

Sharding



- Sharding is the process of storing data records across multiple machines and is MongoDB's approach to meeting 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 problem with horizontal scaling.
- With sharding, you add more machines to support data growth and the demands of read and write operations.

Purpose of Sharding

- Database systems with large data sets and high throughput applications can challenge the capacity of a single server.
- High query rates can exhaust the CPU capacity of the server.
- Larger data sets exceed the storage capacity of a single machine.
- Finally, working set sizes larger than the system's RAM stress the I/O capacity of disk drives.
- To address these issues of scales, database systems have two basic approaches:
 - ▣ vertical scaling
 - ▣ sharding
- 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, or *horizontal scaling*, by contrast, 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.

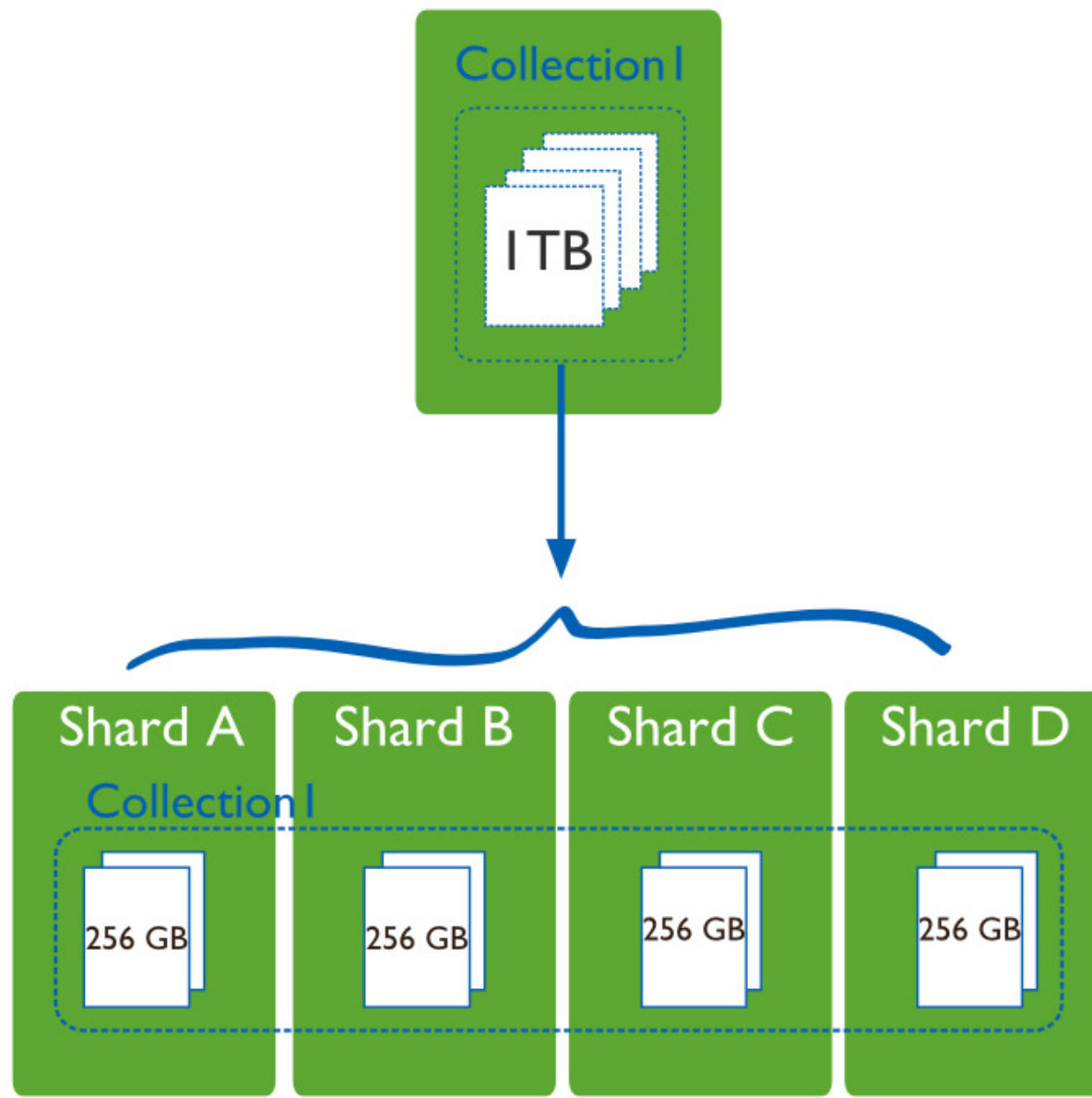


Diagram of a large collection with data distributed across 4 shards.

