Dockerizar Spring Boot MySQL

Volver a hacerlo pero con una aplicación que no sea crud

Me voy a basar en estos tutoriales:

**Dockerize Spring Boot Application with MySQL**

<https://youtu.be/S2s28PCg4M4>

May 24, 2022, 16 minutos.

**Dockerize Spring Boot Application with MySQL using Docker compose**

<https://youtu.be/6hMHziv0T2Y>

May 27, 2022, 11 minutos.

Funciona bien todo.

Voy a hacer muchos cambios. Realmente lo que quiero es tener un apunte para las materias.

# Dos opciones para dockerizar

Hay dos opciones para dockerizar una aplicación de Spring Boot con MySQL:

* Crear una network de containers Docker.
* Usando docker-compose.

Nosotros vamos a usar docker-compose en este documento.

# Crear una network de containers Docker

En la extensión para Docker de VS Code, borrar todos los containers, imágenes, volúmenes y networks. Esto no es estrictamente necesario, pero trabajar con un espacio limpio ayuda a pensar con claridad.

## PASO 1 Descargar la imagen de mysql

Descargar la imagen de mysql desde docker hub:

docker pull mysql

Al terminar la descarga, comprobamos que tenemos la imagen mysql:latest.

## PASO 2 Crear una network

Crear una network para que los containers se puedan comunicar entre sí. Es un bridge entre dos servicios:

docker network create springboot-mysql-net

Comprobamos que tenemos la network de nombre springboot-mysql-net, pero todavía no tiene ningún container asociado.

## PASO 3 Correr un container para MySQL

Correr un container para MySQL en la network:

docker run -d --name mysqldb -p 3306:3306 --net springboot-mysql-net -e MYSQL\_ROOT\_PASSWORD=1234 -e MYSQL\_DATABASE=anime -e MYSQL\_USER=cruduser -e MYSQL\_PASSWORD=1234 mysql

Comprobamos que ahora tenemos corriendo el container llamado mysqldb. Vemos que tiene asociado un volumen, aunque no le pusimos nada explícitamente. Y vemos que está asociado a la network correspondiente.

Si vamos comprobar los volúmenes, vemos que ahora tenemos uno asociado con nuetro container.

Si vamos a las networks, comprobamos que tenemos la network que creamos. Si nos dice que no tiene containers asociados, refrescamos la vista, y aparecerá el correspondiente container.

## PASO 4 Verificar la base de datos

Verificar que se ha creado la base de datos. En la terminal de Windows ejecutamos el comando:

docker ps

Nos fijamos en los primeros tres caracteres de la ID de nuestro container. En este caso eran f76:

docker exec -it f76 bash

Aparece una ventana de bash:

mysql -ucruduser -p1234

Estamos en la consola de MySQL:

show databases;

Vemos que tenemos anime más las otras propias de la instalación de MySQL:

use anime;

Estamos en la base de datos:

show tables;

Vemos que la base de datos está vacía.

Para salir de la consola de MySQL:

exit

Estamos en bash. Para salir de bash:

exit

Estamos de nuevo en la terminal de Windows.

## PASO 5 application.properties

Actualizar el archivo application.properties:

spring.datasource.url=jdbc:mysql://mysqldb:3306/anime

spring.datasource.username=cruduser

spring.datasource.password=1234

## PASO 6 Imagen de la aplicación Spring Boot

Construir la imagen de la aplicación de Spring Boot. Antes de compilar, ir al paquete test y comentar todos los tests. Esto es porque si algún test falla la compilación no será exitosa. Recompilar la aplicación. En la terminal de VS Code:

mvn clean install

Deberíamos tener un jar recién creado en target.

En el directorio crud crear un archivo Dockerfile y pegar:

# eclipse-temurin soporta versiones recientes del JDK.

# En muchos tutoriales usan openjdk, pero la documentación

# oficial dice que está deprecado.

FROM eclipse-temurin

COPY target/\*.jar app.jar

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

Construir la imagen de la aplicación Spring Boot:

docker build -t springbootmysql .

Comprobamos en la extensión de Docker que apareció esta nueva imagen. No hay nuevos containers, ni tampoco volúmenes, ni se agregó nada a la network. Solo tenemos la imagen nueva.

