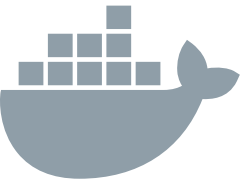

mariadb

Official Image ☆

MariaDB Server is a high performing open source relational database, forked from MySQL.

1B+

Container Linux x86-64 ARM 64 PowerPC 64 LE IBM Z 386 Databases Official Image

Copy and paste to pull this image

```
docker pull mariadb
```

View Available Tags

Description Reviews Tags

Note: the description for this image is longer than the Hub length limit of 25000, so has been trimmed. The full description can be found at https://github.com/docker-library/docs/tree/master/mariadb/README.md. See docker/hub-beta-feedback#238 for more information.

Quick reference

- **Maintained by**: MariaDB developer community

- **Where to get help**: Database Adminstrators (Stack Exchange), MariaDB Knowledge Base (Ask a Question here available).

Also see the "Getting Help with MariaDB" article on the MariaDB Knowledge Base.

Supported tags and respective `Dockerfile` links

- `10.8.2-rc-focal`, `10.8-rc-focal`, `10.8.2-rc`, `10.8-rc`
- `10.7.3-focal`, `10.7-focal`, `10-focal`, `focal`, `10.7.3`, `10.7`, `10`, `latest`
- `10.6.7-focal`, `10.6-focal`, `10.6.7`, `10.6`
- `10.5.15-focal`, `10.5-focal`, `10.5.15`, `10.5`
- `10.4.24-focal`, `10.4-focal`, `10.4.24`, `10.4`
- `10.3.34-focal`, `10.3-focal`, `10.3.34`, `10.3`
- `10.2.43-bionic`, `10.2-bionic`, `10.2.43`, `10.2`

Quick reference (cont.)

- **Where to file issues**: Issues can be filed on https://jira.mariadb.org/ under the "MDEV" Project and "Docker" Component, or on GitHub
- **Supported architectures**: (more info) `amd64`, `arm64v8`, `ppc64le`, `s390x`
- **Published image artifact details**: repo-info repo's `repos/mariadb/` directory (history) (image metadata, transfer size, etc)
- **Image updates**: official-images repo's `library/mariadb` label
  official-images repo's `library/mariadb` file (history)
- **Source of this description**: docs repo's `mariadb/` directory (history)

What is MariaDB?

MariaDB Server is one of the most popular database servers in the world. It's made by the original developers of MySQL and guaranteed to stay open source. Notable users include Wikipedia, DBS Bank, and ServiceNow.

The intent is also to maintain high compatibility with MySQL, ensuring a library binary equivalency and exact matching with MySQL APIs and commands. MariaDB developers continue to develop new features and improve performance to better serve its users.

How to use this image

Start a `mariadb` server instance

Starting a MariaDB instance with the latest version is simple:

```
$ docker run --detach --name some-mariadb --env MARIADB_USER=example-user --env MARIADB_PASSWORD=my_cool_secret --env MAR
```

or:

```
$ docker network create some-network
$ docker run --detach --network some-network --name some-mariadb --env MARIADB_USER=example-user --env MARIADB_PASSWORD=m
```

... where `some-network` is a newly created network (other than `bridge` as the default network), `some-mariadb` is the name you want to assign to your container, `my-secret-pw` is the password to be set for the MariaDB root user. See the list above for relevant tags to match your needs and environment.

