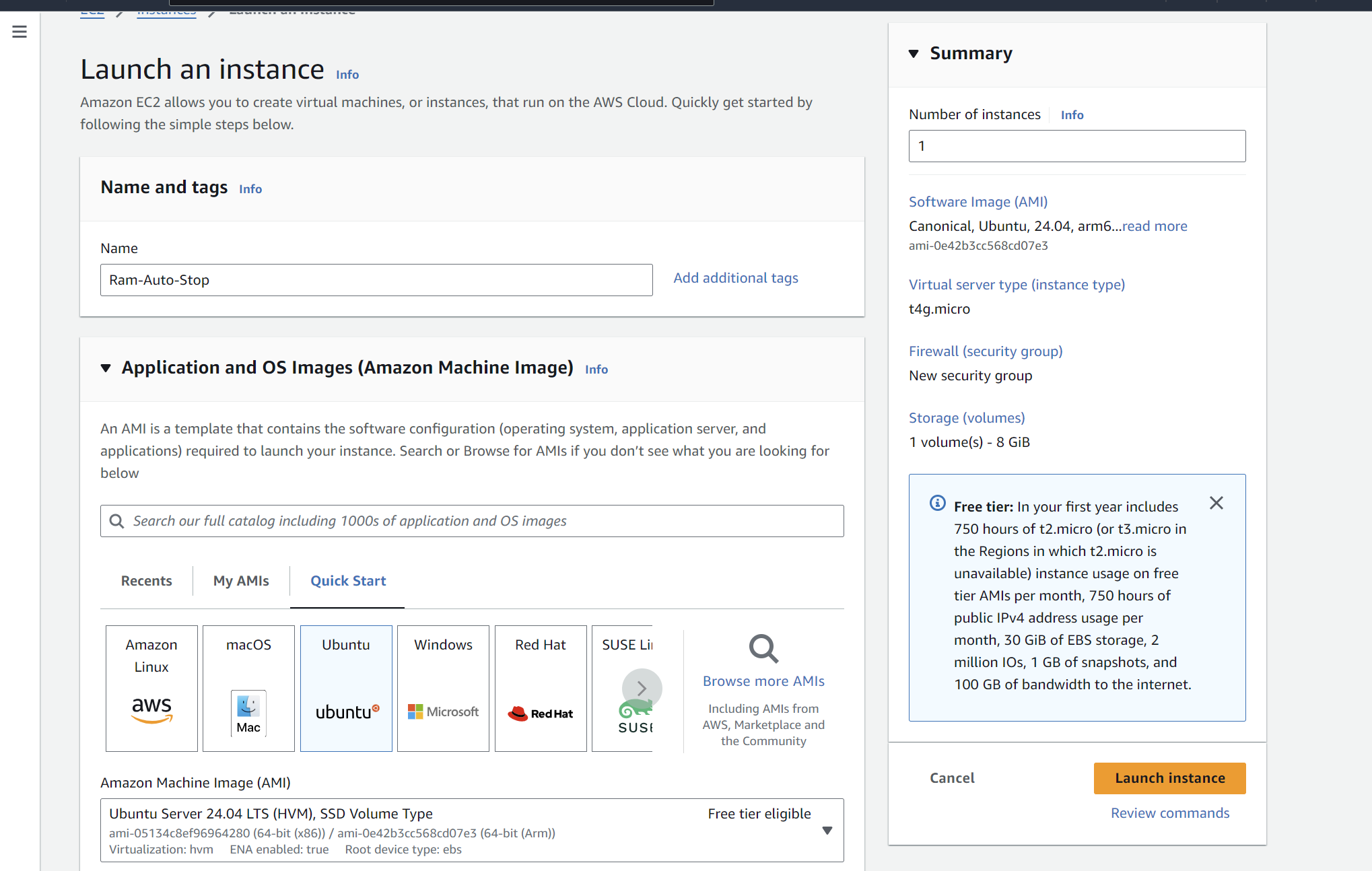
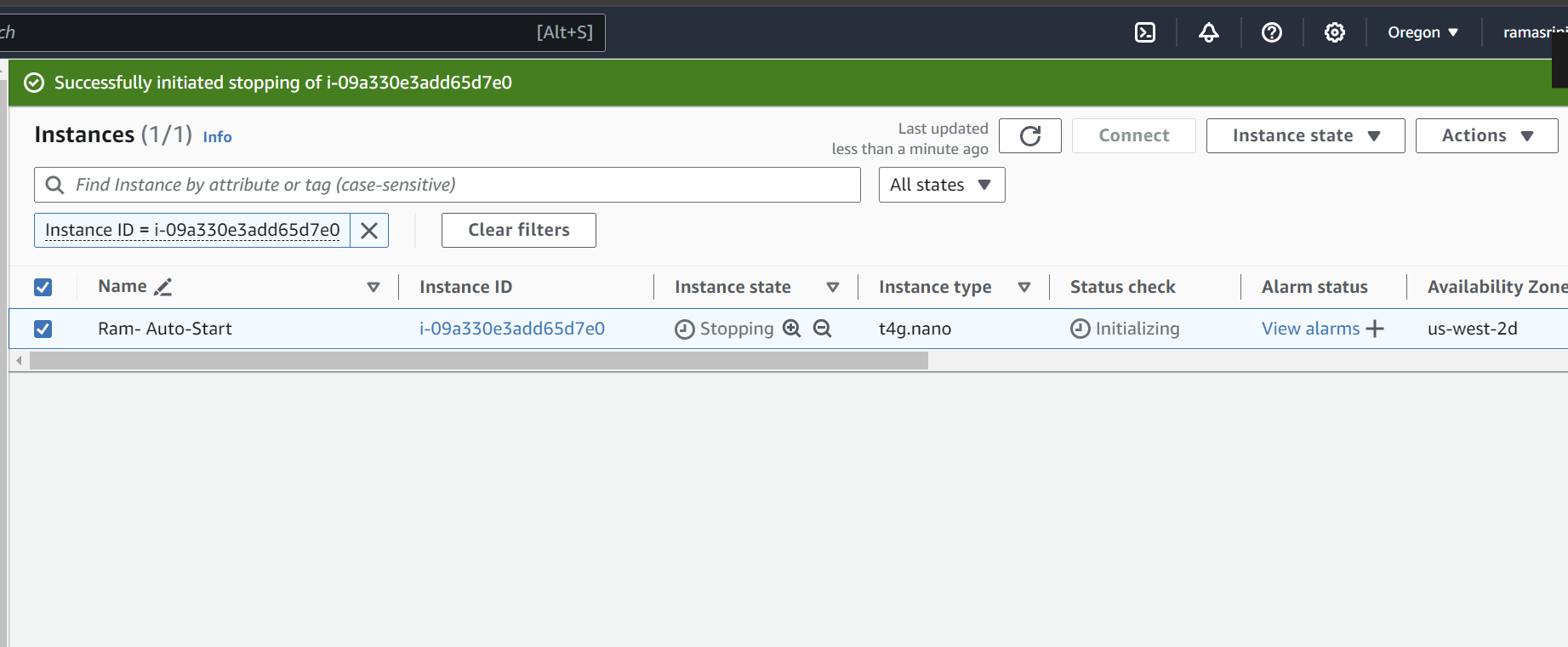
**Graded-Assignment-On-Serverless-Architecture**

**Git hub repo: https://github.com/Ram495-ctrl/Serverless-Architecture\_HV.git**

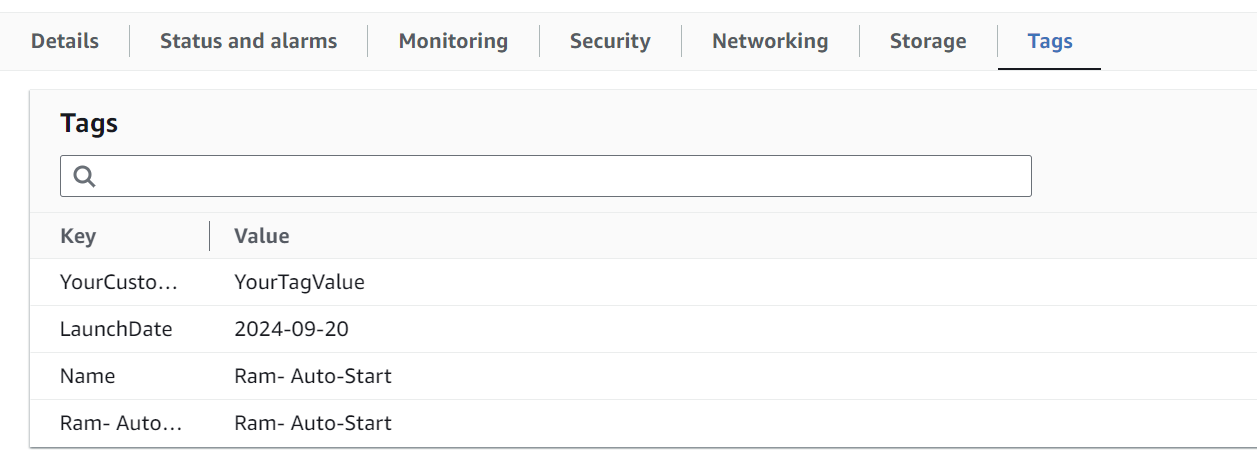
Assignment 1: Automated Instance Management Using AWS Lambda and Boto3

