

Part 2

Horizontal Fragmentation

We have successfully completed the implementation of horizontal fragmentation in our database, a process that organizes our data more efficiently. This method involves dividing our music playlists into separate groups based on the first letter of each playlist name. For example, all playlists starting with 'A' are grouped together, then those starting with 'B', and so on. This approach makes it easier and faster to find and manage specific playlists, especially as our collection grows. During this process, we also created a special category for playlists that begin with non-letter characters like numbers or emojis. This ensures that every playlist is sorted appropriately.

```
Category 'b': ['B0DYRXLL B3AT5']
Category 'c': ['CHORE']
Category 'd': ['dissociation']
Category 'f': ['funeral']
Category 'm': ['me♥']
Category 'p': ['promise', 'pew pew']
Category 's': ['sasasa', 'summer school']
Category 'u': ['uh uh uh uh']
Category 'w': ['wah!']
Category 'others': ['💖 songs to listen to when im yeah 💖']
```

We've taken a pretty cool leap with our MongoDB setup, diving into ranged sharding based on letters for our `playlists` collection. The idea was to distribute the data across three shards, roughly aligning with different groups of alphabet letters. This wasn't the most straightforward task, given MongoDB's nature of not directly supporting sharding by specific alphabet ranges into a fixed number of shards. However, we worked around this by creatively using shard keys. We added a field to each document representing the first letter of the playlist's name, then sharded based on this field. While MongoDB handled the distribution automatically, our approach nudged it towards our goal.

```
mkdir -p ~/mongodb/configdb

mongod --configsvr --replSet configReplSet --dbpath ~/mongodb/configdb --port 27019 --bind_ip localhost

mongo --host localhost --port 27019
```

```

rs.initiate(
{
  _id: "configReplSet",
  configsvr: true,
  members: [
    { _id : 0, host : "localhost:27019" }
    // Add more members if needed
  ]
}
)

mkdir -p ~/mongodb/shard1
mkdir -p ~/mongodb/shard2
mkdir -p ~/mongodb/shard3

mongod --shardsvr --replSet shardReplSet1 --dbpath ~/mongodb/shard1 --port 27018 --bind_ip localhost
mongod --shardsvr --replSet shardReplSet2 --dbpath ~/mongodb/shard2 --port 27020 --bind_ip localhost
mongod --shardsvr --replSet shardReplSet3 --dbpath ~/mongodb/shard3 --port 27021 --bind_ip localhost

// For shard 1
rs.initiate(
{
  _id: "shardReplSet1",
  members: [
    { _id : 0, host : "localhost:27018" }
  ]
}
)

// For shard 2
rs.initiate(
{
  _id: "shardReplSet2",
  members: [
    { _id : 0, host : "localhost:27020" }
  ]
}
)

// For shard 3
rs.initiate(
{
  _id: "shardReplSet3",
  members: [
    { _id : 0, host : "localhost:27021" }
  ]
}
)

mongos --configdb configReplSet/localhost:27019 --bind_ip localhost

mongosh --host localhost --port 27017

```

```
sh.addShard("shardReplSet1/localhost:27018")
sh.addShard("shardReplSet2/localhost:27020")
sh.addShard("shardReplSet3/localhost:27021")
```

