### **Report: Automated Deployment with Monitoring**

**Mayur Jaiswal**

**D15B**

**26**

#### **Objective:**

The goal of this experiment is to implement an automated CI/CD pipeline using Jenkins to deploy a web application on an Amazon EC2 instance. Additionally, configure Nagios to monitor the deployed application’s availability and ensure that the deployment process is running smoothly.

### **Abstract:**

In this project, we will set up a Jenkins Continuous Integration/Continuous Deployment (CI/CD) pipeline that automates the deployment of a web application to an EC2 instance. Post-deployment, Nagios will be configured to monitor the HTTP status of the deployed application. This approach is essential for modern DevOps practices, ensuring that applications are deployed quickly and monitored for uptime, all with minimal human intervention.

### **Prerequisites:**

#### **1. Jenkins Installation**

Jenkins will be installed to automate the build and deployment process.

**Ubuntu Installation Commands**:  
1. sudo yum update -y

2. sudo yum install java-11-openjdk-devel -y

3. sudo wget -O /etc/yum.repos.d/jenkins.repo <https://pkg.jenkins.io/redhat-stable/jenkins.repo>

4. sudo rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io.key>

5. sudo yum install jenkins -y

sudo systemctl start jenkins

sudo systemctl enable jenkins

6. aws configure

7. aws s3api create-bucket --bucket myproject-case1 --region ap-south-1 --create-bucket-configuration LocationConstraint=ap-south-1

**Commands on CMD:**

scp -i "C:\Users\Hp\Downloads\mayur-key.pem" -r "C:\Users\Hp\OneDrive\Documents\exxp10\practice" ec2-user@ec2-35-154-30-48.ap-south-1.compute.amazonaws.com:/home/ec2-user/

#### **2. AWS EC2 Setup**

Launch an EC2 instance that will host the web application. Ensure you have a security group allowing HTTP (port 80) and SSH (port 22) traffic.

Steps:

1. Go to **AWS Console**.
2. Launch an EC2 instance with Ubuntu or Amazon Linux.
3. Choose **t2.micro** for free-tier usage.
4. Set up a security group that allows inbound rules for HTTP and SSH traffic.

#### **3. AWS Credentials**

Ensure you have created an **IAM User** in AWS with permissions for EC2 and S3 services. Store the **Access Key** and **Secret Key** for use in Jenkins.

You can find and create keys in the **IAM Management Console** under **Access Management -> Users**.

### **System Design:**

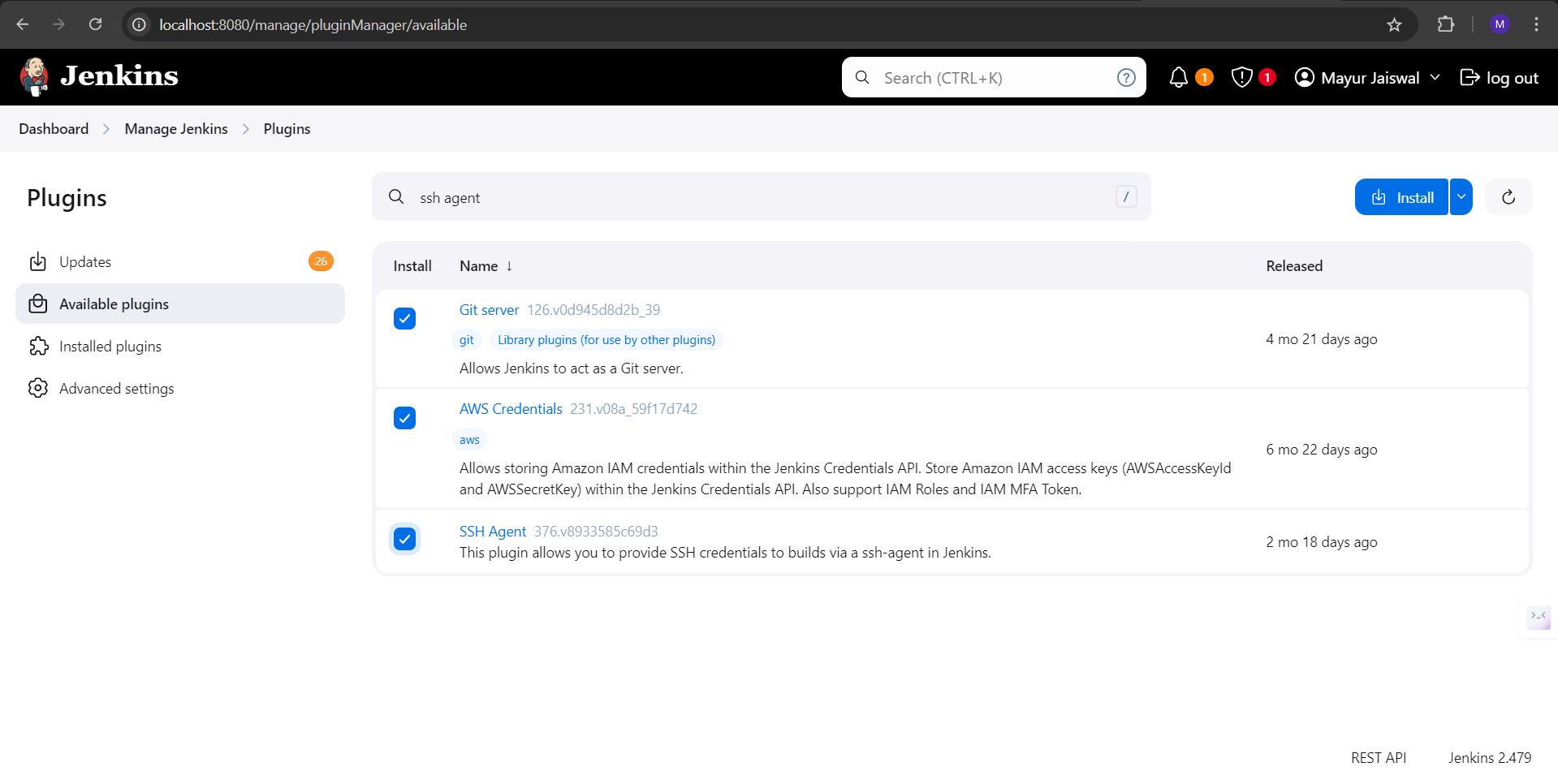
The overall system is designed in two stages:

1. **CI/CD Pipeline using Jenkins**: Jenkins will be used to build and deploy the web application on an EC2 instance.
2. **Monitoring with AWS CloudWatch**: CloudWatch will continuously monitor the availability of the web application by sending HTTP requests and verifying the status code.

