MongoDB - Replication & Sharding

1

Course Map

- Understand about Replication
- Purpose of Replication
Understand Replica Set
Sharding
- Sharding Mechanics
- GridFS

Module Objectives

What you will learn

At the end of this module, you will learn:

- Replication and Sharding in MongoDB

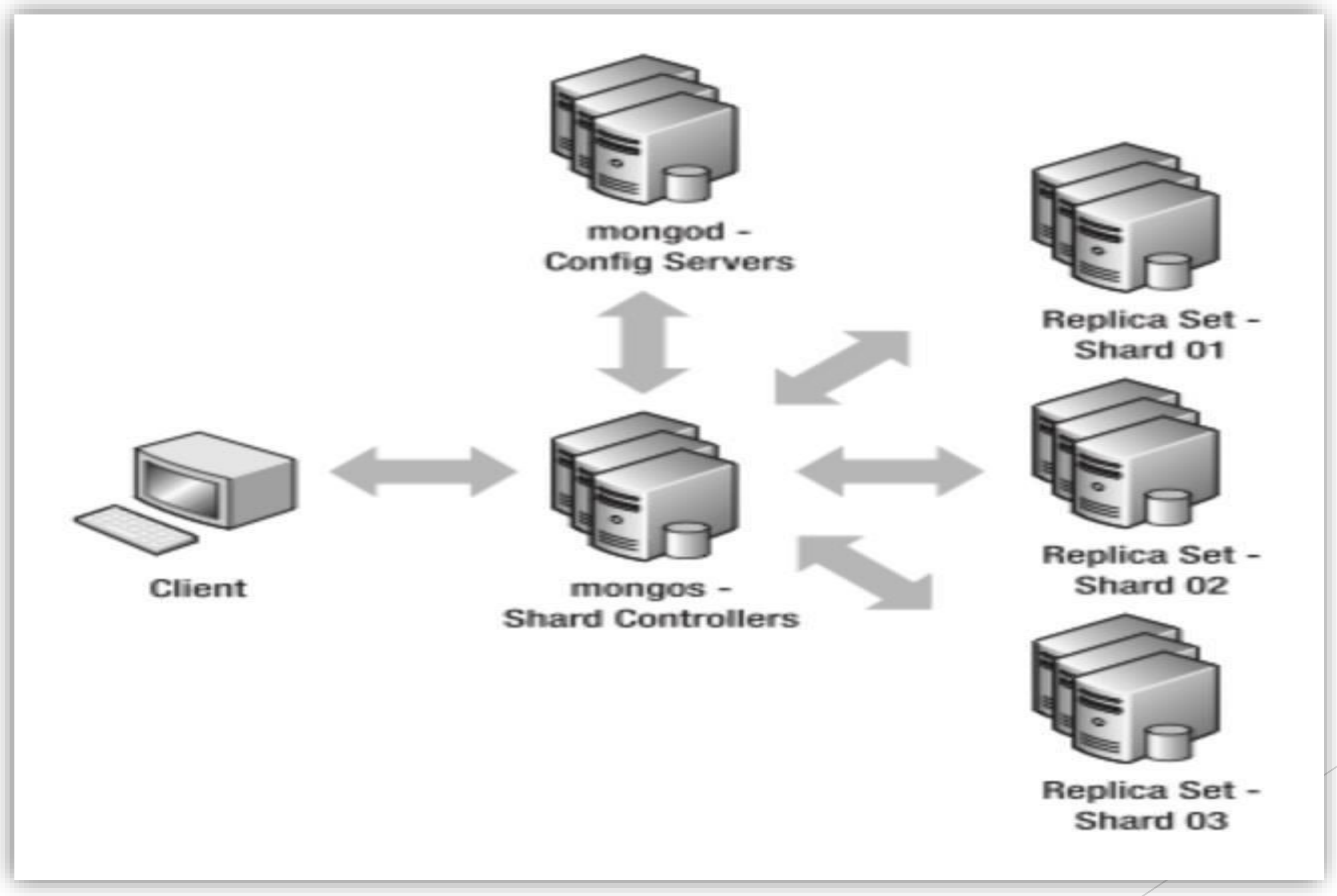
What you will be able to do

At the end of this module, you be able to:

- Understand what is Replication and Sharding
- Explain the purpose of Replication
- Describe what is a Replica Set
- State the features of Sharding mechanics
- Explain GridFS

3

Replication & Sharding



Replication

What is replication?

Purpose of replication / Redundancy

- Fault tolerance
- Availability
  - Increase read capacity

Why Replication?

How many have faced node failures?

How many have been woken up from sleep to do a fail-over(s)?

How many have experienced issues due to network latency?

Different uses for data:

- Normal processing
- Simple analytics

Replication in MongoDB

Replica Set Members

- **Primary**
  - Read, Write operations
- **Secondary**
  - Asynchronous Replication
  - Can be primary
- **Arbiter**
  - Voting
  - Can't be primary
- **Delayed Secondary**
  - Can't be primary

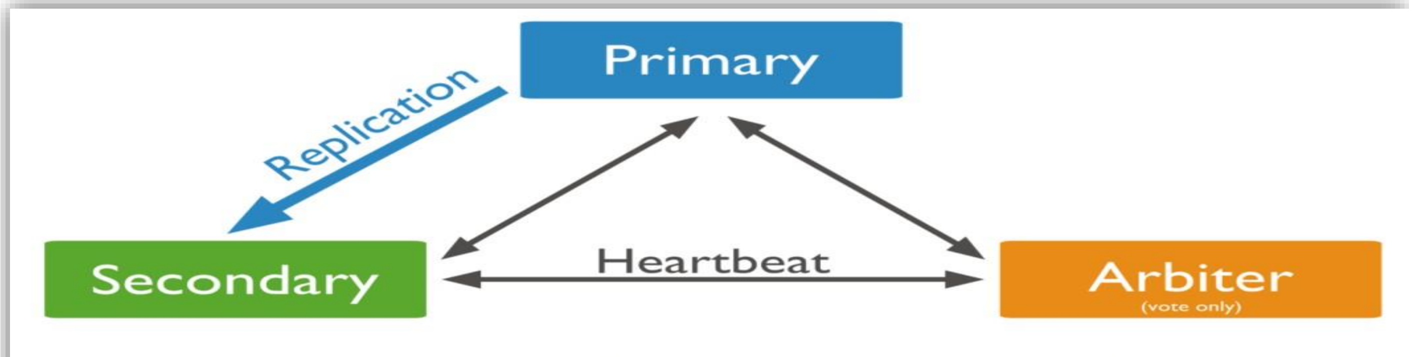


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

Replication in MongoDB

Automatic Failover

- Heartbeats
- Elections

The Standard Replica Set Deployment

Deploy an Odd Number of Members

Rollback

Security

- SSL/TLS

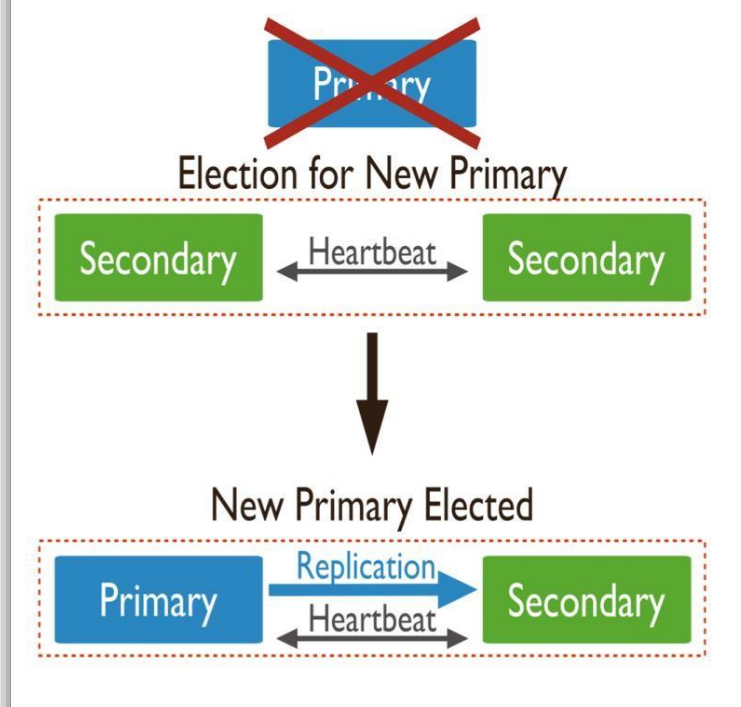


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

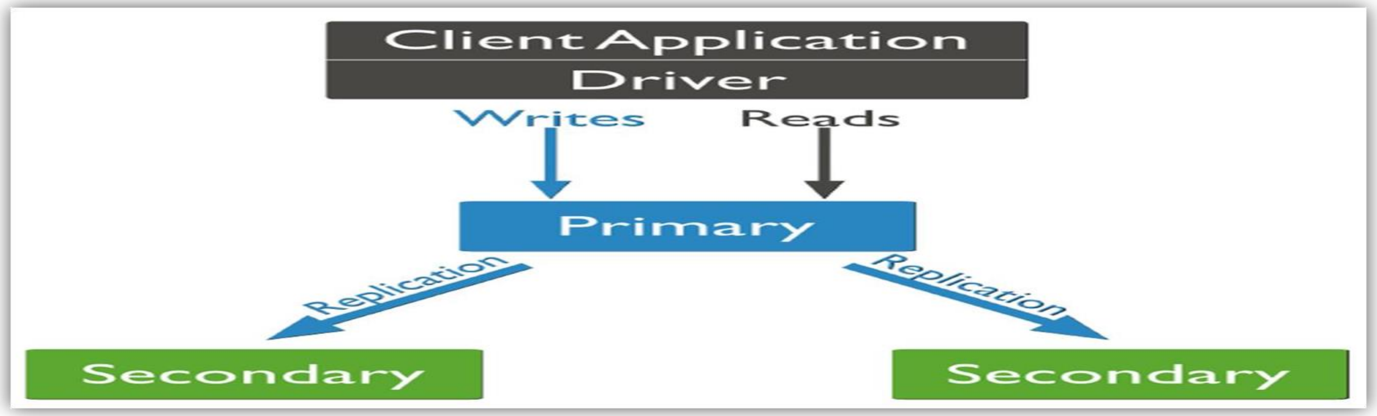
Replication

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication protects a database from the loss of a single server. Replication also allows to recover from hardware failure and service interruptions. With additional copies of the data, can dedicate one to disaster recovery, reporting, or backup.

In some cases, we can use replication to increase read capacity. Clients have the ability to send read and write operations to different servers. We can also maintain copies in different data centers to increase the locality and availability of data for distributed applications.

