Spring Boot MySQL

# Docker spring boot MySQL example | Spring boot MySQL docker

<https://youtu.be/PAQvxqocb6A>

# Aplicación Spring Boot

El autor usa su propia aplicación. Yo voy a usar la que ya tengo hecha. Está en la carpeta crud. Es una aplicación de Spring Boot, con API REST. Usa una base de datos MySQL también propia de él, pero yo sigo con la que ya tenía hecha para mi aplicación.

## src\main\resources\application.properties

# Debe ser igual al artifactId del pom

spring.application.name=crud

server.port=8080

spring.datasource.driver-class-name =com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/anime

spring.datasource.username=cruduser

spring.datasource.password=crudPassword

# https://stackoverflow.com/questions/50322550/hibernate-dialect-for-mysql-8

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.hibernate.ddl-auto=update

#spring.jpa.show-sql: true

logging.level.org.hibernate.SQL=debug

# Arrancar WampServer

Vamos a armar la base de datos en WampServer, porque es muy simple. Antes de arrancar WampServer necesitamos asegurarnos de que el puerto 3306 (el de MySQL) no está en uso. En la consola de Windows:

for /f "tokens=5" %a in ('netstat -aon ^| find /i "LISTENING" ^| find ":3306"') do taskkill /f /pid %a

Una vez en WampServer, arrancar phpMyAdmin.

# \40-docker\crud\sql\anime.sql

En este archivo está el script SQL para crear la base de datos y el usuario. Copiar y pegar todo en el panel SQL de phpMyAdmin. Una vez ejecutado, tenemos la base de datos y el usuario.

# Ejecutar en la PC local desde el IDE con MySQL en WampServer

<https://youtu.be/PAQvxqocb6A?t=38>

Si uno tiene corriendo WampServer y ejecuta la aplicación desde el IDE, se ve que todo funciona correctamente. Hay que asegurarse de que el puerto 3306 no está siendo usado por un container, por ejemplo.

<https://youtu.be/PAQvxqocb6A?t=73>

En la URL

<http://localhost:8080/all>

se puede ver la lista de todos los animes cargados en la base de datos. Esto es lo que hemos hecho muchas veces, es conocido y no tiene nada de sorprendente.

Recordar que la aplicación Java se conecta a una base de datos MySQL, y por eso necesitamos que el motor de la base de datos esté corriendo, lo cual hemos hecho hasta ahora usando WampServer.

# Docker Desktop y CLI

Descargar el instalador desde

<https://www.docker.com/products/docker-desktop/>

Se instala una aplicación de Windows que se llama Docker Desktop. Además se instala un comando, que se puede ejecutar desde la consola de Windows, escribiendo docker. Este comando es un gestor de containers autosuficiente.

# MySQL: descargar la imagen

<https://youtu.be/PAQvxqocb6A?t=134>

Si no tengo la imagen de MySQL ya descargada en Docker, la tengo que bajar de hub.docker.com. En la consola de Windows:

docker pull mysql

Explicamos este comando[[1]](#footnote-1). La sintaxis es:

docker pull [OPTIONS] NAME[:TAG|@DIGEST]

docker pull: baja a la PC local una imagen o un repositorio desde un registro. Por defecto, descargará las imágenes desde Docker Hub[[2]](#footnote-2)

En este caso no tenemos opciones.

mysql es el nombre de la imagen. El formato general del parámetro NAME es:

PATH\_AL\_REGISTRO:PUERTO/NOMBRE\_DE\_LA\_IMAGEN: TAG|@DIGEST]

Por ejemplo:

docker pull myregistry.local:5000/testing/test-image

descarga la imagen llamada testing/test-image desde un registro local por el puerto 5000. El path del registro es similar a una URL, pero no tiene el protocolo. Docker usa por defecto https:// . Se puede configurar un registro para que acepte conexiones no seguras, pero nosotros no lo vamos a hacer.

# MySQL: crear un container

Ahora que tengo la imagen en mi PC local, tengo que instanciar un container para MySQL. Asegurarse de que el puerto no está en uso, por ejemplo, hay que apagar WampServer:

docker run --name mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql

Ahora tengo MySQL server corriendo en un container, con las credenciales que elegí.

# MySQL Workbench 8.0 CE

Como apagué el Wamp, no puedo usar phpMyAdmin. Descargué MySQL Workbench Community Edition de:

<https://dev.mysql.com/downloads/workbench/>

Lo instalé, y ahora lo ejecuto. Me conecto usando las credenciales del MySQL que está corriendo en el container, y que está listening en el puerto 3306 de mi localhost, porque así lo configuré. Y como esto es lo que asume MySQL Workbench por defecto, no hay problemas.

En el file system del container, MySQL guarda las bases de datos en /var/lib/mysql

# File system del container

El container tiene un file system bastante complejo.

Un modo de explorarlo es desde Docker Desktop. Vamos a la solapa containers. Veremos la lista de los containers que tenemos. En el costado derecho hay varios botones o iconos, que nos permiten arrancar, pausar, reiniciar, detener o borrar cada container. Entre estos botones, los dos de la izquierda nos permiten ver el container en el browser, o bien en una terminal. Justamente en la terminal tipo Linux podremos ver el file system.