1. Creating EC2 Instances for AutoStart and AutoStop.

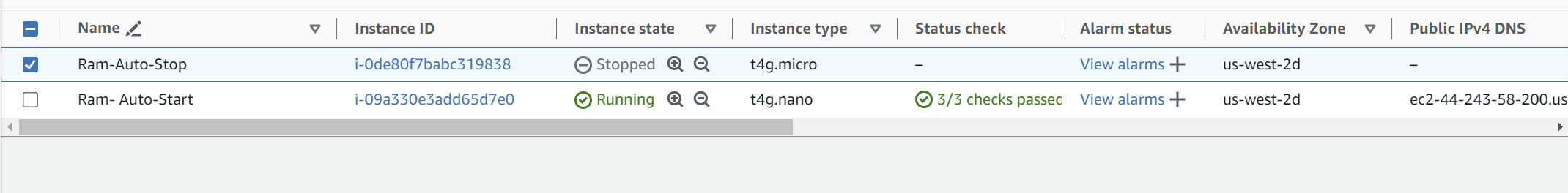


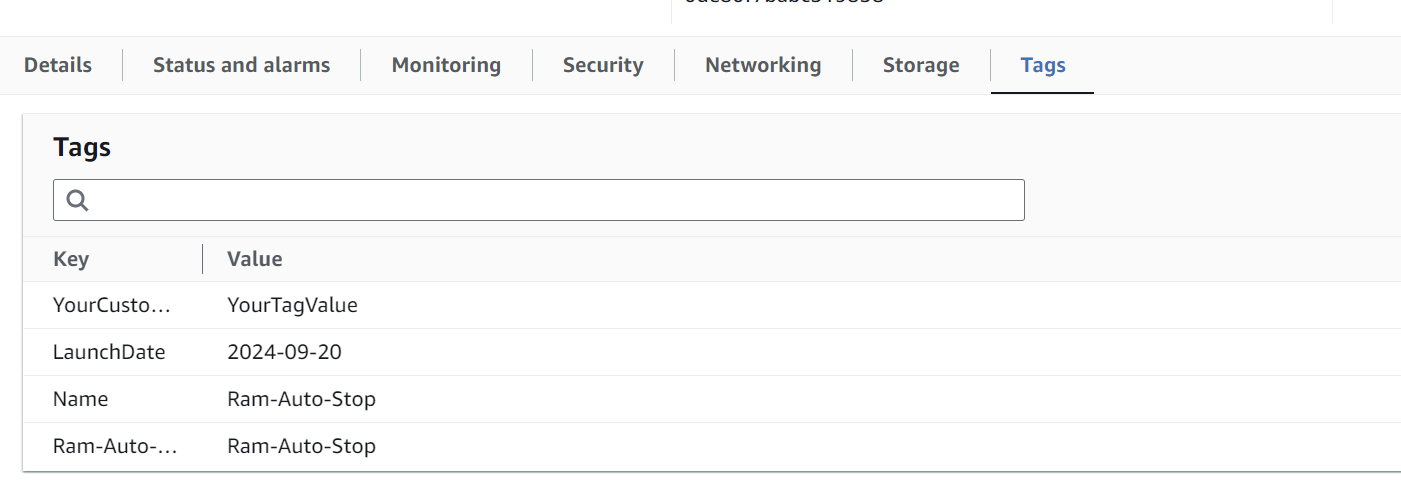


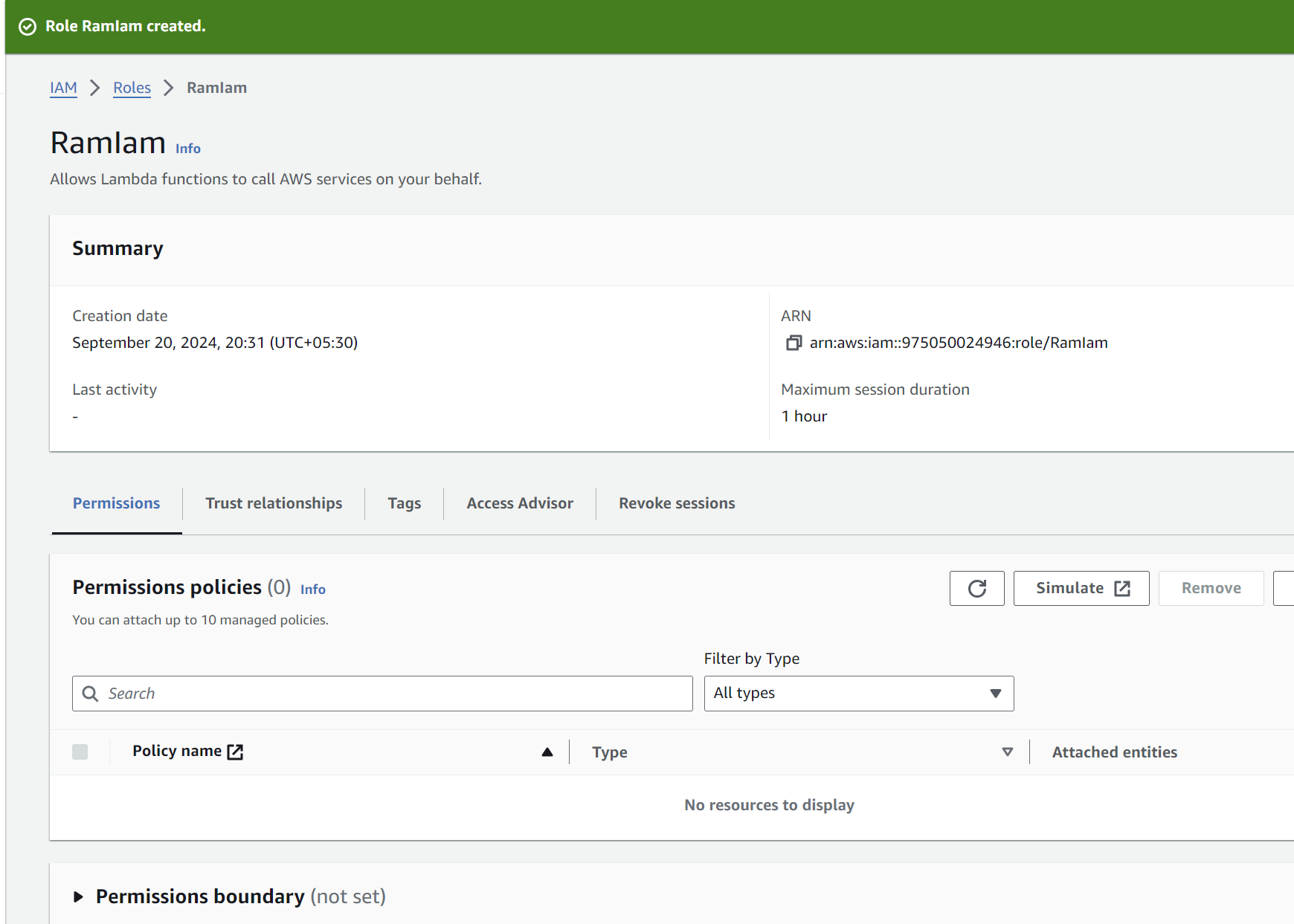
Ram Auto-start Create Tag:

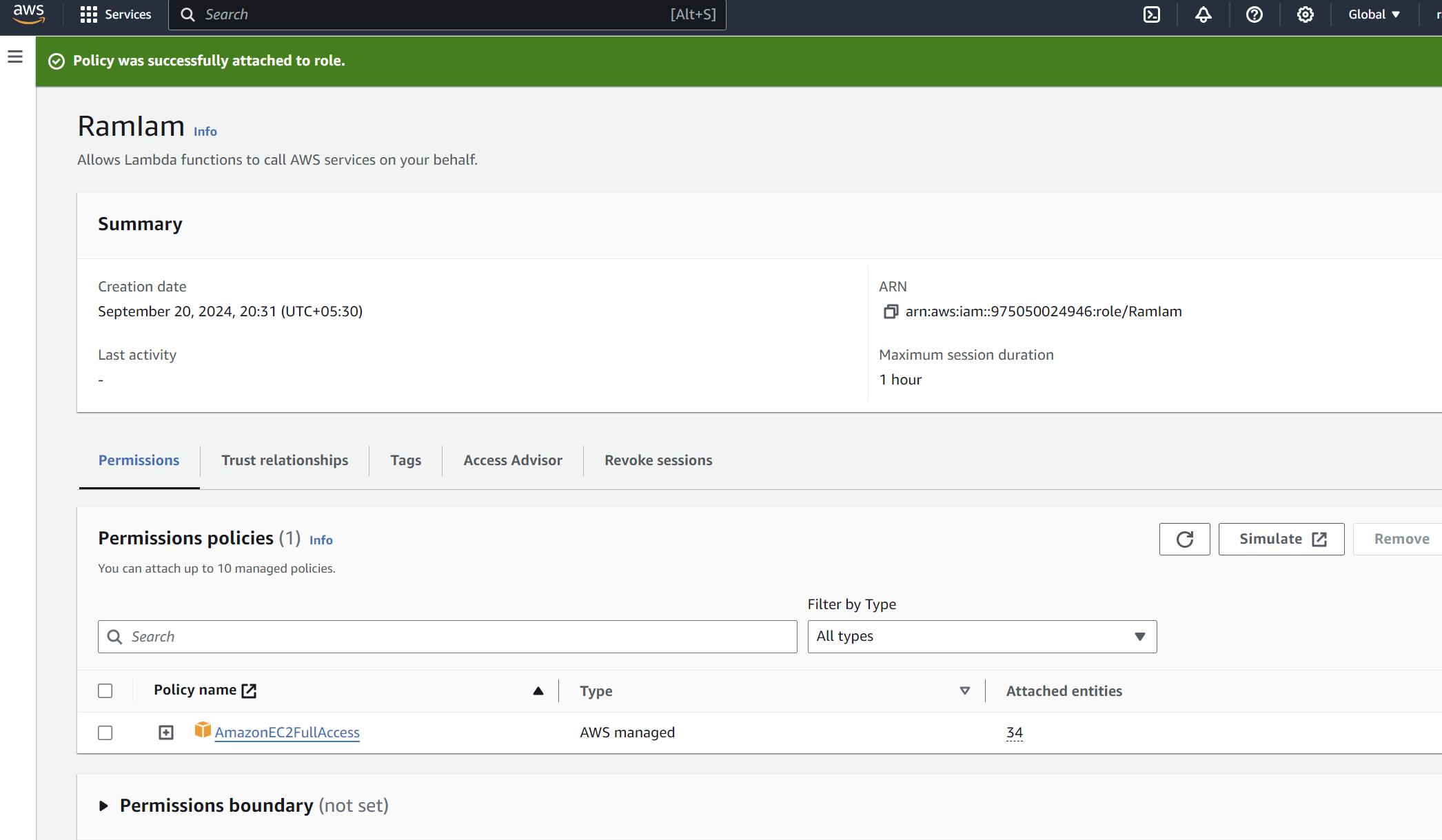


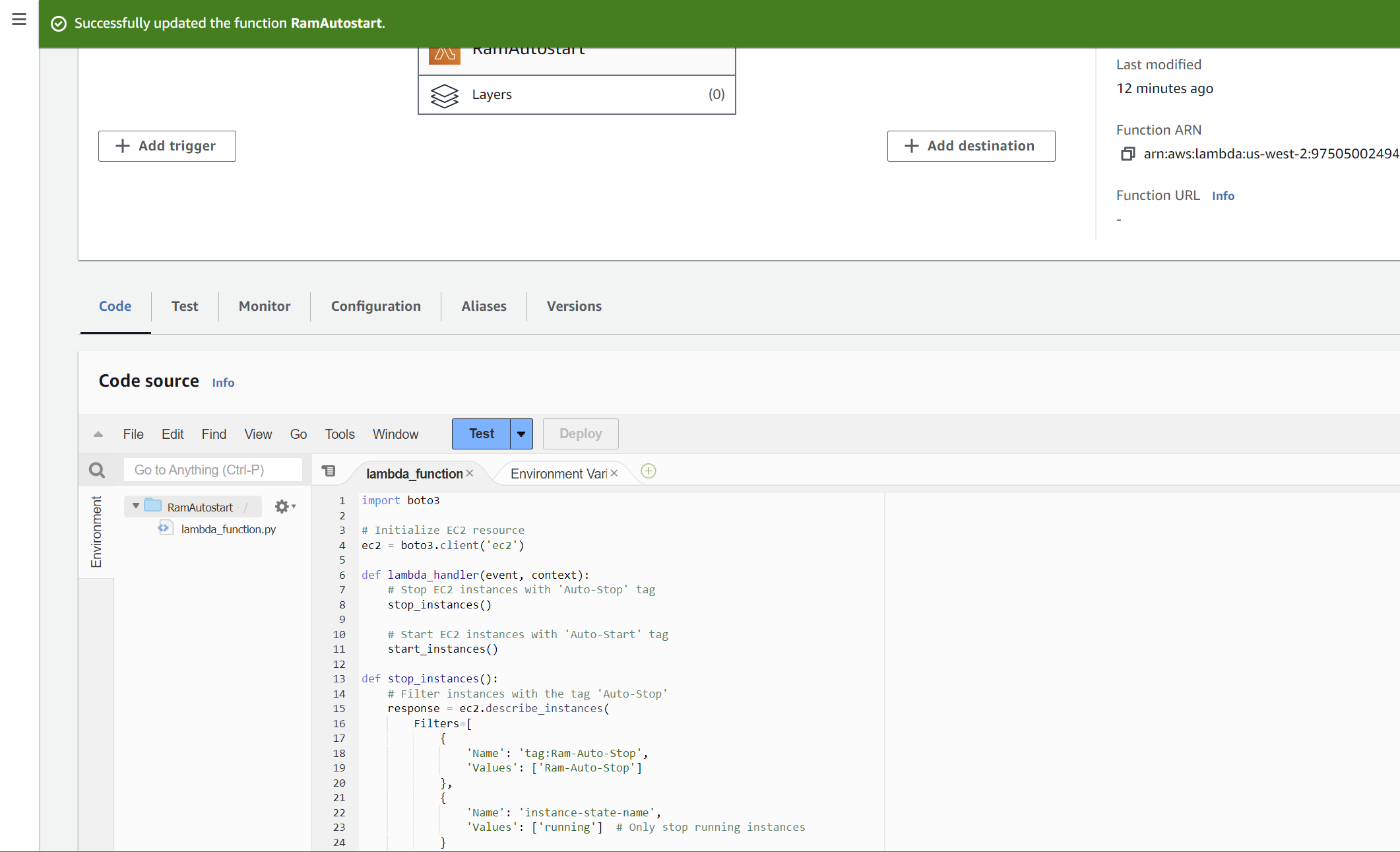
Similarly create EC2 for Auto stop (Ram-Auto-Stop) and make sure it is in stopped





1. Created the IAM role for lamda function



1. Added the python code in lamda function.

**Python code:**

import boto3

# Initialize EC2 resource

ec2 = boto3.client('ec2')

def lambda\_handler(event, context):

# Stop EC2 instances with 'Auto-Stop' tag

stop\_instances()

# Start EC2 instances with 'Auto-Start' tag

start\_instances()

def stop\_instances():

# Filter instances with the tag 'Auto-Stop'

response = ec2.describe\_instances(

Filters=[

{

'Name': 'tag:Ram-Auto-Stop',

'Values': ['Ram-Auto-Stop']

},

{

'Name': 'instance-state-name',

'Values': ['running'] # Only stop running instances

}

]

)

instances\_to\_stop = []

for reservation in response['Reservations']:

for instance in reservation['Instances']:

instances\_to\_stop.append(instance['InstanceId'])

# Stop instances if any are found

if instances\_to\_stop:

print(f"Stopping instances: {instances\_to\_stop}")

ec2.stop\_instances(InstanceIds=instances\_to\_stop)

else:

print("No instances found with the 'Auto-Stop' tag that are running.")

def start\_instances():

# Filter instances with the tag 'Auto-Start'

response = ec2.describe\_instances(

Filters=[

{

'Name': 'tag:Ram- Auto-Start',

'Values': ['Ram- Auto-Start']

},

{

'Name': 'instance-state-name',

'Values': ['stopped'] # Only start stopped instances

}

]

)

instances\_to\_start = []

for reservation in response['Reservations']:

for instance in reservation['Instances']:

instances\_to\_start.append(instance['InstanceId'])

# Start instances if any are found

if instances\_to\_start:

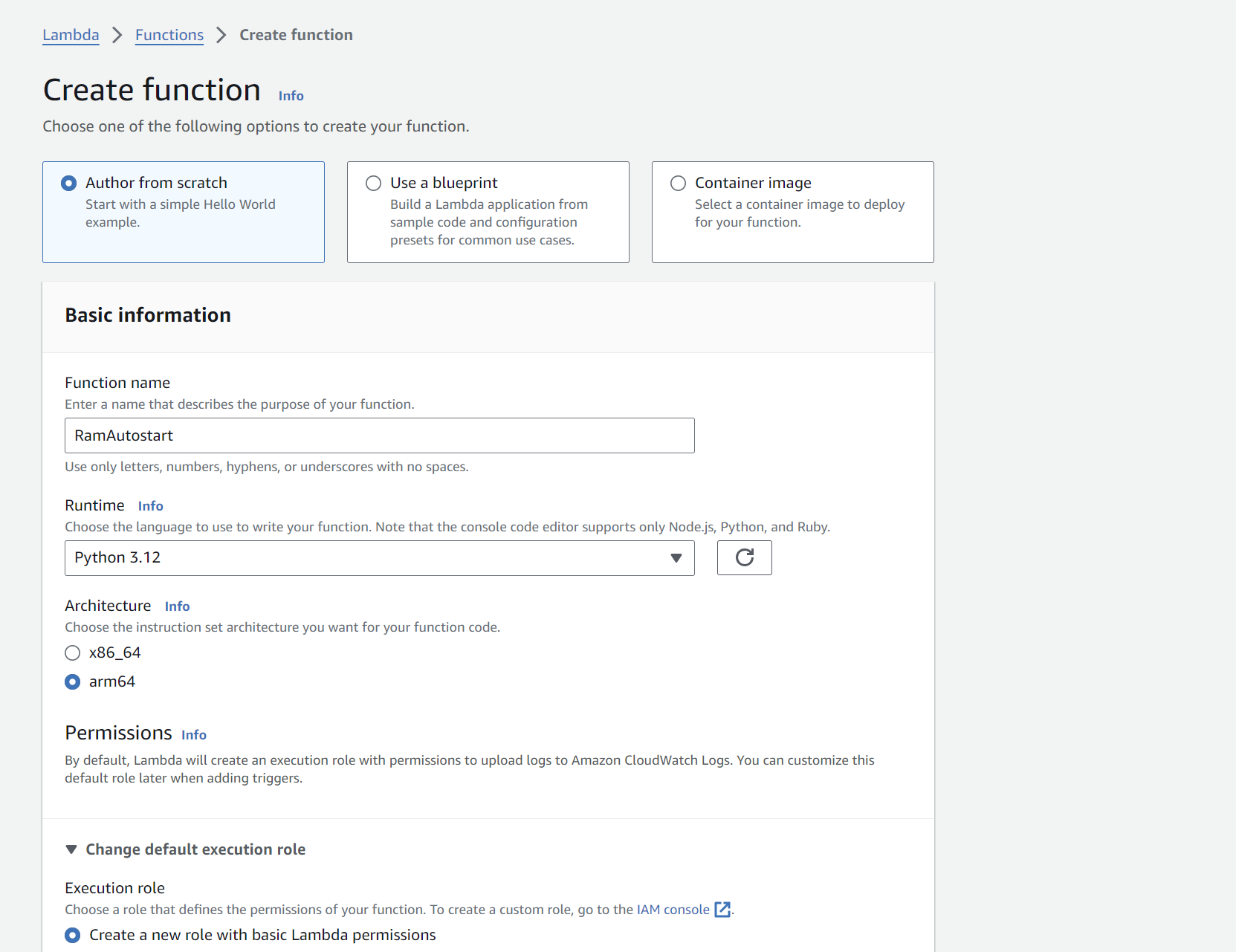
print(f"Starting instances: {instances\_to\_start}")

