**Static Web Application Deployment on AWS**

**Overview**

In this project, I have deployed a scalable and secure static web application on AWS. The application serves a simple "Hello World!" message through a basic HTML page, and I've followed best practices for security, monitoring, and scaling using AWS services. The project covers the following key elements:

- EC2 (Ubuntu Server)

- Elastic Load Balancer (ELB)

- Auto Scaling Group

- CloudWatch for Monitoring

- HTTPS Redirection with a Self-Signed SSL Certificate

The goal of this project is to automate the deployment of the web application using AWS services and a configuration management approach.

What I Did in This Project: -

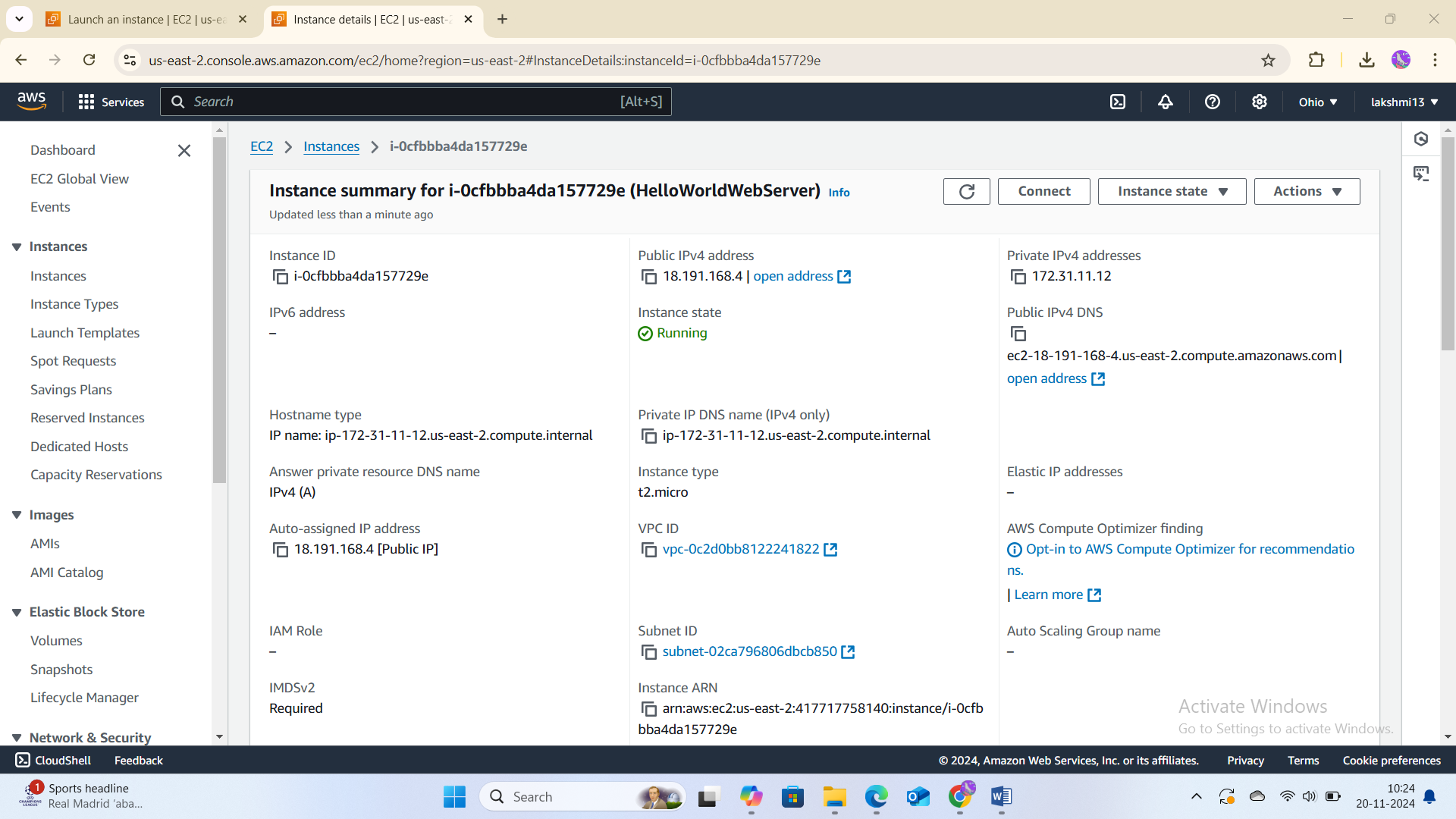
Step 1: Set Up EC2 Instance

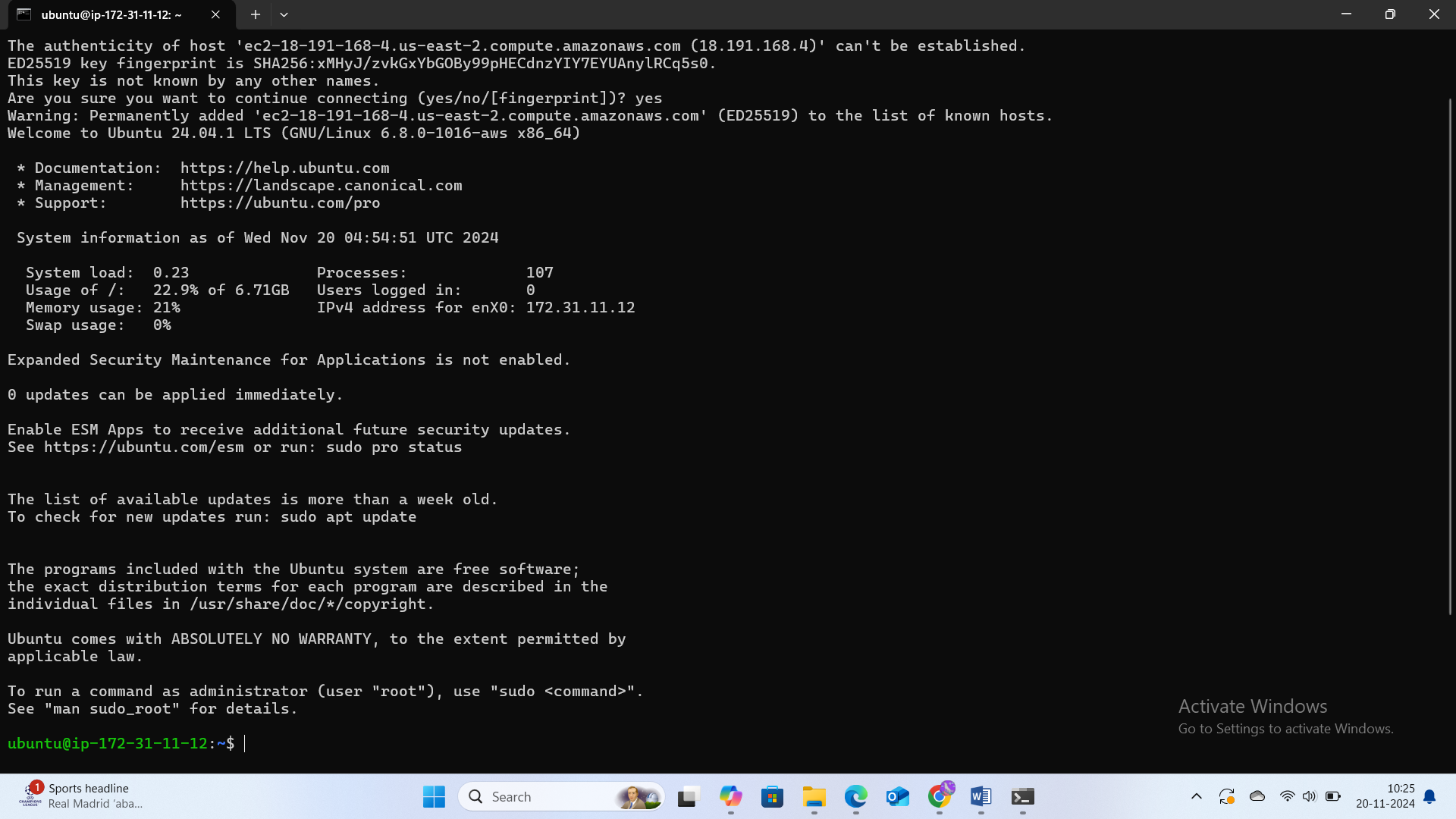
1. Created an EC2 instance with the Ubuntu Server AMI (Amazon Machine Image).

2. Configured the security group to allow:

- SSH (port 22) for remote access.

- HTTP (port 80) and HTTPS (port 443) for web traffic.





