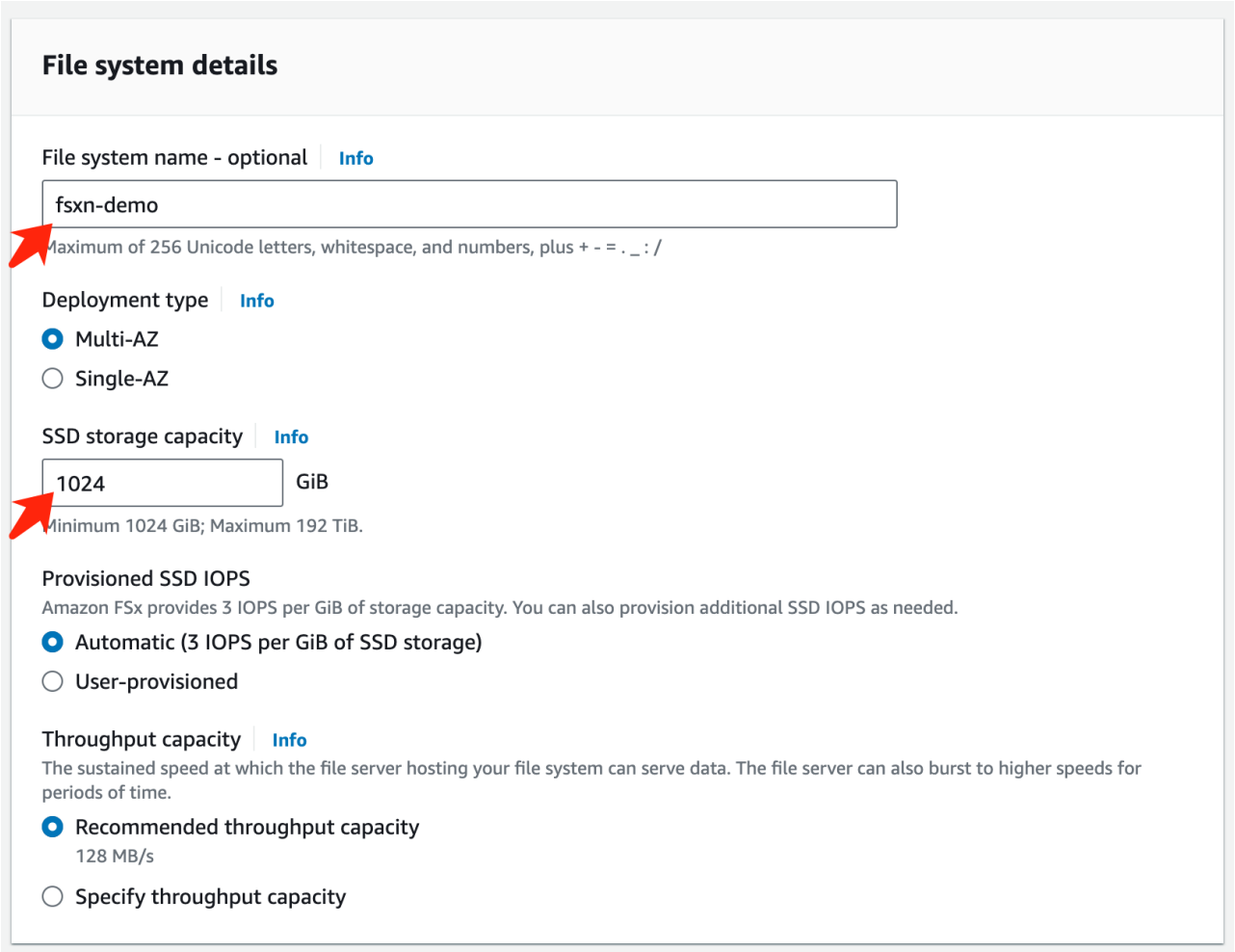
3. Selezionare la prima scheda **FSx ONTAP** e fare clic su **Avanti**.



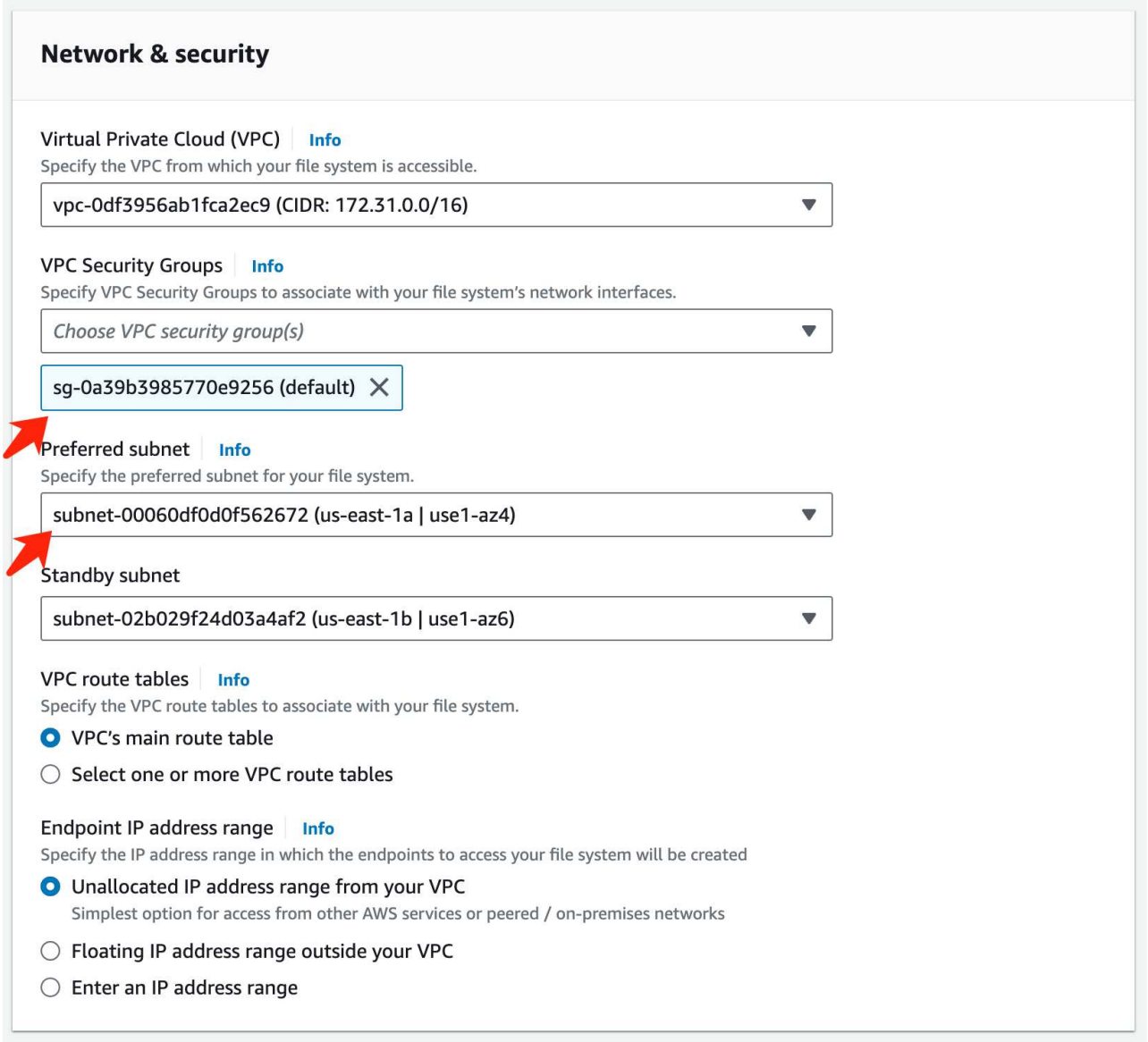
4. Nella pagina di configurazione dei dettagli.
   a. Selezionare l'opzione **Creazione standard**.



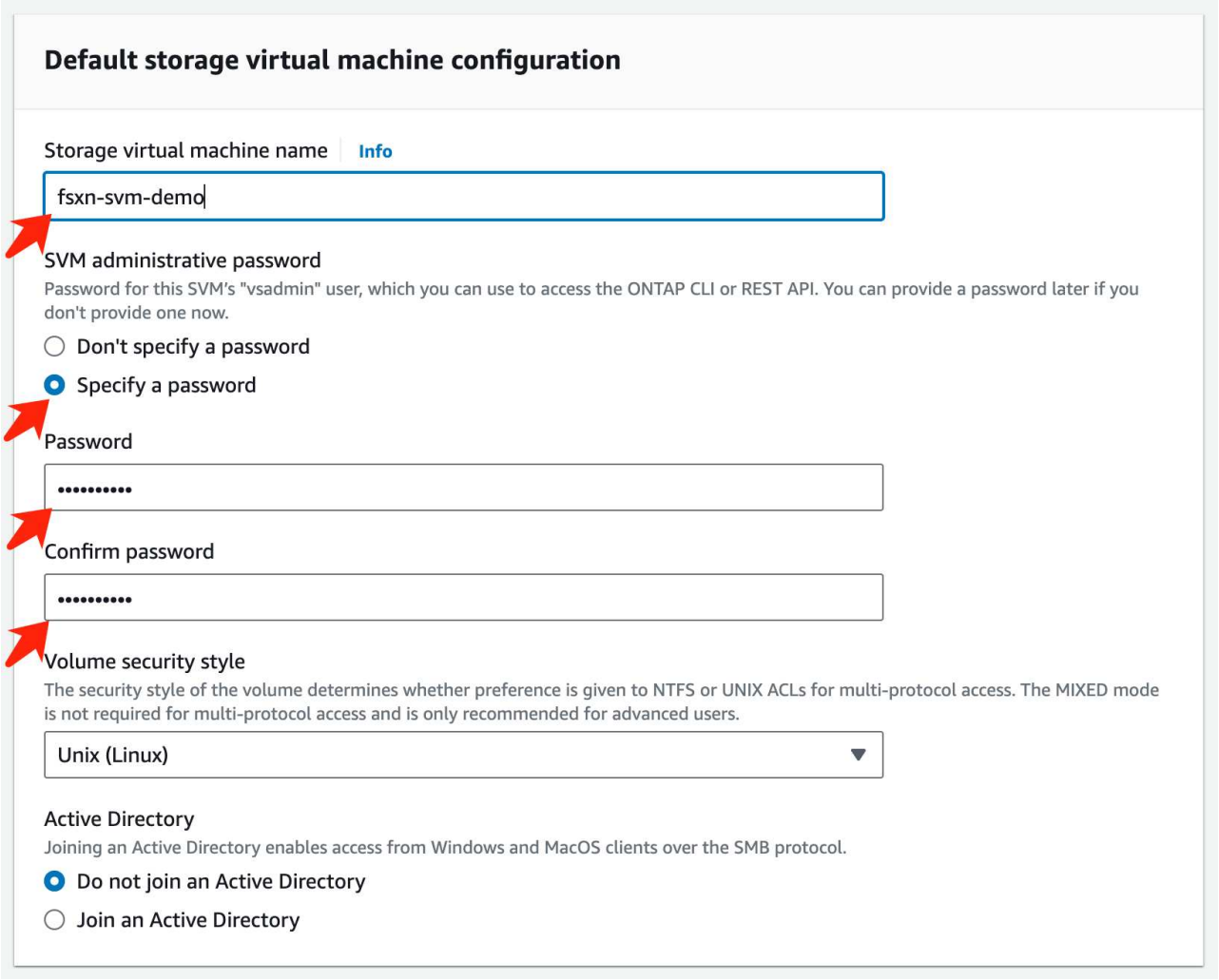
   b. Immettere il **Nome del file system** e la **Capacità di archiviazione SSD**.



