**Supported tags and respective Dockerfile links**

* [8.0.0, 8.0, 8 (*8.0/Dockerfile*)](https://github.com/docker-library/mysql/blob/4dd33136c4739667a223d39b6f829beb27b235cf/8.0/Dockerfile)
* [5.7.17, 5.7, 5, latest (*5.7/Dockerfile*)](https://github.com/docker-library/mysql/blob/c207cc19a272a6bfe1916c964ed8df47f18479e7/5.7/Dockerfile)
* [5.6.35, 5.6 (*5.6/Dockerfile*)](https://github.com/docker-library/mysql/blob/fd3bdabe513643b62fc379f365b2b5dd10886311/5.6/Dockerfile)
* [5.5.54, 5.5 (*5.5/Dockerfile*)](https://github.com/docker-library/mysql/blob/ee989d0666458d08dd79c55f7abb62be29a9cb3d/5.5/Dockerfile)

For more information about this image and its history, please see [the relevant manifest file (library/mysql)](https://github.com/docker-library/official-images/blob/master/library/mysql). This image is updated via [pull requests to the docker-library/official-images GitHub repo](https://github.com/docker-library/official-images/pulls?q=label%3Alibrary%2Fmysql).

For detailed information about the virtual/transfer sizes and individual layers of each of the above supported tags, please see [the repos/mysql/tag-details.md file](https://github.com/docker-library/repo-info/blob/master/repos/mysql/tag-details.md) in [the docker-library/repo-info GitHub repo](https://github.com/docker-library/repo-info).

**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](http://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 an application in another Docker container**

This image exposes the standard MySQL port (3306), so container linking makes the MySQL instance available to other application containers. Start your application container like this in order to link it to the MySQL container:

$ docker run --name some-app --link some-mysql:mysql -d application-that-uses-mysql

**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 --link some-mysql:mysql --rm mysql sh -c 'exec mysql -h"$MYSQL\_PORT\_3306\_TCP\_ADDR" -P"$MYSQL\_PORT\_3306\_TCP\_PORT" -uroot -p"$MYSQL\_ENV\_MYSQL\_ROOT\_PASSWORD"'

... where some-mysql is the name of your original mysql container.

This image can also be used as a client for non-Docker or remote MySQL 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](http://dev.mysql.com/doc/en/mysql.html)

**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 MySQL Server log is available through Docker's container log:

$ docker logs some-mysql

**Using a custom MySQL configuration file**

The MySQL startup configuration is specified in the file /etc/mysql/my.cnf, and that file in turn includes any files found in the /etc/mysql/conf.d directory that end with .cnf. Settings in files in this directory will augment and/or override settings in /etc/mysql/my.cnf. If you want to use a customized MySQL configuration, you can create your alternative configuration file in a directory on the host machine and then mount that directory location as /etc/mysql/conf.d inside the mysql container.

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.

Note that users on host systems with SELinux enabled may see issues with this. The current workaround is to assign the relevant SELinux policy type to your new config file so that the container will be allowed to mount it:

$ chcon -Rt svirt\_sandbox\_file\_t /my/custom

**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\_unicode\_ci

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.

**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](http://dev.mysql.com/doc/en/adding-users.html)) 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.

**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](https://docs.docker.com/engine/tutorials/dockervolumes/#mount-a-host-file-as-a-data-volume) and provide [custom images](https://docs.docker.com/reference/builder/) 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](https://docs.docker.com/engine/tutorials/dockervolumes/#adding-a-data-volume). 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](https://docs.docker.com/engine/tutorials/dockervolumes/#mount-a-host-directory-as-a-data-volume). 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:
3. $ 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.

Note that users on host systems with SELinux enabled may see issues with this. The current workaround is to assign the relevant SELinux policy type to the new data directory so that the container will be allowed to access it:

$ chcon -Rt svirt\_sandbox\_file\_t /my/own/datadir

**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.

**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.

**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

**Supported Docker versions**

This image is officially supported on Docker version 1.12.5.

Support for older versions (down to 1.6) is provided on a best-effort basis.

Please see [the Docker installation documentation](https://docs.docker.com/installation/) for details on how to upgrade your Docker daemon.

**User Feedback**

**Issues**

If you have any problems with or questions about this image, please contact us through a [GitHub issue](https://github.com/docker-library/mysql/issues). If the issue is related to a CVE, please check for [a cve-tracker issue on the official-images repository first](https://github.com/docker-library/official-images/issues?q=label%3Acve-tracker).

You can also reach many of the official image maintainers via the #docker-library IRC channel on [Freenode](https://freenode.net).

**Contributing**

You are invited to contribute new features, fixes, or updates, large or small; we are always thrilled to receive pull requests, and do our best to process them as fast as we can.

Before you start to code, we recommend discussing your plans through a [GitHub issue](https://github.com/docker-library/mysql/issues), especially for more ambitious contributions. This gives other contributors a chance to point you in the right direction, give you feedback on your design, and help you find out if someone else is working on the same thing.

**Documentation**

Documentation for this image is stored in the [mysql/ directory](https://github.com/docker-library/docs/tree/master/mysql) of the [docker-library/docs GitHub repo](https://github.com/docker-library/docs). Be sure to familiarize yourself with the [repository's README.md file](https://github.com/docker-library/docs/blob/master/README.md) before attempting a pull request.