

MongoDB Sharding Documentation

1. Overview

This documentation outlines the steps taken to set up MongoDB sharding, create a sharded collection, insert data, and perform queries. The process involves setting up a MongoDB cluster using Docker Compose, enabling sharding on a database, creating a sharded collection, inserting data, and querying it.

2. Setup MongoDB Cluster with Docker Compose

First, a MongoDB cluster is set up using Docker Compose. This involves the following steps:

- Create MongoDB Instances: Define MongoDB instances for shard servers, config servers, and mongos routers in the Docker Compose file.
- Execute Docker Compose: Run Docker Compose to create and start MongoDB instances.

```
version: '3'
services:
  mongo-shard1-1:
    image: mongo:5
    ports:
      - "27101:27017"
    command: mongod --shardsvr --replSet mongo-shard1-rs --port 27017
    --bind_ip_all
    networks:
      - mongodb-network

  mongo-shard1-2:
    image: mongo:5
    command: mongod --shardsvr --replSet mongo-shard1-rs --port 27017
    --bind_ip_all
    networks:
      - mongodb-network

  mongo-shard1-3:
    image: mongo:5
    command: mongod --shardsvr --replSet mongo-shard1-rs --port 27017
    --bind_ip_all
    networks:
      - mongodb-network

  mongo-shard2-1:
```

```
image: mongo:5
ports:
  - "27201:27017"
command: mongod --shardsvr --replSet mongo-shard2-rs --port 27017
--bind_ip_all
networks:
  - mongodb-network

mongo-shard2-2:
image: mongo:5
command: mongod --shardsvr --replSet mongo-shard2-rs --port 27017
--bind_ip_all
networks:
  - mongodb-network

mongo-shard2-3:
image: mongo:5
command: mongod --shardsvr --replSet mongo-shard2-rs --port 27017
--bind_ip_all
networks:
  - mongodb-network

mongo-shard3-1:
image: mongo:5
ports:
  - "27301:27017"
command: mongod --shardsvr --replSet mongo-shard3-rs --port 27017
--bind_ip_all
networks:
  - mongodb-network

mongo-shard3-2:
image: mongo:5
command: mongod --shardsvr --replSet mongo-shard3-rs --port 27017
--bind_ip_all
networks:
  - mongodb-network

mongo-shard3-3:
image: mongo:5
command: mongod --shardsvr --replSet mongo-shard3-rs --port 27017
--bind_ip_all
networks:
```

```
- mongodb-network

mongo-config-server-1:
  image: mongo:5
  ports:
    - "27019:27017"
  command: mongod --configsvr --replSet mongo-config-server-rs --port
27017 --bind_ip_all
  networks:
    - mongodb-network

mongo-config-server-2:
  image: mongo:5
  command: mongod --configsvr --replSet mongo-config-server-rs --port
27017 --bind_ip_all
  networks:
    - mongodb-network

mongo-config-server-3:
  image: mongo:5
  command: mongod --configsvr --replSet mongo-config-server-rs --port
27017 --bind_ip_all
  networks:
    - mongodb-network

mongos-router:
  image: mongo:5
  ports:
    - "27017:27017"
  command: mongos --configdb
mongo-config-server-rs/mongo-config-server-1:27017,mongo-config-server-2:27
017,mongo-config-server-3:27017 --bind_ip_all
  depends_on:
    - mongo-config-server-1
    - mongo-config-server-2
    - mongo-config-server-3
  networks:
    - mongodb-network

networks:
  mongodb-network:
    driver: bridge
```

```
zamlamb@ZamLaptop:~/mongodb$ vim docker-compose.yaml
zamlamb@ZamLaptop:~/mongodb$ docker-compose -up -d
```

Then connect the components to one another:

In the provided commands, several actions are being performed to set up a MongoDB sharded cluster. Let's break down each command and explain what it does:

1. MongoDB Shard Initialization

```
docker exec -it mongodb_mongo-shard1-1_1 mongosh --eval 'rs.initiate({_id:
"mongo-shard1-rs", members: [{_id: 0, host:
"mongodb_mongo-shard1-1_1:27017"}, {_id: 1, host:
"mongodb_mongo-shard1-2_1:27017"}, {_id: 2, host:
"mongodb_mongo-shard1-3_1:27017"}]})'
```

- This command initializes the first shard replica set (`mongo-shard1-rs`) by executing the `rs.initiate()` function.
- It specifies three members (`members`) for the replica set, each with an `_id` and `host` indicating the Docker container and port where the MongoDB instances are running.

This is also done for the other sets

2. Configuration Server Initialization

```
docker exec -it mongodb_mongo-config-server-1_1 mongosh --eval
'rs.initiate({_id: "mongo-config-server-rs", configsvr: true, members:
[ {_id: 0, host: "mongodb_mongo-config-server-1_1:27017"}, {_id: 1, host:
"mongodb_mongo-config-server-2_1:27017"}, {_id: 2, host:
"mongodb_mongo-config-server-3_1:27017"} ]})'
```

- This command initializes the configuration server replica set (`mongo-config-server-rs`) with the `configsvr: true` option, indicating that these servers will store cluster metadata.
- Similar to the shard initialization, it specifies three members for the replica set.

