PRÁCTICA DOCKER COMPOSE

Este proyecto consiste en una aplicación web con dos interfaces de usuario, una desarrollada con Angular y la otra con Spring Boot. Además, cuenta con una base de datos MySQL para el almacenamiento de los datos.

Funcionalidad

El backend (desarrollado con Spring Boot) es el encargado de proveer servicios RESTful para la manipulación de los datos almacenados en la base de datos. Los servicios son consumidos por el frontend (desarrollado con Angular) para presentar la información al usuario y permitir la manipulación de los datos.

DOCKERFILE

En el Dockerfile de la base de datos, se establece la imagen base, se crean y asignan las variables de entorno para la base de datos, se copia el archivo de configuración de MySQL al contenedor y se especifica el volumen en el que se almacenarán los datos persistentes de la base de datos.

En el Dockerfile del backend, se establece la imagen base, se copian los archivos del proyecto al contenedor, se establece el directorio de trabajo y se expone el puerto necesario para la comunicación con el frontend. Luego se instalan las dependencias necesarias y se compila el proyecto. Finalmente, se especifica el comando que se ejecutará al iniciar el contenedor.

En el Dockerfile del frontend, se establece la imagen base, se copian los archivos del proyecto al contenedor y se establece el directorio de trabajo. Luego se instalan las dependencias necesarias y se construye la aplicación. Finalmente, se establece el comando que se ejecutará al iniciar el contenedor.

database

FROM mysql:latest: esta línea indica que la imagen base a partir de la cual se creará esta imagen será la imagen más reciente de MySQL que se encuentre disponible en el repositorio de Docker Hub.

ENV MYSQL_ROOT_PASSWORD: esta línea define la contraseña del usuario root de MySQL en la imagen de Docker creada.

ENV MYSQL_DATABASE: esta línea define el nombre de la base de datos que se creará en la imagen de Docker.

ENV MYSQL_USER: esta línea define el nombre de usuario que se utilizará para conectarse a la base de datos.

ENV MYSQL_PASSWORD: esta línea define la contraseña del usuario definido en la línea anterior.

COPY my.cnf /etc/mysql/my.cnf: esta línea copia el archivo my.cnf de la máquina host (en la misma carpeta que el Dockerfile) a la imagen de Docker. Este archivo se utilizará para configurar el servidor MySQL dentro del contenedor.

VOLUME /var/lib/mysql: esta línea indica que se creará un volumen para almacenar los datos de la base de datos. Esto permite que los datos persistan incluso si el contenedor se detiene o se elimina.

EXPOSE 3306: esta línea indica que se expondrá el puerto 3306 del contenedor de Docker. Este es el puerto utilizado por MySQL para aceptar conexiones.

CMD ["mysqld"]: esta línea define el comando que se ejecutará cuando se inicie el contenedor. En este caso, se iniciará el servidor de MySQL.

backend

FROM openjdk:11-jre-slim: Esta línea especifica la imagen base que se va a utilizar para construir la imagen de Docker. En este caso, se está utilizando la imagen slim de OpenJDK 11.

WORKDIR /app: Esta línea establece el directorio de trabajo en el cual se van a copiar los archivos necesarios para la construcción de la imagen.

COPY target/crud-test-back-0.0.1-SNAPSHOT.jar app.jar: Esta línea copia el archivo JAR del proyecto de Spring Boot en el directorio de trabajo especificado anteriormente.

EXPOSE 8080: Esta línea expone el puerto 8080 del contenedor de Docker para que se pueda acceder al proyecto de Spring Boot desde otros contenedores o desde el host.

ENTRYPOINT ["java","-jar","app.jar"]: Esta línea establece el comando de inicio del contenedor de Docker. En este caso, se está especificando que se debe ejecutar el archivo JAR del proyecto de Spring Boot al iniciar el contenedor.

frontend

FROM node:14.17.3-alpine3.14 AS build: esta línea indica que la imagen de Docker se basará en la imagen de Node.js con la versión 14.17.3 y el sistema operativo Alpine 3.14, y se crea una etiqueta de construcción llamada "build".

