Práctica 5. Creación de un contenedor Docker con MySQL Server

Apuntes de BD para DAW, DAM y ASIR

José Juan Sánchez Hernández

Curso 2023/2024

Índice

Índice de figuras

Índice de cuadros

1 Creación de un contenedor Docker con MySQL Server

1.1 Requisitos

Para poder ejecutar contenedores Docker es necesario tener instalado Docker Community Edition (CE) en nuestro equipo.

En la web oficial encontrará la información necesaria para realizar la instalación de Docker CE sobre Windows, macOS, Ubuntu, Debian, Fedora y CentOS.

1.2 Cómo crear un contenedor sin persistencia de datos

Un contenedor Docker que no tiene persistencia de datos quiere decir que cuando finalice la ejecución perderá todo el contenido que hayamos creado durante la ejecución. Es decir, si durante la ejecución del contenedor hemos creado varias bases de datos en MySQL Server, éstas se perderán cuando el contenedor se detenga.

El comando que podríamos usar para lanzar nuestro contenedor Docker con MySQL Server sin persistencia de datos podría ser el siguiente:

```
1  docker run -d --rm --name mysql -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 mysql
       :8.0
```

- `docker run` es el comando que nos permite crear un contenedor a partir de una imagen Docker.
- El parámetro `-d` nos permite ejecutar el contenedor en modo *detached*, es decir, ejecutándose en segundo plano.
- El parámetro `--rm` hace que cuando salgamos del contenedor, éste se elimine y no ocupe espacio en nuestro disco.
- El parámetro `--name` nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre Docker nos asignará un nombre automáticamente.
- El parámetro `e` es para pasarle al contenedor una variable de entorno. En este caso le estamos pasando la variable de entorno `MYSQL_ROOT_PASSWORD` con el valor de la contraseña que tendrá el usuario `root` para MySQL Server.
- El parámetro `-p` nos permite mapear los puertos entre nuestra máquina local y el contenedor. En este caso, estamos mapeando el puerto `3306` de nuestra máquina local con el puerto `3306` del contenedor.

- `mysql:8.0` es el nombre de la imagen y la versión que vamos a utilizar para crear el contenedor. Si no se indica lo contrario buscará las imágenes en el repositorio oficial Docker Hub.

1.3 Cómo cambiar el *plugin* de autenticación en las versiones de MySQL Server superiores a la 8.0

A partir de la versión 8.0 de MySQL Server, el *plugin* de autenticación que se utiliza por defecto es `caching_sha2_password`, mientras que en las versiones inferiores o iguales a la 5.7 se utiliza `mysql_native_password`.

El *plugin* de autenticación `caching_sha2_password` no está soportado por las versiones de PHP anteriores a la 7.4, por lo tanto, si queremos conectar con MySQL Server 8.0 desde una apliación web que utilice una versión de PHP inferior a la 7.4, tendremos que utilizar el *plugin* de autenticación `mysql_native_password` .

El comando que podríamos usar para lanzar nuestro contenedor Docker con MySQL Server 8.0 sin persistencia de datos y con el *plugin* de autenticación `mysql_native_password` podría ser el siguiente:

```
1 docker run -d --rm --name mysql -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 mysql
      :8 --default-authentication-plugin=mysql_native_password
```

Observe que en este caso la imagen que hemos escogido ha sido `mysql:8` y después del nombre de la imagen hemos utilizado el comando:

```
1 --default-authentication-plugin=mysql_native_password
```

para indicar cuál será el método de autenticación que queremos utilizar.

