

## Purpose:

Our project aims to develop a Lambda function tailored to optimize storage costs within the AWS environment. By leveraging this function, we will systematically identify EBS snapshots that have become disassociated from active EC2 instances and subsequently execute their deletion. This initiative is driven by the goal of reducing unnecessary storage overheads and fostering a more efficient resource management strategy within the AWS Cloud infrastructure. Through the implementation of this Lambda function, we anticipate significant cost savings while maintaining operational agility and scalability.

## Python script for the lambda function (base script, can be modified as per usecase):

```
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 a running instance
    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)

            print(f"Deleted EBS snapshot {snapshot_id} as it was taken from a volume not attached to
any running instance.")

    except ec2.exceptions.ClientError as e:

        if e.response['Error']['Code'] == 'InvalidVolume.NotFound':

            # The volume associated with the snapshot is not found (it might have been deleted)

            ec2.delete_snapshot(SnapshotId=snapshot_id)

            print(f"Deleted EBS snapshot {snapshot_id} as its associated volume was not found.")

```

Function name

Enter a name that describes the purpose of your function.

cost-optimization-project

Use only letters, numbers, hyphens, or underscores with no spaces.

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 3.12

Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

☒ x86\_64
☐ arm64

Permissions [Info](#)

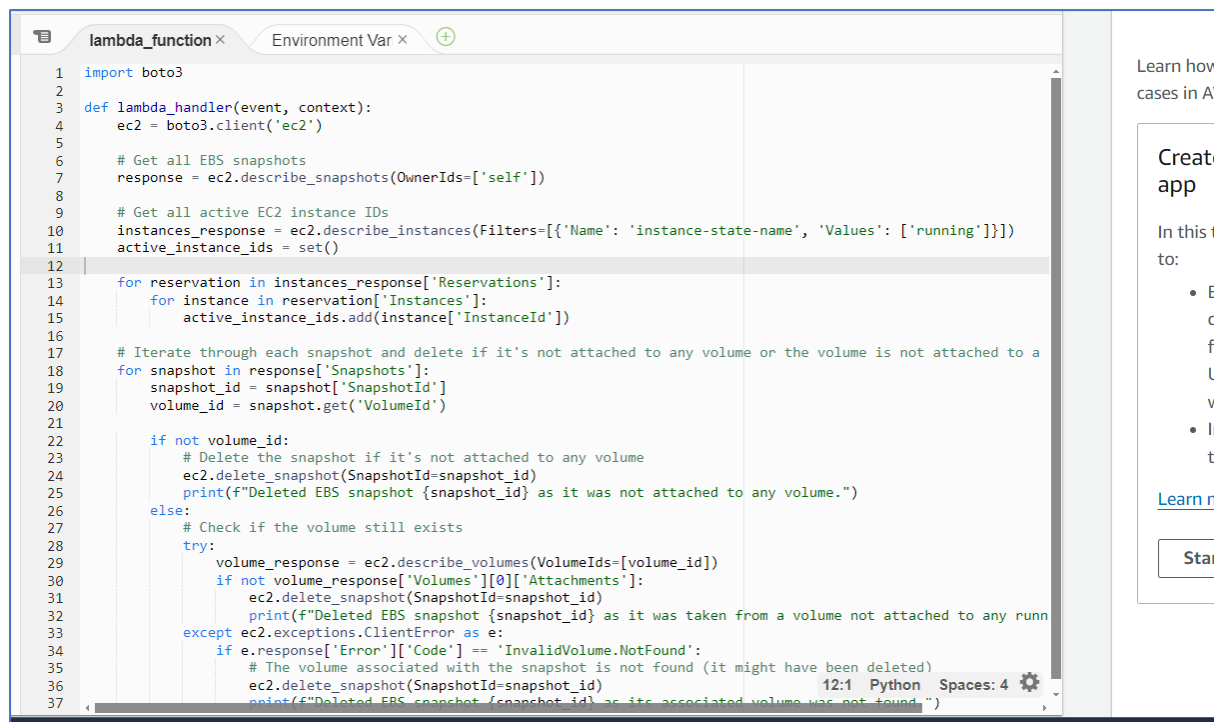
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► Change default execution role

