Web Application Containerization with Docker and Nginx

Your Name

August 18, 2024

1 Introduction

This document outlines the architecture and setup of a web application optimized using Docker containerization. The application consists of a React frontend, a Node.js API backend, and a MongoDB database, all running in separate Docker containers. Additionally, an Nginx reverse proxy has been configured to handle HTTPS requests and forward them to the appropriate internal services.

2 Project Components

2.1 React Frontend

The React frontend provides the user interface for the application. It runs on port 5173 within its Docker container and communicates with the API backend to handle user authentication and task management.

2.2 Node.js API Backend

The API backend, built with Express.js, handles user authentication and CRUD operations for tasks. It runs on port 3002 within its Docker container and connects to the MongoDB database to store user and task data.

2.3 MongoDB Database

MongoDB is used as the database to store all the application's data, including user accounts and tasks. It runs on port 27017 within its Docker container.

3 Docker Setup

3.1 Dockerfiles

Each component of the application has its own Dockerfile:

- **React Frontend:** Builds the React application and serves it.
- **Node.js API Backend:** Installs dependencies and runs the Express server.
- **MongoDB Database:** Uses the official MongoDB image.

3.2 Docker Compose

Docker Compose is used to orchestrate the three containers, allowing them to communicate with each other internally:

```
version: '3.8'

services:
  front:
    build: ./front
    ports:
      - "5173:5173"
    depends_on:
    - api

  api:
    build: ./back
    ports:
      - "3002:3002"
    depends_on:
    - mongo

  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
```

4 Nginx Reverse Proxy

To secure the application and handle HTTPS traffic, an Nginx container is configured as a reverse proxy. It terminates SSL connections and forwards requests to the appropriate service:

4.1 Nginx Configuration

```
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        proxy_pass http://front:5173;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/ {
        proxy_pass http://api:3002;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

4.2 SSL Setup

For HTTPS, SSL certificates are installed on the Nginx server. The Nginx container handles HTTPS requests and forwards them internally over HTTP.

5 Conclusion

This setup effectively containerizes the web application, making it easier to manage and scale. The use of Docker and Docker Compose simplifies deployment, while the Nginx reverse proxy adds an additional layer of security.

Next Steps: As a future improvement, learning Kubernetes and AWS will be essential to mastering orchestration and cloud infrastructure management for more advanced projects.