Spring Boot MySQL

# Docker spring boot MySQL example | Spring boot MySQL docker

<https://youtu.be/PAQvxqocb6A>

# 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

# Crear la base de datos y el usuario

Una vez en WampServer, arrancar phpMyAdmin. En el archivo

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

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.

# Aplicación Spring Boot

El autor usa su propia aplicación. Yo voy a usar la que ya tengo hecha. Está en la carpeta crud. Abrirla con VS Code. 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.

# Credenciales de la aplicación para la base de datos

Las credenciales y otros parámetros de la configuración necesarios para que la aplicación se conecte a la base de datos están en el archivo:

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

# 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 ejemplo por un container.

<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.

En el resto de este tutorial, siempre que queremos usar un comando docker, antes debemos asegurarnos de que esté corriendo Docker Desktop.

# Imágenes en Docker

En la consola de Windows:

docker images

# 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 general 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).

[OPTIONS] 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.

Una vez que termina de descargar la imagen, podemos comprobar en la extensión de Visual Studio Code, en la solapa Images. Deberíamos ver que tiene el tag latest.

# 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 -d --name mysqlCnt -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -v "path-absoluto-a-una-carpeta-local":/var/lib/mysql mysql

docker run: primero crea una capa de contenedor grabable sobre la imagen especificada y luego la inicia usando el comando especificado.

-d: ejecutar el container en background (detach)

--name mysqlCnt: asigna el nombre mysqlCnt al container

-p 3306:3306: publica un puerto del container al host puerto\_local:puerto\_en\_el\_container

-e MYSQL\_ROOT\_PASSWORD=my-secret-pw: establece la variable de entorno

-v "path-absoluto-a-una-carpeta-local":/var/lib/mysql crea un volumen, es decir que la base de datos va a estar guardada en una carpeta local y no se va a perder si borramos el container.

mysql: nombre de la imagen a ejecutar.

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

Podemos comprobar en la extensión de Visual Studio Code, en la solapa Containers.

Si voy a <http://localhost:3306/> en realidad no voy a ver nada. El server está listening, pero esa URL no le está mandando ningún request. Para eso necesitaría el phpMyAdmin, pero solo lo tengo en el WampServer, así que no puedo.

# MySQL Workbench 8.0 CE

Como apagué el WampServer, 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.

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.

Otro 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.

# \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, refrescamos el panel de bases de datos, y vemos que ya tenemos la base de datos y el usuario, todo en el container.

Notar que el container que creamos hace un momento tiene ahora una base de datos, y un usuario que no es el root, con una password diferente.

Notar que estas credenciales deben ser exactamente iguales a las que están en el archivo application.properties, que es de donde las tomará el jar cuando vayamos a ejecutarlo.

Este script solo tendremos que ejecutarlo la primera vez que arranquemos el container. Como ese container tiene asociado un volumen local, la base de datos que se crea queda grabada en esa carpeta local. Si borramos el container la base de datos no se pierde. Al crear o reiniciar el container, los datos están.

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

Arranco la aplicación desde el IDE, como antes. 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. Lo que ahora sucede es que en el puerto 8080 está listening la aplicación Java. A esa aplicación Java le mando un request. La aplicación Java lo recibe, y como tiene un controlador REST le manda a su vez un request al puerto 3306, que es donde está listening el servicio MySQL. Y por eso me llega la respuesta. Notar que ahora no me vino desde el MySQL que está dentro de WampServer, sino desde el MySQL que está corriendo en el container que arranqué desde Docker.

# Detener el IDE

Detener la aplicación que está corriendo en el IDE. Comprobar que la URL <http://localhost:8080/all> no tiene la aplicación.

# Crear network en Docker

Cada container que se crea como una instancia de una imagen corre aislado, solo se ve a sí mismo. Para que dos containers puedan verse entre sí, necesitamos tener una network, y que ambos containers estén en ella.

docker network create crudNet

Crea una network y le llama crudNet.

docker network connect crudNet mysqlCnt

Conecta a la network antes creada el container mysqlCnt, que hasta ahora estaba aislado.

# Dokerfile para construir la imagen

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

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 archivos de tipo jar que hay en la carpeta target (<src>, relativo a la carpeta en la que está el archivo Dockerfile) en la carpeta app.jar (<dest> es el path en el file system de la imagen).

