Bonjour,

J’ai fait un POC sur l’utilisation et le partage d’une base de données « MySql ».

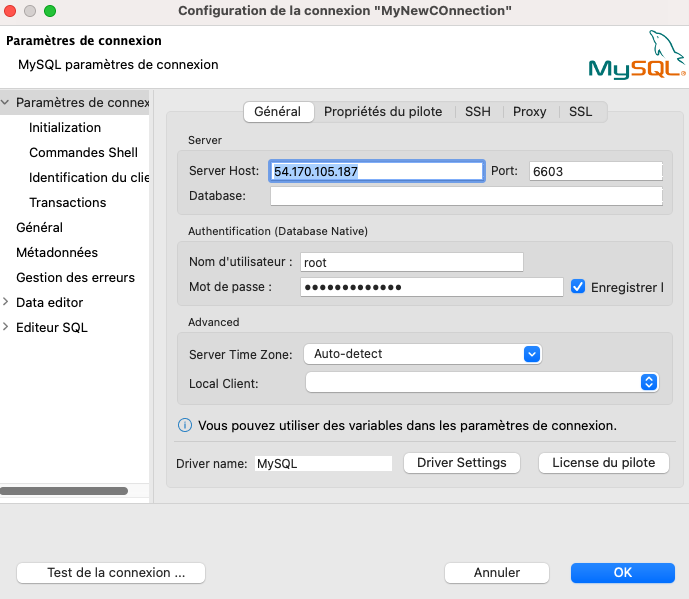
1. **Info image docker « MySql »**

Sur ma VM, j’ai déployé cette image (j’ai simplement suivi les indications du site <https://hevodata.com/learn/docker-mysql/>).

1. **Utilisation de « Dbeaver » (**[**https://dbeaver.io/download/**](https://dbeaver.io/download/)**)**

J’ai eu des soucis avec MySqlWorkbench…

Voici les paramètres de connexion (prérequis : l’image docker qui tourne, je l’arrête à la fin de la journée, dites-moi si vous testez) :



* Host : **54.170.105.187**
* Port : **6603**
* User : **root**
* Mot de passe : **dstmysqlpwd01**

DB\_USER=usr\_sel

DB\_PASSWORD=pwdairline01

DB\_USER=root

DB\_PASSWORD=dstmysqlpwd01

1. **Vérification de la connexion :**

**SELECT** nom, prenom

**FROM** testdbdst.testtabledst;

1. **En local chez moi et depuis la VM :**

Une image contenant texte

Description générée automatiquement

MySQL is a relational database management system based on the Structured Query Language (SQL), the most widely used language for accessing and managing database records. Under the GNU license, MySQL is open-source and free software. Oracle Corporation backs it up.

This blog talks about the 3 steps you can follow to configure and seamlessly set up **Docker MySQL Container** for your business requirements. It also gives a brief introduction to Docker and MySQL before diving into the advantages, installation process, configuration, updating, and management tips for a Docker MySQL Container.

## Table of Contents

* [What is MySQL?](https://hevodata.com/learn/docker-mysql/#t1)
* [What is Docker?](https://hevodata.com/learn/docker-mysql/#t2)
* [What is the Advantage of Running a Docker MySQL Container?](https://hevodata.com/learn/docker-mysql/#t3)
* [How to Install the Docker MySQL Container?](https://hevodata.com/learn/docker-mysql/#t4)
  + [Step 1: Pull the Docker Image for MySQL](https://hevodata.com/learn/docker-mysql/#s1)
  + [Step 2: Deploy and Start the MySQL Container](https://hevodata.com/learn/docker-mysql/#s2)
  + [Step 3: Connect with the Docker MySQL Container](https://hevodata.com/learn/docker-mysql/#s3)
* [How to Configure your Docker MySQL Container?](https://hevodata.com/learn/docker-mysql/#t5)
* [Bonus Setup: How to Connect Host Machine with Docker MySQL?](https://hevodata.com/learn/docker-mysql/#t6)
* [Managing your Docker MySQL Container](https://hevodata.com/learn/docker-mysql/#t7)
  + [How to Start the Container for MySQL?](https://hevodata.com/learn/docker-mysql/#m1)
  + [How to Restart the Container for MySQL?](https://hevodata.com/learn/docker-mysql/#m2)
  + [How to Stop the Container for MySQL?](https://hevodata.com/learn/docker-mysql/#m3)
* [How to Delete your Docker MySQL Container?](https://hevodata.com/learn/docker-mysql/#t8)
* [How to Upgrade your Dockerised MySQL Container?](https://hevodata.com/learn/docker-mysql/#t9)
* [Cautions](https://hevodata.com/learn/docker-mysql/#1)
* [Conclusion](https://hevodata.com/learn/docker-mysql/#t10)

