Introduction to Replication in MongoDB

- Replica set architecture
- Primary, Secondary, Arbiter
- Elections & failover
- Read preference & write concerns
- Lab: Set up a 3-node replica set locally
- Lab: Test failover + observe elections

1. Set up a **3-node replica set locally**
2. **Test failover** and observe elections, read preferences, and write concerns

---

0. CSC Story / Use Case (use this narrative in training)

CSC runs a **Compliance & Filings Platform** for thousands of corporate clients.

They **cannot afford downtime** during US year-end filing season.

- Production runs on a **3-node MongoDB replica set** (Linux or Atlas).
- Bangalore SRE team is responsible for **high availability** and **disaster recovery**.
- Mumbai team builds reporting and analytics that must read from **secondaries** without impacting primary.

To do this safely, everyone must understand:

- **Replica set architecture**
- **Primary / Secondary / Arbiter**
- **Elections & failover**
- **Read preferences & write concerns**
- How to **set up and break** a replica set in the lab, *before* doing anything in production.

---

1. Concepts – Replica Set Basics (Short & Clear)

1.1 Replica Set Architecture

A **replica set** is a group of mongod processes that maintain the **same data set**.

Typical CSC production pattern:

- **Primary** – handles all writes and default reads.
- **Secondaries (1–2 or more)** – replicate data from primary; can serve **read-only** traffic.
- (Optional) **Arbiter** – only votes in elections; **no data**.

Architecture example (3-node):

- csc-rs-1 – PRIMARY
- csc-rs-2 – SECONDARY
- csc-rs-3 – SECONDARY or ARBITER (depending on design)

1.2 Primary, Secondary, Arbiter

- **Primary**
  - Accepts all writes.
  - Replicates changes to secondaries via **oplog**.
- **Secondary**
  - Copies data from primary (oplog).
  - Can be used for **reads**, backups, reporting.
- **Arbiter**
  - No data.
  - Only participates in **elections** to avoid ties (odd number of votes).

CSC-style use:

- Primary: in Bangalore or primary DC
- Secondaries: one more in same DC, another in DR region or Mumbai DC
- Arbiter: sometimes used in small clusters where you want 3 votes but only 2 data-bearing nodes.

1.3 Elections & Failover

If a **primary goes down**:

1. Replica set members detect primary is unavailable.
2. Eligible secondary calls an **election**.
3. New primary is elected based on:
   - Priority
   - Data “freshness”

    - Voting majority

Failover is **automatic** (within a few seconds) → this is what keeps CSC app **online**.

1.4 Read Preferences & Write Concerns

- **Read Preference**
    - primary (default) – safest, latest data.
    - secondary – offload reads (e.g., reporting in Mumbai).
    - primaryPreferred, secondaryPreferred, nearest – hybrid strategies.
- **Write Concern**
    - w: 1 – acknowledge from primary only.
    - w: "majority" – acknowledge when majority of nodes have written (safer).
    - j: true – wait for journal, improving durability.

CSC production typical choice:

- Most critical writes: writeConcern: { w: "majority", j: true }
- Some internal logs/metrics: w: 1

---

2. Lab 1 – Set up a 3-node Replica Set LOCALLY

We'll simulate a 3-node replica set on a **single machine** using **three different data folders and ports**.

We'll show the setup generically, then highlight OS-specific differences:

- **Linux / Mac** – similar shell commands.
- **Windows** – use cmd or PowerShell.

For simplicity, we'll name:

- Replica set name: cscReplSet
- Ports: 27017, 27018, 27019
- Data dirs: data/rs1, data/rs2, data/rs3

Important: Make sure mongod is installed and on PATH.

