AWS Cost Optimization using Lambda, EC2, SNS, IAM, and Python (boto3)

Introduction

In cloud computing, resources such as EC2 instances often remain running even when they are not being used, leading to unnecessary costs. This project aims to automate AWS cost optimization by identifying and stopping idle EC2 instances using AWS Lambda, IAM, SNS, and Python (boto3). The system runs automatically, analyzes the resource utilization, and sends email notifications to the user about stopped or idle instances — thus reducing manual monitoring and saving cloud expenses.

Objective

- Automatically monitor AWS EC2 resource utilization.
- Detect idle instances based on low CPU usage.
- Stop unused EC2 instances to reduce costs.
- Send notifications via SNS to keep users informed.

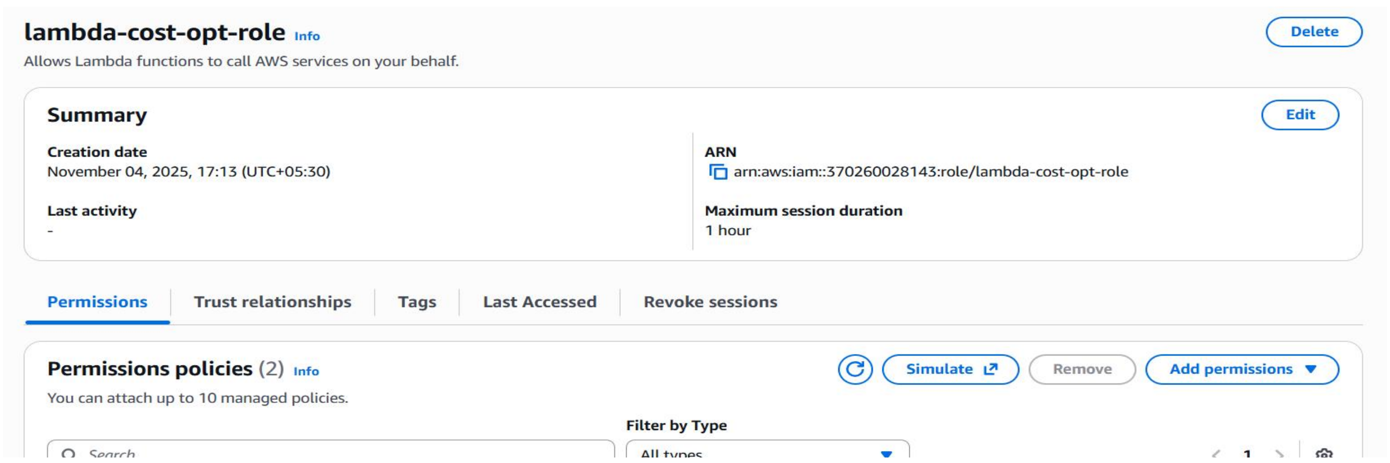
Project Workflow

Step 1: Create EC2 Instance

- Create one EC2 Instance.
- Keep it running.
- No special IAM role for EC2 instance.

Step 2: Create IAM Role

- Created a new IAM role: ***lambda-cost-opt-role***
- Attached an inline policy allowing:
  ec2:DescribeInstances, ec2:StopInstances
  cloudwatch:GetMetricStatistics
  sns:Publish
- This gives Lambda access to EC2, CloudWatch, and SNS.



Allows Lambda functions to call AWS services on your behalf.

Summary

Creation date: November 04, 2025, 17:13 (UTC+05:30)