## PASO 7 Container de Spring Boot

Arrancar el container de Spring Boot en la misma network:

docker run --network springboot-mysql-net --name springboot-container -p 8080:8080 -d springbootmysql

Vemos que apareció un nuevo container llamado springboot-container, instancia de la imagen springbootmysql, que está en la network springboot-mysql-net.

Vamos a las networks, refrescamos, y vemos que nuestra red ahora tiene dos containers.

No hay volúmenes nuevos.

Repetimos los pasos de antes para ir a la consola de MySQL y vemos que ahora tenemos la tabla anime creada, pero está vacía.

Pegamos en la consola de MySQL el contenido del archivo anime.sql.

Vemos que ahora se han creado las dos tablas, y que tienen registros.

Si ahora vamos a la URL <http://localhost:8080/all> tenemos la lista de los registros.

## PASO 8 Se pierden los datos

Detener ambos containers. Repetir los pasos 3 y 7 para arrancarlos ambos nuevamente.

Si ahora vamos a la URL <http://localhost:8080/all> veremos que en lugar de la lista de los registros solo obtenemos una lista vacía. La aplicación funciona bien, pero los datos que habíamos cargado en la aplicación se perdieron.

## PASO 9 Correr un container con volumen para MySQL

Detener y borrar ambos containers y el volumen asociado. Correr un container con volumen para MySQL en la network:

docker run -d --name mysqldb -p 3306:3306 --net springboot-mysql-net -e MYSQL\_ROOT\_PASSWORD=1234 -e MYSQL\_DATABASE=anime -e MYSQL\_USER=cruduser -e MYSQL\_PASSWORD=1234 -v db:/var/lib/mysql mysql

Comprobamos que ahora tenemos corriendo el container llamado mysqldb. Vemos que tiene asociado un volumen, aunque no le pusimos nada explícitamente. Y vemos que está asociado a la network correspondiente.

Si vamos comprobar los volúmenes, vemos que ahora tenemos uno asociado con nuetro container.

Si vamos a las networks, comprobamos que tenemos la network que creamos. Si nos dice que no tiene containers asociados, refrescamos la vista, y aparecerá el correspondiente container.

## PASO 10 Verificar la base de datos

Verificar que se ha creado la base de datos. En la terminal de Windows ejecutamos el comando:

docker ps

Nos fijamos en los primeros tres caracteres de la ID de nuestro container. En este caso eran f76:

docker exec -it f76 bash

Aparece una ventana de bash:

mysql -ucruduser -p1234

Estamos en la consola de MySQL:

show databases;

Vemos que tenemos anime más las otras propias de la instalación de MySQL:

use anime;

Estamos en la base de datos:

show tables;

Vemos que la base de datos no tiene tablas. Pegar en la consola de MySQL el contenido del archivo anime.sql.

Para ver cómo quedó:

show tables;

Para salir de la consola de MySQL:

exit

Estamos en bash. Para salir de bash:

exit

Estamos de nuevo en la terminal de Windows.

## PASO 11 Container de Spring Boot

Arrancar el container de Spring Boot en la misma network:

docker run --network springboot-mysql-net --name springboot-container -p 8080:8080 -d springbootmysql

Vemos que apareció un nuevo container llamado springboot-container, instancia de la imagen springbootmysql, que está en la network springboot-mysql-net.

Vamos a las networks, refrescamos, y vemos que nuestra red ahora tiene dos containers.

No hay volúmenes nuevos.

Repetimos los pasos de antes para ir a la consola de MySQL y vemos que ahora tenemos la tabla anime creada, pero está vacía.

Pegamos en la consola de MySQL el contenido del archivo anime.sql.

Vemos que ahora se han creado las dos tablas, y que tienen registros.

Si ahora vamos a la URL <http://localhost:8080/all> tenemos la lista de los registros.

## PASO 12 Se conservan los datos

Detener ambos containers. No borrar el volumen db. Repetir los pasos 3 y 7 para arrancarlos ambos nuevamente.

Si ahora vamos a la URL <http://localhost:8080/all> veremos la lista de los registros. La aplicación funciona bien, y los datos que habíamos cargado en la aplicación se conservaron.

# Usando Docker Compose

Ahora vamos a dockerizar la misma aplicación, pero usando Docker Compose.

