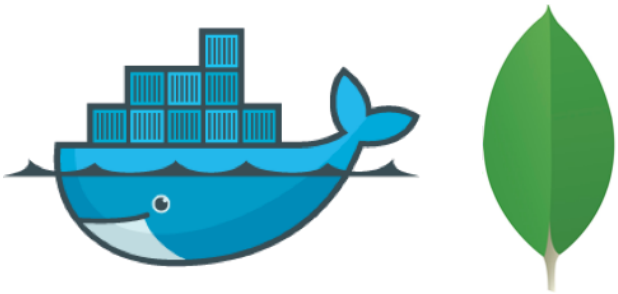
Cristian Ramirez Follow
FullStack MEAN Engineer with some Docker knowledge Linkedin: https://www.linkedin.com/in/cristian-r...
Jan 15 · 11 min read

How to deploy a MongoDB Replica Set using Docker

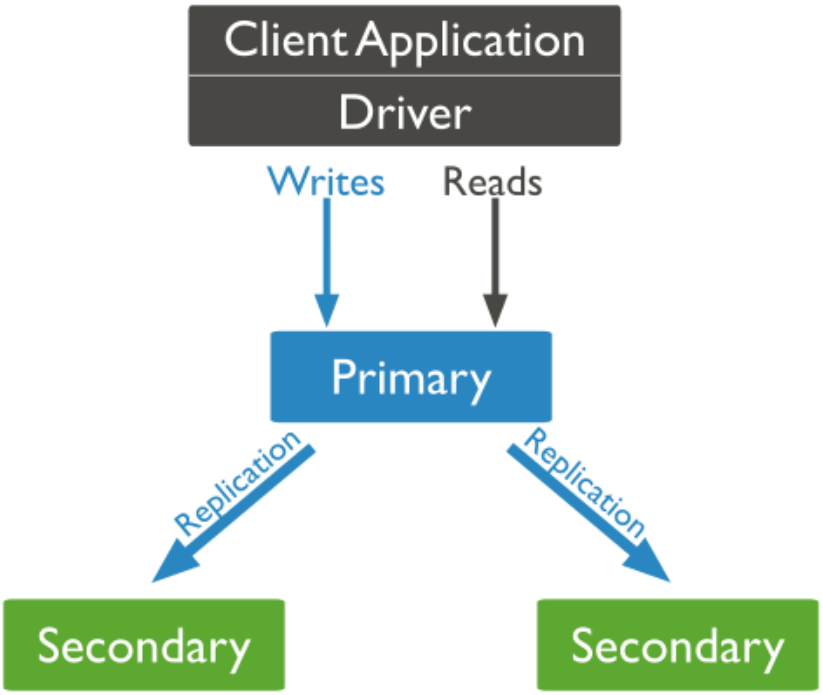


This article is going to be a walk-through in how to set up a MongoDB replica set with authentication using docker.

What we are going to use for this article is:

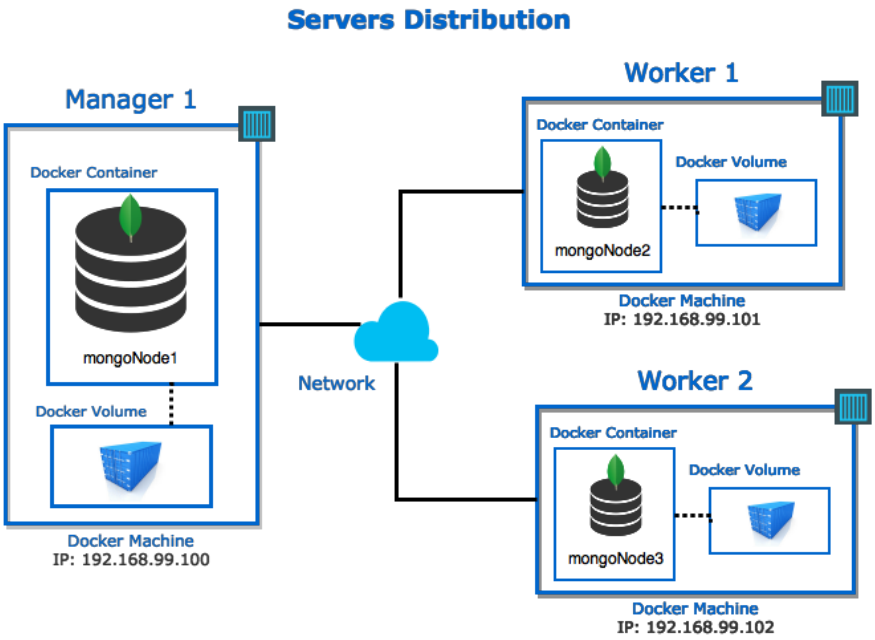
- MongoDB 3.4.1
- Docker for Mac 1.12.6

Architecture of the replica set



MongoDB Replica Set

Architecture for our replica set with docker



Servers Distribution using Docker

In the image above we are seeing what is going to be the result of our replication set with docker.

Prerequisite

- basic knowledge in **docker**
- **docker** and **docker-machine** installed
- basic knowledge in **mongoDB**
- basic knowledge in **bash scripting**

If you are on Mac or Windows, consider using a Virtual Machine. I will use VirtualBox on MacOS Sierra to run our mongoDB instances.

Step 1—Create our 3 docker-machines

To create a docker machine we need to issue the next command in a terminal:

```
$ docker-machine create -d virtualbox manager1
```

This command will create a machine called **manager1** using **virtualbox** as our virtualization provider.

Now let's create the two lefting `docker-machine`

```
$ docker-machine create -d virtualbox worker1
```

```
$ docker-machine create -d virtualbox worker2
```

To verify if our machines are created, let's run the following command:

```
$ docker-machine ls
```

```
// the result will be
NAME    ACTIVE   DRIVER     STATE     URL
manager1   -   virtualbox   Running
tcp://192.168.99.100:2376
worker1    -   virtualbox   Running
tcp://192.168.99.101:2376
worker2    -   virtualbox   Running
tcp://192.168.99.102:2376
```

Step 2—Configuration of master node of MongoDB

Now that we have our three machines lets position it in our **first machine** to start the mongodb configuration, let's run the next command:

```
$ eval `docker-machine env manager1`
```

Before creating our mongoDB containers, there is a very important topic that has been long discussed around **database persistence** in **docker containers**, and to achieve this challenge what we are going to do is to create a **docker volume**.

```
$ docker volume create --name mongo_storage
```

Now let's attached our volume created to start our first mongo container and set the configurations.

```
$ docker run --name mongoNode1 \
-v mongo_storage:/data \
-d mongo \
--smallfiles
```

Next we need to create the **key file.**

> *The contents of the keyfile serves as the shared password for the members of the replica set. The content of the keyfile must be the same for all members of the replica set.*

```
$ openssl rand -base64 741 > mongo-keyfile
$ chmod 600 mongo-keyfile
```

Next let's create the folders where is going to hold the data, keyfile and configurations inside the **mongo_storage** volume:

```
$ docker exec mongoNode1 bash -c 'mkdir /data/keyfile
/data/admin'
```

The next step is to create some admin users, let's create a **admin.js** and a **replica.js** file that looks like this:

```
// admin.js

admin = db.getSiblingDB("admin")

// creation of the admin user
admin.createUser(
  {
    user: "cristian",
    pwd: "cristianPassword2017",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)

// let's authenticate to create the other user
db.getSiblingDB("admin").auth("cristian",
"cristianPassword2017" )


// creation of the replica set admin user
db.getSiblingDB("admin").createUser(
  {
    "user" : "replicaAdmin",
    "pwd" : "replicaAdminPassword2017",
    roles: [ { "role" : "clusterAdmin", "db" : "admin" } ]
  }
)
```

. . .

```
//replica.js
```

```
rs.initiate({
 _id: 'rs1',
 members: [{
  _id: 0, host: 'manager1:27017'
 }]
})
```

IMPORTANT

> *Passwords should be random, long, and complex to ensure system security and to prevent or delay malicious access. See Database User Roles for a full list of built-in roles and related to database administration operations.*

What we have done until know:

- created the **mongo_storage**, docker volume.
- created the **mongo-keyfile**, openssl key generation.
- created the **admin.js file**, admin users for mongoDB.
- created the **replica.js file**, to init the replica set.

