

**Placement Empowerment Program**

***Cloud Computing and DevOps Centre***

# Write a Python Flask application and deploy it on your cloud VM. Configure the firewall to allow HTTP traffic.

Name: MADHUMETHRA AP Department: IT



**Introduction**

This Proof of Concept (PoC) demonstrates how to deploy a simple Python Flask web application on a cloud-based Virtual Machine (VM). The goal is to set up a basic web service, make it accessible over the internet, and configure necessary security settings, such as firewall rules.

**Overview**

The PoC involves:

1. Setting up a cloud VM (AWS, GCP, Azure, etc.).

2. Installing dependencies and writing a basic Flask web application.

3. Running the application and exposing it to external users.

4. Configuring firewall rules to allow HTTP traffic.

5. (Optional) Enhancing the deployment with a process manager (e.g., Gunicorn) and a reverse proxy (e.g., Nginx).

**Objective**

1. Cloud Deployment Basics - It helps in understanding how cloud instances work, including networking, security, and resource management.

2. Real-World Application Hosting - Deploying a Flask application on a cloud VM mimics real-world scenarios where web services are hosted on cloud platforms.

3. Scalability & Performance Testing - Once deployed, this PoC can be extended to test load balancing, auto-scaling, and high availability solutions, preparing the system for production-level traffic.

4. Security & Firewall Configuration - By configuring the firewall to allow HTTP traffic while restricting other ports, this PoC highlights the importance of cloud security practices.

5. Preparation for Advanced Deployments - This experiment sets the stage for more advanced deployment strategies, such as containerization with Docker, Kubernetes orchestration, and CI/CD automation.

**Importance**

Here are the key importance points:

1. Accessibility: Allowing HTTP traffic ensures that your web application is accessible to users.

2. Security: Configuring the firewall to allow specific traffic reduces the attack surface.

3. Reliability: Ensuring that your application is accessible and secure improves overall reliability.

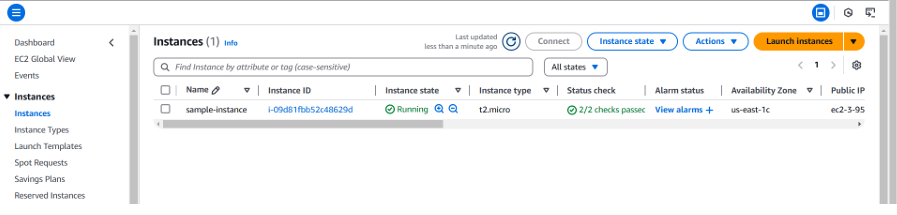
4. Scalability: Properly configuring the firewall and allowing HTTP traffic enables scalability.

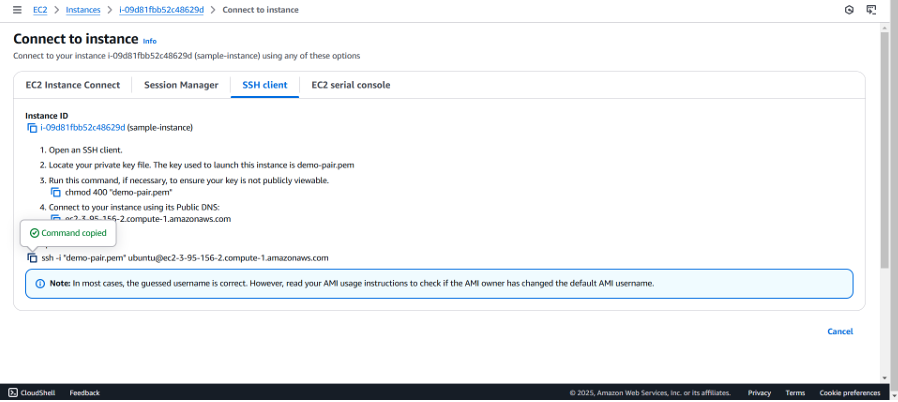
5. Compliance: Depending on your industry, allowing HTTP traffic may be required for compliance.

**Step-by-Step Overview**

Step1:

CREATE AN EC2 INSTANCE • Login into your AWS console and navigate to the EC2 dashboard. • Click on ‘launch instance’ and create your own instance. Ensure your cloud VM is running a Linux distribution (Ubuntu, CentOS, etc.).



