AWS Cost Optimization Report: Automated Snapshot Deletion upon EC2 Instance Termination

Name: Chaithanya M

Date: July 8, 2025

Topic: Automated Snapshot Deletion upon EC2 Instance Termination

1. Introduction

In AWS, cost optimization is crucial to maintaining an efficient and scalable cloud infrastructure. One common area where hidden costs accumulate is in EBS snapshots, especially when they are left behind after EC2 instances are terminated. This report outlines a solution involving a Lambda function that automatically deletes EBS snapshots if the corresponding EC2 instance has been deleted.

2. Architecture Overview

- EC2 Instance Creation: An EC2 instance is launched.
- Snapshot Creation: A snapshot of the instance's attached EBS volume is created.
- Instance Termination: The EC2 instance is deleted, but the snapshot persists.
- Lambda Automation: A Lambda function detects orphaned snapshots and deletes them.

3. Initial Issue

1. Create an EC2 instance.
2. Create a snapshot of its EBS volume.
3. Terminate the EC2 instance.

However, the snapshot remained even after instance deletion. This led to unnecessary snapshot storage charges.

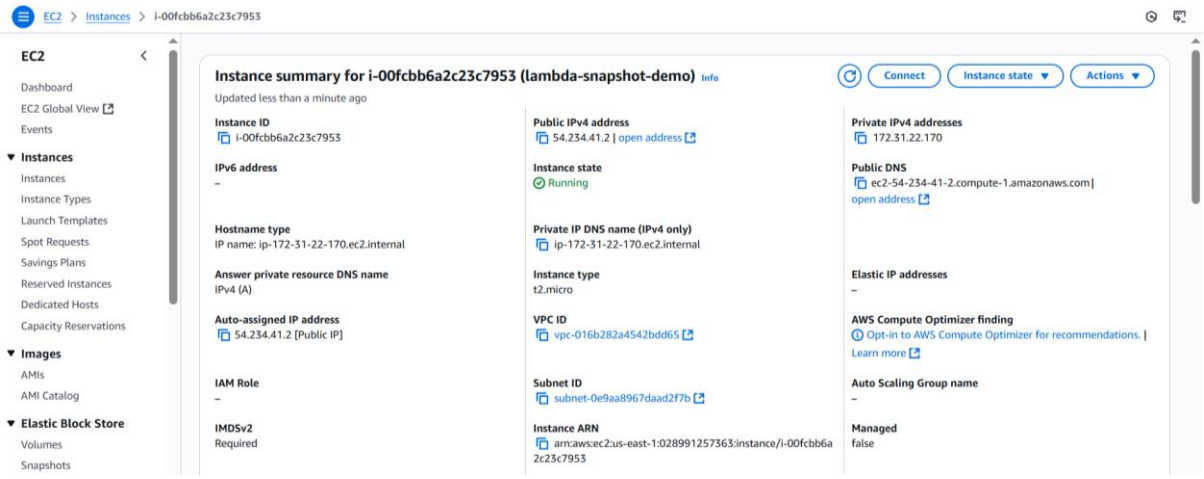
4. Solution: Lambda-Based Clean-up Function