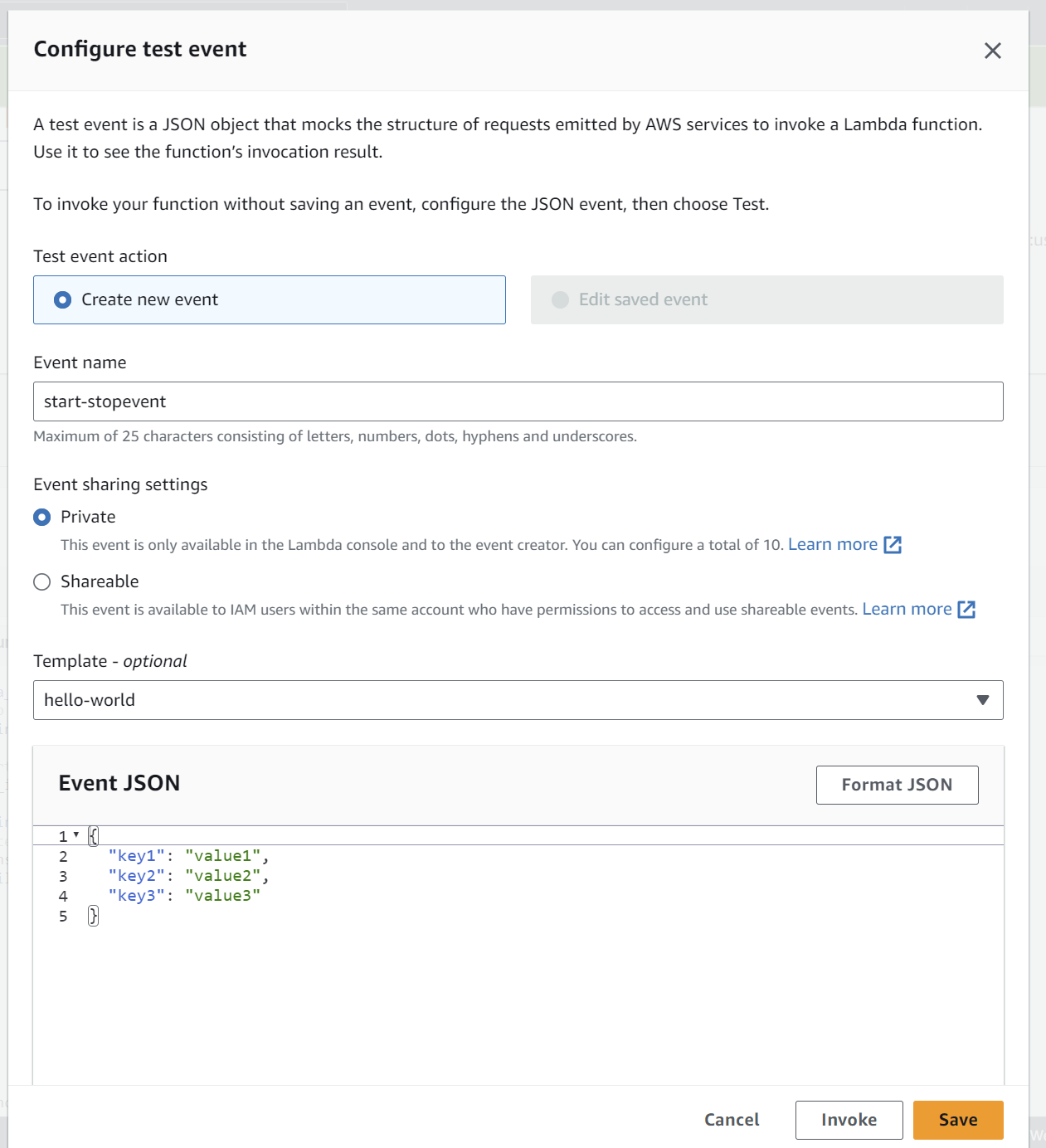
ec2.start\_instances(InstanceIds=instances\_to\_start)

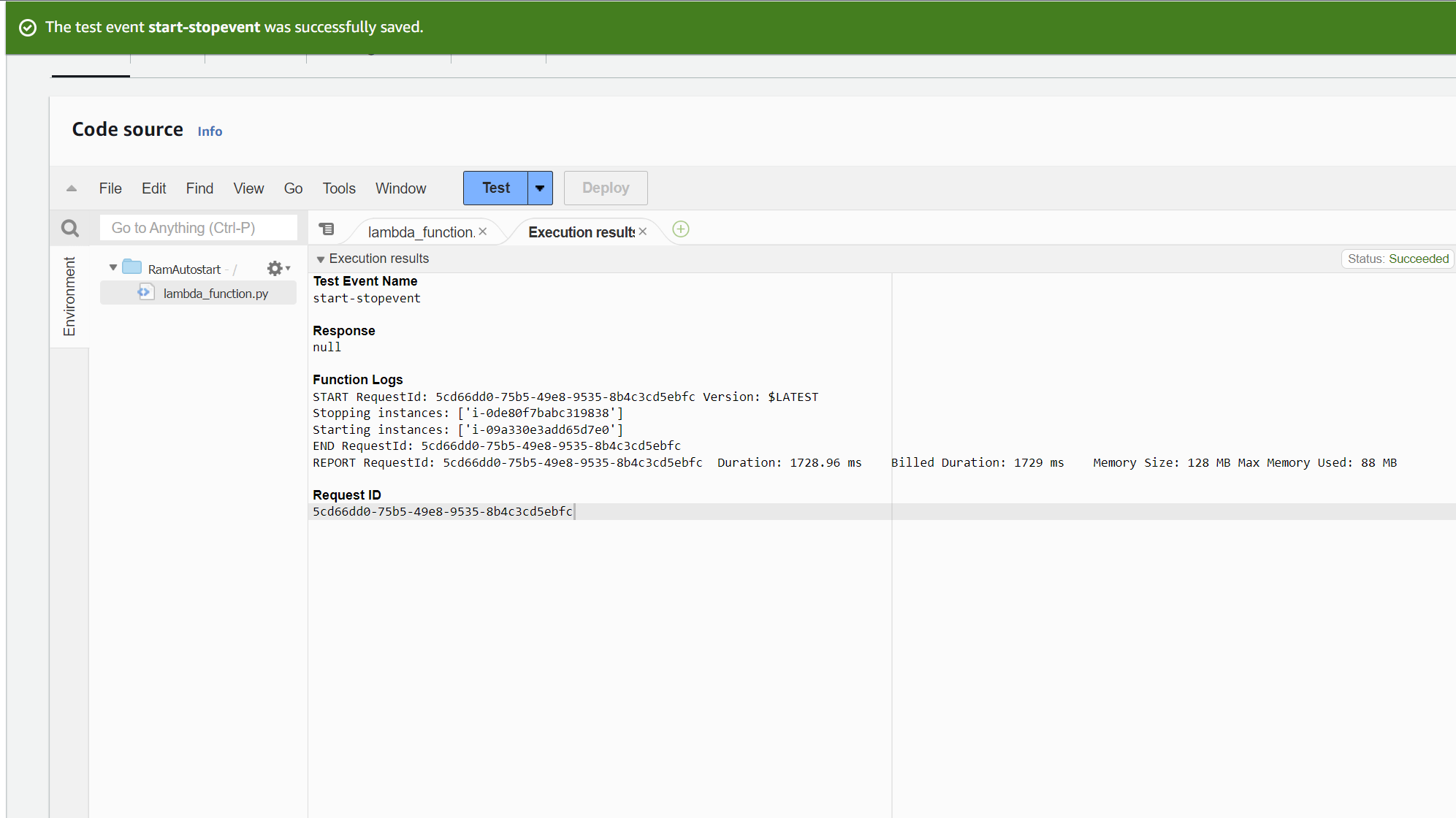
else:

print("No instances found with the 'Auto-Start' tag that are stopped.")