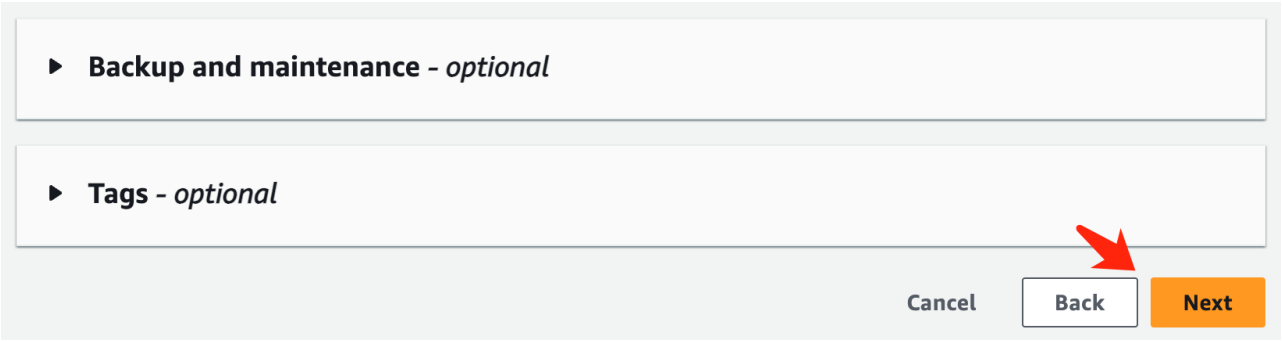
c. Assicurarsi di utilizzare la **VPC** e la **subnet** identiche per l'istanza di **SageMaker Notebook**.



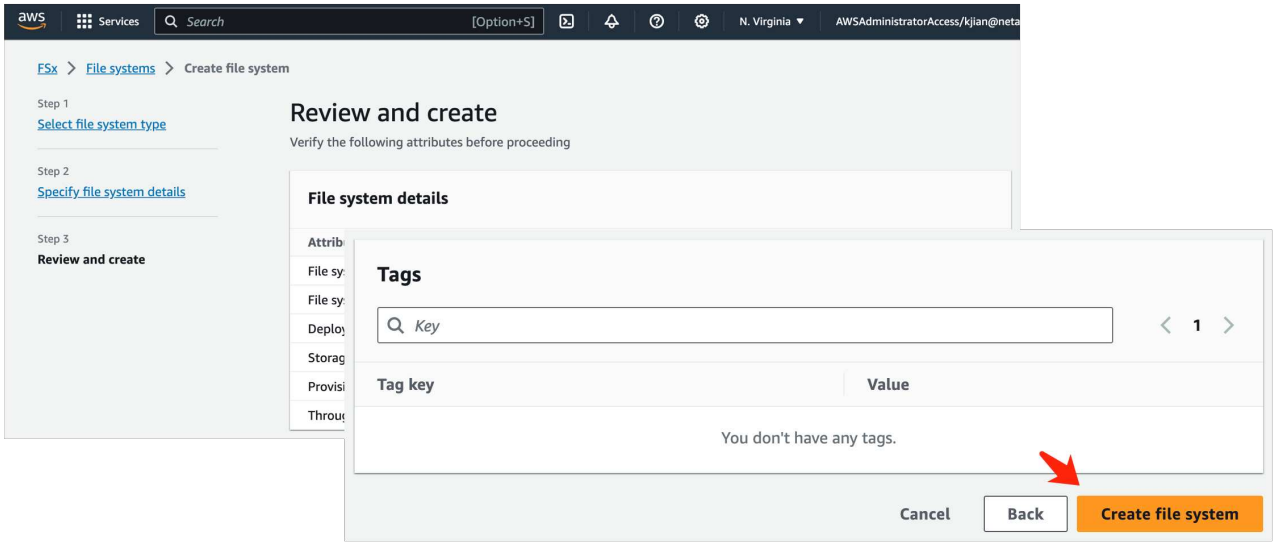
d. Inserisci il nome della **macchina virtuale di archiviazione** e **specifica una password** per la tua SVM (macchina virtuale di archiviazione).



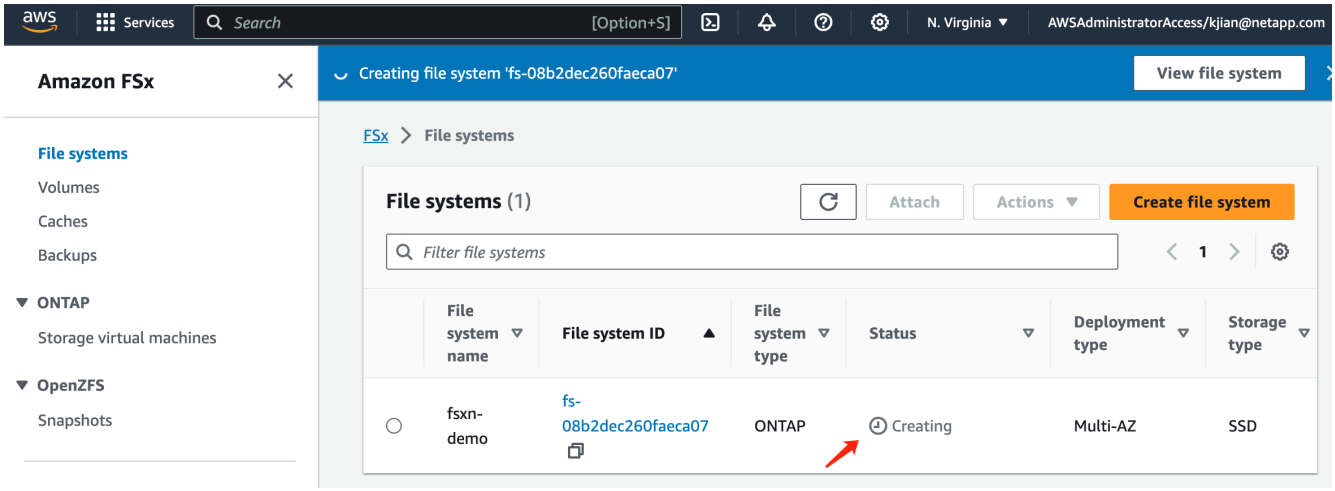
e. Lascia le altre voci predefinite e clicca sul pulsante arancione **Avanti** in basso a destra.



f. Fare clic sul pulsante arancione **Crea file system** in basso a destra nella pagina di revisione.



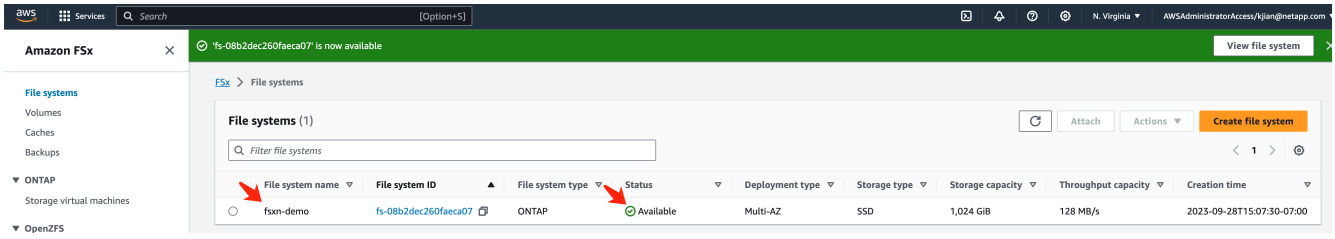
5. Potrebbero essere necessari circa **20-40 minuti** per avviare il file system FSx.



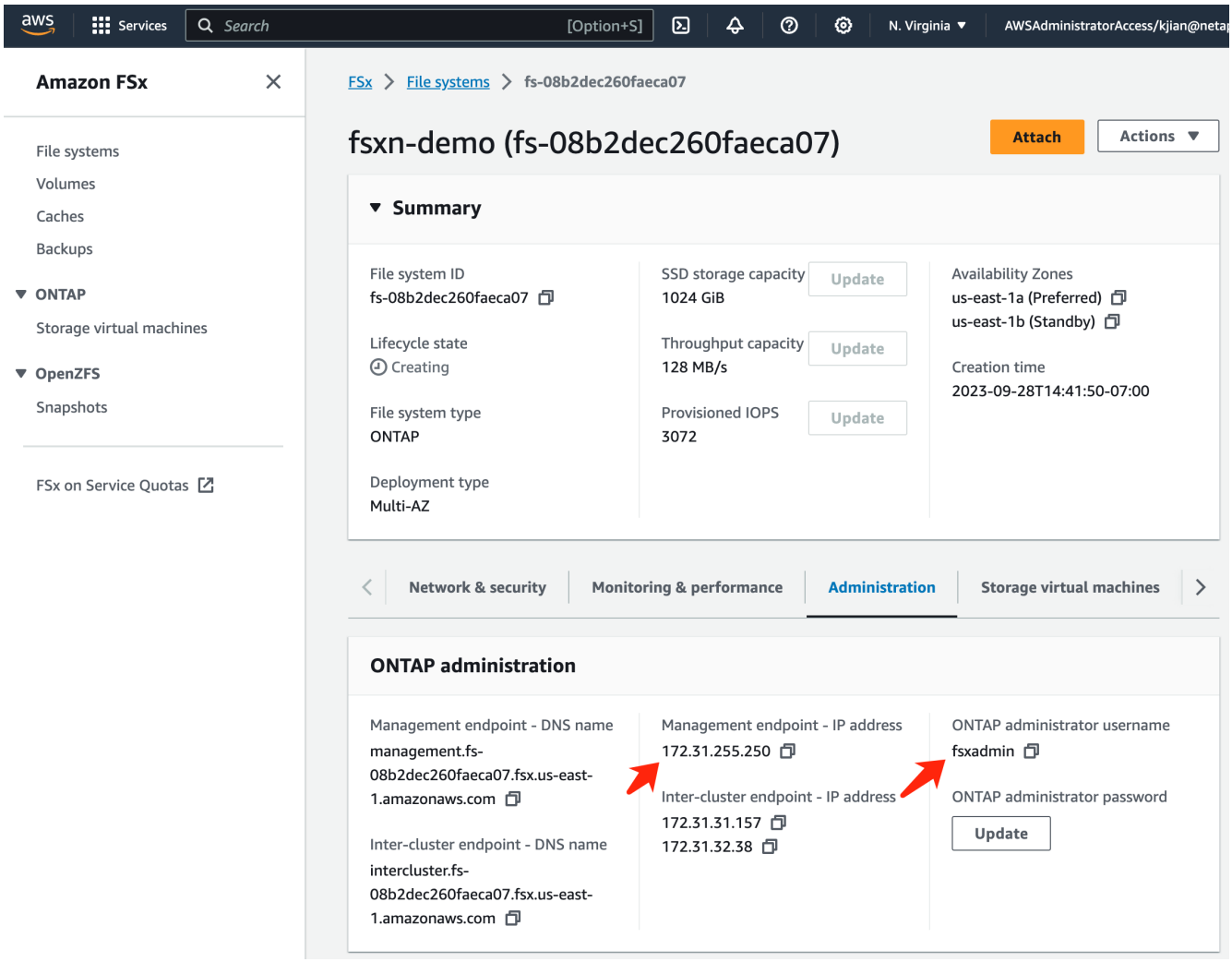
Configurazione del server

Configurazione ONTAP

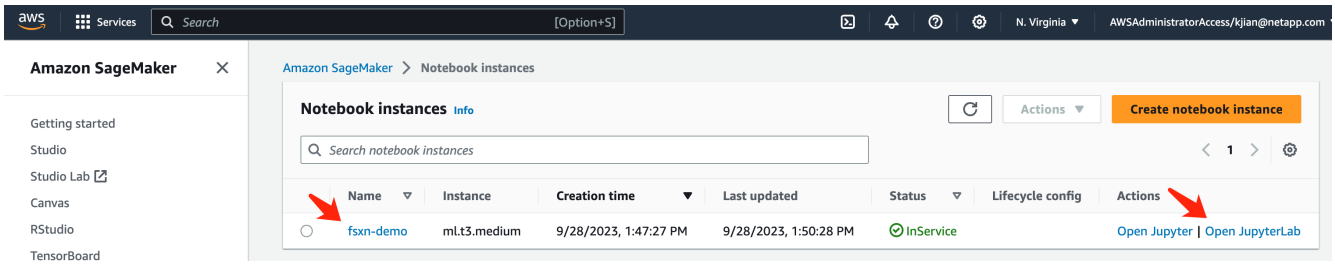
1. Aprire il file system FSx creato. Assicurati che lo stato sia **Disponibile**.