Para construir la imagen de la aplicación Spring Boot, arrancar una consola de Windows en la carpeta en la que está el Dockerfile (que en este caso debería ser 40-docker\crud) y pegar el comando:

docker build -t crud .

El comando docker build[[7]](#footnote-7) construye una imagen Docker según las instrucciones del archivo Dockerfile. El parámetro -t sirve para darle un nombre a la imagen. El dot que está al final es el path, en este caso relativo a la carpeta en la que está físicamente el archivo Dockerfile. Ese path determina cuál es el contexto sobre el que va a trabajar Docker.

En Visual Studio Code vamos a la extensión de Docker, solapa Images, y vemos que hay una imagen llamada crud, con el tag latest.

# Instanciar un container a partir de la imagen

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. En ese caso, habría que borrarlo y recomenzar.

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.

Para instanciar un container a partir de esta imagen y conectarlo a la network:

docker run --name crudCnt --net crudNet -p 8080:8080 -e MYSQL\_HOST=mysqlCnt -e MYSQL\_USER=cruduser -e MYSQL\_PASSWORD=crudPassword -e MYSQL\_PORT=3306 crud

docker run: primero crea una capa de contenedor grabable sobre la imagen especificada y luego la inicia usando el comando especificado.

--name crudCnt: asigna el nombre crudCnt al container.

--net crudNet: conecta este container a la network que antes creamos.

-p 8080:8080: publica un puerto del container al host.

-e MYSQL\_HOST=mysqlCnt: establece la variable de entorno .

-d: ejecutar el container en background (detach).

crud: nombre de la imagen a ejecutar.

Notar que las credenciales que le damos ahora son solo para la base de datos de la aplicación.

# Ejecutar todo en container

Esta vez la aplicación arranca automáticamente. 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. Lo que ahora sucede es que en el puerto 8080 está listening la aplicación Java. A esa aplicación Java le mando un request. La aplicación lo recibe, y como tiene un controlador REST le manda a su vez un request al puerto 3306, que es donde está listening el servicio MySQL. Y por eso me llega la respuesta. Notar que ahora no me vino desde el MySQL que está dentro de WampServer, sino desde el MySQL que está corriendo en el container que arranqué desde Docker.

# Differences Between Docker Compose and Dockerfile by Example

Hasta ahora lo que hemos visto es cómo se usa Dockerfile para construir imágenes de Docker, cómo se crea una network y cómo se conectan dos containers a la misma network para que se vean entre sí.

Vamos a dejar el proyecto anterior en pausa para estudiar un tema nuevo. Vamos a seguir el tutorial:

<https://youtu.be/JmyAMcKUNYA>

May 19, 2022, 11 minutos

Cameron McKenzie. Muy clara la explicación.

En pocas palabras:

* Dockerfile es solo para construir la imagen, pero no para correr containers.
* docker-compose up es solo para correr containers, pero no para construir imágenes.

Si docker-compose necesita una imagen que no tiene disponible, va a tratar de buscarla en el registro default. Si la encuentra la descarga. Si no la encuentra da un error fatal.

# Imagen y container de nginx

En la consola de Windows pegamos:

docker pull nginx

Esto nos descarga desde el registro default la imagen nginx:latest. Arrancamos un container con esta imagen, por ejemplo desde la extensión de VS Code. En la solapa Containers de la extensión de VS Code veremos que aparece un container basado en la imagen nginx:latest, con un nombre aleatorio, porque no le dimos uno. Navegamos por el file system de este container, hasta la carpeta \usr\share\nginx\html\. Veremos que hay dos archivos: index.html y 50x.html. Si abrimos este container en el browser, veremos justamente el contenido de index.html. Si nosotros ponemos en esta carpeta todo nuestro sitio web, lo veremos en el browser.

# Dockerfile con nginx y website

Vamos a la carpeta MYWEBSITE. En esa carpeta creamos un archivo Dockerfile y en ese archivo pegamos las siguientes líneas:

FROM nginx:latest

COPY ./website /usr/share/nginx/html/

Como ya vimos, la primera línea establece la imagen que se tomará como base para todo lo que sigue.

La segunda instrucción[[8]](#footnote-8) copia archivos o directorios desde un origen a un destino. En este caso, copia toda la carpeta website (que contiene el web site) al lugar que antes discutimos dentro del file system del container. Tenemos que reconocer que el website es francamente humilde: tiene un archivo html, un js y un css. Pero es suficiente para mostrar lo que queremos.

