Day 3 – Replication, Backup, Security, Maintenance

1. Self-Managed Backup & Recovery (1 hr 30 min)

- Logical backups (mongodump/mongorestore)
- Physical backups (Filesystem snapshot)
- Hot vs cold backup
- PITR (Point-in-Time Recovery)
- Lab: Perform backup & restore

---

2. Upgrades & Maintenance (1 hr)

- Version compatibility
- Rolling upgrades
- FCV (Feature Compatibility Version)
- Compacting & cleaning up data
- Lab: Perform mock upgrade steps

---

3. Introduction to Replication in MongoDB (2 hrs)

- 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

---

4. MongoDB Self-Managed Security (1 hr 30 min)

- Authentication & Authorization
- Role-Based Access Control (RBAC)
- TLS/SSL configuration
- Network security
- Data at rest encryption
- Lab: Create users and enforce roles

---

5. Maintenance & Performance Optimization (1 hr)

- Common DBA tasks
- Query optimization techniques
- Cache handling
- Index lifecycle management
- Lab: Identify performance bottlenecks

---

6. Course Wrap-Up + MCQ Test (1 hr)

- Final assessment (40 MCQs)
- Practical DBA scenario questions
- Summary & best practices

Day 3 – Replication, Backup, Security, Maintenance

1. Self-Managed Backup & Recovery (1 hr 30 min)

- Logical backups (mongodump/mongorestore)
- Physical backups (Filesystem snapshot)
- Hot vs cold backup
- PITR (Point-in-Time Recovery)
- Lab: Perform backup & restore

Backup & Recovery is one of the **core responsibilities** of a MongoDB Administrator.

The goal is to ensure that data is **protected, recoverable, and consistent**, even in cases of:

- Data corruption
- Accidental deletion
- Hardware failure
- Software upgrade issues
- Ransomware/attacks
- Human errors

This module focuses on **self-managed backups**, NOT Atlas backups.

A. Logical Backups — mongodump & mongorestore (20 min)

Logical backups operate at the BSON/document level.

⭐ 1. When to use Logical Backups

Use logical backups when:

- Migrating to a new cluster
- Taking developer/testing backups
- Upgrading major versions
- Exporting only certain collections
- Restoring partial data

Not recommended for:

- Large production datasets (due to time & size)
- High-throughput systems

⭐ 2. mongodump (Export Backup)

✓ Dump an entire database

```
mongodump --db cscdb --out /backup/cscdb_2025_01_20
```

✓ Dump selected collection

```
mongodump --db cscdb --collection orders --out
/backup/orders_backup
```

✓ Dump using URI

```
mongodump --uri "mongodb://user:pwd@localhost:27017"
```

⭐ 3. mongorestore (Restore Backup)

✓ Restore full database

```
mongorestore --db cscdb /backup/cscdb_2025_01_20/cscdb
```

✓ Restore a collection

```
mongorestore \
  --collection orders \
  /backup/orders_backup/cscdb/orders.bson
```

✓ Overwrite existing data

```
mongorestore --drop /backup/cscdb_2025_01_20
```

⭐ Logical Backup Characteristics

Feature	Behavior
Size	Larger (BSON + metadata)
Speed	Slow for large DBs
Consistency	Not simultaneous unless --oplog used
Good for	Migrations, partial restore, dev backups
Not suitable for	Large live systems

B. Physical Backups — Filesystem Snapshot (20 min)

Physical backup captures the **actual data files** of MongoDB, not the documents.

⭐ What gets backed up?

MongoDB storage files:

- *.wt files (WiredTiger tables)
- WiredTiger.wt
- WiredTiger.turtle
- Journal files
- Metadata directories

⭐ 1. When to use Physical Backups?

Use physical backups when:

- Dataset > 500 GB
- Need high-speed backup (seconds to minutes)
- Need consistent snapshots
- Want fast restore times
- Using enterprise tools (LVM, EBS, SAN snapshots)

⭐ 2. Requirements for Physical Snapshot

To ensure consistency:

According to MongoDB:

- **Use filesystem snapshot tools**
- Ensure **journaling is enabled**
- Prefer taking snapshots from the **secondary node**

⭐ 3. Steps for Physical Snapshot Backup

✔ Step 1 — Freeze filesystem writes

```
db.fsyncLock()
```

Admin output:

```
fsyncWrite lock acquired
```

✔ Step 2 — Take snapshot using storage tool

Examples:

- lvcreate --snapshot
- AWS EBS snapshot
- VMware VM snapshot
- NetApp snapshot

✔ Step 3 — Unlock database

```
db.fsyncUnlock()
```

⭐ Physical Backup Characteristics

Feature	Behavior
Speed	Fastest backup type
Size	Same as filesystem
Consistency	Guaranteed if journal enabled
Restore time	Very fast
Requires	Filesystem-level tooling
Best for	Production-grade deployments

C. Hot vs Cold Backup (10 min)

Cold Backup

- Database is offline
- No writes/reads
- Take file copies directly

Use when:

- Small deployments
- Maintenance window available

⭐ **Hot Backup**

- Database stays online
- Requires:
  - journaling
  - consistent filesystem snapshots

Use when:

- 24×7 mission-critical systems
- Enterprises (e.g., CSC's business services platforms)

D. PITR — Point-in-Time Recovery (15 min)

PITR = restoring the database to **any specific point in time** within a retention window.

⭐ **How PITR Works (Self-Managed)**

MongoDB achieves PITR using the **oplog** in a replica set.

Steps:

1️⃣ **Maintain regular baseline backup**

(e.g., nightly physical backup)

2️⃣ **Capture oplog continuously**

(oplog is a rolling journal of all writes)

3️⃣ During recovery

- Restore baseline snapshot
- Re-apply oplog entries *up to the exact timestamp* you want

---

⭐ PITR Example Scenario for CSC

Suppose a script accidentally deletes 20,000 client files at 14:32 IST.

