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': ['running']}
    ]

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

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

    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.")
```

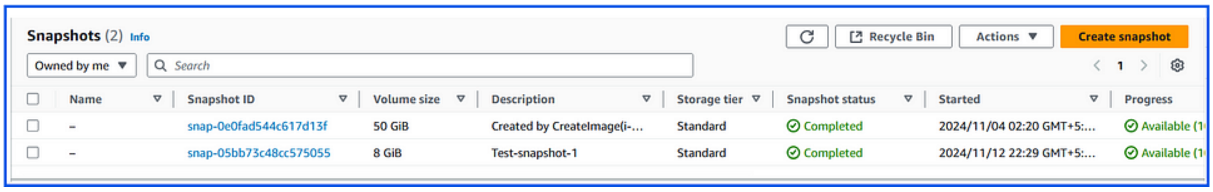
```
        else:
            # Check if the volume still exists
            try:
                volume_response = ec2.describe_volumes(Volu
meIds=[volume_id])
                if not volume_response['Volumes'][0]['Attac
hments']:
                    # Delete snapshot if volume exists but
is not attached 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:

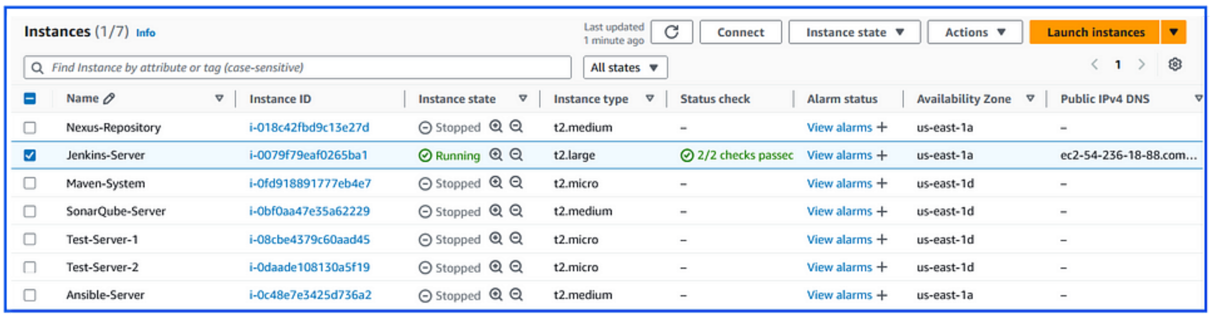
```
1. Lambda Function Triggered
   |
   V
2. Initialize EC2 Client in Target Region
```

```
   |
   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:

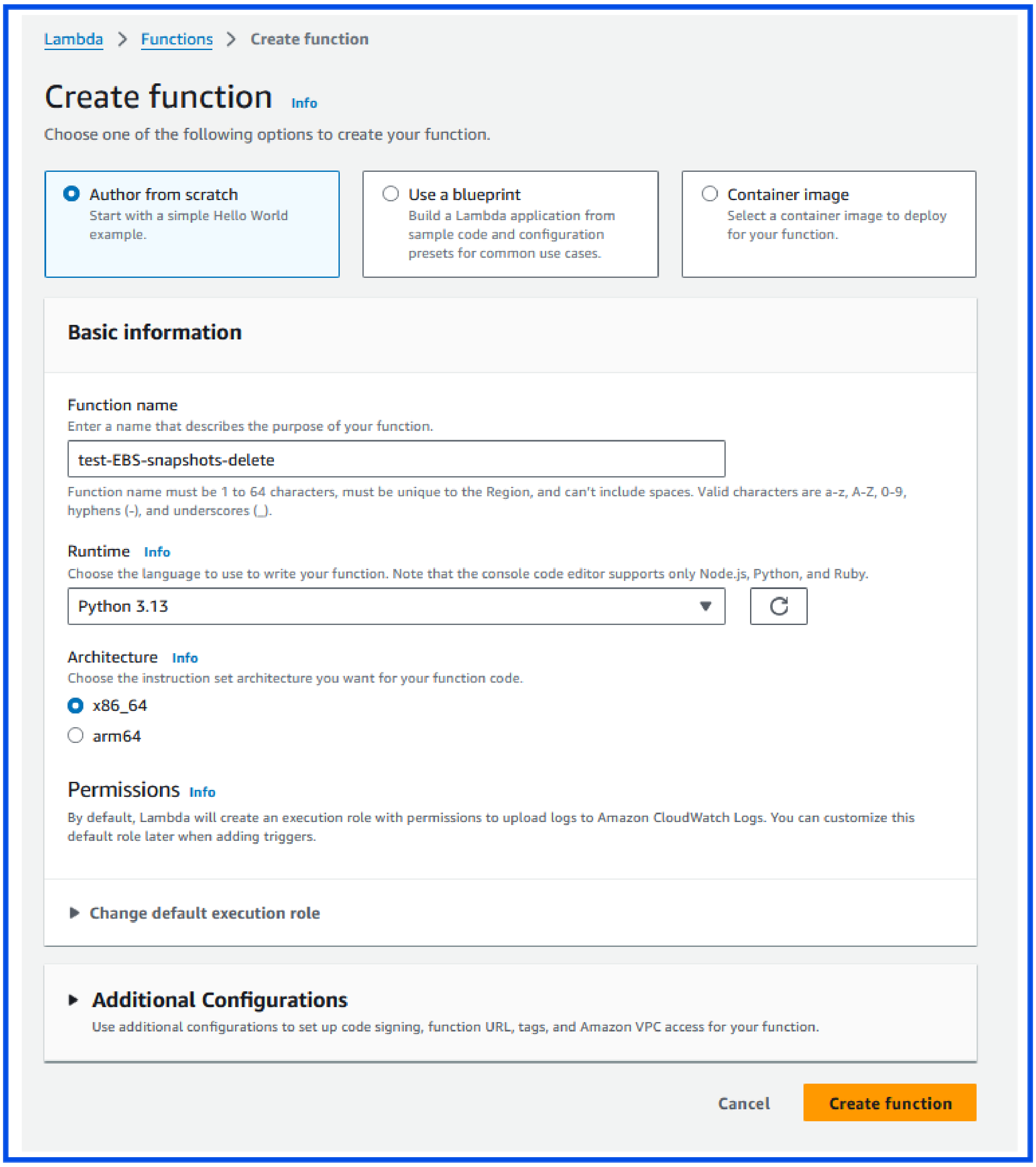


Name	Snapshot ID	Volume size	Description	Storage tier	Snapshot status	Started	Progress
–	snap-0e0fad544c617d13f	50 GiB	Created by CreateImage(i-…	Standard	✓ Completed	2024/11/04 02:20 GMT+5:…	✓ Available (1
–	snap-05bb73c48cc575055	8 GiB	Test-snapshot-1	Standard	✓ Completed	2024/11/12 22:29 GMT+5:…	✓ Available (1


Instances (1/7) Info
Last updated 1 minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
1

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
☐	Nexus-Repository	i-018c42fbd9c13e27d	Stopped	t2.medium	–	View alarms +	us-east-1a	–
✓	Jenkins-Server	i-0079f79eaf0265ba1	Running	t2.large	✓ 2/2 checks passec	View alarms +	us-east-1a	ec2-54-236-18-88.com…
