MONGODB SHARDING: A PRACTICAL, HANDS-ON GUIDE

This comprehensive article explores sharding in MongoDB. We break the article into two sections:

1. Concepts, components, pros & cons
2. A step-by-step tutorial on setting up sharding

(This article is part of our MongoDB Guide. Navigate using the right-hand menu.)

What is sharding?

Sharding is the process of distributing data across multiple hosts. In MongoDB, sharding is achieved by splitting large data sets into small data sets across multiple MongoDB instances.

How sharding works

When dealing with high throughput applications or very large databases, the underlying hardware becomes the main limitation. High query rates can stress the CPU, RAM, and I/O capacity of disk drives resulting in a poor end-user experience.

To mitigate this problem, there are two types of scaling methods.

Vertical scaling

Vertical scaling is the traditional way of increasing the hardware capabilities of a single server. The process involves upgrading the CPU, RAM, and storage capacity. However, upgrading a single server is often challenged by technological limitations and cost constraints.

Horizontal scaling

This method divides the dataset into multiple servers and distributes the database load among each server instance. Distributing the load reduces the strain on the required hardware resources and provides redundancy in case of a failure.

However, horizontal scaling increases the complexity of underlying architecture. MongoDB supports horizontal scaling through sharding—one of its major benefits, as we'll see below.

MongoDB sharding basics

MongoDB sharding works by creating a cluster of MongoDB instances consisting of at least three servers. That means sharded clusters consist of three main components:

- The shard
- Mongos
- Config servers

Shard

A shard is a single MongoDB instance that holds a subset of the sharded data. Shards can be deployed as replica sets to increase availability and provide redundancy. The combination of multiple shards creates a complete data set. For example, a 2 TB data set can be broken down into four shards, each containing 500 GB of data from the original data set.

Mongos

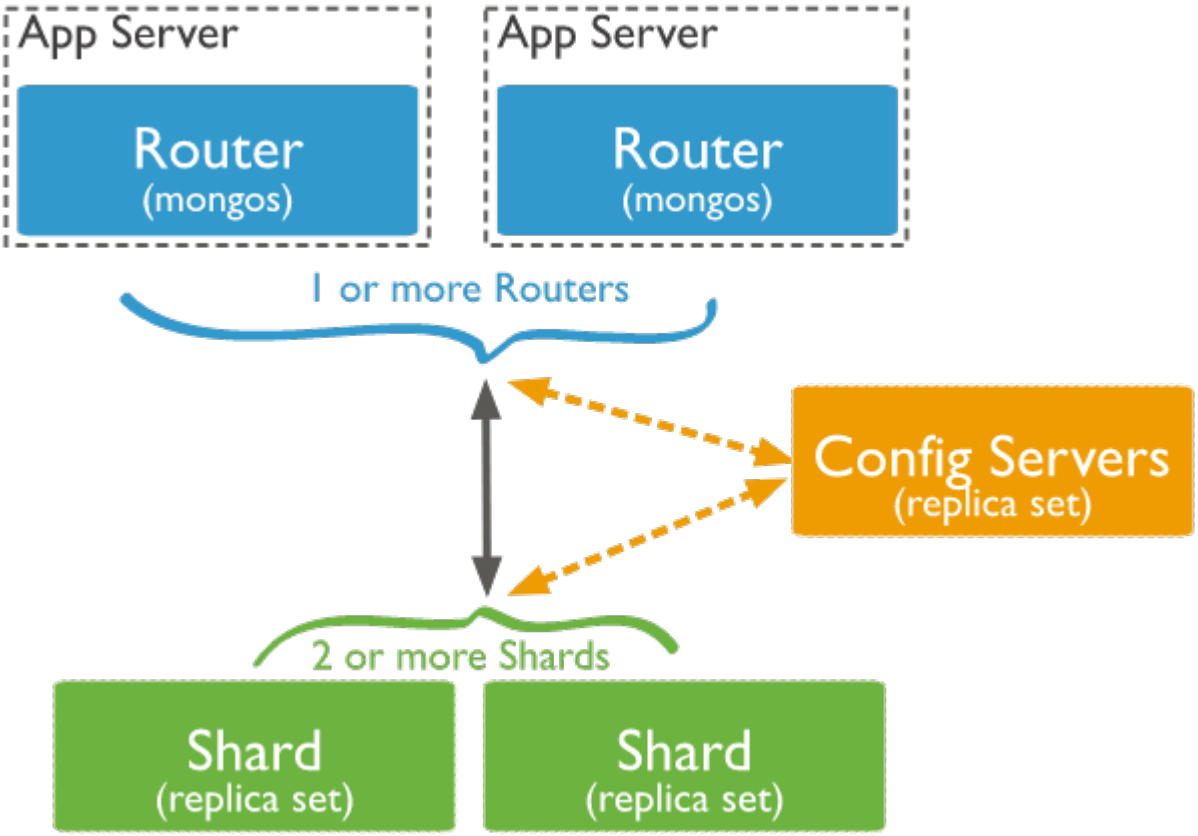
Mongos act as the query router providing a stable interface between the application and the sharded cluster. This MongoDB instance is responsible for routing the client requests to the correct shard.

