AWS Lambda

Created by	SUBBA REDDY SANGHAM
Created time	@November 12, 2024 7:22 PM
Tags	AWS

AWS Lambda:

AWS Lambda is a **serverless compute service** that allows you to run code in response to events without provisioning or managing servers.

It’s designed to execute small, individual functions based on triggers, scaling automatically and billing only for the compute time used.

Why Lambda?

AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources, making it the fastest way to turn an idea into a modern, production, serverless applications.

Benefits of Lambda:

1. **No need for managing servers:** Run code without provisioning or managing infrastructure. Simply write and upload code as a .zip file or container image.
2. **Automatic Scaling:** Automatically respond to code execution requests at any scale, from a dozen events per day to hundreds of thousands per

second.

3. **Pay-as-you-go pricing:** Save costs by paying only for the compute time you use — by the millisecond — instead of provisioning infrastructure upfront for peak capacity.
4. **Performance optimization:** Optimize code execution time and performance with the right function memory size.

What AWS Lambda Actually Is

Lambda is essentially a **function-as-a-service (FaaS)** platform where developers deploy “functions” (small units of code) that run in response to events. These events can be HTTP requests, file uploads, database updates, scheduled tasks, and more.

Here’s the basic flow of a **Lambda function** from beginning to end:

```
1. Event Trigger
   |
   V
2. Lambda Function Invoked
   |
   V
3. Initialize Execution Environment
   |    - AWS sets up a runtime environment if it's a cold
start
   |    - Loads dependencies and code if not already initia
lized
   |
   V
4. Lambda Code Execution Begins
   |    - AWS passes the Event Object and Context Object to
the handler
   |
   V
5. Handler Processes the Event
   |    - Executes the main logic of the function
   |    - Interacts with other AWS services (e.g., S3, Dyna
moDB, SNS) if needed
```

```
   |    - Uses environment variables and other resources
   |
   V
6. Return Response
   |    - Lambda function returns a response to the caller
or trigger service
   |    - If it's an asynchronous invocation, the response
is not directly returned
   |
   V
7. Log Output to CloudWatch
   |    - Logs generated during execution are sent to Cloud
Watch Logs
   |
   V
8. Execution Environment Frozen
   |    - AWS freezes the environment to reuse it for subse
quent invocations (warm start)
   |    - If unused for a period, the environment is eventu
ally shut down
   |
   V
9. Function Ends
```

Example 1: *Real-Time Log Monitoring and Alerting.*

Business Requirement: Suppose a company wants to monitor application logs in real time to identify any security threats, errors, or anomalies and send alerts to the support team.

Traditional Approach (Before Lambda):

```
1. Log Entry Generated
   |
   V
2. Log Sent to Monitoring Server (EC2 Instance)
```

```
   |
   V
3. Server Continuously Monitors and Processes Logs
   |    - Analyzes logs for errors or security threats
   |    - Requires continuous server uptime, even if no log
s are generated
   |
   V
4. If Issue Detected
   |
   V
5. Server Sends Alert
      - Notifies support team via email, SMS, or custom ale
rting system
   |
   V
6. Support Team Takes Action
```

Challenges:

- **High Costs**: Continuous server uptime incurs costs even when no logs are generated.
- **Manual Scaling**: Scaling for high log volumes requires additional EC2 instances, load balancers, and configuration.
- **Operational Overhead**: Requires constant server maintenance, patching, and scaling.

AWS Lambda Solution:

```
1. Log Entry Generated
   |
   V
2. Log Sent to CloudWatch Logs
   |
   V
3. CloudWatch Logs Trigger Lambda Function
   |
   V
```

```
4. Lambda Function Processes Log Entry
   |    - Analyzes log for errors or security threats
   |    - Only runs when new log data is available, minimiz
ing idle costs
   |
   V
5. If Issue Detected
   |
   V
6. Lambda Sends Alert
      - Notifies support team via SNS or Slack
   |
   V
7. Support Team Takes Action
```

Benefits:

- **Cost Savings**: Only pays for Lambda's compute time when processing logs, eliminating idle costs.
- **Automatic Scaling**: Lambda scales automatically with log volume, with no manual configuration required.
- **Reduced Maintenance**: No server patching, scaling, or maintenance, reducing operational burden.

Lambda Function Code

This script assumes you've set up an **SNS topic** and configured **CloudWatch Logs** to trigger the Lambda function.

```
import boto3
import os
import json

# Initialize AWS clients
sns_client = boto3.client('sns')

# Environment variables for SNS topic and keywords to monit
or
```

```
SNS_TOPIC_ARN = os.environ['SNS_TOPIC_ARN']  # Add your SNS
Topic ARN in environment variables
KEYWORDS = ["error", "unauthorized", "threat", "failed"]

def lambda_handler(event, context):
    # Extract log data from the CloudWatch Logs event
    log_data = event['awslogs']['data']

    # Decode and decompress log data
    log_events = decode_log_data(log_data)

    # Analyze each log event for the specified keywords
    for log_event in log_events:
        if detect_issue(log_event):
            # Send an alert if an issue is detected
            send_alert(log_event)

    return {"status": "completed"}

def decode_log_data(log_data):
    import base64
    import gzip
    import json

    # Decode the base64-encoded, gzipped log data
    decoded_data = base64.b64decode(log_data)
    decompressed_data = gzip.decompress(decoded_data)
    log_events = json.loads(decompressed_data)

    return log_events['logEvents']

def detect_issue(log_event):
    # Check for specified keywords in the log message
    log_message = log_event['message'].lower()
    for keyword in KEYWORDS:
        if keyword in log_message:
            print(f"Issue detected with keyword '{keywor
d}': {log_message}")
```

```
            return True
    return False

def send_alert(log_event):
    # Prepare alert message
    alert_message = f"Security Alert: Issue detected in log
s\n\nLog Message:\n{log_event['message']}\nTimestamp: {log_
event['timestamp']}"

    # Publish alert to SNS
    response = sns_client.publish(
        TopicArn=SNS_TOPIC_ARN,
        Subject="Security Alert: Issue Detected in CloudWat
ch Logs",
        Message=alert_message
    )
    print(f"Alert sent via SNS: {response}")
```

Explanation of Each Function:

```
1. lambda_handler:
The main entry point of the Lambda function.
Decodes log data and checks each log event for keywords ind
icating issues.

2. decode_log_data:
Decodes the base64-encoded, gzipped CloudWatch Logs data.
Returns a list of log events from the data.

3. detect_issue:
Searches each log message for the keywords specified in the
KEYWORDS list.
Returns True if an issue is detected, which triggers an ale
rt.

4. send_alert:
Sends an alert via SNS to notify the support team if an iss
```

```
ue is detected in the logs.
The message includes the log message and timestamp.
```

Environment Variables Required:

```
SNS_TOPIC_ARN:
The ARN of the SNS topic to which alerts will be sent.
KEYWORDS:
Customize the list of keywords you want to search for in lo
g messages.
```

Setting Up the CloudWatch Logs Trigger:

1. In the **CloudWatch Logs console**, create a **subscription filter** on the log group you want to monitor.
2. Set the **destination** to the Lambda function created with this script, so it receives log data in real-time.

Example 2: Automated EC2 Instance Shutdown Using AWS Lambda

Scenario: Many organizations have **non-production EC2 instances** (e.g., for development, testing, or staging) that are often left running after business hours or on weekends, leading to unnecessary costs. Additionally, leaving these instances active increases the security risk of potential unauthorized access when not in use.

Solution: Use AWS Lambda with **CloudWatch Events** to automate the shutdown of non-production EC2 instances during off-hours. This reduces operational costs and enhances security by limiting the exposure of non-production instances.

Traditional Approach:

- Manually stopping instances or using custom scripts on a dedicated server to manage instance schedules.

- Higher costs due to the continuous operation of EC2 instances or servers running automation scripts.
- Potential for human error if the shutdown process is missed.

