



22-4-2024

Ejemplo utilizando Docker.

Sistemas Tolerante a Fallas

Brandon Missael Infante Huerta

Dockerización de una Aplicación de Chat con Node.js y Socket.io

Introducción

En este informe, describiremos el proceso de Dockerización de una aplicación de chat simple utilizando Node.js y Socket.io. La Dockerización es el proceso de encapsular una aplicación dentro de un contenedor Docker, lo que permite su ejecución de forma independiente y reproducible en cualquier entorno.

Objetivo

El objetivo de este proyecto es crear una aplicación de chat que permita la comunicación en tiempo real entre múltiples clientes utilizando un servidor Node.js y la biblioteca Socket.io.

Requisitos

Para llevar a cabo este proyecto, necesitarás lo siguiente:

- Docker instalado en tu sistema.
- Conocimientos básicos de Node.js y npm.
- Un editor de código de tu elección.

1.- Pasos:

El primer paso es organizar el proyecto en una estructura de directorios adecuada. El proyecto constará de dos partes: el servidor (backend) y el cliente (frontend).

chat-app/

```
|— server/  
|  |— package.json  
|  |— server.js  
|— client/  
    |— package.json  
    |— index.html
```

El directorio *server/* contendrá el código del servidor Node.js, mientras que el directorio *client/* contendrá el código del cliente HTML.

2. Código de la Aplicación:

Backend (Servidor)

En el archivo *server/package.json*, especificamos las dependencias necesarias para el servidor:

```
{  
  "name": "chat-server",  
  "version": "1.0.0",  
  "description": "Simple chat server using Socket.io",  
  "main": "server.js",  
  "dependencies": {  
    "express": "^4.17.1",  
    "socket.io": "^4.2.0"  
  }  
}
```

El archivo `server/server.js` contiene el código del servidor Node.js que utiliza Socket.io para la comunicación en tiempo real entre clientes.

Frontend (Cliente)

En el archivo `client/package.json`, especificamos la dependencia necesaria para el cliente:

```
{
  "name": "chat-client",
  "version": "1.0.0",
  "description": "Simple chat client using Socket.io",
  "main": "index.html",
  "dependencies": {
    "socket.io-client": "^4.2.0"
  }
}
```

El archivo `client/index.html` contiene el código HTML y JavaScript del cliente que se utilizará para interactuar con el servidor.

3.- Dockerfile:

El archivo **Dockerfile** especifica cómo construir la imagen Docker para nuestra aplicación. En este archivo, dividimos el proceso en tres etapas:

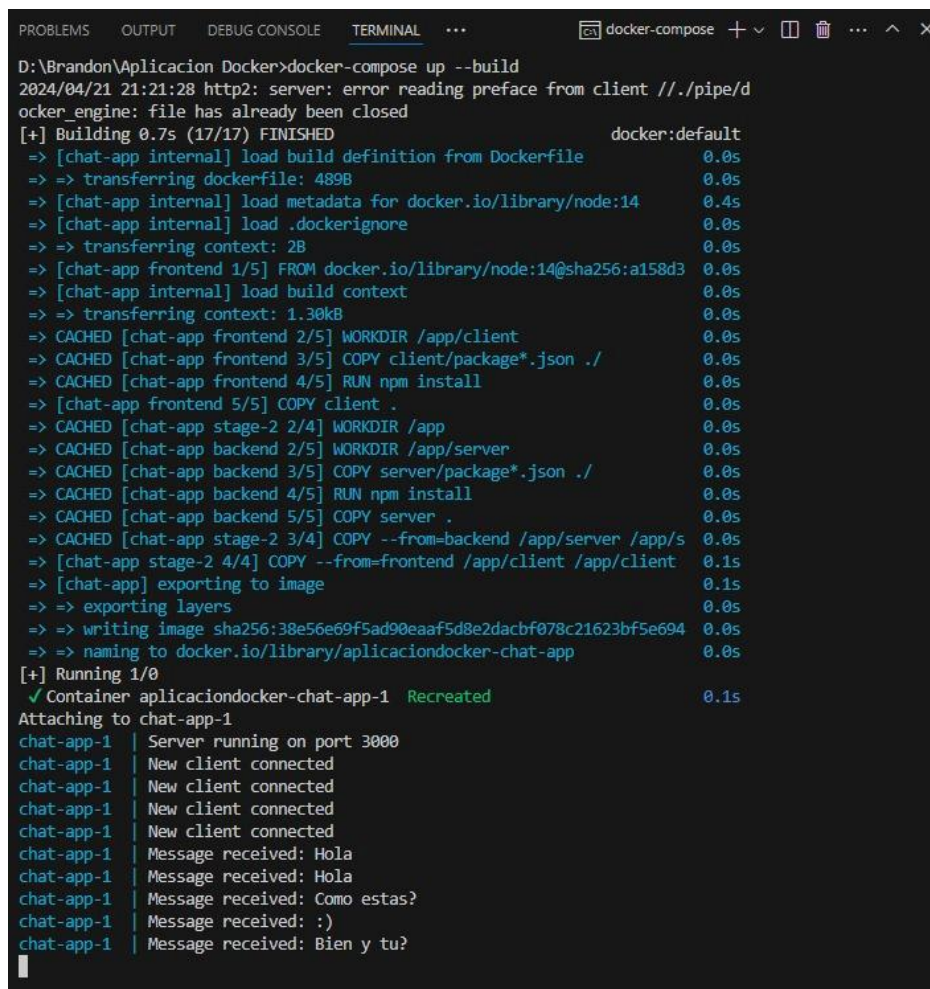
- Backend: Instalamos las dependencias del servidor y copiamos el código del servidor.
- Frontend: Instalamos las dependencias del cliente y copiamos el código del cliente.
- Combinar: Combinamos las etapas de backend y frontend en una sola imagen.

4. Docker-compose.yml

El archivo **docker-compose.yml** define los servicios necesarios para nuestra aplicación. Esto incluye la construcción de la imagen Docker y la exposición de puertos si es necesario.

5. Construcción y Ejecución de Contenedores

Una vez que se han definido los archivos Dockerfile y docker-compose.yml, ejecutamos el comando `docker-compose up --build` en la terminal para construir y ejecutar los contenedores Docker.



