mysql ☆

Docker Official Images

MySQL is a widely used, open-source relational database management system (RDBMS).

10M+

Container Linux x86-64 Databases Official Image

Copy and paste to pull this image

```
docker pull mysql
```

View Available Tags

DESCRIPTION REVIEWS TAGS

Supported tags and respective `Dockerfile` links

- `8.0.16`, `8.0`, `8`, `latest`
- `5.7.26`, `5.7`, `5`
- `5.6.44`, `5.6`

Quick reference

- **Where to get help**:
  the Docker Community Forums, the Docker Community Slack, or Stack Overflow
- **Where to file issues**:
  https://github.com/docker-library/mysql/issues
- **Maintained by**:
  the Docker Community and the MySQL Team
- **Supported architectures**: (more info)
  `amd64`
- **Published image artifact details**:
  repo-info repo's `repos/mysql/` directory (history)
  (image metadata, transfer size, etc)
- **Image updates**:
  official-images PRs with label `library/mysql`
  official-images repo's `library/mysql` file (history)
- **Source of this description**:
  docs repo's `mysql/` directory (history)
- **Supported Docker versions**:
  the latest release (down to 1.6 on a best-effort basis)

What is MySQL?

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, covering the entire range from personal projects and websites, via e-commerce and information services, all the way to high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.

For more information and related downloads for MySQL Server and other MySQL products, please visit www.mysql.com.

How to use this image

Start a `mysql` server instance

Starting a MySQL instance is simple:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

... where `some-mysql` is the name you want to assign to your container, `my-secret-pw` is the password to be set for the MySQL root user and `tag` is the tag specifying the MySQL version you want. See the list above for relevant tags.

Connect to MySQL from the MySQL command line client

The following command starts another `mysql` container instance and runs the `mysql` command line client against your original `mysql` container, allowing you to execute SQL statements against your database instance:

```
$ docker run -it --network some-network --rm mysql mysql -hsome-mysql -uexample-user -p
```

... where `some-mysql` is the name of your original `mysql` container (connected to the `some-network` Docker network).

This image can also be used as a client for non-Docker or remote instances:

```
$ docker run -it --rm mysql mysql -hsome.mysql.host -usome-mysql-user -p
```

More information about the MySQL command line client can be found in the MySQL documentation

... via `docker stack deploy` or `docker-compose`

Example `stack.yml` for `mysql`:

```
# Use root/example as user/password credentials
version: '3.1'

services:

  db:
    image: mysql
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: example

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

Try in PWD

Run `docker stack deploy -c stack.yml mysql` (or `docker-compose -f stack.yml up`), wait for it to initialize completely, and visit `http://swarm-ip:8080`, `http://localhost:8080`, or `http://host-ip:8080` (as appropriate).

Container shell access and viewing MySQL logs

The `docker exec` command allows you to run commands inside a Docker container. The following command line will give you a bash shell inside your `mysql` container:

```
$ docker exec -it some-mysql bash
```

The log is available through Docker's container log:

```
$ docker logs some-mysql
```

Using a custom MySQL configuration file

The default configuration for MySQL can be found in `/etc/mysql/my.cnf`, which may `!includedir` additional directories such as `/etc/mysql/conf.d` or `/etc/mysql/mysql.conf.d`. Please inspect the relevant files and directories within the `mysql` image itself for more details.

If `/my/custom/config-file.cnf` is the path and name of your custom configuration file, you can start your `mysql` container like this (note that only the directory path of the custom config file is used in this command):

```
$ docker run --name some-mysql -v /my/custom:/etc/mysql/conf.d -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

This will start a new container `some-mysql` where the MySQL instance uses the combined startup settings from `/etc/mysql/my.cnf` and `/etc/mysql/conf.d/config-file.cnf`, with settings from the latter taking precedence.

Configuration without a `cnf` file

Many configuration options can be passed as flags to `mysqld`. This will give you the flexibility to customize the container without needing a `cnf` file. For example, if you want to change the default encoding and collation for all tables to use UTF-8 (`utf8mb4`) just run the following:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag --character-set-server=utf8mb4 --collation-server=utf8mb4_unicod
```

If you would like to see a complete list of available options, just run:

```
$ docker run -it --rm mysql:tag --verbose --help
```

Environment Variables

When you start the `mysql` image, you can adjust the configuration of the MySQL instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

See also https://dev.mysql.com/doc/refman/5.7/en/environment-variables.html for documentation of environment variables which MySQL itself respects (especially variables like `MYSQL_HOST`, which is known to cause issues when used with this image).

`MYSQL_ROOT_PASSWORD`

This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account. In the above example, it was set to `my-secret-pw`.

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access (corresponding to `GRANT ALL`) to this database.

`MYSQL_USER`, `MYSQL_PASSWORD`

These variables are optional, used in conjunction to create a new user and to set that user's password. This user will be granted superuser permissions (see above) for the database specified by the `MYSQL_DATABASE` variable. Both variables are required for a user to be created.

Do note that there is no need to use this mechanism to create the root superuser, that user gets created by default with the password specified by the `MYSQL_ROOT_PASSWORD` variable.

`MYSQL_ALLOW_EMPTY_PASSWORD`

This is an optional variable. Set to `yes` to allow the container to be started with a blank password for the root user. *NOTE*: Setting this variable to `yes` is not recommended unless you really know what you are doing, since this will leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access.

`MYSQL_RANDOM_ROOT_PASSWORD`

This is an optional variable. Set to `yes` to generate a random initial password for the root user (using `pwgen`). The generated root password will be printed to stdout (`GENERATED ROOT PASSWORD: .....`).

`MYSQL_ONETIME_PASSWORD`

Sets root (*not* the user specified in `MYSQL_USER`!) user as expired once init is complete, forcing a password change on first login. *NOTE*: This feature is supported on MySQL 5.6+ only. Using this option on MySQL 5.5 will throw an appropriate error during initialization.

Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files. For example:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD_FILE=/run/secrets/mysql-root -d mysql:tag
```

Currently, this is only supported for `MYSQL_ROOT_PASSWORD`, `MYSQL_ROOT_HOST`, `MYSQL_DATABASE`, `MYSQL_USER`, and `MYSQL_PASSWORD`.

Initializing a fresh instance

When a container is started for the first time, a new database with the specified name will be created and initialized with the provided configuration variables. Furthermore, it will execute files with extensions `.sh`, `.sql` and `.sql.gz` that are found in `/docker-entrypoint-initdb.d`. Files will be executed in alphabetical order. You can easily populate your `mysql` services by mounting a SQL dump into that directory and provide custom images with contributed data. SQL files will be imported by default to the database specified by the `MYSQL_DATABASE` variable.

Caveats

Where to Store Data

Important note: There are several ways to store data used by applications that run in Docker containers. We encourage users of the `mysql` images to familiarize themselves with the options available, including:

- Let Docker manage the storage of your database data by writing the database files to disk on the host system using its own internal volume management. This is the default and is easy and fairly transparent to the user. The downside is that the files may be hard to locate for tools and applications that run directly on the host system, i.e. outside containers.
- Create a data directory on the host system (outside the container) and mount this to a directory visible from inside the container. This places the database files in a known location on the host system, and makes it easy for tools and applications on the host system to access the files. The downside is that the user needs to make sure that the directory exists, and that e.g. directory permissions and other security mechanisms on the host system are set up correctly.

The Docker documentation is a good starting point for understanding the different storage options and variations, and there are multiple blogs and forum postings that discuss and give advice in this area. We will simply show the basic procedure here for the latter option above:

1. Create a data directory on a suitable volume on your host system, e.g. `/my/own/datadir`.
2. Start your `mysql` container like this:

```
$ docker run --name some-mysql -v /my/own/datadir:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

The `-v /my/own/datadir:/var/lib/mysql` part of the command mounts the `/my/own/datadir` directory from the underlying host system as `/var/lib/mysql` inside the container, where MySQL by default will write its data files.

No connections until MySQL init completes

If there is no database initialized when the container starts, then a default database will be created. While this is the expected behavior, this means that it will not accept incoming connections until such initialization completes. This may cause issues when using automation tools, such as `docker-compose`, which start several containers simultaneously.

If the application you're trying to connect to MySQL does not handle MySQL downtime or waiting for MySQL to start gracefully, then a putting a connect-retry loop before the service starts might be necessary. For an example of such an implementation in the official images, see WordPress or Bonita.

Usage against an existing database

If you start your `mysql` container instance with a data directory that already contains a database (specifically, a `mysql` subdirectory), the `$MYSQL_ROOT_PASSWORD` variable should be omitted from the run command line; it will in any case be ignored, and the pre-existing database will not be changed in any way.

Running as an arbitrary user

If you know the permissions of your directory are already set appropriately (such as running against an existing database, as described above) or you have need of running `mysqld` with a specific UID/GID, it is possible to invoke this image with `--user` set to any value (other than `root` / `0`) in order to achieve 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

Most of the normal tools will work, although their usage might be a little convoluted in some cases to ensure they have access to the `mysqld` server. A simple way to ensure this is to use `docker exec` and run the tool from the same container, similar to the following:

```
$ 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 restoring data. You can use `docker exec` command with `-i` flag, similar to the following:

```
$ docker exec -i some-mysql sh -c 'exec mysql -uroot -p"$MYSQL_ROOT_PASSWORD"' < /some/path/on/your/host/all-databases.sql
```

License

View license information for the software contained in this image.

As with all Docker images, these likely also contain other software which may be under other licenses (such as Bash, etc from the base distribution, along with any direct or indirect dependencies of the primary software being contained).

Some additional license information which was able to be auto-detected might be found in the `repo-info` repository's `mysql/` directory.

As for any pre-built image usage, it is the image user's responsibility to ensure that any use of this image complies with any relevant licenses for all software contained within.