Data Management and Visualization

Politecnico di Torino

Create a replica set and query a MongoDB database – Practice 5

Goal

The objective of the practice is to configure a replica set in a local environment and connect to the MongoDB database. Then, create and successfully populate a collection of documents. Finally, query the database exploiting different MongoDB functionalities and patterns.

Initial set-up

Two different procedures can be carried out to solve this practice:

1. one exploits MongoDB Server and the shell interface,
2. another exploits Docker to create the replica set

This first procedure is the most general one and can be deployed, theoretically, on any operating systems. However, especially for Windows OS, we could encounter some problems running instances with access control disable. For this reason, we propose an alternative procedure which works properly on Window.

For the first procedure, to create a replica set we will use:

- a command line interpreter
- MongoDB Compass (https://www.mongodb.com/try/download/compass)
- MongoDB Community Server (https://www.mongodb.com/try/download/community?tck=docs_server)

If you want to try out this practice on your own machine, you need to be sure to have all these programs installed and up to date.
You can easily follow the guide https://docs.mongodb.com/manual/installation/ to download the required material.

For the second procedure, to create a replica set we will use the Docker Platform.

Linux requirements: Docker, Docker Compose
Windows requirements: Docker Toolbox for Windows 10 or older.
(for Windows 10 Pro or Enterprise download and run Docker Desktop)

Mac requirements: Docker Desktop (includes Docker Compose)

Place the Docker Compose file (**docker-compose.yml**, provided with practice) in a folder (e.g., $HOME/MongoDB).

Open a terminal in the folder where is located the docker-compose.yml file and run the following command (if the Docker Containers do not start, try renaming the file **docker-compose.yml** into **docker-compose.yaml**):

```
docker-compose up -d
```

The docker services will run in detached mode. If you want to see the output of each container, remove the option -d form the previous command.

Configure the replica set (1st Procedure)

First of all, we need to create the necessary data directories for each member by issuing the following command:

```
mkdir mongodb\polimongodb1 mongodb\polimongodb2 mongodb\polimongodb3
```

Note: if you are using Linux, remember to replace "\" with "/".

Then, we need to start our 3 different mongodb instances in their own shell windows by issuing the following commands:

- mongod --replSet rspoli --port 27017 --bind_ip localhost --dbpath mongodb\polimongodb1 --oplogSize 128
- mongod --replSet rspoli --port 27018 --bind_ip localhost --dbpath mongodb\polimongodb2 --oplogSize 128
- mongod --replSet rspoli --port 27019 --bind_ip localhost --dbpath mongodb\polimongodb3 –oplogSize 128

Note: if you are having some problems connecting to one of the instances, try to replace the value of the "bind_ip" parameters with "0.0.0.0", so that it accepts connection from any IP address.

From now on, we will exploit the Mongo Shell (mongosh) provided by MongoDB Compass.

1) Open MongoDB Compass and connect with one of the instances. Fill out the connections fields individually setting:

   Hostname: localhost
   Port: 27017
   SRV Record: off
   Authentication: None

2) Configure the replica set with the following requirements:
    a) replica set name should be equal to *rspoli*
    b) the priority of polimongodb1 instance should the highest one
    c) the priority of polimongodb3 instance should the lower one

   In order to configure and initiate the replica set you need to:
    - define the configuration in *rsconf*
    - issue the *rs.initiate(rsconf)* command

   To check if your configuration has been correctly deployed, use *rs.conf()*

3) Check the configuration of your replica set with the following command:

```
rs.status()
```

4) Shut down from the windows shell of the primary node. To shut down you just need to interrupt the process with CTRL+C.

5) Identify the new primary node: check your configuration once again and see which one of the remaining nodes has been elected as primary.

6) Restart the shut-down node using the previous command to start the mongod instance

7) Re-check the status of replica set

Configure the replica set (2nd procedure)

The docker-compose.yml file defines 3 instances of MongoDB. The name of each instance can be retrieved using the command:

```
docker ps
```

All the following commands should be launched inside the folder of the project (e.g., $HOME/MongoDB).

1) Connect to the Mongo shell of one instance using the following command

```
docker-compose exec name_docker_mongo mongo
```

   where *name_docker_mongo* is the name of one mongo instance.

2) Configure the replica set with the following requirements:
    a) replica set name should be equal to *rspoli*
    b) the priority of polimongodb1 instance should the highest one
    c) the priority of polimongodb3 instance should the lower one
3) Check the configuration with the following command:

```
rs.status()
```

4) Shut down from a new shell the primary node. To shut down a node use the following command

```
docker-compose stop name_primary_mongo
```

where *name_docker_mongo* is the name of the mongo container running the primary node.

5) Identify the new primary node
6) Restart the shutted down node

```
docker-compose restart name_primary_mongo
```

7) Re-check the status of replica set

Create a database

1) Connect to primary node
2) Create a new database for the practice. (use restaurants)
3) Populate the newly created database by inserting the documents in the **restaurants_collection.txt** (download it from the course website). use the command:

```
db.restaurants.insertMany(<file content>)
```

4) In order to check the success of the insert, run the command:

```
db.restaurants.find().pretty()
```

Running queries of interest

Each document of the collection has a structure with the following fields:

```
{_id: <ObjectId>,
name: <string>, // name of the restaurant
tag: <list[string]>, // tags assigned by the users
orderNeeded: <boolean>, // if the user should reserve
maxPeople:<int>, // maximum number of customers
review:<float>, // average vote
cost:<string>, // classification of the menu price. Categories are: low, medium and high
location:{type:"Point",coordinates:[<lat>,<long>]}, // geographical point
contact:{
    phone:<string>, // telephone of the restaurant
```

facebook:<string> // link to the facebook page

}

}

Run the following queries of interest:

1) Find all restaurants whose cost is medium. Show the result in the pretty format.
2) Select the name and the number of seats (maxPeople) available of all the restaurants whose review is bigger than 4 and cost is medium or low
3) Select the name and the phone of the restaurants that can contain more than 5 people and:
    a) whose tag contains "italian" or "japanese" and cost is medium or high OR
    b) whose tag does not contain neither "italian" nor "japanese", and whose review score is higher than 4.5

   Remove from the output the field _id.
4) Calculate the average review of all restaurants
5) Count the number of restaurants whose review is higher than 4.5 and can contain more than 5 people
6) Find the restaurant in the collection which is nearest to the point [45.0644, 7.6598]

   Hint: remember to create the geospatial index.
7) Find how many restaurants in the collection are within 500 meters from the point [45.0623, 7.6627]
8) Add the tag "pizza" to all the restaurants that contain the tag "italian". If the tag "pizza" is already present, you should not insert it.
9) Decrease the review score of 0.2 for all the restaurants with the tag 'fastfood'
10) For only the restaurants with a review higher than 3, find the tags which appear more than 1 time. For each tag, show how many documents include it.
11) For each cost category, compute the minimum review rate, the maximum review rate, the average review rate and the number of restaurants. Sort the result in descending order according to the number of restaurants in each cost category.
12) Find the median value of maxPeople attribute.