Config Servers

Configuration servers store the metadata and the configuration settings for the whole cluster.

Components illustrated

The following diagram from the official MongoDB docs explains the relationship between each component:



1. The application communicates with the routers (mongos) about the query to be executed.
2. The mongos instance consults the config servers to check which shard contains the required data set to send the query to that shard.
3. Finally, the result of the query will be returned to the application.

It's important to remember that the config servers also work as replica sets.

Shard Keys

When sharding a MongoDB collection, a shard key gets created as one of the initial steps. The "shard key" is used to distribute the MongoDB collection's documents across all the shards. The key consists of a single field or multiple fields in every document. The sharded key is immutable and cannot be changed after sharding. A sharded collection only contains a single shard key.

When sharding a populated collection, the collection must have an index that starts with the shard key. For empty collections that don't have an appropriate index, MongoDB will create an index for the specified shard key.

The shard key can directly have an impact on the performance of the cluster. Hence can lead to bottlenecks in applications associated with the cluster. To mitigate this, before sharding the collection, the shard key must be created based on:

- The schema of the data set
- How the data set is queried

Chunks

Chunks are subsets of shared data. MongoDB separates sharded data into chunks that are distributed across the shards in the shared cluster. Each chunk has an inclusive lower and exclusive upper range based on the shard key. A balancer specific for each cluster handles the chunk distribution.

The balancer runs as a background job and distributes the chunks as needed to achieve an even balance of chunks across all shards. This process is called even chuck distribution.

Sharding benefits & limitations

Now that we've got the concept down, let's look at benefits and limitations of sharding in MongoDB:

Benefits

- In traditional replication scenarios, the primary node handles the bulk of write operations, while the secondary servers are limited to read-only operations or maintaining the backup of the data set. However, as sharding utilizes shards with replica sets, all queries are distributed among all the nodes in the cluster.
- As each shard consists of a subset of the complete data set, simply adding additional shards will increase the cluster's storage capacity without having to do complex hardware restructuring.
- Replication requires vertical scaling when handling large data sets. This requirement can lead to hardware limitations and prohibitive costs compared to the horizontal scaling approach. But, because MongoDB utilizes horizontal scaling, the workload is distributed. When the need arises, additional servers can be added to a cluster.
- In sharding, both read and write performance directly correlates to the number of server nodes in the cluster. This process provides a quick method to increase the cluster's performance by simply adding additional nodes.
- A sharded cluster can continue to operate even if a single or multiple shards are unavailable. While the data on those shards are unavailable, the client application can still access all the other available shards within the cluster without any downtime. In production environments, all individual shards deploy as replica sets, further increasing the availability of the cluster.

Limitations

- Sharding requires careful planning and maintenance to maintain a sharded cluster—because of the complexity involved.
- When you shard a MongoDB collection, there is no way to unshard the sharded collection.
- The shard key directly impacts the overall performance of the underlying cluster, as it is used to identify all the documents within the collections.
- There are some operational restrictions within a MongoDB sharded environment. For example, the **geoSearch** command is not supported within a sharded environment.
- In an instance where a shard key or a prefix of compound shard key is not present, Mongo will perform a broadcast operation that queries all the shards in the cluster, which can result in long-running query tasks.

