Install and Configure mongodb *Development Configuration Sharded Cluster* with Vagrant

Fernando G. Tinetti*, Agustín Terruzzi

Technical Report TR-BDD-03-2018
III-LIDI, Fac. de Informática, UNLP
*Also with CIC, Prov. de Buenos Aires
Argentina
contact e-mail: fernando@info.unlp.edu.ar
July 2018

Abstract. We document the installation and configuration of a so called mongodb sharded cluster. Actually, mongodb defines sharding as for data distribution in a cluster of mongodb servers. We are particularly interested in the so called "Development Configuration" in order to verify the basic installation, configuration, and operation of such "sharded cluster" with minimal infrastructure requirements. Most of the basic information is extracted from the official mongodb site/documentation, but we paid special attention to the development configuration and its implementation in a virtual environment handled by the Vagrant software. More specifically, the configuration environment is not fully covered in the mongodb documentation, so we expect to cover most (if not all) the specific details in this technical report so that we can use it as a "step-by-step" list to have a mongodb development configuration sharded cluster "up and running" with minimum effort/trial-and-error.

1.- Introduction

We consider this technical report as the third of a series and expected to be read after the previous ones, referred to the mongodb replica set [6] and using a nodejs client [7] respectively, both taking advantage of the Vagrant [1] tool for handling virtual environments. As we have made in previous technical reports, we focus on explaining and/or documenting several mongodb distributed (NoSQL) database concepts and issues either not completely documented or for which we had to devote too much time for having a useful experimental environment. In this case, we think that simplifying and defining a simple and yet useful experimentation environment would provide a better understanding of data distribution in a cluster handled by a mongodb database. We are not interested in handling a large amount of data but in understanding the rationale underlying the mongodb database distribution in a cluster.

The mongodb documentation defines "sharding" for data distribution across several computers, mostly for horizontal scaling [2]. Initially, the term *sharding* has been used for the concept referred to as *horizontal fragmentation* in the context of structured/SQL distributed databases [5]. Many NoSQL DBMS, including mongodb, use the term sharding in this way. Furthermore, the mongodb focus on horizontal scaling is also clear by the use of the expression "sharded cluster" [2] for the mongodb sharding implementation.

The suggested sharding configuration (or sharded cluster in terms of the mongodb documentation) for production environments is schematically shown in Fig. 1 [3], where

- The Router (a specific mongodb process: mongos) is the front-end/client interface, for processing every client request. There could be as many Routers as necessary for handling clients, and each Router is completely independent of/transparent to each other.
- The Config Server Replica Set (also referred to by its acronym: CSRS) is a config (configuration) server containing the sharding metadata in a mongodb replica set configuration.
- Two or more Shards, i.e. two or more mongodb data partition servers, each one containing a database partition. Each mongodb data partition server is expected to be replicated, i.e. a mongodb replica set is maintained for each database shard.

