Mongo DB lab

Author : Amat François

Contact : amat.francois@gmail.com

This rapport can be found in markdown and pdf here

Table of Contents

Install MongoDB :

- Create a directory for the server and launch it. Write down the command for launching it; on which port it runs?
- Import the document moviepeople-10.json into the server
- Launch a client (mongo shell), retrieve all the documents by asking a query in the client.

Installation of mongodb on mac :

```
brew install mongodb
```

Getting started in the lab :

```
# create a special folder for mongodb
sudo mkdir -p ./data/db
sudo chown -R `id -un` ./data/db
mongod  -dbpath ./data/db &

mkdir mongodb; cd mongodb
wget https://www.lri.fr/\~maniu/labJSON.zip
unzip labJSON.zip
rm -rf __MACOSX
rm  labJSON.zip
# mongoimport  labJSON/moviepeople-1000.json
# To import moviepeople-1000 as a collection
# mongo does not like number in its collection name.
mongoimport  --db test --collection movie --type json   labJSON/moviepeople-1000.js

# to show dbs :
mongo # we enter mongo shell
show dbs
use test
show collections
db.movie.find() # this will return  all the documents
```

Import query

Import the documents moviepeople-3000.json and cities.json into the server

```
mongoimport  --db test --collection movie --type json   labJSON/moviepeople-3000.js
mongoimport  --db test --collection cities --type json   labJSON/cities.json
```

• In the mongo shell client, write queries for finding:

2. The person named Anabela Teixeira

```
template: db.inventory.find( { status: "D" } )
db.movie.find( { "person-name": "Anabela, Teixeira" } )
db.movie.find( { "person-name": { $regex: /(.*Teixeira.*Anabela.*)/ } } )
db.movie.findOne({ "person-name": { $regex: /(.*[Teixeira|Anabela], [Teixeira|Anabe
```

3. The birthplace of Steven Spielberg
   `db.movie.findOne({ "person-name": { $regex: /(.*[Steven|Spielberg], [Steven|Spielberg].*)/ } }).info.birthnotes db.movie.findOne({ "person-name": "Steven, Spielberg"}) db.movie.findOne({ "person-name": "Spielberg, Steven"}) db.movie.findOne({ "person-name": "Spielberg, Steven"}).info.birthnotes # correct one`
4. The number of people born in Lisbon

```
db.movie.count({"info.birthnotes": "Lisbon, Portugal"})
```

5. The people taller than 170 cm

```
db.movie.find({"info.height":{ $gt: "170"}})
```

It can be noted that due to the fact that some height are in inches, part of the result is false.

6. The names of people whose information contains "Opera"
   `db.movie.find( { "info.trivia": { $regex: /(.*Opera.*)/ } } )`

More Queries

7. For each movie person whose birthplace is known, find the latitude, longitude and population of that city (if that information exists in the city document) You may use functions, several commands, aggregates, lookups etc.

```
db.movie.find({"info.birthnotes":{$exists: true}}).forEach(
  function(movie){
    birthnotes = movie.info.birthnotes[0].split(',')[0]
    city = db.cities.findOne({"name":birthnotes})
```

```
    if(city != null)
      print("movie",movie.info.birthnotes, city.location.latitude,city.location.lon
  }
  )
```

MongoDB replication

• Create working directories for 3 MongoDB servers

```
sudo mkdir -p ./data/db1
sudo mkdir -p ./data/db2
sudo mkdir -p ./data/db3
sudo chown -R `id -un` ./data/db1
sudo chown -R `id -un` ./data/db2
sudo chown -R `id -un` ./data/db3
```

• Create a replication set for a collection named small- movie

```
mongod --replSet small-movie --dbpath ./data/db1 --port 27011
mongod --replSet small-movie --dbpath ./data/db2 --port 27012
mongod --replSet small-movie --dbpath ./data/db3 --port 27013
```

• Launch 3 MongoDB servers (in different shells). The server program is mongod. Leave those shells alone.

