MongoDB Replication and Sharding Configuration Document

1. MongoDB Replication Configuration

1.1 Replication Architecture Design

- **Primary Node:**
  - Handles all write operations
  - Records all data changes to the oplog
  - Maintains cluster configuration
- **Secondary Nodes:**
  - 2 secondary nodes for data replication and read scaling
  - Maintains data synchronization with the primary node through asynchronous replication
  - Can handle read operations (requires read preference configuration)
- **Arbiter Node:**
  - Does not store data, only participates in voting
  - Ensures the replica set always has an odd number of voting members
  - Helps elect a new primary node when the current primary fails

1.2 Detailed Configuration Steps

1. Create data directories for each node:

```
# Primary node data directory
mkdir -p /data/db/primary
# Secondary node data directories
mkdir -p /data/db/secondary1
mkdir -p /data/db/secondary2
# Arbiter node directory
mkdir -p /data/db/arbiter
```

2. Create primary node configuration file `mongod-primary.conf`:

```
systemLog:
  destination: file
  path: "/data/db/primary/mongod.log"
  logAppend: true
storage:
  dbPath: "/data/db/primary"
  journal:
    enabled: true
net:
  bindIp: 0.0.0.0
  port: 27017
replication:
  replSetName: "elearningReplicaSet"
security:
  authorization: enabled
  keyFile: "/data/db/keyfile"
processManagement:
  fork: true
```

3. Create secondary node configuration file `mongod-secondary1.conf`:

```
systemLog:
  destination: file
  path: "/data/db/secondary1/mongod.log"
  logAppend: true
storage:
  dbPath: "/data/db/secondary1"
  journal:
    enabled: true
net:
  bindIp: 0.0.0.0
  port: 27018
replication:
  replSetName: "elearningReplicaSet"
security:
  authorization: enabled
  keyFile: "/data/db/keyfile"
processManagement:
  fork: true
```

4. Create secondary node configuration file `mongod-secondary2.conf`:

```
systemLog:
  destination: file
  path: "/data/db/secondary2/mongod.log"
  logAppend: true
storage:
  dbPath: "/data/db/secondary2"
  journal:
    enabled: true
net:
  bindIp: 0.0.0.0
  port: 27019
replication:
  replSetName: "elearningReplicaSet"
security:
  authorization: enabled
  keyFile: "/data/db/keyfile"
processManagement:
  fork: true
```

5. Create arbiter node configuration file `mongod-arbiter.conf`:

```
systemLog:
  destination: file
  path: "/data/db/arbiter/mongod.log"
  logAppend: true
storage:
  dbPath: "/data/db/arbiter"
  journal:
    enabled: true
net:
  bindIp: 0.0.0.0
  port: 27020
replication:
  replSetName: "elearningReplicaSet"
security:
  authorization: enabled
  keyFile: "/data/db/keyfile"
processManagement:
  fork: true
```

6. Create authentication key file:

```
openssl rand -base64 756 > /data/db/keyfile
chmod 400 /data/db/keyfile
```

7. Start all nodes:

```
# Start primary node
mongod --config mongod-primary.conf

# Start secondary nodes
mongod --config mongod-secondary1.conf
mongod --config mongod-secondary2.conf

# Start arbiter node
mongod --config mongod-arbiter.conf
```

8. Connect to the primary node and initialize the replica set:

```
mongo --port 27017

// Initialize replica set
rs.initiate({
  _id: "elearningReplicaSet",
  members: [
    {
      _id: 0,
      host: "localhost:27017",
      priority: 2
    },
    {
      _id: 1,
      host: "localhost:27018",
      priority: 1
    },
    {
      _id: 2,
      host: "localhost:27019",
      priority: 1
    },
    {
      _id: 3,
      host: "localhost:27020",
      arbiterOnly: true
    }
  ]
})

// Check replica set status
rs.status()
```

9. Configure read/write preferences:

```
// Set read/write preferences in application connection string
const uri = "mongodb://localhost:27017,localhost:27018,localhost:27019/?replicaSet=elearningReplicaSet&readPreference=secondary&w=majority"

// Or set in code
db.getMongo().setReadPref("secondary")
```

10. Monitor replication status:

```
// Check replication lag
rs.printSlaveReplicationInfo()

// Check oplog size
db.getReplicationInfo()
```

1.3 Replica Set Maintenance Operations

1. Add a new secondary node:

```
rs.add("localhost:27021")
```

2. Remove a node:

```
rs.remove("localhost:27021")
```

3. Force re-election:

```
rs.stepDown()
```

4. Modify node priority:

```
var cfg = rs.conf()
cfg.members[1].priority = 2
rs.reconfig(cfg)
```

1.4 Benefits of Replication

1. **High Availability**
    - Automatic failover: Secondary nodes automatically upgrade to primary when the primary node fails
    - No single point of failure: Multiple nodes ensure continuous service availability
    - Service continuity guarantee: The failover process is transparent to applications
    - Maintenance window optimization: Nodes can be maintained in rotation without affecting service

2. **Data Redundancy**
    - Multiple data backups: Each secondary node maintains a complete copy of the data
    - Prevent data loss: Multiple copies ensure data safety
    - Disaster recovery capability: Data can be recovered from any healthy node
    - Geographic distribution: Nodes can be deployed in different geographical locations

3. **Read Scaling**
    - Secondary nodes can handle read operations: Distributes read pressure
    - Reduces primary node load: Primary node can focus on write operations
    - Improves overall system performance: Better concurrent processing capability
    - Proximity reading: Can select the nearest secondary node based on geographic location

2. MongoDB Sharding Configuration

2.1 Sharding Architecture Design

- **Config Servers:** Store cluster metadata
- **Router Servers (Mongos):** Handle client requests
- **Shard Servers:** Store actual data

2.2 Configuration Steps

1. Start config server:

```
mongod --configsvr --port 27019 --dbpath /data/configdb
```

2. Start shard servers:

```
# Shard 1
mongod --shardsvr --port 27018 --dbpath /data/shard1

# Shard 2
mongod --shardsvr --port 27020 --dbpath /data/shard2
```

3. Start router server:

```
mongos --configdb localhost:27019 --port 27017
```

4. Add shards to the cluster:

```
sh.addShard("localhost:27018")
sh.addShard("localhost:27020")
```

5. Enable database sharding:

```
sh.enableSharding("elearning")
```

6. Create shard key:

```
sh.shardCollection("elearning.courses", { "department": "hashed" })
```

2.3 Benefits of Sharding

1. **Horizontal Scaling**
    - Supports unlimited data growth

  - Distributes storage pressure
  - Increases system capacity

2. **Performance Optimization**

   - Parallel query processing
   - Load balancing
   - Improves response time

3. **Resource Utilization**

   - Better hardware utilization
   - Cost-effectiveness optimization
   - Flexible scalability

3. Best Practice Recommendations

1. **Replica Set Configuration**

   - Use an odd number of nodes
   - Set priorities appropriately
   - Regularly monitor replication lag

2. **Sharding Strategy**

   - Choose appropriate shard keys
   - Avoid hotspot data
   - Regularly check data distribution

3. **Operations Management**

   - Regularly backup data
   - Monitor system performance
   - Develop failure recovery plans