## What is MySQL?

[](https://lh6.googleusercontent.com/dQshvOYbQEGTJQtyakuuEovum346UhSu1nqVGrUhTkHcijCmEM3eRFPpIqU_wFZ86VmmBF320fW000dqPBTaqkqMNs-MW4pvcbWDGE7xRTj_EQESdqFVl5s1D9caSN58x34_9JLN)

[Image Source](https://fr.wikipedia.org/wiki/MySQL)

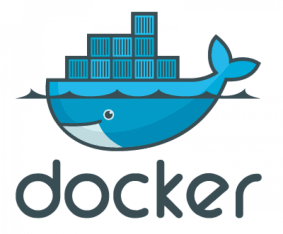
MySQL was used by more than **39% of developers** in the world back in 2019, making it the world’s most popular Database. It certainly lacks the extensive features that are provided by PostgreSQL, but it is still useful for a large number of use cases like **Web Applications**.

Since it comes standard in the LAMP stack, where LAMP stack is an Open-Source suite of web applications that consists of **Linux, Apache HTTP Server, MySQL, PHP**; MySQL is the go-to choice for scalable web applications. Let’s talk about a few salient features that make MySQL such a great catch. Also, see how you can integrate from[MySQL to PostgreSQL](https://hevodata.com/learn/mysql-to-postgresql/).

MySQL is a lightweight database that can be installed and used by developers on production application servers with large multi-tier applications. Leveraging MySQL as a part of your workflow has several advantages:

* Features like [Master-Slave Replication](https://hevodata.com/learn/mysql-master-slave-replication/), Scale-Out are supported by MySQL.
* Offload Reporting, Geographic Data Distribution is also supported by MySQL.
* There’s a very low overhead with the MyISAM Storage Engine when used for read-only applications.
* For frequently used tables, MySQL provides support for the Memory Storage Engine.
* For repeatedly used statements there exists a Query Cache.
* MySQL is easy to learn and troubleshoot given a wide number of helpful sources like blogs, white papers, and books on the subject.
* MySQL is a highly flexible and scalable Database Management System.

## What is Docker?

[](https://res.cloudinary.com/hevo/image/upload/f_auto,q_auto/v1641529586/hevo-learn/homepage-docker-logo.png?_i=AA)

[Image Source](https://www.docker.com/company/newsroom/media-resources/)

Docker aims to make software development predictable and efficient. Docker achieves this by taking away mundane, tedious configuration tasks throughout the development lifecycle for easy, fast, and portable application development – Cloud and Desktop. Docker’s extensive end-to-end platform consists of **CLIs, UIs, and APIs**along with robust security measures engineered to work together across the entire application delivery lifecycle.

Docker Containers are a standardized unit of software that lets developers isolate their app from the environment, therefore, solving the localized operation problem. Docker has grown to become the go-to standard for sharing and building containerized apps for virtually millions of developers- from the Cloud to Desktops. With Docker, you can install from a single package to get up and running in no time flat. Therefore, you can test and code locally with Docker while ensuring consistency between production and development.

## What is the Advantage of Running a Docker MySQL Container?

By running a Docker MySQL Container you are developing code with repeatable configuration and infrastructure, thus speeding up the development process considerably. A key point to note here is that Docker Images are generally small, to begin with. Apart from this, you get speedy delivery and shorter deployment for new application containers. Another key benefit of running a Docker MySQL Container is straightforward maintenance. Whenever you containerize an application, it gets isolated from other apps running in the same system. So, since applications don’t intermix, it makes application maintenance that much easier.

The MySQL Operator for Kubernetes effectively teaches Kubernetes what it requires to run MySQL correctly. It educates Kubernetes on how to treat MySQL Clusters as a first-party resource type in Kubernetes’ API. It also helps Kubernetes understand how many nodes you might need to have a minimal HA cluster within your Kubernetes environment.