### **Steps for Implementation:**

1. Install necessary plugins:

* Go to **Manage Jenkins** > **Manage Plugins**.
* Install **Git**, **SSH Agent**, and **AWS** plugins.



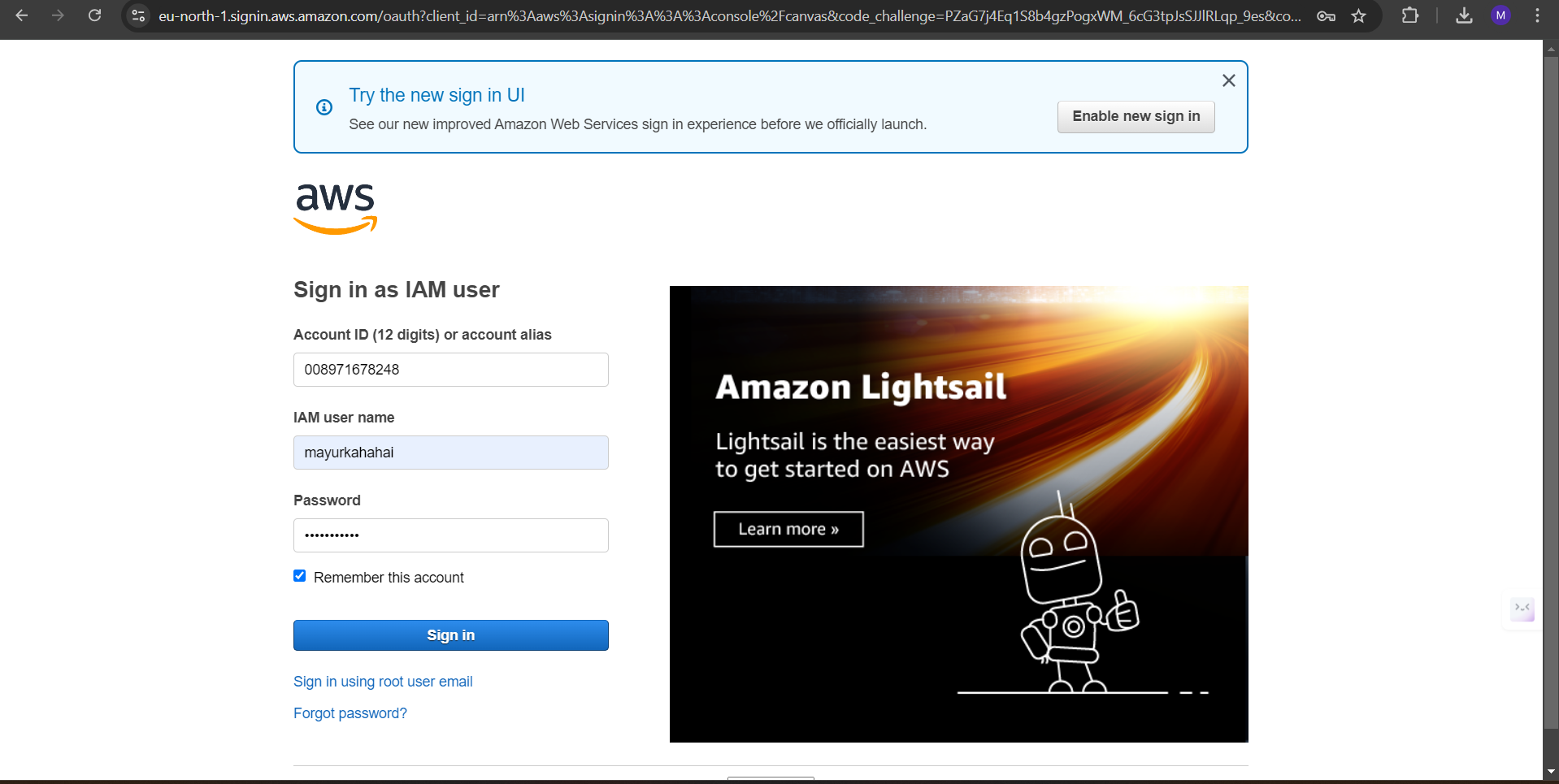
##### **Install AWS CLI**

##### Download and install the **AWS CLI** for Windows.

Open **Command Prompt** and configure AWS credentials:  
 aws configure

Provide your **Access Key ID**, **Secret Access Key**, **region**, and **output format**.

You can obtain your credentials from the **IAM Management Console** in AWS.



**ERROR: I encountered an error while attempting to connect to my AWS EC2 instance because the existing security group did not permit inbound traffic on port 5666.** This port is often used for specific network protocols or applications. To resolve the issue, I created a new security group and configured it to allow inbound traffic on port 5666. After associating this new security group with the EC2 instance, I was able to successfully establish a connection.

1. **Create a Security Group:**

* Click the "Create Security Group" button.
* **Name:** Provide a meaningful name like groupforcasestudy
* **Description:** Enter a description, e.g., "Security group for Jenkins and Nagios monitoring."

