Practical -6

Aim - Implementation of Sharding in MongoDB.

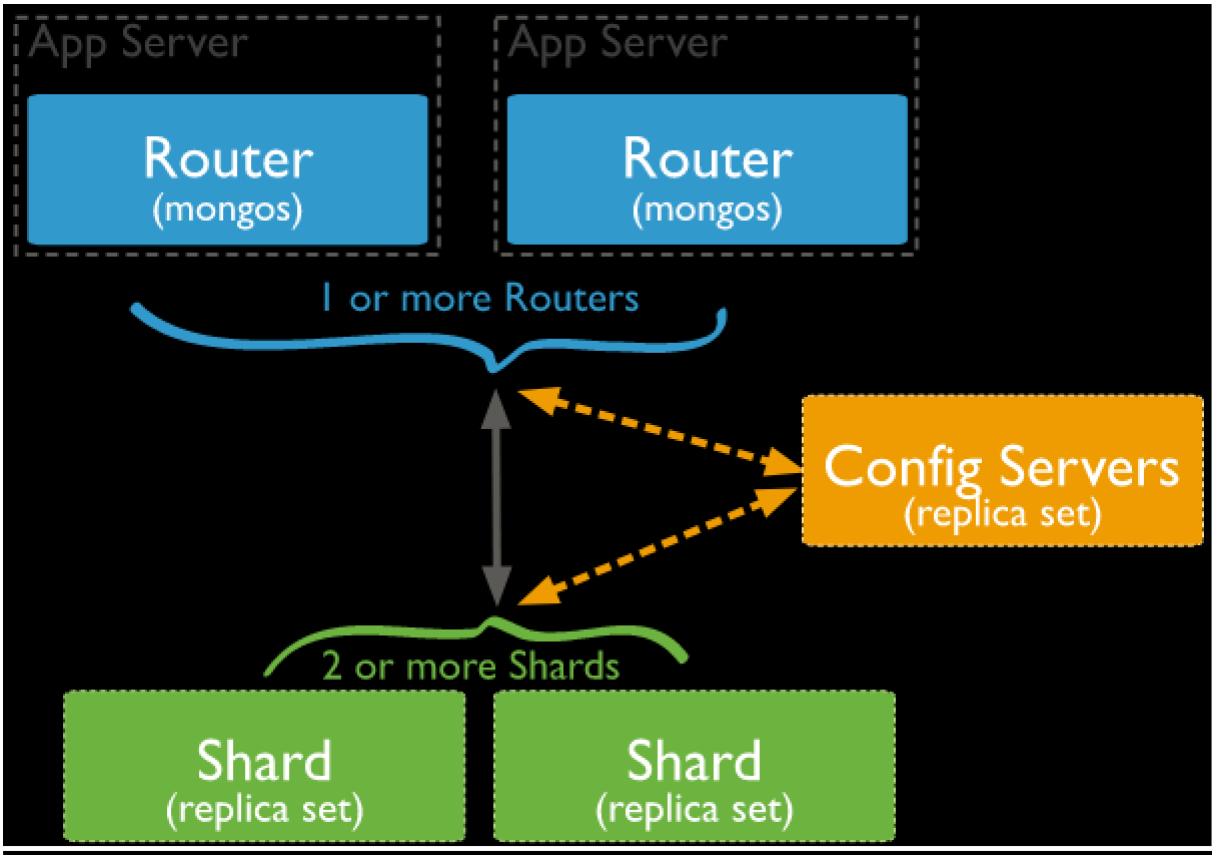
Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Sharded Cluster

A MongoDB sharded cluster consists of the following components:

- shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- config servers: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

The following graphic describes the interaction of components within a sharded cluster:



MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

Shard Keys

MongoDB uses the shard key to distribute the collection's documents across shards. The shard key consists of a field or fields that exist in every document in the target collection.

You 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. See Shard Key Specification.

Advantages of Sharding

Reads / Writes

MongoDB distributes the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster

operations. Both read and write workloads can be scaled horizontally across the cluster by adding more shards.

