Universidad de las Américas Puebla

LIS4112 Cloud Computing and Big Data

José Antonio Solís Martínez

ID: 162442

Hands On 1: MongoDB Sharding

20/02/2022

Exercise walkthrough

The first step is logging into the https://labs.play-with-docker.com/ website and creating a new instance, from these new instance is where the following commands are executed.

Installation and configuration

The first step is to get the hands-on material from the GitHub repository. To do this, we run the command:

```
git clone https://github.com/javieraespinosa/dxlab-sharding.git
```

Then we download the MongoDB docker image, since this hands-on is about sharding in MongoDB

```
# Enter lab folder
cd dxlab-sharding

# Downloads images specified in docker-compose.yml
docker-compose pull
```

To verify that the image was downloaded correctly, we run the following command:

```
docker images
```

Now that we have installed all the prerequisites, we can move on to playing with sharding. The execution of the above steps can be seen in the image below.

```
[node1] (local) root@192.168.0.8 ~
$ git clone https://github.com/javieraespinosa/dxlab-sharding.git
Cloning into 'dxlab-sharding'...
remote: Enumerating objects: 77, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 77 (delta 10), reused 3 (delta 0), pack-reused 55
Receiving objects: 100% (77/77), 747.58 KiB | 18.23 MiB/s, done.
Resolving deltas: 100% (41/41), done.
[node1] (local) root@192.168.0.8 ~
$ cd dxlab-sharding/
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$ docker-compose pull
Pulling cli                ... done
Pulling configserver       ... done
Pulling queryrouter.docker ... done
Pulling shard1.docker      ... done
Pulling shard2.docker      ... done
Pulling shard3.docker      ... done
```

```
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
mongo        3.0       fdab8031e252   3 years ago   232MB
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$
```

Preparing a Sharded Cluster

The material explains that this cluster is partially configured already, composed of:

- 1 query router
- 1 config server
- 3 shards

The first step for this section is to start the cluster using the following command:

```
# Start docker containers
docker-compose up -d
```

We can see a list of docker containers and their IP's using the following commands:

```
# List containers
docker ps

# List containers IPs
docker network inspect dxlab-sharding_default
```