Ok let's continue with passing the files to the container.

```
$ docker cp admin.js mongoNode1:/data/admin/

$ docker cp replica.js mongoNode1:/data/admin/

$ docker cp mongo-keyfile mongoNode1:/data/keyfile/
```

. . .

```
// change folder owner to the user container

$ docker exec mongoNode1 bash -c 'chown -R mongodb:mongodb
/data'
```

What we have done is that we pass the files needed to the container, and then **change the /data folder owner to the container user**, since the the container user is the user that will need access to this folder and files.

Now everything has been set, and we are ready to restart the mongod instance with the replica set configurations.

Before we start the authenticated mongo container let's create an `env` file to set our users and passwords.

```
MONGO_USER_ADMIN=cristian
MONGO_PASS_ADMIN=cristianPassword2017

MONGO_REPLICA_ADMIN=replicaAdmin
MONGO_PASS_REPLICA=replicaAdminPassword2017
```

Now we need to remove the container and start a new one. Why ?, because we need to provide the replica set and authentication parameters, and to do that we need to run the following command:

```
// first let's remove our container

$ docker rm -f mongoNode1

// now lets start our container with authentication

$ docker run --name mongoNode1 --hostname mongoNode1 \
-v mongo_storage:/data \
--env-file env \
--add-host manager1:192.168.99.100 \
--add-host worker1:192.168.99.101 \
--add-host worker2:192.168.99.102 \
-p 27017:27017 \
-d mongo --smallfiles \
```

```
  --keyFile /data/keyfile/mongo-keyfile \
  --replSet 'rs1' \
  --storageEngine wiredTiger \
  --port 27017
```

What is going on here… 🤔 it seems that is an abuse of flags.

let me explaint to you in two parts:

Docker Flags:

1. The `--env-file` reads an env file and sets `environment` variables inside the container.
2. The `--add-host` flag adds entries into the docker container's `/etc/hosts` file so we can use hostnames instead of IP addresses. Here we are mapping our 3 docker-machines that we have created before.

For more deep understanding in *docker run commands*, read Docker's documentation.

Mongo Flags:

For setting up the mongo replica set we need a variety of mongo flags

1. `--keyFile` this flag is for telling mongo where is the `mongo-keyfile.`
2. `--replSet` this flag is for setting the name of the replica set.
3. `--storageEngine` this flag is for setting the engine of the mongoDB, is not requiered, since **mongoDB 3.4.1** its default engine is **wiredTiger.**

For more deep understanding in mongo replica set, read MongoDB documentation, also i recommend the MongoUniversity Courses for learning more about this topic.

Final step for the mongoNode1 container, is to start the replica set, and we are going to do that by running the following command:

```
$ docker exec mongoNode1 bash -c 'mongo <
/data/admin/replica.js'
```

You should see something like this:

```
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
{ "ok" : 1 }
bye
```

And now let's create the admin users with the following command:

```
$ docker exec mongoNode1 bash -c 'mongo <
/data/admin/admin.js'
```

You should see something like this:

```
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
admin
Successfully added user: {
 "user" : "cristian",
 ...
Successfully added user: {
 "user" : "replicaAdmin",
...
bye
```

And now to get in to the replica run the following command:

```
$ docker exec -it mongoNode1 bash -c 'mongo -u
$MONGO_REPLICA_ADMIN -p $MONGO_PASS_REPLICA --eval
"rs.status()" --authenticationDatabase "admin"'
```

And you should be ready and see something like this:

```
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
{
 "set" : "rs1",
 ...
 "members" : [
  {
   "_id" : 0,
   "name" : "manager1:27017",
   ...
 "ok" : 1
}
```

Step 3— Adding 2 more mongo node containers

Now that everything is ready, let's start 2 more nodes and join them to the replica set.