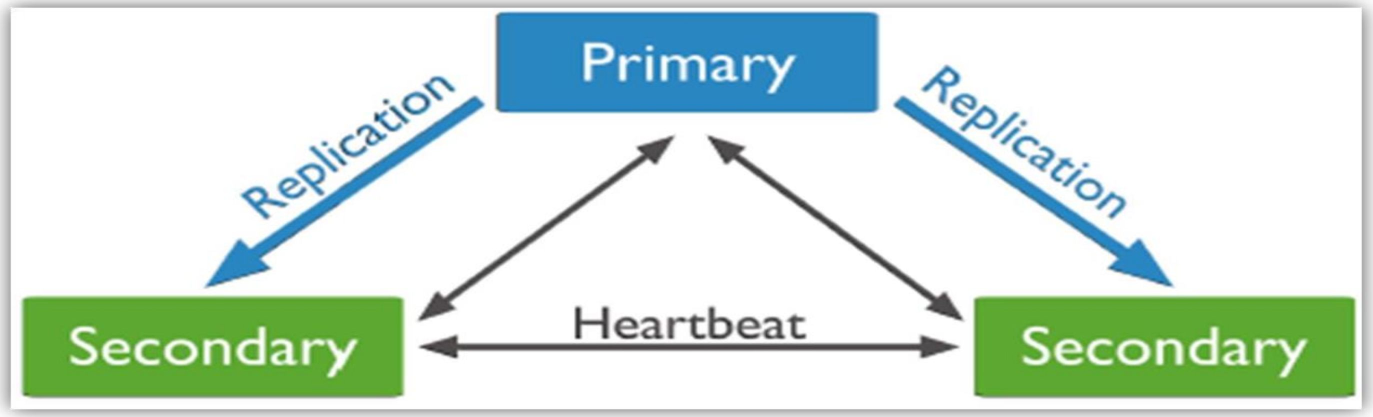
A replica set in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments.

Replication in MongoDB



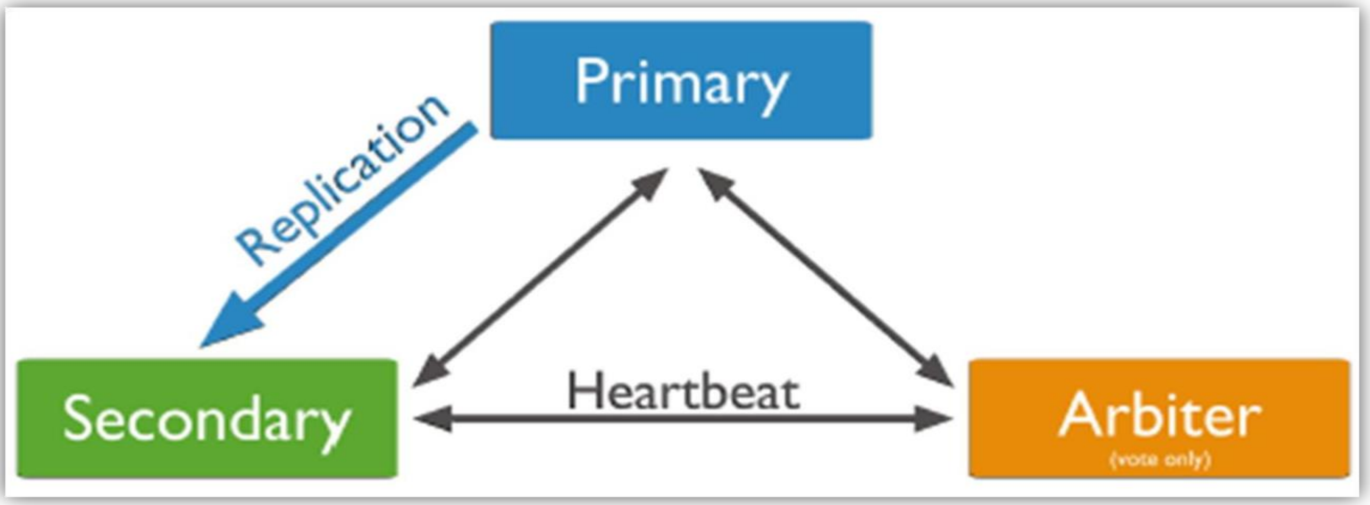
A replica set is a group of mongod instances that host the same data set. One mongod, the primary, receives all write operations. All other instances, secondaries, apply operations from the primary so that they have the same data set.	The primary accepts all write operations from clients. Replica set can have only one primary. Because only one member can accept write operations, replica sets provide strict consistency for all reads from the primary. To support replication, the primary logs all changes to its data sets in its oplog.

Replication in MongoDB (contd.)



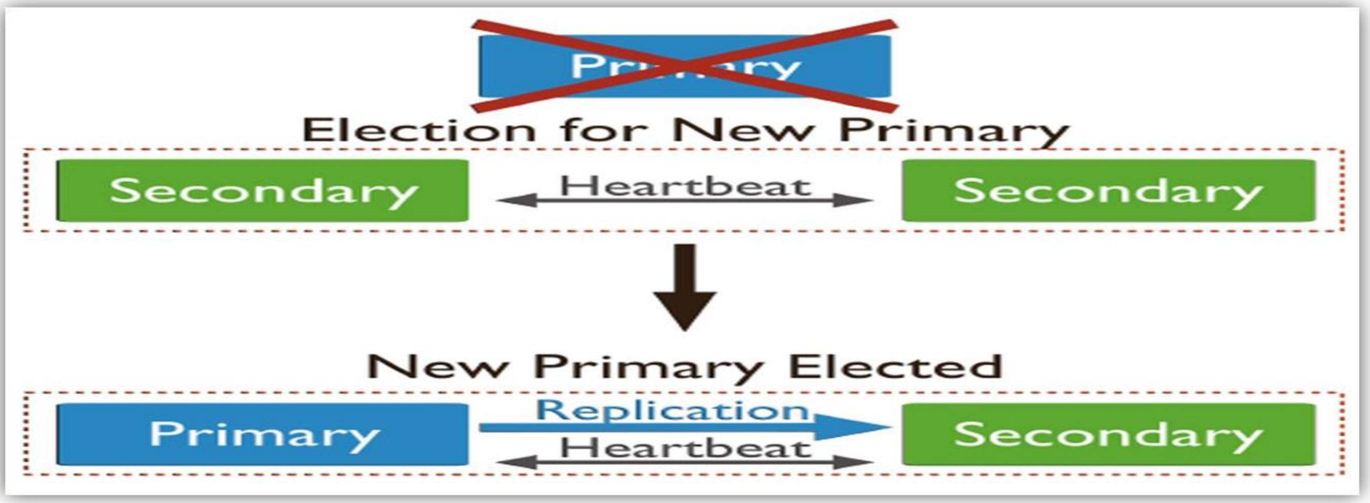
The secondaries replicate the primary’s oplog and apply the operations to their data sets. Secondary's data sets reflect the primary’s data set. If the primary is unavailable, the replica set will elect a secondary to be primary.	By default, clients read from the primary, however, clients can specify a read preferences to send read operations to secondaries.

Replication in MongoDB (contd.)



We can add an extra mongod instance to a replica set as an arbiter. Arbiters do not maintain a data set. Arbiters only exist to vote in elections. If your replica set has an even number of members, add an arbiter to obtain a majority of votes in an election for primary. Arbiters do not require dedicated hardware.

Replication in MongoDB (contd.)



Automatic Failover

When a primary does not communicate with the other members of the set for more than 10 seconds, the replica set will attempt to select another member to become the new primary. The first secondary that receives a majority of the votes becomes primary.

Replica Set Members

Primary:	• Receives all write operations.
Secondaries:	• Replicate operations from the primary to maintain an identical data set. Secondaries may have additional configurations for special usage profiles.
Arbiters:	• Do not keep a copy of the data. However, arbiters play a role in the elections that select a primary if the current primary is unavailable.
Priority 0 Replica Set Members:	• A priority 0 member is a secondary that cannot become primary. Priority 0 members cannot trigger elections. Otherwise these members function as normal secondaries. A priority 0 member maintains a copy of the data set, accepts read operations, and votes in elections. Configure a priority 0 member to prevent secondaries from becoming primary, which is particularly useful in multi-data center deployments. It can be used as standby.
Hidden Replica Set Members:	• A hidden member maintains a copy of the primary's data set but is invisible to client applications. Hidden members are good for workloads with different usage patterns from the other members in the replica set. Hidden members must always be priority 0 members and so cannot become primary.
Delayed Replica Set Members:	• Delayed members contain copies of a replica set's data set. However, a delayed member's data set reflects an earlier, or delayed, state of the set. For example, if the current time is 09:52 and a member has a delay of an hour, the delayed member has no operation more recent than 08:52. It help to recover from various kinds of human error. For example, a delayed member can make it possible to recover from unsuccessful application upgrades and operator errors including dropped databases and collections.

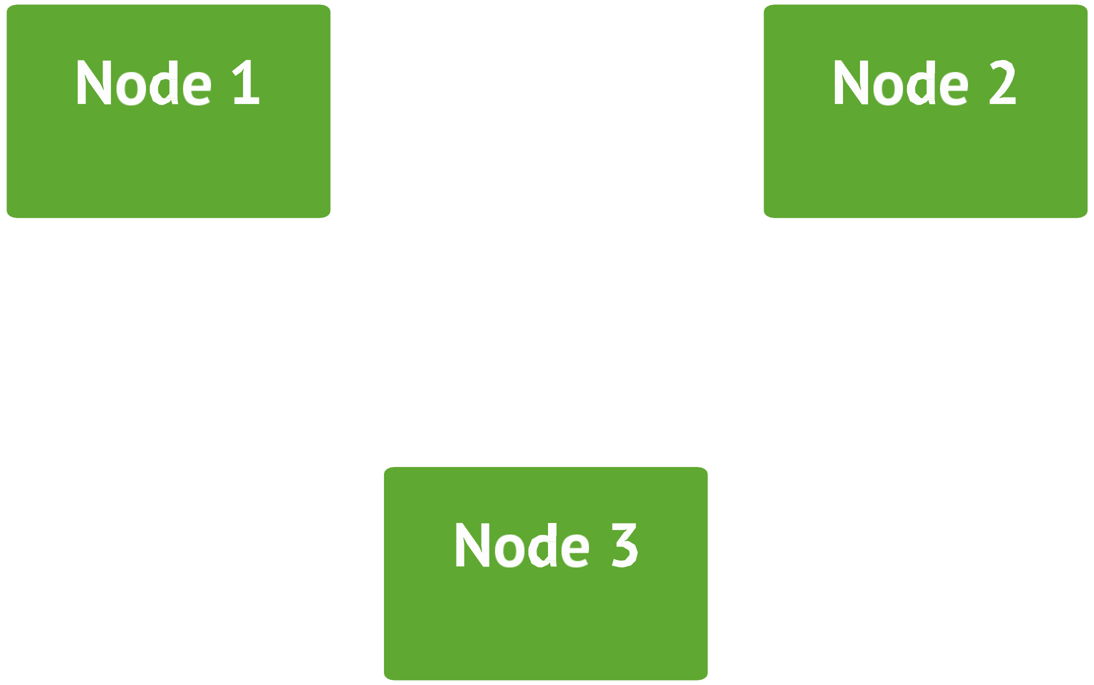
Replica Set Members (contd.)

A replica set can have up to 12 members. However, only 7 members can vote at a time.

