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. Use the right-hand menu to navigate.)

**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 chunk 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/
```
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**

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

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

Check the logs to verify if the server is running.

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

Connect to the config server.

```
mongo 10.10.10.56:27019
```

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()
```
Configuring the query router

Create the directory structure and an empty log file.

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

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 4.4.1 | Connecting to: mongodb://10.10.10.57:27017/test?compressors=disabled&gssapiServiceName=mongodb
| Implicit session: session { "id" : UUID("01335db3-f824-4ece-a223-9907920e8f3e") } |
| MongoDB server version: 4.4.1 | The server generated these startup warnings when booting:

2020-11-07T12:21:34.259+08:00: ***** SERVER RESTARTED *****
2020-11-07T12:21:34.276+08:00: Access control is not enabled for the database. Read and write access to data 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/
```

---
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.

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

```
File - mongoShard.conf

storage:
  dbPath: /home/barry/mongodb/data/sharddb

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

net:
  port: 27018
  bindIp: 10.10.10.58

sharding:
  clusterRole: shardsvr
```
Start the shard server.

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

Check the logs to verify if the server is running.

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

Login to the shard server.

`mongo 10.10.10.58:27018`

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()`
Adding the shard to the Cluster

Connect to the Query Router.

mongo 10.10.10.57:27017

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

sh.addShard("ShardRep1Set/10.10.10.58:27018")
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:

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

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.

```plaintext
sh.status()
```
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.

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

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

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

Add a `personid` with value 10001.

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

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

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

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

If the operation is successful, the following output will be displayed.
Verify if sharding is working as intended. Use the `getShardDistribution()` command to verify the status of the sharding operation.

```plaintext
db.personscollection.getShardDistribution()
```

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