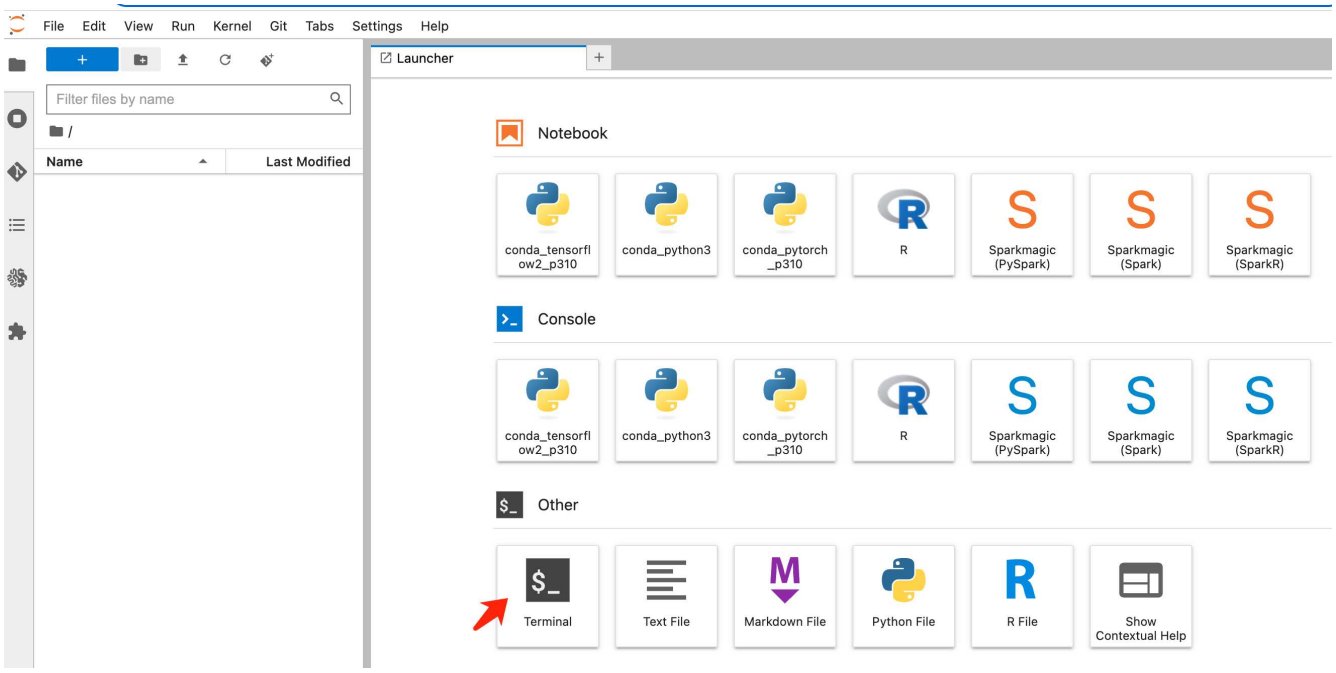
2. Selezionare la scheda **Amministrazione** e mantenere **Endpoint di gestione - Indirizzo IP** e *Nome utente amministratore ONTAP *.



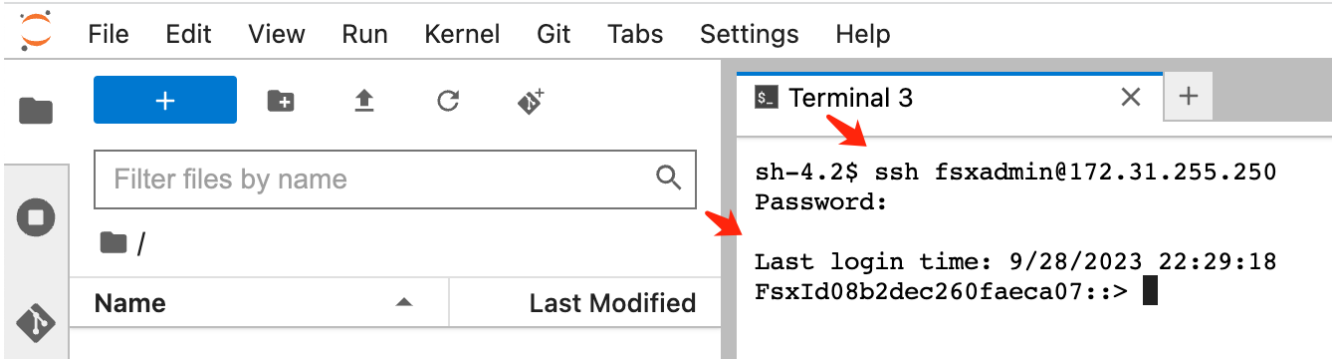
3. Aprire l'istanza **SageMaker Notebook** creata e fare clic su **Apri JupyterLab**.



4. Nella pagina Jupyter Lab, apri un nuovo **Terminale**.



5. Immettere il comando ssh ssh <nome utente amministratore>@< IP server ONTAP > per accedere al file system FSx ONTAP . (Il nome utente e l'indirizzo IP vengono recuperati dal passaggio 2) Utilizzare la password utilizzata durante la creazione della **macchina virtuale di archiviazione**.



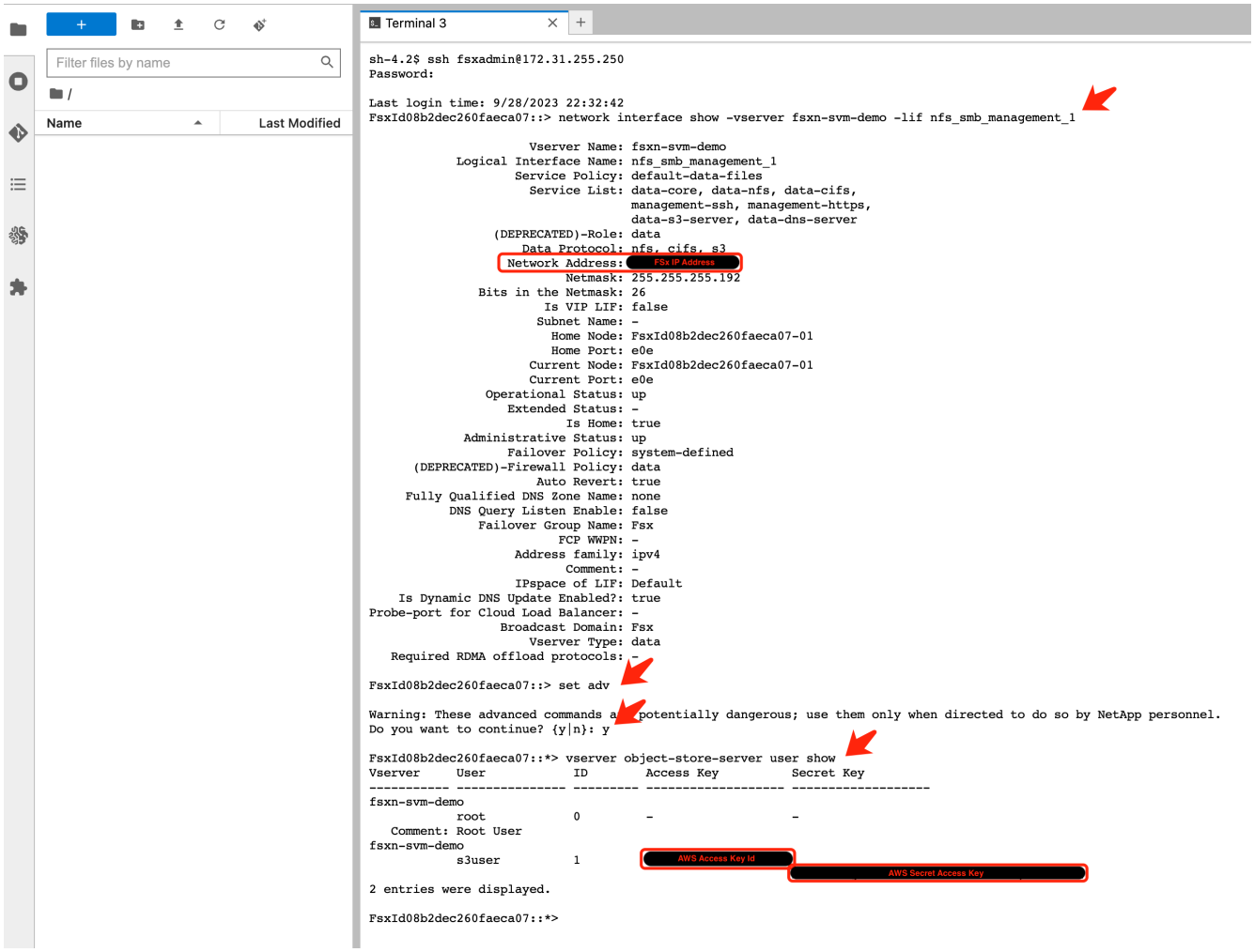
6. Eseguire i comandi nel seguente ordine. Utilizziamo **fsxn-ontap** come nome per il **nome del bucket S3 privato FSx ONTAP *. Utilizzare *nome della macchina virtuale di archiviazione** per l'argomento **-vserver**.

```
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. Eseguire i comandi seguenti per recuperare l'IP dell'endpoint e le credenziali per FSx ONTAP privato S3.

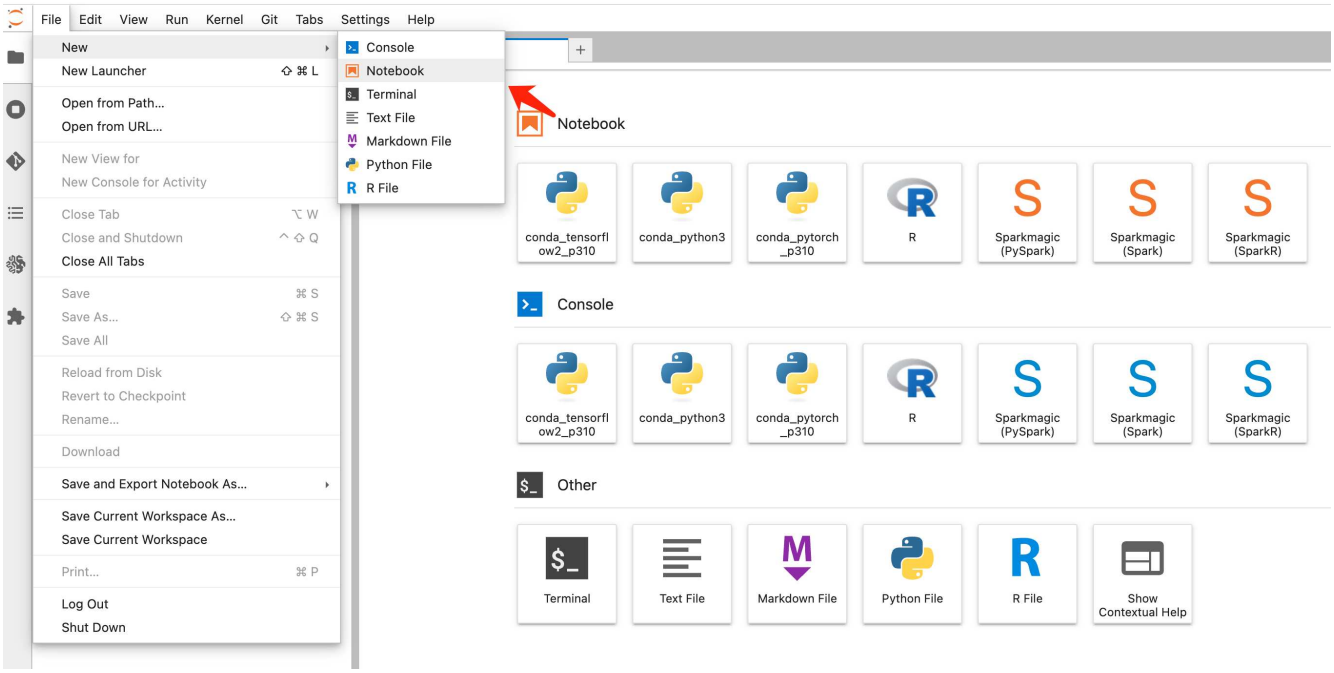
```
network interface show -vserver fsxn-svm-demo -lif nfs_smb_management_1