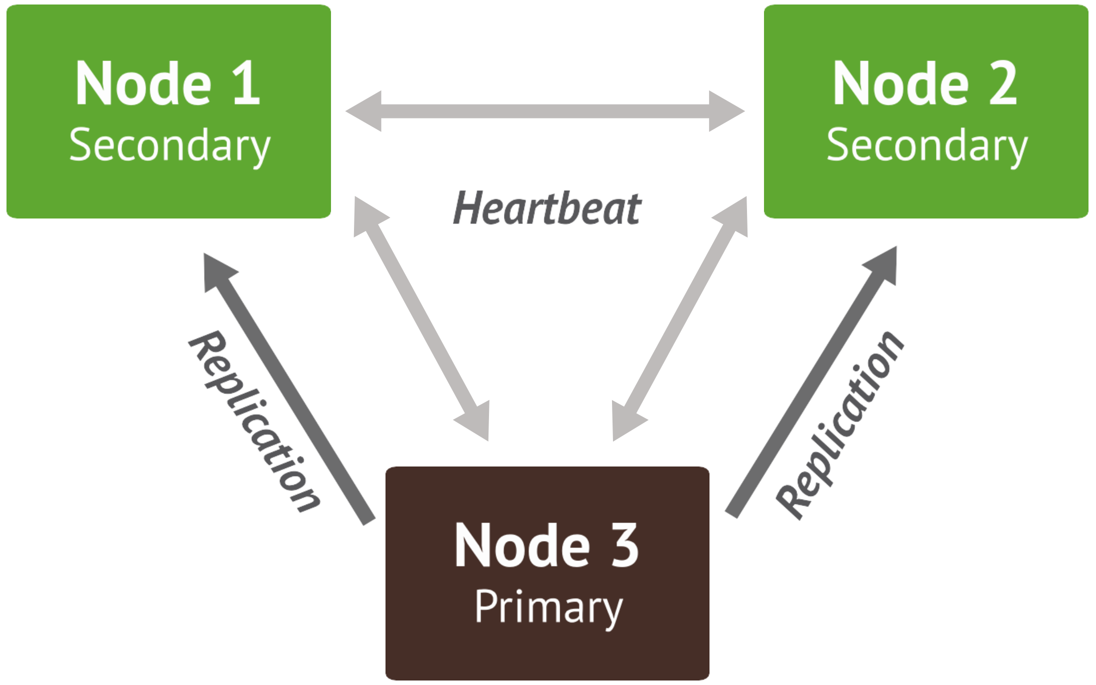
The minimum requirements for a replica set are:

- Primary
- Secondary
- Arbiter.

