Deploying a Flask App to a Remote Ubuntu (20.04.6 LTS x86_64) Server

This documentation will guide you to deploy your pre-existing flask app on a remote Ubuntu 20.04 Server.

Step 1: Connect to the remote server via SSH

Open Windows Terminal and enter the following command

```
$ ssh <username>@<IP_address>
```

Replace with your Ubuntu server username and <IP_address> with the server's IP Address.
For Example:

```
$ ssh ubuntu@172.15.3.94
```

Step 2: Update the Ubuntu System

Enter the following command:

```
$ sudo apt update && sudo apt upgrade
```

Step 3: Install the required software

Install Python, pip, Nginx, and set-up a virtual environment

```
$ sudo apt install python3 python3-pip python3-venv nginx
```

Note: The following versions have been used and confirmed to work

Ubuntu: 20.04.6 LTS
Kernel: 5.4.0-156
Python: 3.8.10
Flask: 2.3.2

```
gunicorn: 21.2.0
pip: 20.0.2
```

The versions of applications used are as follows

Step 4: Set Up your Flask App on the Server

- Since this is a small API, we could set-up the app by copying the code from our python file and pasting it on the server.
- The directory structure of the app is:

```
├── python-apis
|   ├── app.py
```

- To replicate the structure, make a folder called `python-apis`:

```
$ mkdir python-apis
```

- Make the required python files inside the directory:

```
$ cd python-apis
$ touch app.py
```

- Copy the contents of app.py from your machine to the remote server.

```
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/add', methods=['POST'])
def add_nums():
    try:
        data =  request.get_json()
        num1 = data['num1']
        num2 = data['num2']
        result = num1 + num2
        return jsonify({'result': result})
    except KeyError:
        return jsonify({"error": "Invalid input."})

@app.route('/subtract', methods=['POST'])
def subtract_numbers():
    data = request.get_json()
    num1 = data['num1']
    num2 = data['num2']
    result = num1 - num2
    return jsonify({'result': result})

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True, port=8989)
```

Step 5: Create and Activate the Python Virtual Environment

```
$ python3 -m venv venv
$ source venv/bin/activate
```

Your python virtual environment should now be active, indicated by `(venv)` before your prompt on the server.
Example:

```
(venv) ubuntutest@ubuntu:~/python-apis$
```

Step 6: Install the required dependencies

```
$ pip install Flask gunicorn
```

Step 7: Test Gunicorn

Test that your Flask app works using gunicorn:

```
$ ./venv/bin/gunicorn --bind 0.0.0.0:8989 -w 3 app:app
```

Here, the first `app` corresponds to the filename `app.py` and the second `app` corresponds to the `app` module inside `app.py`. Hence, make sure to change the names accordingly.

⚠ **Warning**

Check the port number in the command. Use the same port as you have specified in the Flask App. In this case, the port number is 8989.

Step 8: Configure Gunicorn systemd service

This step runs the gunicorn command in the background whenever the system is started. So the API is always served and accessible.
You may use your editor of choice instead of nano.

```
$ sudo nano /etc/systemd/system/python-apis.service
```

Add the following content:

```
[Unit]
Description=Gunicorn instance to serve your Flask app
After=network.target

[Service]
User=<username>
Group=www-data
WorkingDirectory=/home/<username>/python-apis
ExecStart=/home/<username>/python-apis/./venv/bin/gunicorn --bind 0.0.0.0:8989
-w 3 app:app

[Install]
WantedBy=multi-user.target
```

⚠ **Attention**

Replace the username with your ubuntu server username. Make sure to use the gunicorn from the virtual environment and not from your ubuntu installation (if exists). The working directory is the root directory of your Flask App.

Save and close the file.

Step 9: Configure Nginx

Create an Nginx server block configuration file:

```
$ sudo nano /etc/nginx/sites-available/python-apis
```

Add the following content:

```
server {
    listen 80;
    server_name your_server_domain_or_ip;

    location / {
        include proxy_params;
        proxy_pass http://your_server_ip:8000;
    }
}
```

Replace `your_server_domain_or_ip` and `your_server_ip` and port number accordingly.

