Universidad de las Américas Puebla
Cloud Computing and Big Data
LIS4102-1

Hands-On:

MongoDB Sharding

Author:
Fernando Nieto Morales (ID: 163930)

Professor:
Genoveva Vargas Solar

Introduction

MongoDB is a NoSQL document database based on storing BSON (a binary representation of data structures and maps) data structures with a dynamic schema, making data integration in certain applications easier and faster. It is characterized by its ad hoc queries, indexing, replication, sharding and aggregation.

Focusing on sharding, this implemented method on MongoDB distributes data across multiple machines to support deployments that implies big amounts of documents in the database and high throughput operations. A MongoDB sharded cluster is composed of 3 components:

1. Shards: Subsets of the sharded documents.
2. Mongos: Query routers used as interface between clients and the shards.
3. Config servers: Settings of the sharded cluster.

To distribute the documents through the shards, MongoDB uses shard keys, a set of fields of the documents, that are indexed by specific id's. The keys are selected through the cardinality (the maximum number of chunks the balancer can create), frequency (how often a given shard key value occurs in the data) and if is monotonically changing (distribute inserts to a single chunk in a cluster). This index improves the performance, efficiency, and scalability of a sharded cluster.

The sharding can be done by range-based or hashed. The range-based sharding, the default sharding methodology, consists of dividing all the data into contiguous ranges determined by the shard key values. On the other hand, the hashed sharding computes the hash value of a single field as the index value and used as a shard key. The hashed sharding works better if we want to equally distribute the data, focusing on broadcast operations. Meanwhile, the range-based sharding works better with targeted operations.

Methodology

To understand this process, I will create a sharded cluster with a query router, a config server and three shards. Aided by the Docker web service, I will do the next steps:

1. Start the cluster environment by connecting to the query router and add a shard named *shard1*.
2. Import the specified data (cities.txt) into the database and verify by executing the queries:
    a. *db.cities.find().pretty()*[1]
    b. db.cities.count()
3. Create a new collection named *cities1* and enable sharding on this collection using attribute state as shard key.
4. Populate the collection using the *cities* collection.
5. Repeat the step 3 naming the collection *cities2* and establishing the shard key as "*hash*".
6. Repeat the step 4 with *cities2*.
7. Create a new collection named *cities3* and balance the process by adding two shards: *shard2* & *shard3*.

[1] The pretty() method is to notchange the output format in mongosh.

8. Associate a specific tag for each shard (*shard2->"CA", shard2->"NY", shard3->"Others"*). Populate *shard2* & *shard3* with the same database and associate key ranges based on the tags values.
9. Disconnect the query router, stop containers and remove the docker images

Results

```
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("61feb02fdd3bacac4e2ae1c6")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1.docker:27017" }
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
```

Figure 1: Shard status after creating the cluster environment and adding one shard.

As we can see on the Figure 1, the sharding was successfully done in the "shards" section, where shard1 was deployed correctly from the database in docker.

```
mongos> db.cities.find().pretty()
{
        "_id" : "01012",
        "city" : "CHESTERFIELD",
        "loc" : [
                -72.833309,
                42.38167
        ],
        "pop" : 177,
        "state" : "MA"
}
```

Figure 2: Query 'a.' from step 2. Example of one result of the query.

```
mongos> db.cities.count()
29353
```

Figure 3: Query 'b.' from step 2.

Based on the information shown in each query, I can mention from the database that: There is a total of 29353 cities documented (Figure 3) where each city contains the next fields: ID, name, coordinates of its location, Post Office Protocol and state (Figure 2).

Figure 4: Databases shown by the method sh.status(). Create the collection "citites1" and enable its sharding with the attribute "state" as shard key.

From the sh.status() method on Figure 4 it is possible to see that a new collections was created with the name *cities1* inside the database *mydb*. This collection has the shard key with the attribute *state* and one shard. It only contains one chunk that has a range from the minimum value to the maximum value of the shard key field (*state*).

Figure 5: sh.status() after populating the collection "cities1".

After populating the collection, it added two chunks into the shard that used the key shard field "state" to establish the range of documents that each chunk will have. In other words, the first chunk has a range of the "state" value from the minimum value to the "MA" value, the second chunk has a range from "MA" to "RI", and the third is from "RI" to the maximum value.

