MongoDB Clustering/Sharding Lab

THIS LAB HAS BEEN WORKED ON BY BOTH MAHNOOR ADEEL(26913) AND ZUHA AQIB(26106). BOTH OF US HAVE USED MAHNOOR'S VM FOR THIS LAB.

Objective

Measure and compare the performance of a single MongoDB instance against a sharded cluster using a 100,000 document dataset.

Verify Setup

```
docker --version
docker-compose --version
```

Lab files structure

```
mongodb-lab/
├── docker-compose.single.yml
├── docker-compose.cluster.yml
└── scripts/
    ├── init_single.py
    └── init_cluster.py
```

Ensure docker-compose files are in root and Python scripts are in the scripts/ folder.

Architecture Overview

Single Instance Setup

Application → MongoDB → All Data
Drawback: Single point of failure, limited by one machine's resources.

Sharded Cluster Setup

```
Application → Mongos Router
            ↓
      ┌────────────────┴────────────────┐
                                                  ↓
      ↓

                                                    Shard B
  Shard A
                                                  (3 replicas)
 (3 replicas)
                                                    50% data
  50% data
```

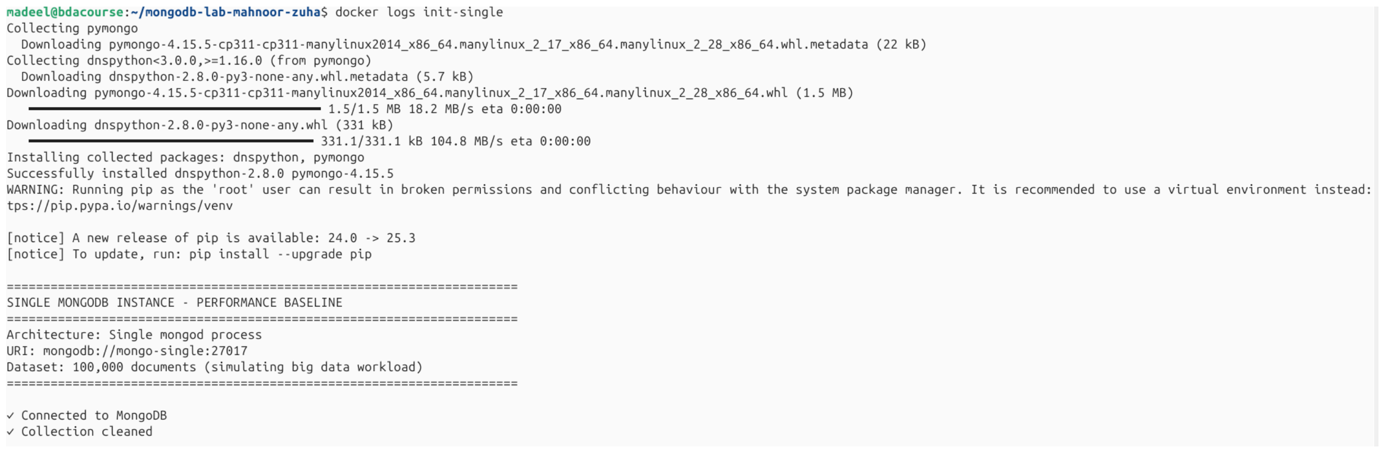
Key components

- Mongos: Query router directing traffic to appropriate shards
- Config Servers: Metadata storage (3-node replica set)
- Shards: Data storage nodes (3-node replica sets)
- Shard Key: Distribution field (userId, hashed

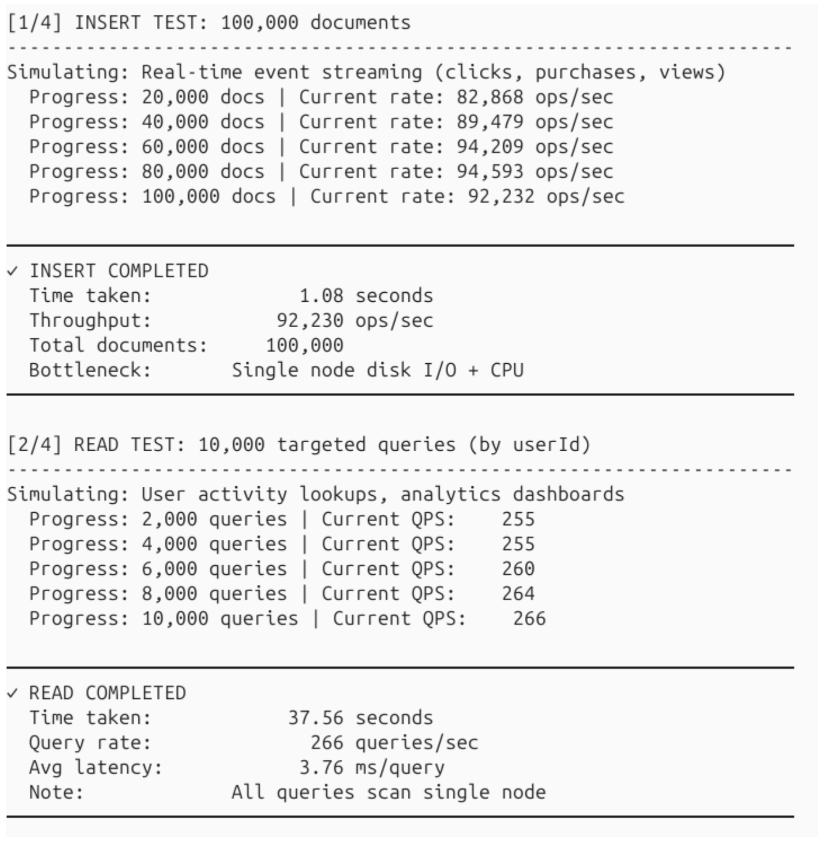
Experiment 1: Single Instance Baseline

Run a single MongoDB server and benchmark insert, read, update, and aggregation operations.

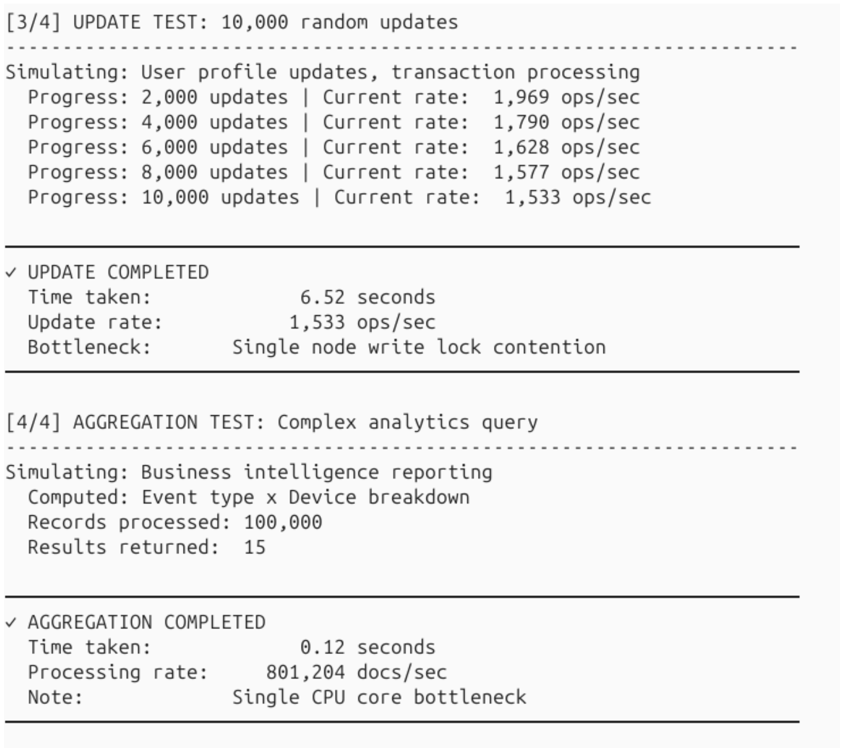
```
# Start single instance
docker-compose -f docker-compose.single.yml up

# View output (wait 3-5 minutes for completion)

# Check results
docker logs init-single
```



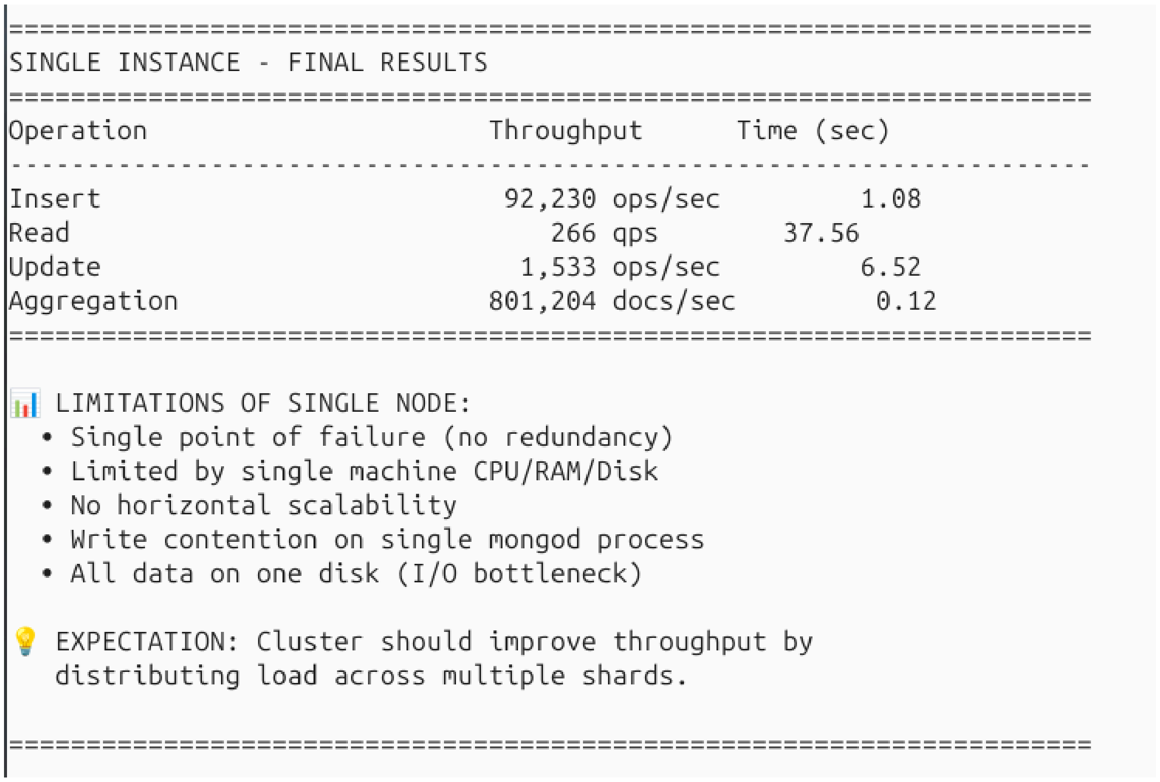
```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker logs init-single
Collecting pymongo
  Downloading pymongo-4.15.5-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.8.0-py3-none-any.whl.metadata (5.7 kB)
Downloading pymongo-4.15.5-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (1.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 18.2 MB/s eta 0:00:00
Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 331.1/331.1 kB 104.8 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.8.0 pymongo-4.15.5
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead:
tps://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: pip install --upgrade pip

======================================================================
SINGLE MONGODB INSTANCE - PERFORMANCE BASELINE
======================================================================
Architecture: Single mongod process
URI: mongodb://mongo-single:27017
Dataset: 100,000 documents (simulating big data workload)
======================================================================

✓ Connected to MongoDB
✓ Collection cleaned
```



```
[1/4] INSERT TEST: 100,000 documents
----------------------------------------------------------------------
Simulating: Real-time event streaming (clicks, purchases, views)
  Progress: 20,000 docs | Current rate: 82,868 ops/sec
  Progress: 40,000 docs | Current rate: 89,479 ops/sec
  Progress: 60,000 docs | Current rate: 94,209 ops/sec
  Progress: 80,000 docs | Current rate: 94,593 ops/sec
  Progress: 100,000 docs | Current rate: 92,232 ops/sec

──────────────────────────────────────────────────────────────────────
✓ INSERT COMPLETED
  Time taken:             1.08 seconds
  Throughput:           92,230 ops/sec
  Total documents:     100,000
  Bottleneck:       Single node disk I/O + CPU
──────────────────────────────────────────────────────────────────────

[2/4] READ TEST: 10,000 targeted queries (by userId)
----------------------------------------------------------------------
Simulating: User activity lookups, analytics dashboards
  Progress: 2,000 queries | Current QPS:    255
  Progress: 4,000 queries | Current QPS:    255
  Progress: 6,000 queries | Current QPS:    260
  Progress: 8,000 queries | Current QPS:    264
  Progress: 10,000 queries | Current QPS:    266

──────────────────────────────────────────────────────────────────────
✓ READ COMPLETED
  Time taken:            37.56 seconds
  Query rate:              266 queries/sec
  Avg latency:            3.76 ms/query
  Note:             All queries scan single node
──────────────────────────────────────────────────────────────────────
```



```
[3/4] UPDATE TEST: 10,000 random updates
----------------------------------------------------------------------
Simulating: User profile updates, transaction processing
  Progress: 2,000 updates | Current rate:  1,969 ops/sec
  Progress: 4,000 updates | Current rate:  1,790 ops/sec
  Progress: 6,000 updates | Current rate:  1,628 ops/sec
  Progress: 8,000 updates | Current rate:  1,577 ops/sec
  Progress: 10,000 updates | Current rate:  1,533 ops/sec

──────────────────────────────────────────────────────────────────────
✓ UPDATE COMPLETED
  Time taken:             6.52 seconds
  Update rate:           1,533 ops/sec
  Bottleneck:       Single node write lock contention
──────────────────────────────────────────────────────────────────────

[4/4] AGGREGATION TEST: Complex analytics query
----------------------------------------------------------------------
Simulating: Business intelligence reporting
  Computed: Event type x Device breakdown
  Records processed: 100,000
  Results returned:  15

──────────────────────────────────────────────────────────────────────
✓ AGGREGATION COMPLETED
  Time taken:             0.12 seconds
  Processing rate:     801,204 docs/sec
  Note:             Single CPU core bottleneck
──────────────────────────────────────────────────────────────────────
```