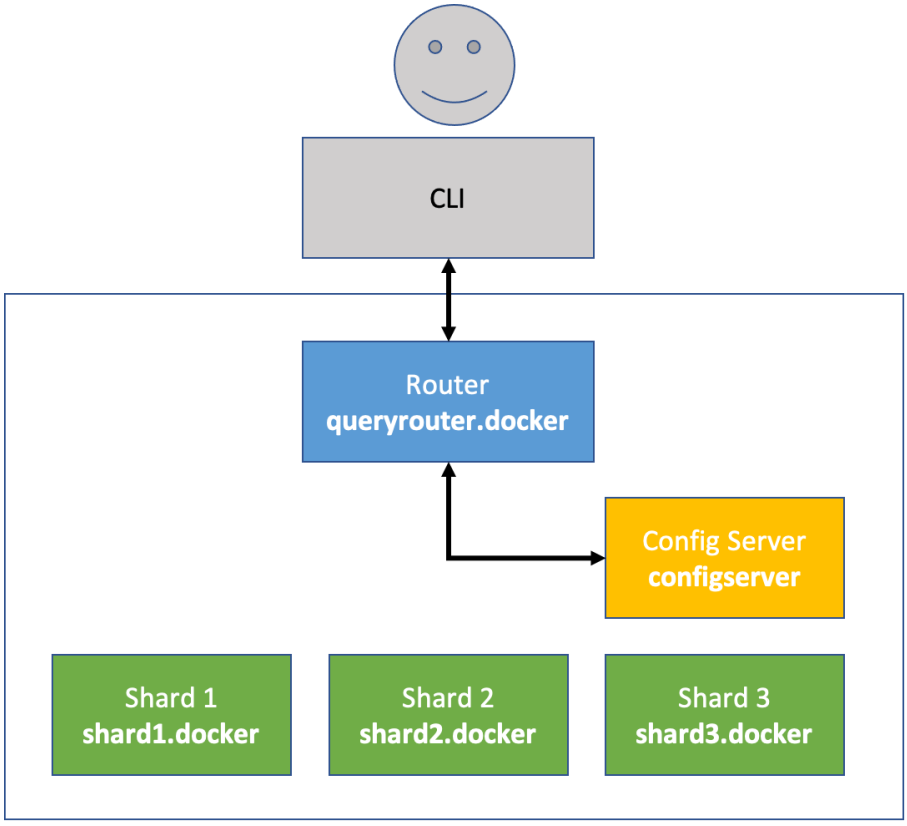
The execution is show below:

```
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$ docker-compose up -d
Creating network "dxlab-sharding_default" with the default driver
Creating dxlab-sharding_cli_1 ... done
Creating shard3.docker        ... done
Creating shard1.docker        ... done
Creating shard2.docker        ... done
Creating configserver         ... done
Creating queryrouter.docker   ... done
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED
 STATUS              PORTS                       NAMES
0880fb57e907   mongo:3.0   "docker-entrypoint.s…"   About a minute ago
 Up About a minute   0.0.0.0:27017->27017/tcp   queryrouter.docker
3a5cb3edfa92   mongo:3.0   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:27118->27017
/tcp   shard2.docker
92f71d18dbed   mongo:3.0   "docker-entrypoint.s…"   About a minute ago   Up About a minute   27017/tcp
       configserver
ce0517b92dfc   mongo:3.0   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:27119->27017
/tcp   shard3.docker
c34794cddf12   mongo:3.0   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:27117->27017
/tcp   shard1.docker
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$ docker network inspect dxlab-sharding_default
[
    {
        "Name": "dxlab-sharding_default",
        "Id": "722cd58efeb02fb89f8ea9c73a550215e745b839623dd292a8343c93148904f9",
        "Created": "2022-02-18T04:58:04.222163896Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0880fb57e907e1351e9111366a2a3248c7c45af0654b04fbb4773e7848a0f585": {
                "Name": "queryrouter.docker",
                "EndpointID": "1914f2ac3f7714abf66b2f99663925cfab308c52d501a669e0820ce0522b3225",
                "MacAddress": "02:42:ac:13:00:07",
                "IPv4Address": "172.19.0.7/16",
                "IPv6Address": ""
```

```
            },
            "3a5cb3edfa92bfc79000a19d091071ca94dae90155f6f43bc1a313a1c5eda8c2": {
                "Name": "shard2.docker",
                "EndpointID": "dc732eb5872cc216f4ad269c5a46d383bb65cd08d2b6afe2cf82754c2aafcd00",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            },
            "92f71d18dbed24406a974dc510d43beff98a0178aba6d4b7b0d5761c4cb5150f": {
                "Name": "configserver",
                "EndpointID": "6bba3d0d2980f4a5304cf9601823f9c786505a93742fab642ca2dafd3705c9c9",
                "MacAddress": "02:42:ac:13:00:06",
                "IPv4Address": "172.19.0.6/16",
                "IPv6Address": ""
            },
            "c34794cddf12858f726a097a1271af4d402a25a4683468aa35461f2c4bb8d43d": {
                "Name": "shard1.docker",
                "EndpointID": "7636facf04294c9ef126c71191fe417785e7589645bb9c6438b51107ce27fa08",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            },
            "ce0517b92dfcae50f7087b6ee0f3d6ed3f6e57e3cf476ba294112cb20a4f7060": {
                "Name": "shard3.docker",
                "EndpointID": "b5cb315baa788fb5265adfb917333e976c83c900d96740f350ff9e3acbcf25b3",
                "MacAddress": "02:42:ac:13:00:04",
                "IPv4Address": "172.19.0.4/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "default",
            "com.docker.compose.project": "dxlab-sharding",
            "com.docker.compose.version": "1.26.0"
        }
    }
]
```


Cluster configuration

The figure below shows how the query router and config server are the only components configured in the cluster. To complete it, this steps adds a shard server.



