MONGODB SHARDING

Cognitive Assistant for Networks (CAN) Release 5.5

JULY 16, 2021

AVANSEUS TECHNOLOGY PVT. LTD.

Revision History

Version	Date	Change description	Created by	Updated by	Reviewed by
V 1.0	July, 2021	Initial Release	Sunil	Sandeep Singh	Chiranjib

Table of Contents

1. Objective

This document focuses on MongoDB sharding setup & configuration on VM based environment. Existing customers with VM based MongoDB setup can use this documentation for sharding the database or collection in MongoDB.

2. MongoDB Sharding

To setup MongoDB Sharding environment, you should ideally have the below setups:

- A Config Databases Replica Set of 3 or more members
- Multiple Shard Database Replica Sets
- Mongos Database instances

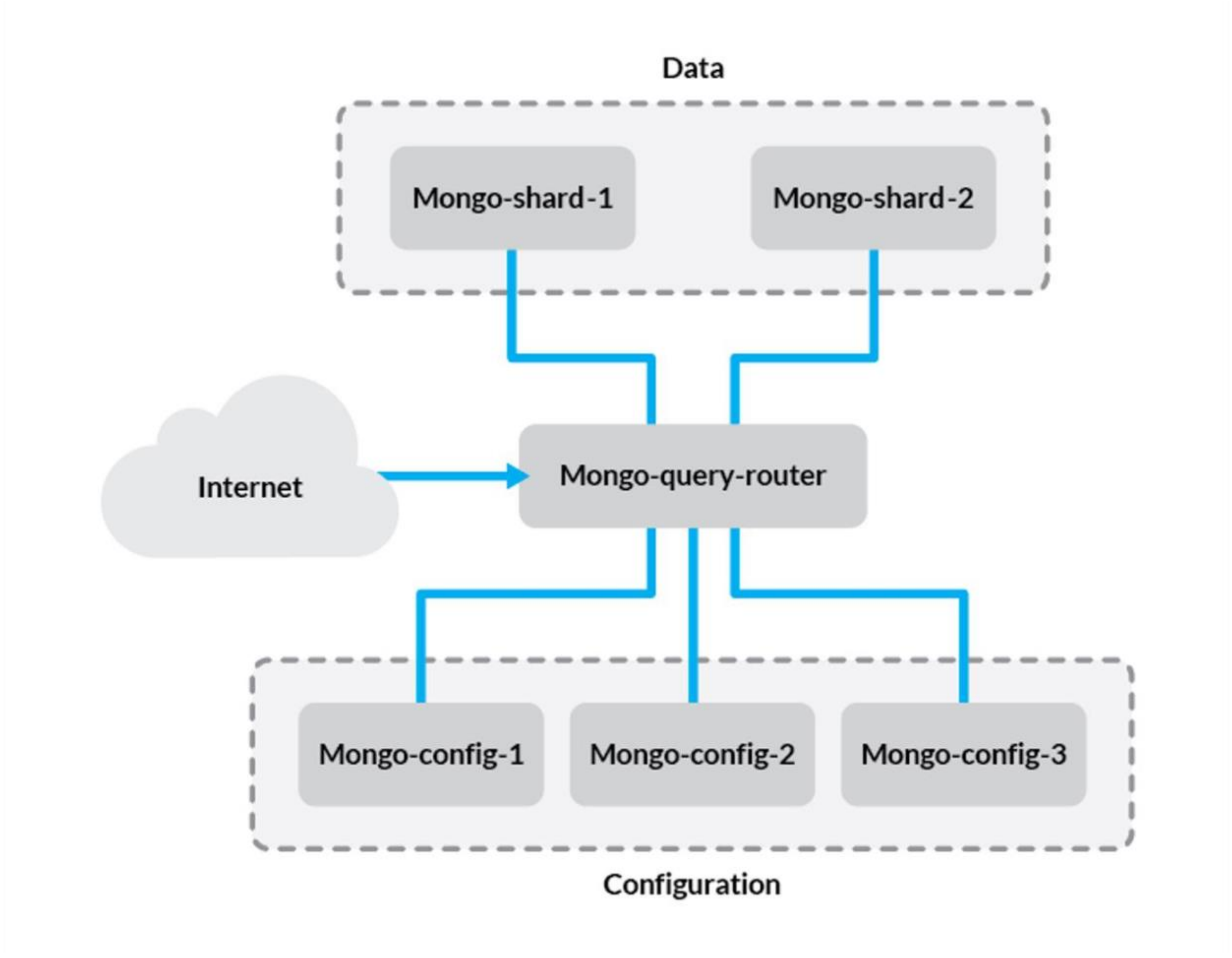

Figure 1 - MongoDB Sharding Environment

2.1. Cluster Architecture

Before getting started, review the components of the setup you will create:

- **Config Server** - Config server is used to store the metadata of the cluster server. This metadata contains information of cluster data set mapping. Query router or mongos uses this metadata information to perform operations on the specific shards. You can implement three config server sharded clusters in a production environment.