• Connect a client (mongo) to one server. Through the client, initialize the replication: add the other replica server, and the arbiter.

```
mongo localhost:27011

rs.initiate({_id: 'small-movie',
members: [ {_id: 1, host: 'localhost:27011'}, {_id: 2, host: 'localhost:27012'},
{_id: 3, host: 'localhost:27013'} ] })
```

```
answer : > rs.initiate({_id: 'small-movie',
... members: [ {_id: 1, host: 'localhost:27011'}, {_id: 2, host: 'localhost:27012'}
... {_id: 3, host: 'localhost:27013'} ] })
{
```

```
    "ok" : 1,
    "operationTime" : Timestamp(1538488558, 1),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1538488558, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}
small-movie:SECONDARY>
```

Just after we initiate, the server become secondary and show it to the command prompt.

```
rs.conf()
{
    "_id" : "small-movie",
    "version" : 1,
    "protocolVersion" : NumberLong(1),
    "writeConcernMajorityJournalDefault" : true,
    "members" : [
        {
            "_id" : 1,
            "host" : "localhost:27011",
            "arbiterOnly" : false,
            "buildIndexes" : true,
            "hidden" : false,
            "priority" : 1,
            "tags" : {

            },
            "slaveDelay" : NumberLong(0),
            "votes" : 1
        },
        {
            "_id" : 2,
            "host" : "localhost:27012",
            "arbiterOnly" : false,
            "buildIndexes" : true,
            "hidden" : false,
            "priority" : 1,
            "tags" : {

            },
            "slaveDelay" : NumberLong(0),
            "votes" : 1
        },
        {
            "_id" : 3,
            "host" : "localhost:27013",
            "arbiterOnly" : false,
```

```
            "buildIndexes" : true,
            "hidden" : false,
            "priority" : 1,
            "tags" : {

            },
            "slaveDelay" : NumberLong(0),
            "votes" : 1
        }
    ],
    "settings" : {
        "chainingAllowed" : true,
        "heartbeatIntervalMillis" : 2000,
        "heartbeatTimeoutSecs" : 10,
        "electionTimeoutMillis" : 10000,
        "catchUpTimeoutMillis" : -1,
        "catchUpTakeoverDelayMillis" : 30000,
        "getLastErrorModes" : {

        },
        "getLastErrorDefaults" : {
            "w" : 1,
            "wtimeout" : 0
        },
        "replicaSetId" : ObjectId("5bb378eec22addc10c10aa47")
    }
}
```

• Identify the master from the outputs in the servers' shell and by requesting replica set information from the servers.

```
rs.status()
{
 "set" : "small-movie",
 "date" : ISODate("2018-10-02T14:01:06.367Z"),
 "myState" : 1,
 "term" : NumberLong(1),
 "syncingTo" : "",
 "syncSourceHost" : "",
 "syncSourceId" : -1,
 "heartbeatIntervalMillis" : NumberLong(2000),
 "optimes" : {
   "lastCommittedOpTime" : {
     "ts" : Timestamp(1538488861, 1),
     "t" : NumberLong(1)
   },
   "readConcernMajorityOpTime" : {
     "ts" : Timestamp(1538488861, 1),
     "t" : NumberLong(1)
   },
   "appliedOpTime" : {
     "ts" : Timestamp(1538488861, 1),
```

```
        "t" : NumberLong(1)
      },
      "durableOpTime" : {
        "ts" : Timestamp(1538488861, 1),
        "t" : NumberLong(1)
      }
    },
    "lastStableCheckpointTimestamp" : Timestamp(1538488811, 1),
    "members" : [
      {
        "_id" : 1,
        "name" : "localhost:27011",
        "health" : 1,
        "state" : 1,
        "stateStr" : "PRIMARY",
        "uptime" : 398,
        "optime" : {
          "ts" : Timestamp(1538488861, 1),
          "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2018-10-02T14:01:01Z"),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "electionTime" : Timestamp(1538488569, 1),
        "electionDate" : ISODate("2018-10-02T13:56:09Z"),
        "configVersion" : 1,
        "self" : true,
        "lastHeartbeatMessage" : ""
      },
      {
        "_id" : 2,
        "name" : "localhost:27012",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 307,
        "optime" : {
          "ts" : Timestamp(1538488861, 1),
          "t" : NumberLong(1)
        },
        "optimeDurable" : {
          "ts" : Timestamp(1538488861, 1),
          "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2018-10-02T14:01:01Z"),
        "optimeDurableDate" : ISODate("2018-10-02T14:01:01Z"),
        "lastHeartbeat" : ISODate("2018-10-02T14:01:05.906Z"),
        "lastHeartbeatRecv" : ISODate("2018-10-02T14:01:06.140Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "localhost:27011",
        "syncSourceHost" : "localhost:27011",
        "syncSourceId" : 1,
        "infoMessage" : "",
```

```
      "configVersion" : 1
    },
    {
      "_id" : 3,
      "name" : "localhost:27013",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 307,
      "optime" : {
        "ts" : Timestamp(1538488861, 1),
        "t" : NumberLong(1)
      },
      "optimeDurable" : {
        "ts" : Timestamp(1538488861, 1),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2018-10-02T14:01:01Z"),
      "optimeDurableDate" : ISODate("2018-10-02T14:01:01Z"),
      "lastHeartbeat" : ISODate("2018-10-02T14:01:05.906Z"),
      "lastHeartbeatRecv" : ISODate("2018-10-02T14:01:06.140Z"),
      "pingMs" : NumberLong(0),
      "lastHeartbeatMessage" : "",
      "syncingTo" : "localhost:27011",
      "syncSourceHost" : "localhost:27011",
      "syncSourceId" : 1,
      "infoMessage" : "",
      "configVersion" : 1
    }
  ],
  "ok" : 1,
  "operationTime" : Timestamp(1538488861, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1538488861, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

I Identify the server 1 (port 27011) as master (PRIMARY), That is shown in the latter command (rs.status) but also in the command prompt that became "small-movie:PRIMARY>"
In the following there will be a selected log ouput to show the mecanism of the replication.
The full logs can be found here

OUTPUT SERVER 1