First we have to enter the cluster environment:

```
docker-compose run --rm cli
```

Then we connect to the query router:

```
mongo --host queryrouter.docker
```

Inside the query router we can add a shard to the cluster. For this example, the shard is named `shard1`.

```
// Change database
use admin

// Add shard to cluster
db.runCommand({
  addShard: "shard1.docker",
  name: "shard1"
})
```

To verify our changes and the cluster status, we run the following command:

```
sh.status()
```

Note: To disconnect from the query router press `ctrl + c`

> **Q1. Which is the important information reported by sh.status()**
> *A. The shards that exist on the server, the status of the server (which in our case is enabled but not running yet) and a list of the databases*

Execution:

```
[node1] (local) root@192.168.0.8 ~/dxlab-sharding
$ docker-compose run --rm cli
root@5785f177f648:~# mongo --host queryrouter.docker
MongoDB shell version: 3.0.15
connecting to: queryrouter.docker:27017/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
mongos> use admin
switched to db admin
mongos> db.runCommand({ addShard: "shard1.docker", name: "shard1" })
{ "shardAdded" : "shard1", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
  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" }

mongos>
```

Inserting and querying data

For this step we have to import data into our cluster, using the following command:

```
mongoimport \
  --host queryrouter.docker \
  --db mydb \
  --collection cities \
  --file ./cities.txt
```

Now with the content of cities.txt imported into the cities collection on the mydb database, we can check if the operation worked by connecting to the query router:

```
# Connect to query router
mongo --host queryrouter.docker
```

Then we can see the collections in the database:

```
# List databases
show dbs

# Change to database mydb
use mydb

# List collections
show collections
```

And we can execute some queries over the cities collection:

```
# SELECT * FROM Cities
db.cities.find().pretty()

# SELECT COUNT(*) FROM Cities
db.cities.count()
```

> **Q2. Describe in natural language the content of the collection**
> *A. For each city the collection contains the internal id, the name of the city, where it is located by its coordinates, the population, and in what state it is located.

Execution:

```
root@5785f177f648:~# mongoimport \
>    --host queryrouter.docker \
>    --db mydb \
>    --collection cities \
>    --file ./cities.txt
2022-02-18T05:31:59.073+0000    connected to: queryrouter.docker
2022-02-18T05:31:59.807+0000    imported 29353 documents
root@5785f177f648:~# mongo --host queryrouter.docker
MongoDB shell version: 3.0.15
connecting to: queryrouter.docker:27017/test
mongos> show dbs
admin   (empty)
config  0.016GB
mydb    0.078GB
mongos> use mydb
switched to db mydb
mongos> show collections
cities
system.indexes
```

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

```
Type "it" for more
mongos> db.cities.count()
29353
mongos>
```

Sharding Database Collections

MongoDB uses two kinds of partitioning strategies based on the shard key (attribute that must exist in every document in a collection.

Range Based Partitioning

In this strategy, the data is partitioned into intervals called chunks. Lets explore this strategy.

We create a new collection:

```
db.createCollection("cities1")
show collections
```

Then we enable sharding by using the attribute `state` as the shard key:

```
sh.enableSharding("mydb")
sh.shardCollection("mydb.cities1", { "state": 1} )
```

Verify the cluster state:

```
sh.status()
```

> **Q3. How many chunks did you create? Which are their associated ranges? Include a screen copy of the results of the command in your answer to support your answer.**
> *A. Only one chunk was created, and the range is min: 1, max: 1. However, this makes sense as the collection is empty. We can expect to see more information once we populate the collection.*