## Prerequisites

* An updated Docker Installation.
* Access to a Terminal Window/ Command-Line.
* A User Account with either access to the root account or sudo privileges.

## ****Simplify your MySQL ETL with Hevo’s No-code Data Pipeline****

A fully managed No-code Data Pipeline platform like[Hevo](https://hevodata.com/) helps you integrate data from [100+ data sources](https://hevodata.com/integrations/) (**including 40+ Free Data Sources**) like **MySQL** to a destination of your choice in real-time in an effortless manner. Hevo with its minimal learning curve can be set up in just a few minutes allowing the users to load data without having to compromise performance. Its strong integration with umpteenth sources provides users with the flexibility to bring in data of different kinds, in a smooth fashion without having to code a single line.

[**GET STARTED WITH HEVO FOR FREE**](https://hevodata.com/signup?step=email)

**Check Out Some of the Cool Features of Hevo:**

* **Completely Automated:** The Hevo platform can be set up in just a few minutes and requires minimal maintenance.
* **Real-Time Data Transfer:** Hevo provides real-time data migration, so you can have analysis-ready data always.
* **Transformations**: Hevo provides preload transformations through Python code. It also allows you to run transformation code for each event in the Data Pipelines you set up. You need to edit the event object’s properties received in the transform method as a parameter to carry out the transformation. Hevo also offers drag and drop transformations like Date and Control Functions, JSON, and Event Manipulation to name a few. These can be configured and tested before putting them to use.
* **Connectors**: Hevo supports 100+ Integrations to SaaS platforms, files, databases, analytics, and BI tools. It supports various destinations including Amazon Redshift, Firebolt, [Snowflake Data Warehouses](https://hevodata.com/blog/snowflake-data-warehouse/); Databricks, Amazon S3 Data Lakes; **MySQL**, SQL Server, TokuDB, DynamoDB, PostgreSQL databases to name a few.
* **100% Complete & Accurate Data Transfer:** Hevo’s robust infrastructure ensures reliable data transfer with zero data loss.
* **Scalable Infrastructure:** Hevo has in-built integrations for 100+ sources, that can help you scale your data infrastructure as required.
* **24/7 Live Support:** The Hevo team is available round the clock to extend exceptional support to you through chat, email, and support calls.
* **Schema Management:** Hevo takes away the tedious task of schema management & automatically detects the schema of incoming data and maps it to the destination schema.
* **Live Monitoring:** Hevo allows you to monitor the data flow so you can check where your data is at a particular point in time.

Simplify your Data Analysis with Hevo today!

[**SIGN UP HERE FOR A 14-DAY FREE TRIAL!**](https://hevodata.com/signup?step=email)

## How to Install the Docker MySQL Container?

Here are the steps you can follow to install the Dockerhub MySQL Container:

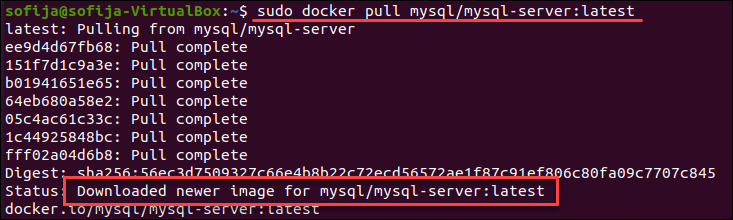
* [Step 1: Pull the Docker Image for MySQL](https://hevodata.com/learn/docker-mysql/#s1)
* [Step 2: Deploy and Start the MySQL Container](https://hevodata.com/learn/docker-mysql/#s2)
* [Step 3: Connect with the Docker MySQL Container](https://hevodata.com/learn/docker-mysql/#s3)

### Step 1: Pull the Docker Image for MySQL

* Begin by taking the appropriate Docker Image for MySQL. You can either download a specific version or choose the latest release as depicted in the following code snippet:

sudo docker pull mysql/mysql-server:latest

* However, if you want a specific version of MySQL, you need to amend the latest value with the version number as shown in the figure below:

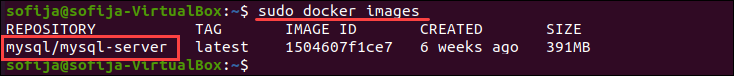
[](https://lh6.googleusercontent.com/ke_3x71rrW_osel6K0R5ipG3yigB4ZW-W8oUqzCCu-Uc1USNfU0odR59gt16WAk0APJRPjdVma47mxkztsY1NR8thTmggGAotvACph_0t5xddUPfU8n0PHvq-K7UDI5kxEf5boJl)

[Image Source](https://phoenixnap.com/)

* Finally, you need to verify that the image is now stored locally by listing the downloaded Docker images:

sudo docker images

* The output should contain mysql/mysql-server as a constituent of the listed images:

[](https://lh6.googleusercontent.com/kjytwGpSWZVfyy6TtxIcIlY6uHlZpJ_panpg7ONAZb4btnhnt9eQtr4ZhJ5aplu2wclGbb-8GhgODyDRaxn7NXGWyhTy9XfgLrP6SBN5Wk8cto4re_hmu6ElNTPA0ObnbifQNIgq)

[Image Source](https://phoenixnap.com/)

### Step 2: Deploy and Start the MySQL Container

* Now that you have extracted the image from the Docker repository, you can move on to deploying a new MySQL Container with the following code snippet:

sudo docker run --name=[container\_name] -d [image\_tag\_name]

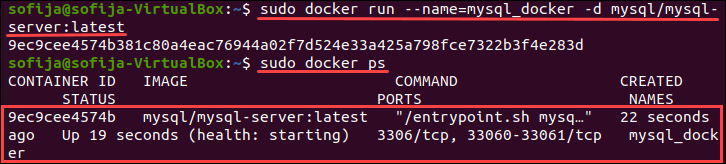
* Amend the [container\_name] with your desired name. If you don’t give a name, Docker chooses one for you, by default.
* The -d option tells Docker to run the **Container as a Service** in the background. Next, amend the [image\_tag\_name] with the name of the image you downloaded in the previous step. For this example, you can create a container called mysql\_docker with the latest version tag based on the following code snippet:

sudo docker run --name=[container\_name] -d mysql/mysql-server:latest

* To check if the Docker MySQL Container is running use the following command:

docker ps

* With this command, you can view the newly created container listed in the output. This consists of essential container details, for instance, the status of the virtual environment. The status is modified from health: **starting to healthy**, once you have completed the setup process.

[](https://lh6.googleusercontent.com/Ib7dArxgYcgLpQof3G8Vu9OBXioOMkVKB5NfCJxePvs7QLv7W_dRwdddI-Hiszn9BW0XgmeT2xhc0YSd6s1QQIKwSDcArPMHrJrL784lrXFwjazNup4Ug_wEkFmtFuBS3F8htmCZ)

[Image Source](https://phoenixnap.com/)

### Step 3: Connect with the Docker MySQL Container

* Before you go about connecting the MySQL Server Container with the host, you need to ensure that the MySQL Client Package has been installed with the following command:

apt-get install mysql-client

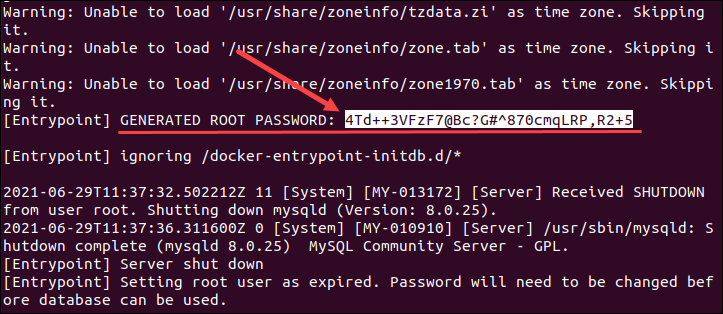
* Next, you need to open the log files for the Docker MySQL Container to find the root password you generated:

sudo docker logs [container\_name]

* Now that you have the mysql\_docker container, run the following command:

sudo docker logs mysql\_docker

* Look through the output and locate the line ‘**[Entrypoint] GENERATED ROOT PASSWORD:**’. Next, copy and paste the password in a text editor or a notepad for future use.

