Dockerización de un proyecto fullstack con React y Node.js

Julian F. Latorre

17 de septiembre de 2024

Índice

1. Introducción

En esta guía, aprenderemos cómo llevar un proyecto fullstack con frontend en React y backend en Node.js a Docker. Docker nos permite empaquetar nuestra aplicación y sus dependencias en contenedores, facilitando la implementación y el despliegue.

2. Requisitos previos

- Docker instalado en tu sistema
- Un proyecto fullstack existente con React (frontend) y Node.js (backend)
- Conocimientos básicos de React, Node.js y Docker

3. Estructura del proyecto

Asumiremos la siguiente estructura de proyecto:

```
proyecto-fullstack/

 frontend/
    src/
    package.json
    ...

 backend/
    src/
    package.json
    ...

 docker-compose.yml
```

4. Dockerización del frontend (React)

4.1. Opción 1: Construcción de la aplicación React

Esta opción es ideal para producción o cuando quieres servir una versión optimizada de tu aplicación.

Crea un archivo llamado `Dockerfile` en el directorio `frontend/`:

```
# Etapa de construcción
FROM node:14 as build

WORKDIR /app

COPY package*.json ./
```

```
RUN npm install

COPY . .

RUN npm run build

# Etapa de producción
FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

4.2. Explicación del Dockerfile del frontend (Opción 1)

- Usamos una construcción de múltiples etapas para optimizar el tamaño de la imagen final.
- La primera etapa construye la aplicación React.
- La segunda etapa copia los archivos construidos a un servidor Nginx para servir el contenido estático.

4.3. Opción 2: Iniciar la aplicación React en modo de desarrollo

Esta opción es preferible durante el desarrollo o cuando no es posible o deseable construir la aplicación.

Crea un archivo llamado `Dockerfile.dev` en el directorio `frontend/`:

```
FROM node:14

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

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

4.4. Explicación del Dockerfile.dev del frontend (Opción 2)

- Usamos la imagen base de Node.js.

- Copiamos los archivos de dependencias y las instalamos.
- Copiamos todo el código fuente.
- Exponemos el puerto 3000 (el puerto por defecto para aplicaciones Create React App).
- Iniciamos la aplicación en modo de desarrollo con `npm start`.

5. Dockerización del backend (Node.js)

Crea un archivo llamado `Dockerfile` en el directorio `backend/`:

```
FROM node:14

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5000

CMD ["node", "src/index.js"]
```

5.1. Explicación del Dockerfile del backend

- Usamos la imagen base de Node.js.
- Copiamos los archivos de dependencias y las instalamos.
- Copiamos el resto del código fuente.
- Exponemos el puerto 5000 (asegúrate de que coincida con el puerto de tu aplicación).
- Iniciamos la aplicación Node.js.

6. Creación del archivo docker-compose.yml

En la raíz del proyecto, crea un archivo `docker-compose.yml`:

```
version: '3'
services:
  frontend-prod:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "80:80"
```

```
    depends_on:
      - backend

  frontend-dev:
    build:
      context: ./frontend
      dockerfile: Dockerfile.dev
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
      - /app/node_modules
    environment:
      - CHOKIDAR_USEPOLLING=true
    depends_on:
      - backend

  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - NODE_ENV=production
```

6.1. Explicación de docker-compose.yml

- Definimos tres servicios: frontend-prod, frontend-dev y backend.
- El frontend en modo producción usa el puerto 80.
- El frontend en modo desarrollo usa el puerto 3000 y tiene configuración para desarrollo en tiempo real.
- El backend usa el puerto 5000.
- Usamos volúmenes para el frontend en modo desarrollo para permitir la recarga en caliente.

7. Ajustes necesarios en el backend

Actualiza el archivo principal de tu servidor Node.js (por ejemplo, `index.js` o `server.js`) para usar el puerto 5000:

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 5000;

// ... resto de la configuración del servidor

app.listen(PORT, () => {
  console.log(`Servidor corriendo en el puerto ${PORT}`);
});
```

8. Ajustes en el frontend

Si tu frontend hace llamadas al backend, asegúrate de actualizar la URL del backend en tu código. Por ejemplo, si estás usando una variable de entorno para la URL del API:

En tu archivo `.env` del frontend:

```
REACT_APP_API_URL=http://localhost:5000
```

Y en tu código React:

```
const apiUrl = process.env.REACT_APP_API_URL || 'http://localhost
    :5000';
// Usa apiUrl para tus llamadas fetch o axios
```

9. Ejecutar la aplicación con Docker Compose

Para iniciar la aplicación en modo de producción:

```
docker-compose up --build frontend-prod backend
```

Para iniciar la aplicación en modo de desarrollo:

```
docker-compose up --build frontend-dev backend
```

10. Consideraciones adicionales

10.1. Variables de entorno

Para manejar configuraciones específicas de entorno, considera usar archivos `.env` y la directiva `env_file` en `docker-compose.yml`.

10.2. Volúmenes

Para persistencia de datos, puedes usar volúmenes de Docker. Agrega la siguiente sección a tu servicio en `docker-compose.yml`:

```
volumes:
  - ./backend/data:/app/data
```

10.3. Redes

Docker Compose crea una red por defecto, pero puedes definir redes personalizadas si es necesario.

10.4. Comunicación entre contenedores

En una red Docker, los contenedores pueden comunicarse entre sí usando los nombres de los servicios como nombres de host. Por ejemplo, desde el frontend podrías acceder al backend usando `http://backend:5000` en lugar de `localhost`.

10.5. Variables de entorno en producción

Para un entorno de producción, considera usar variables de entorno para configurar las URLs y puertos, permitiendo mayor flexibilidad en diferentes entornos de despliegue.

11. Conclusión

Has aprendido cómo dockerizar un proyecto fullstack con React y Node.js. Esta configuración te permite desarrollar, probar y desplegar tu aplicación de manera consistente en diferentes entornos. Recuerda adaptar estas instrucciones según las necesidades específicas de tu proyecto.