Then we populate the collection using the collection from `mydb`:

```
db.cities.find().forEach(
   function(doc) {
      db.cities1.insert(doc);
   }
)
```

And again we verify this new state:

```
sh.status()
```

> **Q4. How many chunks are there now? Which are their associated ranges? Which changes can you identify in particular? Include a screen copy of the results of the**

> **command in your answer to support your answer.**
> *A. Now with the collection populated we have 3 chunks. The first one goes from the start of the collection until "MA". The second goes from "MA" to "RI". And the third chunk goes from "RI" to the end of the collection.*

Execution:

```
mongos> db.createCollection("cities1")
{ "ok" : 1 }
mongos> show collections
cities
cities1
system.indexes
mongos> sh.enableSharding("mydb")
{ "ok" : 1 }
mongos> sh.shardCollection("mydb.cities1", { "state": 1} )
{ "collectionsharded" : "mydb.cities1", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
  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
" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1"
}
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  1
                        { "state" : { "$minKey" : 1 } } -->> { "state" :
 { "$maxKey" : 1 } } on : shard1 Timestamp(1, 0)
```

```
mongos> db.cities.find().forEach(
...     function(doc) {
...        db.cities1.insert(doc);
...     }
... )

mongos>
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
  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
" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1"
}
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  3
                        { "state" : { "$minKey" : 1 } } -->> { "state" :
 "MA" } on : shard1 Timestamp(1, 1)
                        { "state" : "MA" } -->> { "state" : "RI" } on :
shard1 Timestamp(1, 2)
                        { "state" : "RI" } -->> { "state" : { "$maxKey"
: 1 } } on : shard1 Timestamp(1, 3)
```

Hash Based Partitioning

In this partitioning strategy data is partitioned using a hash function.

Create a new collection for this strategy:

```
db.createCollection("cities2")
show collections
```

Enable sharding on this collection also using `state` as the shard key:

```
sh.shardCollection(
  "mydb.cities2", { "state": "hashed" }
)
```

Verify the state of the cluster:

```
sh.status()
```

> **Q5. How many chunks did you create? What differences do you see with respect to the same task in the range sharding strategy? Include a screen copy of the results of the command in your answer to support your answer.**
> *A. Two chunks were created, and the difference seems to be that the chunks are taking some kind of 0 hash as a placeholder intermedate value between the start and end of the collection, which right now is empty.*

Populate the collection using mydb.cities

```
db.cities.find().forEach(
   function(doc) {
      db.cities2.insert(doc);
   }
)
```

Verify new cluster state:

```
sh.status()
```

> **Q6. How many chunks are there now? Include a screen copy of the results of the command in your answer to support your answer. Compare the result with respect to the range sharding. Include a screen copy of the results of the command in your answer to support your answer.**
> *A. Now there are 4 chunks. In comparison to the range sharding, with hashing it isn't clear what value of the state attribute is being used to perform the sharding. By the nature of hashing we can make a guess that its an existing value, but not much more with the information that we have at the moment.*