Create a symbolic link to enable the configuration:

```
$ sudo ln -s /etc/nginx/sites-available/python-apis /etc/nginx/sites-enabled
```

Test the Nginx configuration:

```
$ sudo nginx -t
```

Restart Nginx to apply the changes:

```
$ sudo systemctl restart nginx
```

Step 10: Start Gunicorn Service

Start and enable the Gunicorn systemd service:

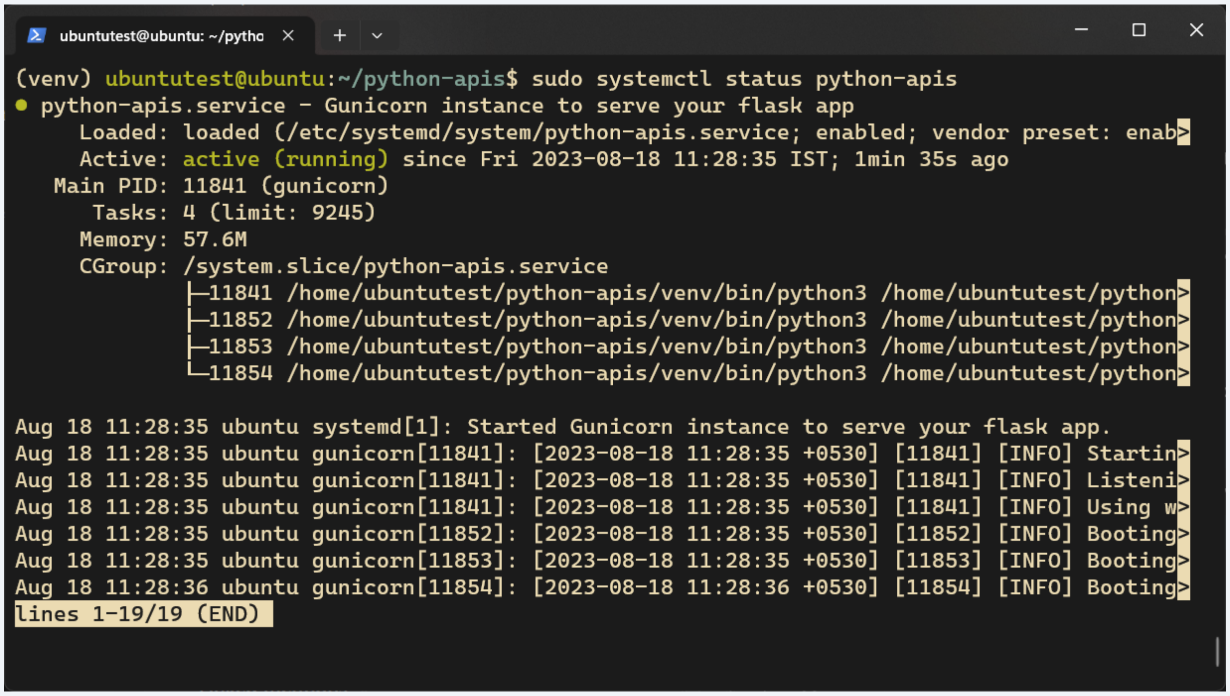
```
$ sudo systemctl start python-apis
$ sudo systemctl enable python-apis
```

Note

Make sure the service is up and running by using the following command:

```
$ sudo systemctl status python-apis
```

The output will look something like this:



```
(venv) ubuntutest@ubuntu:~/python-apis$ sudo systemctl status python-apis
● python-apis.service - Gunicorn instance to serve your flask app
     Loaded: loaded (/etc/systemd/system/python-apis.service; enabled; vendor preset: enab>
     Active: active (running) since Fri 2023-08-18 11:28:35 IST; 1min 35s ago
   Main PID: 11841 (gunicorn)
      Tasks: 4 (limit: 9245)
     Memory: 57.6M
     CGroup: /system.slice/python-apis.service
             ├─11841 /home/ubuntutest/python-apis/venv/bin/python3 /home/ubuntutest/python>
             ├─11852 /home/ubuntutest/python-apis/venv/bin/python3 /home/ubuntutest/python>
             ├─11853 /home/ubuntutest/python-apis/venv/bin/python3 /home/ubuntutest/python>
             └─11854 /home/ubuntutest/python-apis/venv/bin/python3 /home/ubuntutest/python>

Aug 18 11:28:35 ubuntu systemd[1]: Started Gunicorn instance to serve your flask app.
Aug 18 11:28:35 ubuntu gunicorn[11841]: [2023-08-18 11:28:35 +0530] [11841] [INFO] Startin>
Aug 18 11:28:35 ubuntu gunicorn[11841]: [2023-08-18 11:28:35 +0530] [11841] [INFO] Listeni>
Aug 18 11:28:35 ubuntu gunicorn[11841]: [2023-08-18 11:28:35 +0530] [11841] [INFO] Using w>
Aug 18 11:28:35 ubuntu gunicorn[11852]: [2023-08-18 11:28:35 +0530] [11852] [INFO] Booting>
Aug 18 11:28:35 ubuntu gunicorn[11853]: [2023-08-18 11:28:35 +0530] [11853] [INFO] Booting>
Aug 18 11:28:36 ubuntu gunicorn[11854]: [2023-08-18 11:28:36 +0530] [11854] [INFO] Booting>
lines 1-19/19 (END)
```

If the service is not active, return to previous steps and make sure that all the steps have been followed properly.