☐	Maven-System	i-0fd918891777eb4e7	Stopped	t2.micro	–	View alarms +	us-east-1d	–
☐	SonarQube-Server	i-0bf0aa47e35a62229	Stopped	t2.medium	–	View alarms +	us-east-1d	–
☐	Test-Server-1	i-08cbe4379c60aad45	Stopped	t2.micro	–	View alarms +	us-east-1d	–
☐	Test-Server-2	i-0daade108130a5f19	Stopped	t2.micro	–	View alarms +	us-east-1d	–
☐	Ansible-Server	i-0c48e7e3425d736a2	Stopped	t2.medium	–	View alarms +	us-east-1a	–

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 (_).

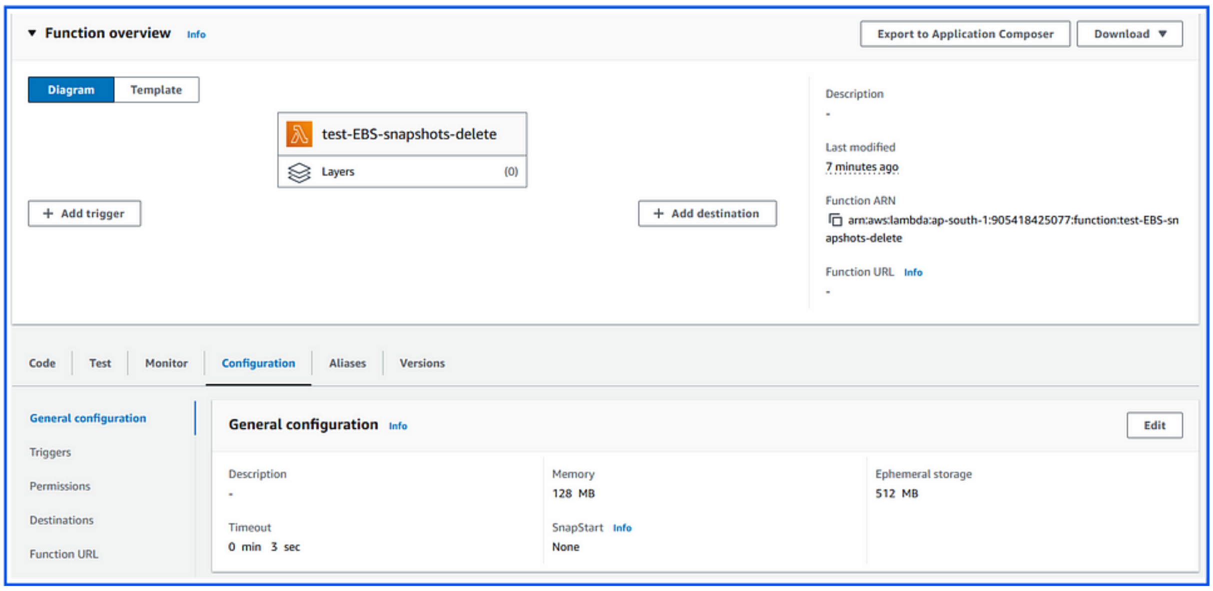
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.13

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.

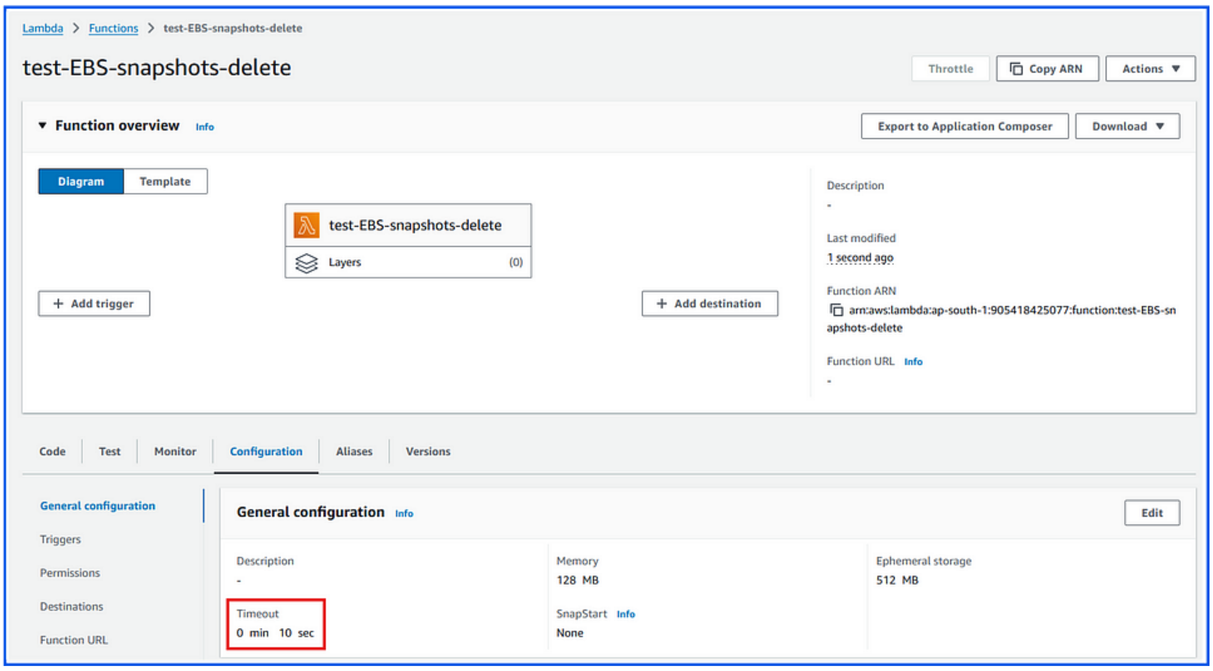


Description
-
Last modified
7 minutes ago
Function ARN
arn:aws:lambda:ap-south-1:905418425077:function:test-EBS-snapshots-delete
Function URL
-

General configuration

Description
-
Timeout
0 min 3 sec
Memory
128 MB
SnapStart
None
Ephemeral storage
512 MB

Changed the Timeout to 10 sec:

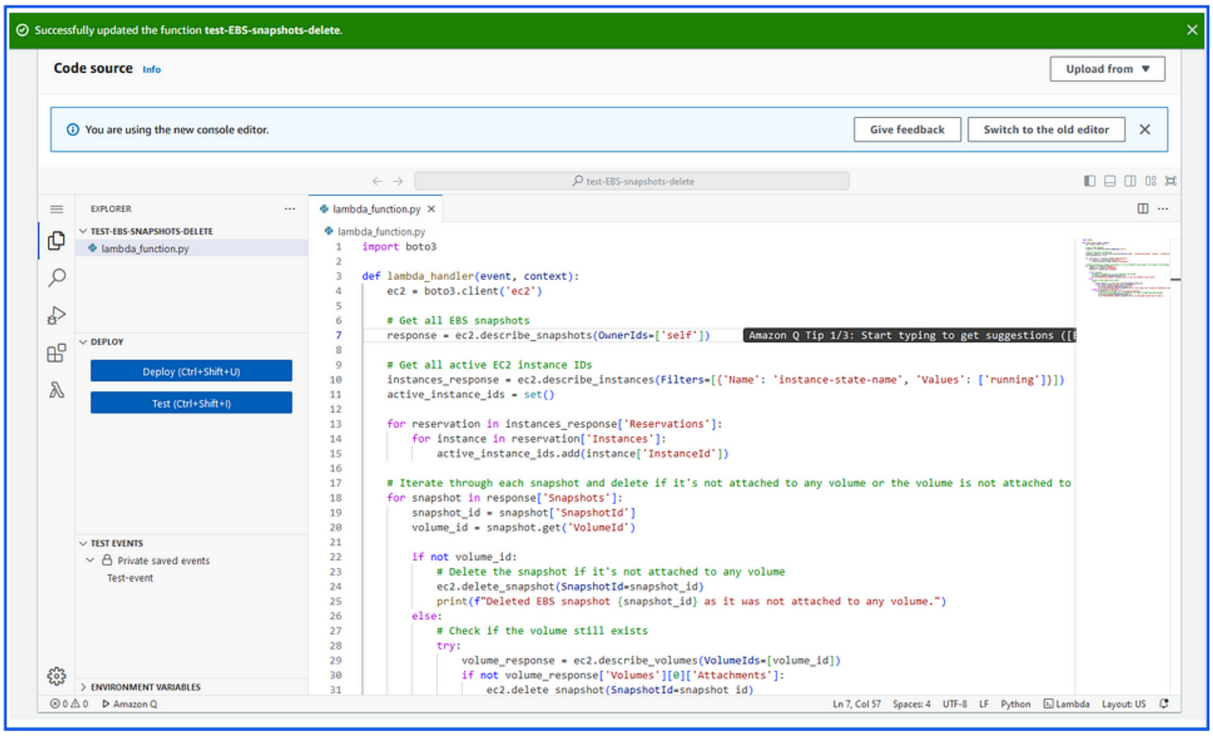


Description
-
Last modified
1 second ago
Function ARN
arn:aws:lambda:ap-south-1:905418425077:function:test-EBS-snapshots-delete
Function URL
-

General configuration

Description
-
Timeout
0 min 10 sec
Memory
128 MB
SnapStart
None
Ephemeral storage
512 MB

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


Successfully updated the function **test-EBS-snapshots-delete**.

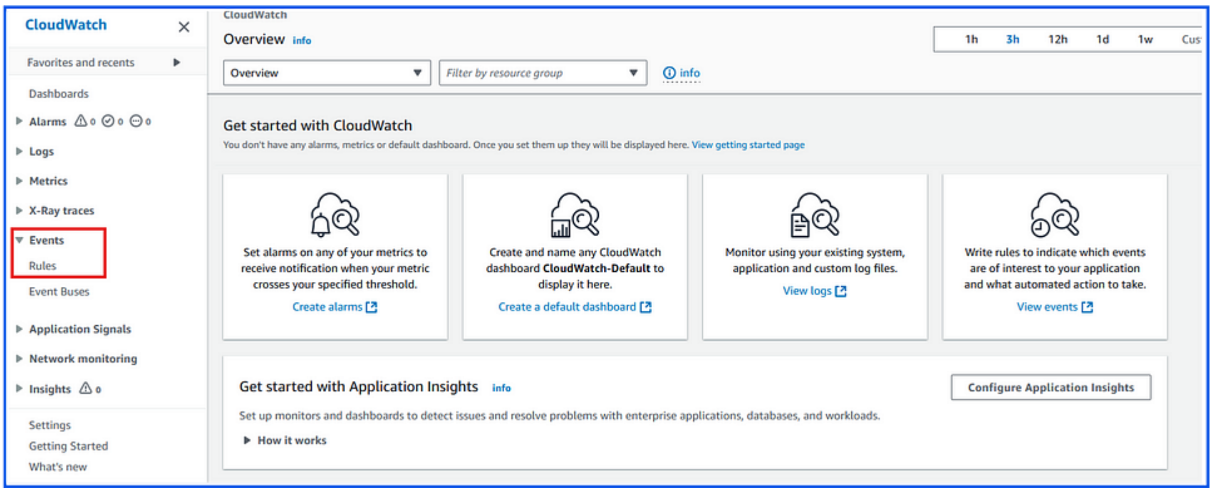
Code source Info

You are using the new console editor.