Number of shards: 3

Shard ID: shardReplSet1, Host: shardReplSet1/localhost:27018

Shard ID: shardReplSet2, Host: shardReplSet2/localhost:27020

Shard ID: shardReplSet3, Host: shardReplSet3/localhost:27021

```
shardingVersion
{ _id: 1, clusterId: ObjectId('656424d36b26956e5271f92b') }
---
shards
[
  {
    _id: 'shardReplSet1',
    host: 'shardReplSet1/localhost:27018',
    state: 1,
    topologyTime: Timestamp({ t: 1701062227, i: 2 })
  },
  {
    _id: 'shardReplSet2',
    host: 'shardReplSet2/localhost:27020',
    state: 1,
    topologyTime: Timestamp({ t: 1701062227, i: 6 })
  },
  {
    _id: 'shardReplSet3',
    host: 'shardReplSet3/localhost:27021',
    state: 1,
    topologyTime: Timestamp({ t: 1701062227, i: 19 })
  }
]
---
active mongoses
[ { '7.0.2': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
---
databases
[
```

```

{
  database: { _id: 'config', primary: 'config', partitioned: true },
  collections: {
    'config.system.sessions': {
      shardKey: { _id: 1 },
      unique: false,
      balancing: true,
      chunkMetadata: [ { shard: 'shardReplSet1', nChunks: 1 } ],
      chunks: [
        { min: { _id: MinKey() }, max: { _id: MaxKey() }, 'on shard': 'shardReplSet
1', 'last modified': Timestamp({ t: 1, i: 0 }) }
      ],
      tags: []
    }
  },
},
{
  database: {
    _id: 'streaming_dds',
    primary: 'shardReplSet3',
    partitioned: false,
    version: {
      uuid: UUID('51696630-9574-4b74-bf81-691a130bfbd9'),
      timestamp: Timestamp({ t: 1701062516, i: 1 }),
      lastMod: 1
    }
  },
  collections: {
    'streaming_dds.playlists': {
      shardKey: { first_char: 1 },
      unique: false,
      balancing: true,
      chunkMetadata: [ { shard: 'shardReplSet3', nChunks: 1 } ],
      chunks: [
        { min: { first_char: MinKey() }, max: { first_char: MaxKey() }, 'on shard':
'shardReplSet3', 'last modified': Timestamp({ t: 1, i: 0 }) }
      ],
      tags: []
    }
  }
}
]

```

Vertical Fragmentation

Vertical fragmentation, splitting a table into smaller ones based on columns, is not supported in MongoDB due to its design as a NoSQL database. Unlike traditional relational databases, MongoDB uses flexible, JSON-like documents instead of fixed tables and rows. This allows for efficient data management without needing to split data across multiple tables, making vertical fragmentation unnecessary in MongoDB's document-oriented structure.

Replica Sets










```
docker network create mongoCluster

docker run -d --rm -p 27031:27017 --name mongo1 --network mongoCluster mongo:5 mongod
--replSet myReplicaSet --bind_ip localhost,mongo1

docker run -d --rm -p 27032:27017 --name mongo2 --network mongoCluster mongo:5 mongod
--replSet myReplicaSet --bind_ip localhost,mongo2

docker run -d --rm -p 27033:27017 --name mongo3 --network mongoCluster mongo:5 mongod
--replSet myReplicaSet --bind_ip localhost,mongo3

docker exec -it mongo1 mongosh --eval "rs.initiate({
  _id: \"myReplicaSet\",
  members: [
    { _id: 0, host: \"mongo1\" },
    { _id: 1, host: \"mongo2\" },
    { _id: 2, host: \"mongo3\" }
  ]
})"
```