```
I REPL     [conn1] replSetInitiate admin command received from client
I REPL     [conn1] replSetInitiate config object with 3 members parses ok
I ASIO     [Replication] Connecting to localhost:27012
```

```
I ASIO     [Replication] Connecting to localhost:27013
[...]

I REPL     [replexec-0] Starting an election, since we've seen no PRIMARY in the pa
I REPL     [replexec-0] conducting a dry run election to see if we could be elected
[...]

I REPL     [replexec-2] election succeeded, assuming primary role in term 1
I REPL     [replexec-2] transition to PRIMARY from SECONDARY
```

OUTPUT SERVER 2

```
[...]
I REPL     [replexec-3] Member localhost:27011 is now in state PRIMARY
[...]
```

OUTPUT Server 3

```
[...]
I REPL     [rsBackgroundSync] Changed sync source from empty to localhost:27011
[...]
```

Import moviepeople-10.json through the master; note the output of the two other servers.

```
mongoimport --host localhost:27011  --db test --collection movie --type json   labJ
```

SERVER 1

```
[...]
 I COMMAND  [conn59] command test.movie command:
 insert { insert: "movie", writeConcern: { getLastError: 1, w: "majority" }, ordere
    ninserted:10 keysInserted:10 numYields:0 reslen:245 locks:{ Global:
      {
       acquireCount: { r: 5, w: 5 } }, Database: { acquireCount: { w: 4, W: 1 } },
       Collection: { acquireCount: { w: 2 } }, oplog: { acquireCount: { w: 2 } }
     }
       protocol:op_query 333ms

[...]
```

SERVER 2

```
[...]
I STORAGE  [repl writer worker 10] createCollection: test.movie with provided UUID:
[...]
```

SERVER 3

```
[...]
 I STORAGE  [repl writer worker 9] createCollection: test.movie with provided UUID:
[...]
```

When the master is killed (server 1):

The two others servers really don't like it :

SERVER 2

```
2018-10-02T16:21:11.558+0200 I ASIO     [Replication] Failed to connect to localhos
2018-10-02T16:21:11.558+0200 I ASIO     [Replication] Dropping all poole
```

SERVER 3

