Docker Project

Contents

1. Introduction

1.1 Project Overview

This project involves building a three-tier application using Docker, consisting of a Backend service, a Database, and a Proxy service. The Backend is built using a multi-stage Dockerfile, the Database credentials are securely managed, and the Proxy handles HTTPS traffic.

1.2 Objectives

- **Backend**: Implement a backend service with a multi-stage Dockerfile.
- **Database**: Set up a PostgreSQL database with credentials managed securely.
- **Proxy**: Configure a reverse proxy with HTTPS support.
- **Networking**: Ensure all containers operate on separate Docker networks.
- **Automation**: Utilize Docker Compose to manage the entire stack.

2. Architecture

2.1 System Design

The application is structured into three distinct layers:

1. **Backend**: Handles application logic and business rules.
2. **Database**: Stores application data securely.
3. **Proxy**: Manages secure HTTPS traffic and routing.

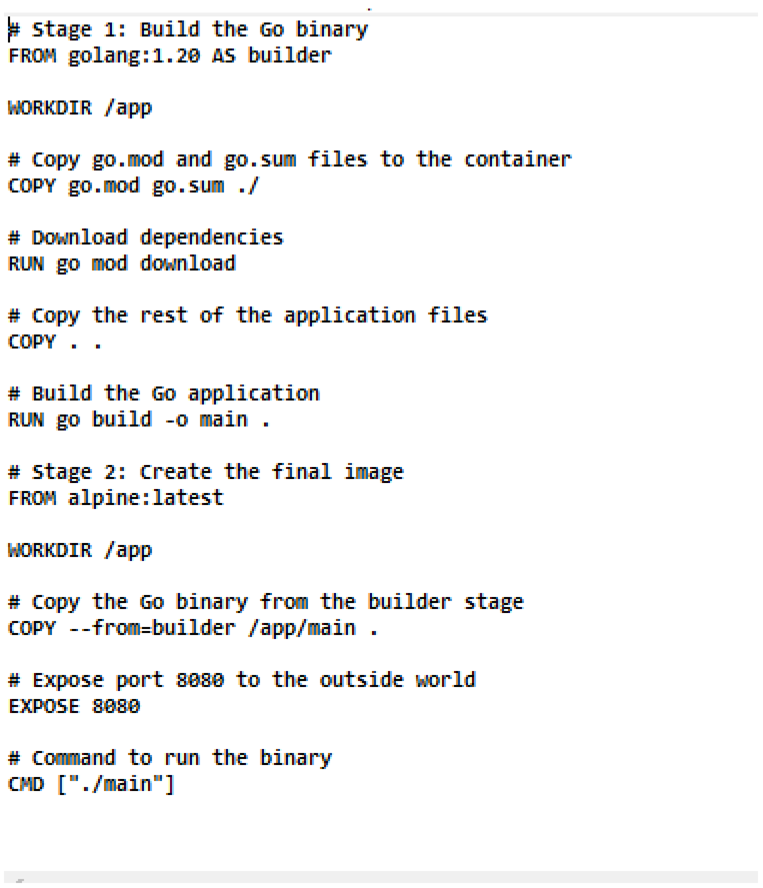
2.2 Network Architecture

Each service operates on its dedicated Docker network:

- **Backend Network**: Handles communication between the Backend and the Database.
- **Database Network**: Isolated for database access.
- **Proxy Network**: Manages traffic routing to ensure security.

3. Implementation

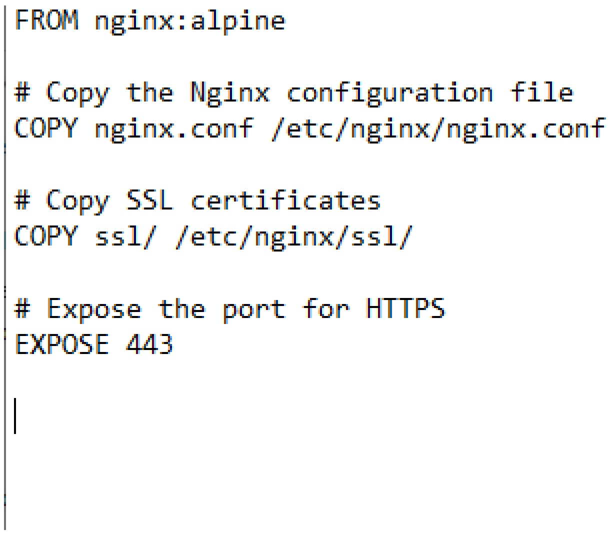
3.1 Dockerfile for Backend



```
# Stage 1: Build the Go binary
FROM golang:1.20 AS builder

WORKDIR /app

# Copy go.mod and go.sum files to the container
COPY go.mod go.sum ./

# Download dependencies
RUN go mod download

# Copy the rest of the application files
COPY . .

# Build the Go application
RUN go build -o main .

# Stage 2: Create the final image
FROM alpine:latest

WORKDIR /app

# Copy the Go binary from the builder stage
COPY --from=builder /app/main .

# Expose port 8080 to the outside world
EXPOSE 8080

# Command to run the binary
CMD ["./main"]
```

- **Stage 1**: Builds the application in a Golang environment.
- **Stage 2**: Uses a lightweight Alpine image to run the built application.

