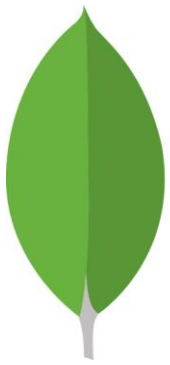


mongoDB

Replication

Prof. P. M. Jadav
Associate Professor
Computer Engineering Department
Faculty of Technology
D. D. University, Nadiad

Introduction

- **Replication** is the process of **synchronizing data across multiple servers**
- In MongoDB, this is achieved using a **replica set** — a group of MongoDB instances that maintain the same data set

Advantages

1. **High Availability**

- **Automatic failover** ensures that if the primary node fails, a secondary is elected to take over
- Applications experience **minimal downtime** during failover
- Useful for mission-critical systems that require **24/7 availability**

Advantages

2. Data Redundancy

- Each node in the replica set has a **copy of the data**
- Even if a server crashes or is corrupted, other replicas retain the data
- Reduces risk of **data loss** due to hardware failures

Advantages

3. Automatic Recovery and Failover

- MongoDB handles failover **automatically** without manual intervention
- The system remains operational even if one or more nodes are offline

Advantages

4. Read Scalability

- You can **offload read operations** to secondary nodes using **read preferences**
- Helps in **balancing load** in read-heavy applications
- Supports geo-distributed reads from the **nearest secondary**

Advantages

5. Backup and Analytics without Impact

- Secondaries can be used for:
  - Backups (e.g., mongodump on secondary)
  - Reporting and analytics queries
- This avoids impacting the primary node and production performance

Advantages

6. Geo-Distributed Deployments

- Replica set members can be distributed across different **data centers** or **regions**
- Enhances:
  - **Disaster recovery**
  - **Read latency reduction** by serving from the nearest replica

Advantages

7. Data Consistency Control

- MongoDB provides write concern levels to control how many nodes must acknowledge a write
- Example:

```
{ writeConcern: { w: "majority" } }
```

- Ensures writes are acknowledged by a majority of replicas before confirming success

Advantages

8. Support for Maintenance and Upgrades

- You can perform **rolling upgrades**:
  - Take down one secondary at a time
  - Upgrade or maintain it
  - Rejoin the replica set
- No need for **complete downtime** during upgrades

Advantages

9. Integration with Sharding

- Replication works seamlessly with MongoDB's **sharding** feature
- Each shard in a sharded cluster is typically a replica set — combining **scalability** with **availability**