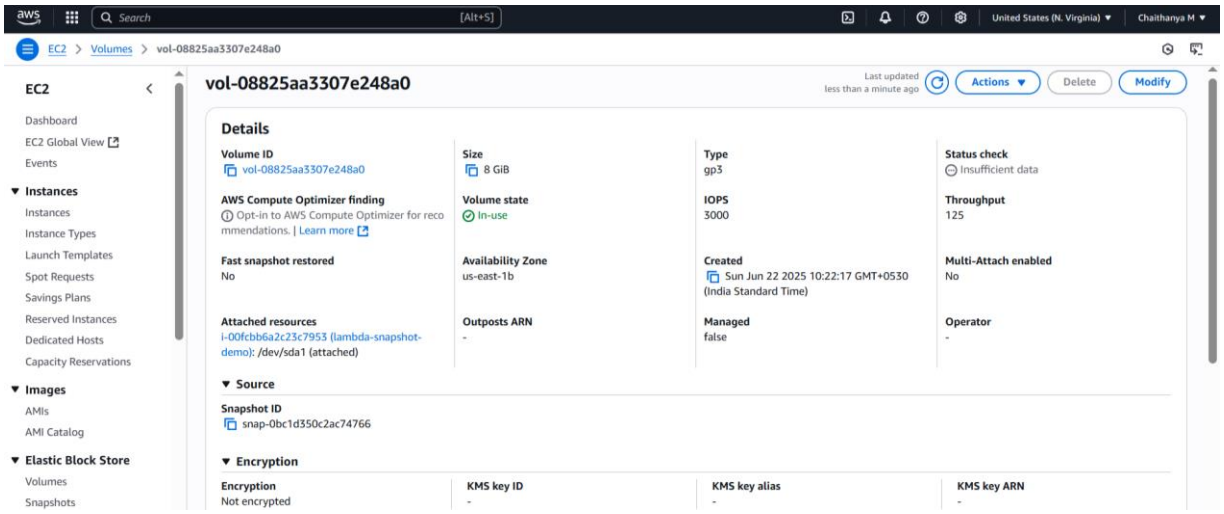
- Periodically check all EBS snapshots.
- Identify if the instance associated with the snapshot no longer exists.
- Automatically delete orphaned snapshots.

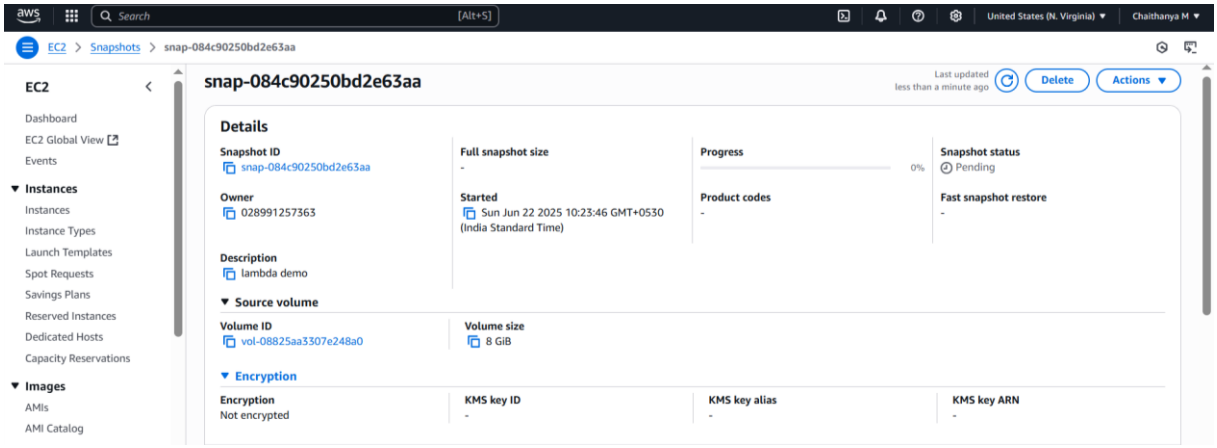
5. Implementation

Step 1: Launch EC2 Instance

- Go to EC2 Dashboard → Instances → Launch Instances
- Select: AMI: Ubuntu, Instance type: t2.micro (Free Tier eligible)
- Click Launch