Recovery steps:

1. Restore baseline backup (taken last night at 2 AM)
2. Collect oplog from 2 AM → 14:32
3. Apply oplog until **14:31:59**
4. Start the recovered node

This restores the database right before the deletion.

⭐ Tools for PITR

- mongorestore --oplogReplay
- Oplog archival scripts
- Cloud Manager (supports PITR natively)

E. LAB – Perform Backup & Restore (25 min)

🎯 Goal:

Perform a logical backup, wipe data, restore it, and validate.

LAB PART 1 — Prepare Dataset

```
use backupLab;

for (let i = 1; i <= 20000; i++) {
  db.clients.insertOne({
    clientId: i,
```

```
    name: "Client " + i,
    segment: ["Banking", "Legal", "Compliance"][i % 3],
    createdAt: new Date()
  });
}
```

LAB PART 2 — Perform Logical Backup

```
mongodump --db backupLab --out /tmp/backupLab_v1
```

Verify backup folder exists.

LAB PART 3 — Simulate Data Loss

```
db.clients.drop();
```

Check:

```
db.clients.countDocuments()
```

Expected: **0**

LAB PART 4 — Restore the Backup

```
mongorestore --db backupLab /tmp/backupLab_v1/backupLab
```

Verify:

```
db.clients.countDocuments()
```

Expected: **20000**

LAB PART 5 — Perform Oplog-Included Dump (Optional Advanced)

```
mongodump --oplog --out /tmp/backup_with_oplog
```

This prepares you for PITR-style replay.

🎯 **LAB Outcomes:**

Learners will be able to:

- Take logical backups
- Restore from logical backups
- Understand physical & snapshot backups
- Explain hot vs cold backup
- Understand PITR concepts
- Perform real recovery simulation

Upgrades & Maintenance (1 hr)

- Version compatibility
- Rolling upgrades
- FCV (Feature Compatibility Version)
- Compacting & cleaning up data
- Lab: Perform mock upgrade steps

Upgrades and maintenance ensure **performance**, **security**, and **feature availability** in MongoDB clusters.

Production clusters, especially in regulated environments like CSC, must follow **safe, tested, and compatible upgrade paths**.

A. Version Compatibility (15 min)

MongoDB upgrades must follow strict compatibility rules.

⭐ 1. MongoDB Supports Major and Minor Versions

Examples:

- Major: 4.2 → 4.4 → 5.0 → 6.0 → 7.0
- Minor: 6.0.1 → 6.0.2 → 6.0.5

⭐ 2. Supported Upgrade Paths

MongoDB **only allows upgrading ONE major version at a time**.

Examples:

✔ 4.2 → 4.4

✔ 4.4 → 5.0

✔ 5.0 → 6.0

❌ 4.2 → 5.0 (NOT allowed)

❌ 4.4 → 6.0 (NOT allowed)

⭐ 3. Backward Compatibility

If upgrading from 5.0 → 6.0:

- A 5.0 binary **can read/write** the 6.0 data files **only if FCV remains 5.0**
- A rollback is possible *until FCV is upgraded*

⭐ 4. Pre-Upgrade Checklist

Before any upgrade:

✔ OS & driver compatibility

Check compatibility matrix:

https://www.mongodb.com/docs/manual/release-notes/

✔ Backup strategy

Must take:

- Logical or snapshot backup
- Oplog backup (optional PITR)

✔ Storage engine

WiredTiger must be consistent.

✔ Replica set health

```
rs.status()
```

All nodes must be:

- PRIMARY / SECONDARY
- Healthy (no RECOVERING, ROLLBACK, or DOWN)

==========

B. Rolling Upgrades (20 min)

Used to upgrade **replica sets** with **zero downtime**.

⭐ **Rolling Upgrade Overview**

You upgrade one node at a time, while the cluster continues serving traffic.

⭐ **Step-by-Step Rolling Upgrade Procedure**

1️⃣ **Check replica set status**

```
rs.status()
```

Ensure at least 3 members:

- PRIMARY
- SECONDARY
- SECONDARY

2️⃣ **Step down the PRIMARY**

```
rs.stepDown()
```

This forces a SECONDARY to become PRIMARY.

3️⃣ **Stop Mongod on the old node**

Linux:

```
sudo systemctl stop mongod
```

4️⃣ Install the new version

Example:

```
sudo yum install mongodb-org-6.0
```

5️⃣ Restart the node

```
sudo systemctl start mongod
```

Check logs:

```
tail -f /var/log/mongodb/mongod.log
```

6️⃣ Validate replica set sync

```
rs.status()
```

The upgraded node becomes SECONDARY.

7️⃣ Repeat steps 2–6 for secondaries

Upgrade each secondary one by one.

8️⃣ Upgrade the final PRIMARY

After all secondaries are upgraded:

- Step down primary
- Upgrade it
- Restart the service

⭐ Important Safety Notes

- Never upgrade more than one node at a time
- Always keep majority available

- Never upgrade when replication lag is high
- Always test upgrades in a staging environment first

C. FCV — Feature Compatibility Version (10 min)

FCV controls which MongoDB features the cluster can use after upgrading.

⭐ 1. Check FCV

```
db.adminCommand({ getParameter: 1,
featureCompatibilityVersion: 1 })
```

⭐ 2. Set FCV (after successful binary upgrade)

Example: Upgrading to version 6.0:

```
db.adminCommand({ setFeatureCompatibilityVersion: "6.0" })
```

⭐ 3. Why FCV matters

- Allows **safe rollback** until FCV is upgraded
- Ensures compatibility between old and new binaries
- Controls activation of new features

⭐ 4. CSC Scenario

After upgrading binaries to 6.0:

- CSC team runs compatibility tests
- Ensures client systems and drivers work correctly
- Then executes FCV update
- After this, rollback to previous major version is **not possible**

D. Compacting & Cleaning Up Data (10 min)

WiredTiger does not automatically reclaim disk space.