3. Adding Shards to the Router

```
docker exec -it mongodb_mongos-router-1_1 mongosh --eval  
'sh.addShard("mongo-shard1-rs/mongodb_mongo-shard1-1_1:27017,mongodb_mongo-  
shard1-2_1:27017,mongodb_mongo-shard1-3_1:27017")'
```

- This command adds the first shard (`mongo-shard1-rs`) to the mongos router. It specifies the hosts of the shard members.
- Similar commands are used to add the second and third shards (`mongo-shard2-rs` and `mongo-shard3-rs`) to the mongos router.

Overall, these commands initialize shard replica sets, configuration server replica set, and add shards to the mongos router, effectively setting up a MongoDB sharded cluster.

3. Insert Data from SQL Query

```
SELECT  
  t.id AS treasure_id,  
  t.difficulty,  
  t.terrain,  
  c.city_name,  
  co.name AS country_name,  
  co.code3 AS country_code,  
  s.container_size,  
  s.description AS stage_description,  
  s.latitude AS stage_latitude,  
  s.longitude AS stage_longitude,  
  s.sequence_number,  
  s.type AS stage_type,  
  s.visibility  
FROM  
  treasure AS t  
    INNER JOIN  
  city AS c ON t.city_city_id = c.city_id  
    INNER JOIN  
  country AS co ON c.country_code = co.code  
    INNER JOIN  
  treasure_stages AS ts ON t.id = ts.treasure_id  
    INNER JOIN  
  stage AS s ON ts.stages_id = s.id
```

After the MongoDB cluster is set up, data is inserted from a SQL query output CSV file ('result2.csv').

	treasure_id (UUID)	difficulty	terrain	city_name	country_name
1	4a519a26-47c2-43f1-a570-430c3dda0c58	1	2 São Sebastião do Maranhão		Brazil
2	60feda0e-4881-4a1a-abad-75e34c300ea4	4	0 Chak 459 E.B.(Reformatory Farm)		Pakistan
3	b5d17859-b785-4f66-9bcb-29d238be8109	0	2 Varnell		United States of America
4	3269930a-eef1-4ff1-8782-35b393cca557	0	1 Santa Cruz Do Bolo		Spain
5	934ba90a-2da1-498d-9218-83fe6778c110	2	2 Rayamadharam		India
6	934ba90a-2da1-498d-9218-83fe6778c110	2	2 Rayamadharam		India
7	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
8	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
9	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
10	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
11	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
12	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
13	e43f6a9c-1a0b-4dfd-95e4-385400bb3e62	4	1 Nur Pur Machiwala		Pakistan
14	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
15	e43f6a9c-1a0b-4dfd-95e4-385400bb3e62	4	1 Nur Pur Machiwala		Pakistan
16	45e20496-c47f-4bea-90f6-085c1c798904	2	1 Paial		Brazil
17	934ba90a-2da1-498d-9218-83fe6778c110	2	2 Rayamadharam		India
18	934ba90a-2da1-498d-9218-83fe6778c110	2	2 Rayamadharam		India
19	934ba90a-2da1-498d-9218-83fe6778c110	2	2 Rayamadharam		India

Export Data

Extractor:
CSV

☐ Transpose

☒ Add column header

☒ Add row header

Output file:
E:\downloads\Result_2.csv

Export preview:

1	#,treasure_id,difficulty,terrain,city_name,countr
2	1,4a519a26-47c2-43f1-a570-430c3dda0c58,1,2,São Se
3	2,60feda0e-4881-4a1a-abad-75e34c300ea4,4,0,Chak 4
4	3,b5d17859-b785-4f66-9bcb-29d238be8109,0,2,Varne1
5	4,3269930a-eef1-4ff1-8782-35b393cca557,0,1,Santa
6	5,934ba90a-2da1-498d-9218-83fe6778c110,2,2,Rayama
7	6,934ba90a-2da1-498d-9218-83fe6778c110,2,2,Rayama
8	7,45e20496-c47f-4bea-90f6-085c1c798904,2,1,Paial,
9	8,45e20496-c47f-4bea-90f6-085c1c798904,2,1,Paial,
10	9,45e20496-c47f-4bea-90f6-085c1c798904,2,1,Paial,
11	10,45e20496-c47f-4bea-90f6-085c1c798904,2,1,Paial
12	

