**Assignment On Serverless Architecture**

**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.

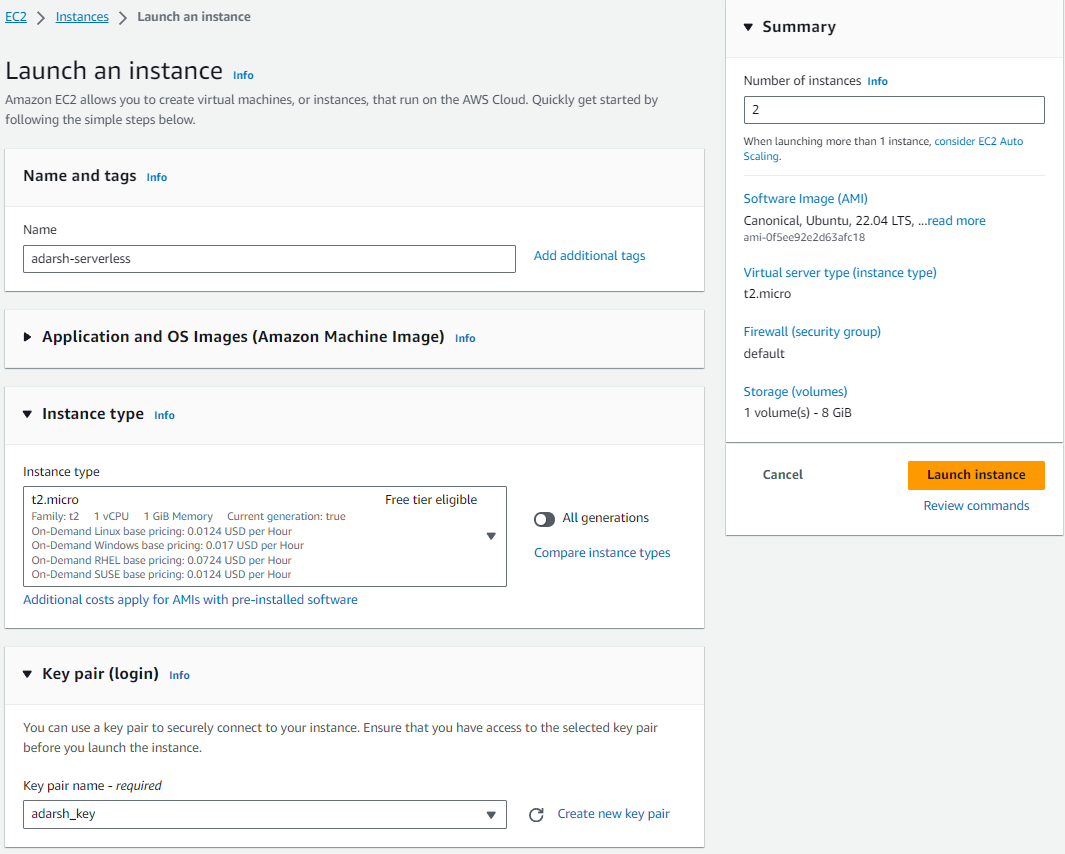
**Task:** You're tasked to automate the stopping and starting of EC2 instances based on tags.

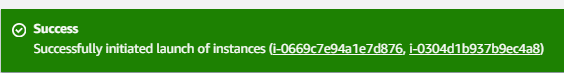
1. Setup:

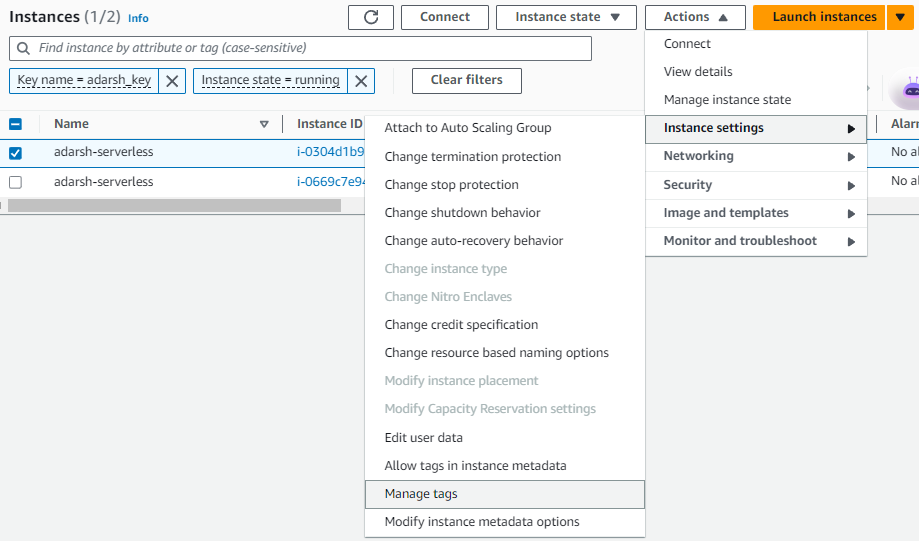
* Create two EC2 instances.
* Tag one of them as `Auto-Stop` and the other as `Auto-Start`.

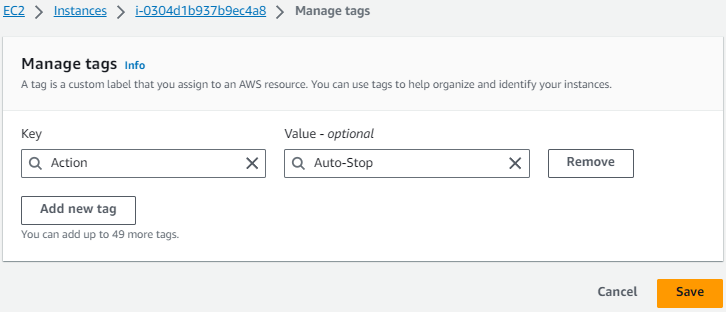
**Instructions**:

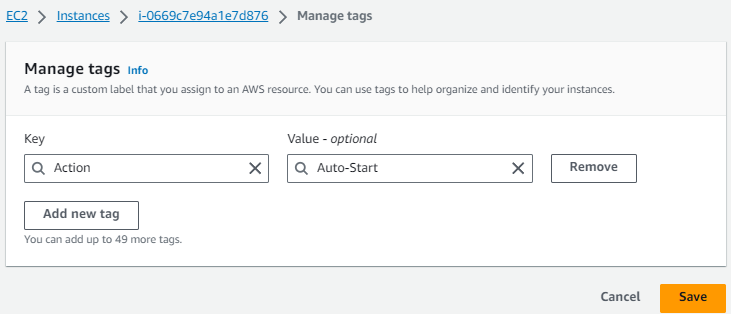
* Navigate to the EC2 dashboard and create two new t2.micro instances (or any other available free-tier type).
* Tag the first instance with a key `Action` and value `Auto-Stop`.
* Tag the second instance with a key `Action` and value `Auto-Start`.











