
📄 Dockerized Flask App with Database & CI/CD

📌 Objective

To containerize a Flask application with PostgreSQL using Docker and Docker Compose, and implement automated build and deployment using GitHub Actions with a self-hosted runner on an Ubuntu server.

🏗️ Project Architecture

Developer → Git Push → GitHub → Self-Hosted Runner → Docker Compose → Flask + PostgreSQL → Live Application

---

🧱 Technologies Used

- Python 3.10
- Flask
- PostgreSQL 15
- Docker
- Docker Compose
- GitHub Actions
- Ubuntu (AWS Lightsail)

📂 Project Structure

```
flask-docker-app/
│
├── app.py
├── requirements.txt
├── Dockerfile
├── docker-compose.yml
└── .github/
    └── workflows/
        └── deploy.yml
```

🐳 Step 1: Flask Application Setup

app.py

```python
from flask import Flask
import psycopg2
import os

app = Flask(__name__)

def get_db_connection():
    conn = psycopg2.connect(
        host=os.environ.get("POSTGRES_HOST"),
        database=os.environ.get("POSTGRES_DB"),
        user=os.environ.get("POSTGRES_USER"),
        password=os.environ.get("POSTGRES_PASSWORD")
    )
    return conn

@app.route("/")
def home():
    try:
        conn = get_db_connection()
        conn.close()
        return "Flask + PostgreSQL is connected successfully!"
    except Exception as e:
        return f"Database connection failed: {e}"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

📦 Step 2: Dockerfile

```
FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

This Dockerfile:

- Uses lightweight Python base image
- Installs dependencies
- Copies application code
- Exposes port 5000
- Runs Flask application

🧩 Step 3: Docker Compose Configuration

```
version: "3.9"

services:
  web:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      POSTGRES_HOST: db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: postgres

  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Docker Compose:

- Runs Flask and PostgreSQL services
- Enables internal networking
- Uses service name `db` as database host
- Creates persistent volume for database data

🔄 Step 4: GitHub Actions CI/CD Pipeline

Workflow File: `.github/workflows/deploy.yml`

```
name: Deploy Flask App

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: self-hosted

    steps:
```

```
  - name: Checkout code
    uses: actions/checkout@v3

  - name: Stop existing containers
    run: docker-compose -p flaskapp down || true

  - name: Build and start containers
    run: docker-compose -p flaskapp up -d --build
```

🚀 Step 5: Self-Hosted Runner Setup

The self-hosted GitHub Actions runner was installed on an Ubuntu server.

Steps:

1. Created runner directory in home folder
2. Downloaded runner package
3. Configured with repository URL and token
4. Installed as system service:

```
sudo ./svc.sh install
sudo ./svc.sh start
```

Runner runs in background and listens for deployment jobs.

🔥 Deployment Flow

1. Developer pushes code to `main` branch.

2. GitHub Actions workflow triggers automatically.
3. Self-hosted runner executes workflow on Ubuntu server.
4. Docker Compose:
    - Stops old containers
    - Rebuilds images
    - Starts updated containers
5. Application becomes live automatically.

🌐 Accessing Application

Application URL:

http://<server-ip>:5000

✅ Task Completion Checklist

✔ Flask application created
✔ PostgreSQL database configured
✔ Dockerfile created
✔ Docker Compose configured
✔ Inter-service communication verified
✔ GitHub Actions workflow added
✔ Self-hosted runner configured
✔ Automatic deployment implemented

🎯 Conclusion

Successfully implemented a Dockerized Flask application integrated with PostgreSQL and automated deployment using GitHub Actions with a self-hosted runner on Ubuntu.

The solution ensures continuous deployment and zero manual intervention during updates.