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, shared clusters can increase capacity and throughput horizontally.
 - ▣ For example, to insert data, the application only needs to access the shard responsible for that records.

- Sharding reduces the amount of data that each server needs to store. Each shard stores less data as the cluster grows.
 - ▣ For example, if a database has a 1 terabyte data set, and there are 4 shards, then each shard might hold only 256GB of data.
 - ▣ If there are 40 shards, then each shard might hold only 25GB of data.

Sharding in MongoDB

MongoDB supports sharding through the configuration of a sharded clusters.

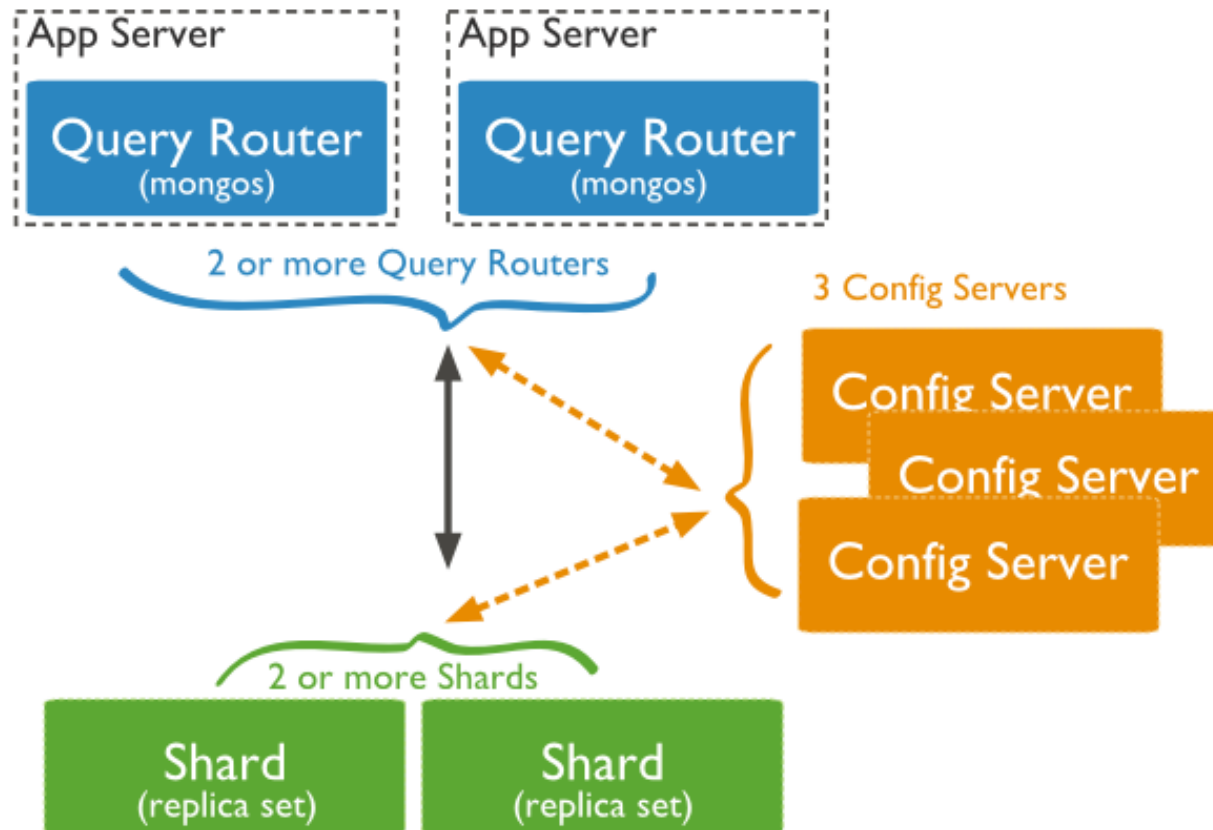



Diagram of a sample sharded cluster for production purposes. Contains exactly 3 config server, 2 or more **mongos** query router, and at least 2 shards. The shards are replica sets.

