**Question 1**

**Scenario:**

Your customer currently manually manages EC2 instances, sometimes leaving unused instances running, resulting in unnecessary costs. They're asking if you can automate the identification and termination of EC2 instances tagged as "Environment": "Dev" that have been running for more than 7 days.

**Task:**

Explain step-by-step, how you would automate this scenario using Python and boto3. Please provide the Python boto3 code.

**EC2 Idle Instance Cleanup Automation (Step-by-Step Instructions)**

Objective:

This document outlines the step-by-step process for automating the identification and termination of AWS EC2 instances tagged with "Environment=Dev" that have been running for more than ***7 days*** and have low CPU utilization**(unused)**.

The logic behind the script is, first setting a threshold to identify older instances and filtering them based on tags. It then checks each instance using CPU CloudWatch metrics to find idle instances based on the average threshold. If an instance is idle, it gets terminated. Additionally, as an optional step, the script can send an email to the instance owner using SES and take snapshots of the AMI

The screenshots shared below are from a demo project I set up specifically on the AWS cloud provider for this case.

Below are the step-by-step instructions on how to run this script,

### **Step 1: Prerequisites**

* AWS CLI must be installed and configured on your local machine.

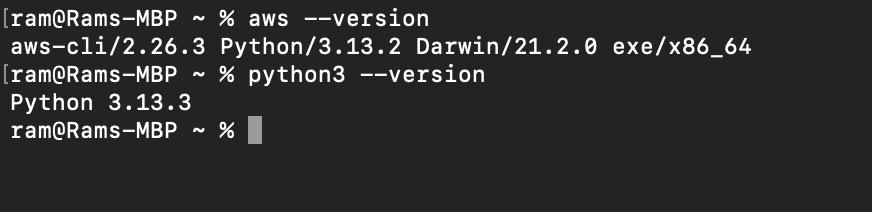
aws configure

* Boto3 (AWS SDK for Python) must be installed.

pip install boto3

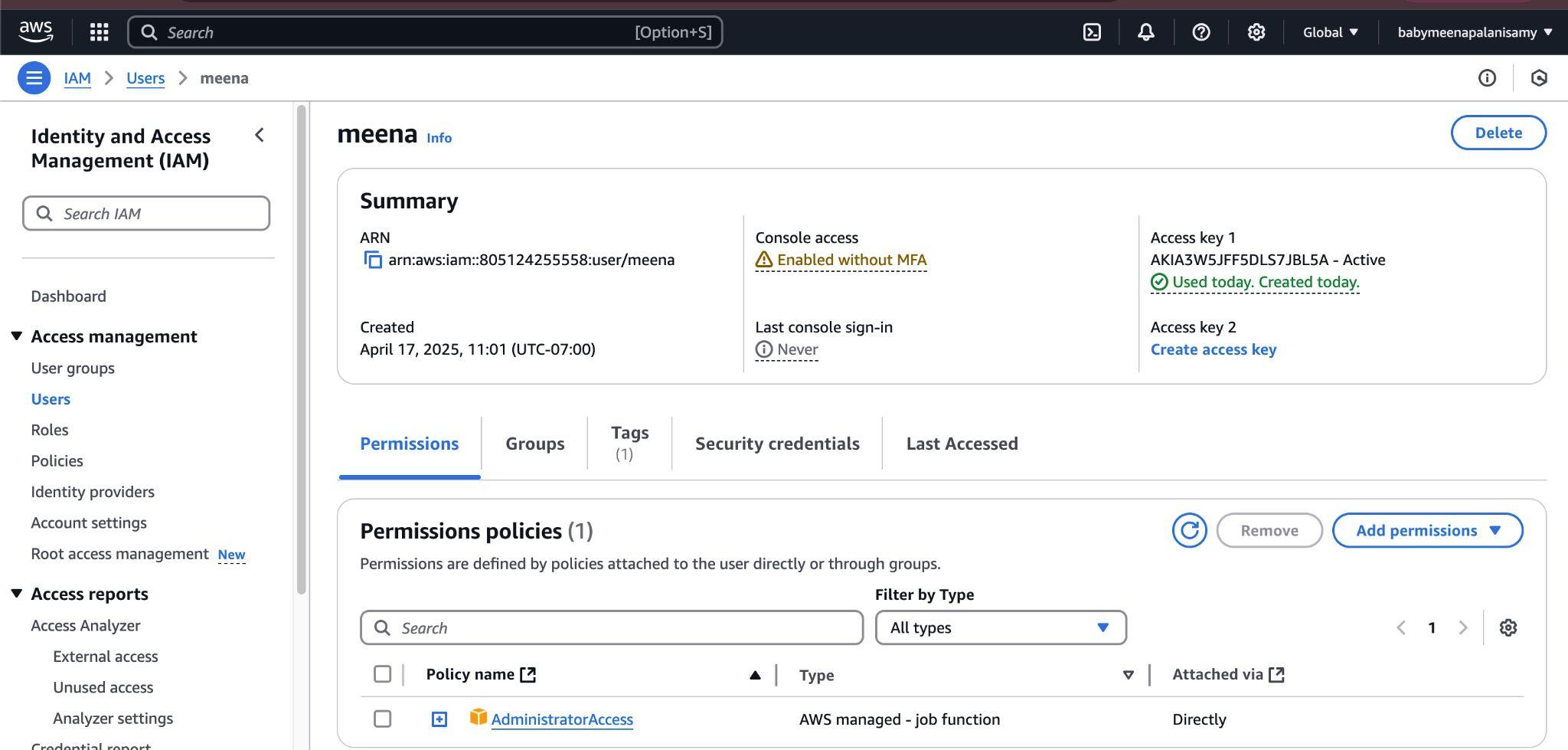
* Python 3.x should be available on the system.

