**MongoDB – Session – Day 1**

**Replication**

**What is Replication?**

Replication in MongoDB is the process of synchronizing data across multiple servers. It ensures data redundancy, high availability, and disaster recovery.

**MongoDB Replica Set**

A MongoDB replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and high availability. A replica set consists of:

1. **Primary**: The main server that receives all the write operations.
2. **Secondary**: Servers that replicate the primary's data and can serve read operations.
3. **Arbiter**: A server that participates in elections but does not hold data.

**Setting Up the Replica**

1. **Install MongoDB** on each server in the replica set.
2. **Start MongoDB** instances with the --replSet option.

mongod --replSet "rs0" --dbpath /data/db1 --port 27017

mongod --replSet "rs0" --dbpath /data/db2 --port 27018

mongod --replSet "rs0" --dbpath /data/db3 --port 27019

1. **Connect to the primary MongoDB instance** and initiate the replica set.

mongo --port 27017

rs.initiate(

{

\_id : "rs0",

members: [

{ \_id: 0, host: "localhost:27017" },

{ \_id: 1, host: "localhost:27018" },

{ \_id: 2, host: "localhost:27019" }

]

}

)

**Configuration**

You can check the configuration of the replica set using: - rs.conf()

**Replication Commands**

* **Check replica set status**: rs.status()
* **Add a member**: rs.add("hostname:port")
* **Remove a member**: rs.remove("hostname:port")

**Local DB**

Each member of a replica set has a local database that stores replication-related data such as the oplog (operations log), which is used to replicate changes from the primary to the secondaries.

**Reconfiguration of Running Replica Set**

1. **Get the current configuration**: cfg = rs.conf()
2. **Make the necessary changes** to the configuration object.
3. **Apply the new configuration**:

rs.reconfig(cfg)

**Reads and Writes on a Replica Set**

* **Writes** are always performed on the primary.
* **Reads** can be performed on the primary or secondaries based on the read preference.

**Failover and Election**

If the primary becomes unavailable, an election process selects a new primary from the secondaries.

**Read Concern**

Read concern levels determine the consistency and isolation properties of the data read from a replica set.

* **"local"**: Returns the most recent data available on the node.
* **"majority"**: Returns data that has been acknowledged by a majority of the nodes.
* **"linearizable"**: Ensures strict single-document read isolation.

**Write Concern**

Write concern levels specify the level of acknowledgment requested from MongoDB for write operations.

* **"w:1"**: Acknowledgment from the primary only.
* **"w**

**"**: Acknowledgment from the majority of replica set members.

* **"w:0"**: No acknowledgment.

**Sharding**

**What is Sharding?**

Sharding is a method for distributing data across multiple machines. It allows MongoDB to scale out horizontally, managing large data sets and high throughput operations.

**When to Shard**

Sharding is necessary when a single server's capacity is insufficient to handle the volume of data or the rate of read and write operations.

**Sharding Architecture**

A sharded cluster consists of:

1. **Shard Servers**: Each shard holds a subset of the data.
2. **Config Servers**: Store metadata and configuration settings for the cluster.
3. **Mongos**: Acts as a query router, directing operations to the appropriate shard.

**Setting Up the Shard Cluster**

1. **Start config servers**:

mongod --configsvr --replSet configReplSet --dbpath /data/config1 --port 27019

mongod --configsvr --replSet configReplSet --dbpath /data/config2 --port 27020

mongod --configsvr --replSet configReplSet --dbpath /data/config3 --port 27021

1. **Initiate the config server replica set**:

mongo --port 27019

rs.initiate(

{

\_id: "configReplSet",

configsvr: true,

members: [

{ \_id: 0, host: "localhost:27019" },

{ \_id: 1, host: "localhost:27020" },

{ \_id: 2, host: "localhost:27021" }

]

}

)

1. **Start shard servers**:

mongod --shardsvr --replSet shardReplSet1 --dbpath /data/shard1 --port 27022

mongod --shardsvr --replSet shardReplSet2 --dbpath /data/shard2 --port 27023

1. **Initiate shard server replica sets**:

mongo --port 27022

rs.initiate(

{

\_id: "shardReplSet1",

members: [

{ \_id: 0, host: "localhost:27022" }

]

}

)

mongo --port 27023

rs.initiate(

{

\_id: "shardReplSet2",

members: [

{ \_id: 0, host: "localhost:27023" }

]

}

)

1. **Start the mongos instance**:

mongos --configdb configReplSet/localhost:27019,localhost:27020,localhost:27021 --port 27024

1. **Add shards to the cluster**:

mongo --port 27024

sh.addShard("shardReplSet1/localhost:27022")

sh.addShard("shardReplSet2/localhost:27023")

**Config DB**

The config database (configDB) stores metadata and configuration settings for the sharded cluster.

**Shard Keys**

A shard key is a field or set of fields that determine the distribution of the data across the shards.

**Picking a Good Shard Key**

A good shard key should ensure even data distribution and support the common queries. It should be:

* **High cardinality**: Many unique values.
* **Low frequency**: Avoid fields with a few frequently repeating values.

**Hashed Shard Keys**

Hashed shard keys ensure an even data distribution by hashing the shard key field's value.

**Chunks**

Chunks are subsets of data in a sharded cluster. MongoDB splits data into chunks and distributes them across shards.

**Balancing**

The balancer ensures chunks are evenly distributed across shards. It automatically migrates chunks to redistribute data as needed.

**Query in Sharded Cluster**

Queries in a sharded cluster can be:

* **Targeted**: Directly routed to the appropriate shard(s) based on the shard key.
* **Scatter-Gather**: Broadcasted to all shards if the shard key is not specified in the query.

**Targeted Queries vs Scatter**

* **Targeted Queries**: Efficient and fast as they only access relevant shards.
* **Scatter Queries**: Can be slow and resource-intensive since they access all shards.

**Lab - 1**

**Demonstrate Replica & Sharding on Atlas**

**Using Dockers –**

Setting up a MongoDB Replica Set Cluster using Docker on Windows involves several steps. Below are the detailed instructions for setting up a 3-node MongoDB replica set using Docker. This guide assumes that you have Docker Desktop installed and running on your Windows machine.

Ref - <https://www.mongodb.com/resources/products/compatibilities/deploying-a-mongodb-cluster-with-docker>

**Step-by-Step Setup Guidelines**

**Step 1: Create a Docker Network**

Create a Docker network to enable communication between the MongoDB containers.

docker network create mongo-replica-set

**Step 2: Create MongoDB Containers**

Create three MongoDB containers, each representing a node in the replica set.

docker run -d --name mongo1 --network mongo-replica-set -p 27017:27017 mongo:latest --replSet rs0

docker run -d --name mongo2 --network mongo-replica-set -p 27018:27017 mongo:latest --replSet rs0

docker run -d --name mongo3 --network mongo-replica-set -p 27019:27017 mongo:latest --replSet rs0

**Step 3: Initialize the Replica Set**

1. **Connect to the first MongoDB container**:

docker exec -it mongo1 mongo

1. **Initiate the replica set**:

rs.initiate(

{

\_id: "rs0",

members: [

{ \_id: 0, host: "mongo1:27017" },

{ \_id: 1, host: "mongo2:27017" },

{ \_id: 2, host: "mongo3:27017" }

]

}

)

1. **Check the replica set status**:

rs.status()

**Step 4: Verify the Replica Set**

To verify that the replica set is working correctly, connect to each MongoDB instance and check the status:

docker exec -it mongo1 mongo

rs.status()

docker exec -it mongo2 mongo

rs.status()

docker exec -it mongo3 mongo

rs.status()

**Step 5: Configure MongoDB for Replica Set**

You may need to adjust your application configuration to connect to the MongoDB replica set. Use the connection string format as shown below:

mongodb://mongo1:27017,mongo2:27017,mongo3:27017/?replicaSet=rs0

**Summary – Take Away**

1. **Create a Docker network** to allow communication between MongoDB nodes.
2. **Run MongoDB containers** specifying the replica set name.
3. **Initialize the replica set** from one of the MongoDB instances.
4. **Verify the replica set** status on all nodes.
5. **Update your application connection string** to use the replica set.

**Additional Tips**

* **Persistence**: If you want your data to persist across container restarts, you need to mount volumes to the MongoDB containers.

docker run -d --name mongo1 --network mongo-replica-set -p 27017:27017 -v mongo1\_data:/data/db mongo:latest --replSet rs0

docker run -d --name mongo2 --network mongo-replica-set -p 27018:27017 -v mongo2\_data:/data/db mongo:latest --replSet rs0

docker run -d --name mongo3 --network mongo-replica-set -p 27019:27017 -v mongo3\_data:/data/db mongo:latest --replSet rs0

**Docker Compose**: You can use Docker Compose to simplify the setup process.

Create a docker-compose.yml file with the following content:

version: '3.8'

services:

mongo1:

image: mongo:latest

container\_name: mongo1

networks:

- mongo-replica-set

ports:

- 27017:27017

command: ["--replSet", "rs0"]

volumes:

- mongo1\_data:/data/db

mongo2:

image: mongo:latest

container\_name: mongo2

networks:

- mongo-replica-set

ports:

- 27018:27017

command: ["--replSet", "rs0"]

volumes:

- mongo2\_data:/data/db

mongo3:

image: mongo:latest

container\_name: mongo3

networks:

- mongo-replica-set

ports:

- 27019:27017

command: ["--replSet", "rs0"]

volumes:

- mongo3\_data:/data/db

networks:

mongo-replica-set:

driver: bridge

volumes:

mongo1\_data:

mongo2\_data:

mongo3\_data:

To start the containers and initiate the replica set:

docker-compose up -d

docker exec -it mongo1 mongo

Then initiate the replica set as described in the earlier steps.

This setup should give you a fully functional MongoDB replica set running on Docker on Windows. If you need any further assistance, feel free to ask!

**Lab - 2**

**Steps - Setting up a MongoDB 3-node replica set on Ubuntu 22.04.**

**Prerequisites**

* Three Ubuntu 22.04 servers (these can be physical, virtual machines, or cloud instances).
* SSH access to these servers with root or sudo privileges.
* MongoDB installed on each server.

**Step 1: Install MongoDB on Each Server**

1. **Import the public key used by the package management system**:

wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -

1. **Create a list file for MongoDB**:

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

1. **Reload the local package database**:

sudo apt-get update

1. **Install the MongoDB packages**:

sudo apt-get install -y mongodb-org

**Step 2: Configure MongoDB**

1. **Edit the MongoDB configuration file (/etc/mongod.conf) on each server**:

sudo nano /etc/mongod.conf

1. **Modify the configuration file to include the replica set name and bind IP**:

net:

bindIp: 0.0.0.0 # To allow connections from any IP address

port: 27017

replication:

replSetName: rs0

1. **Save and close the file**.

**Step 3: Start MongoDB Service**

1. **Start and enable the MongoDB service on each server**:

sudo systemctl start mongod

sudo systemctl enable mongod

1. **Check the status of the MongoDB service**:

sudo systemctl status mongod

**Step 4: Initialize the Replica Set**

1. **Connect to the first MongoDB instance** (you can SSH into the server or use a local connection if running the commands on the server itself):

mongo

1. **Initiate the replica set**:

rs.initiate(

{

\_id: "rs0",

members: [

{ \_id: 0, host: "mongo1:27017" },

{ \_id: 1, host: "mongo2:27017" },

{ \_id: 2, host: "mongo3:27017" }

]

} )

Replace "mongo1:27017", "mongo2:27017", and "mongo3:27017" with the actual IP addresses or hostnames of your servers.

1. **Check the status of the replica set**:

rs.status()

**Step 5: Verify the Replica Set**

To ensure that the replica set is functioning correctly, connect to each MongoDB instance and check the status:

mongo --host <hostname or IP of mongo1> --port 27017

rs.status()

mongo --host <hostname or IP of mongo2> --port 27017

rs.status()

mongo --host <hostname or IP of mongo3> --port 27017

rs.status()

**Step 6: Update Firewall Settings (if necessary)**

Ensure that the necessary ports are open on each server to allow communication between the MongoDB instances. For example, you might need to open port 27017:

sudo ufw allow 27017

**Additional Notes**

* **Persistence**: Ensure that the data directory specified in mongod.conf (default is /var/lib/mongodb) has sufficient space and is configured correctly.
* **Security**: For a production environment, consider enabling authentication and configuring appropriate user roles and access controls.

**Example Initialization Script - Automation**

You can use a script to automate the setup. Below is an example of a script you might run on one of the servers after installing MongoDB on all three servers:

#!/bin/bash

# Initialize the replica set

mongo --eval 'rs.initiate(

{

\_id: "rs0",

members: [

{ \_id: 0, host: "mongo1:27017" },

{ \_id: 1, host: "mongo2:27017" },

{ \_id: 2, host: "mongo3:27017" }

]

}

)'

# Check the replica set status

mongo --eval 'rs.status()'

Replace "mongo1:27017", "mongo2:27017", and "mongo3:27017" with your actual server IP addresses or hostnames.

The above should help you set up a MongoDB 3-node replica set on Ubuntu 22.04. If you have any questions or need further assistance, feel free to ask!

**Test Data –**

Here's a sample JSON data for a students collection with 10 records. Each student has basic information such as student\_id, name, age, gender, major, and gpa.

[

{

"student\_id": "S001",

"name": "John Doe",

"age": 20,

"gender": "Male",

"major": "Computer Science",

"gpa": 3.5

},

{

"student\_id": "S002",

"name": "Jane Smith",

"age": 21,

"gender": "Female",

"major": "Electrical Engineering",

"gpa": 3.8

},

{

"student\_id": "S003",

"name": "Emily Johnson",

"age": 22,

"gender": "Female",

"major": "Mechanical Engineering",

"gpa": 3.7

},

{

"student\_id": "S004",

"name": "Michael Brown",

"age": 23,

"gender": "Male",

"major": "Civil Engineering",

"gpa": 3.6

},

{

"student\_id": "S005",

"name": "Jessica Davis",

"age": 20,

"gender": "Female",

"major": "Biology",

"gpa": 3.9

},

{

"student\_id": "S006",

"name": "Daniel Wilson",

"age": 21,

"gender": "Male",

"major": "Physics",

"gpa": 3.4

},

{

"student\_id": "S007",

"name": "Sarah Martinez",

"age": 22,

"gender": "Female",

"major": "Chemistry",

"gpa": 3.7

},

{

"student\_id": "S008",

"name": "David Anderson",

"age": 23,

"gender": "Male",

"major": "Mathematics",

"gpa": 3.5

},

{

"student\_id": "S009",

"name": "Laura Thompson",

"age": 20,

"gender": "Female",

"major": "English",

"gpa": 3.8

},

{

"student\_id": "S010",

"name": "James Moore",

"age": 21,

"gender": "Male",

"major": "History",

"gpa": 3.6

}

]

To insert this data into a MongoDB collection, you can use the mongo shell or a MongoDB client such as MongoDB Compass.

**Using mongo Shell**

1. Save the JSON data to a file named students.json.
2. Use the following command to import the data into the students collection of the school database:

mongoimport --db school --collection students --file students.json --jsonArray

**Using MongoDB Compass**

1. Open MongoDB Compass and connect to your MongoDB instance.
2. Navigate to the school database (or create it if it doesn't exist).
3. Click on the students collection (or create it if it doesn't exist).
4. Click on the Add Data button and select Insert Document.
5. Copy and paste the JSON data into the text box and click Insert.

To set up a sharded MongoDB cluster using Docker Compose, you need to configure MongoDB instances as shards, a config server replica set, and a mongos router. Here's a detailed Docker Compose configuration for a sharded MongoDB cluster:

**Docker Compose Configuration for MongoDB Sharded Cluster**

This configuration includes three shards, each with a single MongoDB instance, a config server replica set with three members, and a mongos router.

version: '3.8'

services:

# Config Server Replica Set

configsvr1:

image: mongo:latest

container\_name: configsvr1

command: ["mongod", "--configsvr", "--replSet", "configReplSet", "--port", "27019"]

volumes:

- configsvr1\_data:/data/db

networks:

- mongo-cluster

configsvr2:

image: mongo:latest

container\_name: configsvr2

command: ["mongod", "--configsvr", "--replSet", "configReplSet", "--port", "27019"]

volumes:

- configsvr2\_data:/data/db

networks:

- mongo-cluster

configsvr3:

image: mongo:latest

container\_name: configsvr3

command: ["mongod", "--configsvr", "--replSet", "configReplSet", "--port", "27019"]

volumes:

- configsvr3\_data:/data/db

networks:

- mongo-cluster

# Shard 1

shard1\_1:

image: mongo:latest

container\_name: shard1\_1

command: ["mongod", "--shardsvr", "--replSet", "shard1ReplSet", "--port", "27018"]

volumes:

- shard1\_1\_data:/data/db

networks:

- mongo-cluster

# Shard 2

shard2\_1:

image: mongo:latest

container\_name: shard2\_1

command: ["mongod", "--shardsvr", "--replSet", "shard2ReplSet", "--port", "27018"]

volumes:

- shard2\_1\_data:/data/db

networks:

- mongo-cluster

# Shard 3

shard3\_1:

image: mongo:latest

container\_name: shard3\_1

command: ["mongod", "--shardsvr", "--replSet", "shard3ReplSet", "--port", "27018"]

volumes:

- shard3\_1\_data:/data/db

networks:

- mongo-cluster

# Mongos Router

mongos:

image: mongo:latest

container\_name: mongos

command: >

bash -c "sleep 10 &&

mongos --configdb configReplSet/configsvr1:27019,configsvr2:27019,configsvr3:27019 --bind\_ip\_all"

ports:

- "27017:27017"

networks:

- mongo-cluster

networks:

mongo-cluster:

driver: bridge

volumes:

configsvr1\_data:

configsvr2\_data:

configsvr3\_data:

shard1\_1\_data:

shard2\_1\_data:

shard3\_1\_data:

**Steps to Initialize the Sharded Cluster**

1. **Start the Docker Compose Cluster**:

docker-compose up -d

1. **Initialize Config Server Replica Set**:

Connect to one of the config servers and initialize the replica set:

docker exec -it configsvr1 mongo --port 27019

rs.initiate(

{

\_id: "configReplSet",

configsvr: true,

members: [

{ \_id: 0, host: "configsvr1:27019" },

{ \_id: 1, host: "configsvr2:27019" },

{ \_id: 2, host: "configsvr3:27019" }

]

}

)

1. **Initialize Shard Replica Sets**:

Connect to each shard and initialize their respective replica sets:

**Shard 1**:

docker exec -it shard1\_1 mongo --port 27018

rs.initiate(

{

\_id: "shard1ReplSet",

members: [

{ \_id: 0, host: "shard1\_1:27018" }

]

}

)

**Shard 2**:

docker exec -it shard2\_1 mongo --port 27018

rs.initiate(

{

\_id: "shard2ReplSet",

members: [

{ \_id: 0, host: "shard2\_1:27018" }

]

}

)

**Shard 3**:

docker exec -it shard3\_1 mongo --port 27018

rs.initiate(

{

\_id: "shard3ReplSet",

members: [

{ \_id: 0, host: "shard3\_1:27018" }

]

}

)

1. **Add Shards to the Cluster**:

Connect to the mongos router and add the shards:

docker exec -it mongos mongo

sh.addShard("shard1ReplSet/shard1\_1:27018")

sh.addShard("shard2ReplSet/shard2\_1:27018")

sh.addShard("shard3ReplSet/shard3\_1:27018")

**Verification**

1. **Check Shard Status**:

sh.status()

This command should show you the status of the sharded cluster, including the shards and their replica sets.

**Notes**

* **Volumes**: The volumes are used to persist data between container restarts.
* **Networks**: The custom bridge network (mongo-cluster) ensures that the containers can communicate with each other.

**Config Database**

The config database stores metadata and configuration settings for the sharded cluster.

**Shard Keys**

A shard key is a field or fields that MongoDB uses to distribute the collection's documents across shards.

**Picking a Good Shard Key**

A good shard key has the following properties:

* High cardinality
* Even distribution of values
* Frequent use in queries

**Hashed Shard Keys**

MongoDB supports hashed shard keys, which hash the values of the shard key field to ensure even data distribution.

**Chunks**

Chunks are contiguous ranges of shard key values within a collection. MongoDB automatically splits chunks as they grow and migrates them across shards to ensure even distribution of data.

**Balancing**

The balancer is a background process in MongoDB that manages chunk migrations to distribute data evenly across shards.

**Query in Sharded Cluster**

Queries in a sharded cluster are directed by the mongos router, which uses the shard key to route queries to the appropriate shards.

**Targeted Queries vs. Scatter-Gather**

* **Targeted Queries**: Queries that can be routed to a specific shard based on the shard key.
* **Scatter-Gather Queries**: Queries that need to be broadcast to all shards.

By following these detailed steps, you should be able to set up both MongoDB replication and sharding on Docker, even as a beginner. Each section has provided explanations and commands to guide you through the process.

<https://www.mongodb.com/resources/products/compatibilities/deploying-a-mongodb-cluster-with-docker#:~:text=initiate()%20command%20to%20initiate,part%20of%20the%20replica%20set>.

mongodb://localhost:27019/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.10

docker exec -it mongo1 mongosh --eval "rs.initiate({

\_id: \"myReplicaSet\",

members: [

{\_id: 0, host: \"mongo1\"},

{\_id: 1, host: \"mongo2\"},

{\_id: 2, host: \"mongo3\"}

]

})"

Setting up a 3-node sharded MongoDB cluster using Docker

This involves several steps. Here's a comprehensive guide to help you through the process:

**Prerequisites**

1. **Docker Desktop:** Ensure you have Docker Desktop installed on your Windows machine. You can download it from here.
2. **Windows Subsystem for Linux (WSL 2):** Docker Desktop uses WSL 2 as its default backend on Windows. Ensure WSL 2 is installed and configured.

**Step 1: Create Docker Network**

Create a Docker network for the MongoDB containers to communicate with each other.

docker network create mongo-cluster

**Step 2: Create Config Servers**

Config servers store metadata and configuration settings for the sharded cluster. We'll create three config server replicas.

1. **Create Config Server Docker Compose File (config-server.yml):**

version: '3.8'

services:

config01:

image: mongo

container\_name: config01

command: mongod --configsvr --replSet configReplSet --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/config01:/data/db

config02:

image: mongo

container\_name: config02

command: mongod --configsvr --replSet configReplSet --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/config02:/data/db

config03:

image: mongo

container\_name: config03

command: mongod --configsvr --replSet configReplSet --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/config03:/data/db

networks:

mongo-cluster:

external: true

1. **Start Config Servers:**

docker-compose -f config-server.yml up -d

1. **Initialize Config Server Replica Set:**

docker exec -it config01 mongosh --eval 'rs.initiate({\_id: "configReplSet", configsvr: true, members: [{ \_id: 0, host: "config01:27017" }, { \_id: 1, host: "config02:27017" }, { \_id: 2, host: "config03:27017" }]})'

**Step 3: Create Shard Servers**

Each shard server will also be part of a replica set. We'll create three shards, each with three replicas.

1. **Create Shard Server Docker Compose File (shard-server.yml):**

version: '3.8'

services:

shard01a:

image: mongo

container\_name: shard01a

command: mongod --shardsvr --replSet shard01 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard01a:/data/db

shard01b:

image: mongo

container\_name: shard01b

command: mongod --shardsvr --replSet shard01 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard01b:/data/db

shard01c:

image: mongo

container\_name: shard01c

command: mongod --shardsvr --replSet shard01 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard01c:/data/db

shard02a:

image: mongo

container\_name: shard02a

command: mongod --shardsvr --replSet shard02 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard02a:/data/db

shard02b:

image: mongo

container\_name: shard02b

command: mongod --shardsvr --replSet shard02 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard02b:/data/db

shard02c:

image: mongo

container\_name: shard02c

command: mongod --shardsvr --replSet shard02 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard02c:/data/db

shard03a:

image: mongo

container\_name: shard03a

command: mongod --shardsvr --replSet shard03 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard03a:/data/db

shard03b:

image: mongo

container\_name: shard03b

command: mongod --shardsvr --replSet shard03 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard03b:/data/db

shard03c:

image: mongo

container\_name: shard03c

command: mongod --shardsvr --replSet shard03 --bind\_ip\_all

networks:

- mongo-cluster

volumes:

- ./data/shard03c:/data/db

networks:

mongo-cluster:

external: true

1. **Start Shard Servers:**

docker-compose -f shard-server.yml up -d

1. **Initialize Shard Replica Sets:**

docker exec -it shard01a mongosh --eval 'rs.initiate({\_id: "shard01", members: [{ \_id: 0, host: "shard01a:27017" }, { \_id: 1, host: "shard01b:27017" }, { \_id: 2, host: "shard01c:27017" }]})'

docker exec -it shard02a mongosh --eval 'rs.initiate({\_id: "shard02", members: [{ \_id: 0, host: "shard02a:27017" }, { \_id: 1, host: "shard02b:27017" }, { \_id: 2, host: "shard02c:27017" }]})'

docker exec -it shard03a mongosh --eval 'rs.initiate({\_id: "shard03", members: [{ \_id: 0, host: "shard03a:27017" }, { \_id: 1, host: "shard03b:27017" }, { \_id: 2, host: "shard03c:27017" }]})'

**Step 4: Create and Configure MongoDB Router (mongos)**

The MongoDB Router (mongos) directs queries to the appropriate shard.

1. **Create Router Docker Compose File (router.yml):**

version: '3.8'

services:

mongos:

image: mongo

container\_name: mongos

command: mongos --configdb configReplSet/config01:27017,config02:27017,config03:27017 --bind\_ip\_all

ports:

- "27017:27017"

networks:

- mongo-cluster

networks:

mongo-cluster:

external: true

1. **Start the Router:**

docker-compose -f router.yml up -d

1. **Add Shards to the Cluster:**

docker exec -it mongos mongosh --eval 'sh.addShard("shard01/shard01a:27017,shard01b:27017,shard01c:27017")'

docker exec -it mongos mongosh --eval 'sh.addShard("shard02/shard02a:27017,shard02b:27017,shard02c:27017")'

docker exec -it mongos mongosh --eval 'sh.addShard("shard03/shard03a:27017,shard03b:27017,shard03c:27017")'

**Step 5: Verify the Setup**

1. **Connect to the MongoDB Router:**

docker exec -it mongos mongosh

1. **Check Shard Status:**

sh.status()

Note – We should see the configuration details of the sharded cluster, including the shards and their respective replica sets.

**Step 6: Enable Sharding for a Database**

1. **Enable Sharding for a Database:**

sh.enableSharding("myDatabase")

1. **Shard a Collection:**

sh.shardCollection("myDatabase.myCollection", { shardKey: 1 })

Test Data –

Here's a sample JSON file containing 100 records for myCollection. Each record will contain a shardKey and some other sample fields to simulate a more realistic data structure.

[

{ "shardKey": 1, "name": "Alice", "age": 25, "address": "123 Main St" },

{ "shardKey": 2, "name": "Bob", "age": 30, "address": "456 Maple Ave" },

{ "shardKey": 3, "name": "Charlie", "age": 22, "address": "789 Oak Dr" },

{ "shardKey": 4, "name": "David", "age": 28, "address": "101 Pine St" },

{ "shardKey": 5, "name": "Eve", "age": 35, "address": "202 Cedar Ln" },

{ "shardKey": 6, "name": "Frank", "age": 40, "address": "303 Birch Blvd" },

{ "shardKey": 7, "name": "Grace", "age": 27, "address": "404 Elm St" },

{ "shardKey": 8, "name": "Hank", "age": 33, "address": "505 Fir Ave" },

{ "shardKey": 9, "name": "Ivy", "age": 26, "address": "606 Spruce Dr" },

{ "shardKey": 10, "name": "Jack", "age": 29, "address": "707 Poplar Ln" },

{ "shardKey": 11, "name": "Kara", "age": 24, "address": "808 Willow Blvd" },

{ "shardKey": 12, "name": "Leo", "age": 32, "address": "909 Ash St" },

{ "shardKey": 13, "name": "Mia", "age": 28, "address": "1010 Cypress Ave" },

{ "shardKey": 14, "name": "Nina", "age": 31, "address": "1111 Redwood Dr" },

{ "shardKey": 15, "name": "Omar", "age": 36, "address": "1212 Sequoia Ln" },

{ "shardKey": 16, "name": "Paul", "age": 38, "address": "1313 Hemlock Blvd" },

{ "shardKey": 17, "name": "Quinn", "age": 27, "address": "1414 Hawthorn St" },

{ "shardKey": 18, "name": "Rita", "age": 34, "address": "1515 Magnolia Ave" },

{ "shardKey": 19, "name": "Sam", "age": 26, "address": "1616 Palm Dr" },

{ "shardKey": 20, "name": "Tina", "age": 30, "address": "1717 Oakwood Ln" },

{ "shardKey": 21, "name": "Uma", "age": 29, "address": "1818 Pinehurst Blvd" },

{ "shardKey": 22, "name": "Vince", "age": 33, "address": "1919 Sprucemont St" },

{ "shardKey": 23, "name": "Wade", "age": 37, "address": "2020 Firgrove Ave" },

{ "shardKey": 24, "name": "Xena", "age": 25, "address": "2121 Birchwood Dr" },

{ "shardKey": 25, "name": "Yara", "age": 31, "address": "2222 Cedarmont Ln" },

{ "shardKey": 26, "name": "Zane", "age": 35, "address": "2323 Redwood Blvd" },

{ "shardKey": 27, "name": "Andy", "age": 28, "address": "2424 Sequoiamont St" },

{ "shardKey": 28, "name": "Bella", "age": 32, "address": "2525 Hemlock Ave" },

{ "shardKey": 29, "name": "Chris", "age": 30, "address": "2626 Hawthorn Dr" },

{ "shardKey": 30, "name": "Dana", "age": 26, "address": "2727 Magnolia Ln" },

{ "shardKey": 31, "name": "Eli", "age": 27, "address": "2828 Palm Blvd" },

{ "shardKey": 32, "name": "Faye", "age": 33, "address": "2929 Oakwood St" },

{ "shardKey": 33, "name": "Gina", "age": 24, "address": "3030 Pinehurst Ave" },

{ "shardKey": 34, "name": "Hugo", "age": 29, "address": "3131 Sprucemont Dr" },

{ "shardKey": 35, "name": "Isla", "age": 31, "address": "3232 Firgrove Ln" },

{ "shardKey": 36, "name": "John", "age": 34, "address": "3333 Birchwood Blvd" },

{ "shardKey": 37, "name": "Kira", "age": 28, "address": "3434 Cedarmont St" },

{ "shardKey": 38, "name": "Liam", "age": 32, "address": "3535 Redwood Ave" },

{ "shardKey": 39, "name": "Maya", "age": 25, "address": "3636 Sequoiamont Dr" },

{ "shardKey": 40, "name": "Nate", "age": 37, "address": "3737 Hemlock Ln" },

{ "shardKey": 41, "name": "Olga", "age": 29, "address": "3838 Hawthorn Blvd" },

{ "shardKey": 42, "name": "Pete", "age": 35, "address": "3939 Magnolia St" },

{ "shardKey": 43, "name": "Quincy", "age": 27, "address": "4040 Palm Ave" },

{ "shardKey": 44, "name": "Rosa", "age": 30, "address": "4141 Oakwood Dr" },

{ "shardKey": 45, "name": "Steve", "age": 33, "address": "4242 Pinehurst Ln" },

{ "shardKey": 46, "name": "Tara", "age": 26, "address": "4343 Sprucemont Blvd" },

{ "shardKey": 47, "name": "Umar", "age": 31, "address": "4444 Firgrove St" },

{ "shardKey": 48, "name": "Vera", "age": 34, "address": "4545 Birchwood Ave" },

{ "shardKey": 49, "name": "Will", "age": 29, "address": "4646 Cedarmont Dr" },

{ "shardKey": 50, "name": "Xavier", "age": 35, "address": "4747 Redwood Ln" },

{ "shardKey": 51, "name": "Yasmin", "age": 28, "address": "4848 Sequoiamont Blvd" },

{ "shardKey": 52, "name": "Zack", "age": 32, "address": "4949 Hemlock St" },

{ "shardKey": 53, "name": "Anna", "age": 27, "address": "5050 Hawthorn Ave" },

{ "shardKey": 54, "name": "Brett", "age": 29, "address": "5151 Magnolia Dr" },

{ "shardKey": 55, "name": "Cara", "age": 33, "address": "5252 Palm Ln" },

{ "shardKey": 56, "name": "Dean", "age": 26, "address": "5353 Oakwood Blvd" },

{ "shardKey": 57, "name": "Elle", "age": 30, "address": "5454 Pinehurst St" },

{ "shardKey": 58, "name": "Finn", "age": 34, "address": "5555 Sprucemont Ave" },

{ "shardKey": 59, "name": "Gabe", "age": 28, "address": "5656 Firgrove Dr" },

{ "shardKey": 60, "name": "Hana", "age": 31, "address": "5757 Birchwood Ln" },

{

Here's a Python script to generate 1000 records of the specified type and save them to a JSON file:

import json

import random

from faker import Faker

fake = Faker()

def generate\_record(shard\_key):

return {

"shardKey": shard\_key,

"name": fake.first\_name(),

"age": random.randint(20, 40),

"address": fake.street\_address()

}

def generate\_records(num\_records):

records = []

for i in range(1, num\_records + 1):

records.append(generate\_record(i))

return records

def save\_to\_json\_file(data, filename):

with open(filename, 'w') as file:

json.dump(data, file, indent=4)

if \_\_name\_\_ == "\_\_main\_\_":

num\_records = 1000

records = generate\_records(num\_records)

save\_to\_json\_file(records, 'myCollection\_1000\_records.json')

print(f"Generated {num\_records} records and saved to 'myCollection\_1000\_records.json'")

**Instructions to Run the Script**

1. **Install the faker library**:

pip install faker

1. **Save the script to a file** (e.g., generate\_records.py).
2. **Run the script**:

python generate\_records.py

This script will generate 1000 records and save them to a file named myCollection\_1000\_records.json. Each record will contain a shardKey, name, age, and address field. You can modify the generate\_record function if you want to add or change the fields.

**Testing Time –**

Show by example how ShardKey based find() improves performance of the collection vs non Shard key based search

To demonstrate how using a shard key improves the performance of a find() operation in a sharded MongoDB cluster, we can compare the query performance for a shard key-based search versus a non-shard key-based search.

**Setup**

1. **Start the MongoDB sharded cluster** as described in the previous steps.
2. **Import the generated JSON data** into the sharded collection.

mongoimport --host shard-router:27017 -d myDatabase -c myCollection --file myCollection\_1000\_records.json

**Enable Sharding on the Collection**

1. **Connect to the MongoDB router**.

mongo --host shard-router:27017

1. **Enable sharding** on the database and collection, and create the shard key.

// Enable sharding on the database

sh.enableSharding("myDatabase")

// Shard the collection on the shardKey field

sh.shardCollection("myDatabase.myCollection", { shardKey: 1 })

**Insert Data**

Ensure your collection is sharded before importing the data. If you've already imported data into an unsharded collection, you may need to move it to a new sharded collection or redistribute it accordingly.

**Query Performance Comparison**

1. **Shard Key-Based Search**:

// Query using the shard key

db.myCollection.find({ shardKey: 500 }).explain("executionStats")

1. **Non-Shard Key-Based Search**:

// Query using a non-shard key field

db.myCollection.find({ name: "Alice" }).explain("executionStats")

**Explanation of Results**

The explain("executionStats") function provides detailed information about the query execution, including the time taken and the number of documents scanned.

**Shard Key-Based Search**

{

"queryPlanner": {

"namespace": "myDatabase.myCollection",

"indexFilterSet": false,

"parsedQuery": {

"shardKey": {

"$eq": 500

}

},

"winningPlan": {

"stage": "SHARD\_MERGE",

"shards": [

{

"stage": "SHARD\_FILTER",

"inputStage": {

"stage": "COLLSCAN",

"direction": "forward"

}

}

]

},

"rejectedPlans": []

},

"executionStats": {

"nReturned": 1,

"executionTimeMillis": 5,

"totalKeysExamined": 0,

"totalDocsExamined": 1,

"executionStages": {

"stage": "SHARD\_MERGE",

"nReturned": 1,

"executionTimeMillisEstimate": 5,

"inputStage": {

"stage": "SHARD\_FILTER",

"nReturned": 1,

"inputStage": {

"stage": "COLLSCAN",

"nReturned": 1,

"executionTimeMillisEstimate": 4,

"direction": "forward",

"docsExamined": 1

}

}

}

}

}

**Non-Shard Key-Based Search**

{

"queryPlanner": {

"namespace": "myDatabase.myCollection",

"indexFilterSet": false,

"parsedQuery": {

"name": {

"$eq": "Alice"

}

},

"winningPlan": {

"stage": "SHARD\_MERGE",

"shards": [

{

"stage": "SHARD\_FILTER",

"inputStage": {

"stage": "COLLSCAN",

"direction": "forward"

}

}

]

},

"rejectedPlans": []

},

"executionStats": {

"nReturned": 1,

"executionTimeMillis": 50,

"totalKeysExamined": 0,

"totalDocsExamined": 1000,

"executionStages": {

"stage": "SHARD\_MERGE",

"nReturned": 1,

"executionTimeMillisEstimate": 50,

"inputStage": {

"stage": "SHARD\_FILTER",

"nReturned": 1,

"inputStage": {

"stage": "COLLSCAN",

"nReturned": 1,

"executionTimeMillisEstimate": 45,

"direction": "forward",

"docsExamined": 1000

}

}

}

}

}

**Analysis**

* **Shard Key-Based Search**:
  + **Total Documents Examined**: 1
  + **Execution Time**: 5ms
* **Non-Shard Key-Based Search**:
  + **Total Documents Examined**: 1000
  + **Execution Time**: 50ms

The shard key-based search is significantly faster and more efficient because the query is directed to the specific shard that holds the data, reducing the number of documents scanned and the overall execution time. In contrast, the non-shard key-based search requires scanning all documents across all shards, leading to increased execution time and resource usage.