2. Lambda Function Creation:

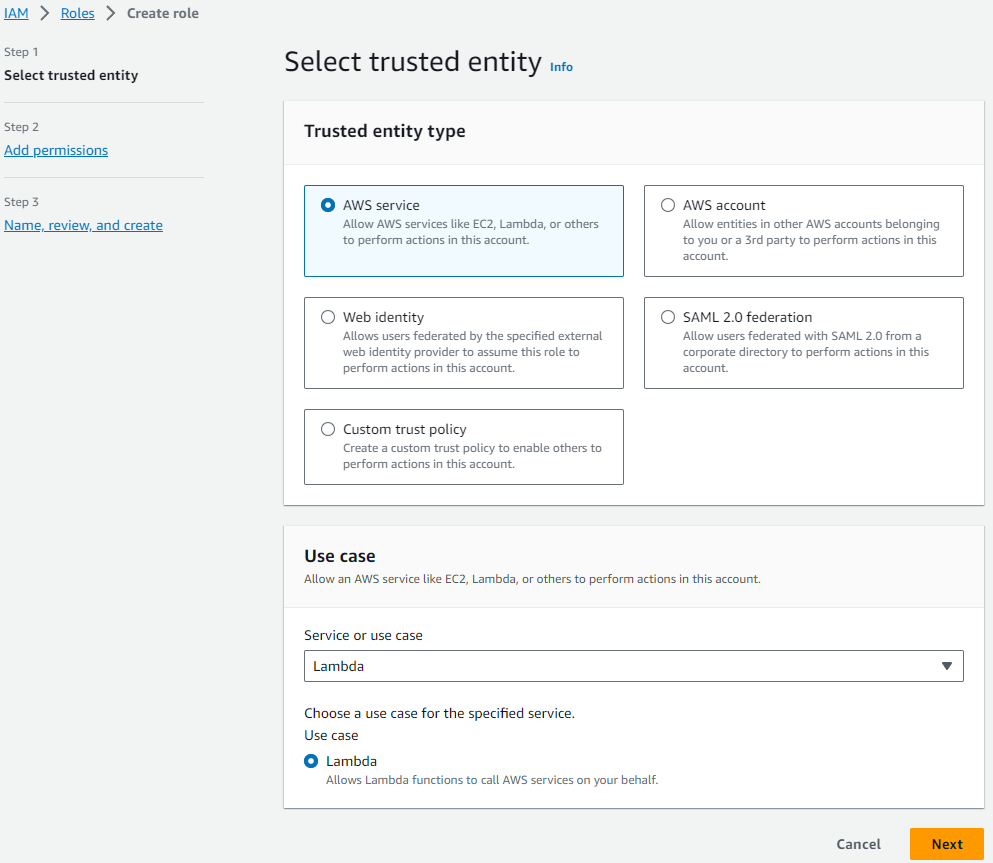
* Set up an AWS Lambda function
* Ensure that the Lambda function has the necessary IAM permissions to describe, stop, and start EC2 instances.

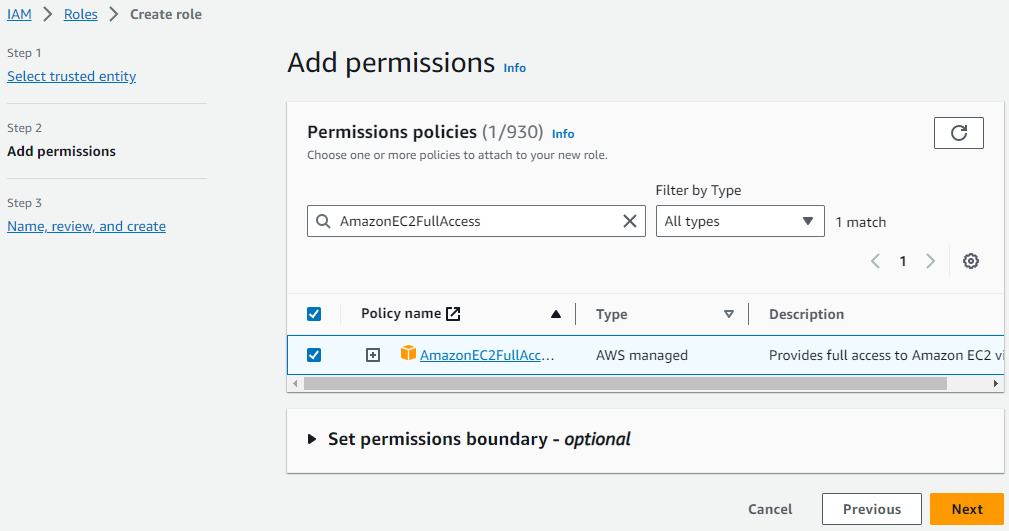
**Instructions**

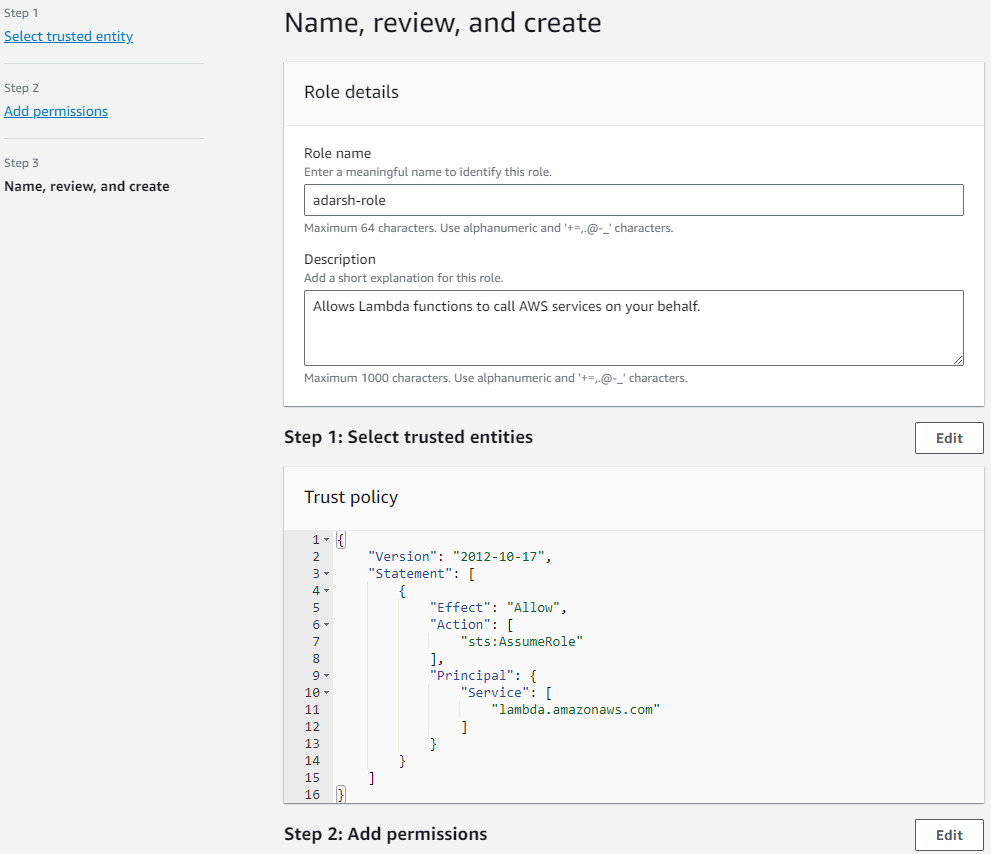
2. Lambda IAM Role:

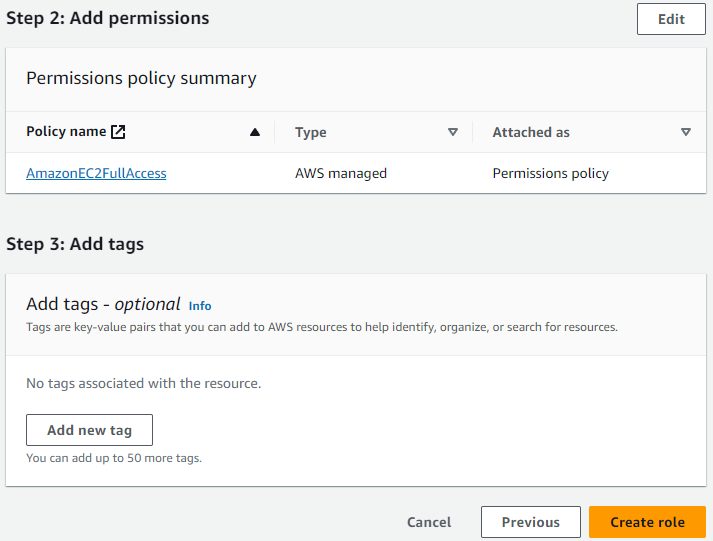
- In the IAM dashboard, create a new role for Lambda.

