Composing a Sharded MongoDB Cluster on Docker Containers

by Ayberk Cansever · Aug. 04, 17 · Database Zone · Tutorial

In this article, we will write a `docker-compose.yaml` file and a cluster initiation scripts which will deploy a sharded MongoDB cluster on Docker containers.

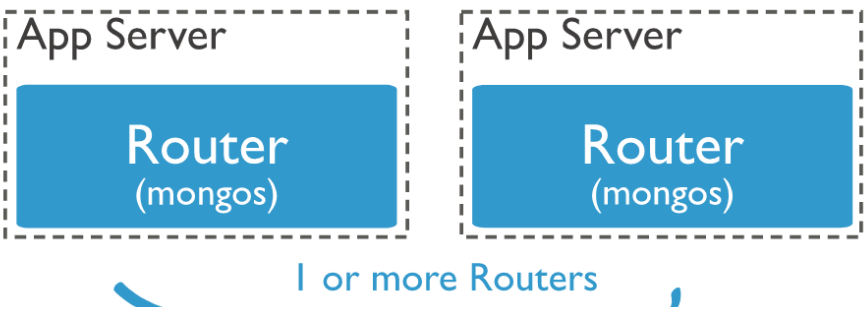
Initially, let's look what kind of components we are going to need for a sharded MongoDB cluster. If we look at the official documentation, we need three main components which are obviously defined:

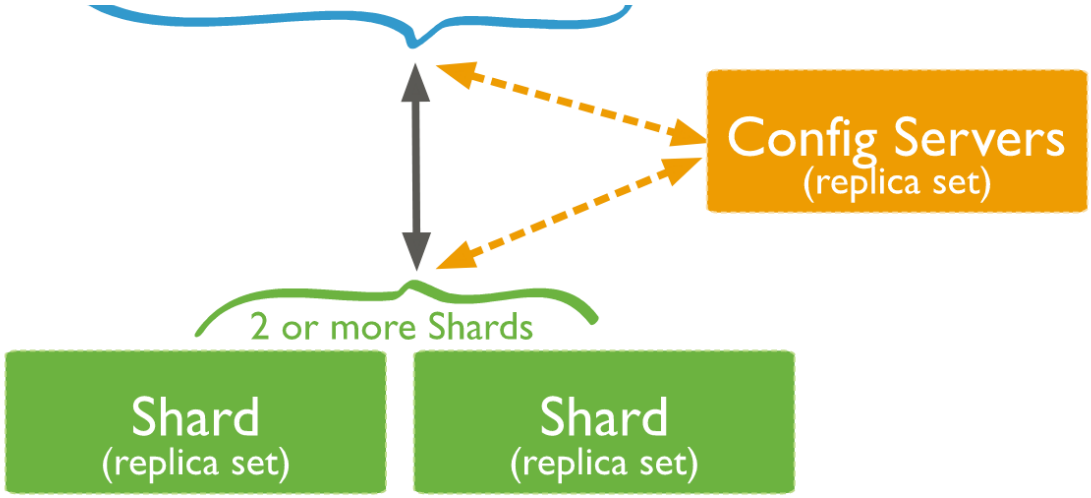
1. **Shard**: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.

2. **Mongos (router):** The mongos acts as query routers, providing an interface between client applications and the sharded cluster.

3. **Config servers:** Config servers store metadata and configuration settings for the cluster.

In the official documentation, the main architecture of a sharded cluster looks like this:





Now, we are beginning to build a cluster that consists of a shard that is a replica set (three nodes), config servers (three nodes replica set), and two router nodes. In total, we will have eight Docker containers running for our MongoDB sharded cluster. Of course, we can expand our cluster according to our needs.

