Unit-4 MongoDB: Sharding

What is Sharding?

- Sharding is the process of storing data records across multiple machines.

Why sharding is required?

- To meet the demands of data growth.
- As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.
- Sharding solves the above problem with horizontal scaling.
- With sharding, more machines are added to support data growth and the demands of read and write operations.
- Database systems applications with large data sets and high throughput requirements can challenge the capacity of a single server due to:
  - CPU capacity of the server
  - Storage capacity of a single machine
  - RAM
  - I/O capacity of disk drives
- There are two solution to solve large data sets and high throughput requirements
  - Vertical scaling
  - Horizontal scaling - Sharding

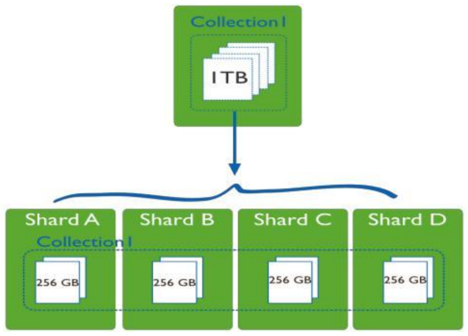
Vertical scaling

- Vertical scaling adds more CPU and storage resources to increase capacity.
- Scaling by adding capacity has limitations:
  - High performance systems with large numbers of CPUs and large amount of RAM are disproportionately more expensive than smaller systems.
- Additionally, cloud-based providers may only allow users to provision smaller instances.
- As a result, there is a practical maximum capability for vertical scaling.

Sharding

- Sharding, or horizontal scaling divides the data set and distributes the data over multiple servers, or shards.
- Each shard is an independent database, and collectively, the shards make up a single logical database.

- Sharding addresses the challenge of scaling to support high throughput and large data sets:
  - Sharding reduces the number of operations each shard handles.
  - Each shard processes fewer operations as the cluster grows.
  - As a result, a cluster can increase capacity and throughput horizontally.
- For example, to insert data, the application only needs to access the shard responsible for that record.
- For example, if a database has a 1 terabyte data set, and there are 4 shards, then each shard might hold only 256 GB of data.
- If there are 40 shards, then each shard might hold only 25 GB of data.



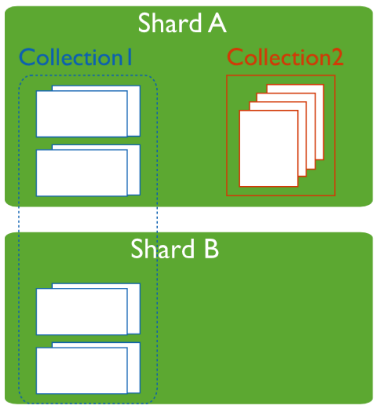
- Sharded cluster has the following components:
  - Shards
  - Query routers or mongos Instances
  - Config servers

Shard

- Shards store the data.
- Each shard is either a single mongod instance or a replica set
- To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set.
- A shard is a replica set or a single mongod that contains a subset of the data for the sharded cluster.
- MongoDB shards data on a per collection basis. You must access all data in a sharded cluster via the mongos instances.
- Two or More Replica Sets may work as Shards in deployment architecture
- **If you connect directly to a shard, you will see only its fraction of the cluster's data.**

Primary Shard

- Every database has a “primary” shard that holds all the un-sharded collections in that database.
- sh.status() to see an overview of the cluster



Query Routers / Mongos

- Query Routers, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards.
- A client sends requests to mongos, which then routes the operations to the shards and returns the results to the clients.
- A sharded cluster can contain more than one mongos to divide the client request load, and most sharded clusters have more than one mongos for this reason.
- Applications do not access the shards directly.

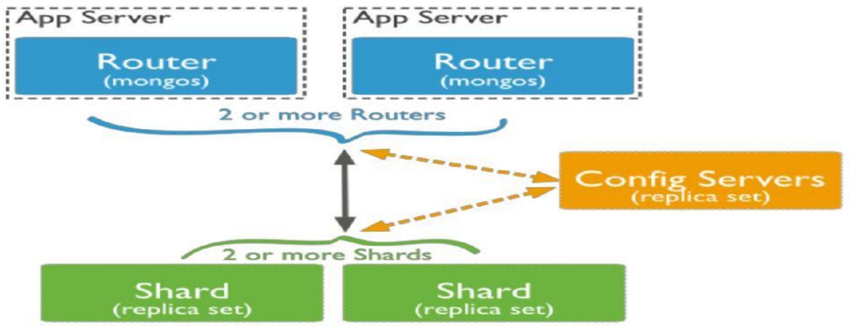
One or More Query Routers (mongos)

