**Full Stack Application Deployment with Docker (React + Flask + PostgreSQL)**

This guide explains how to deploy a full stack application using **Docker**, including a **React frontend**, **Flask backend**, and **PostgreSQL database**. It is designed for clarity and step-by-step execution.

**Prerequisites**

* Ubuntu / Linux system
* Docker installed

**1) Frontend Setup (React)**

**1.1 Install Node and npm**

Update packages:

sudo apt update && sudo apt upgrade -y

sudo apt install -y nodejs npm

**1.2 Create React Project**

npx create-react-app reactapp

cd reactapp/src

**1.3 Update App.js**

* Copy your React code from the repository.
* Change the API base URL to point to the backend container:

const API\_BASE = "http://host.docker.internal:5000";

**1.4 Test React App**

npm start

* Verify the frontend works in your browser before containerizing.

**2) Backend Setup (Python + Flask)**

**2.1 Check Python**

python3 --version

If not installed:

sudo apt update

sudo apt install -y python3 python3-pip python3-venv

**2.2 Create and Activate Virtual Environment**

python3 -m venv venv

source venv/bin/activate

**2.3 Install Required Packages**

pip install flask flask-cors psycopg2-binary

**2.4 Create requirements.txt**

Flask

flask-cors

psycopg2-binary

gunicorn

**2.5 Create Procfile**

web: gunicorn application:application

**2.6 Update application.py**

* Copy the code from your repository.
* Update database configuration:

def get\_db\_connection():

return psycopg2.connect(

host="host.docker.internal",

database="postgres",

user="postgres",

password="123456",

port=5432

)

**2.7 Test Backend**

python3 application.py

* If permission issues occur:

sudo chown -R $USER:$USER /webapp/backend/venv

**3) Database Setup (PostgreSQL)**

**3.1 Run PostgreSQL Container**

docker run -d \

--name my-postgres \

-e POSTGRES\_USER=postgres \

-e POSTGRES\_PASSWORD=123456 \

-e POSTGRES\_DB=postgres \

-p 5432:5432 \

postgres:16

**3.2 Access Database**

docker exec -it my-postgres psql -U postgres -d postgres

**3.3 Create Table**

CREATE TABLE entries (

id SERIAL PRIMARY KEY,

text\_entry TEXT NOT NULL

);

**4) Dockerizing Backend**

**4.1 Backend Dockerfile**

# Base image

FROM python:3.12-slim

# Set working directory

WORKDIR /app

# Copy requirements and install

COPY requirements.txt .

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

# Copy backend code

COPY . .

# Expose port

EXPOSE 5000

# Command to run

CMD ["python", "application.py"]

**4.2 Build and Run Backend Container**

docker build -t backend .

docker run -d -p 5000:5000 backend

**5) Dockerizing Frontend**

**5.1 Frontend Dockerfile**

# Use official Node image

FROM node:20

# Copy package.json and package-lock.json

COPY package\*.json ./

# Install dependencies

RUN npm install

# Copy all app code

COPY . .

# Expose the port your app runs on

EXPOSE 3000

# Run the app

CMD ["npm", "start"]

**5.2 Build and Run Frontend Container**

docker build -t react-frontend .

docker run -d -p 3000:3000 react-frontend

**6) Final Checks**

1. Verify **frontend** can reach the **backend API**.
2. Verify **backend** connects to **PostgreSQL database**.
3. Ensure **all containers are running** without issues:

docker ps

* You should see react-frontend, backend, and my-postgres containers running.

Now your full stack application is deployed and accessible at:

* Frontend: <http://localhost:3000>
* Backend API: <http://localhost:5000>

**DevMasterMind ( Muhammad Usman )**  
Follow me for more tutorials and projects:

* GitHub: https://github.com/UsmanNadeem575
* Website: <https://dev-master-mind.vercel.app/>
* App: <https://play.google.com/store/apps/dev?id=5401515301307159565>
* Youtube: <https://www.youtube.com/@Dev-Master-Mind>

Thanks for reading and watching!