**Assignment 1: Automated Instance Management Using AWS Lambda and Boto3**

**Objective:** In this assignment, you will gain hands-on experience with AWS Lambda and Boto3, Amazon's SDK for Python. You will create a Lambda function that will automatically manage EC2 instances based on their tags.

**Task:** You're tasked to automate the stopping and starting of EC2 instances based on tags. Specifically:

1. Setup:

- Create two EC2 instances.

- Tag one of them as `Auto-Stop` and the other as `Auto-Start`.

2. Lambda Function Creation:

- Set up an AWS Lambda function.

- Ensure that the Lambda function has the necessary IAM permissions to describe, stop, and start EC2 instances.

3. Coding:

- Using Boto3 in the Lambda function:

- Detect all EC2 instances with the `Auto-Stop` tag and stop them.

- Detect all EC2 instances with the `Auto-Start` tag and start them.

4. Testing:

- Manually invoke the Lambda function.

- Confirm that the instance tagged `Auto-Stop` stops and the one tagged `Auto-Start` starts.

**Instructions:**

1. EC2 Setup:

- Navigate to the EC2 dashboard and create two new t2.micro instances (or any other available free-tier type).

- Tag the first instance with a key `Action` and value `Auto-Stop`.

- Tag the second instance with a key `Action` and value `Auto-Start`.

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.

- Attach the `AmazonEC2FullAccess` policy to this role. (Note: In a real-world scenario, you would want to limit permissions for better security.)

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.

- Choose Python 3.x as the runtime.

- Assign the IAM role created in the previous step.

- Write the Boto3 Python script to:

1. Initialize a boto3 EC2 client.

2. Describe instances with `Auto-Stop` and `Auto-Start` tags.

3. Stop the `Auto-Stop` instances and start the `Auto-Start` instances.

4. Print instance IDs that were affected for logging purposes.

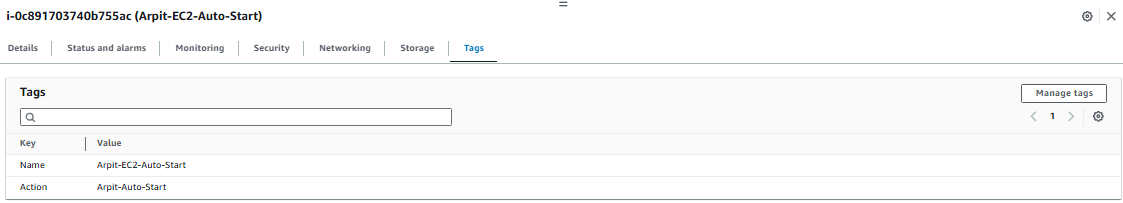
4. Manual Invocation:

- After saving your function, manually trigger it.

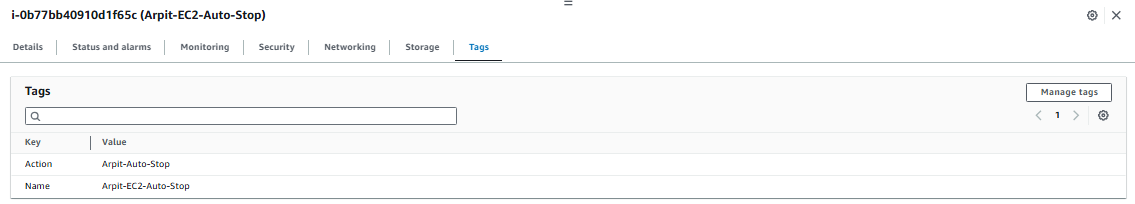
- Go to the EC2 dashboard and confirm that the instances' states have changed according to their tags.

1. Create two EC2 instances:

Created an EC2 instance and associated with the Tag with Key as ‘Action’ and Value ‘Arpit-Auto-Start’:

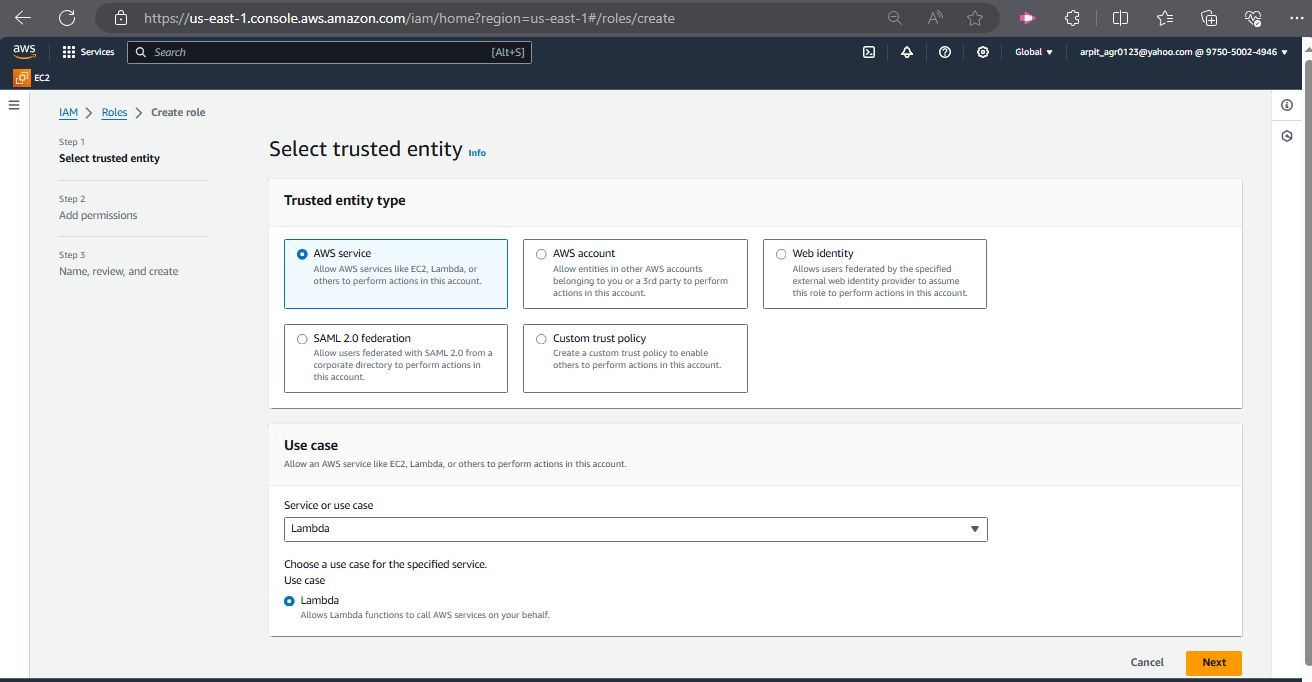


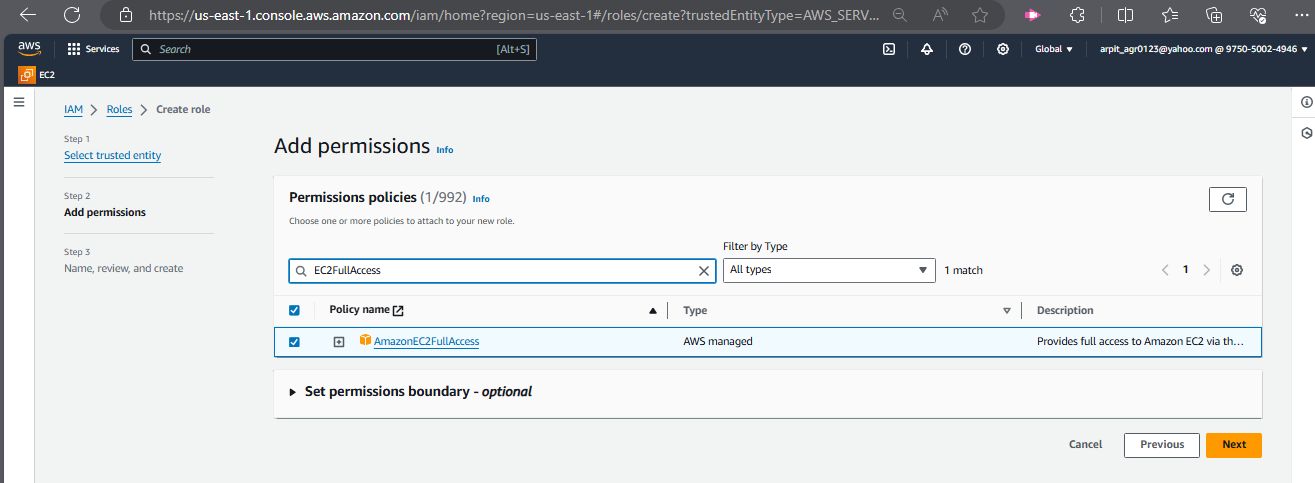
Created an EC2 instance and associated with the Tag with Key as ‘Action’ and Value ‘Arpit-Auto-Stop’:



