ISE – Cloud Computing Lab(AI423)

Name : Juber Husen Shaikh

PRN : 2217027

Batch : S1

Title : Deploy a Python-Based Weather Information App on a Virtual Machine in the Cloud

Aim :

- To develop a Python-based application that retrieves and displays current weather information for a given city using a free weather API.
- To deploy and run the developed application on a Virtual Machine (VM) or cloud environment so it can be accessed via a web browser.

Steps of Execution :

1. **Set Up Environment:**

- Install VirtualBox, VMware, or create a cloud VM instance.
- Install Python and required packages (flask, flask-bootstrap, requests).

2. **Create Project Folder:**

```
weather_app/
  ├── app.py
   ├── requirements.txt
   ├── templates/
     |   └── index.html
  └── static/
```

3. **Develop Backend (Flask Application):**

- Create app.py to handle routing, API calls, and data rendering.
- Use requests to call the Open-Meteo API and fetch weather data based on the entered city.

4. **Design Frontend (HTML Template):**

- Create index.html inside the templates/ folder using Bootstrap for styling.
- Display temperature and weather description dynamically.

5. **Create requirements.txt:**

- List dependencies required to run the project.
- Install them using pip install -r requirements.txt.

6. **Run the Application:**

- Start Flask server with python3 app.py.
- Access the app in a browser using:
  http://<VM_IP>:8000

7. **Testing:**

- Enter a city name and verify weather data is fetched and displayed correctly.
- Ensure the app runs properly inside the VM environment.

Folder Structure:

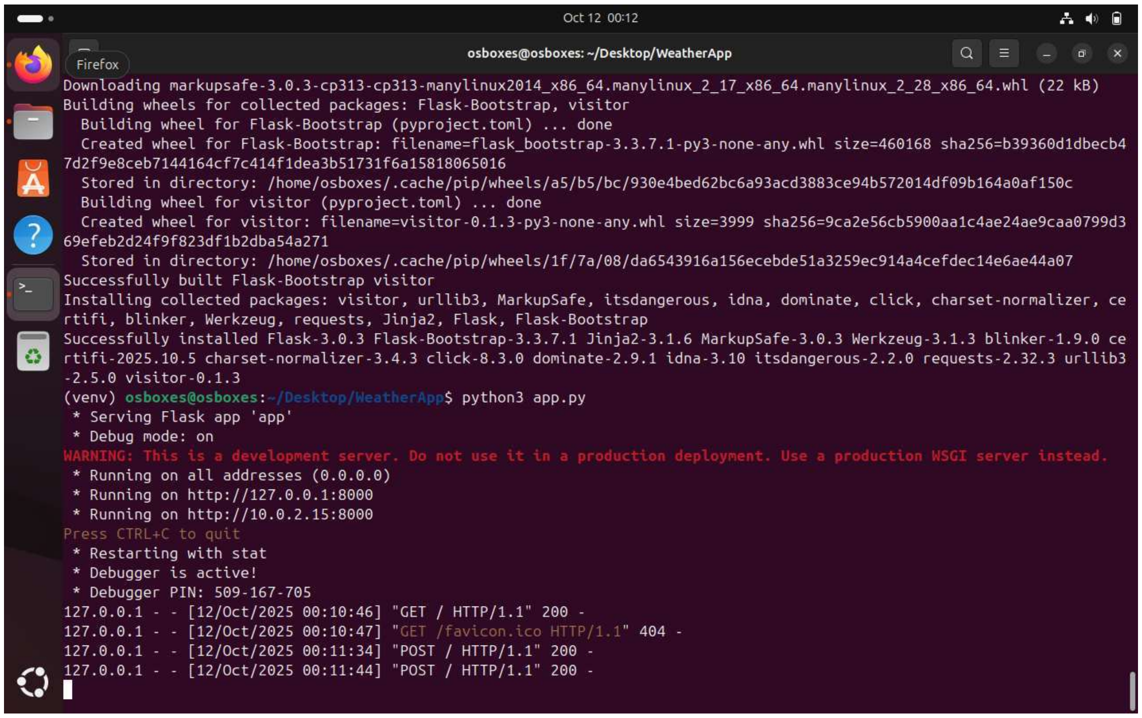
```
weather_app/

|

 ├── app.py

 ├── requirements.txt

 ├── templates/

|   └── index.html

└── static/
```

