Module 10: Deployment & CI/CD

🚀 Overview

- **Dockerizing** a FastAPI app
- Managing secrets with `.env`
- Deploying to **Render**, **Railway**, or **AWS**
- Intro to **CI/CD** with GitHub Actions

🐳 Dockerizing a FastAPI App

1. Create a `Dockerfile`
2. Use an official Python base image
3. Install dependencies
4. Run with **Uvicorn**
5. Test locally before deployment

🐳 Backend Dockerfile

```dockerfile
FROM python:3.11-slim

WORKDIR /app

# Install system dependencies for psycopg2
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
 && rm -rf /var/lib/apt/lists/*

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

# Copy backend code
COPY . .

EXPOSE 8000

# Start FastAPI with Uvicorn
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

🐳 Frontend Dockerfile

```dockerfile
# Build stage
FROM node:18-alpine AS build

WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install

COPY . .
RUN npm run build

# Serve stage
FROM nginx:alpine

# Copy Vite build output to Nginx HTML folder
COPY --from=build /app/dist /usr/share/nginx/html

# Optional: SPA routing support
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

🐳 Docker compose file

```yaml
version: "3.9"

services:
  db:
    image: postgres:15
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres123!
      - POSTGRES_DB=fastapi_week10
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5440:5432"
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://postgres:postgres123!@db:5440/fastapi_week10
    depends_on:
      - db

  frontend:
    build: ./frontend/my-app
    ports:
      - "3000:80"
    depends_on:
      - backend

volumes:
  postgres_data:
```

Database

```yaml
db:
    image: postgres:15
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres123!
      - POSTGRES_DB=fastapi_week10
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5440:5432"
```

Database - explanation (1)

- **environments:**
  - POSTGRES_USER, POSTGRES_PASSWORD → Creates a database superuser and sets the password
  - POSTGRES_DB=fastapi_week10 → Creates a database named fastapi_week10 on startup.
- **volumes:**
  - Stores PostgreSQL's data files inside the container --> your database data persists even if the container is deleted.
- **ports: "5440:5432"**
  - container port: 5432, localhost:5440.

Database - explanation (2)

- How you would connect?
  - From your host (outside Docker):

```
psql -h localhost -p 5440 -U postgres -d fastapi_week10
```

- Inside another container (e.g., your FastAPI app):

```
host=db
port=5432
user=postgres
password=postgres123!
database=fastapi_week10
```

Backend

```yaml
backend:
    build: ./backend
    ports:
      - "8000:8000"
    depends_on:
      - db
```

Backend - explanation

- `build: ./backend`
  - Tells Docker to build an image from the Dockerfile located in the `./backend` directory.
- `ports:`
  - "8000:8000": maps host port 8000 → container port 8000.
  - you can access it at http://localhost:8000 on your machine.
- `depends_on:`
  - db: Ensures that Docker starts the `db` service before backend

Frontend

```yaml
frontend:
   build: ./frontend/my-app
   ports:
     - "3000:80"
   depends_on:
     - backend
```

Frontend - explanation

- `build: ./frontend/my-app`
  - Docker will build an image using the Dockerfile inside ./frontend/my-app.
- `ports:`
  - "3000:80": maps host port 3000 → container port 80.
  - Inside the container, the app is served on port 80 (like Nginx serving static files).
  - On your local machine, you open the app at 👉 http://localhost:3000.

🔑 Using .env & Environment Variables

- Store secrets like:
  - Database URLs
  - API keys
  - Debug flags
- Use **python-dotenv** or **Pydantic Settings**
- Never commit `.env` to GitHub

☁️ Deployment Options

- **Render**
  - Simple, free tier
  - Auto deploy from GitHub
- **Railway**
  - Great for quick prototypes
  - Easy database integration
- **AWS**
  - Full flexibility
  - Options: EC2, Elastic Beanstalk, ECS, Lambda

⚙️ Introduction to CI/CD

- **Continuous Integration (CI)**
  - Automated tests
  - Linting & type checks
- **Continuous Deployment (CD)**
  - Auto-deploy after successful CI
- Tools: **GitHub Actions**, GitLab CI, CircleCI

💡 GitHub Actions Workflow Example

```yaml
name: CI/CD
on:
  push:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with:
          python-version: "3.11"
      - run: pip install -r requirements.txt
      - run: pytest
```

🎯 Remember

- Containerize with Docker for portability
- Manage secrets via .env
- Choose a hosting platform that fits your needs
- Automate testing & deployment with CI/CD