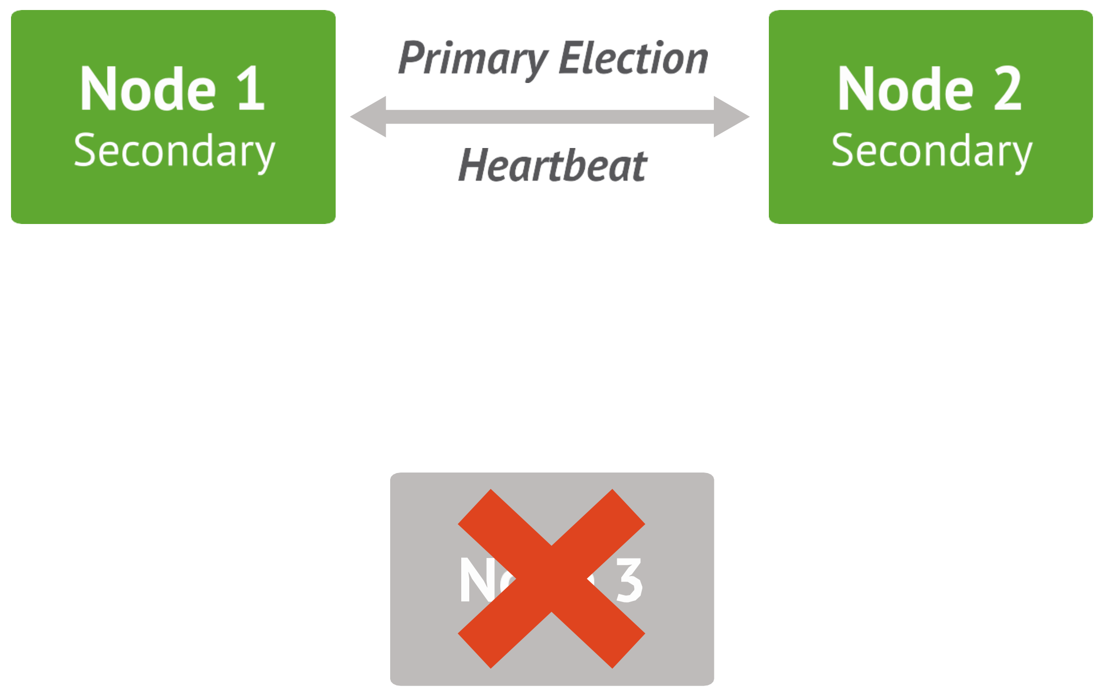
Replica Set Lifecycle - Creation



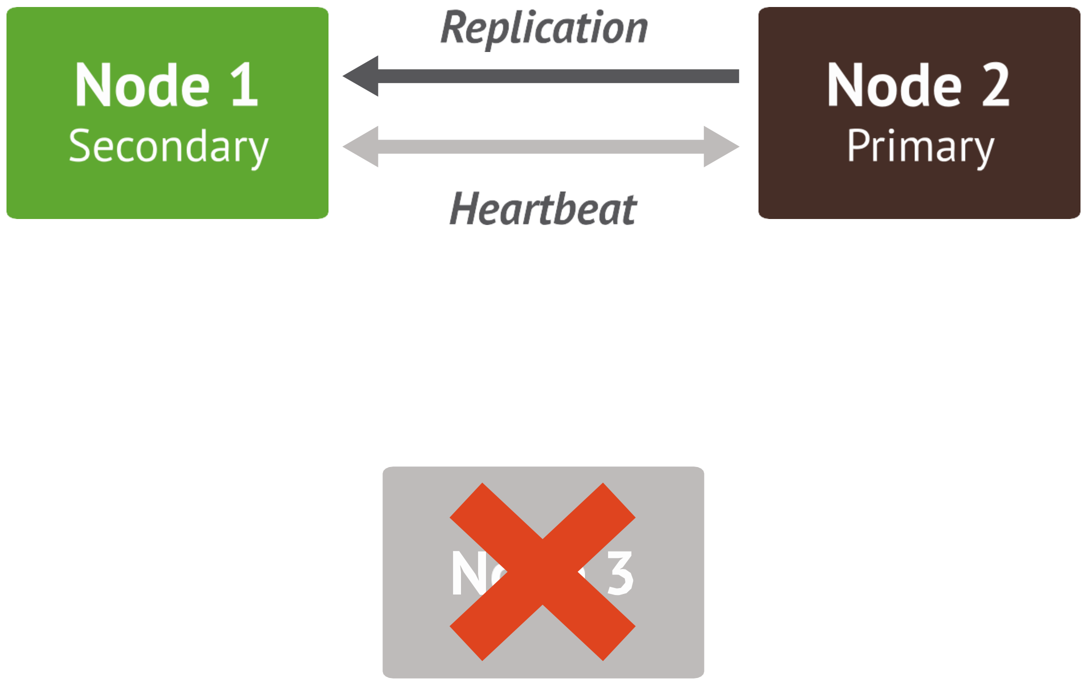
Replica Set Lifecycle - Initialize



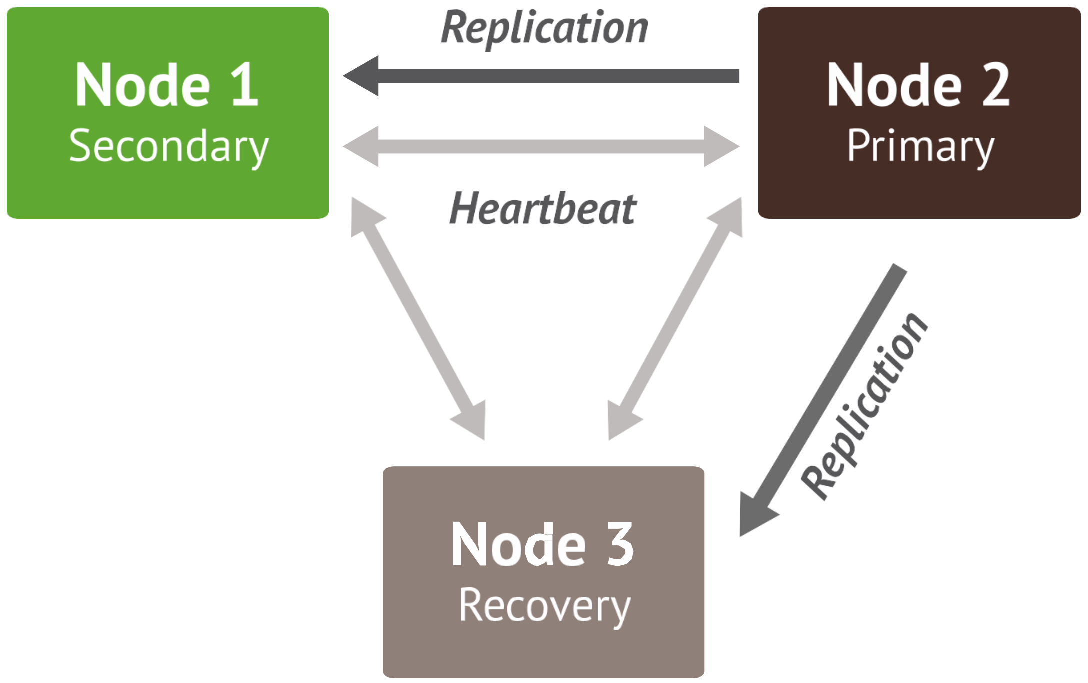
Replica Set Lifecycle - Failure



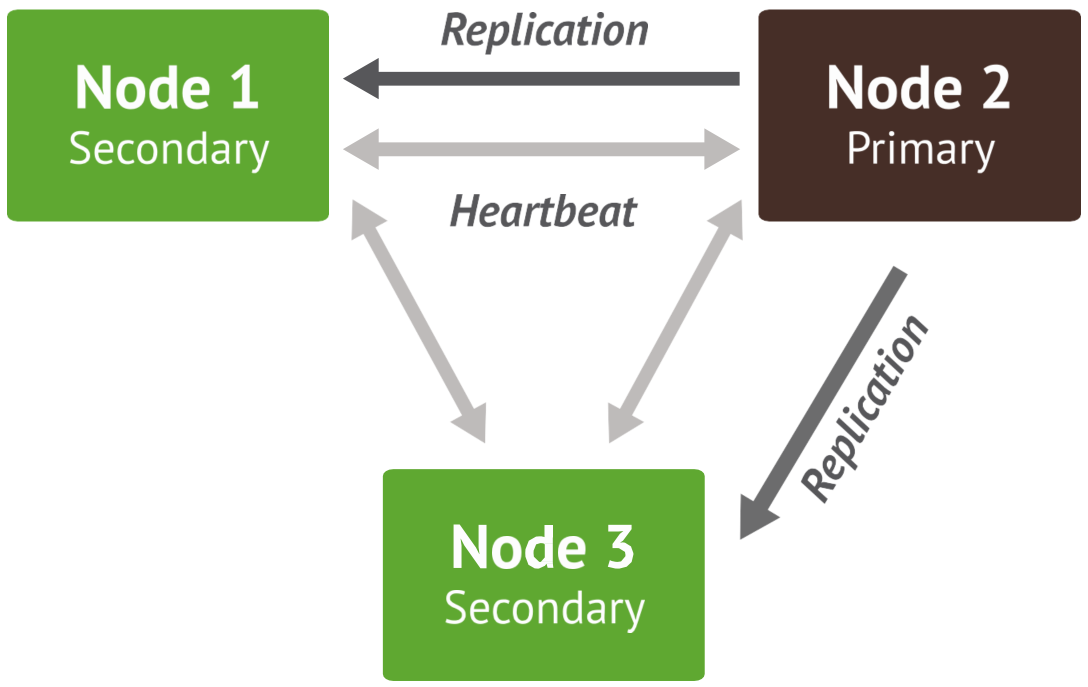
Replica Set Lifecycle - Failover



Replica Set Lifecycle - Recovery



Replica Set Lifecycle - Recovered



Setting up Local Replica Sets

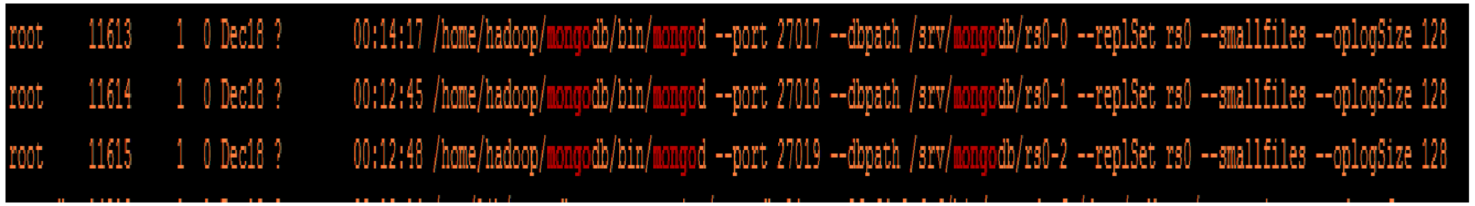
Make 3 directories to set up 3 replicas on local node.

mkdir -p /data/mongodb/rs0-0 /data/mongodb/rs0-1 /data/mongodb/rs0-2

Start 3 mongo daemons on local node with 3 different available ports.

- mongod --port 27017 --dbpath /data/mongodb/rs0-0 --replSet rs0 --smallfiles --oplogSize 128
- mongod --port 27018 --dbpath /data/mongodb/rs0-1 --replSet rs0 --smallfiles --oplogSize 128
- mongod --port 27019 --dbpath /data/mongodb/rs0-2 --replSet rs0 --smallfiles --oplogSize 128

Check the mongo daemons running with “ps –ef|grep mongo” command.



```
root     11613     1  0 Dec18 ?        00:14:17 /home/hadoop/mongodb/bin/mongod --port 27017 --dbpath /srv/mongodb/rs0-0 --replSet rs0 --smallfiles --oplogSize 128
root     11614     1  0 Dec18 ?        00:12:45 /home/hadoop/mongodb/bin/mongod --port 27018 --dbpath /srv/mongodb/rs0-1 --replSet rs0 --smallfiles --oplogSize 128
root     11615     1  0 Dec18 ?        00:12:48 /home/hadoop/mongodb/bin/mongod --port 27019 --dbpath /srv/mongodb/rs0-2 --replSet rs0 --smallfiles --oplogSize 128
```

Failover and Primary Election

This election process will be initiated by any node that cannot reach the primary.

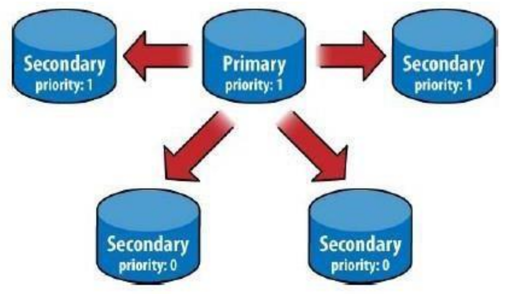
- The new primary must be elected by a *majority of the nodes in the set. The new primary will be the node with the highest priority, using freshness of data to break ties between nodes with the same priority.*

The primary node uses a heartbeat to track how many nodes in the cluster are visible to it.

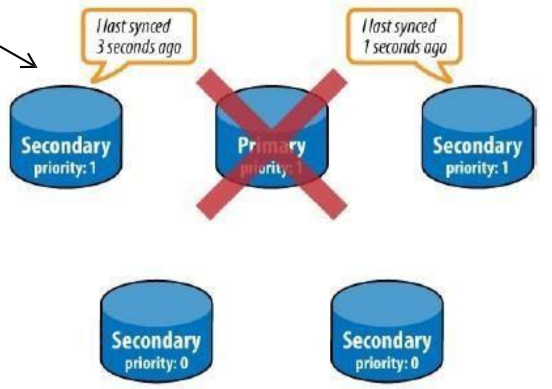
- If this falls below a majority, the primary will automatically fall back to secondary status.

Whenever the primary changes, the data on the new primary is assumed to be the most up-to-date data in the system.

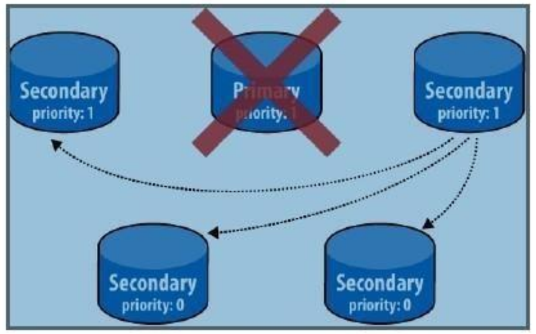
Failover and Primary Election (contd.)



replica set can have several servers of different priority levels



If the primary goes down, the highest-priority servers will compare how up-to-date they are



The highest-priority most-up-to-date server will become the new primary

Sharding

What is Sharding?

Purpose of Sharding

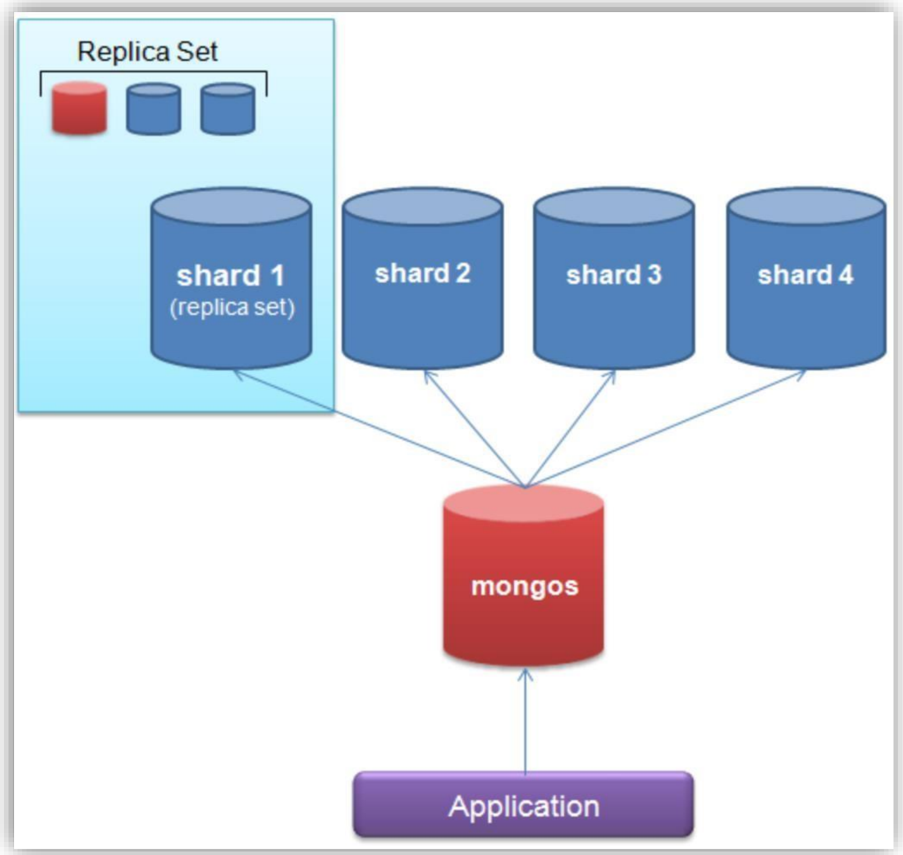
- Horizontal scaling out

Query Routers

- mongos

Shard keys

- Range based Sharding
- Cardinality
- Avoid hot spotting



Main Components

Shard

- A Shard is a node of the cluster.
- Each Shard can be a single mongod or a replica set.

Config Server (meta data storage)

- Stores cluster chunk ranges and locations.
- Can be only 1 or 3 (production must have 3).
- Not a replica set.

Mongos

- Acts as a router / balancer.
- No local data (persists to config database).
- Can be 1 or many.

Sharding

Sharding is MongoDB’s approach to scaling out.

Sharding allows you to add more machines to handle increasing load and data size without affecting your application.

Manual sharding will work well but become difficult to maintain when adding or removing nods from the cluster or in face of changing data distributions or load patterns.

MongoDB supports autosharding, which eliminates some of the administrative headaches of manual sharding.

Auto Sharding in MongoDB

The basic concept behind MongoDB's sharding is to

break up collections into small chunks.

- We don't need to know what shard has what data, so we run a *router in front of the application, it knows where the data located, so applications can connect to it and issue requests normally.*
- An application will connected to a normal mongod, the router, knowing what data is on which shard, is able to forward the requests to the appropriate shard(s).

When to Shard

In general, you should start with a non sharded setup and convert it to a sharded one, if and when you need.

When the situations like this, you should probably start to shard:

- You’ve run out of disk space on your current machine.
- You want to write data faster than a single mongod can handle.
- You want to keep a larger proportion of data in memory to improve performance.

Shard Keys

When you set up sharding, you choose a key from a collection and use that key’s values to split up the data. This key is call a *shard key.*

For example, If we chose "name" as our shard key, one shard could hold documents where the "name" started with A–F, the next shard could hold names from G–P, and the final shard would hold names from Q–Z.

As you added (or removed) shards, MongoDB would rebalance this data so that each shard was getting a balanced amount of traffic and a sensible amount of data.

Chunks

Suppose we add a new shard. Once this shard is up and running, MongoDB will break up the collection into two pieces, called chunks.

A chunk contains all of the documents for a range of values for the shard key.

- For example, if we use "timestamp" as the shard key, so one chunk would have documents with a timestamp value between -∞ and, say, June 26, 2003, and the other chunk would have timestamps between June 27, 2003 and ∞.

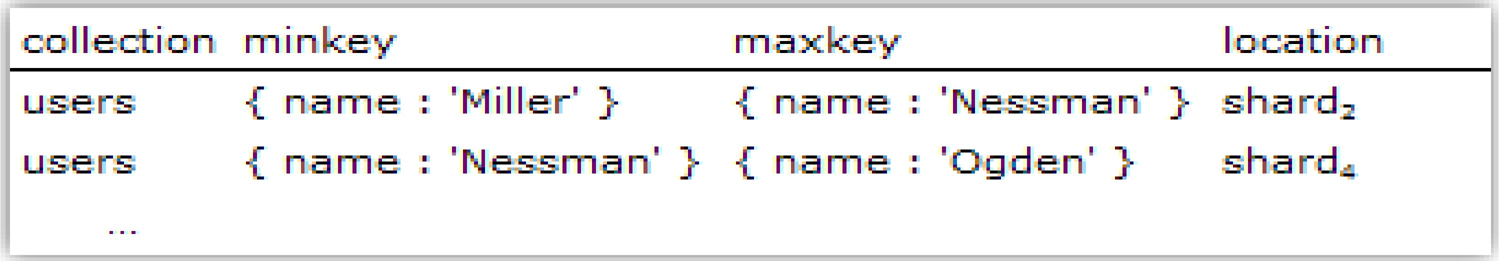
Pre Sharding a Table

Determine a shard key

- Define how we distribute data.
- MongoDB's sharding is order-preserving; adjacent data by shard key tends to be on the same server.
- The config database stores all the metadata indicating the location of data by range:
- It should be granular enough to ensure an even distribution of data.