<input type="checkbox"/>		mongo1 9aacc79cf9e5 	mongo:5	Running	27031:27017 
<input type="checkbox"/>		mongo2 24c75e7ea518 	mongo:5	Running	27032:27017 
<input type="checkbox"/>		mongo3 13caf853982f 	mongo:5	Running	27033:27017 

```
docker exec -it mongo1 mongosh --eval "rs.status()";
```

```
{ ok: 1 }
→ Streaming-DDS git:(main) x docker exec -it mongo1 mongosh --eval "rs.status()"
Current Mongosh Log ID: 655fd282e1c7333a644a22a2
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeout
MS=2000&appName=mongosh+2.0.2
Using MongoDB: 5.0.22
Using Mongosh: 2.0.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/
```

```

{
  set: 'myReplicaSet',
  date: ISODate("2023-11-23T22:30:26.878Z"),
  myState: 1,
  term: Long("1"),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long("2000"),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
    lastCommittedWallTime: ISODate("2023-11-23T22:30:19.022Z"),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
    appliedOpTime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
    durableOpTime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
    lastAppliedWallTime: ISODate("2023-11-23T22:30:19.022Z"),
    lastDurableWallTime: ISODate("2023-11-23T22:30:19.022Z")
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1700778606, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate("2023-11-23T22:30:17.501Z"),
    electionTerm: Long("1"),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1700778606, i: 1 }), t: Long("-1") },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1700778606, i: 1 }), t: Long("-1") },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long("10000"),
    numCatchUpOps: Long("0"),
    newTermStartDate: ISODate("2023-11-23T22:30:17.537Z"),
    wMajorityWriteAvailabilityDate: ISODate("2023-11-23T22:30:18.982Z")
  },
  members: [
    {
      _id: 0,
      name: 'mongo1:27017',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 43,
      optime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
      optimeDate: ISODate("2023-11-23T22:30:19.000Z"),
      lastAppliedWallTime: ISODate("2023-11-23T22:30:19.022Z"),
      lastDurableWallTime: ISODate("2023-11-23T22:30:19.022Z"),
      syncSourceHost: '',
      syncSourceId: -1,
      infoMessage: '',
      electionTime: Timestamp({ t: 1700778617, i: 1 }),
      electionDate: ISODate("2023-11-23T22:30:17.000Z"),
      configVersion: 1,
      configTerm: 1,

```

```

self: true,
lastHeartbeatMessage: ''
},
{
  _id: 1,
  name: 'mongo2:27017',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 20,
  optime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
  optimeDurable: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
  optimeDate: ISODate("2023-11-23T22:30:19.000Z"),
  optimeDurableDate: ISODate("2023-11-23T22:30:19.000Z"),
  lastAppliedWallTime: ISODate("2023-11-23T22:30:19.022Z"),
  lastDurableWallTime: ISODate("2023-11-23T22:30:19.022Z"),
  lastHeartbeat: ISODate("2023-11-23T22:30:25.525Z"),
  lastHeartbeatRecv: ISODate("2023-11-23T22:30:25.028Z"),
  pingMs: Long("0"),
  lastHeartbeatMessage: '',
  syncSourceHost: 'mongo1:27017',
  syncSourceId: 0,
  infoMessage: '',
  configVersion: 1,
  configTerm: 1
},
{
  _id: 2,
  name: 'mongo3:27017',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 20,
  optime: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
  optimeDurable: { ts: Timestamp({ t: 1700778619, i: 5 }), t: Long("1") },
  optimeDate: ISODate("2023-11-23T22:30:19.000Z"),
  optimeDurableDate: ISODate("2023-11-23T22:30:19.000Z"),
  lastAppliedWallTime: ISODate("2023-11-23T22:30:19.022Z"),
  lastDurableWallTime: ISODate("2023-11-23T22:30:19.022Z"),
  lastHeartbeat: ISODate("2023-11-23T22:30:25.525Z"),
  lastHeartbeatRecv: ISODate("2023-11-23T22:30:25.028Z"),
  pingMs: Long("0"),
  lastHeartbeatMessage: '',
  syncSourceHost: 'mongo1:27017',
  syncSourceId: 0,
  infoMessage: '',
  configVersion: 1,
  configTerm: 1
}
],
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1700778619, i: 5 }),
  signature: {
    hash: Binary.createFromBase64("AAAAAAAAAAAAAAAAAAAAAAAAAAAA=", 0),
    keyId: Long("0")
  }
}

```

```
}  
},  
operationTime: Timestamp({ t: 1700778619, i: 5 })  
}
```

We have successfully set up MongoDB replica sets, improving our database's reliability and backup capabilities. This setup includes several MongoDB servers working together. The main server handles all the data changes, while the others keep copies of this data. This approach ensures that our data is safe and always available, even if one server has problems. We carefully configured each server and tested the entire system to ensure it works well together. This successful setup shows our commitment to a strong and reliable data management system.

The replica set also allows for automatic switching to a backup server if the main one fails. This means our data remains accessible with minimal interruption. We regularly check the system's performance to quickly fix any issues.