```python
import boto3

def lambda_handler(event, context):
    ec2 = boto3.client('ec2')

    # Get all EBS snapshots
    response = ec2.describe_snapshots(OwnerIds=['self'])

    # Get all active EC2 instance IDs
    instances_response = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
    active_instance_ids = set()

    for reservation in instances_response['Reservations']:
        for instance in reservation['Instances']:
            active_instance_ids.add(instance['InstanceId'])

    # Iterate through each snapshot and delete if it's not attached to any volume or the volume is not attached to
    for snapshot in response['Snapshots']:
        snapshot_id = snapshot['SnapshotId']
        volume_id = snapshot.get('VolumeId')

        if not volume_id:
            # Delete the snapshot if it's not attached to any volume
            ec2.delete_snapshot(SnapshotId=snapshot_id)
            print(f"Deleted EBS snapshot {snapshot_id} as it was not attached to any volume.")
        else:
            # Check if the volume still exists
            try:
                volume_response = ec2.describe_volumes(VolumeIds=[volume_id])
                if not volume_response['Volumes'][0]['Attachments']:
                    ec2.delete_snapshot(SnapshotId=snapshot_id)
```

Amazon Q Tip 1/3: Start typing to get suggestions

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.



Get started with CloudWatch

You don't have any alarms, metrics or default dashboard. Once you set them up they will be displayed here. View getting started page

Set alarms on any of your metrics to receive notification when your metric crosses your specified threshold.

Create and name any CloudWatch dashboard **CloudWatch-Default** to display it here.

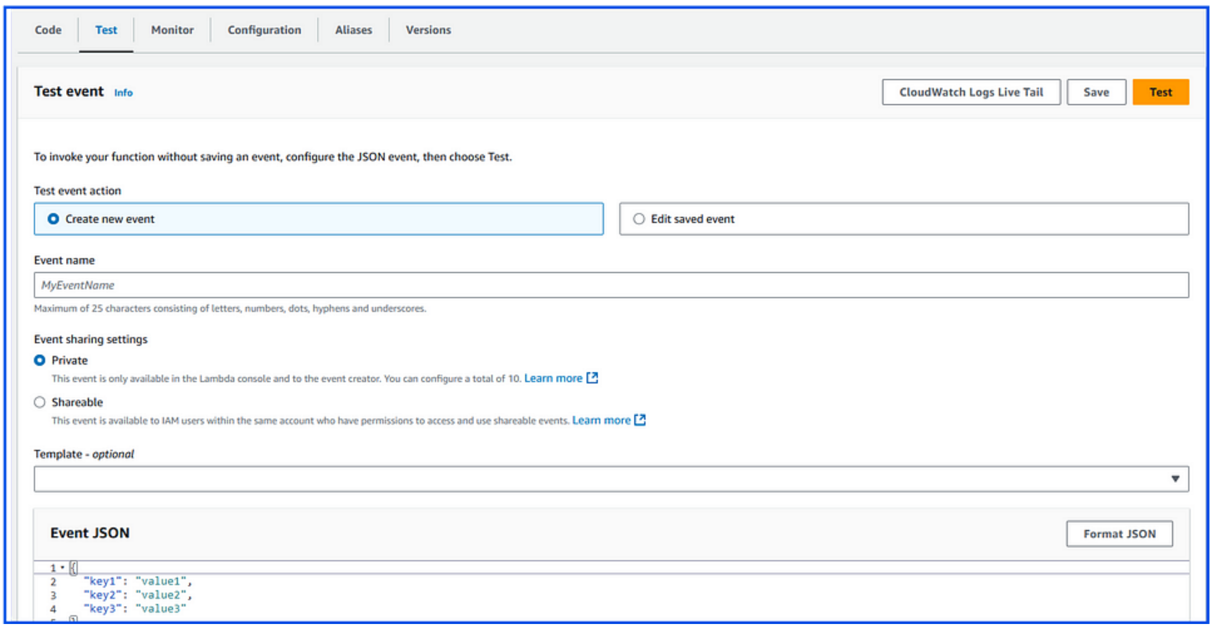
Monitor using your existing system, application and custom log files.

Write rules to indicate which events are of interest to your application and what automated action to take.

Get started with Application Insights info

Set up monitors and dashboards to detect issues and resolve problems with enterprise applications, databases, and workloads.

Creating a manual test for Lambda function:



Test event Info

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

[✓ Create new event] Edit saved event

Event name

MyEventName

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

[✓ Private]

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

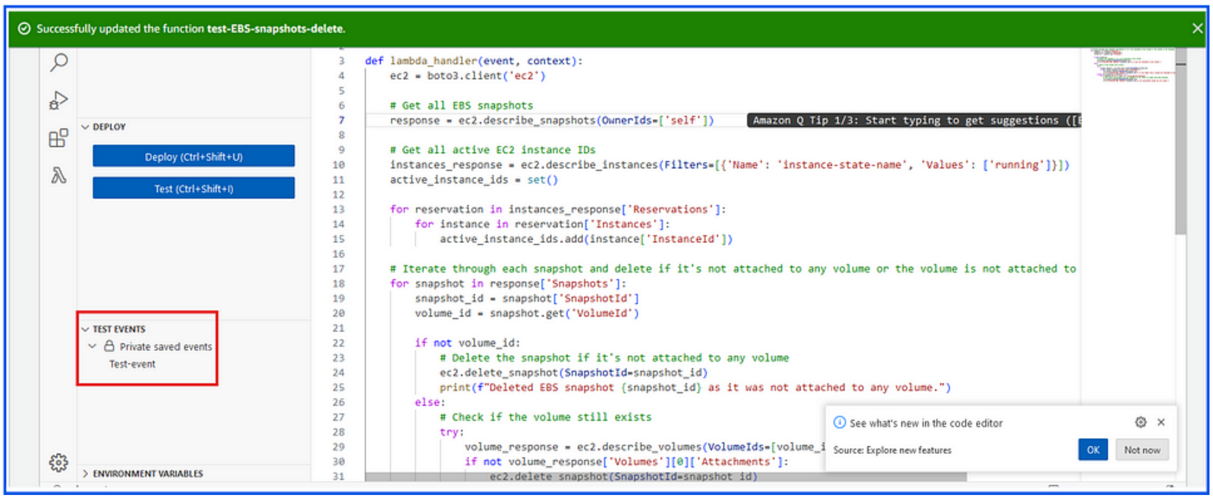
Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

Event JSON

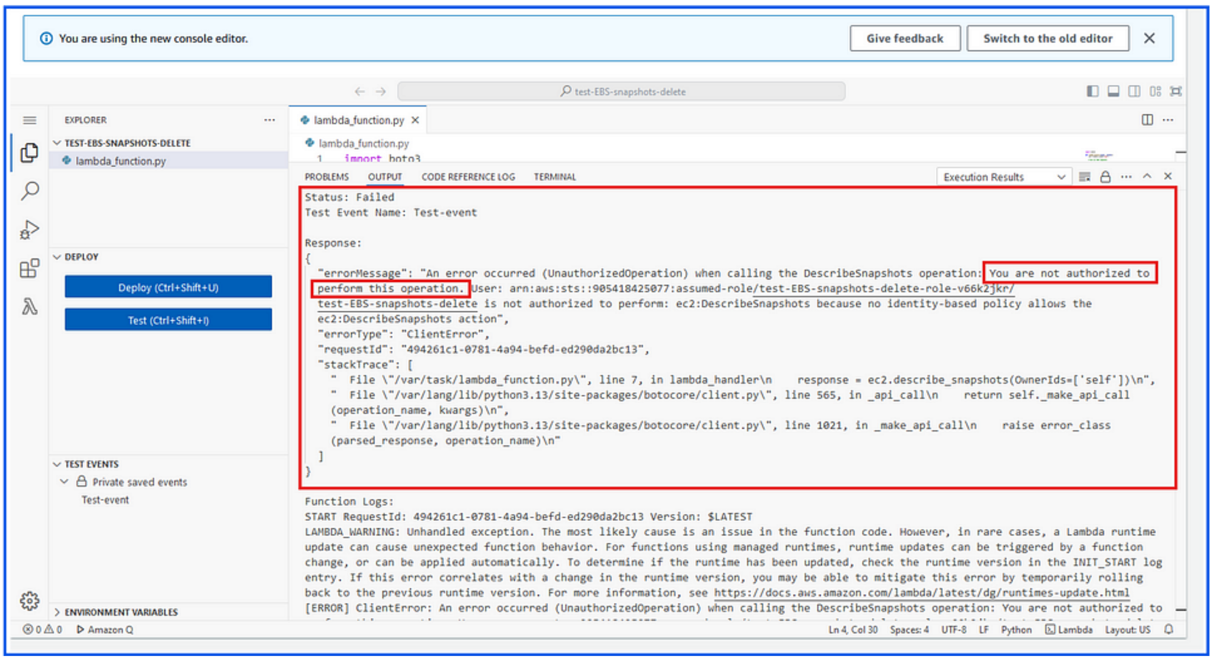
```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
```


Successfully updated the function **test-EBS-snapshots-delete**.