Chunks

- A contiguous range of data from a particular collection.
- Once a chunk has reached about 200M size, the chunk splits into two new chunks. When a particular shard has excess data, chunks will then migrate to other shards in the system.
- The addition of a new shard will also influence the migration of chunks.



collection	minkey	maxkey	location
users	{ name : 'Miller' }	{ name : 'Nessman' }	shard$_2$
users	{ name : 'Nessman' }	{ name : 'Ogden' }	shard$_4$
...			

Sharding a Table

Enable sharding on a database

```
db.runCommand({"enablesharding" : "foo"})
```

Enable sharding on collection.

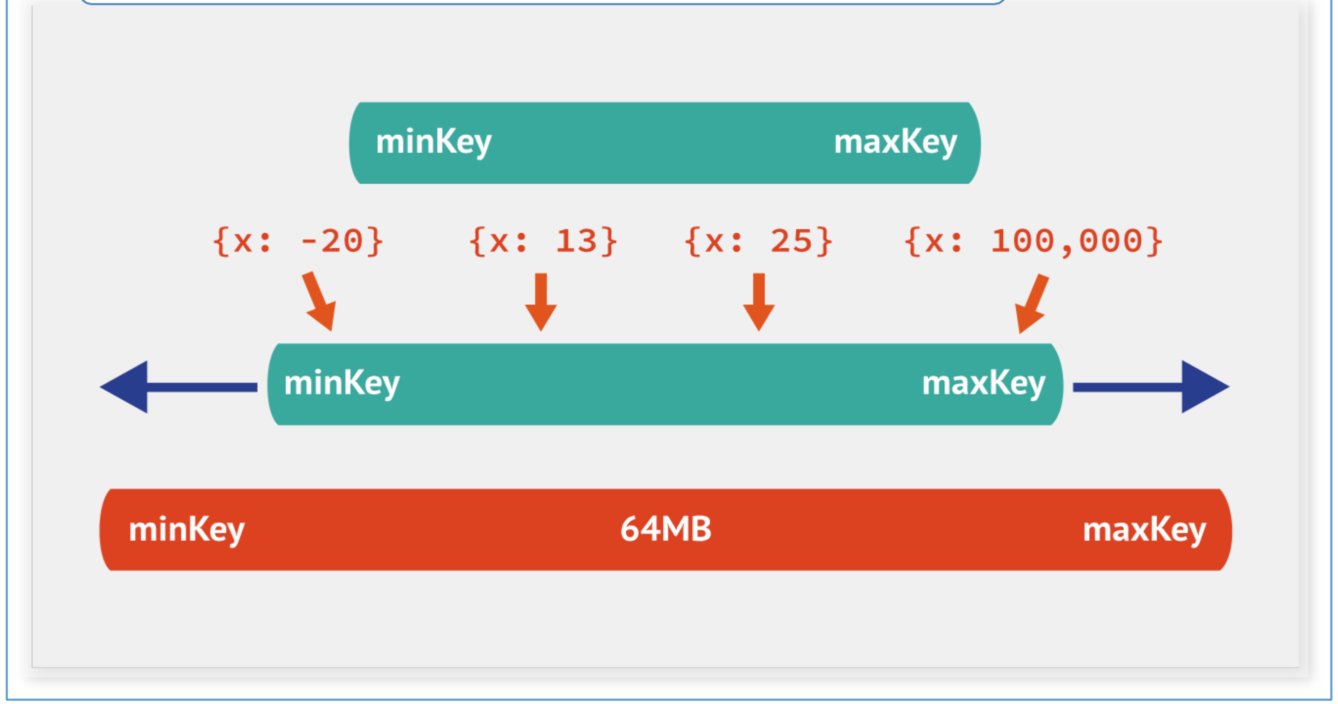
```
db.runCommand({"shardcollection" : "foo.bar", "key" : {"_id" : 1}})
```

Show autosharding status

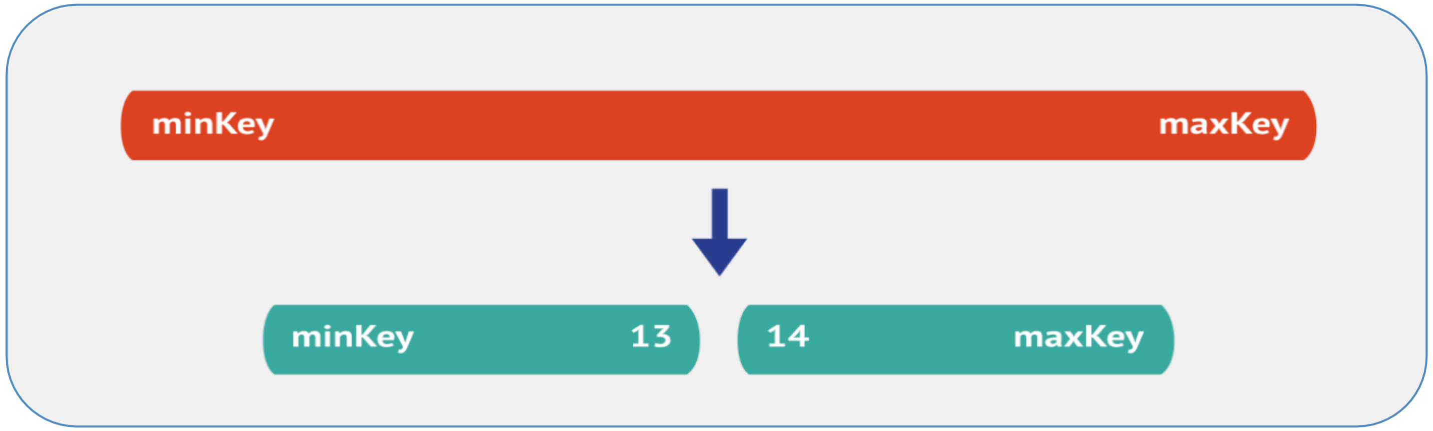
```
> db.printShardingStatus()
--- Sharding Status ---
sharding version: { "_id" : 1, "version" : 3 }
shards:
{ "_id" : "shard0", "host" : "localhost:10000" }
{ "_id" : "shard1", "host" : "localhost:10001" }
databases:
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "foo", "partitioned" : false, "primary" : "shard1" }
{ "_id" : "x", "partitioned" : false, "primary" : "shard0" }
{ "_id" : "test", "partitioned" : true, "primary" : "shard0",
"sharded" : { "test.foo" : { "key" : { "x" : 1 }, "unique" : false } } }
test.foo chunks:
{ "x" : { $minKey : 1 } } -->> { "x" : { $maxKey : 1 } } on : shard0
{ "t" : 1276636243000, "i" : 1 }
```

Chunk Partitioning

Chunk is a section of the entire range.