ARN: arn:aws:iam::370260028143:role/lambda-cost-opt-role

Last activity: -

Maximum session duration: 1 hour

Permissions policies (2) Info

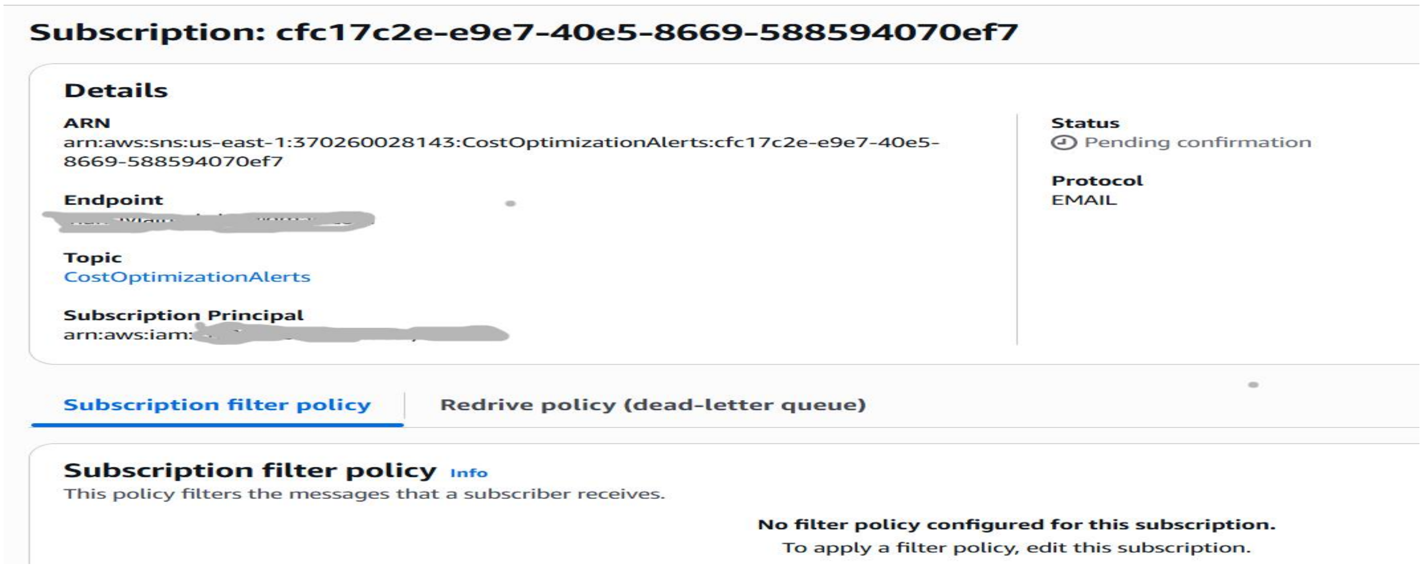
You can attach up to 10 managed policies.

Filter by Type

Fig 1: Snapshot of the IAM Role

Step 3: SNS Setup

- Created an SNS topic : ***cost-optimizer-alerts.***
- Subscribed an email address for notifications.
- Verified the subscription through the email confirmation link.



Subscription: cfc17c2e-e9e7-40e5-8669-588594070ef7

Details

ARN
arn:aws:sns:us-east-1:370260028143:CostOptimizationAlerts:cfc17c2e-e9e7-40e5-8669-588594070ef7

Endpoint
[redacted]

Topic
CostOptimizationAlerts

Subscription Principal
arn:aws:iam:[redacted]

Status
Pending confirmation

Protocol
EMAIL

Subscription filter policy
Redrive policy (dead-letter queue)