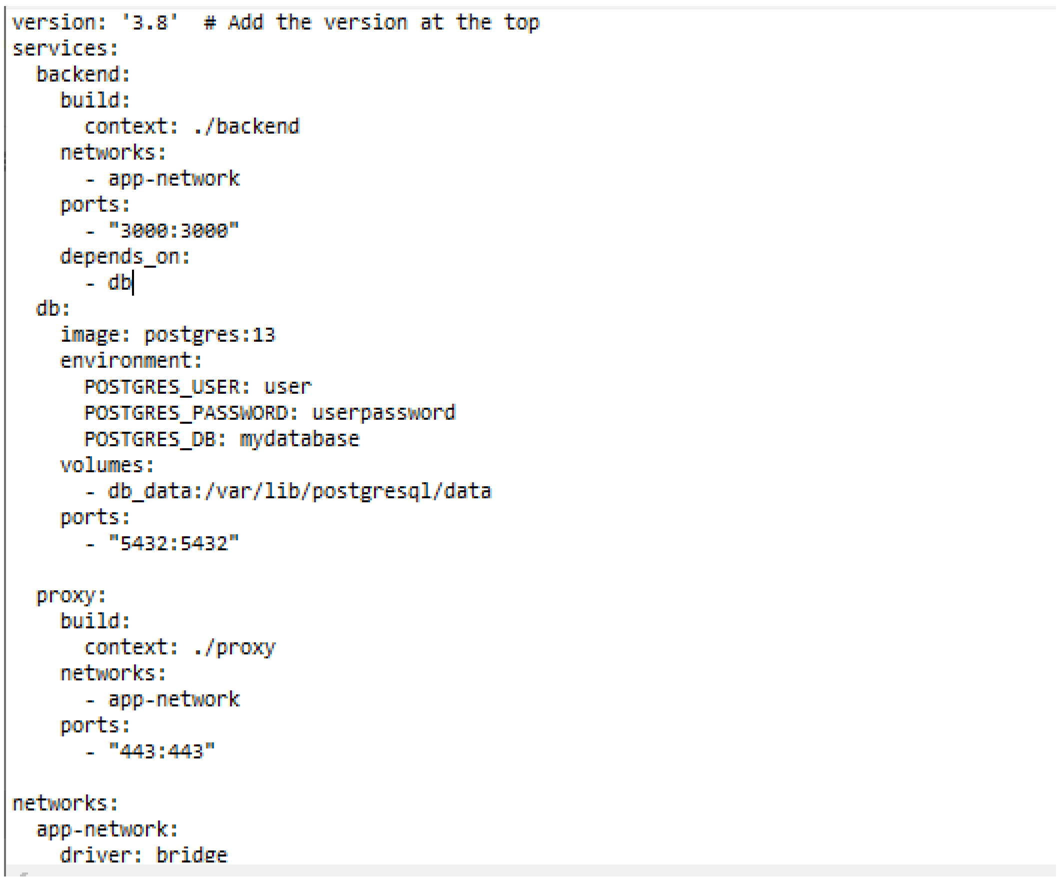
3.2 Dockerfile for Proxy



```
FROM nginx:alpine

# Copy the Nginx configuration file
COPY nginx.conf /etc/nginx/nginx.conf

# Copy SSL certificates
COPY ssl/ /etc/nginx/ssl/

# Expose the port for HTTPS
EXPOSE 443
```

Configures Nginx with SSL certificates for HTTPS support.

3.3 Docker Compose Configuration



```
version: '3.8'  # Add the version at the top
services:
  backend:
    build:
      context: ./backend
    networks:
      - app-network
    ports:
      - "3000:3000"
    depends_on:
      - db
  db:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: userpassword
      POSTGRES_DB: mydatabase
    volumes:
      - db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  proxy:
    build:
      context: ./proxy
    networks:
      - app-network
    ports:
      - "443:443"

networks:
  app-network:
    driver: bridge
```

- **Backend Service**: Built from the `./backend` directory.
- **Database Service**: Configured with environment variables for secure credential management.
- **Proxy Service**: Configured with SSL certificates for HTTPS.

4. Security Considerations

4.1 Database Security

- **Credentials Management**: Database credentials are managed using environment variables to avoid hardcoding sensitive information.
- **Network Isolation**: The database runs on a separate network to limit access.

4.2 Proxy Security

- **HTTPS Configuration**: Proxy uses SSL certificates to ensure encrypted traffic.
- **Configuration Files**: Nginx configuration is managed from the host machine to ensure secure and flexible updates.

5. Testing and Validation

5.1 Verifying Services

- **Backend**: Ensure the backend service runs without errors and can connect to the database.
- **Database**: Verify that the database initializes correctly and accepts connections.
- **Proxy**: Confirm that the proxy correctly routes traffic and serves HTTPS.

5.2 Troubleshooting

- **Common Issues**: Resolve issues related to missing files or configuration errors by reviewing logs and configuration files.
- **Error Logs**: Use Docker logs to diagnose and address service-specific issues.

6. Conclusion

This documentation outlines the setup and configuration of a three-tier application using Docker. By following these guidelines, you can ensure that the application is secure, scalable, and maintainable.