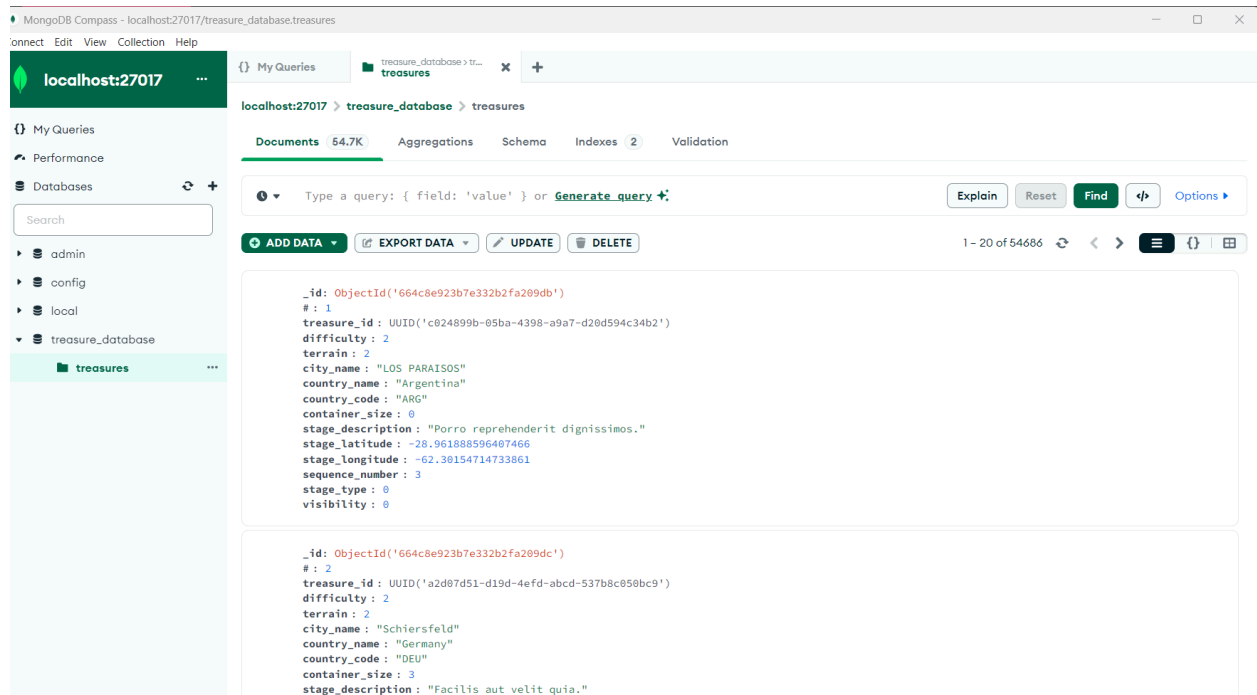
?

Export to File

Cancel

Copy to Clipboard

The data is inserted into MongoDB using MongoDB Compass or similar tool by connecting to `mongodb://localhost:27017` where we create the treasure_database then create the treasures collections. Below is the output of the inserted data as documents:



4. Enable Sharding

Sharding is enabled on the target database (`treasure_database`) using `mongosh` or through docker. This is done using the following command:

```
sh.enableSharding("treasure_database")
```

5. Create Index

An index is created on the `city_name` field of the `treasures` collection. This index is essential for efficient querying and sharding based on city names. The command used to create the index is:

```
use treasure_database
db.treasures.createIndex({ "city_name": 1 })
```

6. Shard the Collection

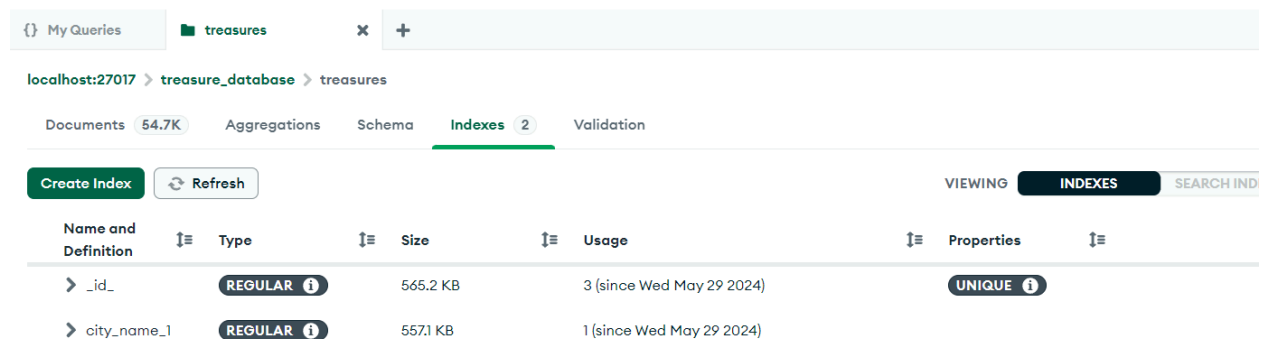
The `treasures` collection is sharded based on the `city_name` field. This ensures that data is distributed across shards based on city names, enabling efficient querying. The shard key used is `{ "city_name": 1 }`, indicating ascending order. The command to shard the collection is:

```
sh.shardCollection("treasure_database.treasures", { "city_name": 1 })
```

7. Query Data

Data is queried from the sharded collection to verify successful sharding and querying. The query used is to find treasures in the city of "Dörtağaç". The query is executed using the following command:

```
use treasure_database
db.treasures.find({ "city_name": /Dörtağaç/i }).pretty()
```



The screenshot shows the MongoDB Compass interface for the 'treasures' collection. The 'Indexes' tab is selected, showing two indexes:

Name and Definition	Type	Size	Usage	Properties
> _id_	REGULAR ⓘ	565.2 KB	3 (since Wed May 29 2024)	UNIQUE ⓘ
> city_name_1	REGULAR ⓘ	557.1 KB	1 (since Wed May 29 2024)	