WORKDIR /app: esta línea establece el directorio de trabajo en el contenedor como /app, que es donde se almacenará y ejecutará el código del frontend.

COPY package*.json ./: esta línea copia los archivos package.json y package-lock.json del directorio local al directorio de trabajo del contenedor. Estos archivos se utilizan para instalar las dependencias de la aplicación.

RUN npm install: esta línea ejecuta el comando npm install en el contenedor, que instala las dependencias de la aplicación.

COPY . .: esta línea copia todo el contenido del directorio local al directorio de trabajo del contenedor.

RUN npm run build: esta línea ejecuta el comando npm run build en el contenedor, que compila el código fuente del frontend y genera los archivos estáticos que se servirán al navegador.

FROM nginx:1.21.1-alpine: esta línea indica que la imagen final de Docker se basará en la imagen de NGINX con la versión 1.21.1 y el sistema operativo Alpine.

COPY --from=build /app/build /usr/share/nginx/html: esta línea copia los archivos estáticos generados en el paso anterior (etiqueta "build") desde el directorio de trabajo del contenedor de compilación al directorio de trabajo del contenedor de NGINX, donde se servirán los archivos.

EXPOSE 80: esta línea indica que el puerto 80 debe estar expuesto para que el contenedor de NGINX pueda recibir solicitudes HTTP.

CMD ["nginx", "-g", "daemon off;"]: esta línea establece el comando de inicio para el contenedor de NGINX, que ejecuta el servidor NGINX en modo demonio y mantiene el contenedor en ejecución.

ARCHIVO DOCKER COMPOSE

Este archivo de Docker Compose define tres servicios:

crudtestdb: Es el servicio de base de datos. Se construye a partir de una imagen creada en el directorio ./crud-test-db. También se especifica un nombre de imagen personalizado. Se crea un contenedor con el nombre de crudtestdb. Se monta un volumen para persistir los datos de la base de datos. Se agrega a la red crudtest y se reinicia automáticamente a menos que se detenga manualmente.

crudtestback: Es el servicio de backend, construido a partir de una imagen creada en el directorio ./spring-crud-example. Se especifica un nombre de imagen personalizado. El servicio se conecta a la red crudtest y se especifica el DATABASE_URL para conectarse a la base de datos. Se reinicia automáticamente a menos que se detenga manualmente.

crudtestfront: Es el servicio de frontend, construido a partir de una imagen creada en el directorio ./angular-15-crud-example. Se especifica un nombre de imagen personalizado. Se

mapea el puerto 8081 del contenedor al puerto 8081 del host. Se agrega a la red crudtest y se reinicia automáticamente a menos que se detenga manualmente.

Al final del archivo se define la red crudtest con el controlador de puente como controlador de red predeterminado. Esta red es utilizada por los tres servicios para comunicarse entre sí.

ARCHIVO DE CONFIGURACIÓN DEL PROXY

Este archivo es una configuración de servidor NGINX. En este caso, se está configurando un servidor NGINX para que escuche en el puerto 8081 y la dirección IP local. La sección location / se encarga de la gestión de las rutas y de enviar el archivo index.html cuando el cliente solicita una URL en la raíz del sitio web. La sección error_page establece la página de error que se mostrará si hay errores en el servidor. La sección location /api configura un proxy para enviar solicitudes a la aplicación backend que se está ejecutando en el puerto 8080 y el contenedor llamado crudtestback. Esta sección es importante porque permite que el servidor NGINX enrute las solicitudes a la aplicación backend.

TAG Y PUSH

el archivo tag_n_push.sh y el archivo manually-creation.sh

se encargar de hacer la creación de las imagenes, taguearlas y hacerles push al repositorio.