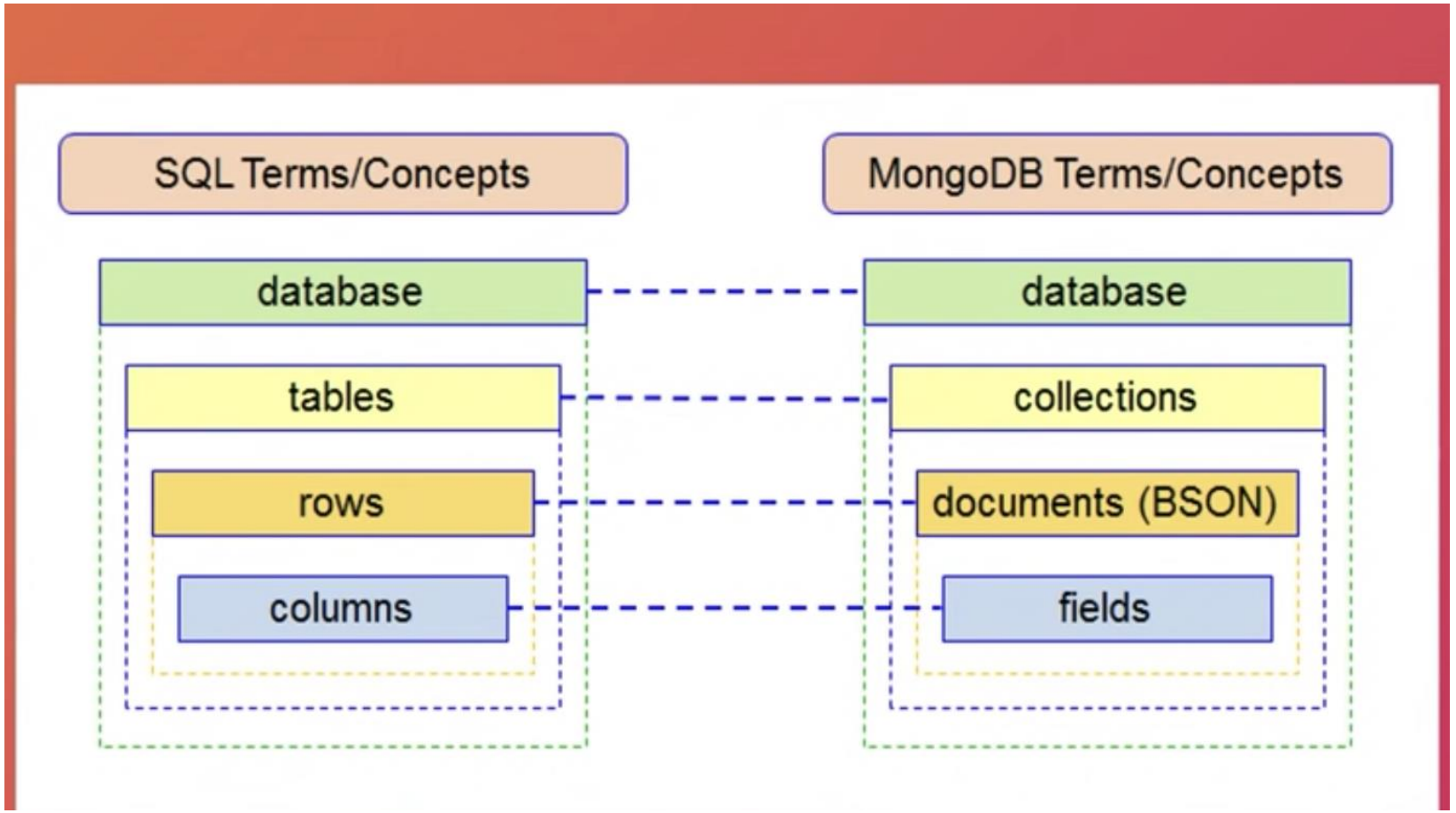
NoSQL Databases:MongoDB

the name Mongo comes from “humongous”—with performance and easy data access as core design goals.

