CSCI 5380 – Network Virtualization and Orchestration

Lab 4

Amazon Web Services (AWS)

University of Colorado Boulder

Department of Computer Science

Network Engineering

Professor Levi Perigo, Ph.D.

# Summary

Amazon Web Services (AWS) is a cloud-computing platform offered by Amazon. Cloud technology and tools are popular for network engineering, and the skills learned in this lab will enhance students’ resumes in a desirable skillset in the current market. The AWS tools and skills used throughout this lab include EC2, security policies, S3 buckets, SNS, and Cloud Watch. Although these services could be managed using the AWS Management Console, AWS also offers a powerful tool (Software Development Kit) known as Boto3. This gives the power to manage the above-mentioned services for a large number of resources using simple Python scripts integrated with the Boto module.

# Objectives

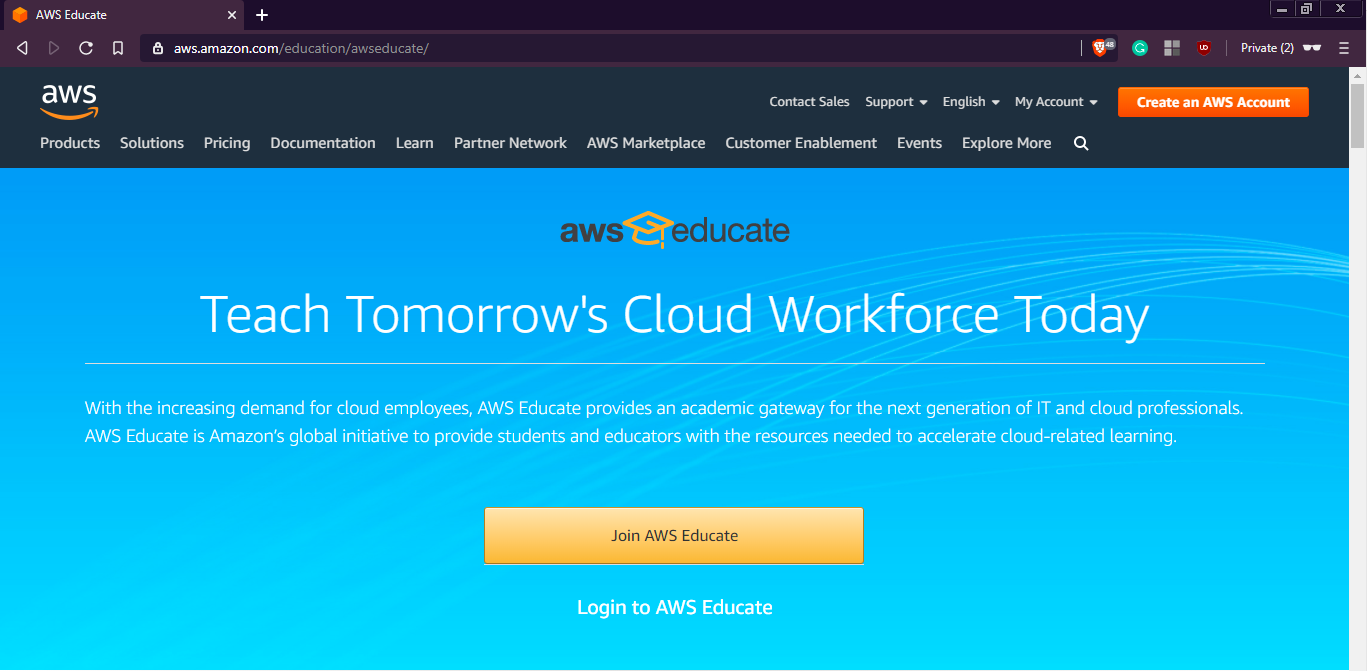
1. Learn about Amazon Web Services (AWS) tools
2. Lean how to deploy EC2 server instances
3. Learn how to deploy applications on those instances
4. Learn how to create security policies/firewall rules
5. Learn how to backup configurations into S3 buckets
6. Learn how to setup and use Simple Notification Service (SNS)
7. Learn how to setup Cloud Watch monitoring system
8. Manage AWS resources using Boto3 (Python based SDK)

# Part 1

# Objective 1.1 – Create an AWS account

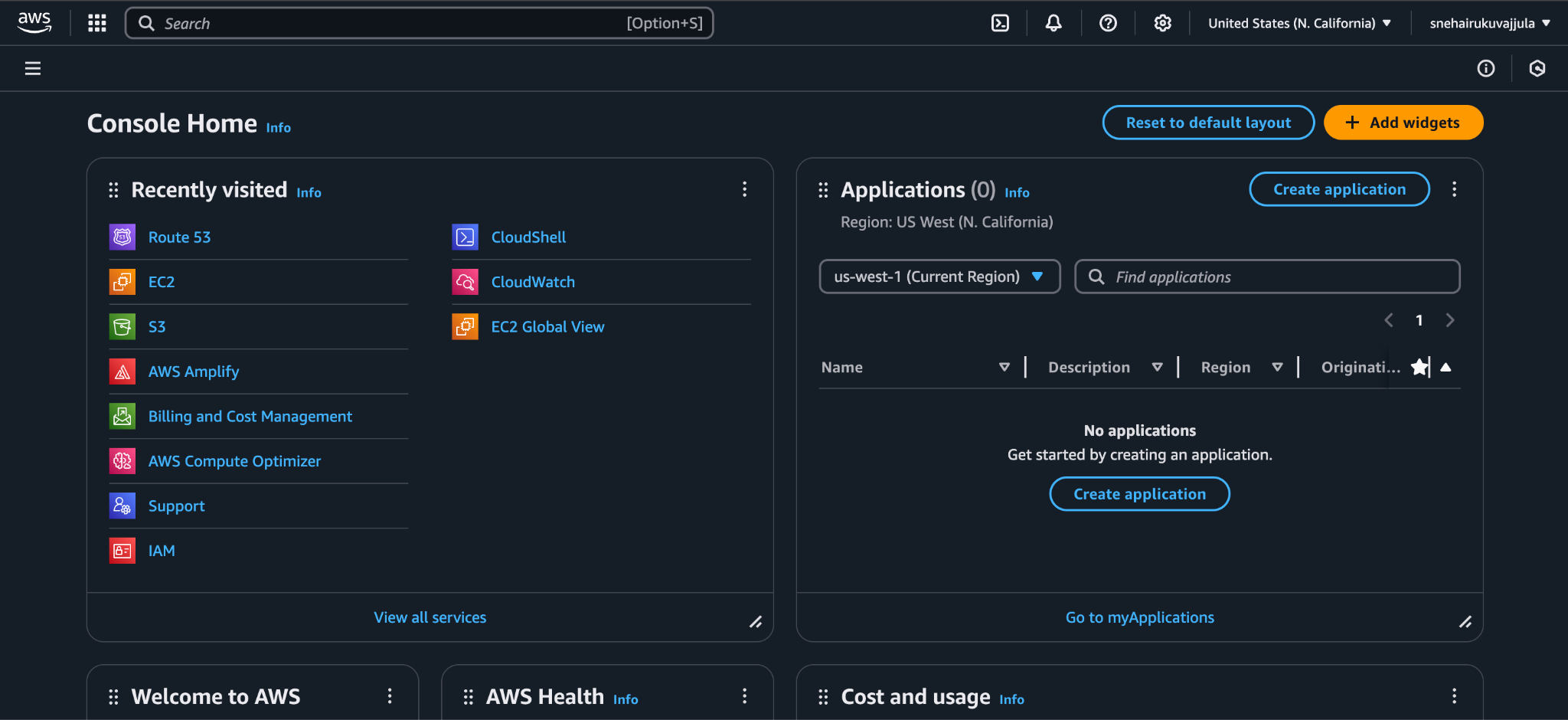
In order to use Amazon Web Services, you must first create a user account. Amazon provides one year of free usage (limited resources).

1. Create an AWS Educate account at <https://aws.amazon.com/education/awseducate/> to get AWS promotional credits. You can also use your amazon.com credentials to log in, but please ensure you create a student account to get the AWS credits.



[**NOTE:** You might need to provide your debit/credit card number for future billing]

1. After creating the account, you will be directed to AWS management console which presents the user with a number of AWS tools and services. Provide a screenshot of your management console. [**1 point**]

****

*Screenshot 1: AWS console*

Objective 1.2 – Tweaking AWS settings

We shall now look at tweaking the AWS management settings to make efficient use of these resources.

1. We do not want Amazon to charge us for either using their premium services (Multi-Core CPU’s, Terabytes of data, CDN services, etc.) or breaching their ONE-year free usage tier.

**To prevent extra charges on your account please shut down and remove all instances at the end of the lab.**

To ensure you get the most efficient performance, we need to change our region to any of the US West regions (N. California/Oregon). You can do this from the AWS management console. Provide a screenshot of the region you selected [**1 point**].

# 

*Screenshot 2: Selected N. California as my region*

1. Set up security credentials for your account: <https://aws.amazon.com/premiumsupport/knowledge-center/create-access-key/> (Please ensure that you download and save the keys on your machine, as it will be required for further objectives).