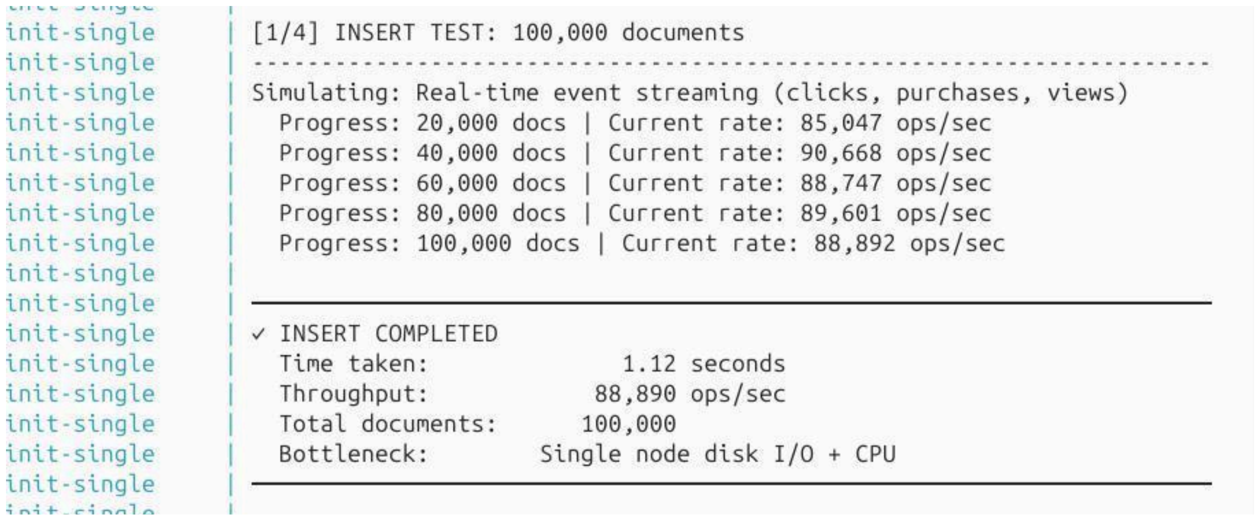


```
======================================================================
SINGLE INSTANCE - FINAL RESULTS
======================================================================
Operation                       Throughput      Time (sec)
----------------------------------------------------------------------
Insert                           92,230 ops/sec         1.08
Read                                266 qps        37.56
Update                            1,533 ops/sec         6.52
Aggregation                     801,204 docs/sec         0.12
======================================================================

📊 LIMITATIONS OF SINGLE NODE:
  • Single point of failure (no redundancy)
  • Limited by single machine CPU/RAM/Disk
  • No horizontal scalability
  • Write contention on single mongod process
  • All data on one disk (I/O bottleneck)

💡 EXPECTATION: Cluster should improve throughput by
   distributing load across multiple shards.

======================================================================
```

# Save for comparison
docker logs init-single > single_results.txt

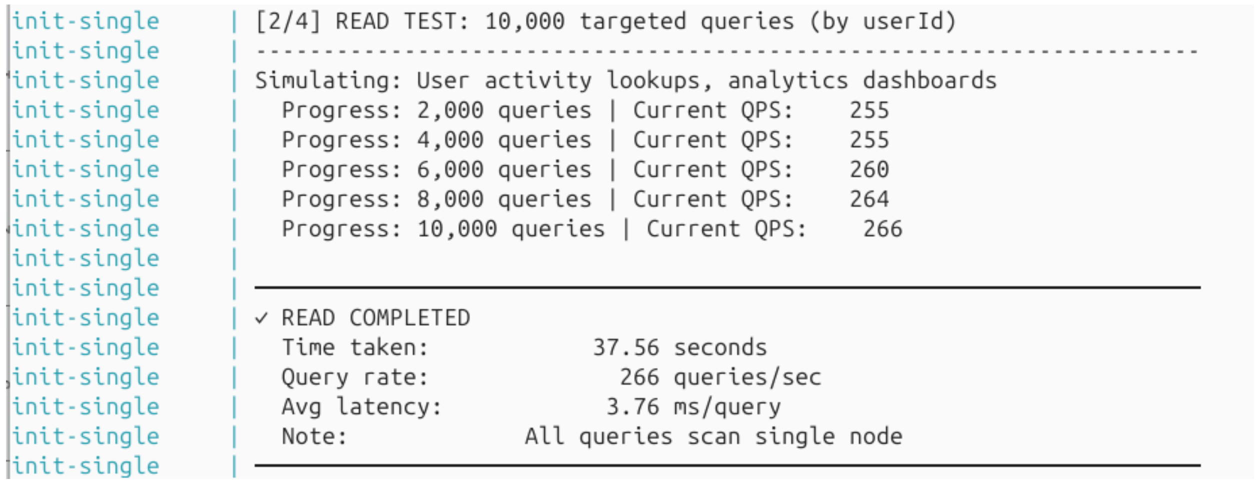
# Stop and cleanup
docker-compose -f docker-compose.single.yml down -v

- **Insert throughput (ops/sec)**



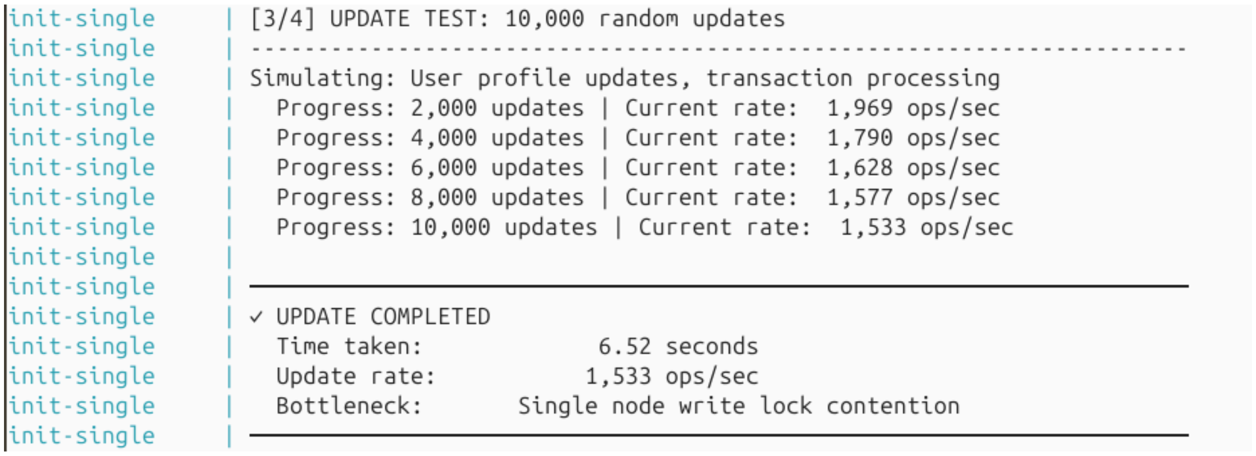
```
init-single     | [1/4] INSERT TEST: 100,000 documents
init-single     | ----------------------------------------------------------------------
init-single     | Simulating: Real-time event streaming (clicks, purchases, views)
init-single     |   Progress: 20,000 docs | Current rate: 85,047 ops/sec
init-single     |   Progress: 40,000 docs | Current rate: 90,668 ops/sec
init-single     |   Progress: 60,000 docs | Current rate: 88,747 ops/sec
init-single     |   Progress: 80,000 docs | Current rate: 89,601 ops/sec
init-single     |   Progress: 100,000 docs | Current rate: 88,892 ops/sec
init-single     |
init-single     | ──────────────────────────────────────────────────────────────────────
init-single     | ✓ INSERT COMPLETED
init-single     |   Time taken:              1.12 seconds
init-single     |   Throughput:            88,890 ops/sec
init-single     |   Total documents:      100,000
init-single     |   Bottleneck:         Single node disk I/O + CPU
init-single     | ──────────────────────────────────────────────────────────────────────
```

- **Read throughput (queries/sec)**



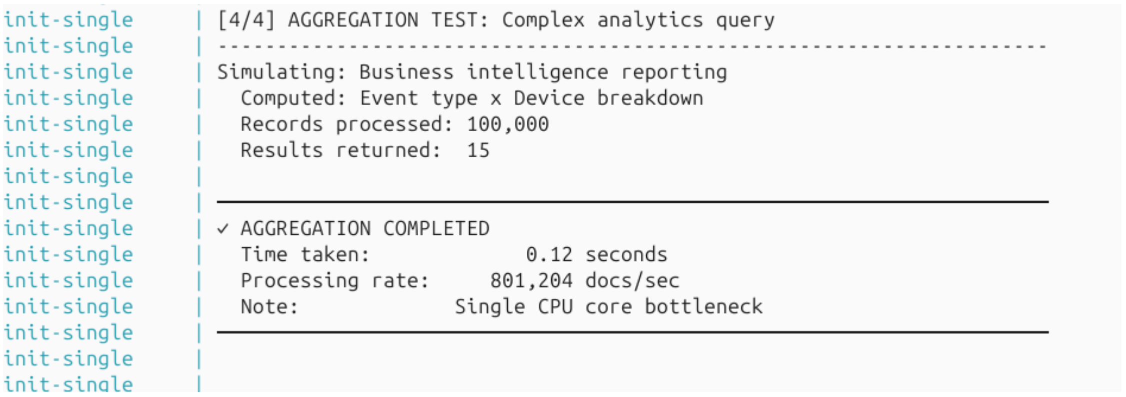
```
init-single     | [2/4] READ TEST: 10,000 targeted queries (by userId)
init-single     | ----------------------------------------------------------------------
init-single     | Simulating: User activity lookups, analytics dashboards
init-single     |   Progress: 2,000 queries | Current QPS:    255
init-single     |   Progress: 4,000 queries | Current QPS:    255
init-single     |   Progress: 6,000 queries | Current QPS:    260
init-single     |   Progress: 8,000 queries | Current QPS:    264
init-single     |   Progress: 10,000 queries | Current QPS:    266
init-single     |
init-single     | ──────────────────────────────────────────────────────────────────────
init-single     | ✓ READ COMPLETED
init-single     |   Time taken:             37.56 seconds
init-single     |   Query rate:               266 queries/sec
init-single     |   Avg latency:             3.76 ms/query
init-single     |   Note:              All queries scan single node
init-single     | ──────────────────────────────────────────────────────────────────────
```

- **Update throughput (ops/sec)**



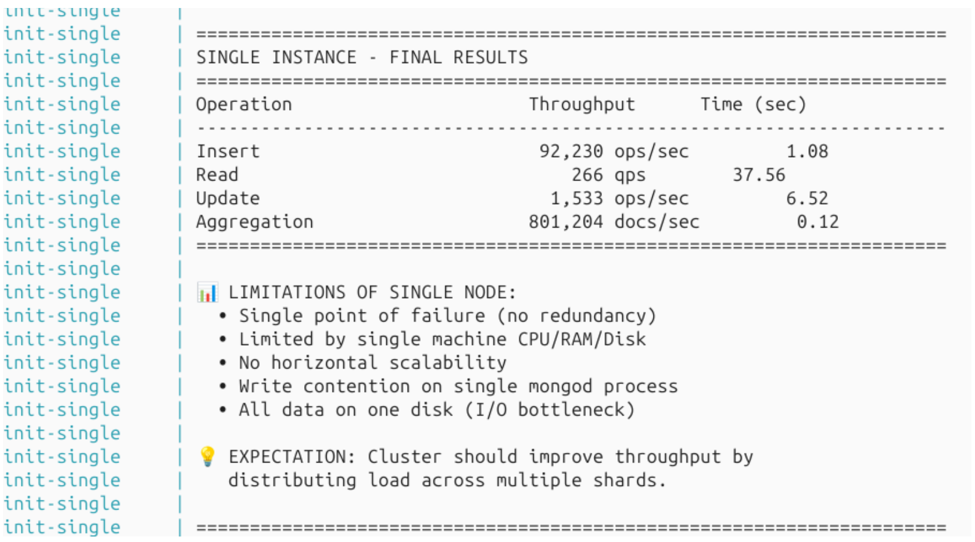
```
init-single      | [3/4] UPDATE TEST: 10,000 random updates
init-single      | ----------------------------------------------------------------------
init-single      | Simulating: User profile updates, transaction processing
init-single      |   Progress: 2,000 updates | Current rate:  1,969 ops/sec
init-single      |   Progress: 4,000 updates | Current rate:  1,790 ops/sec
init-single      |   Progress: 6,000 updates | Current rate:  1,628 ops/sec
init-single      |   Progress: 8,000 updates | Current rate:  1,577 ops/sec
init-single      |   Progress: 10,000 updates | Current rate:  1,533 ops/sec
init-single      |
init-single      | ──────────────────────────────────────────────────────────────────────
init-single      | ✓ UPDATE COMPLETED
init-single      |   Time taken:             6.52 seconds
init-single      |   Update rate:           1,533 ops/sec
init-single      |   Bottleneck:       Single node write lock contention
init-single      | ──────────────────────────────────────────────────────────────────────
```

- **Aggregation speed (docs/sec)**



```
init-single      | [4/4] AGGREGATION TEST: Complex analytics query
init-single      | ----------------------------------------------------------------------
init-single      | Simulating: Business intelligence reporting
init-single      |   Computed: Event type x Device breakdown
init-single      |   Records processed: 100,000
init-single      |   Results returned:  15
init-single      |
init-single      | ──────────────────────────────────────────────────────────────────────
init-single      | ✓ AGGREGATION COMPLETED
init-single      |   Time taken:             0.12 seconds
init-single      |   Processing rate:     801,204 docs/sec
init-single      |   Note:             Single CPU core bottleneck
init-single      | ──────────────────────────────────────────────────────────────────────
init-single      |
init-single      |
```

- **All together**



```
init-single      |
init-single      | ======================================================================
init-single      | SINGLE INSTANCE - FINAL RESULTS
init-single      | ======================================================================
init-single      | Operation                      Throughput      Time (sec)
init-single      | ----------------------------------------------------------------------
init-single      | Insert                          92,230 ops/sec         1.08
init-single      | Read                               266 qps         37.56
init-single      | Update                           1,533 ops/sec         6.52
init-single      | Aggregation                    801,204 docs/sec         0.12
init-single      | ======================================================================
init-single      |
init-single      | 📊 LIMITATIONS OF SINGLE NODE:
init-single      |   • Single point of failure (no redundancy)
init-single      |   • Limited by single machine CPU/RAM/Disk
init-single      |   • No horizontal scalability
init-single      |   • Write contention on single mongod process
init-single      |   • All data on one disk (I/O bottleneck)
init-single      |
init-single      | 💡 EXPECTATION: Cluster should improve throughput by
init-single      |    distributing load across multiple shards.
init-single      |
init-single      | ======================================================================
```

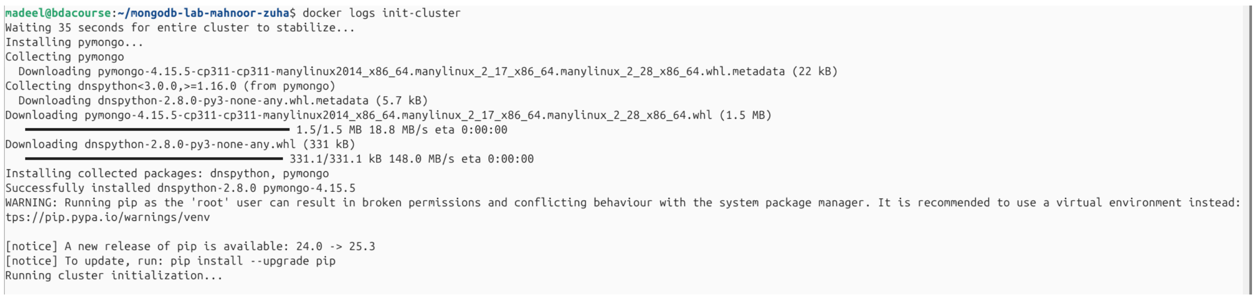
Experiment 2: Sharded Cluster

Deploy a 10-container cluster (3 config servers, 6 shard nodes, 1 router, 1 initialization script).