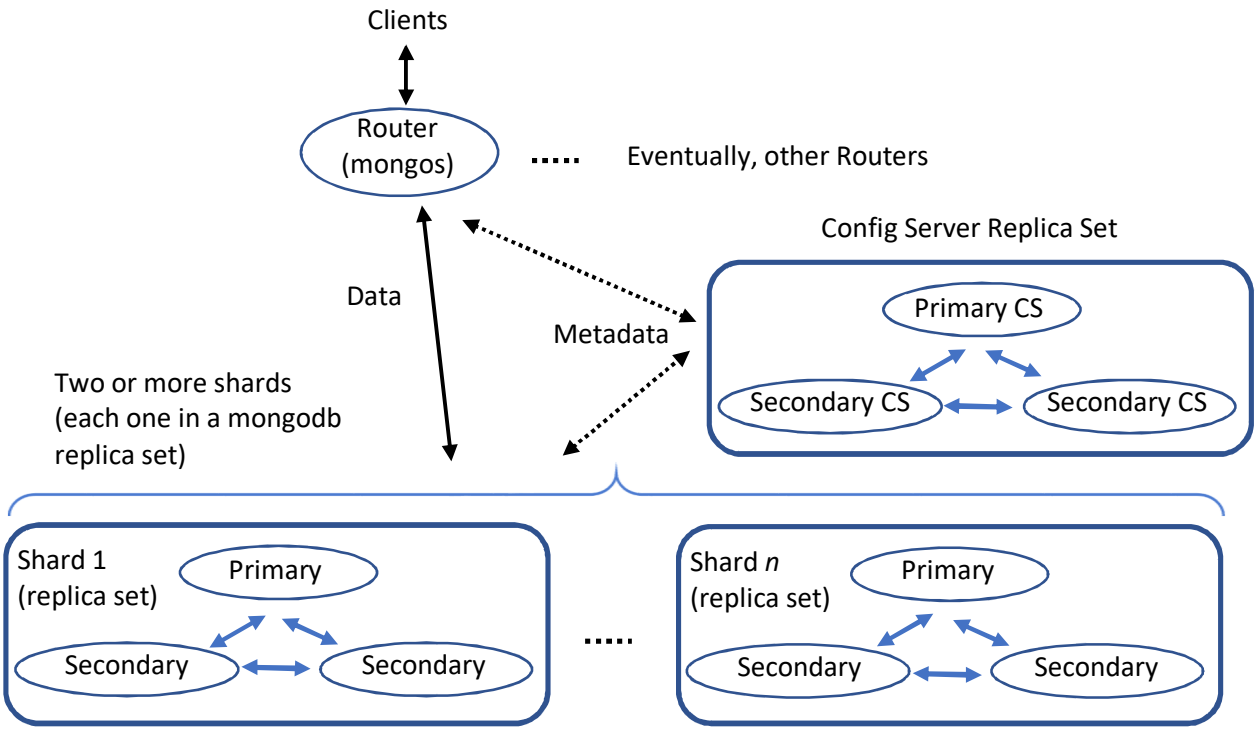

Figure 1: Sharded Cluster in a Production Environment.

Each process in Fig. 1, identified with an ellipse, is expected to be in a single computer, mostly for high availability/fault tolerance. Rounded corner rectangles in Fig. 1 are used to group the mongodb servers/processes by the conceptual service they provide. Having into account that each process is assigned to a different (virtual) computer, the minimum number of computer servers in a sharded cluster would be 10: 1 for the Router, 3 for the CSRS, 3 for the "Shard 1" replica set, and 3 for the "Shard 2" replica set, i.e. a sharded cluster with two shards. Even for a virtualized environment, having ten virtual computers would require a significant amount of resources, being the RAM the most constrained in standard computers (laptop and desktop ones).

Clearly, requiring ten computers for the initial steps of learning, experimentation of functionality/ies, etc. is (or, at least, seems to be) too much. Actually, setting the whole environment (even a virtualized one) would require an effort similar to or even greater than the one required for the task with the sharded cluster itself. The sharded cluster configuration we suggest for a learning, development, and minimal experimentation environments is based on that in [3], and is schematically shown in Fig. 2, where

- The Router (a specific mongodb process: mongos) is the front-end/client interface, and at least initially, we can set only one of them for serving clients requests.

- The Config (configuration) Server containing the sharding metadata, now running as a single process. Besides, at least initially, we allocate the process in the computer where the Router is assigned. We are not specifically interested in performance in the verification of functionality and debugging process, so we are able to reduce the number of (virtual) computers for the whole infrastructure.
- Two shards, each one in a different (virtual) computer, where some data can be stored and specific experiments for data distribution verification can be carried out.

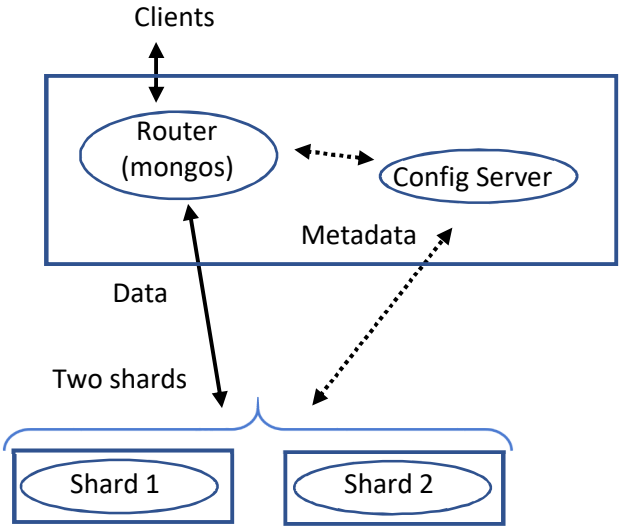

Figure 2: Sharded Cluster in a Development and Minimal Experimentation Environment.

Each process in Fig. 2 is identified with an ellipse and (virtual) computers to which they are assigned/they are running in is identified with a rectangle. Thus, the minimum number of computer servers in the sharded cluster would be 3: 1 for the Router and the Config Server and 1 for each of the shards, i.e. 1 computer "Shard 1", and 1 for the "Shard 2". None of the sharded cluster services is replicated, so there is no mongodb replica set definition for any service. The step-by-step guide described in the mongodb documentation [4] is focused on the production environment, so we focus on the minimal experimentation environment in this technical report.

2.- Command Line Deploy of a Minimal Sharded Cluster

Most (if not all) of the technical details explained in the mongodb documentation [2], e.g. shard keys and sharding strategy, chunks, sharded and non-sharded collections, etc. are better understood having an "up and running" sharded cluster. Furthermore, we think it is necessary to know not only to explain how to have such an up and running sharded cluster (by following the guide in [4]) but also the involved mongodb processes/services and their functionality in the sharded cluster. As a bonus, having less processes implies not only less computers/hardware requirements but also a reduced time to set up and maintain the whole environment.

