

# HOW TO RUN MONGODB AS A DOCKER CONTAINER



MongoDB is among the most popular [NoSQL databases](#) today. And containers offer easy app usage and scalability. In this article, I'll show you how to:

- Configure MongoDB as a container in Docker
- Set up the Docker platform with docker-compose
- Create a docker-compose file to create the MongoDB container
- And more

The last part of this tutorial will look at advanced configurations. These can give you a glimpse of the extensibility of a containerized project. So, we'll create a self-containing project with a MongoDB instance and Mongo Express web interface on a dedicated network and docker volume to maximize the portability of the project.

Let's get started.

*(This article is part of our [MongoDB Guide](#). Use the right-hand menu to navigate.)*

## Docker containers & MongoDB

[Docker](#) is a tool to create, deploy, and run applications using containers easily. A [container](#) is a standard unit of software that can be used to package applications and all the dependencies to a single package. These containers can be run on any server platform regardless of the underlying configuration or hardware structure.

Docker can be used to run MongoDB instances. Setting up MongoDB as a container allows the user to create a portable and extensible NoSQL database. A containerized MongoDB instance behaves

exactly like a non-containerized MongoDB instance without having to worry about the underlying configuration.

---

**Interested in Enterprise DevOps? [Learn more about DevOps Solutions and Tools with BMC.](#) >**

---

## Installing Docker

In this section, we'll set up a simple Docker installation to run containers on a Ubuntu-based server. We can get the Docker installation packages from [the official](#) Docker repository. Here are the installation steps:

1. Update existing packages.

```
sudo apt update && sudo apt upgrade -y
```

```
barry@ubser001 ~  
$ sudo apt update && sudo apt upgrade -y  
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease  
Hit:2 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease  
Hit:3 https://download.docker.com/linux/ubuntu focal InRelease  
Hit:4 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease  
Hit:5 http://us.archive.ubuntu.com/ubuntu focal-security InRelease  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
All packages are up to date.  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
Calculating upgrade... Done  
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

2. Install prerequisite packages.

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

```
barry@ubser001 ~  
$ sudo apt install apt-transport-https ca-certificates curl software-properties-common  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  python3-certifi python3-requests python3-requests-unixsocket  
Suggested packages:  
  python3-cryptography python3-openssl python3-socks  
The following NEW packages will be installed:  
  apt-transport-https ca-certificates curl python3-certifi python3-requests  
  python3-requests-unixsocket software-properties-common  
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.  
Need to get 529 kB of archives.  
After this operation, 1,774 kB of additional disk space will be used.  
Do you want to continue? [Y/n] Y
```

3. Add the GPG key from the official Docker repository.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
barry@ubser001 ~  
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
OK
```

4. Add the official docker repository to APT sources.

```
sudo add-apt-repository
```

```
"deb https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
barry@ubser001 ~  
$ sudo add-apt-repository \  
"deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"  
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease  
Hit:2 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease  
Hit:3 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease  
Get:4 https://download.docker.com/linux/ubuntu focal InRelease [36.2 kB]  
Hit:5 http://us.archive.ubuntu.com/ubuntu focal-security InRelease  
Get:6 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [4,226 B]  
Fetched 40.4 kB in 2s (26.0 kB/s)  
Reading package lists... Done
```

5. Update the Ubuntu package list.

```
sudo apt update
```

```
barry@ubser001 ~  
$ sudo apt update  
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease  
Hit:2 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease  
Hit:3 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease  
Hit:4 https://download.docker.com/linux/ubuntu focal InRelease  
Hit:5 http://us.archive.ubuntu.com/ubuntu focal-security InRelease  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
All packages are up to date.
```