3. Connected to the EC2 instance using SSH and updated the system packages.

Step 2: Install and Configure Apache Web Server

1. Installed Apache web server on the EC2 instance to serve the static HTML page.

2. Created an HTML file at `/var/www/html/index.html` with the following content:

```html

<html>

<head>

<title>Hello World</title>

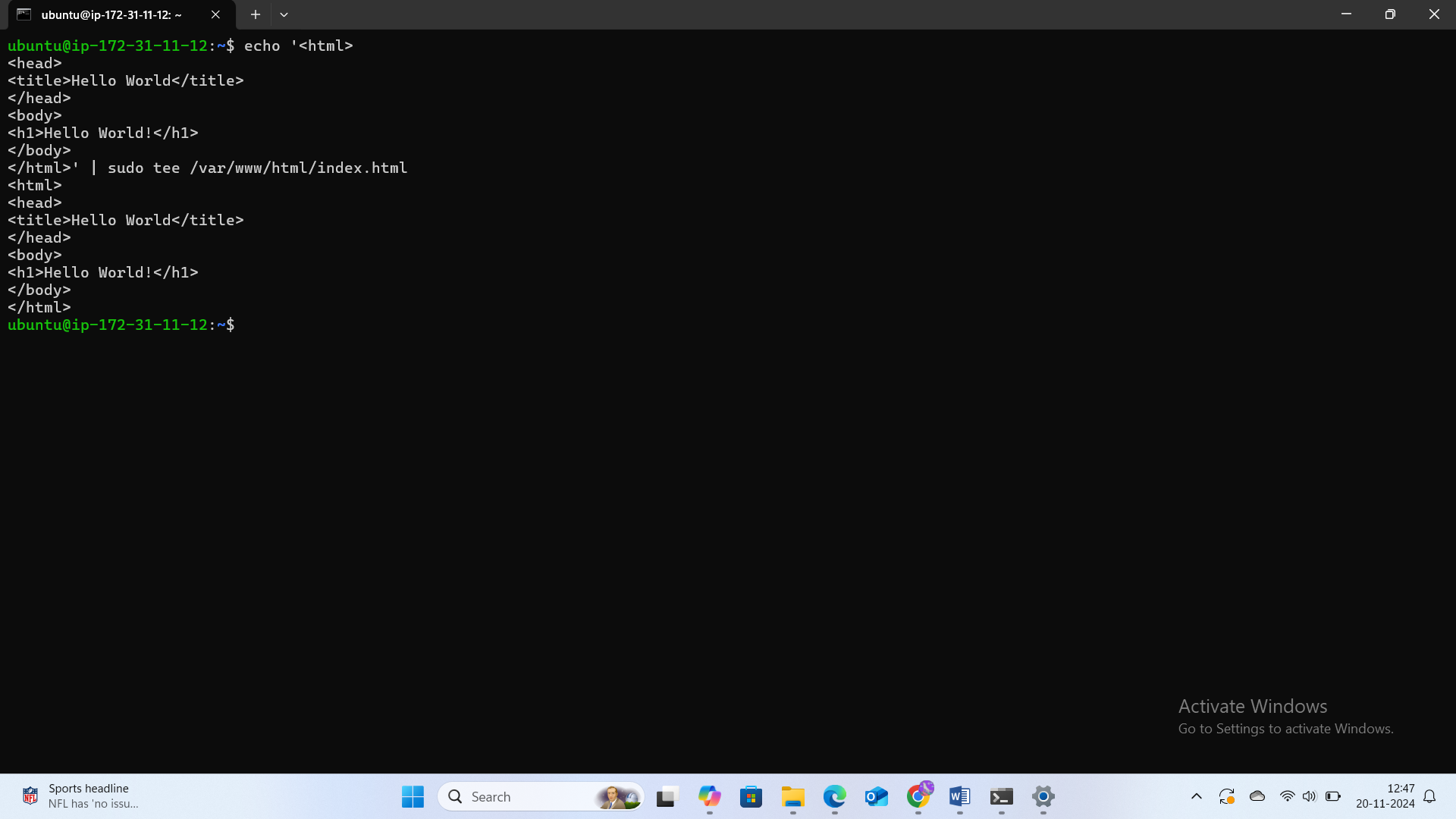
</head>

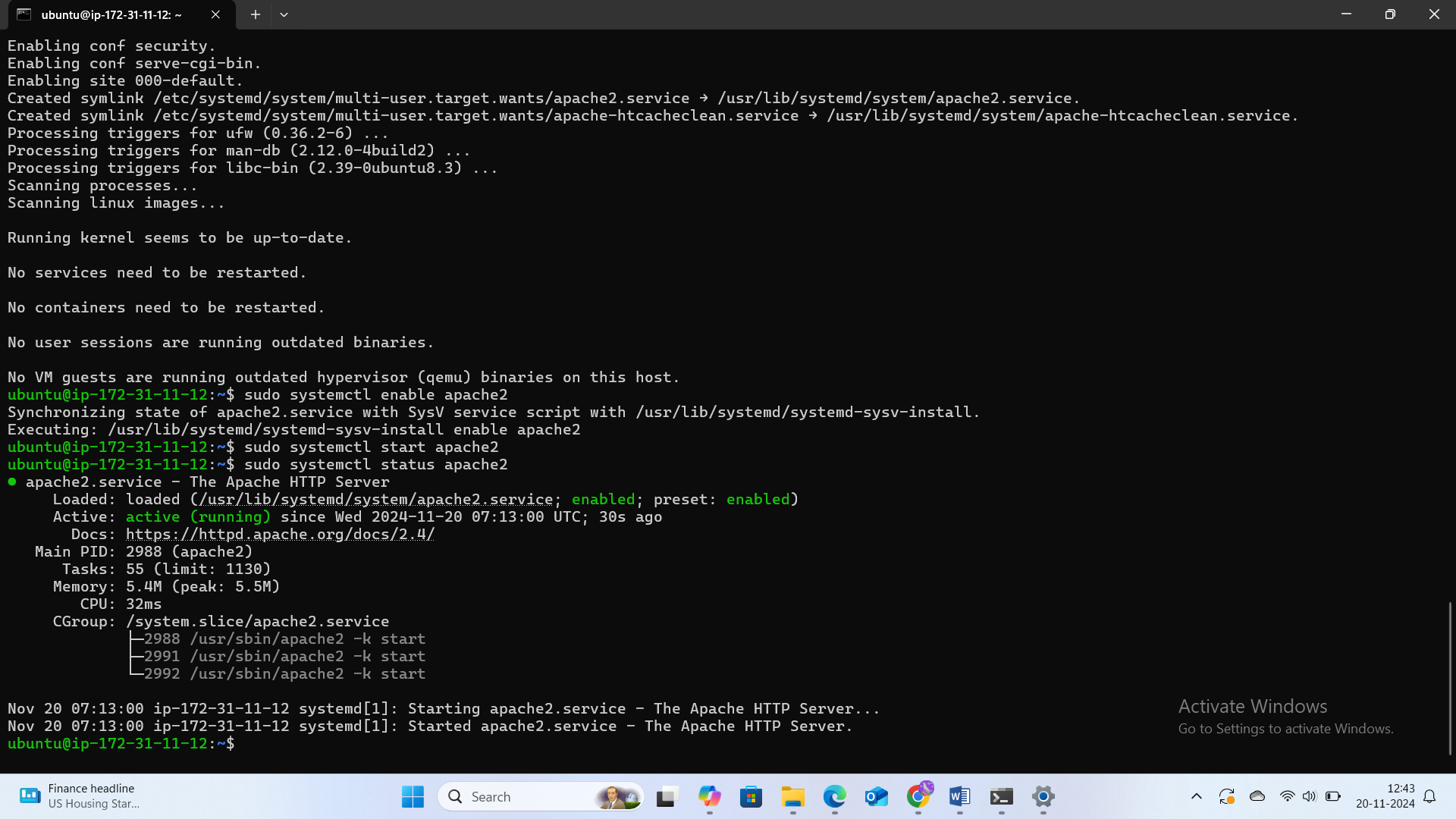
<body>

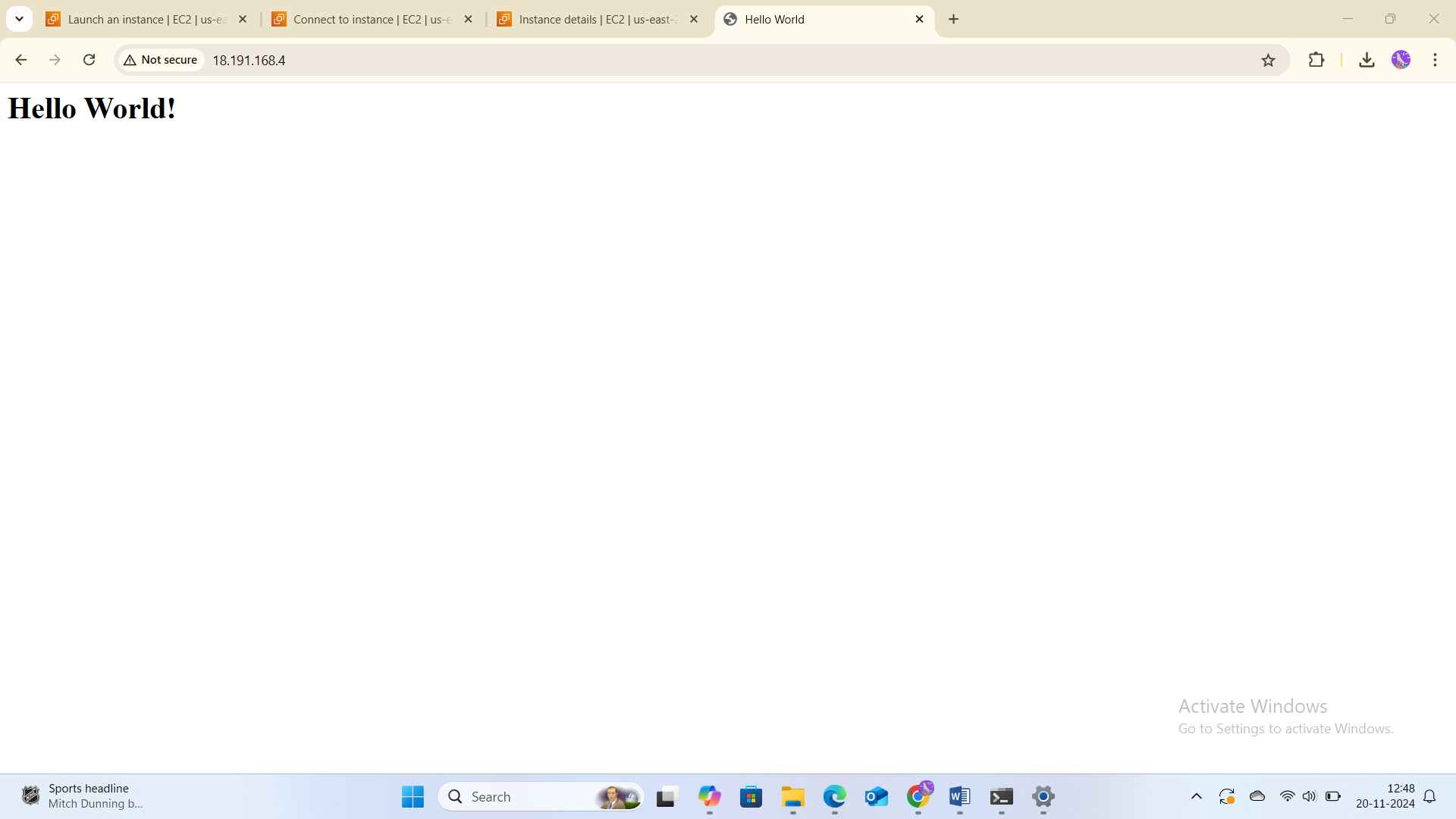
<h1>Hello World!</h1>

</body>

</html>







Secure the Web Server: -