How to set up a MongoDB sharding

Now that we understand the concepts of sharding, let's get started with some tutorials.

The following steps demonstrate how to set up a sharded cluster can on three servers. Ubuntu 20.04 LTS and MongoDB 4.4.1 are installed on all three servers. We can define servers as shown below:

- mongodb01 - 10.10.10.56 - Config Server
- mongodb02 - 10.10.10.57 - Query Router (mongos)
- mongodb03 - 10.10.10.58 – Shard

Now, we'll walk you through several actions, including how to:

- Configure the config server
- Configure the query router
- Configure a shard
- Add shard to cluster
- Enable sharding
- Create the sharding dataset
- Enable sharding on a collection

Configuring the config server

Create the directory structure for the database and log file.

```
mkdir -pv  mongodb/data/configdb/
mkdir -pv  mongodb/data/logs
touch mongodb/data/logs/configsvr.log
ls -alRv mongodb/
```

```
barry@mongodb01:~$ mkdir -pv  mongodb/data/configdb/
mkdir: created directory 'mongodb'
mkdir: created directory 'mongodb/data'
mkdir: created directory 'mongodb/data/configdb/'
barry@mongodb01:~$ mkdir -pv  mongodb/data/logs
mkdir: created directory 'mongodb/data/logs'
barry@mongodb01:~$ touch mongodb/data/logs/configsvr.log
barry@mongodb01:~$ ls -alRv mongodb/
mongodb/:
total 12
drwxrwxr-x 3 barry barry 4096 Nov  7 11:49 .
drwxr-xr-x 4 barry barry 4096 Nov  7 11:49 ..
drwxrwxr-x 4 barry barry 4096 Nov  7 11:49 data

mongodb/data:
total 16
drwxrwxr-x 4 barry barry 4096 Nov  7 11:49 .
drwxrwxr-x 3 barry barry 4096 Nov  7 11:49 ..
drwxrwxr-x 2 barry barry 4096 Nov  7 11:49 configdb
drwxrwxr-x 2 barry barry 4096 Nov  7 11:49 logs

mongodb/data/configdb:
total 8
drwxrwxr-x 2 barry barry 4096 Nov  7 11:49 .
drwxrwxr-x 4 barry barry 4096 Nov  7 11:49 ..

mongodb/data/logs:
total 8
drwxrwxr-x 2 barry barry 4096 Nov  7 11:49 .
drwxrwxr-x 4 barry barry 4096 Nov  7 11:49 ..
-rw-rw-r-- 1 barry barry    0 Nov  7 11:49 configsvr.log
barry@mongodb01:~$
```

Create the config file.

When you execute the above commands, a config file will be created along with those directories. Next, open the config file and assign the port and the IP address of the server. Also, don't forget to set the cluster role as **configsvr**.

```
sudo nano /etc/mongodConfig.conf
```

File - mongodConfig.conf

```
storage:
  dbPath: /home/barry/mongodb/data/configdb
  journal:
  enabled: true

systemLog:
  destination: file
  logAppend: true
  path: /home/barry/mongodb/data/logs/configsvr.log

net:
  port: 27019
  bindIp: 10.10.10.56

sharding:
  clusterRole: configsvr
```

```
replication:
  replSetName: ConfigReplSet
```

Start the config server. Now, you can start the config server with the following command.

```
mongod --config /etc/mongodConfig.conf&
```

```
barry@mongodb01:~$ mongod --config /etc/mongodConfig.conf&
[1] 9042
barry@mongodb01:~$
```