6. Verify the docker repository.

```
apt-cache policy docker-ce
```

```
barry@ubser001 ~  
$ apt-cache policy docker-ce  
docker-ce:  
  Installed: (none)  
  Candidate: 5:19.03.14~3-0~ubuntu-focal  
  Version table:  
   5:19.03.14~3-0~ubuntu-focal 500  
     500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
   5:19.03.13~3-0~ubuntu-focal 500  
     500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
   5:19.03.12~3-0~ubuntu-focal 500  
     500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
   5:19.03.11~3-0~ubuntu-focal 500  
     500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
   5:19.03.10~3-0~ubuntu-focal 500  
     500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages  
   5:19.03.9~3-0~ubuntu-focal 500  
     500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
```

7. Install the Docker community edition.

```
sudo apt install docker-ce
```

```
barry@ubser001 ~  
$ sudo apt install docker-ce  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  aufs-tools cgroupfs-mount containerd.io docker-ce-cli pigz  
The following NEW packages will be installed:  
  aufs-tools cgroupfs-mount containerd.io docker-ce docker-ce-cli pigz  
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.  
Need to get 91.5 MB of archives.  
After this operation, 413 MB of additional disk space will be used.  
Do you want to continue? [Y/n] Y
```

8. Check the status of the installation with the following command. If the **service** status returns active (running), Docker is successfully installed and active on the system.

```
sudo systemctl status docker
```

```
barry@ubser001 ~  
$ sudo systemctl status docker  
● docker.service - Docker Application Container Engine  
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)  
   Active: active (running) since Sun 2020-12-06 08:55:33 UTC; 44s ago  
TriggeredBy: ● docker.socket  
   Docs: https://docs.docker.com  
  Main PID: 41782 (dockerd)  
    Tasks: 10  
   Memory: 38.0M  
    CGroup: /system.slice/docker.service  
            └─41782 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock  
  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.311419692Z" level=warning msg=">  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.311581309Z" level=warning msg=">  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.311695934Z" level=warning msg=">  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.313060065Z" level=info msg="Loa>  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.424500856Z" level=info msg="Def>  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.473118189Z" level=info msg="Loa>  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.514913524Z" level=info msg="Doc>  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.515221329Z" level=info msg="Dae>  
Dec 06 08:55:33 ubser001 dockerd[41782]: time="2020-12-06T08:55:33.531208321Z" level=info msg="API>  
Dec 06 08:55:33 ubser001 systemd[1]: Started Docker Application Container Engine.  
lines 1-21/21 (END)
```

## Installing Docker Compose

We can use the command line interface (CLI) to create and manage Docker containers. However, the CLI can be tedious when dealing with multiple containers and configurations.

Docker Compose allows users to take multiple containers and integrate them into a single application. Docker Compose uses the YAML format to create the compose files that can be easily executed using docker-compose up or down commands that will create or remove all the containers and configurations within a compose file, respectively.

Let's install Docker Compose on the Ubuntu server.

1. Install the current stable release of Docker Compose.

```
sudo curl -L
```



```
"https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
barry@ubser001 ~  
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 651 100 651 0 0 1190 0 --:--:-- --:--:-- --:--:-- 1190  
100 11.6M 100 11.6M 0 0 2297k 0 0:00:05 0:00:05 --:--:-- 2645k
```

2. Apply executable permissions for the downloaded binary.

```
>sudo chmod +x /usr/local/bin/docker-compose
```

```
barry@ubser001 ~  
$ sudo chmod +x /usr/local/bin/docker-compose
```

3. Verify the Docker Compose installation.

```
docker-compose --version
```

```
barry@ubser001 ~  
$ docker-compose --version  
docker-compose version 1.27.4, build 40524192
```

## Setting up a MongoDB container

This section will cover how to set up a MongoDB container using a Docker Compose file.

Before creating the compose file, let's search for the official MongoDB container image using the **search** command.

```
sudo docker search mongodb
```

```
barry@ubser001 ~  
$ sudo docker search mongodb  
NAME DESCRIPTION STARS OFFICIAL  
mongo MongoDB document databases provide high avail... 7363 [OK]  
mongo-express Web-based MongoDB admin interface, written w... 823 [OK]
```

The search results show us that an official MongoDB container image called mongo exists in the docker container registry.

By default, the MongoDB container stores the databases within the /data/db directory within the container.