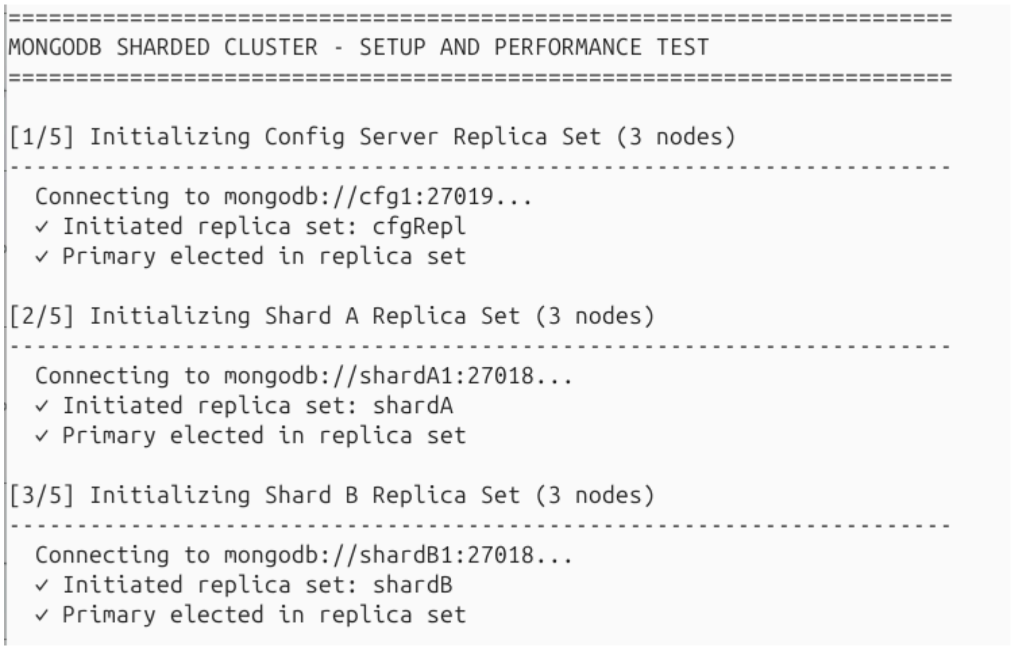
# Start cluster
docker-compose -f docker-compose.cluster.yml up

# The initialization process:
# - Sets up config server replica set
# - Initializes shard A (3 nodes)
# - Initializes shard B (3 nodes)
# - Registers shards with cluster
# - Enables sharding on database
# - Runs performance benchmark

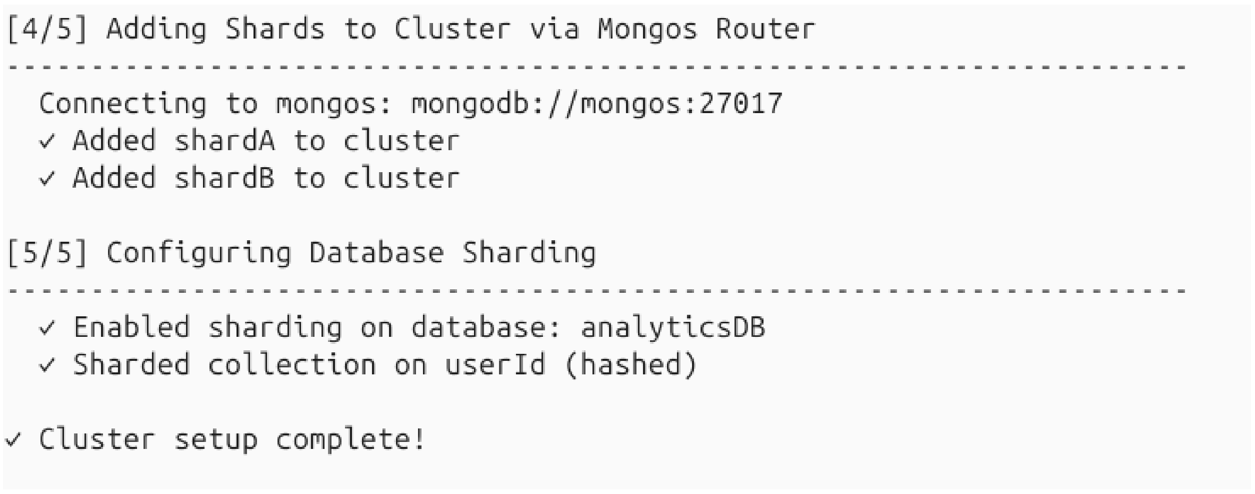
# View results
docker logs init-cluster



```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker logs init-cluster
Waiting 35 seconds for entire cluster to stabilize...
Installing pymongo...
Collecting pymongo
  Downloading pymongo-4.15.5-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.8.0-py3-none-any.whl.metadata (5.7 kB)
Downloading pymongo-4.15.5-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (1.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 18.8 MB/s eta 0:00:00
Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 331.1/331.1 kB 148.0 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.8.0 pymongo-4.15.5
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead:
tps://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: pip install --upgrade pip
Running cluster initialization...
```

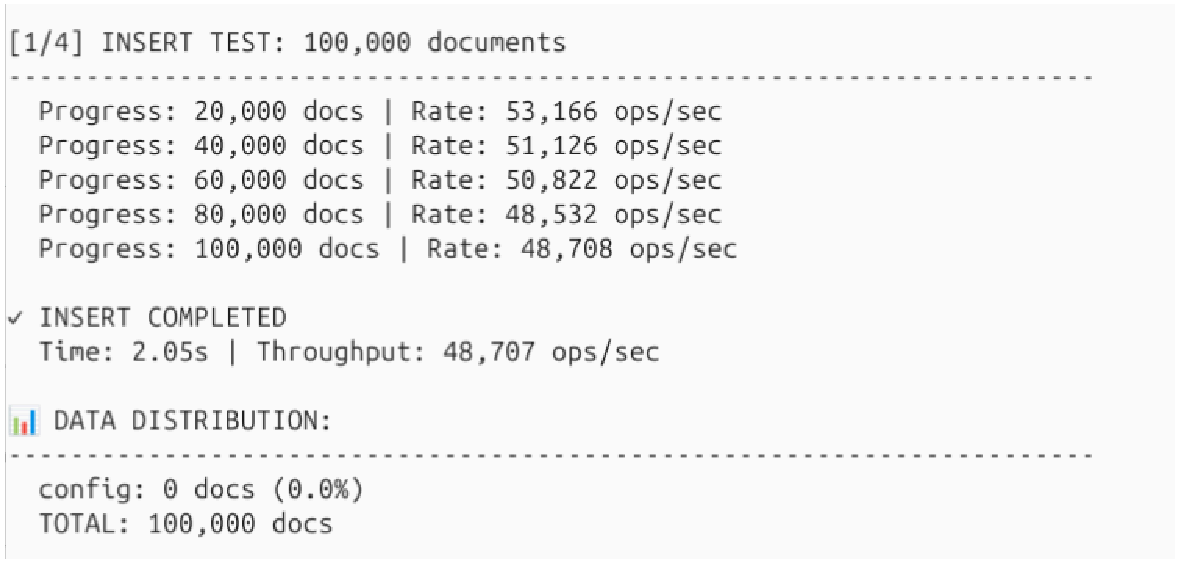


```
======================================================================
MONGODB SHARDED CLUSTER - SETUP AND PERFORMANCE TEST
======================================================================

[1/5] Initializing Config Server Replica Set (3 nodes)
----------------------------------------------------------------------
  Connecting to mongodb://cfg1:27019...
  ✓ Initiated replica set: cfgRepl
  ✓ Primary elected in replica set

[2/5] Initializing Shard A Replica Set (3 nodes)
----------------------------------------------------------------------
  Connecting to mongodb://shardA1:27018...
  ✓ Initiated replica set: shardA
  ✓ Primary elected in replica set

[3/5] Initializing Shard B Replica Set (3 nodes)
----------------------------------------------------------------------
  Connecting to mongodb://shardB1:27018...
  ✓ Initiated replica set: shardB
  ✓ Primary elected in replica set
```



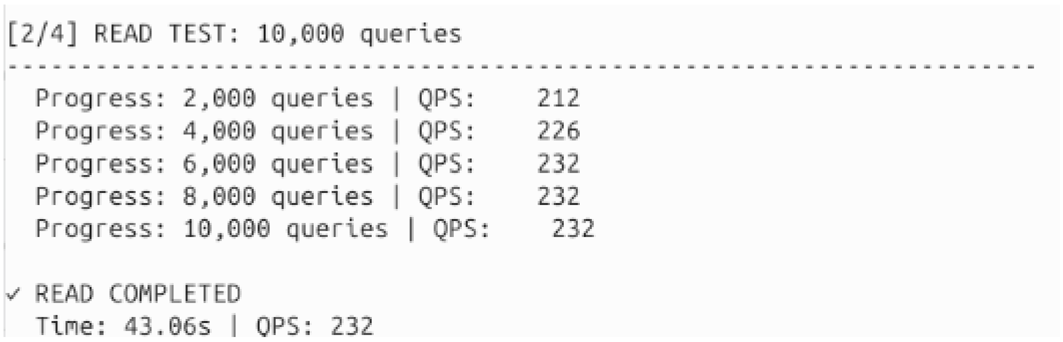
```
[4/5] Adding Shards to Cluster via Mongos Router
----------------------------------------------------------------------
  Connecting to mongos: mongodb://mongos:27017
  ✓ Added shardA to cluster
  ✓ Added shardB to cluster

[5/5] Configuring Database Sharding
----------------------------------------------------------------------
  ✓ Enabled sharding on database: analyticsDB
  ✓ Sharded collection on userId (hashed)

✓ Cluster setup complete!
```

- **Insert throughput (ops/sec)**



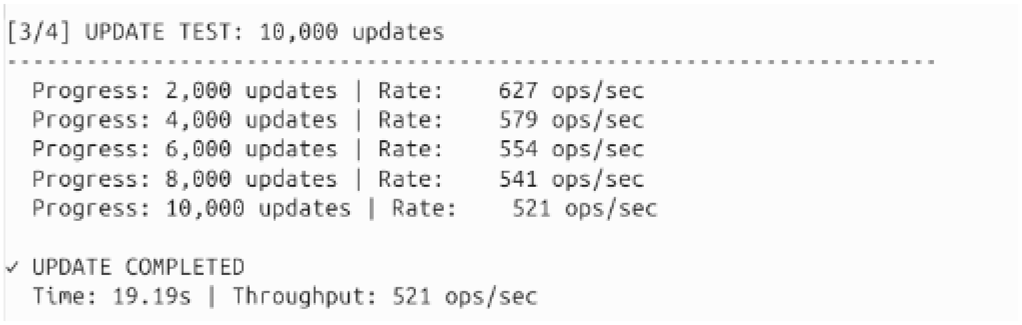
```
[1/4] INSERT TEST: 100,000 documents
----------------------------------------------------------------------
  Progress: 20,000 docs | Rate: 53,166 ops/sec
  Progress: 40,000 docs | Rate: 51,126 ops/sec
  Progress: 60,000 docs | Rate: 50,822 ops/sec
  Progress: 80,000 docs | Rate: 48,532 ops/sec
  Progress: 100,000 docs | Rate: 48,708 ops/sec

✓ INSERT COMPLETED
  Time: 2.05s | Throughput: 48,707 ops/sec

📊 DATA DISTRIBUTION:
----------------------------------------------------------------------
  config: 0 docs (0.0%)
  TOTAL: 100,000 docs
```

- **Read throughput (queries/sec)**



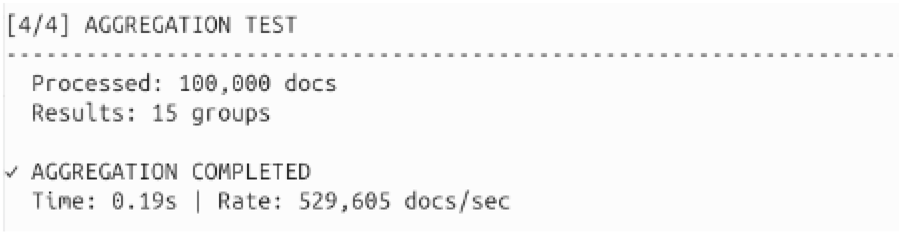
```
[2/4] READ TEST: 10,000 queries
------------------------------------------------------------------------
  Progress: 2,000 queries | QPS:    212
  Progress: 4,000 queries | QPS:    226
  Progress: 6,000 queries | QPS:    232
  Progress: 8,000 queries | QPS:    232
  Progress: 10,000 queries | QPS:    232

✓ READ COMPLETED
  Time: 43.06s | QPS: 232
```

- **Update throughput (ops/sec)**



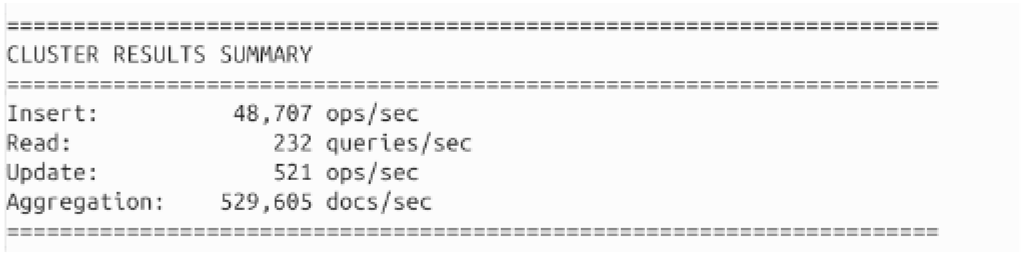
```
[3/4] UPDATE TEST: 10,000 updates
----------------------------------------------------------------------
  Progress: 2,000 updates | Rate:    627 ops/sec
  Progress: 4,000 updates | Rate:    579 ops/sec
  Progress: 6,000 updates | Rate:    554 ops/sec
  Progress: 8,000 updates | Rate:    541 ops/sec
  Progress: 10,000 updates | Rate:    521 ops/sec

✓ UPDATE COMPLETED
  Time: 19.19s | Throughput: 521 ops/sec
```

- **Aggregate throughput (docs/sec)**



```
[4/4] AGGREGATION TEST
----------------------------------------------------------------------
  Processed: 100,000 docs
  Results: 15 groups

✓ AGGREGATION COMPLETED
  Time: 0.19s | Rate: 529,605 docs/sec
```