App.py: Code

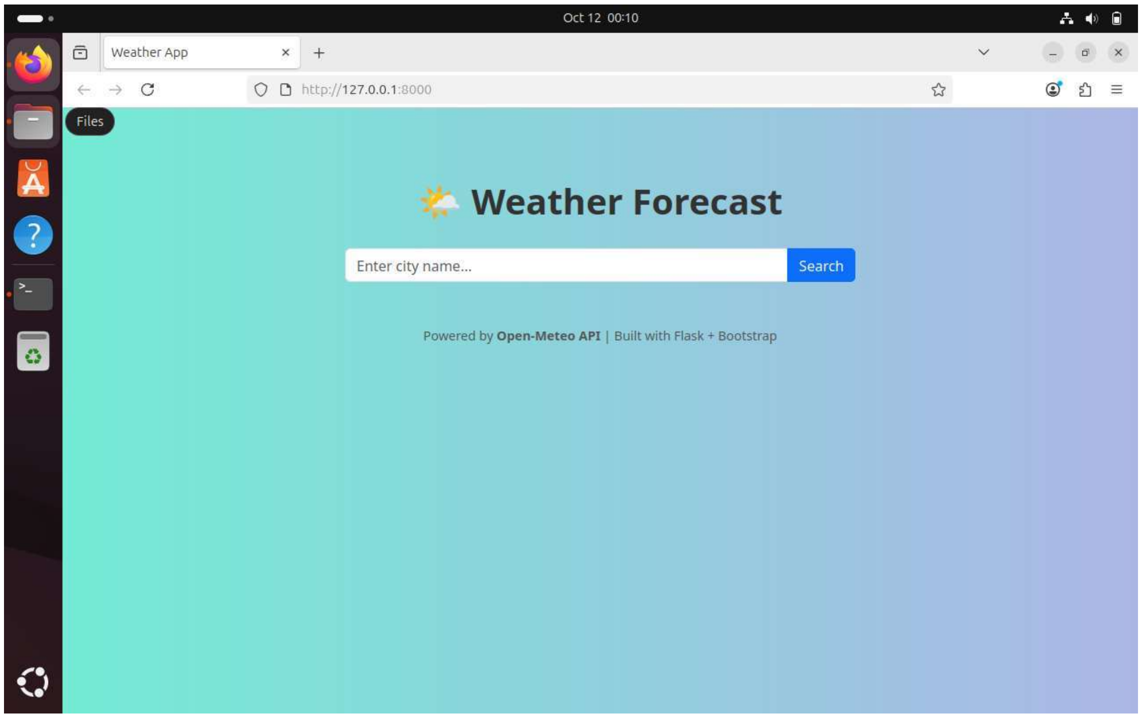
```
import requests

from flask import Flask, render_template, request

from flask_bootstrap import Bootstrap


app = Flask(__name__)

Bootstrap(app)
```

```
GEOCODING_BASE_URL = "http://geocoding-api.open-meteo.com/v1/search"
WEATHER_BASE_URL = "http://api.open-meteo.com/v1/forecast"


def interpret_weather_code(code):
    """Convert Open-Meteo weather codes into human-readable descriptions."""
    codes = {
        0: "Clear sky",
        1: "Mainly clear",
        2: "Partly cloudy",
        3: "Overcast",
        45: "Fog",
        48: "Depositing rime fog",
        61: "Slight rain",
        63: "Moderate rain",
        65: "Heavy rain",
        80: "Slight rain showers",
        95: "Thunderstorm",
    }
    return codes.get(code, "Unknown condition")


def get_color_class(description):
    """Return a Bootstrap color class based on weather condition."""
    desc = description.lower()

    if "clear" in desc:
        return "clear-card"
    elif "rain" in desc or "shower" in desc:
        return "rain-card"
    elif "cloud" in desc or "overcast" in desc:
        return "cloud-card"
    elif "thunder" in desc:
        return "storm-card"
    elif "fog" in desc:
        return "fog-card"
    else:
        return "default-card"


@app.route("/", methods=["GET", "POST"])
def index():
    weather_data = None
    error = None

    if request.method == "POST":
        city = request.form["city"]

        if city:
            try:
                # Fetch latitude and longitude for the given city
                geo_res = requests.get(
                    GEOCODING_BASE_URL,
                    params={"name": city, "count": 1},
                    timeout=10
                )
                geo_res.raise_for_status()
                geo_data = geo_res.json()

                if not geo_data.get("results"):
                    raise ValueError(f"City '{city}' not found.")

                location = geo_data["results"][0]
                lat = location["latitude"]
                lon = location["longitude"]
        city_name = location["name"]

        # Fetch current weather data
        weather_params = {
            "latitude": lat,
            "longitude": lon,
            "current_weather": "true"
        }
        weather_res = requests.get(
            WEATHER_BASE_URL,
            params=weather_params,
            timeout=10
        )
        weather_res.raise_for_status()
        data = weather_res.json()

        current = data["current_weather"]
        description = interpret_weather_code(current["weathercode"])
        color_class = get_color_class(description)

        weather_data = {
            "city": city_name,
            "temperature": current["temperature"],
            "description": description,
            "color_class": color_class
        }

    except Exception as e:
        error = str(e)
else:
    error = "City name cannot be empty."
    return render_template("index.html", weather=weather_data, error=error)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, debug=True)
```

Output: (screenshots)