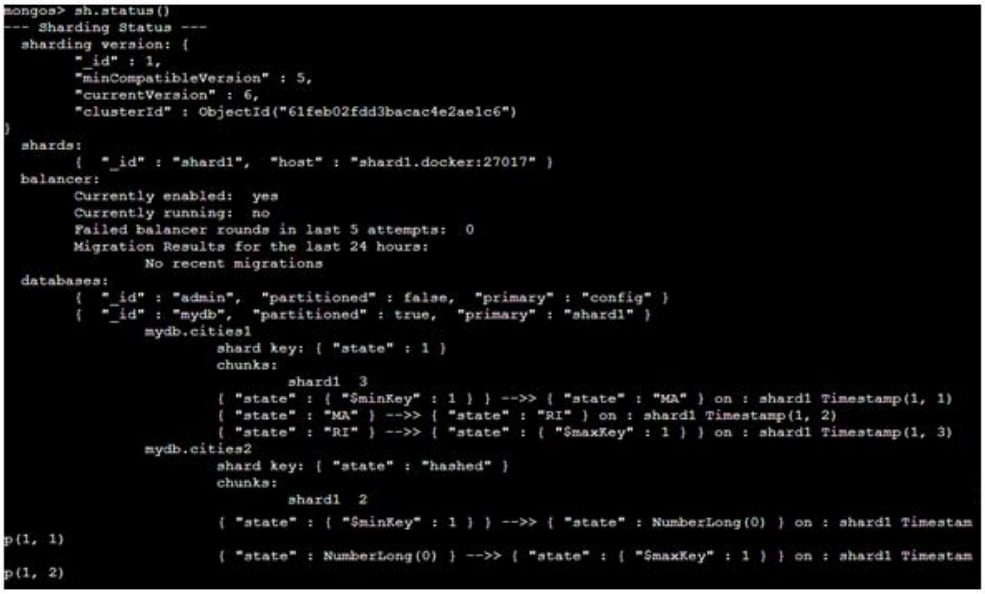

Figure 6: sh.status() after cretaing the collection "cities2".

The difference between cities1 and cities2 is the existence of a shard in cities2. It does not only contain the chunk from minimum value to maximum value, it also has a chunk with the NumberLong()[2] as the minimum key shard statement.

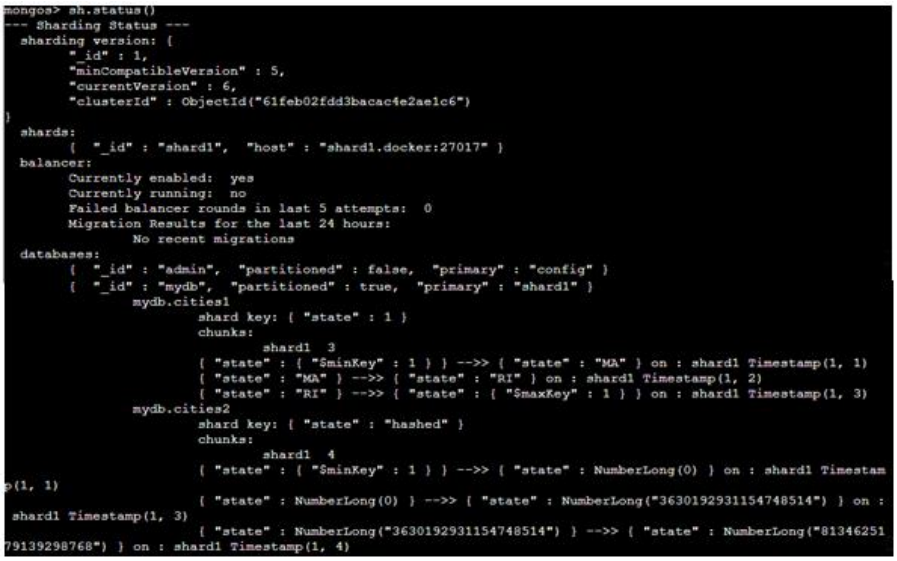

Figure 7: sh.status() after populating the collection "cities2".

Similar to Figure 5, on this collection it added two new chunks distributing the documents based on the NumberLong() value: From the minimum value to NumberLong(0), the second range is from NumberLong(0) to NumberLong("3630192931154748514"), and the third chunk from NumberLong("3630192931154748514") to NumberLong("8134625179139298768").

[2] Wrapper to handle 64-bit integers.