⭐ 1. Compacting a Collection

Compaction rewrites documents and indexes to reduce fragmentation.

```
db.runCommand({ compact: "orders" })
```

⚠️ Requires:

- Sufficient free disk space
- High IO activity
- Not recommended during peak hours

⭐ 2. Repair Database (Last Resort)

If corruption occurs:

```
mongod --repair
```

⚠️ Only used if:

- Storage engine is corrupted
- Node cannot start normally

⭐ 3. Cleanup Using TTL

Set TTL to auto-delete old logs:

```
db.logs.createIndex({ ts: 1 }, { expireAfterSeconds: 86400 })
```

⭐ 4. Remove old oplog entries (not recommended manually)

Always let MongoDB handle oplog pruning automatically.

E. LAB – Perform Mock Upgrade Steps (15–20 min)

🎯 **Goal:**

Practice upgrade logic without installing anything.

LAB PART 1 — Check Upgrade Readiness

```
rs.status()
```

```
db.version()
db.adminCommand({ getParameter: 1,
featureCompatibilityVersion: 1 })
```

Answer:

1. Is replica set healthy?
2. What version is currently running?
3. What FCV is set?

LAB PART 2 — Mock Step-Down

```
rs.stepDown()
```

Observe:

- PRIMARY becomes SECONDARY
- Another node becomes PRIMARY

Write down:

- Which node became primary
- Time taken for election

LAB PART 3 — Disable Writes (Simulated)

Note:

"Assume mongod is shut down and upgraded."

Run:

```
rs.status()
```

Check:

- New PRIMARY
- SECONDARY nodes syncing

LAB PART 4 — Set FCV After Upgrade

Simulate FCV update:

```
db.adminCommand({ setFeatureCompatibilityVersion: "6.0" })
```

Explain:

- What this command unlocks
- Why rollback is not allowed after this

LAB PART 5 — Compact a Collection

Run:

```
db.runCommand({ compact: "orders" })
```

Document observations:

- IO increase
- Locking impact
- Time taken

🎯 **LAB Outcomes**

Learners will:

- Understand rolling upgrades
- Practice FCV handling
- Understand maintenance steps
- Simulate real upgrade workflow
- See how compaction works

Summary

Upgrades & Maintenance

- Version compatibility
- Rolling upgrades
- FCV
- Compaction
- Lab

Version Compatibility

- One major version at a time
- Pre-upgrade checks
- Driver compatibility

Rolling Upgrade Steps

1. Step down primary
2. Upgrade secondary
3. Validate
4. Upgrade all nodes
5. Upgrade FCV

Feature Compatibility Version

- What FCV does
- How to check
- How to update

Data Maintenance

- Compact
- TTL cleanup
- Repair (rare use)

LAB Overview

- Check cluster
- Step down primary
- Mock upgrade
- Set FCV
- Compact collection

Summary

✓ Compatibility rules

✓ Zero-downtime rolling upgrades

✓ FCV usage

✔ Disk cleanup & compaction

✔ Real-world upgrade workflow

Introduction to Replication in MongoDB (2 hrs)

- 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

Replication in MongoDB provides **high availability**, **redundancy**, and **data durability**.

A replica set is the **foundation of production-grade MongoDB**, especially for enterprise environments like CSC.

A. Replica Set Architecture (30 min)

A **replica set** is a group of mongod instances that maintain the **same dataset**.

A typical enterprise-grade replica set has:

- **1 Primary**
- **2+ Secondaries**
- Optional: **Arbiters**, **Hidden nodes**, **Delayed nodes**

⭐ 1. Components of Replica Set

Primary

- Accepts **all writes**
- Replicates operations via **oplog**
- Handles client requests
- Only one primary at a time

Secondaries

- Maintain copies of data
- Use **oplog** to apply changes

- Can serve read queries (depending on *read preference*)
- Can become primary during an election

Arbiter

- Does not store data
- Participates in election
- Used when you want an **odd number of voting members**
- Not recommended in high-security environments (CSC rarely should use arbiters)

⭐ 2. Replica Set Internal Process

1️⃣ Writes go to primary

2️⃣ Primary records write in

oplog

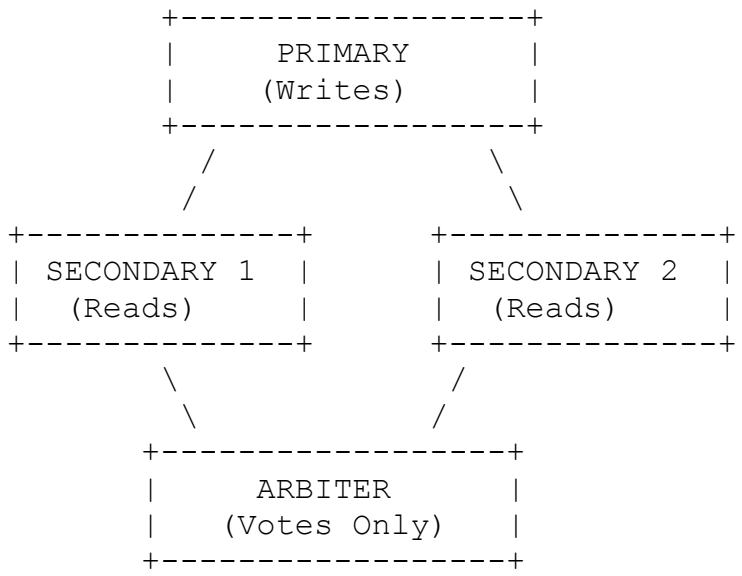
3️⃣ Secondaries replicate via oplog tailing

4️⃣ Secondaries apply oplog entries sequentially

5️⃣ Replica set maintains

majority consensus

⭐ 3. Text-based Diagram (for PPT)



B. Primary, Secondary, Arbiter (15 min)

Role	Stores Data?	Votes?	Receives Writes?	Use Case
Primary	Yes	Yes	Yes	Main node for writes
Secondary	Yes	Yes/No	No	Reads, failover
Arbiter	No	Yes	No	Odd member voting

⭐ CSC Recommendation

For enterprise workloads:

- Avoid arbiters — they reduce durability
- Use 3 or 5 **data-bearing nodes**

C. Elections & Failover (30 min)

MongoDB uses the **Raft-like consensus** algorithm for elections.