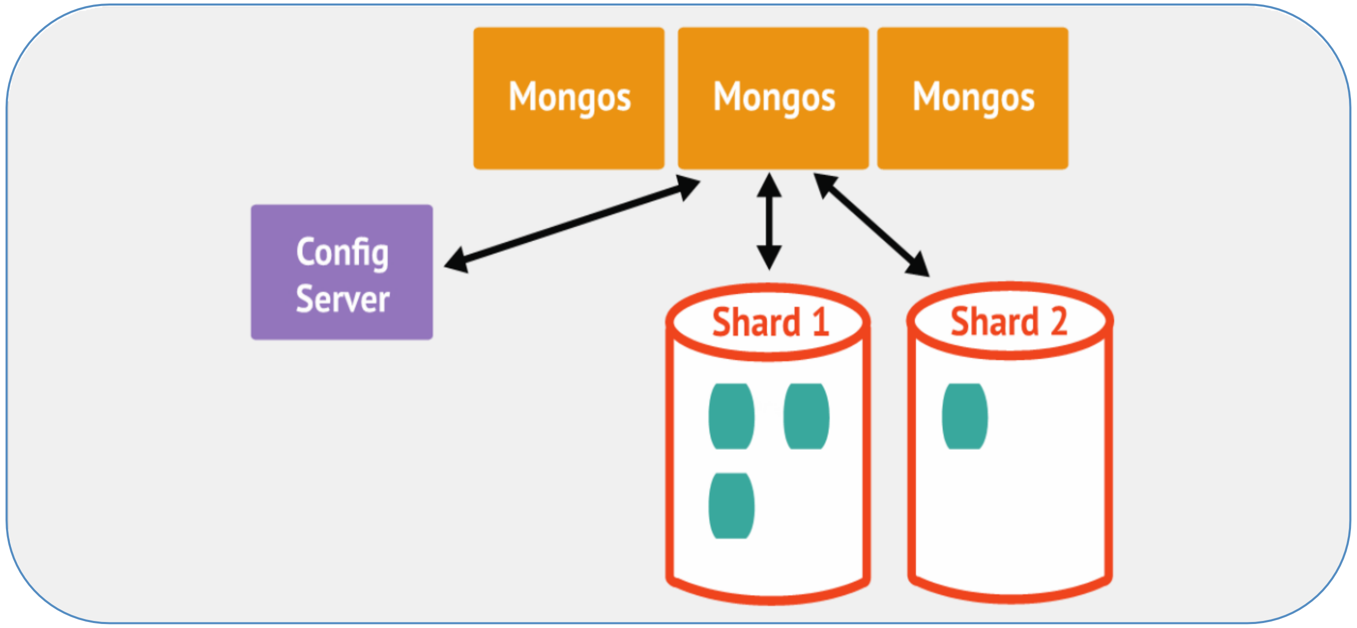


Chunk Splitting



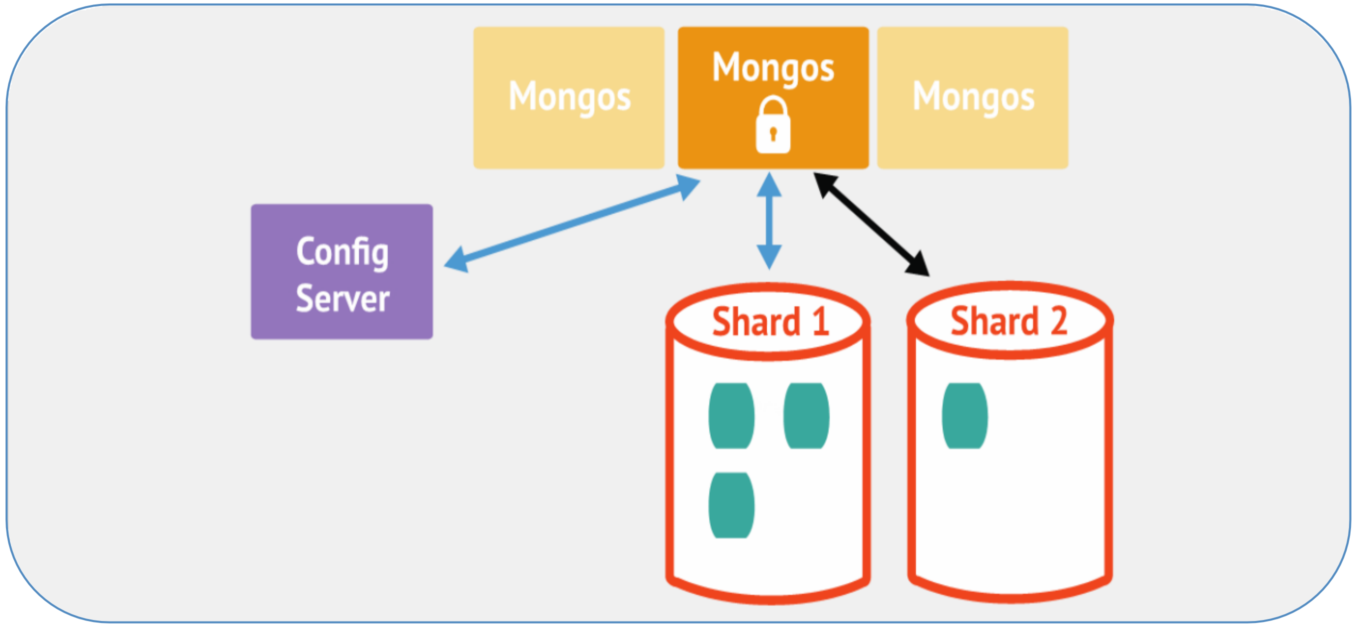
- A chunk is split once it exceeds the maximum size
- There is no split point if all documents have the same shard key.
- Chunk split is a logical operation (no data is moved)

Balancing



- Balancer is running on mongos.
- Once the difference in chunks between the most dense shard and the least dense shard is above the migration threshold, a balancing round starts.

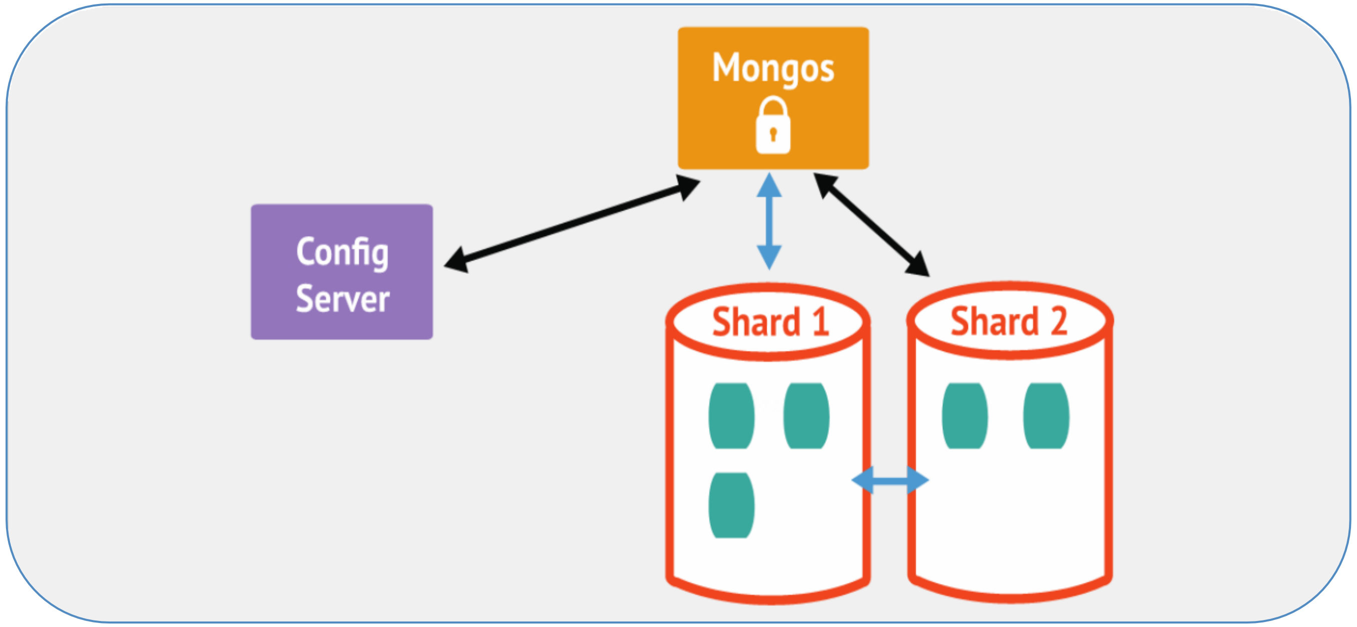
Acquiring the Balancer Lock



- The balancer on mongos takes out a “balancer lock”
- To see the status of these locks:
  - `use config`
  - `db.locks.find({ _id: “balancer” })`

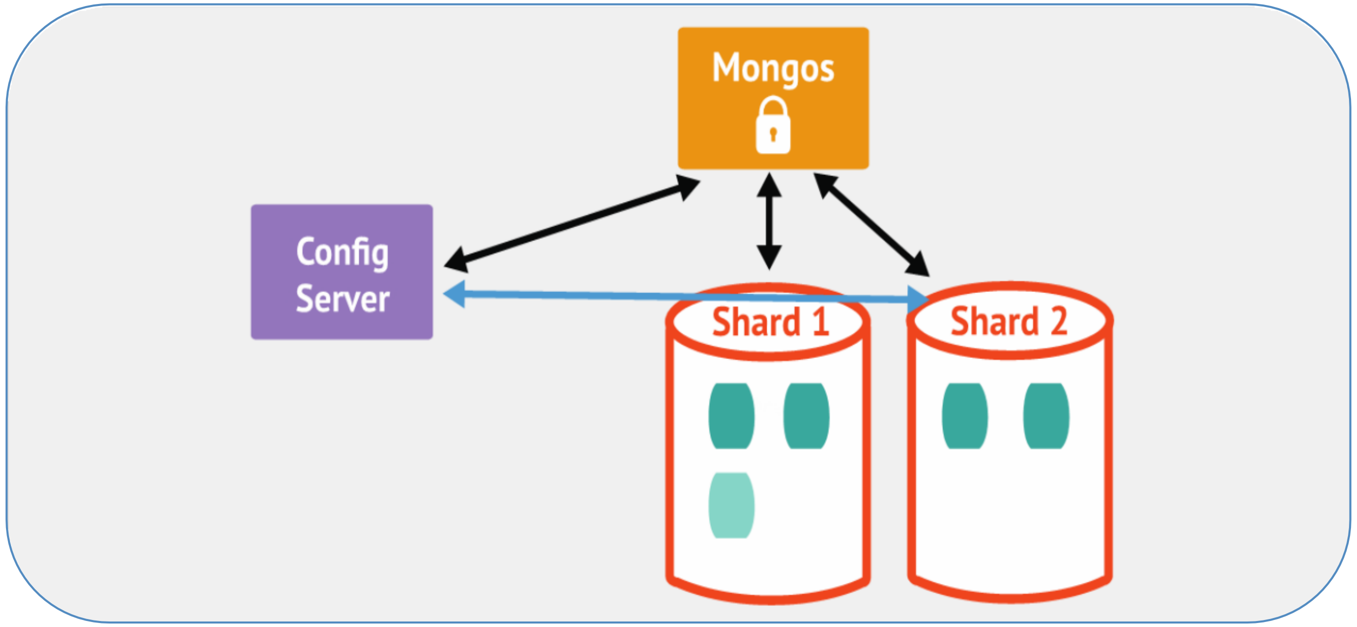
37

Moving the Chunk



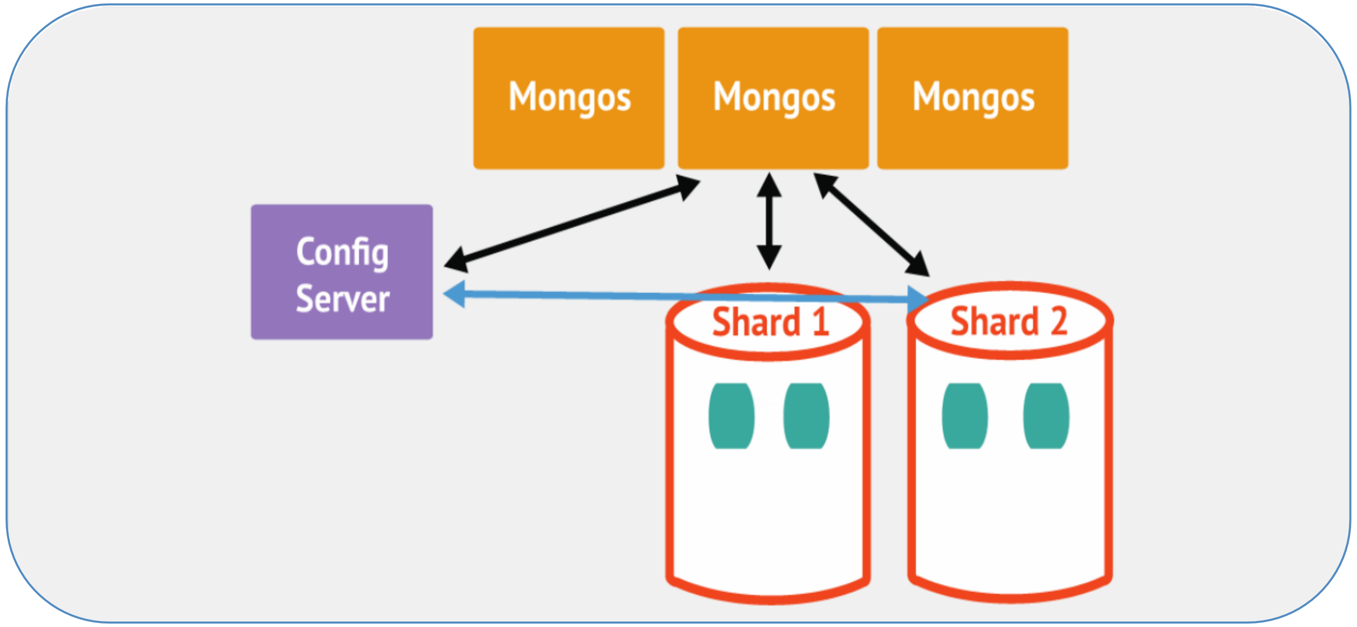
- The mongos sends a **`moveChunk`** command to source shard.
- The source shard then notifies destination shard.
- Destination shard starts pulling documents from source shard.