To add the first node let's change to the **worker1** docker machine, if you are using a local computer run the following command:

```
eval `docker-machine env worker1`
```

If you're not running on local, just point your terminal to the next server.

Now since we are going to repeat almost all the steps we made for **mongoNode1** let's make a script that runs all of our commands for us.

Let's create a file called **create-replica-set.sh** and let's see how is going to be composed the **main** function:

```
function main {
  init_mongo_primary
  init_mongo_secondaries
```

```
  add_replicas manager1 mongoNode1
  check_status manager1 mongoNode1
}


main
```

Now let me show you how are composed this functions:

INIT MONGO PRIMARY FUNCTION

```
function init_mongo_primary {
  # @params name-of-keyfile
  createKeyFile mongo-keyfile

  # @params server container volume
  createMongoDBNode manager1 mongoNode1 mongo_storage

  # @params container
  init_replica_set mongoNode1
}
```

This function has calls to functions inside also, there is nothing new added, we already saw all the functionality before, let me describe it for you what it does:

1. creation of the **keyfile** for the **replica set authentication.**
2. creation of a mongodb container, and recieves 2 parameters: a) the server where is going to be located, b) the name of the container, c) the name of the docker volume, all this functionality we saw it before.
3. and finally, it will initiate the replica with the exact same steps, we do before.

INIT MONGO SECONDARY FUNCTION

```
function init_mongo_secondaries {
  # @Params server container volume
  createMongoDBNode worker1 mongoNode1 mongo_storage
  createMongoDBNode worker2 mongoNode2 mongo_storage
}
```

This function what it does is to **creates the other 2 mongo containers for the replica set**, and executes the same steps as the mongoNode1, but here we don't include the replica set instantiation, and the admin users creation, because those aren't necessary, since the replica set, will share with all the nodes of the replica the database configurations, and later on they will be added to the primary database.

ADD REPLICAS FUNCTION

```
# @params server container
function add_replicas {
  echo '·· adding replicas >>>> '$1' ··'


  switchToServer $1

  for server in worker1 worker2
   do
    rs="rs.add('$server:27017')"
    add='mongo --eval "'$rs'" -u $MONGO_REPLICA_ADMIN
         -p $MONGO_PASS_REPLICA --
authenticationDatabase="admin"'
    sleep 2
    wait_for_databases $server
    docker exec -i $2 bash -c "$add"
  done
}
```

Here what are we doing, is finally adding the 2 other mongo containers to the **primary database** on the **replica set configuration,** first we loop through the machines left to add the containers, in the loop we prepare the configuration, then we check if the container is ready, we do that by calling the function `wait_for_databases` and we pass the machine to check as the parameter, then we execute the configuration inside the primary database and we should se messages like this:

```
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
{ "ok" : 1 }
```

That means that the mongo container was added successfully to the replica.

And finally we check the status of the replica set with the last function in the main:

```
# @params server container
function check_status {
  switchToServer $1
  cmd='mongo -u $MONGO_REPLICA_ADMIN -p $MONGO_PASS_REPLICA
        --eval "rs.status()" --authenticationDatabase
"admin"'
  docker exec -i $2 bash -c "$cmd"
}
```

Now that we have seen the functions of our automated script and that we know is going to do, it's time to execute the automated bash script like the following:

> *note if you have made all the steps above, you need to reset everything that we have implemented, to avoid any collision name problems, to reset the configurations there is a reset.sh file on the github repository*

```
# and this how we can execute the script that will configure
# everything for us.
```

```
$ bash < create-replica-set.sh
```