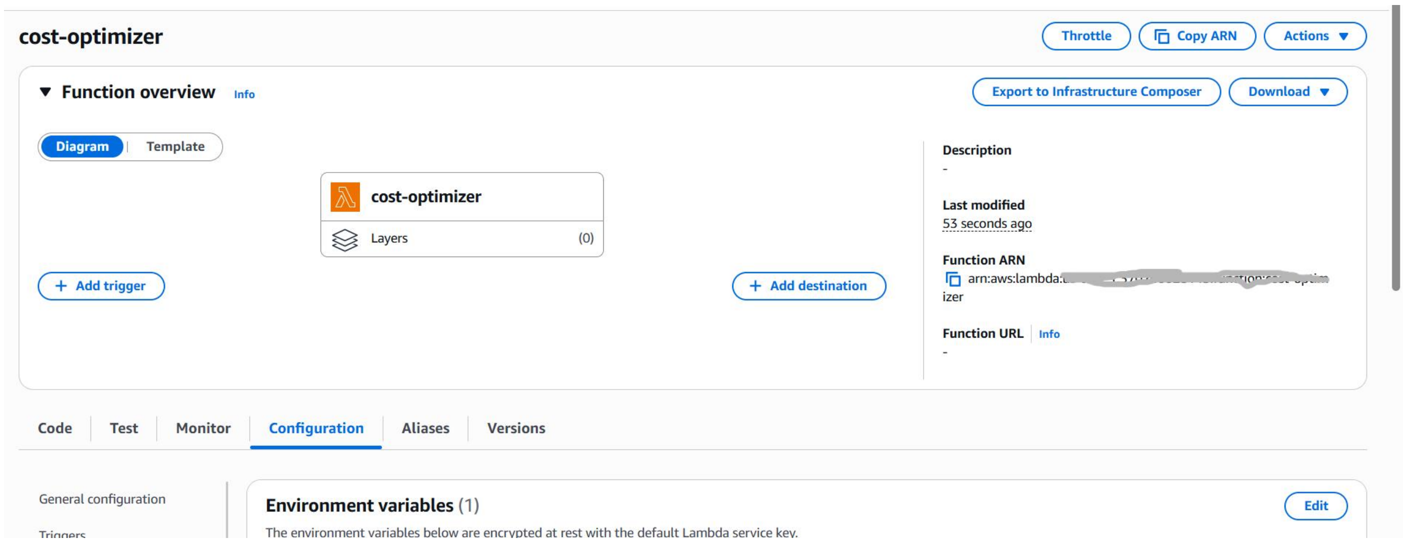
Subscription filter policy Info
This policy filters the messages that a subscriber receives.

No filter policy configured for this subscription.
To apply a filter policy, edit this subscription.

Fig 2: Snapshot of the SNS Topic

Step 4: Lambda Function

- Created a Lambda function named *cost-optimizer.*
- Selected the Python runtime.
- Attached the IAM role *lambda-cost-opt-role.*



cost-optimizer

Description
-

Last modified
53 seconds ago

Function ARN
arn:aws:lambda:[redacted]izer

Function URL Info
-

Environment variables (1)
The environment variables below are encrypted at rest with the default Lambda service key.