Lambda Solution:

- Fully automated with scheduled triggers, eliminating the need for manual intervention.
- Only pays for the milliseconds Lambda runs, leading to cost savings compared to dedicated automation servers.
- Scalable and easily customizable, supporting any number of instances across accounts and regions.

Automated EC2 Instance Shutdown Using AWS Lambda solution:

```
1. CloudWatch Event Trigger (Scheduled at 7 PM)
   |
   V
2. Lambda Function Invoked
   |
   V
3. Lambda Checks for Non-Production Instances
   |    - Filters instances by tags
          (e.g., "Environment: Development" or "Testing")
   |    - Identifies instances in the "running" state
   |
   V
4. Stop Command Issued for Each Matching Instance
   |    - Lambda sends stop command to each tagged instance
   |
   V
5. Optional Notification
   |    - Lambda sends an alert via SNS to notify team of t
he shutdown
   |
   V
6. Instances Stopped, Reducing Costs
```

Optional Scheduled Start (Next Morning):

```
1. CloudWatch Event Trigger (Scheduled at 8 AM)
   |
   V
2. Lambda Function Invoked
   |
   V
3. Lambda Checks for Non-Production Instances
   |    - Filters instances by tags
          (e.g., "Environment: Development" or "Testing")
   |    - Identifies instances in the "stopped" state
   |
   V
4. Start Command Issued for Each Matching Instance
   |    - Lambda sends start command to each tagged instanc
e
   |
   V
5. Optional Notification
   |    - Lambda sends an alert via SNS to notify team of t
he instance startup
   |
   V
6. Instances Started, Ready for Use
```

Steps for Automated Stop and Start of EC2 Instances:

1. **Tag EC2 Instances**:

- Tag the instances you want to automatically stop and start (e.g., add a tag `Environment=Dev` or `AutoSchedule=True`).

2. **Create IAM Role for Lambda**:

- Create an IAM role with permissions for Lambda and EC2.
- Attach the policy with `ec2:StopInstances`, `ec2:StartInstances`, and `ec2:DescribeInstances` permissions.

3. **Create Lambda Functions**:

- **Stop Function**: Write a Lambda function to stop instances based on the tag.

- **Start Function**: Write a separate Lambda function to start instances based on the tag.

4. **Create CloudWatch Events Rules**:

- Set up **two CloudWatch Events rules** (one for stopping and one for starting).
- Configure the **stop rule** to trigger the Stop Lambda function at the end of business hours (e.g., 7 PM).
- Configure the **start rule** to trigger the Start Lambda function at the beginning of business hours (e.g., 8 AM).

5. **Test the Setup**:

- Run the Lambda functions manually to ensure they stop and start the tagged instances as expected.

6. **Monitor Logs**:

- Use **CloudWatch Logs** to monitor the Lambda execution logs and verify that instances are being stopped and started as scheduled.

1. Lambda Function to Stop EC2 Instances

This function stops all EC2 instances with a specified tag (`AutoSchedule=True`), which is typically run after business hours (e.g., at 7 PM).

```
import boto3
import os

# Initialize the EC2 client
ec2_client = boto3.client('ec2')

# Lambda handler function to stop instances
def lambda_handler(event, context):
    # Define the tag key and value to filter instances
    tag_key = 'AutoSchedule'
    tag_value = 'True'

    # Filter instances by tag and running state
    filters = [
```

```
        {'Name': f'tag:{tag_key}', 'Values': [tag_value]},
        {'Name': 'instance-state-name', 'Values': ['runnin
g']}
    ]

    # Describe instances with the specified filters
    instances = ec2_client.describe_instances(Filters=filte
rs)

    # Collect instance IDs to stop
    instance_ids = [instance['InstanceId'] for reservation
in instances['Reservations'] for instance in reservation['I
nstances']]

    if instance_ids:
        # Stop instances
        ec2_client.stop_instances(InstanceIds=instance_ids)
        print(f"Stopping instances: {instance_ids}")
    else:
        print("No instances found to stop.")
```

2. Lambda Function to Start EC2 Instances

This function starts all EC2 instances with the specified tag (`AutoSchedule=True`), which is typically run at the beginning of business hours (e.g., at 8 AM).

```
import boto3
import os

# Initialize the EC2 client
ec2_client = boto3.client('ec2')

# Lambda handler function to start instances
def lambda_handler(event, context):
    # Define the tag key and value to filter instances
    tag_key = 'AutoSchedule'
```

```
    tag_value = 'True'

    # Filter instances by tag and stopped state
    filters = [
        {'Name': f'tag:{tag_key}', 'Values': [tag_value]},
        {'Name': 'instance-state-name', 'Values': ['stoppe
d']}
    ]

    # Describe instances with the specified filters
    instances = ec2_client.describe_instances(Filters=filte
rs)

    # Collect instance IDs to start
    instance_ids = [instance['InstanceId'] for reservation
in instances['Reservations'] for instance in reservation['I
nstances']]

    if instance_ids:
        # Start instances
        ec2_client.start_instances(InstanceIds=instance_id
s)
        print(f"Starting instances: {instance_ids}")
    else:
        print("No instances found to start.")
```

Example 3: AWS Cost Optimization Example using Lambda: Identifying Stale EBS Snapshots

(***This example is credited to Mr. Abhishek Veeramalla. Thank you for your invaluable support and guidance to the DevOps community!***)

In this example, we'll create a Lambda function that identifies EBS snapshots that are no longer associated with any active EC2 instance and deletes them to save on storage costs.

Description:
The Lambda function fetches all EBS snapshots owned by the account in a specified
target region and checks if each snapshot's associated volume (if any) is attached to an active instance. If a snapshot is **stale** (i.e., its volume is deleted or unattached), the function deletes it, optimizing storage costs by removing unnecessary snapshots.

```
import boto3
import os

# Specify the target region where your EC2 resources are lo
cated
TARGET_REGION = 'us-east-1'  # Replace 'us-east-1' with the
correct region

# Initialize EC2 client in the target region
ec2 = boto3.client('ec2', region_name=TARGET_REGION)

def lambda_handler(event, context):
    # Get all EBS snapshots in the specified region
    response = ec2.describe_snapshots(OwnerIds=['self'])
    print(f"Total snapshots found: {len(response['Snapshot
s'])}")

    # Iterate through each snapshot
    for snapshot in response['Snapshots']:
        snapshot_id = snapshot['SnapshotId']
        volume_id = snapshot.get('VolumeId')
        print(f"Checking snapshot: {snapshot_id}, Volume I
D: {volume_id}")

        if not volume_id:
            # Delete the snapshot if it's not attached to a
ny volume
            ec2.delete_snapshot(SnapshotId=snapshot_id)
            print(f"Deleted EBS snapshot {snapshot_id} as i
t was not attached to any volume.")
```

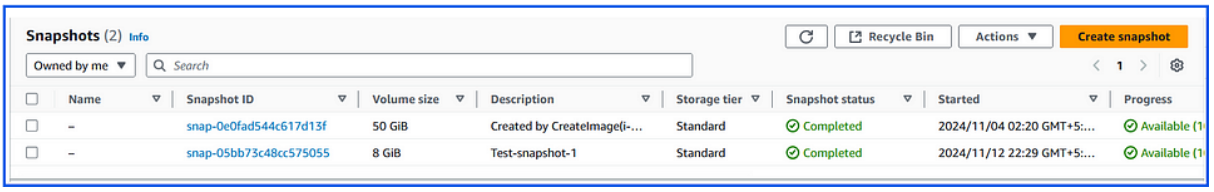
```
        else:
            # Check if the volume still exists
            try:
                volume_response = ec2.describe_volumes(Volu
meIds=[volume_id])
                if not volume_response['Volumes'][0]['Attac
hments']:
                    # Delete snapshot if volume exists but
is not attached to a running instance
                    ec2.delete_snapshot(SnapshotId=snapshot
_id)
                    print(f"Deleted EBS snapshot {snapshot_
id} as it was taken from a volume not attached to any runni
ng instance.")
                else:
                    print(f"Volume {volume_id} is still att
ached; skipping snapshot {snapshot_id}.")
            except ec2.exceptions.ClientError as e:
                if e.response['Error']['Code'] == 'InvalidV
olume.NotFound':
                    # Delete snapshot if associated volume
is not found
                    ec2.delete_snapshot(SnapshotId=snapshot
_id)
                    print(f"Deleted EBS snapshot {snapshot_
id} as its associated volume was not found.")
                else:
                    # Log other errors
                    print(f"Error processing snapshot {snap
shot_id}: {e}")
```