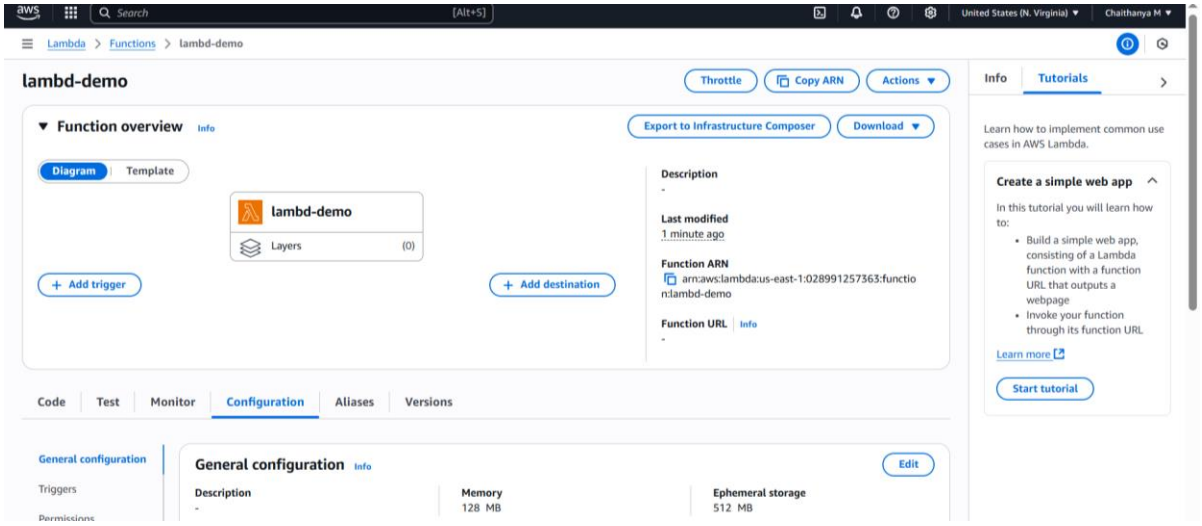






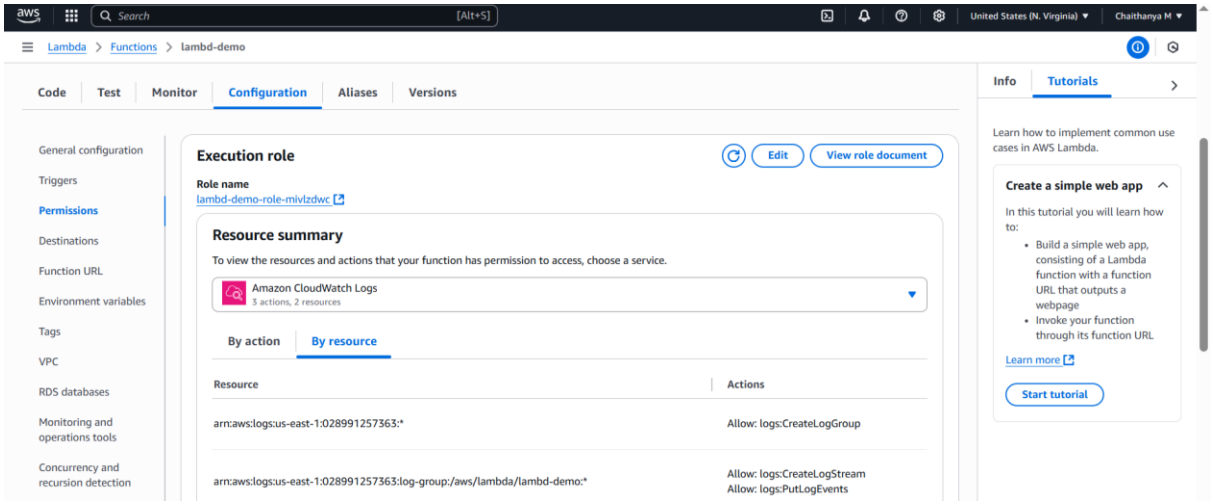
Step 2: Create the Lambda Function

1. Go to Lambda > Functions → Create function
2. Choose:
    - Author from scratch
    - Function name: lambda-demo
    - Runtime: Python 3.9
3. Click Create Function