## PASO 1 application.properties

Actualizar el archivo application.properties:

spring.datasource.url=jdbc:mysql://mysqldb:3306/anime

spring.datasource.username=cruduser

spring.datasource.password=1234

Notar que el puerto 3306 es el default. Este jar va a intentar conectarse con un servidor de MySQL que esté listening en el puerto 3306. Pero esto va a correr en una network de containers. Por lo tanto, lo que tengo que poner es el puerto interno del container de MySQL, no el puerto local publicado por ese container.

## PASO 2 Imagen de la aplicación Spring Boot

Para no dejar cabos sueltos, vamos a borrar los containers, los volúmenes, las networks y las imágenes. Vamos al directorio target y borramos los jar. Arrancamos de nuevo con todo limpio. Recompilar la aplicación. En la terminal de VS Code:

mvn clean install

Verificamos que tenemos un jar recién creado en target, con el puerto 3307.

## PASO 3 compose.yaml

Muchos tutoriales usan docker-compose.yml, pero la documentación oficial dice que se debe preferir compose.yaml. De modo que esto es lo que yo voy a usar.

Para ver la versión de Docker que tengo instalada, en la consola de Windows:

docker -v

El tutorial usa la propiedad version en la primera línea, pero la documentación oficial[[1]](#footnote-1) dice que está deprecada, de modo que yo no la uso.

Pegar el siguiente texto:

services:

  mysqldb:

    environment:

      MYSQL\_DATABASE: anime

      MYSQL\_USER: cruduser

      MYSQL\_PASSWORD: 1234

      MYSQL\_ROOT\_PASSWORD: 4321

    image: mysql

    ports:

      - 3307:3306

    restart: always

  server:

    build: .

    restart: always

    ports:

      - 8080:8080

    depends\_on:

      - mysqldb

El container en el que va a correr la aplicación Spring Boot se llama server. Podemos poner cualquier nombre, pero esta es una elección razonable, ya que estamos hablando de una API que es como el backend.

La opción build está pidiendo que construyamos la imagen Docker, a partir del jar que acabamos de compilar en el paso anterior. Eso lo haremos siguiendo las instrucciones de Dockerfile.

Después exponemos los ports. Lo que estamos haciendo es mapear el puerto local 8080 al puerto interno del container 8080.

Finalmente, decimos que este containder depends on mysqlbd, que es el servicio de MySQL. Todavía no existe, de modo que el comando docker-compose primero lo levanta, y una vez que está corriendo levanta el container del server.

Notar que el puerto local de MySQL es 3306, o sea el default. Es el que va a usar el jar, porque están en una network.

Ejecutamos el comando:

docker-compose up

Si vamos a la URL <http://localhost:8080/all> veremos que la aplicación está funcionando. Esto es porque el jar usa el puerto 3306 y el container mysqldb usa el puerto interno 3306.

### El puerto publicado por el container

Si intentamos conectarnos a este container mysqkdb que está corriento con una aplicación que esté fuera de la network de Docker, tendremos que usar el puerto 3307, que es el que publica el container. No el interno, 3306, que es el usamos dentro de la network.

# Volúmenes de Docker frente a montajes de enlace

## Fuentes

Usé los siguientes artículos, y algunos otros. No pongo las citas puntuales, porque cambié, edité, mezclé, y recorté:

<https://blog.logrocket.com/docker-volumes-vs-bind-mounts/>

<https://www.freecodecamp.org/news/docker-mount-volume-guide-how-to-mount-a-local-directory/>

<https://maximorlov.com/docker-compose-syntax-volume-or-bind-mount/>

<https://www.baeldung.com/ops/docker-volumes>

## Introducción

Los contenedores Docker se utilizan para ejecutar aplicaciones en un entorno aislado. De forma predeterminada, todos los cambios dentro del contenedor se pierden cuando el contenedor se detiene. Pero Docker proporciona volúmenes y bind mounts, que son dos mecanismos para conservar datos en su contenedor de Docker. Vamos a ver cómo vincular directorios locales a su contenedor Docker y usar volúmenes administrados por Docker alternativamente. Saber ambos le permite usar contenedores Docker para muchos más casos de uso que pueden aumentar su productividad.