```
def lambda_handler(event, context):
    ec2 = boto3.client('ec2')

    # Get all EBS snapshots
    response = ec2.describe_snapshots(OwnerIds=['self'])    Amazon Q Tip 1/3: Start typing to get suggestions ([

    # Get all active EC2 instance IDs
    instances_response = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
    active_instance_ids = set()

    for reservation in instances_response['Reservations']:
        for instance in reservation['Instances']:
            active_instance_ids.add(instance['InstanceId'])

    # Iterate through each snapshot and delete if it's not attached to any volume or the volume is not attached to
    for snapshot in response['Snapshots']:
        snapshot_id = snapshot['SnapshotId']
        volume_id = snapshot.get('VolumeId')

        if not volume_id:
            # Delete the snapshot if it's not attached to any volume
            ec2.delete_snapshot(SnapshotId=snapshot_id)
            print(f"Deleted EBS snapshot {snapshot_id} as it was not attached to any volume.")
        else:
            # Check if the volume still exists
            try:
                volume_response = ec2.describe_volumes(VolumeIds=[volume_i
                if not volume_response['Volumes'][0]['Attachments']:
                    ec2.delete_snapshot(SnapshotId=snapshot_id)
```

See what's new in the code editor
Source: Explore new features
OK
Not now

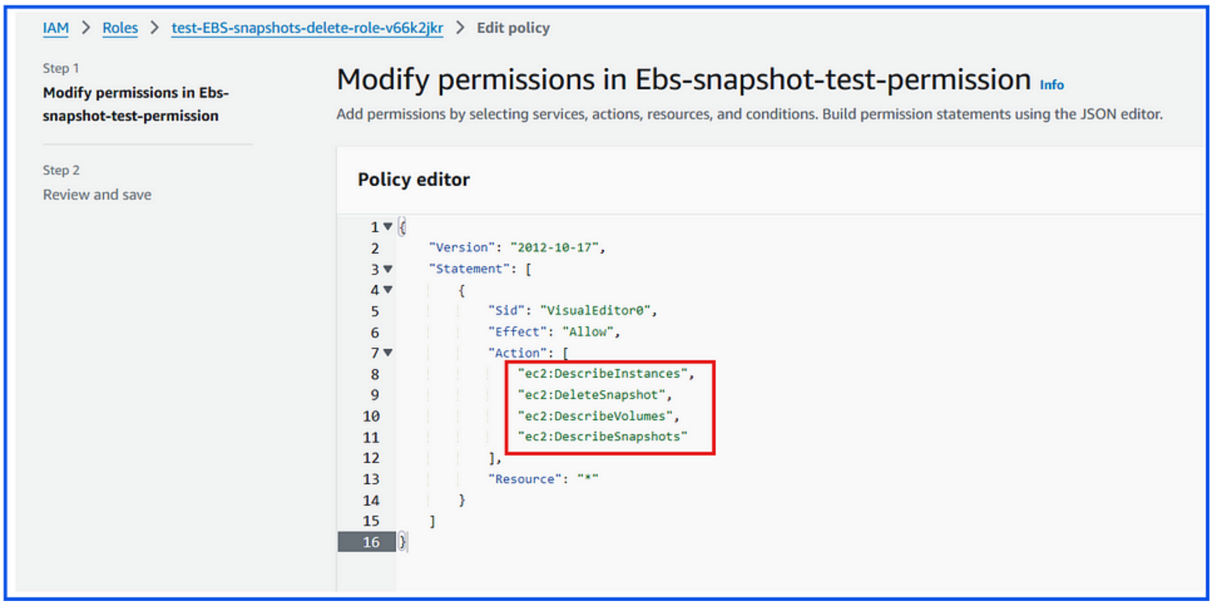
If you test this function without required IAM permissions, it will get failed:



```
Status: Failed
Test Event Name: Test-event

Response:
{
  "errorMessage": "An error occurred (UnauthorizedOperation) when calling the DescribeSnapshots operation: You are not authorized to
  perform this operation. User: arn:aws:sts::905418425077:assumed-role/test-EBS-snapshots-delete-role-v66k2jkr/
  test-EBS-snapshots-delete is not authorized to perform: ec2:DescribeSnapshots because no identity-based policy allows the
  ec2:DescribeSnapshots action",
  "errorType": "ClientError",
  "requestId": "494261c1-0781-4a94-befd-ed290da2bc13",
  "stackTrace": [
    "  File \"/var/task/lambda_function.py\", line 7, in lambda_handler\n    response = ec2.describe_snapshots(OwnerIds=['self'])\n",
    "  File \"/var/lang/lib/python3.13/site-packages/botocore/client.py\", line 565, in _api_call\n    return self._make_api_call
    (operation_name, kwargs)\n",
    "  File \"/var/lang/lib/python3.13/site-packages/botocore/client.py\", line 1021, in _make_api_call\n    raise error_class
    (parsed_response, operation_name)\n"
  ]
}

Function Logs:
START RequestId: 494261c1-0781-4a94-befd-ed290da2bc13 Version: $LATEST
LAMBDA_WARNING: Unhandled exception. The most likely cause is an issue in the function code. However, in rare cases, a Lambda runtime
update can cause unexpected function behavior. For functions using managed runtimes, runtime updates can be triggered by a function
change, or can be applied automatically. To determine if the runtime has been updated, check the runtime version in the INIT_START log
entry. If this error correlates with a change in the runtime version, you may be able to mitigate this error by temporarily rolling
back to the previous runtime version. For more information, see https://docs.aws.amazon.com/lambda/latest/dg/runtimes-update.html
[ERROR] ClientError: An error occurred (UnauthorizedOperation) when calling the DescribeSnapshots operation: You are not authorized to
```

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.

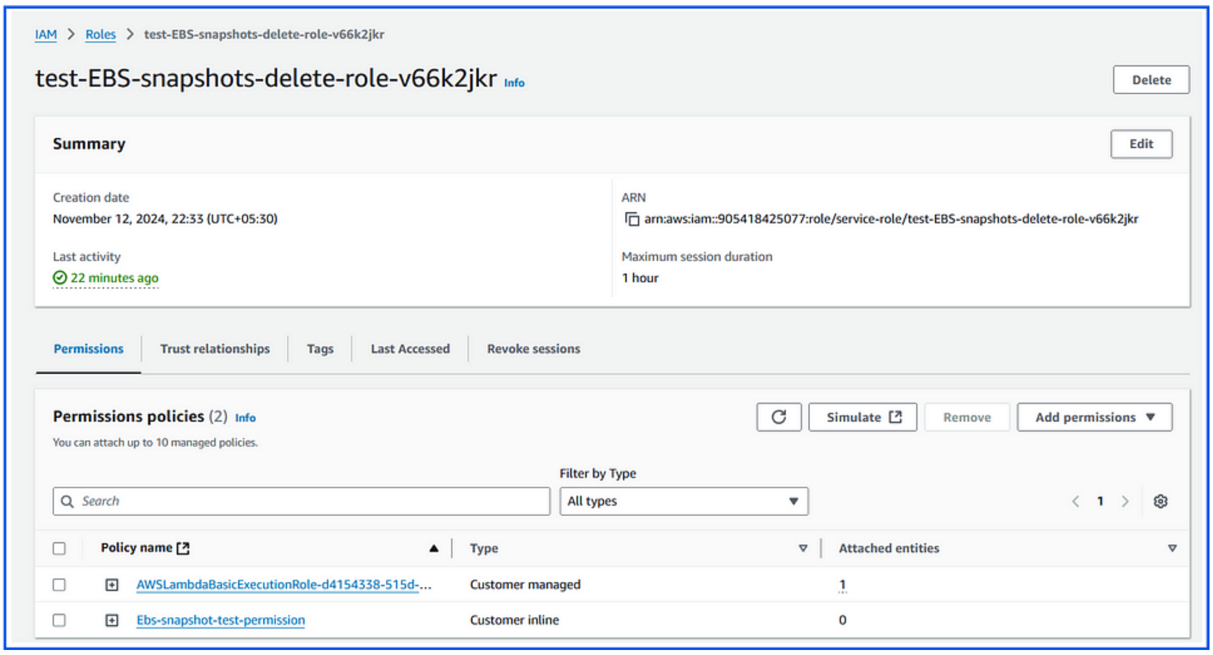

IAM > Roles > test-EBS-snapshots-delete-role-v66k2jkr > Edit policy
Step 1
Modify permissions in Ebs-snapshot-test-permission
Step 2
Review and save

Modify permissions in Ebs-snapshot-test-permission Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DeleteSnapshot",
                "ec2:DescribeVolumes",
                "ec2:DescribeSnapshots"
            ],
            "Resource": "*"
        }
    ]
}
```



IAM > Roles > test-EBS-snapshots-delete-role-v66k2jkr

test-EBS-snapshots-delete-role-v66k2jkr Info

Summary

Creation date
November 12, 2024, 22:33 (UTC+05:30)

Last activity
22 minutes ago

ARN
arn:aws:iam::905418425077:role/service-role/test-EBS-snapshots-delete-role-v66k2jkr

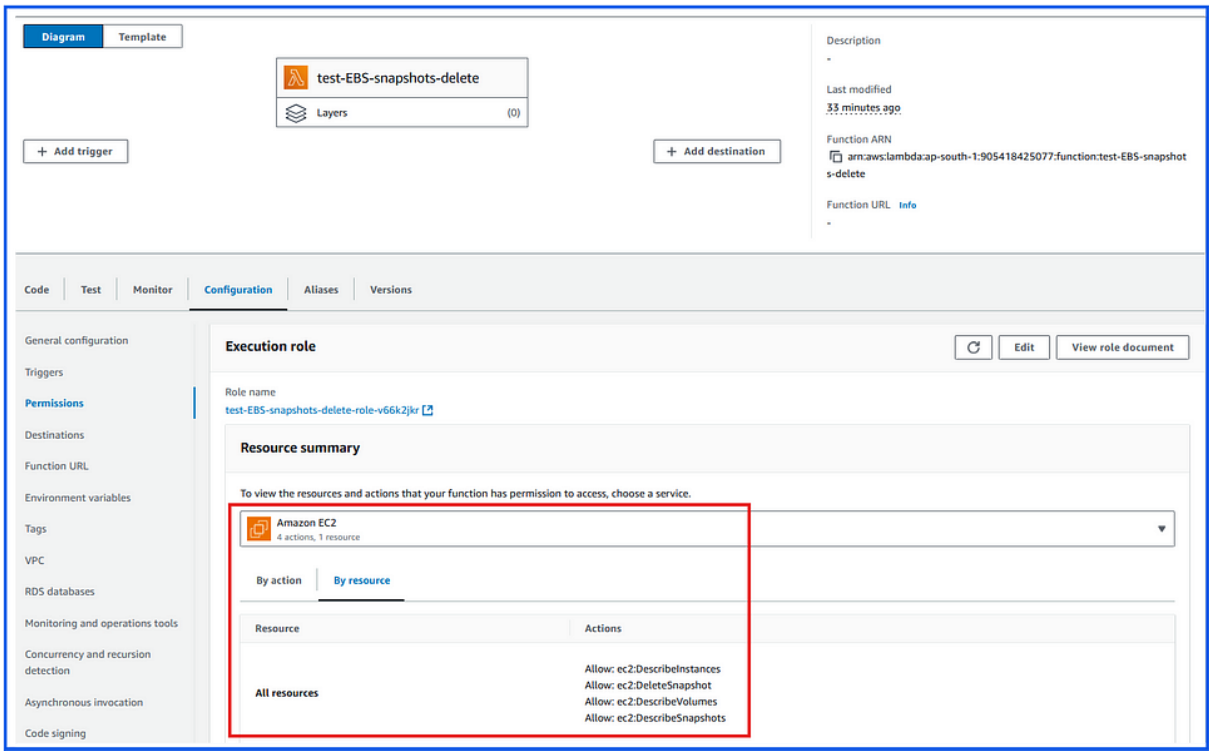
Maximum session duration
1 hour

Permissions policies (2) Info
You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
AWSLambdaBasicExecutionRole-d4154338-515d-...	Customer managed	1
Ebs-snapshot-test-permission	Customer inline	0