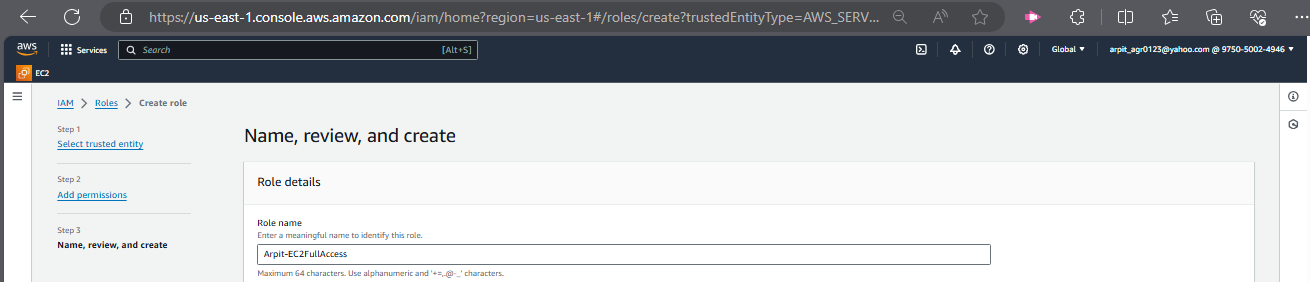
1. Create an IAM role for EC2 permissions:

Creating an IAM role for Lambda function:

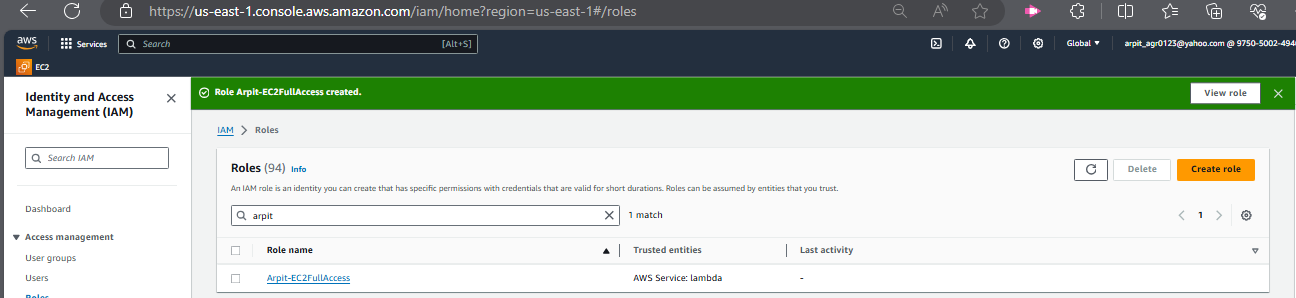


Adding AmazonEC2FullAccess permission:

Naming the role:



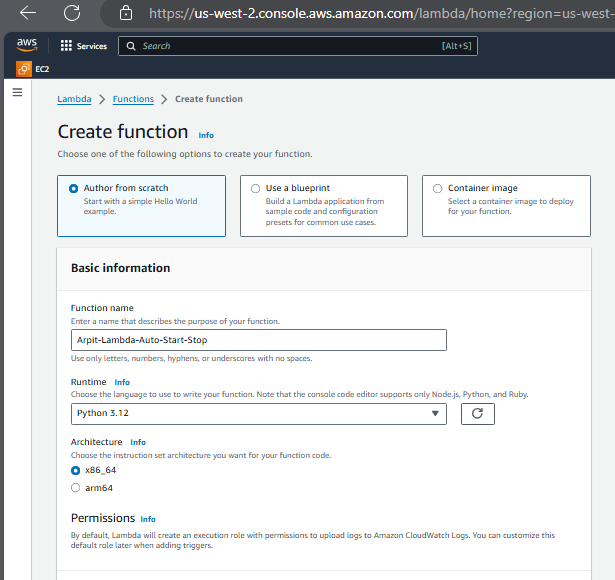
New IAM role has been created successfully:

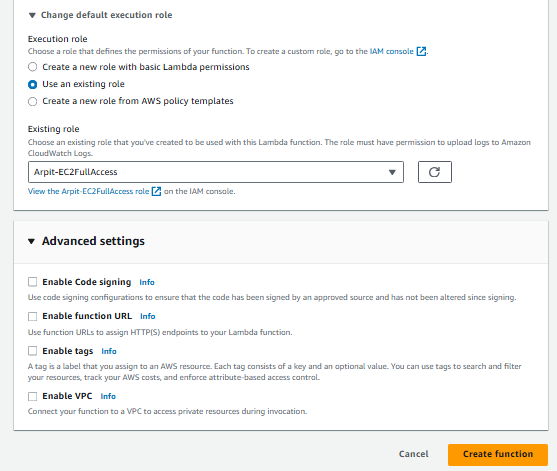


1. Set up an AWS Lambda function:

Creating a Lambda Function for Auto Start and Auto Stop EC2 instances:

Please remember to set a timeout of at least 10 seconds to prevent the function from failing in case of long execution time.



Selecting the new IAM role that we just created for the Lambda function:

Code: Boto3 Python script to automatically manage EC2 instances based on their tags.

import boto3

def lambda\_handler(event, context):

ec2\_client = boto3.client('ec2')

# Get all instances with the 'Auto-Stop' tag

auto\_stop\_instances = ec2\_client.describe\_instances(

Filters=[

{'Name': 'tag:Action', 'Values': ['Arpit-Auto-Stop']},

{'Name': 'instance-state-name', 'Values': ['running']}

]

)

# Get all instances with the 'Auto-Start' tag

auto\_start\_instances = ec2\_client.describe\_instances(

Filters=[

{'Name': 'tag:Action', 'Values': ['Arpit-Auto-Start']},

{'Name': 'instance-state-name', 'Values': ['stopped']}

]

)

# Stop the instances with the tag 'Auto-Stop'

stop\_instance\_ids = [instance['InstanceId']

for reservation in auto\_stop\_instances['Reservations']

for instance in reservation['Instances']]

if stop\_instance\_ids:

ec2\_client.stop\_instances(InstanceIds=stop\_instance\_ids)

print(f'Stopped instances: {stop\_instance\_ids}')

else:

print('No running instances with Auto-Stop Tag to stop.')

# Start the instances with the tag 'Auto-Start'

start\_instance\_ids = [instance['InstanceId']

for reservation in auto\_start\_instances['Reservations']

for instance in reservation['Instances']]

if start\_instance\_ids:

ec2\_client.start\_instances(InstanceIds=start\_instance\_ids)

print(f'Started instances: {start\_instance\_ids}')

else:

print('No stopped instances with Auto-Start Tag to start.')

return {

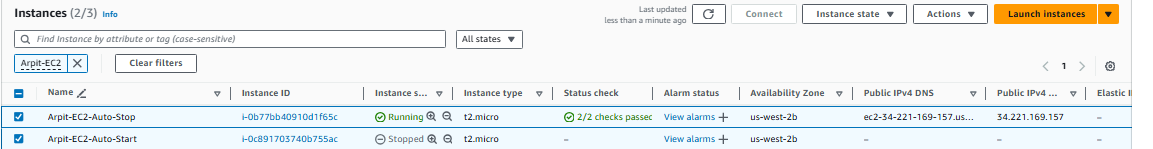
'statusCode': 200,

'body': ' Automated Instance Management is completed'