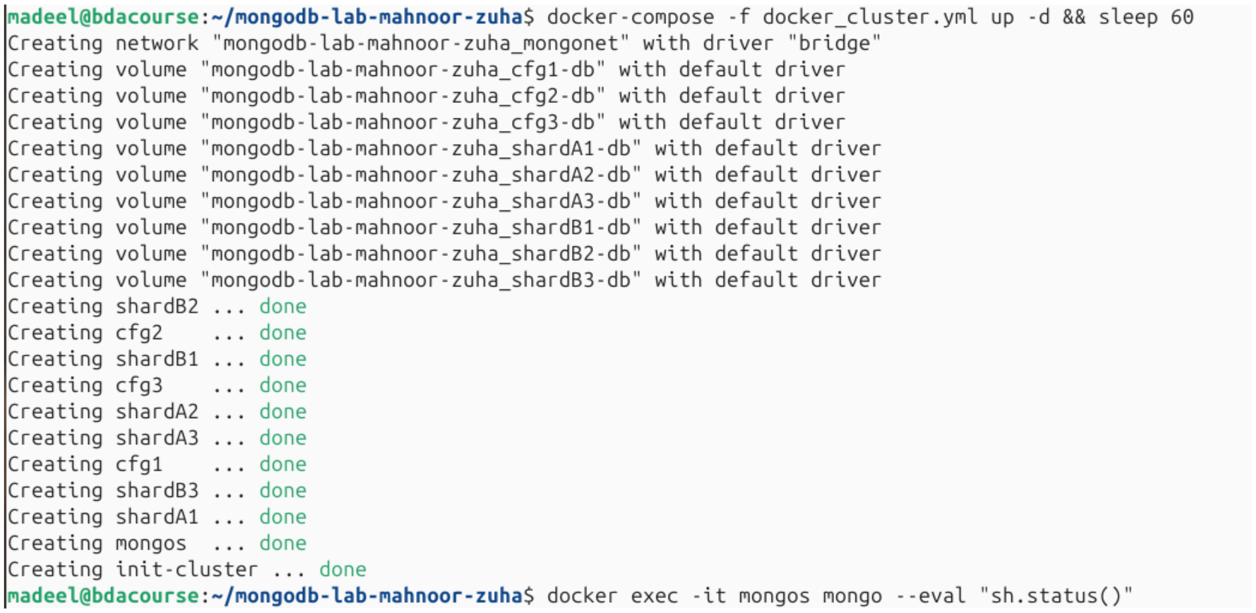
- **All together**



```
======================================================================
CLUSTER RESULTS SUMMARY
======================================================================
Insert:          48,707 ops/sec
Read:               232 queries/sec
Update:             521 ops/sec
Aggregation:    529,605 docs/sec
======================================================================
```

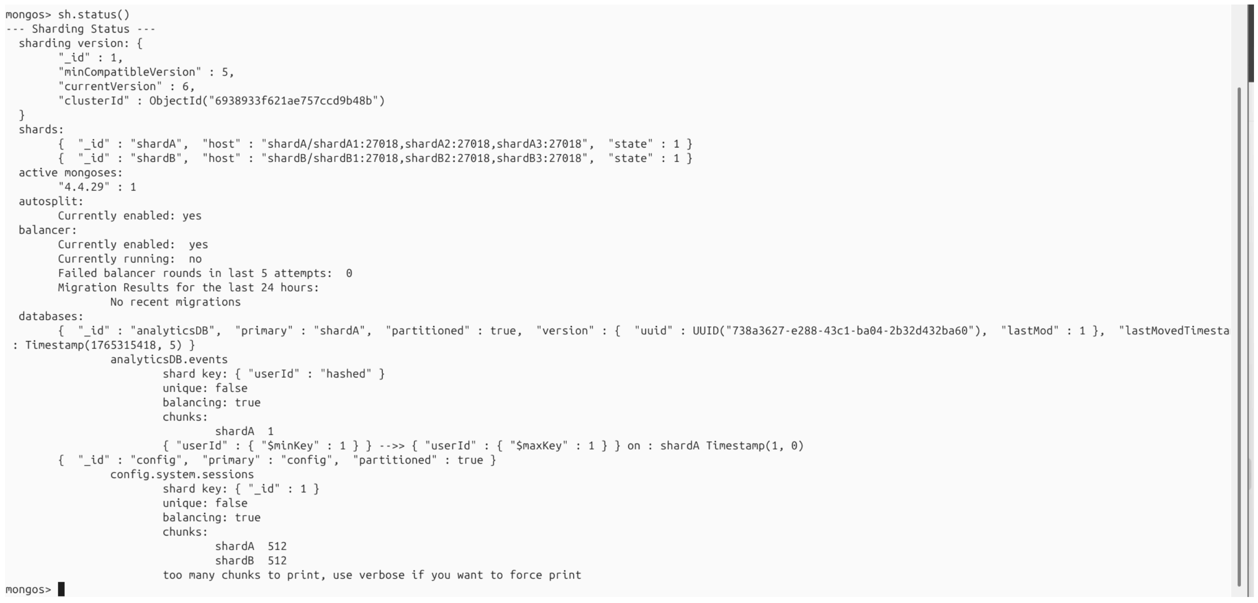
Cluster Inspection (Required)

# Start cluster in detached mode docker-compose -f docker-compose.cluster.yml up -d sleep 60



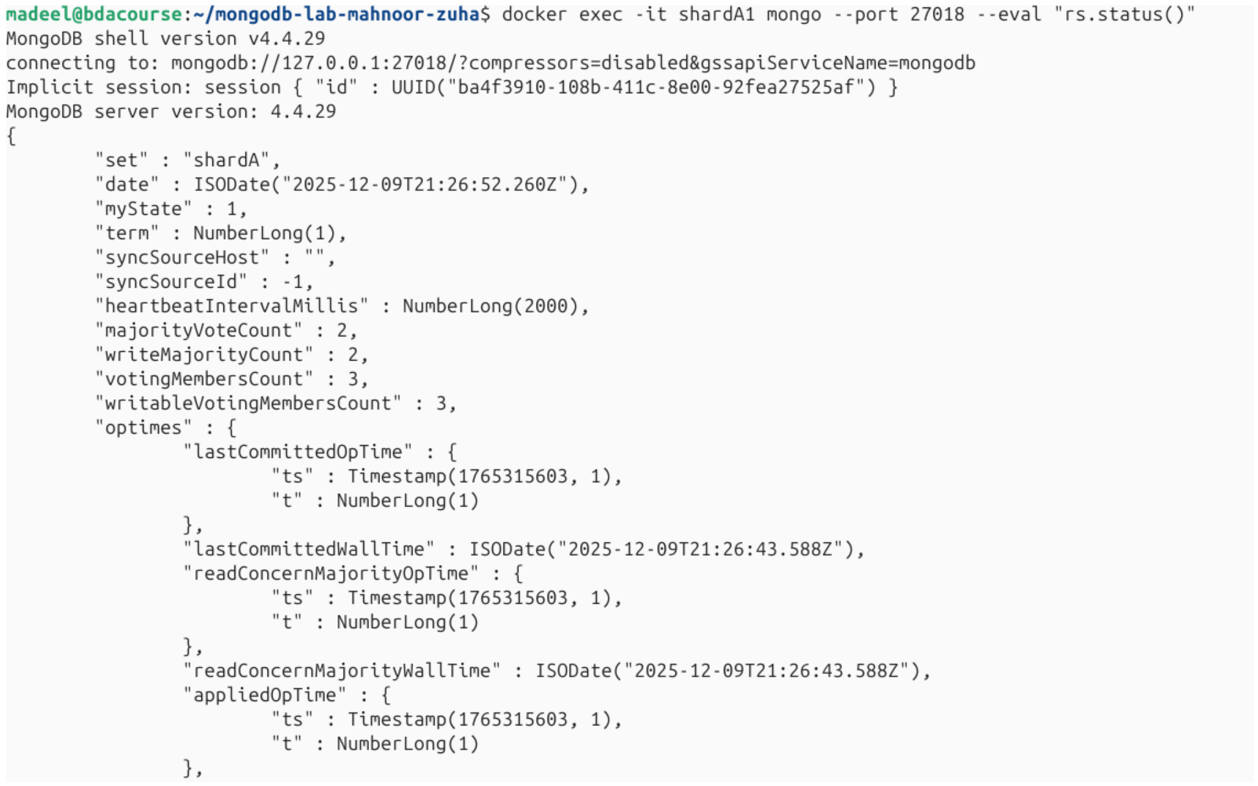
```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker-compose -f docker_cluster.yml up -d && sleep 60
Creating network "mongodb-lab-mahnoor-zuha_mongonet" with driver "bridge"
Creating volume "mongodb-lab-mahnoor-zuha_cfg1-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_cfg2-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_cfg3-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_shardA1-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_shardA2-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_shardA3-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_shardB1-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_shardB2-db" with default driver
Creating volume "mongodb-lab-mahnoor-zuha_shardB3-db" with default driver
Creating shardB2 ... done
Creating cfg2    ... done
Creating shardB1 ... done
Creating cfg3    ... done
Creating shardA2 ... done
Creating shardA3 ... done
Creating cfg1    ... done
Creating shardB3 ... done
Creating shardA1 ... done
Creating mongos  ... done
Creating init-cluster ... done
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker exec -it mongos mongo --eval "sh.status()"
```

# View cluster configuration
docker exec -it mongos mongo --eval "sh.status()"

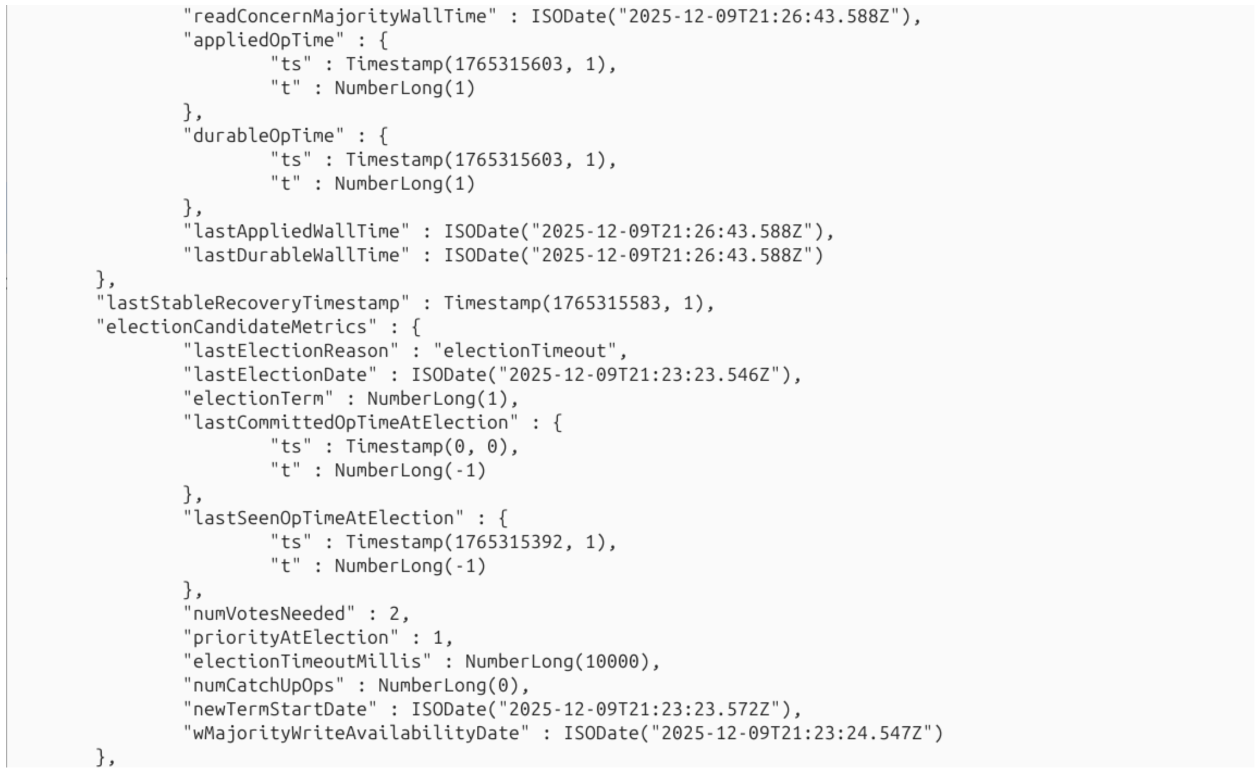


```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("6938933f621ae757ccd9b48b")
  }
  shards:
        {  "_id" : "shardA",  "host" : "shardA/shardA1:27018,shardA2:27018,shardA3:27018",  "state" : 1 }
        {  "_id" : "shardB",  "host" : "shardB/shardB1:27018,shardB2:27018,shardB3:27018",  "state" : 1 }
  active mongoses:
        "4.4.29" : 1
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "analyticsDB",  "primary" : "shardA",  "partitioned" : true,  "version" : {  "uuid" : UUID("738a3627-e288-43c1-ba04-2b32d432ba60"),  "lastMod" : 1 },  "lastMovedTimesta
 : Timestamp(1765315418, 5) }
                analyticsDB.events
                        shard key: { "userId" : "hashed" }
                        unique: false
                        balancing: true
                        chunks:
                                shardA  1
                        { "userId" : { "$minKey" : 1 } } -->> { "userId" : { "$maxKey" : 1 } } on : shardA Timestamp(1, 0)
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                shardA  512
                                shardB  512
                        too many chunks to print, use verbose if you want to force print
mongos> 
```

# Check shard A status
docker exec -it shardA1 mongo --port 27018 --eval "rs.status()"



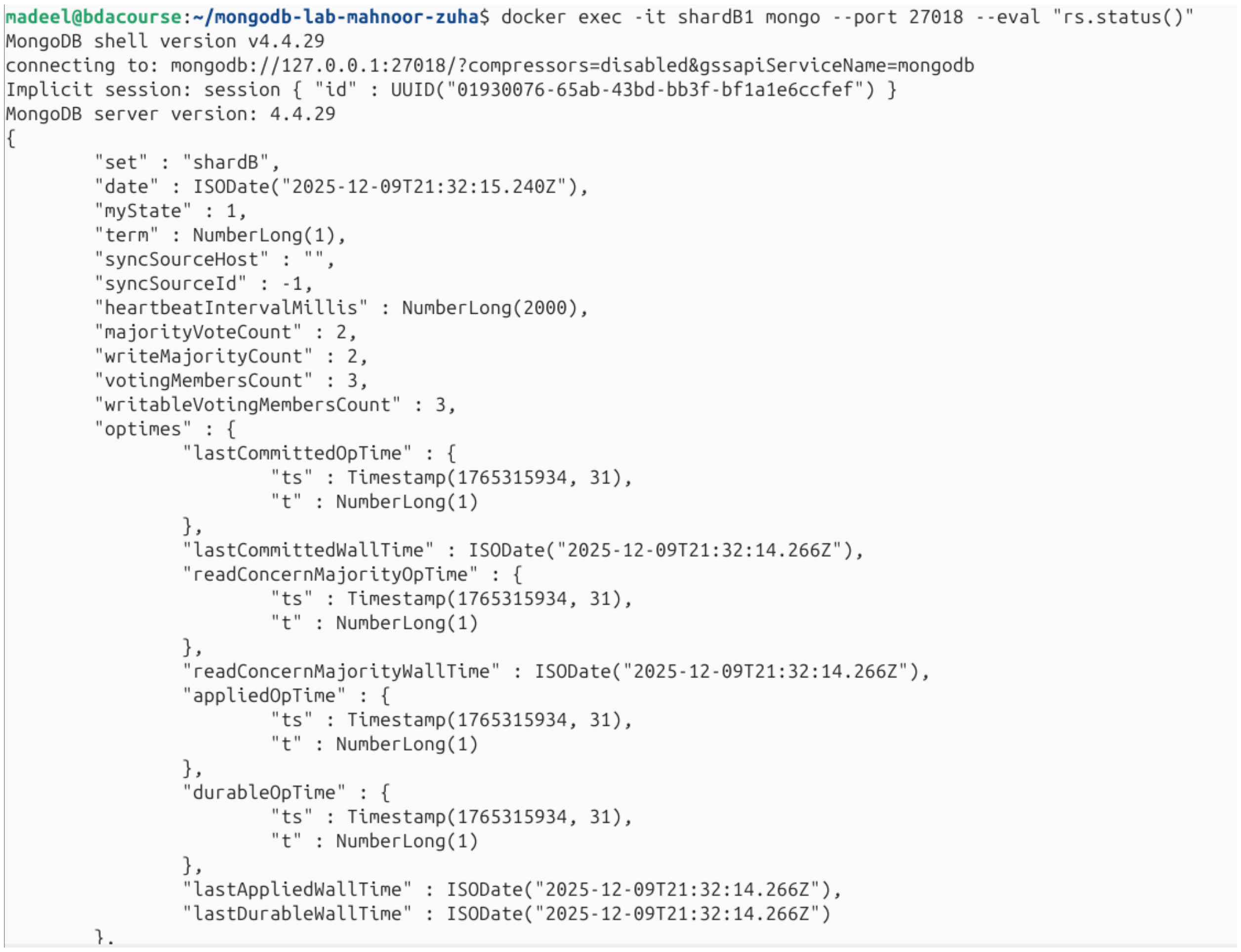
```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker exec -it shardA1 mongo --port 27018 --eval "rs.status()"
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27018/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("ba4f3910-108b-411c-8e00-92fea27525af") }
MongoDB server version: 4.4.29
{
        "set" : "shardA",
        "date" : ISODate("2025-12-09T21:26:52.260Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "majorityVoteCount" : 2,
        "writeMajorityCount" : 2,
        "votingMembersCount" : 3,
        "writableVotingMembersCount" : 3,
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1765315603, 1),
                        "t" : NumberLong(1)
                },
                "lastCommittedWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1765315603, 1),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                "appliedOpTime" : {
                        "ts" : Timestamp(1765315603, 1),
                        "t" : NumberLong(1)
                },
```



