Step 1: Launch an AWS EC2 Instance

Sign in to the AWS Management Console.

Navigate to the EC2 Dashboard.

Click on "Launch instance".

Choose an Amazon Machine Image (AMI): Select Ubuntu Server (a recent LTS version like 22.04 or 20.04 is a good choice). This will be our operating system.

Choose an Instance Type: For a simple application, a t2.micro or t3.micro instance (which are part of the AWS Free Tier) is sufficient.

Configure Key Pair: Create a new key pair or use an existing one. This will be a .pem file that you'll use to connect to your instance securely. Download and save this file in a secure location. You will not be able to download it again.

Network Settings (Security Group):

Click "Edit" in the Network settings section.

Create a security group. This acts as a virtual firewall.

Add the following inbound rules:

SSH: Type SSH, Protocol TCP, Port Range 22, Source My IP (for security, this restricts access to your current IP address. You can also select Anywhere, but this is less secure).

HTTP: Type HTTP, Protocol TCP, Port Range 80, Source Anywhere.

HTTPS: Type HTTPS, Protocol TCP, Port Range 443, Source Anywhere.

Configure Storage: The default storage is usually sufficient for a small application.

Launch Instance: Review your settings and click "Launch Instance".

It will take a few minutes for your instance to launch. Once it's running, you can find its Public IPv4 address in the EC2 Dashboard.

Step 2: Connect to Your EC2 Instance

You will use the .pem key file you downloaded to connect to your instance via SSH.

For macOS/Linux:

Open your terminal.

Change the permissions of your .pem file:

chmod 400 /path/to/your-key.pem

Connect to the instance using the following command, replacing your-key.pem with the path to your key and your\_public\_ip with the instance's public IP address:

ssh -i /path/to/your-key.pem ubuntu@your\_public\_ip

For Windows:

You can use an SSH client like PuTTY or the built-in SSH client in PowerShell.

To use PowerShell, follow the same commands as for macOS/Linux.

Once connected, you will be in the terminal of your EC2 instance.

Step 3: Prepare the Server

Now we'll install the software needed to run your Flask application.

Update and Upgrade System Packages:

sudo apt update

sudo apt upgrade -y

Install Python, Pip, and Virtual Environment:

sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-setuptools

sudo apt install python3-venv

Install Nginx:

sudo apt install nginx -y

Step 4: Deploy Your Flask Application

Create a Project Directory:

mkdir ~/myproject

cd ~/myproject

Create a Virtual Environment:

python3 -m venv venv

Activate the Virtual Environment:

source venv/bin/activate

Your terminal prompt should now be prefixed with (venv).

Transfer Your Flask App Files:

You can use git to clone your repository or scp to copy files from your local machine.

Using Git (Recommended):

git clone https://your-repository-url.git .

Using scp (from your local machine's terminal):

scp -i /path/to/your-key.pem /path/to/your/app.py ubuntu@your\_public\_ip:~/myproject/

Install Flask and Other Dependencies:

Make sure you have a requirements.txt file in your project.

pip install -r requirements.txt

pip install gunicorn

Test the App with Gunicorn:

Let's assume your main Flask file is named app.py and the Flask app instance is also named app.

gunicorn --bind 0.0.0.0:5000 app:app

This will start your app. You won't be able to access it from the browser yet because we haven't configured Nginx. Press Ctrl+C to stop the server.

Note: Deactivate the virtual environment for now by typing deactivate.

Step 5: Configure Gunicorn with a Systemd Service

We will create a service file so that systemd can manage Gunicorn for us. This ensures the app runs on startup and restarts if it fails.

Create a Systemd Service File:

sudo nano /etc/systemd/system/myproject.service

Add the following content to the file. Be sure to replace your\_username (which is likely ubuntu) and the paths with your actual project details.

[Unit]

Description=Gunicorn instance to serve myproject

After=network.target

[Service]

User=ubuntu

Group=www-data

WorkingDirectory=/home/ubuntu/myproject

Environment="PATH=/home/ubuntu/myproject/venv/bin"

ExecStart=/home/ubuntu/myproject/venv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi:app

[Install]

WantedBy=multi-user.target

wsgi:app: Assumes you have a wsgi.py file that contains your Flask app instance. If your main file is app.py, you can use app:app. The wsgi.py file is a good practice for production and would simply contain:

# wsgi.py

from app import app

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

app.run()

Start and Enable the Service:

sudo systemctl start myproject

sudo systemctl enable myproject

Check the Status:

sudo systemctl status myproject

You should see an "active (running)" status.

Step 6: Configure Nginx as a Reverse Proxy

Nginx will sit in front of Gunicorn and forward client requests to it.

Create an Nginx Server Block:

sudo nano /etc/nginx/sites-available/myproject

Add the following configuration. Replace your\_public\_ip with your server's IP address or your domain name if you have one.

server {

listen 80;

server\_name your\_public\_ip;

location / {

include proxy\_params;

proxy\_pass http://unix:/home/ubuntu/myproject/myproject.sock;

}

}

Enable the file by creating a symbolic link to sites-enabled:

sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled

Test the Nginx Configuration for syntax errors:

sudo nginx -t

If it's successful, you will see a message saying the syntax is ok.

Restart Nginx:

sudo systemctl restart nginx

You're Live!

Your Flask application should now be accessible from your web browser by navigating to your EC2 instance's public IP address (http://your\_public\_ip).

Next Steps for a Production Environment

Domain Name: Point a custom domain name to your EC2 instance's IP address.

SSL Certificate: Use Let's Encrypt to add a free SSL certificate to secure your application with HTTPS.

Database: For a production application, you'll want to use a more robust database like PostgreSQL or MySQL, possibly using AWS RDS.

Monitoring and Logging: Set up monitoring and logging to track your application's performance and errors.