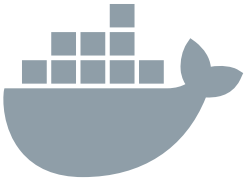

mongo

Official Image ☆

MongoDB document databases provide high availability and easy scalability.

1B+

Container Windows Linux IBM Z x86-64 ARM 64 Databases Official Image

Copy and paste to pull this image

```
docker pull mongo
```

View Available Tags

Description Reviews Tags

Quick reference

- **Maintained by**: the Docker Community
- **Where to get help**: the Docker Community Forums, the Docker Community Slack, or Stack Overflow

Supported tags and respective `Dockerfile` links

Note: the description for this image is longer than the Hub length limit of 25000, so the "Supported tags" list has been trimmed to compensate. See docker/hub-beta-feedback#238 for more information.

- See "Supported tags and respective `Dockerfile` links" at https://github.com/docker-library/docs/tree/master/mongo/README.md

Quick reference (cont.)

- **Where to file issues**: https://github.com/docker-library/mongo/issues
- **Supported architectures**: (more info) `amd64`, `arm64v8`, `windows-amd64`
- **Published image artifact details**: repo-info repo's `repos/mongo/` directory (history) (image metadata, transfer size, etc)
- **Image updates**: official-images repo's `library/mongo` label
  official-images repo's `library/mongo` file (history)
- **Source of this description**: docs repo's `mongo/` directory (history)

What is MongoDB?

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata. MongoDB is developed by MongoDB Inc., and is published under a combination of the Server Side Public License and the Apache License.

First developed by the software company 10gen (now MongoDB Inc.) in October 2007 as a component of a planned platform as a service product, the company shifted to an open source development model in 2009, with 10gen offering commercial support and other services. Since then, MongoDB has been adopted as backend software by a number of major websites and services, including MetLife, Barclays, ADP, UPS, Viacom, and the New York Times, among others. MongoDB is the most popular NoSQL database system.

> wikipedia.org/wiki/MongoDB

How to use this image

Start a `mongo` server instance

```
$ docker run --name some-mongo -d mongo:tag
```

... where `some-mongo` is the name you want to assign to your container and `tag` is the tag specifying the MongoDB version you want. See the list above for relevant tags.

Connect to MongoDB from another Docker container

The MongoDB server in the image listens on the standard MongoDB port, `27017`, so connecting via Docker networks will be the same as connecting to a remote `mongod`. The following example starts another MongoDB container instance and runs the `mongo` command line client against the original MongoDB container from the example above, allowing you to execute MongoDB statements against your database instance:

```
$ docker run -it --network some-network --rm mongo mongo --host some-mongo test
```

... where `some-mongo` is the name of your original `mongo` container.

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

Example `stack.yml` for `mongo`:

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

services:

  mongo:
    image: mongo
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example

  mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: example
      ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
```

Try in PWD

Run `docker stack deploy -c stack.yml mongo` (or `docker-compose -f stack.yml up`), wait for it to initialize completely, and visit `http://swarm-ip:8081`, `http://localhost:8081`, or `http://host-ip:8081` (as appropriate).

Container shell access and viewing MongoDB 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 `mongo` container:

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

The MongoDB Server log is available through Docker's container log:

```
$ docker logs some-mongo
```

Configuration

See the MongoDB manual for information on using and configuring MongoDB for things like replica sets and sharding.

Customize configuration without configuration file

Most MongoDB configuration can be set through flags to `mongod`. The entrypoint of the image is created to pass its arguments along to `mongod`. See below an example of setting MongoDB to use a different threading and execution model via `docker run`.

```
$ docker run --name some-mongo -d mongo --serviceExecutor adaptive
```

And here is the same with a `docker-compose.yml` file

```
version: '3.1'
services:
  mongo:
    image: mongo
    command: --serviceExecutor adaptive
```

To see the full list of possible options, check the MongoDB manual on `mongod` or check the `--help` output of `mongod`:

```
$ docker run -it --rm mongo --help
```

Setting WiredTiger cache size limits

By default Mongo will set the `wiredTigerCacheSizeGB` to a value proportional to the host's total memory regardless of memory limits you may have imposed on the container. In such an instance you will want to set the cache size to something appropriate, taking into account any other processes you may be running in the container which would also utilize memory.

Taking the examples above you can configure the cache size to use 1.5GB as:

```
$ docker run --name some-mongo -d mongo --wiredTigerCacheSizeGB 1.5
```

See the upstream "WiredTiger Options" documentation for more details.

Using a custom MongoDB configuration file

For a more complicated configuration setup, you can still use the MongoDB configuration file. `mongod` does not read a configuration file by default, so the `--config` option with the path to the configuration file needs to be specified. Create a custom configuration file and put it in the container by either creating a custom Dockerfile `FROM mongo` or mounting it from the host machine to the container. See the MongoDB manual for a full list of configuration file options.

For example, `/my/custom/mongod.conf` is the path to the custom configuration file. Then start the MongoDB container like the following:

```
$ docker run --name some-mongo -v /my/custom:/etc/mongo -d mongo --config /etc/mongo/mongod.conf
```

Environment Variables

When you start the `mongo` image, you can adjust the initialization of the MongoDB 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.

`MONGO_INITDB_ROOT_USERNAME`, `MONGO_INITDB_ROOT_PASSWORD`

These variables, used in conjunction, create a new user and set that user's password. This user is created in the `admin` authentication database and given the role of `root`, which is a "superuser" role.

The following is an example of using these two variables to create a MongoDB instance and then using the `mongo` cli to connect against the `admin` authentication database.