- **Shard -** A shard is a database server that holds a portion of your data. Items in the database are divided among shards either by range or hashing. Shard will provide high availability and data consistency of the database server. You can implement each replica set to have a separate shard. Each shard will contain a subset of sharded data.

- **Query Router (Mongos) -** The mongos daemon acts as an interface between the client application and the cluster shards. Since data is distributed among multiple servers, queries need to be routed to the shard where a given piece of information is stored. The query router runs on the application server.

To set up all the Config, Shard and Mongos instances, follow the below steps:

1. Create the Config Server Replica Set (For a production deployment, deploy a config server replica set with at least three members):

   a. Create a directory for storing Config Server data and logs

   ```
   mkdir config_server_data
   mkdir config_server_logs
   ```

   Go to the config_server_data directory and create the directories cfg1 for storing config database replica sets.

   ```
   cd config_server_data
   mkdir cfg1
   ```

   Follow the above steps in other two different servers (cfg2 & cfg3).

   b. Create the configuration files required for config servers cfg1, cfg2 and cfg3:

   #open the file cfg1.conf

   ```
   vi cfg1.conf
   ```

   #Save the below configuration in this file:

   ```
   storage:
     dbPath: /home/ec2-user/mongodb/config_server_data/cfg1
     journal:
       enabled: true

   systemLog:
     destination: file
     logAppend: true
     path: /home/ec2-user/mongodb/config_server_logs/cfg1.log
   ```

```
net:
  port: 26050
  bindIp: 172.31.31.7 #internal IP address

sharding:
  clusterRole: configsvr

replication:
  replSetName: cfgReplicaSet

#security:
   #keyFile: /home/ec2-user/mongodbKeyFile/keyFile
```

Similarly, create the configuration file for cfg2 & cfg3 in other servers. Update the DB path, log path and port number for cfg2 & cfg3.

c. Start all the three config server instances in three different servers:

```
Server 1:   mongod --config cfg1.conf --fork
Server 2:   mongod --config cfg2.conf --fork
Server 3:   mongod --config cfg3.conf --fork
```

d. Initiate the config server with the three-member replica set and then check the status of the server with the status () function:

```
mongo --host 172.31.31.7 --port 26050
>rs.initiate();
>rs.add("172.31.31.6:26050");
>rs.add("172.31.31.5:26050");
>rs.status();
```

2. Create the Shard Replica Sets

Note: You can create as many shards as required. Here, you will create three shards:

a. Create the directory for storing sharding server data and logs:

```
mkdir sharding_data
mkdir sharding_logs
```

Go to the sharding_data and create the folders shda1, shdb1 and shdc1 for storing shard a, shard b & shard c's replica sets data. Sharding 1 (shard a) will have three replica sets shda1, shda2 & shda3, and similarly, sharding 2 (shard b) will have three replica sets shdb1, shdb2 & shdb3, and sharding 3 (shardc) will have three replica sets shdc1, shdc2 & shdc3. You will use shda1 for storing primary replica set data and the other two shda2 & shda3 for secondary replica sets of shard a. Same follows for the second shard b and c as well.

```
cd sharding_data
```

```
mkdir shda1  shdb1  shdc1
```

Follow the same procedure in the other two servers as well for storing secondary replica set data ((shda2, shdb2 & shdc2) & (shda3, shdb3 shdc3))

b. Create the configuration files required for sharding a, b & c

Shard a configuration:

#open the file shda1.conf

```
vi shda1.conf
```

#Save the below configuration in this file.

```
storage:
  dbPath: /home/ec2-user/mongodb/sharding_data/shda1
  journal:
    enabled: true

systemLog:
  destination: file
  logAppend: true
  path: /home/ec2-user/mongodb/sharding_logs/shda1.log

net:
  port: 27000
  bindIp: 172.31.31.7 #internal IP

sharding:
  clusterRole: shardsvr

replication:
  replSetName: sharda

#security:
   #keyFile: /home/ec2-user/mongodbKeyFile/keyFile
```