Sharded Cluster Components

- Sharded cluster has the following components:
 - ▣ Shards
 - ▣ query routers
 - ▣ config servers
- For development and testing purposes only, each shard can be a single mongod instead of a replica set. Do not deploy production clusters without 3 config servers.

Shards

- 
- ❑ Shards store the data.
 - ❑ To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set.

Query Routers

- 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 cluster have many query routers.

Config 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.
- ❑ Production sharded clusters have exactly 3 config servers.

Data Partitioning

- MongoDB distributes data, or shards, at the collection level.
- Sharding partitions a collection's data by the shard key.
- Shard Keys
 - ▣ To shard a collection, you need to select a 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
 - hash based partitioning

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**. Consider a numeric shard key: If you visualize a number line that goes from negative infinity to positive infinity, each value of the shard key falls at some point on that line. MongoDB partitions this line into smaller, non-overlapping ranges called **chunks** where a chunk is range of values from some minimum value to some maximum value.

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.

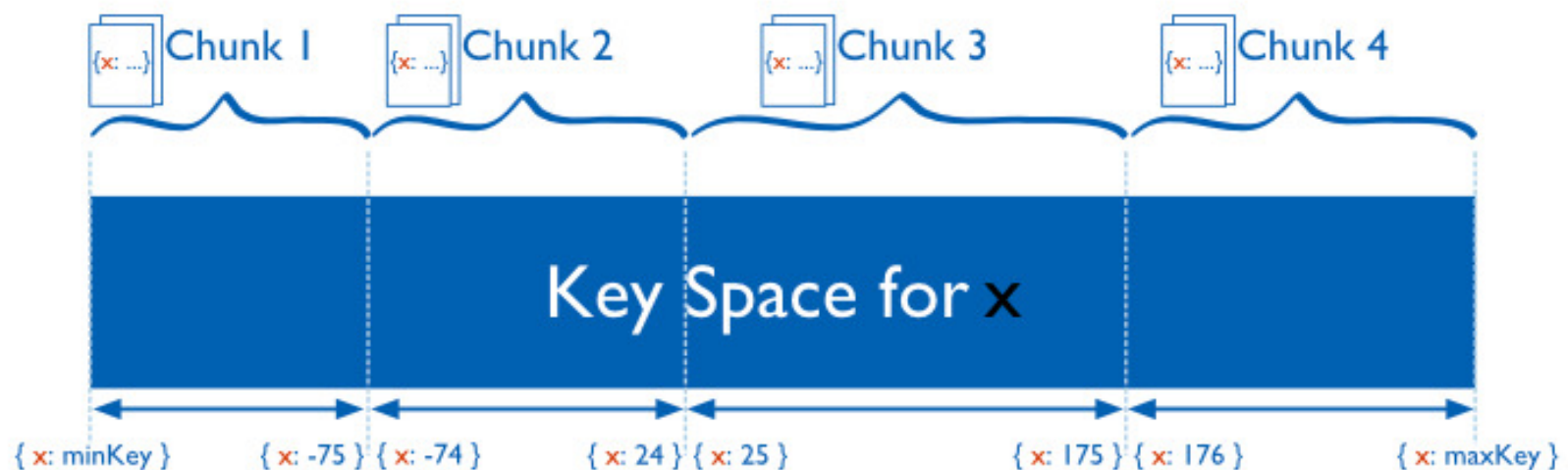


Diagram of the shard key value space segmented into smaller ranges or chunks.

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.