set adv

vserver object-store-server user show
```

8. Conservare l'IP e le credenziali dell'endpoint per un utilizzo futuro.



Configurazione del client

1. Nell'istanza di SageMaker Notebook, crea un nuovo notebook Jupyter.



2. Utilizzare il codice seguente come soluzione alternativa per caricare i file nel bucket S3 privato di FSx ONTAP . Per un esempio di codice completo fare riferimento a questo notebook."fsxn_demo.ipynb"

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

Questo conclude l'integrazione tra FSx ONTAP e l'istanza SageMaker.

Lista di controllo utile per il debug

- Assicurarsi che l'istanza di SageMaker Notebook e il file system FSx ONTAP si trovino nella stessa VPC.
- Ricordarsi di eseguire il comando **set dev** su ONTAP per impostare il livello di privilegio su **dev**.

FAQ (aggiornato al 27 settembre 2023)

D: Perché ricevo l'errore **"Si è verificato un errore (NotImplemented) durante la chiamata dell'operazione CreateMultipartUpload: il comando s3 richiesto non è implementato"** quando carico file su FSx ONTAP?

R: In quanto bucket S3 privato, FSx ONTAP supporta il caricamento di file fino a 100 MB. Quando si utilizza il protocollo S3, i file più grandi di 100 MB vengono divisi in blocchi da 100 MB e viene chiamata la funzione 'CreateMultipartUpload'. Tuttavia, l'attuale implementazione di FSx ONTAP private S3 non supporta questa funzione.

D: Perché ricevo l'errore **"Si è verificato un errore (Accesso negato) durante la chiamata delle operazioni PutObject: Accesso negato"** quando carico file su FSx ONTAP?

R: Per accedere al bucket S3 privato FSx ONTAP da un'istanza di SageMaker Notebook, sostituire le credenziali AWS con le credenziali FSx ONTAP . Tuttavia, per concedere l'autorizzazione di scrittura all'istanza è necessaria una soluzione alternativa che prevede il montaggio del bucket e l'esecuzione del comando shell 'chmod' per modificare le autorizzazioni.

D: Come posso integrare il bucket S3 privato FSx ONTAP con altri servizi SageMaker ML?

R: Purtroppo, l'SDK dei servizi SageMaker non fornisce un modo per specificare l'endpoint per il bucket S3 privato. Di conseguenza, FSx ONTAP S3 non è compatibile con i servizi SageMaker quali Sagemaker Data Wrangler, Sagemaker Clarify, Sagemaker Glue, Sagemaker Athena, Sagemaker AutoML e altri.

Parte 2 - Utilizzo di AWS Amazon FSx for NetApp ONTAP (FSx ONTAP) come origine dati per l'addestramento del modello in SageMaker

Questo articolo è un tutorial sull'utilizzo di Amazon FSx for NetApp ONTAP (FSx ONTAP) per l'addestramento di modelli PyTorch in SageMaker, in particolare per un progetto di classificazione della qualità degli pneumatici.

Introduzione

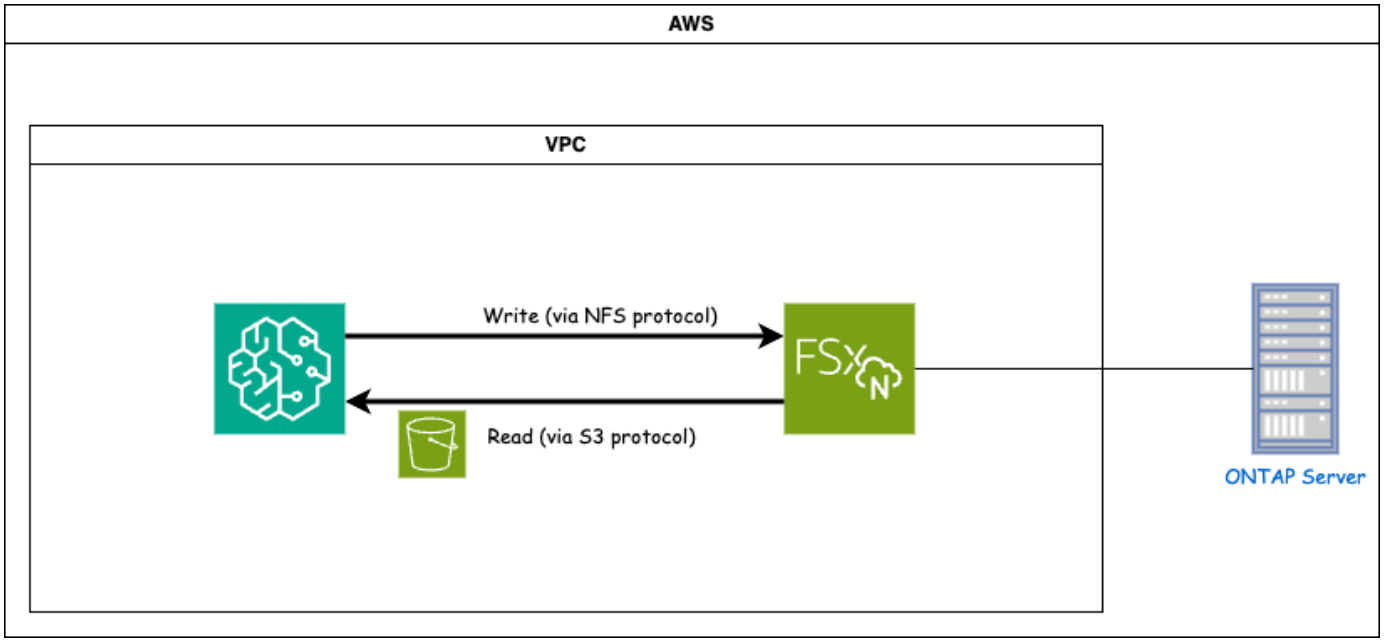
Questo tutorial offre un esempio pratico di un progetto di classificazione della visione artificiale, fornendo un'esperienza pratica nella creazione di modelli di apprendimento automatico che utilizzano FSx ONTAP come origine dati all'interno dell'ambiente SageMaker. Il progetto si concentra sull'utilizzo di PyTorch, un framework di deep learning, per classificare la qualità degli pneumatici in base alle immagini degli stessi. Si concentra sullo sviluppo di modelli di apprendimento automatico utilizzando FSx ONTAP come origine dati in Amazon SageMaker.

Che cos'è FSx ONTAP

Amazon FSx ONTAP è effettivamente una soluzione di storage completamente gestita offerta da AWS. Sfrutta il file system ONTAP di NetApp per fornire uno storage affidabile e ad alte prestazioni. Grazie al supporto di protocolli come NFS, SMB e iSCSI, consente un accesso senza interruzioni da diverse istanze di elaborazione e container. Il servizio è progettato per offrire prestazioni eccezionali, garantendo operazioni sui dati rapide ed efficienti. Offre inoltre elevata disponibilità e durabilità, garantendo che i tuoi dati rimangano accessibili e protetti. Inoltre, la capacità di archiviazione di Amazon FSx ONTAP è scalabile, consentendoti di adattarla facilmente in base alle tue esigenze.

Prerequisito

Ambiente di rete



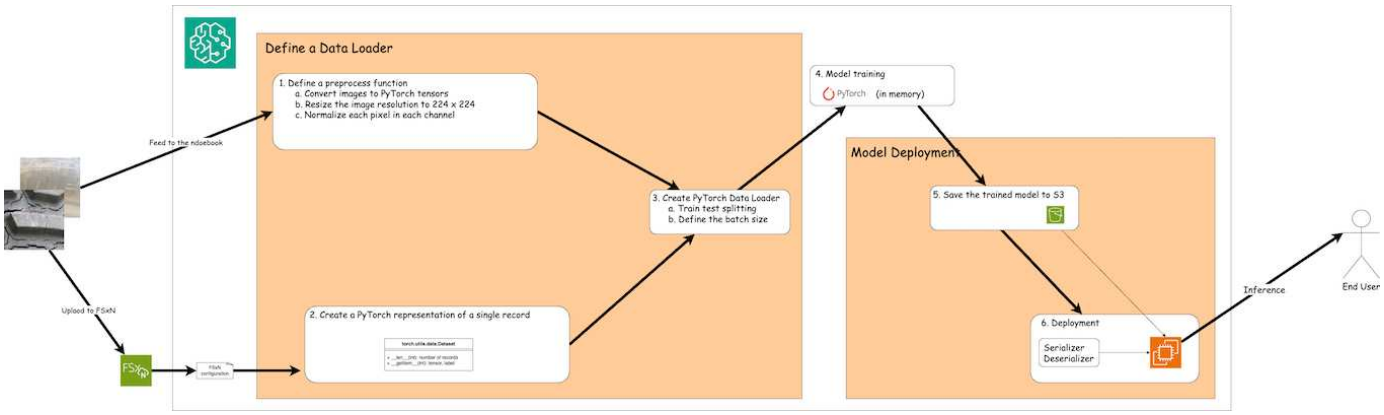
FSx ONTAP (Amazon FSx ONTAP) è un servizio di archiviazione AWS. Include un file system in esecuzione sul sistema NetApp ONTAP e una macchina virtuale (SVM) gestita da AWS che si connette ad esso. Nel diagramma fornito, il server NetApp ONTAP gestito da AWS si trova all'esterno della VPC. L'SVM funge da intermediario tra SageMaker e il sistema NetApp ONTAP , ricevendo le richieste operative da SageMaker e inoltrandole allo storage sottostante. Per accedere a FSx ONTAP, SageMaker deve essere posizionato nella stessa VPC della distribuzione FSx ONTAP . Questa configurazione garantisce la comunicazione e l'accesso ai dati tra SageMaker e FSx ONTAP.

Accesso ai dati

In scenari reali, gli scienziati dei dati in genere utilizzano i dati esistenti archiviati in FSx ONTAP per creare i loro modelli di apprendimento automatico. Tuttavia, a scopo dimostrativo, poiché il file system FSx ONTAP è inizialmente vuoto dopo la creazione, è necessario caricare manualmente i dati di addestramento. Ciò può essere ottenuto montando FSx ONTAP come volume su SageMaker. Una volta montato correttamente il file system, puoi caricare il tuo set di dati nella posizione montata, rendendolo accessibile per l'addestramento dei tuoi modelli nell'ambiente SageMaker. Questo approccio consente di sfruttare la capacità di archiviazione e le funzionalità di FSx ONTAP mentre si lavora con SageMaker per lo sviluppo e la formazione dei modelli.

Il processo di lettura dei dati prevede la configurazione di FSx ONTAP come bucket S3 privato. Per apprendere le istruzioni di configurazione dettagliate, fare riferimento a"Parte 1 - Integrazione di Amazon FSx for NetApp ONTAP (FSx ONTAP)) come bucket S3 privato in AWS SageMaker"

Panoramica sull'integrazione



Il flusso di lavoro per l'utilizzo dei dati di training in FSx ONTAP per creare un modello di deep learning in SageMaker può essere riassunto in tre passaggi principali: definizione del caricatore dati, training del modello e distribuzione. Ad alto livello, questi passaggi costituiscono la base di una pipeline MLOps. Tuttavia, ogni fase prevede diversi sotto-fasi dettagliati per un'implementazione completa. Questi sotto-passaggi comprendono varie attività, quali la pre-elaborazione dei dati, la suddivisione del set di dati, la configurazione del modello, l'ottimizzazione degli iperparametri, la valutazione del modello e l'implementazione del modello. Questi passaggi garantiscono un processo completo ed efficace per la creazione e l'implementazione di modelli di deep learning utilizzando i dati di training di FSx ONTAP all'interno dell'ambiente SageMaker.

Integrazione passo dopo passo

Loader dati

Per addestrare una rete di deep learning PyTorch con i dati, viene creato un caricatore di dati per facilitare l'alimentazione dei dati. Il caricatore dati non solo definisce la dimensione del batch, ma determina anche la procedura per la lettura e la preelaborazione di ciascun record all'interno del batch. Configurando il caricatore di dati, possiamo gestire l'elaborazione dei dati in batch, consentendo l'addestramento della rete di deep learning.

Il caricatore dati è composto da 3 parti.

Funzione di pre-elaborazione

```
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]
    )
])
```

Il frammento di codice sopra riportato illustra la definizione delle trasformazioni di pre-elaborazione delle immagini utilizzando il modulo **torchvision.transforms**. In questo tutorial, l'oggetto preprocess viene creato per applicare una serie di trasformazioni. Innanzitutto, la trasformazione **ToTensor()** converte l'immagine in una rappresentazione tensoriale. Successivamente, la trasformazione **Resize 224,224** ridimensiona

l'immagine a una dimensione fissa di 224x224 pixel. Infine, la trasformazione **Normalize()** normalizza i valori del tensore sottraendo la media e dividendo per la deviazione standard lungo ciascun canale. I valori di media e deviazione standard utilizzati per la normalizzazione sono comunemente impiegati nei modelli di reti neurali pre-addestrati. Nel complesso, questo codice prepara i dati dell'immagine per un'ulteriore elaborazione o per l'inserimento in un modello pre-addestrato, convertendoli in un tensore, ridimensionandoli e normalizzando i valori dei pixel.

La classe del set di dati 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
```

Questa classe fornisce funzionalità per ottenere il numero totale di record nel set di dati e definisce il metodo per leggere i dati per ciascun record. All'interno della funzione ***getitem***, il codice utilizza l'oggetto bucket S3 boto3 per recuperare i dati binari da FSx ONTAP. Lo stile del codice per l'accesso ai dati da FSx ONTAP è simile a quello per la lettura dei dati da Amazon S3. La spiegazione successiva approfondisce il processo di creazione dell'oggetto S3 privato **bucket**.