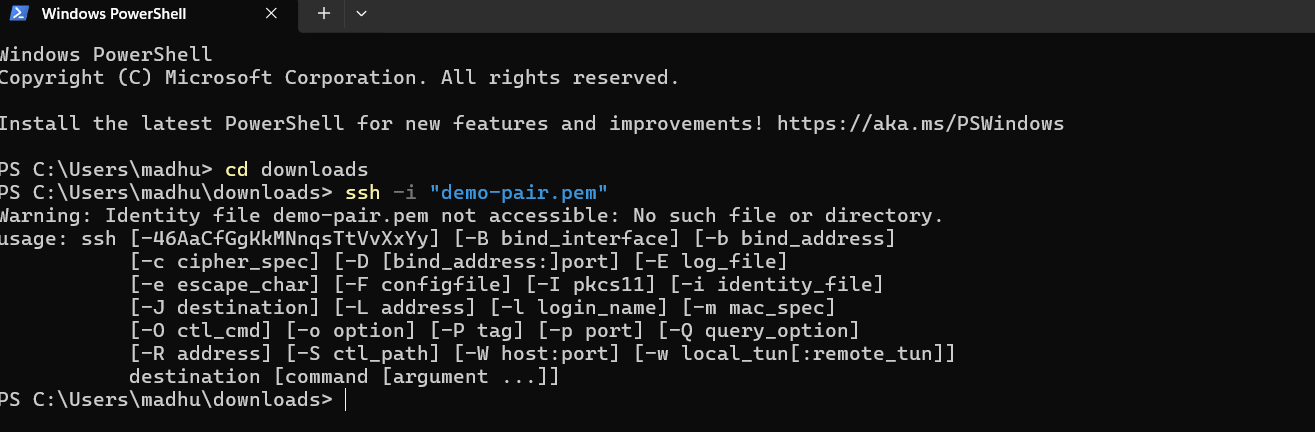


Connect your EC2 instance and copy the ssh command.

Step 2:

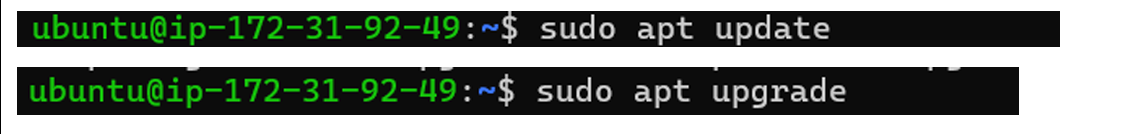
INSTALLING AND SET-UP

Open your PowerShell, change the file directory.



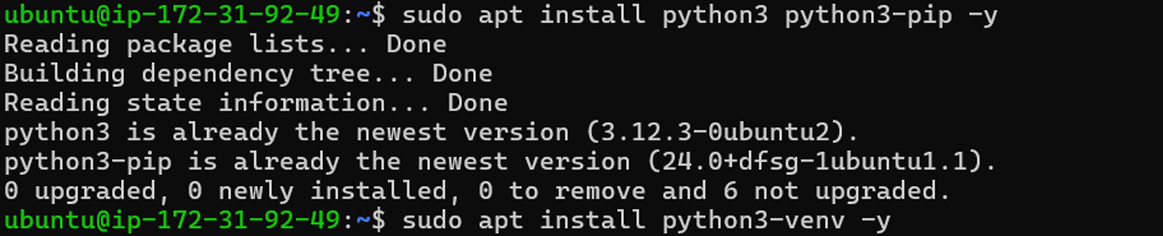
Step 3:

Update & upgrade the package.



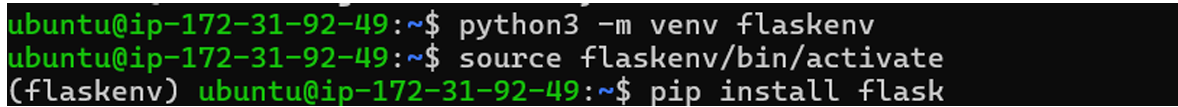
Step 4:

Install python3 and pip.



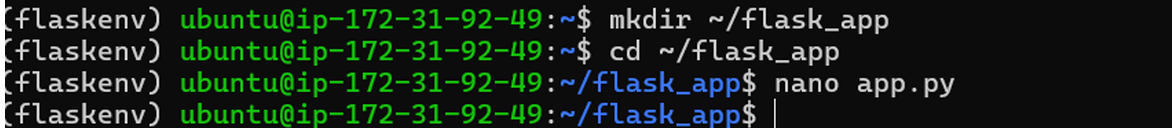
Step 5:

Install Flask.



Step 6:

Create a directory for your app and create a file called ‘app.py’.



