**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 container linking or 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 --link some-mongo:mongo --rm mongo mongo --host mongo test

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

**... via** [**docker stack deploy**](https://docs.docker.com/engine/reference/commandline/stack_deploy/) **or** [**docker-compose**](https://github.com/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

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](https://docs.mongodb.com/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 [smaller default file size](https://docs.mongodb.com/manual/reference/program/mongod/#cmdoption-mongod-smallfiles) via docker run.

$ docker run --name some-mongo -d mongo --smallfiles

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

version: '3.1'

services:

mongo:

image: mongo

command: --smallfiles

To see the full list of possible options, check the MongoDB manual on [mongod](https://docs.mongodb.com/manual/reference/program/mongod/) or check the --help output of mongod:

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

**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](https://docs.mongodb.com/manual/reference/configuration-options/) 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. 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 *Initializing a fresh instance* below). 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 --name some-mongo -e MONGO\_INITDB\_ROOT\_USERNAME=mongoadmin -e MONGO\_INITDB\_ROOT\_PASSWORD=secret mongo

$ docker run -it --rm --link some-mongo:mongo mongo mongo --host mongo -u mongoadmin -p secret --authenticationDatabase admin some-db

> db.getName();

some-db

If you do not provide these two variables or do not set the --auth flag with your own custom user setup, then MongoDB will not require authentication. For more details about the functionality described here, please see the sections in the official documentation which describe [authentication](https://docs.mongodb.com/manual/core/authentication/) and [authorization](https://docs.mongodb.com/manual/core/authorization/) in more detail.

**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.

**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](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.

**WARNING (Windows & OS X)**: The default Docker setup on Windows and OS X uses a VirtualBox VM to host the Docker daemon. Unfortunately, the mechanism VirtualBox uses to share folders between the host system and the Docker container is not compatible with the memory mapped files used by MongoDB (see [vbox bug](https://www.virtualbox.org/ticket/819), [docs.mongodb.org](https://docs.mongodb.com/manual/administration/production-notes/#fsync-on-directories) and related [jira.mongodb.org](https://jira.mongodb.org/browse/SERVER-8600) bug). This means that it is not possible to run a MongoDB container with the data directory mapped to the host.

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:
3. $ 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](https://docs.mongodb.com/v3.4/reference/program/mongod/#cmdoption-configsvr)