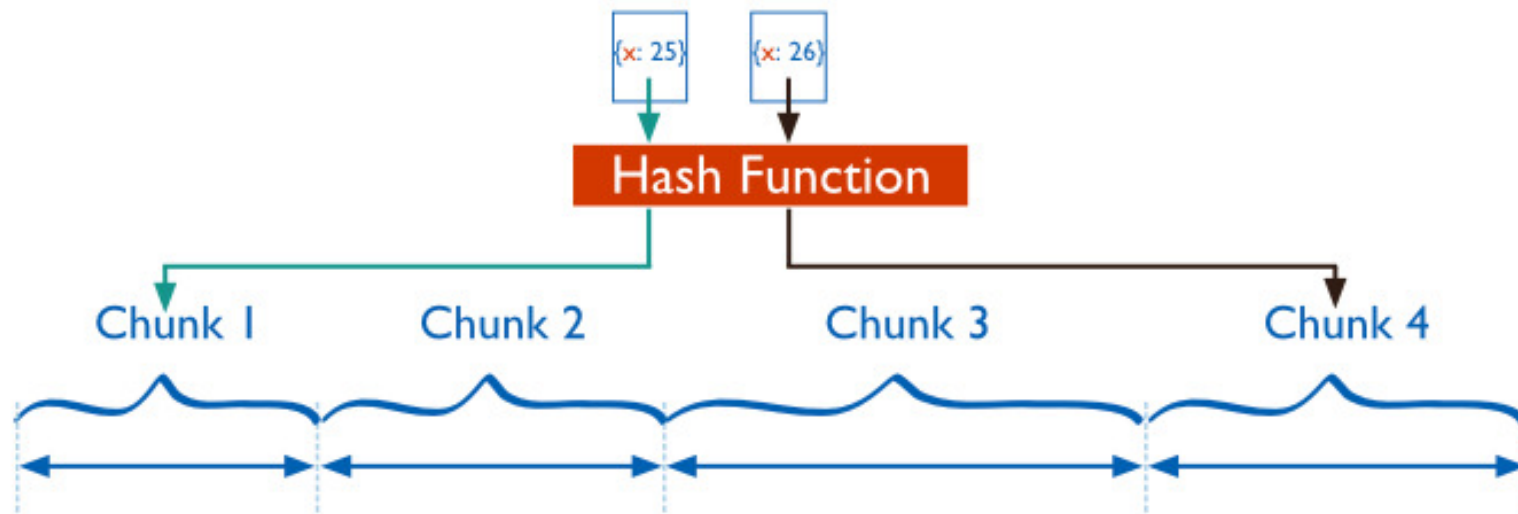


Diagram of the hashed based segmentation.

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

Splitting is a background process that keeps chunks from growing too large. When a chunk grows beyond a [specified chunk size](#), MongoDB splits the chunk in half. Inserts and updates triggers splits. Splits are an efficient meta-data change. To create splits, MongoDB does *not* migrate any data or affect the shards.

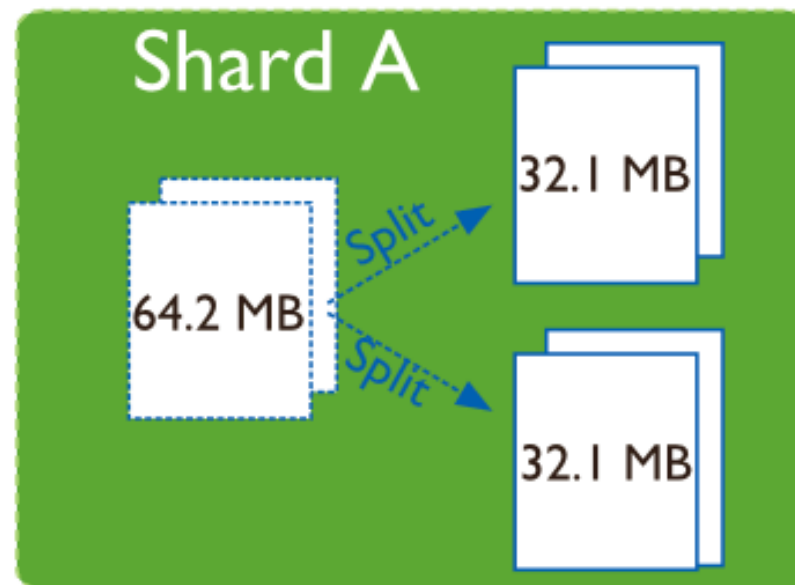


Diagram of a shard with a chunk that exceeds the default [chunk size](#) of 64 MB and triggers a split of the chunk into two chunks.