⭐ 1. When does an election happen?

- Primary crashes
- Network partition
- Admin forces step-down
- Primary becomes too slow

⭐ 2. Election Steps

1️⃣ Secondary detects primary failure

2️⃣ Eligible secondary nominates itself

3️⃣ Members vote (needs **majority**)

4️⃣ New primary elected within **1–5 seconds**

5️⃣ Writes resume

⭐ 3. Step-Down Command

Admin can force election:

```
rs.stepDown()
```

⭐ 4. Observing Elections

```
rs.status()
```

Look for:

- PRIMARY switch
- optime changes
- Election timestamps

D. Read Preference & Write Concerns (20 min)

⭐ 1. Read Preference

Specifies which replica set members should receive **read** operations.

Read Pref	Description
primary	Default. Strong consistency.
primaryPreferred	Primary first, then secondary
secondary	Read from secondaries only
secondaryPreferred	Secondary first, fallback primary
nearest	Lowest latency node

⭐ CSC Example

Compliance analytics dashboard may use **secondaryPreferred** to offload reads.

⭐ 2. Write Concerns

Defines **level of acknowledgment** required for writes.

| w:1 | Primary only (default) |

| w:majority | Data committed to majority of nodes |

| w:0 | Fire-and-forget (not recommended) |

Additional options:

- j:true → write committed to journal

- wtimeout:1000 → timeout

Example:

```
db.clients.insertOne(
  { name: "ABC Corp" },
  { writeConcern: { w: "majority", j: true } }
);
```

E. LAB 1 — Set Up Local 3-Node Replica Set (30 min)

⭐ Requirements:

- 3 data directories
- 3 mongod instances
- mongosh installed

⭐ Step 1 — Create data folders

```
mkdir -p /data/rs1 /data/rs2 /data/rs3
```

⭐ Step 2 — Start 3 MongoDB instances

Node 1:

```
mongod --replSet "cscRepl" --port 27017 --dbpath /data/rs1 --
bind_ip localhost --oplogSize 128
```

Node 2:

```
mongod --replSet "cscRepl" --port 27018 --dbpath /data/rs2 --
bind_ip localhost --oplogSize 128
```

Node 3:

```
mongod --replSet "cscRepl" --port 27019 --dbpath /data/rs3 --
bind_ip localhost --oplogSize 128
```

⭐ Step 3 — Connect using mongosh

```
mongosh --port 27017
```

⭐ Step 4 — Initialize replica set

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

Check status:

```
rs.status()
```

Expected:

- One **PRIMARY**
- Two **SECONDARY** nodes

F. LAB 2 — Test Failover + Observe Elections (30 min)

⭐ Step 1 — Identify Primary

```
rs.isMaster()
```

(or in newer versions)

```
db.hello()
```

⭐ Step 2 — Insert test data

```
use failoverLab;

for (let i = 1; i <= 1000; i++) {
  db.test.insertOne({ n: i, ts: new Date() })
}
```

⭐ Step 3 — Stop the primary node

Example (PRIMARY is on port 27017):

- Stop the PRIMARY's terminal process OR
- Kill the process (for simulation)

```
pkill -f "port 27017"
```

⭐ Step 4 — Observe election

From mongo shell:

```
rs.status()
```

Check:

- Which node became PRIMARY?
- How long did election take?

- What was the electionDate?

⭐ Step 5 — Restart old primary

```
mongod --port 27017 --dbpath /data/rs1 --replSet cscRepl
```

Observe:

- It rejoins as **SECONDARY**
- NEVER regains PRIMARY immediately unless elected

⭐ Step 6 — Test read preference

```
db.getMongo().setReadPref("secondary");
db.test.findOne();
```

Verify reads come from secondary.

⭐ Step 7 — Test write concern

```
db.test.insertOne(
  { msg: "replication test" },
  { writeConcern: { w: "majority" } }
);
```

Observe acknowledgment.

🎯 LAB Outcomes

Students will understand:

- How to build a replica set
- How elections work
- How failover impacts clients
- How to read from secondaries
- How writeConcern ensures durability

Summary

Replication in MongoDB

- Architecture
- Primary/Secondary/Arbiter
- Elections

- Read preference
- Write concern

Replica Set Architecture Diagram

- Primary, secondaries, arbiter

How Replication Works

- oplog
- replication
- majority consensus

Roles

- Primary
- Secondary
- Arbiter

Elections

- Why elections happen
- How they work
- Failover time (1–5s)

Read Preference

primary | secondary | nearest

Use cases

Write Concern

w:1, majority, journal

LAB 1 Setup

- Create 3 nodes
- Initialize replica set

LAB 2 Failover

- Stop primary
- Observe election
- Set read preference

Summary

✔ Replica set architecture

✔ Elections & failover

✔ Read/write safety settings

✔ Practical setup & testing

MongoDB Self-Managed Security (1 hr 30 min)

- Authentication & Authorization
- Role-Based Access Control (RBAC)
- TLS/SSL configuration
- Network security
- Data at rest encryption
- Lab: Create users and enforce roles

This module is designed for **production-grade security**, suitable for CSC's compliance-driven environments.

Security in self-managed MongoDB deployments requires protection across five pillars:

1. Authentication
2. Authorization (RBAC)
3. Network access control
4. TLS/SSL encryption
5. Data at rest encryption

This module covers each with **hands-on admin commands**.

