MongoDB Log File Lab

CSC compliance platform where MongoDB is the backend, and we need to **find & fix slow queries** using logs + profiler.

1. Log file structure
2. Slow query logs
3. Profiling levels
4. Hands-on lab (step-by-step) for CSC + explanation of each step

Assume a **self-managed mongod** (Linux/Windows) with access to logs and mongosh.

---

1. MongoDB Log File Structure (Basics)

MongoDB writes a **diagnostic log** (usually called mongod.log).

Typical locations:

- Linux: /var/log/mongodb/mongod.log
- Windows (service install): C:\Program Files\MongoDB\Server\<version>\log\mongod.log (or wherever you configured)

A typical log line looks like:

```
2025-01-25T14:32:10.123+0530 I  COMMAND  [conn12] Slow query: ...
```

Breakdown:

- 2025-01-25T14:32:10.123+0530 → **timestamp**
- I → severity level (I=Info, W=Warning, E=Error, F=Fatal)
- COMMAND → component (e.g., NETWORK, STORAGE, QUERY, COMMAND)
- [conn12] → connection / thread id
- Slow query: ... → message text

You will see entries for:

- Server startup/shutdown
- Connections opening/closing
- Replication, elections (if replica set)
- Slow operations (if slow logging enabled)

For CSC, your SRE/DBA team would tail this log to see **spikes in slow operations** when end-of-year filings peak.

---

2. Slow Query Logs

MongoDB can log **slow operations** automatically.

Key parameters (typically in mongod.conf):

```
setParameter:
  slowOpThresholdMs: 100  # ops slower than 100 ms are slow
  logLevel: 0             # base log verbosity
```

- **slowOpThresholdMs** – any operation taking longer than this is logged as a slow op.
- MongoDB logs slow operations with details like collection, filter, sort, execution time, etc.

Example slow query log snippet (simplified):

```
2025-01-25T14:35:02.456+0530 I  COMMAND  [conn15]
Slow query: find csc_compliance.filings
planSummary: COLLSCAN
keysExamined:0 docsExamined:500000
keysInserted:0 numYields:390
query: { state: "DE", status: "OPEN" }
millis: 732
```

For **CSC** this might correspond to:

"Show all OPEN Delaware filings"
– taking 732 ms because there is no proper index.

This is the kind of evidence you later use to justify **index creation**.

---

3. Profiling Levels (Database Profiler)

The **database profiler** stores detailed information about operations in the system.profile collection.

Profiling Levels

Configured per **database**:

```
db.setProfilingLevel(level, { slowms: <ms> })
```

- **0** – Off (no profiling, only very minimal metadata)

- **1** – Profile **slow operations** only (slower than slowms)
- **2** – Profile **all operations** (VERY verbose; use only temporarily)

You can check current status:

```
db.getProfilingStatus()
```

Profiler output is written to:

- db.system.profile collection inside that database
- Not to the mongod.log file (logs and profiler are separate but complementary)

---

4. Hands-On Lab – CSC Scenario

CSC Scenario

CSC runs a **Compliance Dashboard** on database csc_compliance.

Product team complains:

"The **Open Filings** screen for Delaware clients is slow around peak time. Please analyze and fix."

We'll:

1. Enable profiler with a small threshold
2. Run a few "bad" queries
3. Inspect system.profile
4. Relate findings to the log file
5. Suggest improvements

Step 0 – Setup (switch DB and sample data)

```
use csc_compliance
```

Ensure you have collections:

- filings (as in previous exercises)
- entities

If needed, quickly insert some sample data:

```
db.filings.insertMany([
  {
```

```
    entity_id: 1001,
    filing_type: "Annual Report",
    state: "DE",
    due_date: ISODate("2025-03-01"),
    filed_date: null,
    status: "OPEN",
    amount: 5000
  },
  {
    entity_id: 1001,
    filing_type: "Franchise Tax",
    state: "DE",
    due_date: ISODate("2025-03-15"),
    filed_date: null,
    status: "OPEN",
    amount: 20000
  },
  {
    entity_id: 1002,
    filing_type: "Annual Report",
    state: "CA",
    due_date: ISODate("2025-02-10"),
    filed_date: ISODate("2025-02-01"),
    status: "FILED",
    amount: 1500
  }
])
```

In a real lab, you would bulk-load 10k+ filings so slow queries are visible.

Step 1 – Locate & Understand Log File (Theory + Quick Check)

Linux:

```
ps aux | grep mongod

# find mongod.conf path, then:

grep path /etc/mongod.conf        # often shows log path

cat /etc/mongod.conf | grep log
```

You might see:

```
systemLog:
  destination: file
  path: /var/log/mongodb/mongod.log
  logAppend: true
```

Windows Service install:

- Open services.msc → right-click MongoDB service → Properties → see config path or startup parameters.
- Open the referenced config or the log folder in C:\Program Files\MongoDB\....