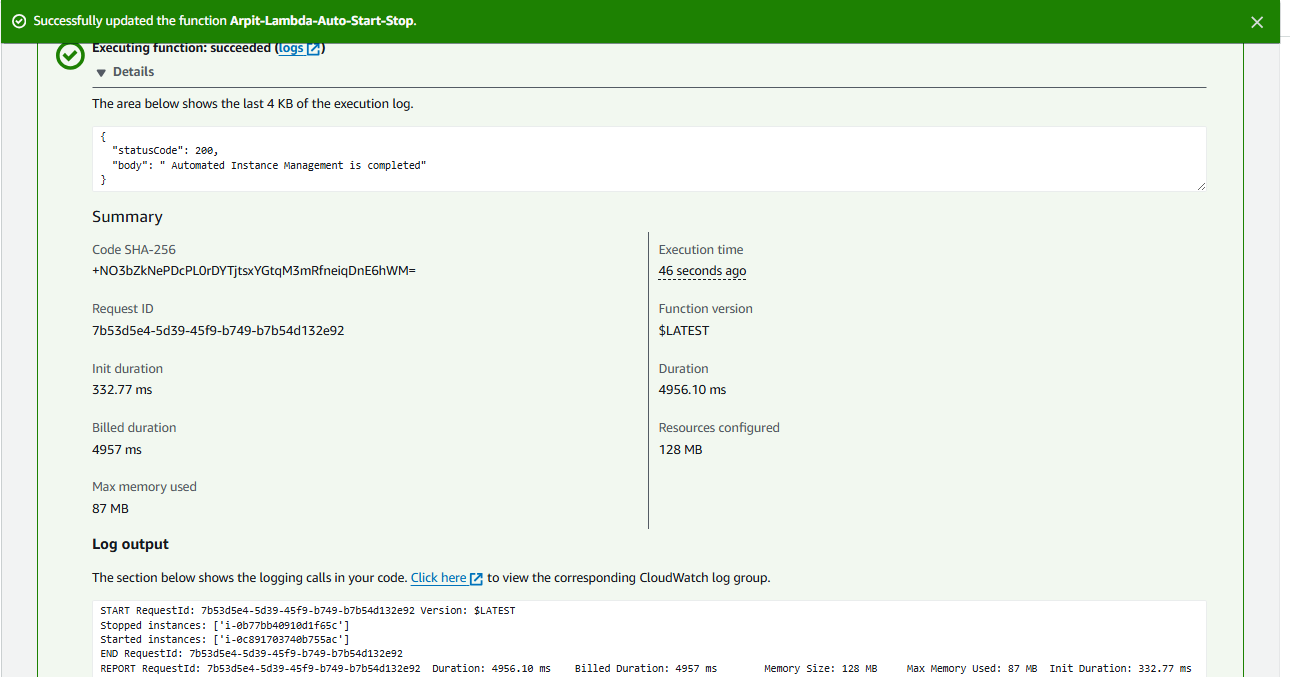
}

1. Testing:

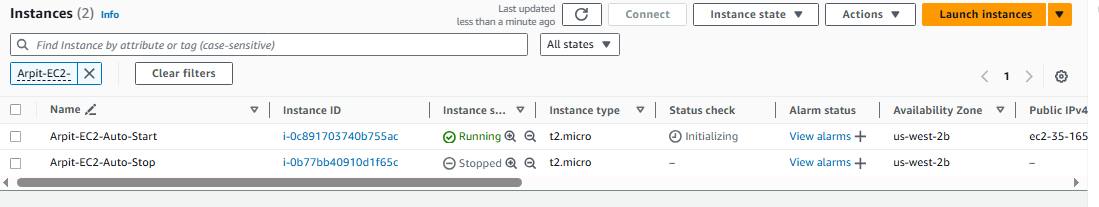
State of EC2 instances before executing the Lambda function: The instance with ‘Arpit-Auto-Start’ tag is in stopped state and ‘Arpit-Auto-Stop' tag is in running state.



Deployed the changes and manually executed the Lambda Function and it executed successfully as shown below:



State of EC2 instances post execution the Lambda function: The instance with ‘Arpit-Auto-Start’ tag has started running and ‘Arpit-Auto-Stop' tag has been stopped.



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

**Objective**: To gain experience with AWS Lambda and Boto3 by creating a Lambda function that will automatically clean up old files in an S3 bucket.

**Task:** Automate the deletion of files older than 30 days in a specific S3 bucket.

**Instructions:**

1. S3 Setup:

- Navigate to the S3 dashboard and create a new bucket.

- Upload multiple files to this bucket, ensuring that some files are older than 30 days (you may need to adjust your system's date temporarily for this or use old files).

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.

- Attach the `AmazonS3FullAccess` policy to this role. (Note: For enhanced security in real-world scenarios, use more restrictive permissions.)

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.

- Choose Python 3.x as the runtime.

- Assign the IAM role created in the previous step.

- Write the Boto3 Python script to:

1. Initialize a boto3 S3 client.

2. List objects in the specified bucket.

3. Delete objects older than 30 days.

4. Print the names of deleted objects for logging purposes.

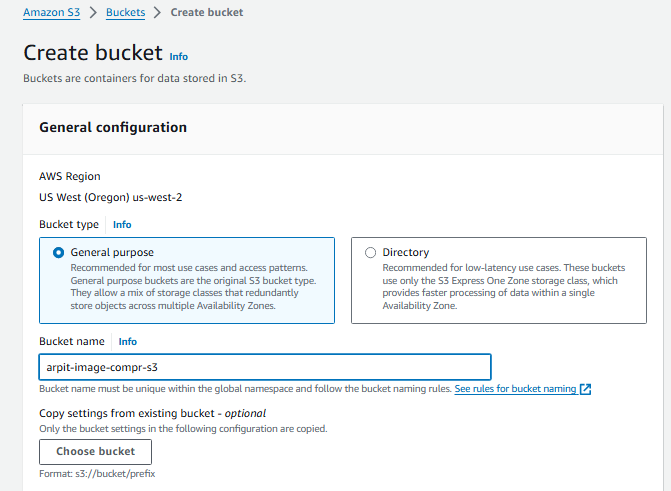
4. Manual Invocation:

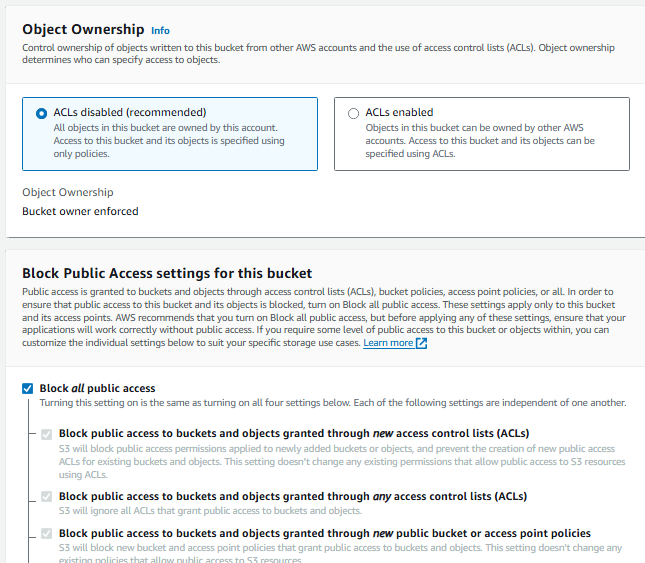
- After saving your function, manually trigger it.

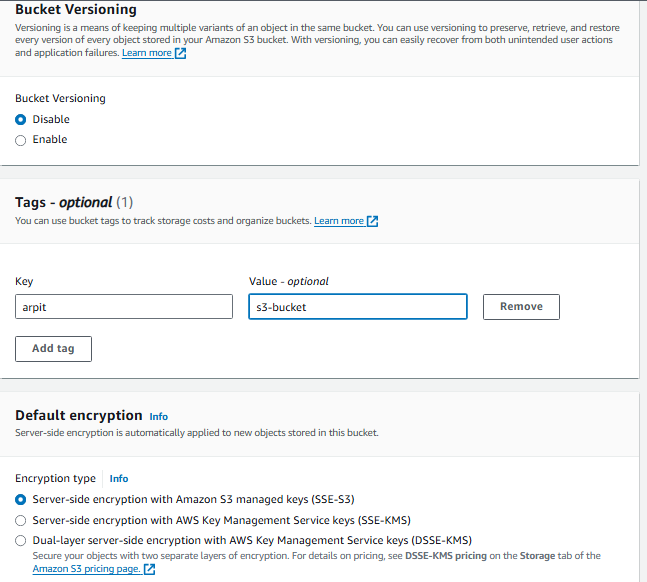
- Go to the S3 dashboard and confirm that only files newer than 30 days remain.