## Bidireccional

Vincular un directorio es una sincronización bidireccional. Cada archivo que cambia en el host se cambia en el contenedor y cada archivo que se cambia en el contenedor se cambia en el host. Entonces, si detiene e inicia la base de datos, puede montar el mismo directorio y su configuración y los datos almacenados estarán disponibles.

Las ventajas de este método son que es fácil de usar y de fácil acceso. Debe usar directorios locales vinculados para los archivos que desea cambiar u observar en el host, como archivos de configuración y archivos de registro.

## Montajes de enlace

Los montajes de enlace montan en su contenedor un archivo o directorio de su máquina host, al que luego puede hacer referencia a través de su ruta absoluta. Un mount bind de Docker es una conexión de alto rendimiento desde el contenedor a un directorio en la máquina host. Permite que el host comparta su propio sistema de archivos con el contenedor, que puede ser de solo lectura o de lectura y escritura. Esto nos permite usar un contenedor para ejecutar herramientas que no queremos instalar en nuestro host y aún así trabajar con los archivos de nuestro host. Por ejemplo, si quisiéramos usar una versión personalizada de bash para un script en particular, podríamos ejecutar ese script en un contenedor bash, montado en nuestro directorio de trabajo actual.

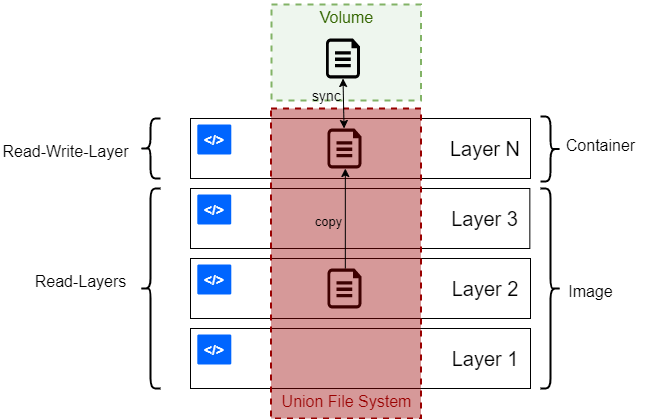
Para usar montajes de enlace, no es necesario que el archivo o directorio ya exista en su host Docker. Si no existe, se creará bajo demanda. Los montajes de enlace dependen de que el sistema de archivos de la máquina host tenga una estructura de directorio específica disponible. Debe crear explícitamente una ruta al archivo o carpeta para colocar el almacenamiento.

**IMPORTANTE: los montajes de enlace dan acceso a archivos confidenciales. De acuerdo con los documentos de Docker, puede cambiar el sistema de archivos del host a través de procesos que se ejecutan en un contenedor. Esto incluye la creación, modificación y eliminación de archivos y directorios del sistema, lo que puede tener implicaciones de seguridad bastante graves. Incluso podría afectar los procesos que no son de Docker.**

## Volúmenes Docker

Los volúmenes son un excelente mecanismo para agregar una capa persistente de datos en sus contenedores Docker, especialmente para una situación en la que necesita conservar los datos después de cerrar sus contenedores. Docker maneja completamente los volúmenes de Docker y, por lo tanto, son independientes tanto de la estructura de directorios como del sistema operativo de la máquina host. Cuando usa un volumen, se crea un nuevo directorio dentro del directorio de almacenamiento de Docker en la máquina host, y Docker administra el contenido de ese directorio.

## El sistema de archivos Docker

Un contenedor docker ejecuta la pila de software definida en una imagen. Las imágenes están formadas por un conjunto de capas de solo lectura que funcionan en un sistema de archivos llamado Union File System. Cuando iniciamos un nuevo contenedor, Docker agrega una capa de lectura y escritura en la parte superior de las capas de la imagen, lo que permite que el contenedor se ejecute como si fuera un sistema de archivos estándar de Linux. Por lo tanto, cualquier cambio de archivo dentro del contenedor crea una copia de trabajo en la capa de lectura y escritura. **Sin embargo, cuando el contenedor se detiene o elimina, esa capa de lectura y escritura se pierde.**

Podemos probar esto ejecutando un comando que escribe y luego lee un archivo:

docker run bash:latest bash -c "echo hello > file.txt && cat file.txt"

hello

Pero si ejecutamos la misma imagen con solo el comando para mostrar el contenido del archivo:

docker run bash:latest bash -c "cat file.txt"

cat: can't open 'file.txt': No such file or directory

La segunda ejecución del contenedor se ejecuta en un sistema de archivos limpio, por lo que no se encuentra el archivo.