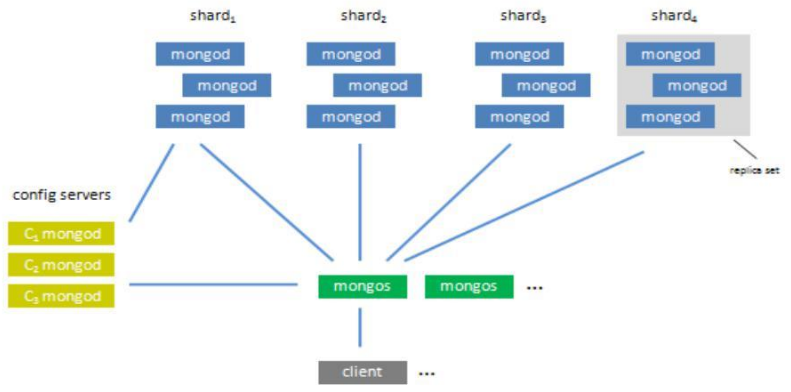
- The mongos instances are the routers for the cluster.
- Deployments have one mongos instance on each application server.
- Each client must interact with only one mongos instance.

Configuration Servers

- 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.
- Config servers store the metadata for a sharded cluster.
- If the config servers become inaccessible, the cluster is not accessible.
- If the data cannot be recovered on a config server, the cluster will be inoperable.
- The following restrictions apply to a replica set configuration when used for config servers:
  - Must have zero arbiters.
  - Must have no delayed members.
  - Must build indexes

- Config Servers for sharded clusters can be deployed as a replica set.
- The replica set config servers must run the WiredTiger storage engine.
- A single sharded cluster must have exclusive use of its config servers.
- If there are multiple sharded clusters, each cluster must have its own replica set config servers.





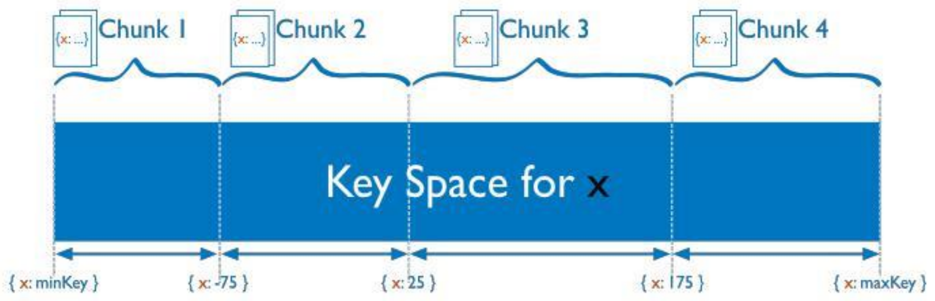
Shard Key

- To shard a collection, there is a need to select a shard key.
- The shard key consists of an immutable field or fields that exist in every document in the target collection.
- Choose the shard key when sharding a collection.
- The choice of shard key cannot be changed after sharding.
- A sharded collection can have only one 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.
- For a sharded collection, only the _id field index and the index on the shard key or a compound index where the shard key is a prefix can be unique:
  - A collection that has unique indexes on other fields cannot be sharded.
  - Unique indexes on other fields for a sharded collection cannot be created.

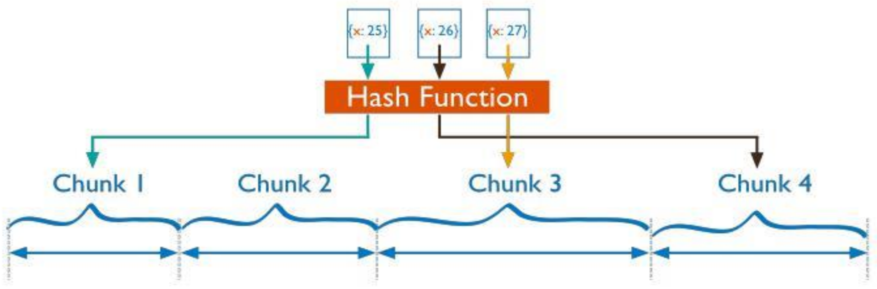
Range Based Sharding

- For range-based sharding, MongoDB divides the data set into ranges determined by the shard key values to provide range-based partitioning.
- Given a range based partitioning system, documents with "close" shard key values are likely to be in the same chunk, and therefore on the same shard.