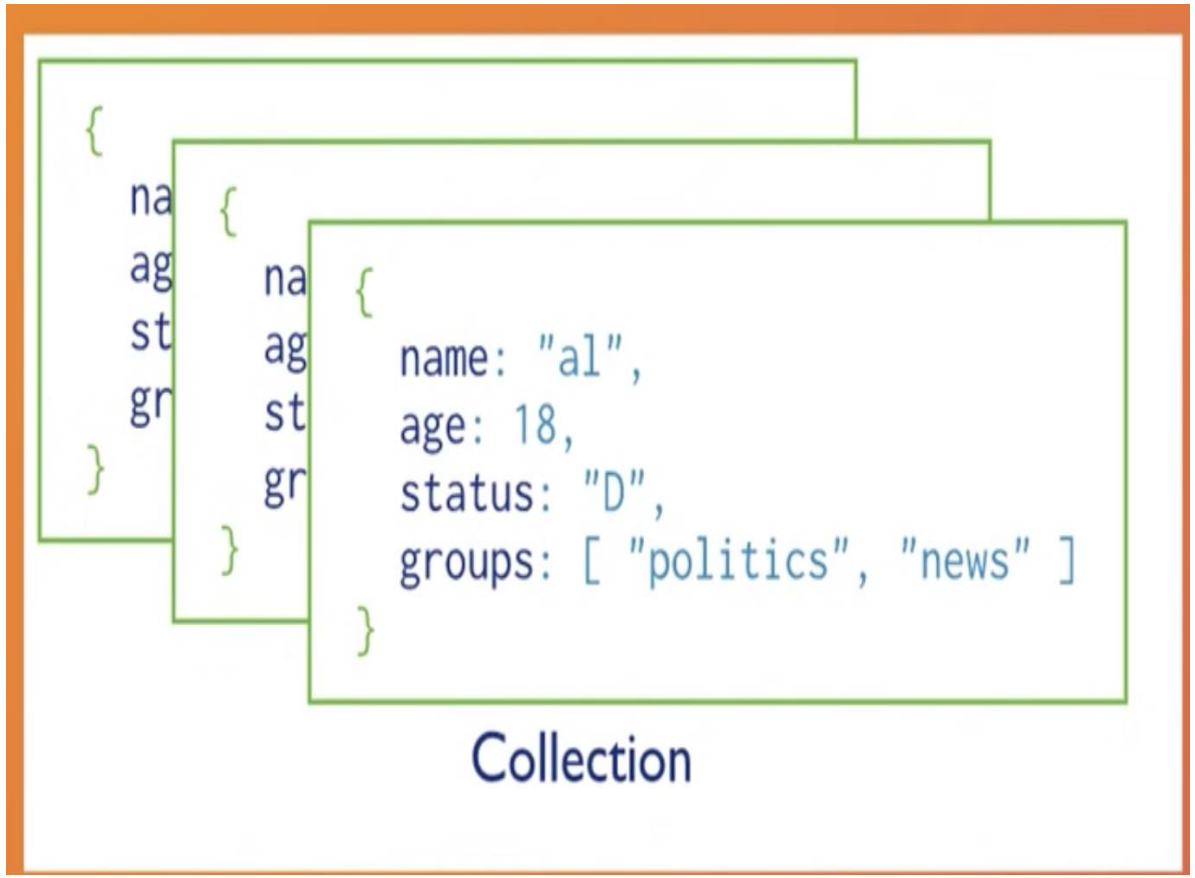
Comparison of Mongo with MySQL

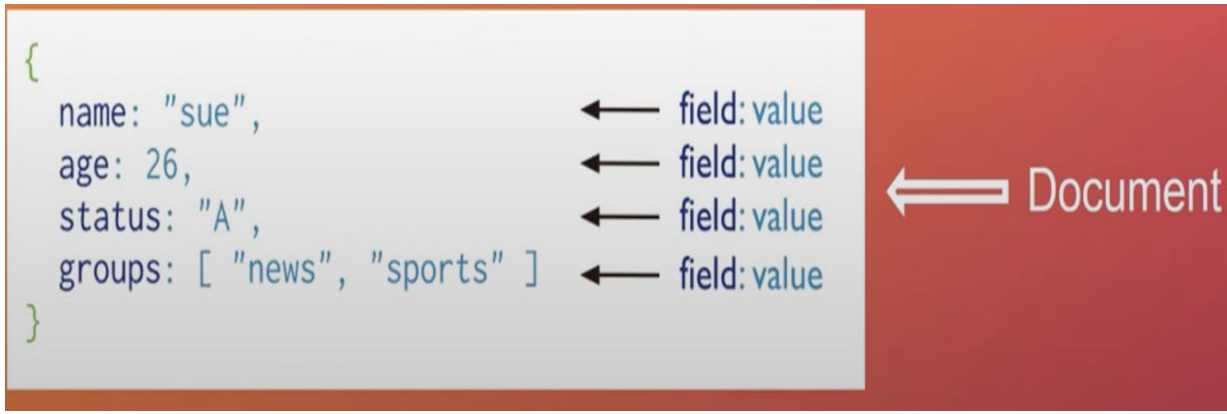


NoSQL Databases:MongoDB

the name Mongo comes from “humongous”—with performance and easy data access as core design goals.

Comparison of Mongo with MySQL





Comparison

The following operators can be used in queries to compare values:

- `$eq` : Values are equal
- `$ne` : Values are not equal
- `$gt` : Value is greater than another value
- `$gte` : Value is greater than or equal to another value
- `$lt` : Value is less than another value
- `$lte` : Value is less than or equal to another value
- `$in` : Value is matched within an array

Logical

The following operators can logically compare multiple queries.

- `$and` : Returns documents where both queries match
- `$or` : Returns documents where either query matches
- `$nor` : Returns documents where both queries fail to match
- `$not` : Returns documents where the query does not match

Evaluation

The following operators assist in evaluating documents.

- `$regex` : Allows the use of regular expressions when evaluating field values
- `$text` : Performs a text search
- `$where` : Uses a JavaScript expression to match documents

Fields

The following operators can be used to update fields:

- `$currentDate`: Sets the field value to the current date
- `$inc`: Increments the field value
- `$rename`: Renames the field
- `$set`: Sets the value of a field
- `$unset`: Removes the field from the document

Array

The following operators assist with updating arrays.

- `$addToSet`: Adds distinct elements to an array
- `$pop`: Removes the first or last element of an array
- `$pull`: Removes all elements from an array that match the query
- `$push`: Adds an element to an array

Aggregations

Expression	Description
$sum	adds up the definite values of every document of a collection.
$avg	computes the average values of every document of a collection.
$min	finds and returns the minimum of all values from within a collection.
$max	finds and returns the maximum of all values from within a collection.
$push	feeds in the values to an array in the associated document.
$first	fetches out the first document.
$last	fetches out the last document.
$addToSet	feeds in the values to an array without duplication.