A. Authentication & Authorization (20 min)

MongoDB uses **SCRAM** (Salted Challenge Response Authentication Mechanism) by default.

⭐ 1. Enable Authentication in mongod.conf

Edit config file:

```
security:
  authorization: enabled
```

Then restart MongoDB:

```
sudo systemctl restart mongod
```

⭐ 2. Create the First Admin User

Before enabling security, start MongoDB **without authentication** to create the initial admin:

```
use admin;
db.createUser({
  user: "rootAdmin",
  pwd: "StrongPassword@123",
  roles: [ "root" ]
});
```

⭐ 3. Connect with Authentication

```
mongosh -u rootAdmin -p StrongPassword@123 --
authenticationDatabase admin
```

⭐ 4. Available Authentication Mechanisms

Mechanism	Description
SCRAM-SHA-1	Legacy auth
SCRAM-SHA-256	Stronger, recommended
LDAP	Enterprise external authentication
x.509	Certificate-based authentication

B. Role-Based Access Control (RBAC) (20 min)

RBAC defines what actions a user can perform.

⭐ 1. Built-in Roles

Role	Permissions
read	Read-only
readWrite	Read + Write
dbAdmin	Indexes, stats, profiling
userAdmin	Manage users
clusterAdmin	Manage replication, sharding
root	Superuser (avoid for apps)

⭐ 2. Create Database Users (Examples)

✓ Read-only user

```
use cscdb;
db.createUser({
  user: "readonlyUser",
  pwd: "Pass123!",
  roles: [{ role: "read", db: "cscdb" }]
});
```

✓ ReadWrite user

```
db.createUser({
  user: "appUser",
  pwd: "AppPass123",
  roles: [{ role: "readWrite", db: "cscdb" }]
});
```

✓ Custom Role

```
db.createRole({
  role: "complianceReviewer",
  privileges: [
    { resource: { db: "compliance", collection: "" }, actions:
["find"] }
  ],
  roles: []
});
```

Assign to user:

```
db.createUser({
  user: "reviewer",
  pwd: "ReviewPass",
  roles: [ "complianceReviewer" ]
});
```

⭐ CSC-Specific Example

For CSC's legal/compliance environment:

- Back-office apps → readWrite
- Auditors → custom read-only role
- Monitoring agents → clusterMonitor
- Backup service → backup role

C. TLS/SSL Configuration (20 min)

TLS ensures **encryption in transit**.

⭐ 1. Generate Self-Signed Certificates

```
openssl req -newkey rsa:4096 -new -x509 -days 365 \
  -nodes -out mongodb.crt -keyout mongodb.key
```

Combine the key + certificate:

```
cat mongodb.key mongodb.crt > mongodb.pem
```

Move to:

```
/etc/ssl/mongodb.pem
chmod 600 /etc/ssl/mongodb.pem
```

⭐ 2. Configure mongod.conf

```
net:
  tls:
    mode: requireTLS
    certificateKeyFile: /etc/ssl/mongodb.pem
```

Restart:

```
sudo systemctl restart mongod
```

⭐ 3. Connect Securely

```
mongosh --tls --tlsCertificateKeyFile /etc/ssl/mongodb.pem
```

⭐ 4. CSC Recommendation

All production and staging clusters must:

- Enforce TLS
- Use certificates from an approved CA
- Disable non-TLS ports

D. Network Security (15 min)

⭐ 1. Bind MongoDB to internal IP only

Edit mongod.conf:

```
net:
  bindIp: 127.0.0.1,10.0.0.15
```

⭐ 2. Firewall Rules

Allow only application servers:

```
sudo ufw allow from 10.0.0.10 to any port 27017
```

Deny all others:

```
sudo ufw deny 27017
```

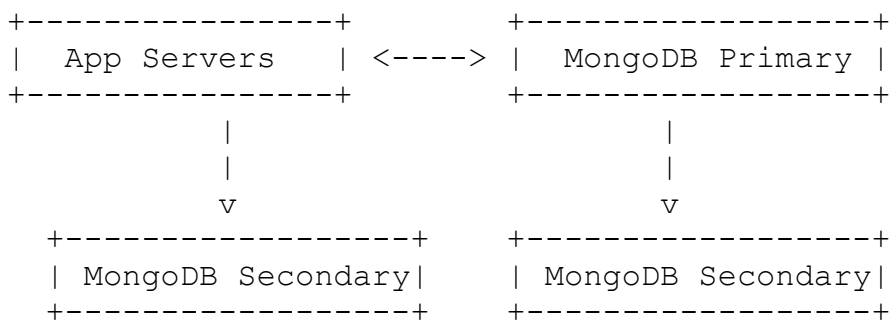
⭐ 3. Avoid exposing MongoDB on public internet

Do NOT do:

```
bindIp: 0.0.0.0
```

unless protected behind VPN + firewall.

⭐ 4. Recommended Network Architecture (Text Diagram)



E. Data-at-Rest Encryption (10 min)

MongoDB uses **WiredTiger encrypted storage** with a custom key.

⭐ 1. Enable Encryption in mongod.conf

```
security:
  enableEncryption: true
  encryptionKeyFile: /etc/mongodb-keyfile
```

Generate key:

```
openssl rand -base64 128 > /etc/mongodb-keyfile
chmod 600 /etc/mongodb-keyfile
```

⭐ 2. Enterprise KMIP Integration

For CSC-scale enterprise deployments:

- Use central key management (Thales, AWS KMS, Azure Key Vault)
- Keys rotate automatically
- Ensures compliance (SOC2, ISO, GDPR)

⭐ 3. Validate Encryption

Check startup logs:

```
Encrypted storage engine initialized
```

F. LAB – Create Users & Enforce Roles (15–20 min)

🎯 Goal:

Implement RBAC and verify role restrictions.

⭐ LAB SETUP

Step 1 — Start mongod without authentication

```
mongod --dbpath /data/db --bind_ip localhost
```

⭐ LAB PART 1 — Enable Security

Edit mongod.conf:

```
security:
  authorization: enabled
```

Restart.