Hash Based Sharding

- For hash based partitioning, MongoDB computes a hash of a field's value, and then uses these hashes to create chunks.
- With hash based partitioning, two documents with "close" shard key values are unlikely to be part of the same chunk.
- This ensures a more random distribution of a collection in the cluster.



Performance Distinctions between Range and Hash Based Partitioning

- Range based partitioning supports more efficient range queries.
- Given a range query on the shard key, the query router can easily determine which chunks overlap that range and route the query to only those shards that contain these chunks.
- However, range based partitioning can result in an uneven distribution of data, which may negate some of the benefits of sharding.
- For example, if the shard key is a linearly increasing field, such as time, then all requests for a given time range will map to the same chunk, and thus the same shard.
- In this situation, a small set of shards may receive the majority of requests and the system would not scale very well.
- Hash based partitioning, by contrast, ensures an even distribution of data at the expense of efficient range queries.
- Hashed key values results in random distribution of data across chunks and therefore shards.
- But random distribution makes it more likely that a range query on the shard key will not be able to target a few shards but would more likely query every shard in order to return a result.

Customized Data Distribution with Tag Aware Sharding

- MongoDB allows administrators to direct the balancing policy using tag aware sharding.
- Administrators create and associate tags with ranges of the shard key, and then assign those tags to the shards.
- Then, the balancer migrates tagged data to the appropriate shards and ensures that the cluster always enforces the distribution of data that the tags describe.
- Tags are the primary mechanism to control the behavior of the balancer and the distribution of chunks in a cluster.
- Most commonly, tag aware sharding serves to improve the locality of data for sharded clusters that span multiple data centers.

Maintaining a Balanced Data Distribution

- The addition of new data or the addition of new servers can result in data distribution imbalances within the cluster, such as a particular shard contains significantly more chunks than another shard or a size of a chunk is significantly greater than other chunk sizes.
- MongoDB ensures a balanced cluster using two background process:
  - splitting

₪ balancer

Splitting

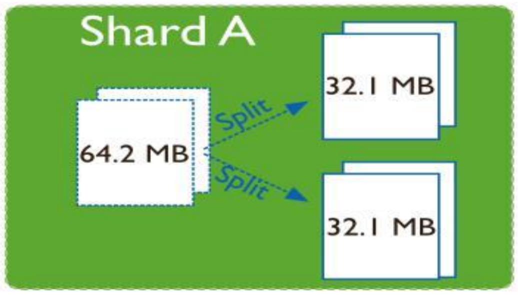
ϖ Splitting is a background process that keeps chunks from growing too large.

ϖ When a chunk grows beyond a specified chunk size, MongoDB splits the chunk in half.

ϖ Inserts and updates triggers splits.

ϖ Splits are an efficient meta-data change.

ϖ To create splits, MongoDB does not migrate any data or affect the shards.



Balancing

ϖ The balancer is a background process that manages chunk migrations.

ϖ The balancer can run from any of the mongos instances in a cluster.

ϖ When the distribution of a sharded collection in a cluster is uneven, the balancer process migrates chunks from the shard that has the largest number of chunks to the shard with the least number of chunks until the collection balances.

ϖ For example: if collection users has 100 chunks on shard 1 and 50 chunks on shard 2, the balancer will migrate chunks from shard 1 to shard 2 until the collection achieves balance.

ϖ The shards manage chunk migrations as a background operation between an origin shard and a destination shard.

ϖ During a chunk migration, the destination shard is sent all the current documents in the chunk from the origin shard.

ϖ Next, the destination shard captures and applies all changes made to the data during the migration process.

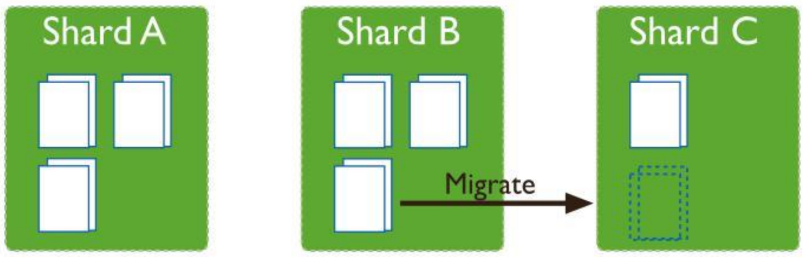
ϖ Finally, the metadata regarding the location of the chunk on config server is updated.

ϖ If there's an error during the migration, the balancer aborts the process leaving the chunk unchanged on the origin shard.

ϖ MongoDB removes the chunk's data from the origin shard after the migration completes successfully.

ϖ MongoDB partitions sharded data into chunks.