8. Check Sharding Status

To ensure that sharding is set up correctly and to view the components and shards present, the `sh.status()` command is used on the mongos router and see that the shard key is also in use

```
docker exec -it mongodb_mongos-router-1_1 mongosh --eval 'sh.status()'
```

```
shardingVersion
{ _id: 1, clusterId: ObjectId('664c822e540d3f05ebe9c7bd') }
```

```
---
shards
[
  {
    _id: 'mongo-shard1-rs',
    host:
'mongo-shard1-rs/mongodb_mongo-shard1-1_1:27017,mongodb_mongo-shard1-2_1:27
017,mongodb_mongo-shard1-3_1:27017',
    state: 1,
    topologyTime: Timestamp({ t: 1716290109, i: 1 })
  },
  {
    _id: 'mongo-shard2-rs',
    host:
'mongo-shard2-rs/mongodb_mongo-shard2-1_1:27017,mongodb_mongo-shard2-2_1:27
017,mongodb_mongo-shard2-3_1:27017',
    state: 1,
    topologyTime: Timestamp({ t: 1716290113, i: 2 })
  },
  {
    _id: 'mongo-shard3-rs',
    host:
'mongo-shard3-rs/mongodb_mongo-shard3-1_1:27017,mongodb_mongo-shard3-2_1:27
017,mongodb_mongo-shard3-3_1:27017',
    state: 1,
    topologyTime: Timestamp({ t: 1716290116, i: 2 })
  }
]
---
active mongoses
[ { '5.0.26': 2 } ]
---
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: 'mongo-shard1-rs', nChunks: 342 },
          { shard: 'mongo-shard2-rs', nChunks: 341 },
          { shard: 'mongo-shard3-rs', nChunks: 341 }
        ],
        chunks: [
          'too many chunks to print, use verbose if you want to force
print'
        ],
        tags: []
      }
    }
  },
  {
    database: {
      _id: 'test',
      primary: 'mongo-shard1-rs',
      partitioned: false,
      version: {
        uuid: UUID('2975478d-560c-414b-a386-10b5e87dd68d'),
        timestamp: Timestamp({ t: 1716406088, i: 1 }),
        lastMod: 1
      }
    },
    collections: {}
  },
  {
    database: {
      _id: 'treasure_database',
      primary: 'mongo-shard1-rs',
      partitioned: true,
      version: {
        uuid: UUID('fbb2a25e-7a23-4cbc-8ffc-e5688877443b'),
        timestamp: Timestamp({ t: 1716406551, i: 1 }),

```

```

        lastMod: 1
      }
    },
    collections: {
      'treasure_database.treasures': {
        shardKey: { city_name: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'mongo-shard1-rs', nChunks: 1 } ],
        chunks: [
          { min: { city_name: MinKey() }, max: { city_name: MaxKey() }, 'on
shard': 'mongo-shard1-rs', 'last modified': Timestamp({ t: 1, i: 0 }) }
        ],
        tags: []
      }
    }
  }
]

```

Additionally, the `.explain("executionStats")` method is used to analyze the query execution statistics.

```
treasure_database> db.treasures.find({ "city_name": /Dörtağaç/i
}).pretty().explain("executionStats")
```

```

[
  {
    _id: ObjectId('664c8e923b7e332b2fa209de'),
    '#': 4,
    treasure_id: UUID('d15e24c4-db3e-45a4-b001-48548dbe0324'),
    difficulty: 1,
    terrain: 4,
    city_name: 'Dörtağaç',
    country_name: 'Turkey',
    country_code: 'TUR',
    container_size: 0,
    stage_description: 'Necessitatibus a ipsum totam est.',
    stage_latitude: 38.45152889714736,
    stage_longitude: 42.1584209983017,
    sequence_number: 2,
  }
]

```

```
    stage_type: 0,
    visibility: 1
  },
  {
    _id: ObjectId('664c8e933b7e332b2fa21a36'),
    '#': 4188,
    treasure_id: UUID('d15e24c4-db3e-45a4-b001-48548dbe0324'),
    difficulty: 1,
    terrain: 4,
    city_name: 'Dörtağaç',
    country_name: 'Turkey',
    country_code: 'TUR',
    container_size: 0,
    stage_description: 'Dolores veritatis rerum at maxime tenetur.',
    stage_latitude: 38.446185355519056,
    stage_longitude: 42.16195164280413,
    sequence_number: 0,
    stage_type: 0,
    visibility: 0
  },
  {
    _id: ObjectId('664c8e953b7e332b2fa2912b'),
    '#': 34641,
    treasure_id: UUID('d15e24c4-db3e-45a4-b001-48548dbe0324'),
    difficulty: 1,
    terrain: 4,
    city_name: 'Dörtağaç',
    country_name: 'Turkey',
    country_code: 'TUR',
    container_size: 3,
    stage_description: 'Corporis id quos.',
    stage_latitude: 38.446255602924126,
    stage_longitude: 42.16170521034321,
    sequence_number: 1,
    stage_type: 0,
    visibility: 0
  },
  {
    _id: ObjectId('664c8e953b7e332b2fa295b1'),
    '#': 35799,
    treasure_id: UUID('d15e24c4-db3e-45a4-b001-48548dbe0324'),
    difficulty: 1,
    terrain: 4,
```

```

    city_name: 'Dörtağaç',
    country_name: 'Turkey',
    country_code: 'TUR',
    container_size: 0,
    stage_description: 'Nihil consequatur recusandae.',
    stage_latitude: 38.454396539541904,
    stage_longitude: 42.15389879750495,
    sequence_number: 3,
    stage_type: 0,
    visibility: 2
  }
]
treasure_database> db.treasures.find({ "city_name": /Dörtağaç/i
}).pretty().explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'treasure_database.treasures',
    indexFilterSet: false,
    parsedQuery: { city_name: { '$regex': 'Dörtağaç', '$options': 'i' } },
    queryHash: '87FC048A',
    planCacheKey: '528C6CB9',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        filter: { city_name: { '$regex': 'Dörtağaç', '$options': 'i' } },
        keyPattern: { city_name: 1 },
        indexName: 'city_name_1',
        isMultiKey: false,
        multiKeyPaths: { city_name: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { city_name: [ '["", {}]', '[/Dörtağaç/i,
/Dörtağaç/i]' ] }
      }
    },
  },