**Add Inbound Rules:**

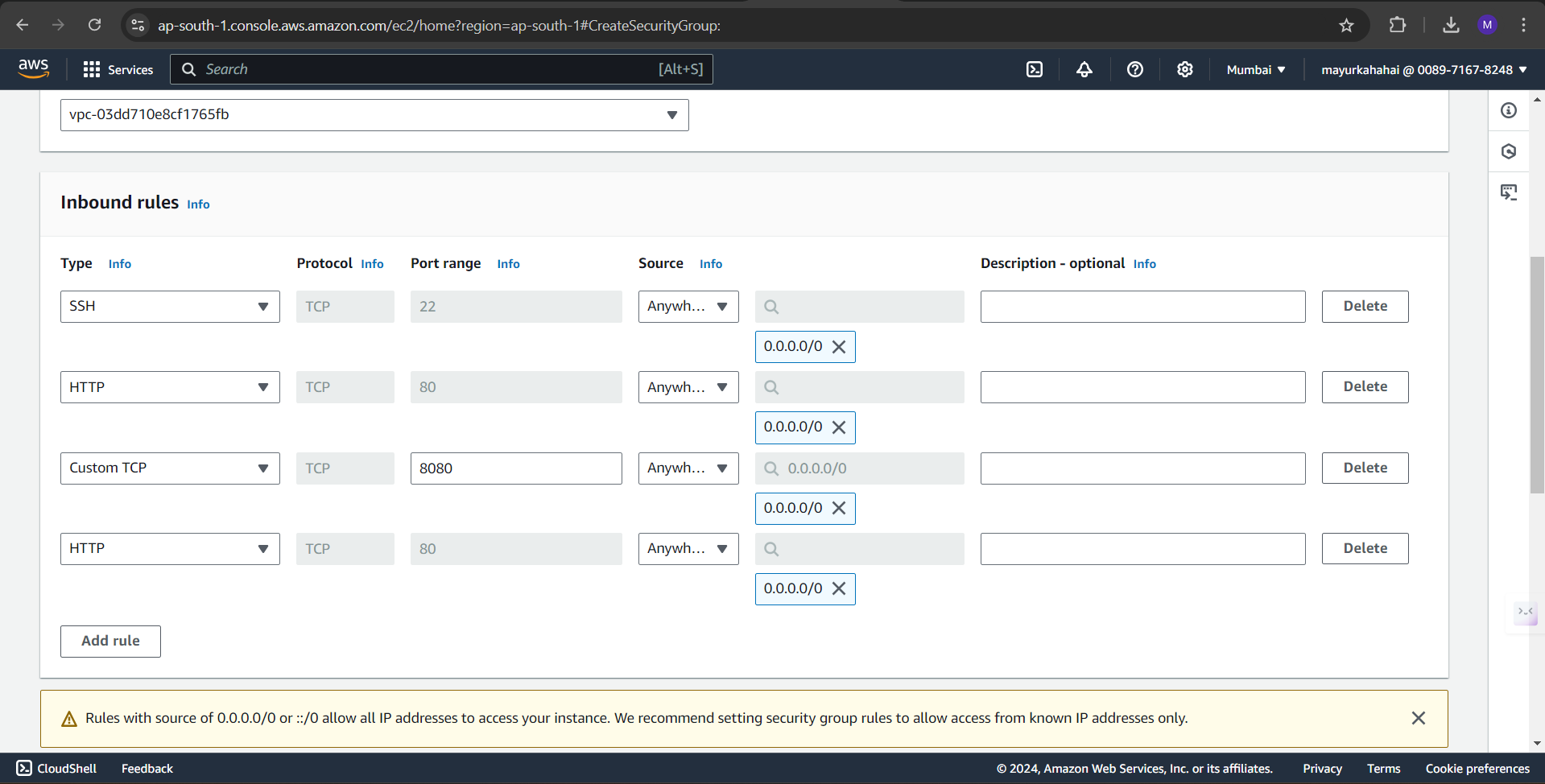
* **SSH:** Type: SSH, Protocol: TCP, Port: 22, Source: Your IP or Anywhere.
* **HTTP:** Type: HTTP, Protocol: TCP, Port: 80, Source: Anywhere (0.0.0.0/0).
* **Custom TCP for Jenkins:** Type: Custom TCP, Protocol: TCP, Port: 8080, Source: Anywhere (0.0.0.0/0).

**Add Outbound Rules:**

By default, outbound rules will allow all traffic. This is fine unless you need to restrict outbound traffic.

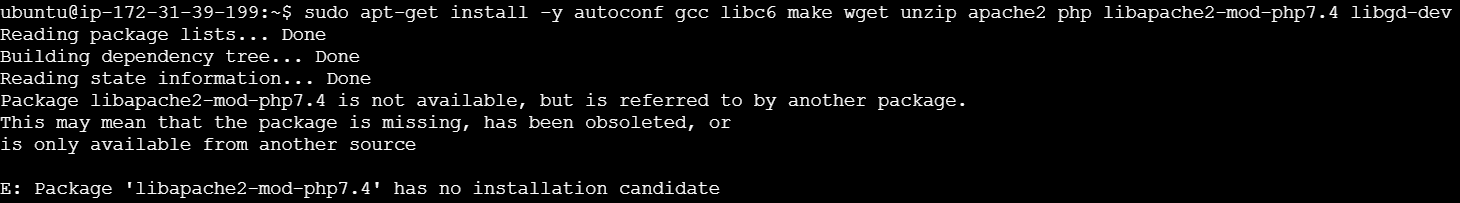
**Save and Assign Security Group:**

After creating the security group, assign it to your EC2 instance.

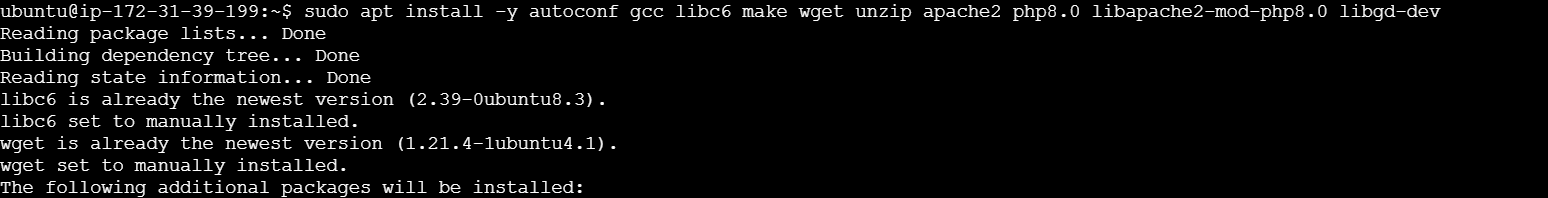


After creating an EC2 Instance, these commands are used to install dependencies:

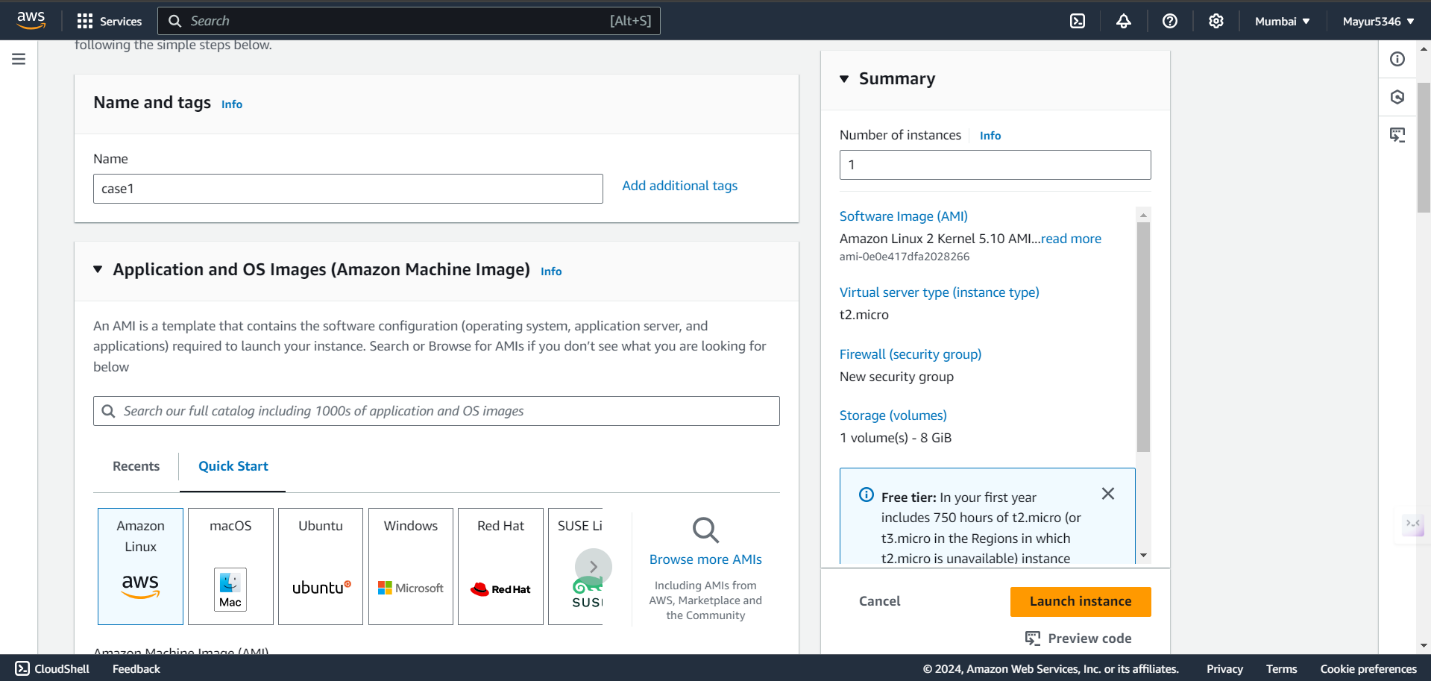
**ERROR:** the package libapache2-mod-php7.4 is no longer available. Since PHP 7.4 has reached its end of life, it has been removed from many repositories. You can use a more up-to-date version of PHP, such as PHP 8.0 or PHP 8.1, instead.



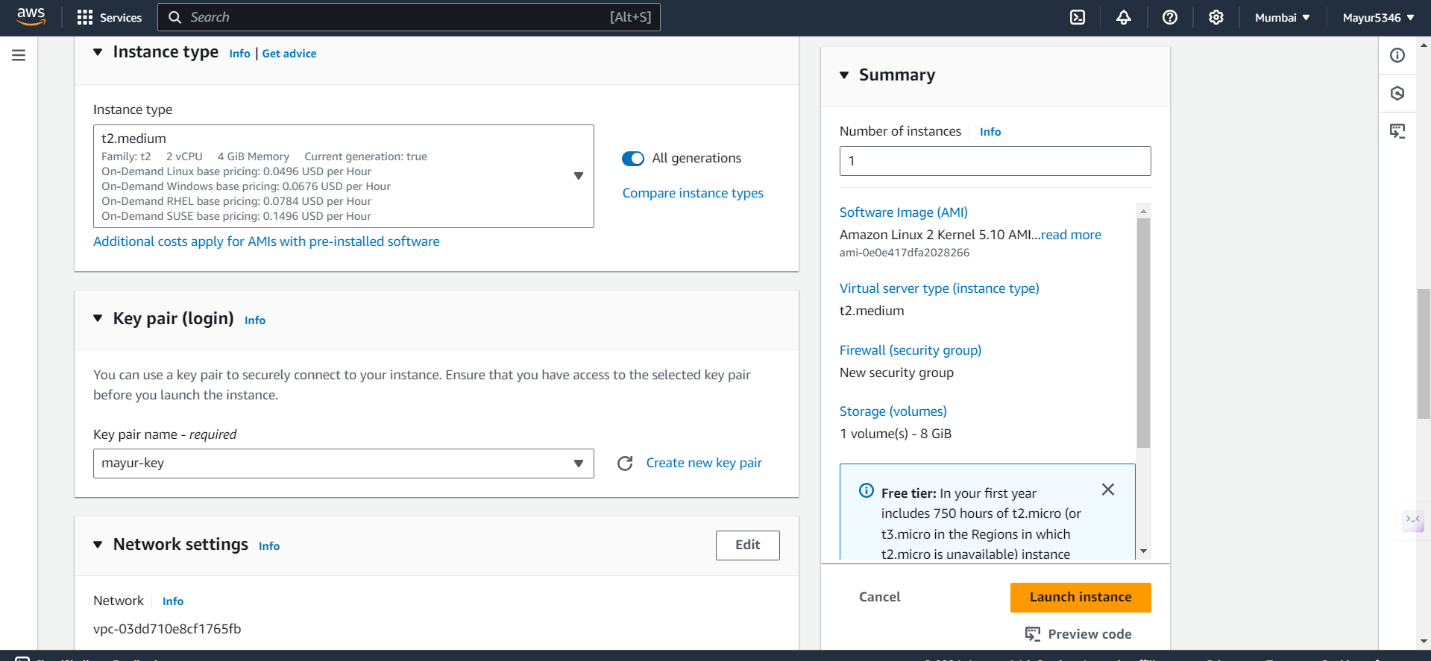
Solution:



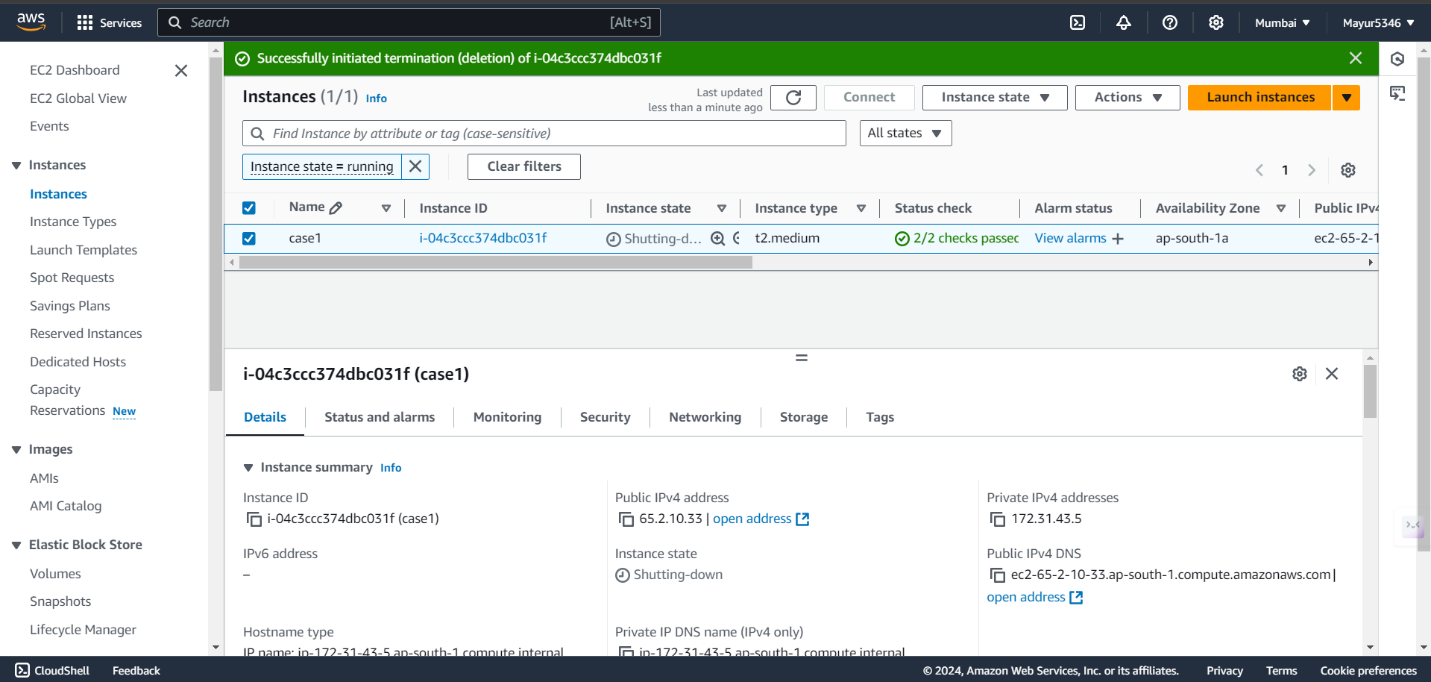
1. Create an EC2 Instance in **Amazon Linux** Image



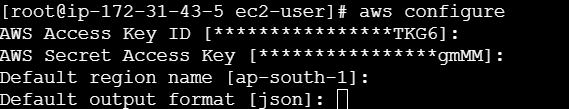
1. T2.medium Size of the instance would be sufficient for this experiment



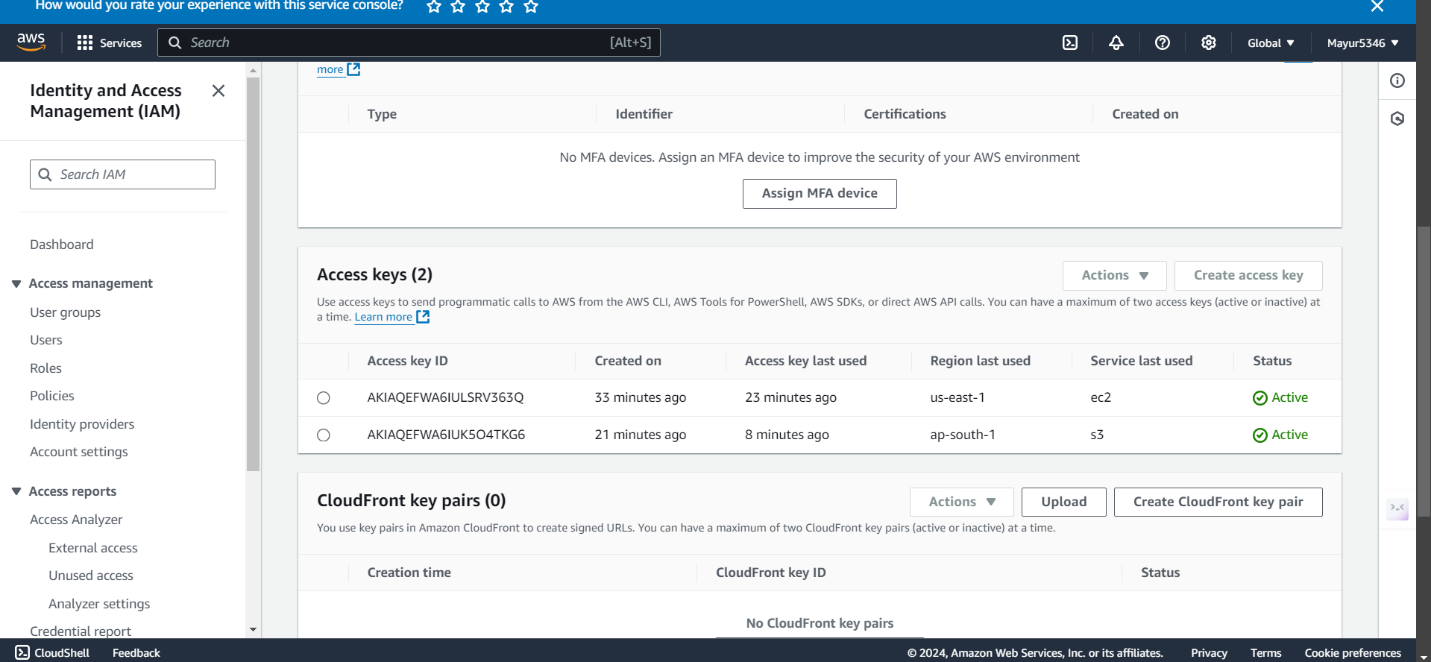
1. This is the EC2 Dashboard after creation



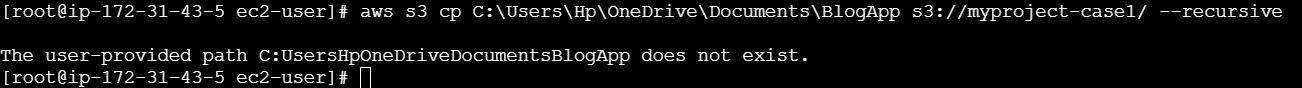
**Steps to configure AWS CLI**



1.To get the **Access Key** and **Secret Access Key**, go to Profile > Security Credentials > then create a new **Access Key**