Connect to MariaDB from the MySQL/MariaDB command line client

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

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

... where `some-mariadb` is the name of your original `mariadb` 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 mariadb mysql -h <server container IP> -u example-user -p
```

That will give you a standard MariaDB prompt. You can test it with:

```
MariaDB [(none)]> SELECT VERSION();
```

... which should give you the version. You can then use `exit` to leave the MariaDB command line client and the client container.

More information about the MariaDB command-line client can be found in the MariaDB Knowledge Base

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

Example `stack.yml` for `mariadb`:

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

services:

  db:
    image: mariadb
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: example

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

Run `docker stack deploy -c stack.yml mariadb` (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 MariaDB 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 `mariadb` container:

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

The log is available through Docker's container log:

```
$ docker logs some-mariadb
```

Using a custom MariaDB configuration file

The 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 MariaDB 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 `mariadb` container.

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

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

This will start a new container `some-mariadb` where the MariaDB 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 run on port 3808 just run the following:

```
$ docker run --name some-mariadb -e MARIADB_ROOT_PASSWORD=my-secret-pw -d mariadb:latest --port 3808
```

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

```
$ docker run -it --rm mariadb:latest --verbose --help
```

Environment Variables

When you start the `mariadb` image, you can adjust the initialization of the MariaDB 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.

From tag 10.2.38, 10.3.29, 10.4.19, 10.5.10 onwards, and all 10.6 and later tags, the `MARIADB_*` equivalent variables are provided. `MARIADB_*` variants will always be used in preference to `MYSQL_*` variants.

One of `MARIADB_ROOT_PASSWORD`, `MARIADB_ALLOW_EMPTY_ROOT_PASSWORD`, or `MARIADB_RANDOM_ROOT_PASSWORD` (or equivalents, including `*_FILE`), is required. The other environment variables are optional.

`MARIADB_ROOT_PASSWORD` / `MYSQL_ROOT_PASSWORD`

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

`MARIADB_ALLOW_EMPTY_ROOT_PASSWORD` / `MYSQL_ALLOW_EMPTY_PASSWORD`

Set to a non-empty value, like `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 MariaDB instance completely unprotected, allowing anyone to gain complete superuser access.

`MARIADB_RANDOM_ROOT_PASSWORD` / `MYSQL_RANDOM_ROOT_PASSWORD`

Set to a non-empty value, like `yes`, to generate a random initial password for the root user. The generated root password will be printed to stdout (`GENERATED ROOT PASSWORD: .....`).

`MARIADB_ROOT_HOST` / `MYSQL_ROOT_HOST`

This is the hostname part of the root user created. By default this is `%`, however it can be set to any default MariaDB allowed hostname component. Setting this to `localhost` will prevent any root user being accessible except via the unix socket.

`MARIADB_MYSQL_LOCALHOST_USER` / `MARIADB_MYSQL_LOCALHOST_GRANTS`

Set `MARIADB_MYSQL_LOCALHOST_USER` to a non-empty value to create the `mysql@locahost` database user. This user is especially useful for a variety of health checks and backup scripts.

The `mysql@localhost` user gets USAGE privileges by default. If more access is required, additional global privileges in the form of a comma separated list can be provided. If you are sharing a volume containing MariaDB's unix socket (`/var/run/mysqld` by default), privileges beyond `USAGE` can result in confidentiality, integrity and availability risks, so use a minimal set. See the example below on using Mariabackup. The `healthcheck.sh` script also documents the required privileges for each health check test.

`MARIADB_DATABASE` / `MYSQL_DATABASE`

This variable allows you to specify the name of a database to be created on image startup.

`MARIADB_USER` / `MYSQL_USER`, `MARIADB_PASSWORD` / `MYSQL_PASSWORD`

These are used in conjunction to create a new user and to set that user's password. Both user and password variables are required for a user to be created. This user will be granted all access (corresponding to `GRANT ALL`) to the `MARIADB_DATABASE` database.

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 `MARIADB_ROOT_PASSWORD` / `MYSQL_ROOT_PASSWORD` variable.

`MARIADB_INITDB_SKIP_TZINFO` / `MYSQL_INITDB_SKIP_TZINFO`

By default, the entrypoint script automatically loads the timezone data needed for the `CONVERT_TZ()` function. If it is not needed, any non-empty value disables timezone loading.

`MARIADB_AUTO_UPGRADE` / `MARIADB_DISABLE_UPGRADE_BACKUP`

Set `MARIADB_AUTO_UPGRADE` to a non-empty value to have the entrypoint check whether `mysql_upgrade` / `mariadb-upgrade` needs to run, and if so, run the upgrade before starting the MariaDB server.

Before the upgrade, a backup of the system database is created in the top of the datadir with the name `system_mysql_backup_*.sql.zst`. This backup process can be disabled with by setting `MARIADB_DISABLE_UPGRADE_BACKUP` to a non-empty value.

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 MARIADB_ROOT_PASSWORD_FILE=/run/secrets/mysql-root -d mariadb:latest
```