For queries that include the shard key or the prefix of a compound shard key, mongos can target the query at a specific shard or set of shards. These targeted operations are generally more efficient than broadcasting to every shard in the cluster.

Storage Capacity

Sharding distributes data across the shards in the cluster, allowing each shard to contain a subset of the total cluster data. As the data set grows, additional shards increase the storage capacity of the cluster.

High Availability

A sharded cluster can continue to perform partial read / write operations even if one or more shards are unavailable. While the subset of data on the unavailable shards cannot be accessed during the downtime, reads or writes directed at the available shards can still succeed.

Starting in MongoDB 3.2, you can deploy config servers as replica sets. A sharded cluster with a Config Server Replica Set (CSRS) can continue to process reads and writes as long as a majority of the replica set is available.

In production environments, individual shards should be deployed as replica sets, providing increased redundancy and availability.

Considerations Before Sharding

Sharded cluster infrastructure requirements and complexity require careful planning, execution, and maintenance.

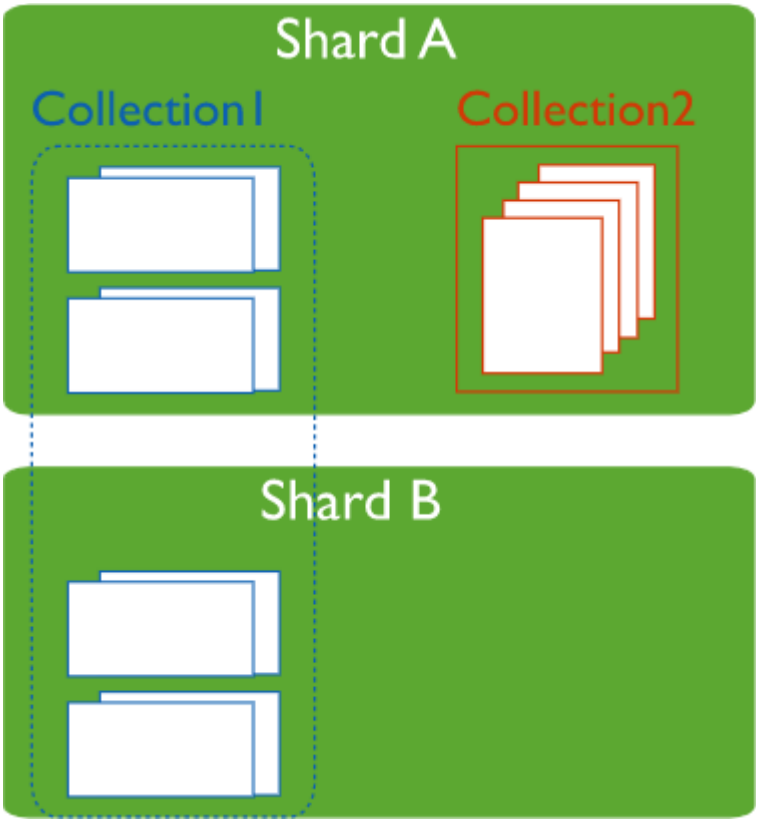
Careful consideration in choosing the shard key is necessary for ensuring cluster performance and efficiency. You cannot change the shard key after sharding, nor can you unshard a sharded collection. See Choosing a Shard Key.

Sharding has certain operational requirements and restrictions. See Operational Restrictions in Sharded Clusters for more information.

If queries do not include the shard key or the prefix of a compound shard key, mongos performs a broadcast operation, querying all shards in the sharded cluster. These scatter/gather queries can be long running operations.