# Part 2

# Objective 2.1 – Deploying EC2 Instances

1. What is EC2? [**5 points**]

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. I’d like to think of it just like any other Virtual Machine. It is designed to make web-scale cloud computing easier for developers. It offers different instance types optimized for compute, memory, storage, and GPU-intensive workloads.

1. Briefly explain the below types of EC2 instances. [**15 points**]

* Spot instances

1. Spot Instances allow users to bid for unused AWS capacity at a lower price than On-Demand instances.
2. Suitable for batch processing, big data, and CI/CD pipelines.
3. AWS can reclaim the instance with short notice when capacity is needed.

* Reserved instances

1. Users commit to 1- or 3-year contracts, receiving discounts compared to On-Demand pricing.
2. Ideal for applications with steady-state usage, such as databases and enterprise applications.
3. Standard (highest discount), Convertible (switch instance types), and Scheduled (run at specific times).

* Dedicated hosts

1. Provides a dedicated physical server for regulatory compliance and licensing requirements.
2. Allows the use of existing software licenses (Windows, SQL Server, etc.).
3. Users can control instance placement for better security and compliance.

* Dedicated instances

1. Instances run on AWS hardware dedicated to a single customer but do not provide complete host-level visibility.
2. Provides physical isolation from other AWS customers.
3. AWS manages the host allocation.

* Elastic GPUs

1. Enables adding GPU capability to EC2 instances without using expensive GPU-based instance types.
2. Supports workloads such as rendering, gaming, and deep learning inference.
3. Users can select different GPU sizes based on workload needs.

To deploy an EC2 instance, click on Services> EC2 > Launch Instance.

**Step 1:** Search for ubuntu and select Ubuntu Server 16.04 LTS 64-bit (x86) image.

[Or you can select an image of your choice.]  
**Step 2:** t2.micro (Free tier eligible)

**NOTE:** Selecting anything else will result in OS costs.

**Step 3:** Deploy **TWO** instances.

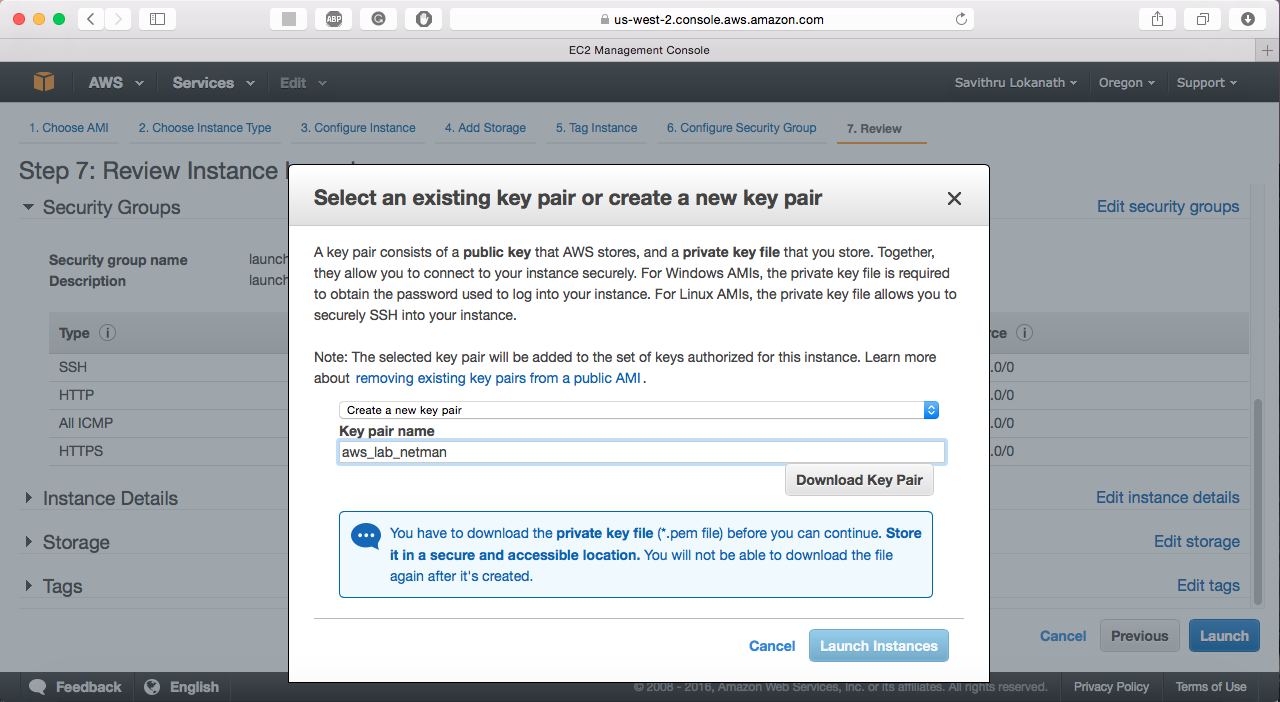
**Step 4:** Add storage (8 GB is fine, can add more later)

**Step 5:** Add name as NVO-Lab

This uniquely identifies the instance and will come handy in a clustered environment. You can define something like a **“webserver”** if you are deploying a webserver or **“appserver”** for an application server, etc.

**Step 6:** Create a new security group and select the source to be your IP address for SSH rule. Add security rules allowing ICMP, HTTP and HTTPS traffic from your IP address.

**Step 7:** Review the configuration parameters before launching the instance. Create a new key-pair and click on download. [**NOTE:** If you miss this step, you will have to repeat the whole process again.] After the download completes, click on the **“Launch”** button.



1. Explain how an AMI is related to the instance. [**3 points**]
2. An AMI serves as a pre-configured template containing the operating system, application stack, and settings required to launch an EC2 instance.
3. When launching an EC2 instance, users select an AMI, which defines the base configuration.
4. Users can create custom AMIs from existing instances, allowing quick recovery, cloning, or replication.
5. What are the disadvantages of allowing SSH traffic into the server from anywhere? Is this a security concern? [**5 points**]

* If SSH is open to all IPs, anyone on the internet can attempt to connect to the server.
* A compromised user account or weak authentication method increases security risks.
* Attackers can flood SSH with connection requests, overwhelming server resources and causing service disruption.
* Constant login attempts from bots and hackers fill up logs, making it harder to detect legitimate issues and excessive failed login attempts may consume server resources unnecessarily.

1. What are some best practices one should follow to secure the north-south and east-west traffic in the cloud? [**5 points**]

**North-South Traffic Security**

Implement security groups and firewall rules to allow only necessary inbound/outbound traffic.

Implement multi-factor authentication (MFA) for all external access.

Deploy rate limiting and anomaly detection tools.

Deploy cloud-native or third-party IDS/IPS solutions to monitor and block threats.

Restrict access based on identity, device posture, and contextual factors.

**East-West Traffic Security**

Implement network segmentation using VLANs or SDN-based policies.

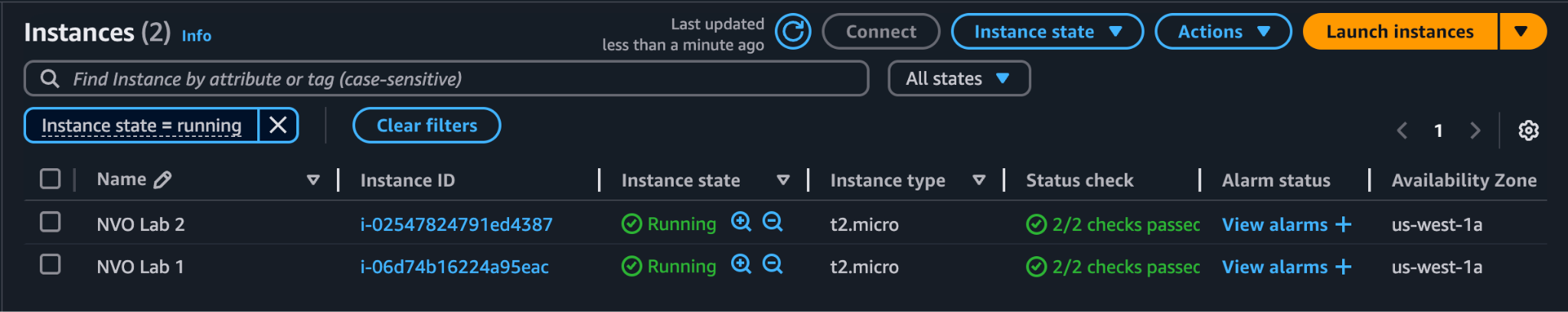
Use host-based firewalls to limit internal traffic.

Apply fine-grained access controls at the application layer.

Deploy workloads in separate VPCs/VNets with peering or transit gateways.