FSx ONTAP come repository S3 privato

```
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 ---
```

Per leggere i dati da FSx ONTAP in SageMaker, viene creato un gestore che punta allo storage FSx ONTAP utilizzando il protocollo S3. Ciò consente di trattare FSx ONTAP come un bucket S3 privato. La configurazione del gestore include la specifica dell'indirizzo IP dell'SVM FSx ONTAP , del nome del bucket e delle credenziali necessarie. Per una spiegazione completa su come ottenere questi elementi di configurazione, fare riferimento al documento all'indirizzo"Parte 1 - Integrazione di Amazon FSx for NetApp ONTAP (FSx ONTAP) come bucket S3 privato in AWS SageMaker" .

Nell'esempio menzionato sopra, l'oggetto bucket viene utilizzato per istanziare l'oggetto dataset PyTorch. L'oggetto dataset verrà spiegato più dettagliatamente nella sezione successiva.

Il Loader dati 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)
```

Nell'esempio fornito, viene specificata una dimensione batch pari a 64, a indicare che ogni batch conterrà 64 record. Combinando la classe PyTorch **Dataset**, la funzione di pre-elaborazione e la dimensione del batch di addestramento, otteniamo il caricatore di dati per l'addestramento. Questo caricatore di dati semplifica il processo di iterazione del set di dati in batch durante la fase di addestramento.

Formazione del modello

```
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')
```

Questo codice implementa un processo di addestramento PyTorch standard. Definisce un modello di rete neurale denominato **TyreQualityClassifier** che utilizza livelli convoluzionali e un livello lineare per classificare la qualità degli pneumatici. Il ciclo di addestramento esegue iterazioni su batch di dati, calcola la perdita e aggiorna i parametri del modello utilizzando la backpropagation e l'ottimizzazione. Inoltre, stampa l'ora corrente, l'epoca, il lotto e la perdita a scopo di monitoraggio.

Distribuzione del modello

Distribuzione

```
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())
```

Il codice salva il modello PyTorch su **Amazon S3** perché SageMaker richiede che il modello venga archiviato in S3 per la distribuzione. Caricando il modello su **Amazon S3**, questo diventa accessibile a SageMaker, consentendo la distribuzione e l'inferenza sul modello distribuito.

```
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
)
```

Questo codice facilita l'implementazione di un modello PyTorch su SageMaker. Definisce un serializzatore personalizzato, **TyreQualitySerializer**, che preelabora e serializza i dati di input come un tensore PyTorch. La classe **TyreQualityPredictor** è un predittore personalizzato che utilizza il serializzatore definito e un **JSONDeserializer**. Il codice crea anche un oggetto **PyTorchModel** per specificare la posizione S3 del modello, il ruolo IAM, la versione del framework e il punto di ingresso per l'inferenza. Il codice genera un

timestamp e costruisce un nome di endpoint basato sul modello e sul timestamp. Infine, il modello viene distribuito utilizzando il metodo deploy, specificando il conteggio delle istanze, il tipo di istanza e il nome dell'endpoint generato. Ciò consente di distribuire il modello PyTorch e di renderlo accessibile per l'inferenza su SageMaker.

Inferenza