► Advanced settings

Cancel

Create function



```
1 import boto3
2
3 def lambda_handler(event, context):
4     ec2 = boto3.client('ec2')
5
6     # Get all EBS snapshots
7     response = ec2.describe_snapshots(OwnerIds=['self'])
8
9     # Get all active EC2 instance IDs
10    instances_response = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
11    active_instance_ids = set()
12
13    for reservation in instances_response['Reservations']:
14        for instance in reservation['Instances']:
15            active_instance_ids.add(instance['InstanceId'])
16
17    # Iterate through each snapshot and delete if it's not attached to any volume or the volume is not attached to a
18    for snapshot in response['Snapshots']:
19        snapshot_id = snapshot['SnapshotId']
20        volume_id = snapshot.get('VolumeId')
21
22        if not volume_id:
23            # Delete the snapshot if it's not attached to any volume
24            ec2.delete_snapshot(SnapshotId=snapshot_id)
25            print(f"Deleted EBS snapshot {snapshot_id} as it was not attached to any volume.")
26        else:
27            # Check if the volume still exists
28            try:
29                volume_response = ec2.describe_volumes(VolumeIds=[volume_id])
30                if not volume_response['Volumes'][0]['Attachments']:
31                    ec2.delete_snapshot(SnapshotId=snapshot_id)
32                    print(f"Deleted EBS snapshot {snapshot_id} as it was taken from a volume not attached to any running instance.")
33            except ec2.exceptions.ClientError as e:
34                if e.response['Error']['Code'] == 'InvalidVolume.NotFound':
35                    # The volume associated with the snapshot is not found (it might have been deleted)
36                    ec2.delete_snapshot(SnapshotId=snapshot_id)
37                    print(f"Deleted EBS snapshot {snapshot_id} as its associated volume was not found.")
```

## Explanation of the code:

**Importing boto3:** boto3 is the AWS SDK for Python, which allows Python developers to write software that makes use of services like Amazon S3 and Amazon EC2.

**Defining the lambda\_handler function:** This is the entry point for the Lambda function. It takes two parameters: event and context. In this code, only the context parameter is being used.

**Creating an EC2 client:** The code initializes an EC2 client using `boto3.client('ec2')`. This client will be used to interact with EC2 resources in AWS.

**Getting all EBS snapshots:** The code calls `ec2.describe_snapshots(OwnerIds=['self'])` to retrieve information about all EBS snapshots that are owned by the current AWS account ('self').

**Getting all active EC2 instance IDs:** The code calls `ec2.describe_instances()` with a filter to get information about all running EC2 instances. It then extracts the instance IDs and stores them in a set called `active_instance_ids`.

Iterating through each snapshot: The code loops through each snapshot obtained in step 4. For each snapshot, it checks if it's attached to any volume.

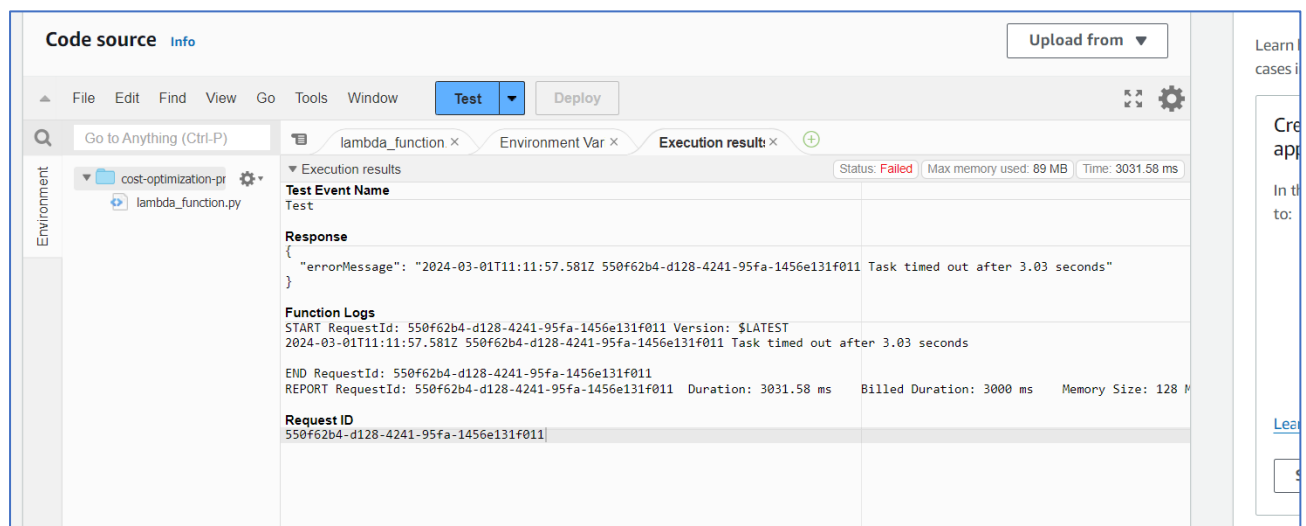
a. If the snapshot is not attached to any volume (volume\_id is None), it deletes the snapshot using `ec2.delete_snapshot(SnapshotId=snapshot_id)` and prints a message indicating that the snapshot was deleted because it was not attached to any volume.