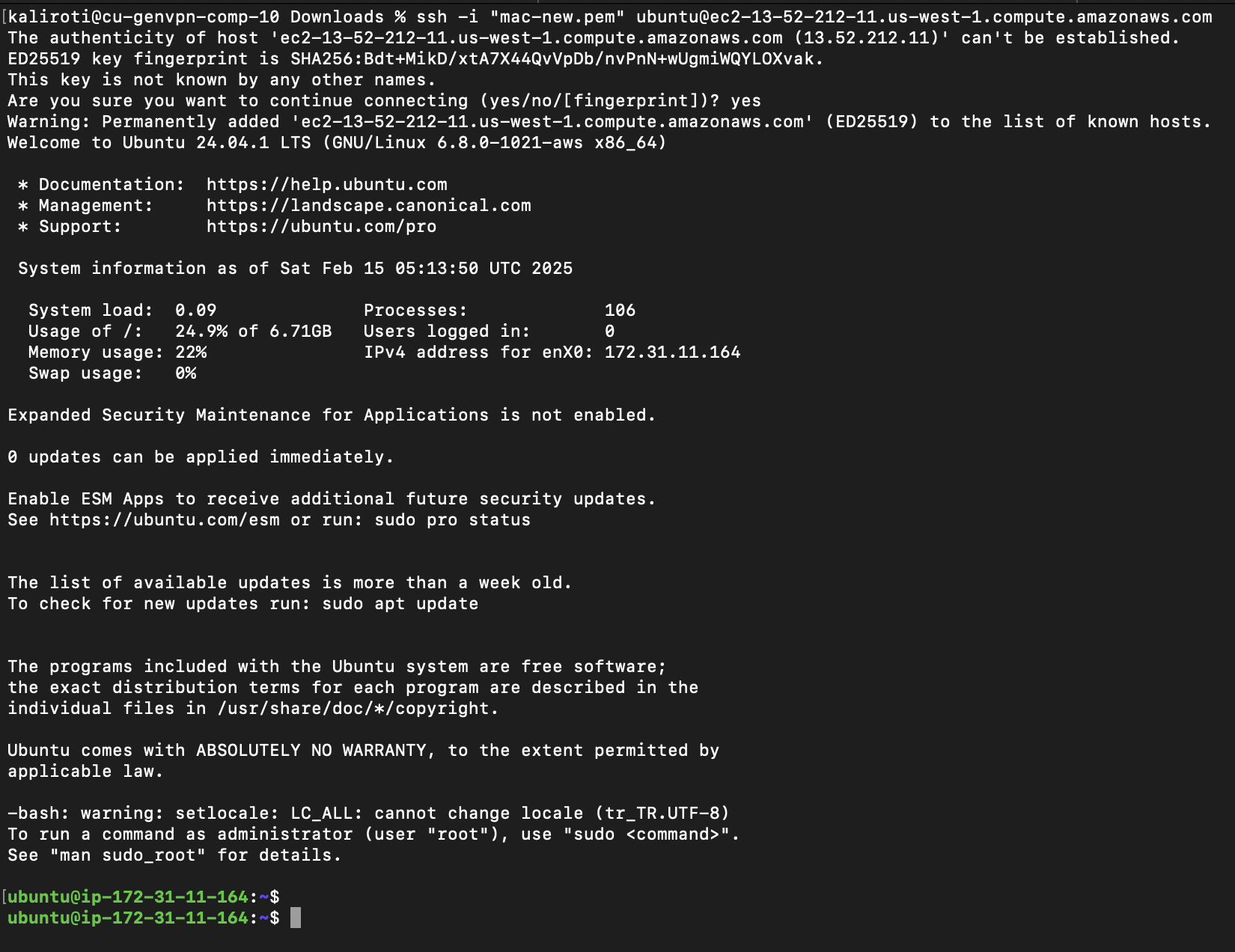
Use dedicated network interfaces for sensitive workloads.

1. Paste a screenshot showing the running instances. [**10 points**]



*Screenshot 3: Running EC2 instances*

1. Select an instance and click on connect. Follow the instructions and SSH into one of the EC2 instances. Paste a screenshot showing the bash prompt. [Login for the Ubuntu instance is: ubuntu] [**5 points**]



*Screenshot 4: Connected to an EC2 instance*

1. What IP address did you SSH to? And what is the IP address on the interface of the instance? Explain the flow of traffic from your laptop to the EC2 instance. [**5 points**]

I connected using my public IP address.

Public IP: 13.52.212.11

Private IP: 172.31.11.164/20

* My Laptop Initiates an SSH Connection

I run an SSH command from my laptop:

| ssh -i mac.pem ubuntu@13.52.212.11 |
| --- |

* My laptop sends a TCP packet to port 22 of my AWS EC2's public IP (13.52.212.11).
* Packet Travels Over the Internet
* The packet goes through my home router/modem.
* It reaches my ISP (Internet Service Provider).
* The ISP routes it to the AWS data center in my selected AWS region (US-West-1, N. California).
* AWS Public Network Receives the Request
* AWS Elastic IP (EIP) or public IP is mapped to my EC2 instance.
* AWS routes the packet to my instance’s private IP (172.31.11.164) inside my VPC (Virtual Private Cloud).
* My EC2 Instance Processes the Request
* The SSH request arrives at port 22.
* My security group allows SSH from my IP, hence the EC2 instance accepts the connection.
* My SSH keys are verified for authentication.
* The EC2 instance sends a response back through the AWS public internet gateway.
* The response follows the reverse path (AWS → ISP → Home Router → My Laptop).
* I successfully see the command prompt of my EC2 instance.

1. Create another user in your instance with the username being your name and enable SSH with password for just your user. Explain how you achieved this. [**3 points**]

I created a new user named sneha using the following command:

| sudo adduser sneha |
| --- |

By default, AWS does not allow SSH login with passwords, so I had to enable it.

I edited the AWS SSH configuration override file:

| sudo vim /etc/ssh/sshd\_config.d/60-cloudimg-settings.conf |
| --- |

Changed this line from:

| PasswordAuthentication no |
| --- |

To:

| PasswordAuthentication yes |
| --- |

Then restarted the ssh service.

To restrict SSH login to only sneha and ubuntu, I edited the SSH configuration:

| sudo vim /etc/ssh/sshd\_config |
| --- |

I added this line:

| AllowUsers sneha ubuntu |
| --- |

This ensures that only sneha and ubuntu can log in via SSH.

I saved and exited, then restarted SSH again:

| sudo systemctl restart ssh |
| --- |

1. What is the difference between stopping and terminating an instance? [**2 points**]

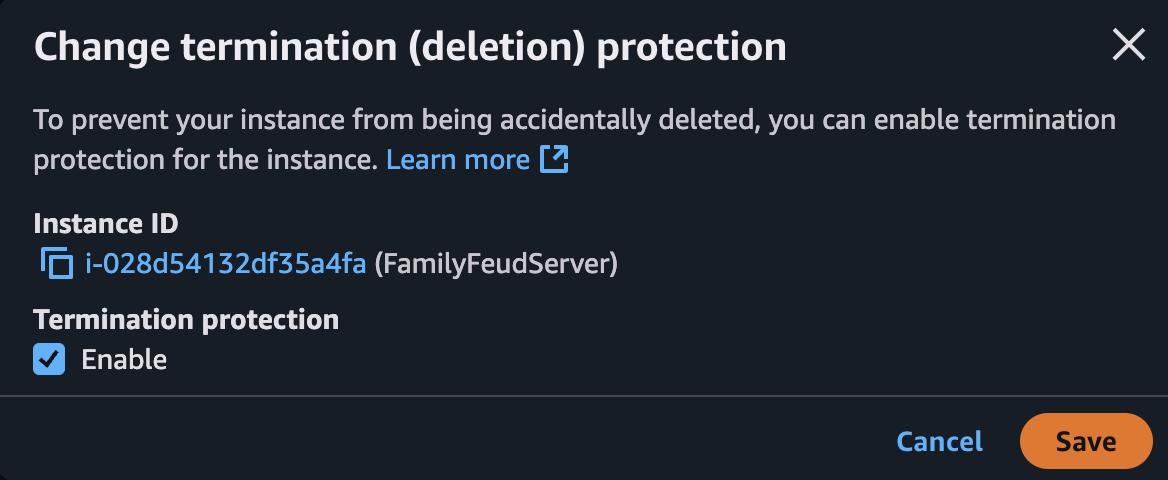
**Stopping an EC2 Instance**

The EC2 instance is shut down, but its EBS (Elastic Block Storage) volume remains attached. Data on the root EBS volume is preserved, so you can start the instance later with the same configuration.

**Terminating an EC2 Instance**

The EC2 instance is permanently deleted. All associated EBS volumes are deleted.

1. Protect the second instance that you created from unauthorized termination. How did you achieve this? [**2 points**]



*Screenshot 5: Enabling unauthorized termination*

Objective 2.2 – Deploying an application on your instance

Install Apache web server on one of the instances. Create an index.html file in the /var/www/html/ directory which displays your name and ‘NVO Lab’. You can use the below sample.

| <HTML>  <HEAD>  <TITLE>NVO Lab</TITLE>  </HEAD>  <BODY>  <H2>Your\_Name</H2>  </BODY>  </HTML> |
| --- |

How do you access this webpage hosted on the instance from your laptop? Paste relevant screenshots. Can you access it using an IP address or DNS name or both? [**20 points**]

To access the webpage, I installed apache2 and created an index.html file under /var/www/html. Next, I accessed the webpage through the public IP address of my EC2 instance. This can also be accessed via the DNS name as well.