Taking into account the Fig. 2 above, we need three computers for our *minimal* sharded cluster: one for running the Router and Config Server processes and one for each of the shard servers. We will take advantage of the vagrant [1] tool for setting up the three initial virtual computers, in a similar way as that explained in [6] [7]. Thus, we will start with three virtual computers with the minimum number of tools/software and the mongodb installation. The vagrant file we use for defining the initial infrastructure is

```
base_network = "192.168.33."
base_host = 20

Vagrant.configure("2") do |config|
  3.times do |num|
    config.vm.define ("monshrd%01d"  % (num + 1)) do |machine|
      machine.vm.box = "bento/ubuntu-16.04"
      machine.vm.network "private_network", ip: "#{base_network}#{base_host + num}"
      machine.vm.hostname = "monshrd#{num+1}"

      machine.vm.provider "virtualbox" do |vb|
        vb.memory = "512"
      end

      machine.vm.provision "shell", inline: <<-SHELL
        # Adding IPs and hostnames... Warning: this is hardcoded... and has a long
list of dependencies...
        echo ""                       | sudo tee -a /etc/hosts
        echo "192.168.33.20 monshrd1" | sudo tee -a /etc/hosts
        echo "192.168.33.21 monshrd2" | sudo tee -a /etc/hosts
        echo "192.168.33.22 monshrd3" | sudo tee -a /etc/hosts

        # Get updated urls/packages sources?
        sudo apt-get update

        # The joe editor
        sudo apt-get install -y joe

        # The alias for ls
        grep -q "alias ls='ls -la'" /home/vagrant/.bashrc || echo -e "\\nalias ls='ls
-la'" >> /home/vagrant/.bashrc

        # Install mongodb if needed
        if ! which mongod > /dev/null ; then
          sudo apt-get install -y mongodb
        fi

        # Save mongodb.conf
        sudo cp /etc/mongodb.conf /etc/mongodb.conf.orig

        # Stop the mongodb server, we'll do it initially in the command line...
        sudo service mongodb stop
      SHELL
    end
  end
end
```

After the setup time, i.e. after running

$ vagrant up

we will have three identical virtual machines, with names and IPs set as

monshrd1 192.168.33.20
monshrd2 192.168.33.21
monshrd3 192.168.33.22

respectively, each one with mongodb installed, but the mongodb server (mongod) is stopped, because we are going to start up the processes in each computer according to their functionality in the sharded cluster.

The underlying idea it to define and use monshrd1 as the one with the "general view" and service provider of the sharded cluster, i.e. in this computer will run the Router and the Config Server. The other two computers, monshrd2 and monshrd3, will be devoted to hold and serve a single shard each. Since the data is expected to be distributed in two shards, each document of the sharded data will be in one of the shards at either monshrd1 or monshrd2.

We will follow the guide in [4], but specifically adapted to avoid having replication sets for the Config Server, Shard 1, and Shard 2, as shown in Fig. 2 above. Unlike the previous reports in this series [6] [7], there will be involved two different mongodb servers: mongos and mongod. Furthermore, the mongodb mongod server will be configured as either a Config Server or a Shard Server. More specifically, we could add information to Fig. 3 above as shown in Fig. 4 below, about hostnames, IPs, and process names and settings.

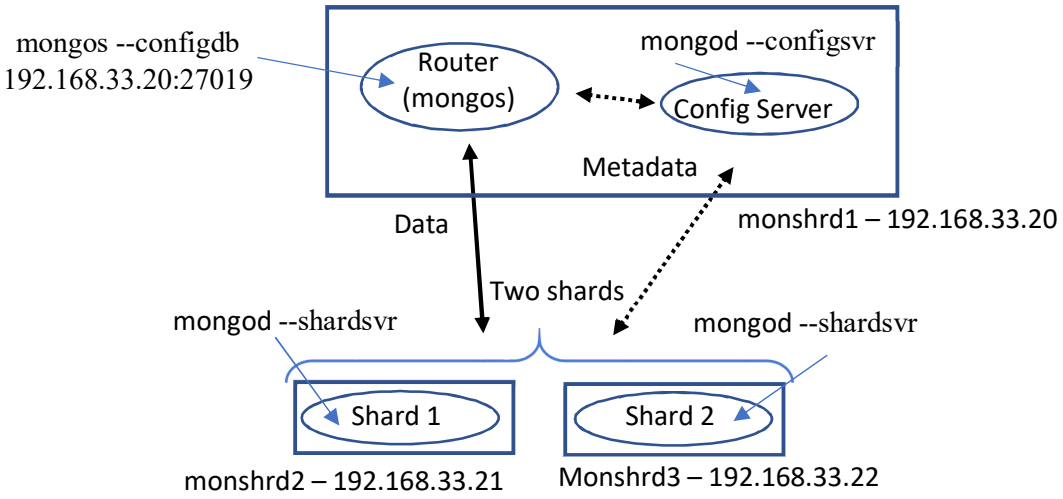

Figure 3: Sharded Cluster Processes and Configuration.