b. If the snapshot is attached to a volume, it checks if the volume still exists by calling `ec2.describe_volumes(VolumeIds=[volume_id])`. If the volume exists but is not attached to any running instance, it deletes the snapshot and prints a message indicating that the snapshot was deleted because it was taken from a volume not attached to any running instance.

c. If the volume associated with the snapshot is not found (possibly deleted), it catches the `InvalidVolume.NotFound` exception, deletes the snapshot, and prints a message indicating that the snapshot was deleted because its associated volume was not found.

## Challenges faced during the project :

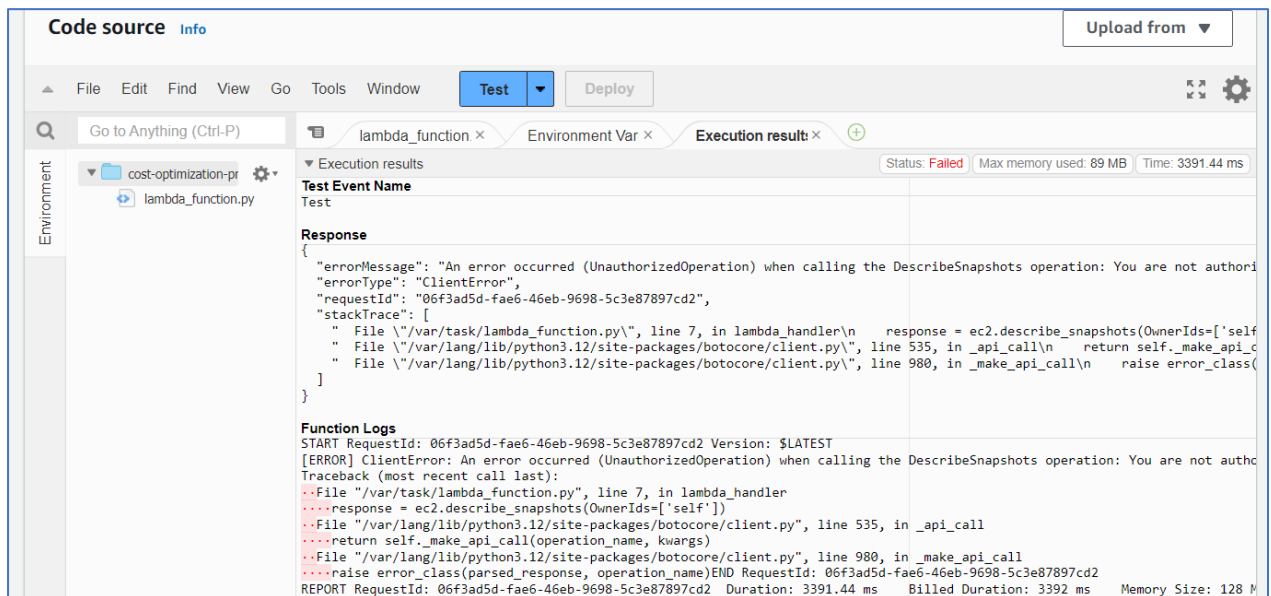
### 1) Lambda function execution time out error :



## Solution :

Increased the lambda execution time from 3 seconds to 10 seconds (for demo purposes, can vary according to use case)