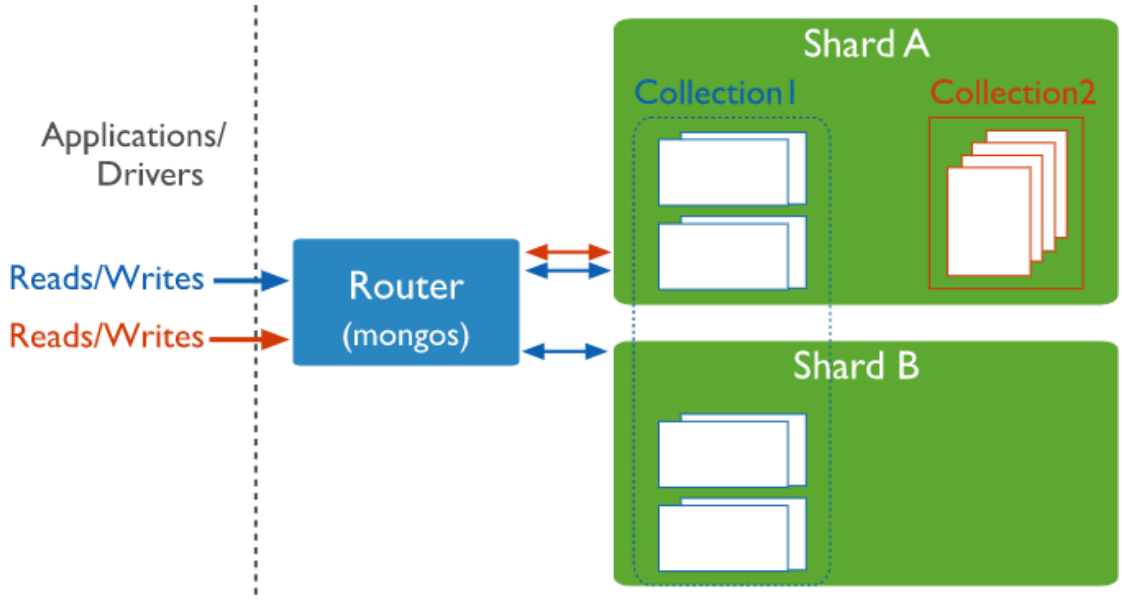
Sharded and Non-Sharded Collections

A database can have a mixture of sharded and unsharded collections. Sharded collections are partitioned and distributed across the shards in the cluster. Unsharded collections are stored on a primary shard. Each database has its own primary shard.



Connecting to a Sharded Cluster

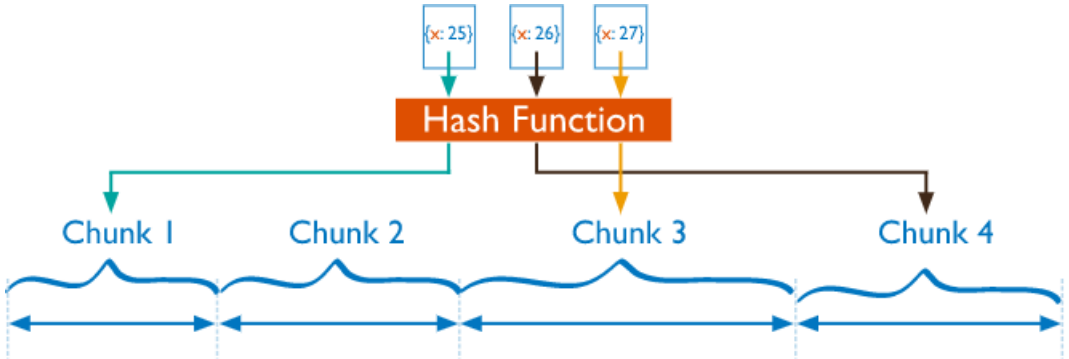
You must connect to a mongos router to interact with any collection in the sharded cluster. This includes sharded and unsharded collections. Clients should never connect to a single shard in order to perform read or write operations.



You can connect to a mongos the same way you connect to a mongod, such as via the mongo shell or a MongoDB driver.

Hashed Sharding

Hashed sharding uses a hashed index to partition data across your shared cluster. Hashed indexes compute the hash value of a single field as the index value; this value is used as your shard key.



Hashed sharding provides more even data distribution across the sharded cluster at the cost of reducing Broadcast Operations..