Check the logs to verify if the server is running.

```
tail -100 mongodb/data/logs/configsvr.log
```

```
barry@mongodb01:~$ tail -100 mongodb/data/logs/configsvr.log
{"t":{"$date":"2020-11-07T11:54:32.271+00:00"},"s":"I",  "c":"REPL",     "id":21311,   "ctx":"initandlisten","msg
":"Did not find local initialized voted for document at startup"}
{"t":{"$date":"2020-11-07T11:54:32.272+00:00"},"s":"I",  "c":"REPL",     "id":21529,   "ctx":"initandlisten","msg
":"Initializing rollback ID","attr":{"rbid":1}}
{"t":{"$date":"2020-11-07T11:54:32.272+00:00"},"s":"I",  "c":"REPL",     "id":21313,   "ctx":"initandlisten","msg
":"Did not find local replica set configuration document at startup","attr":{"error":{"code":47,"codeName":"NoMat
chingDocument","errmsg":"Did not find replica set configuration document in local.system.replset"}}}
{"t":{"$date":"2020-11-07T11:54:32.272+00:00"},"s":"I",  "c":"SHARDING", "id":22649,   "ctx":"thread1","msg":"Cre
ating distributed lock ping thread","attr":{"processId":"ConfigServer","pingIntervalMillis":30000}}
{"t":{"$date":"2020-11-07T11:54:32.274+00:00"},"s":"I",  "c":"REPL",     "id":40440,   "ctx":"initandlisten","msg
":"Starting the TopologyVersionObserver"}
```

Connect to the config server.

```
mongo 10.10.10.56:27019
```

```
barry@mongodb01:~$ mongo 10.10.10.56:27019
MongoDB shell version v4.4.1
connecting to: mongodb://10.10.10.56:27019/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("fbddc554-af8c-4984-8c80-5e5ee654fbf2") }
MongoDB server version: 4.4.1
---
The server generated these startup warnings when booting:
        2020-11-07T12:06:19.964+00:00: ***** SERVER RESTARTED *****
        2020-11-07T12:06:19.971+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger stora
ge engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
        2020-11-07T12:06:20.481+00:00: Access control is not enabled for the database. Read and write access to d
ata and configuration is unrestricted
        2020-11-07T12:06:20.481+00:00: Soft rlimits too low
        2020-11-07T12:06:20.481+00:00:         currentValue: 1024
        2020-11-07T12:06:20.481+00:00:         recommendedMinimum: 64000
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

Initiate the Config Server. You can use the **initiate()** function to initiate the config server with the default configuration. Then check the status of the server with the **status()** function.

```
rs.initiate()
rs.status()
```

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "10.10.10.56:27019",
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(1604750904, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604750904, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1604750904, 1)
}
ConfigReplSet:SECONDARY> rs.status()
{
        "set" : "ConfigReplSet",
        "date" : ISODate("2020-11-07T12:09:15.904Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "configsvr" : true,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "majorityVoteCount" : 1,
        "writeMajorityCount" : 1,
        "votingMembersCount" : 1,
        "writableVotingMembersCount" : 1,
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1604750955, 1),
```

Configuring the query router

Create the directory structure and an empty log file.

```
mkdir -pv  mongodb/data/logs
touch mongodb/data/logs/mongorouter.log
ls -alRv mongodb/
```

```
barry@mongodb02:~$ mkdir -pv  mongodb/data/logs
mkdir: created directory 'mongodb'
mkdir: created directory 'mongodb/data'
mkdir: created directory 'mongodb/data/logs'
barry@mongodb02:~$ touch mongodb/data/logs/queryrouter.log
barry@mongodb02:~$ ls -alRv mongodb/
mongodb/:
total 12
drwxrwxr-x 3 barry barry 4096 Nov  7 12:18 .
drwxr-xr-x 4 barry barry 4096 Nov  7 12:18 ..
drwxrwxr-x 3 barry barry 4096 Nov  7 12:18 data

mongodb/data:
total 12
drwxrwxr-x 3 barry barry 4096 Nov  7 12:18 .
drwxrwxr-x 3 barry barry 4096 Nov  7 12:18 ..
drwxrwxr-x 2 barry barry 4096 Nov  7 12:18 logs

mongodb/data/logs:
total 8
drwxrwxr-x 2 barry barry 4096 Nov  7 12:18 .
drwxrwxr-x 3 barry barry 4096 Nov  7 12:18 ..
-rw-rw-r-- 1 barry barry    0 Nov  7 12:18 queryrouter.log
barry@mongodb02:~$
```

Create the config file. To

create the query router, we provide the log location, IP, and port of the server instance. Further, we need to define the config server to which this query router belongs.

```
sudo nano /etc/mongoRouter.conf
```