```

```
rejectedPlans: []
},
executionStats: {
  executionSuccess: true,
  nReturned: 4,
  executionTimeMillis: 143,
  totalKeysExamined: 54686,
  totalDocsExamined: 4,
  executionStages: {
    stage: 'FETCH',
    nReturned: 4,
    executionTimeMillisEstimate: 18,
    works: 54687,
    advanced: 4,
    needTime: 54682,
    needYield: 0,
    saveState: 54,
    restoreState: 54,
    isEOF: 1,
    docsExamined: 4,
    alreadyHasObj: 0,
    inputStage: {
      stage: 'IXSCAN',
      filter: { city_name: { '$regex': 'Dörtağaç', '$options': 'i' } },
      nReturned: 4,
      executionTimeMillisEstimate: 18,
      works: 54687,
      advanced: 4,
      needTime: 54682,
      needYield: 0,
      saveState: 54,
      restoreState: 54,
      isEOF: 1,
      keyPattern: { city_name: 1 },
      indexName: 'city_name_1',
      isMultiKey: false,
      multiKeyPaths: { city_name: [] },
      isUnique: false,
      isSparse: false,
      isPartial: false,
      indexVersion: 2,
      direction: 'forward',
      indexBounds: { city_name: [ '["", {}]', '/Dörtağaç/i,
```

```

/Dörtağaç/i]' ] },
  keysExamined: 54686,
  seeks: 1,
  dupsTested: 0,
  dupsDropped: 0
}
},
command: {
  find: 'treasures',
  filter: { city_name: /Dörtağaç/i },
  '$db': 'treasure_database'
},
serverInfo: {
  host: 'ZamLaptop',
  port: 27017,
  version: '7.0.9',
  gitVersion: '3ff3a3925c36ed277cf5eafca5495f2e3728dd67'
},
serverParameters: {
  internalQueryFacetBufferSizeBytes: 104857600,
  internalQueryFacetMaxOutputDocSizeBytes: 104857600,
  internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
  internalDocumentSourceGroupMaxMemoryBytes: 104857600,
  internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
  internalQueryProhibitBlockingMergeOnMongoS: 0,
  internalQueryMaxAddToSetBytes: 104857600,
  internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
  internalQueryFrameworkControl: 'trySbeRestricted'
},
ok: 1
}

```

Why did I do it this way?

The reason is:

Shard Initialization: I initialized three shards using replica sets to ensure data redundancy and fault tolerance within each shard. This ensures that even if one replica goes down, data remains accessible from other replicas.

1. **Configuration Server Initialization:** I initialized the configuration server replica set to store metadata about the sharded cluster. This is essential for managing the cluster's configuration and metadata.
2. **Adding Shards to the Router:** I added each shard to the mongos router to allow the router to distribute queries across the shards. This enables efficient query routing and load balancing across the cluster.
3. **Checking Sharding Status:** I used `sh.status()` command on mongos router to verify the status of the sharded cluster. This provided important information about the cluster's components, including shards, active mongoses, and balancer status, ensuring that the cluster was set up correctly.
4. **Query Execution Analysis:** I executed a query to find documents with a specific city name in the treasures collection. I analyzed the query execution statistics to optimize performance and understand how MongoDB processes the query within a sharded environment.

Regarding the choice of using `city_name` as the shard key:

- **Data Distribution:** Shard key ensures even distribution of data based on geographical locations, preventing hotspots and ensuring efficient query execution.
- **Query Performance:** Efficient routing of queries targeting specific cities to the appropriate shard enhances query performance.
- **Scalability:** Horizontal scalability is enabled by sharding on `city_name`, allowing for the addition of more shards as the dataset grows.
- **Data Locality:** Improved data locality for queries related to specific cities, reducing network latency.
- **Natural Hierarchical Structure:** City names naturally form a hierarchical structure, making `city_name` a logical choice for efficient range-based sharding strategies.

In summary, my actions were aimed at ensuring proper setup, verification, and optimization of a MongoDB sharded cluster, with the choice of `city_name` as the shard key optimizing data distribution, query performance, and scalability for location-based data management.

