Full Description

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 online shops 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 http://www.mysql.com.

MySQL Server Docker Images

These are optimized MySQL Server Docker images, created and maintained by the MySQL team at Oracle. The available versions are:

```
MySQL Server 5.5 (tag: 5.5)
MySQL Server 5.6 (tag: 5.6)
MySQL Server 5.7, the latest GA version (tag: 5.7 or latest)
```

Images are updated when new MySQL Server maintenance releases and development milestones are published.

We also publish experimental early previews of MySQL Server from time to time. Please visit the MySQL Labs Docker image page to see what is available.

How to Use the MySQL Images

Start a MySQL Server Instance

Start a MySQL instance as follows (but make sure you also read the sections *Secure Container Startup* and *Where to Store Data* below):

```
docker run --name my-container-name -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql/mysql-server:tag
```

... where `my-container-name` 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, or look at the full list of 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 app-container-name --link my-container-name:mysql -d app-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:

```
docker run -it --link my-container-name:mysql --rm mysql/mysql-server:tag sh -c 'exec mysql -
h"$MYSQL_PORT_3306_TCP_ADDR" -P"$MYSQL_PORT_3306_TCP_PORT" -uroot -p"$MYSQL_ENV_MYSQL_ROOT_PASSWORD"'
```

... where `my-container-name` is the name of your original MySQL Server container.

More information about the MySQL command line client can be found in the MySQL reference documentation at http://dev.mysql.com/doc/refman/en/

Container Shell Access and Viewing MySQL Log Files

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 my-container-name bash
```

The MySQL Server log is located at `/var/log/mysqld.log` inside the container, and the following command line from a shell inside the container will let you inspect it:

```
more /var/log/mysqld.log
```

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.

Most of the variables listed below are optional, but one of the variables `MYSQL_ROOT_PASSWORD`, `MYSQL_ALLOW_EMPTY_PASSWORD`, `MYSQL_RANDOM_ROOT_PASSWORD` must be given.

`MYSQL_ROOT_PASSWORD`
This variable specifies a password that will be set for the MySQL root superuser account. In the above example, it was set `to my-secret-pw`. **NOTE:** Setting the MySQL root user password on the command line is insecure. See the section *Secure Container Startup* below for an alternative.

`MYSQL_RANDOM_ROOT_PASSWORD`
When this variable is set to `yes`, a random password for the server's root user will be generated. The password will be printed to stdout in the container,

and it can be obtained by using the command `docker logs my-container-name`.

`MYSQL_ONETIME_PASSWORD`
This variable is optional. When set to `yes`, the root user's password will be set as expired, and must be changed before MySQL can be used normally. This is only supported by MySQL 5.6 or newer.

`MYSQL_DATABASE`
This variable is optional. It 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 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 set by either of the mechanisms (given or generated) discussed above.

`MYSQL_ALLOW_EMPTY_PASSWORD`
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.

Notes, Tips, Gotchas

Secure Container Startup

In many use cases, employing the `MYSQL_ROOT_PASSWORD` variable to specify the MySQL root user password on initial container startup is insecure. Instead, to keep your setup as secure as possible, we strongly recommend using the `MYSQL_RANDOM_ROOT_PASSWORD` option. To further secure your instance, we also recommend using the `MYSQL_ONETIME_PASSWORD` variable if you use MySQL version 5.6 or higher.

This is the full procedure:

```
docker run --name my-container-name -e MYSQL_RANDOM_ROOT_PASSWORD=yes -e MYSQL_ONETIME_PASSWORD=yes -d
mysql/mysql-server:tag
docker logs my-container-name
```

Look for the "GENERATED ROOT PASSWORD" line in the output.

If you also set the `MYSQL_ONETIME_PASSWORD` variable, you must now start a bash shell inside the container in order to set a new root password:

```
docker exec -it my-container-name bash
```

Start the MySQL command line client and log in using the randomly set root password:

```
mysql -u root -p
```

And finally, on the mysql client command line, set a new, secure root password for MySQL:

```
ALTER USER root IDENTIFIED BY 'my-secret-pw';
```

Where to Store Data

There are basically two ways to store data used by applications that run in Docker containers. We encourage users of MySQL with Docker 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 blog 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 my-container-name -v /my/own/datadir:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql/mysql-server: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 systems with SELinux enabled may experience problems 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
```

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 `docker run` command line; it will in any case be ignored, and the pre-existing database will not be changed in any way.

Port forwarding

Docker allows mapping of ports on the container to ports on the host system by using the -p option. If you start the container as follows, you can connect to the database by connecting your client to a port on the host machine, in this example port 6603:

```
docker run --name my-container-name -p 6603:3306 -d mysql/mysql-server
```

Passing options to the server

You can pass arbitrary command line options to the MySQL server by appending them to the `run command`:

```
docker run --name my-container-name -d mysql/mysql-server --option1=value --option2=value
```

In this case, the values of option1 and option2 will be passed directly to the server when it is started. The following command will for instance start your container with UTF-8 as the default setting for character set and collation for all databases in MySQL:

```
docker run --name my-container-name -d mysql/mysql-server --character-set-server=utf8 --collation-
server=utf8_general_ci
```

Using a Custom MySQL Config File

The MySQL startup configuration in these Docker images is specified in the file `/etc/my.cnf`. If you want to customize this configuration for your own purposes, you can create your alternative configuration file in a directory on the host machine and then mount this file in the appropriate location inside the MySQL container, effectively replacing the standard configuration file.

If you want to base your changes on the standard configuration file, start your MySQL container in the standard way described above, then do:

```
docker exec -it my-container-name cat /etc/my.cnf > /my/custom/config-file
```

... where ´/my/custom/config-file´ is the path and name of the new configuration file. Then start a new MySQL container like this:

```
docker run --name my-new-container-name -v /my/custom/config-file:/etc/my.cnf -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql/mysql-server:tag
```

This will start a new MySQL container ´my-new-container-name´ where the MySQL instance uses the startup options specified in ´/my/custom/config-file´.

Note that users on systems where SELinux is enabled may experience problems 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/config-file
```

Docker Optimized MySQL Install

These Docker images are optimized for size, which means that we have reduced the contents to what is expected to be relevant for a large majority of users who run Docker based MySQL instances. The key differences compared to a default MySQL install are:

- All binaries are stripped, non-debug only
- Included binaries are limited to:

```
    /usr/bin/my_print_defaults
    /usr/bin/mysql
    /usr/bin/mysql_config
    /usr/bin/mysql_install_db
    /usr/bin/mysql_tzinfo_to_sql
    /usr/bin/mysql_upgrade
    /usr/bin/mysqldump
    /usr/sbin/mysqld
```

Supported Docker Versions

These images are officially supported by the MySQL team on Docker version 1.9. Support for older versions (down to 1.0) is provided on a best-effort basis, but we strongly recommend running on the most recent version, since that is assumed for parts of the documentation above.

User Feedback

We welcome your feedback! For general comments or discussion, please drop us a line in the Comments section below. For bugs and issues, please submit a bug report at http://bugs.mysql.com under the category "MySQL Package Repos and Docker Images".

Docker Pull Command