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









3. Coding:

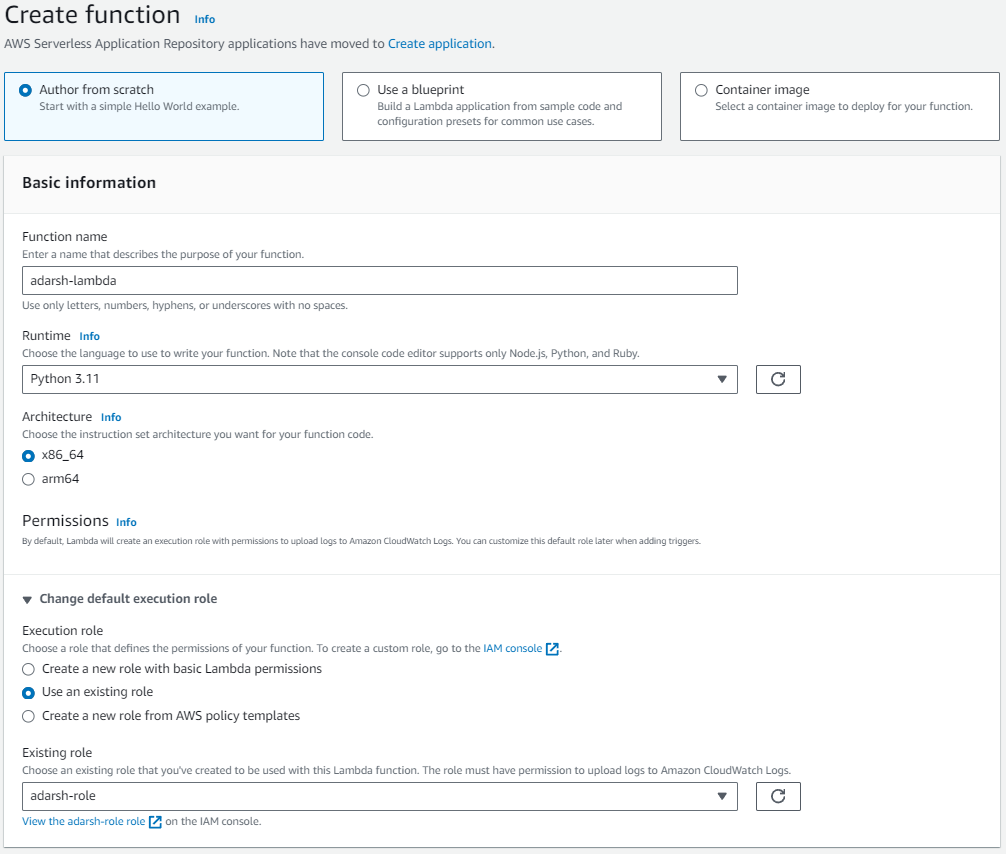
* Using Boto3 in the Lambda function:
* Detect all EC2 instances with the `Auto-Stop` tag and stop them.
* Detect all EC2 instances with the `Auto-Start` tag and start them.

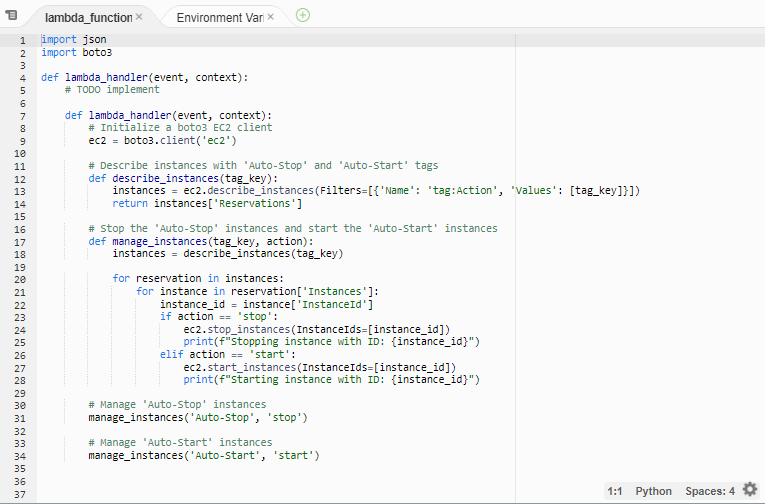
**Instructions**

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.





#-------------------------------------------------Lambda Function------------------------------------------------------------

import json

import boto3

def lambda\_handler(event, context):

ec2 = boto3.client('ec2')

# Describe instances with 'Auto-Stop' and 'Auto-Start' tags

def describe\_instances(tag\_key):

instances = ec2.describe\_instances(Filters=[{'Name': 'tag:Action', 'Values': [tag\_key]}])

return instances['Reservations']

def manage\_instances(tag\_key, action):

instances = describe\_instances(tag\_key)

for reservation in instances:

for instance in reservation['Instances']:

instance\_id = instance['InstanceId']

if action == 'stop':

ec2.stop\_instances(InstanceIds=[instance\_id])

print(f"Stopping instance with ID: {instance\_id}")

elif action == 'start':

ec2.start\_instances(InstanceIds=[instance\_id])

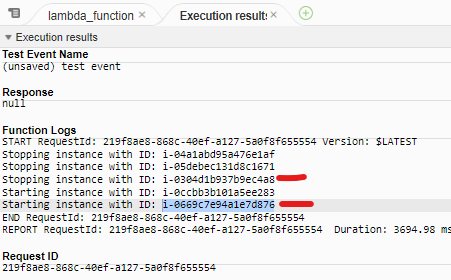
print(f"Starting instance with ID: {instance\_id}")

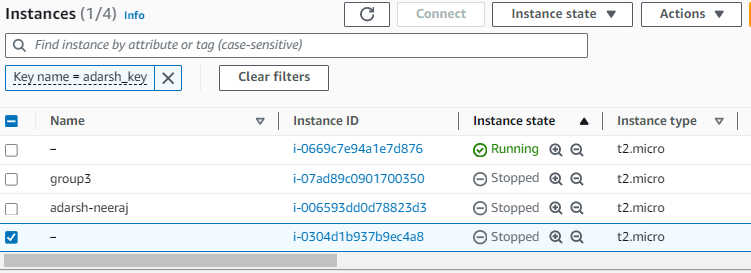
manage\_instances('Auto-Stop', 'stop')

#---------------------------------------------------------------------------------------------------------------------------------

4. Testing:

* Manually invoke the Lambda function.
* Confirm that the instance tagged `Auto-Stop` stops and the one tagged `Auto-Start` starts.