Now How to do it using the 3 shards

Now to make sure my data is split on the 3 shards:

Shard the collection using the `splitAt()` command to divide data into chunks based on the shard key's values.

javascript

Copy code

```
use treasure_database
```

```
// Split at 'A' to create two chunks: [-Inf, A), [A, +Inf)
```

```
sh.splitAt("treasure_database.treasures", { city_name: "A" })

// Split at 'M' to create three chunks: [-Inf, A), [A, M), [M, +Inf)
sh.splitAt("treasure_database.treasures", { city_name: "M" })

// Split at 'Z' to create four chunks: [-Inf, A), [A, M), [M, Z), [Z,
+Inf)
sh.splitAt("treasure_database.treasures", { city_name: "Z" })
```

- 1.
2. **Balancer Configuration:**
 - Ensure the balancer is enabled and running to automatically redistribute chunks and maintain data balance across shards.

javascript
Copy code

```
// Check balancer status
sh.getBalancerState()

// Start balancer if not running
sh.startBalancer()
```

- 3.
4. **Data Insertion and Monitoring:**
 - Insert sample data into the sharded collection to simulate load and trigger chunk splitting.
 - Monitor the distribution of data across shards using
`db.treasures.find().explain("executionStats")`.

javascript
Copy code

```
// Insert sample data
db.treasures.insertMany([
  { city_name: "A City", treasure: "Gold" },
  { city_name: "M City", treasure: "Silver" },
  { city_name: "Z City", treasure: "Diamonds" },
  // Add more documents to simulate load
])

// Check data distribution
db.treasures.find().explain("executionStats")
```

then did a `db.treasures.find().explain("executionStats")` and it revealed this: Mongo-shard3-rs: Returned 62 documents in 14 milliseconds, performing a COLLSCAN. Mongo-shard2-rs: Returned 27,961 documents in 46 milliseconds, also performing a COLLSCAN. Mongo-shard1-rs: Returned 26,666 documents in 90 milliseconds, with a COLLSCAN

5.

Recommendations

1. Index Optimization:

- Create indexes on fields commonly used for querying to improve query performance and reduce the need for COLLSCAN operations.
- Analyze query patterns and create indexes to support efficient data retrieval.

2. Data Distribution:

- Ensure that data is evenly distributed across shards to maximize the efficiency of the sharded cluster.
- Monitor data distribution regularly and rebalance shards as needed to maintain an even distribution.

3. Monitoring and Adjustment:

- Continuously monitor the performance and distribution of data within the sharded cluster.
- Adjust shard keys and indexes as needed based on evolving application requirements and data access patterns.

Conclusion

Efficient data distribution and index optimization are essential aspects of sharding MongoDB databases. By carefully selecting shard keys, properly sharding collections, and optimizing indexes, you can ensure optimal performance and scalability in a sharded environment. Regular monitoring and adjustment are key to maintaining the effectiveness of sharding in handling large datasets and high query loads.

Using `db.treasures.getShardDistribution()` to check whether they are evenly distributed and below is the shard distribution:

Shard mongo-shard1-rs:

Data: 8.82 MiB

Documents: 26,666

Chunks: 2

Estimated data per chunk: 4.41 MiB

Estimated documents per chunk: 13,333

Shard mongo-shard3-rs:

Data: 20 KiB

Documents: 62

Chunks: 1

Estimated data per chunk: 20 KiB

Estimated documents per chunk: 62

Shard mongo-shard2-rs:

Data: 9.18 MiB

Documents: 27,961

Chunks: 1

Estimated data per chunk: 9.18 MiB

Estimated documents per chunk: 27,961

Here's a summary of the actions taken to achieve a more even distribution of data across shards:

1. Initially, a chunk was moved from shard 2 to shard 3 to balance the distribution:

```
sh.moveChunk("treasure_database.treasures", { city_name: "A" },  
"mongo-shard3-rs")
```

2. However, this resulted in overlapping chunks, so another adjustment was made by moving a chunk from shard 3 back to shard 2:

```
sh.moveChunk("treasure_database.treasures", { city_name: "M" },  
"mongo-shard2-rs")
```

3. Despite the previous moves, the distribution was still not optimal. To further improve it, a new chunk was split at 'S' to create a more balanced distribution:

```
sh.splitAt("treasure_database.treasures", { city_name: 'S' })
```

This series of actions led to a more even distribution of data across shards, with each shard holding a more balanced amount of data and documents. The improved distribution was verified by checking the distribution using `db.treasures.getShardDistribution()`, which showed a more equitable distribution across the shards.

With this distribution, shard mongo-shard2-rs and shard mongo-shard1-rs have 36.49% and 31.07% of the data, respectively, while shard mongo-shard3-rs has 32.43% of the data. This distribution is more evenly spread compared to the previous state.

Now we run the same query as before:

```
db.treasures.find({ "city_name": /Dörtağaç/i }).pretty().explain("executionStats")
```

```
{  
  queryPlanner: {  
    mongosPlannerVersion: 1,  
    winningPlan: {  
      stage: 'SHARD_MERGE',  
      shards: [  
        {  
          shardName: 'mongo-shard2-rs',  
          connectionString:  
'mongo-shard2-rs/mongoddb_mongo-shard2-1_1:27017,mongoddb_mongo-shard2-2_1:27  
017,mongoddb_mongo-shard2-3_1:27017',  
          serverInfo: {  
            host: '96e5124bcf21',  
            port: 27017,  
            version: '5.0.26',  
            gitVersion: '0b4f1ea980b5380a66425a90b414106a191365f4'
```

```

    },
    namespace: 'treasure_database.treasures',
    indexFilterSet: false,
    parsedQuery: { city_name: { '$regex': 'Dörtağaç', '$options': 'i'
} }},

    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'SHARDING_FILTER',
        inputStage: {
          stage: 'IXSCAN',
          filter: {
            city_name: { '$regex': 'Dörtağaç', '$options': 'i' }
          },
          keyPattern: { city_name: 1 },
          indexName: 'city_name_1',
          isMultiKey: false,
          multiKeyPaths: { city_name: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: {
            city_name: [ '['", {}))', '/Dörtağaç/i, /Dörtağaç/i]' ]
          }
        }
      }
    },
    rejectedPlans: []
  },
  {
    shardName: 'mongo-shard1-rs',
    connectionString:
'mongo-shard1-rs/mongodb_mongo-shard1-1_1:27017,mongodb_mongo-shard1-2_1:27
017,mongodb_mongo-shard1-3_1:27017',
    serverInfo: {
      host: '3d2937e1da27',
      port: 27017,
      version: '5.0.26',