**Step 3: Set Up HTTPS with a Self-Signed SSL Certificate**

1. **Installed OpenSSL** to generate a self-signed SSL certificate.
2. Created a directory to store the SSL certificate and key

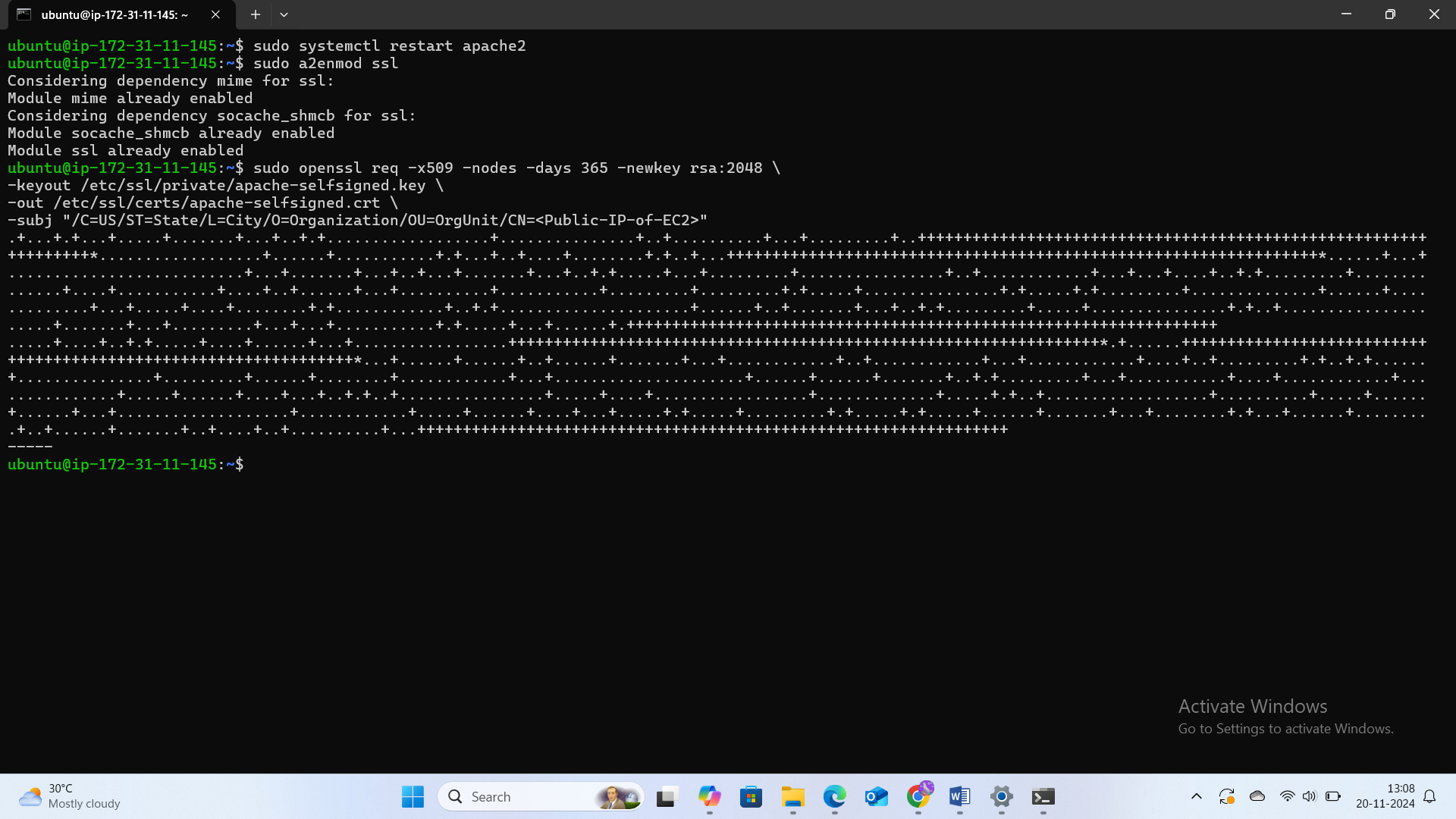
sudo mkdir /etc/ssl/certs/apache

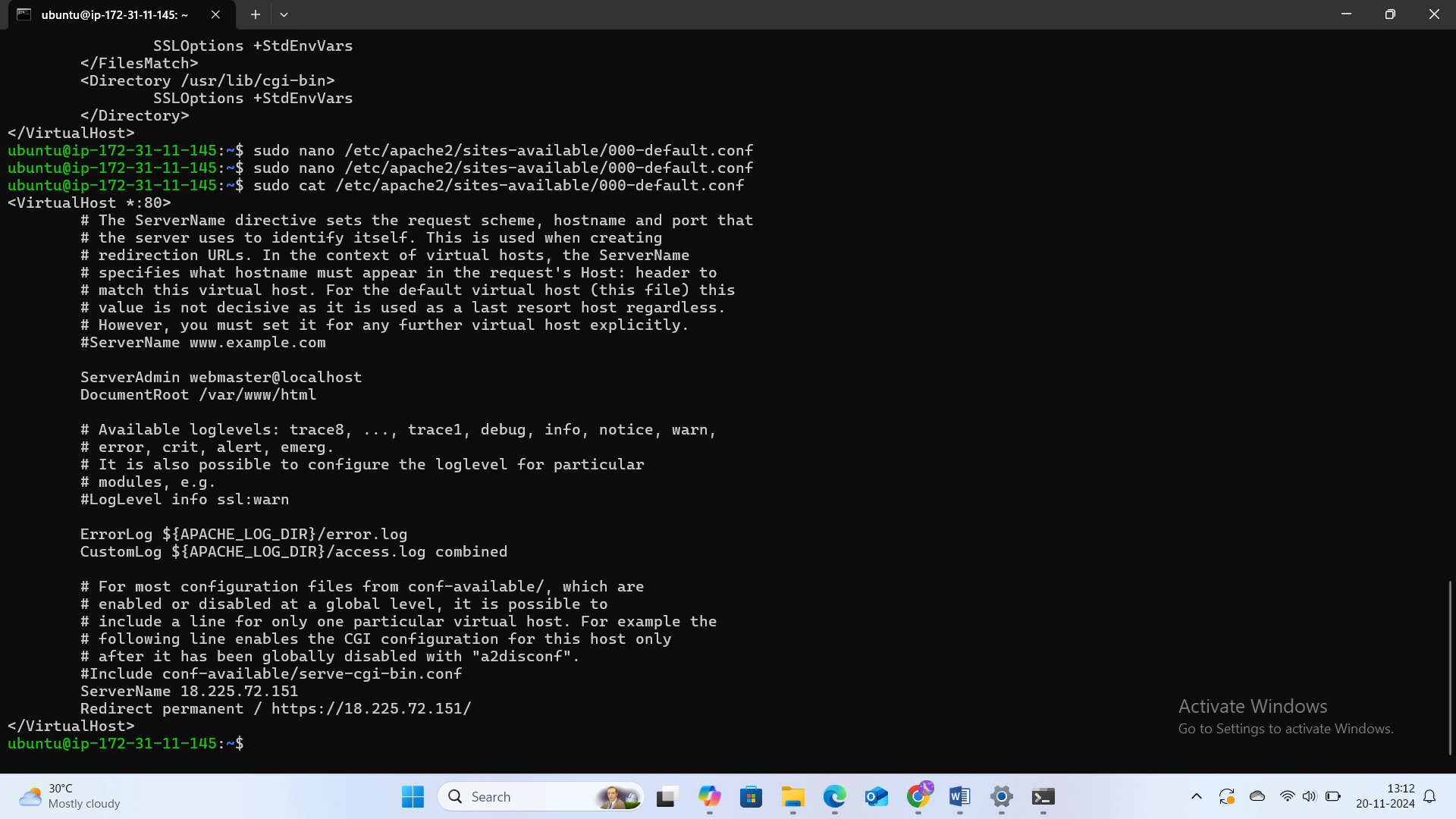
Generated the **self-signed SSL certificate**:

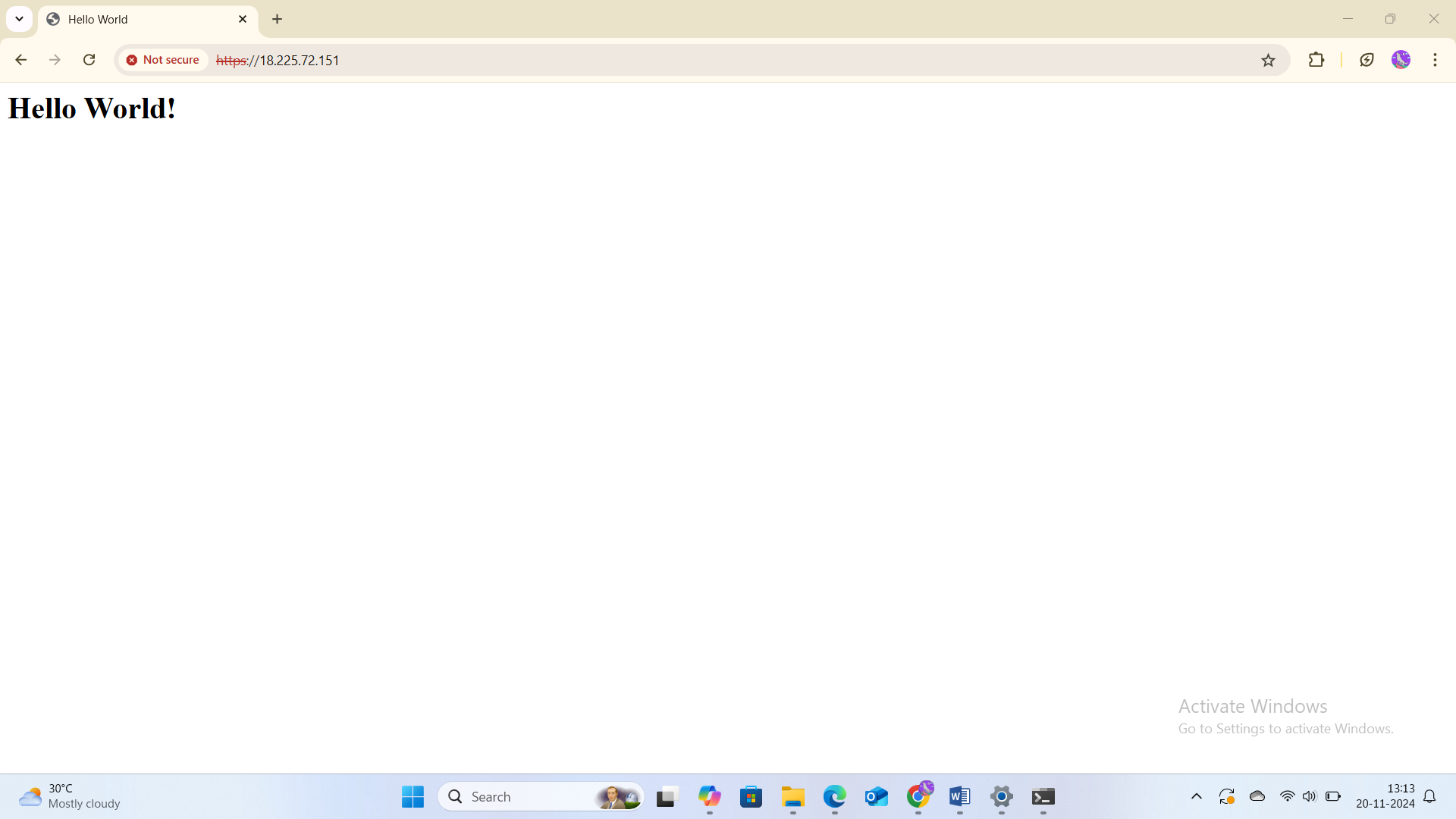
sudo openssl req -x509 -newkey rsa:4096 -keyout /etc/ssl/certs/apache/server.key -out /etc/ssl/certs/apache/server.crt -days 365

 Configured Apache to use the SSL certificate and enabled the SSL module.