1. Created Lamda Fuction

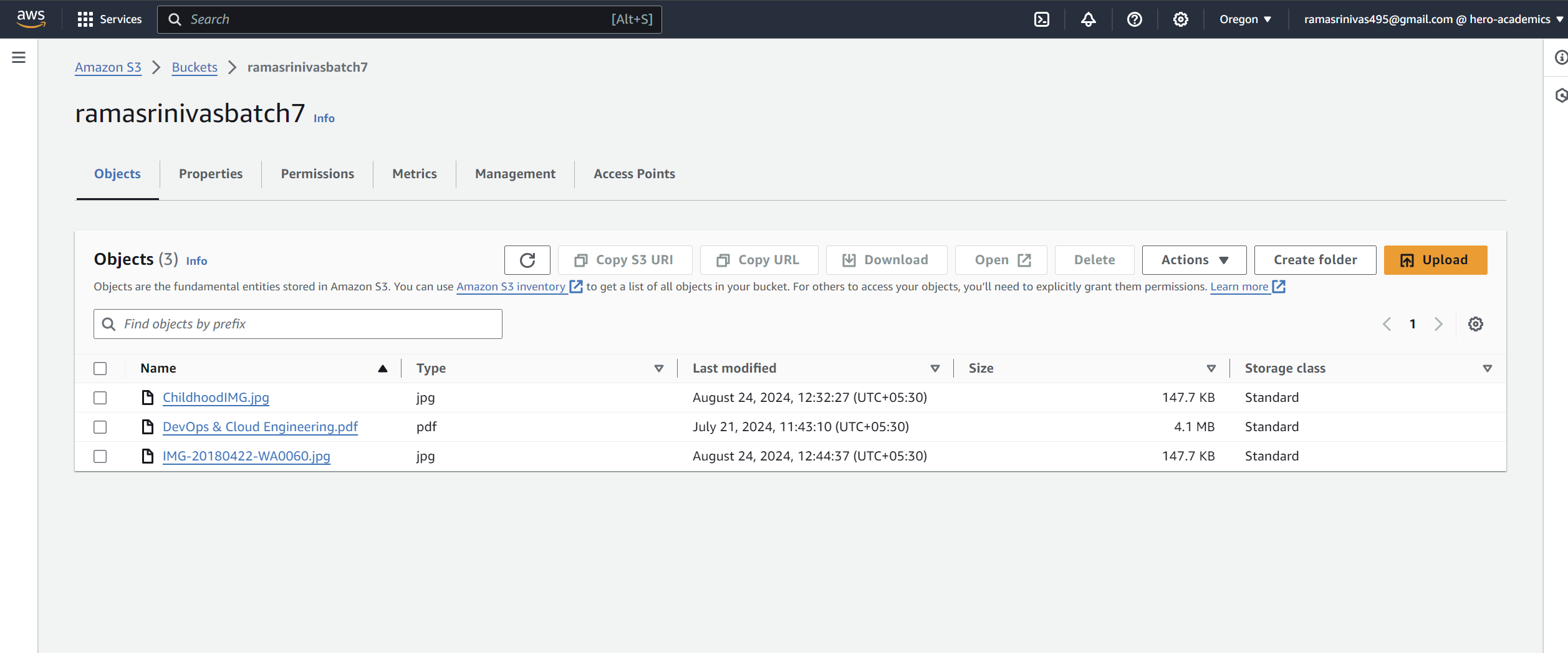




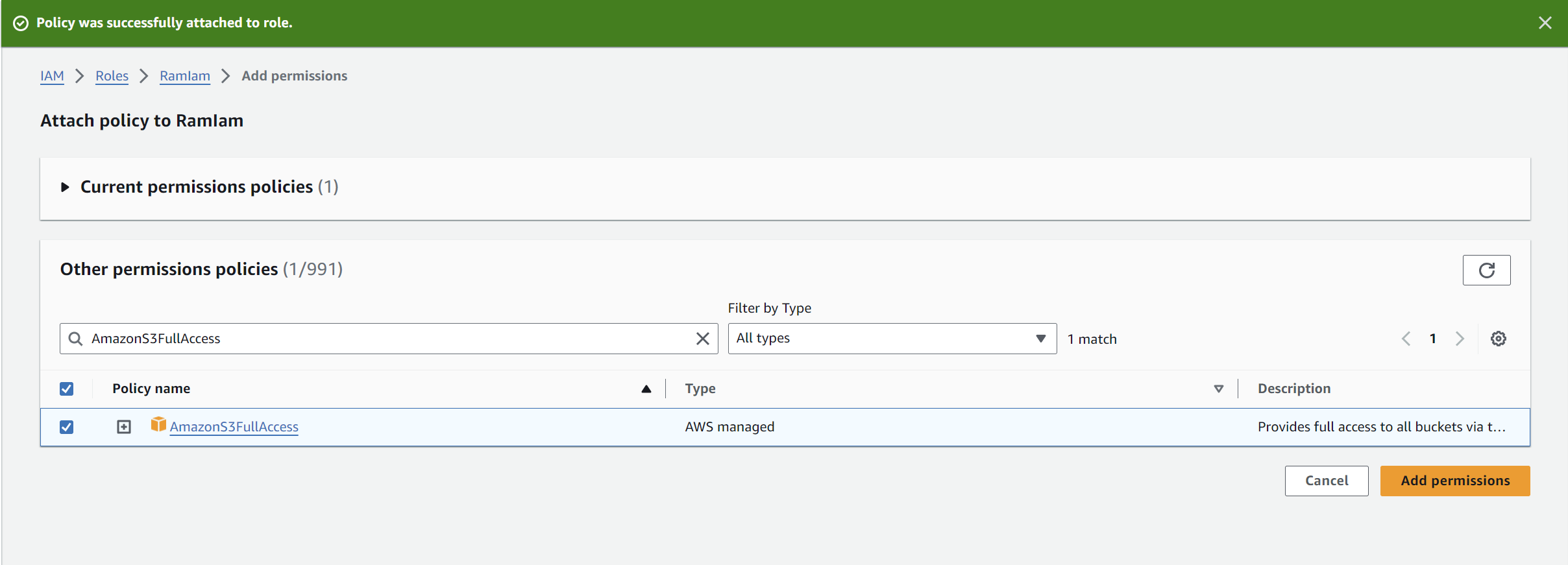
1. Deployed and tested the code. code executed sucesfully.

**Assignment 2**: Automated S3 Bucket Cleanup Using AWS Lambda and Boto3

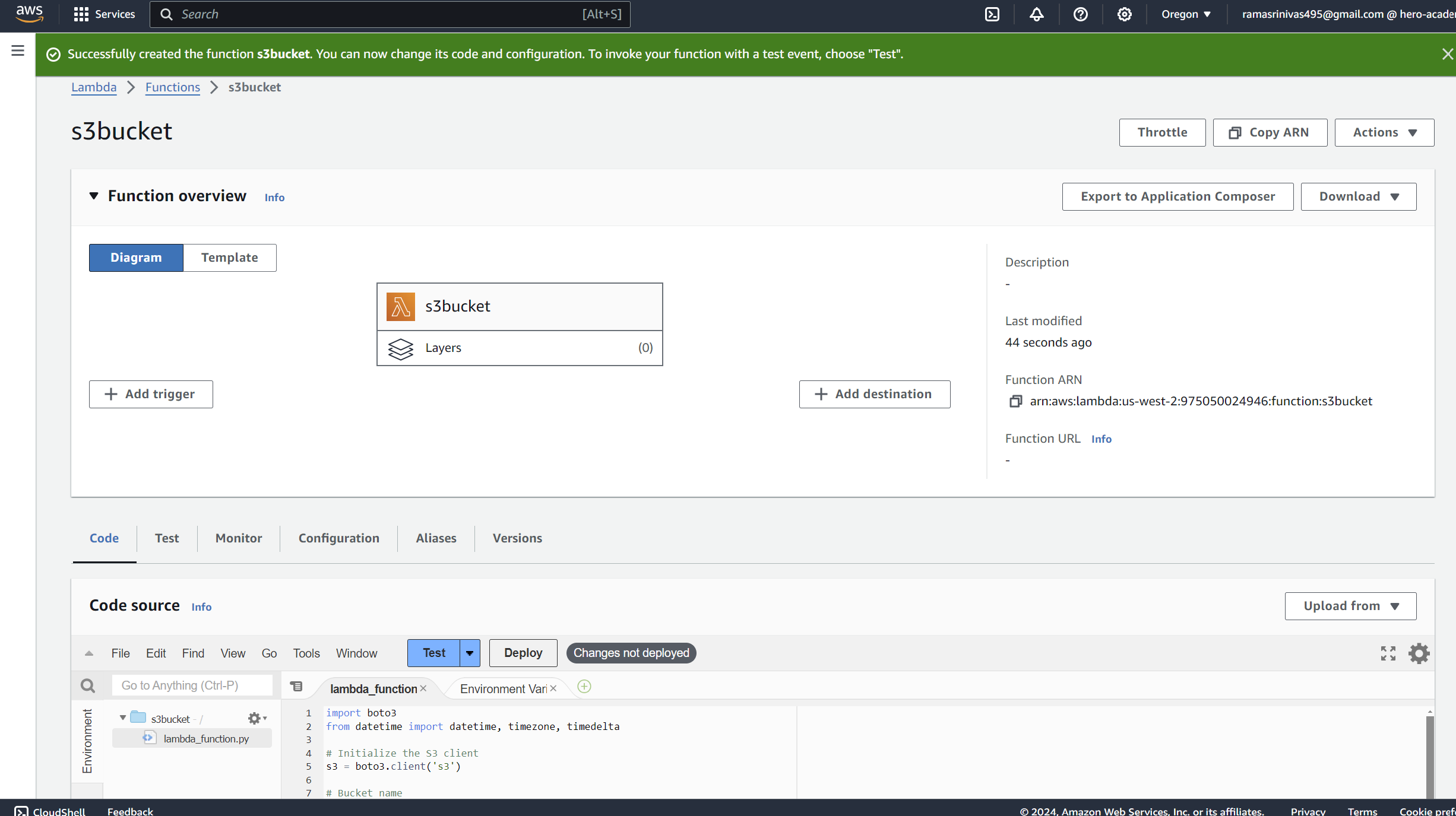
1. S3 bucket Setup



1. created the IAM role for lamda function



1. Using existing Lamda function, Modified the Python code



**Python code:**

import boto3

from datetime import datetime, timezone, timedelta

# Initialize the S3 client

s3 = boto3.client('s3')

# Bucket name

BUCKET\_NAME = 'ramasrinivasbatch7'

# Time delta for 30 days

DAYS\_OLD = 30

now = datetime.now(timezone.utc)

cutoff\_date = now - timedelta(days=DAYS\_OLD)

def delete\_old\_objects():

# List all objects in the bucket

response = s3.list\_objects\_v2(Bucket=BUCKET\_NAME)

if 'Contents' not in response:

print("No objects found in the bucket.")

return

deleted\_objects = []

for obj in response['Contents']:

# Get the object's LastModified time

last\_modified = obj['LastModified']