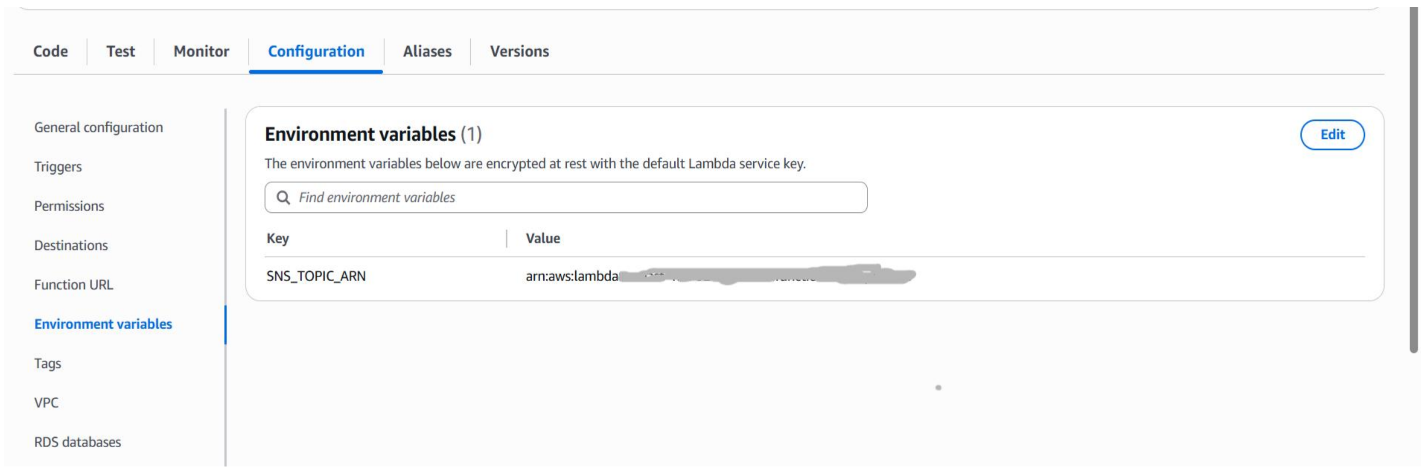
Fig 3: Snapshot of the Lambda Function

- Added the Python Code using Boto3

```
import boto3
import datetime
import os

ec2 = boto3.client('ec2')
cloudwatch = boto3.client('cloudwatch')
sns = boto3.client('sns')

SNS_TOPIC = os.environ.get('SNS_TOPIC_ARN')
CPU_THRESHOLD = 5  # 5% CPU usage

def check_idle_ec2(instance_id):
    end = datetime.datetime.utcnow()
    start = end - datetime.timedelta(hours=3)
    metrics = cloudwatch.get_metric_statistics(
        Namespace='AWS/EC2',
        MetricName='CPUUtilization',
        Dimensions=[{'Name': 'InstanceId', 'Value': instance_id}],
        StartTime=start,
        EndTime=end,
        Period=3600,
        Statistics=['Average']
    )
    datapoints = metrics.get('Datapoints', [])
    if not datapoints:
        return False
    avg_cpu = sum(dp['Average'] for dp in datapoints) / len(datapoints)
    return avg_cpu < CPU_THRESHOLD

def lambda_handler(event, context):
    report = []
    instances = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
    for reservation in instances['Reservations']:
        for instance in reservation['Instances']:
            instance_id = instance['InstanceId']
            if check_idle_ec2(instance_id):
                report.append(f"EC2 {instance_id} is idle — stopping...")
                ec2.stop_instances(InstanceIds=[instance_id])
    if SNS_TOPIC and report:
        sns.publish(
            TopicArn=SNS_TOPIC,
            Subject="AWS Cost Optimization Report",
            Message="\n".join(report)
        )
    return {"report": report}
```

Step 5: Environment Variable

- Added environment variable:


Code | Test | Monitor | Configuration | Aliases | Versions

General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases

Environment variables (1)

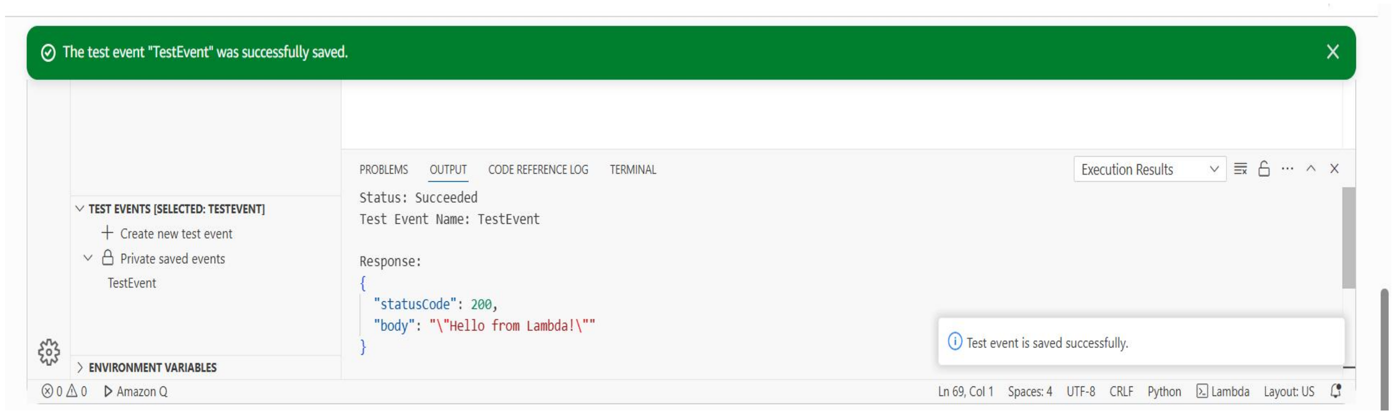
The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
SNS_TOPIC_ARN	arn:aws:lambda[redacted]

Fig 4: Snapshot of the Environment Variable

Step 6: Test Execution

- Created a test event named ***TestEvent***.
- Clicked **Test** in the Lambda console.
- The **execution result** showed success


The test event "TestEvent" was successfully saved.