- MongoDB migrates chunks across the shards in the sharded cluster using the sharded cluster balancer.
- The balancer attempts to achieve an even balance of chunks across all shards in the cluster.



Adding and Removing Shards from the Cluster

- Adding a shard to a cluster creates an imbalance since the new shard has no chunks.
- While MongoDB begins migrating data to the new shard immediately, it can take some time before the cluster balances.
- When removing a shard, the balancer migrates all chunks from a shard to other shards.
- After migrating all data and updating the meta data, the shard can be safely removed.

Read and Write Operations on Config Servers

- Config servers store the cluster's metadata in the config database.
- The mongos instances cache this data and use it to route reads and writes to shards.
- MongoDB only writes data to the config servers when the metadata changes, such as
  - after a chunk migration, or
  - after a chunk split.
  - When writing to the replica set config servers, MongoDB uses a write concern of "majority".
- MongoDB reads data from the config server in the following cases:
  - A new mongos starts for the first time, or an existing mongos restarts.
  - After change in the cluster metadata, such as after a chunk migration.
  - When reading from the replica set config servers, MongoDB uses a Read Concern level of "majority".

Config Servers Availability

- If the config server replica set loses its primary and cannot elect a primary, the cluster's metadata becomes read only.

ϖ Data can be still read and written from the shards, but no chunk migration or chunk splits will occur until the replica set can elect a primary.

ϖ If all config databases become unavailable, the cluster can become inoperable.

Sharded Cluster Requirements

ϖ Sharded clusters have significant infrastructure requirements and increases the overall complexity of a deployment.

ϖ Deploy sharded clusters when indicated by application and operational requirements

  ₪ data set approaches or exceeds the storage capacity of a single MongoDB instance.

  ₪ the size of system's active working set will soon exceed the capacity of your system's maximum RAM.

  ₪ a single MongoDB instance cannot meet the demands of write operations

ϖ If these attributes are not present in system, sharding will only add complexity to system without adding much benefit.

Data Quantity Requirements

ϖ The cluster should manage a large quantity of data if sharding is to have an effect.

ϖ The default chunk size is 64 megabytes.

ϖ The balancer will not begin moving data across shards until the imbalance of chunks among the shards exceeds the migration threshold.

ϖ In practical terms, unless cluster has many hundreds of megabytes of data, your data will remain on a single shard.

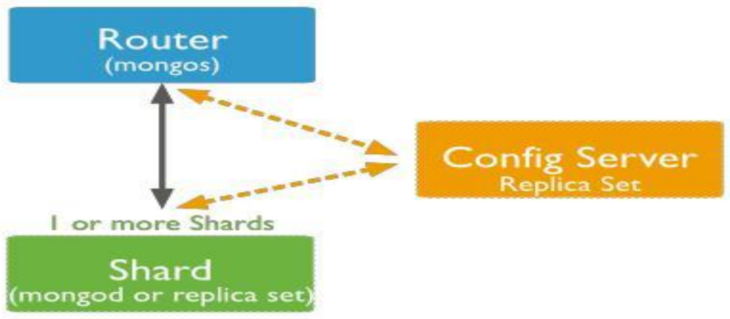
Sharded Cluster Test Architecture

ϖ A replica set config server with one member

ϖ At least one shard.

ϖ Shards are either replica sets or a standalone mongod instances

ϖ One mongos instance



Choosing a Shard Key

- For many collections there may be no single, naturally occurring key that possesses all the qualities of a good shard key.
  1. Compute a more ideal shard key in the application layer, and store this in all of the documents, potentially in the _id field.
  2. Use a compound shard key that uses two or three values from all documents that provide the right mix of cardinality with scalable write operations and query isolation.
  3. Determine that the impact of using a less than ideal shard key is insignificant in use case, given:
     1. limited write volume
     2. expected data size
     3. application query patterns
  4. Use a hashed shard key
     1. Choose a field that has high cardinality and create a hashed index on that field.
     2. MongoDB uses these hashed index values as shard key values, which ensures an even distribution of documents across the shards.

Considerations for Selecting Shard Key

- Create a Shard Key that is Easily Divisible
- Create a Shard Key that has High Degree of Randomness
- Create a Shard Key that Targets a Single Shard

MongoDB Limits and Thresholds

- Replica sets can have up to 50 members.
- Number of Voting Members of a Replica SetReplica sets can have up to 7 voting members. For replica sets with more than 7 total members,
- A shard key cannot exceed 512 bytes.
- A shard key index can be an ascending index on the shard key, a compound index that start with the shard key and specify ascending order for the shard key, or a hashed index.