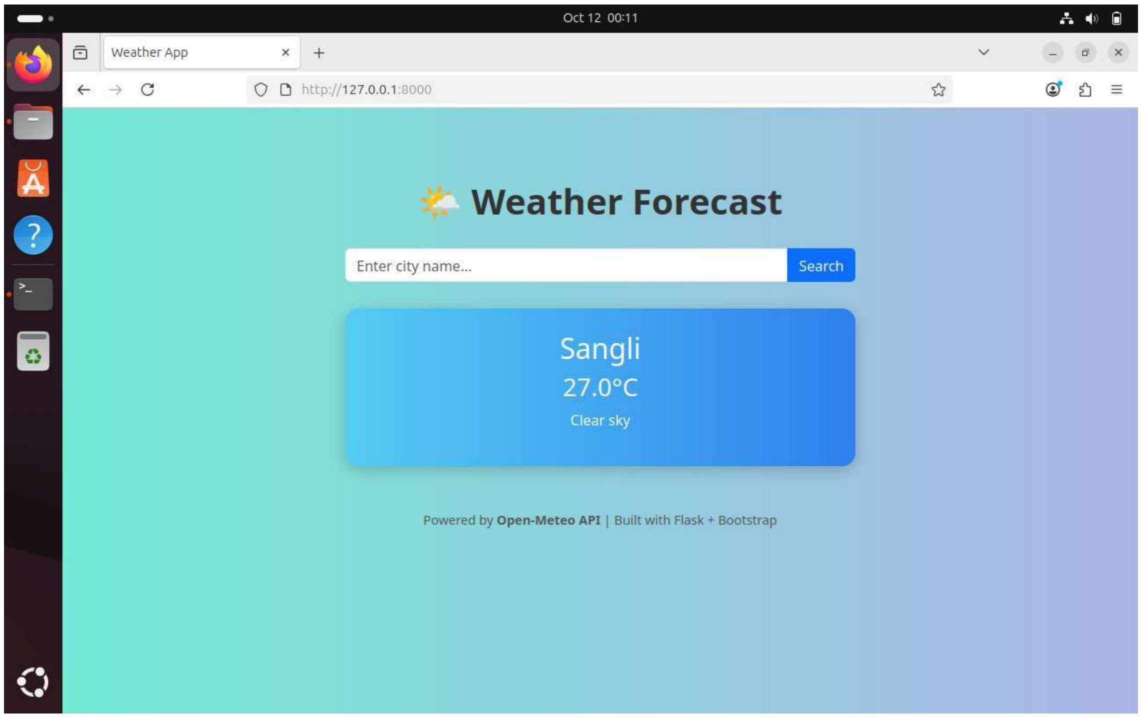
```
Downloading markupsafe-3.0.3-cp313-cp313-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (22 kB)
Building wheels for collected packages: Flask-Bootstrap, visitor
  Building wheel for Flask-Bootstrap (pyproject.toml) ... done
  Created wheel for Flask-Bootstrap: filename=flask_bootstrap-3.3.7.1-py3-none-any.whl size=460168 sha256=b39360d1dbecb4
7d2f9e8ceb7144164cf7c414f1dea3b51731f6a15818065016
  Stored in directory: /home/osboxes/.cache/pip/wheels/a5/b5/bc/930e4bed62bc6a93acd3883ce94b572014df09b164a0af150c
  Building wheel for visitor (pyproject.toml) ... done
  Created wheel for visitor: filename=visitor-0.1.3-py3-none-any.whl size=3999 sha256=9ca2e56cb5900aa1c4ae24ae9caa0799d3
69efeb2d24f9f823df1b2dba54a271
  Stored in directory: /home/osboxes/.cache/pip/wheels/1f/7a/08/da6543916a156ecebde51a3259ec914a4cefdec14e6ae44a07
Successfully built Flask-Bootstrap visitor
Installing collected packages: visitor, urllib3, MarkupSafe, itsdangerous, idna, dominate, click, charset-normalizer, ce
rtifi, blinker, Werkzeug, requests, Jinja2, Flask, Flask-Bootstrap
Successfully installed Flask-3.0.3 Flask-Bootstrap-3.3.7.1 Jinja2-3.1.6 MarkupSafe-3.0.3 Werkzeug-3.1.3 blinker-1.9.0 ce
rtifi-2025.10.5 charset-normalizer-3.4.3 click-8.3.0 dominate-2.9.1 idna-3.10 itsdangerous-2.2.0 requests-2.32.3 urllib3
-2.5.0 visitor-0.1.3
(venv) osboxes@osboxes:~/Desktop/WeatherApp$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8000
 * Running on http://10.0.2.15:8000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 509-167-705
127.0.0.1 - - [12/Oct/2025 00:10:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Oct/2025 00:10:47] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [12/Oct/2025 00:11:34] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [12/Oct/2025 00:11:44] "POST / HTTP/1.1" 200 -
```



Weather Forecast

Enter city name... Search

Powered by **Open-Meteo API** | Built with Flask + Bootstrap



Weather Forecast

Enter city name... Search

Sangli
27.0°C
Clear sky

Powered by **Open-Meteo API** | Built with Flask + Bootstrap