Describing the processes and process configuration in each computer of Fig. 3:

- monshrd1:
  mongos (mongo server, handling the client requests)
  +
  mongod (mongodb configuration server, handling metadata)
- monshrd2: mongod (mongodb shard 1, handling a fraction of the total data)
- monshrd3: mongod (mongodb shard 2, handling a fraction of the total data)

And the most relevant information per computer would be:

- Each one of the computers monshrd2 and monshrd3 holds a mongod process so that
  - It is run as a shard server, defined by the usage of the --shardsvr option.
  - At least initially, it does not have any information about another shard server/s, shard configuration server or shard router server. It works mostly as a standalone data server (except for the --shardsvr option) from the point of view of the programmer.
  - When replicated, the only difference is setting a mongodb replica set and start the whole replica set as a replicated shard server.
- The computer monshrd1, which could be considered as the sharded cluster "front-end", holds two processes
  - A mongod process, which is runing as a shard configuration server, defined by the usage of the --configsvr option. At least initially, it does not have any information about shard server/s or the

shard router server. When replicated, the only difference is setting a mongodb replica set and start the whole replica set as a replicated shard configuration server.

- A mongos process, which should be started with the information about where the shard configuration server is running with the option --configdb (in this specific case, the shard configuration server is in the same computer).

The mongod processes (as shard server and shard config server configurations) can be started in parallel, since they all start running independently of each other. In terms of the vagrant infrastructure defined above (the ">" identifies a windows/host OS command line cursor, the "$" identifies a Linux/guest OS command line cursor):

1) Start the mongo config server

> vagrant up (if not already made)
> vagrant ssh monshrd1

$ sudo mongod --configsvr –dbpath /var/lib/mongodb --bind_ip localhost,192.168.33.20

As explained before, the mongod process is defined as a configuration server, no other information given, i.e. no client handling and no sharding information so far, this mongod process listens to requests in port 27019:

```
vagrant@monshrd1:~$ sudo mongod --configsvr -dbpath /var/lib/mongodb --bind_ip
localhost,192.168.33.20
2018-07-17T15:50:38.648+0000 [initandlisten] MongoDB starting : pid=13125 port=27019
dbpath=/var/lib/mongodb master=1 64-bit host=monshrd1
2018-07-17T15:50:38.648+0000 [initandlisten] db version v2.6.10
2018-07-17T15:50:38.649+0000 [initandlisten] git version: nogitversion
2018-07-17T15:50:38.649+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-07-17T15:50:38.649+0000 [initandlisten] build info: Linux lgw01-12 3.19.0-25-generic
#26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 BOOST_LIB_VERSION=1_58
2018-07-17T15:50:38.649+0000 [initandlisten] allocator: tcmalloc
2018-07-17T15:50:38.650+0000 [initandlisten] options: { net: { bindIp:
"localhost,192.168.33.20" }, sharding: { clusterRole: "configsvr" }, storage: { dbPath:
"/var/lib/mongodb" } }
2018-07-17T15:50:38.730+0000 [initandlisten] journal dir=/var/lib/mongodb/journal
2018-07-17T15:50:38.731+0000 [initandlisten] recover : no journal files present, no recovery
needed
2018-07-17T15:50:39.051+0000 [initandlisten] preallocateIsFaster=true 4.3
2018-07-17T15:50:39.265+0000 [initandlisten] ******
2018-07-17T15:50:39.266+0000 [initandlisten] creating replication oplog of size: 5MB...
2018-07-17T15:50:39.267+0000 [initandlisten] ******
2018-07-17T15:50:39.267+0000 [initandlisten] waiting for connections on port 27019
```

2) Start the mongo shard 1 server

> vagrant ssh monshrd2

$ sudo mongod --shardsvr --dbpath /var/lib/mongodb --bind_ip localhost,192.168.33.21