[](https://lh3.googleusercontent.com/1Pr8ohRW62KMKNPCgNTUKGCPSGeaWMlhifUQUQvauxBOL3yyhIErwP_cUBl2nK2mhXBS96-7MkoVOhGj557LoLYcihU5BVDaUPiDH_sP07-R_dn3DOCcP3cnF6Zqbm5-qRvdR329)

[Image Source](https://phoenixnap.com/)

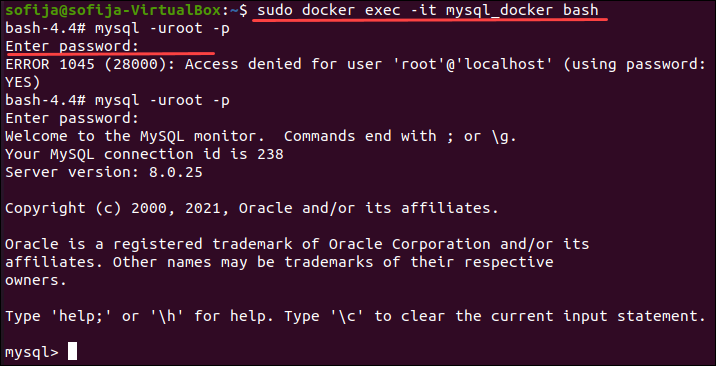
* Next, hover to the bash shell of the MySQL Container and type out the following command:

sudo docker exec -it [container\_name] bash

* For the container you created as an example, you can run this command:

sudo docker -it mysql\_docker bash

* Give the root password that you just copied from the logs file whenever you get prompted. This completes the connection process from the MySQL Client to the Server.

[](https://lh4.googleusercontent.com/BVf0b0N2bVaFFvdr1ZAxFCtUYkmkqeeHs6KmJ1hJeNEcavjzNUO21mUWbUGld8xI4WKHAXTADo-ZRf3B48Yk5gtQe1yYHWWDOFbD8QNKqZpDNauekLmjOxaPfWETVf-TbM3jiYgn)

[Image Source](https://phoenixnap.com/)

* Having completed the previous step for the Docker MySQL Container setup, change the root server password to protect your sensitive information with this snippet:

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '[newpassword]';

* For the above command, replace the [newpassword] clause with a desired strong password.

[Docker MySQL: Step 3 of Installation Part 3 | Hevo Data](https://lh4.googleusercontent.com/HT7XuKVp-qRRS3gn3aIH6upTjO4o65s3OzgLeXw5ZILn-zKsOZnIiOChDL4Sk-VzaNqqf6ZEO9yHPAL4ffje3uD7SIxWlt3pMY79fZkjNf5MIcV71VTgRAhC8CHg5eRvbozaENQS)

[Image Source](https://phoenixnap.com/)

## How to Configure your Docker MySQL Container?

If you wish to take a peek at the configuration of your Docker MySQL Container, you can find it in the ‘**/etc/mysql/my.cnf=**’ directory. If you want to modify the configuration, you need to create an alternative config file on the host machine and mount them inside the Docker MySQL Container.

* **Step 1**: Create a new directory on the host machine with this code snippet:

sudo mkdir -p /root/docker/[container\_name]/conf.d

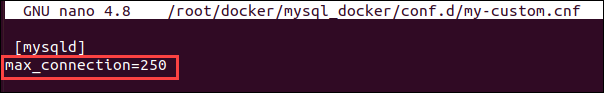
* **Step 2**: Next, you need to create a customized MySQL config file within that directory with the following command:

sudo nano /root/docker/[container\_name]/conf.d/my-custom.cnf

* **Step 3**: Having opened the file, you can add lines with your desired configuration. For instance, if you want to increment the maximum number of connections to 250 (as opposed to the default value 151), you can add the following lines to the configuration file:

[mysqld]

max\_connections=250

[](https://lh5.googleusercontent.com/Jt4DK16Q4nhsi7VvRqZ80PcnMG_JuJEOiOvzTkbzk4kEQYStmoXuqpKoW2cfqKxgEQnTsGt4lX4cNp63lqI4lL-pSo-htauOwYu6CnFeZgH8JpEcKA0ghpPUWDgjG1yosGG1Y1Ob)

[Image Source](https://phoenixnap.com/)

You can change the MySQL instance’s configuration by passing one or more environment variables to the docker run command line when starting the mysql image. If you start the container with a data directory that already contains a database, none of the variables below will have any effect: any pre-existing database will be ignored when the container starts.