Similarly, create the configuration file required for other replica sets. Change the DB path, log path and port number. ReplSetName sharda will remain the same for shda1, shda2 & shda3 ReplSetName shardb for shdb1, shdb2 & shdb3 and ReplSetName shardc for shdc1, shdc2 & shdc3. Port number 27000 will be assigned for shard a replica set members and 27100 & 27200 for shard b & shard c replica set members.

c. Start all the sharding server instances:

#Shard a replica set members

```
Server 1:  mongod --config shda1.conf --fork
Server 2:  mongod --config shda2.conf --fork
Server 3:  mongod --config shda3.conf –fork
```

#Shard b replica set members

```
Server 1:  mongod --config shdb1.conf --fork
Server 2:  mongod --config shdb2.conf --fork
Server 3:  mongod --config shdb3.conf --fork
```

#Shard c replica set members

```
Server 1:  mongod --config shdc1.conf --fork
Server 2:  mongod --config shdc2.conf --fork
Server 3:  mongod --config shdc3.conf –fork
```

d. Initiate replication in sharding servers. You can use the initiate () function to initiate the config server with the default configuration. Add the replication sets and then check the status of the server with the status () function:

#for shard a

```
mongo --host 172.31.31.7 --port 27000
>rs.initiate()
>rs.add("172.31.31.6:27000");
>rs.add("172.31.31.5:27000");
>rs.status();
```

#for shard b

```
mongo --host 172.31.31.7 --port 27100
>rs.initiate()
>rs.add("172.31.31.6:27100");
>rs.add("172.31.31.5:27100");
>rs.status();
```

#for shard c

```
mongo --host 172.31.31.7 --port 27200
>rs.initiate()
>rs.add("172.31.31.6:27200");
>rs.add("172.31.31.5:27200");
>rs.status();
```

3. Configure mongos for the Sharded Cluster:

   a. Create the configuration files required for the sharded cluster. This sharded cluster will route the queries to different shards based on the shard key.

      #open the file queryRouter.conf.

```
vi queryRouter.conf
```

      #Save the below configuration in this file.

```
systemLog:
  destination: file
  logAppend: true
  path: /home/ec2-user/mongodb/query_router_logs/query_router.log

net:
  port: 27017
  bindIp: 172.31.31.7

sharding:
  configDB: cfgReplicaSet/172.31.31.7:26050,172.31.31.6:26050,172.31.31.5:26050

#security:
   #keyFile: /home/ec2-user/mongodbKeyFile/keyFile
```

   b. Start the mongos service.

```
mongos --config query-router.conf --fork
```

   c. Connect to the Sharded Cluster and add Shards to the cluster. Connect a mongo shell to the mongos. Specify the host and port on which the mongos are running.

```
mongo --host 172.31.31.7 --port 27017
>sh.addShard("sharda/172.31.31.7:27000,172.31.31.6:27000,172.31.31.5:27000");
>sh.addShard("shardb/172.31.31.7:27100,172.31.31.6:27100,172.31.31.5:27100");
>sh.addShard("shardc/172.31.31.7:27200,172.31.31.6:27200,172.31.31.5:27200");
>sh.status();
```

   d. Add credentials to admin and other required databases:

      Admin db credentials

```
>db.createUser(
  {
    user: "admin",
    pwd: "Avanseus$0", #give complex passwords
    roles: [
    { role: "clusterAdmin", db: "admin" },
    { role: "userAdmin", db: "admin" }
  ]
 }
);
```

Update admin roles

```
>db.updateUser("admin",{roles :
["userAdminAnyDatabase","userAdmin","readWrite","dbAdmin","clusterAdmin","read
WriteAnyDatabase","dbAdminAnyDatabase"]});
```

User db credentials

```
>use testdb
db.createUser(
  {
   "user": "testdbuser",
   "pwd": "Avanseus$0",  #give complex passwords
   "roles": ["userAdmin", "dbAdmin", "readWrite"]}
  }
```

To connect to admin DB using mongo

```
mongo --authenticationDatabase admin -u admin -p 'Avanseus$0' --host localhost --
port 27017
```

To connect to test DB using mongo

```
mongo --authenticationDatabase testdb -u testdbuser -p 'Avanseus$0' --host
localhost --port 27017
```

e. Generate the key file using openssl tool and restart all the instances with this key file.