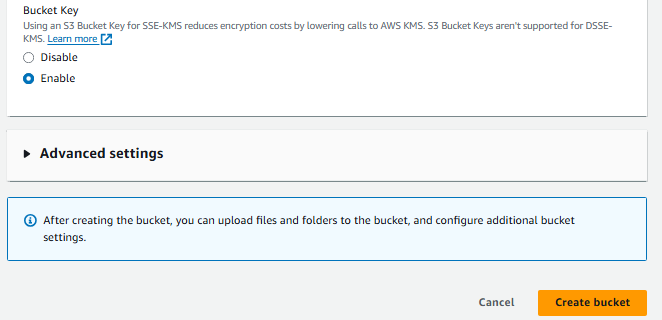
1. S3 Setup:

Create an S3 bucket as below:

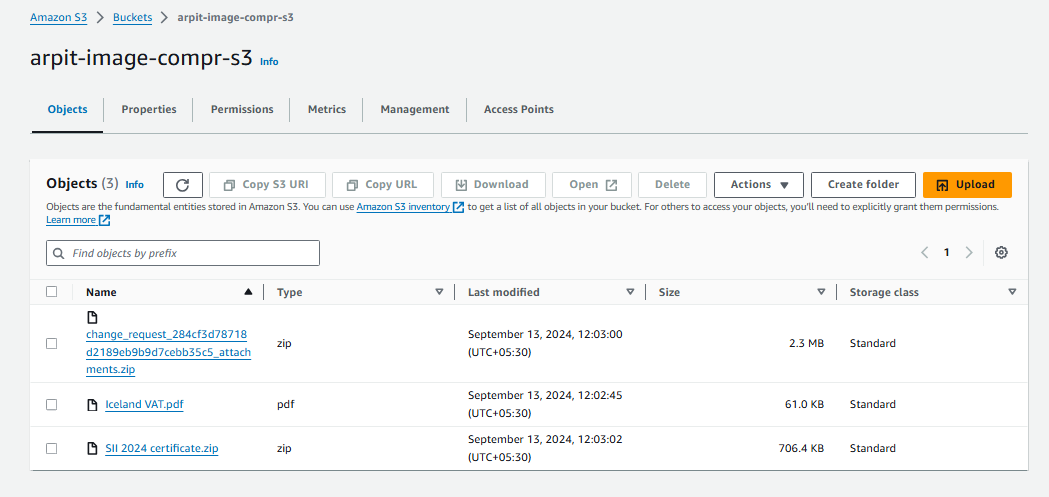






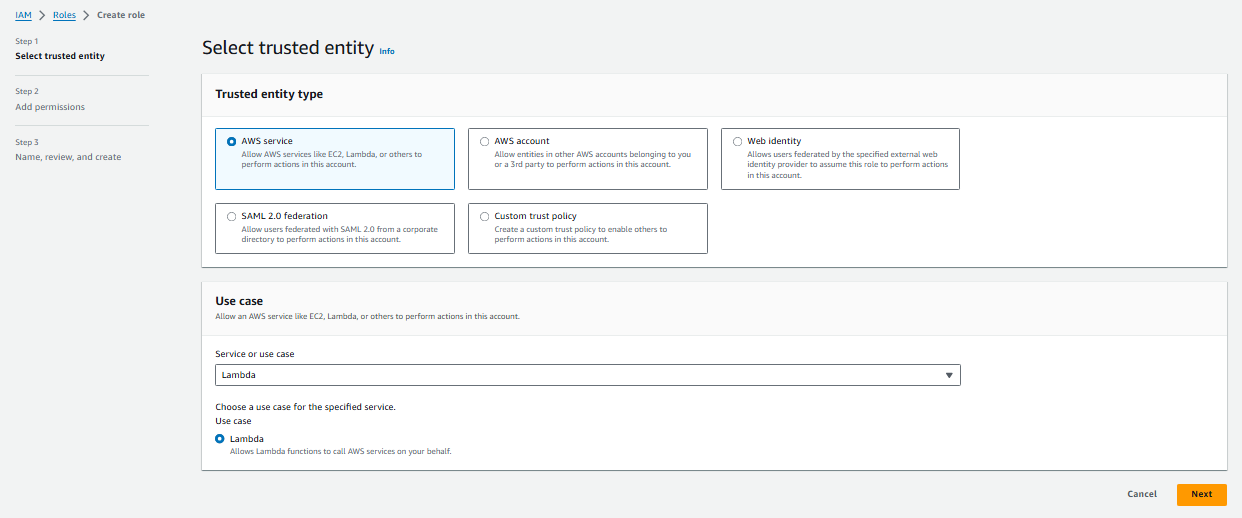


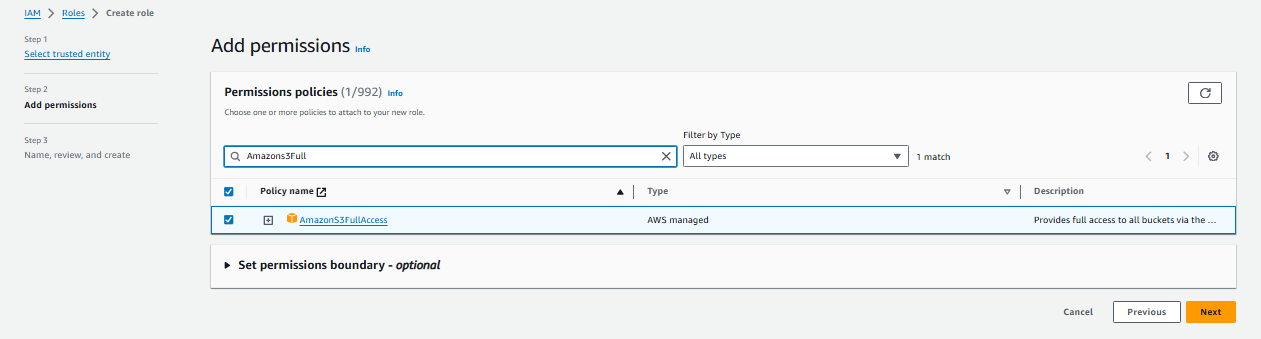
Uploaded some files as below:



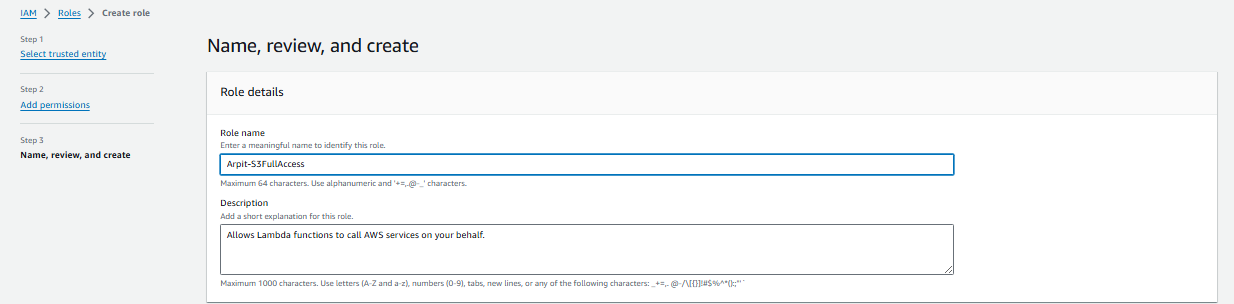
1. Lambda IAM Role:

Create a new IAM role for Lambda function:

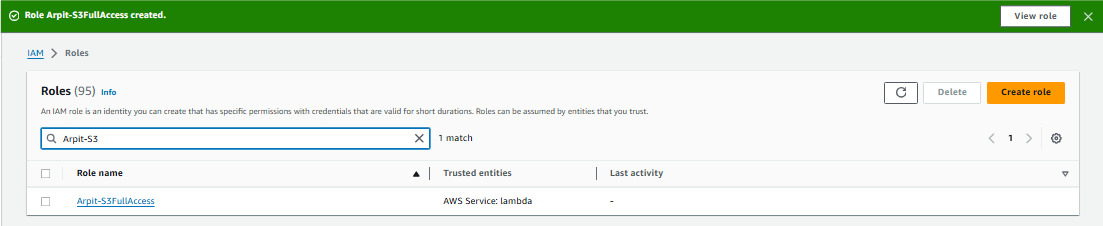


Add permission for AmazonS3Full Access:

Name the Lambda function:



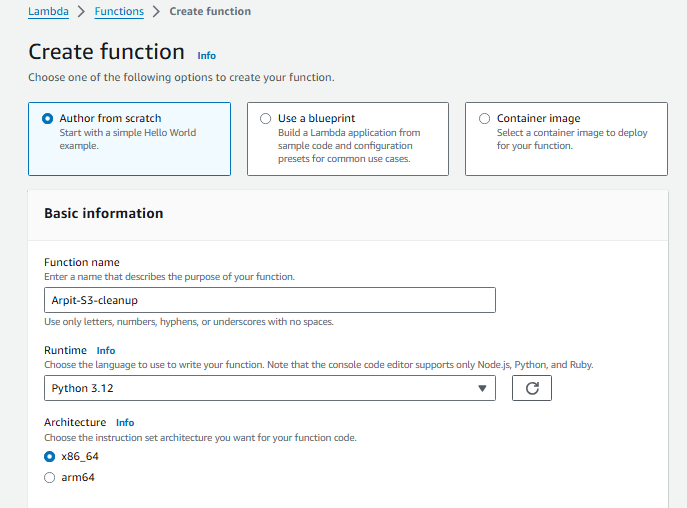
Lambda function has been created successfully:

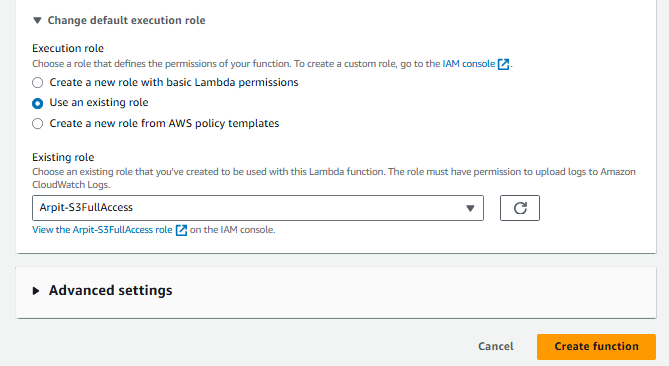


1. Set up an AWS Lambda function:

Creating a Lambda Function for Automated S3 bucket cleanup:

Please remember to set a timeout of at least 1 minute to prevent the function from failing in case of long execution time.