Let's start to write our `docker-compose.yaml` file by defining our shard replica set:

```
version: '2'
services:
  mongorsn1:
    container_name: mongors1n1
    image: mongo
    command: mongod --shardsvr --replSet mongors1 --dbpath /data/db --port 27017
    ports:
      - 27017:27017
    expose:
      - "27017"
    environment:
      TERM: xterm
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /mongo_cluster/data1:/data/db
  mongors1n2:
    container_name: mongors1n2
    image: mongo
    command: mongod --shardsvr --replSet mongors1 --dbpath /data/db --port 27017
    ports:
      - 27027:27017
    expose:
      - "27017"
    environment:
      TERM: xterm
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /mongo_cluster/data2:/data/db
  mongors1n3:
    container_name: mongors1n3
    image: mongo
    command: mongod --shardsvr --replSet mongors1 --dbpath /data/db --port 27017
    ports:
      - 27037:27017
```

```
      - 27037:27017
    expose:
      - "27017"
    environment:
      TERM: xterm
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /mongo_cluster/data3:/data/db
```

As you see, we defined our shard nodes by running them with the `shardsvr` parameter. Also, we mapped the default MongoDB data folder (*/data/db*) of the container, as you see. We will build a replica set with these three nodes when we finish writing our `docker-compose.yaml` file.

Now, let's define our three config servers:

```
mongocfg1:
    container_name: mongocfg1
    image: mongo
    command: mongod --configsvr --replSet mongors1conf --dbpath /data/db --port 27017
    environment:
      TERM: xterm
    expose:
      - "27017"
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /mongo_cluster/config1:/data/db
  mongocfg2:
    container_name: mongocfg2
    image: mongo
    command: mongod --configsvr --replSet mongors1conf --dbpath /data/db --port 27017
    environment:
      TERM: xterm
    expose:
      - "27017"
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /mongo_cluster/config2:/data/db
  mongocfg3:
    container_name: mongocfg3
    image: mongo
    command: mongod --configsvr --replSet mongors1conf --dbpath /data/db --port 27017
    environment:
      TERM: xterm
    expose:
      - "27017"
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /mongo_cluster/config3:/data/db
```

Our config servers are running with the `configsvr` parameter, as you see.

Finally, we are going to define our mongos (router) instances:

```
mongos1:
    container_name: mongos1
    image: mongo
    depends_on:
      - mongocfg1
      - mongocfg2
    command: mongos --configdb mongors1conf/mongocfg1:27017,mongocfg2:27017,mongocfg3:27017
    ports:
      - 27019:27017
    expose:
      - "27017"
    volumes:
      - /etc/localtime:/etc/localtime:ro
  mongos2:
    container_name: mongos2
    image: mongo
    depends_on:
      - mongocfg1
      - mongocfg2
    command: mongos --configdb mongors1conf/mongocfg1:27017,mongocfg2:27017,mongocfg3:27017
    ports:
      - 27020:27017
    expose:
      - "27017"
    volumes:
      - /etc/localtime:/etc/localtime:ro
```

These mongos are dependent on our config servers. They take the `configdb` parameter to obtain metadata and configuration settings.

At last, we built our `docker-compose.yaml` file. If we compose it up, we will see eight running docker containers: 3 shard date replicate set + 3 config servers + 2 mongos (routers):

```
docker-compose up
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                      NAMES
14d32948734f        mongo               "docker-entrypoint..."   41 seconds ago      Up 40 seconds       0.0.0.0:27017->27017/tcp   mongors1n1
04092b6c6b80        mongo               "docker-entrypoint..."   2 minutes ago       Up 38 seconds       0.0.0.0:27020->27017/tcp   mongos2
425cd4d71c43        mongo               "docker-entrypoint..."   2 minutes ago       Up 38 seconds       0.0.0.0:27019->27017/tcp   mongos1
1282f77acc12        mongo               "docker-entrypoint..."   2 minutes ago       Up 39 seconds       27017/tcp                  mongocfg1
f151d9410ff2        mongo               "docker-entrypoint..."   2 minutes ago       Up 40 seconds       27017/tcp                  mongocfg2
f10daf7ce7e8        mongo               "docker-entrypoint..."   23 hours ago        Up 40 seconds       0.0.0.0:27027->27017/tcp   mongors1n2
543131d95fba        mongo               "docker-entrypoint..."   23 hours ago        Up 40 seconds       0.0.0.0:27037->27017/tcp   mongors1n3
```

But we're not finished yet. Our sharding cluster needs to be configured. For this purpose, we will run some commands, which will build our cluster on related nodes.