Next, we need to create a directory called "mongodb" to hold the docker-compose file. We will create another directory called "database" inside the "mongodb" directory to map to the database location of the container. This will enable local access to the database. We use the **-pv** operator to create those parent folders.

```
>mkdir -pv mongodb/database
```

```
barry@ubser001 ~  
$ mkdir -pv mongodb/database  
mkdir: created directory 'mongodb'  
mkdir: created directory 'mongodb/database'
```

The following docker-compose.yml file will be created within the "mongodb" directory to construct

the MongoDB container.

docker-compose.yml

```
version: "3.8"
services:
  mongodb:
    image : mongo
    container_name: mongodb
    environment:
      - PUID=1000
      - PGID=1000
    volumes:
      - /home/barry/mongodb/database:/data/db
    ports:
      - 27017:27017
    restart: unless-stopped
```

We used version 3.8 to create the above compose file. The compose file version directly correlates to:

- Which options are available within the compose file
- The minimum supported Docker engine version

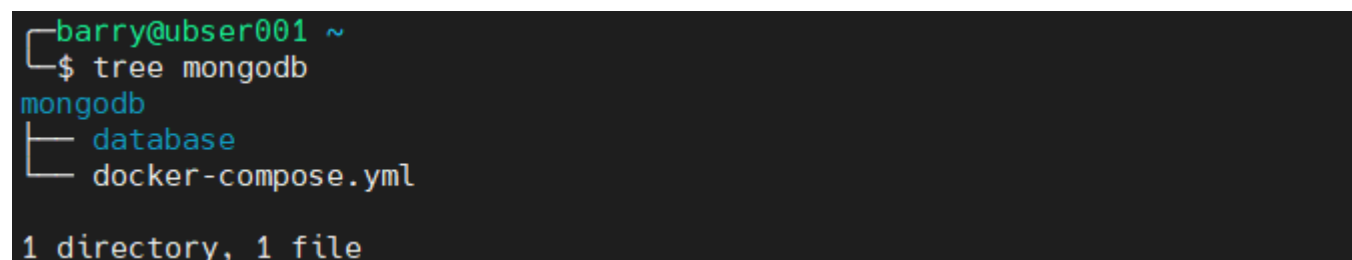
In this case, It's Docker Engine 19.03.0 or newer.

In the compose file, we have created a service called mongodb using the Docker image mongo. We have named the container "mongodb" and mapped the database folder within the container to the local database folder (/home/barry/mongodb/database). These kinds of mappings are known as **bind-mount volumes**.

The environment variables are used to define the "user" and "group" of the container. Finally, we mapped the local port 27017 to internal port 27017. Then the restart policy is set to restart unless stopped by the user.

Here's the file structure of the project:

tree mongodb



```
barry@ubser001 ~
$ tree mongodb
mongodb
├── database
└── docker-compose.yml

1 directory, 1 file
```

Go to the "mongodb" folder and run the **docker-compose up** command to start the MongoDB container. The **-d** operator runs the detached container as a background process.

```
sudo docker-compose up -d
```

```

barry@ubser001 ~/mongodb
$ sudo docker-compose up -d
[sudo] password for barry:
Creating network "mongodb_default" with the default driver
Pulling mongo (mongo:)...
latest: Pulling from library/mongo
f22ccc0b8772: Pull complete
3cf8fb62ba5f: Pull complete
e80c964ece6a: Pull complete
329e632c35b3: Pull complete
3e1bd1325a3d: Pull complete
4aa6e3d64a4a: Pull complete
035bca87b778: Pull complete
874e4e43cb00: Pull complete
08cb97662b8b: Pull complete
f623ce2ba1e1: Pull complete
f100ac278196: Pull complete
461b064aece5: Pull complete
Digest: sha256:00878f3d8e0a61997f2ea67351934b815a77c5ff8985df3ec041bca1c88258f4
Status: Downloaded newer image for mongo:latest
Creating mongodb ... done

```

The **up** command will pull the mongo image from the docker registry and create the container using the given parameters in the docker-compose.yml file.