## Beneficios de usar volúmenes

En los volúmenes de Docker, el almacenamiento no está acoplado al ciclo de vida del contenedor, sino que existe fuera de él. Esto tiene muchos beneficios. Por un lado, puede matar su contenedor tantas veces como quiera y aún así conservar sus datos. También es fácil reutilizar el almacenamiento en varios contenedores; por ejemplo, un contenedor escribe en el almacenamiento mientras que otro lee de él.

Dado que los volúmenes no están vinculados a ningún contenedor, puede adjuntarlos fácilmente a varios contenedores en ejecución al mismo tiempo. También encontrará que los volúmenes no aumentan el tamaño del contenedor Docker que los usa. Por último, puede usar la CLI de Docker para administrar volúmenes, por ejemplo, recuperar la lista de volúmenes o eliminar volúmenes no utilizados.

## Uso de montajes de enlace

Para usar montajes de enlace en un contenedor, tiene dos opciones: --mount y -v. La diferencia más notable entre las dos opciones es que --mount es más detallado y explícito, mientras que -v es más una forma abreviada de --mount. Combina todas las opciones que pasa a --mount en un campo.

En la superficie, ambos comandos hacen lo mismo. Sin embargo, hay algunos escenarios en los que la diferencia entre usar --mount y -v será notablemente diferente. Por ejemplo, es una buena práctica usar --mount cuando trabaja con servicios porque necesitará especificar más opciones de las que son posibles con -v.

## Montar directorios locales con la opción -v

Eso se puede hacer usando el comando docker run -v.

El comando docker run primero crea una capa de contenedor grabable sobre la imagen especificada y luego comienza a usar el comando especificado. (Fuente docker.com)

El uso del parámetro -v le permite vincular un directorio local.

-v o --volume le permite montar directorios y archivos locales en su contenedor. Por ejemplo, puede iniciar una base de datos MySQL y montar el directorio de datos para almacenar los datos reales en su directorio montado.

# En Linux bash

# run mysql container in the background

docker run --name mysql-db -v $(pwd)/datadir:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:8.0.28-debian

# show content of data directory

ls -la datadir

# stop mysql container

docker rm -f mysql-db

# Lo mismo pero en Windows cmd

# run mysql container in the background

docker run --name mysql-db -v %CD%/datadir:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:8.0.28-debian

# show content of data directory

dir /og datadir

# stop mysql container

docker rm -f mysql-db

El primer comando arranca un container llamado mysql-db, y monta un volumen en la carpeta datadir relativa a la carpeta en la que estamos en ese momento. Esto se hace con $(pwd) si estamos en bash, o con %CD% si estamos en la consola de Windows. Essta carpeta que está físicamente en el file system de nuestra computadora local, se asocia con /var/lib/mysql, que es el lugar dentro del file system del container donde MySQL guarda las bases de datos. La password del root se establece a my-secret-pw y el tag de mysql es 8.0.28-debian.

Otro ejemplo: ejecute en Linux el comando:

docker run --rm --name postgres-db -e POSTGRES\_PASSWORD=password --v "$pwd":/var/lib/postgresql/data -p 2000:5432 -d postgres

o en Windows:

docker run --name postgres-db -v %CD%/postgresdb:/var/lib/postgresql/data -e POSTGRES\_PASSWORD=password -p 2000:5432 -d postgres

Este comando arranca un container con el nombre postgres-db, crea un bind mount a la carpeta local llamada postgresdb que se encuentra dentro del directorio actual, y la vincula con la carpeta default que usa PostgreSQL dentro del file system del container, establece la password del root, expone el puerto 2000 del localhost y lo vincula con el puerto 5432 interno del container, el container estará dettachado, y la imagen a usar es postgres.