*Screenshot 6: Accessing the EC2 instance webpage.*

# Part 3

Objective 3.1 – Deploying S3 Backups

**NOTE:** It is preferred to use the course VM provided to complete this objective.

1. What is S3? [**5 points**]

Amazon Simple Storage Service (*S3*) is a scalable, durable, and highly available object storage service provided by AWS. It allows users to store, retrieve, and manage data in the cloud in the form of objects within buckets.

1. You shall now backup some of our router configurations to the cloud. Before you proceed, ensure you have configuration files of your routers stored in a separate folder (Eg. /home/nvo/routerConfigs)

[**NOTE:** If you do not have configuration files present on your system, feel free to use other files (images, files, text,etc.)]

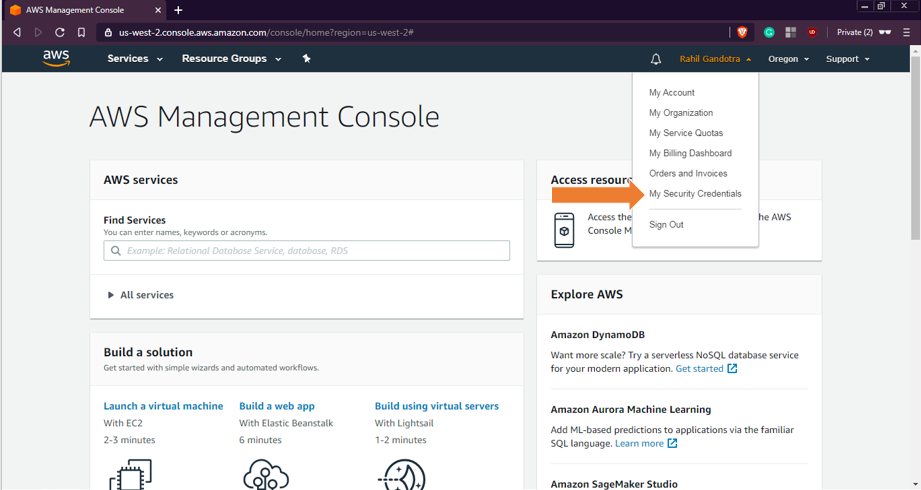
1. Update your VM and install S3 command-line utility

**(VM)# sudo apt-get install s3cmd -y**

1. Configure S3 parameters by entering your AWS access/secret credentials and enter the encryption password (Your choice) when prompted. Save the settings.

**(VM)# s3cmd –configure**

You can create/find an AWS access/secret key on your “username” > Security Credentials > Continue > Access Keys > Create/Use



1. Create an S3 bucket

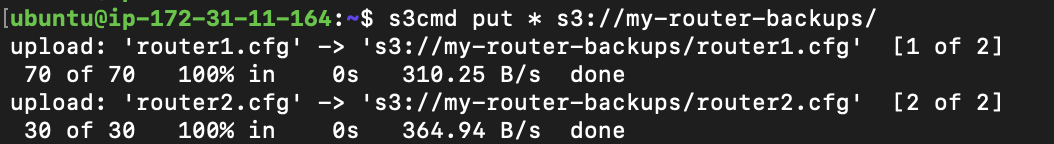
**(VM)# s3cmd mb s3://<S3\_BUCKET\_NAME\_CREATED>**  
Confirm by issuing **“s3cmd ls”**. You can also check using the AWS Management Console. S3 is located under Storage and Content Delivery.

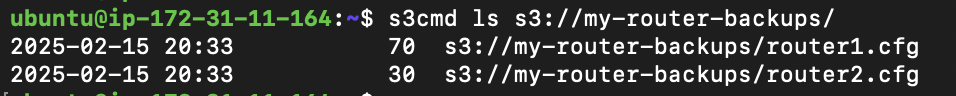


*Screenshot 7: Creating S3 bucket*

1. Push the configuration folder into the bucket

**(VM)# s3cmd put <PATH\_TO\_LOCAL\_CONFIG\_FOLDER> s3://< S3\_BUCKET\_NAME\_CREATED>**  
Verify that the files are updated. Paste a screenshot showing the same. [**10 points**]



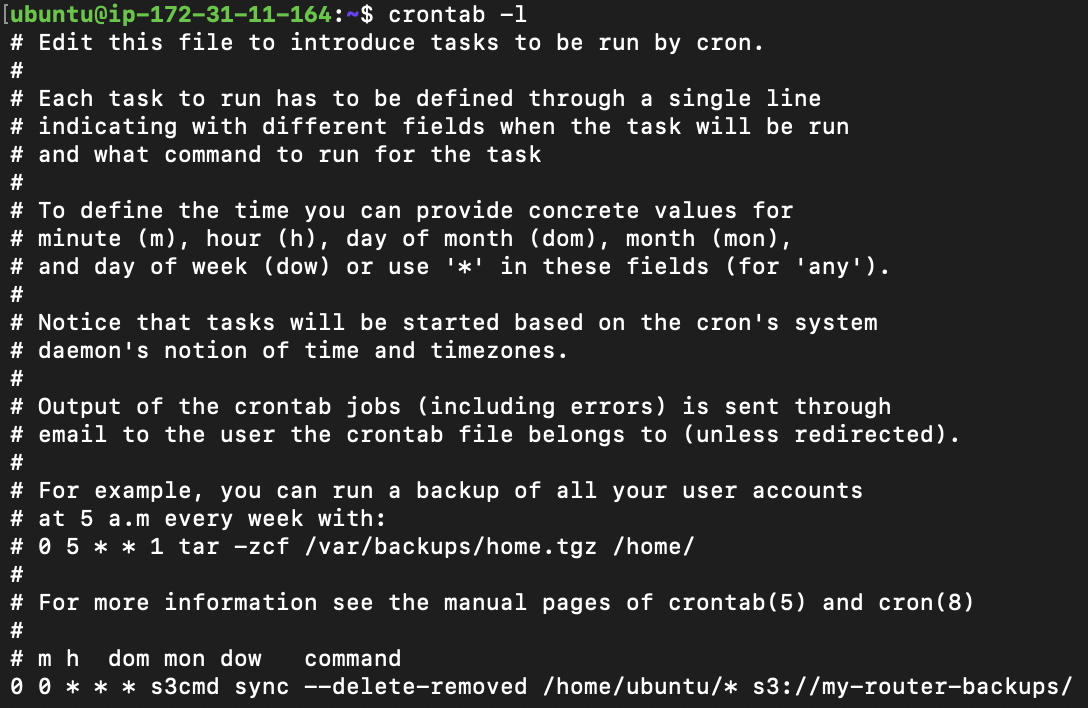


*Screenshot 8: Transferring files to S3 bucket*

1. What is another way of transferring data to your Amazon S3 bucket? [**2 points**]

Another method to transfer data to an Amazon S3 bucket is by using the AWS Command Line Interface (AWS CLI) or uploading files directly through the S3 dashboard in the AWS Management Console.