2.1 Prepare Data Directories

Linux / Mac

```
mkdir -p ~/csc-repl/data/rs1 ~/csc-repl/data/rs2 ~/csc-repl/data/rs3
```

Windows (PowerShell)

```
mkdir C:\csc-repl\data\rs1
mkdir C:\csc-repl\data\rs2
mkdir C:\csc-repl\data\rs3
```

2.2 Start 3 mongod Instances

Linux / Mac

Open **three terminals**, one for each node:

Node 1:

```
mongod --replSet cscReplSet --port 27017 --dbpath ~/csc-repl/data/rs1 --
bind_ip 127.0.0.1
```

Node 2:

```
mongod --replSet cscReplSet --port 27018 --dbpath ~/csc-repl/data/rs2 --
bind_ip 127.0.0.1
```

Node 3:

```
mongod --replSet cscReplSet --port 27019 --dbpath ~/csc-repl/data/rs3 --
bind_ip 127.0.0.1
```

Windows (cmd or PowerShell)

Again, three separate windows:

```
mongod --replSet cscReplSet --port 27017 --dbpath C:\csc-repl\data\rs1 --
bind_ip 127.0.0.1
mongod --replSet cscReplSet --port 27018 --dbpath C:\csc-repl\data\rs2 --
bind_ip 127.0.0.1
```

```
mongod --replSet cscReplSet --port 27019 --dbpath C:\csc-repl\data\rs3 --
bind_ip 127.0.0.1
```

At this point, 3 standalone mongods are running with the **same replica set name**.

2.3 Initialize the Replica Set

Connect to **node 1** using mongosh:

```
mongosh "mongodb://127.0.0.1:27017"
```

Run:

```
rs.initiate({
  _id: "cscReplSet",
  members: [
    { _id: 0, host: "127.0.0.1:27017" },
    { _id: 1, host: "127.0.0.1:27018" },
    { _id: 2, host: "127.0.0.1:27019" }
  ]
})
```

Check status:

```
rs.status()
```

You should see:

- One node as **PRIMARY**
- Two nodes as **SECONDARY**

✅ **CSC use case explanation**

This mimics CSC's **3-node production replica set.**

In reality, instead of ports on one machine, these hosts would be:

- mongo1.bangalore.csc.com:27017
- mongo2.bangalore.csc.com:27017
- mongo3.mumbai.csc.com:27017

2.4 Insert & Read Data from Replica Set

Still on primary:

```
use csc_compliance

db.filings.insertOne({
  entity_id: 1001,
  filing_type: "Annual Report",
  state: "DE",
  status: "OPEN",
  amount: 5000
})
```

Check:

```
db.filings.find().pretty()
```

Now connect to a **secondary** to see replication (for test, we’ll enable secondary reads):

```
mongosh "mongodb://127.0.0.1:27018"
```

By default, secondaries don’t allow reads, so run:

```
rs.slaveOk()  // for test only (legacy name; still works)
use csc_compliance
db.filings.find().pretty()
```

You should see the same document → replication working.

✅ **CSC use case explanation**

Mumbai reporting service reads from **secondaries** to generate heavy monthly compliance reports without overloading the primary.

---

3. Lab 2 – Elections, Failover, Read Preference, Write Concerns

Now we’ll **break things on purpose** and see how the replica set behaves.

3.1 Trigger an Election (Primary Step Down)

Connect to the **PRIMARY** (usually 27017):

```
mongosh "mongodb://127.0.0.1:27017"
```

Run:

```
rs.status()  // confirm this is PRIMARY
rs.stepDown(60)  // step down for 60 seconds
```

What happens:

- This node becomes SECONDARY.
- Another node (27018 or 27019) becomes PRIMARY.

Check from a different node:

```
mongosh "mongodb://127.0.0.1:27018"
rs.isMaster()
```

(Or in newer versions, db.hello().)

You should see:

```
{ isWritablePrimary: true, ... }
```

✅ **CSC use case explanation**

If the primary server in Bangalore goes down, the secondary in another Bangalore rack or Mumbai DC can **take over automatically**, so client applications keep working.

3.2 Simulate Primary Crash

Instead of stepDown, you can **kill the mongod** process of the current primary window (Ctrl+C).

Watch rs.status() from another node and see election logs.