1. **Code: Boto3 Python script Automated S3 Bucket Cleanup Using AWS Lambda**

**import boto3**

**from datetime import datetime, timezone, timedelta**

**def lambda\_handler(event, context):**

**# Initialize the S3 client**

**s3\_client = boto3.client('s3')**

**# Define the S3 bucket**

**bucket\_name = 'arpit-image-compr-s3' # Replace with your bucket name**

**# Set the files deletion cutoff date to 30 days**

**days\_old = 7 # For testing purpose, keeping it to 7 days**

**cutoff\_date = datetime.now(timezone.utc) - timedelta(days=days\_old)**

**# List objects in the bucket**

**objects = s3\_client.list\_objects\_v2(Bucket=bucket\_name)**

**if 'Contents' in objects:**

**for obj in objects['Contents']:**

**object\_key = obj['Key']**

**last\_modified = obj['LastModified']**

**# Delete the files that are older than the cutoff date**

**if last\_modified < cutoff\_date:**

**s3\_client.delete\_object(Bucket=bucket\_name, Key=object\_key)**

**print(f"Deleted {object\_key}, last modified: {last\_modified}")**

**else:**

**print(f"Retained {object\_key}, last modified: {last\_modified}")**

**else:**

**print("Bucket is either empty or no objects found.")**

**return {**

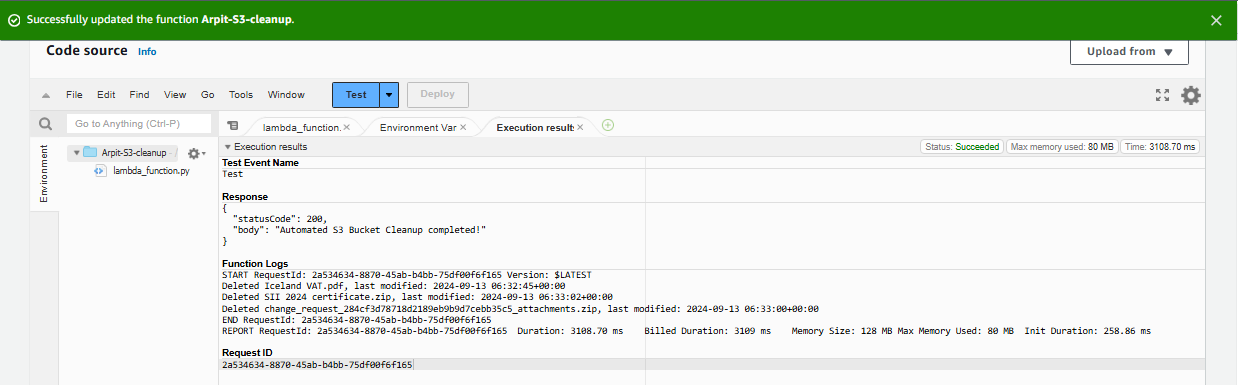
**'statusCode': 200,**

**'body': 'Automated S3 Bucket Cleanup completed!'**

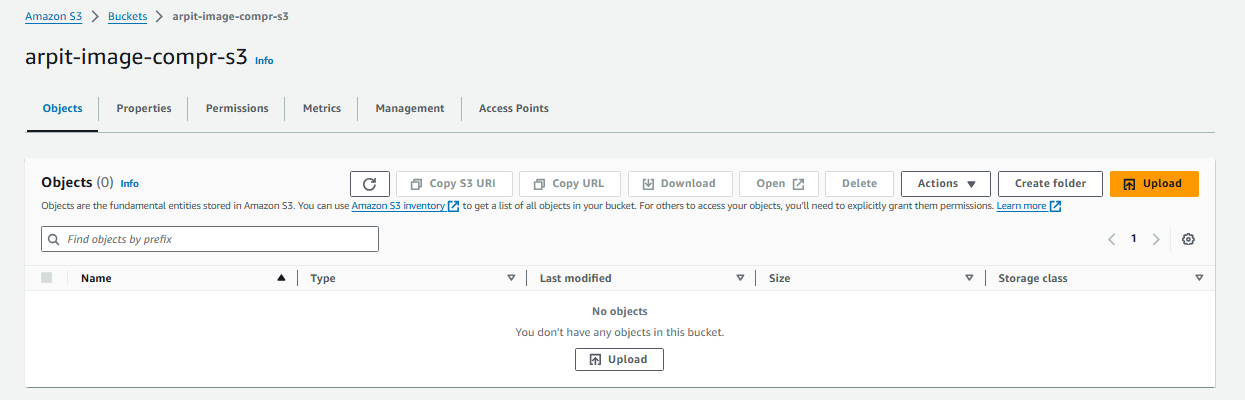
**}**

1. **Testing:**

**Manually executed the lambda function to delete the files older than 30 days:**



**All the files were 8 days older so all of them have been deleted successfully.**



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

**Objective:** To automate the backup process for your EBS volumes and ensure that backups older than a specified retention period are cleaned up to save costs.

**Task:** Automate the creation of snapshots for specified EBS volumes and clean up snapshots older than 30 days.

**Instructions:**

1. EBS Setup:

- Navigate to the EC2 dashboard and identify or create an EBS volume you wish to back up.

- Note down the volume ID.

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.

- Attach policies that allow Lambda to create EBS snapshots and delete them (`AmazonEC2FullAccess` for simplicity, but be more restrictive in real-world scenarios).

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.

- Choose Python 3.x as the runtime.

- Assign the IAM role created in the previous step.

- Write the Boto3 Python script to:

1. Initialize a boto3 EC2 client.

2. Create a snapshot for the specified EBS volume.

3. List snapshots and delete those older than 30 days.

4. Print the IDs of the created and deleted snapshots for logging purposes.

4. Event Source (Bonus):

- Attach an event source, like Amazon CloudWatch Events, to trigger the Lambda function at your desired backup frequency (e.g., every week).

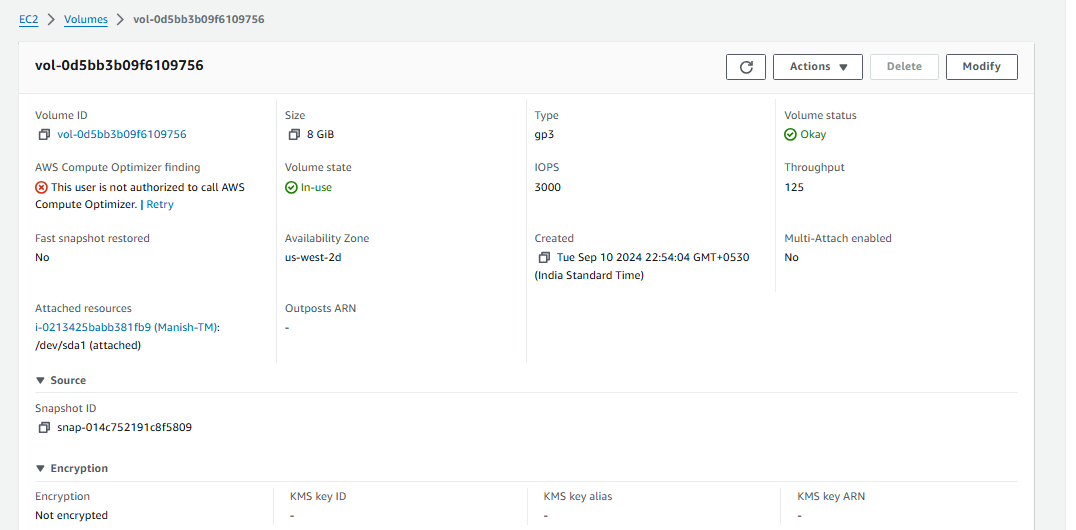
5. Manual Invocation:

- After saving your function, either manually trigger it or wait for the scheduled event.