Replication and Sharding

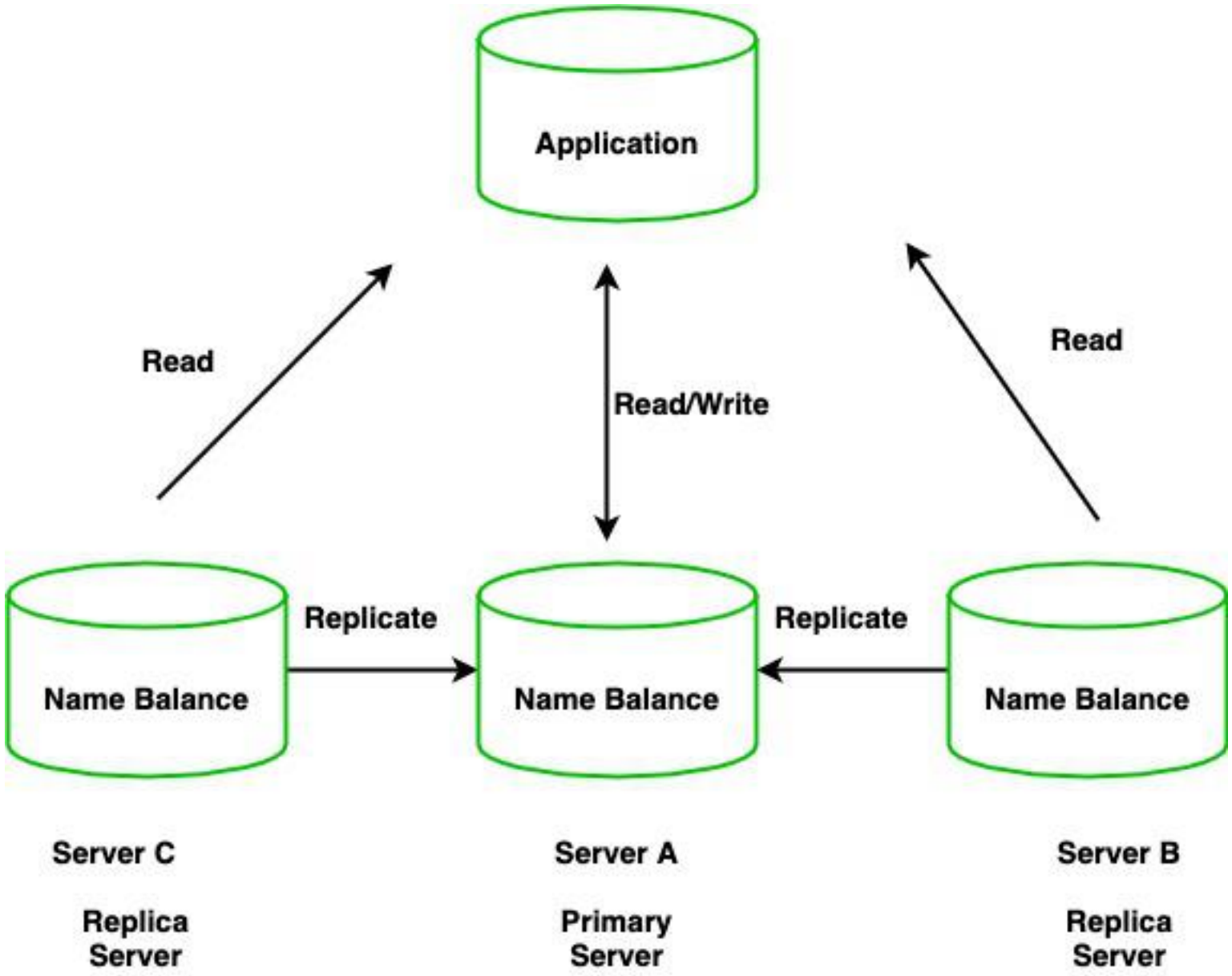
Replication can be simply understood as the duplication of the data-set whereas sharding is partitioning the data-set into discrete parts.By sharding, we divided your collection into different parts.

Replicating your database means you make imagers of your data-set. In terms of functionality delivered.

Replication

Replication is the method of duplication of data across multiple servers. For example, we have an application and it reads and writes data to a database and says this server A has a name and balance which will be copied/replicate to two other servers in two different locations.

Replication



By doing this, will get redundancy and increases data availability with multiple copies of data on different database servers.

So, it will increase the performance of reading scaling. The set of servers that maintain the same copy of data is known as replica servers or MongoDB instances

Replication

Replication Key Features :

- Replica sets are the clusters of N different nodes that maintain the same copy of the data set.
- The primary server receives all write operations and record all the changes to the data i.e, oplog.
- The secondary members then copy and apply these changes in an asynchronous process.
- All the secondary nodes are connected with the primary nodes. there is one heartbeat signal from the primary nodes. If the primary server goes down an eligible secondary will hold the new primary.