3.3 Read Preference – Reading from Secondary

Let's connect using a **replica set connection string** and set read preference.

```
mongosh
"mongodb://127.0.0.1:27017,127.0.0.1:27018,127.0.0.1:27019/csc_compliance?r
eplicaSet=cscReplSet"
```

By default:

```
db.getMongo().getReadPref()  // primary
```

Change read preference to secondary (for test):

```
db.getMongo().setReadPref("secondary")

db.filings.find().limit(1)
```

Mongo will route reads to a secondary.

✅ **CSC use case explanation**

Reporting jobs from Mumbai can be configured with readPreference: secondary to **offload read traffic** from the primary.

3.4 Write Concerns – Testing Safety Levels

In mongosh:

```
db.filings.insertOne(
  { entity_id: 1002, filing_type: "Franchise Tax", state: "CA", status:
"OPEN", amount: 8000 },
  { writeConcern: { w: 1 } }
)
```

This only waits for **primary**.

Now try majority:

```
db.filings.insertOne(
  { entity_id: 1003, filing_type: "Annual Report", state: "NY", status: 
"OPEN", amount: 12000 },
  { writeConcern: { w: "majority", j: true } }
)
```

This waits until a **majority of nodes** have the write and it's **journaled**.

✅ **CSC use case explanation**

For critical filings, CSC uses w: "majority", j: true so a write is guaranteed to be on at least 2 data-bearing nodes *and* durable on disk before the application proceeds.

---

4. Replication in MongoDB Atlas (Concept + Quick Hands-On)

In Atlas, **replication is built-in**. A standard Atlas cluster is already a replica set.

4.1 View Replica Set Members

1. Log in to **Atlas**.
2. Go to your project → open cluster (e.g., csc-compliance-prod).
3. Click **Connect** → **Connect with MongoDB Shell** to get the mongosh URI.
4. Connect:

```
mongosh 
"mongodb+srv://csc_app_user@cluster0.xxxxx.mongodb.net/csc_compliance"
```

5. Check replica set config:

```
rs.status()
```

You'll see the Atlas nodes (3+ members).

4.2 Test Read Preference on Atlas

In mongosh (Atlas):

```
use csc_compliance
db.getMongo().getReadPref()   // should show primary

db.getMongo().setReadPref("secondaryPreferred")
db.filings.find().limit(1)
```

In real apps (Java, Node, .NET), you set read preference in the **driver connection string** or code.

4.3 Test Failover on Atlas (Carefully – Use Non-Prod)

In Atlas UI:

1. Go to your cluster.
2. Click the **…** menu → **Test Failover** (or similar action, depending on UI and permissions).
3. Atlas will **step down** the primary and elect a new one.
4. In mongosh, watch:

```
while (true) {
  printjson(rs.isMaster());
  sleep(2000);
}
```

You'll see isWritablePrimary flip from one node to another.

✅ **CSC use case explanation**

This is how CSC tests **application resilience**: the app should automatically reconnect to the new primary during Atlas failover tests.

---

5. How It Looks Across Platforms (Quick Summary Table)

Platform	What You Do for This Lab
Linux	Run 3 mongod processes with different ports and data dirs, initialize replica set, test elections.
Windows	Same idea, but with C:\csc-repl paths and cmd/PowerShell windows.
Mac	Same as Linux, usually installed via Homebrew, using ~/csc-repl.
Atlas	Replica set is auto-managed. You inspect with rs.status(), set readPreference in the driver, and use “Test Failover” in UI.

---

6. Suggested Exercise Flow for a 90-Minute Session

1. **(15 min)** Explain replica set architecture + primary/secondary/arbiter.
2. **(20–30 min)** Lab 1: Set up 3-node replica set (Windows/Mac/Linux).
3. **(20 min)** Lab 2: Insert data, test secondary reads, step down primary, observe elections.
4. **(10–15 min)** Explain read preferences & write concerns; try a couple of combinations.
5. **(15–20 min)** Atlas: Show rs.status(), read preferences, and a test failover using Atlas UI (non-prod cluster).