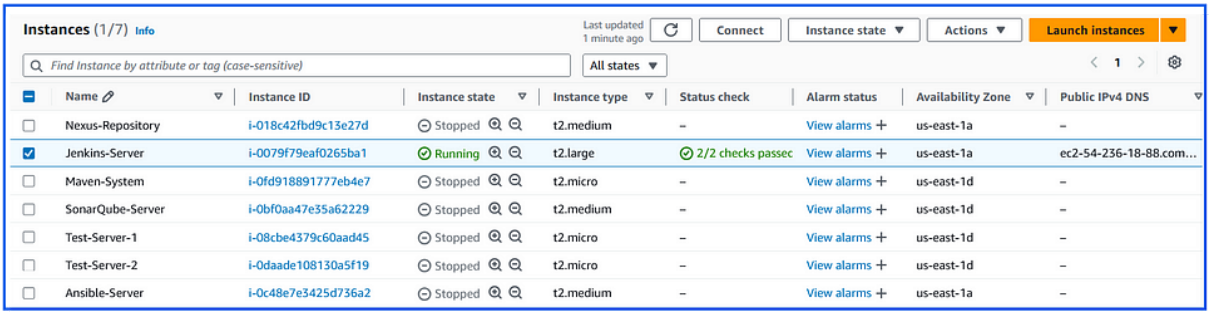
Here's a structured flow diagram for the **Lambda Function to Clean Up EBS Snapshots** that targets a specified AWS region:

```
1. Lambda Function Triggered
   |
   V
2. Initialize EC2 Client in Target Region
```

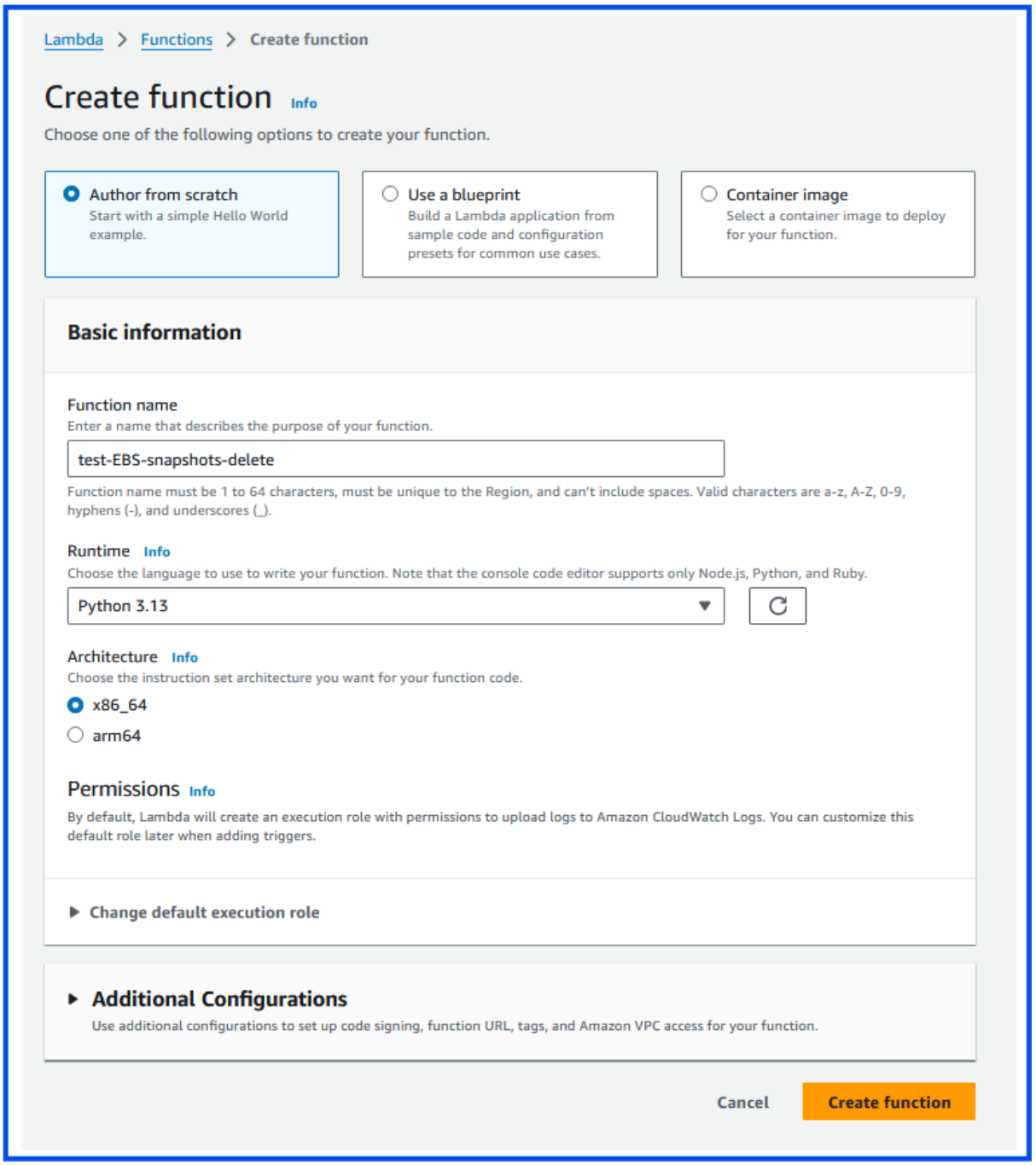
```
   |
   V
3. Retrieve All EBS Snapshots (Owned by Account) in Target
Region
   |
   V
4. Iterate Through Each Snapshot
   |
   |    A. Get Snapshot ID and Associated Volume ID
   |
   |    B. If No Volume ID:
   |       - Delete Snapshot (Not Attached to Any Volume)
   |
   |    C. If Volume ID Exists:
   |       - Check if Volume Still Exists
   |       |
   |       |    i. If Volume Exists but Has No Attachments:
   |       |       - Delete Snapshot (Volume Not Attached t
o Any Instance)
   |       |
   |       |    ii. If Volume Not Found (Error Code 'Invali
dVolume.NotFound'):
   |       |       - Delete Snapshot (Volume Deleted)
   |       |
   |       |    iii. If Volume Exists and Has Attachments:
   |       |       - Skip Deletion (Volume is Attached to a
n Instance)
   |
   V
5. End of Snapshot Iteration
   |
   V
6. Function Completes
```