1. Create a **cronjob** to sync every night. Paste a screenshot of the **cronjob** created. [**5 points**]



*Screenshot 9: Creating cron job*

**0 0 \* \* \*** → Runs at midnight (12:00 AM) every day.

**s3cmd sync** → Synchronizes local files to the S3 bucket.

**--delete-removed** → Deletes files from S3 if they are no longer present locally.

**/home/ubuntu/** → Local directory to sync.

**s3://my-router-backups/** → Target S3 bucket.

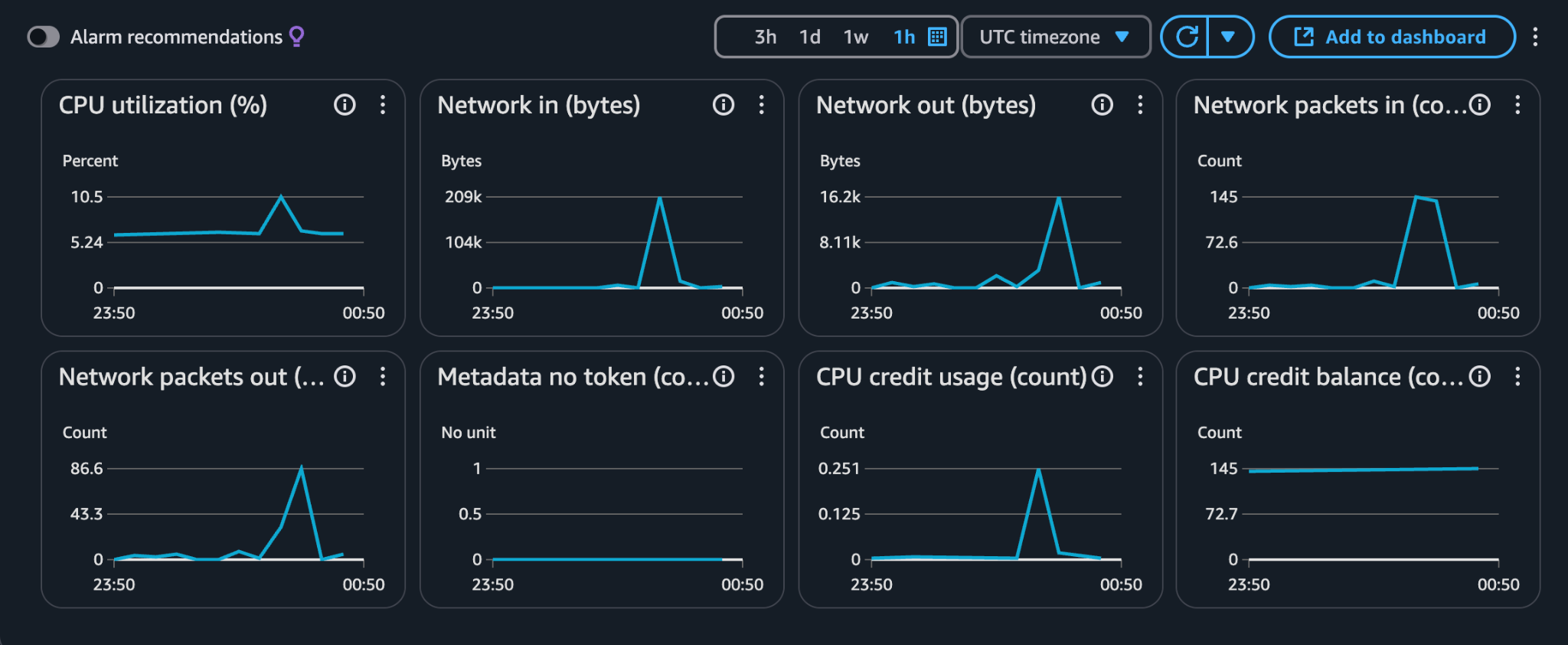
1. What is Amazon Glacier and how is it different from S3? [**3 points**]

Amazon S3 Glacier (now called Amazon S3 Glacier and S3 Glacier Deep Archive) is a low-cost cloud storage service designed for long-term archival and backup of infrequently accessed data. It offers high durability at a much lower cost than standard S3 storage. Amazon S3 is a general-purpose object storage for frequently accessed data whereas Amazon Glacier is for long-term archival storage for infrequently accessed data. Retrieval in S3 take a few milliseconds whereas it can take minutes or hours to retrieve from Glacier.

# Part 4

# Objective 4.1 – Monitoring using CloudWatch

# Click on any of the running EC2 instances and check the monitoring tab for EC2 metrics. What are the key metrics that you see and why is this important for an organization? [5 points]

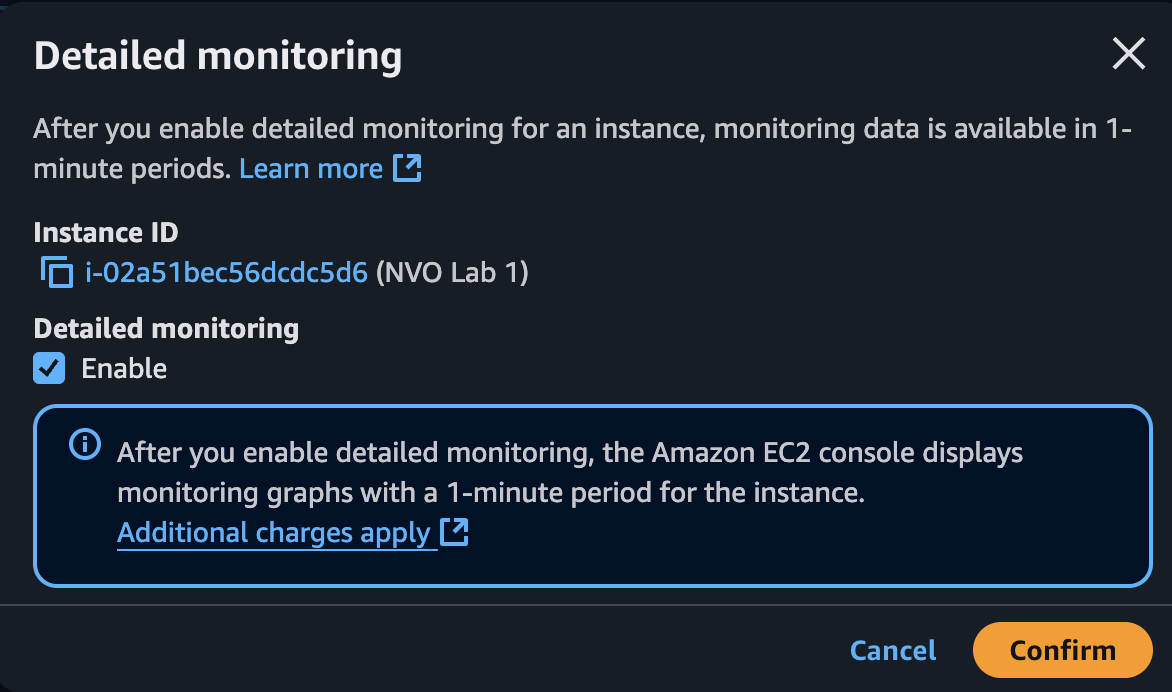


*Screenshot 10: Cloudwatch dashboard*

| **Metric** | **Description** | **Importance** |
| --- | --- | --- |
| **CPU Utilization (%)** | Percentage of the instance’s CPU capacity being used. | Helps monitor workload, detect spikes, and optimize scaling. |
| **Network In (bytes)** | Total data received by the instance. | Ensures proper network traffic flow and helps identify bottlenecks. |
| **Network Out (bytes)** | Total data sent from the instance. | Helps track outbound traffic and detect abnormal spikes. |
| **Network Packets In (count)** | Number of network packets received. | Helps detect unusual network activity or potential DDoS attacks. |
| **Network Packets Out (count)** | Number of network packets sent. | Indicates network performance and outbound traffic volume. |

The metrics help organizations in various ways.

1. Helps optimize resource usage (CPU, memory, disk).
2. Enables early detection of performance issues.
3. Helps automate scaling (e.g., auto-scaling based on CPU utilization).
4. Improves cost efficiency by identifying underutilized resources.
5. Supports security monitoring (unusual spikes in network traffic could indicate attacks).
6. Enable detail monitoring for CloudWatch metrics. What is the difference between Basic monitoring and Detailed monitoring? [**5 points**]



*Screenshot 11: Enabled detail monitoring*

Basic Monitoring collects metrics at 5-minute intervals and is available for free. It is suitable for low-priority workloads where real-time monitoring is not critical. This option is useful when occasional performance insights are sufficient, such as for development environments or non-critical applications.

Detailed Monitoring, on the other hand, provides 1-minute interval metric collection. It comes with an additional cost per instance but is beneficial for applications that require real-time performance tracking and auto-scaling decisions. It is particularly useful in production environments where quick responses to performance spikes are necessary.

1. Create a new **CloudWatch alarm** to monitor **average CPU utilization**. The alarm should take effect when average CPU utilization is greater than a user-defined threshold. Alarm should send an **E-Mail** to the recipients entered during the alarm configuration. Paste a screenshot of the alarm created and the Email that you received. [**10 points**]

[**NOTE:** For simplicity, enter the threshold to be less than 1%. Or you can use the [Linux stress tool](https://www.tecmint.com/linux-cpu-load-stress-test-with-stress-ng-tool/) to generate artificial stress on your instance to increase its CPU utilization.]

Steps to Create an Alarm

Go to AWS Console → Open CloudWatch.

Click on Alarms → Create Alarm.

Select a Metric → Choose EC2 → Per-Instance Metrics.

Select CPU Utilization for your instance.

Click Select Metric.

Set the threshold:

Threshold type: Static

Condition: Greater than 1%

Click Next → Set actions.

Set Notification:

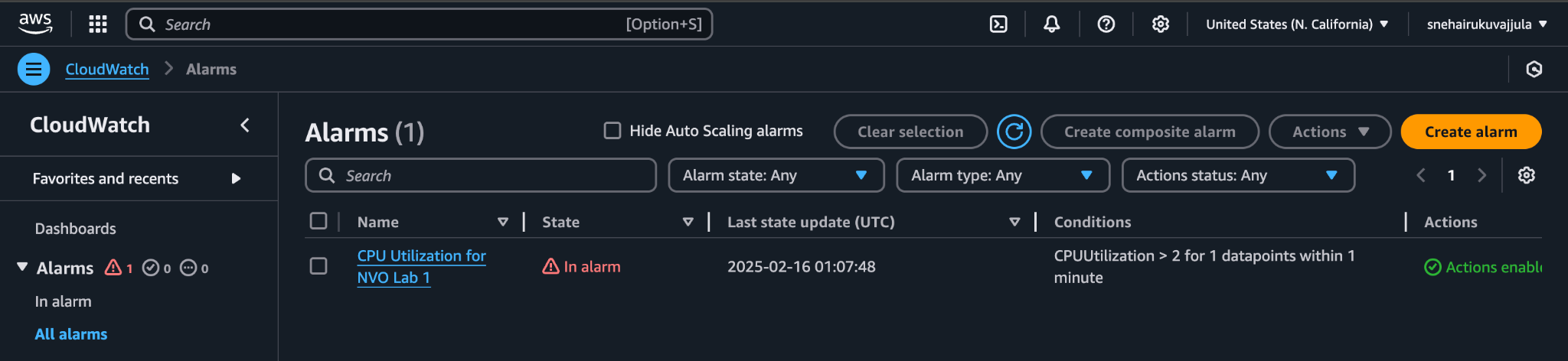
Send a notification to an SNS topic.

