Setting Up a Full Node.js and Docker Project

Eng. Ahmed M'etwalli

December 12, 2024

Contents

1 Introduction

This document provides step-by-step instructions for setting up a Dockerized project using Node.js for the backend, Nginx for the frontend, and a docker-compose.yml file to orchestrate the setup. It explains the folder structure, backend and frontend configurations, and instructions for running the project.

2 Project Structure

The project has the following structure:

```
form-docker/
          app/                      # Frontend files
                index.html            # Main HTML file
                style.css             # CSS for styling
                script.js             # Frontend JavaScript
          server/                   # Backend files
                server.js             # Node.js backend server
                package.json          # Node.js dependencies and scripts
                package-lock.json     # Dependency lock file
          Dockerfile                # Dockerfile for backend and frontend
          docker-compose.yml        # Docker Compose configuration
          .env                      # Environment variables (optional)
```

3 Backend: Node.js Server

The backend handles form submissions using Node.js and Express.js.

3.1 Backend Configuration

The Dockerfile defines the backend environment:

```

Backend stage

FROM node:18-alpine as backend

WORKDIR /app

Install dependencies

COPY server/package*.json ./
RUN npm install

Copy application files

COPY server/ ./

Expose backend port

EXPOSE 3000

Start the backend

CMD ["npm", "start"]
```

3.2 Backend Code

The server.js file contains the backend logic to handle form submissions:

```
const express = require("express");
const bodyParser = require("body-parser");

const app = express();
```

```
const PORT = 3000;

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Handle form submission
app.post("/submit", (req, res) => {
const { name, email, password } = req.body;

if (!name || !email || !password) {
return res.status(400).json({ error: "All fields are required." });
}

if (password.length < 6) {
return res.status(400).json({ error: "Password must be at least 6 characters." });
}

console.log("Form submitted:", { name, email, password });

res.status(200).json({ message: "Form submitted successfully!" });
});

app.listen(PORT, () => {
console.log(Server running on http://localhost:${PORT});
});
```

3.3 Package Configuration

The package.json file defines the backend dependencies:

```
{
"name": "form-backend",
"version": "1.0.0",
"description": "Backend for form submission",
"main": "server.js",
"scripts": {
"start": "node server.js"
},
"dependencies": {
"express": "^4.18.2",
"body-parser": "^1.20.2"
}
}
```

4 Frontend: HTML Form

The frontend is served using an Nginx container and includes a simple HTML form with validation.

4.1 Frontend Configuration

The Dockerfile includes the frontend setup:

```

Frontend stage

FROM nginx:alpine as frontend

Copy frontend files

COPY app/ /usr/share/nginx/html

Expose frontend port

EXPOSE 80
```

4.2 HTML File

The index.html contains the form:

```
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>
  <span class="error" id="emailError"></span>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required minlength="6">
  <span class="error" id="passwordError"></span>

  <button type="submit">Submit</button>
</form>

\subsection{JavaScript File}
The script.js handles client-side validation:

\begin{lstlisting}[language=javascript]
document.getElementById("userForm").addEventListener("submit", async function (e) {
e.preventDefault();
const name = document.getElementById("name").value.trim();
const email = document.getElementById("email").value.trim();
const password = document.getElementById("password").value.trim();

try {
const response = await fetch("http://localhost:3000/submit", {
method: "POST",
headers: {
"Content-Type": "application/json"
},
body: JSON.stringify({ name, email, password })
});

const result = await response.json();

if (response.ok) {
  alert(result.message);
} else {
  alert(result.error);
}

} catch (error) {
console.error("Error submitting form:", error);
}
});
```

5 Docker Compose Configuration

The docker-compose.yml orchestrates the setup:

```
services:
backend:
build:
context: .
dockerfile: Dockerfile
target: backend
container_name: backend-container
ports:
- "3000:3000"
volumes:
- ./server:/app

frontend:
build:
```

```
context: .
dockerfile: Dockerfile
target: frontend
container_name: frontend-container
ports:
- "8080:80"
volumes:
- ./app:/usr/share/nginx/html
```

6 How to Start the Project

1. Clone or set up the project structure as described.
2. Build and run the containers:

```
docker-compose up --build
```

3. Open a browser and navigate to `http://localhost:8080`.
4. Submit the form and verify that the backend processes the data.
5. Check logs to confirm:
   - Backend logs: Submitted data.
   - Frontend logs: No errors.