# If the object is older than the cutoff date

if last\_modified < cutoff\_date:

print(f"Deleting object: {obj['Key']}")

s3.delete\_object(Bucket=BUCKET\_NAME, Key=obj['Key'])

deleted\_objects.append(obj['Key'])

# Log the names of deleted objects

if deleted\_objects:

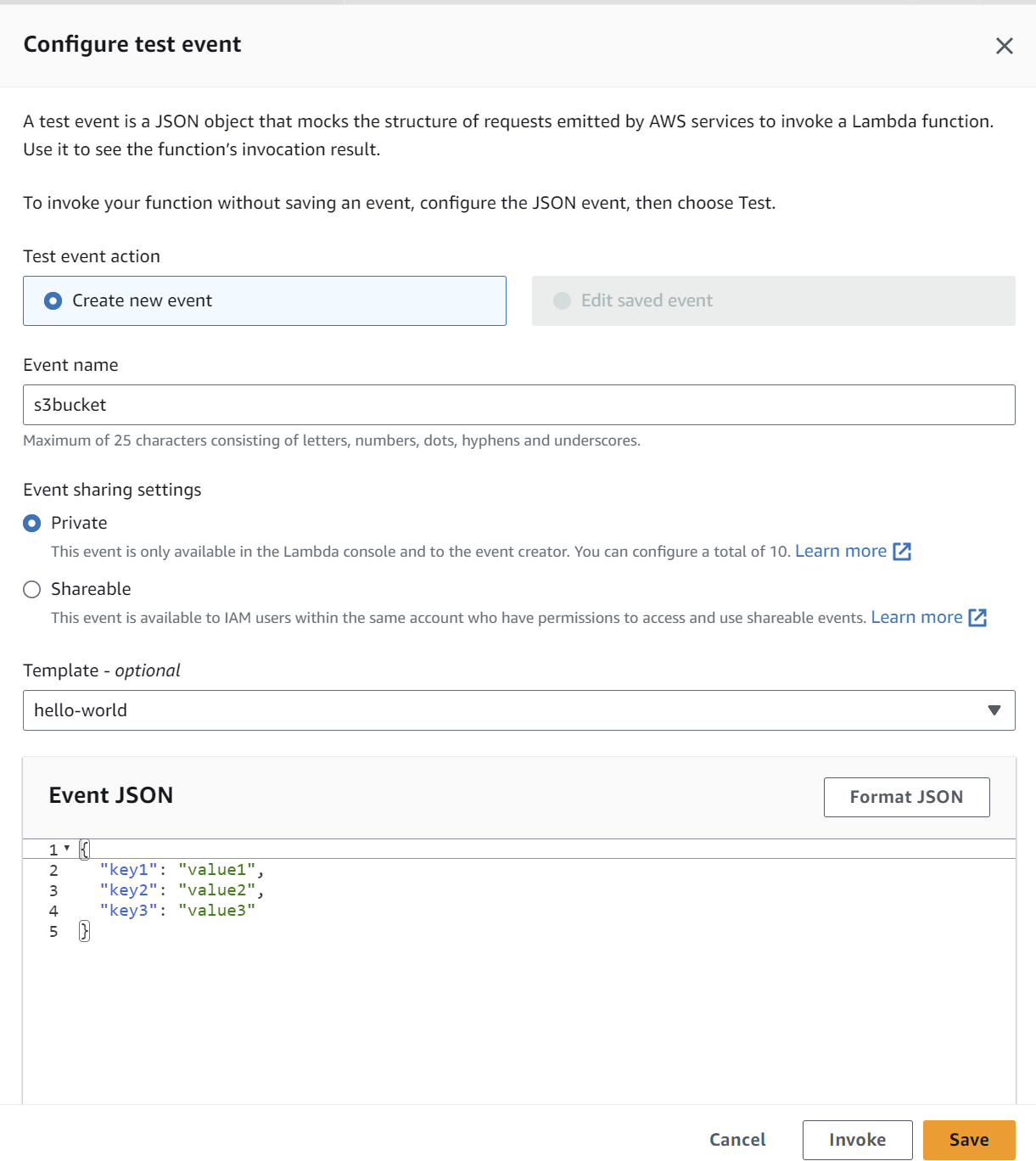
print(f"Deleted objects: {deleted\_objects}")

else:

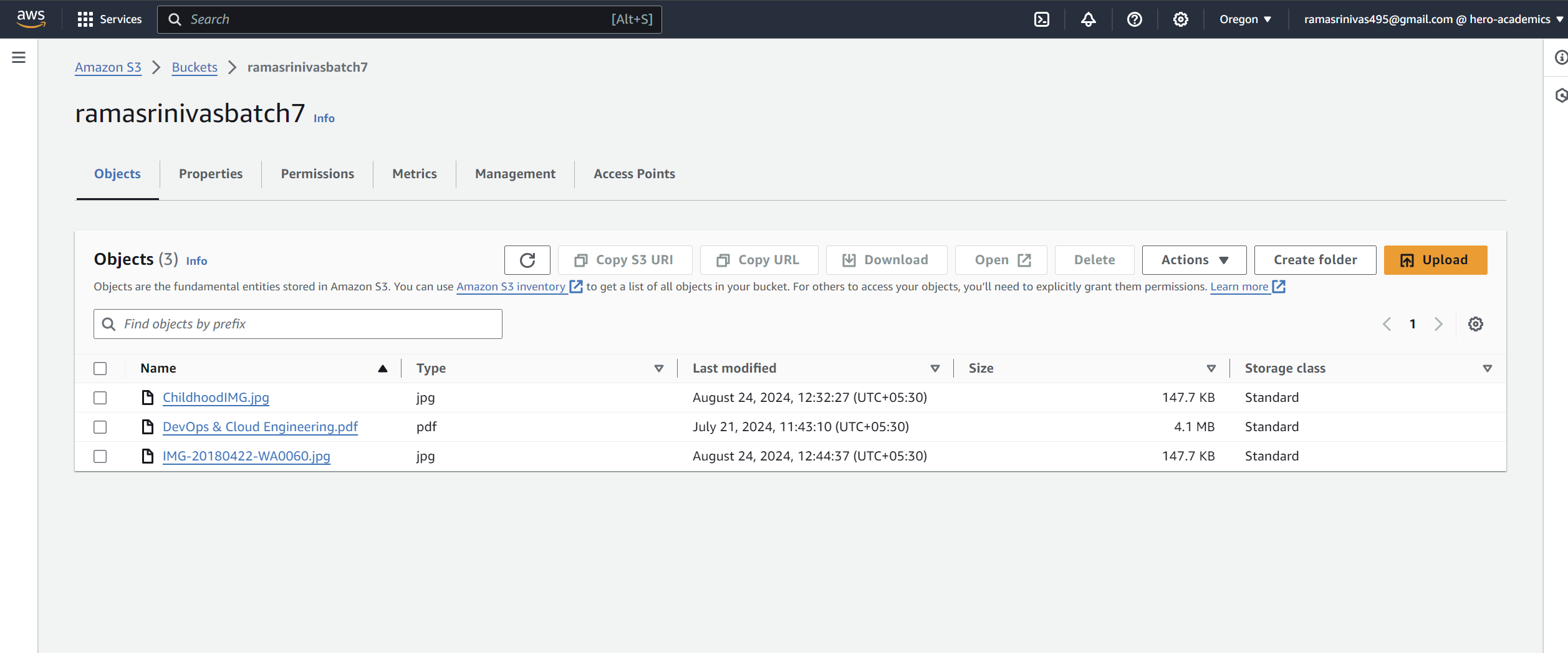
print("No objects older than 30 days found.")

def lambda\_handler(event, context):

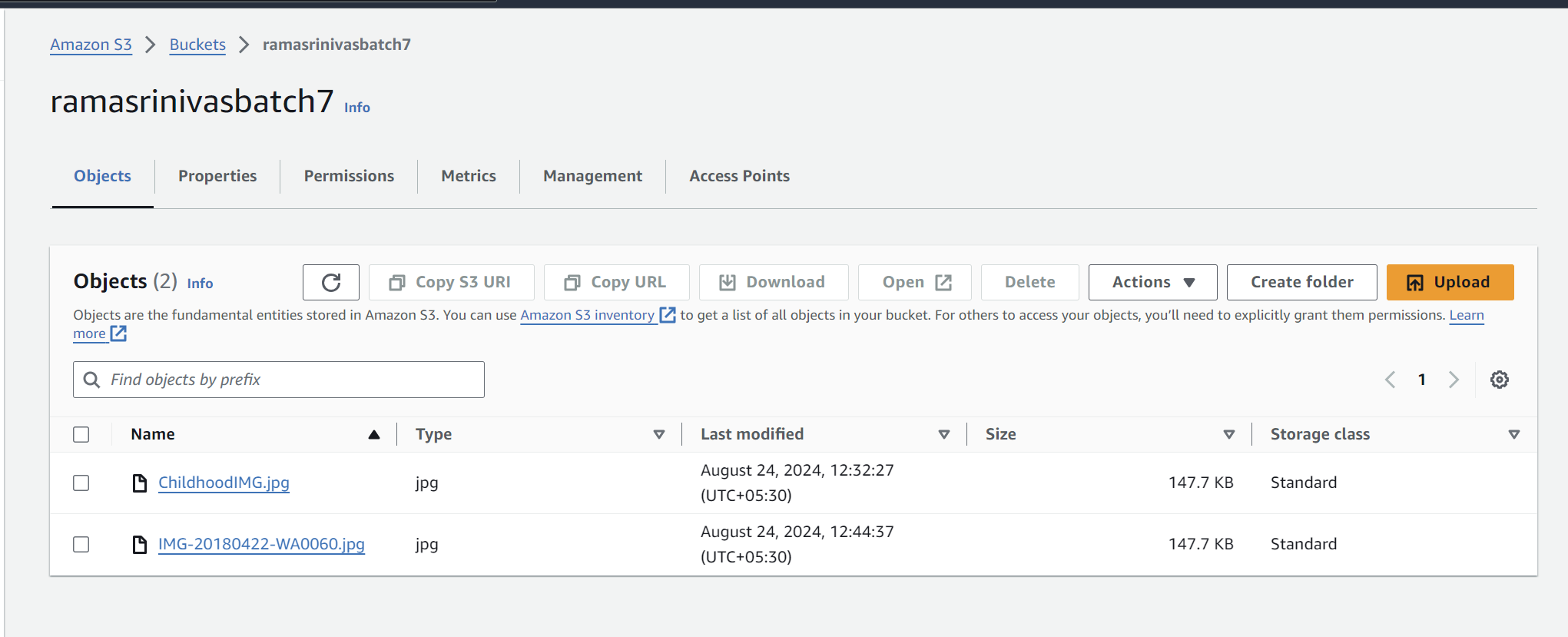
delete\_old\_objects()