Abrimos una terminal de Windows en la carpeta MYWEBSITE y pegamos el comando:

docker build -t my-nginx-image .

Como ya hemos visto, este comando construirá una imagen siguiendo las instrucciones del archivo Dockerfile, cuyo contexto será el directorio en el que estamos, y le pondrá como nombre my-nginx-image. Podemos verficar esto en VS Code. Si no lo vemos en Docker Desktop, tenemos que ir a Troubleshoot y reniniciar el daemon.

Ya tenemos la imagen construida, pero todavía no hay ningún container.

# Archivo compose.yaml

Este es el nombre preferido, según la documentación oficial. El formato del archivo compose.yaml está definido en:

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

La sintaxis[[9]](#footnote-9) es simple. Lo único que vamos a decir por ahora es que la indentación es muy importante, porque se requiere para dar estructura al contenido. Solo se usan espacios, no tabulaciones, y cada nivel está indentado exactamente dos espacios.

# Version top-level element

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

Muchos tutoriales y ejemplos ponen al principio del archivo el parámetro version. Pero la documentación oficial dice que está deprecado. Yo no lo voy a usar.

# Docker compose

En lugar de correr los containers con docker run, vamos a usar Docker Compose. Para eso, navegamos a la carpeta MYWEBSITE. En esa carpeta creamos un archivo compose.yaml y en ese archivo pegamos las siguientes líneas:

services:

  my-nginx-service:

    container\_name: my-website

    image: my-nginx-image:latest

    cpus: 1.5

    mem\_limit: 2048m

    ports:

      - "8080:80"

Este archivo establece cómo queremos ejecutar el container a partir de la imagen.

La primera línea dice que vamos a correr servicios. Un servicio es un server corriendo en un puerto, algo que podemos consumir con un cliente.

La segunda línea dice que nuestro servicio se va a llamar my-nginx-service.

Este servicio va a correr en un container que se va a llamar my-website.

El container va a ser una instancia de la imagen my-nginx-image:latest, que es la que acabamos de crear en el paso anterior.

Queremos que tenga 1.5 cpus y 2 gigabytes de memoria.

Finalmente mapeamos el puerto 8080 de nuestro localhost al puerto 80 interno del container, que es el puerto default de nginx.

# Correr un container con Docker Compose

En la consola de Windows pegamos el comando:

docker-compose up

Si vamos a <http://localhost:8080/> veremos nuestro sitio web.

# Ahora varios servicios

Para empezar apagamos todos los containers anteriores. Inmediatamente comprobaremos que el sitio no se ve más en el browser.

Ahora vamos a crear un container con varios servicios distintos, corriendo en distintos puertos del host. Como este container está corriendo en mi PC local, el host será mi localhost. Si estuviera en la nube, el host tendría su correspondiente URL.

Abrimos en VS Code la carpeta MYOTHERWEBSITE. Creamos un archivo compose.yaml y pegamos lo siguiente:

services:

  apache-service:

    container\_name: myotherwebsite

    image: httpd:latest

    cpus: 1.5

    mem\_limit: 2048m

    ports:

      - "80:80"

    volumes:

      - ./website:/usr/local/apache2/htdocs

Ejecutamos el comando docker-compose up. Vemos que hay una imagen que antes no estaba, que es hpptd:latest. Se descargó automáticamente del repositorio default. Vemos que hay un nuevo container, que antes no estaba, que se llama myotherwebsite. Si vamos al file system de este container, y navegamos hasta :/usr/local/apache2/htdocs, veremos que allí están copiados los tres archivos de la carpeta website, aunque nosotros no lo hicimos explícitamente. Lo que sucede es que la primera vez que se levana el container se crea el volumen, para lo cual se copia la carpeta del origen al destino. El destino es, obviamente, el punto de arranque default de httpd.

Probamos esto con la URL <http://localhost/>, que usa el puerto 80, y ahora dirá en el título MYOTHERWEBSITE, simplemente para que estemos seguros de que no nos estamos confundiendo con lo que hicimos en los pasos anteriores.