Let's verify if the container is running and the local folder is populated with the following commands. The **-a** operator will display all the containers within the system regardless of their status.

```
sudo docker ps -a
```

```

barry@ubser001 ~
$ sudo docker ps -a

```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS
f38465a007ea	mongo		"docker-entrypoint.s..."	8 minutes ago	Up 8 minutes
0.0.0.0:27017->27017/tcp		mongodb			

```
sudo tree mongodb
```

```
barry@ubser001 ~  
$ sudo tree mongodb  
mongodb  
├── database  
│   ├── collection-0--3194752319981630697.wt  
│   ├── collection-2--3194752319981630697.wt  
│   ├── collection-4--3194752319981630697.wt  
│   ├── diagnostic.data  
│   │   ├── metrics.2020-12-06T11-42-32Z-00000  
│   │   └── metrics.interim  
│   ├── index-1--3194752319981630697.wt  
│   ├── index-3--3194752319981630697.wt  
│   ├── index-5--3194752319981630697.wt  
│   ├── index-6--3194752319981630697.wt  
│   └── journal  
│       ├── WiredTigerLog.00000000001  
│       ├── WiredTigerPrelog.00000000001  
│       └── WiredTigerPrelog.00000000002  
├── _mdb_catalog.wt  
├── mongod.lock  
├── sizeStorer.wt  
├── storage.bson  
├── WiredTiger  
├── WiredTigerHS.wt  
├── WiredTiger.lock  
├── WiredTiger.turtle  
├── WiredTiger.wt  
└── docker-compose.yml  
  
3 directories, 22 files
```

## Interacting with the MongoDB container

Using the docker **exec** command, we can access the terminal of the MongoDB container. As the container runs in a detached mode, we will use the Docker interactive terminal to establish the connection.

```
sudo docker exec -it mongodb bash
```



```

barry@ubser001 ~
$ sudo docker exec -it mongodb bash
root@f38465a007ea:/# mongo
MongoDB shell version v4.4.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("5fb25459-bbcd-4c89-97c3-cd4d5609a2ff") }
MongoDB server version: 4.4.2
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2020-12-06T11:42:32.453+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
  2020-12-06T11:42:32.964+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>

```

In the bash terminal of the container, we call the mongo command to access MongoDB. We will create a database called "food" and a collection called "fruits", along with three documents.

1. Switch the database.

use food

2. Create collection.

```
db.createCollection("fruits")
```

3. Insert documents

```
db.fruits.insertMany()
```

```

> use food
switched to db food
> db.createCollection("fruits")
{ "ok" : 1 }
> db.fruits.insertMany([
... {name: "apple", origin: "usa", price: 5},
... {name: "orange", origin: "italy", price: 3},
... {name: "mango", origin: "malaysia", price: 3}
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5fccccd5c79274820ab6c3b54"),
    ObjectId("5fccccd5c79274820ab6c3b55"),
    ObjectId("5fccccd5c79274820ab6c3b56")
  ]
}
> █

```

Se

arch for the documents using the **find** command:

```
db.fruits.find().pretty()
```

```

> db.fruits.find().pretty()
{
  "_id" : ObjectId("5fccccd5c79274820ab6c3b54"),
  "name" : "apple",
  "origin" : "usa",
  "price" : 5
}
{
  "_id" : ObjectId("5fccccd5c79274820ab6c3b55"),
  "name" : "orange",
  "origin" : "italy",
  "price" : 3
}
{
  "_id" : ObjectId("5fccccd5c79274820ab6c3b56"),
  "name" : "mango",
  "origin" : "malaysia",
  "price" : 3
}
> █

```

The MongoDB container will act like any normal MongoDB installation without any concerns about the underlying software and hardware configuration. Using the **exit** command, we can exit both the MongoDB shell and container shell.

```

> exit
bye
root@f38465a007ea:/# exit
exit
barry@ubser001 ~
$ █

```

## External connections to MongoDB container

While creating the MongoDB container, we mapped the internal MongoDB port to the

corresponding port in the server, exposing the MongoDB container to external networks.