Committing Migration



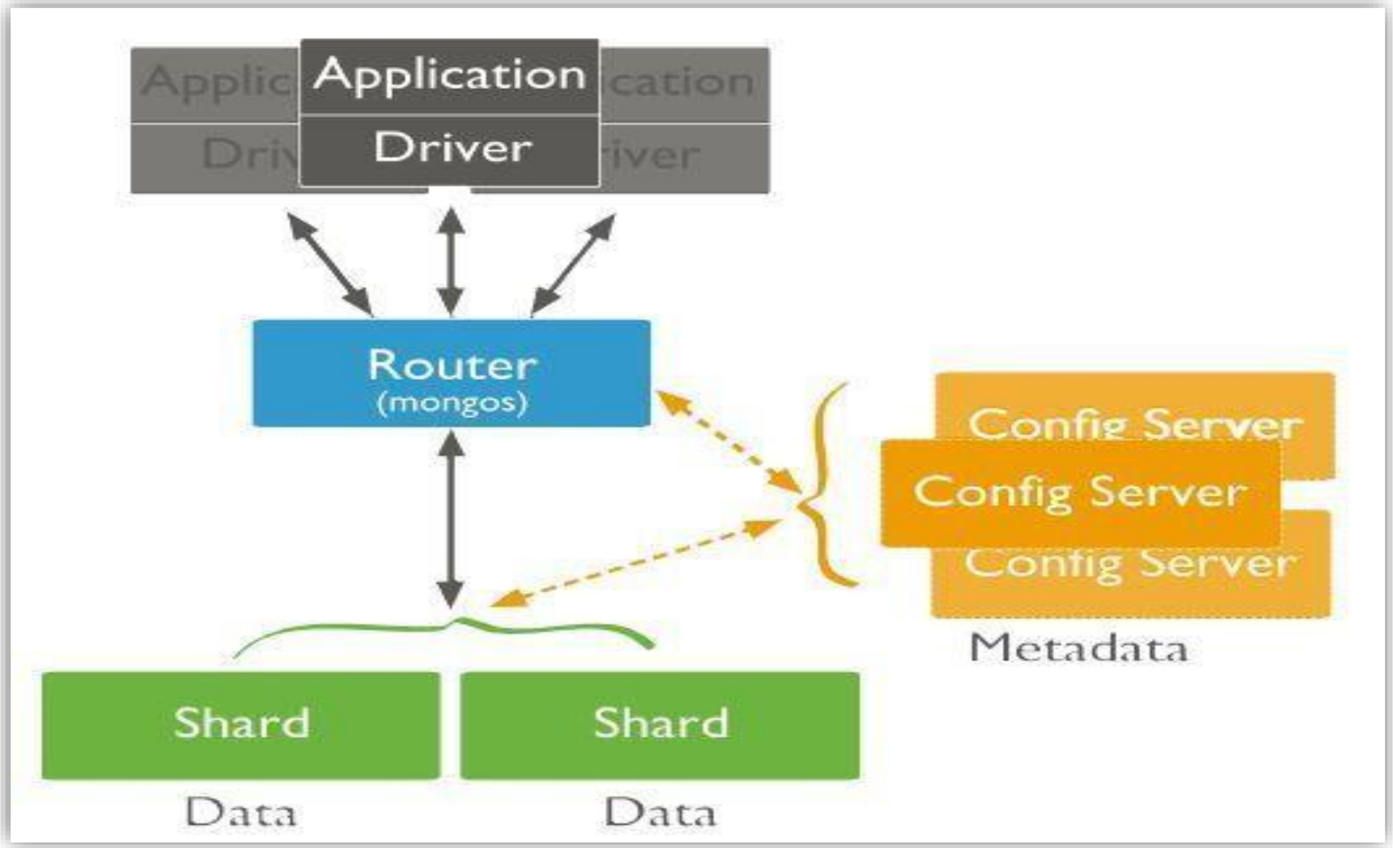
- When complete, destination shard updates config server.
  - Provides new locations of the chunks.

Cleanup



- Source shard deletes moved data.
  - Must wait for open cursors to either close or time out.
- The **`mongos`** releases the balancer lock after old chunks are deleted.

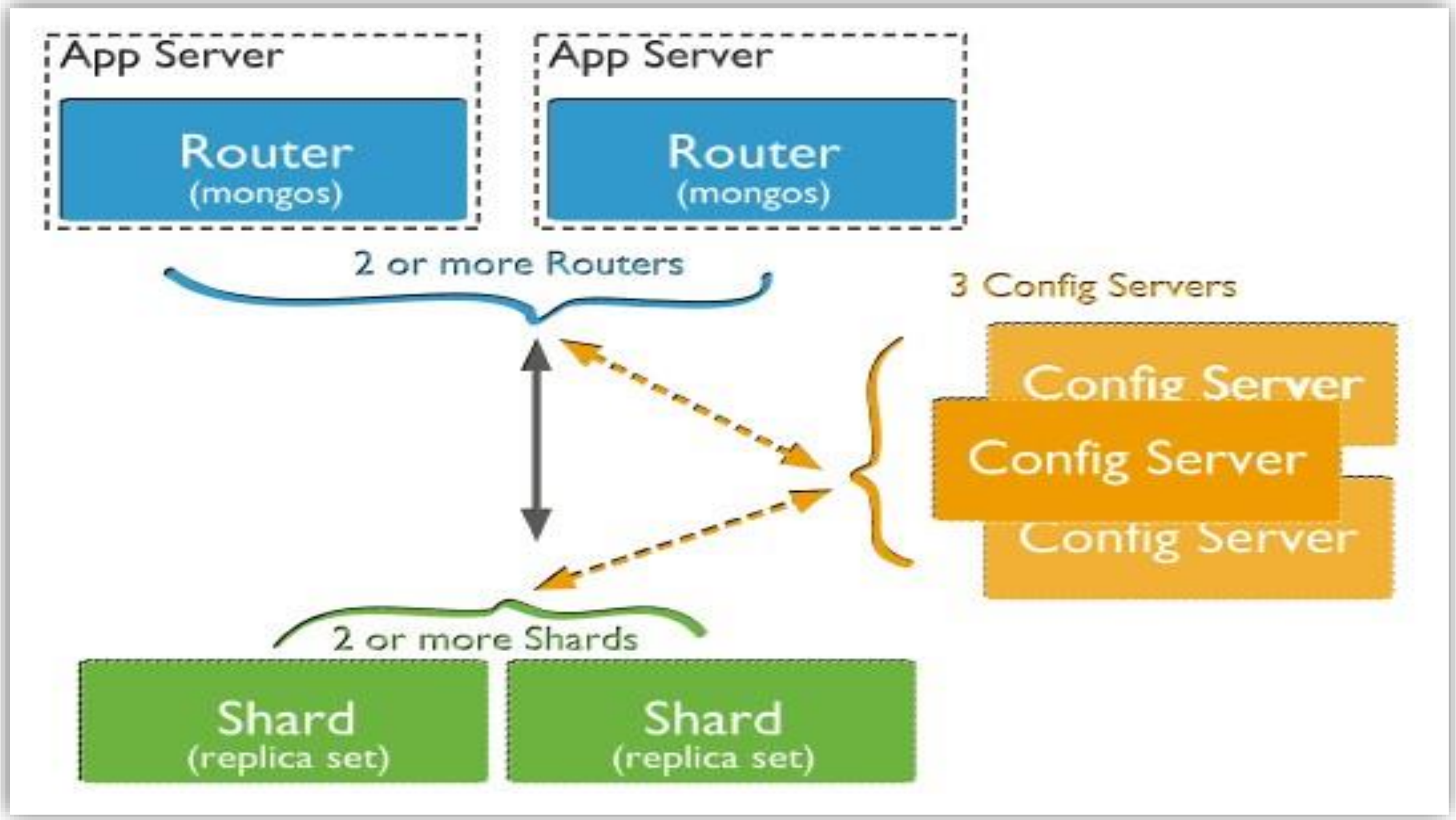
Sharding Mechanics



Sharding in MongoDB

Sharded cluster has the following components:		
Shards store the data. To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set	**Query Routers**, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. The query router processes and targets operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Most sharded clusters have many query routers.	**Config servers** store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. Production sharded clusters have exactly 3 config servers.

Sharding in MongoDB (contd.)



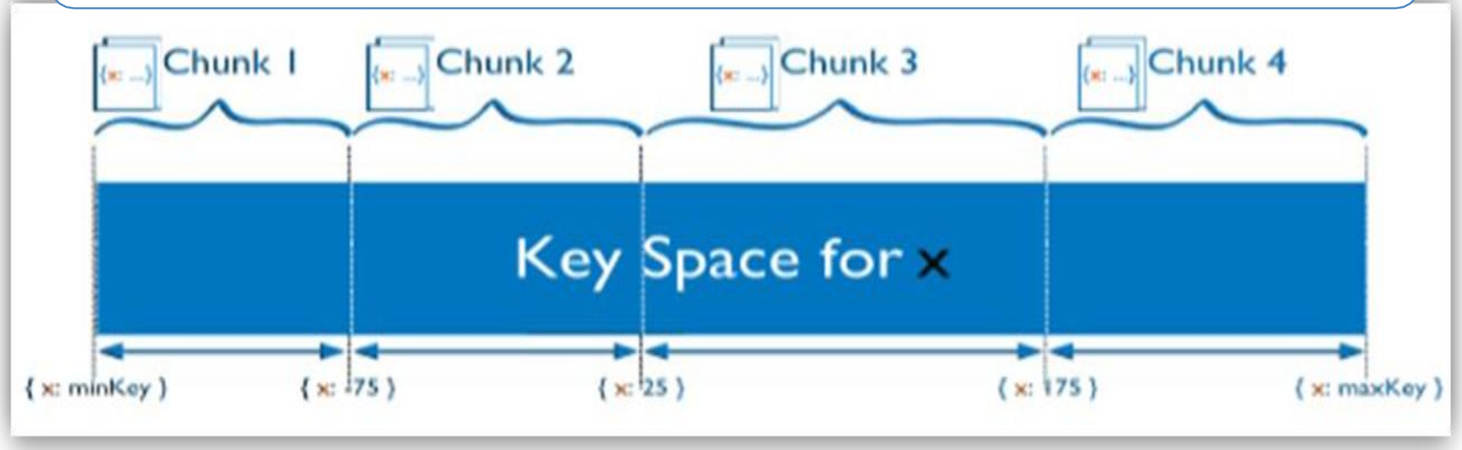
Data Partitioning

MongoDB distributes data, or shards, at the collection level. Sharding partitions a collection's data by the shard key.A shard key is either an indexed field or an indexed compound field that exists in every document in the collection.

MongoDB divides the shard key values into chunks and distributes the chunks evenly across the shards. To divide the shard key values into chunks, MongoDB uses either range based partitioning or hash based partitioning.

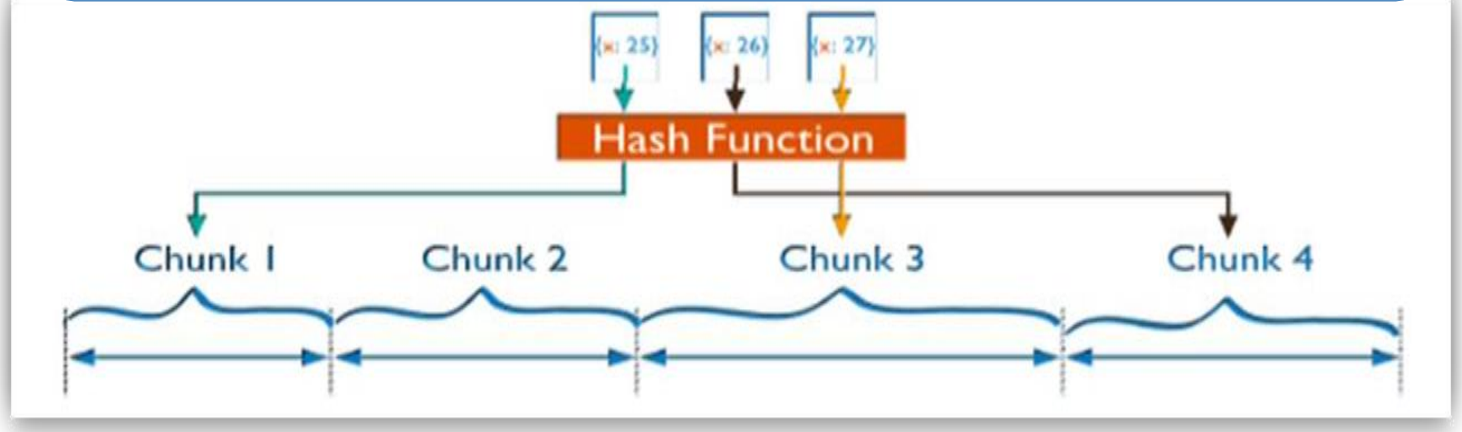
Data Partitioning (contd.)