**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.

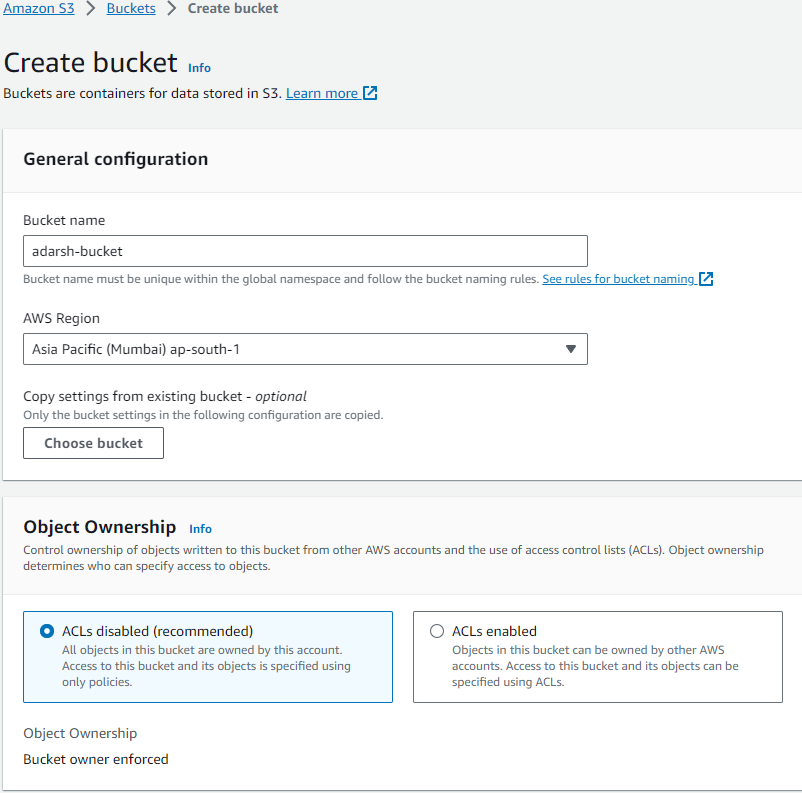
**Task:** Automate the deletion of files older than 30 days in a specific S3 bucket.

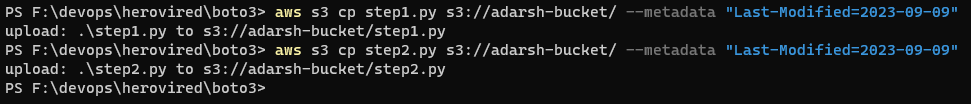
**Instructions:**

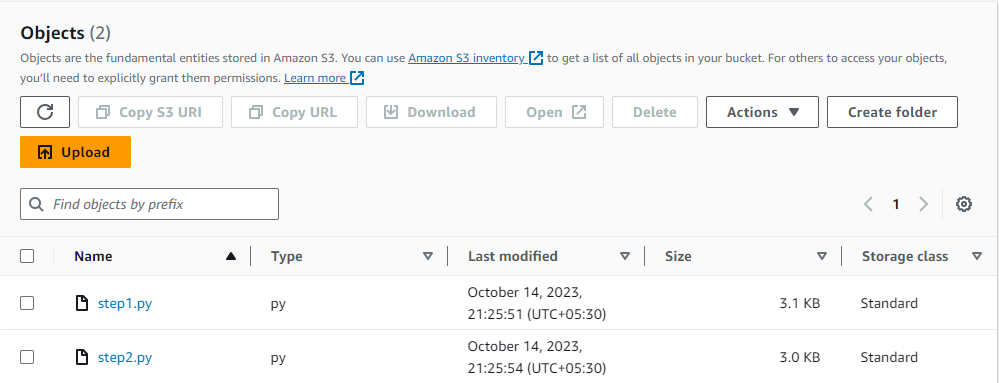
1. S3 Setup:

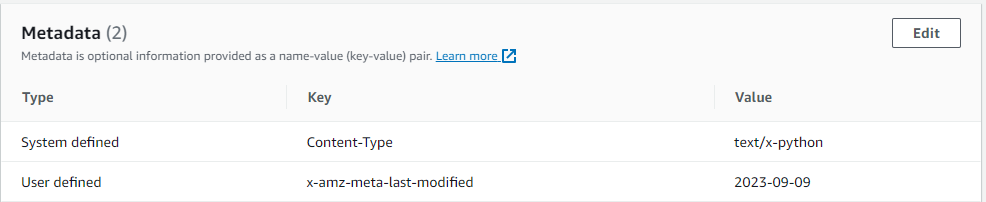
- Navigate to the S3 dashboard and create a new bucket.

- 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).





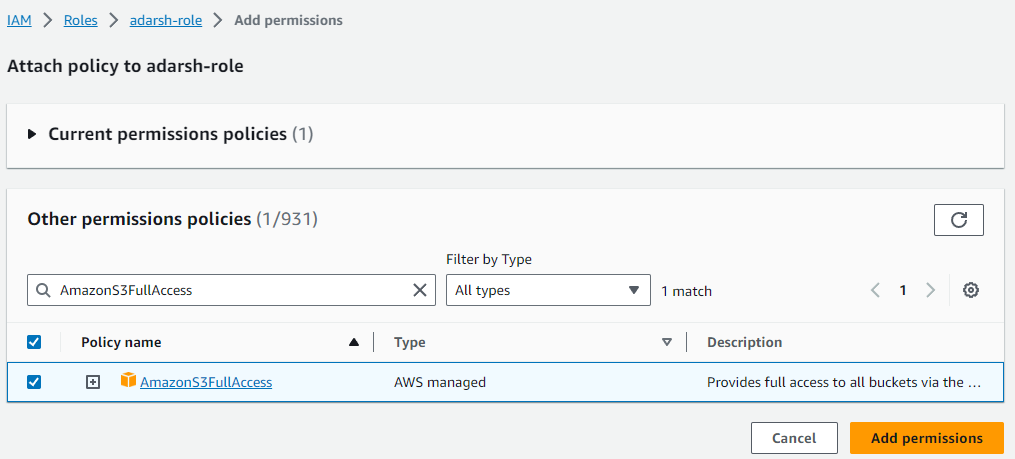


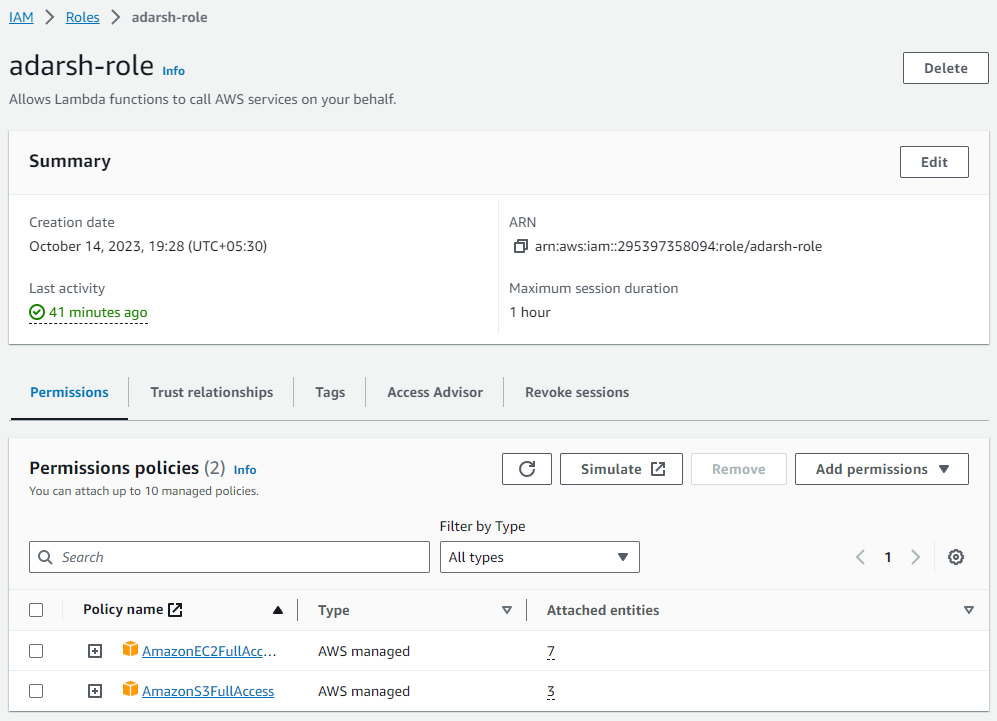


2. Lambda IAM Role:

- 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.

