Simple Docker and Docker Compose Configuration for a Three-Service Application

Grok

June 1, 2025

1 Introduction

This document provides basic Dockerfiles for a front-end (React), backend (Node.js/Express), and database (PostgreSQL) service, along with a minimal Docker Compose configuration. The setup is designed for simplicity, with essential configurations and comments explaining adaptability to requirement changes.

2 Docker Compose Configuration

Basic orchestration of the three services.

Listing 1: docker-compose.yml

```yaml
version: '3.8'
services:
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:3000" # Change if frontend framework uses a different port (e.g.,
          Angular: 4200)
    depends_on:
      - backend # Update if additional dependencies are added
    environment:
      - REACT_APP_API_URL=http://backend:5000 # Update if backend URL or
          protocol changes
    networks:
      - app-network

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "5000:5000" # Change if backend framework uses a different port (e.g.,
          FastAPI: 8000)
    depends_on:
      - database # Update if additional dependencies are added
    environment:
      - DATABASE_URL=postgresql://user:password@database:5432/mydb # Update for
          DB type/credentials
    networks:
      - app-network

  database:
    image: postgres:16 # Change to mysql:8.0 or mongodb:latest if DB type
        changes
    environment:
      - POSTGRES_USER=user # Update for new user
      - POSTGRES_PASSWORD=password # Update for security
      - POSTGRES_DB=mydb # Update for new database name
    volumes:
      - db-data:/var/lib/postgresql/data # Change volume path for different DB
    networks:
      - app-network

volumes:
  db-data: # Remove or rename if volume requirements change

networks:
  app-network:
    driver: bridge # Change to overlay for swarm mode if needed
```

3 Frontend Dockerfile

Basic Dockerfile for a React front-end.

Listing 2: frontend/Dockerfile

```
# Use Node.js base image; change version (e.g., node:18) if required
FROM node:20
WORKDIR /app
# Copy and install dependencies
COPY package.json .
RUN npm install
# Copy source code
COPY . .
# Expose port; align with docker-compose.yml
EXPOSE 3000
# Start app; change if using different start command (e.g., 'npm start' for
    dev)
CMD ["npx", "serve", "-s", "build"]
```

4 Backend Dockerfile

Basic Dockerfile for a Node.js/Express backend.

Listing 3: backend/Dockerfile

```
# Use Node.js base image; change version if needed
FROM node:20
WORKDIR /app
# Copy and install dependencies
COPY package.json .
RUN npm install
# Copy source code
COPY . .
# Expose port; align with docker-compose.yml
EXPOSE 5000
# Start app; change if using different framework (e.g., 'python app.py' for
    FastAPI)
CMD ["node", "server.js"]
```

5 How to Run

1. Create directories `frontend` and `backend`, and place Dockerfiles.
2. Create `docker-compose.yml` in the project root.
3. Ensure environment variables are set in a `.env` file or directly in `docker-compose.yml`.
4. Run `docker-compose up -build` to start services.
5. Access the front-end at `http://localhost:3000`.
6. Stop services with `docker-compose down`.

6 Adapting to Requirement Changes

- **Change Frontend Framework**: Update `frontend/Dockerfile` (e.g., use `ng build` for Angular) and port in `docker-compose.yml`.
- **Change Backend Framework**: Update `backend/Dockerfile` (e.g., use Python image for FastAPI) and port in `docker-compose.yml`.
- **Change Database**: Update `docker-compose.yml` image and volume path (e.g., `mysql:8.0` for MySQL).
- **Add Services**: Add new services to `docker-compose.yml` with appropriate dependencies and networks.