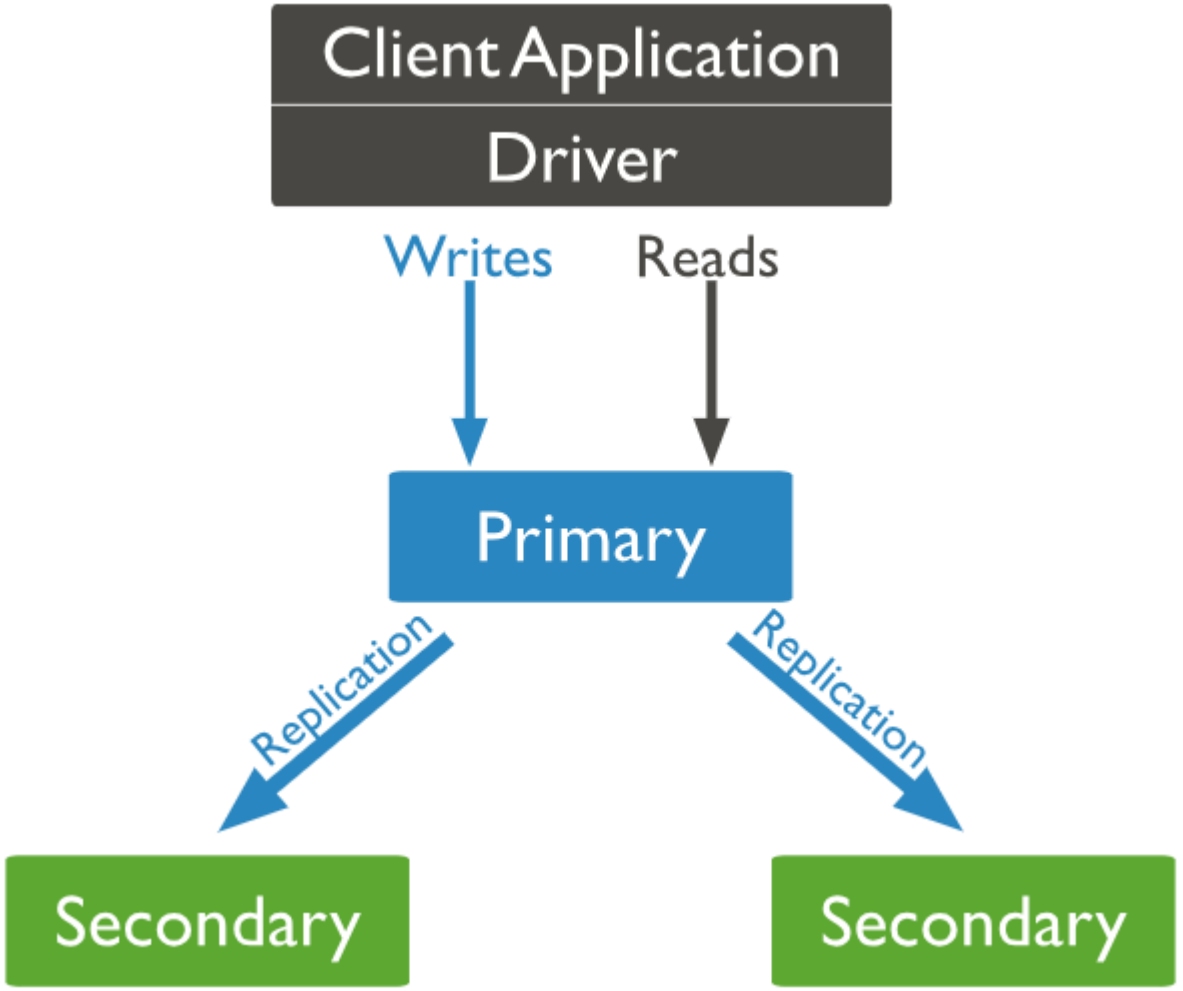
Replica Set Components

Member Type	Description
Primary	The main server that receives all write and read operations (by default)
Secondary	Replicates data from the primary. Used for failover or read scaling
Arbiter	Does not store data. Only participates in elections (to break ties)