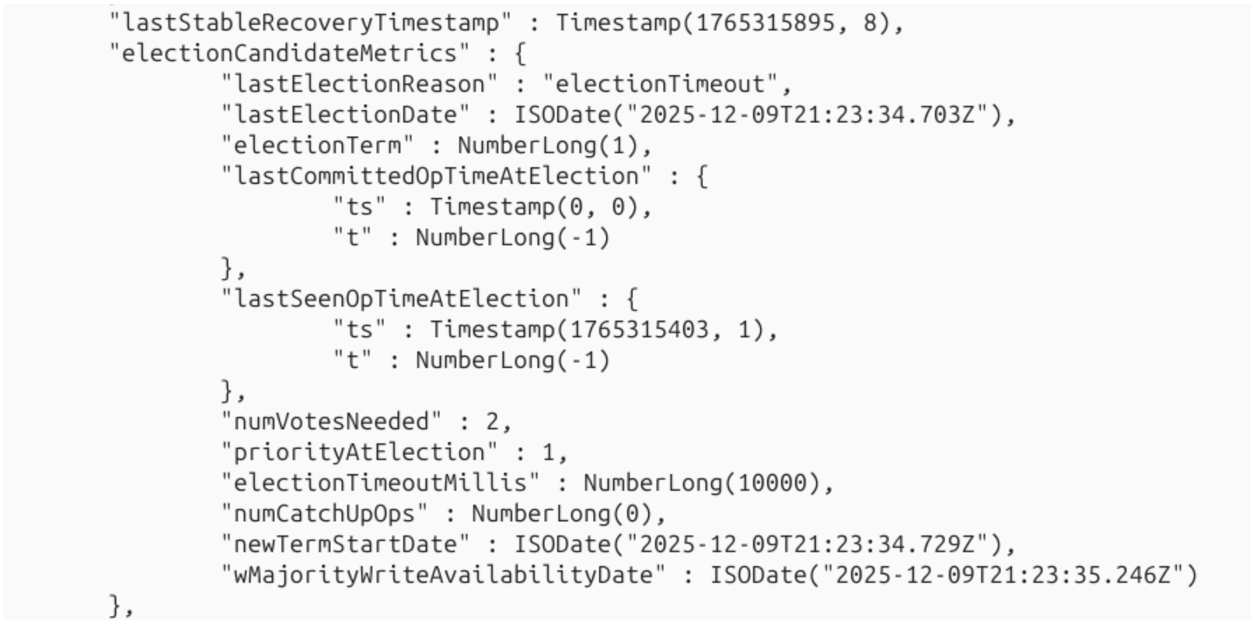
```
                "readConcernMajorityWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                "appliedOpTime" : {
                        "ts" : Timestamp(1765315603, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1765315603, 1),
                        "t" : NumberLong(1)
                },
                "lastAppliedWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                "lastDurableWallTime" : ISODate("2025-12-09T21:26:43.588Z")
        },
        "lastStableRecoveryTimestamp" : Timestamp(1765315583, 1),
        "electionCandidateMetrics" : {
                "lastElectionReason" : "electionTimeout",
                "lastElectionDate" : ISODate("2025-12-09T21:23:23.546Z"),
                "electionTerm" : NumberLong(1),
                "lastCommittedOpTimeAtElection" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "lastSeenOpTimeAtElection" : {
                        "ts" : Timestamp(1765315392, 1),
                        "t" : NumberLong(-1)
                },
                "numVotesNeeded" : 2,
                "priorityAtElection" : 1,
                "electionTimeoutMillis" : NumberLong(10000),
                "numCatchUpOps" : NumberLong(0),
                "newTermStartDate" : ISODate("2025-12-09T21:23:23.572Z"),
                "wMajorityWriteAvailabilityDate" : ISODate("2025-12-09T21:23:24.547Z")
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "shardA1:27018",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 272,
                        "optime" : {
                                "ts" : Timestamp(1765315603, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2025-12-09T21:26:43Z"),
                        "lastAppliedWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                        "lastDurableWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                        "syncSourceHost" : "",
                        "syncSourceId" : -1,
                        "infoMessage" : "",
                        "electionTime" : Timestamp(1765315403, 1),
                        "electionDate" : ISODate("2025-12-09T21:23:23Z"),
                        "configVersion" : 2,
                        "configTerm" : 1,
                        "self" : true,
                        "lastHeartbeatMessage" : ""
                },
```

```
                        "_id" : 1,
                        "name" : "shardA2:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 219,
                        "optime" : {
                                "ts" : Timestamp(1765315603, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1765315603, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2025-12-09T21:26:43Z"),
                        "optimeDurableDate" : ISODate("2025-12-09T21:26:43Z"),
                        "lastAppliedWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                        "lastDurableWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                        "lastHeartbeat" : ISODate("2025-12-09T21:26:50.787Z"),
                        "lastHeartbeatRecv" : ISODate("2025-12-09T21:26:51.821Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "",
                        "syncSourceHost" : "shardA1:27018",
                        "syncSourceId" : 0,
                        "infoMessage" : "",
                        "configVersion" : 2,
                        "configTerm" : 1
                },
                {
                        "_id" : 2,
                        "name" : "shardA3:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 219,
                        "optime" : {
                                "ts" : Timestamp(1765315603, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1765315603, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2025-12-09T21:26:43Z"),
                        "optimeDurableDate" : ISODate("2025-12-09T21:26:43Z"),
                        "lastAppliedWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                        "lastDurableWallTime" : ISODate("2025-12-09T21:26:43.588Z"),
                        "lastHeartbeat" : ISODate("2025-12-09T21:26:50.783Z"),
                        "lastHeartbeatRecv" : ISODate("2025-12-09T21:26:50.328Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "",
                        "syncSourceHost" : "shardA1:27018",
                        "syncSourceId" : 0,
                        "infoMessage" : "",
                        "configVersion" : 2,
                        "configTerm" : 1
                }
        ],
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(0, 0),
                "electionId" : ObjectId("7fffffff0000000000000001")
        },
        "lastCommittedOpTime" : Timestamp(1765315603, 1),
        "$configServerState" : {
                "opTime" : {
                        "ts" : Timestamp(1765315603, 1),
                        "t" : NumberLong(1)
                }
        },
        "$clusterTime" : {
                "clusterTime" : Timestamp(1765315603, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1765315603, 1)
}
```

# Check shard B status
docker exec -it shardB1 mongo --port 27018 --eval "rs.status()"



```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker exec -it shardB1 mongo --port 27018 --eval "rs.status()"
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27018/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("01930076-65ab-43bd-bb3f-bf1a1e6ccfef") }
MongoDB server version: 4.4.29
{
        "set" : "shardB",
        "date" : ISODate("2025-12-09T21:32:15.240Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "majorityVoteCount" : 2,
        "writeMajorityCount" : 2,
        "votingMembersCount" : 3,
        "writableVotingMembersCount" : 3,
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1765315934, 31),
                        "t" : NumberLong(1)
                },
                "lastCommittedWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1765315934, 31),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
                "appliedOpTime" : {
                        "ts" : Timestamp(1765315934, 31),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1765315934, 31),
                        "t" : NumberLong(1)
                },
                "lastAppliedWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
                "lastDurableWallTime" : ISODate("2025-12-09T21:32:14.266Z")
        },
```

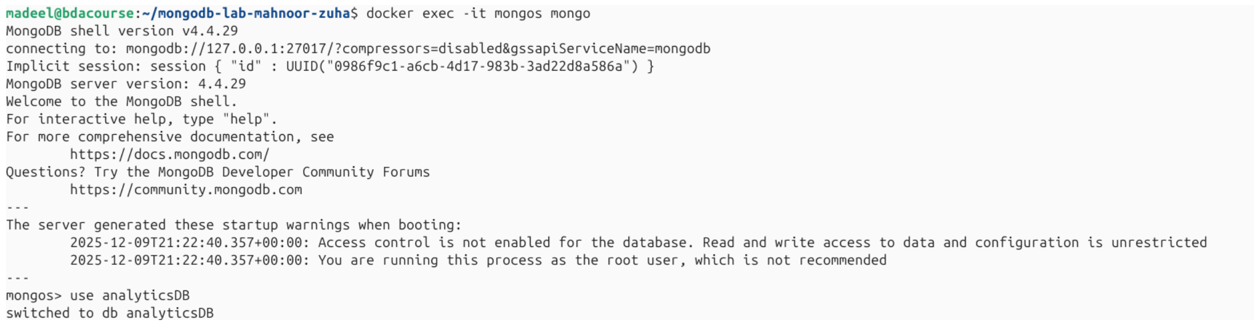


```
        "lastStableRecoveryTimestamp" : Timestamp(1765315895, 8),
        "electionCandidateMetrics" : {
                "lastElectionReason" : "electionTimeout",
                "lastElectionDate" : ISODate("2025-12-09T21:23:34.703Z"),
                "electionTerm" : NumberLong(1),
                "lastCommittedOpTimeAtElection" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "lastSeenOpTimeAtElection" : {
                        "ts" : Timestamp(1765315403, 1),
                        "t" : NumberLong(-1)
                },
                "numVotesNeeded" : 2,
                "priorityAtElection" : 1,
                "electionTimeoutMillis" : NumberLong(10000),
                "numCatchUpOps" : NumberLong(0),
                "newTermStartDate" : ISODate("2025-12-09T21:23:34.729Z"),
                "wMajorityWriteAvailabilityDate" : ISODate("2025-12-09T21:23:35.246Z")
        },
	"members" : [
		{
			"_id" : 0,
			"name" : "shardB1:27018",
			"health" : 1,
			"state" : 1,
			"stateStr" : "PRIMARY",
			"uptime" : 596,
			"optime" : {
				"ts" : Timestamp(1765315934, 31),
				"t" : NumberLong(1)
			},
			"optimeDate" : ISODate("2025-12-09T21:32:14Z"),
			"lastAppliedWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
			"lastDurableWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
			"syncSourceHost" : "",
			"syncSourceId" : -1,
			"infoMessage" : "",
			"electionTime" : Timestamp(1765315414, 1),
			"electionDate" : ISODate("2025-12-09T21:23:34Z"),
			"configVersion" : 2,
			"configTerm" : 1,
			"self" : true,
			"lastHeartbeatMessage" : ""
		},
		{
			"_id" : 1,
			"name" : "shardB2:27018",
			"health" : 1,
			"state" : 2,
			"stateStr" : "SECONDARY",
			"uptime" : 531,
			"optime" : {
				"ts" : Timestamp(1765315934, 31),
				"t" : NumberLong(1)
			},
			"optimeDurable" : {
				"ts" : Timestamp(1765315934, 31),
				"t" : NumberLong(1)
			},
			"optimeDate" : ISODate("2025-12-09T21:32:14Z"),
			"optimeDurableDate" : ISODate("2025-12-09T21:32:14Z"),
			"lastAppliedWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
			"lastDurableWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
			"lastHeartbeat" : ISODate("2025-12-09T21:32:14.914Z"),
			"lastHeartbeatRecv" : ISODate("2025-12-09T21:32:14.961Z"),
			"pingMs" : NumberLong(0),
			"lastHeartbeatMessage" : "",
			"syncSourceHost" : "shardB1:27018",
			"syncSourceId" : 0,
			"infoMessage" : "",
			"configVersion" : 2,
			"configTerm" : 1
		},
                {
                        "_id" : 2,
                        "name" : "shardB3:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 531,
                        "optime" : {
                                "ts" : Timestamp(1765315934, 31),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1765315934, 31),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2025-12-09T21:32:14Z"),
                        "optimeDurableDate" : ISODate("2025-12-09T21:32:14Z"),
                        "lastAppliedWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
                        "lastDurableWallTime" : ISODate("2025-12-09T21:32:14.266Z"),
                        "lastHeartbeat" : ISODate("2025-12-09T21:32:14.915Z"),
                        "lastHeartbeatRecv" : ISODate("2025-12-09T21:32:14.958Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "",
                        "syncSourceHost" : "shardB1:27018",
                        "syncSourceId" : 0,
                        "infoMessage" : "",
                        "configVersion" : 2,
                        "configTerm" : 1
                }
        ],
```

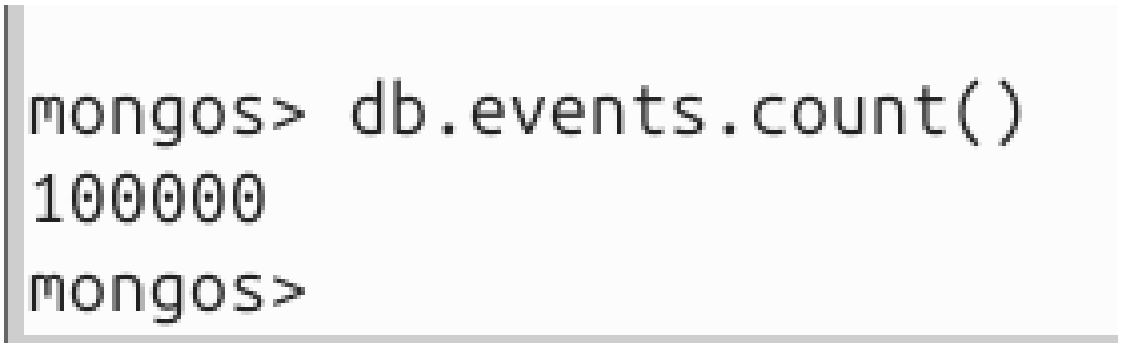
```
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(0, 0),
                "electionId" : ObjectId("7fffffff0000000000000001")
        },
        "lastCommittedOpTime" : Timestamp(1765315934, 31),
        "$configServerState" : {
                "opTime" : {
                        "ts" : Timestamp(1765315934, 24),
                        "t" : NumberLong(1)
                }
        },
        "$clusterTime" : {
                "clusterTime" : Timestamp(1765315934, 31),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1765315934, 31)
}
```