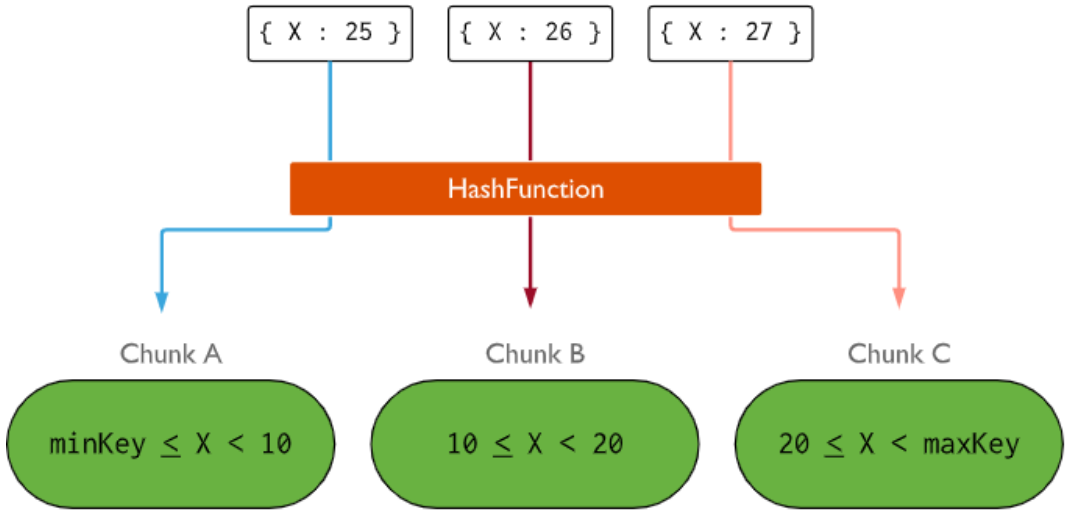
Hashed Sharding Shard Key

The field you choose as your hashed shard key should have a good cardinality, or large number of different values. Hashed keys are ideal for shard keys with fields that change monotonically like ObjectId values or timestamps. A good example of this is the default _id field, assuming it only contains ObjectID values.

To shard a collection using a hashed shard key, see Shard a Collection.

Hashed vs Ranged Sharding

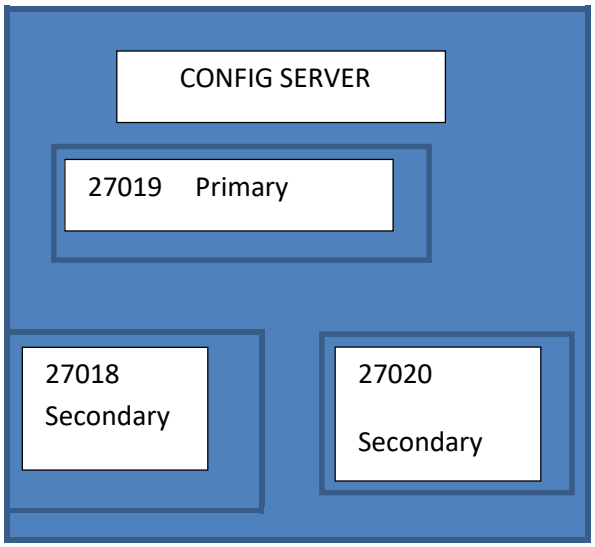
Given a collection using a monotonically increasing value X as the shard key, using ranged sharding results in a distribution of incoming inserts similar to the following:

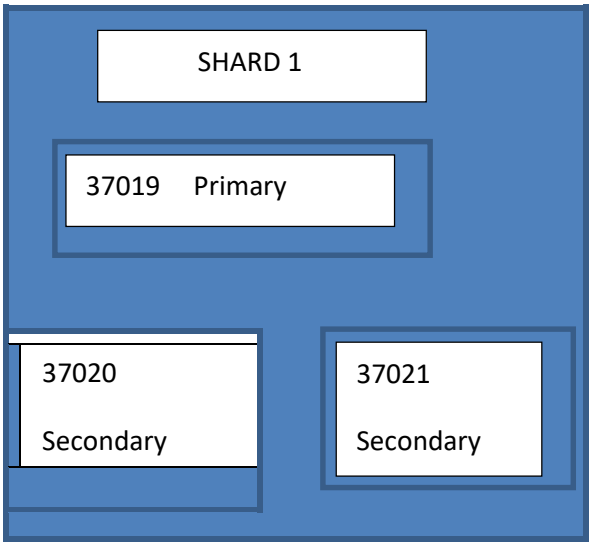


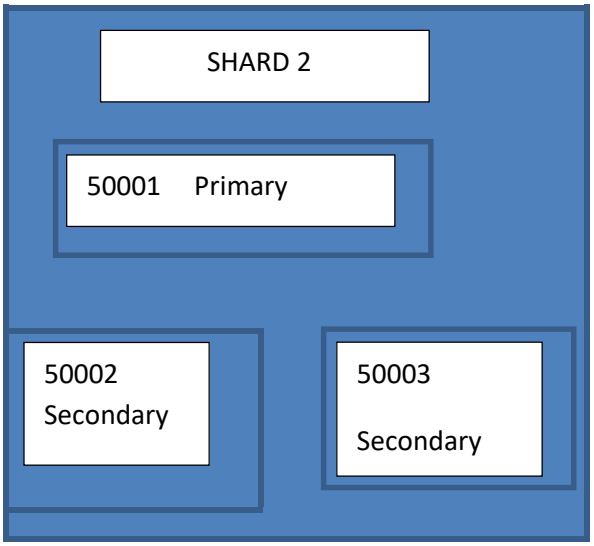
Since the data is now distributed more evenly, inserts are efficiently distributed throughout the cluster.

Shard the Collection

Use the sh.shardCollection() method, specifying the full namespace of the collection and the target hashed index to use as the shard key.