The following example demonstrates how we can connect to the container from an external endpoint by simply pointing the **mongo** command to the appropriate server and port.

**mongo 10.10.10.60:27017**

```
barry@mongodb04:~$ mongo 10.10.10.60:27017
MongoDB shell version v4.4.2
connecting to: mongodb://10.10.10.60:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0912297b-c0d7-4219-b76d-eb33689127ae") }
MongoDB server version: 4.4.2
---
The server generated these startup warnings when booting:
  2020-12-06T11:42:32.453+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
  2020-12-06T11:42:32.964+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> █
```

The **find** command will search for the fruits collection and its documents to verify that we are connected to the MongoDB container.

show databases

use food

show collections

db.fruits.find().pretty()

## Data resilience

We've mapped the database to a local folder. As a result of that, even if the container is removed, the saved data in the local folder can be used to recreate a new MongoDB container.

Let's test that. We'll:

- Remove the container using the **docker-compose down**
- Delete the associated images.
- Recreate a new MongoDB database using the compose file and local database files.

```
> show databases
admin    0.000GB
config  0.000GB
food     0.000GB
local    0.000GB
> use food
barry@subser001 ~/mongodb
$ sudo docker-compose down
[sudo] password for barry:
Stopping mongodb ... done
Removing mongodb ... done
Removing network mongodb_default
Remove the local mongo image.
}
{
  "price" : 5
}
{
  "name" : "orange",
  "origin" : "italy",
  "price" : 3
}
}
{
  "_id" : ObjectId("5fcccc5c79274820ab6c3b55"),
  "name" : "orange",
  "origin" : "italy",
  "price" : 3
}
}
{
  "_id" : ObjectId("5fcccc5c79274820ab6c3b56"),
  "name" : "mango",
  "origin" : "malaysia",
  "price" : 3
}
}
>
```

```

barry@ubser001 ~
$ sudo docker rmi mongo
Untagged: mongo:latest
Untagged: mongo@sha256:00878f3d8e0a61997f2ea67351934b815a77c5ff8985df3ec041bca1c88258f4
Deleted: sha256:60930105324282947c57d98729be68394062249ab347e4afc78e0cab9777a4eb
Deleted: sha256:d3c340b9fb3756edbf106bc38081447ee3a362ec2cef54d2cb9496310d211f55
Deleted: sha256:df411d930f08ff19c01fff7498a7c161ffe353027d1ee0be99aef4150ef2f237
Deleted: sha256:9da06454ee6eff2960465ba510f4180810bd95d22caa65d958923aeccaa12a12
Deleted: sha256:4fc8092849352849654c290a66285e0b989f01e73a95f869a8ee9bb2eadaebf6
Deleted: sha256:4545ce32b03c37edb33aedeac6ad411047ed6dd94650697e2465db1a86507f93
Deleted: sha256:9abb615f3c3c97191bf206d3228b5bd0978e370394e71984a0d183efb0e0bd9c
Deleted: sha256:eb6c9f1a2a53c149123b96a258dcd25c29b0b8c1709b8a326bc6b9ebaa43dec8
Deleted: sha256:1acc9a7abd496942e1469e91d2f7d2b7d9e0c696f2b392ef1e85d2eae9f91e96b
Deleted: sha256:2589aa72cefbc28ecbf0798bf2ca438af965daedbbb434e8f92d4bb2d689831
Deleted: sha256:9459b6a89846db0723e467610b841e6833dbb2aae6133319a91f2f70c388afac
Deleted: sha256:9a9311f7fcddf94f7476ce89f9703e8360e8cf347ef486a280735f5cf98888cd
Deleted: sha256:b43408d5f11b7b2faf048ae4eb25c296536c571fb2f937b4f1c3883386e93d64

```

Verify the local database files.

From the output below, we can identify that even though we removed the containers, the data mapped to a local directory did not get removed.