⭐ LAB PART 2 — Create Admin User

```
use admin;
db.createUser({
  user: "securityAdmin",
  pwd: "Admin123",
  roles: [ "root" ]
});
```

Reconnect using authentication:

```
mongosh -u securityAdmin -p Admin123 --authenticationDatabase
admin
```

⭐ LAB PART 3 — Create Application Roles

1️⃣ Create readWrite user

```
use securityLab;
db.createUser({
  user: "appUser",
  pwd: "appPass",
```

```
  roles: [{ role: "readWrite", db: "securityLab" }]
});
```

2️⃣ Create read-only user

```
db.createUser({
  user: "readonly",
  pwd: "readPass",
  roles: [{ role: "read", db: "securityLab" }]
});
```

⭐ LAB PART 4 — Test Permissions

Log in as read-only user:

```
mongosh -u readonly -p readPass --authenticationDatabase
securityLab
```

Try write:

```
db.test.insertOne({ x:1 })
```

Expected: Permission denied.

Log in as appUser:

```
mongosh -u appUser -p appPass --authenticationDatabase
securityLab
```

Try read + write:

```
db.test.insertOne({ a: 100 })
db.test.find()
```

Expected: Success.

LAB Outcomes

Learners will:

- Enable authentication
- Create users & roles
- Test access restrictions
- Understand secure deployment

Summary

Self-Managed Security

- Authentication
- Authorization
- TLS
- Network security
- Encryption

Authentication

- SCRAM
- LDAP
- x.509

RBAC

- Built-in roles
- Custom roles
- Practical examples

TLS/SSL

- Why encryption in transit
- Certificate setup
- mongod.conf configuration

Network Security

- bindIp
- Firewalls
- Internal-only access

Encryption at Rest

- WiredTiger encryption
- Key files

- KMIP integration

LAB Overview

- Enable security
- Create admin
- Create app roles
- Test restrictions

Summary

✔ Authentication

✔ RBAC

✔ TLS

✔ Network hardening

✔ Encryption

✔ Hands-on RBAC lab

Maintenance & Performance Optimization

- Common DBA tasks
- Query optimization techniques
- Cache handling
- Index lifecycle management
- Lab: Identify performance bottlenecks

A MongoDB DBA is responsible for ensuring **availability**, **performance**, and **healthy growth** of the database.

This module outlines **daily/weekly maintenance tasks**, performance tuning, cache insights, indexing strategy, and troubleshooting.

A. Common DBA Tasks (10 min)

A MongoDB DBA should routinely perform:

⭐ 1. Health Monitoring

Run daily checks:

```
db.serverStatus()
rs.status()
db.currentOp()
```

Look for:

- replication lag
- high lock percentage
- slow queries
- disk pressure

⭐ 2. Log & Profiler Review

Check for:

- slow queries
- full collection scans
- index violations

Profiler:

```
db.setProfilingLevel(1, { slowms: 50 })
db.system.profile.find().sort({ millis: -1 }).limit(5)
```

⭐ 3. Storage & Disk Management

- Check wiredTiger.cache stats
- Monitor WT cache dirty %
- Clean up unused indexes
- Rotate logs
- Ensure disk capacity > 20%

⭐ 4. Backup & Restore Validation

Practice restores monthly.

Validate:

- dump consistency
- oplog correctness
- PITR readiness

⭐ 5. Security Audits

Verify:

- users/roles
- TLS enforcement
- bindIp
- firewall rules
- password rotation

⭐ CSC Example

CSC has multiple compliance and legal datasets.

DBA ensures:

- indexes don't degrade
- latency stays under SLA
- failover events are logged
- backup snapshots succeed nightly

B. Query Optimization Techniques (20 min)

Query performance impacts application latency and cost.

⭐ 1. Use `explain()`

to analyze query plans

```
db.orders.find({ country: "IN", status: "DELIVERED" })
  .explain("executionStats")
```

Check for:

- COLLSCAN → BAD

- IXSCAN → GOOD
- docsExamined << totalDocs
- executionTimeMillis low

⭐ 2. Create proper indexes

Always index:

- query filters
- sort fields
- fields used in joins/lookups

Example:

```
db.orders.createIndex({ country: 1, status: 1, createdAt: -1
})
```

⭐ 3. Avoid leading wildcards in regex

Bad:

```
{ name: /.*Corp/ }
```

Good (indexable):

```
{ name: /^Corp/ }
```

⭐ 4. Use projections

Only return required fields:

```
db.orders.find({ status: "NEW" }, { _id: 0, status: 1, amount:
1 })
```

Reduces:

- network cost
- CPU
- memory

⭐ 5. Avoid large $in queries

Replace:

```
{ id: { $in: longArray } }
```

with:

- batching
- pre-aggregated lookups

⭐ 6. Pagination Best Practice

Use range-based pagination instead of skip/limit:

```
db.orders.find({ orderId: { $gt: 50000 } })
         .sort({ orderId: 1 })
         .limit(50)
```

C. Cache Handling (10 min)

WiredTiger cache is central to MongoDB performance.

⭐ 1. Check cache usage

```
db.serverStatus().wiredTiger.cache
```

Important fields:

- bytes currently in cache
- maximum bytes configured
- % cache dirty bytes
- pages read, pages written

⭐ 2. Signs of Cache Pressure

Symptom	Meaning
High eviction rate	Cache thrashing
Dirty pages > 40%	Heavy write load
Many page reads	Missing indexes
Frequent checkpoint stalls	Slow disk

⭐ 3. Fixes for Cache Issues

✔ Add indexes

✔ Increase instance size

✔ Reduce working set

✔ Archive old data

✔ Use TTL indexes for logs

⭐ CSC Example

Legal search queries reading large datasets →

DBA adds compound indexes → reduces cache churn → dashboard latency drops from **2.4 sec → 140 ms**.