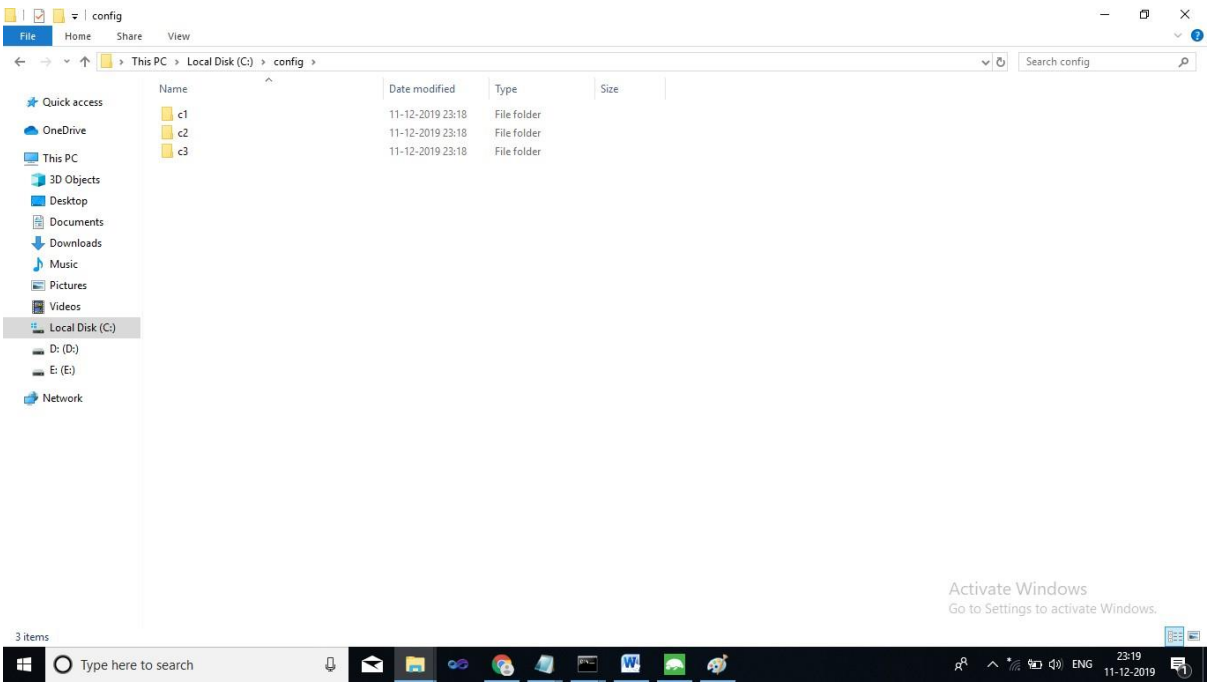






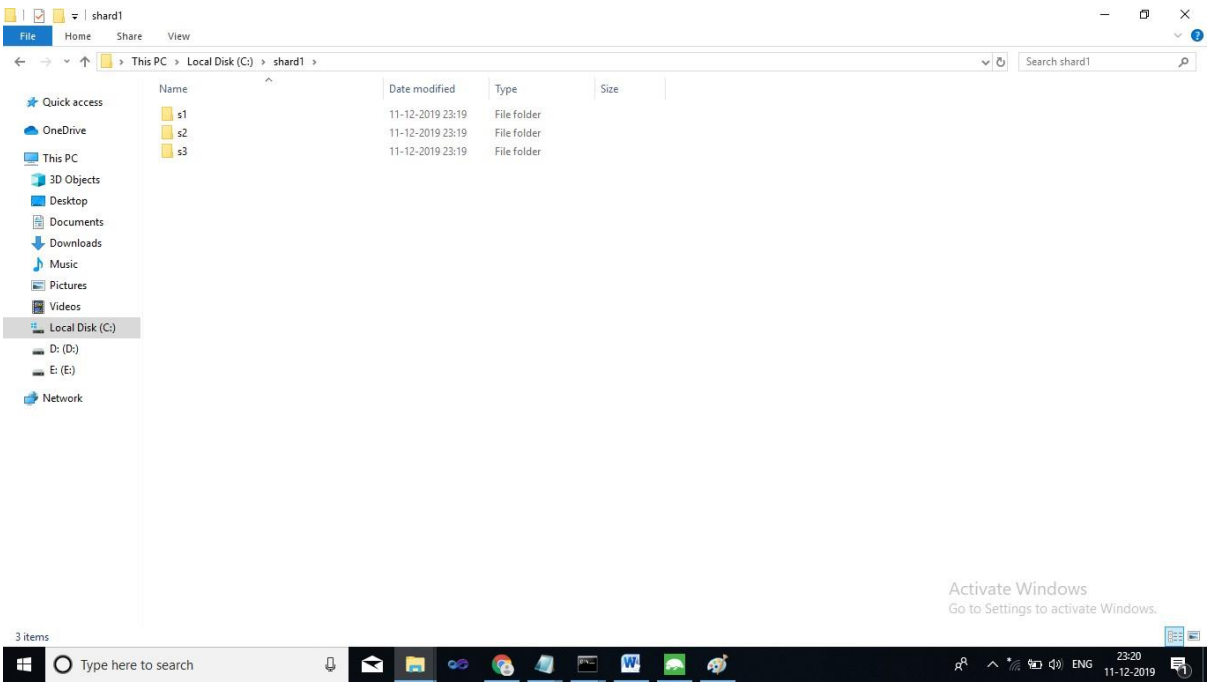
STEP 1

Create config folder in C Drive.Inside that create three folders c1,c2,c3.