As explained before, the mongod process is defined as a shard server, no other information given, i.e. no client handling and no sharding information so far, this mongod listen to requests in port 27018:

```
vagrant@monshrd2:~$ sudo mongod --shardsvr --dbpath /var/lib/mongodb --bind_ip
localhost,192.168.33.21
2018-07-17T16:26:21.691+0000 [initandlisten] MongoDB starting : pid=13128 port=27018
dbpath=/var/lib/mongodb 64-bit host=monshrd2
2018-07-17T16:26:21.692+0000 [initandlisten] db version v2.6.10
2018-07-17T16:26:21.692+0000 [initandlisten] git version: nogitversion
2018-07-17T16:26:21.692+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-07-17T16:26:21.692+0000 [initandlisten] build info: Linux lgw01-12 3.19.0-25-generic
#26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 BOOST_LIB_VERSION=1_58
2018-07-17T16:26:21.692+0000 [initandlisten] allocator: tcmalloc
2018-07-17T16:26:21.692+0000 [initandlisten] options: { net: { bindIp:
"localhost,192.168.33.21" }, sharding: { clusterRole: "shardsvr" }, storage: { dbPath:
"/var/lib/mongodb" } }
2018-07-17T16:26:21.737+0000 [initandlisten] journal dir=/var/lib/mongodb/journal
2018-07-17T16:26:21.737+0000 [initandlisten] recover : no journal files present, no recovery
needed
2018-07-17T16:26:21.953+0000 [initandlisten] waiting for connections on port 27018
2018-07-17T16:27:21.969+0000 [clientcursormon] mem (MB) res:43 virt:379
2018-07-17T16:27:21.969+0000 [clientcursormon]  mapped (incl journal view):160
2018-07-17T16:27:21.969+0000 [clientcursormon]  connections:0
```

3) Start the mongo shard 2 server

> vagrant ssh monshrd3

$ sudo mongod --shardsvr --dbpath /var/lib/mongodb --bind_ip localhost,192.168.33.22
(identical to shard 1, only the bind_ip changes accordingly)
The process output is similar (almost identical) to the mongo shard 1.

So far, we have three mongod processes, one running as a configuration server and two as shard servers, no one process has information on any of the other/s, all three processes are running independently of each other.

4) Start the router

> vagrant ssh monshrd1

$ sudo mongos --configdb 192.168.33.20:27019 --bind_ip localhost,192.168.33.20

The mongos process is started in computer monsrhd1 where the configuration server is already running. As explained before, the information on where the config server is running is given, in this case via the command line parameter --configdb as shown above. This would be the only process started so far with some information of the sharded cluster. However, the router process (mongos) does not have any knowledge of the shard servers, so the shard is not “up and running” yet. The output of this command is shown below (large number of similar output text lines have been replaced by a single line with “…”)

```
vagrant@monshrd1:~$ sudo mongos --configdb 192.168.33.20:27019 --bind_ip
localhost,192.168.33.20
2018-07-17T16:39:31.963+0000 warning: running with 1 config server should be done only for
testing purposes and is not recommended for production
2018-07-17T16:39:31.966+0000 [mongosMain] MongoS version 2.6.10 starting: pid=13221 port=27017
64-bit host=monshrd1 (--help for usage)
2018-07-17T16:39:31.966+0000 [mongosMain] db version v2.6.10
2018-07-17T16:39:31.966+0000 [mongosMain] git version: nogitversion
2018-07-17T16:39:31.966+0000 [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-07-17T16:39:31.966+0000 [mongosMain] build info: Linux lgw01-12 3.19.0-25-generic
#26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 BOOST_LIB_VERSION=1_58
2018-07-17T16:39:31.967+0000 [mongosMain] allocator: tcmalloc
2018-07-17T16:39:31.967+0000 [mongosMain] options: { net: { bindIp: "localhost,192.168.33.20"
}, sharding: { configDB: "192.168.33.20:27019" } }
2018-07-17T16:39:31.976+0000 [LockPinger] creating distributed lock ping thread for
192.168.33.20:27019 and process monshrd1:27017:1531845571:1804289383 (sleeping for 30000ms)
2018-07-17T16:39:32.148+0000 [LockPinger] cluster 192.168.33.20:27019 pinged successfully at
Tue Jul 17 16:39:31 2018 by distributed lock pinger
'192.168.33.20:27019/monshrd1:27017:1531845571:1804289383', sleeping for 30000ms
2018-07-17T16:39:32.153+0000 [mongosMain] distributed lock
'configUpgrade/monshrd1:27017:1531845571:1804289383' acquired, ts : 5b4e1bc39600f1089c6064de
2018-07-17T16:39:32.154+0000 [mongosMain] starting upgrade of config server from v0 to v5
2018-07-17T16:39:32.154+0000 [mongosMain] starting next upgrade step from v0 to v5
2018-07-17T16:39:32.154+0000 [mongosMain] about to log new metadata event: { _id: "monshrd1-
2018-07-17T16:39:32-5b4e1bc49600f1089c6064df", server: "monshrd1", clientAddr: "N/A", time:
new Date(1531845572154), what: "starting upgrade of config database", ns: "config.version",
details: { from: 0, to: 5 } }
2018-07-17T16:39:32.156+0000 [mongosMain] creating WriteBackListener for: 192.168.33.20:27019
serverID: 000000000000000000000000
…
2018-07-17T17:07:27.361+0000 [Balancer] distributed lock
'balancer/monshrd1:27017:1531845571:1804289383' unlocked.
2018-07-17T17:07:32.227+0000 [LockPinger] cluster 192.168.33.20:27019 pinged successfully at
Tue Jul 17 17:07:32 2018 by2018-07-17T17:08:02.228+0000 [LockPinger] cluster
192.168.33.20:27019 pinged successfully at Tue Jul 17 17:08:02 2018 by distributed lock pinger
'192.168.33.20:27019/monshrd1:27017:1531845571:1804289383', sleeping for 30000msacquired, ts :
5b4e22559600f1089c6065fc
2018-07-17T17:07:33.363+0000 [Balancer] distributed lock
'balancer/monshrd1:27017:1531845571:1804289383' unlocked.
…
'balancer/monshrd1:27017:1531845571:1804289383' acquired, ts : 5b4e22739600f1089c606601
2018-07-17T17:08:03.374+0000 [Balancer] distributed lock
'balancer/monshrd1:27017:1531845571:1804289383' unlocked.
```