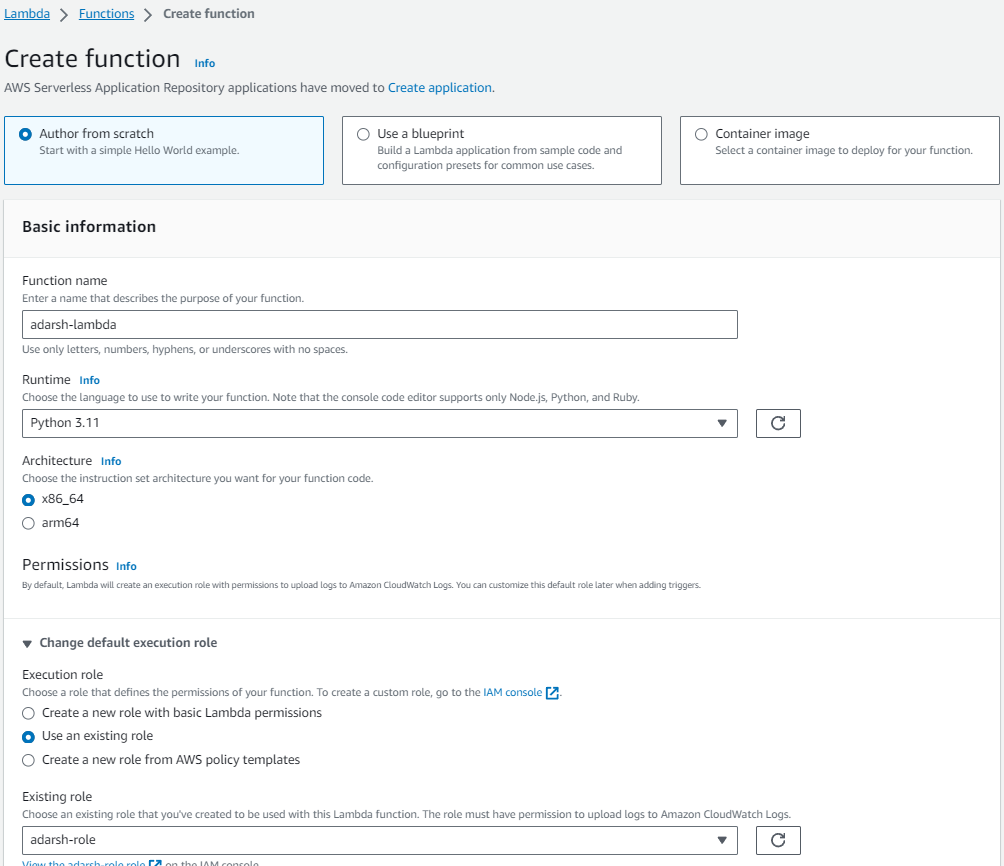
- Write the Boto3 Python script to:

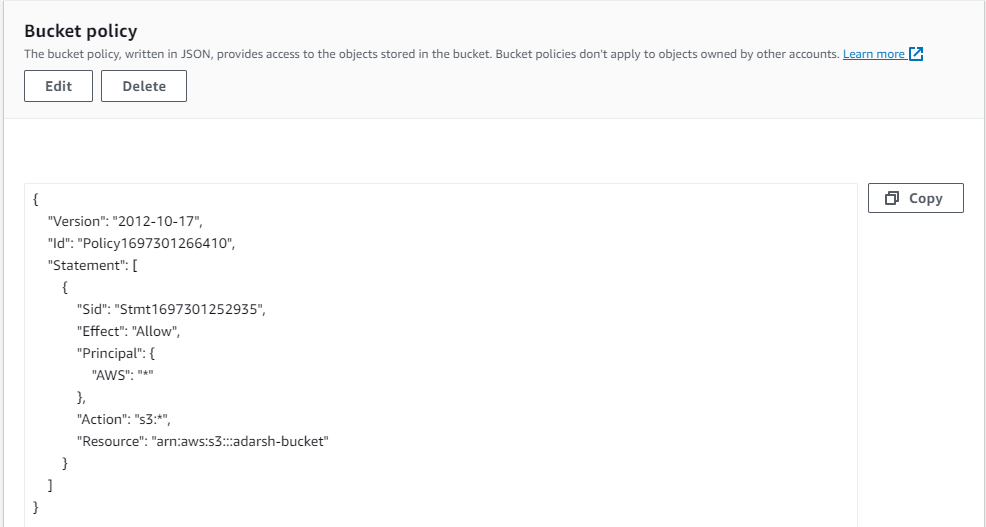
1. Initialize a boto3 S3 client.

2. List objects in the specified bucket.

3. Delete objects older than 30 days.

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





#----------------------------------------------Lambda Function--------------------------------------------------------

import json

import boto3

from datetime import datetime, timedelta

def lambda\_handler(event, context):

s3 = boto3.client('s3')

# Get the current date

current\_date = datetime.now()

# Calculate the date 30 days ago

thirty\_days\_ago = current\_date - timedelta(days=30)

objects = s3.list\_objects(Bucket='adarsh-bucket')

object\_list = [obj['Key'] for obj in objects['Contents']] # print(object\_list)

if 'Contents' in objects:

for obj in objects['Contents']:

object\_key = obj['Key']

# Get object metadata tags

metadata = s3.head\_object(Bucket='adarsh-bucket', Key=object\_key)

metadata\_tags = metadata.get('Metadata', {})

# print(f"Object Key: {object\_key}")

if 'last-modified' in metadata\_tags:

last\_modified\_date = datetime.strptime(metadata\_tags['last-modified'], '%Y-%m-%d')

# Check if the tag value indicates a date older than 30 days

if last\_modified\_date < thirty\_days\_ago:

# Delete the object

s3.delete\_object(Bucket='adarsh-bucket', Key=object\_key)

print(f"Deleted object: {object\_key}")

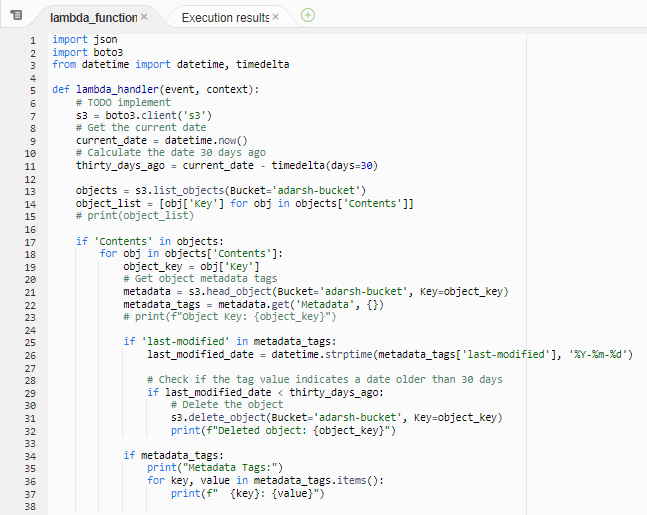
if metadata\_tags:

print("Metadata Tags:")

for key, value in metadata\_tags.items():

print(f" {key}: {value}")

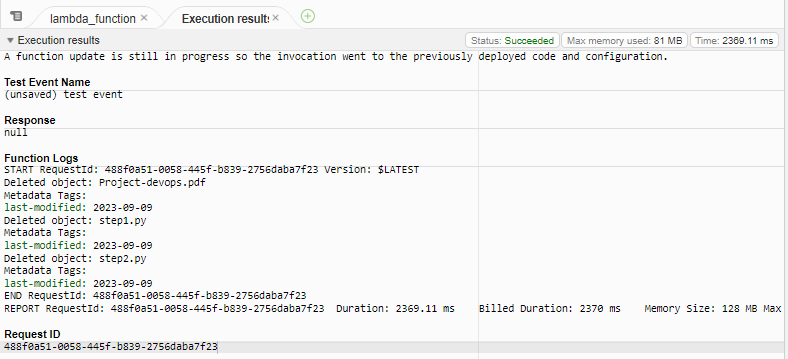
#---------------------------------------------------------------------------------------------------------------------------