Balancing

The [balancer](#) is a background process that manages chunk migrations. The balancer runs in all of the query routers in a cluster.

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. 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.

The shards manage *chunk migrations* as a background operation. During migration, all requests for a chunk's data address the "origin" shard.

In a chunk migration, the *destination shard* receives all the documents in the chunk from the *origin shard*. Then, the destination shard captures and applies all changes made to the data during migration process. Finally, the destination shard updates the metadata regarding the location of the chunk on the *config server*.

If there's an error during the migration, the balancer aborts the process leaving the chunk on the origin shard. MongoDB removes the chunk's data from the origin shard **after** the migration completes successfully.

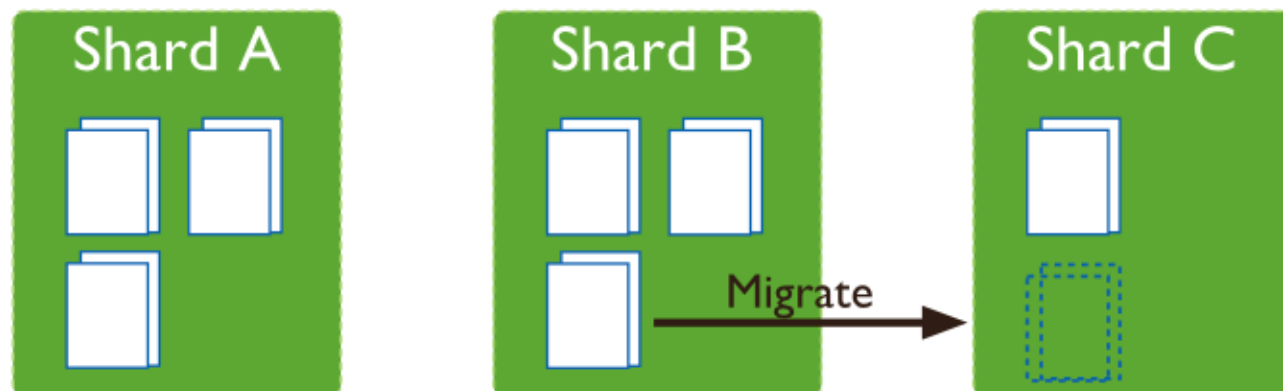


Diagram of a collection distributed across three shards. For this collection, the difference in the number of chunks between the shards reaches the [migration thresholds](#) (in this case, 2) and triggers migration.

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 to other shards.
- After migrating all data and updating the meta data, you can safely remove the shard.

Sharded Cluster Requirements

- While sharding is a powerful and compelling, sharded clusters have significant infrastructure requirements and increases the overall complexity of a deployment.
- As a result, only deploy sharded clusters when indicated by application and operational requirements.
- Sharding is the only solution for some classes of deployments. Use sharded clusters if:
 - ▣ your data set approaches or exceeds the storage capacity of a single MongoDB instance.
 - ▣ the size of your 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 your write operations, and all other approaches have not reduced contention.
- If these attributes are not present in your system, sharding will only add complexity to your system without adding much benefit.

Sharding



- ❑ It takes time and resources to deploy sharding.
- ❑ If your system has already reached or exceeded its capacity, it will be difficult to deploy sharding without impacting your application.
- ❑ As a result, if you think you will need to partition your database in the future, do not wait until your system is overcapacity to enable sharding.
- ❑ When designing your data model, take into consideration your sharding needs.

Production Cluster Architecture

In a production cluster, you must ensure that data is redundant and that your systems are highly available. To that end, a production cluster must have the following components:

- Three **config servers**. Each config servers must be on separate machines. A single **sharded cluster** must have exclusive use of its **config servers**. If you have multiple sharded clusters, you will need to have a group of config servers for each cluster.
- Two or more **replica sets**. These replica sets are the **shards**.
- One or more **mongos** instances. **mongos** is the routers for the cluster. Typically, deployments have one **mongos** instance on each application server. You may also may deploy a group of **mongos** instances and use a proxy/load balancer between the application and the **mongos**.