```

```

    gitVersion: '0b4f1ea980b5380a66425a90b414106a191365f4'
  },
  namespace: 'treasure_database.treasures',
  indexFilterSet: false,
  parsedQuery: { city_name: { '$regex': 'Dörtağaç', '$options': 'i'
} } },

  maxIndexedOrSolutionsReached: false,
  maxIndexedAndSolutionsReached: false,
  maxScansToExplodeReached: false,
  winningPlan: {
    stage: 'FETCH',
    inputStage: {
      stage: 'SHARDING_FILTER',
      inputStage: {
        stage: 'IXSCAN',
        filter: {
          city_name: { '$regex': 'Dörtağaç', '$options': 'i' }
        },
        keyPattern: { city_name: 1 },
        indexName: 'city_name_1',
        isMultiKey: false,
        multiKeyPaths: { city_name: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          city_name: [ '['", {}))', '/Dörtağaç/i, /Dörtağaç/i]' ]
        }
      }
    }
  },
  rejectedPlans: []
},
{
  shardName: 'mongo-shard3-rs',
  connectionString:
'mongo-shard3-rs/mongodb_mongo-shard3-1_1:27017,mongodb_mongo-shard3-2_1:27
017,mongodb_mongo-shard3-3_1:27017',
  serverInfo: {
    host: '574aa9f451aa',
    port: 27017,

```

```

        version: '5.0.26',
        gitVersion: '0b4f1ea980b5380a66425a90b414106a191365f4'
    },
    namespace: 'treasure_database.treasures',
    indexFilterSet: false,
    parsedQuery: { city_name: { '$regex': 'Dörtağaç', '$options': 'i'
} } },

    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
        stage: 'FETCH',
        inputStage: {
            stage: 'SHARDING_FILTER',
            inputStage: {
                stage: 'IXSCAN',
                filter: {
                    city_name: { '$regex': 'Dörtağaç', '$options': 'i' }
                },
                keyPattern: { city_name: 1 },
                indexName: 'city_name_1',
                isMultiKey: false,
                multiKeyPaths: { city_name: [] },
                isUnique: false,
                isSparse: false,
                isPartial: false,
                indexVersion: 2,
                direction: 'forward',
                indexBounds: {
                    city_name: [ '['", {}))', '/Dörtağaç/i, /Dörtağaç/i]' ]
                }
            }
        }
    },
    rejectedPlans: []
}
]
}
},
executionStats: {
    nReturned: 4,
    executionTimeMillis: 42,
    totalKeysExamined: 58063,

```



```
totalDocsExamined: 4,
executionStages: {
  stage: 'SHARD_MERGE',
  nReturned: 4,
  executionTimeMillis: 42,
  totalKeysExamined: 58063,
  totalDocsExamined: 4,
  totalChildMillis: Long('74'),
  shards: [
    {
      shardName: 'mongo-shard2-rs',
      executionSuccess: true,
      nReturned: 0,
      executionTimeMillis: 4,
      totalKeysExamined: 3436,
      totalDocsExamined: 0,
      executionStages: {
        stage: 'FETCH',
        nReturned: 0,
        executionTimeMillisEstimate: 1,
        works: 3437,
        advanced: 0,
        needTime: 3436,
        needYield: 0,
        saveState: 3,
        restoreState: 3,
        isEOF: 1,
        docsExamined: 0,
        alreadyHasObj: 0,
        inputStage: {
          stage: 'SHARDING_FILTER',
          nReturned: 0,
          executionTimeMillisEstimate: 1,
          works: 3437,
          advanced: 0,
          needTime: 3436,
          needYield: 0,
          saveState: 3,
          restoreState: 3,
          isEOF: 1,
          chunkSkips: 0,
          inputStage: {
            stage: 'IXSCAN',
```

```

    filter: {
      city_name: { '$regex': 'Dörtağaç', '$options': 'i' }
    },
    nReturned: 0,
    executionTimeMillisEstimate: 1,
    works: 3437,
    advanced: 0,
    needTime: 3436,
    needYield: 0,
    saveState: 3,
    restoreState: 3,
    isEOF: 1,
    keyPattern: { city_name: 1 },
    indexName: 'city_name_1',
    isMultiKey: false,
    multiKeyPaths: { city_name: [] },
    isUnique: false,
    isSparse: false,
    isPartial: false,
    indexVersion: 2,
    direction: 'forward',
    indexBounds: {
      city_name: [ '['", {}))', '[/Dörtağaç/i, /Dörtağaç/i]' ]
    },
    keysExamined: 3436,
    seeks: 1,
    dupsTested: 0,
    dupsDropped: 0
  }
}
},
{
  shardName: 'mongo-shard1-rs',
  executionSuccess: true,
  nReturned: 0,
  executionTimeMillis: 30,
  totalKeysExamined: 26666,
  totalDocsExamined: 0,
  executionStages: {
    stage: 'FETCH',
    nReturned: 0,
    executionTimeMillisEstimate: 3,