So far, we have four terminals, each one running the config server, the shard 1 server, the shard server 2, and the router, respectively.

5) Complete the sharded cluster configuration

As expected, once we have every process successfully started, it is necessary to the shard servers so that the router and the config servers use it. These are the final steps of the sharded cluster configuration and they are made in a mongo console connected to the router process, i.e. to the mongos process.

> vagrant ssh monshrd1

mongo

```
vagrant@monshrd1:~$ mongo
MongoDB shell version: 2.6.10
connecting to: 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>
```

Note that the shell cursor is “mongos”, not “mongo”. At this point, add the shard 1 and shard 2 servers:

```
mongos> sh.addShard( "192.168.33.21:27018")
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos> sh.addShard( "192.168.33.22:27018")
{ "shardAdded" : "shard0001", "ok" : 1 }
```

We are able to request the status of sharded cluster:

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("5b4e1bc49600f1089c6064e0")
}
  shards:
        {  "_id" : "shard0000",  "host" : "192.168.33.21:27018" }
        {  "_id" : "shard0001",  "host" : "192.168.33.22:27018" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
```

i.e. there is a sharded cluster and no database is currently sharded.

6) Enable Sharding on a database

Sharding is made on collections in a database, so the database must be “enabled” for having sharded collections. Set a database other than the default one and enable that database to contains sharded collections of documents.

```
mongos> use sharded
switched to db sharded
mongos> show dbs
admin   (empty)
config  0.016GB
mongos> sh.enableSharding('sharded')
{ "ok" : 1 }
```

Now looking at the sharded cluster status:

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("5b4e1bc49600f1089c6064e0")
}
  shards:
        {  "_id" : "shard0000",  "host" : "192.168.33.21:27018" }
        {  "_id" : "shard0001",  "host" : "192.168.33.22:27018" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "sharded",  "partitioned" : true,  "primary" : "shard0000" }
```

i.e. the sharded databse is identified as enabled for “partitioning” (containing sharded collections).

7) Define a collection for sharding/to be sharded

We will define sharding a collection of a sharded enabled database in a two steps process: 1) defining the index, and 2) start sharding the collection.

7.1) Define the collection sharding index

```
mongos> db.coll2shard.ensureIndex( { _id : "hashed" } )
{
        "raw" : {
                "192.168.33.21:27018" : {
                        "createdCollectionAutomatically" : true,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "ok" : 1
                }
        },
        "ok" : 1
}
```

In this example, we have defined the _id as the index of the coll2shard collection in the sharded database we are working on, i.e. on which we have enabled sharding.

7.2) Start sharding the sharding process in the collection

```
mongos> sh.shardCollection('sharded.coll2shard', {_id:"hashed"})
{ "collectionsharded" : "sharded.coll2shard", "ok" : 1 }
```

3.- The Scope/Abstraction Levels in a Sharded Cluster

Note that the previous section on deploying a sharded cluster contains different scopes of commands or configurations, which can be associated to different abstraction levels in a sharded cluster. Some of them refer to the whole cluster or set of computers used for the sharded cluster, some to specific databases, and the last one/s to specific database collections.

Steps 1) to 5) of the step-by-step guide in the previous section can be referred to as "cluster-wide" steps. There is no reference to any database, their scope is the whole infrastructure/environment in which the mongodb databases are going to be "handled". Actually, there is no "up and running" sharded cluster prior to step 5), in which the shard servers are included in the router (mongos), which now has the whole sharded cluster "vision": it was started with the information of where to find the config server and in the mongo shell we were able to include every shard included in the sharded cluster. Notet that, for example, you are able to start much more shard servers than those given in the mongos shell, but shard servers not added with the addShard() command will be not taken into account in any sharding operation, they will not contain any sharded data. Clearly, the steps 1) to 5) enable any number of databases (and, thus, their corresponding data) to be sharded/distributed in the shard servers.