`sudo tree mongodb`

```

barry@ubser001 ~
$ sudo tree mongodb
mongodb
├── database
│   ├── collection-0--3194752319981630697.wt
│   ├── collection-11--3194752319981630697.wt
│   ├── collection-2--3194752319981630697.wt
│   ├── collection-4--3194752319981630697.wt
│   ├── diagnostic.data
│   │   └── metrics.2020-12-06T11-42-32Z-00000
│   ├── index-12--3194752319981630697.wt
│   ├── index-1--3194752319981630697.wt
│   ├── index-3--3194752319981630697.wt
│   ├── index-5--3194752319981630697.wt
│   ├── index-6--3194752319981630697.wt
│   └── journal
│       ├── WiredTigerLog.0000000001
│       ├── WiredTigerPrelog.0000000001
│       └── WiredTigerPrelog.0000000002
├── _mdb_catalog.wt
├── mongod.lock
├── sizeStorer.wt
├── storage.bson
├── WiredTiger
├── WiredTigerHS.wt
├── WiredTiger.lock
├── WiredTiger.turtle
├── WiredTiger.wt
└── docker-compose.yml

3 directories, 23 files

```

Recreate a new MongoDB container. Now, we will recreate the container using the original docker-compose.yml file. We execute the following command in the mongodb folder.

`sudo docker-compose up -d`



```
barry@ubser001 ~/mongodb
$ sudo docker-compose up -d
Creating network "mongodb_default" with the default driver
Pulling mongo (mongo:)...
latest: Pulling from library/mongo
f22ccc0b8772: Pull complete
3cf8fb62ba5f: Pull complete
e80c964ece6a: Pull complete
329e632c35b3: Pull complete
3e1bd1325a3d: Pull complete
4aa6e3d64a4a: Pull complete
035bca87b778: Pull complete
874e4e43cb00: Pull complete
08cb97662b8b: Pull complete
f623ce2ba1e1: Pull complete
f100ac278196: Pull complete
461b064aece5: Pull complete
Digest: sha256:00878f3d8e0a61997f2ea67351934b815a77c5ff8985df3ec041bca1c88258f4
Status: Downloaded newer image for mongo:latest
Creating mongodb ... done
```

Verify the Data in the MongoDB container. Let's now access the bash shell in the container and check for the "fruits" collections.

```
sudo docker exec -it mongodb bash
```

```
barry@ubser001 ~/mongodb
$ sudo docker exec -it mongodb bash
root@e1c300bfe465:/# mongo
MongoDB shell version v4.4.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("dd9841d4-bbcf-4e04-b25b-2ff8755d670f") }
MongoDB server version: 4.4.2
```

```
show databases
```

```
use food
```

```
db.fruits.find().pretty()
```

```
> show databases
admin    0.000GB
config   0.000GB
food     0.000GB
local    0.000GB
> use food
switched to db food
> db.fruits.find().pretty()
{
  "_id" : ObjectId("5fccccd5c79274820ab6c3b54"),
  "name" : "apple",
  "origin" : "usa",
  "price" : 5
}
{
  "_id" : ObjectId("5fccccd5c79274820ab6c3b55"),
  "name" : "orange",
  "origin" : "italy",
  "price" : 3
}
{
  "_id" : ObjectId("5fccccd5c79274820ab6c3b56"),
  "name" : "mango",
  "origin" : "malaysia",
  "price" : 3
}
> █
```

The result indicates that the new container was created with the local database information associated with the new container.

Additionally, we can simply move the container by moving the local folder structure to a new server and creating a container using the docker-compose.yml file. Docker volumes can be used instead of locally saving the data to increase the portability of the database.