# Interactive shell
docker exec -it mongos mongo

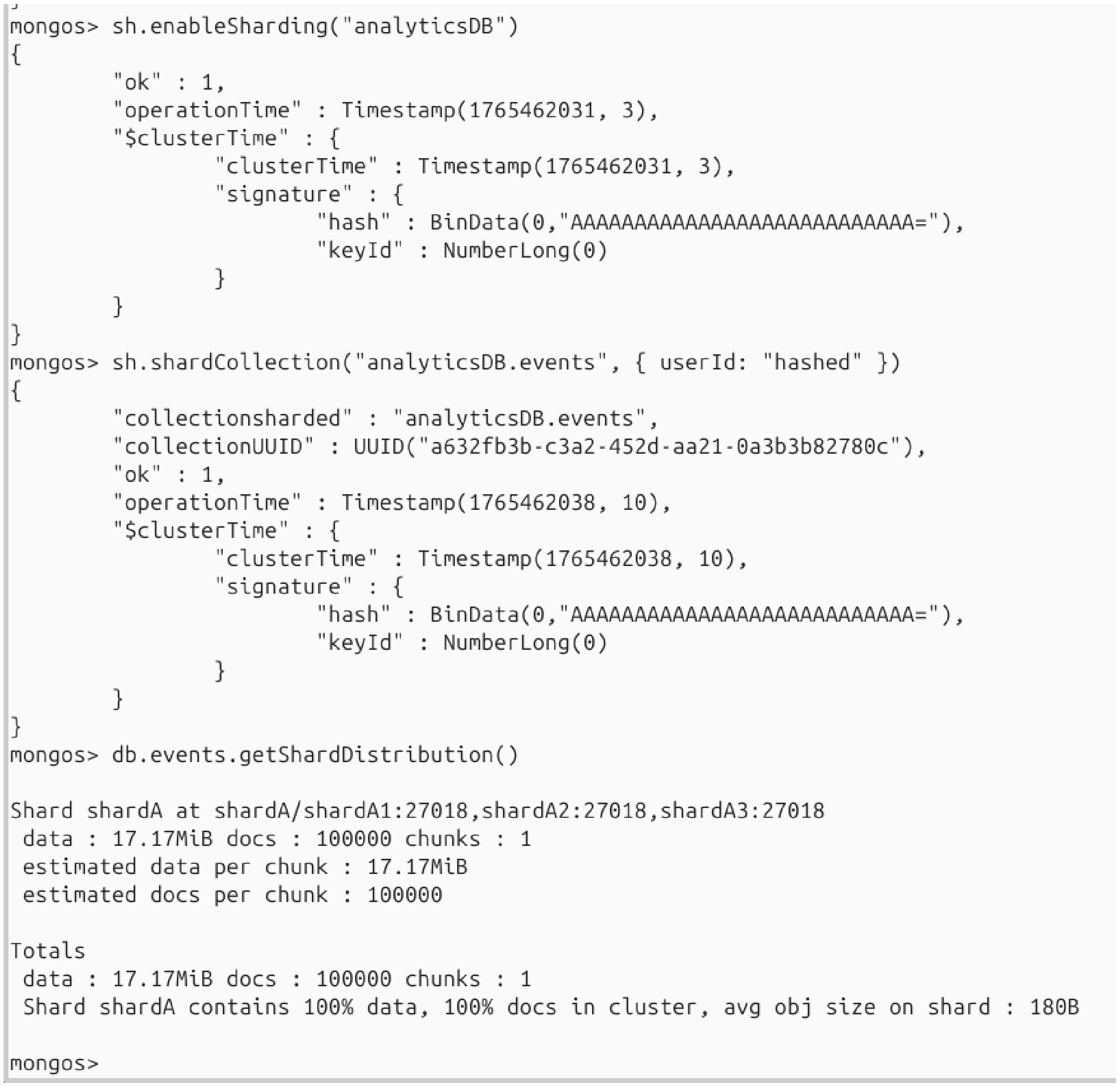
Inside mongo shell:
use analyticsDB


```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker exec -it mongos mongo
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0986f9c1-a6cb-4d17-983b-3ad22d8a586a") }
MongoDB server version: 4.4.29
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2025-12-09T21:22:40.357+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
        2025-12-09T21:22:40.357+00:00: You are running this process as the root user, which is not recommended
---
mongos> use analyticsDB
switched to db analyticsDB
```

db.events.count()


```
mongos> db.events.count()
100000
mongos>
```

db.events.getShardDistribution()



```
mongos> sh.enableSharding("analyticsDB")
{
        "ok" : 1,
        "operationTime" : Timestamp(1765462031, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1765462031, 3),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.shardCollection("analyticsDB.events", { userId: "hashed" })
{
        "collectionsharded" : "analyticsDB.events",
        "collectionUUID" : UUID("a632fb3b-c3a2-452d-aa21-0a3b3b82780c"),
        "ok" : 1,
        "operationTime" : Timestamp(1765462038, 10),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1765462038, 10),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> db.events.getShardDistribution()

Shard shardA at shardA/shardA1:27018,shardA2:27018,shardA3:27018
 data : 17.17MiB docs : 100000 chunks : 1
 estimated data per chunk : 17.17MiB
 estimated docs per chunk : 100000

Totals
 data : 17.17MiB docs : 100000 chunks : 1
 Shard shardA contains 100% data, 100% docs in cluster, avg obj size on shard : 180B

mongos>
```

The Issue

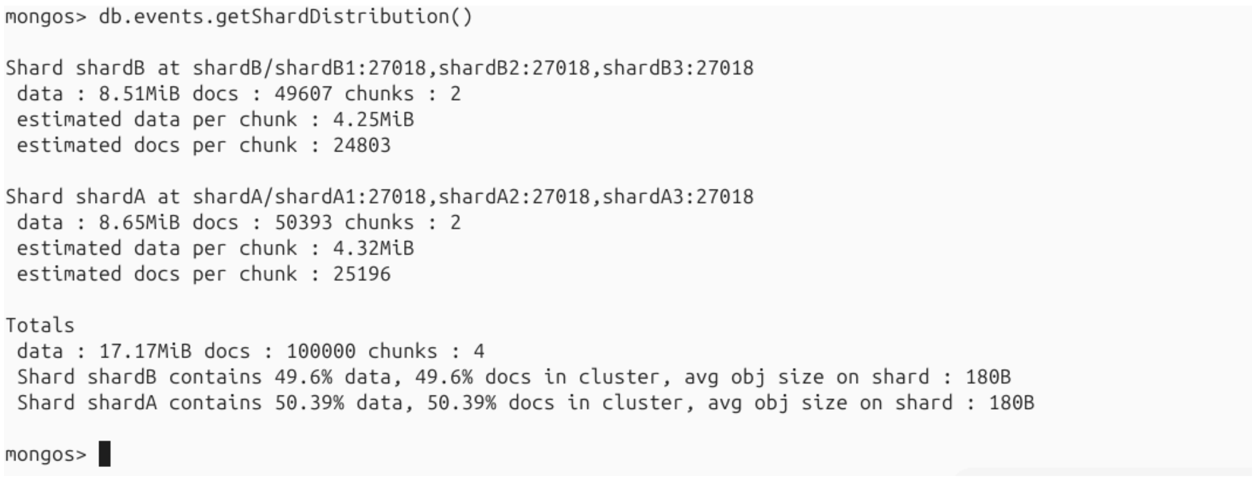
db.events.getShardDistribution() shows all 100,000 docs on shardA. This occurred because, although sharding was enabled on the `analyticsDB` database, the `events` collection itself was never explicitly sharded. Without a proper shard key, MongoDB cannot distribute documents across shards, so all data defaults to a single shard.

Solution

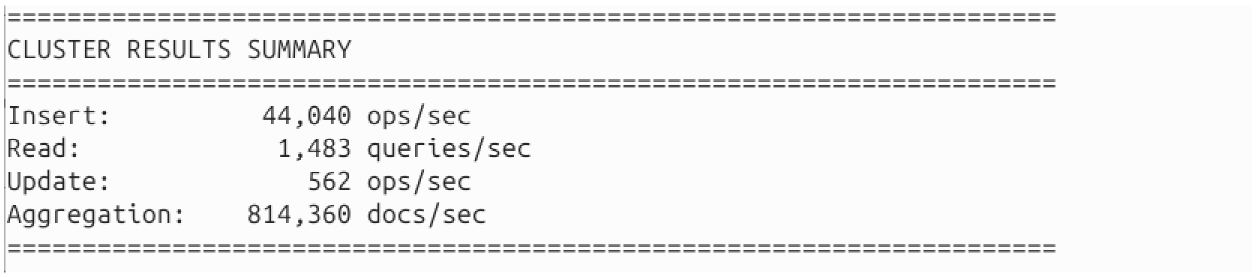
To resolve this, we modified the script to drop the collection before inserting any data, ensuring a clean slate. We then explicitly sharded the `events` collection using `userId` as the shard key and applied hashed sharding with this command:

```
mongos.admin.command('shardCollection', 'analyticsDB.events',
key={'userId': 'hashed'})
```

After implementing these changes and inserting the data, documents were properly balanced across the shards. This not only fixed the distribution issue but also improved performance metrics.

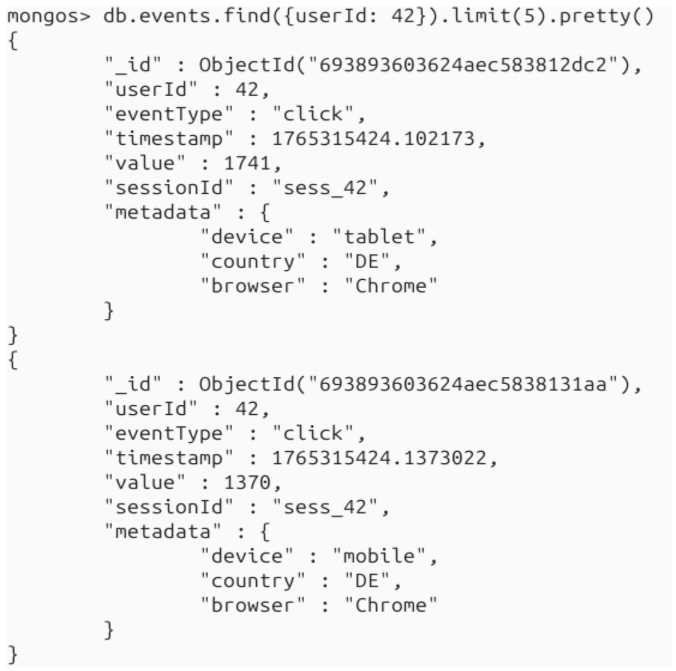


```
mongos> db.events.getShardDistribution()

Shard shardB at shardB/shardB1:27018,shardB2:27018,shardB3:27018
 data : 8.51MiB docs : 49607 chunks : 2
 estimated data per chunk : 4.25MiB
 estimated docs per chunk : 24803

Shard shardA at shardA/shardA1:27018,shardA2:27018,shardA3:27018
 data : 8.65MiB docs : 50393 chunks : 2
 estimated data per chunk : 4.32MiB
 estimated docs per chunk : 25196

Totals
 data : 17.17MiB docs : 100000 chunks : 4
 Shard shardB contains 49.6% data, 49.6% docs in cluster, avg obj size on shard : 180B
 Shard shardA contains 50.39% data, 50.39% docs in cluster, avg obj size on shard : 180B

mongos> █
```

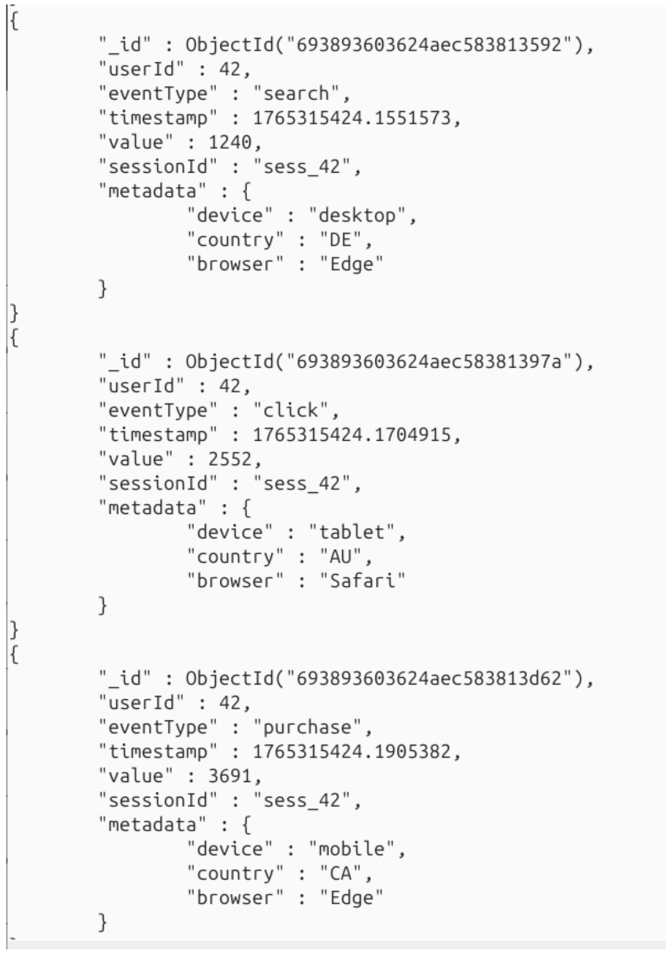


```
======================================================================
CLUSTER RESULTS SUMMARY
======================================================================
Insert:          44,040 ops/sec
Read:             1,483 queries/sec
Update:             562 ops/sec
Aggregation:    814,360 docs/sec
======================================================================
```

db.events.find({userId: 42}).limit(5).pretty()



```
mongos> db.events.find({userId: 42}).limit(5).pretty()
{
        "_id" : ObjectId("693893603624aec583812dc2"),
        "userId" : 42,
        "eventType" : "click",
        "timestamp" : 1765315424.102173,
        "value" : 1741,
        "sessionId" : "sess_42",
        "metadata" : {
                "device" : "tablet",
                "country" : "DE",
                "browser" : "Chrome"
        }
}
{
        "_id" : ObjectId("693893603624aec5838131aa"),
        "userId" : 42,
        "eventType" : "click",
        "timestamp" : 1765315424.1373022,
        "value" : 1370,
        "sessionId" : "sess_42",
        "metadata" : {
                "device" : "mobile",
                "country" : "DE",
                "browser" : "Chrome"
        }
}
```



```
{
        "_id" : ObjectId("693893603624aec583813592"),
        "userId" : 42,
        "eventType" : "search",
        "timestamp" : 1765315424.1551573,
        "value" : 1240,
        "sessionId" : "sess_42",
        "metadata" : {
                "device" : "desktop",
                "country" : "DE",
                "browser" : "Edge"
        }
}
{
        "_id" : ObjectId("693893603624aec58381397a"),
        "userId" : 42,
        "eventType" : "click",
        "timestamp" : 1765315424.1704915,
        "value" : 2552,
        "sessionId" : "sess_42",
        "metadata" : {
                "device" : "tablet",
                "country" : "AU",
                "browser" : "Safari"
        }
}
{
        "_id" : ObjectId("693893603624aec583813d62"),
        "userId" : 42,
        "eventType" : "purchase",
        "timestamp" : 1765315424.1905382,
        "value" : 3691,
        "sessionId" : "sess_42",
        "metadata" : {
                "device" : "mobile",
                "country" : "CA",
                "browser" : "Edge"
        }
```