Range Based Sharding: MongoDB divides the data set into ranges determined by the shard key values to provide range based partitioning.



Data Partitioning (contd.)

Hash Based Sharding: MongoDB computes a hash of a field's value, and then uses these hashes to create chunks.



GridFS

MongoDB provides GridFS specification for storing and retrieving files such as images, audio files, video files that exceed the BSON-document size limit of 16MB.

GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.

By default GridFS limits chunk size to 255k.

GridFS uses two collections to store files:

- fs.files
- fs.chunks

When we query a GridFS store for a file, the driver or client will reassemble the chunks as needed.

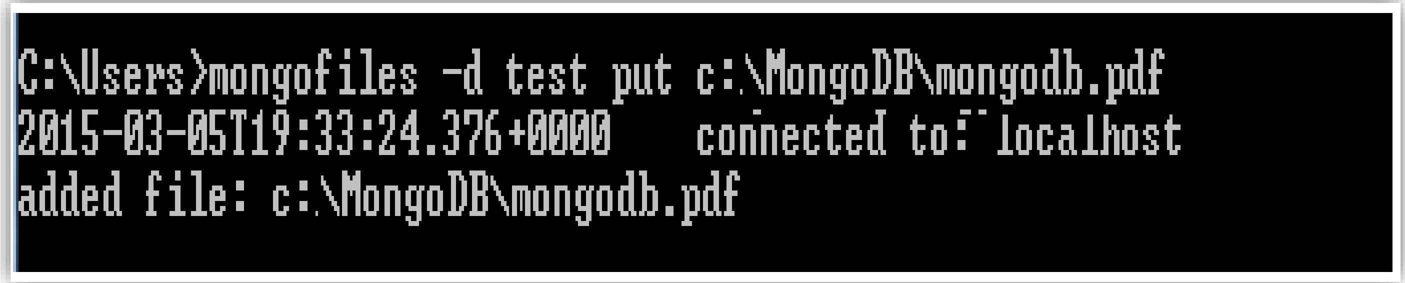
We can also access information from arbitrary sections of files, which allows you to “skip” into the middle of a video or audio file.

GridFSCan store any types of file.

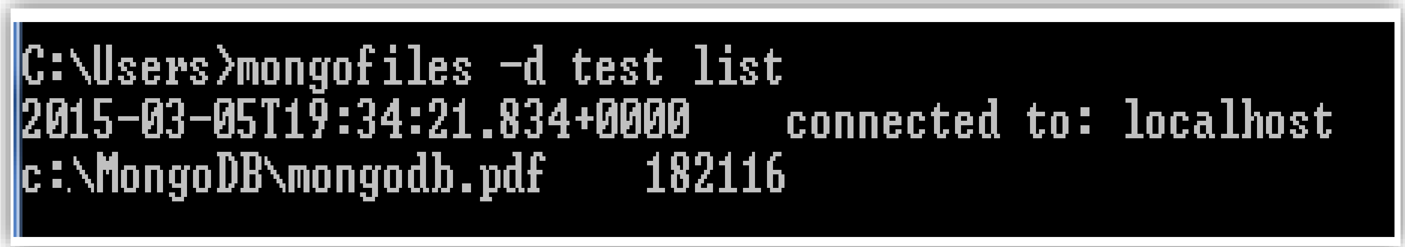
Adding Files in GridFS

Run mongofiles exe from command prompt as shown below:

mongofiles.exe -d <database> -c <collection> -prefix <string> put <filename>
mongofiles.exe -d <database> get <filename>
mongofiles -d records list [to list lists in GridFS]

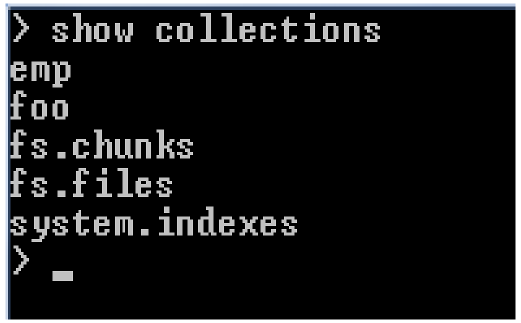


```
C:\Users>mongofiles -d test put c:\MongoDB\mongodb.pdf
2015-03-05T19:33:24.376+0000    connected to: localhost
added file: c:\MongoDB\mongodb.pdf
```

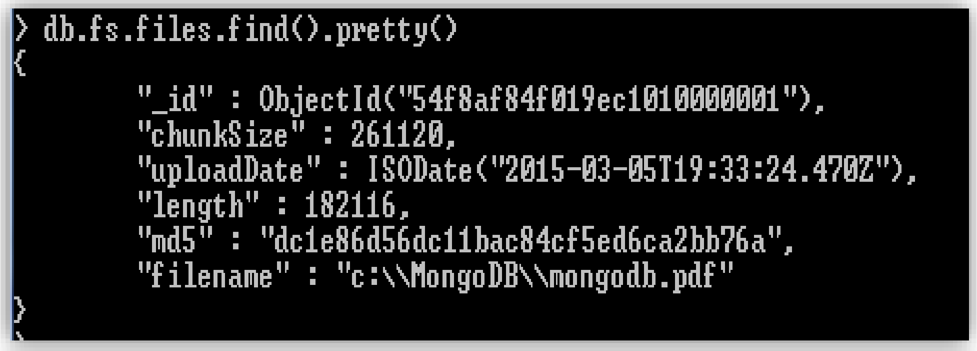


```
C:\Users>mongofiles -d test list
2015-03-05T19:34:21.834+0000    connected to: localhost
c:\MongoDB\mongodb.pdf    182116
```

GridFS Collections



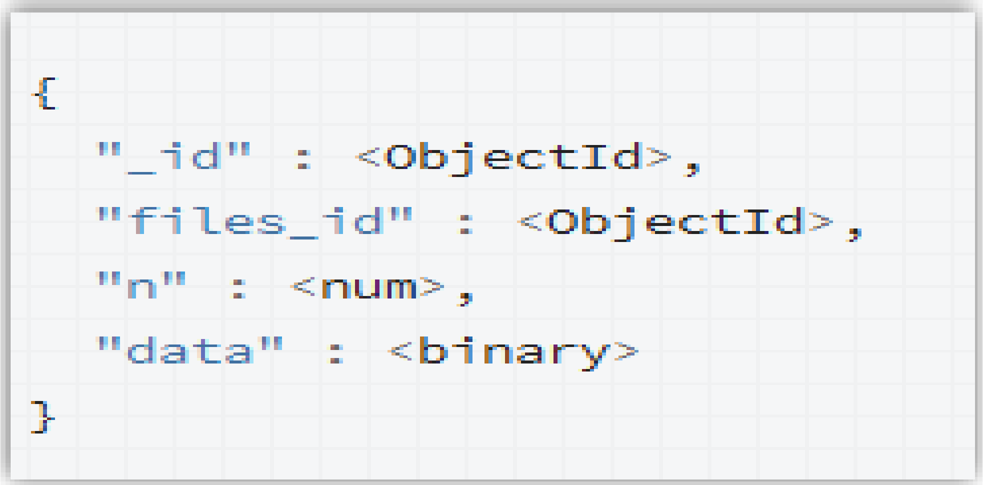
```
> show collections
emp
foo
fs.chunks
fs.files
system.indexes
> _
```



```
> db.fs.files.find().pretty()
{
        "_id" : ObjectId("54f8af84f019ec1010000001"),
        "chunkSize" : 261120,
        "uploadDate" : ISODate("2015-03-05T19:33:24.470Z"),
        "length" : 182116,
        "md5" : "dc1e86d56dc11bac84cf5ed6ca2bb76a",
        "filename" : "c:\\MongoDB\\mongodb.pdf"
}
```

Note: fs.chunks store binary data and querying this collection will return binary data of file.

Structure is shown right side.



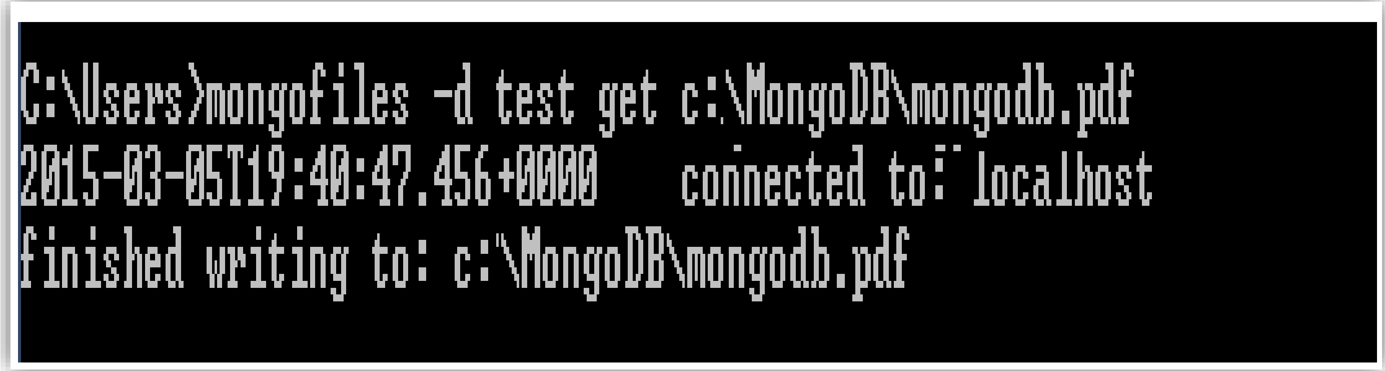
```
{
  "_id" : <ObjectId>,
  "files_id" : <ObjectId>,
  "n" : <num>,
  "data" : <binary>
}
```

Retrieving Files from GridFS

Run **mongofiles exe** from command prompt as shown below:

- mongofiles.exe -d <database> get <filename>

Note: MongoDB copy file to same location from where it has copied. To copy to different location change the file location by renaming filename in fs.files collection.



```
C:\Users>mongofiles -d test get c:\MongoDB\mongodb.pdf
2015-03-05T19:40:47.456+0000    connected to: localhost
finished writing to: c:\MongoDB\mongodb.pdf
```

Summary

Understand about Replication

Purpose of Replication

Understand Replica Set

Sharding

Sharding Mechanics

GridFS