Now open log:

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

"This is where we'll see slow query lines and other diagnostic messages."

Step 2 – Enable Profiler (Level 1, small threshold)

We want to capture **slow ops only**, but make "slow" small enough for the lab.

In mongosh:

```
use csc_compliance

db.setProfilingLevel(
  1,
  { slowms: 20 }  // anything slower than 20 ms will be captured
)
```

Confirm:

```
db.getProfilingStatus()
```

Expected:

```
{ was: 0, slowms: 20, sampleRate: 1, level: 1 }
```

Use Case Explanation (CSC):

"We are enabling profiling for the csc_compliance database so we can see which operations our dashboard is performing slowly, without recording every single tiny operation."

Step 3 – Run "Bad" Queries (Simulate Slow Screen)

Simulate the **Open Filings for Delaware** screen:

```
// Query 1 - no index yet, may be slow if dataset is large

db.filings.find({
  status: "OPEN",
  state: "DE"
}).sort({ due_date: 1 })
```

Simulate another heavy query:

```
// Query 2 - using a range on amount

db.filings.find({
  status: "OPEN",
  amount: { $gt: 10000 }
})
```

In a real lab with many documents, these will cross 20 ms and be profiled.

Step 4 – Inspect Profiler Output (system.profile)

Now look at the last few profiled operations:

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

You'll see documents like:

```
{
  op: "query",
  ns: "csc_compliance.filings",
  command: {
    find: "filings",
    filter: { status: "OPEN", state: "DE" },
    sort: { due_date: 1 }
  },
  keysExamined: 0,
  docsExamined: 500000,
  millis: 732,
  planSummary: "COLLSCAN",
  ts: ISODate("2025-01-25T09:05:02.456Z")
}
```

Explain each important field:

- ns – namespace → which collection
- command.filter – what filter was used
- docsExamined – how many docs scanned
- millis – how long it took
- planSummary – COLLSCAN vs IXSCAN

CSC Use Case Explanation:

"This is our CSC dashboard query.
It scans 500k filings and takes 732 ms – clearly a performance issue.
We now know *exactly* which query is slow and how bad it is."

Step 5 – Relate to Log File (Slow Query Entry)

If slowOpThresholdMs is configured (say 100 ms), the same operation should appear in the mongod.log.

While logs are being tailed:

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

Re-run the slow query.

You should see a line similar to:

```
I  COMMAND [conn15] Slow query: find csc_compliance.filings
planSummary: COLLSCAN
...
docsExamined:500000 keysExamined:0
...
millis:732
```

Explain:

- Logs give **high-level overview** across all DBs.
- Profiler gives **structured, query-level detail** inside the DB.

For CSC production SRE:

- Logs → quick alerting, dashboards (e.g., via ELK, Datadog, Splunk).
- Profiler → deep dive into one database's behavior.

Step 6 – Use Findings to Improve (Optional, but Powerful)

From profiler we saw:

- planSummary: "COLLSCAN"
- Filters on status and state
- Sort on due_date

Design an index:

```
db.filings.createIndex({ status: 1, state: 1, due_date: 1 })
```

Re-run the query:

```
db.filings.explain("executionStats").find({
  status: "OPEN",
  state: "DE"
}).sort({ due_date: 1 })
```

Profiler again:

```
db.system.profile.find()
  .sort({ ts: -1 })
```

```
  .limit(1)
  .pretty()
```

Now you should see:

- planSummary: "IXSCAN"
- docsExamined ≈ nReturned (much smaller)
- millis significantly lower

Tie back to CSC:

"We used logs + profiler to *identify* the slow Open Filings query,
then created an index to *fix* it, and verified improvement scientifically."

Step 7 – Reset Profiling After Lab

After the lab, turn profiling off to avoid overhead:

```
db.setProfilingLevel(0)
db.getProfilingStatus()
```

Final Recap

MongoDB Logging Basics @ CSC

- **Log file structure:** timestamp, severity, component, connection, message – used by SREs for monitoring and alerts.
- **Slow query logs:** configured by slowOpThresholdMs, show slow operations with collection, filter, plan, and millis.
- **Profiler levels:**
  - 0 – off
  - 1 – slow ops only (with slowms)
  - 2 – all ops (short-term diagnostics only).
- **Lab flow (CSC use case):**

  1. Enable profiler on csc_compliance with small slowms
  2. Run "Open Filings" queries
  3. Inspect system.profile for slow operations
  4. Correlate with mongod.log slow query entries
  5. Design indexes / changes and re-measure