Currently, this is only supported for `MARIADB_ROOT_PASSWORD`, `MARIADB_ROOT_HOST`, `MARIADB_DATABASE`, `MARIADB_USER`, and `MARIADB_PASSWORD` (and `MYSQL_*` equivalents of these).

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`, `.sql.gz`, `.sql.xz` and `.sql.zst` that are found in `/docker-entrypoint-initdb.d`. Files will be executed in alphabetical order. `.sh` files without file execute permission are sourced rather than executed. You can easily populate your `mariadb` 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 `MARIADB_DATABASE` / `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 `mariadb` 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 `mariadb` container like this:

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

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 MariaDB by default will write its data files.

No connections until MariaDB 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.

Health/Liveness/Readiness Checking

See the "Official Images" FAQ for why there is no default `HEALTHCHECK` directive. However, you can use the `/usr/local/bin/healthcheck.sh` script to choose from a (non-exhaustive) list of tests to check for whatever you consider health/liveness/readiness. Refer to the script's sources to learn about how to use it and which exact tests are provided.

Usage against an existing database

If you start your `mariadb` container instance with a data directory that already contains a database (specifically, a `mysql` subdirectory), no environment variables that control initialization will be needed or examined, and no pre-existing databases will not be changed. The only exception is the non-default `MARIADB_AUTO_UPGRADE` environment variable, that might cause `mysql_upgrade` / `mariadb-upgrade` to run, which might change the system tables.

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-mariadb sh -c 'exec mysqldump --all-databases -uroot -p"$MARIADB_ROOT_PASSWORD"' > /some/path/on/your/
```

Restoring data from dump files

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

```
$ docker exec -i some-mariadb sh -c 'exec mysql -uroot -p"$MARIADB_ROOT_PASSWORD"' < /some/path/on/your/host/all-database
```

If one or more databases, but neither `--all-databases` nor the `mysql` database, were dumped, these databases can be restored by placing the resulting sql file in the `/docker-entrypoint-initdb.d` directory.

Creating backups with Mariabackup

To perform a backup using Mariabackup, a second container is started that shares the original container's data directory. An additional volume for the backup needs to be included in the second backup instance. Authentication against the MariaDB database instance is required to successfully complete the backup. In the example below a `mysql@localhost` user is used with the MariaDB server's unix socket shared with the backup container.

```
$ docker volume create some-mariadb-socket
$ docker run --name some-mariadb -v /my/own/datadir:/var/lib/mysql -v some-mariadb-socket:/var/run/mysqld -e MARIADB_MYSQ
```

Note: Privileges listed here are for 10.5+. For an exact list, see the Knowledge Base documentation for Mariabackup: Authentication and Privileges.

Mariabackup will run as the `mysql` user in the container, so the permissions on `/backup` will need to ensure that it can be written to by this user:

```
$ docker volume create some-mariadb-backup
$ docker run --rm some-mariadb-backup -v some-mariadb-backup:/backup mariadb:latest chown mysql: /backup
```

To perform the backup:

```
$ docker run --user mysql -v some-mariadb-socket:/var/run/mysqld -v some-mariadb-backup:/backup -v /my/own/datadir:/var/l
```

Restore backups with Mariabackup

These steps restore the backup made with Mariabackup.

At some point before doing the restore, the backup needs to be prepared. Perform the prepare like this:

```
$ docker run --user mysql --rm -v some-mariadb-backup:/backup mariadb:latest mariabackup --prepare --target-dir=/backup
```

Now that the image is prepared, start the container with both the data and the backup volumes and restore the backup:

```
$ docker run --user mysql --rm -v /my/new/datadir:/var/lib/mysql -v some-mariadb-backup:/backup mariadb:latest mariabacku
```

With `/my/new/datadir` containing the restored backup, start normally as this is an initialized data directory:

```
$ docker run --name some-mariadb -v /my/new/datadir:/var/lib/mysql -d mariadb:latest
```

For further information on Mariabackup, see the Mariabackup Knowledge Base.

How to reset root and user passwords

If you have an existing data directory and wish to reset the root and user passwords, and to create a database on which the user can fully modify, perform the following steps.

First create a `passwordreset.sql` file:

```
CREATE USER IF NOT EXISTS root@localhost IDENTIFIED BY 'thisismyrootpassword';
SET PASSWORD FOR root@localhost = PASSWORD('thisismyrootpassword');
GRANT ALL ON *.* TO root@localhost WITH GRANT OPTION;
CREATE USER IF NOT EXISTS root@'%' IDENTIFIED BY 'thisismyrootpassword';
SET PASSWORD FOR root@'%' = PASSWORD('thisismyrootpassword');
GRANT ALL ON *.* TO root@'%' WITH GRANT OPTION;
CREATE USER IF NOT EXISTS myuser@'%' IDENTIFIED BY 'thisismyuserpassword';
SET PASSWORD FOR myuser@'%' = PASSWORD('thisismyuserpassword');
CREATE DATABASE IF NOT EXISTS databasename;
GRANT ALL ON databasename.* TO myuser@'%';
```

Adjust `myuser`, `databasename` and passwords as needed.

Then:

```
$ docker run --rm -v /my/own/datadir:/var/lib/mysql -v /my/own/passwordreset.sql:/passwordreset.sql:z mariadb:latest --in
```

On restarting the MariaDB container on this `/my/own/datadir`, the `root` and `myuser` passwords will be reset.

How to install MariaDB plugins

MariaDB has many plugins, most are not enabled by default, some are in the mariadb container, others need to be installed from additional packages.

The following methods summarize the MariaDB Blog article - Installing plugins in the MariaDB Docker Library Container on this topic.

Which plugins does the container contain?

To see which plugins are available in the mariadb:

```
$ docker run --rm mariadb:latest ls -C /usr/lib/mysql/plugin
```

Enabling a plugin using flags

Using the `--plugin-load-add` flag with the plugin name (can be repeated), the plugins will be loaded and ready when the container is started:

For example enable the `simple\_password\_check` plugin:

```
$ docker run --name some-mariadb -e MARIADB_ROOT_PASSWORD=my-secret-pw --network=host -d mariadb:latest --plugin-load-add
```

Enabling a plugin in the configuration files

`plugin-load-add` can be used as a configuration option to load plugins. The example below load the FederatedX Storage Engine.

```
$ printf "[mariadb]\nplugin-load-add=ha_federatedx\n" > /my/custom/federatedx.conf
$ docker run --name some-mariadb -v /my/custom:/etc/mysql/conf.d -e MARIADB_ROOT_PASSWORD=my-secret-pw -d mariadb:latest
```

Install a plugin using SQL in /docker-entrypoint-initdb.d

`INSTALL SONAME` can be used to install a plugin as part of the

Overview

What is a Container

Product Overview

Product Offerings

Docker Desktop

Docker Hub

Features

Container Runtime

Developer Tools

Docker App

Kubernetes

Getting Started

Play with Docker

Community

Open Source

Docs

Hub Release Notes

About Us

Resources

Blog

Customers

Partners

Newsroom

Events and Webinars

Careers

Contact Us