Why Replication?

- High Availability of data disasters recovery
- No downtime for maintenance (like backups index rebuilds and compaction)
- Read Scaling (Extra copies to read from)

Sharding

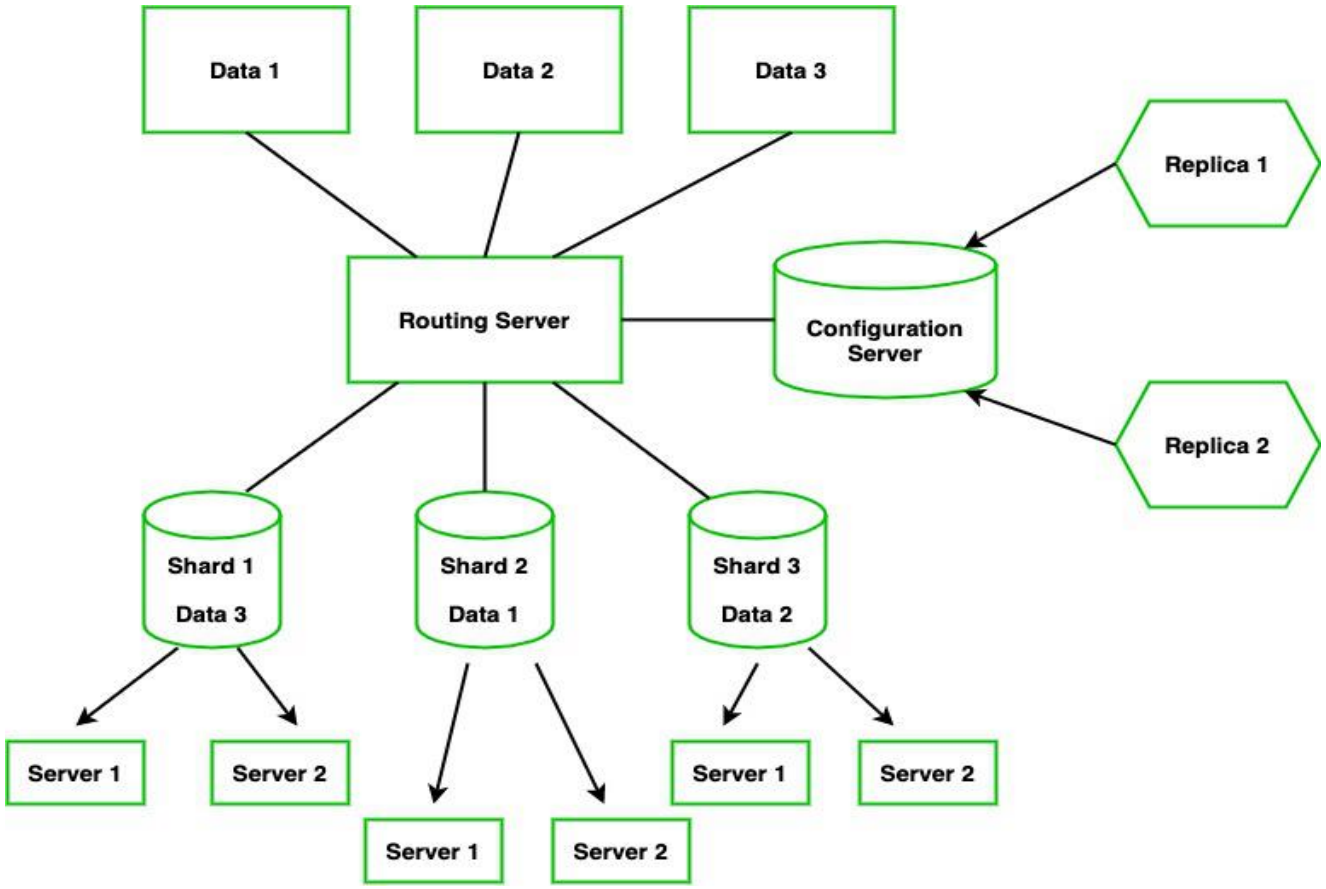
Sharding is a method for allocating data across multiple machines. MongoDB used sharding to help deployment with very big data sets and large throughput the operation. By sharding, you combine more devices to carry data extension and the needs of read and write operations.

