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

Grok

June 1, 2025

1 Introduction

This document provides Dockerfiles for a front-end (React), backend (Node.js/Express), and database (PostgreSQL) service, with a Docker Compose configuration incorporating complex requirements: multi-stage builds, health checks, custom networks with static IPs, resource limits, logging, and conditional dependencies.

2 Docker Compose Configuration

Advanced orchestration with health checks and resource limits.

Listing 1: docker-compose.yml

```
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:
        condition: service_healthy # Wait for backend to be healthy
    environment:
      - REACT_APP_API_URL=http://backend:5000 # Update if backend URL or protocol changes
      - NODE_ENV=production # Change to 'development' for dev mode
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000"] # Update if health endpoint changes
      interval: 30s
      timeout: 10s
      retries: 3
    deploy:
      resources:
```

```
      limits:
        cpus: '0.5' # Adjust CPU limit based on performance needs
        memory: 512M # Adjust memory limit as required
    networks:
      app-network:
        ipv4_address: 172.20.0.2 # Change if network subnet or IP scheme changes
    logging:
      driver: "json-file" # Change to 'syslog' or 'fluentd' for centralized logging
      options:
        max-size: "10m"
        max-file: "3"

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "5000:5000" # Change if backend framework uses a different port (e.g., FastAPI: 8000)
    depends_on:
      database:
        condition: service_healthy # Wait for database to be healthy
    environment:
      - DATABASE_URL=postgresql://user:password@database:5432/mydb # Update for DB type/credentials
      - NODE_ENV=production # Change for dev mode
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"] # Update if health endpoint changes
      interval: 30s
      timeout: 10s
      retries: 3
    deploy:
      resources:
        limits:
          cpus: '0.75' # Adjust based on backend load
          memory: 768M # Adjust memory as needed
    networks:
      app-network:
        ipv4_address: 172.20.0.3 # Change if IP scheme changes
    logging:
      driver: "json-file" # Change for centralized logging
      options:
        max-size: "10m"
        max-file: "3"

  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
      - ./db-backup:/backup # Add backup volume; update path if backup strategy
          changes
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U user"] # Update for different DB (e.g
          ., mysqladmin for MySQL)
      interval: 10s
      timeout: 5s
      retries: 5
    deploy:
      resources:
        limits:
          cpus: '0.5' # Adjust for DB performance
          memory: 512M # Adjust as needed
    networks:
      app-network:
        ipv4_address: 172.20.0.4 # Change if IP scheme changes
    logging:
      driver: "json-file" # Change for centralized logging
      options:
        max-size: "10m"
        max-file: "3"

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

networks:
  app-network:
    driver: bridge # Change to overlay for swarm mode
    ipam:
      config:
        - subnet: 172.20.0.0/16 # Change subnet if network conflicts arise
```

3 Frontend Dockerfile

Multi-stage build for optimized React front-end image.

Listing 2: frontend/Dockerfile

```
# Build stage; change base image if Node.js version changes
FROM node:20 AS build
WORKDIR /app
COPY package.json .
# Install dependencies; change to 'yarn install' if using Yarn
RUN npm install
```

```
COPY . .
# Build app; change if using different framework (e.g., 'ng build' for Angular)
RUN npm run build

# Production stage; change base image if using different server (e.g., nginx:alpine)
FROM node:20-slim
WORKDIR /app
# Install serve; change if using different static server
RUN npm install -g serve
# Copy build artifacts from build stage
COPY --from=build /app/build ./build
# Expose port; align with docker-compose.yml
EXPOSE 3000
# Start server; change if using different server command
CMD ["serve", "-s", "build", "-l", "3000"]
```

4 Backend Dockerfile

Multi-stage build for optimized Node.js/Express backend image.

Listing 3: backend/Dockerfile

```
# Build stage; change base image if Node.js version changes
FROM node:20 AS build
WORKDIR /app
COPY package.json .
# Install dependencies; change to 'yarn install' if using Yarn
RUN npm install
COPY . .

# Production stage; change base image for different runtime needs
FROM node:20-slim
WORKDIR /app
COPY --from=build /app .
# 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, health check, and volume paths (e.g., `mysql:8.0` for MySQL).
- **Adjust Resource Limits**: Modify `deploy.resources` in `docker-compose.yml` for performance tuning.
- **Change Network Configuration**: Update subnet or IPs in `docker-compose.yml` for network conflicts.
- **Modify Logging**: Change `logging.driver` to `syslog` or `fluentd` for centralized logging.