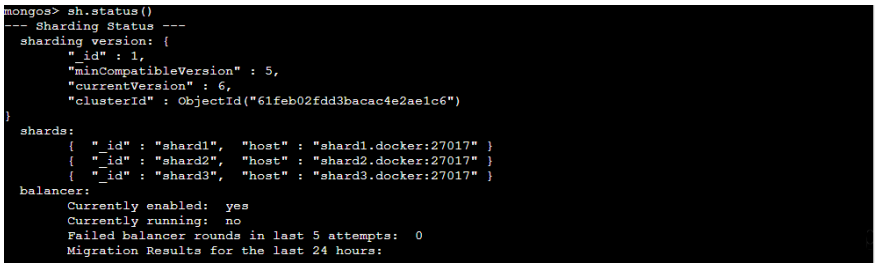

Figure 8: Add the shard2 & shard3.

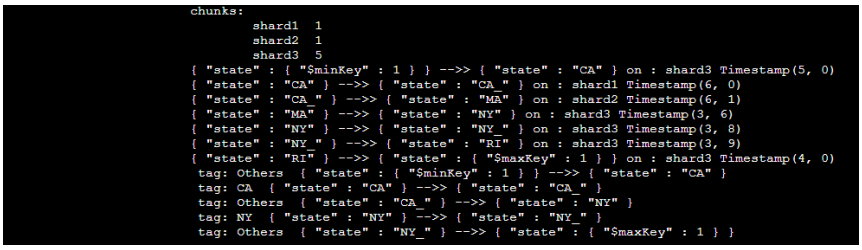

Figure 9: Set the tags into "cities3" and ranges for each chunk.

By establishing the range parameters of each chunk, it is possible to manipulate how the portioning will work. This balance between shards lets the client maintain control over the queries. On Figure 9, the ranges marked for each chunk are: Minimum value -> “CA” -> “CA_” -> “MA” -> “NY” -> “NY_” -> “RI” -> Maximum value. It also shows the referenced tags where the chunks that has a state different to “CA” or “NY” will be tagged in “Others”, any city with the previously mentioned states will be tagged as “CA” or “NY” respectively.

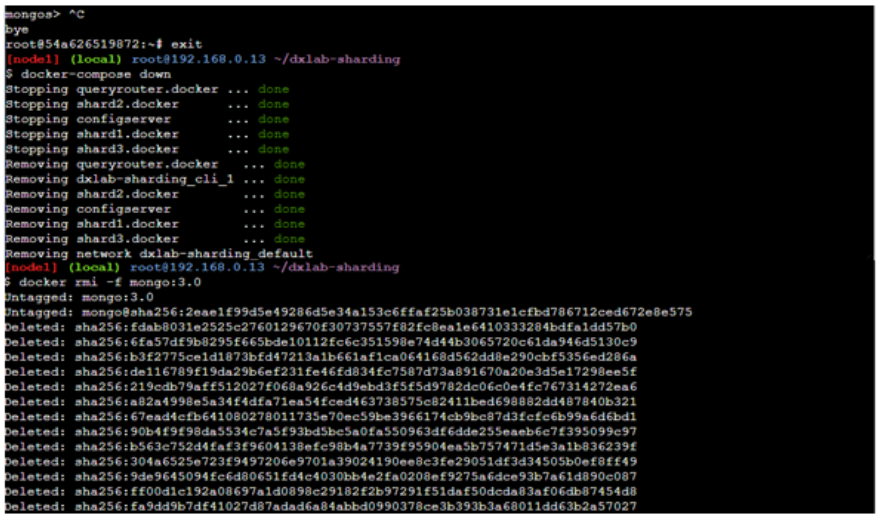

Figure 10: Disconnect the query router and remove the data.

Conclusion

The MongoDB attribute of sharding a database provides a more efficient method of selecting documents and distribute them among different servers. This logistic technique makes easier to ensure where to find each document. The “horizontal” workflow through different machines with tags lets great amount of data to be sharded correctly based on the client’s needs. The use of range-

based or hashed sharding will depend on the type of queries that will be constantly done, for example, range-based queries will work better if we want to create specific clusters where the queries will look for range of values, while hashed works better if the query will look for specific values.

To conclude, MongoDB provides good characteristics to work with databases of big data. Understanding how sharding works implies a better document distribution for each client.

References

MongoDB, Inc. (2021). *Sharding*. Retrieved february 13, 2022, from MongoDB: https://docs.mongodb.com/manual/sharding/