Vamoa a agregar un segundo servicio al archivo compose.yaml:

  nginx-service:

    container\_name: nginx-app

    image: nginx:latest

    cpus: 2.5

    mem\_limit: 4096m

    ports:

      - "99:80"

    volumes:

      - ./nginx-app:/usr/share/nginx/html

Este nuevo servicio se llamará nginx-service, y correrá en un container que se va a llamar nginx-app. Se armará a partir de la imagen nginx:latest, que habíamos bajado antes, pero la borramos después de usarla como base para construir la imagen my-nginx-image. Tampoco puedo usar my-nginx-image, porque si bien a esta la construí a partir de nginx, le copié el otro sitio web al punto de inicio, así que no me va a servir. Por eso arranco de nuevo, y le armo el volumen con el sitio nuevo, que es el que tengo ahora. Vamos a armar el volumen copiando de la carpeta local nginx-app a la carpeta /usr/share/nginx/html del file system del container. Ejecutamos el comando docker-compose up y revisamos. Si ahora comprobamos en la URL <http://localhost:99/>, vemos que allí está nuestra aplicación. En el file system del container navegamos hasta /usr/share/nginx/html, y allí vemos que se copió el archivo index.html.

# Desplegar un war en un container Docker

Seguimos las instrucciones de la documentación oficial:

<https://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/>

The example app has been packaged as a war file and can be downloaded [here](https://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/sample.war) (Note: make sure your browser doesn't change file extension or append a new one).

Descargamos el archivo sample.war y lo guardamos en la carpeta samplewar.

The easiest way to run this application is simply to move the war file to your **CATALINA\_HOME/webapps** directory. Tomcat will automatically expand and deploy the application for you.

Agregamos las siguientes líneas a compose.yaml:

  tomcat-service:

    container\_name: sample-war

    image: tomcat:latest

    cpus: 3.5

    mem\_limit: 4096m

    ports:

      - "8080:8080"

    volumes:

      - ./samplewar:/usr/local/tomcat/webapps

Como ya sabemos, **CATALINA\_HOME** es la carpeta donde está instalado Tomcat dentro del container Docker. En este caso, /usr/local/tomcat.

Cuando ejecutamos el comando docker-compose up vemos que el war fue expandido en la carpeta local samplewar y se creó una nueva carpeta sample, en la cual ahora podemos ver una estructura de directorios que antes no estaban. Si exploramos el file system del container recién creado, veremos que en la carpeta destino designada ahora está copiado no solo el war, sino la carpeta sample con todo su contenido.

You can view it with the following URL (assuming that you're running tomcat on port 8080 as is the default):

<http://localhost:8080/sample>

# Desplegar un jar en un container Docker

Seguimos los mismos pasos que en la primera parte de este tutorial. Dentro de la carpeta java hay una aplicación Spring Boot muy simple. Creamos el archivo Dockerfile y pegamos el siguiente texto:

FROM eclipse-temurin

ADD target/\*.jar app.jar

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

Generamos la imagen de esta aplicación con el comando:

docker build -t java-jar-image .

Comprobamos que tenemos la imagen java-jar-image en VS Code.

Agregamos a compose.yaml el texto:

  tomcat-service2:

    container\_name: sample-jar

    image: java-jar-image:latest

    cpus: 3.5

    mem\_limit: 4096m

    ports:

      - "8081:8080"

    volumes:

      - ./samplewar:/usr/local/tomcat/webapps

Ejecutamos el comando docker-compose up. Vemos que ahora en la categoría services tenemos no uno, sino tres servicios. Cada uno de ellos corre en su propio container.

<http://localhost/>

<http://localhost:99/>

<http://localhost:8080/sample>

<http://localhost:8081/>

# Volvemos al principio

Detenemos todos los containers, cerramos todas las carpetas, navegamos hasta \40-docker\crud, o sea donde estábamos al principio. Creamos un archivo compose.yaml y le pegamos este texto:

## compose.yaml

services:

  mysql:

    image: mysql

    environment:

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

      - MYSQL\_PASSWORD=crudPassword

      - MYSQL\_DATABASE=anime

    ports:

      - "3306:3306"

    volumes:

      - "./sql/anime.sql:/docker-entrypoint-initdb.d/1.sql"

  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.

El primer servicio se llama mysql-service

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)
7. <https://docs.docker.com/engine/reference/commandline/build/> [↑](#footnote-ref-7)
8. <https://docs.docker.com/engine/reference/builder/#copy> [↑](#footnote-ref-8)
9. <https://en.wikipedia.org/wiki/YAML#Syntax> [↑](#footnote-ref-9)