---

**Take IT Service Management to the next level with [BMC Helix ITSM.>](#)**

---

## Container log files

Every container creates logs that can be used to monitor and debug itself. We can access the container logs using the **docker logs** command with the container name to be monitored.

```
sudo docker logs mongodb
```

```

barry@ubser001 ~
$ sudo docker logs mongodb
{"t":{"$date":"2020-12-06T13:15:27.707+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main",
"msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2020-12-06T13:15:27.709+00:00"},"s":"W", "c":"ASIO", "id":22601, "ctx":"main",
"msg":"No TransportLayer configured during NetworkInterface startup"}
{"t":{"$date":"2020-12-06T13:15:27.710+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"main",
"msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t":{"$date":"2020-12-06T13:15:27.711+00:00"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":"initandlisten", "msg":"MongoDB starting", "attr":{"pid":1, "port":27017, "dbPath":"/data/db", "architecture":"64-bit", "host":"e1c300bfe465"}}
{"t":{"$date":"2020-12-06T13:15:27.712+00:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten", "msg":"Build Info", "attr":{"buildInfo":{"version":"4.4.2", "gitVersion":"15e73dc5738d2278b688f8929aee605fe4279b0e", "openSSLVersion":"OpenSSL 1.1.1 11 Sep 2018", "modules":[], "allocator":"tcmalloc", "environment":{"distmod":"ubuntu1804", "distarch":"x86_64", "target_arch":"x86_64"}}}}
{"t":{"$date":"2020-12-06T13:15:27.712+00:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten", "msg":"Operating System", "attr":{"os":{"name":"Ubuntu", "version":"18.04"}}}

```

## Advanced container usage

In this section, we will create a secure MongoDB container that requires a username and password to access the database.

In earlier examples, we mapped the database data to a local folder. However, this is tedious and requires manual intervention when moving the Docker container. Using Docker volumes, we can create Docker native persistent volumes that can be easily transferred between Docker installations.

Although we can use the CLI to manipulate the MongoDB instance, a GUI would be a more convenient option to do that. Mongo Express is a web-based MongoDB administration interface that also can be run as a containerized application.

The docker-compose file comes in handy as a single YAML file that captures all the requirements.

docker-compose.yml

```

version: "3.8"
services:
  mongodb:
    image: mongo
    container_name: mongodb
    environment:
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=pass12345
    volumes:
      - mongodb-data:/data/db
    networks:
      - mongodb_network
    ports:
      - 27017:27017
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongo 10.10.10.60:27017/test --quiet
      interval: 30s
      timeout: 10s

```

```

retries: 3
restart: unless-stopped
mongo-express:
image: mongo-express
container_name: mongo-express
environment:
- ME_CONFIG_MONGODB_SERVER=mongodb
- ME_CONFIG_MONGODB_ENABLE_ADMIN=true
- ME_CONFIG_MONGODB_ADMINUSERNAME=root
- ME_CONFIG_MONGODB_ADMINPASSWORD=pass12345
- ME_CONFIG_BASICAUTH_USERNAME=admin
- ME_CONFIG_BASICAUTH_PASSWORD=admin123
volumes:
- mongodb-data
depends_on:
- mongodb
networks:
- mongodb_network
ports:
- 8081:8081
healthcheck:
test: wget --quiet --tries=3 --spider http://admin:admin123@10.10.10.60:8081
|| exit 1
interval: 30s
timeout: 10s
retries: 3
restart: unless-stopped
volumes:
mongodb-data:
name: mongodb-data
networks:
mongodb_network:
name: mongodb_network

```

Now, let's break down the compose file given above. First, we have created two services:

- mongodb
- mongo-express

## **mongodb service**

The root username and password of the mongodb container are configured using the following environment variables.

- MONGO\_INITDB\_ROOT\_USERNAME
- MONGO\_INITDB\_ROOT\_PASSWORD

The data volume is mapped to mongodb-data docker volume, and the network is defined as mongodb\_network while opening port 27017.

## mongo-express service

The environment variables of the mongo-express container are:

- ME\_CONFIG\_MONGODB\_SERVER - MongoDB service (mongodb)
- ME\_CONFIG\_MONGODB\_ENABLE\_ADMIN - Enable access to all databases as admin
- ME\_CONFIG\_MONGODB\_ADMINUSERNAME - Admin username of the MongoDB database
- ME\_CONFIG\_MONGODB\_ADMINPASSWORD - Admin password of the MongoDB database
- ME\_CONFIG\_BASICAUTH\_USERNAME - Mongo-Express web interface access username
- ME\_CONFIG\_BASICAUTH\_PASSWORD - Mongo-Express web interface access password

Additionally, we have configured the mongo-express service to depend on the mongodb service. The network is assigned the same mongodb\_network, and the volumes are mapped to mongodb-data volume. Then the port 8081 is exposed to allow access to the web interface.