And if everything was set correctly, we should see a message from mongodb like this:

```
MongoDB shell version v3.4.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.1
{
 "set" : "rs1",
 ...
```

```
  },
  "members" : [
   {
    "_id" : 0,
    "name" : "manager1:27017",
    "health" : 1,
    "state" : 1,
    "stateStr" : "PRIMARY",
    ...
   },
   {
    "_id" : 1,
    "name" : "worker1:27017",
    "health" : 1,
    "state" : 2,
    "stateStr" : "SECONDARY",
    ...
   },
   {
    "_id" : 2,
    "name" : "worker2:27017",
    "health" : 1,
    "state" : 0,
    "stateStr" : "STARTUP",
    ...
   }
  ],
  "ok" : 1
 }
```

As you can see every container is now well configured, some things to notice is that we we use the `--add-host` flag from docker as we used before and this adds these entries into the Docker container's /etc/hosts file so we can use hostnames instead of IP addresses.

It might take a minute for both node to complete syncing from mongoNode1.

You can view what is happening in each mongo Docker container by looking at the logs. You can do this by running this command on any of the docker-machine servers.

```
$ docker logs -ft mongoContainerName
```

Now that we have a **MongoDB replica set** service up and running, let's modify our user or you can create another user and grant some permissions to make crud operations over a database, so for

illustration purposes only, this a bad practice, let me add a super role to our admin user.

```
# we are going to assign the root role to our admin user

# we enter to the container
$ docker exec -it mongoNode1 bash -c 'mongo -u
$MONGO_USER_ADMIN -p $MONGO_PASS_ADMIN --
authenticationDatabase "admin"'

# Then we execute the following in the mongo shell
# Mongo 3.4.1 shell
> use admin
> db.grantRolesToUser( "cristian", [ "root" , { role:
"root", db: "admin" } ] )
>
```

Now he have a super user that can make anything, so let's create a database and insert some data.

```
$ docker exec -it mongoNode1 bash -c 'mongo -u
$MONGO_USER_ADMIN -p $MONGO_PASS_ADMIN --
authenticationDatabase "admin"'

# Mongo 3.4.1 shell
> use movies
> db.movies.insertMany([{
  id: '1',
  title: 'Assasins Creed',
  runtime: 115,
  format: 'IMAX',
  plot: 'Lorem ipsum dolor sit amet',
  releaseYear: 2017,
  releaseMonth: 1,
  releaseDay: 6
}, {
  id: '2',
  title: 'Aliados',
  runtime: 124,
  format: 'IMAX',
  plot: 'Lorem ipsum dolor sit amet',
  releaseYear: 2017,
  releaseMonth: 1,
  releaseDay: 13
}, {
  id: '3',
  title: 'xXx: Reactivado',
  runtime: 107,
  format: 'IMAX',
```