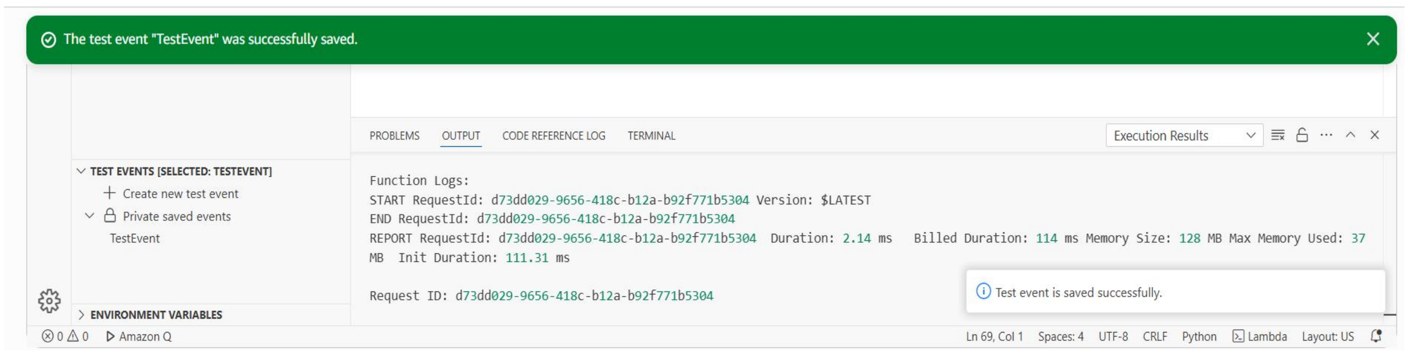
TEST EVENTS [SELECTED: TESTEVENT]
Create new test event
Private saved events
TestEvent
ENVIRONMENT VARIABLES

PROBLEMS | OUTPUT | CODE REFERENCE LOG | TERMINAL — Execution Results

```
Status: Succeeded
Test Event Name: TestEvent

Response:
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Test event is saved successfully.

Ln 69, Col 1 Spaces: 4 UTF-8 CRLF Python Lambda Layout: US


The test event "TestEvent" was successfully saved.

TEST EVENTS [SELECTED: TESTEVENT]
Create new test event
Private saved events
TestEvent
ENVIRONMENT VARIABLES

PROBLEMS | OUTPUT | CODE REFERENCE LOG | TERMINAL — Execution Results

```
Function Logs:
START RequestId: d73dd029-9656-418c-b12a-b92f771b5304 Version: $LATEST
END RequestId: d73dd029-9656-418c-b12a-b92f771b5304
REPORT RequestId: d73dd029-9656-418c-b12a-b92f771b5304  Duration: 2.14 ms   Billed Duration: 114 ms Memory Size: 128 MB Max Memory Used: 37
MB  Init Duration: 111.31 ms

Request ID: d73dd029-9656-418c-b12a-b92f771b5304
```

Test event is saved successfully.

Ln 69, Col 1 Spaces: 4 UTF-8 CRLF Python Lambda Layout: US

Fig 5 & 6: Snapshots of the Test Event

Step 7: Result Verification

- If idle EC2 instances were found, they were **automatically stopped**.
- An **email notification** was received from SNS with details like:

  EC2 i-0abc123def45 is idle — stopping...

- If no idle instances, the report returned:

  No idle EC2 instances found.

Conclusion

This project demonstrates how AWS automation can optimize cloud costs using serverless architecture.
By integrating **Lambda, EC2, SNS, IAM, and Python (boto3)**, idle instances were automatically detected and stopped, reducing unnecessary expenses.
The system is scalable, easily deployable, and requires no manual monitoring, showcasing effective use of AWS services for cost efficiency.