1. Deployed and tested the code(not able to add files older 30 days so modified the code with 30 sec).

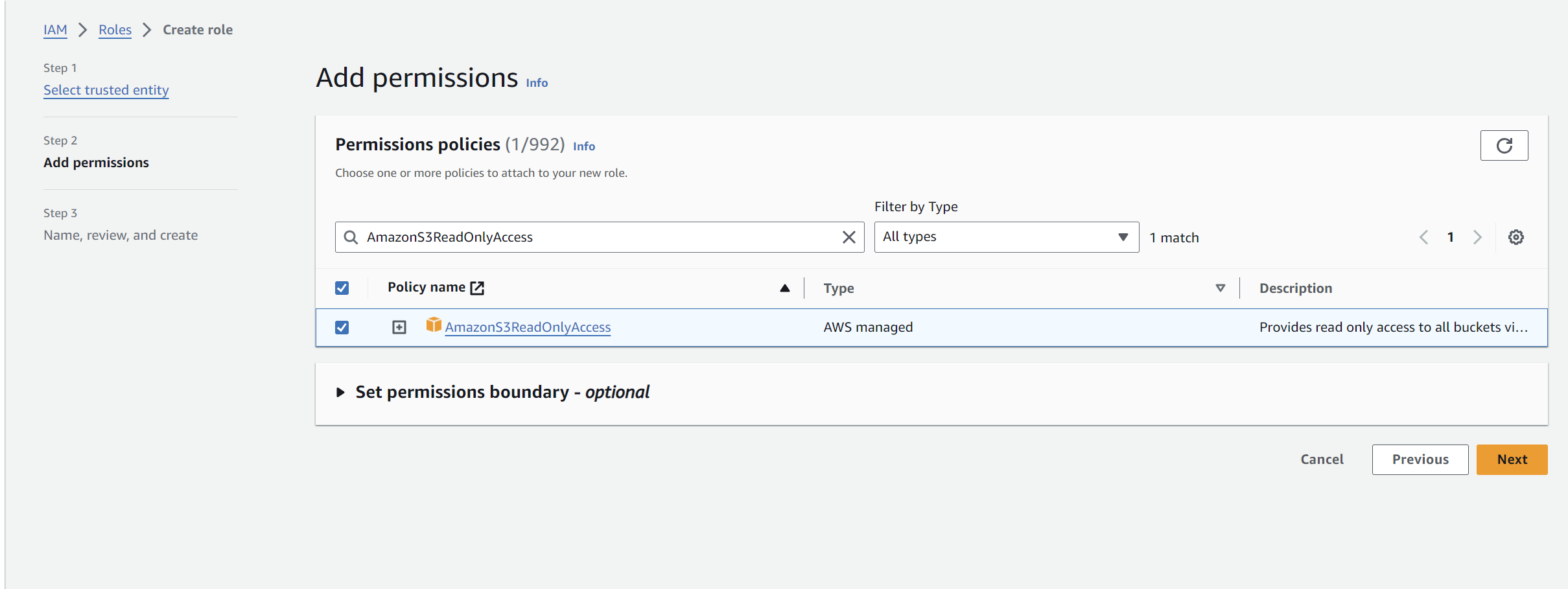


Code executed deleted old file



Q3: **Assignment 3**: Monitor Unencrypted S3 Buckets Using AWS Lambda and Boto3

1. created the IAM role for lamda function



1. Using existing Lamda function, Modified the Python code



**Python code:**

import boto3

from botocore.exceptions import ClientError

s3 = boto3.client('s3')

def lambda\_handler(event, context):

detect\_unencrypted\_buckets()

def list\_all\_buckets():

try:

response = s3.list\_buckets()

return response['Buckets']

except ClientError as e:

print(f"Error listing buckets: {e}")

return []

def check\_bucket\_encryption(bucket\_name):

try:

response = s3.get\_bucket\_encryption(Bucket=bucket\_name)

encryption = response['ServerSideEncryptionConfiguration']

return True # Bucket is encrypted

except ClientError as e:

error\_code = e.response['Error']['Code']

if error\_code == 'ServerSideEncryptionConfigurationNotFoundError':

return False # Bucket is not encrypted

else:

print(f"Error checking encryption for bucket {bucket\_name}: {e}")

return None # Error occurred

def detect\_unencrypted\_buckets():

buckets = list\_all\_buckets()

if not buckets:

print("No buckets found or error in listing buckets.")

return

unencrypted\_buckets = []

for bucket in buckets:

bucket\_name = bucket['Name']

is\_encrypted = check\_bucket\_encryption(bucket\_name)

if is\_encrypted is False:

print(f"Bucket without encryption: {bucket\_name}")

unencrypted\_buckets.append(bucket\_name)

elif is\_encrypted is None:

print(f"Skipping bucket {bucket\_name} due to an error.")

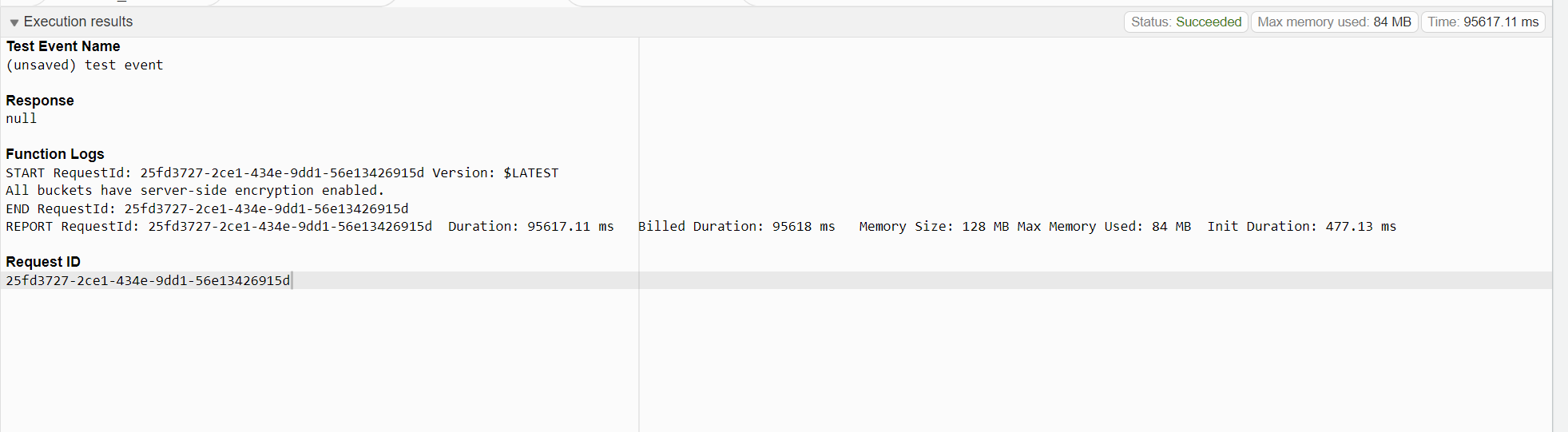
if unencrypted\_buckets:

print(f"Unencrypted buckets: {unencrypted\_buckets}")

else:

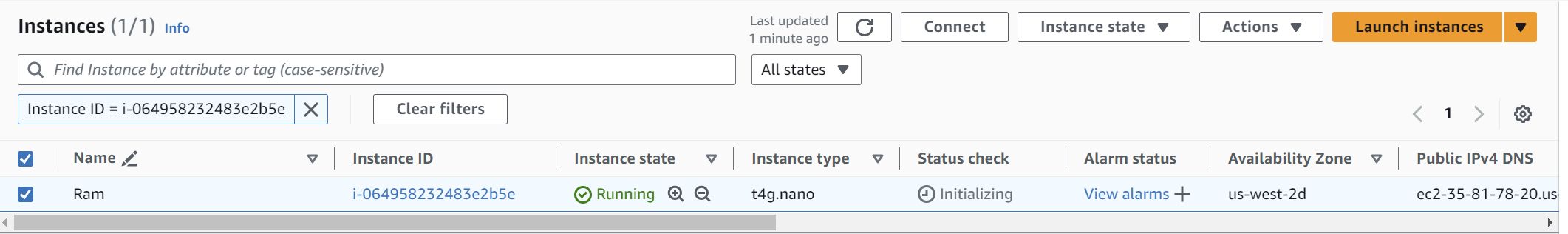
print("All buckets have server-side encryption enabled.")