Step 6) of the step-by-step guide in the previous section can be referred to as "database-wide" step. There is no way of sharding a database that is not enabled for sharding. Moreover, having enabled a database for sharding, all its collections can be sharded, or a subset of collections can be specifically defined to be sharded. By default, no collection is sharded, though, it has to be explicitly defined. It is expected that the collections with most of the data in the database are to be sharded.

Commands described in step 7) of the step-by-step guide in the previous section can be referred to as "collection-wide" commands. Actually, only the documents in a collection enabled for sharding (which, in turn, has to be enabled for sharding) can be distributed in shards, and there is no other data in a mongodb database that would be distributed in any (other) way. Furthermore, the sharding itself involves several criteria explained in [2], and we have used one of them without much explanation. In turn, different sharding strategies fundamentally affect the way in which documents are going to be distributed and, in general, the way in which queries are going to be processed. We expect that at least initially, the setting of a sharded cluster we propose in this technical report will allow to know and experiment in small scale on these issues.

4.- Initial Testing of the Sharded Cluster

We will use a node client as in [7] to test the sharded cluster described above. We will have a windows node client operating with the sharded cluster, i.e. a basic heterogenous distributed system. We will take advantage of the windows host running the virtual machines for running the node client as well. The standard and well known steps for having a node mongodb client are carried out in this case too:

```
> nodevars.bat
Your environment has been set up for using Node.js 8.9.4 (x64) and npm.

> npm init
…

> npm install mongodb@2.2 --save
```

…
With the simple javascript code in connect.js

```
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

var url = 'mongodb://192.168.33.20:27017/sharded';
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected to server.");
  db.close();
});
```

```
> node connect.js
Connected to server.
```

Thus, the client is successfully connected to the router sharded cluster router process. And we will try to insert four documents in the sharded collection with the code in insert.js

```
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');

var url = 'mongodb://192.168.33.20:27017/sharded';

var insertDocuments = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('coll2shard');
  // Insert some documents
  collection.insertMany([
    {a : 1}, {a : 2}, {a : 3}, {a : 4}
  ], function(err, result) {
    assert.equal(err, null);
    assert.equal(4, result.result.n);
    assert.equal(4, result.ops.length);
    console.log("Inserted 4 documents into the collection");
    callback(result);
  });
}

// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  insertDocuments(db, function() {
    db.close();
  });
});
```

```
> node insert.js
Connected successfully to server
Inserted 4 documents into the collection
```

The console report indicates there was no problem/error at inserting the four documents. We will verify at the mongo console in the different mongodb servers/processes. Initially, we will check the shard status as reported by mongos, the sharded cluster router process:

```
mongos> sh.shardCollection('sharded.coll2shard', {_id:"hashed"})
{ "collectionsharded" : "sharded.coll2shard", "ok" : 1 }
mongos>
mongos> sh.status('verbose')
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("5b4e1bc49600f1089c6064e0")
}
  shards:
        {  "_id" : "shard0000",  "host" : "192.168.33.21:27018" }
        {  "_id" : "shard0001",  "host" : "192.168.33.22:27018" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "sharded",  "partitioned" : true,  "primary" : "shard0000" }
              sharded.coll2shard
                    shard key: { "_id" : "hashed" }
                    chunks:
                          shard0000       2
                          shard0001       2
                    { "_id" : { "$minKey" : 1 } } -->> { "_id" : NumberLong("-4611686018427387902") } on
: shard0000 Timestamp(2, 2)
                    { "_id" : NumberLong("-4611686018427387902") } -->> { "_id" : NumberLong(0) } on :
shard0000 Timestamp(2, 3)
                    { "_id" : NumberLong(0) } -->> { "_id" : NumberLong("4611686018427387902") } on :
shard0001 Timestamp(2, 4)
                    { "_id" : NumberLong("4611686018427387902") } -->> { "_id" : { "$maxKey" : 1 } } on
: shard0001 Timestamp(2, 5)
```