Otro ejemplo: ejecute en Linux el comando:

docker run -v $(pwd):/var/opt/project bash:latest \ bash -c "echo Hello > /var/opt/project/file.txt"

o en Windows:

docker run -v %CD%:/var/opt/project bash:latest bash -c "echo Hello > /var/opt/project/file.txt"

Este comando inicia un container sin nombre, a partir de la imagen bash:latest. Define un bind mount entre el directorio de trabajo del file system local y un path absoluto dentro del file system del container. En ese directorio crea y graba un archivo llamado file.txt. Como la sincronización es bidireccional, ese archivo del container se copiará en la carpeta local que es el otro extremo del bind mount. En este caso pusimos como source el directorio de trabajo, que es la carpeta donde está bash o la consola de Windows. Si revisamos esa carpeta, veremos el archivo file.txt.

## Cómo utilizar los volúmenes de Docker para conservar los cambios

En lugar de vincular su directorio local, puede usar volúmenes de Docker. Un volumen de Docker es un directorio en algún lugar de su directorio de almacenamiento de Docker y se puede montar en uno o varios contenedores. Están completamente administrados y no dependen de ciertas características específicas del sistema operativo. Cuando uno instala Docker en su Pc local, esta aplicación hace muchas cosas. Una de ellas es que arma su propio sistema de archivos, que es independiente del file system de nuestra computadora. Docker maneja todas estas cosas de forma independiente. Uno puede mostrarlas e inspeccionarlas por medio de Docket Desktop, del comando docker, o de la extensión Docker de Visual Studio Code. Pero no es aconsejable modificar nada. Hay que dejar que Docker haga su trabajo. Con los directorios locales vinculados a un volumen es distinto. Son realmente directorios nuestros, y podemos trabajar con ellos como siempre hacemos.

Vamos a crear un volumen de Docker y montarlo para conservar los datos de MySQL:

# LINUX

# create volume

docker volume create mysql-data

# run mysql container in the background

docker run --name mysql-db -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:latest

# stop mysql container

docker rm -f mysql-db

:: WINDOWS

:: create volume

docker volume create mysql-data

:: run mysql container in the background

docker run --name mysql-db -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:latest

:: stop mysql container

docker rm -f mysql-db

Antes de eliminar el volumen de Docker, puede abrir su GUI de Docker e inspeccionar el volumen haciendo clic primero sobre el volumen, y una vez en él, clic en la pestaña de datos.

Se ven los archivos, pero están aislados en un volumen de Docker. Se recomienda usarlos para archivos persistentes que no necesita observar o cambiar desde su sistema host. Se sabe que este método tiene un mejor rendimiento que los enlaces de directorios locales.

# LINUX

# remove volume

docker volume remove mysql-data

:: WINDOWS

:: remove volume

docker volume remove mysql-data

## Sintaxis de Docker Compose: ¿volumen o enlace de montaje?

En compose.yaml, la clave de volúmenes puede aparecer en dos lugares diferentes:

services:

database:

# ...

volumes: # Nested key. Configures volumes for a particular service.

volumes: # Top-level key. Declares volumes which can be referenced from multiple services.

# ...

Vamos a hablar sobre la clave de volúmenes anidada. Ahí es donde configura los volúmenes para un servicio o contenedor específico, como una base de datos o un servidor web. Esta configuración tiene un formato de sintaxis corto y uno largo.

## Formato de sintaxis corta y sus variaciones

La configuración del volumen tiene un formato de sintaxis breve que se define como:

[SOURCE:]TARGET[:MODE]

SOURCE puede ser un volumen con nombre o una ruta (relativa o absoluta) en el sistema host.

TARGET es una ruta absoluta en el contenedor.

MODE es una opción de montaje que puede ser de solo lectura o de lectura y escritura. Los corchetes significan que el argumento es opcional.

Esta opcionalidad conduce a tres variaciones únicas que puede usar para configurar los volúmenes de un contenedor. Docker Compose es inteligente al reconocer qué variedad se usa y si usar un volumen o un montaje de enlace.

### ****No hay SOURCE****

/var/lib/postgresql/data

Cuando solo se especifica un TARGET, sin un SOURCE, Docker Compose creará un directorio anónimo y lo montará como un volumen en la ruta de destino dentro del contenedor.

La ruta del directorio en el sistema host y dentro del file system propio de Docker es por defecto

/var/lib/docker/volumes/<uuid>/\_data

donde <uuid> es un ID aleatorio asignado al volumen como su nombre.