Execution:

```
mongos> db.createCollection("cities2")
{ "ok" : 1 }
mongos> show collections
cities
cities1
cities2
system.indexes
mongos> sh.shardCollection(
...   "mydb.cities2", { "state": "hashed" }
... )
{ "collectionsharded" : "mydb.cities2", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
```

```
  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" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1" }
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  3
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shard1 Timestamp(1, 1)
                        { "state" : "MA" } -->> { "state" : "RI" } on : shard1 Timestamp(1, 2)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shard1 Timestamp(1, 3)
                mydb.cities2
```

```
                        shard key: { "state" : "hashed" }
                        chunks:
                                shard1  2
                        { "state" : { "$minKey" : 1 } } -->> { "state" : NumberLong(0) } on : shard1 Timestamp(1,
 1)
                        { "state" : NumberLong(0) } -->> { "state" : { "$maxKey" : 1 } } on : shard1 Timestamp(1,
 2)

mongos> db.cities.find().forEach(
...    function(doc) {
...       db.cities2.insert(doc);
...    }
... )
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
```

```
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
  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" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1" }
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  3
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shard1 Timestamp(1, 1)
                        { "state" : "MA" } -->> { "state" : "RI" } on : shard1 Timestamp(1, 2)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shard1 Timestamp(1, 3)
                mydb.cities2
                        shard key: { "state" : "hashed" }
                        chunks:
                                shard1  4
                        { "state" : { "$minKey" : 1 } } -->> { "state" : NumberLong(0) } on : shard1 Timestamp(1,
 1)
                        { "state" : NumberLong(0) } -->> { "state" : NumberLong("3630192931154748514") } on : sha
rd1 Timestamp(1, 3)
                        { "state" : NumberLong("3630192931154748514") } -->> { "state" : NumberLong("813462517913
9298768") } on : shard1 Timestamp(1, 4)
                        { "state" : NumberLong("8134625179139298768") } -->> { "state" : { "$maxKey" : 1 } } on :
 shard1 Timestamp(1, 5)
```

Balancing Data Across Shards

Cluster balancing is the process that MongoDB uses to distribute chunks across shards.

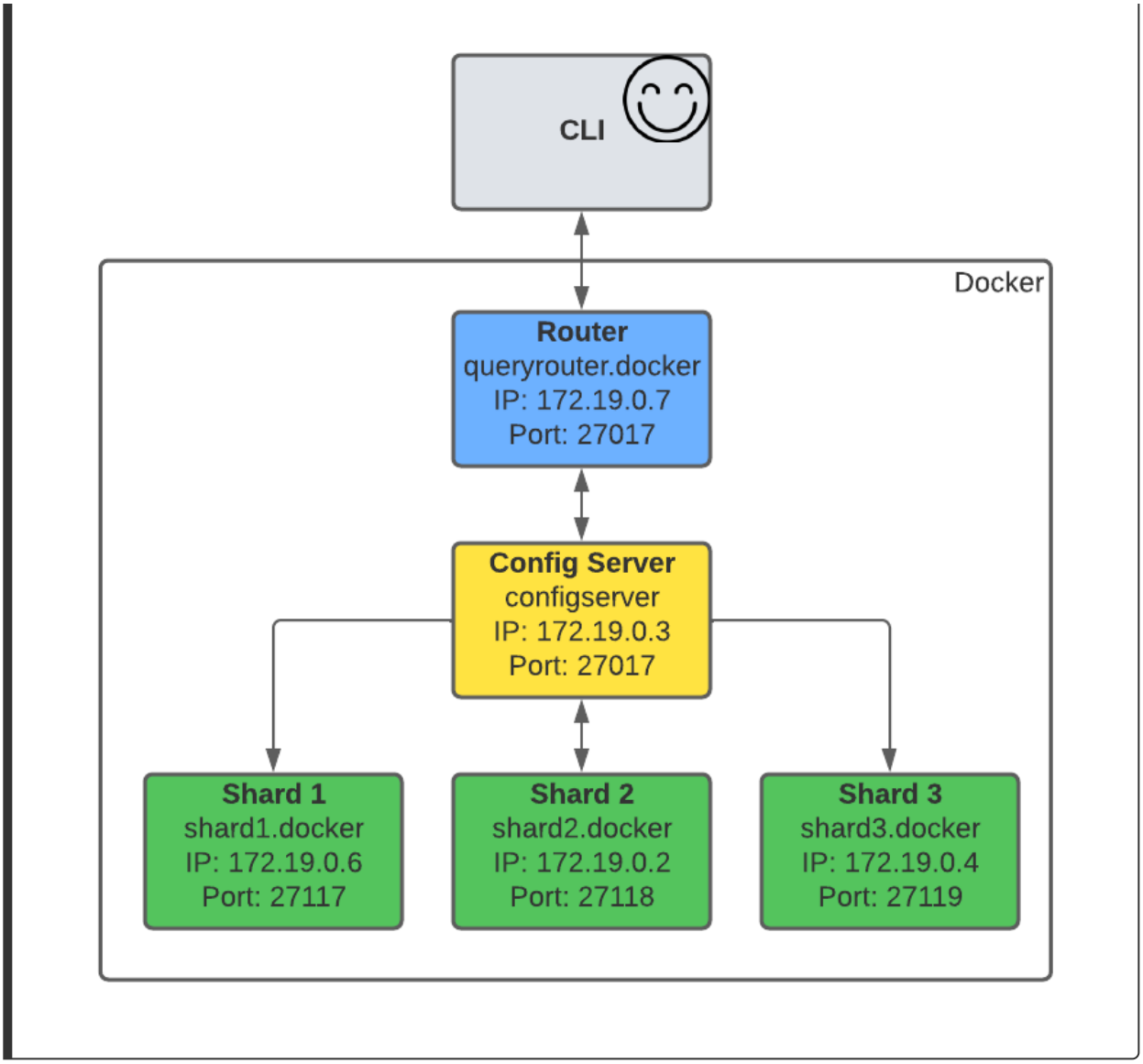
We can trigger it buy running the following commands:

```
use admin
db.runCommand( { addShard: "shard2.docker", name: "shard2" } )
db.runCommand( { addShard: "shard3.docker", name: "shard3" } )
```