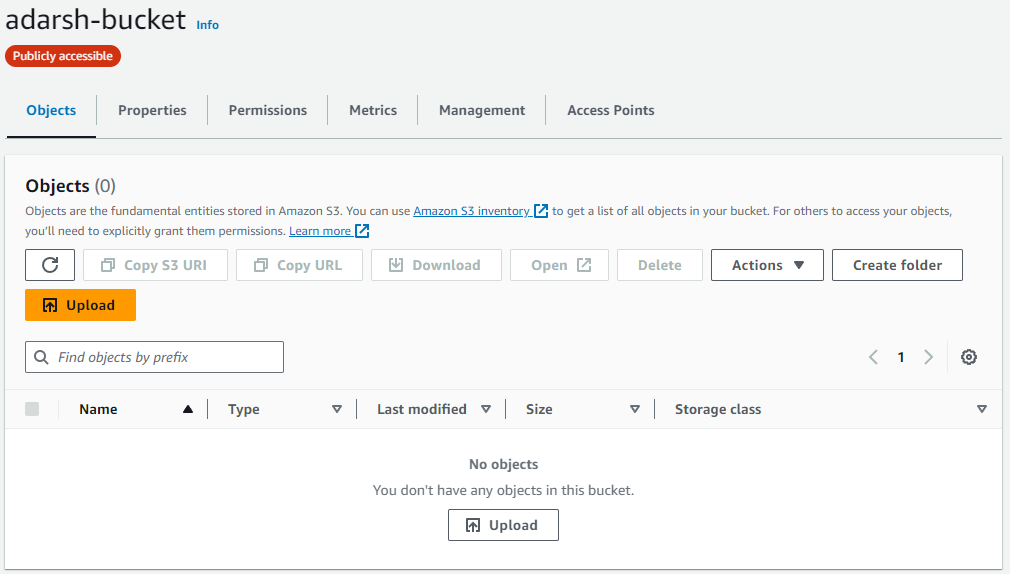


4. Manual Invocation:

- After saving your function, manually trigger it.

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





**Assignment 3: 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.

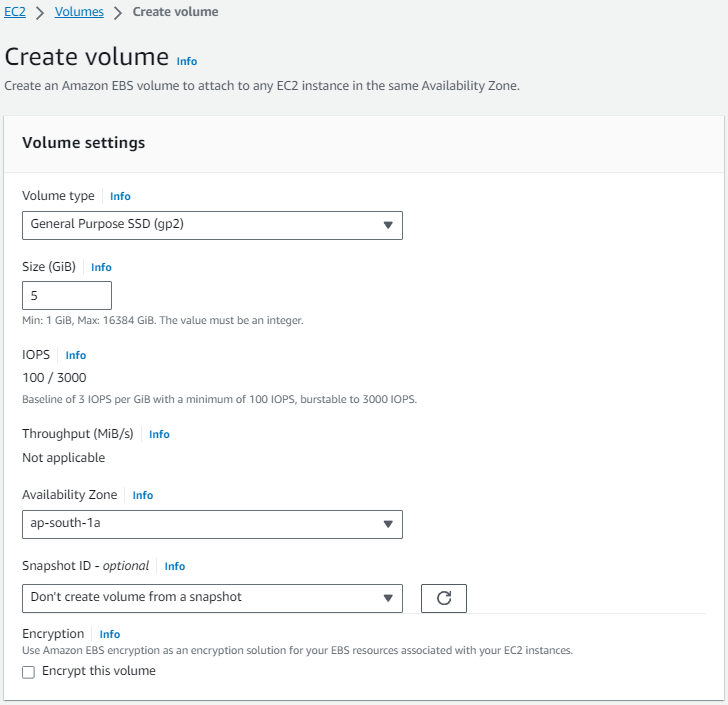
**Task:** Automate the creation of snapshots for specified EBS volumes and clean up snapshots older than 30 days.

**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.



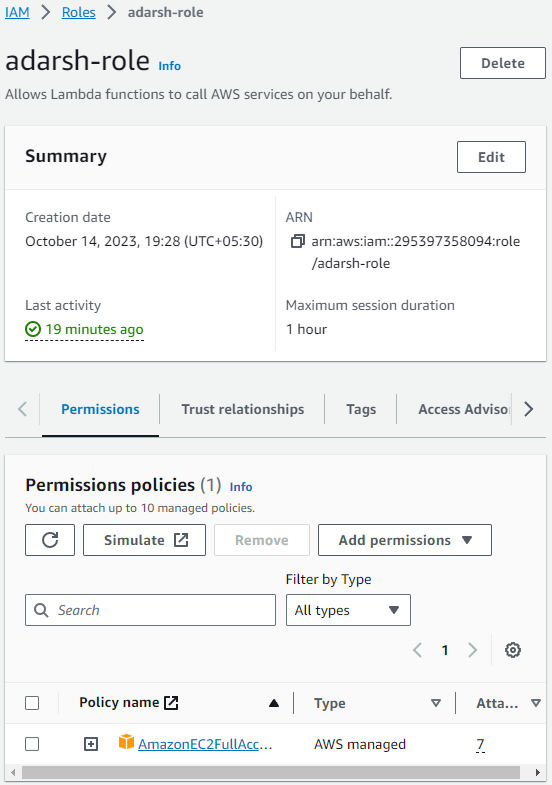


vol-0bb16ec4917256c7c

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.

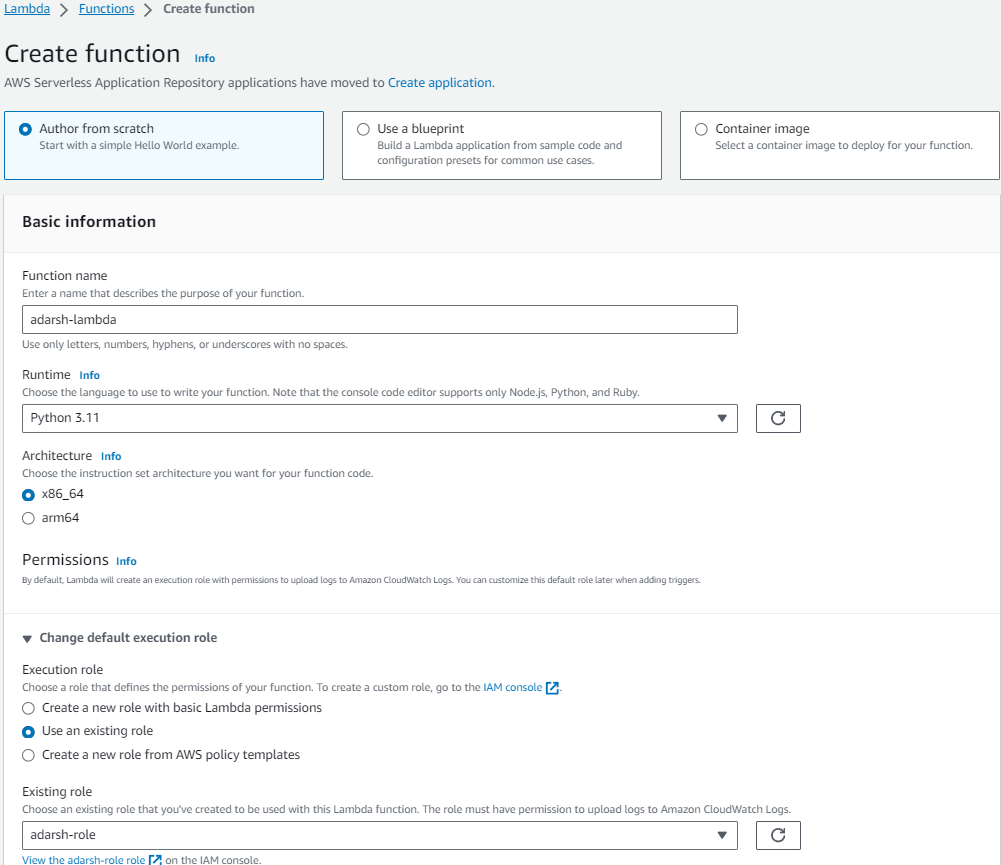
- Write the Boto3 Python script to:

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.