You can change the MySQL instance’s configuration by passing one or more **environment variables** to the docker run command line when starting the mysql image. If you start the container with a data directory that already contains a database, none of the variables below will have any effect: any pre-existing database will be ignored when the container starts.

**MYSQL ROOT PASSWORD**  
This is a required variable that specifies the password for the MySQL root superuser account. It was set to my-secret-pw in the example above.

**MYSQL DATABASE**  
This variable is optional and allows you to name a database that will be created when the image starts up. If a user/password was provided (see below), that user will be granted superuser access to this database (corresponding to GRANT ALL).

**MYSQL USER**and**MYSQL PASSWORD**are two variables in MySQL.  
These variables are optional and are used to create a new user and set the password for that user. For the database specified by the MYSQL DATABASE variable, this user will be granted superuser permissions (see above). Both variables must be present in order to create a user.

It’s worth noting that this mechanism isn’t required to create the root superuser; that user is automatically created with the password specified by the MYSQL ROOT PASSWORD variable.

**MYSQL ALLOW EMPTY PASSWORD**  
This is a non-mandatory variable. Set to a non-empty value, such as yes, to allow the container to start with the root user’s password blank. NOTE: If you set this variable to yes, your MySQL instance will be completely unprotected, allowing anyone to gain complete superuser access.

**MYSQL RANDOM ROOT PASSWORD**  
This is a non-mandatory variable. To generate a random initial password for the root user, set to a non-empty value, such as yes (using pwgen). The root password is generated and printed to stdout (GENERATED ROOT PASSWORD: …..).

**MYSQL ONETIME PASSWORD**  
Once init is complete, the root (not the user specified in MYSQL USER!) user is marked as expired, requiring a password change on the first login. This setting is activated by any non-empty value. NOTE: This feature is only available in MySQL 5.6 and higher. When using this option with MySQL 5.5, an appropriate error will be thrown during initialization.

**MYSQL INITDB SKIP TZINFO**  
The timezone data required for the CONVERT TZ() function is loaded automatically by default by the entrypoint script. Any non-empty value disables timezone loading if it is not required.

* **Step 4**: Save the file and exit. Now, for the changes to take effect you need to rerun the MySQL Container after removing it. For this, the container leverages a combination of configuration settings from the newly created file along with the default config files. You can run the Docker MySQL Container and map the volume path with the following code snippet:

docker run

--detach

--name=[container\_name]

--env="MYSQL\_ROOT\_PASSWORD=[my\_password]"

--publish 6603:3306

--volume=/root/docker/[container\_name]/conf.d:/etc/mysql/conf.d

mysql

* **Step 5**: Now, to check if the Docker MySQL Container has loaded the configuration from the host, you can run the following command:

mysql -uroot -pmypassword -h127.0.0.1 -P6603 -e 'show global variables like "max\_connections"';

* **Step 6**: You can now check the maximum number of connections for your Docker MySQL Container. It should show 250 connections.

## Bonus Setup: How to Connect Host Machine with Docker MySQL?

* **Step 1**: Start the container and map it to a local port to begin the process. In this step, you will be mapping the port from the Docker Container to a port on localhost which can be used to connect to MySQL. First, you need to remove the container you created with the following set of commands:

docker stop learn-mysql-docker

docker rm learn-mysql-docker

* **Step 2**: After removing the container, you can start a new container with the same MySQL Image and map the Docker port to the localhost port. You can do this with the help of the following command:

docker run -p 13306:3306 --name mysql-docker-local -eMYSQL\_ROOT\_PASSWORD=Password -d mysql:latest

* **Step 3**: Next, you need to make sure that there are no errors after the above command has been executed. If everything works out fine, the above command would print out a generated container ID.
* **Step 4**: After starting the container, you need to connect to MySQL from localhost by either using a GUI tool like MySQL Workbench or the command-line.
* **Step 5**: For the command line, you can use this snippet to get things started:

mysql --host=127.0.0.1 --port=13306 -u root -p

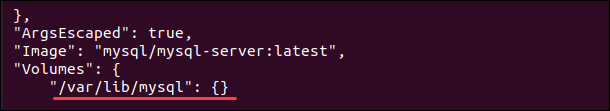
* **Step 6**: After you’ve executed the command, you will see that you are connected to a Docker-based MySQL instance. You can run any command to check if it’s working fine.

## Managing your Docker MySQL Container

* **Step 1**: Docker stores data in its internal volume by default, so, to check the location of Docker’s internal volumes you can use the following code snippet:

sudo docker inspect [container\_name]

* **Step 2**: For this command, you can see the /var/lib/mysql mounted in the internal volume.

[](https://lh5.googleusercontent.com/PhuXdstiRem713hT4bDymqc3AKFmkVTiKsYiArw8nyBuYhp9k9JAimtbBaDOIBwLx5MLMNPsH_J_84kBQxRiP25uZfXMYqWopOhTws0e7NdQFyS2kXtlXYKf_9G6Y60sOqqp26Hk)

[Image Source](https://phoenixnap.com/)

* **Step 3**: You can even modify the location of the data directory and make one that exists on the host of your Docker MySQL Container instead. Having a volume outside the container lets other tools and applications access the volumes when required.
* **Step 4**: Next, you need to find an appropriate volume of the host and make a data directory on it with the following code snippet:

sudo mkdir -p /storage/docker/mysql-data

* **Step 5**: After executing the previous command, start the container again by mounting the previously created directory with this command:

docker run

--detach

--name=[container\_name]

--env="MYSQL\_ROOT\_PASSWORD=my\_password"

--publish 6603:3306

--volume=/root/docker/[container\_name]/conf.d:/etc/mysql/conf.d

--volume=/storage/docker/mysql-data:/var/lib/mysql

mysql

* **Step 6**: If you inspect the container, you should be able to peek at the data within Docker MySQL Container and realize that it’s stored on the host system now.

sudo docker inspect [container\_name]

Here are a few essential components to help you manage your Docker Compose MySQL Container:

* [How to Start the Container for MySQL?](https://hevodata.com/learn/docker-mysql/#m1)
* [How to Restart the Container for MySQL?](https://hevodata.com/learn/docker-mysql/#m2)
* [How to Stop the Container for MySQL?](https://hevodata.com/learn/docker-mysql/#m3)

### How to Start the Container for MySQL?

The code snippet you can use to start the Docker MySQL Container is as follows:

sudo docker start [container\_name]

### How to Restart the Container for MySQL?

The code snippet you can use to restart the Docker MySQL Container is as follows:

sudo docker restart [container\_name]

### How to Stop the Container for MySQL?

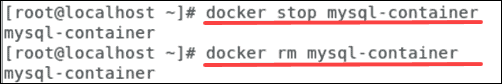
The code snippet you can use to stop the Docker MySQL Container is as follows:

sudo docker stop [container\_name]

## How to Delete your Docker MySQL Container?

If you wish to delete the Docker MySQL Container, you need to make sure that you’ve stopped it first. Next, you can remove the Docker Container with the following code snippet:

sudo docker rm [container\_name]

[](https://lh5.googleusercontent.com/veZqHiZAW73xicQ21fzM3XJH_xQXJezx0ULl5ApHbEn9gpmg_Q9Nr--6v73GBF64BYhVph4pS2_22SRYEK6NOsyUcjzFfbq64-e3dvqf8J_BMEEzxNIOoU8tJNu2kPAc0fBNe9-L)

[Image Source](https://phoenixnap.com/kb/mysql-docker-container#ftoc-heading-3)

## How to Upgrade your Dockerised MySQL Container?

To upgrade a Docker MySQL Container, you can follow the steps mentioned below:

* **Step 1**: Stop the MySQL Server for this instance using the code snippet mentioned below:

docker stop mysql57

* **Step 2**: Next, download the MySQL 8.0 Server Docker image and make sure that you have the right tag for MySQL 8.0.
* **Step 3**: Start a new MySQL 8.0 Docker Container with the old configuration and server data that have been applied on the host. Run the following command for the same:

docker run --name=mysql80

--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf

--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql

-d mysql/mysql-server:8.0

* **Step 4**: Now, you need to wait for the server to finish startup. You can check the status of the server with the docker ps command.
* **Step 5**: For earlier versions of MySQL, you need to run the mysql\_upgrade utility in the MySQL 8.0 Server Container as follows:

docker exec -it mysql80 mysql\_upgrade -uroot -p

* **Step 6**: Enter the root password for your old MySQL Server and finish the upgrade by restarting the MySQL Server Container as follows:

docker restart mysql80

## Cautions

* **Where to Store Data**: Data is stored in a variety of ways by applications that run in Docker containers. Users of the mysql images should become familiar with the options available, which include:
  + Allow Docker to manage the storage of your database files on the host system by writing the database files to disc using its own internal volume management. This is the default setting, and it is simple and transparent to the user. The downside is that for tools and applications that run directly on the host system, such as outside containers, the files may be difficult to locate.
  + Make a data directory on the host system (outside the container) and mount it to a directory that can be seen from inside the container. This places the database files in a well-known location on the host system, making them accessible to the host system’s tools and applications. The disadvantage is that the user must ensure that the directory exists and that the host system’s security mechanisms, such as directory permissions and other security mechanisms, are configured correctly.

The Docker documentation is a good place to start learning about the various storage options and variations, and there are a number of blogs and forum posts that discuss and offer advice in this area. For the latter option, we’ll just show the basic procedure here:

Create a data directory, such as /my/own/datadir, on a suitable volume on your host system.

Begin your mysql container as follows:

-v /my/own/datadir:/var/lib/mysql -e MYSQL ROOT PASSWORD=my-secret-pw -d docker run —name some-mysql mysql:tag  
The -v /my/own/datadir:/var/lib/mysql part of the command mounts the underlying host system’s /my/own/datadir directory as /var/lib/mysql inside the container, where MySQL will write its data files by default.

* **No Connections Until MySQL init Completes**: A default database will be created if there is no database initialised when the container starts. While this is normal behaviour, it means that it will not accept incoming connections until the initialization is complete. This could cause problems when using automation tools like docker-compose, which start multiple containers at the same time. If the application you’re trying to connect to MySQL doesn’t handle MySQL downtime or waiting for the service to start gracefully, a connect-retry loop may be required before the service starts. See WordPress or Bonita for an example of such an implementation in the official images.
* **Usage Against an Existing Database**: The $MYSQL ROOT PASSWORD variable should be omitted from the run command line if you start your mysql container instance with a data directory that already contains a database (specifically, a mysql subdirectory); it will be ignored in any case, and the pre-existing database will not be changed.
* **Running as an Arbitrary User**: If you know your directory permissions are already set correctly (for example, if you’re running against an existing database, as described above), or if you need to run mysqld with a specific UID/GID, you can use this image with —user set to any value (other than root/0) to get the desired access/configuration:

$ mkdir data

$ ls -lnd data

drwxr-xr-x 2 1000 1000 4096 Aug 27 15:54 data

$ docker run -v "$PWD/data":/var/lib/mysql --user 1000:1000 --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:tag

* **Creating Database Dumps**: The majority of standard tools will work, though their use may be a little convoluted in some cases to ensure access to the mysqld server. Using docker exec and running the tool from the same container, as shown below, is a simple way to ensure this:

$ docker exec some-mysql sh -c 'exec mysqldump --all-databases -uroot -p"$MYSQL\_ROOT\_PASSWORD"' > /some/path/on/your/host/all-databases.sql

* **Restoring Data from Dump Files**: For data recovery. You can use the -i flag with the docker exec command, as shown below:

$ docker exec -i some-mysql sh -c 'exec mysql -uroot -p"$MYSQL\_ROOT\_PASSWORD"' < /some/path/on/your/host/all-databases.sql

## Conclusion

This blog talks about the different steps you can follow to set up a **Docker MySQL**Container after a brief introduction to the salient features of MySQL and Docker. It also covers the advantages of leveraging Docker MySQL Containers and various handy management and configuration tips.

Extracting complex data from a diverse set of data sources to carry out an insightful analysis can be challenging, and this is where **Hevo** saves the day![Hevo](https://hevodata.com/) offers a faster way to move data from Databases or SaaS applications like **MySQL** into your Data Warehouse to be visualized in a BI tool. Hevo is fully automated and hence does not require you to code.

[**VISIT OUR WEBSITE TO EXPLORE HEVO**](https://hevodata.com/)

Want to take Hevo for a spin?

[**SIGN UP**](https://hevodata.com/signup?step=email) and experience the feature-rich Hevo suite first hand. You can also have a look at the unbeatable[pricing](https://hevodata.com/pricing/) that will help you choose the right plan for your business needs.