File - mongoRouter.conf

```
systemLog:
  destination: file
  logAppend: true
  path: /home/barry/mongodb/data/logs/queryrouter.log

net:
  port: 27017
  bindIp: 10.10.10.57

sharding:
  configDB: ConfigReplSet/10.10.10.56:27019
```

Start the query router with the mongos command.

```
mongos --config /etc/mongoRouter.conf&
```

Check if the query router instance is reachable.

```
mongo 10.10.10.57:27017
```

```
barry@mongodb02:~$ mongo 10.10.10.57:27017
MongoDB shell version v4.4.1
connecting to: mongodb://10.10.10.57:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("01335db3-f824-4ece-a223-9967926e8f3e") }
MongoDB server version: 4.4.1
---
The server generated these startup warnings when booting:
        2020-11-07T12:21:34.259+00:00: ***** SERVER RESTARTED *****
        2020-11-07T12:21:34.276+00:00: Access control is not enabled for the database. Read and write access to d
ata and configuration is unrestricted
---
mongos>
```

Configuring a shard

Create the directory structure and the log file.

```
mkdir -pv  mongodb/data/sharddb/
mkdir -pv  mongodb/data/logs
touch mongodb/data/logs/shard.log
ls -alRv mongodb/
```

```
barry@mongodb03:~$ mkdir -pv  mongodb/data/sharddb/
mkdir: created directory 'mongodb'
mkdir: created directory 'mongodb/data'
mkdir: created directory 'mongodb/data/sharddb/'
barry@mongodb03:~$ mkdir -pv  mongodb/data/logs
mkdir: created directory 'mongodb/data/logs'
barry@mongodb03:~$ touch mongodb/data/logs/shard.log
barry@mongodb03:~$ ls -alRv mongodb/
mongodb/:
total 12
drwxrwxr-x 3 barry barry 4096 Nov  7 12:30 .
drwxr-xr-x 4 barry barry 4096 Nov  7 12:30 ..
drwxrwxr-x 4 barry barry 4096 Nov  7 12:30 data

mongodb/data:
total 16
drwxrwxr-x 4 barry barry 4096 Nov  7 12:30 .
drwxrwxr-x 3 barry barry 4096 Nov  7 12:30 ..
drwxrwxr-x 2 barry barry 4096 Nov  7 12:30 logs
drwxrwxr-x 2 barry barry 4096 Nov  7 12:30 sharddb

mongodb/data/logs:
total 8
drwxrwxr-x 2 barry barry 4096 Nov  7 12:30 .
drwxrwxr-x 4 barry barry 4096 Nov  7 12:30 ..
-rw-rw-r-- 1 barry barry    0 Nov  7 12:30 shard.log

mongodb/data/sharddb:
total 8
drwxrwxr-x 2 barry barry 4096 Nov  7 12:30 .
drwxrwxr-x 4 barry barry 4096 Nov  7 12:30 ..
barry@mongodb03:~$
```

Create the config file. The shard's config file contains the paths for the database storage, logs, and sharding cluster role, which is set to **shardsvr**. It also includes the network settings to the server instance. Finally, we have set **replSetName** allowing the data to be replicated.

```
sudo nano /etc/mongodShard.conf
```

File - mongoShard.conf

```
storage:
  dbPath: /home/barry/mongodb/data/sharddb
  journal:
    enabled: true

systemLog:
  destination: file
  logAppend: true
  path: /home/barry/mongodb/data/logs/shard.log

net:
  port: 27018
  bindIp: 10.10.10.58

sharding:
  clusterRole: shardsvr
```

```
replication:
  replSetName: ShardReplSet
```

Start the shard server.

```
mongod --config /etc/mongodShard.conf&
```

```
barry@mongodb03:~$ mongod --config /etc/mongodShard.conf&
[1] 6396
barry@mongodb03:~$
```