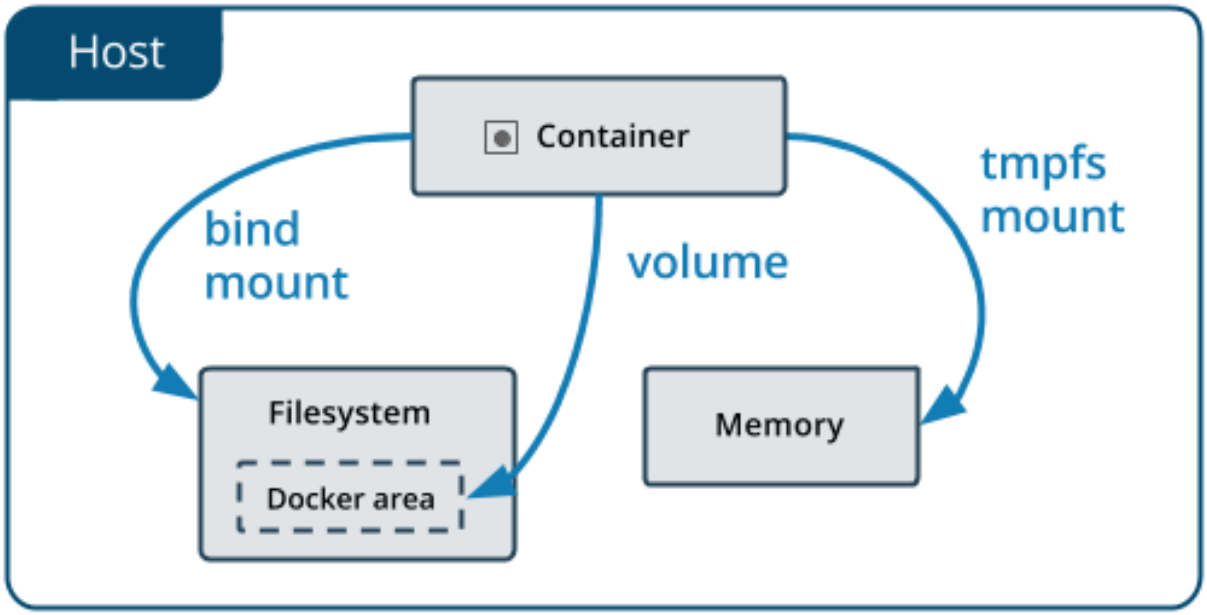
1.4 Cómo crear un contenedor con persistencia de datos

Si queremos que los datos del contenedor sean persistentes tenemos que crear un **volumen** donde vamos a indicar el directorio de nuestra máquina local vamos a almacenar el directorio `/var/lib/mysql`, que es el directorio que utiliza MySQL Server para almacenar las bases de datos.

Para crear un volumen utilizamos el parámetro `-v`.

Docker nos ofrece dos posibilidades para implementar persistencia de datos en los contenedores:

- `bind mount`: pueden estar almacenados en cualquier directorio del sistema de archivos de la máquina host. Estos archivos pueden ser consultados o modificados por otros procesos de la máquina host o incluso por otros contenedores Docker.
- `volume`: se almacenan en la máquina host dentro del área del sistema de archivos que gestiona Docker. Otros procesos de la máquina host no deberían modificar estos archivos, sólo deberían ser modificados por contenedores Docker.



En la documentación oficial podemos encontrar más información sobre el uso de **volumenes** en Docker.

Ejemplo de uso del parámetro `-v` para crear un volumen de tipo `bind_mount`:

```
-v /home/josejuan/data:/var/lib/mysql
```

En este caso el directorio `/home/josejuan/data` de nuestra máquina local estará sincronizado con el directorio `/var/lib/mysql` del contenedor con MySQL Server.

Podemos hacer uso de la variable de entorno `$PWD` para indicar que queremos crear el volumen en nuestro directorio actual.

Ejemplo de uso del parámetro `-v` para crear un volumen de tipo `bind_mount` con la variable de entorno `$PWD`:

```
-v "$PWD":/var/lib/mysql
```

Ejemplo de uso del parámetro `-v` con un volumen de tipo `volume`:

```
-v mysql_data:/var/lib/mysql
```

El comando que podríamos usar para lanzar nuestro contenedor Docker con MySQL Server con persistencia de datos en un volumen podría ser el siguiente:

```
docker run -d --rm --name mysql -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -v
    mysql_data:/var/lib/mysql mysql:8.0
```

- `docker run` es el comando que nos permite crear un contenedor a partir de una imagen Docker.
- El parámetro `-d` nos permite ejecutar el contenedor en modo *detached*, es decir, ejecutándose en segundo plano.
- El parámetro `--rm` hace que cuando salgamos del contenedor, éste se elimine y no ocupe espacio en nuestro disco.