```
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)
```

Questo è un esempio di utilizzo dell'endpoint distribuito per eseguire l'inferenza.

Parte 3 - Creazione di una pipeline MLOps semplificata (CI/CT/CD)

Questo articolo fornisce una guida alla creazione di una pipeline MLOps con i servizi AWS, concentrandosi sulla riqualificazione automatizzata dei modelli, sulla distribuzione e sull'ottimizzazione dei costi.

Introduzione

In questo tutorial imparerai come sfruttare vari servizi AWS per costruire una semplice pipeline MLOps che comprende integrazione continua (CI), formazione continua (CT) e distribuzione continua (CD). A differenza delle pipeline DevOps tradizionali, MLOps richiede considerazioni aggiuntive per completare il ciclo operativo. Seguendo questo tutorial, imparerai come integrare CT nel ciclo MLOps, consentendo un addestramento continuo dei tuoi modelli e una distribuzione senza interruzioni per l'inferenza. Il tutorial ti guiderà attraverso il processo di utilizzo dei servizi AWS per stabilire questa pipeline MLOps end-to-end.

Manifesto

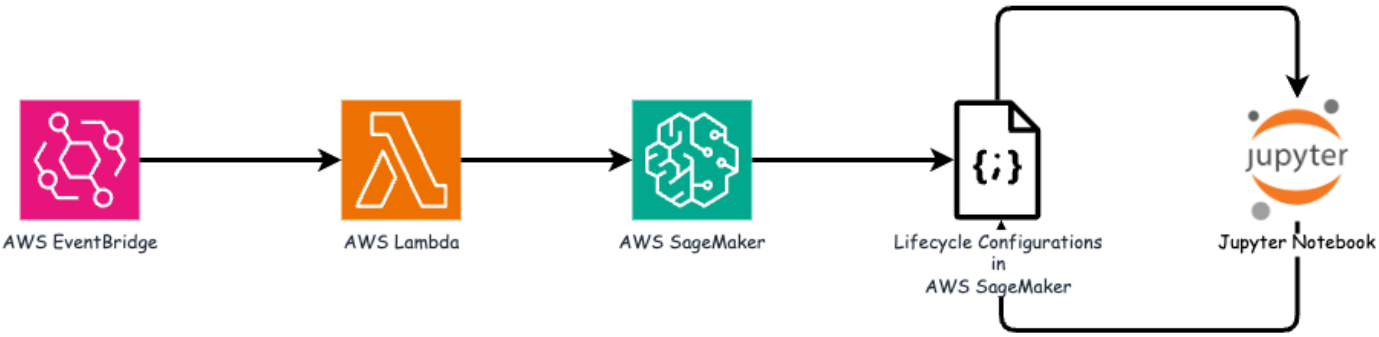
Funzionalità	Nome	Commento
Archiviazione dei dati	AWS FSx ONTAP	Fare riferimento a "Parte 1 - Integrazione di Amazon FSx for NetApp ONTAP (FSx ONTAP) come bucket S3 privato in AWS SageMaker" .
IDE di scienza dei dati	AWS SageMaker	Questo tutorial si basa sul notebook Jupyter presentato in"Parte 2 - Utilizzo di Amazon FSx for NetApp ONTAP (FSx ONTAP) come origine dati per l'addestramento del modello in SageMaker" .

Funzionalità	Nome	Commento
Funzione per attivare la pipeline MLOps	Funzione AWS Lambda	-
Trigger del lavoro cron	AWS EventBridge	-
Framework di apprendimento profondo	PyTorch	-
SDK Python di AWS	boto3	-
Linguaggio di programmazione	Pitone	v3.10

Prerequisito

- Un file system FSx ONTAP preconfigurato. Questo tutorial utilizza i dati memorizzati in FSx ONTAP per il processo di formazione.
- Un'**istanza di SageMaker Notebook** configurata per condividere la stessa VPC del file system FSx ONTAP menzionato sopra.
- Prima di attivare la **funzione AWS Lambda**, assicurarsi che l'**istanza SageMaker Notebook** sia nello stato **arrestato**.
- Il tipo di istanza **ml.g4dn.xlarge** è necessario per sfruttare l'accelerazione GPU necessaria per i calcoli delle reti neurali profonde.

Architettura



Questa pipeline MLOps è un'implementazione pratica che utilizza un cron job per attivare una funzione serverless, che a sua volta esegue un servizio AWS registrato con una funzione di callback del ciclo di vita. **AWS EventBridge** funge da cron job. Periodicamente richiama una **funzione AWS Lambda** responsabile della riqualificazione e della ridistribuzione del modello. Questo processo prevede l'avvio dell'istanza **AWS SageMaker Notebook** per eseguire le attività necessarie.

Configurazione passo passo

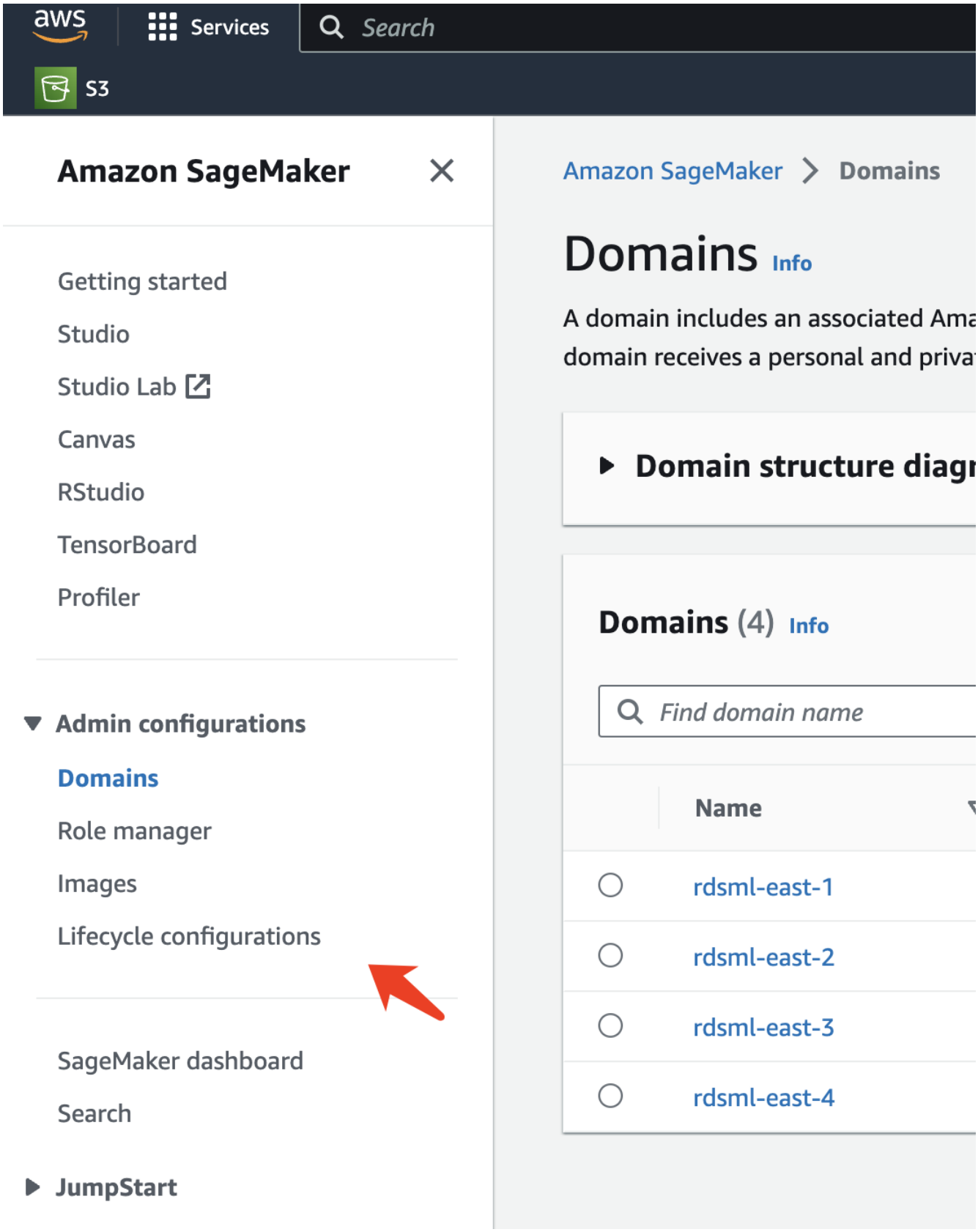
Configurazioni del ciclo di vita

Per configurare la funzione di callback del ciclo di vita per l'istanza AWS SageMaker Notebook, è necessario utilizzare **Configurazioni del ciclo di vita**. Questo servizio consente di definire le azioni necessarie da eseguire durante l'avvio dell'istanza del notebook. Nello specifico, è possibile implementare uno script shell all'interno delle **configurazioni del ciclo di vita** per arrestare automaticamente l'istanza del notebook una volta completati i processi di formazione e distribuzione. Questa è una configurazione obbligatoria poiché il

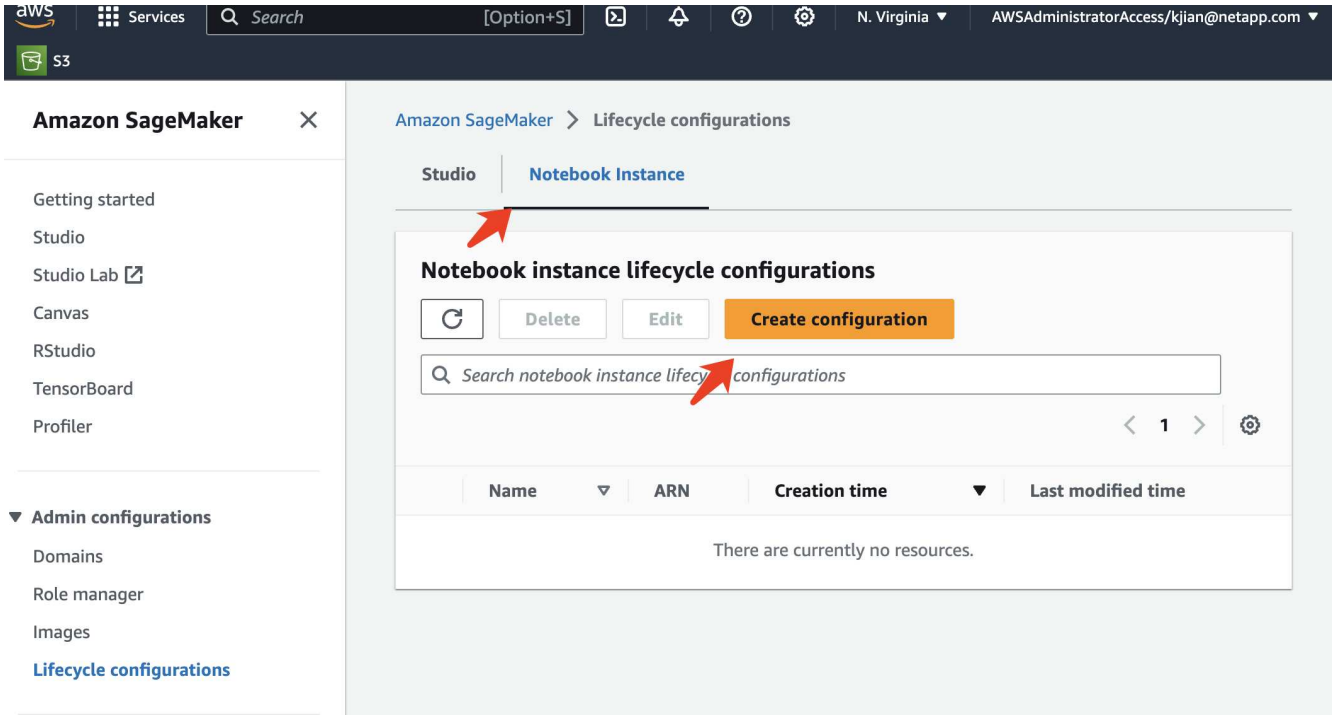
costo è uno degli aspetti più importanti da considerare in MLOps.

È importante notare che la configurazione per le **configurazioni del ciclo di vita** deve essere impostata in anticipo. Pertanto, si consiglia di dare priorità alla configurazione di questo aspetto prima di procedere con l'altra configurazione della pipeline MLOps.

1. Per impostare le configurazioni del ciclo di vita, apri il pannello **Sagemaker** e vai a **Configurazioni del ciclo di vita** nella sezione **Configurazioni amministrative**.



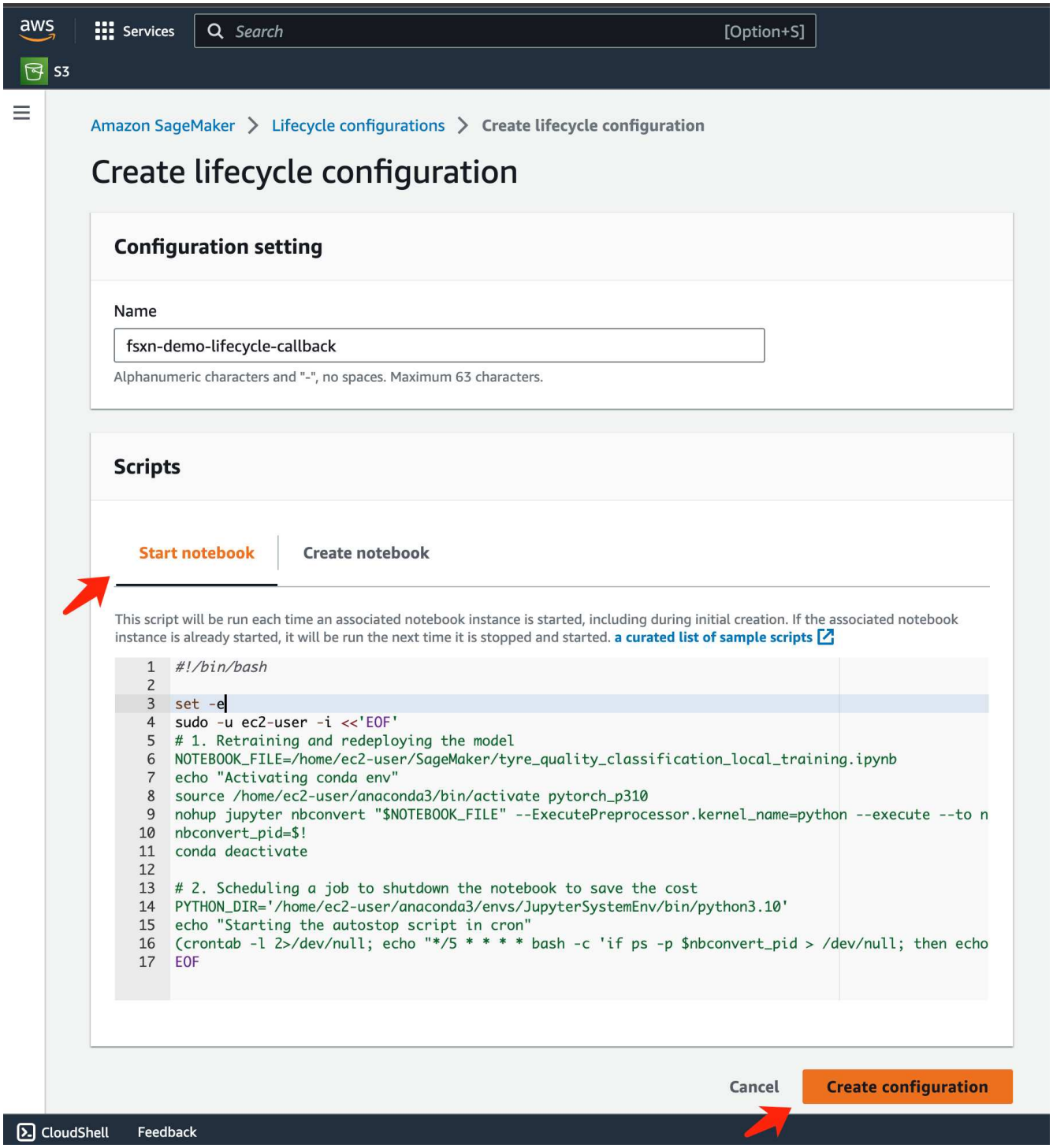
2. Selezionare la scheda **Istanza notebook** e fare clic sul pulsante **Crea configurazione**



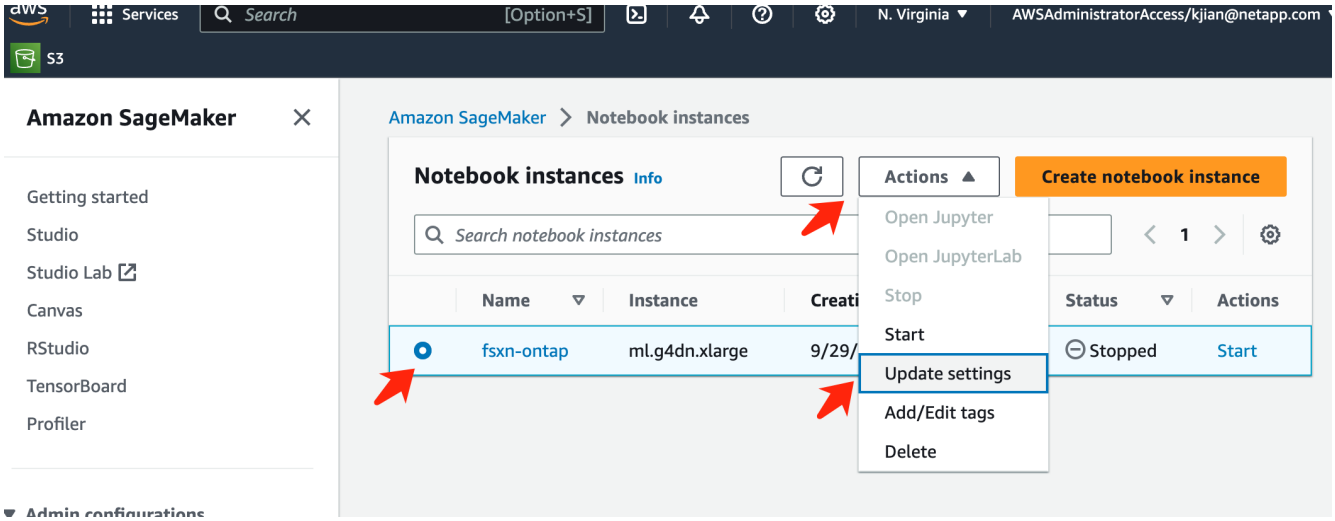
3. Incolla il codice sottostante nell'area di immissione.