Why Sharding?

- Database systems having big data sets or high throughput requests can doubt the ability of a single server.
- For example, High query flows can drain the CPU limit of the server.
- The working set sizes are larger than the system’s RAM to stress the I/O capacity of the disk drive.

How does Sharding work?

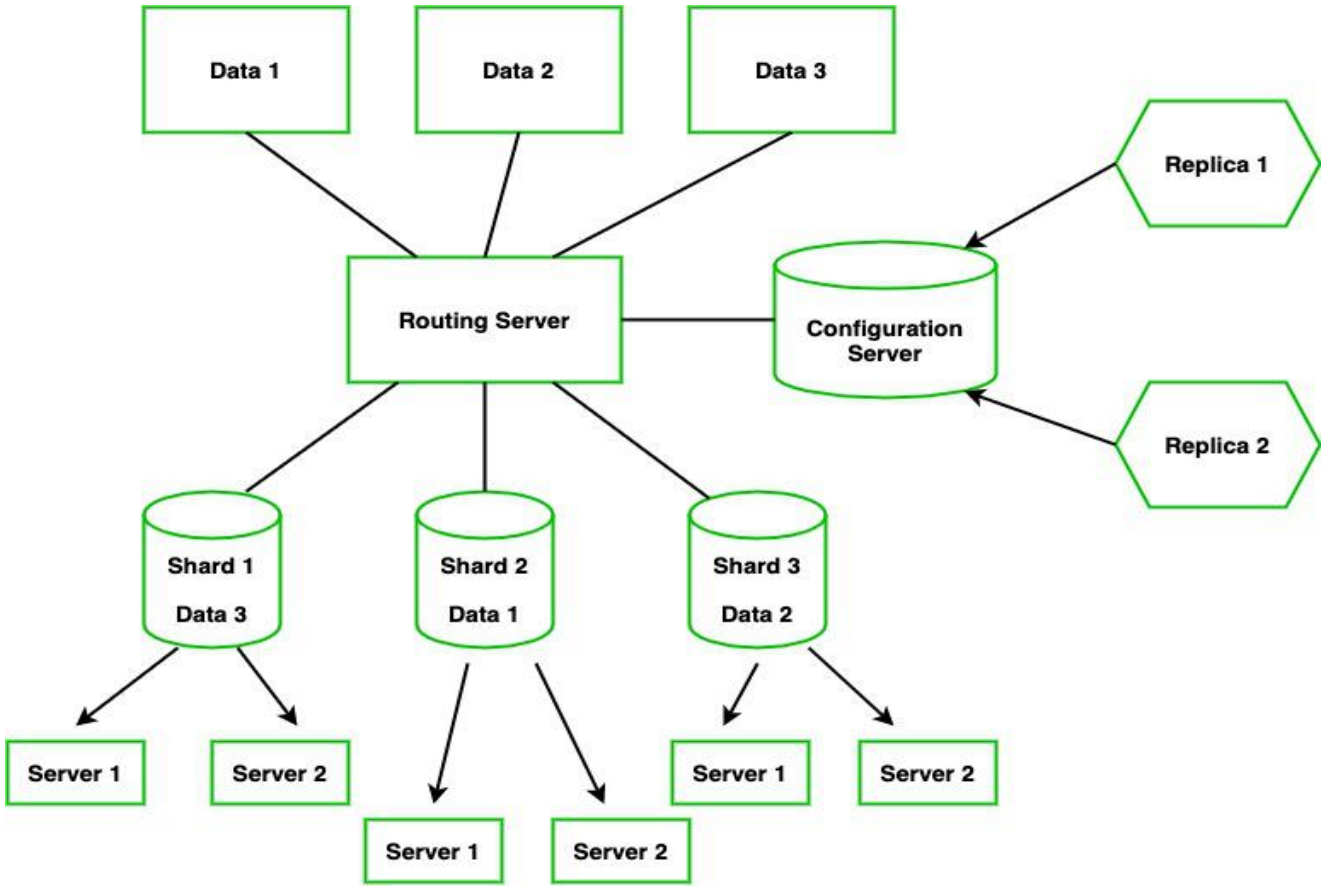
Sharding determines the problem with horizontal scaling breaking the system dataset and store over multiple servers, adding new servers to increase the volume as needed.