If an SNS topic doesn’t exist, create one.

Enter an email address to receive alerts.

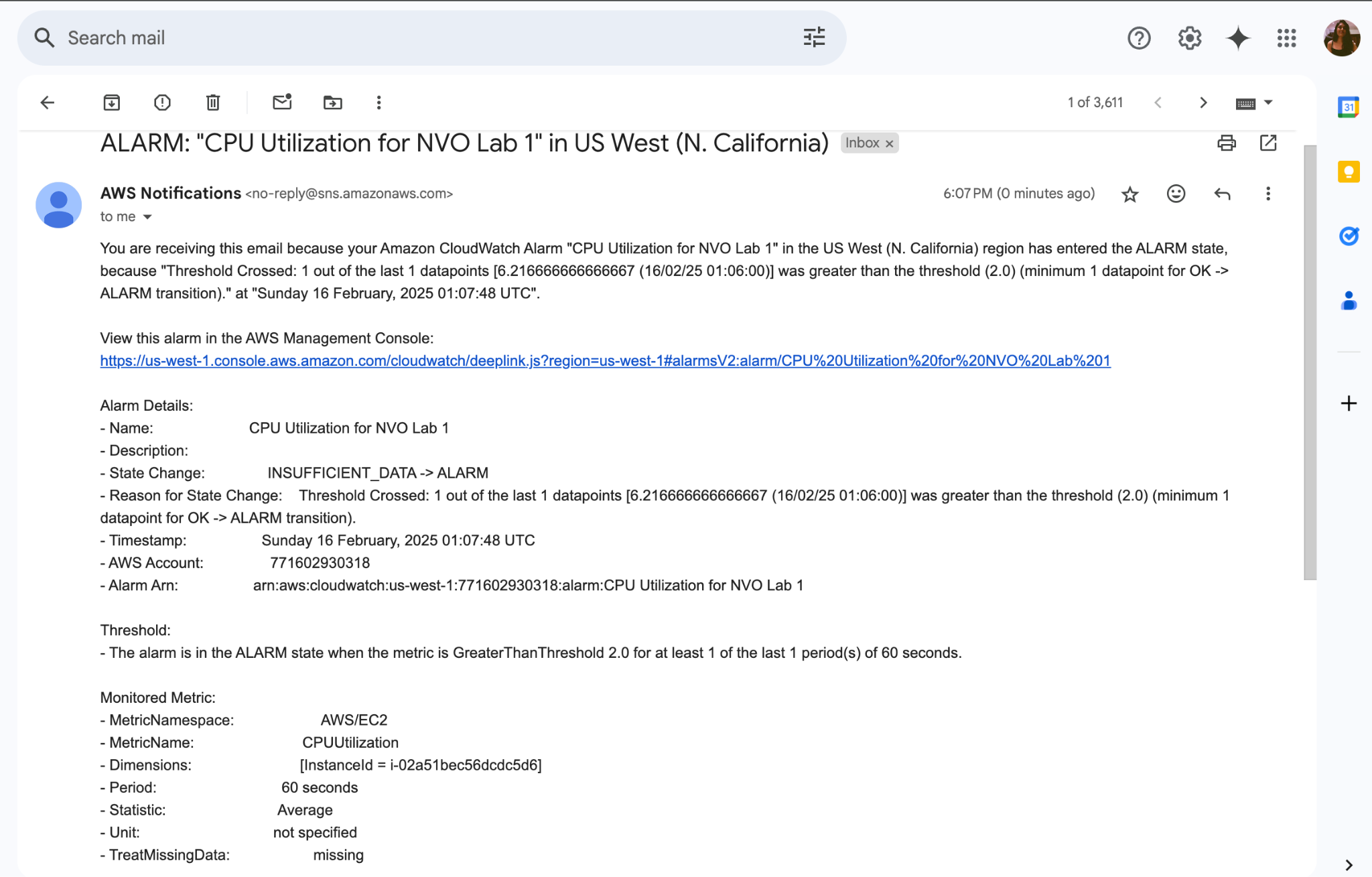
Click Create Alarm.

Now, CloudWatch will send an alert when CPU usage exceeds 1%.





*Screenshot 12: Cloudwatch Alarms*



*Screenshot 13: Cloudwatch alarm sending an alert to my Email ID*

1. Which is the service used by CloudWatch to send out E-Mail notifications? [**5 points**]

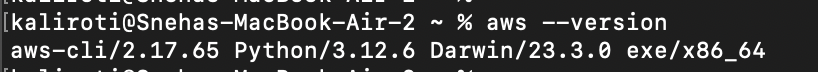
Amazon Simple Notification Service (SNS) is the service used by CloudWatch to send out email notifications when an alarm is triggered.

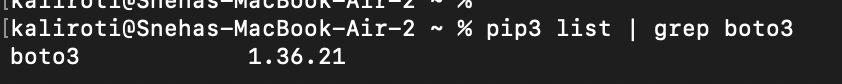
# Part 5

# Objective 5.1 Setting up BOTO3 for AWS resource automation

1. Install Boto 3 on your machine using “sudo -H pip install boto3” to download the required packages.
2. Before we start using Boto, it is mandatory to setup the necessary authentication to the AWS management console. In order to do this, we would need to download the AWS CLI and put in the AWS Access Keys which we have already downloaded in objective 1.3.
3. You may install the awscli for ease of authentication using “sudo -H pip install awscli” command. Further steps to achieve this are found in the below mentioned link: <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html#installation>.
4. If you opt to not set up one-time authentication with AWS CLI, ensure you add Var\_name = Session (aws\_access\_key\_id = ' ', aws\_secret\_access\_key = ‘ ‘ , region\_name = ' ') in your python code for further objectives.

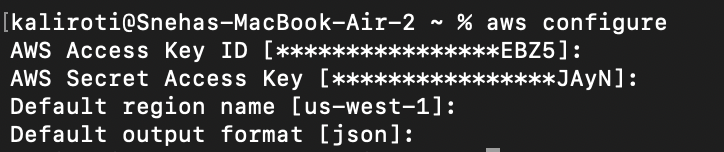
Installed awscli and boto3





*Screenshot 14: Installed aws cli and boto3*

Configured AWS Credentials on my Mac machine using `aws configure`



*Screenshot 15: Configured awscli*

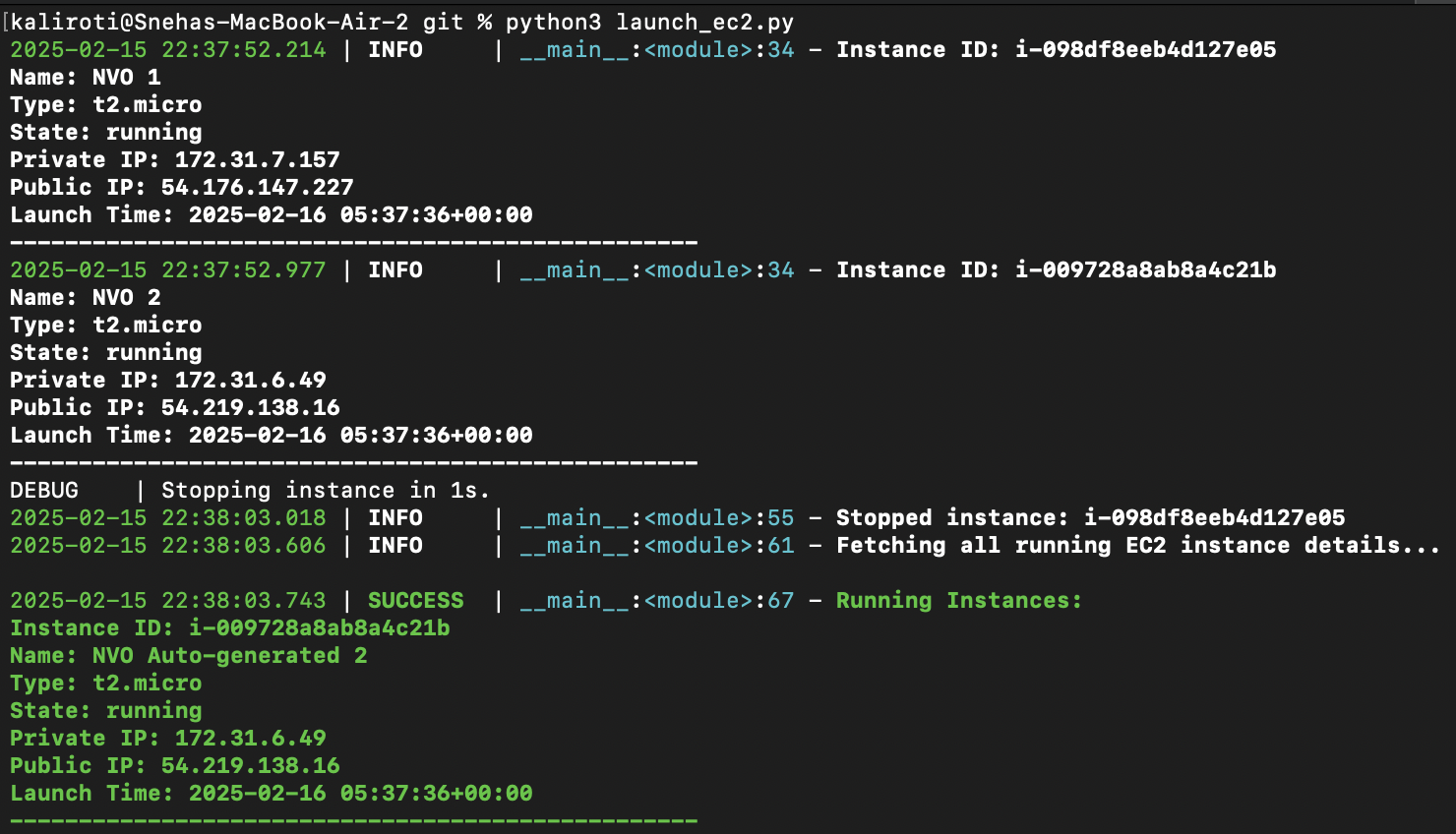
# Objective 5.2 Controlling EC2 using Boto3