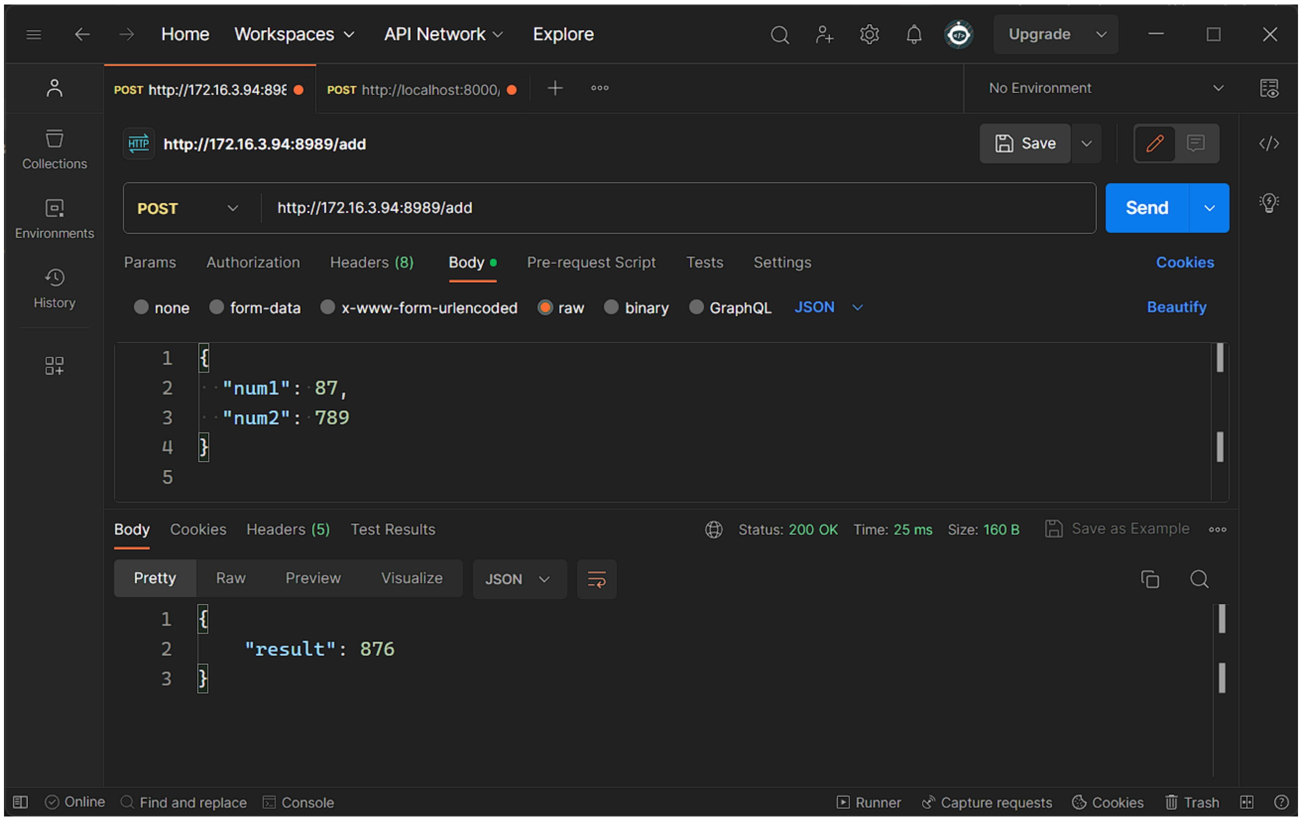
Step 11: Adjust the firewall Settings

Allow Nginx on UFW:

```
$ sudo ufw allow 'Nginx Full'
```

Step 12: Access Your Flask App via Postman

Send an appropriate request via Postman to your API and make sure it returns the appropriate response to confirm that it works.
Example:



POST http://172.16.3.94:8989/add

```
{
  "num1": 87,
  "num2": 789
}
```

Status: 200 OK Time: 25 ms Size: 160 B

```
{
    "result": 876
}
```

To send the query as JSON, Click on `Body -> raw -> Text` and change it to JSON. Send your query.
For Example, for our API:

```
{
  "num1": 87,
  "num2": 789
}
```

If the response is not received, try disabling the firewall (consider safety requirements before doing so) and then try sending the query via Postman again.

```
$ sudo systemctl disable ufw.service
```

If this works, enable the firewall service and then enable the appropriate ports and try again.

Make sure an appropriate response is received as shown in screenshot.

Conclusion

Congratulations! You've successfully deployed your Flask app on a remote Ubuntu server using Nginx as a reverse proxy and Gunicorn as the application server.