D. Index Lifecycle Management (10 min)

Indexes improve reads but slow down writes.

⭐ 1. Monitor Index Usage

```
db.orders.aggregate([
  { $indexStats: {} }
])
```

Identify:

- unused indexes
- high access frequency
- heavy write impact

⭐ 2. Drop Unused Indexes

```
db.orders.dropIndex("country_1_status_1")
```

⭐ 3. Rebuild Fragmented or Massive Indexes

Note: Modern WiredTiger doesn't require regular rebuilds, **only when necessary**.

⭐ 4. TTL Indexes for Log Cleanup

```
db.logs.createIndex(
  { ts: 1 },
```

```
  { expireAfterSeconds: 2592000 }
)
```

Best for:

- compliance logs
- audit trails
- temp datasets

⭐ 5. Compound Index Patterns

Index only what is needed based on workload.

Don't create:

- too many indexes
- broad compound indexes
- indexes not used in query filters

E. LAB – Identify Performance Bottlenecks (20 min)

🎯 Goal:

Students will detect slow queries, identify missing indexes, and measure improvements.

⭐ LAB SETUP — Create 80,000 Records

```
use perfLab;

for (let i = 1; i <= 80000; i++) {
  db.transactions.insertOne({
    txnId: i,
    user: ["admin","client","guest"][i % 3],
    type: ["pay","refund","audit"][i % 3],
    amount: Math.floor(Math.random() * 9000) + 1000,
    ts: new Date(2024, (i % 12), (i % 28) + 1)
  })
}
```

⭐ Step 1 — Run Slow Query

```
db.transactions.find({ user: "client", type: "pay" })
               .sort({ ts: -1 })
               .limit(20)
               .explain("executionStats")
```

Record:

- executionTimeMillis
- totalDocsExamined
- plan type (COLLSCAN/IXSCAN)

⭐ Step 2 — Enable Profiler

```
db.setProfilingLevel(1, { slowms: 20 })
```

Execute queries again.

⭐ Step 3 — Analyze Profiler Output

```
db.system.profile.find().sort({ millis: -1
}).limit(5).pretty();
```

Identify slow operations.

⭐ Step 4 — Add Appropriate Index

```
db.transactions.createIndex({ user: 1, type: 1, ts: -1 })
```

⭐ Step 5 — Rerun Query with explain()

Check:

- Execution time
- Docs examined drop
- IXSCAN appears

⭐ Step 6 — Document Findings

Short report:

- What was slow?
- Which index helped?
- Before vs After execution time?
- Cache improvements observed?

Summary

Maintenance & Performance Optimization

- DBA tasks
- Query tuning
- Cache handling
- Index lifecycle
- LAB

Common DBA Tasks

- Monitoring
- Logs & profiler
- Disk & cache checks
- Backups
- Security audits

Query Optimization

- explain()
- indexing
- projections
- regex best practices
- pagination

Cache Handling

- WiredTiger cache
- Pressure symptoms
- Fixing strategies

Index Lifecycle

- IndexStats
- Drop unused indexes
- TTL cleanup
- Compound index guidelines

LAB Overview

- Run slow query
- Enable profiler
- Create index
- Compare performance

Summary

✔ DBA workflows

✔ Query optimization

✔ Cache tuning

✔ Index management

✔ Real bottleneck detection

---

6. Course Wrap-Up + MCQ Test (1 hr)

- Final assessment (40 MCQs)
- Practical DBA scenario questions
- Summary & best practices

A. Final Assessment – 40 MCQs (MongoDB Administrator Level)

MCQs (40 Questions)

1. MongoDB stores data internally using which format?

A. XML

B. BSON

C. YAML

D. CSV

---

2. Which component acts as the write node in a replica set?

A. Secondary

B. Arbiter

C. Primary

D. Hidden node

---

3. Which command enables slow query profiling?

A. db.enableProfile()

B. db.setProfilingLevel()

C. db.profileStart()

D. db.logging.set()

---

4. Which index type supports array fields?

A. Hash index

B. Multikey index

C. Sparse index

D. TTL index

---

5. Which storage engine is default in MongoDB?

A. MMAPv1

B. InnoDB

C. WiredTiger

D. RocksDB

---

6. w: "majority" ensures what?