Visual Studio Code tiene una extensión para Docker. Si la tenemos instalada, nos permite examinar todo con gran detalle. Es muy intuitiva y poderosa. Podemos comprobar que las bases de datos están donde dijimos antes.

# \40-docker\crud\sql\anime.sql

En este archivo está el script SQL para crear la base de datos y el usuario. La base de datos que creamos antes con este mismo script está en el sistema de archivos de WampServer, en el lugar donde MySQL guarda las bases de datos. El MySQL que está ahora corriendo en el container está limpio, no tiene ninguna base de datos. Copiar y pegar todo en el panel Query 1 del Workbench. Una vez ejecutado, tenemos la base de datos y el usuario, todo en el container.

# Ejecutar en la PC local desde el IDE con MySQL en container

En la URL

<http://localhost:8080/all>

se puede ver la lista de todos los animes cargados en la base de datos. La aplicación está ahora conectada a la base de datos que está en el container de MySQL.

# Dockerizar la aplicación Spring Boot

Ya tenemos dockerizado el server de MySQL. Ahora tenemos que dockerizar la aplicación de Spring Boot.

# Dokerfile

Docker puede crear imágenes automáticamente leyendo las instrucciones de un Dockerfile[[3]](#footnote-3). Un Dockerfile es un documento de texto que contiene todos los comandos que un usuario podría llamar en la línea de comandos para ensamblar una imagen. Usando docker build se puede crear una construcción automatizada que ejecuta varias instrucciones de la línea de comandos en sucesión. Para eso, se usa el comando docker build. Pero para poder usarlo, antes hay que escribir el Dockerfile.

Vamos a la carpeta de la aplicación, 40-docker\crud, creamos un archivo de texto y lo llamamos Dockerfile, sin extensión. En ese archivo pegamos lo siguiente:

# eclipse-temurin soporta versiones recientes del JDK.

# En muchos tutoriales usan openjdk, pero la documentación

# oficial dice que está deprecado.

FROM eclipse-temurin

ADD target/\*.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]

La instrucción FROM[[4]](#footnote-4) inicializa una nueva etapa de la construcción y establece la imagen base para las instrucciones posteriores. En este caso, lo que estamos diciendo es que queremos arrancar la construcción con eclipse-temurin[[5]](#footnote-5) que es una alternativa a openjdk.

La instrucción ADD[[6]](#footnote-6) copia nuevos archivos, directorios o direcciones URL de archivos remotos desde <src> y los agrega al sistema de archivos de la imagen en la ruta <dest>. En este caso, le estamos diciendo que copie todos los jar que hay en la carpeta target

Para construir la imagen de la aplicación Spring Boot:

docker build -t crud .

Para instanciar un container a partir de esta imagen:

docker run -p 8080:8080 --name crud -e MYSQL\_HOST=mysql -e MYSQL\_USER=root -e MYSQL\_PASSWORD=my-secret-pw -e MYSQL\_PORT=3306 crud

docker run -p 8080:8080 --name crud -e MYSQL\_HOST=mysql -e MYSQL\_USER=cruduser -e MYSQL\_PASSWORD=crudPassword -e MYSQL\_PORT=3306 crud

Antes de ejecutar este comando, hay que asegurarse de que el puerto 8080 está libre. Entre otras cosas, si la aplicación está corriendo en el IDE, hay que apagarla. Si nos olvidamos, y nos da un error, el container de todos modos fue construido.

Puede dar el error de que la app fue compilada con una versión de Java posterior a la del container. Hay que recompilar la app.

# Docker compose

El formato del archivo docker-compose.yml está definido en:

<https://github.com/compose-spec/compose-spec/blob/master/spec.md#compose-file>

# Version top-level element

<https://github.com/compose-spec/compose-spec/blob/master/spec.md#version-top-level-element>

A pesar de que muchos tutoriales y ejemplos ponen este parámetro, está deprecado. Yo no lo voy a usar.

## docker-compose.yml

services:

  mysql:

    image: mysql

    environment:

      - MYSQL\_ROOT\_PASSWORD=mysql\_root\_password

      - MYSQL\_PASSWORD=crudPassword

      - MYSQL\_DATABASE=anime

    ports:

      - "3306:3306"

  app:

    image: crud

    environment:

      - SPRING\_DATASOURCE\_URL=jdbc:mysql://mysql:3306/anime

      - SPRING\_DATASOURCE\_USERNAME=cruduser

      - SPRING\_DATASOURCE\_PASSWORD=crudPassword

    ports:

      - "8080:8080"

    build:

      context: ./

      dockerfile: Dockerfile

    depends\_on:

      - mysql

Este archivo se ejecuta con docker-compose up.

Notar que se ocupa de buildear la imagen de la aplicación crud, si no existe.

1. <https://docs.docker.com/engine/reference/commandline/pull/> [↑](#footnote-ref-1)
2. <https://hub.docker.com/> [↑](#footnote-ref-2)
3. <https://docs.docker.com/engine/reference/builder/> [↑](#footnote-ref-3)
4. <https://docs.docker.com/engine/reference/builder/#from> [↑](#footnote-ref-4)
5. <https://hub.docker.com/_/eclipse-temurin> [↑](#footnote-ref-5)
6. <https://docs.docker.com/engine/reference/builder/#add> [↑](#footnote-ref-6)