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



test-EBS-snapshots-delete
Layers (0)

Description
-

Last modified
33 minutes ago

Function ARN
arn:aws:lambda:ap-south-1:905418425077:function:test-EBS-snapshots-delete

Function URL Info
-

Execution role

Role name
test-EBS-snapshots-delete-role-v66k2jkr

Resource summary

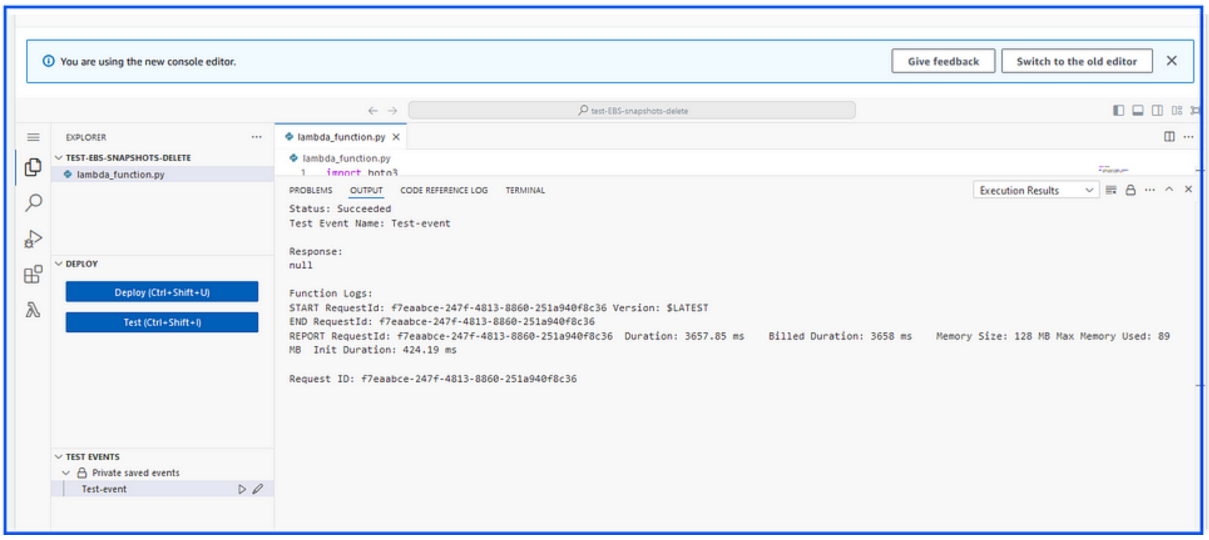
To view the resources and actions that your function has permission to access, choose a service.

Amazon EC2
4 actions, 1 resource

Resource	Actions
All resources	Allow: ec2:DescribeInstances Allow: ec2:DeleteSnapshot Allow: ec2:DescribeVolumes Allow: ec2:DescribeSnapshots

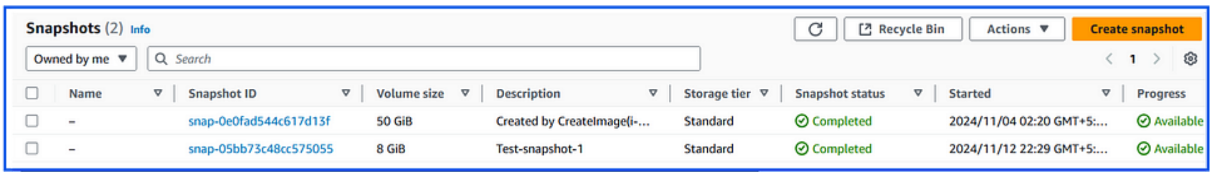
Now we will test the lambda function again:

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

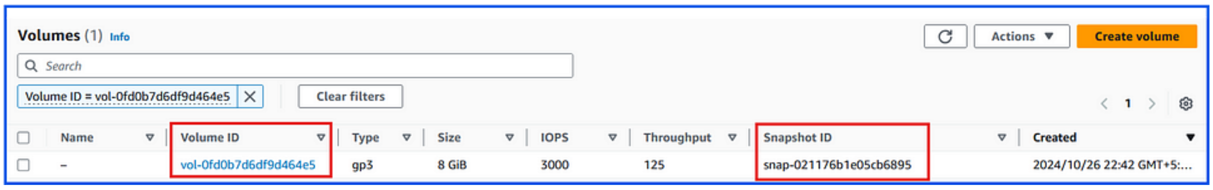


```
import boto3
```

```
Status: Succeeded
Test Event Name: Test-event

Response:
null

Function Logs:
START RequestId: f7eaabce-247f-4813-8860-251a940f8c36 Version: $LATEST
END RequestId: f7eaabce-247f-4813-8860-251a940f8c36
REPORT RequestId: f7eaabce-247f-4813-8860-251a940f8c36  Duration: 3657.85 ms    Billed Duration: 3658 ms    Memory Size: 128 MB Max Memory Used: 89
MB  Init Duration: 424.19 ms

Request ID: f7eaabce-247f-4813-8860-251a940f8c36
```