#-----------------------------------------Lambda Function-------------------------------------------------------------------

import json

import boto3

from datetime import datetime, timedelta, timezone

def lambda\_handler(event, context):

# Initialize a Boto3 EC2 client

ec2 = boto3.client('ec2')

# Specify the EBS volume ID for which you want to create a snapshot

volume\_id = 'vol-0bb16ec4917256c7c'

# Create a snapshot for the specified EBS volume

snapshot = ec2.create\_snapshot(

VolumeId=volume\_id,

Description=f'Snapshot for EBS volume {volume\_id}'

)

# Print the snapshot ID for logging purposes

snapshot\_id = snapshot['SnapshotId']

print(f"Snapshot ID: {snapshot\_id}")

# List snapshots for the specified volume

snapshots = ec2.describe\_snapshots(Filters=[{'Name': 'volume-id', 'Values': [volume\_id]}])

# Calculate the date 30 days ago

thirty\_days\_ago = datetime.now(timezone.utc) - timedelta(days=30)

# Iterate through the snapshots

for snapshot in snapshots['Snapshots']:

snapshot\_id = snapshot['SnapshotId']

snapshot\_start\_time = snapshot['StartTime']

# Check if the snapshot is older than 30 days

if snapshot\_start\_time < thirty\_days\_ago:

# Delete the snapshot

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

print(f"Deleted Snapshot ID: {snapshot\_id}")

# Return the IDs of the created and deleted snapshots

return {

'statusCode': 200,

'body': {

'created\_snapshot\_id': snapshot\_id,

'deleted\_snapshot\_ids': [snapshot['SnapshotId'] for snapshot in snapshots['Snapshots'] if snapshot['StartTime'] < thirty\_days\_ago]

}

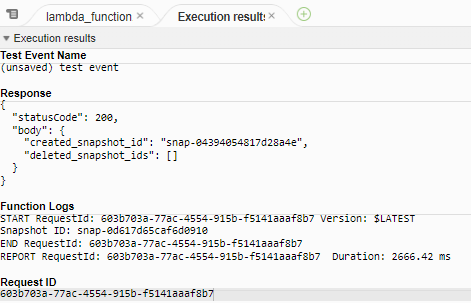
}

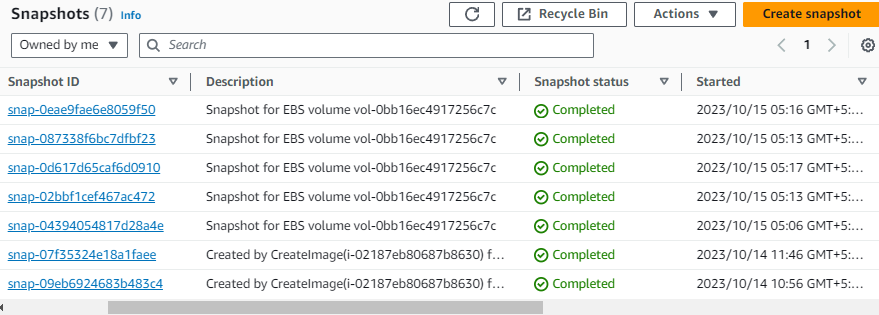
#-------------------------------------------------------------------------------------------------------------------

4. 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.



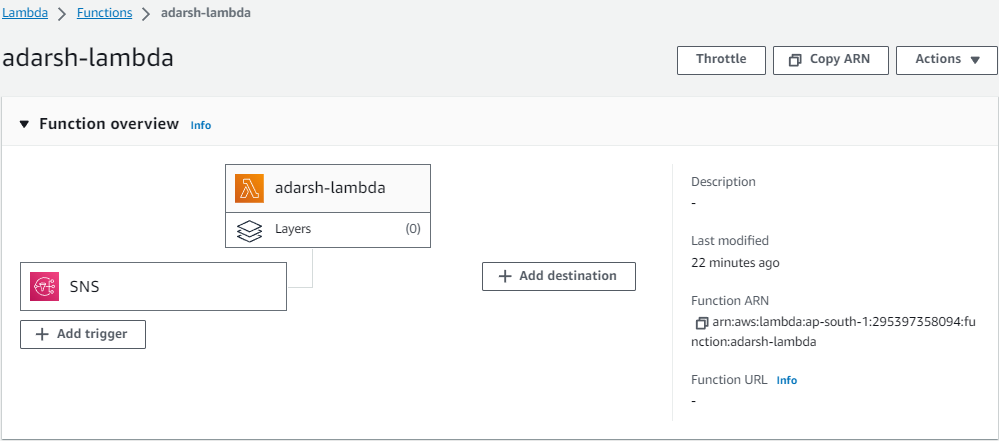


**Assignment 4: Load Balancer Health Checker**

**Objective:** Design a Lambda function that checks the health of registered instances behind an Elastic Load Balancer (ELB) and notifies via SNS if any instances are unhealthy.

**Instructions:**

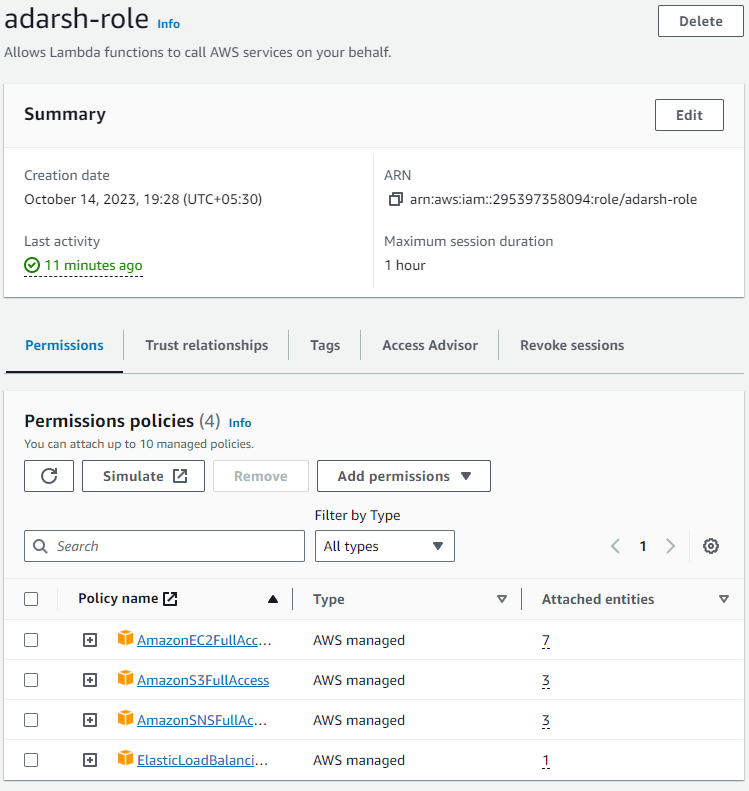
1. Create a Lambda function.