Both services are monitored using Docker health checks. The mongodb service will ping the MongoDB database, while the mongo-express service will try to access the web page using the given credentials.

Finally, we have defined a volume called mongodb-data and a network called mongodb\_network for the project.

Start the Docker compose file.

```
sudo docker-compose up -d
```

```
barry@ubser001 ~/mongodb
$ sudo docker-compose up -d
Creating network "mongodb_network" with the default driver
Creating volume "mongodb-data" with default driver
Pulling mongodb (mongo:)...
latest: Pulling from library/mongo
f22ccc0b8772: Pull complete
3cf8fb62ba5f: Pull complete
e80c964ece6a: Pull complete
329e632c35b3: Pull complete
3e1bd1325a3d: Pull complete
4aa6e3d64a4a: Pull complete
035bca87b778: Pull complete
874e4e43cb00: Pull complete
08cb97662b8b: Pull complete
f623ce2ba1e1: Pull complete
f100ac278196: Pull complete
461b064aece5: Pull complete
Digest: sha256:00878f3d8e0a61997f2ea67351934b815a77c5ff8985df3ec041bca1c88258f4
Status: Downloaded newer image for mongo:latest
Pulling mongo-express (mongo-express:)...
latest: Pulling from library/mongo-express
cbdbe7a5bc2a: Pull complete
da41a38d96d0: Pull complete
3d6d69ed0edb: Pull complete
13618797e148: Pull complete
3a213630ced6: Pull complete
2880b14e420a: Pull complete
9eeb754ce0e7: Pull complete
0b93e97917be: Pull complete
Digest: sha256:4d219ac97564f6de664c15040bb8eabe8951562d13bbb4ca8bb978995f393d02
Status: Downloaded newer image for mongo-express:latest
Creating mongodb ... done
Creating mongo-express ... done
barry@ubser001 ~/mongodb
$
```

The above output contains no errors. So, we can assume that all the services are created



successfully. As we have added health checks for both services, we can verify it by using the docker **ps** command.

```
sudo docker ps -a
```

```
barry@ubser001 ~/mongodb
$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS
925ac02406d9	mongo-express	"tini -- /docker-ent..."	mongo-express	12 minutes ago	Up 12 minutes (healthy)
636eec4b6e12	mongo	"docker-entrypoint.s..."	mongodb	12 minutes ago	Up 12 minutes (healthy)

The docker **ps** command prints the health status of the container. This health status is only available if you have defined a health check for the container.

## Mongo Express

Now, let's go to the Mongo Express web interface using the server IP (<http://10.10.10.60:8081>).

Mongo Express Database

### Mongo Express

Databases

Database Name

+ Create Database

View	admin	Del
View	config	Del
View	local	Del

Server Status

Hostname	1422490249c1	MongoDB Version	4.4.2
Uptime	6 seconds	Server Time	Sun, 06 Dec 2020 16:51:07 GMT
Current Connections	3	Available Connections	838857
Active Clients	0	Queued Operations	0
Clients Reading	0	Clients Writing	0
Read Lock Queue	0	Write Lock Queue	0
Disk Flushes		Last Flush	
Time Spent Flushing	ms	Average Flush Time	ms
Total Inserts	0	Total Queries	2
Total Updates	0	Total Deletes	0

The Mongo Express interface provides a convenient way to interact with the MongoDB database. The Mongo Express interface also provides an overview status of the MongoDB server instance, providing a simple monitoring functionality.

That concludes this tutorial.

## Related reading

- [BMC Machine Learning & Big Data Blog](#)
- [MongoDB vs Cassandra: NoSQL Databases Compared](#), part of our MongoDB Guide
- [Docker Commands: A Cheat Sheet](#)
- [How To Introduce Docker Containers in Enterprise](#)
- [State of Containers: A Report Summary](#)