Results Analysis

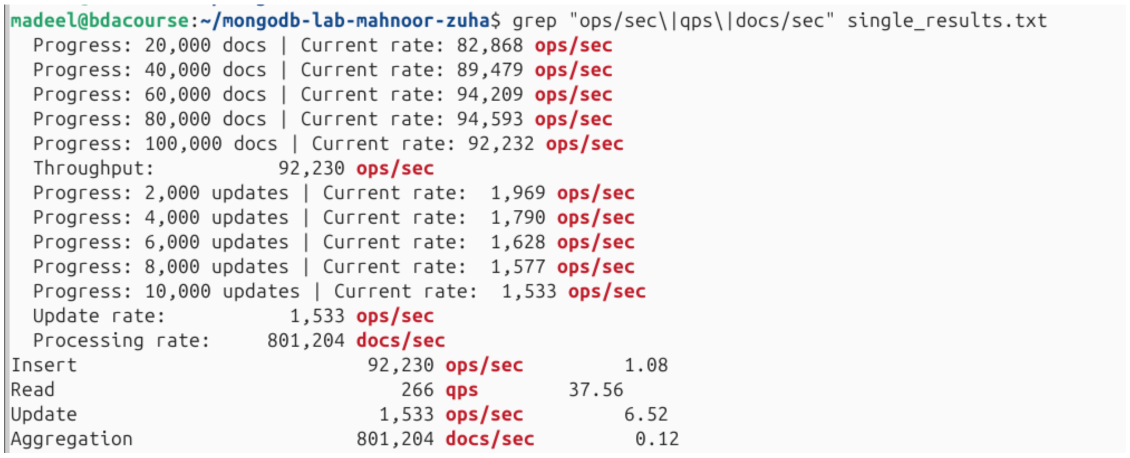
Operation	Single	Cluster (Before Sharding)	Cluster (After Sharding)	Improvement %
				(Cluster-Single) / Single
Insert (ops/sec)	88,890	48,707	44,040	**-50.45%**
Read (qps)	266	232	1,483	**457.4%**
Update (ops/sec)	1533	521	562	**-63.3%**
Aggregation (docs/sec)	801,204	529,605	814,360	**1.64%**

Explanation

- Sharded clusters introduce **overhead**: routing through `mongos`, **writing to multiple replica sets**, and **maintaining metadata** in the config servers. This leads to inserts being slowed down.
- Reads are now **parallelized across shards**. Each shard can process queries independently, so the cluster scales read throughput very well. Thus, harding dramatically improves **read performance**.
- However, updates still **need to locate the correct shard using the shard key**. Coordination overhead **reduces throughput**. Some updates may not be routed efficiently if the update key is not the shard key.
- Aggregations are now executed **in parallel across shards** and a very slight improvement is noticed.

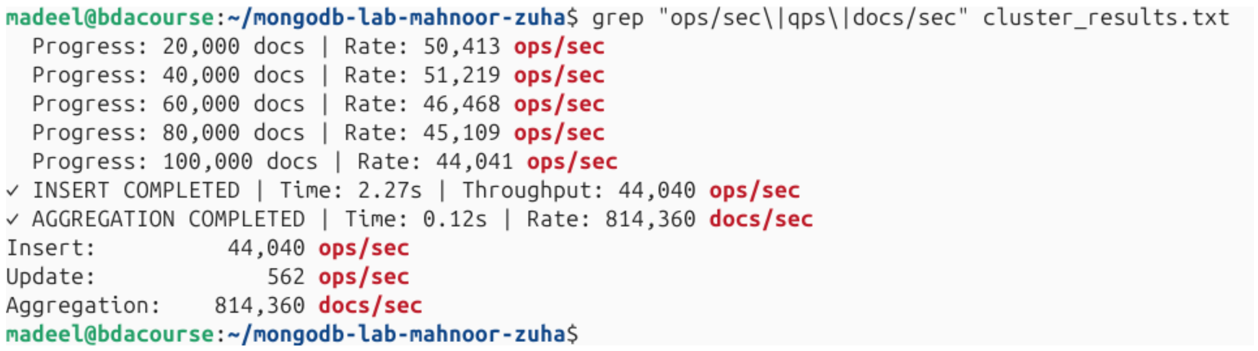
Extract data from logs

grep "ops/sec\|qps\|docs/sec" single_results.txt



```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ grep "ops/sec\|qps\|docs/sec" single_results.txt
  Progress: 20,000 docs | Current rate: 82,868 ops/sec
  Progress: 40,000 docs | Current rate: 89,479 ops/sec
  Progress: 60,000 docs | Current rate: 94,209 ops/sec
  Progress: 80,000 docs | Current rate: 94,593 ops/sec
  Progress: 100,000 docs | Current rate: 92,232 ops/sec
  Throughput:           92,230 ops/sec
  Progress: 2,000 updates | Current rate:  1,969 ops/sec
  Progress: 4,000 updates | Current rate:  1,790 ops/sec
  Progress: 6,000 updates | Current rate:  1,628 ops/sec
  Progress: 8,000 updates | Current rate:  1,577 ops/sec
  Progress: 10,000 updates | Current rate:  1,533 ops/sec
  Update rate:           1,533 ops/sec
  Processing rate:     801,204 docs/sec
Insert                        92,230 ops/sec         1.08
Read                             266 qps        37.56
Update                         1,533 ops/sec         6.52
Aggregation                  801,204 docs/sec         0.12
```

grep "ops/sec\|qps\|docs/sec" cluster_results.txt



```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ grep "ops/sec\|qps\|docs/sec" cluster_results.txt
  Progress: 20,000 docs | Rate: 50,413 ops/sec
  Progress: 40,000 docs | Rate: 51,219 ops/sec
  Progress: 60,000 docs | Rate: 46,468 ops/sec
  Progress: 80,000 docs | Rate: 45,109 ops/sec
  Progress: 100,000 docs | Rate: 44,041 ops/sec
✓ INSERT COMPLETED | Time: 2.27s | Throughput: 44,040 ops/sec
✓ AGGREGATION COMPLETED | Time: 0.12s | Rate: 814,360 docs/sec
Insert:          44,040 ops/sec
Update:             562 ops/sec
Aggregation:    814,360 docs/sec
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$
```

Lab Questions

Question 1

Q1: Performance Analysis Which operation showed the greatest performance gain in the cluster? Explain why based on the architecture differences.

The **READ** operations showed significant improvement with an increase **from 266 queries per second to 1483 queries per second**. This is because we now have parallel processing across shards. The mongos query router directs queries for different userIds to their relevant shards. Both shards can now independently handle queries on their half of the dataset without creating a bottleneck. Unlike a single node, where all queries are handled by a single server and throughput is reduced. Moreover, we used **hashed userId** as the shard key, which is evenly distributed across the shards, directing documents to both shards evenly.

For the **AGGREGATION** operation, results improved only slightly because computation is done in parallel across shards, but the `mongos` merge step offsets much of the gain for small datasets.

For the **INSERT** and **UPDATE** operations, replication to secondaries, and metadata updates to config servers are required. This coordination adds overhead compared to a single-node setup.

Question 2

Q2: Data Distribution What problems would occur if shard A contained 90% of documents instead of 50%?

If shard A contained 90% of the documents, we would face **hotspooting problem** where **shard A becomes the hotspot**. Most reads, writes, and updates would hit shard A's primary, so shard A's CPU/disk would saturate and users will experience slower responses, while **shard B stays underutilized**. This makes the whole cluster behave like a stressed single machine, and the response time would degrade because one shard is doing almost all the work.

It would also create operational overhead because **MongoDB's balancer** would keep trying to move chunks from shard A to shard B to restore balance. That migration creates extra network and disk load, so instead of improving performance, the cluster would waste resources rebalancing.

Moreover, in case shard A primary fails, 90% of the documents would be inaccessible until a new primary is elected for the replica set.

Question 3

Q3: Shard Key Design Analyze these alternative shard key choices:

- **Timestamp field**
- **Boolean field**
- **Auto-incrementing integer**

What issues would each introduce?

For a timestamp shard key, the biggest issue is that timestamps increase sequentially. That means almost all new inserts go to the “latest” range, which typically lives on the same chunk and therefore the same shard for some time. This creates a write hotspot and forces frequent chunk splits and migrations, increasing overhead and preventing good parallelism for ingest-heavy workloads like event streams.

For a boolean shard key, there are only two possible values (true/false), so the data can only be split into two huge groups. If one value is more common (which is very likely), one shard gets overloaded and the other stays mostly idle, making balancing very limited. For an auto-incrementing integer, the problem is again monotonic growth—new IDs keep landing in the same “end” region which causes the same hotspot pattern as timestamps.

Because of these issues, using a hashed shard key (like hashed userId) is typically the correct strategy to get uniform distribution. However, in our case, the results show that even with hashed userId, the collection did not spread across shards (we ended up with 1 chunk on shardA), so the shard key choice alone wasn’t enough, we also needed chunk distribution to actually occur.

Question 4

Q4: Scaling Strategy The dataset will grow from 100K to 10M documents next year, with 10x traffic increase. Propose a scaling plan.

Since our dataset is expected to grow from 100K to 10M documents with a 10× increase in traffic, the first step would be to scale the cluster both horizontally and vertically. First, the current shard key (`userId`, hashed) should be reviewed to ensure it maintains even data distribution at higher volumes. If queries frequently filter by time ranges, using a compound shard key, such as `userId + timestam`p, can further improve distribution and query efficiency.

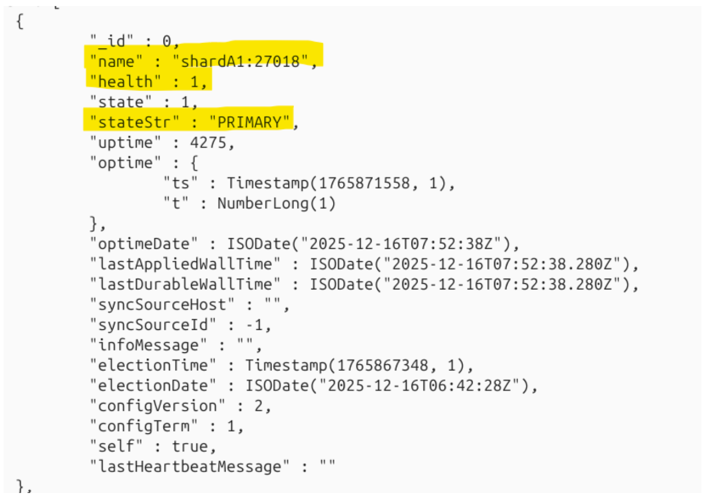
Next, we can scale horizontally by adding more shards, each configured as a 3-node replica set. The existing shards may require upgrades in RAM and CPU to handle the larger working set efficiently. The MongoDB balancer should be actively monitored to ensure chunks are evenly distributed and to prevent **hotspotting** situations that could create bottlenecks.

We would also improve indexing based on query patterns (for example, indexes involving `userId`, and possibly compound indexes involving `userId` + `timestamp` depending on analytics). For heavy read workloads, we can also use replica set secondaries for read scaling where possible, while monitoring replication lag to avoid stale results. Finally, upgrading storage (SSD) and ensuring enough RAM on primaries helps a lot once the working set grows into millions of documents.

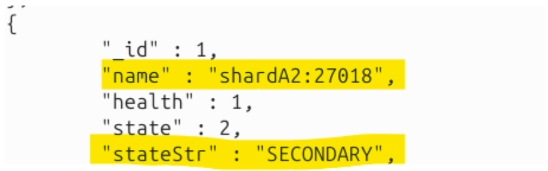
Question 5

Q5: Failover Test Execute this failover scenario:
docker-compose -f docker-compose.cluster.yml up -d
sleep 60
docker exec -it shardA1 mongo --port 27018 --eval "rs.status()"
docker kill shardA1
sleep 30
docker exec -it shardA2 mongo --port 27018 --eval "rs.status()"
docker exec -it mongos mongo --eval "db.getSiblingDB('analyticsDB').events.count()"
docker start shardA1
Measure the downtime and describe shardA1's role after restart.

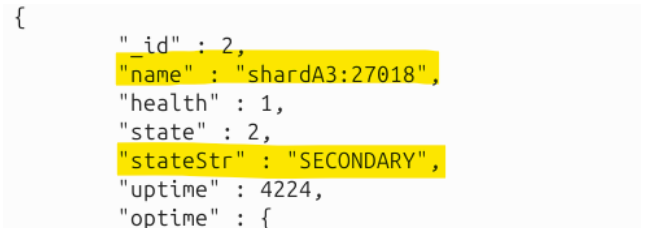
In our failover test, we checked cluster availability through `mongos` and verified the dataset remained accessible. Before simulating the failure, we confirmed the cluster still had the analyticsDB.events collection available through `mongos`, and the command `db.getSiblingDB('analyticsDB').events.count()` returned 100000. At this stage, when we attempted to run `rs.status()` on `shardA`1, Docker reported that `shardA1` was **healthy and active as the primary**, with shardA2 and shardA3 functioning as secondaries. This baseline ensured that the replica set was fully operational prior to simulating a failover scenario, allowing us to accurately observe the behavior of the cluster when the primary becomes unavailable.