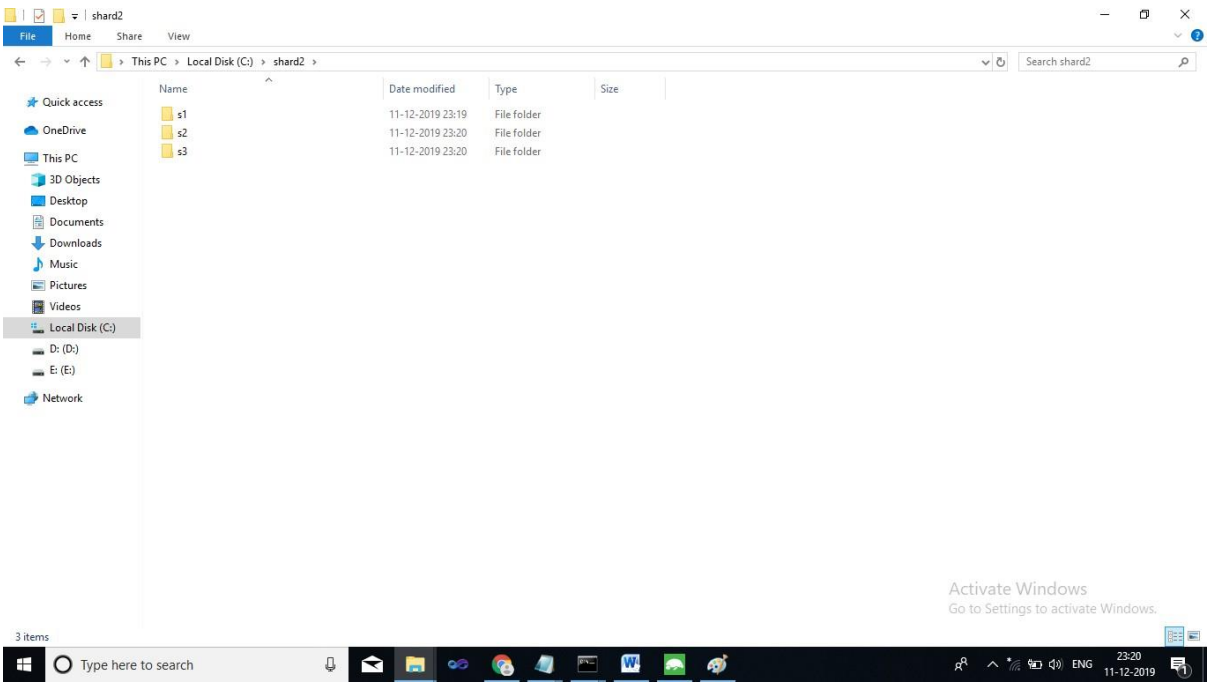
STEP 2

Create shard1 folder in drive. Inside that create three sub folders s1,s2,s3.



STEP 3

Create shard2 folder in drive. Inside that create three sub folders s1,s2,s3.



STEP 4

OPEN command Prompt.

Create three Config Server. Then Set one Config Server as Primary and remaining two as secondary.

Using below commands.

C:\Program Files\MongoDB\Server\4.2\bin>mongod --configsvr --dbpath "C:\config\c1" --port 27018 --replSet=c0

C:\Program Files\MongoDB\Server\4.2\bin>mongod --configsvr --dbpath "C:\config\c2" --port 27019 --replSet=c0

C:\Program Files\MongoDB\Server\4.2\bin>mongod --configsvr --dbpath "C:\config\c3" --port 27020 --replSet=c0

Now open any mongo client among all and connect with server using below command.

C:\>mongo --port 27019

Create a group ,assign ids and replica using below command.

config = { _id: "c0", members:[

{ _id : 0, host : "localhost:27018" },

{ _id : 1, host : "localhost:27019" },

{ _id : 2, host : "localhost:27020" }]};

Set any node as primary node using below given command.

rs.initiate(config)

STEP 5 (SHARD 1)

OPEN command Prompt.

Create three Shard Server. Then Set one shard Server as Primary and remaining two as secondary.

Using below commands.

C:\Program Files\MongoDB\Server\4.2\bin>mongod --replSet s1 --dbpath "C:\shard2\s1" --port 37019 --shardsvr

C:\Program Files\MongoDB\Server\4.2\bin>mongod --replSet s1 --dbpath "C:\shard2\s2" --port 37020 --shardsvr

C:\Program Files\MongoDB\Server\4.2\bin>mongod --replSet s1 --dbpath "C:\shard2\s3" --port 37021 –shardsvr

Now open any mongo client among all and connect with server using below command.