Now, instead of one signal as primary, we have multiple servers called Shard. We have different routing servers that will route data to the shard servers. For example: Let say we have Data 1, Data 2, and Data 3 this will be going to the routing server which will route the data (i.e, Different Data will go to a particular Shard) Each Shard holds some pieces of data. Here the configuration server will hold the metadata and it will configure the routing server to integrate the particular data to a shard however configure server is the MongoDB instance if it goes down then the entire server will go down, So it again has Replica Configure database

How does Sharding work?

Sharding determines the problem with horizontal scaling breaking the system dataset and store over multiple servers, adding new servers to increase the volume as needed.



Advantages of Sharding :

- Sharding adds more server to a data field automatically adjust data loads across various servers.
- The number of operations each shard manage got reduced.
- It also increases the write capacity by splitting the write load over multiple instances.
- It gives high availability due to the deployment of replica servers for shard and config.
- Total capacity will get increased by adding multiple shards.

GeoSpatial Queries

MongoDB provides the functionality to store locations under the object type geoJSON and their coordinates against the coordinate field in [longitude, latitude] form where longitude must lie between [-180, 180] and latitude must lie between [-90,90], both inclusive.

geoJSON types that are provided by MongoDB

1. POINT:

Represents a single location point, declared as below,

```
{
type: "Point",
coordinates: [ 27, 67 ]
}
```

2. LINESTRING:

The LineString coordinates field is an array of two or more points that represent a path in the real world.
It is declared as-

```
{
type: "LineString",
coordinates: [ [ 27, 67 ], [ 52, 4 ] ]
}
```

GeoSpatial Queries

geoJSON types that are provided by MongoDB

3. POLYGON:

The Polygon coordinates field is an array of linear rings. Linear rings have at least four coordinates, and the first coordinate is the same as the last, making a loop.

Polygon coordinates may contain a single linear ring or multiple rings.

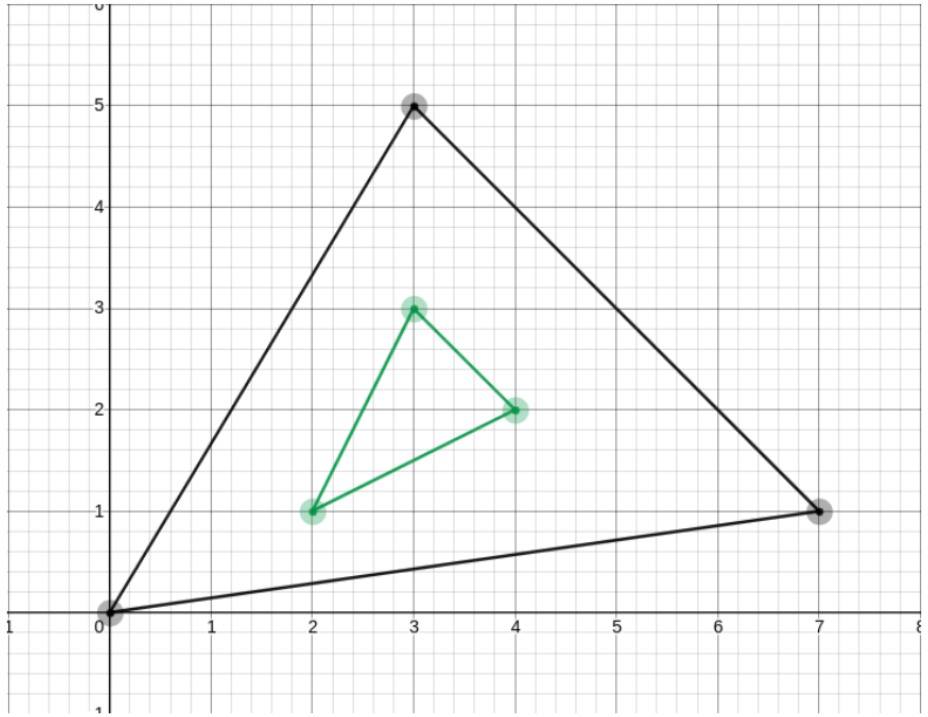
In the case of a single ring, polygon can be declared as,

```
{
type: "Polygon",
coordinates: [[[2, 1], [3, 4], [5, 1], [2, 1]]]
}
```

