# **MongoDB Geo-Distributed Setup and Implementation**

This guide explains how to set up and replicate a MongoDB geo-distributed database with sharding, replication, and zoning. It includes details on performance metrics and queries for a content recommendation system.

## **1. Prerequisites**

1. **Software and Tools:**
   * MongoDB Community/Enterprise Edition (4.2+ recommended)
   * Python (3.x) for query testing and Flask-based UI (optional)
   * mongo shell for executing database commands
   * Hardware/VMs for deploying a distributed cluster (at least 4 machines/containers)
2. **Environment Setup:**
   * Minimum of 4 nodes:
     + 1 for the **Config Server Replica Set**
     + 4 for the **Shard Replica Sets**
     + 1 for the **MongoS Router**
   * Ensure network connectivity between all nodes.

Example structure:  
sql  
Copy code  
/data/mongodb/

├── config

├── shard1

│ ├── primary

│ └── secondary

├── shard2

│ ├── primary

│ └── secondary

├── shard3

│ ├── primary

│ └── secondary

├── shard4

│ ├── primary

│ └── secondary

└── router

1. **Key Concepts:**
   * **Shards**: Distribute data across multiple nodes for scalability.
   * **Replication**: Provide fault tolerance by replicating data across nodes.
   * **Zoning**: Ensure specific data resides on specific shards for locality.

## **2. MongoDB Cluster Setup**

### **Starting MongoDB Sharded Cluster and Checking Status**

#### **1. Start Config Server Replica Set**

**Start the config server**:  
  
mongod --configsvr --replSet configReplSet --dbpath /data/config --port 27019

**Initialize the config replica set** (only needed once during initial setup):  
  
rs.initiate({

\_id: "configReplSet",

configsvr: true,

members: [{ \_id: 0, host: "localhost:27019" }]

})

**Check Config Server Replica Set Status**:  
  
rs.status()

#### **2. Start Shard Servers**

**Shard 1 Primary**:  
  
mongod --shardsvr --replSet shardReplSet1 --dbpath /data/shard1 --port 27018

**Shard 1 Secondary**:  
  
mongod --shardsvr --replSet shardReplSet1 --dbpath /data/shard1\_secondary --port 27020

**Shard 2 Primary**:  
  
mongod --shardsvr --replSet shardReplSet2 --dbpath /data/shard2 --port 27016

**Shard 2 Secondary**:  
  
mongod --shardsvr --replSet shardReplSet2 --dbpath /data/shard2\_secondary --port 27021

* **Initialize Each Shard Replica Set** (only needed once):
  + Connect to each primary shard instance in mongosh and add the secondary node:

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

rs.add("localhost:27020")

For shardReplSet2:  
  
rs.initiate({ \_id: "shardReplSet2", members: [{ \_id: 0, host: "localhost:27016" }] })

rs.add("localhost:27021")

* **Check Shard Replica Set Status**:

For each shard, connect to the primary node and run:  
  
rs.status()

This step is repeated for 4 shards

#### **3. Start Mongos Router**

**Start mongos**:  
  
mongos --configdb configReplSet/localhost:27019 --port 27015

* **Check Sharding Status in mongos**:

Connect to mongos and check sharding status:  
  
sh.status()

#### **4. Adding Shards and Sharding Collections**

**Add Shards** (only needed once during initial setup):  
  
sh.addShard("shardReplSet1/localhost:27018")

sh.addShard("shardReplSet2/localhost:27016")

## **3. Sharding and Zoning**

### **Step 1: Enable Sharding for the Database**

sh.enableSharding("DDS\_Project")

### **Step 2: Shard Collections**

**Users Collection (Range-Based Sharding by Location):**  
sh.shardCollection("DDS\_Project.users", { location: 1 })

**Regional Content (Range-Based Sharding by Region):**  
sh.shardCollection("DDS\_Project.regional\_content", { region: 1 })

### **Step 3: Configure Zoning**

Define zones:  
  
sh.addShardToZone("shardReplSet1", "AsiaShard")

sh.addShardToZone("shardReplSet2", "EuropeShard")

sh.addShardToZone("shardReplSet3", "AmericasShard")

1. Associate data ranges with zones:  
     
   sh.updateZoneKeyRange("DDS\_Project.users", { location: "Asia" }, { location: "AsiaZZZ" }, "AsiaShard")

sh.updateZoneKeyRange("DDS\_Project.users", { location: "Europe" }, { location: "EuropeZZZ" }, "EuropeShard")

Same for North America and South America

## **4. Query Examples**

### **Query 1: Local Recommendations (Top 10)**

db.regional\_trends.aggregate([

{

$match: { region: "South America" } // Filter for the Asia region

},

{

$sort: { // Sort by total views and likes in descending order

"engagement\_metrics.total\_views": -1,

"engagement\_metrics.total\_likes": -1

}

},

{

$lookup: { // Join with the content collection for additional content details

from: "content", // Collection to join

localField: "top\_content", // Field in regional\_trends (top\_content)

foreignField: "title", // Field in content (title)

as: "content\_details" // Name of the array where joined content will be stored

}

},

{

$unwind: "$content\_details" // Unwind the content details to flatten the array

},

{

$project: { // Select and rename the required fields

\_id: 0,

"Content Title": "$top\_content",

"Content Type": "$content\_details.type",

"Total Views": "$engagement\_metrics.total\_views",

"Total Likes": "$engagement\_metrics.total\_likes"

}

},

{

$limit: 10 // Limit to top 10 results

}

]);

### **Query 2: Global Recommendations (Top 5)**

db.regional\_trends.aggregate([

{

$group: { // Group by content title

\_id: "$top\_content",

total\_views: { $sum: "$engagement\_metrics.total\_views" }, // Sum views across regions

total\_likes: { $sum: "$engagement\_metrics.total\_likes" }, // Sum likes across regions

regions: { $addToSet: "$region" } // Collect all regions

}

},

{

$match: { // Ensure content is trending in at least 4 regions

"regions.3": { $exists: true }

}

},

{

$sort: { // Sort by total views and likes in descending order

total\_views: -1,

total\_likes: -1

}

},

{

$lookup: { // Join with the content collection

from: "content",

localField: "\_id", // Field in regional\_trends (top\_content)

foreignField: "title", // Field in content (title)

as: "content\_details"

}

},

{

$unwind: "$content\_details" // Flatten the content\_details array

},

{

$project: { // Select and rename fields

\_id: 0,

"Content Title": "$\_id",

"Content Type": "$content\_details.type",

"Total Views": "$total\_views",

"Total Likes": "$total\_likes"

}

},

{ $limit: 5 } // Limit to top 5 global content

]);

## **5. Performance Metrics**

### **Why These Metrics Were Chosen:**

1. **Execution Time (ms):** Measures query speed for a responsive system.
2. **Throughput (queries/sec):** Assesses the system's capacity under load.
3. **Average Response Time (ms):** Indicates the average delay for a user query.
4. **CPU Utilization (%):** Evaluates resource efficiency.
5. **Memory Utilization (%):** Tracks memory usage for scalability.

## **6. Observations and Results**

* MongoDB query time was the best due to sharding and efficient query routing.
* CPU utilization was higher because MongoDB processes sorting and aggregation in memory.

## **7. Troubleshooting**

**Shard Distribution Check:**  
db.users.getShardDistribution()

* **Zone Range Issues:** Ensure min and max fields in updateZoneKeyRange do not overlap.

**Shard Balancing:**  
sh.startBalancer()

sh.stopBalancer()