It provides information about the sharded databases as well as the sharded collection in the “sharded” database. Now we will check/verify the status/contents in the shard servers, initially in the computer monshrd2:

```
> show dbs
admin    (empty)
local    0.078GB
sharded  0.078GB
> se2018-07-17T20:08:15.163+0000 [clientcursormon] mem (MB) res:59 virt:540
2018-07-17T20:08:15.163+0000 [clientcursormon]  mapped (incl journal view):320
2018-07-17T20:08:15.163+0000 [clientcursormon]  connections:1

> use sharded
switched to db sharded

> show collections
coll2shard
system.indexes

> db.coll2shard.count()
1
> db.coll2shard.find()
{ "_id" : ObjectId("5b4e47f37c6bc91a605cd931"), "a" : 2 }
>
```

i.e. the shard server at monshrd2 contains only one of the four inserted documents, that with JSON contents {a : 2}. Checking the contents/status of the shard 2 server at monsrhd3:

```
> use sharded
switched to db sharded
> db.coll2shard.count()
3
> db.coll2shard.find()
{ "_id" : ObjectId("5b4e47f37c6bc91a605cd930"), "a" : 1 }
{ "_id" : ObjectId("5b4e47f37c6bc91a605cd932"), "a" : 3 }
{ "_id" : ObjectId("5b4e47f37c6bc91a605cd933"), "a" : 4 }
```

i.e. it contains the other three of the four documents inserted, those with JSON contents {a : 1}.{a : 3}, and {a : 4}.

5.- About Dropping a Sharded Database

Since we are in the context of learning and experimenting, it is a rather common task to delete/drop a sharded database and start over. Following the information at

https://jira.mongodb.org/browse/SERVER-17397

it would be necessary to use each shard mongo console:

```
$mongo --port 27018
mongo> use sharded
mongo> db.dropDatabase()
{ "dropped" : "sharded", "ok" : 1 }
```

Besides, in the computer with the router process (mongos):

```
$ mongo
mongos> use config
switched to db config
mongos> db.collections.remove( { _id:/^sharded\./ } )
WriteResult({ "nRemoved" : 1 })
mongos> db.databases.remove( { _id:"sharded" } )
WriteResult({ "nRemoved" : 1 })
mongos> db.chunks.remove( { ns: /^sharded\./ } )
WriteResult({ "nRemoved" : 4 })
mongos> db.locks.remove( { _id: /^sharded\./ } )
WriteResult({ "nRemoved" : 1 })
mongos> db.adminCommand("flushRouterConfig")
{ "flushed" : true, "ok" : 1 }
```

Still, the sharded db is shown when using the “show dbs” command… but the sh.status() does not show it

```
mongos> show dbs
admin    (empty)
config   0.016GB
sharded  0.156GB

mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("5b4e1bc49600f1089c6064e0")
}
  shards:
        {  "_id" : "shard0000",  "host" : "192.168.33.21:27018" }
        {  "_id" : "shard0001",  "host" : "192.168.33.22:27018" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
```

5.- Conclusions and Further Work

We have set up a sharded cluster and made a minimum experimentation on it. The step-by-step procedure is documented, taking into account that this information is not currently found elsewhere. We have taken advantage of the vagrant tool and predefined vagrant virtual machine “boxes” so that the computing and storage resources required are found in almost any standard desktop/laptop computer (e.g. a total of 1.5GB RAM). The experiments also show how the virtual machines host and guest computers are used, in a (virtual) heterogenous distributed system.

Most of the interesting work is yet to be done with the sharded cluster, we only document how to set up and use it. The immediate further step would be to set the configuration files of the mongo processes in

each computer/mongo server. We still think that for learning and experimentation purposes, having every server in a replication set would impose a large amount of resources and effort only for having the system "up-and-running". However, it does not seem to be reasonable to use the command line start of every process (even when it is interesting for knowing a little bit more about the functionality of each of them).

Once the system is handled with confidence, there are a lot of details to learn/experiment with. Many of them are related to the mongodb so called sharding strategy/ies. And maybe that with large amount of data, the real advantage of replica set would be apparent beyond the well known high availability functionality.

References

[1] HashiCorp Vagrant, Development Environments Made Easy
https://www.vagrantup.com/

[2] MongoDB, Inc., "Sharding – MongoDB Manual", (July 2018)
https://docs.mongodb.com/manual/sharding/

[3] MongoDB, Inc., "Sharded Cluster Components – MongoDB Manual", (July 2018)
https://docs.mongodb.com/manual/core/sharded-cluster-components/

[4] MongoDB, Inc., "Deploy a Sharded Cluster – MongoDB Manual", (July 2018)
https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/

[5] Özsu M. T., Valduriez P., Principles of Distributed Database Systems, 3rd. Ed. Springer Science & Business Media, 2011, ISBN 1441988343.

[6] Fernando G. Tinetti, Agustín Terruzzi, "Install a mongodb Replica Set in a Virtual Environment", Technical Report BDD-01-2018, February 2018.
http://fernando.bl.ee/reptec/2018-repl-mongo.pdf

[7] Fernando G. Tinetti, Agustín Terruzzi, "Install and Use mongodb nodejs Driver with a mongdb Replica Set Defined in Vagrant", Technical Report TR-BDD-02-2018, February 2018
http://fernando.bl.ee/reptec/2018-nodemongo.pdf