1. Write a python script to launch two new EC2 instances, stop one of the instances, and then fetch the details of all instances using the Boto3 module. Sample output:

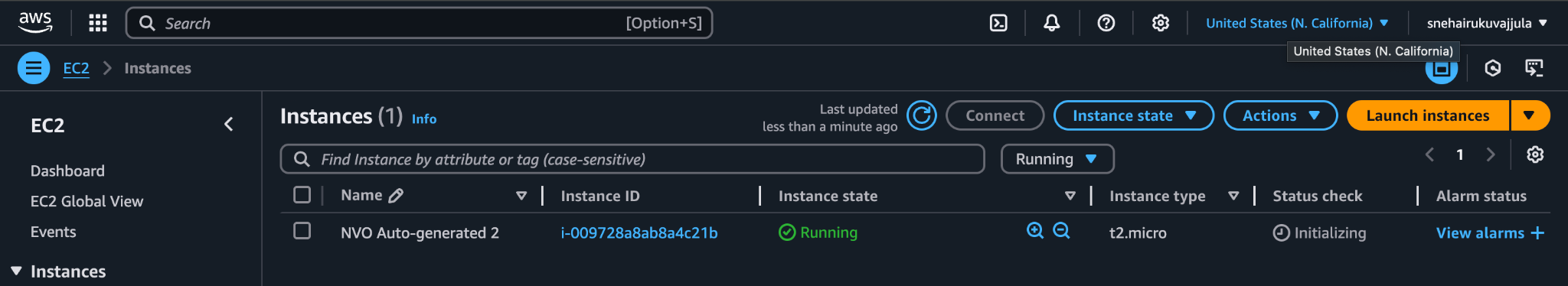
**[Instance Id] [Instance\_type] [Instance\_ip\_address] “Running/Stopped”**

[Hint: refer <https://boto3.readthedocs.org/en/latest/guide/migrationec2.html>]

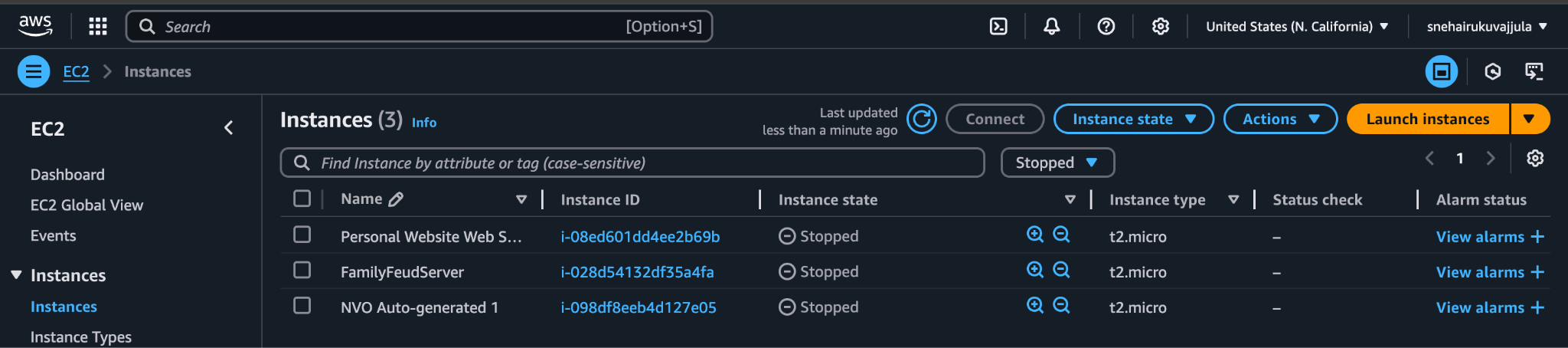
1. Submit the .py file that you created to accomplish this objective, screenshots showing the new instances created on the console, and the instance details as specified in the sample output. [**30 points**]

**Python code output:**

**Running instances:**

****

**Stopped instances:**

****

# Objective 5.3 Fetching Cloudwatch metrics using Boto3

1. Write a python script to create a new AWS session using access keys (refer objective 1), create a cloudwatch session, and fetch the following metrics for one running EC2 instance over a specific time period (at least 30 minutes): Status\_Check, CPU\_Utilization, Network\_In and Network\_Out. Sample output:

**Instance ID: <value>**

**Status Check: <value>**

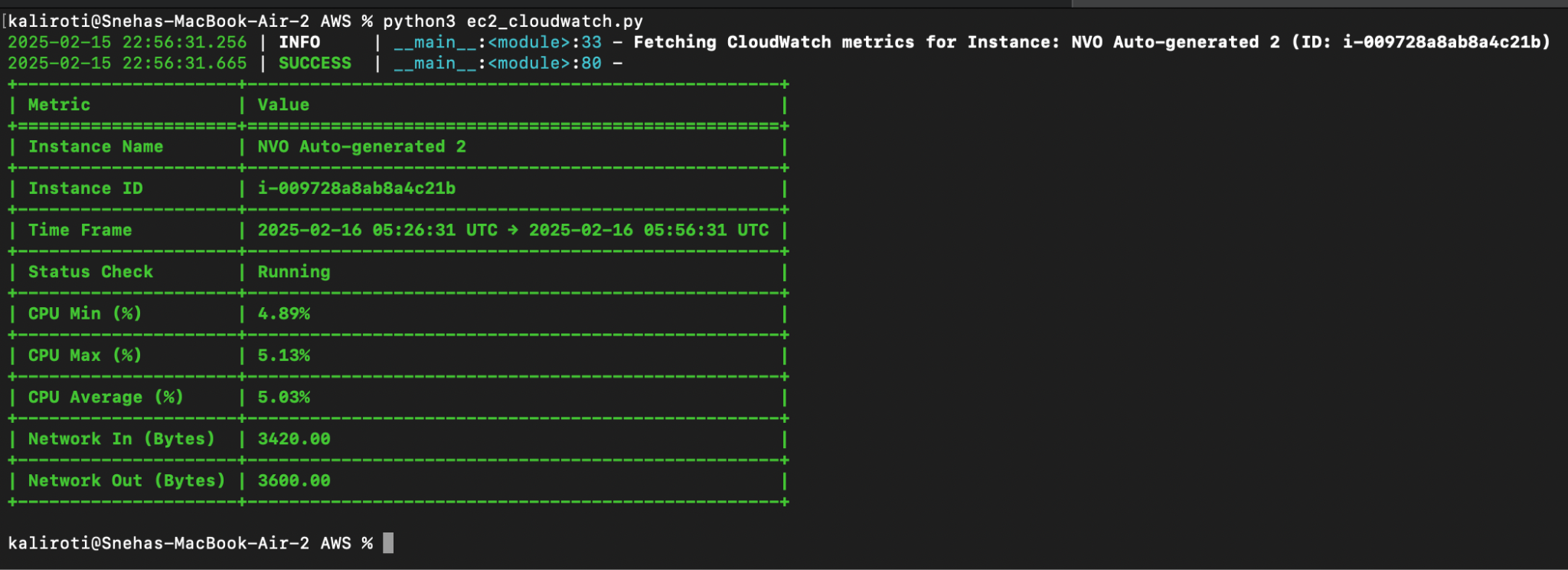
**CPU Utilization: <value>**

**Network In: <value>**

**Network Out: <value>**

[Hint: <https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/cloudwatch.html#CloudWatch.Client.get_metric_statistics>]