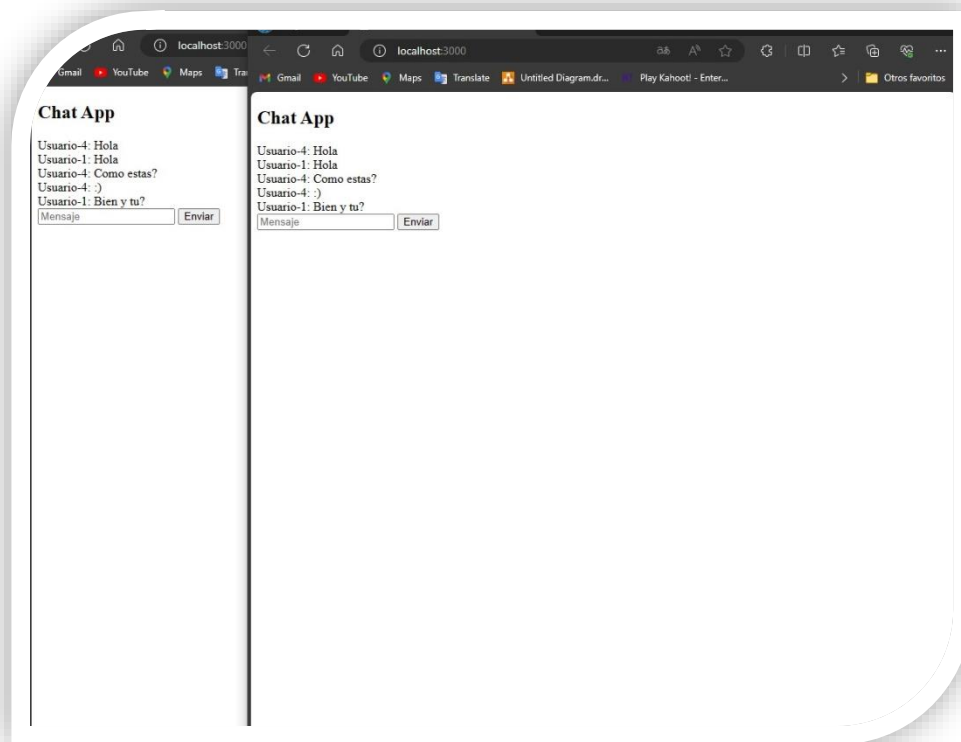
```
D:\Brandon\Aplicacion Docker>docker-compose up --build
2024/04/21 21:21:28 http2: server: error reading preface from client //./pipe/d
ocker_engine: file has already been closed
[+] Building 0.7s (17/17) FINISHED                                docker:default
=> [chat-app internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 489B                               0.0s
=> [chat-app internal] load metadata for docker.io/library/node:14 0.4s
=> [chat-app internal] load .dockerignore                         0.0s
=> => transferring context: 2B                                     0.0s
=> [chat-app frontend 1/5] FROM docker.io/library/node:14@sha256:a158d3 0.0s
=> [chat-app internal] load build context                         0.0s
=> => transferring context: 1.30kB                                0.0s
=> CACHED [chat-app frontend 2/5] WORKDIR /app/client             0.0s
=> CACHED [chat-app frontend 3/5] COPY client/package*.json ./   0.0s
=> CACHED [chat-app frontend 4/5] RUN npm install                 0.0s
=> [chat-app frontend 5/5] COPY client .                           0.0s
=> CACHED [chat-app stage-2 2/4] WORKDIR /app                     0.0s
=> CACHED [chat-app backend 2/5] WORKDIR /app/server              0.0s
=> CACHED [chat-app backend 3/5] COPY server/package*.json ./    0.0s
=> CACHED [chat-app backend 4/5] RUN npm install                   0.0s
=> CACHED [chat-app backend 5/5] COPY server .                     0.0s
=> CACHED [chat-app stage-2 3/4] COPY --from=backend /app/server /app/s 0.0s
=> [chat-app stage-2 4/4] COPY --from=frontend /app/client /app/client 0.1s
=> [chat-app] exporting to image                                  0.1s
=> => exporting layers                                           0.0s
=> => writing image sha256:38e56e69f5ad90eaaf5d8e2dacbf078c21623bf5e694 0.0s
=> => naming to docker.io/library/aplicaciondocker-chat-app      0.0s
[+] Running 1/0
✓ Container aplicaciondocker-chat-app-1 Recreated                0.1s
Attaching to chat-app-1
chat-app-1 | Server running on port 3000
chat-app-1 | New client connected
chat-app-1 | New client connected
chat-app-1 | New client connected
chat-app-1 | New client connected
chat-app-1 | Message received: Hola
chat-app-1 | Message received: Hola
chat-app-1 | Message received: Como estas?
chat-app-1 | Message received: :)
chat-app-1 | Message received: Bien y tu?
```

Podemos apreciar en la parte inferior cuando los clientes se conectan y los mensajes que se envían y reciben.



6. Acceso a la Aplicación

Finalmente, podemos acceder a la aplicación en nuestro navegador web visitando la URL especificada en el archivo docker-compose.yml. En el ejemplo proporcionado, la aplicación de chat está disponible en <http://localhost:3000>.



Conclusiones

En este informe, hemos descrito el proceso de Dockerización de una aplicación de chat simple utilizando Node.js y Socket.io. La Dockerización nos permite ejecutar la aplicación de forma independiente y reproducible en cualquier entorno, lo que facilita el desarrollo y la distribución de la aplicación.