```
$ docker run -d --network some-network --name some-mongo \
    -e MONGO_INITDB_ROOT_USERNAME=mongoadmin \
    -e MONGO_INITDB_ROOT_PASSWORD=secret \
    mongo

$ docker run -it --rm --network some-network mongo \
    mongo --host some-mongo \
        -u mongoadmin \
        -p secret \
        --authenticationDatabase admin \
        some-db
> db.getName();
some-db
```

Both variables are required for a user to be created. If both are present then MongoDB will start with authentication enabled (`mongod --auth`).

Authentication in MongoDB is fairly complex, so more complex user setup is explicitly left to the user via `/docker-entrypoint-initdb.d/` (see the *Initializing a fresh instance* and *Authentication* sections below for more details).

`MONGO_INITDB_DATABASE`

This variable allows you to specify the name of a database to be used for creation scripts in `/docker-entrypoint-initdb.d/*.js` (see *Initializing a fresh instance* below). MongoDB is fundamentally designed for "create on first use", so if you do not insert data with your JavaScript files, then no database is created.

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-mongo -e MONGO_INITDB_ROOT_PASSWORD_FILE=/run/secrets/mongo-root -d mongo
```

Currently, this is only supported for `MONGO_INITDB_ROOT_USERNAME` and `MONGO_INITDB_ROOT_PASSWORD` .

Initializing a fresh instance

When a container is started for the first time it will execute files with extensions `.sh` and `.js` that are found in `/docker-entrypoint-initdb.d`. Files will be executed in alphabetical order. `.js` files will be executed by `mongo` using the database specified by the `MONGO_INITDB_DATABASE` variable, if it is present, or `test` otherwise. You may also switch databases within the `.js` script.

Authentication

As noted above, authentication in MongoDB is fairly complex (although disabled by default). For details about how MongoDB handles authentication, please see the relevant upstream documentation:

- `mongod --auth`
- Security > Authentication
- Security > Role-Based Access Control
- Security > Role-Based Access Control > Built-In Roles
- Security > Enable Auth (tutorial)

In addition to the `/docker-entrypoint-initdb.d` behavior documented above (which is a simple way to configure users for authentication for less complicated deployments), this image also supports `MONGO_INITDB_ROOT_USERNAME` and `MONGO_INITDB_ROOT_PASSWORD` for creating a simple user with the role `root` in the `admin` authentication database, as described in the *Environment Variables* section above.

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 `mongo` 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.

WARNING (Windows & OS X): When running the Linux-based MongoDB images on Windows and OS X, the file systems used to share between the host system and the Docker container is not compatible with the memory mapped files used by MongoDB (docs.mongodb.org and related jira.mongodb.org bug). This means that it is not possible to run a MongoDB container with the data directory mapped to the host. To persist data between container restarts, we recommend using a local named volume instead (see `docker volume create`). Alternatively you can use the Windows-based images on Windows.

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 `mongo` container like this:

```
$ docker run --name some-mongo -v /my/own/datadir:/data/db -d mongo
```

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

This image also defines a volume for `/data/configdb` for use with `--configsvr` (see docs.mongodb.com for more details).

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 `mongod` 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-mongo sh -c 'exec mongodump -d <database_name> --archive' > /some/path/on/your/l
```

Image Variants

The `mongo` images come in many flavors, each designed for a specific use case.

`mongo:<version>`

This is the defacto image. If you are unsure about what your needs are, you probably want to use this one. It is designed to be used both as a throw away container (mount your source code and start the container to start your app), as well as the base to build other images off of.

Some of these tags may have names like bionic, focal, or xenial in them. These are the suite code names for releases of Ubuntu and indicate which release the image is based on. If your image needs to install any additional packages beyond what comes with the image, you'll likely want to specify one of these explicitly to minimize breakage when there are new releases of Ubuntu.

`mongo:<version>-windowsservercore`

This image is based on Windows Server Core (`microsoft/windowsservercore`). As such, it only works in places which that image does, such as Windows 10 Professional/Enterprise (Anniversary Edition) or Windows Server 2016.

For information about how to get Docker running on Windows, please see the relevant "Quick Start" guide provided by Microsoft:

- Windows Server Quick Start
- Windows 10 Quick Start

License

View license information for the software contained in this image.

It is relevant to note the change from AGPL to SSPLv1 for all versions after October 16, 2018.

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 `mongo/` 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.