### SOURCE sin un path

postgresql-data:/var/lib/postgresql/data

Si SOURCE está presente y no tiene una ruta, entonces Docker Compose asume que te estás refiriendo a un volumen con nombre. Este volumen debe declararse en el mismo archivo en la declaración de clave de volúmenes de nivel superior. Recordemos que, como dijimos al principio, estamos hablando sobre la clave de volúmenes anidada, no sobre la de top level.

La clave de volúmenes de nivel superior siempre declara volúmenes, nunca montajes vinculados. Los montajes vinculados no tienen nombre y no pueden ser nombrados.

### SOURCE con un path

/some/content:/usr/share/nginx/html

o

./public:/usr/share/nginx/html

Si SOURCE es una ruta, absoluta o relativa, Docker Compose enlazará la carpeta en el contenedor. Rutas relativas que comienzan con . o .. son relativos a la ubicación de compose.yaml.

Se desaconsejan los montajes de enlace para los contenedores de bases de datos, ya que los hace menos portátiles. Los montajes de enlace son específicos del sistema host y Docker no los administra. Los documentos oficiales tienen una sección sobre volúmenes frente a monturas vinculantes que explica las diferencias entre los dos con más detalle.

# Volúmenes en Docker

Voy a seguir este tutorial:

**Docker Tutorial For Beginners**

<https://youtu.be/SlV5qb1m5og>

May 23, 2022, 45 minutos.

Excelente. El tipo tiene un fuerte acento, pero se le entiende bien. En su perfil de LinkedIn dice que es irlandés.

# Docker Compose en detalle

## Docker Crash Course #11 - Docker Compose

<https://youtu.be/TSySwrQcevM>

## Docker Crash Course #12 - Dockerizing a React App

<https://youtu.be/QePBbG5MoKk>

## Docker Crash Course #13 - Sharing Images on Docker Hub

<https://youtu.be/YS35VHsbS-0>

compose01.yaml

los siguientes todavía no los miré, más allá de haberlos elegido por la primera impresión

Dockerize Spring Boot Application with MySQL using Docker compose

<https://youtu.be/6hMHziv0T2Y>

docker compose example | spring boot & mysql docker-compose

<https://youtu.be/-ekBqIvAGY4>

Differences Between Docker Compose and Dockerfile by Example

<https://youtu.be/JmyAMcKUNYA>

DOCKER desde ZERO: Siguiente nivel con Docker-Compose

<https://youtu.be/yWgL0GSbFuo>

Docker compose tutorial for beginners by example [all you need to know]

<https://youtu.be/4EqysCR3mjo>

What is Docker Compose | How to create docker compose file | How to use Compose

<https://youtu.be/HUpIoF_conA>

Spring Boot with MySQL Docker deployment using DOCKER COMPOSE

<https://youtu.be/J2YhWG994Iw>

Introducción a Docker Compose

<https://youtu.be/i-45V0ojtlI>

Using Docker Compose with Mysql And Node

<https://youtu.be/9-iLqe-E9iI>

Docker Compose vs Dockerfile - Dockerfile Explained - Docker Tutorial

<https://youtu.be/Z44UJUXsOGA>

Docker Compose in 12 Minutes

<https://youtu.be/Qw9zlE3t8Ko>

Docker Compose Tutorial - Docker in Practice || Docker Tutorial 9

<https://youtu.be/MVIcrmeV_6c>

Docker Compose Tutorial

<https://youtu.be/HG6yIjZapSA>

los siguientes todavía no los miré, más allá de haberlos elegido por la primera impresión

# How do Docker Volumes work? | Docker made easy #4

<https://medium.com/techmormo/how-do-docker-volumes-enable-persistence-for-containers-docker-made-easy-4-2093a1783b87>

# Top Tips and Use Cases for Managing Your Volumes

<https://www.docker.com/blog/top-tips-and-use-cases-for-managing-your-volumes/>

# Everything you need to know about docker volumes - Docker tips and tricks

<https://cloudnweb.dev/2020/10/everything-you-need-to-know-about-docker-volume/>

# Learn Docker, from the beginning, part II

<https://softchris.github.io/pages/docker-two.html#resources>

# The Complete Guide to Docker Volumes

<https://towardsdatascience.com/the-complete-guide-to-docker-volumes-1a06051d2cce>

1. <https://docs.docker.com/compose/compose-file/#compose-file> [↑](#footnote-ref-1)