- El parámetro `--name` nos permite asignarle un nombre a nuestro contenedor. Si no le asignamos un nombre Docker nos asignará un nombre automáticamente.
- El parámetro `e` es para pasarle al contenedor una variable de entorno. En este caso le estamos pasando la variable de entorno `MYSQL_ROOT_PASSWORD` con el valor de la contraseña que tendrá el usuario `root` para MySQL Server.
- El parámetro `-p` nos permite mapear los puertos entre nuestra máquina local y el contenedor. En este caso, estamos mapeando el puerto `3306` de nuestra máquina local con el puerto `3306` del contenedor.
- El parámetro `-v` nos permite crear un volumen para tener persistencia de datos al finalizar el contenedor.
- `mysql:8.0` es el nombre de la imagen y la versión que vamos a utilizar para crear el contenedor. Si no se indica lo contrario buscará las imágenes en el repositorio oficial Docker Hub.

1.5 Cómo comprobar que el contenedor está en ejecución

Una vez que hemos iniciado el contenedor podemos comprobar que se está ejecutando con el siguiente comando:

```
1 docker ps
```

Deberíamos obtener una salida similar a esta.

```
1 CONTAINER ID        IMAGE               COMMAND                  CREATED
                 STATUS              PORTS                    NAMES
2 3082ce645213        mysql:5.7.22        "docker-entrypoint.…s"   4 seconds ago
          Up 2 seconds        0.0.0.0:3306->3306/tcp   mysql
```

1.6 Cómo conectar con MySQL Server

Una vez que MySQL Server está en ejecución podemos conectarnos con cualquier cliente: MySQL Workbench, PHPMyAdmin, Adminer, etc.

Los datos de conexión serán:

- Host: 127.0.0.1
- Puerto: 3306
- Usuario: root
- Password: root

1.7 Cómo detener el contenedor

Para detener el contenedor en primer lugar tenemos que conocer cuál es su ID. Para obtenerlo podemos hacer uso del comando `docker ps`.

```
docker ps

CONTAINER ID        IMAGE               COMMAND                  CREATED
                  STATUS              PORTS                    NAMES
3082ce645213        mysql:8.0           "docker-entrypoint.…s"   4 seconds ago
           Up 2 seconds        0.0.0.0:3306->3306/tcp   mysql
```

En la primera columna podemos ver cuál es el `CONTAINER ID`. Una vez localizado el identificador ejecutamos el comando `docker stop` y le pasamos como parámetro el identificador del contenedor que queremos detener.

Para el caso anterior deberíamos ejecutar:

```
docker stop 3082ce645213
```

1.8 Ejemplo de `docker-compose.yml` con MySQL Server

Docker Compose es una utilidad que nos permite gestionar varios contenedores Docker de una forma sencilla. Para ello, utiliza un archivo YAML donde se definen y configuran los servicios, los volúmenes y las redes que queremos crear.

Vamos a crear un archivo `docker-compose.yml` para definir un servicio llamado `mysql` a partir de la imagen de MySQL Server 8.0.

```
version: '3'

services:
  mysql:
    image: mysql:8.0
    ports:
      - 3306:3306
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=database
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
  mysql_data:
```

Puede descargar el archivo `docker-compose.yml` de GitHub.

Para iniciar los servicios en segundo plano con la utilidad `docker-compose` tenemos que ejecutar el siguiente comando.

```
docker-compose up -d
```

Para detener los servicios y mantener el volumen que hemos creado podemos ejecutar:

```
docker-compose down
```

Si además de detener los servicios queremos eliminar el volumen que hemos creado tenemos que añadir el parámetro `-v`.

```
docker-compose down -v
```

Para comprobar el estado de los servicios podemos ejecutar:

```
docker-compose ps
```

1.9 Referencias

- Web oficial de MySQL Server
- Imagen de Mysql Server en Docker Hub
- Web oficial de Docker
- Instalación de Docker Community Edition (CE)
- Instalación de Docker Community Edition (CE) en Windows
- Instalación de Docker Community Edition (CE) en macOS
- Instalación de Docker Community Edition (CE) en Ubuntu
- Instalación de Docker Community Edition (CE) en Debian
- Instalación de Docker Community Edition (CE) en Fedora
- Instalación de Docker Community Edition (CE) en CentOS
- Docker Hub
- Uso de *volumes* en Docker
- Docker Compose
- YAML

2 Licencia