```
#!/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
```

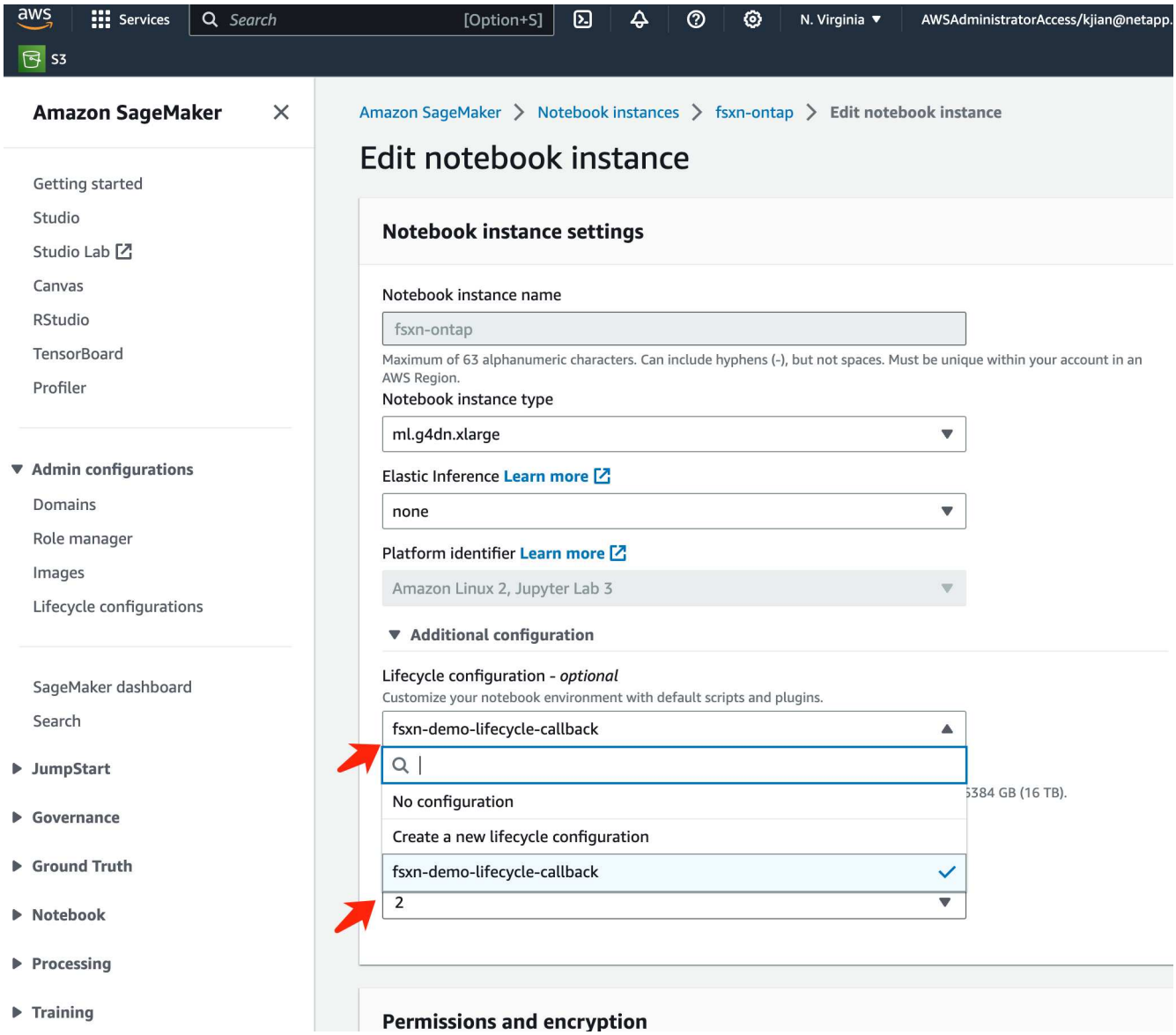
4. Questo script esegue Jupyter Notebook, che gestisce il riaddestramento e la ridistribuzione del modello per l'inferenza. Una volta completata l'esecuzione, il notebook si spegnerà automaticamente entro 5 minuti. Per saperne di più sulla definizione del problema e sull'implementazione del codice, fare riferimento a"Parte 2 - Utilizzo di Amazon FSx for NetApp ONTAP (FSx ONTAP) come origine dati per l'addestramento del modello in SageMaker" .



5. Dopo la creazione, vai alle istanze del Notebook, seleziona l'istanza di destinazione e fai clic su **Aggiorna impostazioni** nel menu a discesa Azioni.



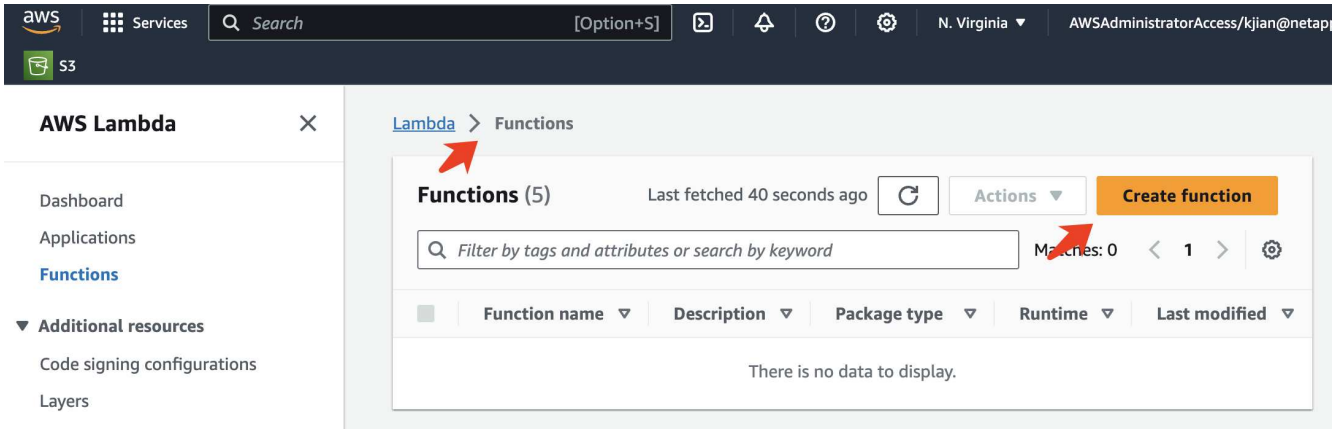
6. Selezionare la **Configurazione del ciclo di vita** creata e fare clic su **Aggiorna istanza del notebook**.



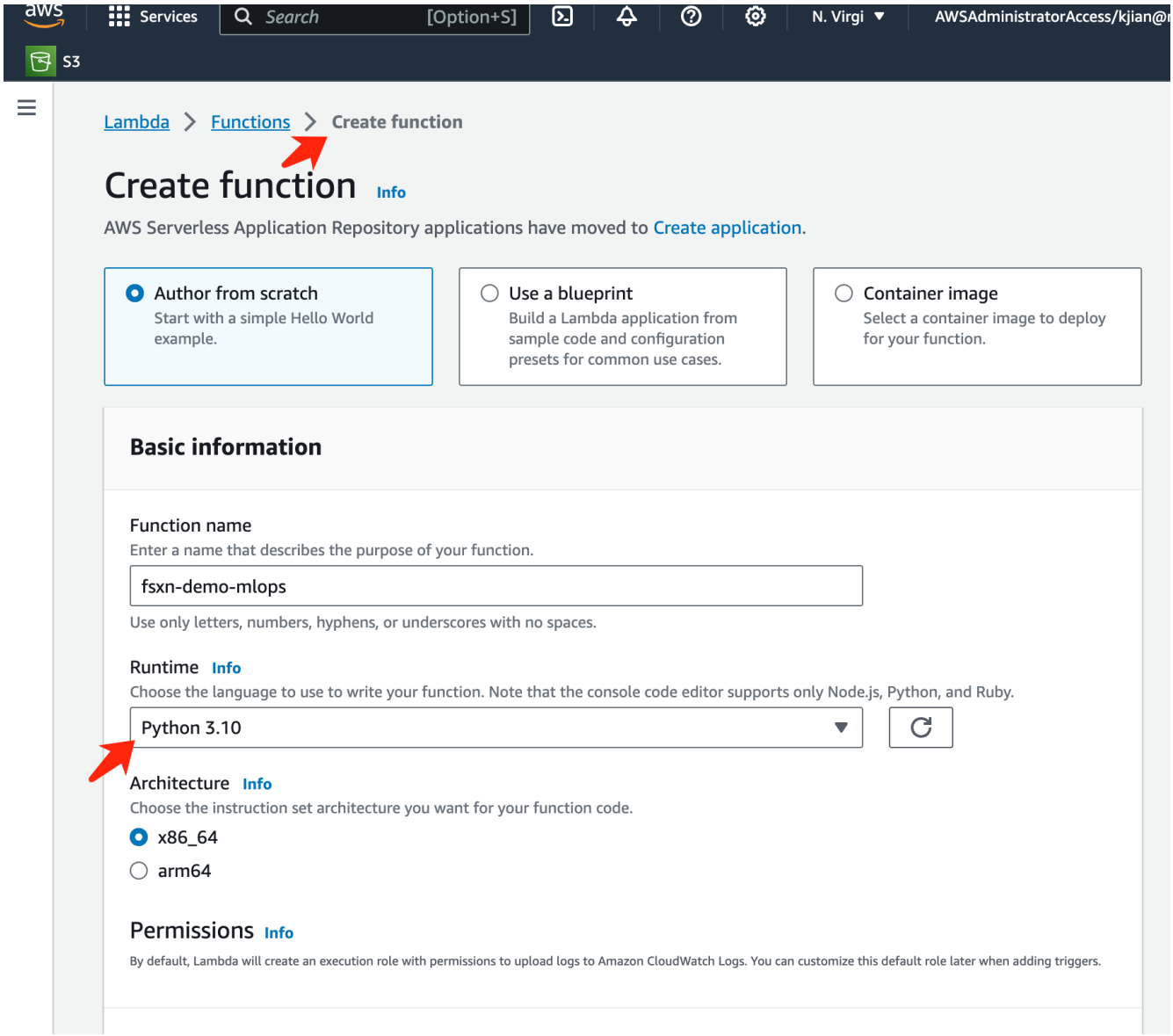
Funzione serverless AWS Lambda

Come accennato in precedenza, la **funzione AWS Lambda** è responsabile dell'avvio dell'**istanza AWS SageMaker Notebook**.

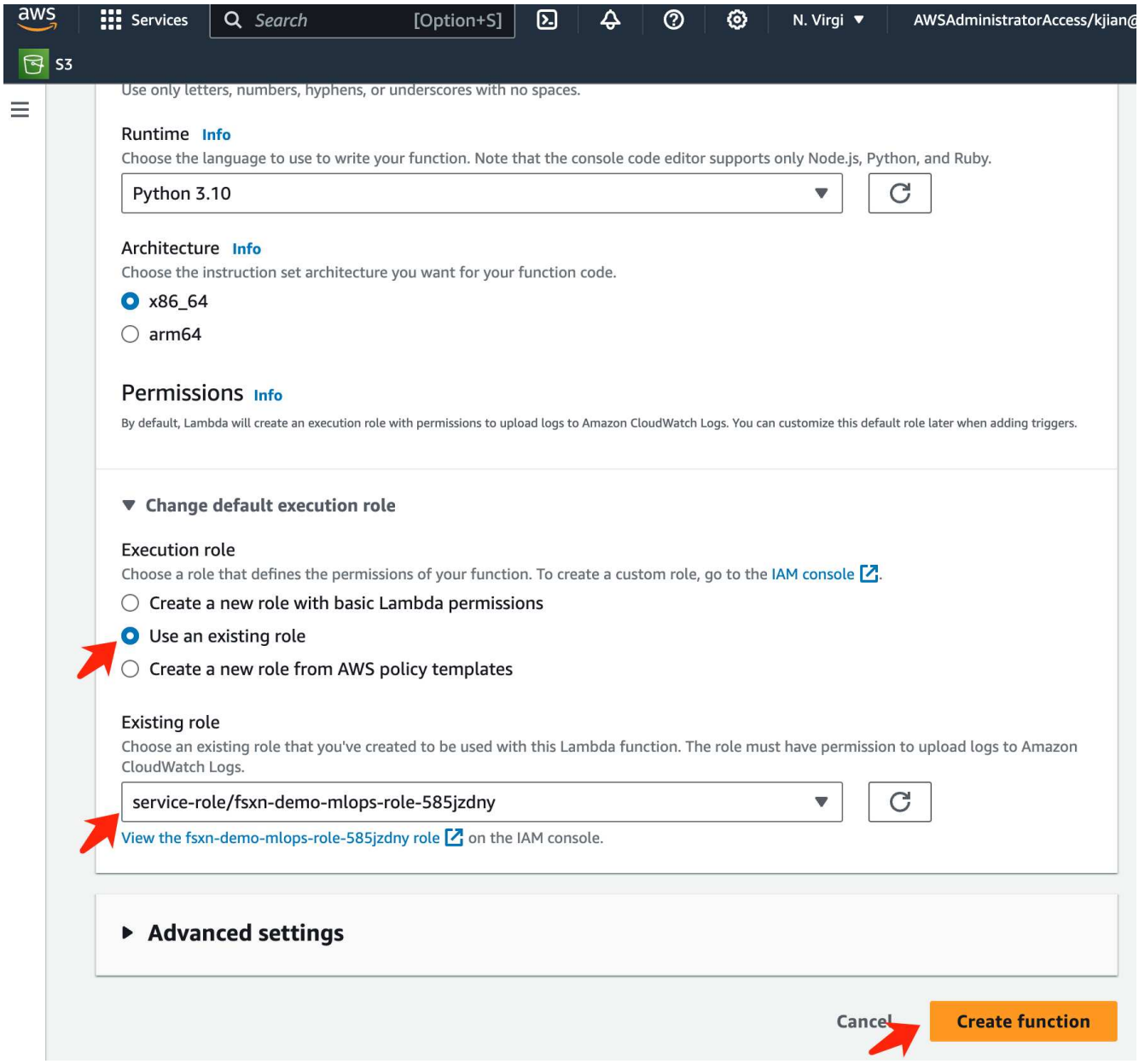
1. Per creare una **funzione AWS Lambda**, vai al pannello corrispondente, passa alla scheda **Funzioni** e fai clic su **Crea funzione**.



2. Si prega di compilare tutte le voci richieste nella pagina e di ricordarsi di impostare il Runtime su **Python 3.10**.



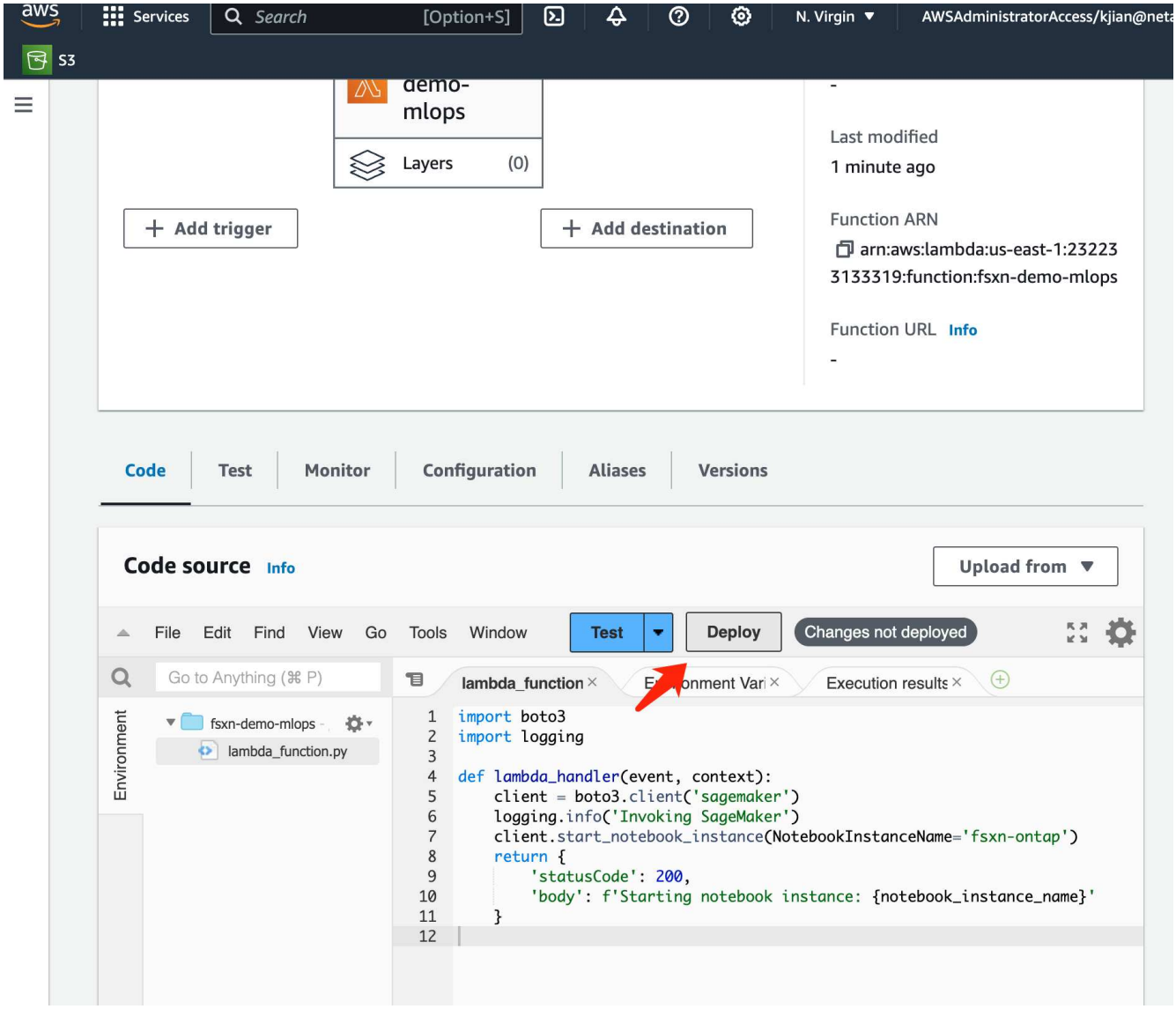
3. Verificare che il ruolo designato disponga dell'autorizzazione richiesta **AmazonSageMakerFullAccess** e fare clic sul pulsante **Crea funzione**.



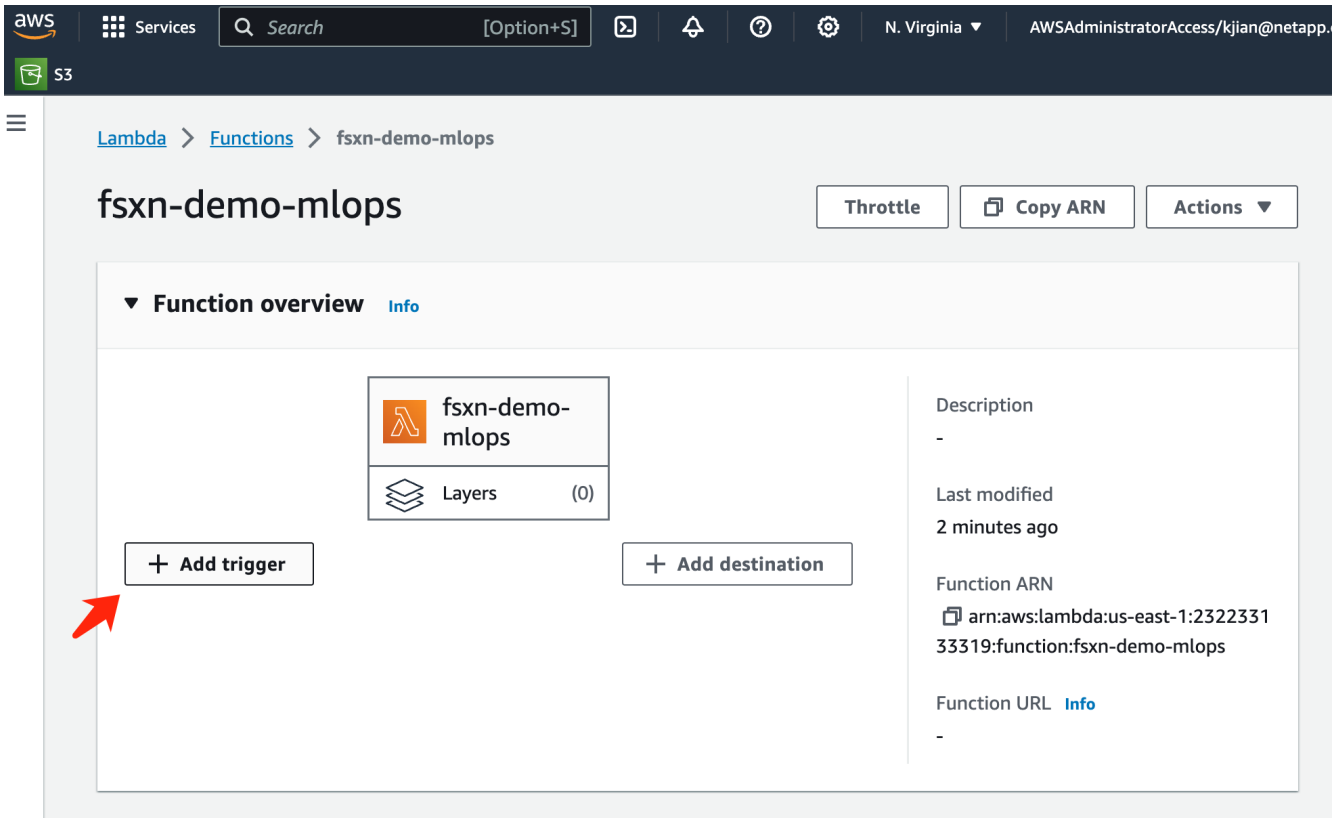
4. Selezionare la funzione Lambda creata. Nella scheda codice, copia e incolla il seguente codice nell'area di testo. Questo codice avvia l'istanza del notebook denominata **fsxn-ontap**.