After waiting a moment for the changes to take effect, we can check the status to see if it changed:

```
sh.status()
```

> **Q7. Draw the new configuration of the cluster and label each element (router, config server and shards) with its corresponding IP.**



Execution:

```
mongos> use admin
switched to db admin
mongos> db.runCommand( { addShard: "shard2.docker", name: "shard2" } )
{ "shardAdded" : "shard2", "ok" : 1 }
mongos> db.runCommand( { addShard: "shard3.docker", name: "shard3" } )
{ "shardAdded" : "shard3", "ok" : 1 }
```

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1.docker:27017" }
        {  "_id" : "shard2",  "host" : "shard2.docker:27017" }
        {  "_id" : "shard3",  "host" : "shard3.docker:27017" }
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                4 : Success
  databases:
```

```
      {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1" }
            mydb.cities1
                  shard key: { "state" : 1 }
                  chunks:
                        shard1  1
                        shard2  1
                        shard3  1
                  { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shard2 Timestamp(2, 0)
                  { "state" : "MA" } -->> { "state" : "RI" } on : shard3 Timestamp(3, 0)
                  { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shard1 Timestamp(3, 1)
            mydb.cities2
                  shard key: { "state" : "hashed" }
                  chunks:
                        shard1  2
                        shard2  1
                        shard3  1
                  { "state" : { "$minKey" : 1 } } -->> { "state" : NumberLong(0) } on : shard2 Timestamp(2,
0)
                  { "state" : NumberLong(0) } -->> { "state" : NumberLong("3630192931154748514") } on : sha
rd3 Timestamp(3, 0)
                  { "state" : NumberLong("3630192931154748514") } -->> { "state" : NumberLong("813462517913
9298768") } on : shard1 Timestamp(3, 1)
                  { "state" : NumberLong("8134625179139298768") } -->> { "state" : { "$maxKey" : 1 } } on :
 shard1 Timestamp(1, 5)
```

Guiding Partitioning Using Tags

Tagging can be performed on a range of shard key values, and they provide isolation of data and collocation of shards in geographical related regions.

In this section tags will be used to isolate cities into specific shard servers

First we associate tags to shards:

```
sh.addShardTag("shard1", "CA")
sh.addShardTag("shard2", "NY")
sh.addShardTag("shard3", "Others")
```

Then we create, populate and enable sharding on a new collection:

```
use mydb

db.createCollection("cities3")
sh.shardCollection("mydb.cities3", { "state": 1} )

db.cities.find().forEach(
   function(doc) {
      db.cities3.insert(doc);
   }
)
```

Now we define and associate key ranges `[from, to)` to shards:

```
sh.addTagRange("mydb.cities3", { state: MinKey }, { state: "CA" }, "Others")
sh.addTagRange("mydb.cities3", { state: "CA" }, { state: "CA_" }, "CA")
sh.addTagRange("mydb.cities3", { state: "CA_" }, { state: "NY" }, "Others")
sh.addTagRange("mydb.cities3", { state: "NY" }, { state: "NY_" }, "NY")
sh.addTagRange("mydb.cities3", { state: "NY_" }, { state: MaxKey }, "Others")
```

And as usual we analyze the new cluster state:

```
sh.status()
```

> **Q8. Analyze the results and explain the logic behind this tagging strategy. Connect to the shard that contains the data about California, and count the documents. Do the same operation with the other shards. Is the sharded data collection complete with respect to initial one? Are shards orthogonal?**
> *The idea for this tagging strategy is to have an indentification of which documents correspond to the specific state. After counting the documents, we can infer that shards are orthogonal and no information gets lost in the sharding process.*

Execution:

```
mongos> sh.addShardTag("shard1", "CA")
mongos> sh.addShardTag("shard2", "NY")
mongos> sh.addShardTag("shard3", "Others")
mongos> use mydb
switched to db mydb
mongos> db.createCollection("cities3")
{ "ok" : 1 }
mongos> sh.shardCollection("mydb.cities3", { "state": 1} )
{ "collectionsharded" : "mydb.cities3", "ok" : 1 }
mongos> db.cities.find().forEach(
...     function(doc) {
...        db.cities3.insert(doc);
...     }
... )
mongos> sh.addTagRange("mydb.cities3", { state: MinKey }, { state: "CA"
}, "Others")
mongos> sh.addTagRange("mydb.cities3", { state: "CA" }, { state: "CA_" }
, "CA")
mongos> sh.addTagRange("mydb.cities3", { state: "CA_" }, { state: "NY" }
```

```
, "Others")
mongos> sh.addTagRange("mydb.cities3", { state: "NY" }, { state: "NY_" }
, "NY")
mongos> sh.addTagRange("mydb.cities3", { state: "NY_" }, { state: MaxKey
 }, "Others")
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("620f275ee079bad15a655d60")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1.docker:27017",  "tags" : [ "CA" ] }
        {  "_id" : "shard2",  "host" : "shard2.docker:27017",  "tags" : [ "NY" ] }
        {  "_id" : "shard3",  "host" : "shard3.docker:27017",  "tags" : [ "Others" ] }
  balancer:
        Currently enabled:  yes
```

```
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                7 : Success
 databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1" }
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  1
                                shard2  1
                                shard3  1
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shard2 Timestamp(2, 0)
                        { "state" : "MA" } -->> { "state" : "RI" } on : shard3 Timestamp(3, 0)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shard1 Timestamp(3, 1)
                mydb.cities2
                        shard key: { "state" : "hashed" }
                        chunks:
```

```
                                shard1  2
                                shard2  1
                                shard3  1
                        { "state" : { "$minKey" : 1 } } -->> { "state" : NumberLong(0) } on : shard2 Timestamp(2,
 0)
                        { "state" : NumberLong(0) } -->> { "state" : NumberLong("3630192931154748514") } on : sha
rd3 Timestamp(3, 0)
                        { "state" : NumberLong("3630192931154748514") } -->> { "state" : NumberLong("813462517913
9298768") } on : shard1 Timestamp(3, 1)
                        { "state" : NumberLong("8134625179139298768") } -->> { "state" : { "$maxKey" : 1 } } on :
 shard1 Timestamp(1, 5)
                mydb.cities3
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  1
                                shard3  5
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "CA" } on : shard3 Timestamp(4, 1)
                        { "state" : "CA" } -->> { "state" : "CA_" } on : shard3 Timestamp(4, 3)
                        { "state" : "CA_" } -->> { "state" : "MA" } on : shard3 Timestamp(4, 4)
```