Step 3: Verify and Update Lambda Execution Role Permissions

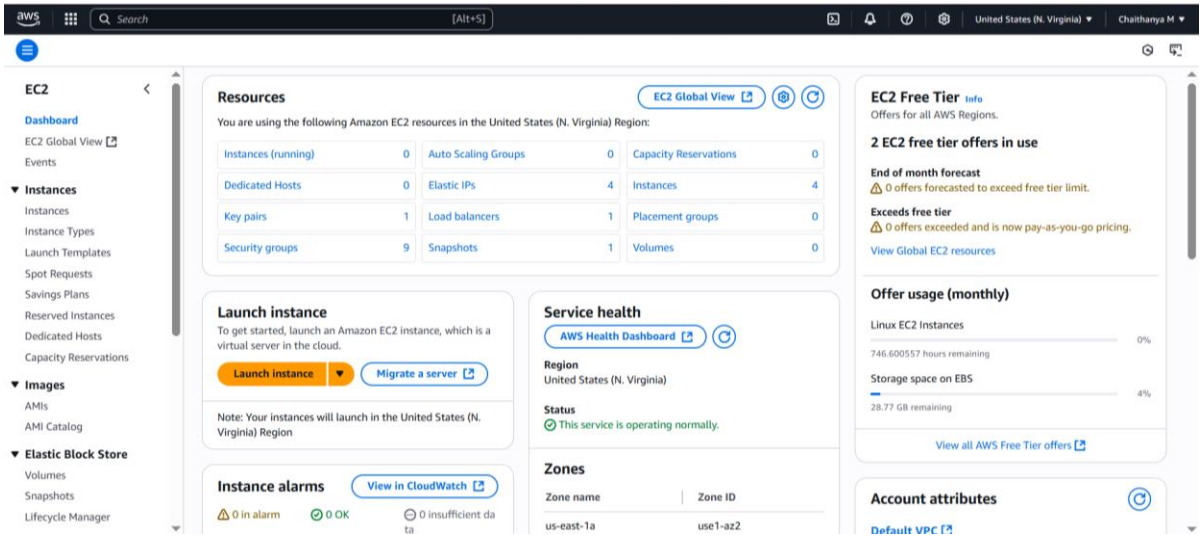
The Lambda function uses an IAM execution role (lambda-demo-role-mivlzdwc) with permissions to write logs to Amazon CloudWatch. As shown, it has access to create log groups, streams, and put log events. These permissions help monitor the function's execution and troubleshooting.



Step 4: Terminate the EC2 Instance

1. Go to EC2 > Instances
2. Select your instance → Instance state → Terminate instance

The snapshot remained even after instance deletion. This led to unnecessary snapshot storage charges.



Step 5: Add Lambda Code

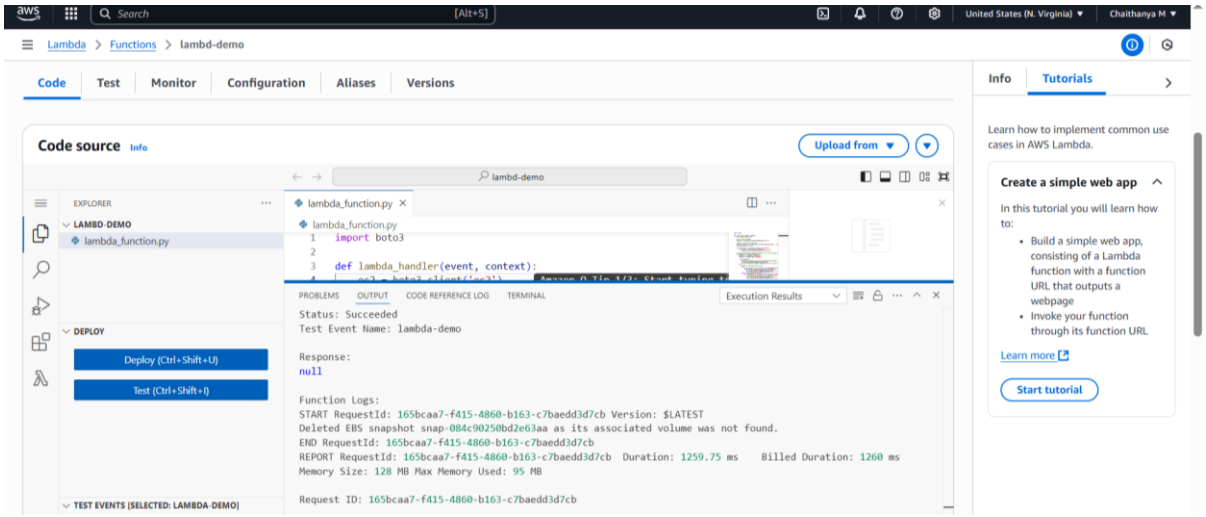
1. Scroll to Function code
2. Replace default code with the following:

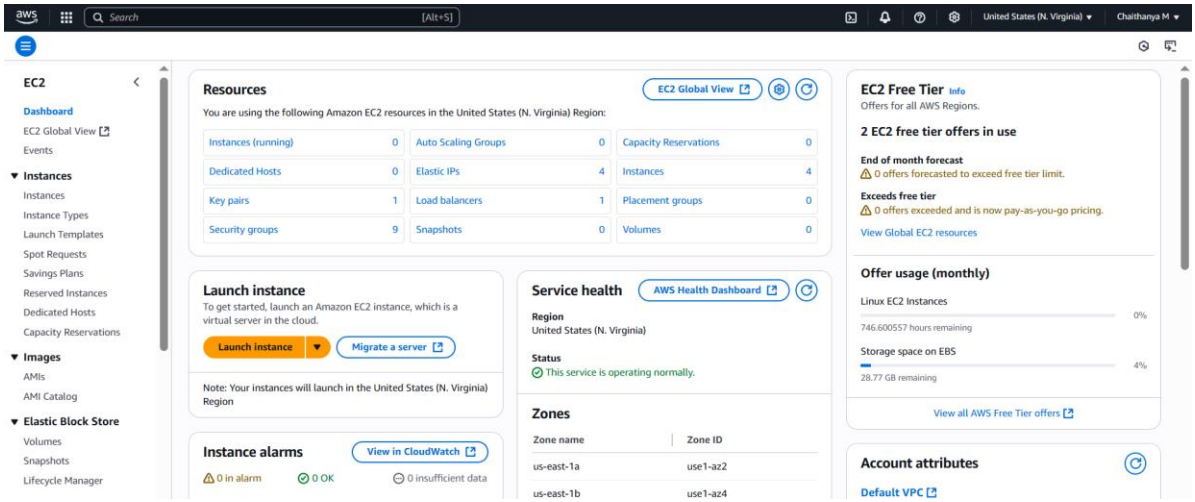
```
import boto3

def lambda_handler(event, context):
    ec2 = boto3.client('ec2')
    # Get all EBS snapshots
    response = ec2.describe_snapshots(OwnerIds=['self'])
    # Get all active EC2 instances
    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'])
        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':
                                ec2.delete_snapshot(SnapshotId=snapshot_id)
                    print(f"Deleted EBS snapshot {snapshot_id} as its associated volume was not
found.")
```

```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 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.")
```

Step 6: Test & Monitor

1. In Lambda, click Test → Configure a new test event
2. Run the test — check Logs tab
3. Verify:
    - Orphaned snapshot is deleted
    - Snapshots of active instances are untouched





6. Conclusion

This project demonstrated an effective and automated method for cost optimization in AWS by targeting one of the most overlooked areas: unused EBS snapshots. By leveraging AWS Lambda, we built a lightweight serverless solution that periodically checks and deletes EBS snapshots that are no longer associated with active EC2 instances or attached volumes.

The key outcomes include:

- Automated Cleanup: No manual intervention is required, reducing administrative overhead.
- Improved Cost Efficiency: Unused snapshots are automatically deleted, preventing unnecessary EBS snapshot storage charges.
- Resource Optimization: The function ensures storage is only used for relevant, active resources.
- Scalability: The solution can be run periodically (daily/weekly) using CloudWatch Events and scales well with increasing AWS usage.
- Security & Safety: Only snapshots no longer in use or linked to deleted resources are deleted, reducing the risk of data loss.

In conclusion, this Lambda-based snapshot cleanup system ensures that AWS environments remain cost-effective, clean, and well-managed, making it a recommended best practice for all teams using EC2 and EBS resources in production or development environments.