Check the logs to verify if the server is running.

```
tail -100 mongodb/data/logs/shard.log
```

```
barry@mongodb03:~$ tail -100 mongodb/data/logs/shard.log

{"t":{"$date":"2020-11-07T12:33:47.841+00:00"},"s":"W",  "c":"CONTROL",  "id":20698,   "ctx":"main","msg":"*****
SERVER RESTARTED *****","tags":["startupWarnings"]}
{"t":{"$date":"2020-11-07T12:33:47.846+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main","msg":"Automa
tically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2020-11-07T12:33:47.852+00:00"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx":"main","msg":"No Tra
nsportLayer configured during NetworkInterface startup"}
{"t":{"$date":"2020-11-07T12:33:47.853+00:00"},"s":"I",  "c":"NETWORK",  "id":4648601, "ctx":"main","msg":"Implic
it TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOp
enQueueSize."}
{"t":{"$date":"2020-11-07T12:33:47.853+00:00"},"s":"I",  "c":"STORAGE",  "id":4615611, "ctx":"initandlisten","msg
":"MongoDB starting","attr":{"pid":6396,"port":27018,"dbPath":"/home/barry/mongodb/data/sharddb","architecture":"
64-bit","host":"mongodb03"}}
```

Login to the shard server.

```
mongo 10.10.10.58:27018
```

```
barry@mongodb03:~$ mongo 10.10.10.58:27018
MongoDB shell version v4.4.1
connecting to: mongodb://10.10.10.58:27018/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d4d0544d-4289-4f05-b7a0-89253e0423f9") }
MongoDB server version: 4.4.1
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2020-11-07T12:33:47.841+00:00: ***** SERVER RESTARTED *****
        2020-11-07T12:33:47.855+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger stora
ge engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
        2020-11-07T12:33:48.403+00:00: Access control is not enabled for the database. Read and write access to d
ata and configuration is unrestricted
        2020-11-07T12:33:48.403+00:00: Soft rlimits too low
        2020-11-07T12:33:48.403+00:00:         currentValue: 1024
        2020-11-07T12:33:48.403+00:00:         recommendedMinimum: 64000
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

Initiate the shard server. The replication with the default configuration is enabled using the **initiate()** function. Then check the status of the initialization with the **status()** function.

```
rs.initiate()
```

```
rs.status()
```

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "10.10.10.58:27018",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604752870, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1604752870, 1)
}
ShardReplSet:SECONDARY> rs.status()
{
        "set" : "ShardReplSet",
        "date" : ISODate("2020-11-07T12:41:32.901Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "majorityVoteCount" : 1,
        "writeMajorityCount" : 1,
        "votingMembersCount" : 1,
        "writableVotingMembersCount" : 1,
```

Adding the shard to the Cluster

Connect to the Query Router.

```
mongo 10.10.10.57:27017
```

Add the Shard to the clustet. Using the **addShard()** command, we provide the **replSetName** with the IP address and the port of the shard instance.

```
sh.addShard( "ShardReplSet/10.10.10.58:27018")
```

```
barry@mongodb01:~$ mongo 10.10.10.57:27017
MongoDB shell version v4.4.1
connecting to: mongodb://10.10.10.57:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("bc752614-0a4c-43e1-a053-739089260103") }
MongoDB server version: 4.4.1
---
The server generated these startup warnings when booting:
        2020-11-07T12:21:34.259+00:00: ***** SERVER RESTARTED *****
        2020-11-07T12:21:34.276+00:00: Access control is not enabled for the database. Read and write access to
ata and configuration is unrestricted
---
mongos> sh.addShard( "ShardReplSet/10.10.10.58:27018")
{
        "shardAdded" : "ShardReplSet",
        "ok" : 1,
        "operationTime" : Timestamp(1604753143, 9),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604753143, 9),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

Enabling the sharding

Create the database. A database named “persons” will be created using the query router. This database will be used in the sharding operation. Here the necessary steps to enable sharding:

```
use persons
sh.enableSharding("persons")
```

```
mongos> use persons
switched to db persons
mongos> sh.enableSharding("persons")
{
        "ok" : 1,
        "operationTime" : Timestamp(1604753646, 8),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604753646, 8),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

Check sharding status. Using the **status()** command, we check if the sharding is enabled to the database. As shown below, sharding is enabled on the “persons” database.

```
sh.status()
```