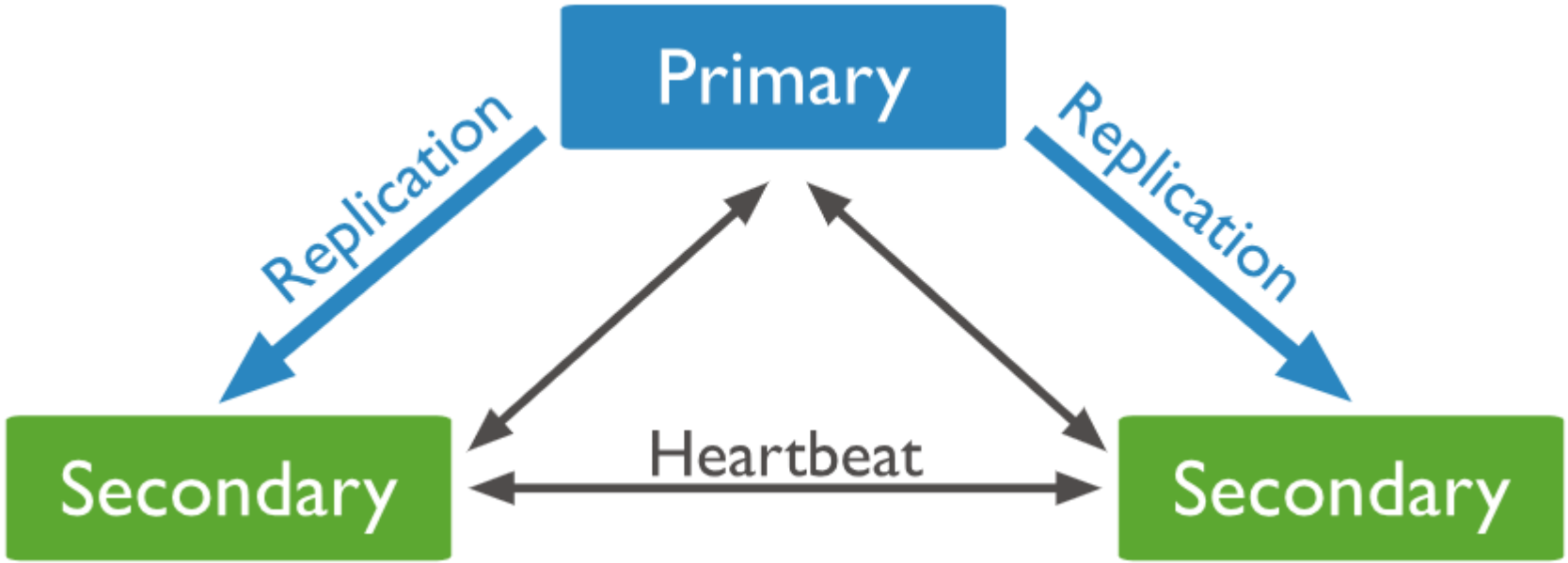
Working of Replication

1) **Write operations** go to the **primary**
2) **Secondaries** replicate the primary's **oplog** (operations log)
3) Each secondary applies the operations to stay in sync with the primary
4) If the **primary fails**, an **election** is triggered to select a new primary
5) Once the failed primary is back online, it re-joins as a **secondary**

Primary


Client Application
Driver
Writes
Reads
Primary
Replication
Replication
Secondary
Secondary

Working of Replication



Operations Log (Oplog)

- Special capped collection: local.oplog.rs
- Records every write operation in order
- Secondaries continuously read from it to replicate changes

Automatic Failover

- If the primary goes down:
  - Replica set holds an **election**
  - A secondary is promoted to **new primary**
  - Writes resume with minimal downtime

Read Preferences

- MongoDB allows **reads from secondaries** based on application needs

Read Preference	Description
primary (default)	Read from primary only
primaryPreferred	Try primary, fall back to secondary
secondary	Read from secondaries only
nearest	Read from node with lowest network latency