GeoSpatial Queries

geoJSON types that are provided by MongoDB

In the case of multiple rings:
The first ring is the outer ring,
Interior rings are completely contained by the outer ring,
Interior rings do not overlap, intersect, or share edges,
No ring self intersects.



Declaration is given as,

```
{
  type: "Polygon",
  coordinates: [
      [[0.01, 0.02], [7.01, 1.03], [3.01, 5.1], [0.01, 0.02]],
      [[2.0, 1.01], [4.0, 2.0], [3.02, 3.0], [2.0, 1.01]]
   ]
}
```

GeoSpatial Queries

geoJSON types that are provided by MongoDB

4. MULTIPOINT:

In LineString coordinates the field is an array of points.It is defined as,

```
{
  type: "MultiPoint",
  coordinates: [
      [162.07985, 20.66911],
      [126.73567, -21.59424],
      [-57.94873, 0.03998]
  ]
}
```

5. MULTILINESTRING:

In MultiLineString, the coordinates field is array of linestrings,

```
{
  type: "MultiLineString",
  coordinates: [
      [[162.07985, 20.66911], [163.07945, 21.56721]],
      [[126.73567, -21.59424], [127.74537, -22.54623]],
      [[-57.94873, 0.03998], [-58.73823, 1.12934]]
  ]
}
```

6. MULTIPOLYGON:

In MultiPolygon, the coordinates field is array of polygons,

```
{
  type: "MultiPolygon",
  coordinates: [
      [[[0.01, 0.02], [7.01, 1.03], [3.01, 5.1], [0.01, 0.02]]],
      [[[2.2, 1.01], [4.02, 2.01], [3.2, 3], [2.2, 1.01]]]
  ]
}
```

6. GEOMETRYCOLLECTION:

It is a complex geoJSON object that is used to store the above-mentioned geometries under one variable. It is defined as,

```
{
  type: "GeometryCollection",
  geometries: [
    {
      type: "MultiPoint",
      coordinates: [
          [162.07985, 20.66911],
          [126.73567, -21.59424],
          [-57.94873, 0.03998]
      ]
    },
    {
      type: "MultiLineString",
      coordinates: [
          [[162.07985, 20.66911], [163.07945, 21.56721]],
          [[126.73567, -21.59424], [127.74537, -22.54623]],
          [[-57.94873, 0.03998], [-58.73823, 1.12934]]
```

GEOSPATIAL INDEXES:

In MongoDB we need to create geospatial indexes to perform geospatial queries as we do in text search.
There are two types of indexes for geospatial search.

1. 2D

The 2d index is used when we are performing calculations on two-dimensional geometrical planes. We will not discuss these in detail in this article. A 2d index can be created by running

```
db.collection.createIndex( { <location field> : "2d" } )
```

Where location_field whose value is in legacy coordinates ([x,y] instead of [long,lat])

2. 2DSPHERE

A 2dsphere index is used when performing queries/calculation on earth-like spheres.
It can be created as,

```
db.collection.createIndex( { <location_field> : "2dsphere" } )
```

Here, location field can be of type geoJSON object or legacy coordinates.