- Go to the EC2 dashboard and confirm that the snapshot is created and old snapshots are deleted.

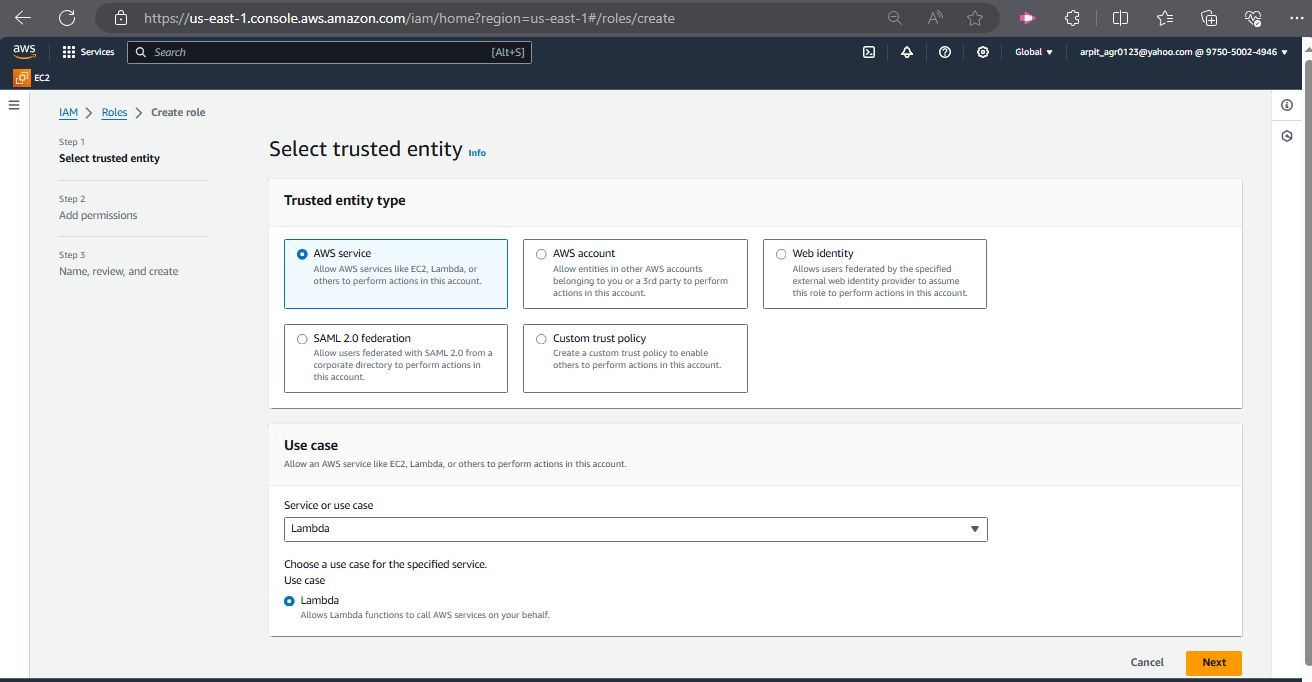
1. EBS Setup:

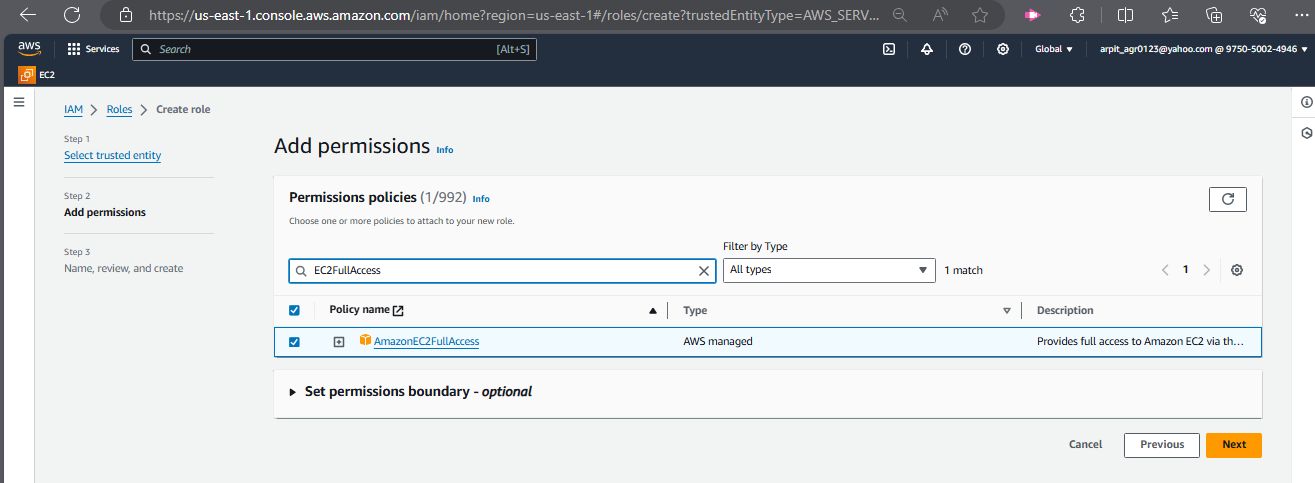
Navigated to the EC2 dashboard and identified an EBS volume we wish to back up.



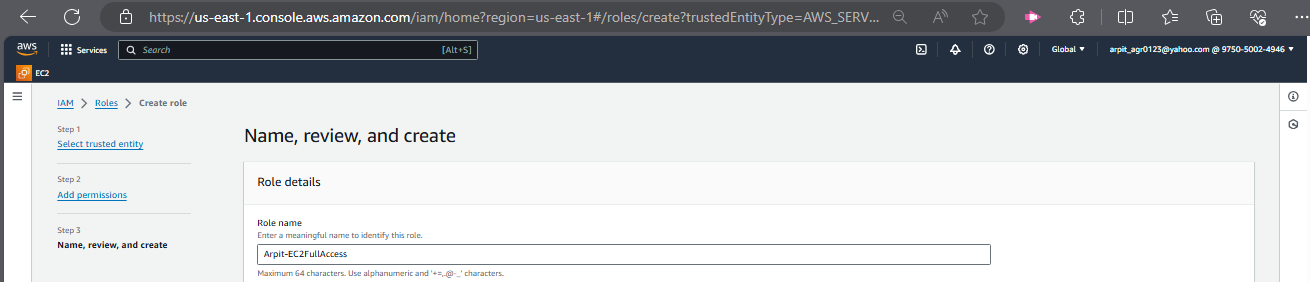
1. Create an IAM role for EC2 permissions:

Creating an IAM role for Lambda function:

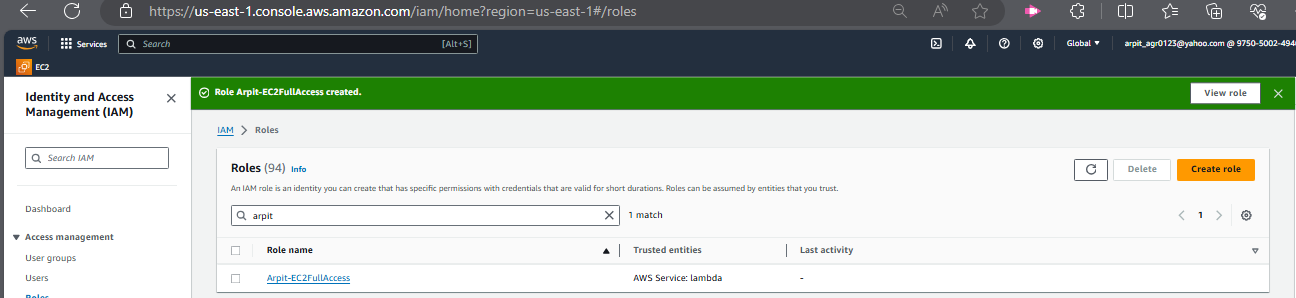


Adding AmazonEC2FullAccess permission:

Naming the role:



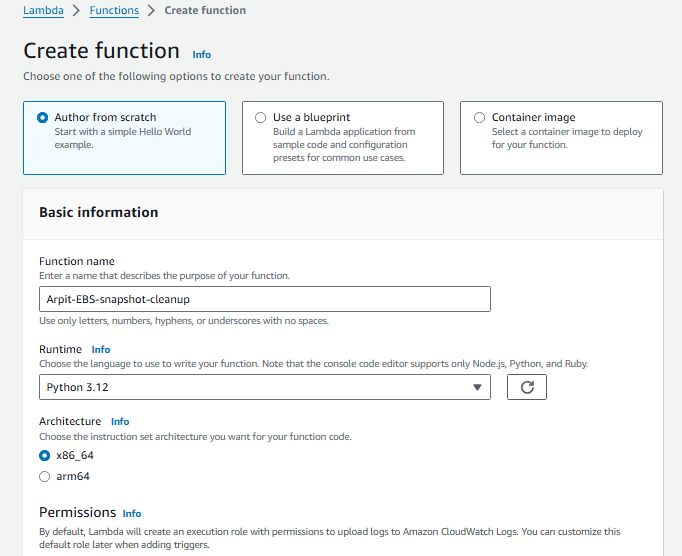
New IAM role has been created successfully:



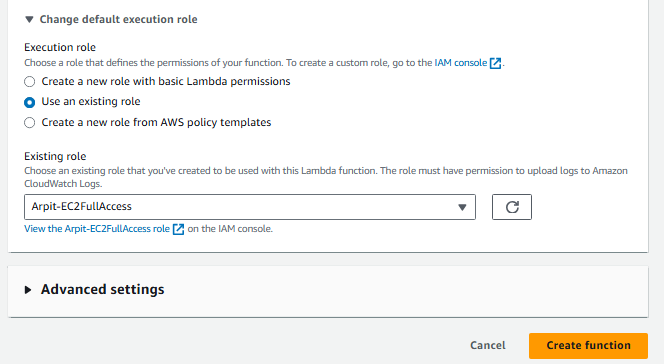
1. Set up an AWS Lambda function:

Creating a Lambda Function for Automatic EBS Snapshot and Cleanup:

Please remember to set a timeout of at least 1 minute to prevent the function from failing in case of long execution time.



Selecting the new IAM role that we just created for the Lambda function:



Code: Boto3 Python script for Automatic EBS Snapshot and Cleanup.

import boto3

import datetime

# Initialize EC2 client

ec2\_client = boto3.client('ec2')

def lambda\_handler(event, context):

# Specify the EBS volume ID to back up

volume\_id = 'vol-0d5bb3b09f6109756'

# Create a snapshot of the specified EBS volume

response = ec2\_client.create\_snapshot(

VolumeId=volume\_id,

Description=f'Snapshot of volume {volume\_id} on {datetime.date.today()}'

)

snapshot\_id = response['SnapshotId']

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

# Set the retention period to 7 days

retention\_days = 7

deletion\_date = datetime.datetime.now() - datetime.timedelta(days=retention\_days)

# Find and delete snapshots older than the retention period

snapshots = ec2\_client.describe\_snapshots(

Filters=[{'Name': 'volume-id', 'Values': [volume\_id]}],

OwnerIds=['self'] # Owned by the current account

)

for snapshot in snapshots['Snapshots']:

snapshot\_date = snapshot['StartTime'].replace(tzinfo=None)

if snapshot\_date < deletion\_date:

snapshot\_id = snapshot['SnapshotId']

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

print(f'Deleted old snapshot: {snapshot\_id}')

return {

'statusCode': 200,

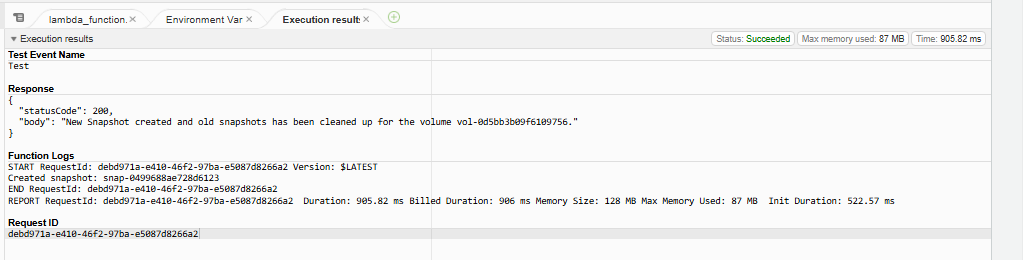
'body': f'New Snapshot created and old snapshots has been cleaned up for the volume {volume\_id}.'

}

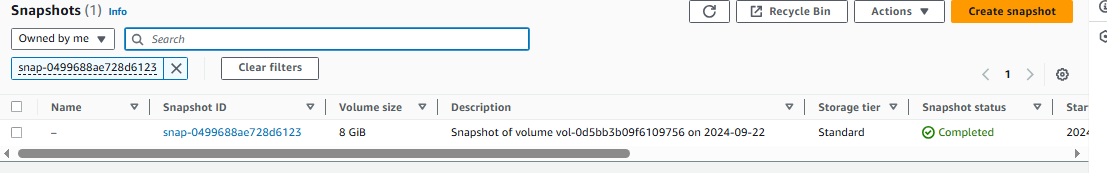
1. Testing:

Manually invoked the Lambda function.

New snapshot snap-0499688ae728d6123 has been created as shown below in the execution result:



snap-0499688ae728d6123 is present in the Snapshots list as below:



**Assignment 8: Analyze Sentiment of User Reviews Using AWS Lambda, Boto3, and Amazon Comprehend**

**Objective**: Automatically analyze and categorize the sentiment of user reviews using Amazon Comprehend.

**Task:** Set up a Lambda function to receive user reviews, analyze their sentiment using Amazon Comprehend, and log the results.

**Instructions:**

1. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.

- Attach policies that allow Lambda to use Amazon Comprehend.

2. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.

- Choose Python 3.x as the runtime.

- Assign the IAM role created previously.

- Write the Boto3 Python script to:

1. Extract the user review from an event.

2. Use Amazon Comprehend to analyze the sentiment of the review.

3. Log the sentiment result.

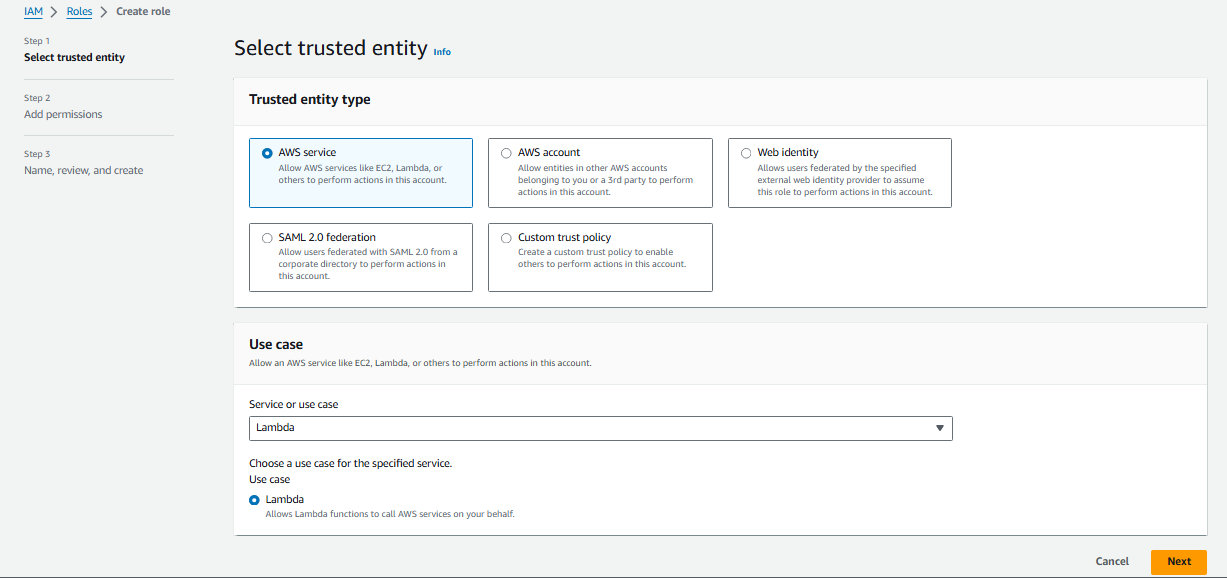
3. Testing:

- Manually trigger the Lambda function with sample reviews.