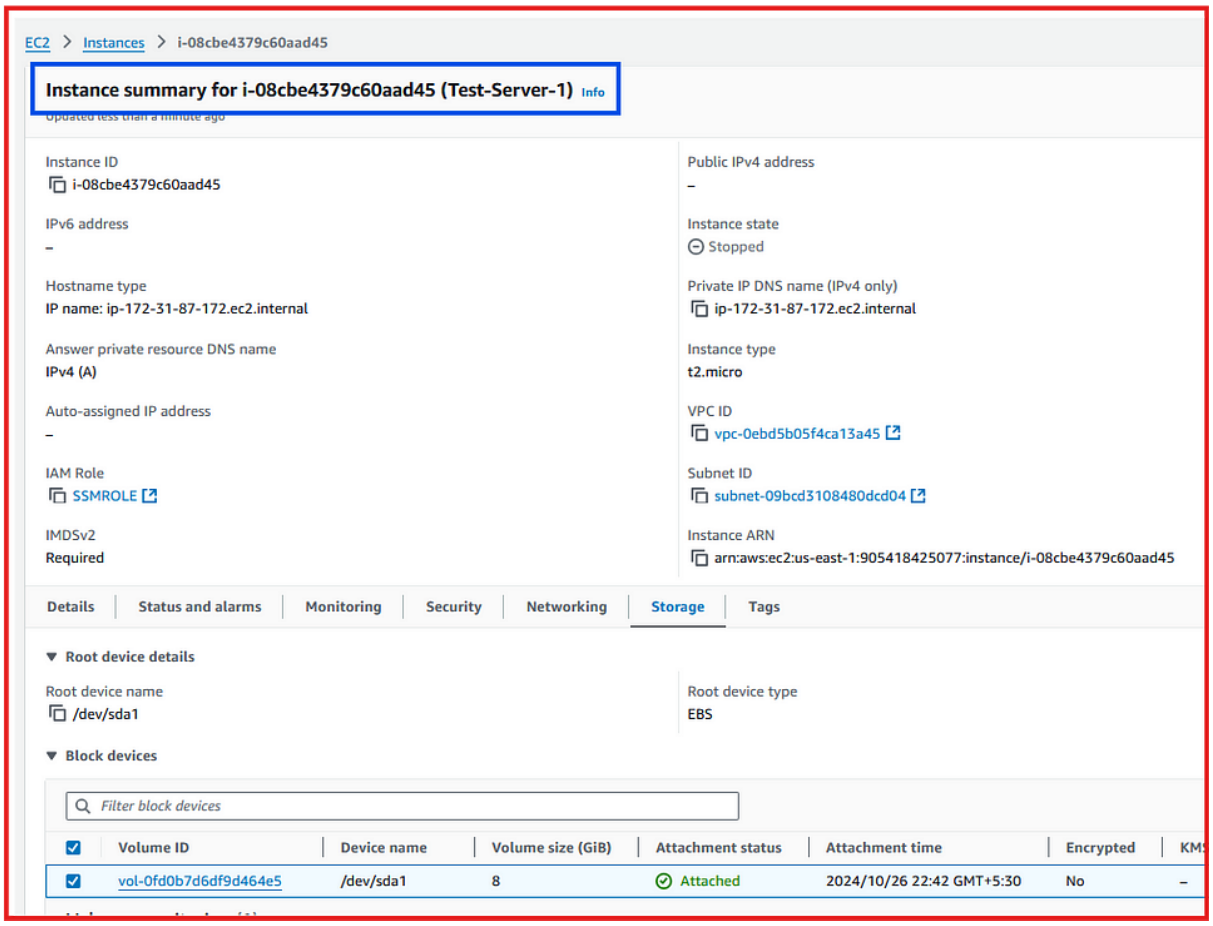


Name	Snapshot ID	Volume size	Description	Storage tier	Snapshot status	Started	Progress
–	snap-0e0fad544c617d13f	50 GiB	Created by CreateImage(i-...	Standard	Completed	2024/11/04 02:20 GMT+5:...	Available
–	snap-05bb73c48cc575055	8 GiB	Test-snapshot-1	Standard	Completed	2024/11/12 22:29 GMT+5:...	Available



Name	Volume ID	Type	Size	IOPS	Throughput	Snapshot ID	Created
–	vol-0fd0b7d6df9d464e5	gp3	8 GiB	3000	125	snap-021176b1e05cb6895	2024/10/26 22:42 GMT+5:...

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



EC2 > Instances > i-08cbe4379c60aad45

Instance summary for i-08cbe4379c60aad45 (Test-Server-1) Info

Updated less than a minute ago

Instance ID i-08cbe4379c60aad45	Public IPv4 address –
IPv6 address –	Instance state Stopped
Hostname type IP name: ip-172-31-87-172.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-87-172.ec2.internal
Answer private resource DNS name IPv4 (A)	Instance type t2.micro
Auto-assigned IP address –	VPC ID vpc-0ebd5b05f4ca13a45
IAM Role SSMROLE	Subnet ID subnet-09bcd3108480dcd04
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:905418425077:instance/i-08cbe4379c60aad45

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

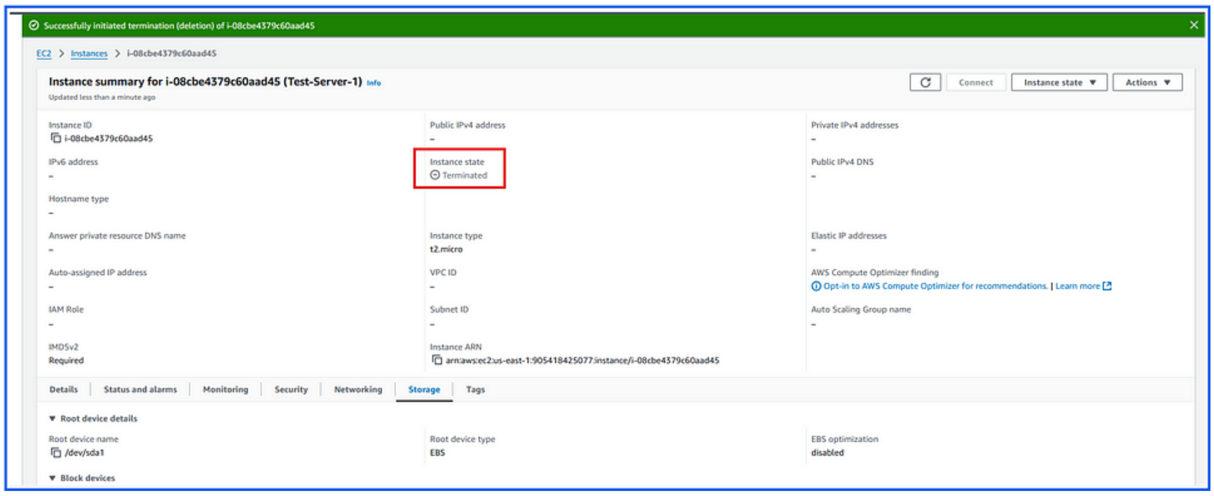
▼ Root device details

Root device name /dev/sda1	Root device type EBS

▼ Block devices

Filter block devices

✓	Volume ID	Device name	Volume size (GiB)	Attachment status	Attachment time	Encrypted	KMS
✓	vol-0fd0b7d6df9d464e5	/dev/sda1	8	Attached	2024/10/26 22:42 GMT+5:30	No	–



Successfully initiated termination (deletion) of i-08cbe4379c60aad45

EC2 > Instances > i-08cbe4379c60aad45

Instance summary for i-08cbe4379c60aad45 (Test-Server-1) Info

Updated less than a minute ago

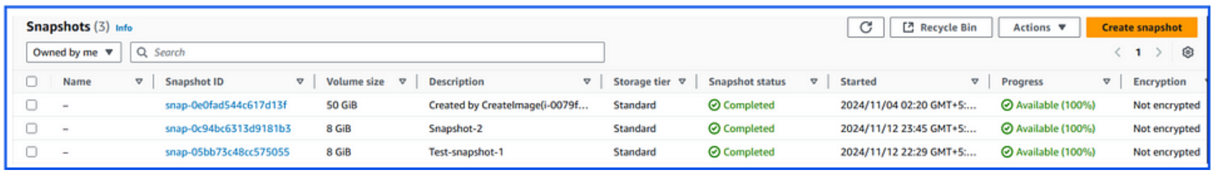
Instance ID i-08cbe4379c60aad45	Public IPv4 address –	Private IPv4 addresses –
IPv6 address –	Instance state Terminated	Public IPv4 DNS –
Hostname type –		
Answer private resource DNS name –	Instance type t2.micro	Elastic IP addresses –
Auto-assigned IP address –	VPC ID –	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role –	Subnet ID –	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:905418425077:instance/i-08cbe4379c60aad45	