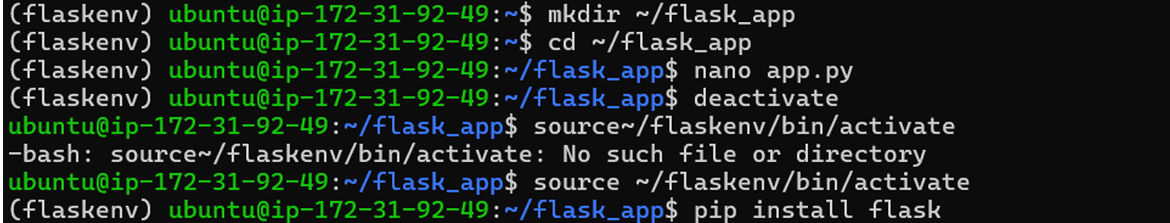
Step 7:

This will open an editor. Write the following code and click enter.



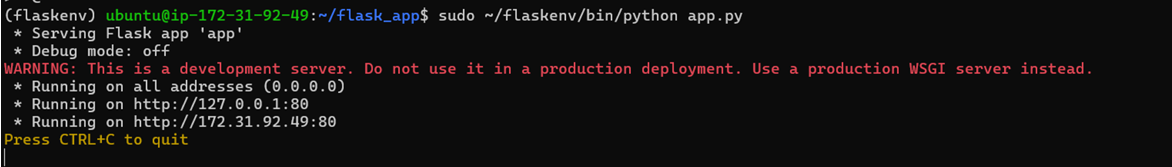
Step 8:

Now, exit the virtual environment. • Then, add your virtual environment’s python path and run the application.



Step 9:

Your Flask is now running!



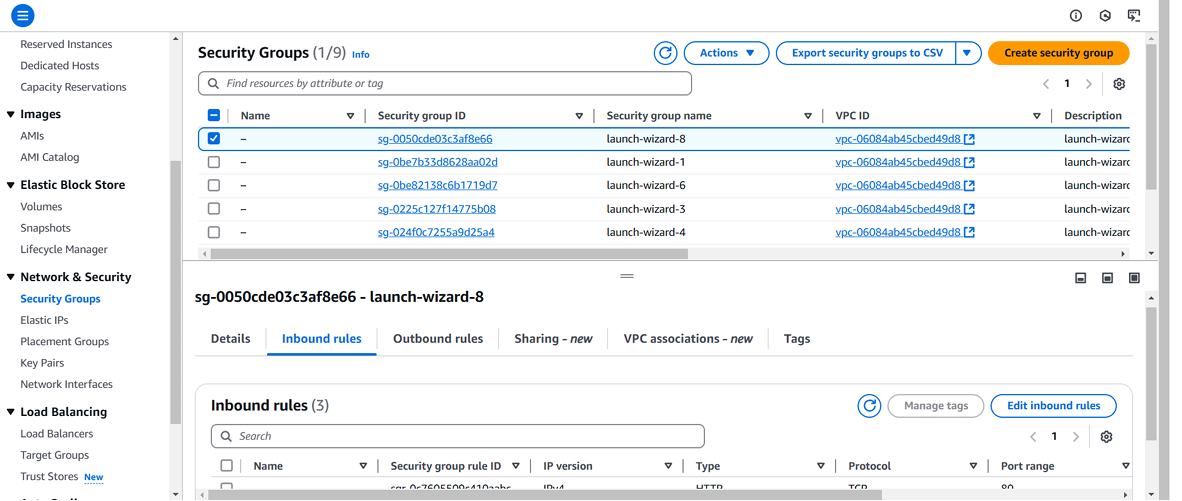
Step 10:

EDIT INBOUND RULES

Go to the EC2 dashboard, > Instances

Find your security group attached to it in ‘network and Security’ section.

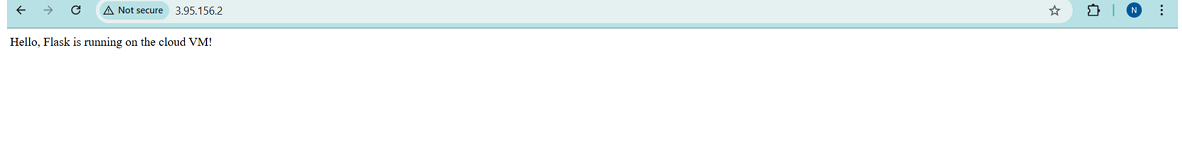
Under the inbound rules, ensure that there is a rule for HTTP(port 80)



Step 11:

TESTING AND ACCESSING

Open your browser to navigate



**Output**

By completing this PoC, you will be able to:

Launch and configure Ec2 instance with Ubuntu as the OS.

Install and configure Python for the flask framework.

Write a simple flask application code.

Host and access the flask web application on the EC2 instance.