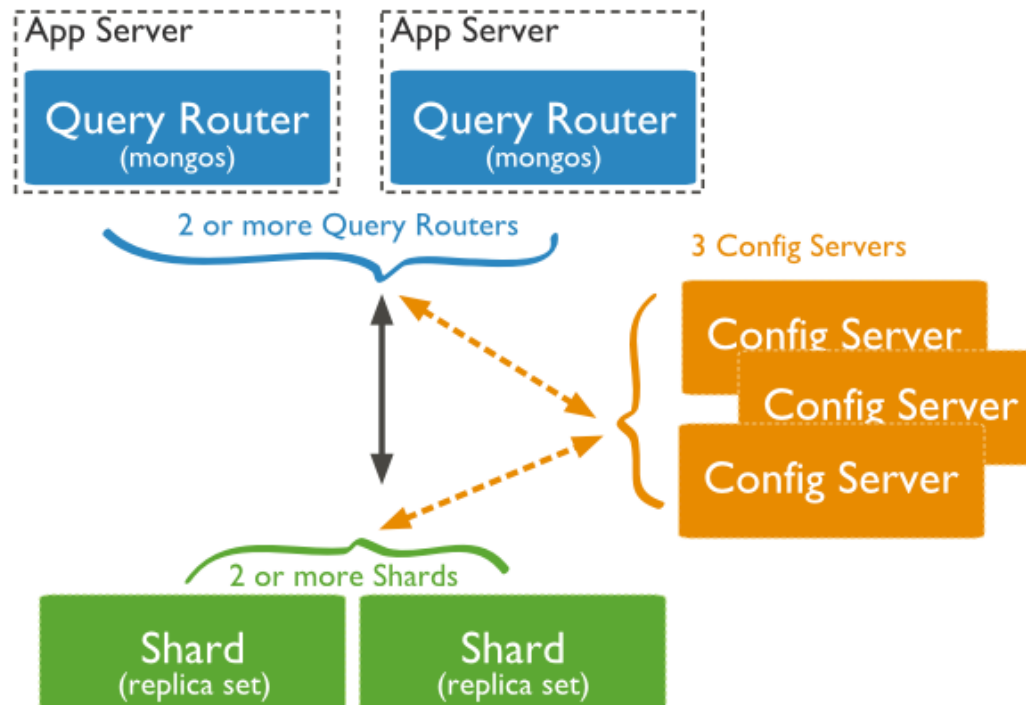


Diagram of a sample sharded cluster for production purposes. Contains exactly 3 config server, 2 or more **mongos** query router, and at least 2 shards. The shards are replica sets.

Sharded Cluster Test Architecture

Warning: Use the test cluster architecture for testing and development only.

For testing and development, you can deploy a minimal sharded clusters cluster. These **non-production** clusters have the following components:

- One [config server](#).
- At least one shard. Shards are either [replica sets](#) or a standalone [mongod](#) instances.
- One [mongos](#) instance.

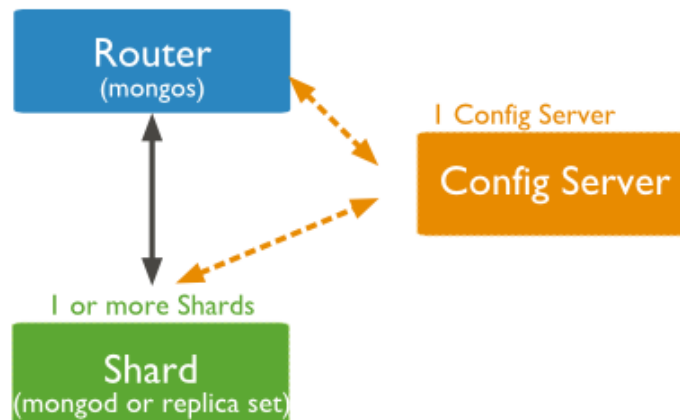


Diagram of a sample sharded cluster for testing/development purposes only. Contains only 1 config server, 1 [mongos](#) router, and at least 1 shard. The shard can be either a replica set or a standalone [mongod](#) instance.