Pontificia Universidad Católica Madre y Maestra

CSTI-1880-5597 - Programación Web Avanzada

Tema:

Dockerizar CRUD con respecto a la Práctica 2.

Integrantes:

1014-3611 Starlin Frías Luna

Profesor:

Carlos Camacho

Santiago de los Caballeros, República Dominicana

Lunes 17 de Febrero del 2025

Introducción:

El objetivo de esta práctica es dockerizar un CRUD realizado con Spring Boot en Java. Además, debemos publicar la imagen en Docker Hub para futuros usos o compartir la misma. Este es el enlace con respecto a la practica que se realizara la dockerizacion:https://github.com/FriasLuna0102/Programacion-Web-Avanzada/tree/main/practica-2

Objetivos a realizar:

1. Utilizar una imagen base de Java que ocupe la menor cantidad de espacio.
2. Utilizar el concepto de multi-stage build para la creación de la imagen.
3. Utilizar un volumen donde se guardan los logs de la aplicación.
4. Utilizar variable de ambiente en la definición de la imagen para cambiar el puerto de arranque de la aplicación.
5. Subir la imagen creada a Docker Hub.

Imagen Utilizada:

La imagen que conozco actualmente que ocupa menos espacio es la de alpine, pero sucede que alpine no tiene contiene soporte para jdk 21, y sucede que la aplicación en Java está construida con Java 21, por lo que en su lugar utilice otra bastante pequeña que si me resolvía esta cuestión, que era la de slim (pesa un poco más).

Forma de utilizarlo en el Dockerfile: **FROM openjdk:21-jdk-slim**

Concepto Multi-Stage Build:

Este concepto es bastante útil debido a su funcionalidad para reducir los pesos de las imágenes finales. Este básicamente consiste en construir todo lo necesario que requiere la aplicación como tal para ejecutarse, ya sea compilarse, instalaciones de dependencias, etc... Todo esos pasos necesarios mencionados anteriormente se realizan en imágenes separadas (etapas), donde ya al final se crea la imagen final (que se utilizara), en esta copiamos todo lo esencial para permitir la ejecución final del app.

Uso del Multi-Stage Build:

Para la validacion del gradle compatible con jdk21 consulte esta issue en StackOverFlow, para compilar mi app con jdk21 : https://stackoverflow.com/questions/77140377/starting-from-which-version-does-gradle-support-java-21

Uso en el Dockerfile: **FROM gradle:8.4-jdk21 AS build**

Esta es la forma en la cual realicé este concepto, tengo dos secciones, en la primera compilo la aplicación de java. En esta instaló el gradle, además, copió los archivos necesarios para compilar el proyecto, y otorgó los permisos para su ejecución futura a gradlew. He aqui el codigo que lo realizó:

```
FROM gradle:8.4-jdk21 AS build

WORKDIR /app

COPY . /app

RUN chmod +x gradlew

RUN ./gradlew build
```

Ya para la segunda etapa donde se ha compilado el app, no es necesario contar con gradle:8.4, ya que esta instalación solo se hizo para compilar. Por lo que, en la segunda etapa solo requerimos instalar la imagen SLIM, además, copiamos los archivos generados de la compilación que es el .jar., y ya luego exponemos el puerto, definimos la variable de entorno. También definimos el volumen donde almacenaremos los logs de la app de Java. Por último, ejecutamos el ENTRYPOINT (punto de entrada principal).

```
FROM openjdk:21-jdk-slim

WORKDIR /app

RUN mkdir /app/logs

COPY --from=build /app/build/libs/practica-2-0.0.1-SNAPSHOT.jar
my-app.jar

ENV SERVER_PORT=8080

EXPOSE ${SERVER_PORT}

VOLUME /app/logs

ENTRYPOINT ["sh", "-c", "java -jar my-app.jar
--server.port=${SERVER_PORT}"]
```

Uso de Volumen:

Los volúmenes son muy utilizados para la persistencia de los datos en los contenedores, ya que si se presenta el escenario de que debemos eliminar un contenedor por x o y razón, pues los datos dentro de este serán eliminados siempre y cuando no se cuente con volúmenes.

Para definir el uso de los volúmenes podemos lograrlo con lo siguiente, donde creamos el directorio y definimos donde tendremos el volumen:

```
RUN mkdir /app/logs

VOLUME /app/logs
```

Para poder capturar los logs de la aplicación corriendo en Java, tuve que modificar el application.properties de la misma, donde se define de este modo:

```
logging.file.name=/app/logs/application.log
logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %msg%n
logging.level.root=INFO
logging.level.org.springframework.web=INFO
logging.level.org.example.practica2=DEBUG
logging.level.org.springframework.web.servlet.DispatcherServlet=DE
BUG
```

Donde en la ejecución debemos especificar el mapeo del volumen al directorio, esta es la forma de ejecutar la imagen una vez construida:

Para construir la imagen simplemente ejecutamos: docker build -t <nombreImagen> .

docker run -d -p 8080:8080 -e SERVER_PORT=8080 -v logs:/app/logs my-app-java

Docker Hub:

Para subir la imagen a Docker Hub lo que debemos realizar es lo siguiente:

Instalar dependencias necesarias:

sudo apt-get install pass gnupg2

Generar clave GPG:

gpg --generate-key

Inicializar almacenamiento Docker:

pass init <clave_pública>

Autenticarse en Docker Hub:

docker login -u <usuario>

Etiquetar y subir la imagen:

docker tag <id_imagen> <usuario>/my-app-java:latest

docker push <usuario>/my-app-java:latest

En el repositorio está la documentación más detallada del mismo:

https://github.com/FriasLuna0102/Programacion-Web-Avanzada/tree/main/DockerBasico_Practica