Deploy a MongoDB Cluster in 9 steps Using Docker

This tutorial basically walks you through what is outlined here http://docs.mongodb.org/manual/tutorial/deploy-replica-set-with-auth/ but translated into how you would do this if you were deploying this using Docker.

Will Deploy:

- MongoDB version 2.6.5
- Replica set with 3 nodes
- Authentication enabled
- Persistent data to the local file system

The only **prerequisite** is to have 3 working Docker server. This means you can bring up a local Vagrant box with Docker installed, use CoreOS, instance on AWS, or however you like to get 3 Docker server.

Steps

Step 1: Get the IP address of all three servers and export the following IP addresses variables on all servers by running the following commands on all servers (replace the IP addresses).

Ideally you would not have and the IPs can be resolved via DNS. Since this is a test setup, this is easier.

```
root@node*:/# export node1=10.11.32.174
root@node*:/# export node2=10.11.33.37
root@node*:/# export node3=10.11.31.176
```

Step 2: Create key file to be used on all nodes. Run this once on one of the server and then copy the file over to the other 2 servers to the same location.

For this tutorial, I will be putting everything in "/home/core" directory.

```
root@node*:/# mkdir -p /home/core
root@node*:/# cd /home/core
root@node*:/# openssl rand -base64 741 > mongodb-keyfile
root@node*:/# chmod 600 mongodb-keyfile
root@node*:/# sudo chown 999 mongodb-keyfile
```

The file owner was changed to a user id of "999″ because the user in the MongoDB Docker container is the one that needs to access this key file.

Step 3: On node1, start the following mongodb container. This will start the container with no authentication so that we can provision it.

```
root@node1:/# docker run --name mongo \
-v /home/core/mongo-files/data:/data/db \
-v /home/core/mongo-files:/opt/keyfile \
--hostname="node1.example.com" \
-p 27017:27017 \
-d mongo:2.6.5 --smallfiles
```

Create an admin user. We will connect to the mongo container we just started and start an interactive shell.

```
root@node1:/# docker exec -it mongo /bin/bash
```

At this point, we are inside the mongo Docker container and we will start a mongo shell

```
root@node1:/# mongo
```

Which will start the mongo shell.

```
MongoDB shell version: 2.6.5
connecting to: test


Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
>
```

Switch to the admin user

```
> use admin
switched to db admin
```

Create a new site admin user

```
> db.createUser( {
     user: "siteUserAdmin",
     pwd: "password",
     roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
   });
```

You should see a successful message

```
Successfully added user: {
"user" : "siteUserAdmin",
"roles" : [
         {
              "role" : "userAdminAnyDatabase",
              "db" : "admin"
         }
      ]
}
```

Create a root user

```
> db.createUser( {
     user: "siteRootAdmin",
     pwd: "password",
     roles: [ { role: "root", db: "admin" } ]
   });
```

You should see a successful message

```
Successfully added user: {
            "user" : "siteRootAdmin",
                  "roles" : [
            {
            "role" : "root",
                  "db" : "admin"
            }
        ]
}
```

We are done creating users that we can use to log back in later. We are now going to exit out of the interactive shells: mongo and Docker.

```
> exit
bye
root@node1:/# exit
```

Step 4: Stop the first Mongo instance

```
root@node1:/# docker stop mongo
```

Step 5: Start up the first Mongo instance but this time with the key file. This is still on node1.

```
root@node1:/# docker rm mongo


root@node1:/# docker run \
--name mongo \
-v /home/core/mongo-files/data:/data/db \
-v /home/core/mongo-files:/opt/keyfile \
--hostname="node1.example.com" \
--add-host node1.example.com:${node1} \
--add-host node2.example.com:${node2} \
--add-host node3.example.com:${node3} \
-p 27017:27017 -d mongo:2.6.5 \
--smallfiles \
--keyFile /opt/keyfile/mongodb-keyfile \
--replSet "rs0"
```

Note:

- The "—keyFile" path is "/opt/keyfile/mongodb-keyfile". This is correct. This is the path to the key file inside the Docker container. We mapped the key file with the "-v" option to this location inside the container.

- The “—add-host” adds these entries into the Docker container’s /etc/hosts file so we can use hostnames instead of IP addresses. In a production environment where these entries are in DNS, these parameters are not needed.

Step 6: Connect to the replica set and configure it. This is still on node1. We will start another interactive shell into the mongo container and start a mongo shell.

```
root@node1:/# docker exec -it mongo /bin/bash
root@node1:/# mongo
MongoDB shell version: 2.6.5
>
```

Switch to the admin user

```
> use admin
switched to db admin
```

This time since we have setup a password, we will have to authenticate. We have set the password to “password” =)

```
> db.auth("siteRootAdmin", "password");
1
```

We can now initiate the replica set

```
> rs.initiate()
{
         "info2" : "no configuration explicitly specified --
making one",
         "me" : "node1.example.com:27017",
         "info" : "Config now saved locally.  Should come online
in about a minute.",
        "ok" : 1
}
>
```

Step 7: Verify the initial replica set configuration

```
>
rs0:PRIMARY> rs.conf()
{
        "_id" : "rs0",
        "version" : 1,r
        "members" : [
              {
                  "_id" : 0,
                  "host" : "node1.example.com:27017"
              }
        ]
}
```

Step 8: Start Mongo on the other 2 nodes

Perform on Node 2

```
root@node2:/# docker run \
--name mongo \
-v /home/core/mongo-files/data:/data/db \
-v /home/core/mongo-files:/opt/keyfile \
--hostname="node2.example.com" \
--add-host node1.example.com:${node1} \
--add-host node2.example.com:${node2} \
--add-host node3.example.com:${node3} \
-p 27017:27017 -d mongo:2.6.5 \
--smallfiles \
--keyFile /opt/keyfile/mongodb-keyfile \
--replSet "rs0"
```

Perform on Node 3

```
root@node3:/# docker run \
--name mongo \
-v /home/core/mongo-files/data:/data/db \
-v /home/core/mongo-files:/opt/keyfile \
--hostname="node3.example.com" \
--add-host node1.example.com:${node1} \
--add-host node2.example.com:${node2} \
--add-host node3.example.com:${node3} \
-p 27017:27017 -d mongo:2.6.5 \
--smallfiles \
--keyFile /opt/keyfile/mongodb-keyfile \
--replSet "rs0"
```

Step 9: Add the other 2 nodes into the replica set

Back to node1 where we are in the mongo shell. If you hit enter a few times here, your prompt should have changed to "rso:PRIMARY". This is because this is the primary node now for replica set "rso".

```
rs0:PRIMARY> rs.add("node2.example.com")
rs0:PRIMARY> rs.add("node3.example.com")
```

We can validate that the other 2 nodes added in correctly by running:

```
rs0:PRIMARY> rs.status()
```

It might take a minute for both node to complete syncing from node1.

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 servers.

```
root@node*:/# docker logs -ft mongo
```

Conclusion

Now you have a working MongoDB cluster in a replica set. You are free to add nodes to this cluster at any time. You can even stop one of the nodes or primary node and watch another node take over as the new primary. Since the data is written to the local file system, restarting any of these nodes is not a problem. The data will persist and rejoin the cluster perfectly fine.