C:\>mongo --port 37019

C***reate a group ,assign ids and replica using below command.***

config = { _id: "s1", members:[

{ _id : 0, host : "localhost:37019" },

{ _id : 1, host : "localhost:37020" },

{ _id : 2, host : "localhost:37021" }]};

Set any node as primary node using below given command.

rs.initiate(config)

STEP 6 (SHARD 2)

OPEN command Prompt.

Create three Shard Server. Then Set one shard Server as Primary and remaining two as secondary.

Using below commands.

C:\Program Files\MongoDB\Server\4.2\bin>mongod --replSet s0 --dbpath "C:\shard1\s1" --port 50001 --shardsvr

C:\Program Files\MongoDB\Server\4.2\bin>mongod --replSet s0 --dbpath "C:\shard1\s2" --port 50002 --shardsvr

C:\Program Files\MongoDB\Server\4.2\bin>mongod --replSet s0 --dbpath "C:\shard1\s3" --port 50003 --shardsvr

Now open any mongo client among all and connect with server using below command.

C:\>mongo --port 50001

Create a group ,assign ids and replica using below command

config = { _id: "s0", members:[

{ _id : 0, host : "localhost:50001" },

{ _id : 1, host : "localhost:50002" },

{ _id : 2, host : "localhost:50003" }]};

Set any node as primary node using below given command.

rs.initiate(config)

STEP 7

Communication with both shards is possible using config servers.

So open cmd and write command.This creates mongos config server.

C:\>mongos.exe --configdb c0/localhost:27018,localhost:27019,localhost:27020 --port 1000

Open a client mongos command window to communicate with server.

C:\>mongo.exe --host localhost --port 1000

And then both the shards which we are created add that shards into shard cluster using given below commands.

mongos> sh.addShard("s0/localhost:50001");

mongos> sh.addShard("s1/localhost:37019");

STEP 8

To see the shards which are in shard cluster run the command in the same command prompt(1000)

db.runCommand({listShards:1});

1. **Now create database.**
   mongos> use projectionDB
   switched to db projectionDB
2. **Move to admin mode.**
   mongos> use admin
   switched to db admin
3. **Enable Database Sharding property.So that table data can be distributed on multiple shards using given command.**
   mongos> db.runCommand({enableSharding:"projectionDB"});
4. **Create collection and set hash key.**

```
mongos> sh.shardCollection("projectionDB.bios", {"name":"hashed"})
```

5. **Now insert data into bios table.**

```
mongos> db.bios.insert({
 name:"Prachi",
 lname:"Shah",
 city:"Mehsana",
    state:"Gujarat"
 });

mongos> db.bios.insert({
name:"sunita",
lname:"chauhan",
city:"Ahmedabad",
state:"Gujarat" })

mongos> db.bios.insert({
...    name:"agam",
...    lname:"shah",
...    city:"Mehsana",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Shraddha",
...    lname:"Mehta",
...    city:"Baroda",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Shurya",
...    lname:"Pandya",
...    city:"Mehsana",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Khushi",
...    lname:"Patel",
...    city:"Mehsana",
```

```
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Pooja",
...    lname:"Mevada",
...    city:"Ahmedabad",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Meeta",
...    lname:"Shah",
...    city:"Ahmedabad",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Avinash",
...    lname:"Mehta",
...    city:"Ahmedabad",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Vency",
...    lname:"Parekh",
...    city:"Ahmedabad",
...     state:"Gujarat"
... })

mongos> db.bios.insert({
...    name:"Dhruv",
...    lname:"Shah",
...    city:"Ahmedabad",
...     state:"Gujarat"
... })
```

Run the command to see the total number of documents in bios table.
db.gnu.count();

STEP 9

To see how data is distributed on shard 1.

Open the mongo client of shard 1 which is on port 50001.

Run the commands like.

Show dbs

Show collection

use projectionDB;

db.bios.count();

STEP 10

To see how data is distributed on shard 2.

Open the mongo client of shard 2 which is on port 37019.

Run the commands like.

Show dbs

Show collection

use projectionDB;

db.bios.count();