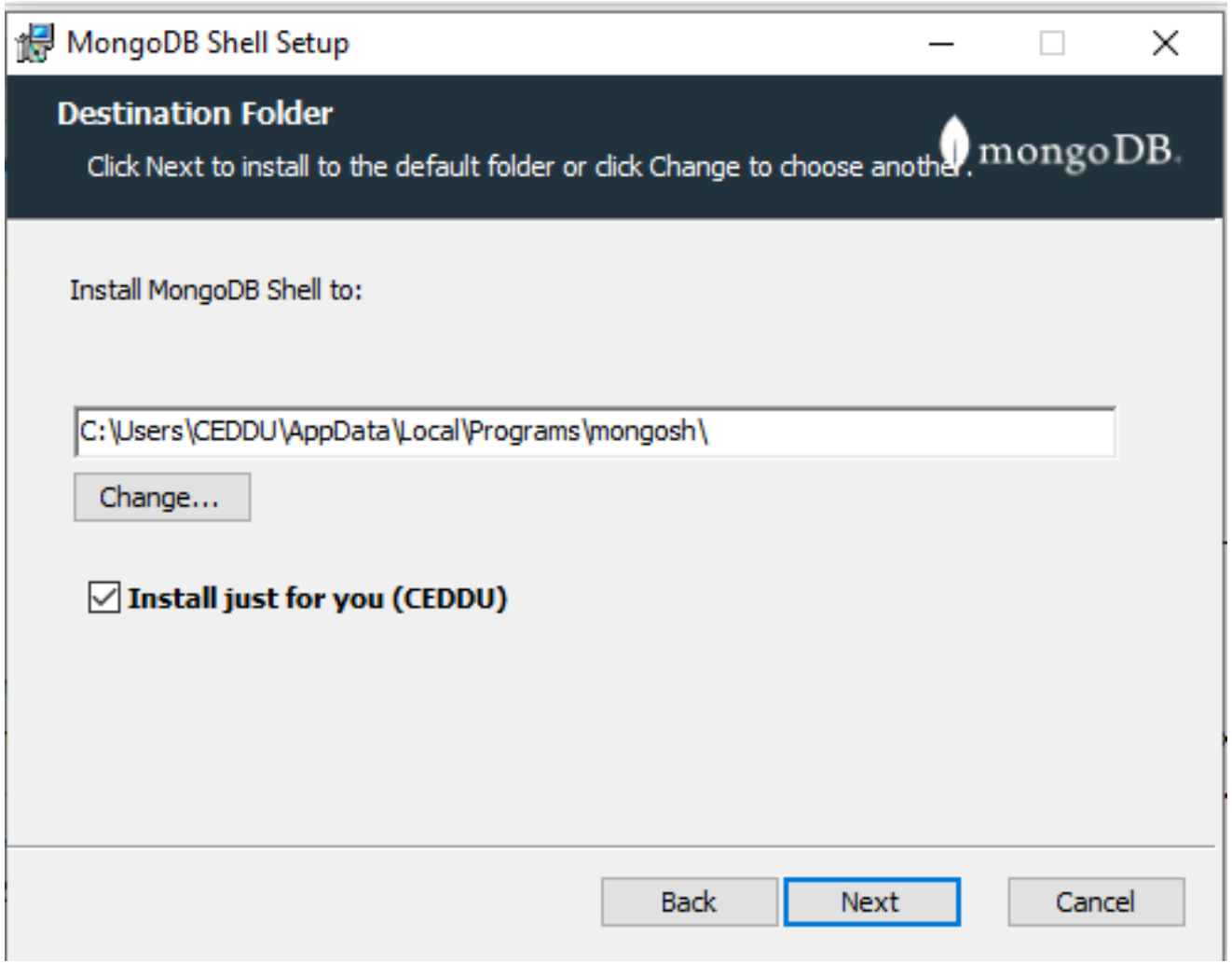
Write Concerns

- Controls **write acknowledgment level**

Write Concern	Description
{ w: 1 }	Acknowledged by primary only
{ w: "majority" }	Acknowledged by majority of nodes
{ w: 0 }	No acknowledgment (fast but risky)

Download and Install MongoDB Shell (CLI)

https://www.mongodb.com/try/download/shell



Setting Up Replica Set (3 Nodes)

1) **Start 3 MongoDB instances (on different ports - in different terminals):**

```
mongod --replSet rs0 --port 27017 --dbpath e:\mongo\db1

mongod --replSet rs0 --port 27018 --dbpath e:\mongo\db2

mongod --replSet rs0 --port 27019 --dbpath e:\mongo\db3
```

Setting Up Replica Set (3 Nodes)

2) Connect to MongoDB shell:

```
C:\Program Files\MongoDB\mongosh-1.5.4-win32-x64\bin>mongosh
Current Mongosh Log ID: 6874c768a9be5d8da859d8bd
Connecting to:          mongodb://127.0.0.1:27017/?directConne
5.4
Using MongoDB:          8.0.11
Using Mongosh:          1.5.4

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-07-14T14:25:11.308+05:30: Access control is not enable
guration is unrestricted
------

test> use admin
switched to db admin
```

Setting Up Replica Set (3 Nodes)

3) Initiate the replica set:

- The rs.initiate() command is used to initialize a replica set
- it starts replication between MongoDB instances and **designates the current node as the primary** (if eligible)

```
test> use admin
switched to db admin
admin> rs.initiate({
...   _id: "rs0",
...   members: [
...     { _id: 0, host: "localhost:27017" }
...   ]
... })
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1752483771, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1752483771, i: 1 })
}
```