```
                        { "state" : "MA" } -->> { "state" : "NY" } on : shard3 Timestamp(4, 5)
                        { "state" : "NY" } -->> { "state" : "RI" } on : shard3 Timestamp(4, 6)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shard1 Timestamp(3, 1)
                         tag: Others  { "state" : { "$minKey" : 1 } } -->> { "state" : "CA" }
                         tag: CA  { "state" : "CA" } -->> { "state" : "CA_" }
                         tag: Others  { "state" : "CA_" } -->> { "state" : "NY" }
                         tag: NY  { "state" : "NY" } -->> { "state" : "NY_" }
                         tag: Others  { "state" : "NY_" } -->> { "state" : { "$maxKey" : 1 } }
```

Debate on data exploration Material

Consider that you must develop a “vademecum” for helping decision making processes related to data sharding using MongoDB. Use the following questions to interact with the members of your group to develop a set of hints that would contain your vademecum.

What is a sharding key? Is the choice of a sharding key directly dependent of the sharding strategy? Explain and give examples.

A shard key must an attribute that all documents share. By transforming the attribute to a shard key it becomes the attribute by which the chunks will be created and shards distributed. The sharding key seemed independent of the sharding strategy; even text could be used as the key.
Another possible example for the shard key could be the population; by using the range sharding strategy we could distribute the cities based on their population. This could be useful in applications where we want to directly query a set of just the cities with a big population.

Explain how does apparently MongoDB chooses the number of intervals used to shard a collection in the Interval oriented strategy? In which situations would such a strategy be well adapted for sharding a collection?

Its hard to tell if the behavior would be different for a collection of different size or properties, but based on what I saw I believe that in the range strategy the number of intervals is selected depending on the alphabetic order of the states, as "M" is more or less in the middle of the alphabet. Maybe if it were numeric values and there where a lot of them the strategy would be similar to a binary search; selecting the middle values and splitting in half the data. With more possible values, I would expect a higher number of intervals.
I think this strategy is ideal in situations where the data in the collection is more or less evenly distributed; if most of the cities are in states that start with the first letters of the alphabet, then the shards would be too uneven.

In which situations would the hash-based strategy be interesting for a collection to be sharded?

The hash-based strategy attempts to solve the problem stated in the last question, since it provides a more even distribution of the data.

Which of the strategies interval or sharding would lead to a more balanced distribution of data across shards, interval, or hash?

The safest answer would be hash, but it really depends on the data. Hash would be more even but we would be missing out on the semantics that the interval provides.

What are the advantages and disadvantages of allowing access to shards directly through their server and not only through the query router?

Advantages:

- Less clutter if you already know what you are looking for.
- One less step could mean more efficiency.

Disadvantages:

- Without prior knowledge about the distribution of the data, finding what you are looking for would be tedious; even imposible when having a lot of shards.
- Removal of one layer of abstraction makes usage harder.

Give an example of situation where tag based sharding would be an interesting option?

Maybe if we have a collection of people and we have information about their income, we could create tags to divide them into "classes" and perform queries on a specific class. (Note: the idea of dividing people into classes is awful, I don't agree with it but it was the best example I came up with)

What happens when a new shard is added to a cluster containing already other shards with data?

If the balancer is enabled, then it would automatically redistribute them in the way that it considers is best.

How would you test whether a sharded collection was an interesting solution in comparison to a centralized one?

I think the easiest example is in efficiency of the queries. Reducing the amount of documents to analyze may mean a quicker output. Another example may be in the correctness of the results; if we want to perform a query about an specific group of people but don't want to use filters every time, then sharding is an interesting option.