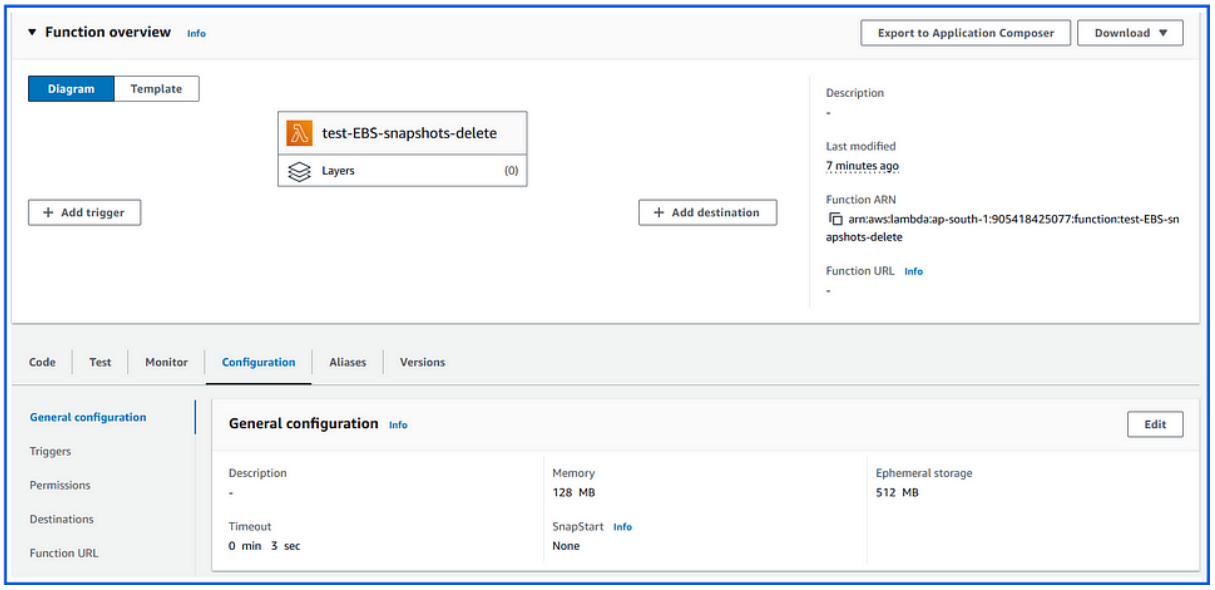
List of snapshots are available:



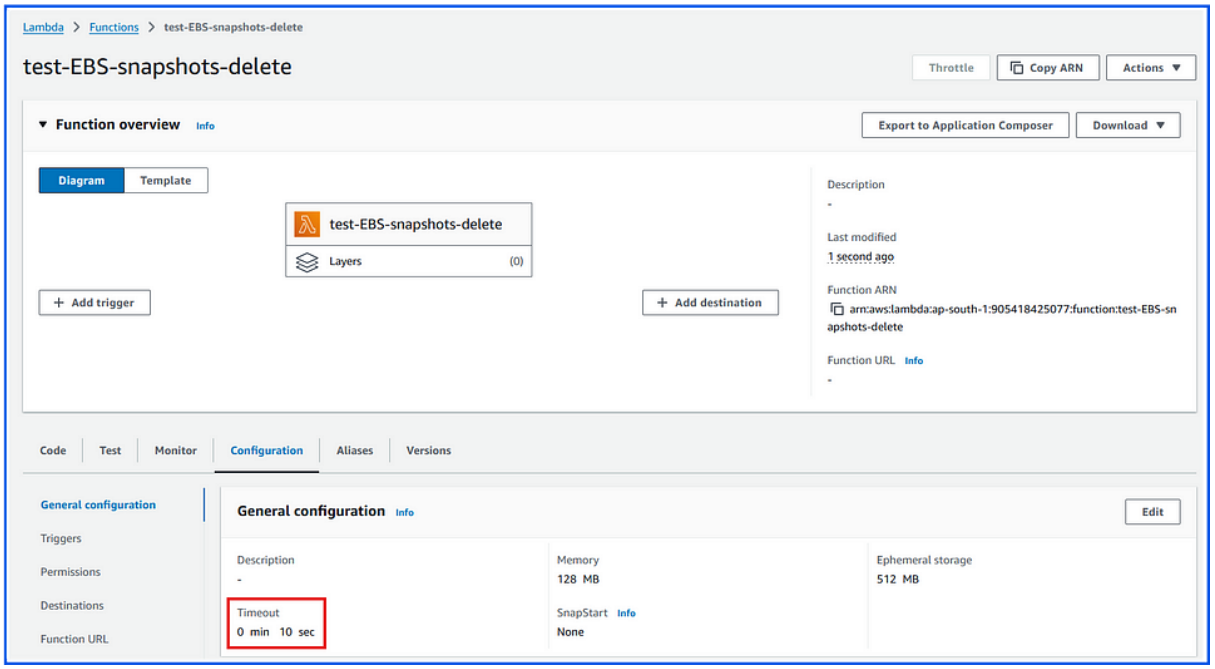


Step-1: Create a Lambda Function with Python Runtime

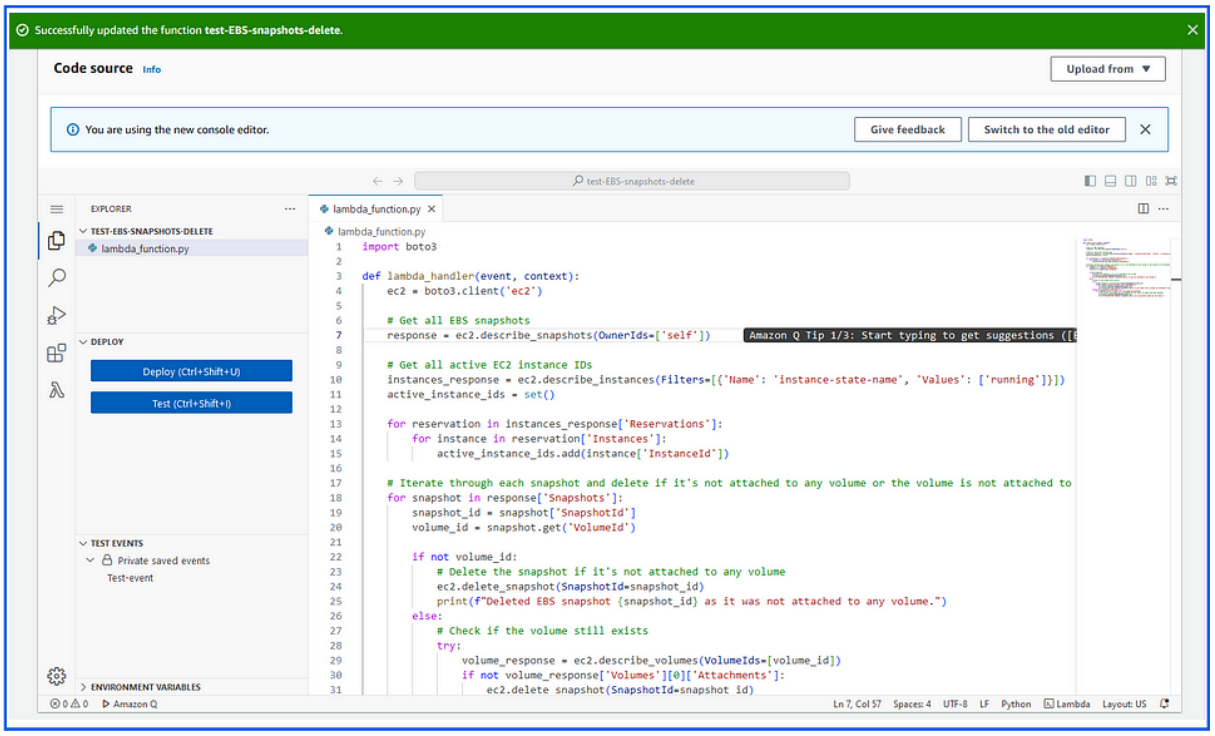

Lambda > Functions > Create function
Create function Info
Choose one of the following options to create your function.
Author from scratch
Start with a simple Hello World example.
Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.
Container image
Select a container image to deploy for your function.
Basic information
Function name
Enter a name that describes the purpose of your function.
test-EBS-snapshots-delete
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.13
Architecture Info
Choose the instruction set architecture you want for your function code.
x86_64
arm64
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.
Change default execution role
Additional Configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.
Cancel
Create function