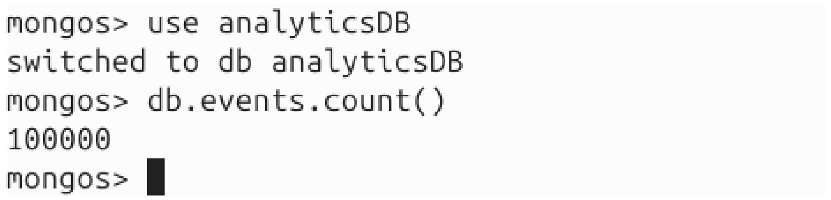
```
{
        "_id" : 0,
        "name" : "shardA1:27018",
        "health" : 1,
        "state" : 1,
        "stateStr" : "PRIMARY",
        "uptime" : 4275,
        "optime" : {
                "ts" : Timestamp(1765871558, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2025-12-16T07:52:38Z"),
        "lastAppliedWallTime" : ISODate("2025-12-16T07:52:38.280Z"),
        "lastDurableWallTime" : ISODate("2025-12-16T07:52:38.280Z"),
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "",
        "electionTime" : Timestamp(1765867348, 1),
        "electionDate" : ISODate("2025-12-16T06:42:28Z"),
        "configVersion" : 2,
        "configTerm" : 1,
        "self" : true,
        "lastHeartbeatMessage" : ""
},
```



```
{
        "_id" : 1,
        "name" : "shardA2:27018",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
```

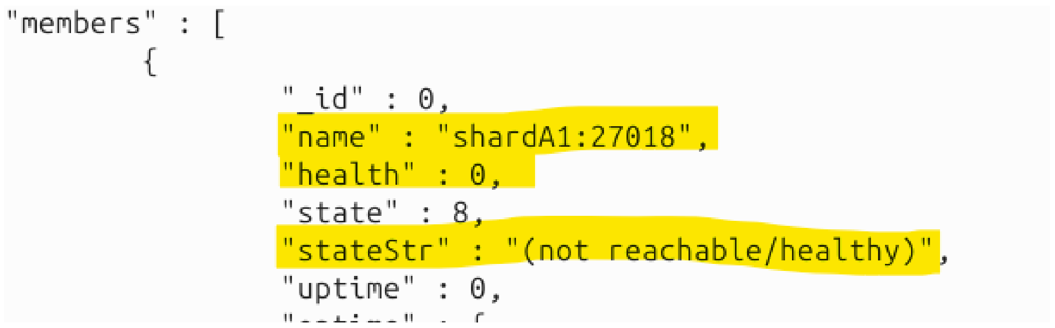


```
{
        "_id" : 2,
        "name" : "shardA3:27018",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 4224,
        "optime" : {
```



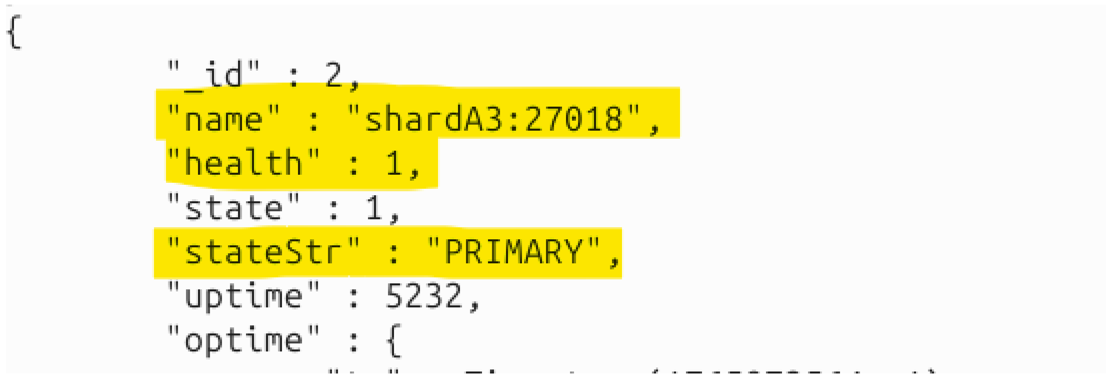
```
mongos> use analyticsDB
switched to db analyticsDB
mongos> db.events.count()
100000
mongos> 
```

From the `rs.status()` output (queried via `shardA`2), we observed that **shardA1** became **`(not reachable/healthy)` with health = 0**.



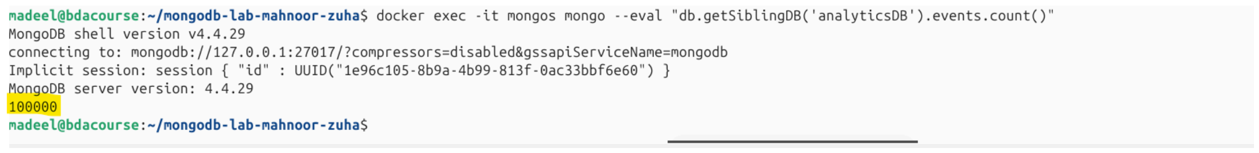
```
"members" : [
        {
                "_id" : 0,
                "name" : "shardA1:27018",
                "health" : 0,
                "state" : 8,
                "stateStr" : "(not reachable/healthy)",
                "uptime" : 0,
```

ShardA3 was elected as the **new PRIMARY** and shardA2 stayed SECONDARY. This confirms that MongoDB replica-set failover worked correctly: when the previously expected node (`shardA1`) was down, another member (`shardA3`) automatically took over as PRIMARY.



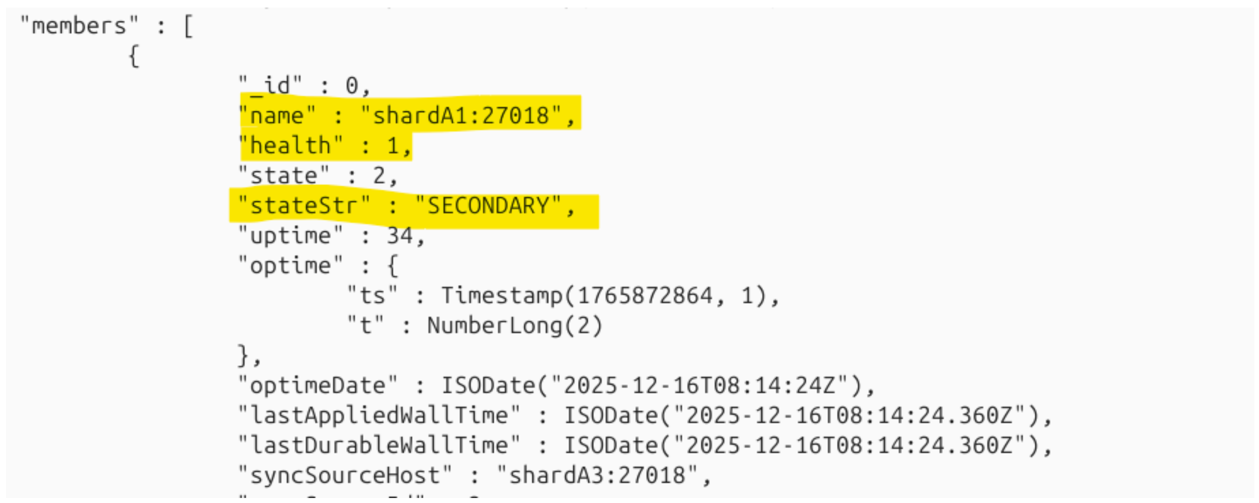
```
{
        "_id" : 2,
        "name" : "shardA3:27018",
        "health" : 1,
        "state" : 1,
        "stateStr" : "PRIMARY",
        "uptime" : 5232,
        "optime" : {
```

We then verified cluster continuity again through **mongos**, and **events.count()** still returned 100000 at, meaning the cluster remained available for reads during/after the election.



```
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$ docker exec -it mongos mongo --eval "db.getSiblingDB('analyticsDB').events.count()"
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("1e96c105-8b9a-4b99-813f-0ac33bbf6e60") }
MongoDB server version: 4.4.29
100000
madeel@bdacourse:~/mongodb-lab-mahnoor-zuha$
```

After that, we restarted **shardA1** using **docker start shardA1** and waited ~20 seconds for it to rejoin. When we checked the status again, **shardA1** returned as SECONDARY (health = 1), while **shardA3** stayed as PRIMARY and shardA2 remained SECONDARY. This is expected replica-set behavior: after a failover, the restarted node does not immediately become PRIMARY; instead, it rejoins as a SECONDARY and synchronizes from the current primary. Overall, our downtime was minimal from the application perspective (counts remained available), and the leadership change shows the cluster achieved high availability through automatic primary election and replica recovery.

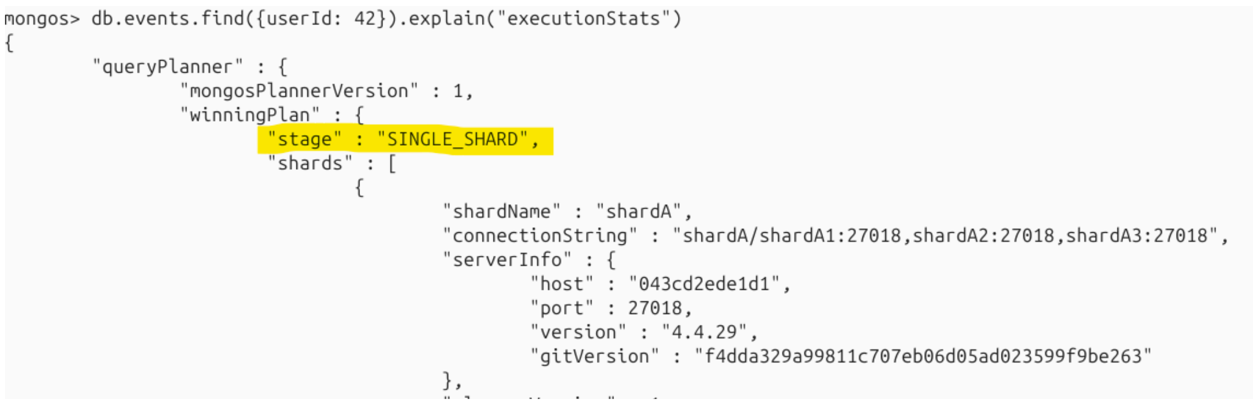


```
"members" : [
        {
                "_id" : 0,
                "name" : "shardA1:27018",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 34,
                "optime" : {
                        "ts" : Timestamp(1765872864, 1),
                        "t" : NumberLong(2)
                },
                "optimeDate" : ISODate("2025-12-16T08:14:24Z"),
                "lastAppliedWallTime" : ISODate("2025-12-16T08:14:24.360Z"),
                "lastDurableWallTime" : ISODate("2025-12-16T08:14:24.360Z"),
                "syncSourceHost" : "shardA3:27018",
```

Question 6

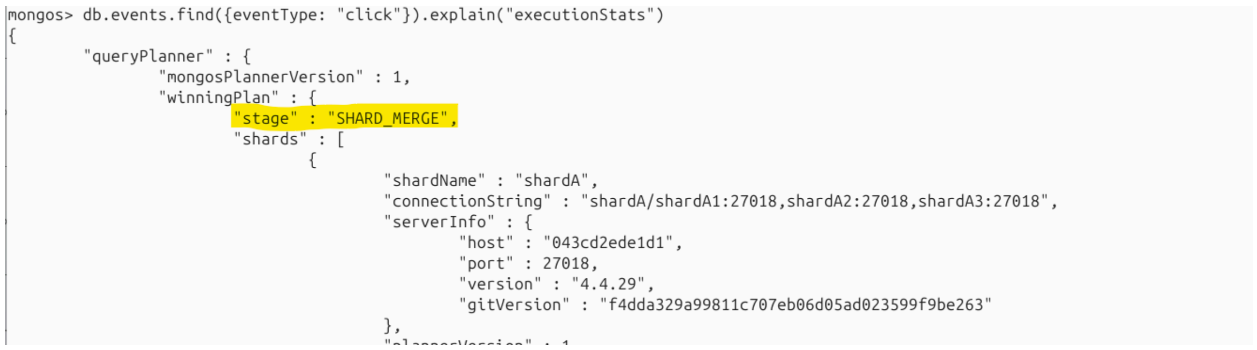
Q6: Query Routing: Compare these queries using .explain("executionStats"):
You can directly run from terminal too:
db.events.find({userId: 42})   // targeted
db.events.find({eventType: "click"})   // scatter-gather
How many shards does each query hit?

When we compare the two queries, **db.events.find({userId: 42})** is supposed to be a targeted query because **userId** is our shard key (hashed). Mongos would route the query directly to the shard that owns the chunk containing the hashed value, so this query hits only one shard, shard A in this case, as shown below.



```
mongos> db.events.find({userId: 42}).explain("executionStats")
{
        "queryPlanner" : {
                "mongosPlannerVersion" : 1,
                "winningPlan" : {
                        "stage" : "SINGLE_SHARD",
                        "shards" : [
                                {
                                        "shardName" : "shardA",
                                        "connectionString" : "shardA/shardA1:27018,shardA2:27018,shardA3:27018",
                                        "serverInfo" : {
                                                "host" : "043cd2ede1d1",
                                                "port" : 27018,
                                                "version" : "4.4.29",
                                                "gitVersion" : "f4dda329a99811c707eb06d05ad023599f9be263"
                                        },
```

On the other hand, **`db.events.find({eventType: "click"})`** is a scatter–gather query because **`eventType`** is not part of the shard key. Mongos would need to query all shards (or all shards that may contain matching chunks) and merge results as shown below in the snapshot. That makes it slower and more expensive as the number of shards grows.



```
mongos> db.events.find({eventType: "click"}).explain("executionStats")
{
        "queryPlanner" : {
                "mongosPlannerVersion" : 1,
                "winningPlan" : {
                        "stage" : "SHARD_MERGE",
                        "shards" : [
                                {
                                        "shardName" : "shardA",
                                        "connectionString" : "shardA/shardA1:27018,shardA2:27018,shardA3:27018",
                                        "serverInfo" : {
                                                "host" : "043cd2ede1d1",
                                                "port" : 27018,
                                                "version" : "4.4.29",
                                                "gitVersion" : "f4dda329a99811c707eb06d05ad023599f9be263"
                                        },
```

Question 7

Q7: Bottleneck Analysis: Identify the primary bottleneck in the single instance (CPU/disk/RAM/locks). Cite specific log evidence.

From our single-instance logs, we can identify the main bottleneck as being the limitations of running everything on one mongos process. The insert section explicitly states: “Bottleneck: Single node disk I/O + CPU”, which makes sense because all writes, journaling, and storage engine operations happen on the same machine. This becomes the limiting factor once concurrency increases or the dataset grows.

For updates and aggregation, the output also indicates single-node constraints clearly. The update results mention “Single node write lock contention”, which means updates compete for write capacity and internal contention within one instance. The aggregation section also notes “Single CPU core bottleneck,” meaning complex pipelines are CPU-bound when everything runs locally on one node. So overall, our evidence points to disk + CPU saturation and write contention being the core reasons a single instance doesn’t scale under higher load.

Question 8

Q8: Infrastructure Economics: The cluster uses 10x the containers of single instance. In what scenarios does this cost multiply justify itself?

The cluster uses significantly more components (multiple replica set members across config servers and shards, plus mongos), so it costs more than a single instance. That cost becomes justified when we need high availability (no single point of failure), horizontal scaling (data larger than one node can hold), and sustained high throughput that one machine cannot handle. In real production systems with huge event streams and strict uptime requirements, these benefits outweigh the operational cost.

In our case, while insert and update throughput experienced a slight decrease due to routing and replication overhead, we still achieved high overall performance, including efficient aggregation and query processing across the cluster. The read throughput improved significantly, but more importantly, the architecture allows all operations, inserts, updates, reads, and aggregations, to be distributed across multiple shards, reducing the load on any single node and preventing bottlenecks.

For larger enterprise datasets, containing tens or hundreds of millions of documents and higher traffic, this architecture can be scaled horizontally by adding more shards and replica sets, ensuring even data distribution and maintaining high availability. Proper shard key selection and indexing further optimize performance, prevent hot shards, and enable parallelized writes and aggregations. Therefore, while the upfront cost is higher, a sharded cluster provides a robust, scalable, and fault-tolerant infrastructure capable of supporting enterprise-level workloads across all types of operations, not just reads.

End of Lab