1. Code executed properly

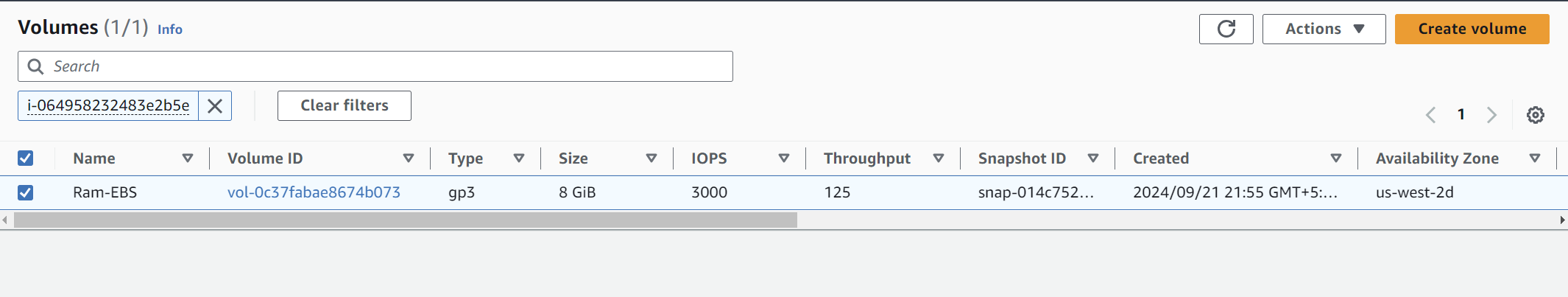


**Assignment 4**: Automatic EBS Snapshot and Cleanup Using AWS Lambda and Boto3

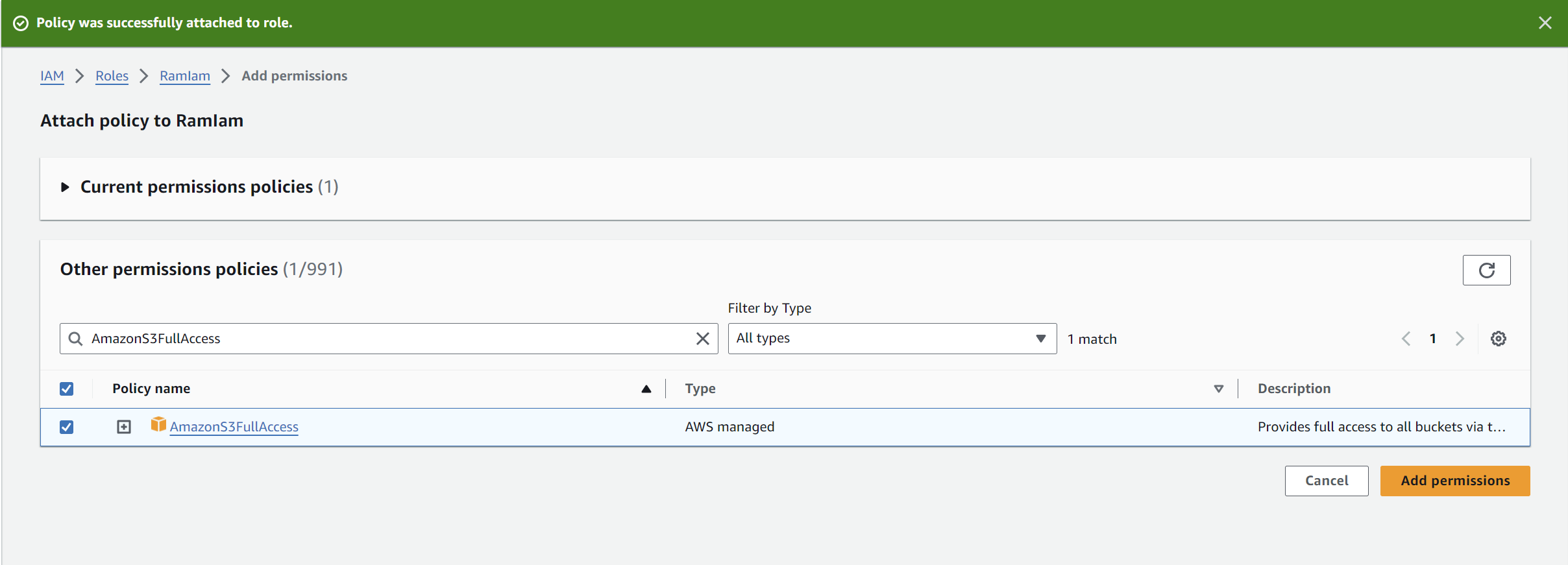
Created EC2 instance



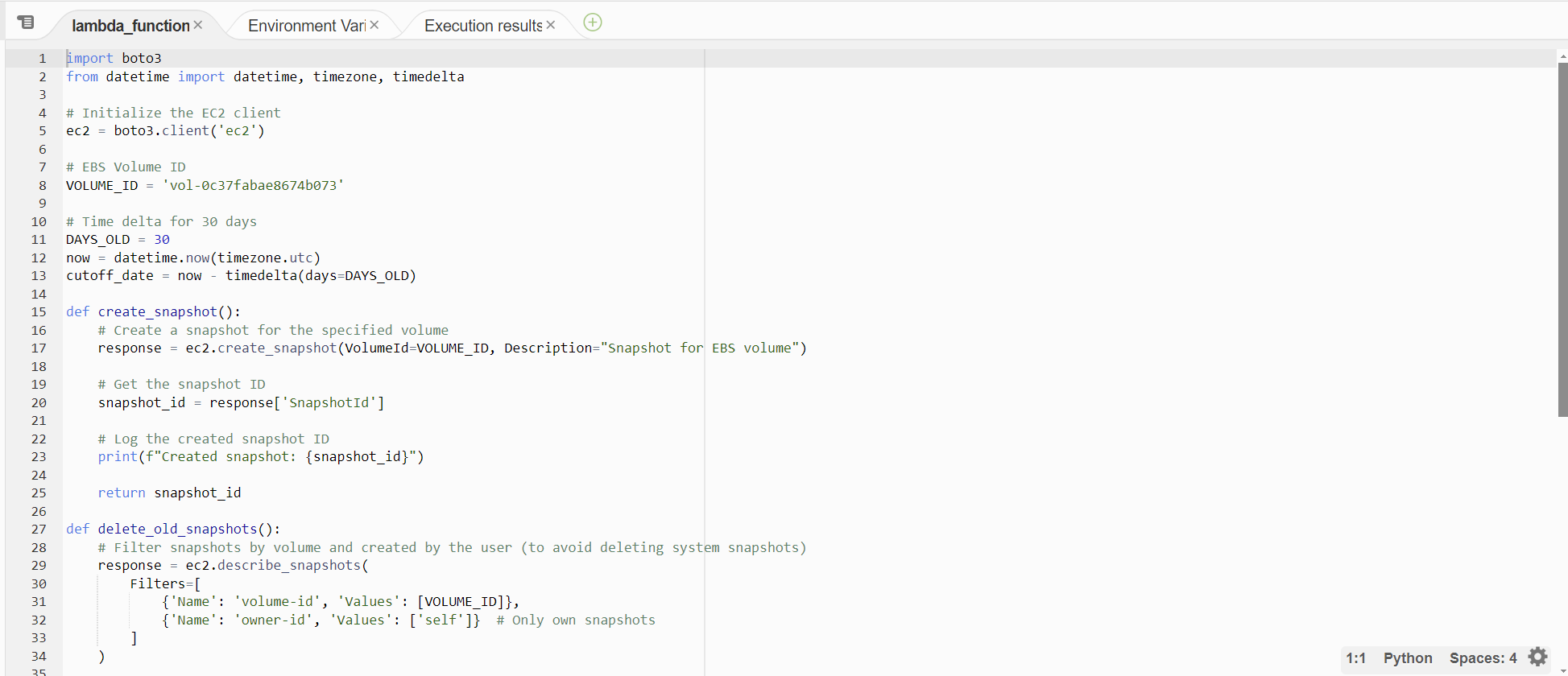
Created EBS volume



Added role AmazonEC2FullAccess to IAM



Using existing Lamda function, Modified the Python code



**Python code:**

import boto3

from datetime import datetime, timezone, timedelta

# Initialize the EC2 client

ec2 = boto3.client('ec2')

# EBS Volume ID

VOLUME\_ID = 'vol-0c37fabae8674b073'

# Time delta for 30 days

DAYS\_OLD = 30

now = datetime.now(timezone.utc)

cutoff\_date = now - timedelta(days=DAYS\_OLD)

def create\_snapshot():

# Create a snapshot for the specified volume

response = ec2.create\_snapshot(VolumeId=VOLUME\_ID, Description="Snapshot for EBS volume")

# Get the snapshot ID

snapshot\_id = response['SnapshotId']

# Log the created snapshot ID

print(f"Created snapshot: {snapshot\_id}")

return snapshot\_id

def delete\_old\_snapshots():

# Filter snapshots by volume and created by the user (to avoid deleting system snapshots)

response = ec2.describe\_snapshots(

Filters=[

{'Name': 'volume-id', 'Values': [VOLUME\_ID]},

{'Name': 'owner-id', 'Values': ['self']} # Only own snapshots

]

)

deleted\_snapshots = []

for snapshot in response['Snapshots']:

snapshot\_id = snapshot['SnapshotId']

start\_time = snapshot['StartTime']

# If the snapshot is older than the cutoff date

if start\_time < cutoff\_date:

print(f"Deleting snapshot: {snapshot\_id} (Created on {start\_time})")

ec2.delete\_snapshot(SnapshotId=snapshot\_id)

deleted\_snapshots.append(snapshot\_id)

# Log the deleted snapshot IDs

if deleted\_snapshots:

print(f"Deleted snapshots: {deleted\_snapshots}")

else:

print("No snapshots older than 30 days found.")

def lambda\_handler(event, context):

# Step 1: Create a new snapshot

created\_snapshot\_id = create\_snapshot()

# Step 2: Delete old snapshots older than 30 days

delete\_old\_snapshots()

Code executed