First, we will configure our config servers replica set:

```
docker exec -it mongocfg1 bash -c "echo 'rs.initiate({_id: \"mongors1conf\",configsvr: true,
```

We can check our config server replica set status by running the below command on the first config server node:

```
docker exec -it mongocfg1 bash -c "echo 'rs.status()' | mongo"
```

We are going to see three replica set members.

Secondly, we are going to build our shard replica set:

```
docker exec -it mongors1n1 bash -c "echo 'rs.initiate({_id : \"mongors1\", members: [{ _id :
```

Now, our shard nodes know each other. One of them is primary and two are secondary. We can check the replica set status by running the status check command on the first shard node:

```
docker exec -it mongors1n1 bash -c "echo 'rs.status()' | mongo"
```

Finally, we will introduce our shard to the routers:

```
 docker exec -it mongos1 bash -c "echo 'sh.addShard(\"mongors1/mongors1n1\")' | mongo "
```

Now our routers, which are the interfaces of our cluster to the clients, have the knowledge about our shard. We can check the shard status by running the command below on the first router node:

```
 docker exec -it mongos1 bash -c "echo 'sh.status()' | mongo "
```

We see the shard status:

```
MongoDB shell version v3.4.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.6
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5981af29870d1fc551e91a9e")
}
  shards:
        {  "_id" : "mongors1",  "host" : "mongors1/mongors1n1:27017,mongors1n2:27017,mongors1n3:27017",  "state" : 1 }
  active mongoses:
        "3.4.6" : 2
 autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
                Balancer lock taken at Wed Aug 02 2017 10:53:29 GMT+0000 (UTC) by ConfigServer:Balancer
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
```

We see that we have a single shard named `mongors1` , which has three mongod instances. But we do not have any databases yet, as you see. Let's create a database named testDb:

```
docker exec -it mongors1n1 bash -c "echo 'use testDb' | mongo"
```

This is not enough; we should enable sharding on our newly created database:

```
docker exec -it mongos1 bash -c "echo 'sh.enableSharding(\"testDb\")' | mongo "
```

Now, we have a sharding-enabled database on our sharded cluster! It's time to create a collection on our sharded database:

```
docker exec -it mongors1n1 bash -c "echo 'db.createCollection(\"testDb.testCollection\")' |
```

We created a collection named `testCollection` on our database, but it is not sharded yet again. We must shard our collection by choosing a sharding key. Let's assume that we have decided to shard our collection on a field named `shardingField` then:

```
docker exec -it mongos1 bash -c "echo 'sh.shardCollection(\"testDb.testCollection\", {\"shar
```

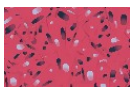
The sharding key must be chosen very carefully because it is for distributing the documents throughout the cluster. It is a *must* to read the official documentation about shard keys.

At the end, we have a sharded cluster, a sharded database, and a sharded collection. If we need to expand our cluster architecture in the future, we can add some new nodes as demanded!

Like This Article? Read More From DZone

DZone Article

Pitfalls and Workarounds for Tailing the Oplog on a MongoDB Sharded Cluster



DZone Article

When Should I Enable MongoDB Sharding?

DZone Article

MongoDB Replication and Sharding [Video]

Free DZone Refcard

Distributed SQL Essentials

Topics: DATABASE, DOCKER, MONGODB, NOSQL, SHARDED CLUSTER, TUTORIAL