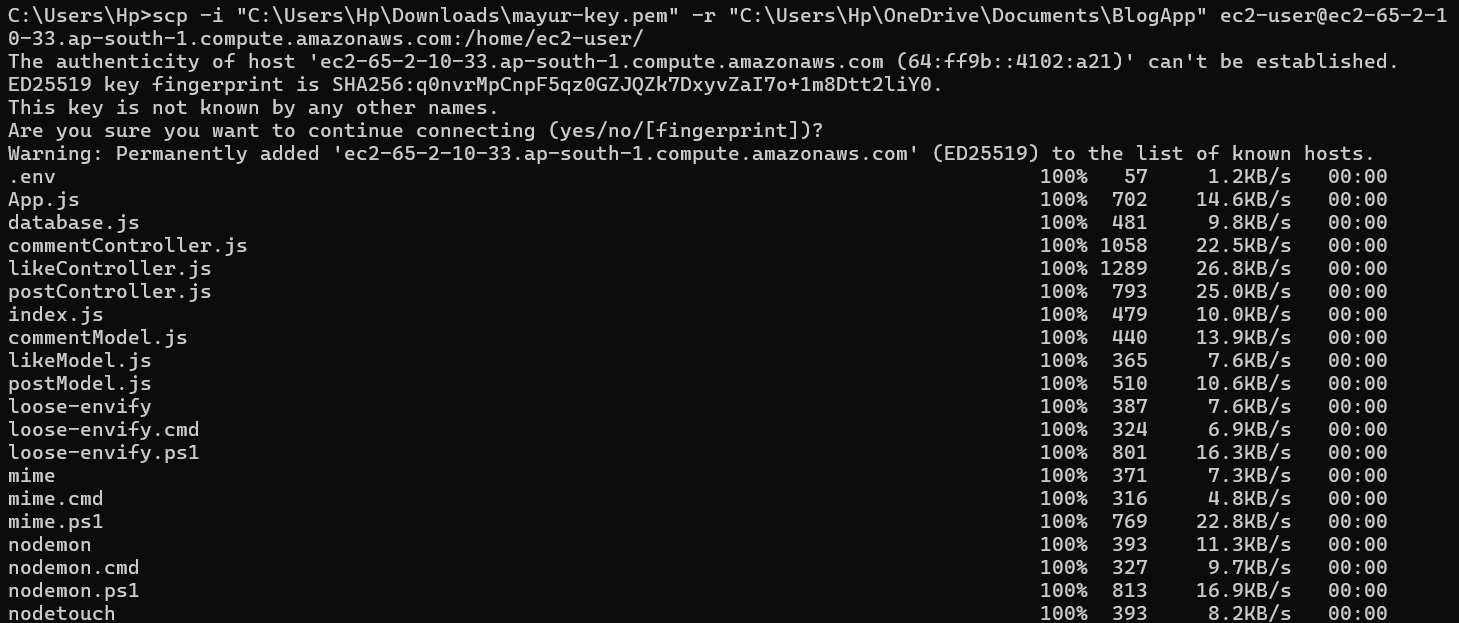
**ERROR**: The folder for monitoring has to be uploaded on an S3 bucket for Automated Deployment



**Steps to Upload Local Folder on Cloud**

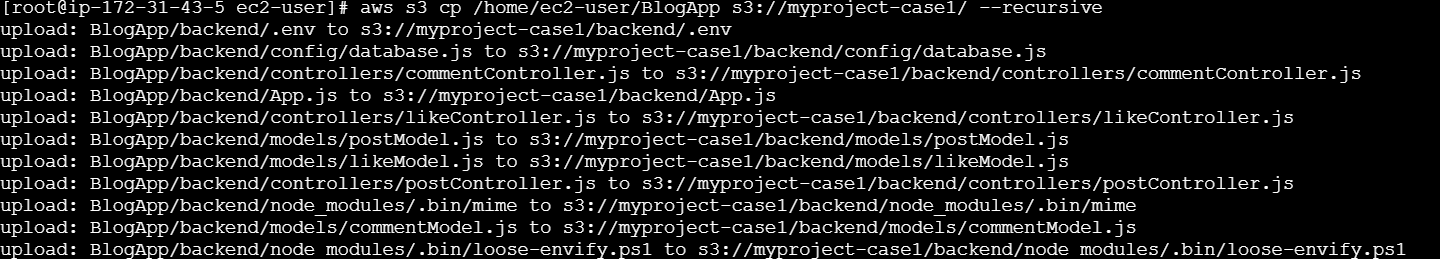
Run the following command in **CMD**

scp -i “C:\Users\Hp\Downloads\mayur-key.pem” -r “project path” ec2-user@ ec2-35-154-30-48.ap-south-1.compute.amazonaws.com:/home/ec2-user/



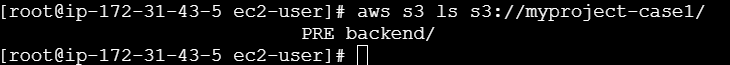
Run the following command on **EC2 Terminal**