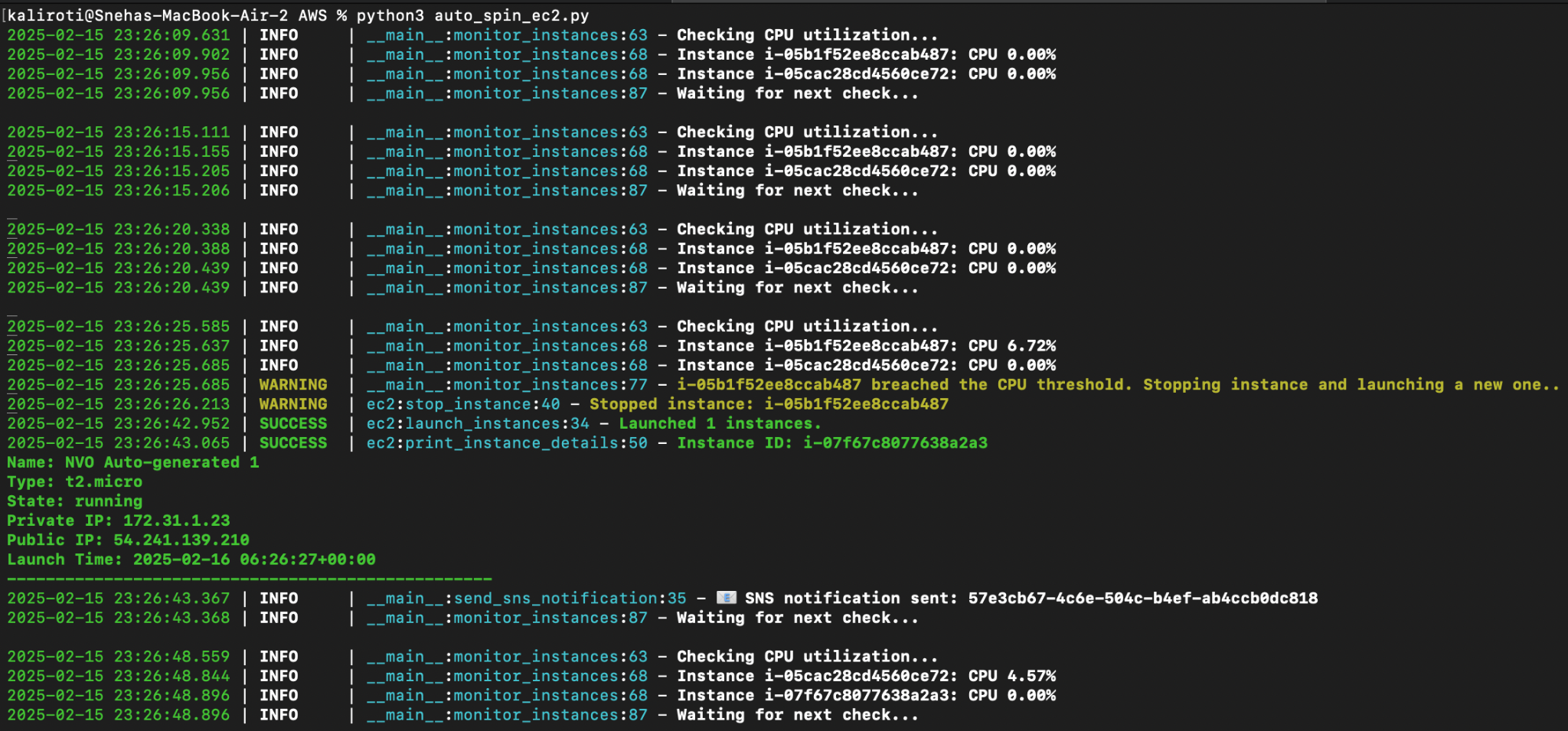
1. Submit the .py file that you created to accomplish this objective and screenshots of the details as specified in the sample output. [**20 points**]



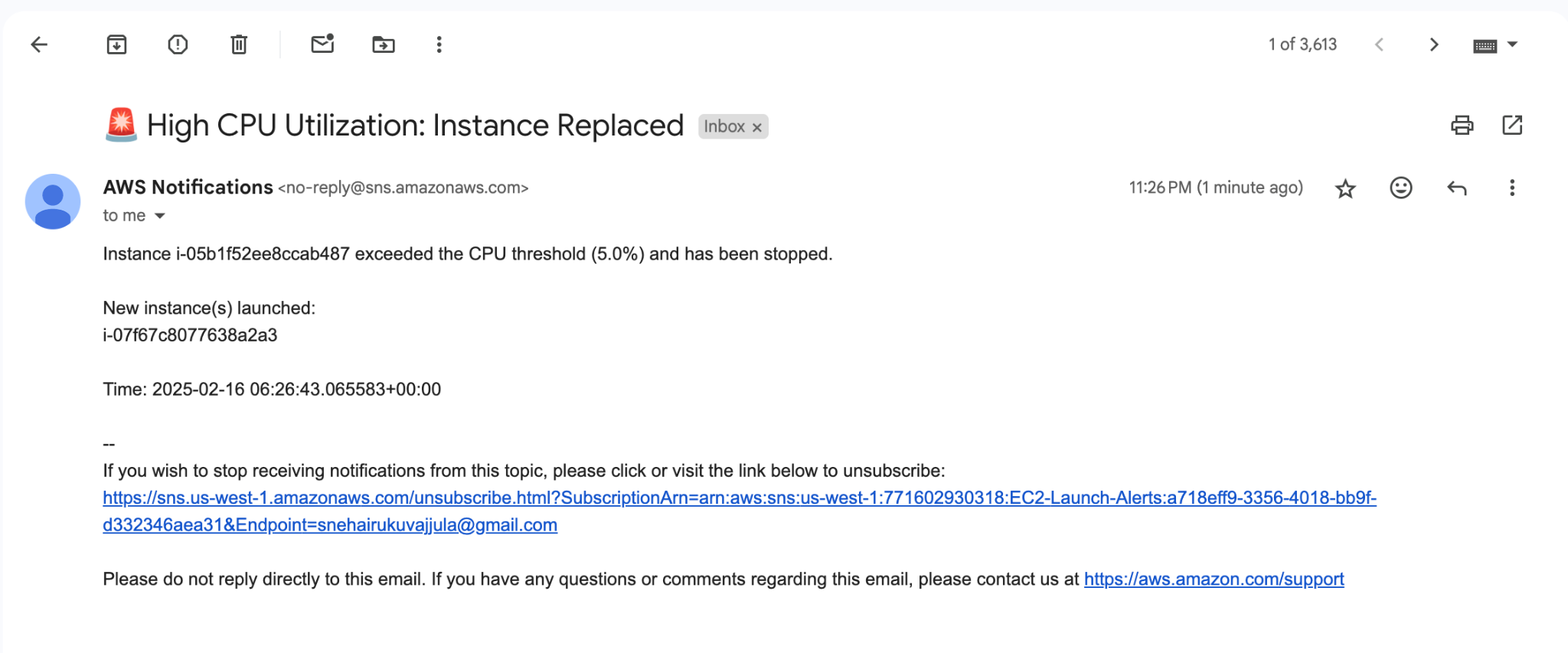
# Objective 5.4 Spinning new instances based on CPU utilization

Write a python script that uses Boto3 to continuously fetch CPU utilization of two running EC2 instances. When a specific threshold is reached, the script should automatically shut the instances down, spin up identical new instances and send out an alert email to your email id. Submit the .py file that you created to accomplish this and relevant screenshots. [**20 points**]

The code running:



An email alert sent to my email ID:



*Screenshot 16: CPU utilization alert sent to my email ID*

Reflection:

1. Now that you have learnt the basics, what do you think are the most important reasons for an organization to use AWS? [**5 points**]

Most Important Reasons for an Organization to Use AWS:

Scalability & Flexibility – AWS allows businesses to scale resources up or down based on demand, ensuring cost efficiency and performance optimization.

Cost-Effective – With AWS’s pay-as-you-go model, organizations avoid large upfront infrastructure costs and only pay for what they use.

Global Reach & Reliability – AWS operates in multiple geographic regions, ensuring high availability and disaster recovery capabilities.

Wide Range of Services – AWS offers compute (EC2), storage (S3), networking (VPC), AI/ML, analytics, databases, and serverless computing (Lambda) to meet diverse business needs.

1. Suggest any other AWS modules that you would like to learn about. Why?

I’d like to learn more about AWS Lambda to learn how to build applications that run without provisioning or managing servers. Further, I’d also like to learn AWS Kubernetes (EKS) to explore container orchestration and how to efficiently run, scale, and manage containerized applications.

1. Suggest any other public cloud platforms that you would like to learn about. Why?

I’d like to explore two other major cloud platforms: Microsoft Azure, which is widely adopted in enterprise environments and provides seamless integration with Microsoft products such as Active Directory and Office 365, and Google Cloud Platform (GCP), which is recognized for its AI, machine learning, and data analytics capabilities, including BigQuery and TensorFlow integration.

# Total Points \_\_\_\_\_\_\_\_\_\_\_\_ / 207