   Generate ssl file using openssl tool.

```
openssl rand -base64 756 > /home/ec2-user/mongodbKeyFile/keyFile
```

   Change the permission of this file to read only.

```
chmod 400 /home/ec2-user/mongodbKeyFile/keyFile
```

   Now kill all the instances and uncomment the below code in all the conf files (ConfigServers ,sharded clusters and query router).

```
security:
  keyFile: /home/ec2-user/mongodbKeyFile/keyFile
```

   Note: Make sure you have that key file on the right path and copy the same key file to other servers.

Once the above changes are made in all the conf files, restart the instance in this order.

1) Config servers,
2) Sharded clusters and
3) Query router.

f. To enable sharding at Database level, follow the below step:

```
>sh.enableSharding("testdb");
```

Once you enable sharding for a database, MongoDB assigns a primary shard for that database where MongoDB stores all data in that database.

g. Check sharding status using sh.status() command, You can see the database as being sharded and can check to which server it is sharded to.

```
>sh.status();
```

Output of above command would look like below:

```
--- Sharding Status ---
  sharding version: {
      "_id" : 1,
      "minCompatibleVersion" : 5,
      "currentVersion" : 6,
      "clusterId" : ObjectId("60df2de46f468b83fa1d5105")
  }
  shards:
        {  "_id" : "sharda",  "host" :
"sharda/172.31.31.7:27000,172.31.31.6:27000,172.31.31.5:27000",  "state" : 1 }
        {  "_id" : "shardb",  "host" :
"shardb/172.31.31.7:27100,172.31.31.6:27100,172.31.31.5:27100",  "state" : 1 }
        {  "_id" : "shardc",  "host" :
"shardc/172.31.31.7:27200,172.31.31.6:27200,172.31.31.5:27200",  "state" : 1 }
  active mongoses:
        "3.6.23" : 1
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  5
        Last reported error:  Could not find host matching read preference { mode:
"primary" } for set shardb
        Time of Reported error:  Sun Jul 04 2021 15:45:03 GMT+0200 (CEST)
        Migration Results for the last 24 hours:
            No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
            config.system.sessions
                  shard key: { "_id" : 1 }
                  unique: false
                  balancing: true
                  chunks:
                        sharda  512
                        shardb  512
                        shardc      512
                  too many chunks to print, use verbose if you want to force print
        {  "_id" : "testdb",  "primary" : "sharda",  "partitioned" : true }
```

h. To enable sharding at Collection level, add the shard key.

   To enable shard on a particular collection you need to provide a shard key. Ensure that you add the index to the field on which sharding is done.

   Note: Before you can shard a collection you must first enable sharding for the database where the collection resides.

   MongoDB provides two strategies to shard collections:

   i. Hashed sharding uses a hashed index of a single field as the shard key to partition data across your sharded cluster.

```
   > sh.shardCollection("<database>.<collection>", { <shard key field> : "hashed" } )
```

   Consider the below example:

   The sharding is performed on the Alarm collection with equipmentComponent_id Field. Provide a shard key as equipmentComponent_id.

   Create the collection Alarm.

```
   > db.createCollection("Alarm");
```

   Add hashed index to the euipmentComponent_id field.

```
   > db.Alarm.createIndex({equipmentComponent_id: "hashed"});
```

   Add the sharding key information

```
   > sh.shardCollection("testdb.Alarm", {equipmentComponent_id : "hashed"});
```

   Once this is done, insert the data into the Alarm table. Data will be evenly distributed across the shards in case of hashed sharding.

   ii. Range-based sharding can use multiple fields as the shard key and divides data into contiguous ranges determined by the shard key values.

```
   > sh.shardCollection("<database>.<collection>", { <shard key field> : 1, ... } )
```

Consider the below example:

The sharding is performed on the Alarm collection with equipmentComponent_id and cause_id field

Create the collection Alarm.

```
> db.createCollection("Alarm");
```

Add an index to the equipmentComponent_id and cause_id field".

```
> db.Alarm.createIndex({equipmentComponent_id: 1, cause_id:1 });
```

Add the sharding key information.

```
> sh.shardCollection("canDemo.Alarm", {equipmentComponent_id : 1,
cause_id:1});
```

Once this is done, insert the data into the Alarm table. Data might not be evenly distributed across the shards in the case of range-based sharding. One shard may have data than the other shards.