 Redirected HTTP traffic (port 80) to HTTPS (port 443) by modifying Apache's configuration file.







### Step 4: ****Configure Elastic Load Balancer (ELB)****

1. **Created an Application Load Balancer (ALB)** to distribute incoming traffic across multiple EC2 instances.
2. Configured a listener to accept both **HTTP** and **HTTPS** requests.
3. Registered the EC2 instance with the **Target Group** of the Load Balancer to ensure it routes traffic to the instance.

### Step 5: ****Set Up Auto Scaling Group****

1. **Created an Auto Scaling Group (ASG)** to ensure high availability and automatic scaling of EC2 instances based on traffic load.
2. Configured the desired capacity (1 instance), minimum capacity (1), and maximum capacity (3).
3. Set up **scaling policies** to automatically scale the EC2 instances based on CPU usage.
4. Attached the Auto Scaling Group to the Elastic Load Balancer for traffic distribution.

### Step 6: ****Monitoring with CloudWatch****

1. **Enabled CloudWatch monitoring** for the EC2 instance to track CPU utilization, memory, and network usage.
2. Set up **CloudWatch Alarms** to monitor CPU utilization and notify me when it exceeds 80%.
3. Created CloudWatch Logs to capture Apache access logs and monitor web traffic.

### Step 7: ****Test and Validate****

1. Accessed the **Load Balancer's DNS** name in a web browser to check if the static web page was loading correctly.
2. Simulated high traffic using the ab (Apache Benchmark) tool to test auto-scaling:

bash

Copy code

ab -n 10000 -c 100 http://<Load-Balancer-DNS>

1. Verified that the Auto Scaling Group scaled up EC2 instances as traffic increased, and the ELB distributed traffic across instances.

### Step 8: ****Clean Up Resources****

1. After completing the project, I **terminated** the EC2 instances, **deleted** the Load Balancer, and **removed** the Auto Scaling Group to avoid extra costs.

## Conclusion

This project successfully demonstrates how to deploy a **secure and scalable static web application** using AWS. I used **EC2**, **Elastic Load Balancer**, and **Auto Scaling** to handle web traffic, along with **CloudWatch** for monitoring. Additionally, I configured **HTTPS** with a self-signed certificate to ensure secure communication. The application is highly available and scalable, with automated scaling based on CPU usage.