python3 --version



* IAM credentials with permission to describe instances, get CloudWatch metrics, create AMIs, and terminate instances.

The below IAM user has the administrator access policy attached.



* Optional: Verified email in AWS SES and relevant IAM permissions if email notification is used.

**Step 2: Create the Python Script**

* Open a text editor like Visual Studio Code.
* Create a new Python file (e.g., terminate\_unused\_dev\_instances.py).
* Copy and paste the Python code provided separately.

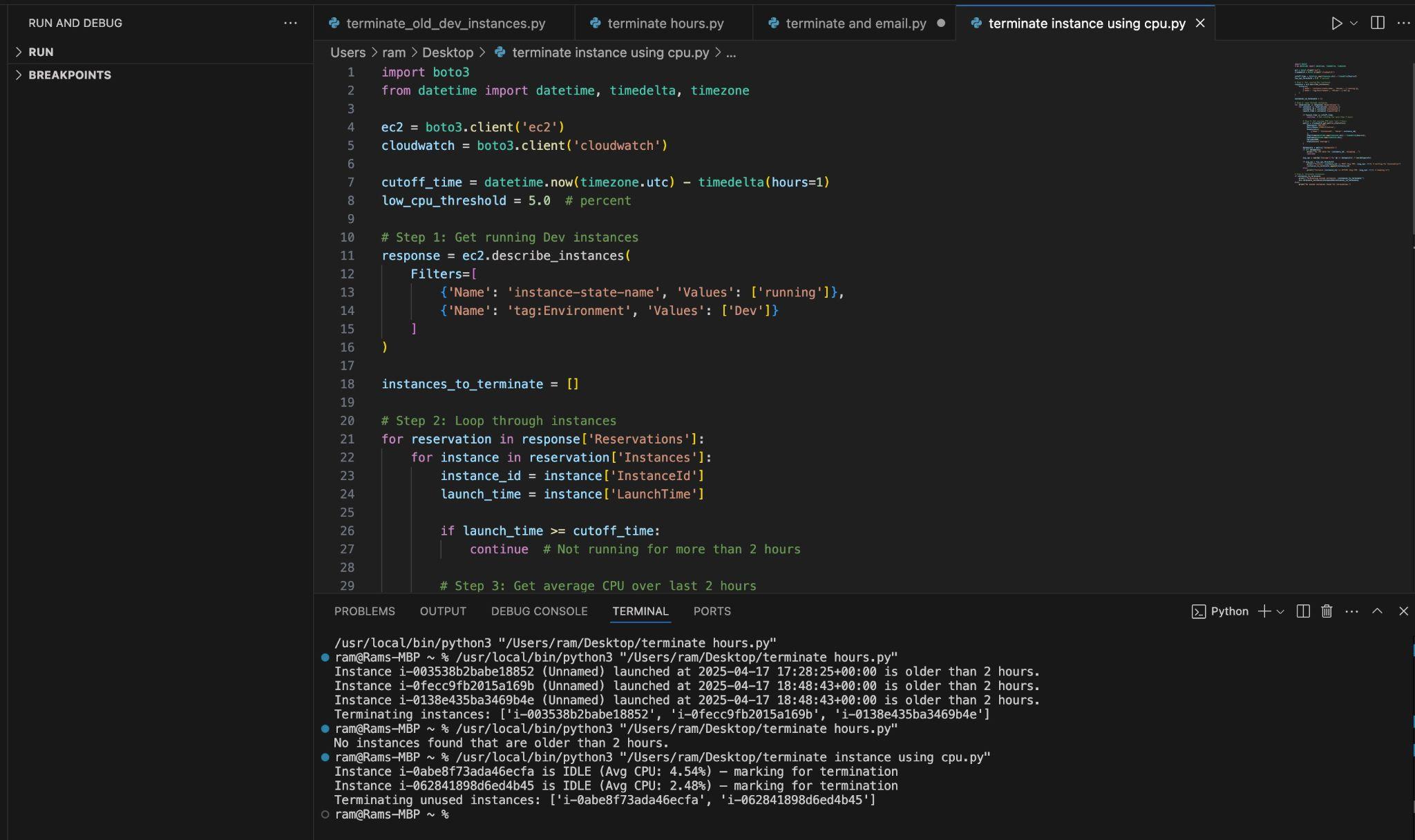
### **Step 3: Script Logic Overview**

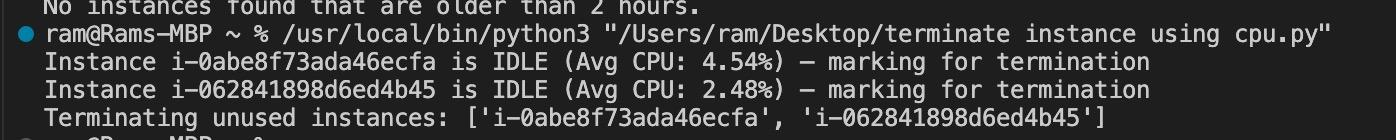
1. **Initialization**: Connect to AWS EC2 and CloudWatch services using boto3.