```
2018-10-02T16:21:48.550+0200 I ASIO     [Replication] Connecting to localhost:27011
2018-10-02T16:21:48.551+0200 I ASIO     [Replication] Failed to connect to localhos
```

I should have observed a mecanism with another vote to determine where should be the next primary server but the
servers 2 and 3 were traps in a hostUnreachable error loop.
The servers 2 and 3 might be in a mode where they wait for server 1 to be up again in the network but after 10/15minutes no other votes took place.

Sharding

Start two shard servers; shard the cities by the country.

In mongodb v4.0.2 we have to define the shard server as replica set in order to map them to the
router.
I run into quite a lot of problems due to the fact that the replica set servers used did not want to change to primary for some reason, that's why I needed to use a third server in order to get a second primary server. I got this problem even by setting a configuration with rs.config as above.
I need to use 8 terminals in order to launch the three replicat sets, the config server, a router server and 3 terminals to connect to the two primary replicat sets (to configure them) and to connect to the router server and do the sharding operations.

```
mkdir -p ./data/db4
mkdir -p ./data/db5
mkdir -p ./data/db6

sudo chmod -R go+w ./data/db4
sudo chmod -R go+w ./data/db5
sudo chmod -R go+w ./data/db6

# to be open in a (new) terminal
mongod --shardsvr --replSet rs1 --dbpath ./data/db4 --port 27014 --logpath=../logs/

# to be open in a (new) terminal
mongod --shardsvr --replSet rs2 --dbpath ./data/db5 --port 27015 --logpath=../logs/

# to be open in a (new) terminal
mongod --shardsvr --replSet rs1 --dbpath ./data/db6 --port 27019 --logpath=../logs/

mkdir ./mongoconfig

mongod --configsvr  --replSet c1 --dbpath ./mongoconfig --port 27016 &



#mongo --port 27014 --eval "rs.initiate({ _id: "rs1", members: [ { _id: 0, host: #"
#mongo --port 27015 --eval "rs.initiate({ _id: "rs2", members: [ { _id: 0, host: #"



mongo --port 27014 --eval "rs.initiate()"
```

```
mongo --port 27015 --eval "rs.initiate()"
mongo --port 27019 --eval "rs.initiate()"
# this will set the two primary servers 27015 and (27014|27019)
# In my case it was 27019

# we also need to initiate the config server and allow to connect to the router ser

mongo --port 27016 --eval "rs.initiate()"
mongo --port 27016 --eval "rs.add("local:27020")"

# launch of the router server
mongos --configdb c1/localhost:27016  --port 27020


mongo localhost:27020/admin
# the following commands are in the 27020 shell
sh.addShard( "rs1/localhost:27014");
sh.addShard( "rs2/localhost:27015");
sh.addShard( "rs2/localhost:27019");

db.runCommand({ enablesharding: "test"})
db.runCommand( { shardcollection : "test.cities", key : {country : 1} } )
```

Here are the results from the router server :

```
sh.addShard( "rs1/localhost:27019");
{
    "shardAdded" : "rs1",
    "ok" : 1,
    "operationTime" : Timestamp(1539280689, 5),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1539280689, 5),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}
mongos> sh.addShard( "rs1/localhost:27015");
{
    "shardAdded" : "rs1",
    "ok" : 1,
    "operationTime" : Timestamp(1539280703, 1),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1539280703, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}
```

```
mongos> db.runCommand({ enablesharding: "test"})
{
    "ok" : 1,
    "operationTime" : Timestamp(1539280719, 5),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1539280719, 5),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}
mongos> db.runCommand( { shardcollection : "test.cities", key : {country : 1} } )
{
    "collectionsharded" : "test.cities",
    "collectionUUID" : UUID("df2af57c-fd96-4dae-bfa9-e1cb3df7043d"),
    "ok" : 1,
    "operationTime" : Timestamp(1539280877, 14),
    "$clusterTime" : {
        "clusterTime" : Timestamp(1539280877, 14),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}
```

Conclusions :

This lab let us getting started with mongodb and use its principal functionalities such as how to import a database from a json, how to set up replicat sets or basic sharding.
It can be denoted that the sharding part took me a few hours because the code templated we had was outdated for a former version of mongoDB and then I ran into issues about setting primary into the replicat servers.
The new step would be to implement this with clean config files and test it with docker.

Sources :

Lab link

Course link

Full logs