```

```
works: 26667,
advanced: 0,
needTime: 26666,
needYield: 0,
saveState: 26,
restoreState: 26,
isEOF: 1,
docsExamined: 0,
alreadyHasObj: 0,
inputStage: {
  stage: 'SHARDING_FILTER',
  nReturned: 0,
  executionTimeMillisEstimate: 2,
  works: 26667,
  advanced: 0,
  needTime: 26666,
  needYield: 0,
  saveState: 26,
  restoreState: 26,
  isEOF: 1,
  chunkSkips: 0,
  inputStage: {
    stage: 'IXSCAN',
    filter: {
      city_name: { '$regex': 'Dörtağaç', '$options': 'i' }
    },
    nReturned: 0,
    executionTimeMillisEstimate: 2,
    works: 26667,
    advanced: 0,
    needTime: 26666,
    needYield: 0,
    saveState: 26,
    restoreState: 26,
    isEOF: 1,
    keyPattern: { city_name: 1 },
    indexName: 'city_name_1',
    isMultiKey: false,
    multiKeyPaths: { city_name: [] },
    isUnique: false,
    isSparse: false,
    isPartial: false,
    indexVersion: 2,
```

```

        direction: 'forward',
        indexBounds: {
          city_name: [ '["", {}]', '[/Dörtağaç/i, /Dörtağaç/i]' ]
        },
        keysExamined: 26666,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  {
    shardName: 'mongo-shard3-rs',
    executionSuccess: true,
    nReturned: 4,
    executionTimeMillis: 40,
    totalKeysExamined: 27961,
    totalDocsExamined: 4,
    executionStages: {
      stage: 'FETCH',
      nReturned: 4,
      executionTimeMillisEstimate: 4,
      works: 27962,
      advanced: 4,
      needTime: 27957,
      needYield: 0,
      saveState: 27,
      restoreState: 27,
      isEOF: 1,
      docsExamined: 4,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'SHARDING_FILTER',
        nReturned: 4,
        executionTimeMillisEstimate: 4,
        works: 27962,
        advanced: 4,
        needTime: 27957,
        needYield: 0,
        saveState: 27,
        restoreState: 27,
        isEOF: 1,

```

```

    chunkSkips: 0,
    inputStage: {
      stage: 'IXSCAN',
      filter: {
        city_name: { '$regex': 'Dörtağaç', '$options': 'i' }
      },
      nReturned: 4,
      executionTimeMillisEstimate: 4,
      works: 27962,
      advanced: 4,
      needTime: 27957,
      needYield: 0,
      saveState: 27,
      restoreState: 27,
      isEOF: 1,
      keyPattern: { city_name: 1 },
      indexName: 'city_name_1',
      isMultiKey: false,
      multiKeyPaths: { city_name: [] },
      isUnique: false,
      isSparse: false,
      isPartial: false,
      indexVersion: 2,
      direction: 'forward',
      indexBounds: {
        city_name: [ ['"', {})', '/Dörtağaç/i, /Dörtağaç/i]' ]
      },
      keysExamined: 27961,
      seeks: 1,
      dupsTested: 0,
      dupsDropped: 0
    }
  }
}
}
}
}
],
},
serverInfo: {
  host: 'b4d56766ce5b',
  port: 27017,
  version: '5.0.26',
  gitVersion: '0b4f1ea980b5380a66425a90b414106a191365f4'
}

```

```

},
serverParameters: {
  internalQueryFacetBufferSizeBytes: 104857600,
  internalQueryFacetMaxOutputDocSizeBytes: 104857600,
  internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
  internalDocumentSourceGroupMaxMemoryBytes: 104857600,
  internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
  internalQueryProhibitBlockingMergeOnMongoS: 0,
  internalQueryMaxAddToSetBytes: 104857600,
  internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
},
command: {
  find: 'treasures',
  filter: { city_name: /Dörtağaç/i },
  lsid: { id: UUID('8a1d3274-804d-4329-aeae-052a0975b70a') },
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1717149204, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  '$db': 'treasure_database'
},
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1717149239, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
    keyId: Long('0')
  }
},
operationTime: Timestamp({ t: 1717149237, i: 1 })
}

```

Which shows us that:

1. Query Plan:

- The winning plan involves a stage called **SHARD_MERGE**, which merges results from different shards.

- Each shard (mongo-shard1-rs, mongo-shard2-rs, and mongo-shard3-rs) has its own execution plan.
- 2. **Shard mongo-shard2-rs:**
 - The query planner for this shard indicates that it used an index scan (**IXSCAN**) on the **city_name** index with a regex filter.
 - The execution stages include a fetch operation to retrieve documents matching the query.
- 3. **Shard mongo-shard1-rs:**
 - Similar to shard mongo-shard2-rs, it also used an index scan with a regex filter.
- 4. **Shard mongo-shard3-rs:**
 - This shard also used an index scan with a regex filter.
- 5. **Execution Stats:**
 - The query returned 4 documents in total.
 - The execution time across all shards was 42 milliseconds.
 - Each shard examined a certain number of keys and documents as part of the query execution.

Overall, the query utilized index scans with regex filters on all shards to retrieve documents matching the specified city name (**Dörtağaç**) case-insensitively.