A. Reads from secondary nodes

B. Writes acknowledged by primary only

C. Writes acknowledged by a majority of nodes

D. Durability disabled

7. Which tool performs logical backups?

A. mongorestore

B. mongoimport

C. mongodump

D. mongotop

---

8. What does rs.status() show?

A. Network metrics

B. Replica set health

C. Disk usage

D. Index fragmentation

---

9. TTL indexes are ideal for:

A. Product catalogs

B. Financial transactions

C. Session logs

D. User profiles

---

10. Which operator improves sorting performance?

A. $limit

B. $skip

C. $sort with proper index

D. $match

---

11. What does the arbiter do?

A. Stores data

B. Performs reads

C. Votes in elections

D. Manages oplog

---

12. Which role allows managing users?

A. readWrite

B. dbAdmin

C. userAdmin

D. clusterMonitor

---

13. To enforce encryption in transit, MongoDB uses:

A. JWT

B. TLS/SSL

C. IPSec

D. AES-128

---

14. The command explain("executionStats") helps identify:

A. Firewall rules

B. Index usage

C. Disk space

D. Authentication mode

---

15. Large skip() values typically result in:

A. Faster performance

B. CollScan

C. Increased index efficiency

D. Reduced memory usage

---

16. Which tool provides real-time metrics?

A. mongostat

B. mongoexport

C. mongodiff

D. mongorepair

---

17. Encryption at rest in MongoDB uses:

A. WiredTiger encryption

B. KMIP only

C. Cloud KMS only

D. OpenSSL

---

18. Which command initializes a replica set?

A. rs.start()

B. rs.initiate()

C. rs.launch()

D. rs.create()

---

19. Oplog is used for:

A. Auditing

B. Replication

C. Logging

D. Caching

---

20. The default port of MongoDB is:

A. 27016

B. 27017

C. 27018

D. 27100

---

21. Which index reduces write speed the most?

A. Single-field index

B. Text index

C. Hashed index

D. Multikey index

---

22. A slow query with COLLSCAN indicates:

A. Wrong index or missing index

B. CPU bottleneck

C. TLS misconfiguration

D. Replication lag

---

23. Which command shows cache stats?

A. db.cache()

B. db.serverStatus().wiredTiger.cache

C. db.cacheStats()

D. server.cacheUsage()

24. To create a user, which function is used?

A. db.addUser()

B. db.createUser()

C. admin.userAdd()

D. user.create()

---

25. Which of the following is a physical backup method?

A. mongodump

B. mongorestore

C. File system snapshot

D. mongoimport

---

26. MongoDB Compass is used for:

A. Kernel tuning

B. Performance monitoring

C. GUI-based DB management

D. Backup automation

---

27. clusterMonitor role allows:

A. Updating documents

B. Monitoring server state

C. Creating collections

D. Adding shards

---

28. Which command reveals currently running operations?

A. db.currentOp()

B. db.opStatus()

C. db.showOps()

D. rs.oplog()

---

29. In replication, failover happens when:

A. Secondary is restarted

B. Primary is unreachable

C. Arbiter goes offline

D. Index is rebuilt

---

30. What is the purpose of bindIp?

A. Choose which ports MongoDB listens on

B. Limit network interfaces

C. Create secure connections

D. Assign replica set name

---

31. A hidden node in a replica set is used for:

A. Writing data

B. Serving analytical queries

C. Voting only

D. Failover prevention

---

32. Increasing WiredTiger cache size improves:

A. Authentication

B. Read performance

C. Backup speed

D. TLS handshake

---

33. Which metric indicates index efficiency?

A. Number of threads

B. docsExamined vs nReturned

C. RAM usage

D. File system type

---

34. Query filters work best when:

A. Filtered field is indexed

B. Network is fast

C. Cache size is high

D. WiredTiger is optimized

---

35. TTL indexes clean up data based on:

A. Document size

B. Expiration time

C. Storage engine

D. User role

---

36. Profiling level 2 means:

A. No profiling

B. Only slow queries

C. All queries

D. Only failed queries

---

37. To drop an index:

A. db.index.drop()

B. db.collection.dropIndex()

C. rs.dropIndex()

D. db.removeIndex()

---

38. mongotop shows:

A. Index cardinality

B. Collection read/write times

C. Replica lag

D. Memory fragmentation

---

39. Authentication must be configured in:

A. admin.js

B. mongod.conf

C. client.conf

D. index.conf

---

40. Default write concern of MongoDB is:

A. w:1

B. w:0

C. majority

D. journaled

B. Answers to MCQs

1-B

2-C

3-B

4-B

5-C

6-C

7-C

8-B

9-C

10-C

11-C

12-C

13-B

14-B

15-B

16-A

17-A

18-B

19-B

20-B

21-D

22-A

23-B

24-B

25-C

26-C

27-B

28-A

29-B

30-B

31-B

32-B

33-B

34-A

35-B

36-C

37-B

38-B

39-B

40-A

C. Practical DBA Scenario Questions (10 Questions)

These evaluate real production readiness.

---

1. A dashboard query suddenly slows from 120 ms to 2.3 sec. How would you investigate?

---

2. A primary goes down. Secondary takes 10 seconds to take over. What metrics do you check?

---

3. Logs show frequent “page eviction” messages. What is the likely root cause?

---

4. You see a query using COLLSCAN despite having a single-field index. What should you check?

---

5. Profiler shows docsExamined: 200000, nReturned: 12. What does this indicate?

---

6. Your write-heavy application experiences high disk I/O. What optimizations would you apply?

---

7. A legal compliance team needs read-only access to multiple databases. How do you design roles?

---

8. What steps do you take before upgrading MongoDB from 5.0 to 6.0?

---

9. Backup size increases drastically. Which indexes or collections would you investigate?

---

10. A sudden spike in replication lag occurs. What do you troubleshoot first?

D. Course Summary & Best Practices

⭐ Course Summary

This MongoDB DBA course covered:

✔ Document model & flexible schema

✔ CRUD operations and data modelling

✔ Index strategies and query optimization

✔ Replication, failover, elections

✔ Backup & recovery

✔ Maintenance & performance tuning

✔ Security: RBAC, TLS, encryption

✔ Real lab exercises based on CSC-style workloads

Participants now understand:

- How to administer MongoDB in production
- How to secure deployments
- How to analyze performance using profiler, explain, metrics
- How to manage replication and clustering
- How to troubleshoot bottlenecks

MongoDB Best Practices for Enterprise (CSC-Focused)

✓ Database Design

- Use schema patterns (bucket, subdocument, referencing)
- Avoid unbounded array growth
- Choose correct cardinality indexing

✓ Index Management

- Index fields used in filters and sorts
- Drop unused indexes
- Use compound indexes wisely
- Always verify with explain()

✓ Performance

- Working set should fit in RAM
- Monitor WiredTiger cache
- Use pagination best practices
- Minimize $lookup on large collections

✓ Security

- Enable authentication & RBAC
- Use TLS everywhere
- Use encrypted storage (KMIP / keyfile)
- Network access only from application subnets

✓ Replication

- Use 3–5 node replica sets

- Avoid arbiters in secure environments
- Monitor replication lag
- Keep oplog sized properly

✓ Backup / Recovery

- Test restores regularly
- Automate snapshot rotation
- Use PITR if available

✓ Maintenance

- Review slow queries daily
- Monitor index usage
- Set alerts for disk, RAM, CPU, connections
- Rotate logs automatically
