Dockerizing the Flask Web Application

Aim:The aim of this project is to containerize a simple Flask web application using Docker. By leveraging Docker, we aim to ensure consistency across different environments, simplify deployment, and eliminate dependency conflicts. The primary goal is to build a lightweight, portable, and reproducible web application that can run seamlessly in any environment.

Technologies Used

- Programming Language: Python
- Framework: Flask
- Containerization Platform: Docker
- Operating System: Windows/Linux/Mac (depending on user system)

Detailed Procedure

Step 1: Setting Up the Project Environment

1. Create Project Directory:
    - Open a terminal or command prompt.
    - Run the following commands:

```
mkdir web-app-docker
```

```
cd web-app-docker
```

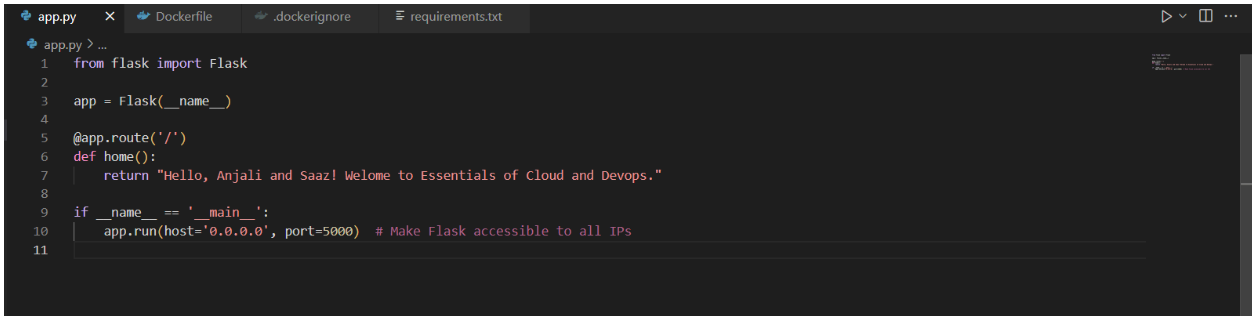
2.Create a Virtual Environment:

- Helps isolate dependencies for the project.

```
python -m venv venv
.\venv\Scripts\activate  # Windows
source venv/bin/activate  # Linux/Mac
```

Step 2: Writing the Application Code

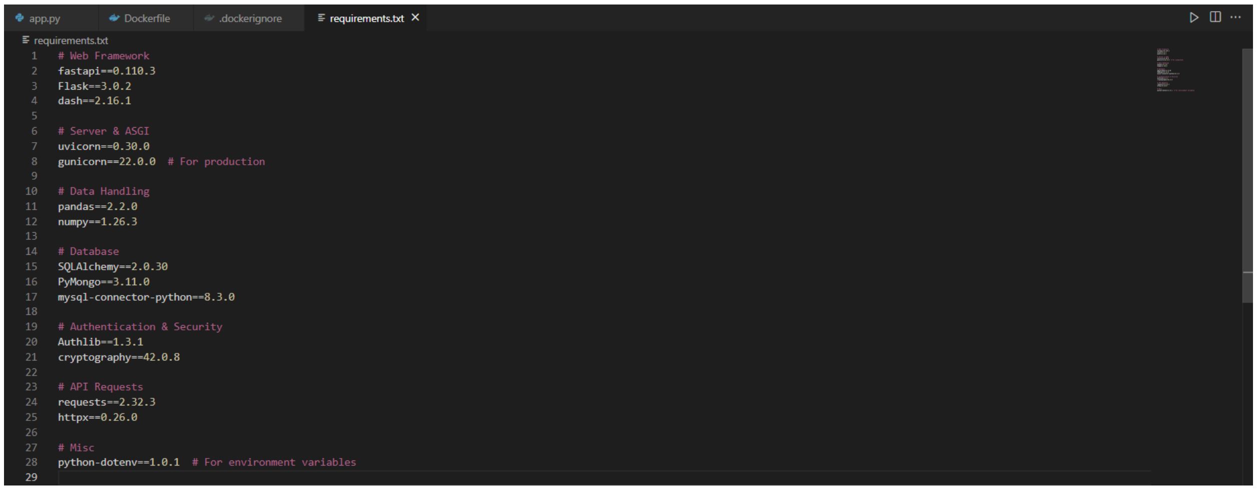
1. Create app.py:
    - Add the following code:



```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Anjali and Saaz! Welome to Essentials of Cloud and Devops."

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)  # Make Flask accessible to all IPs
```

2.Create requirements.txt:

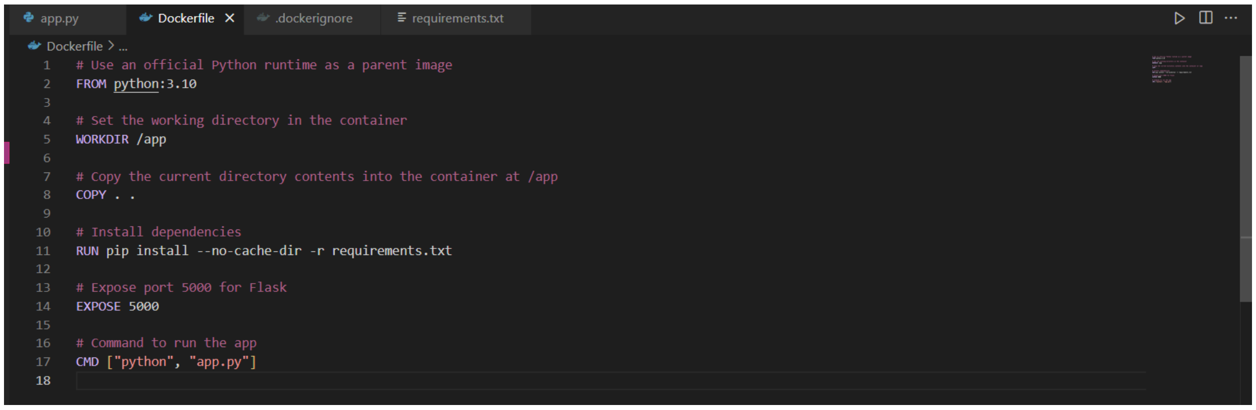
- List the required dependencies:



```
# Web Framework
fastapi==0.110.3
Flask==3.0.2
dash==2.16.1

# Server & ASGI
uvicorn==0.30.0
gunicorn==22.0.0  # For production

# Data Handling
pandas==2.2.0
numpy==1.26.3

# Database
SQLAlchemy==2.0.30
PyMongo==3.11.0
mysql-connector-python==8.3.0

# Authentication & Security
Authlib==1.3.1
cryptography==42.0.8

# API Requests
requests==2.32.3
httpx==0.26.0

# Misc
python-dotenv==1.0.1  # For environment variables
```

Step 3: Dockerizing the Application

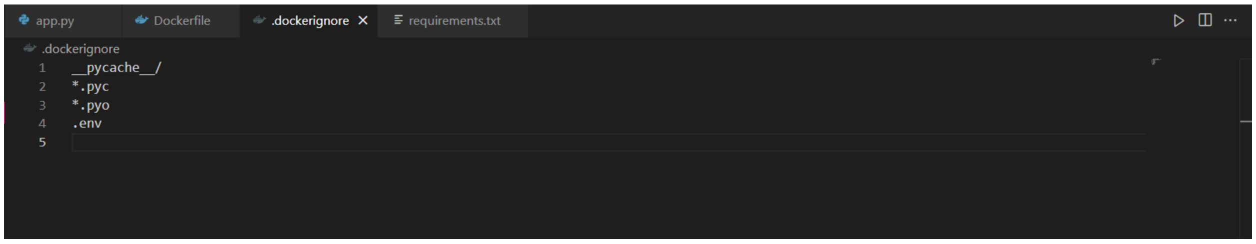
1. Create a Dockerfile:



```dockerfile
# Use an official Python runtime as a parent image
FROM python:3.10

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose port 5000 for Flask
EXPOSE 5000

# Command to run the app
CMD ["python", "app.py"]
```

2.Create a .dockerignore File:

- Exclude unnecessary files from the image.



```
__pycache__/
*.pyc
*.pyo
.env
```

Step 4: Building and Running the Docker Container

1. Build the Docker Image:
    - Run this command in the project directory:

```
docker build -t web-app-docker .
```

2.Run the Docker Container:

- Expose the app on port 5000:

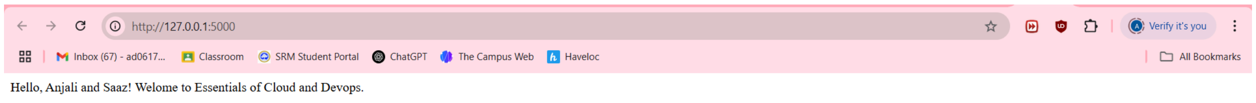
```
docker run -d -p 5000:5000 web-app-docker
```

Step 5: Testing the Application

1. Open a browser and go to:

```
http://localhost:5000
```

2. You should see the message



Hello, Anjali and Saaz! Welome to Essentials of Cloud and Devops.

Step 6: Freezing Dependencies

1. Freeze your project dependencies

```
pip freeze > requirements.txt
```

Step 7: Stopping and Cleaning Up Containers

1. List running containers:

```
docker ps
```

2.Stop the running container using the Container ID:

```
docker stop <container_id>
```

Result:

- Successfully created and ran a basic Flask web application inside a Docker container.
- The application was accessible from a web browser using the URL http://localhost:5000.
- Docker ensured the app ran in an isolated and consistent environment, regardless of the host machine.
- This project demonstrated how Docker simplifies deployment and enhances application portability.