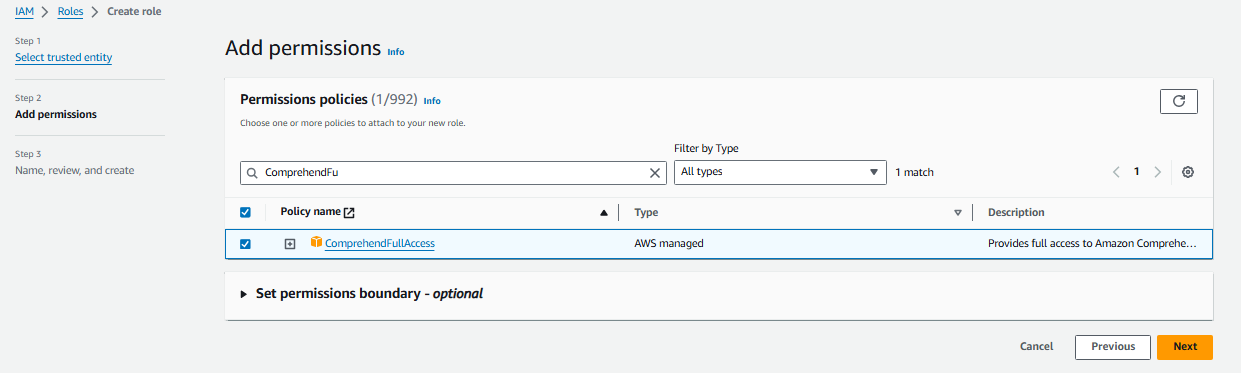
- Confirm the sentiment analysis results in the Lambda logs.

1. Create an IAM role for EC2 permissions:

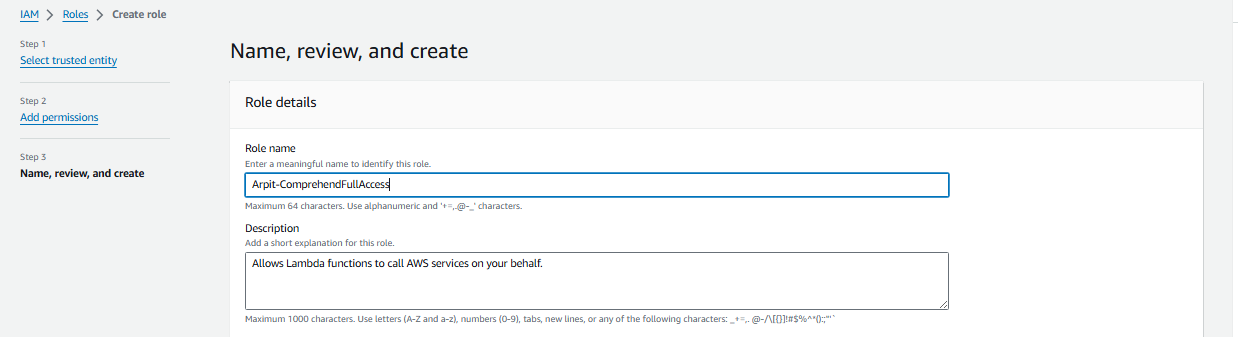
Creating an IAM role for Lambda function:



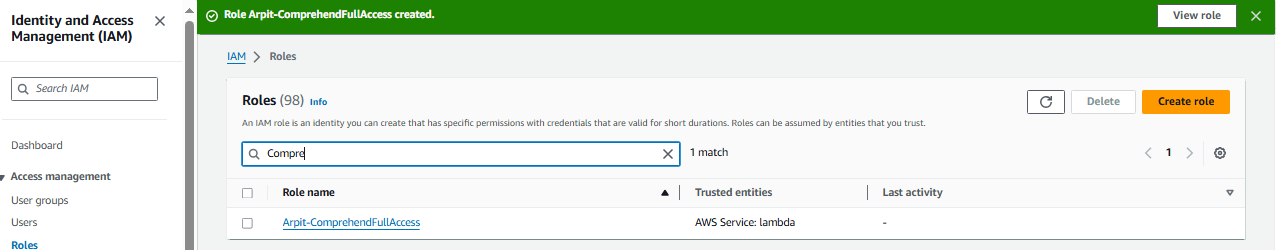
Adding ComprehendFullAccess permission:



Naming the role:



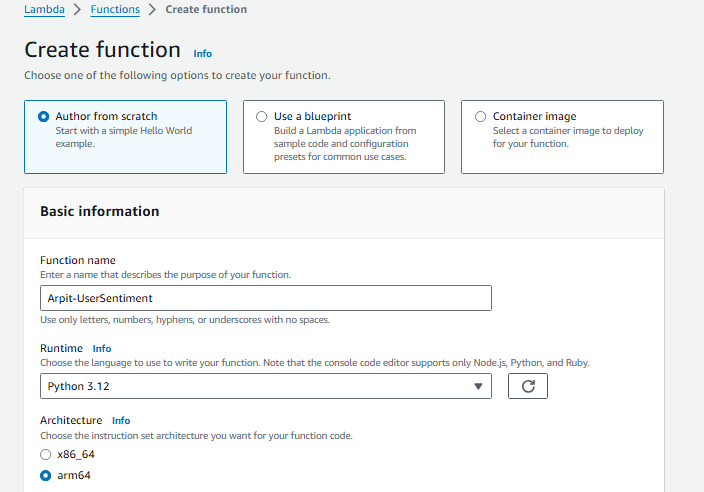
New IAM role has been created successfully:



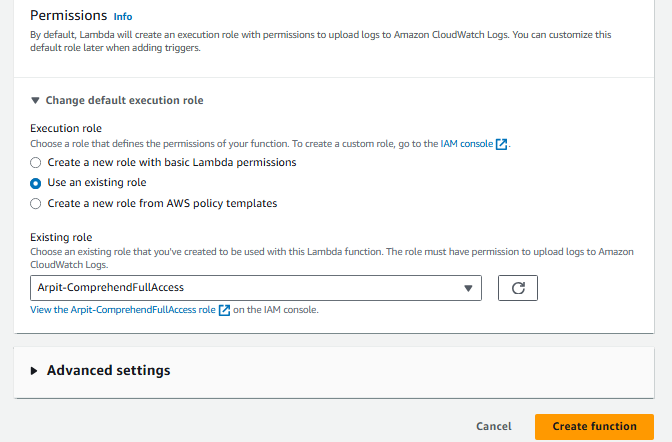
1. Set up an AWS Lambda function:

Creating a Lambda Function for Analyze Sentiment of User Reviews:

Please remember to set a timeout of at least 1 minute to prevent the function from failing in case of long execution time.



Selecting the new IAM role that we just created for the Lambda function:



Code: Boto3 Python script for Automatic EBS Snapshot and Cleanup.

import json

import boto3

def lambda\_handler(event, context):

try:

# Extract the user review from the event

review = event.get('review', '')

if not review:

raise ValueError("No review provided in the event")

# Initialize the Amazon Comprehend client

comprehend = boto3.client('comprehend')

# Analyze the sentiment of the review

response = comprehend.detect\_sentiment(Text=review, LanguageCode='en')

# Extract the sentiment result

sentiment = response.get('Sentiment', 'UNKNOWN')

# Log the sentiment result

print(f"Review: {review}")

print(f"Sentiment: {sentiment}")

return {

'statusCode': 200,

'body': json.dumps({

'review': review,

'sentiment': sentiment

})

}

except Exception as e:

return {

'statusCode': 500,

'body': json.dumps({

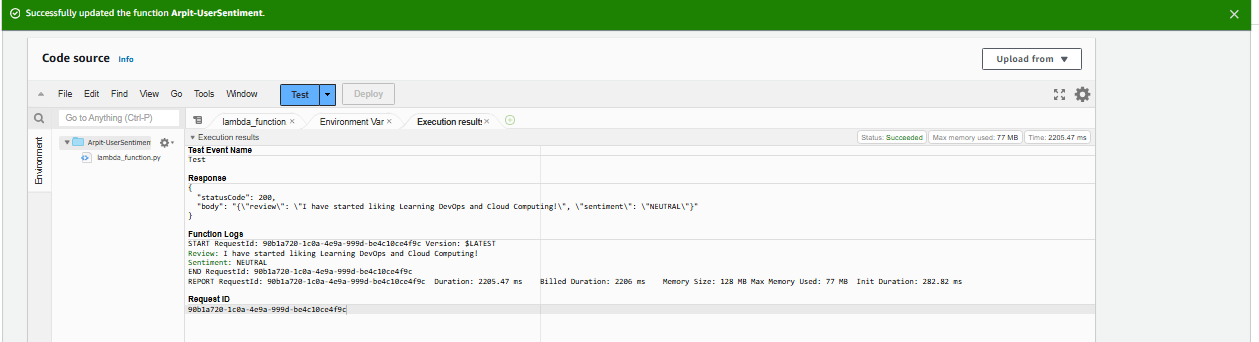
'error': str(e)

})

}

1. Testing:

Click Test to trigger the function.



Cloudwatch Logs:

