**The Floow: Platform Engineer Challenge**

**Solution**

The solution will involve storage of large amount of data so it would be necessary to create a sharded cluster in MongoDB to distribute the workload easily. Sharding is a technique to split large amount of data across multiple server instances. If data grows exponentially, Sharding helps in dividing the whole data set into smaller parts and distribute them across large number of servers (Shards).

**Prerequisites**

MongoDB server 4.2.0

**Steps to create sharded cluster**

**Create the Config Server Replica Set (CSRS)**

1. Start each member of the CSRS

Create a directory of C:\data\config

Open a command prompt and run following command

mongod --configsvr --replSet rst --dbpath "C:\data\config" --bind\_ip localhost

It will automatically connect on port 27019 for config server.

1. Connect to the config server(s)

Open another prompt and run following command to connect to config server

mongo --host localhost --port 27019

1. Initiate the replica set

Run the rs.initiate() method from mongo instance.

rs.initiate({\_id:"rst",configsvr:true,version:1,members:[{\_id:0,host:"localhost:27019"}]})

This sets up config server replica set.

**Create the Shard Replica Sets**

1. Start each member of the shard replica set

Open two command prompt and run following commands

mongod --shardsvr --replSet srt --dbpath "C:\data\shard1" --bind\_ip localhost --port 27018

mongod --shardsvr  --replSet srt  --dbpath "C:\data\shard2" --bind\_ip localhost  --port 27020

1. Connect to one member of the shard replica set

mongo --host localhost --port 27018

1. Initiate the replica set

rs.initiate({\_id:"srt",members:[{\_id:0,host:"localhost:27018"},{\_id:1,host:"localhost:27020"}]})

**Connect a mongos to the Sharded Cluster**

1. Connect a mongos to the cluster

Open a command prompt and run

mongos --configdb rst/localhost:27019 --bind\_ip localhost

1. Connect a mongo shell to the mongos

mongo --host localhost --port 27017

**Add Shards to the Cluster**

Use the sh.addShard() method to add each shard to the cluster. If the shard is a replica set, specify the name of the replica set and specify a member of the set.

sh.addShard("srt/localhost:27018")

sh.addShard("srt/localhost:27020")

Check status of shards by running below commands

sh.status()

**Enable Sharding for a database**

Sh.enableSharding(“wordsdb”)

**Shard a Collection**

For better functionality, capability and performance of the sharding strategy, it’s important to choose the right shard key. The shard key is an indexed field that exists in every document in the collection.

Database Name: wordsdb

Collection Name: wordscount

E.g. {“word”:”the”,”count”:20}

In our case, word will act as a unique indexed shard key. (Could it be useful if word field is hashed based shard key?).

Use wordsdb

Db.createCollection(“wordscount”)

db.wordscount.createIndex({"word":1},{unique,true})

**To shard a collection use following command**

Sh.shardCollection(“wordsdb.wordscount”,{word:1})

Please note above steps of database/collection creation, unique index creation and sharding a collection could be done within the Java application.

**Java application**

The solution involves reading and handling large amount of data with lots of processing. It would be necessary to use multi threading. In the solution one thread will read large file of text line by line and will spawn threads. Each thread will extract words count out of certain chunks of lines (i.e. 1000 to 50000 lines). Then, it will put the words count in the MongoDB collection using bulk write operation.

1. Accept the command line arguments
2. Create a ThreadPoolExecutor of size 20
3. Get a single instance of MongoDB Client using provided hostnames (multiple hosts for sharded cluster) to be shared by all threads.
4. Initialise MongoDB database, sharded collection and create unique indexes (if not done already)
5. Main thread reads file line by line and builds an array list of certain size (i.e. 1000 to 50000 lines)
6. Main thread creates a thread to process each array list of lines and put it in the thread pool executor for execution.
7. Each thread extracts words and word count from an array list of lines.
8. Each thread prepares an array list for MongoDB unordered bulk write operation which
   1. Inserts “word” and “count” in the MongoDB collection if does not exist OR
   2. Updates “count” by incrementing new count of words for a given word

E.g. **new** UpdateOneModel<>(**new** Document("word",entry.getKey())

, **new** Document("$inc", **new** Document("count",entry.getValue()))

, **new** UpdateOptions().upsert(**true**))

1. Each thread executes a bulk write operation to insert or update word count in the MongoDB collection within the sharded cluster.
2. Once all threads finish execution and all words count are stored in the MongoDB collection, print most common and least common words on the screen.