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "10.10.10.56:27019",
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(1604750904, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604750904, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1604750904, 1)
}
ConfigReplSet:SECONDARY> rs.status()
{
        "set" : "ConfigReplSet",
        "date" : ISODate("2020-11-07T12:09:15.904Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "configsvr" : true,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "majorityVoteCount" : 1,
        "writeMajorityCount" : 1,
        "votingMembersCount" : 1,
        "writableVotingMembersCount" : 1,
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1604750955, 1),
```

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5fa68e38167df2c78d404b3d")
  }
  shards:
        {  "_id" : "ShardReplSet",  "host" : "ShardReplSet/10.10.10.58:27018",  "state" : 1 }
  active mongoses:
        "4.4.1" : 1
  autosplit:
        Currently enabled: yes
  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" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                ShardReplSet    1024
                        too many chunks to print, use verbose if you want to force print
        {  "_id" : "persons",  "primary" : "ShardReplSet",  "partitioned" : true,  "version" : {  "uuid" : UUID("
8e4c1ee2-cfde-4eab-9d95-a5ace436d246"),  "lastMod" : 1 } }
mongos>
```

Creating the sharding dataset

In this section, we will create a MongoDB collection and an index called **person_id** to be used as the shard key. The data in the collection will be sharded using the collection and the shard key.

Create the collection.

```
db.createCollection("personscollection")
```

```
mongos> db.createCollection("personscollection")
{
        "ok" : 1,
        "operationTime" : Timestamp(1604754109, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604754109, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

Create the index and add a record. Create the index with **personid** as the field in descending order.

```
db.personscollection.createIndex({personid: -1})
```

Add a **personid** with value 10001.

```
db.personscollection.insertOne({personid: 10001})
```

```
mongos> db.personscollection.createIndex({personid: -1})
{
        "raw" : {
                "ShardReplSet/10.10.10.58:27018" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1604754423, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604754423, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> db.personscollection.insertOne({personid: 10001})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fa69c12d3c67381c2a49461")
}
mongos>
```

Enabling sharding for the “personscollection”

The first step is to make sure the index **personid** is hashed. If not, this will result in an error, and sharding will fail.

To achieve this, we use **ensureIndex()** function within MongoDB.

```
db.personscollection.ensureIndex({personid : "hashed"})
```

If the index is used correctly, we can enable Sharding with “personid” as the shard key.

```
sh.shardCollection("persons.personscollection", {personid : "hashed"})
```

If the operation is successful, the following output will be displayed.

```
mongos> db.personscollection.ensureIndex({personid : "hashed"})
{
        "raw" : {
                "ShardReplSet/10.10.10.58:27018" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1604755070, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604755070, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.shardCollection("persons.personscollection", {personid : "hashed"})
{
        "collectionsharded" : "persons.personscollection",
        "collectionUUID" : UUID("7dfe9dd5-e609-420c-aaf9-0daa9d2af810"),
        "ok" : 1,
        "operationTime" : Timestamp(1604755083, 9),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1604755083, 9),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

Verify if sharding is working as intended. Use the **getShardDistribution()** command to verify the status of the sharding operation.

`db.personscollection.getShardDistribution()`

```
mongos> db.personscollection.getShardDistribution()

Shard ShardReplSet at ShardReplSet/10.10.10.58:27018
 data : 40B docs : 1 chunks : 1
 estimated data per chunk : 40B
 estimated docs per chunk : 1

Totals
 data : 40B docs : 1 chunks : 1
 Shard ShardReplSet contains 100% data, 100% docs in cluster, avg obj size on shard : 40B

mongos>
```

The above output describes that the **personscollection** is sharded in the **ShardRepSet** on the Shard server (10.10.10.58). It consists of a single document in a single chunk. The available document is the single record we entered into the collection.

Summing up MongoDB sharding

MongoDB sharding is a method to manage large data sets efficiently by distributing the workload

across many servers without having any adverse effects on the overall performance of the database. Also, sharding provides the ability to efficiently scale the cluster for future requirements without a complex restructuring of the underlying hardware infrastructure.

Related reading

- BMC Machine Learning & Big Data Blog
- MongoDB Guide, a series of articles and tutorials
- MongoDB: The Mongo Shell & Basic Commands
- Data Storage Explained: Data Lake vs Warehouse vs Database