2. **Set Time Thresholds**: Define the threshold for identifying idle instances (e.g., older than 7 days with average CPU < 5%).
3. **Instance Filtering**:
   * Filter running instances with the tag Environment=Dev.
4. **Evaluate Each Instance**:
   * Check if the instance has been running for over 7 days.
   * Retrieve the average CPU utilization using CloudWatch metrics.
   * Identify idle instances based on CPU usage.
5. **(Optional)**:
   * Create an AMI (Amazon Machine Image) before termination for backup.
   * Send an email notification to the instance owner using their tagged email address.
6. **Terminate Identified Instances**: Proceed with termination for instances meeting the criteria.

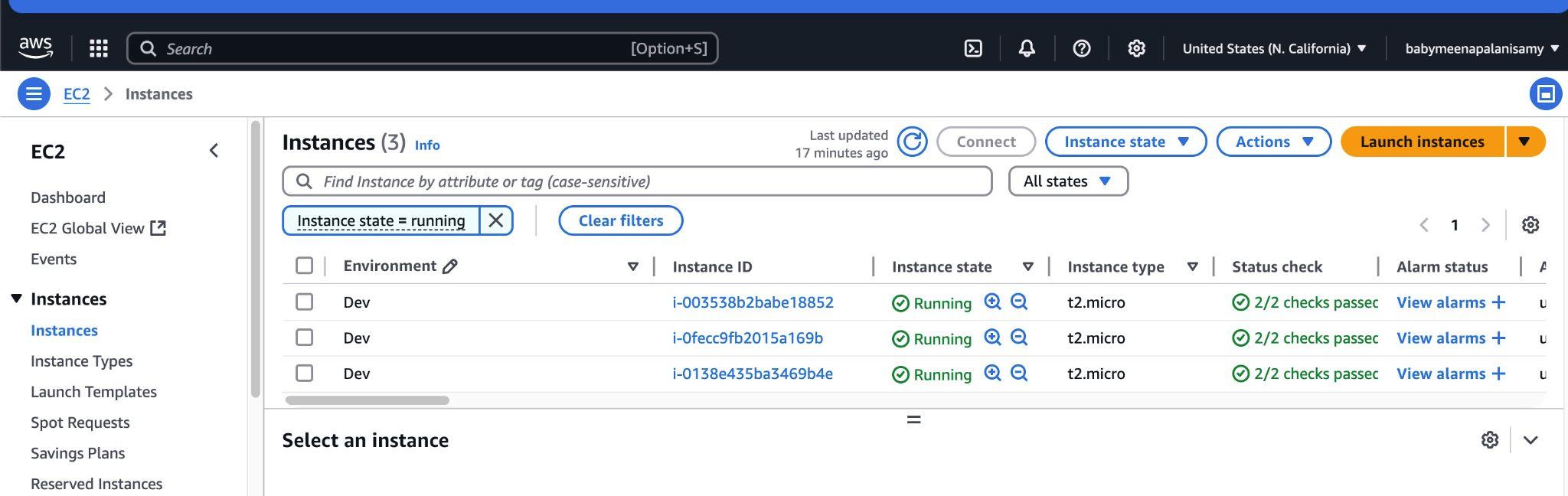
PS: The code has been shared in a separate file

