Deploying a Python Flask Application on AWS EC2 (Amazon Linux)

This guide provides a comprehensive walkthrough for setting up a Python environment, creating a basic Flask web application, and running it on an Amazon EC2 instance.

Step 1: Connect and Install Python

First, connect to your EC2 instance via SSH. Then, install Python and its package manager, `pip`.

1. **Update Your System:** Ensure all your system packages are up-to-date.

   ```
   sudo yum update -y
   ```

2. **Install Python 3:** Amazon Linux 2 comes with Python 3, but this command ensures it's installed.

   ```
   sudo yum install python3 -y
   ```

3. **Install pip:** `pip` is the package installer for Python. You'll use it to install libraries like Flask.

   ```
   sudo yum install python3-pip -y
   ```

4. **Verify the Installation:** Check the versions to confirm that Python and pip are ready to use.

   ```
   # Check Python version (the -V flag shows the version)
   python3 -V
   ```

```
# Check pip version
pip3 -V
```

Step 2: Set Up a Project and Virtual Environment

Using a virtual environment is a crucial best practice in Python development. It isolates your project's dependencies from the system's global packages, preventing conflicts.

1. **Create a Project Directory:**

   ```
   mkdir pythonapp
   cd pythonapp
   ```

2. **Create a Virtual Environment:** We will use the built-in `venv` module to create an environment named `myenv`.

   ```
   python3 -m venv myenv
   ```

3. **Activate the Virtual Environment:** Before you install packages or run your app, you must "activate" the environment. Your terminal prompt will change to show the active environment's name.

   ```
   source myenv/bin/activate
   ```

   (To deactivate later, simply run the command `deactivate`)

Step 3: Create Your Flask Application

With the environment active, you can now create your application files and install the necessary packages.

1. **Create a `requirements.txt` File:** This file lists all the Python packages your project needs.

```
nano requirements.txt
```

Paste the following list of dependencies into the file:

```
click==8.0.3
colorama==0.4.4
Flask==2.0.2
itsdangerous==2.0.1
Jinja2==3.0.3
MarkupSafe==2.0.1
Werkzeug==2.0.2
gunicorn==20.1.0
```

Save the file and exit `nano` (press `CTRL + X`, then `Y`, then `Enter`).

2. **Install the Dependencies:** Use `pip` to install all the packages listed in your requirements file.

```
pip install -r requirements.txt
```

3. **Create the Main Application File:** This script will contain your Flask web server code.

```
nano app.py
```

Paste in the following Python code:

```
from flask import Flask
import os

app = Flask(__name__)

@app.route('/')
def hello_geek():
```

```
    return '<h1>Python App Hosting is Live!</h1>'

@app.route('/hi')
def hello_hi():
    return '<h2>Hiiii from your Flask App!</h2>'

if __name__ == "__main__":
    # Use port 5000 by default, or a port defined in the environment
    port = int(os.environ.get('PORT', 5000))
    # The host '0.0.0.0' makes the server publicly available
    app.run(debug=True, host='0.0.0.0', port=port)
```

Save and exit the file.

Step 4: Configure AWS Security Group

To access your application from the internet, you need to open port 5000 in your EC2 instance's security group.

1. Go to the **EC2 Dashboard** in your AWS Console.
2. Select your instance and click on the **Security** tab.
3. Click on the security group name.
4. Click **Edit inbound rules**, then **Add rule**.
5. Set the following values:
   - **Type:** `Custom TCP`
   - **Port Range:** `5000`
   - **Source:** `Anywhere` (`0.0.0.0/0`).
6. **Save rules**.

Step 5: Run the Flask Application

Here are two ways to run your app.

Method A: Using the Flask Development Server (for Testing)

This is a simple way to test your app, but it's not suitable for production.

1. **Run the App:**

```
python3 app.py
```

2. **Access in Browser:** Open your browser and go to: `http://<Your-EC2-Public-IP-Address>:5000`

   You should see your welcome message. To stop the server, press `CTRL + C` in the terminal.

Method B: Using Gunicorn (Recommended for Production)

Gunicorn is a robust WSGI HTTP server that can handle multiple worker processes, making it much more suitable for running a real application.

1. **Run with Gunicorn:** This command tells Gunicorn to run the `app` object from your `app.py` file. The `b` flag binds it to all available network interfaces on port 5000.

   ```
   gunicorn --bind 0.0.0.0:5000 app:app
   ```

2. **To run it in the background**, you can add the `-daemon` flag:

   ```
   gunicorn --bind 0.0.0.0:5000 app:app --daemon
   ```

3. **To stop the background Gunicorn process**, you'll need to find its process ID (PID) and kill it:

   ```
   # Find the Gunicorn process
   ps aux | grep gunicorn

   # Kill the process using its PID
   kill <PID>
   ```