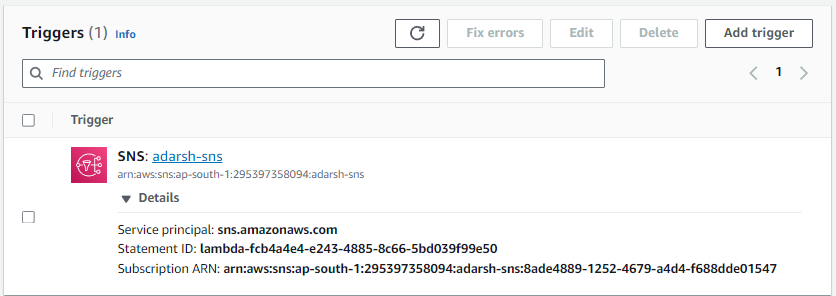


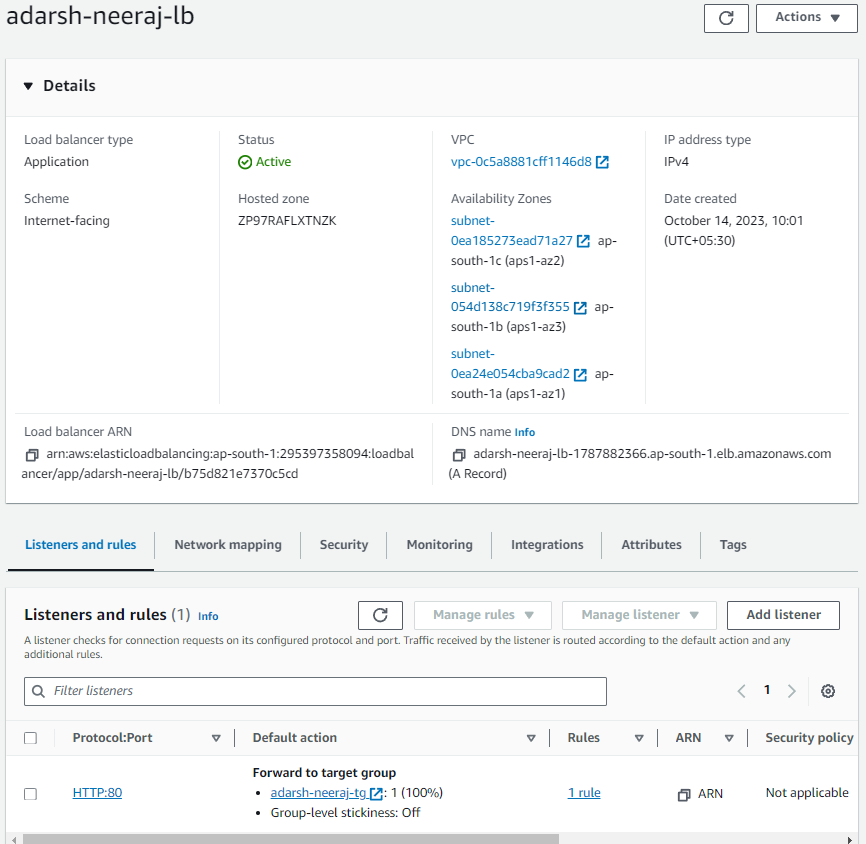
2. With Boto3, configure the function to:

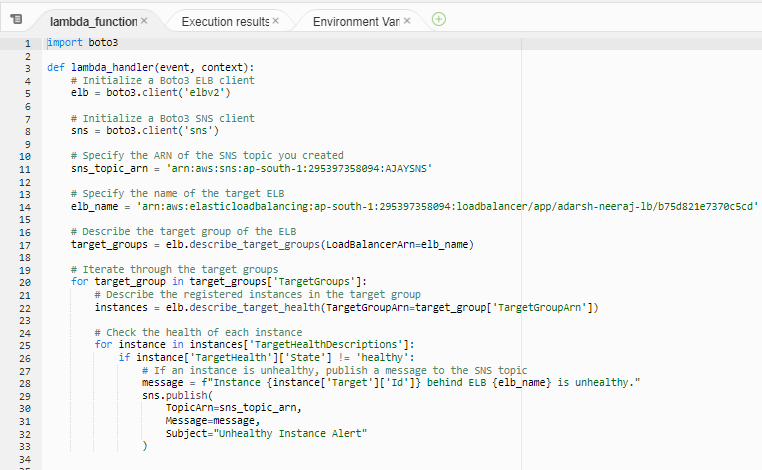
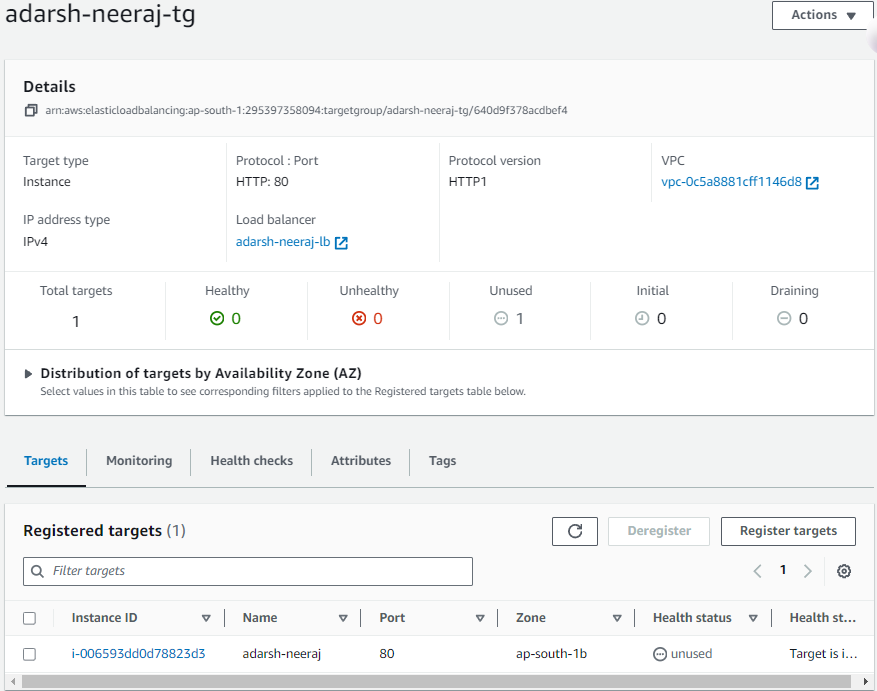
1. Check the health of registered instances behind a given ELB.

2. If any instances are found to be unhealthy, publish a detailed message to an SNS topic.









#----------------------------------------------Lambda Function------------------------------------------------------------

import boto3

def lambda\_handler(event, context):

# Initialize a Boto3 ELB client

elb = boto3.client('elbv2')

# Initialize a Boto3 SNS client

sns = boto3.client('sns')

# Specify the ARN of the SNS topic you created

sns\_topic\_arn = 'arn:aws:sns:ap-south-1:295397358094:AJAYSNS'

# Specify the name of the target ELB

elb\_name = 'arn:aws:elasticloadbalancing:ap-south-1:295397358094:loadbalancer/app/adarsh-neeraj-lb/b75d821e7370c5cd'

# Describe the target group of the ELB

target\_groups = elb.describe\_target\_groups(LoadBalancerArn=elb\_name)

# Iterate through the target groups

for target\_group in target\_groups['TargetGroups']:

# Describe the registered instances in the target group

instances = elb.describe\_target\_health(TargetGroupArn=target\_group['TargetGroupArn'])

# Check the health of each instance

for instance in instances['TargetHealthDescriptions']:

if instance['TargetHealth']['State'] != 'healthy':

# If an instance is unhealthy, publish a message to the SNS topic

message = f"Instance {instance['Target']['Id']} behind ELB {elb\_name} is unhealthy."

sns.publish(

TopicArn=sns\_topic\_arn,

Message=message,

Subject="Unhealthy Instance Alert"

)

#---------------------------------------------------------------------------------------------------------------------------