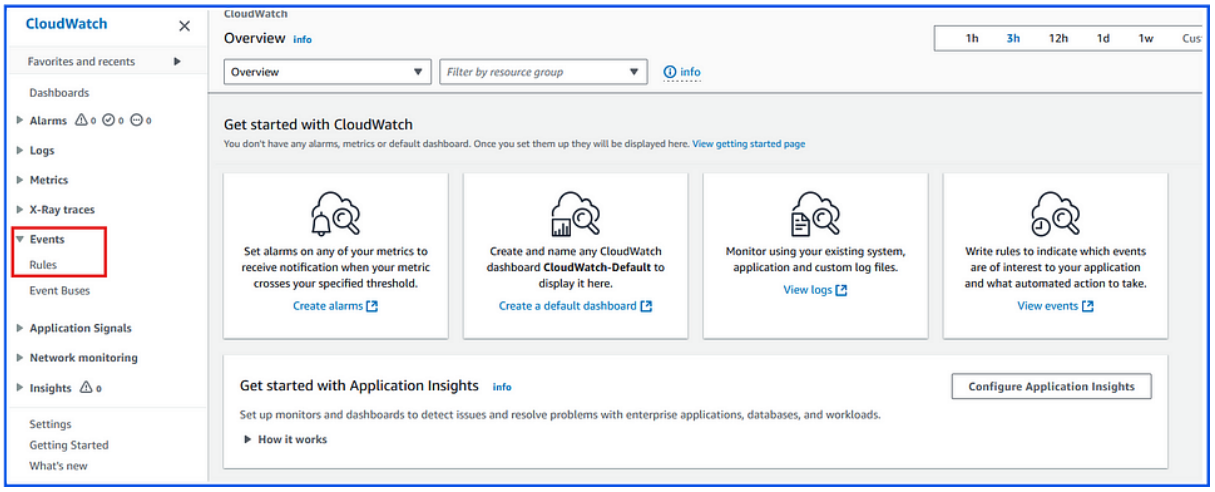
Changed the Timeout to 10 sec:



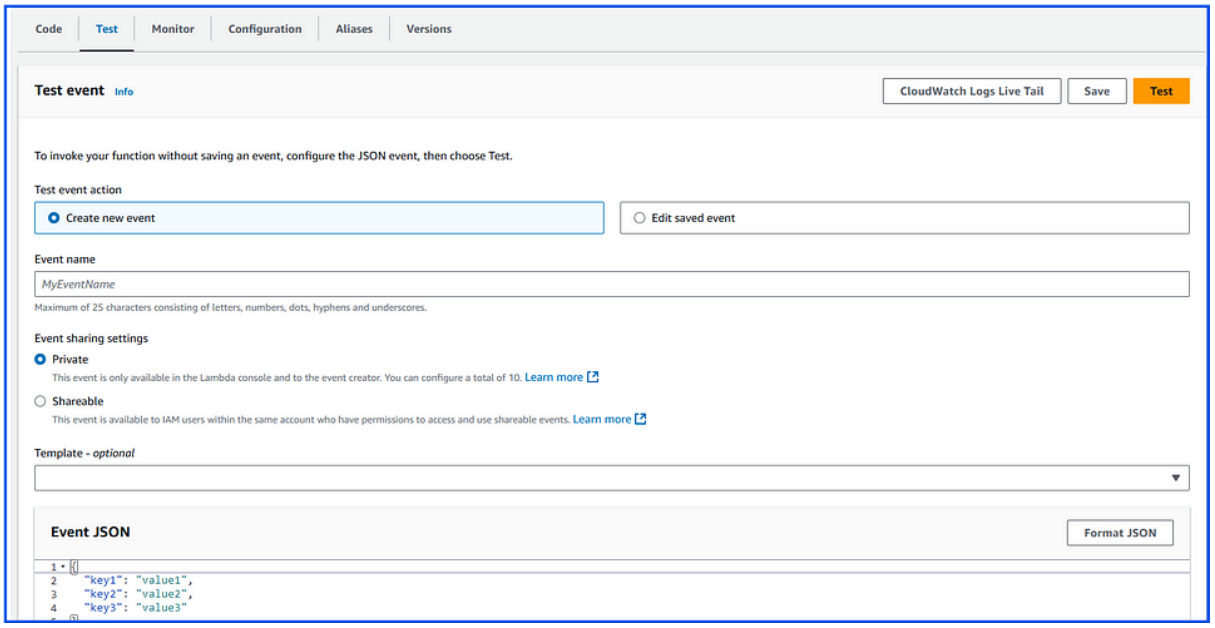
Add the python script code which handles the requirement and deploy the function:

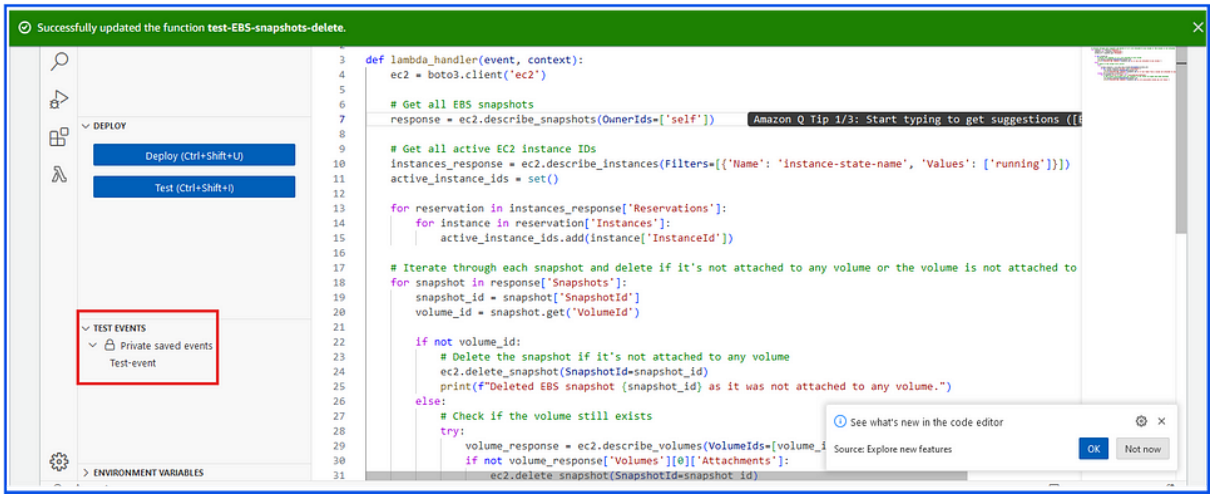


Create a manual test event to run the lambda function. (Note: You can use the Cloud Watch for creating an event to do this job.

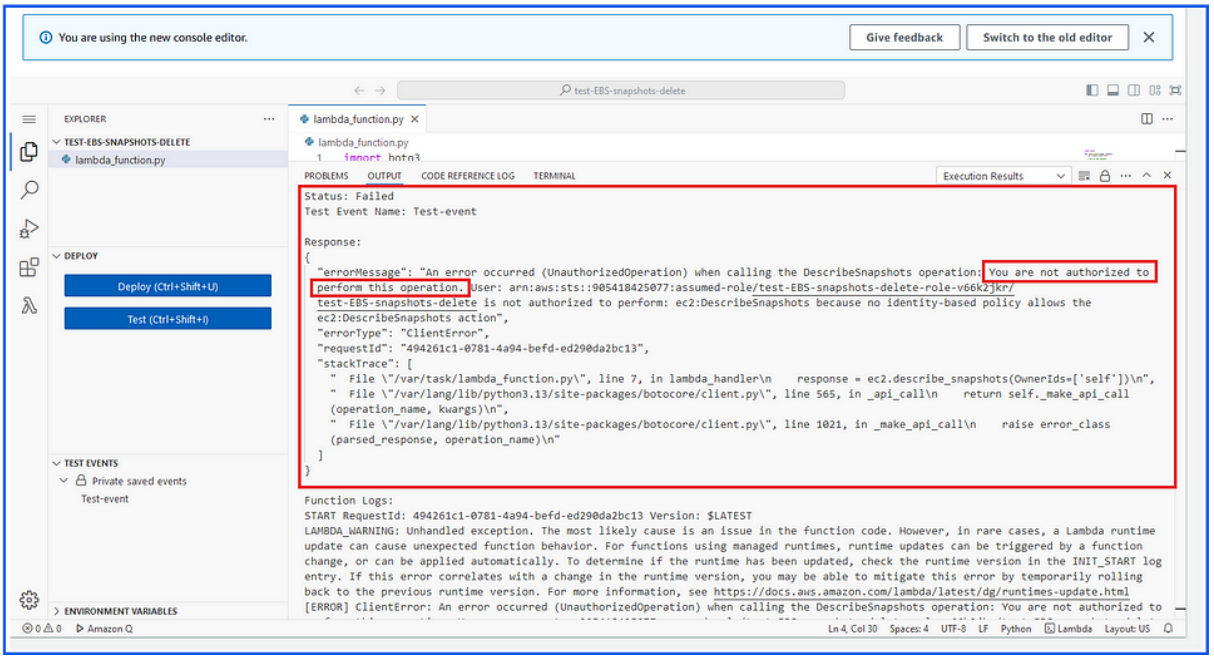


Creating a manual test for Lambda function:



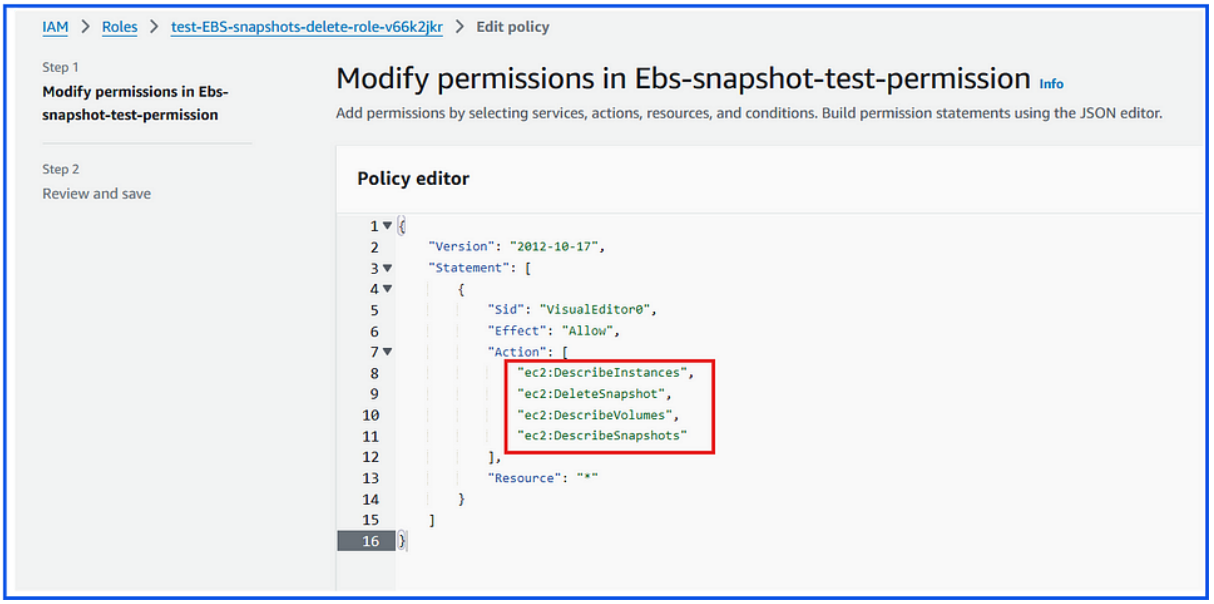


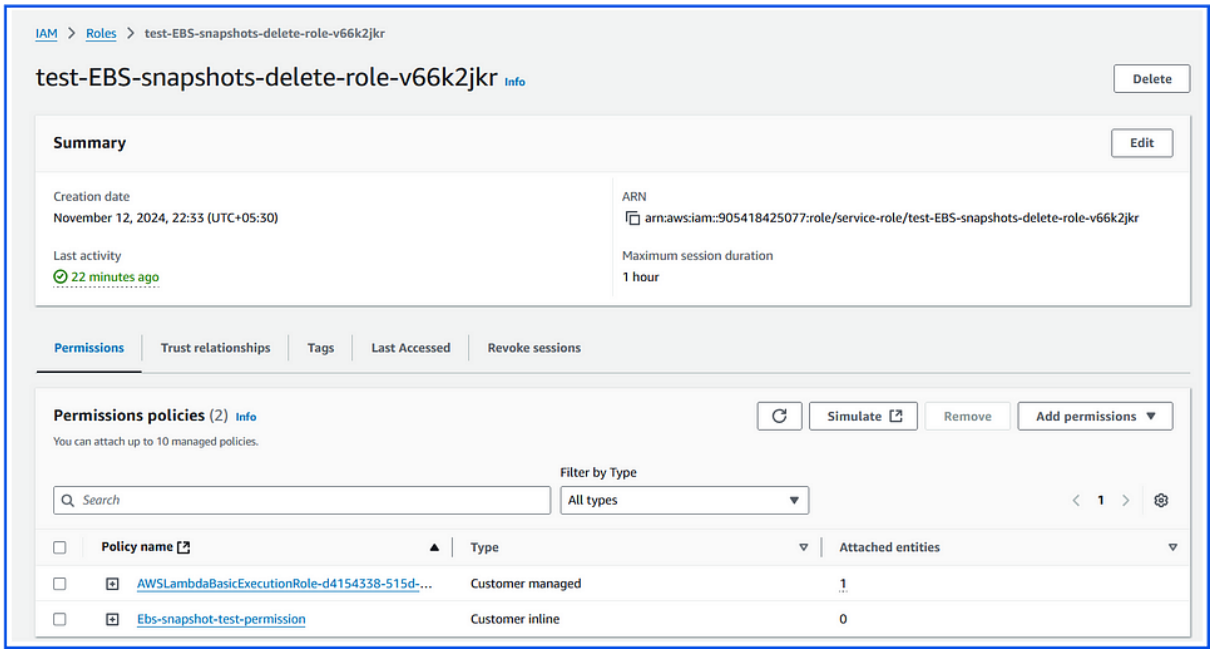
If you test this function without required IAM permissions, it will get failed:



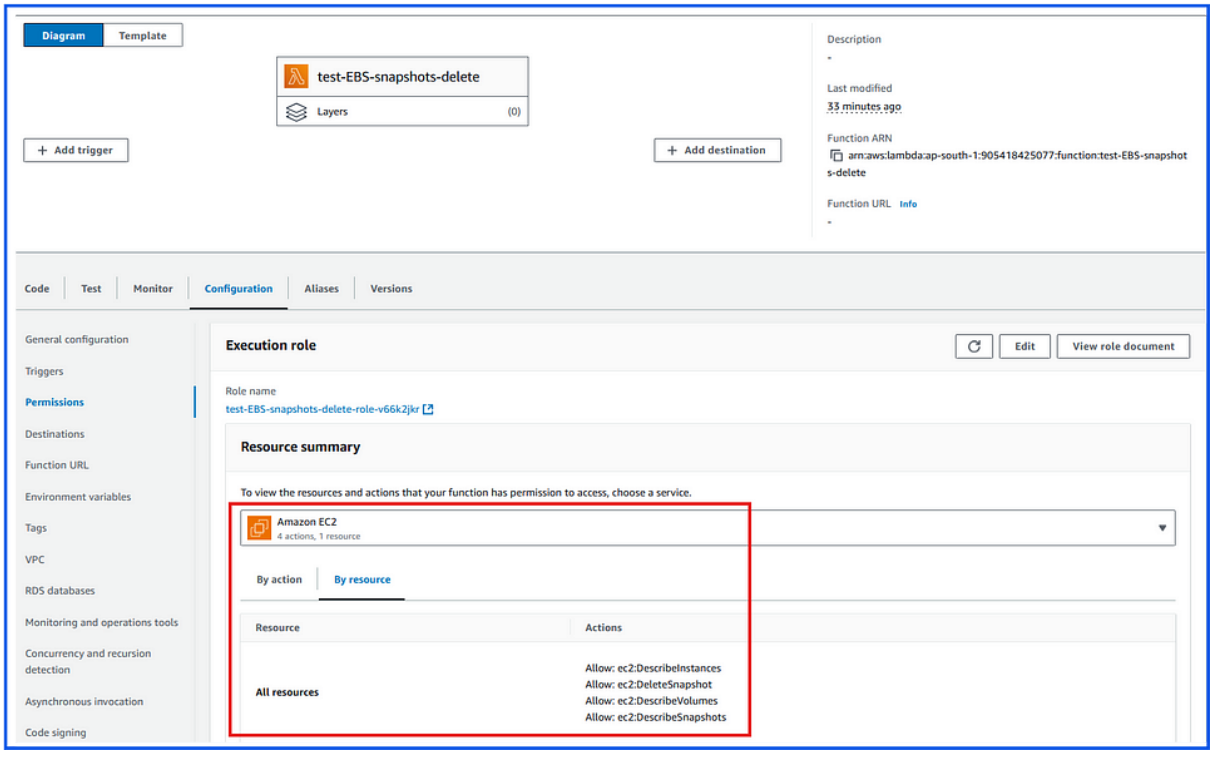
We need to add the following permissions to the existing IAM Role to execute this function:

- `ec2:DescribeSnapshots`: Allows listing of snapshots owned by the account.
- `ec2:DescribeVolumes`: Allows the function to verify if a volume associated with a snapshot still exists.
- `ec2:DescribeInstances`: Allows the function to retrieve details of running EC2 instances.
- `ec2:DeleteSnapshot`: Allows the function to delete snapshots that are no longer in use.



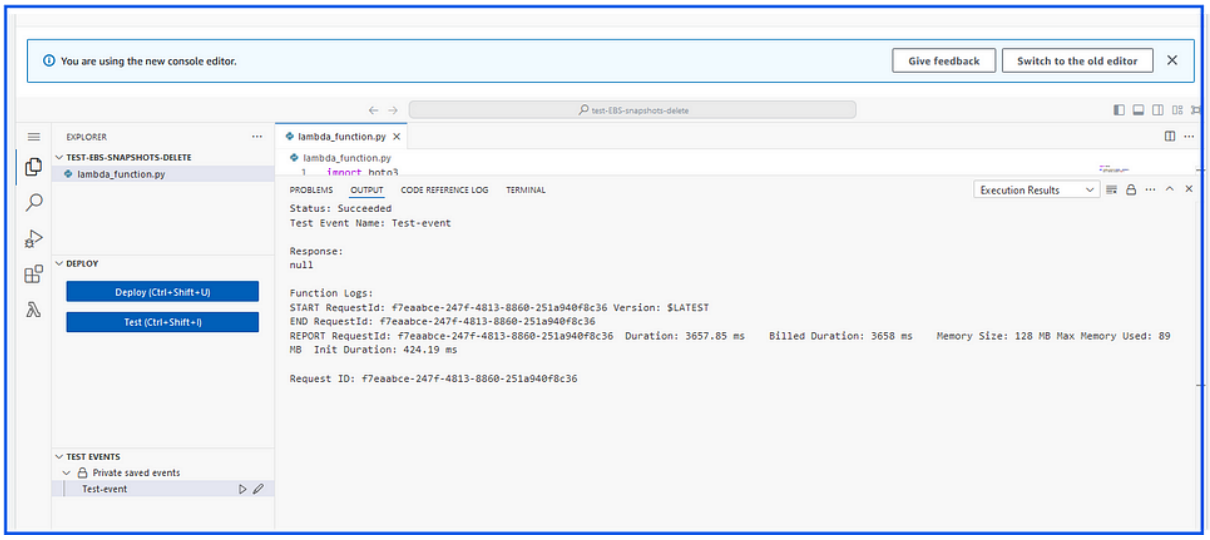


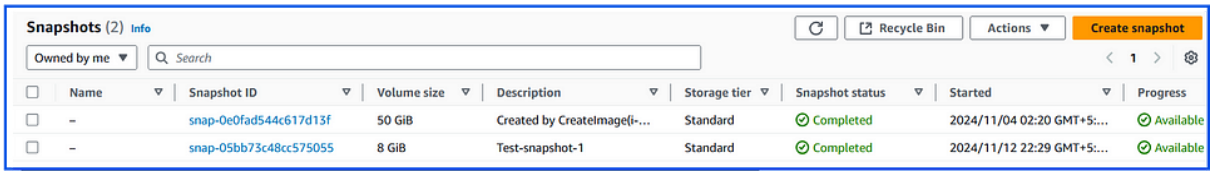
You can check in the Lambda function for updating of newly added IAM permissions:



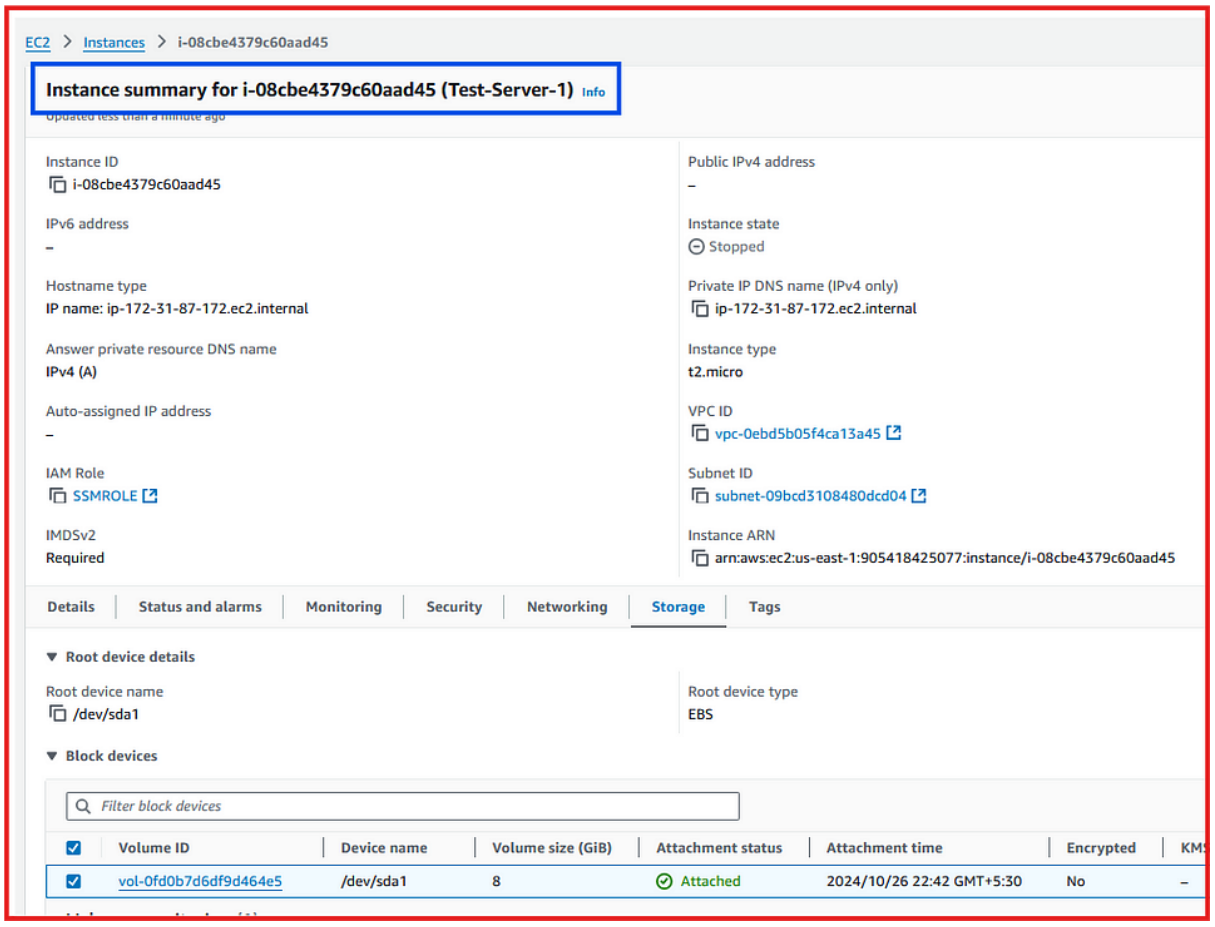
Now we will test the lambda function again:

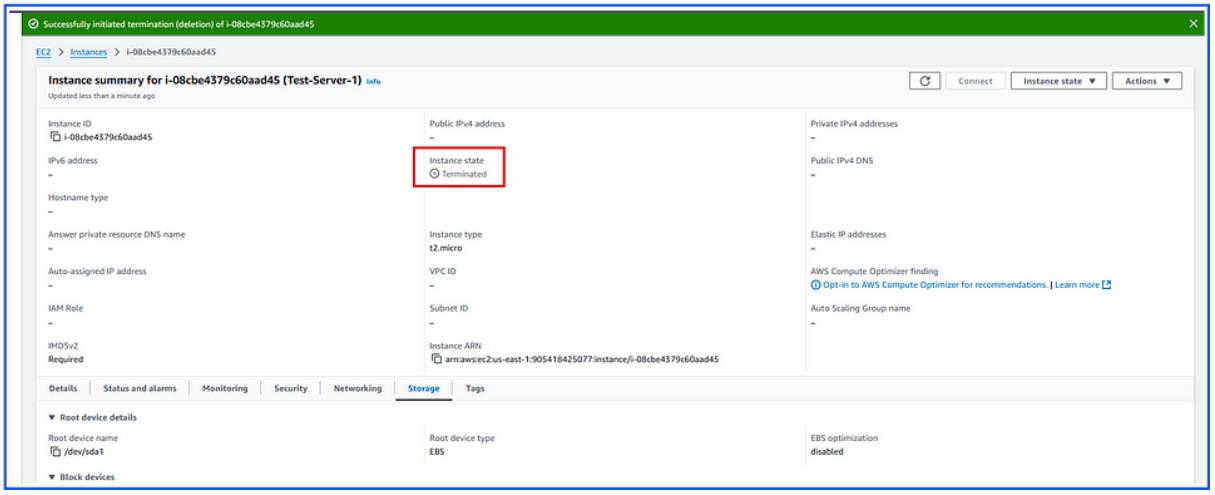
It was not deleted any snapshots because they are attached to volumes, and they are attached to ec2-instances:

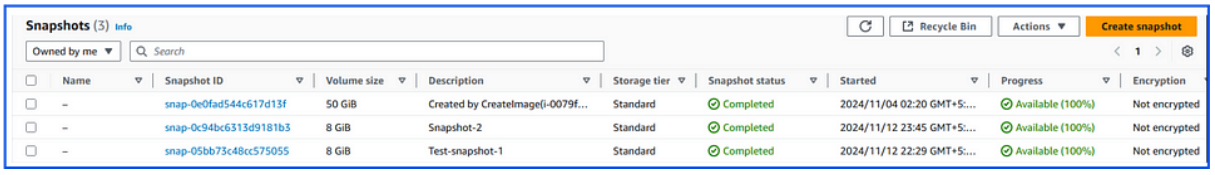




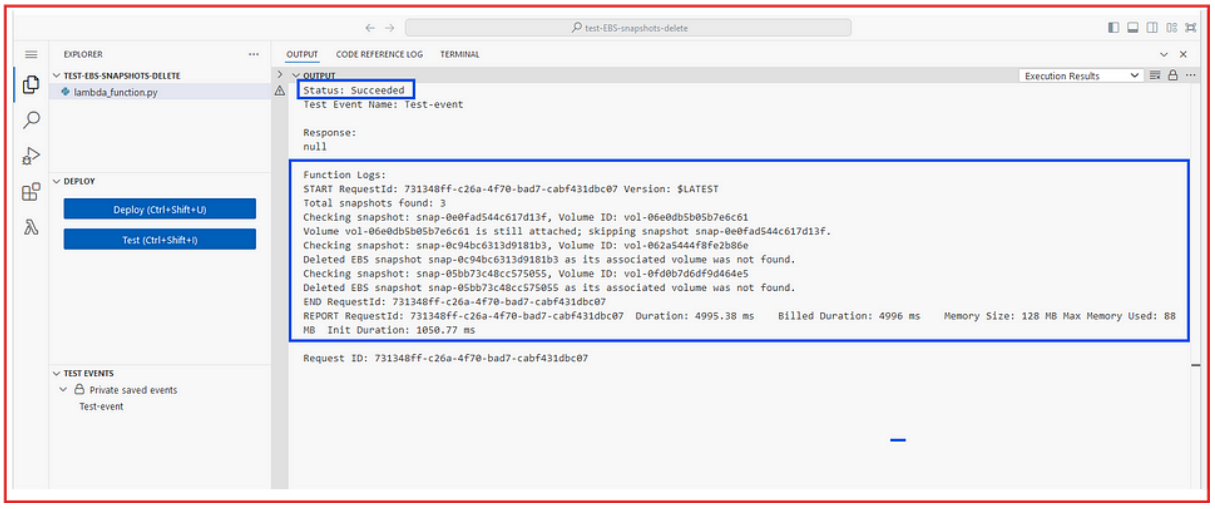
We will delete the ec2-instance that attached by the above EBS volume and run the test once again:

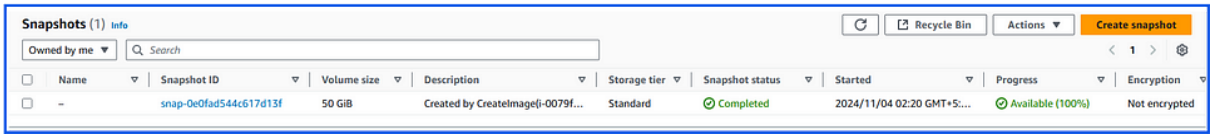






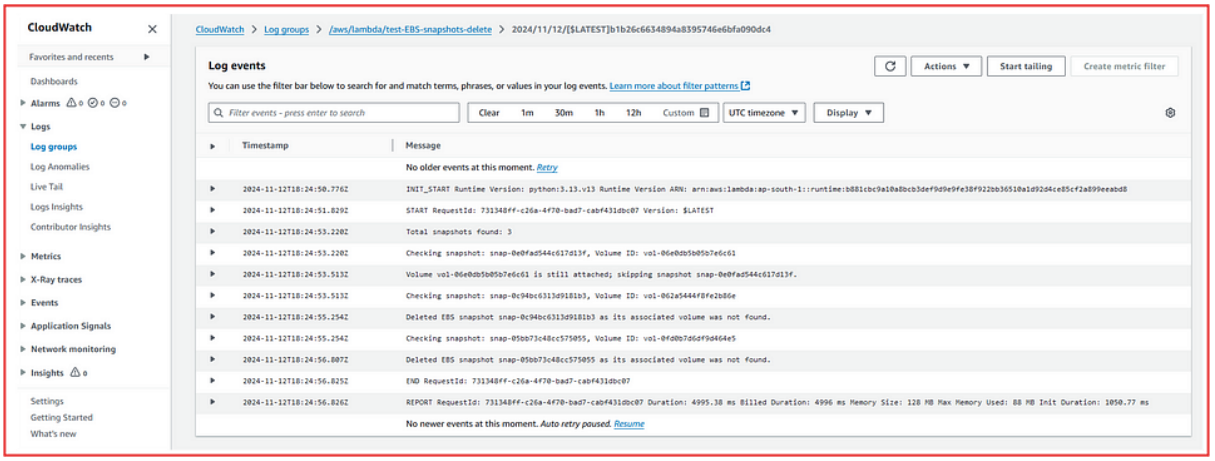
Now run the Test event:





Our Lambda function worked perfectly and successfully deleted snapshot volumes that were not associated with any volumes attached to EC2 instances.

We can also observe the event in CloudWatch Logs:



In this way, we can use AWS Lambda functions for cost optimization and resource security.

> ***Thank you. Happy Learning!***