Setting Up Replica Set (3 Nodes)

4) Add a second node to replica set:

```
rs0 [direct: primary] admin> rs.add("localhost:27018")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1752483973, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1752483973, i: 1 })
}
```

Setting Up Replica Set (3 Nodes)

5) Add a third node to replica set:

```
rs0 [direct: primary] admin> rs.add("localhost:27019")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1752483990, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1752483990, i: 1 })
}
```

Setting Up Replica Set (3 Nodes)

6) Check the current status of the nodes in the replica set:

```
rs0 [direct: primary] admin> rs.status()
{
  set: 'rs0',
  date: ISODate("2025-07-14T09:07:19.301Z"),
  myState: 1,
  term: Long("1"),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long("2000"),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
```

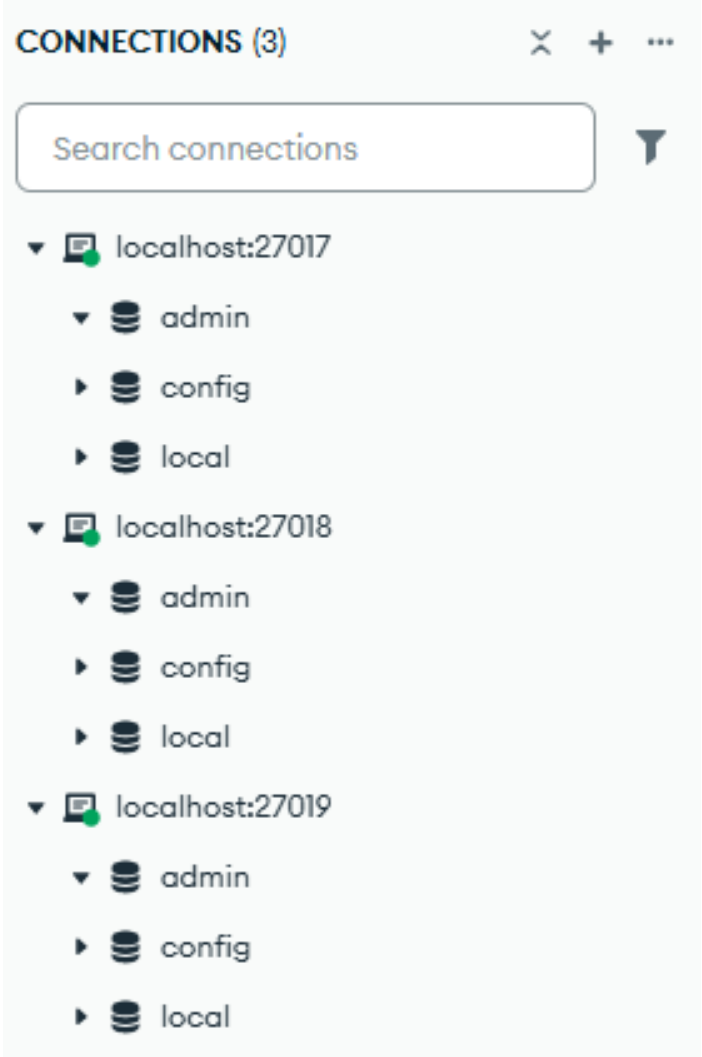
```
members: [
  {
    _id: 0,
    name: 'localhost:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 730,
```

```
{
  _id: 1,
  name: 'localhost:27018',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 66,
```

```
{
  _id: 2,
  name: 'localhost:27019',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 48,
```

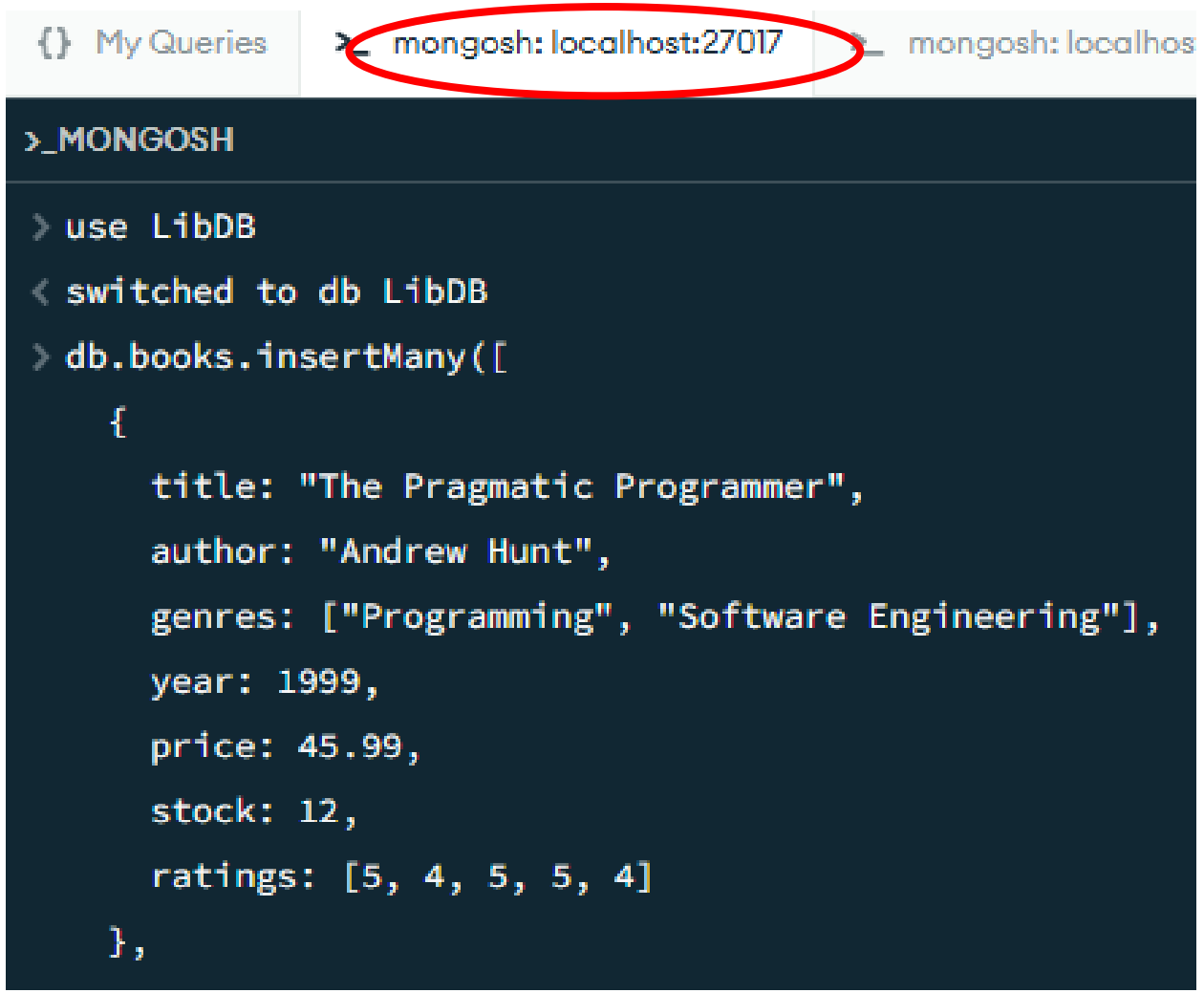
Setting Up Replica Set (3 Nodes)

7) Connect to all three MongoDB instances using Compass:



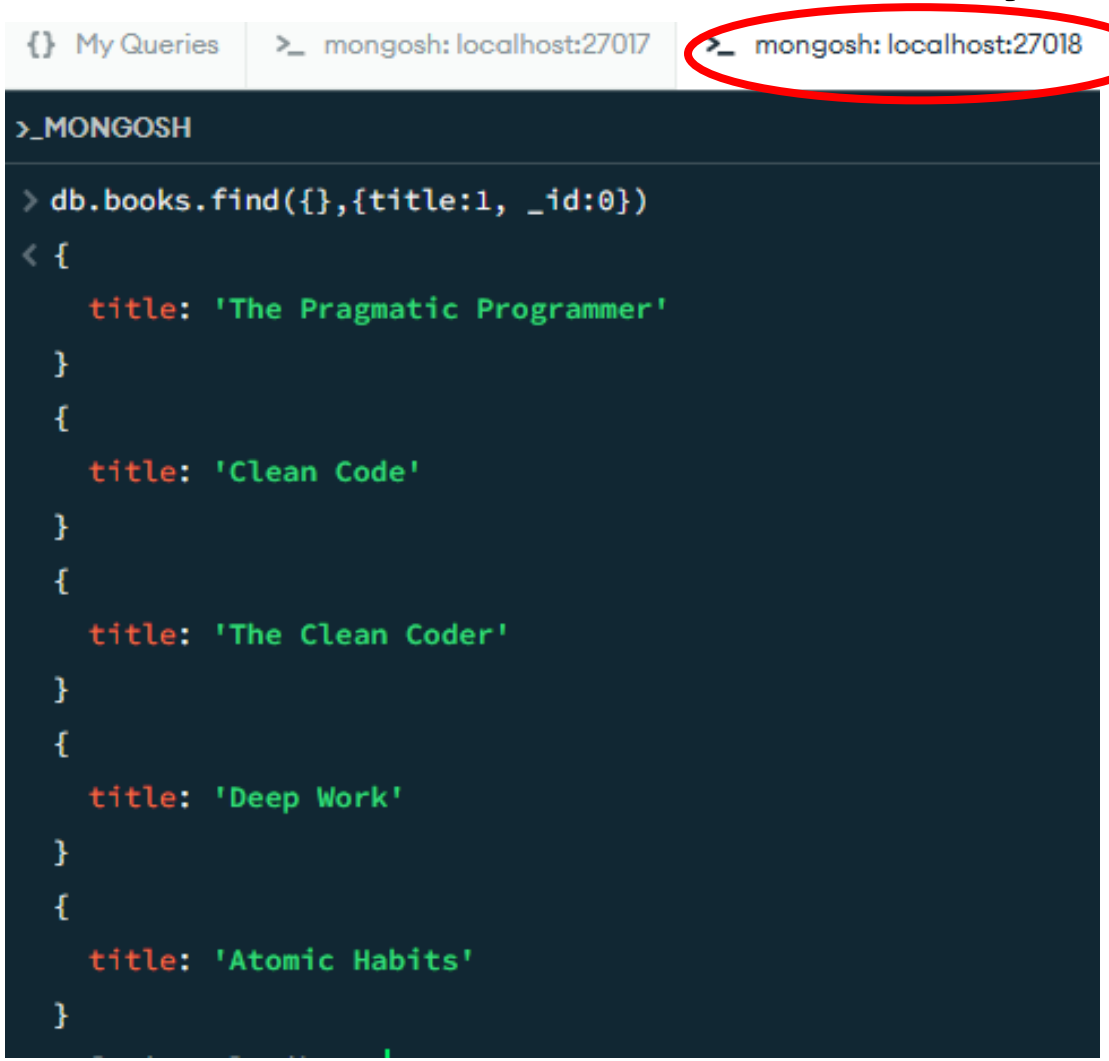
Setting Up Replica Set (3 Nodes)

8) Create a db and collection on primary server (27017) and insert documents:



Setting Up Replica Set (3 Nodes)

9) View the same collection and documents on secondary servers (27018 and 27019):



Setting Up Replica Set (3 Nodes)

10) Shutdown the primary server (27017) by running the following command in compass or you can press Ctrl + C in the terminal where primary server is running:

```
> db.shutdownServer()
```

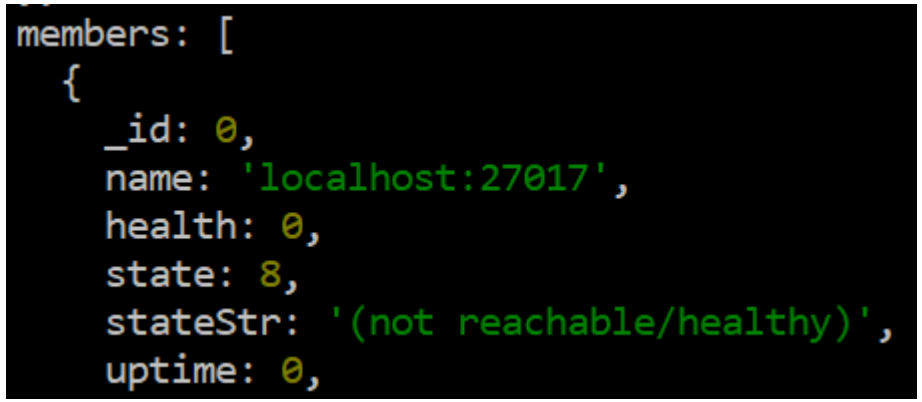
Setting Up Replica Set (3 Nodes)

11) Connect to server (port 27018):

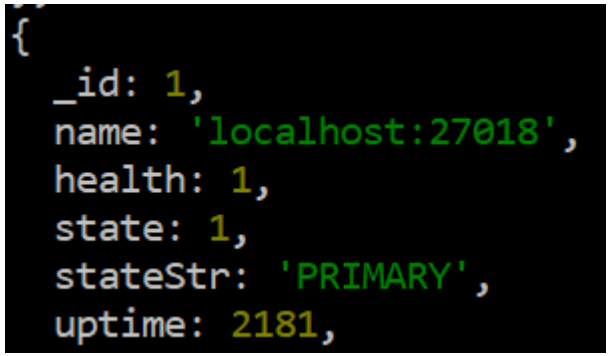
```
C:\Program Files\MongoDB\mongosh-1.5.4-win32-x64\bin>mongosh --port 27018
Current Mongosh Log ID: 6874d024dc1e53c089ad6bd3
Connecting to:          mongodb://127.0.0.1:27018/?directConnection=true&se
5.4
Using MongoDB:          8.0.11
Using Mongosh:          1.5.4
```

Setting Up Replica Set (3 Nodes)

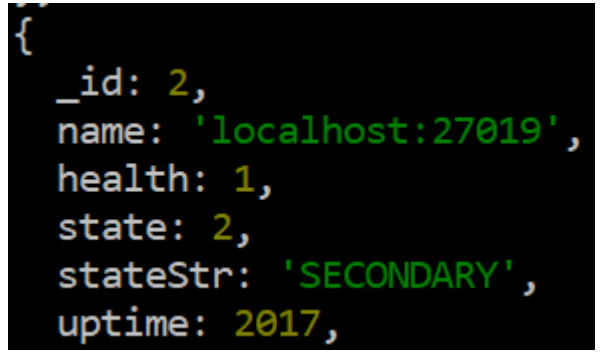
12) Run the rs.status() command to verify new primary server:



```
members: [
  {
    _id: 0,
    name: 'localhost:27017',
    health: 0,
    state: 8,
    stateStr: '(not reachable/healthy)',
    uptime: 0,
```



```
{
  _id: 1,
  name: 'localhost:27018',
  health: 1,
  state: 1,
  stateStr: 'PRIMARY',
  uptime: 2181,
```



```
{
  _id: 2,
  name: 'localhost:27019',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 2017,
```

Purpose of rs.initiate()

1) Start a replica set
   - Converts a standalone MongoDB instance into a **replica set member**
2) Begin configuration
   - Initiates the replica set with the **default or custom configuration**
3) Trigger elections
   - Enables MongoDB to **elect a primary** among members
4) Enable replication
   - Starts replication from primary to secondaries via **oplog**

Replication Commands

- rs.initiate() Initialize replica set
- rs.status() View current replica set status
- rs.add() Add new members to the replica set
- rs.remove() Remove members from the replica set

References

- https://www.mongodb.com/docs/manual/replication/