```
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}'
    }
```

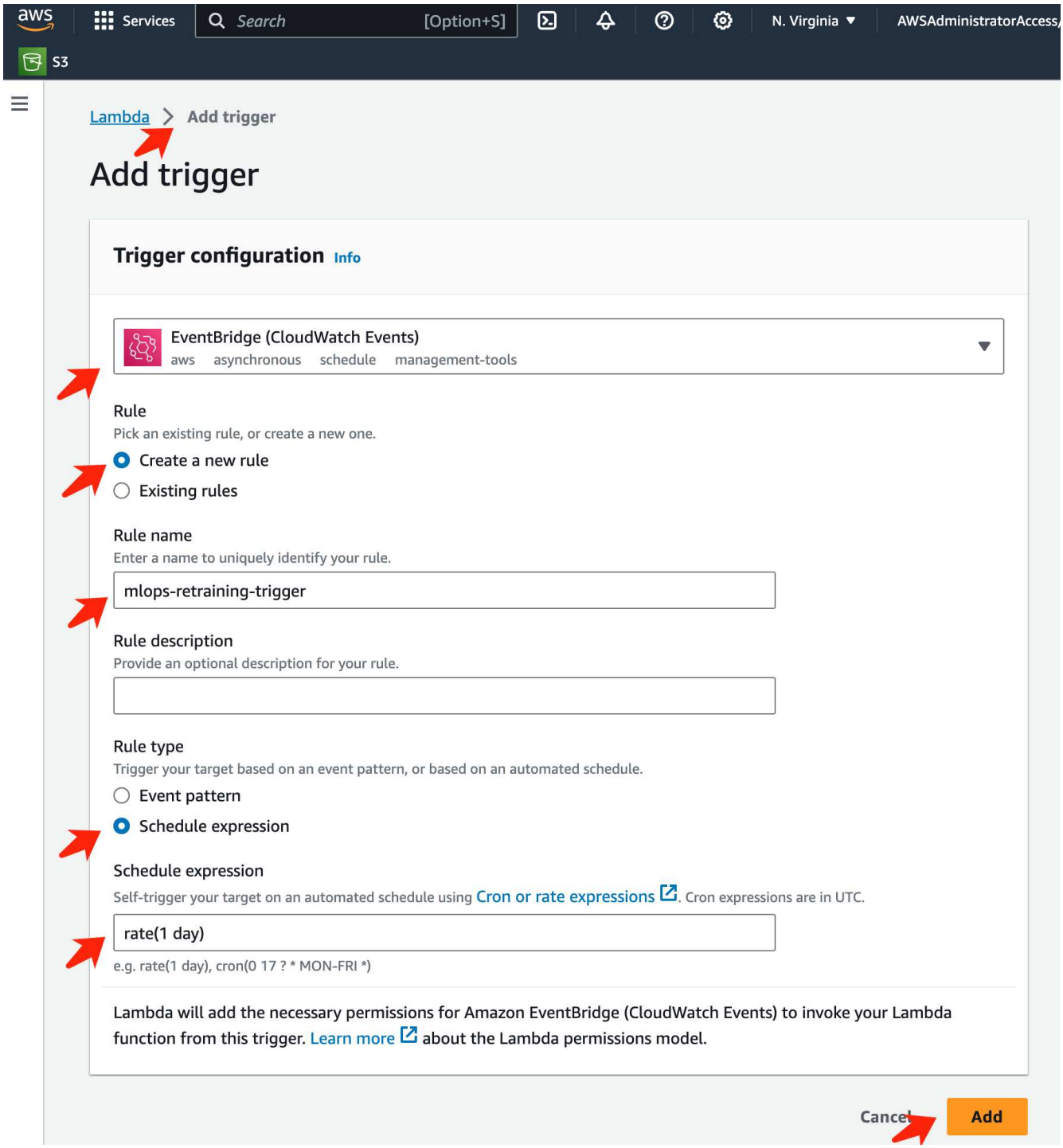
5. Fare clic sul pulsante **Distribuisci** per applicare questa modifica al codice.



6. Per specificare come attivare questa funzione AWS Lambda, fare clic sul pulsante Aggiungi trigger.



7. Selezionare EventBridge dal menu a discesa, quindi fare clic sul pulsante di opzione denominato Crea una nuova regola. Nel campo espressione pianificazione, immettere `rate(1 day)` e fai clic sul pulsante Aggiungi per creare e applicare questa nuova regola cron job alla funzione AWS Lambda.



Dopo aver completato la configurazione in due fasi, ogni giorno la **funzione AWS Lambda** avvierà **SageMaker Notebook**, eseguirà il riaddestramento del modello utilizzando i dati dal repository **FSx ONTAP**, ridistribuirà il modello aggiornato nell'ambiente di produzione e arresterà automaticamente l'**istanza di SageMaker Notebook** per ottimizzare i costi. Ciò garantisce che il modello rimanga aggiornato.

Si conclude qui il tutorial sullo sviluppo di una pipeline MLOps.

Informazioni sul copyright

Informazioni sul marchio commerciale