## 2) Lambda UNAUTHORIZED EXCEPTION :



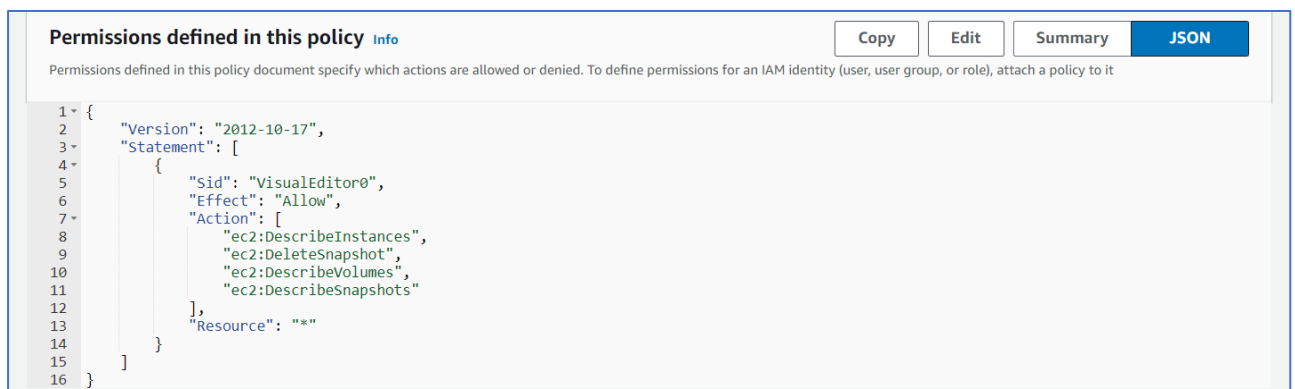
The screenshot shows the AWS Lambda console interface. The 'Execution results' tab is selected, displaying a 'Failed' status. The 'Test Event Name' is 'Test'. The 'Response' section shows an 'UnauthorizedOperation' error. The 'Function Logs' section shows the error message and a traceback.

```
Response
{
  "errorMessage": "An error occurred (UnauthorizedOperation) when calling the DescribeSnapshots operation: You are not authorized to perform this operation.",
  "errorType": "ClientError",
  "requestId": "06f3ad5d-fae6-46eb-9698-5c3e87897cd2",
  "stackTrace": [
    "File \"/var/task/lambda_function.py\", line 7, in lambda_handler\n    response = ec2.describe_snapshots(OwnerIds=['self'])\n    File \"/var/lang/lib/python3.12/site-packages/botocore/client.py\", line 535, in _api_call\n    return self._make_api_call(operation_name, kwargs)\n    File \"/var/lang/lib/python3.12/site-packages/botocore/client.py\", line 980, in _make_api_call\n    raise error_class(parsed_response, operation_name)\n  ]
}
```

```
Function Logs
START RequestId: 06f3ad5d-fae6-46eb-9698-5c3e87897cd2 Version: $LATEST
[ERROR] ClientError: An error occurred (UnauthorizedOperation) when calling the DescribeSnapshots operation: You are not authorized to perform this operation.
Traceback (most recent call last):
  File "/var/task/lambda_function.py", line 7, in lambda_handler
    response = ec2.describe_snapshots(OwnerIds=['self'])
  File "/var/lang/lib/python3.12/site-packages/botocore/client.py", line 535, in _api_call
    return self._make_api_call(operation_name, kwargs)
  File "/var/lang/lib/python3.12/site-packages/botocore/client.py", line 980, in _make_api_call
    raise error_class(parsed_response, operation_name)
END RequestId: 06f3ad5d-fae6-46eb-9698-5c3e87897cd2
REPORT RequestId: 06f3ad5d-fae6-46eb-9698-5c3e87897cd2 Duration: 3391.44 ms Billed Duration: 3392 ms Memory Size: 128 MB
```

## Solution:

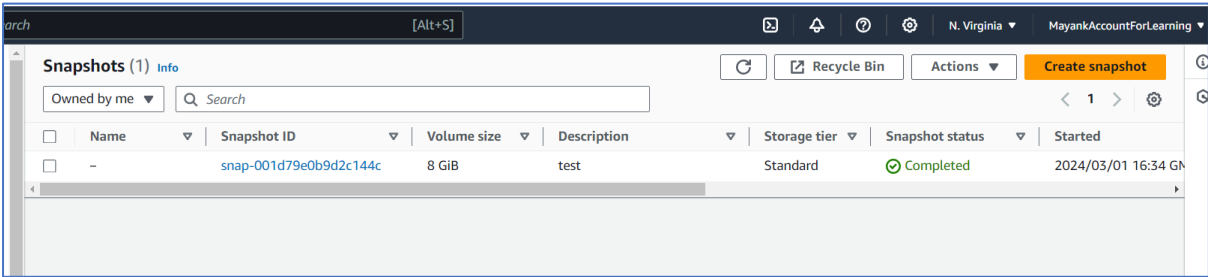
To assign the necessary permissions to the role associated with the lambda function, Here, I included DescribeSnapshots, DescribeInstances, DeleteSnapshots, DescribeVolumes to the policy attached to the role.



The screenshot shows the 'Permissions defined in this policy' section of the AWS IAM console. The policy is named 'VisualEditor0'. The permissions are defined in a JSON document.

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "VisualEditor0",
6       "Effect": "Allow",
7       "Action": [
8         "ec2:DescribeInstances",
9         "ec2:DeleteSnapshot",
10        "ec2:DescribeVolumes",
11        "ec2:DescribeSnapshots"
12      ],
13      "Resource": "*"
14    }
15  ]
16 }
```

Final Result:



Above snapshot got deleted as it was not associated with any running EC2 instance