aws s3 cp /home/ec2-user/practice s3://myproject-case1/ --recursive

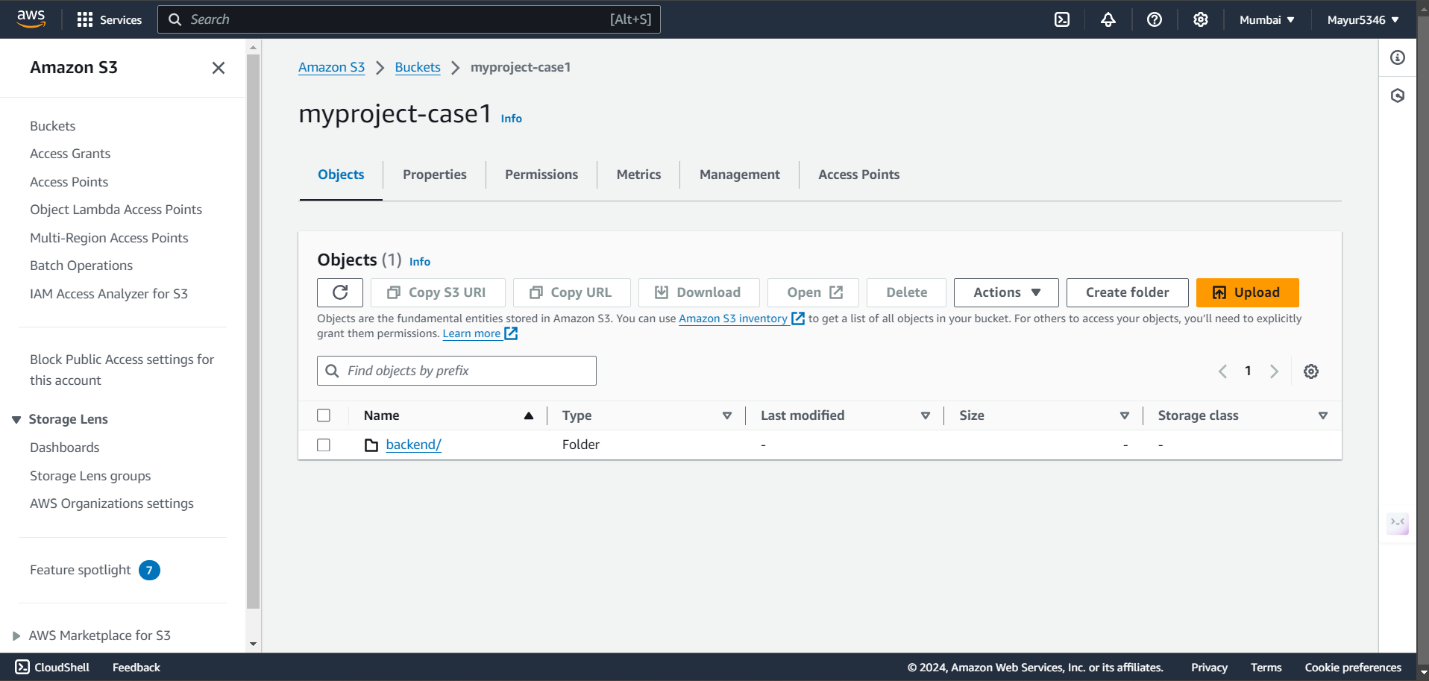


Verify the upload is successful by the following command

aws s3 ls s3://myproject-case1/



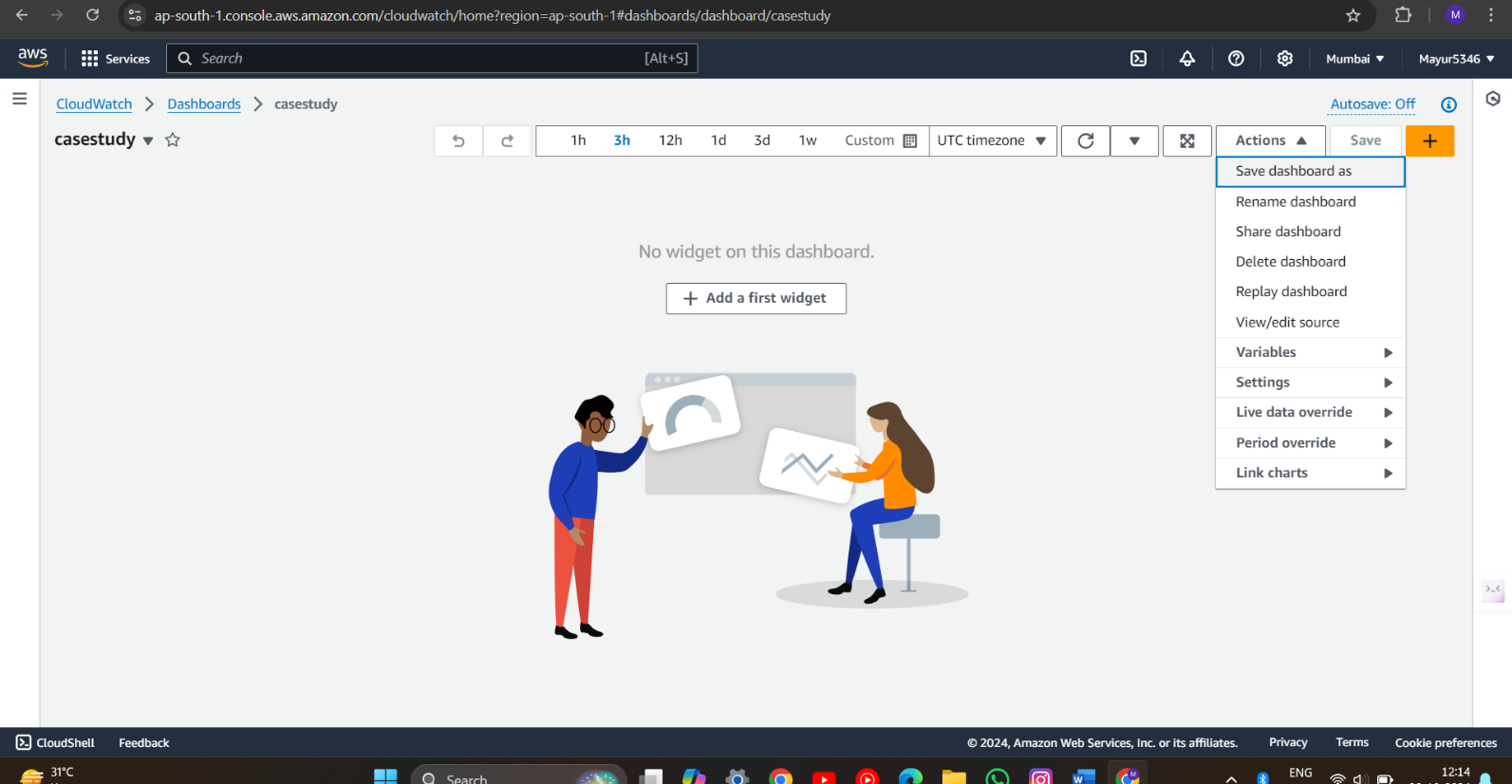
A new S3 bucket will be created automatically and the folder can be seen in the Bucket



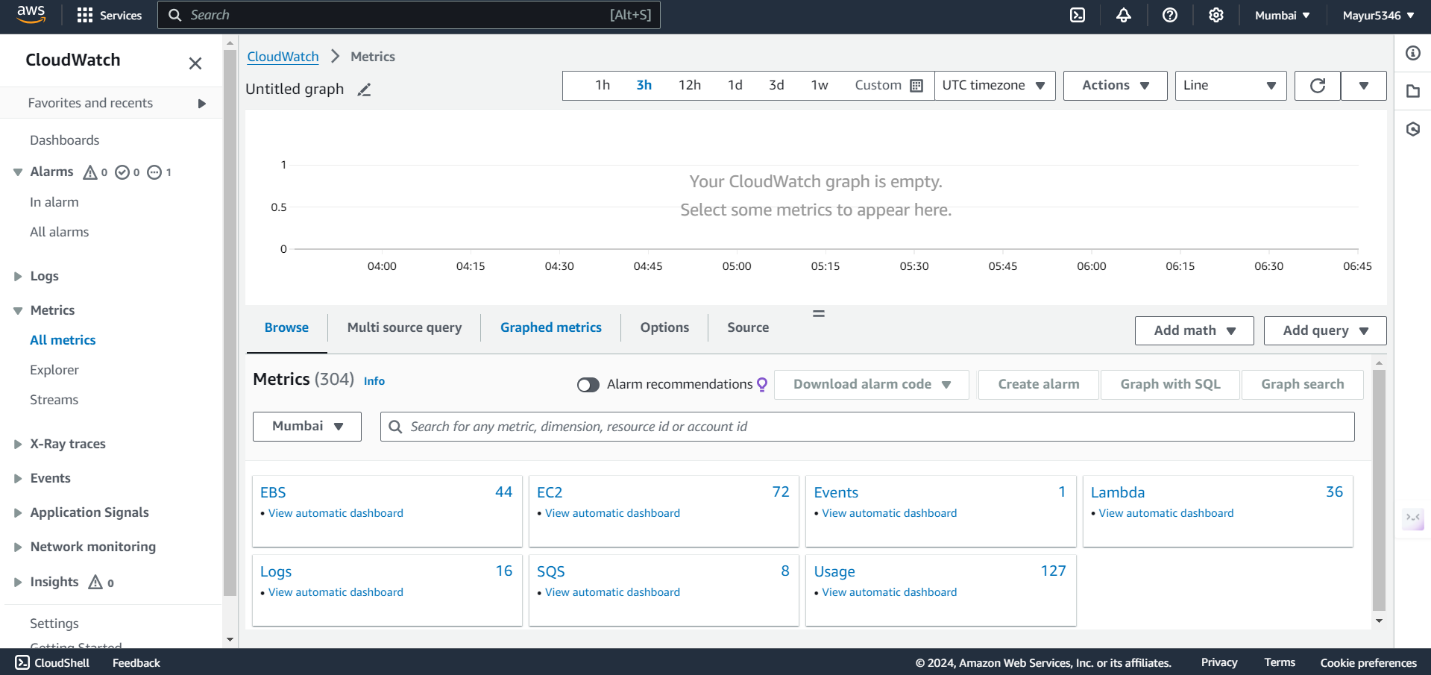
The Process of Automated Deployment using EC2 and Jenkins is successfully performed here, for Monitoring we are using **AWS inbuilt CloudWatch Dashboard** because Nagios was giving unresolvable errors at the time of integration.