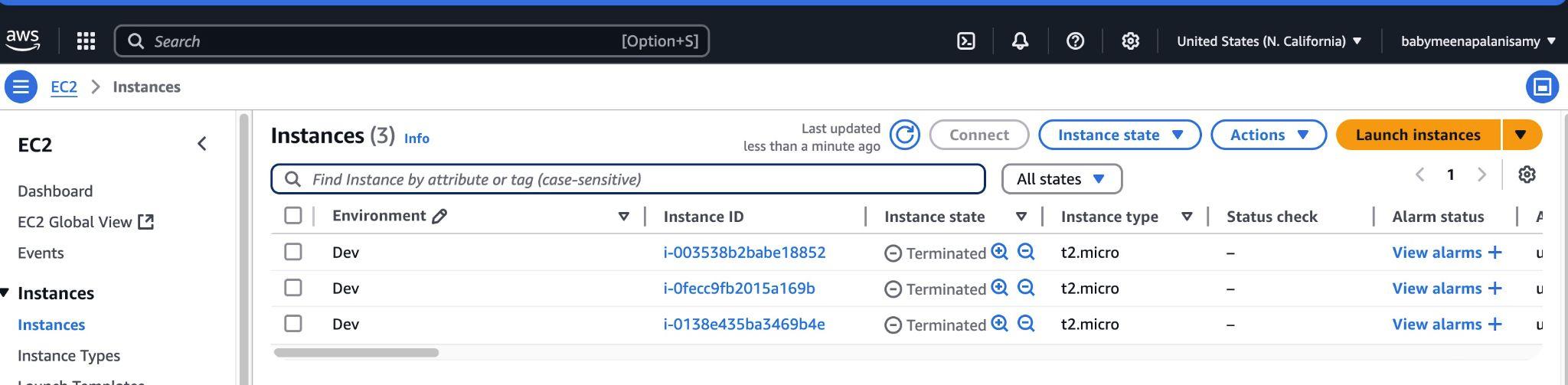
### **Step 4: Run the Script**

1. Open a terminal and navigate to the script's directory.
2. Execute the script using the following command:  
   python3 terminate\_unused\_dev\_instances.py
3. Monitor the output logs to verify actions taken, such as identifying idle instances, creating AMIs, and terminating instances.









### 

### 

### **Step 5: Scheduling the Script**

* Use cron (Linux/macOS) or Task Scheduler (Windows) to automate periodic execution.
* Example cron job to run daily at midnight:  
  0 0 \* \* \* /usr/bin/python3 /path/to/terminate\_unused\_dev\_instances.py

**Step 6: Serverless Way (Lambda + CloudWatch)**

* Create an AWS **Lambda** function and attach the IAM role to access the EC2 instances.
* Upload your Python code. Attached the python code optimized for lambda.
* Set up a **CloudWatch Event Rule** (EventBridge rule) to trigger the Lambda based on a schedule (like every hour or daily).

**(OPTIONAL)Creating CPU Spike on the Server for testing:**

This is an optional step to create a temporary spike on the server. A **simple HTML file + Python script** setup, I used this to simulate CPU usage **above 5%** on a Linux machine by running a small web server that triggers some CPU-intensive operations when accessed.

This script is intended for testing only. Make sure to run this first before executing the Python Boto3 script that terminates unused instances.

**Step 1: Create a simple HTML file**

**File: index.html**

<!DOCTYPE html>

<html>

<head>

<title>CPU Spike Test</title>

</head>

<body>

<h1>Triggering CPU Load</h1>

<p>This page runs a CPU-intensive task on the server.</p>

</body>

</html>

Place this file in the same directory as the below Python script.

## 

## **Step 2: Python script to serve HTML and spike CPU**

**File name:** cpu\_spike\_server.py

from http.server import BaseHTTPRequestHandler, HTTPServer

import time

import threading

PORT = 8080

def cpu\_intensive\_task(duration=30):

print("Starting CPU load...")

end\_time = time.time() + duration

while time.time() < end\_time:

[x\*\*2 for x in range(10000)] # CPU task

print("CPU load ended.")

class RequestHandler(BaseHTTPRequestHandler):

def do\_GET(self):

if self.path == "/":

# Trigger CPU spike in a background thread

threading.Thread(target=cpu\_intensive\_task).start()

self.send\_response(200)

self.send\_header("Content-type", "text/html")

self.end\_headers()

with open("index.html", "r") as f:

self.wfile.write(f.read().encode())

else:

self.send\_error(404)

if \_\_name\_\_ == "\_\_main\_\_":

print(f"Serving on http://localhost:{PORT}")

server = HTTPServer(("", PORT), RequestHandler)

server.serve\_forever()

## 

## **Step 3: Run the server:**

Run the below commands to cause the spike,

1. SSH into your Linux EC2 instance
2. Make sure Python 3 is installed
3. Save both files (cpu\_spike\_server.py and index.html) in the same directory.
4. Run the server:

python3 cpu\_spike\_server.py

## 

## **Step 4: Trigger the CPU usage**

Open a browser and go to:

http://<your-ec2-public-ip>:8080

This triggers a background CPU-intensive task for 30 seconds.