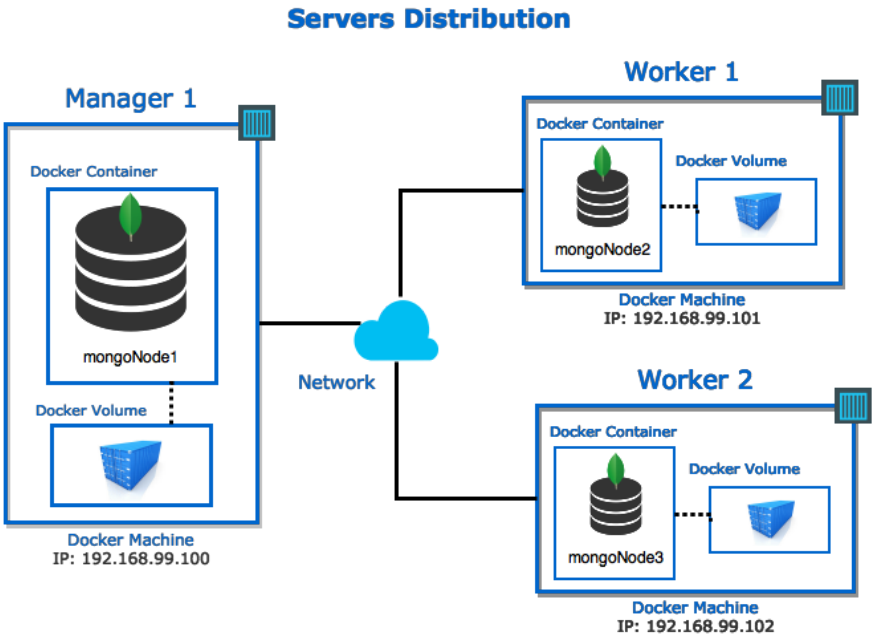
```
    plot: 'Lorem ipsum dolor sit amet',
    releaseYear: 2017,
    releaseMonth: 1,
    releaseDay: 20
  }, {
    id: '4',
    title: 'Resident Evil: Capitulo Final',
    runtime: 107,
    format: 'IMAX',
    plot: 'Lorem ipsum dolor sit amet',
    releaseYear: 2017,
    releaseMonth: 1,
    releaseDay: 27
  }, {
    id: '5',
    title: 'Moana: Un Mar de Aventuras',
    runtime: 114,
    format: 'IMAX',
    plot: 'Lorem ipsum dolor sit amet',
    releaseYear: 2016,
    releaseMonth: 12,
    releaseDay: 2
  }])

  # inserted 5 documents
  >
```

Now we have a movies database with a movies collection that contains 5 movies :D.

Time for a recap

What we have done…



We configure and start a **MongoDB Replica Set** with **Authentication** using **Docker,** with an automated script.

In security terms, we create:

- 2 type of users, the **admin database** and the **cluster admin database.**
- we create a **key file,** and start the replica with **authentication enabled.**

> *If access control is configured correctly for the database, attackers should not have been able to gain access to your data. Review our Security Checklist to help catch potential weaknesses.—@MongoDB Docs*

If we want to add more security to our architecture, we can create a **Swarm Cluster,** with our docker-machines, and docker swarm handles well the network communication, also we can create non-root users in our containers, and we can enable encryption data in mongoDB, but this topics are outside of scope of the pretentions of this article.

. . .

Conclusion

Now we have a working **MongoDB replica set**. You are free to add nodes to this replica set at any time. You can even stop one of the mongo container or the primary mongoNode1 and watch another mongoNode take over as the new primary. Since the **data is written on docker volumes**, restarting any of these nodes is not a problem. **The data will persist** and rejoin the replica set perfectly fine.

One bonus for us is that we saw how to automate this whole process with a bash file.

One challenge for you is to modify the **bash script** and make it more **dynamically** since this script is very attached to this article specifiactions, and another challenge is to add an **Arbitrary Mongo Node,** to the architecture.

. . .

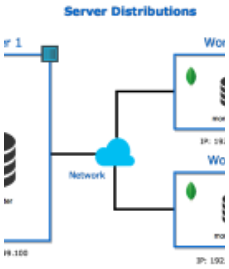
Github repository

To get the complete script files of the article you can check it at the following repo.

GitHub - Crizstian/mongo-replica-with-docker: How to deploy a MongoDB Replica Set using Docker

mongo-replica-with-docker - How to deploy a MongoDB Replica Set using Docker

github.com



Further reading

- Deploy Replica Set With Keyfile Access Control—MongoDB Docs.
- Add Members to a Replica Set—MongoDB Docs.
- MongoDB Docker Hub—Docker Docs.
- How to Avoid a Malicious Attack That Ransoms Your Data
- How to setup a secure mongoDB 3.4 server on ubuntu 16.04

. . .

I hope you enjoyed this article, i'm currently still exploring the MongoDB world, so i am open to accept feedback or contributions, and if you liked it, recommend it to a friend, share it or read it again.

You can follow me at twitter @cramirez_92
https://twitter.com/cramirez_92

Until next time 😁👦🎨👦💻