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

▼ Root device details

Root device name /dev/sda1	Root device type EBS	EBS optimization disabled

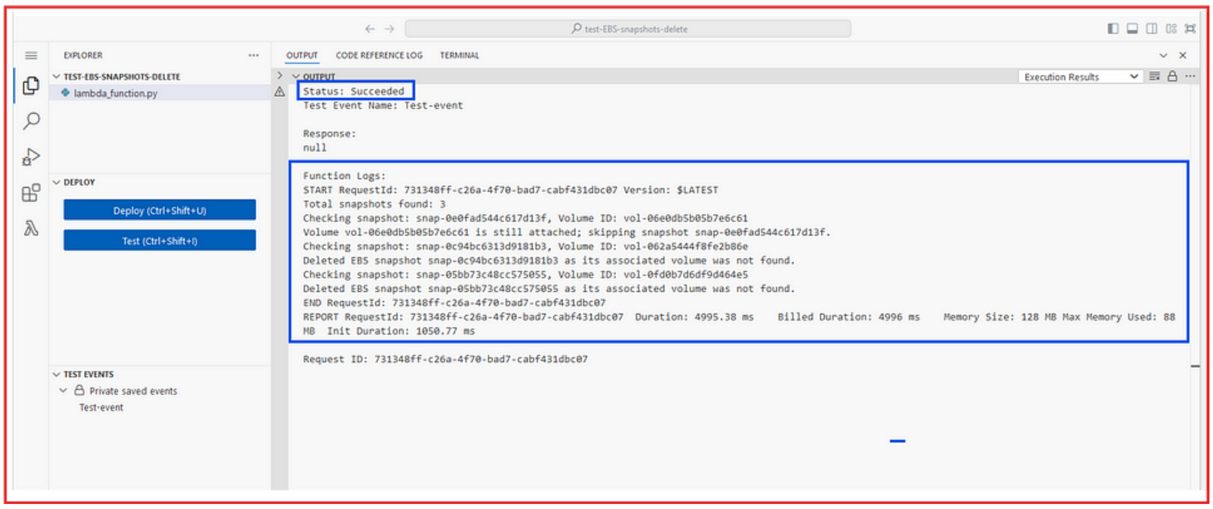
▼ Block devices



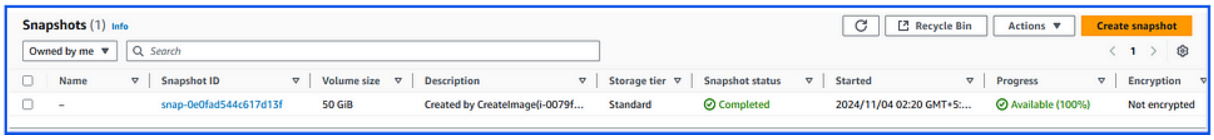
Snapshots (3) Info

Name	Snapshot ID	Volume size	Description	Storage tier	Snapshot status	Started	Progress	Encryption
–	snap-0e0fad544c617d13f	50 GiB	Created by CreateImage(i-0079f...	Standard	Completed	2024/11/04 02:20 GMT+5:...	Available (100%)	Not encrypted
–	snap-0c94bc6313d9181b3	8 GiB	Snapshot-2	Standard	Completed	2024/11/12 23:45 GMT+5:...	Available (100%)	Not encrypted
–	snap-05bb73c48cc575055	8 GiB	Test-snapshot-1	Standard	Completed	2024/11/12 22:29 GMT+5:...	Available (100%)	Not encrypted

Now run the Test event:



```
Status: Succeeded
Test Event Name: Test-event

Response:
null

Function Logs:
START RequestId: 731348ff-c26a-4f70-bad7-cabf431dbc07 Version: $LATEST
Total snapshots found: 3
Checking snapshot: snap-0e0fad544c617d13f, Volume ID: vol-06e0db5b05b7e6c61
Volume vol-06e0db5b05b7e6c61 is still attached; skipping snapshot snap-0e0fad544c617d13f.
Checking snapshot: snap-0c94bc6313d9181b3, Volume ID: vol-062a5444f8fe2b86e
Deleted EBS snapshot snap-0c94bc6313d9181b3 as its associated volume was not found.
Checking snapshot: snap-05bb73c48cc575055, Volume ID: vol-0fd0b7d6df9d464e5
Deleted EBS snapshot snap-05bb73c48cc575055 as its associated volume was not found.
END RequestId: 731348ff-c26a-4f70-bad7-cabf431dbc07
REPORT RequestId: 731348ff-c26a-4f70-bad7-cabf431dbc07  Duration: 4995.38 ms    Billed Duration: 4996 ms    Memory Size: 128 MB Max Memory Used: 88
MB  Init Duration: 1050.77 ms

Request ID: 731348ff-c26a-4f70-bad7-cabf431dbc07
```

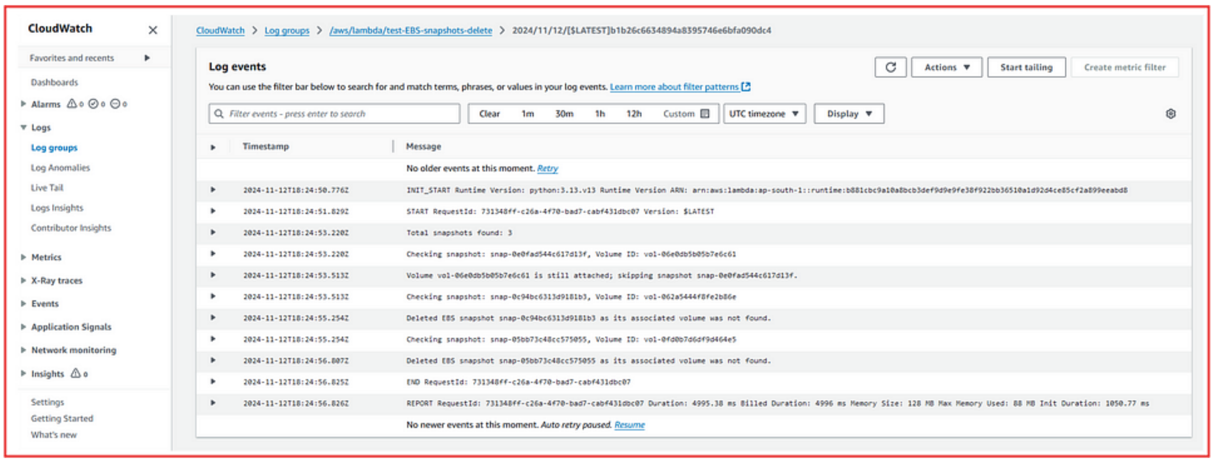


Snapshots (1)

Name	Snapshot ID	Volume size	Description	Storage tier	Snapshot status	Started	Progress	Encryption
–	snap-0e0fad544c617d13f	50 GiB	Created by CreateImage(i-0079f...	Standard	Completed	2024/11/04 02:20 GMT+5:...	Available (100%)	Not encrypted

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:



CloudWatch > Log groups > /aws/lambda/test-EBS-snapshots-delete > 2024/11/12/[$LATEST]b1b26c6634894a8395746e6bfa090dc4

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                  Message
                           No older events at this moment. Retry
2024-11-12T18:24:50.776Z   INIT_START Runtime Version: python:3.13.v13 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:b881cbc9a10a8bcb3def9d9e9fe38f922bb36510a1d92d4ce85cf2a899eeabd8
2024-11-12T18:24:51.829Z   START RequestId: 731348ff-c26a-4f70-bad7-cabf431dbc07 Version: $LATEST
2024-11-12T18:24:53.220Z   Total snapshots found: 3
2024-11-12T18:24:53.220Z   Checking snapshot: snap-0e0fad544c617d13f, Volume ID: vol-06e0db5b05b7e6c61
2024-11-12T18:24:53.513Z   Volume vol-06e0db5b05b7e6c61 is still attached; skipping snapshot snap-0e0fad544c617d13f.
2024-11-12T18:24:53.513Z   Checking snapshot: snap-0c94bc6313d9181b3, Volume ID: vol-062a5444f8fe2b86e
2024-11-12T18:24:55.254Z   Deleted EBS snapshot snap-0c94bc6313d9181b3 as its associated volume was not found.
2024-11-12T18:24:55.254Z   Checking snapshot: snap-05bb73c48cc575055, Volume ID: vol-0fd0b7d6df9d464e5
2024-11-12T18:24:56.807Z   Deleted EBS snapshot snap-05bb73c48cc575055 as its associated volume was not found.
2024-11-12T18:24:56.825Z   END RequestId: 731348ff-c26a-4f70-bad7-cabf431dbc07
2024-11-12T18:24:56.826Z   REPORT RequestId: 731348ff-c26a-4f70-bad7-cabf431dbc07 Duration: 4995.38 ms Billed Duration: 4996 ms Memory Size: 128 MB Max Memory Used: 88 MB Init Duration: 1050.77 ms
                           No newer events at this moment. Auto retry paused. Resume
```

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

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