**Steps to Enabling and Configuring Monitoring**

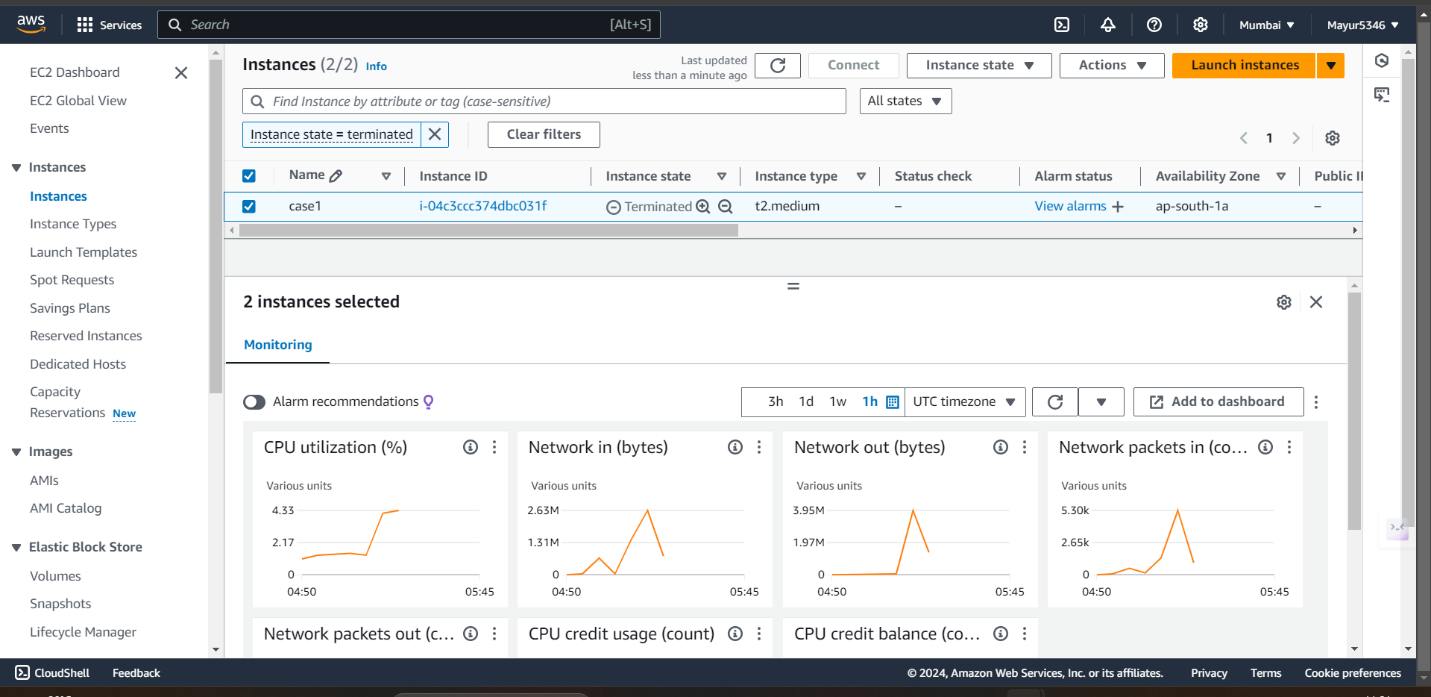
1. **Access CloudWatch Metrics**:
   * After launching the EC2 instance, go to the **CloudWatch** service from the AWS Console.
   * CloudWatch automatically collects metrics such as:
     + CPU utilization
     + Disk I/O
     + Network traffic
     + Memory usage (if enabled with the CloudWatch Agent)



From the dashboard, click on **Metrics** then **All Metrics:**



This the Dashboard to monitor CPU utilization, Network in, Network out, Network Packets, etc.:



**Steps for Cleaning Up Resources**

1. **Terminate the EC2 Instance**:
   * After completing the project, navigate to the EC2 dashboard and **terminate the instance** to avoid further costs.
   * Go to **Instances** > select the instance > click **Actions** > **Instance State** > **Terminate**.
2. **Delete Other Resources**:
   * If you used any additional resources like EBS volumes or S3 buckets for storage, ensure you delete them to avoid unnecessary charges.

**Conclusion**

This project provided a comprehensive introduction to creating and managing an EC2 instance on AWS, with a focus on enabling and configuring monitoring using CloudWatch. The primary goal was to launch an EC2 instance, monitor its performance through key metrics like CPU utilization, network traffic, and disk I/O, and set up CloudWatch alarms to respond to any anomalies or threshold breaches. By enabling detailed monitoring, we were able to capture metrics every minute, allowing for more granular insights into the instance’s performance and resource usage.

However, each obstacle provided an opportunity to dive deeper into AWS’s documentation and learn more about the platform. Referring to the **official AWS documentation** proved invaluable at various stages, helping to clarify complex steps, resolve configuration errors, and understand the full range of capabilities offered by AWS services. For example, issues with SSH access were resolved by referring to the documentation on security groups and public/private key management. Similarly, challenges with CloudWatch alarms were addressed by consulting the monitoring and alarms section of the AWS documentation, which provided step-by-step guidance on creating customized metrics and alarms.