Run MySQL 8.0.18 in a Docker Container

(In a Local Development Environment)

Abstract

Running MySQL in a Docker container is a convenient way to add flexibility to your development environment. This allows you to have different MySQL Docker images with different configurations, e.g. different database names, root password, MySQL versions, etc.
You can also spin up several MySQL Docker images at once, varying the port.
Running MySql in a Docker container is an alternative to installing MySql on your Mac with a DMG file or can be used in conjunction with a local install.

Running Dockerized databases in environments other than your local development machine (QA, Prod, Cloud-Native, etc) involves considerations that are outside the scope of this document.

Installation, Configuration, and Execution of MySQL Docker Image.

The official Docker MySQL image can be found at:

https://hub.docker.com/_/mysql

This command will create a MySQL image, and create and start a container running that image.
It will be accessible on port 3306.

docker run --name mysql-poc -e MYSQL_ROOT_PASSWORD=XXXXXX -e MYSQL_DATABASE=bootdb -e MYSQL_USER=user -e MYSQL_PASSWORD=xxxxx -p 3306:3306 mysql:8

NOTE: You can run docker pull to pull the mysql image down, however the command above will pull the image down if it is not found locally.

You can also run MySQL with no password (Although this is not recommended)
docker run --name mysql-poc -e MYSQL_ALLOW_EMPTY_PASSWORD=true -e MYSQL_DATABASE=bootdb -e MYSQL_USER=user -e MYSQL_PASSWORD=xxxxx -p 3306:3306 mysql:8

1. Connect to the MySQL Docker Container and Running MySQL

You can connect to the MySQL container just created
docker exec -it mysql-poc bash
and then run mysql from the root prompt:

```
root@080830b9818b:/# mysql -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 8.0.18 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Or connect to the MySQL container and run the mysql command
docker exec -it mysql-poc mysql -u root -p

You will be prompted for the root password you set when building the image.

2. Importing Data to MySQL

If you are importing data to your MySQL instance via a DDL script and the "LOAD DATA INFILE" command, you will need to set the ***secure_file_priv*** variable.
This variable restricts where data can be imported from.
You can check the current setting of this variable in your Docker container from the mysql prompt:
show variables like "secure_file_priv";

```
mysql> show variables like "secure_file_priv";
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| secure_file_priv | NULL  |
+------------------+-------+
1 row in set (0.00 sec)
```

This variable is set by editing the /etc/mysql/my.cnf file.
The secure_file_priv variable can be set to a directory, however if the secure_file_priv variable is set to double quotes (e.g. ***secure_file_priv=""***), the default data directory is used.
The default data directory can also be set in the my.cnf file.

```
#
# The MySQL  Server configuration file.
#
# For explanations see
# http://dev.mysql.com/doc/mysql/en/server-system-variables.html

[mysqld]
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
datadir         = /var/lib/mysql
secure_file_priv=""
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

# Custom config should go here
!includedir /etc/mysql/conf.d/
```

MySQL will look for files to be ingested in ${datadir}/${databaseName}.
In the example above this would be /var/lib/mysql/bootdb.

2.1 Modifying the secure_file_priv Variable in the my.cnf File.

When attempting to edit the /etc/mysql/my.cnf file in a Docker container you will discover there is no 'vi' editor available.
There are several solutions to modify the secure_file_priv variable in this file.

Once you modify the /etc/mysql/my.cnf file in your Docker container, you must restart the container for it to take effect.

2.1.1 You can create a local my.cnf and and copy it to your running MySQL container:

docker cp my.cnf mysql-poc:/etc/mysql

This will overwrite the existing my.cnf file in the container, so you might want to back it.

2.1.2 You can also install 'vim' in your container; log into the container.

docker exec -it mysql-poc bash. apt-get update; apt-get -y install vim

2.2 Making Files to be Ingested Available to MySQL Container.

Once the secure_file_priv variable is updated, the container needs access to the data files to be ingested.

2.2.1 You can copy your data files to be ingested to the default MySQL data directory on the container:

docker cp npi1000_simplified.csv mysql-poc:/var/lib/mysql/bootdb

When using the default MySQL directory, the data load statement in your DDL file just needs to mention the file being loaded (CSV, etc), and not the fully qualified path.

```
LOAD DATA INFILE 'npi1000_simplified.csv' IGNORE
```

2.2.2 You can also mount a volume when starting Docker.

docker run -d -v /User/${UserName}/mysql_data:/tmp --name mysql-poc -e MYSQL_ROOT_PASSWORD=XXXXXX -e MYSQL_DATABASE=bootdb -e MYSQL_USER=user -e MYSQL_PASSWORD=xxxxx -p 3306:3306 mysql:8

The command above will mount the mysql_data directory under a user's home directory on to /tmp in the Docker container. Then the LOAD DATA INFILE statement can be modified to reference files in /tmp.

2.3 Persistence

Data is persisted in the data directory defined above on the container. If you stop and restart the container in which MySQL is running, your data will still be there.

However, if you destroy your container, your data will be lost.

If you need to persist your data even if the container is destroyed, you will need to create a volume.
Volumes are the preferred way to persist data generated by and used by Docker containers.

docker run -d -v /User/${UserName}/mysql_data:/var/lib/mysql --name mysql-poc -e MYSQL_ROOT_PASSWORD=XXXXXX -e MYSQL_DATABASE=bootdb -e MYSQL_USER=user -e MYSQL_PASSWORD=xxxxx -p 3306:3306 mysql:8

The above command will mount the local mysql_data directory onto the default data directory of the container. The local directory, ~mysql_data, must be empty this first time this command is run.
All the MySQL files from the container, including the MySQL IBD files, will be persisted to the local filesystem.

Look for the "Mounts" section in output of ***docker inspect ${container}*** to see what volumes are mounted on a running container.

WARNING: If you persist your data to a local filesystem, it will be available when you create and run a new container using the command above. However, if you execute a DDL script in your code that drops and creates your tables, or if you use ***spring.jpa.hibernate.ddl-auto=create*** in your *application.properties* file, your tables will be dropped and recreated. Thus, deleting the data you persisted.
Using ***spring.jpa.hibernate.ddl-auto=update*** is an option this case. It will create your tables only if they do not exist.

2.4 Dockerfile

Instead of configuring your MySQL image on the command line, you can create a Dockerfile to configure the image. The my.cnf copy, root password, database name, and any volumes to be mounted can be included in the Dockerfile.

Example Dockerfile:

```
FROM mysql:8
ENV MYSQL_ROOT_PASSWORD OCA850k$
ENV MYSQL_DATABASE bootdb
COPY ./my.cnf /etc/mysql

#PORT
EXPOSE 3306
```

The image is built from the Dockerfile with the following command: ***docker build -t my-mysql .***
This command creates an image where the my.cnf copy is done, and environment variables are set.
Then a container for this image can be created and run as follows: ***docker run --name my-mysql -p 3306:3306 my-mysql***

Bibliography

ref MySQL 8.0.18 - Installation/Configuration/Application

ref Customize your MySQL Database in Docker

ref Understanding Volumes in Docker

ref **Use volumes**