Release: 20250221

Topics we cover

Database Logs in MongoDB

How to download logs from Atlas?

The getLog command

The log file output

Database Profiler

Finding slow operations

Logging

MongoDB maintains a log of events, including entries such as incoming connections, commands run, and issues encountered

Useful for :

- Diagnosing issues related to slow queries
- Monitoring your deployment
- Tuning performance

Accessing the Database Logs

The Database log file is stored as a text file on the server host

If hosted in Atlas, use the GUI/API to download

Access the last 1024 lines of the log via getLog

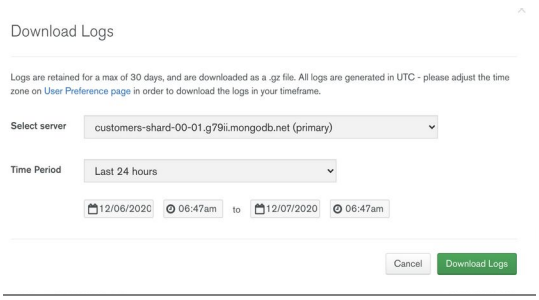
File System	/var/log/mongodb/mongod.log
Hosted in Atlas	Download from the UI or with API
Database	Access with the **getLog** command

The default location of the log is /var/log/mongodb/mongodb.log; however, this often varies depending on how MongoDB has been configured.

Log from Atlas

In the Cluster page:

1. Select your cluster
2. Click the ellipsis icon (...) for to the cluster containing the mongod instance whose logs you want to download
3. Select Download Logs
4. Select host and time period



Note: Database Log is **not available** for download from shared tier servers (M0, M2, M5). It can be accessed using the getLog command (show log global command) but only has information specific to you - not general server information.

getLog command

The getLog command gets the last 1024 log entries

Returns a single document where **logs** are in an array as strings or JSON (from MDB 4.4)

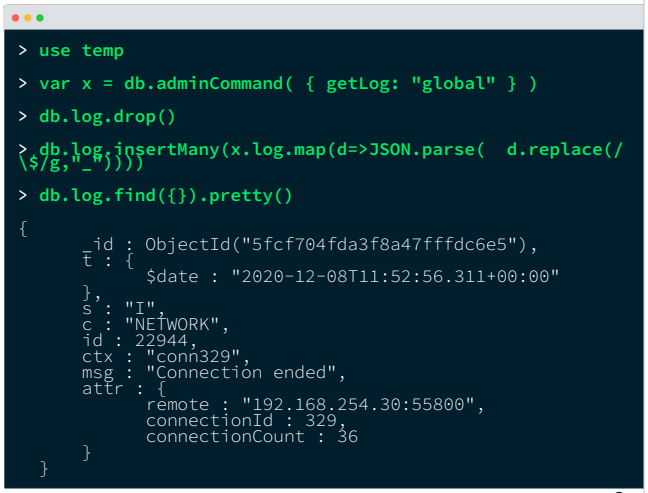
```
> var loglines = db.adminCommand({getLog:"global"})

> printjson(loglines)

{
      totalLinesWritten : 2625918,
      log : [ <LOG LINES HERE> ],
      ok : 1,
      $clusterTime : {
            clusterTime : Timestamp(1594293362, 5),
            signature : {
            }
      },
      operationTime : Timestamp(1594293362, 5)
}
```


- It does not read data from the mongod log file. It reads data from the RAM cache of logged mongod events.
- It returns log data in escaped Relaxed Extended JSON v2.0 format.
- totalLinesWritten is the size of the full on-disk logfile

Copying log to a collection

From MongoDB 4.4, log output is in JSON so can easily copy the getLog output to a temporary collection

And then search or aggregate the data




```
use temp
var x = db.adminCommand( { getLog: "global" } )
db.log.drop()

db.log.insertMany(x.log.map(l=>({l})))
db.log.find({l:/COLLSCAN/}).pretty()
```

We can also , from 4.4 load a log file into mongodb as a collection using mongoimport.

Querying the log database

Can run any query or aggregation on the database you output the log

Aggregation is a separate topic, but **$sortByCount** is a grouping operator useful to know

```
> use temp

> db.log.find({c:"ACCESS",msg:/^Successful/},
{_id:0,"attr.client":1}).pretty()
  { attr : { client : "192.168.248.14:52798" } }
  { attr : { client : "192.168.248.14:52808" } }
  { attr : { client : "192.168.248.14:52824" } }
  { attr : { client : "192.168.248.14:52838" } }
  { attr : { client : "192.168.248.14:52844" } }
  { attr : { client : "192.168.248.14:52860" } }
  { attr : { client : "192.168.248.14:52876" } }

> db.log.aggregate([{$sortByCount:"$c"}])
  { _id : "NETWORK", count : 728 }
  { _id : "ACCESS", count : 176 }
  { _id : "SHARDING", count : 105 }
  { _id : "CONNPOOL", count : 12 }
  { _id : "QUERY", count : 1 }
```


- To view the log in the shell just type **show log global**
- If the above query doesn't return result, try the following query:

```
db.log.find( { c:"ACCESS", msg:"Authentication succeeded" }, { _id:0,
"attr.remote":1 } ).pretty()
```

Finding slow operations using log

Queries appear in the log if their duration exceeds the default slow operation threshold (100ms)

Can see how long they took, their query plan, and many other values

```
> use temp
> query = {"attr.ns":{$ne :"local.oplog.rs"},c:"COMMAND"}
> bytime = {"attr.durationMillis":-1}
> db.log.find(query).sort(bytime).pretty()
```

Log file output format

Severity

- F: Fatal
- E: Error
- W: Warning
- I: Informational

The attr section is the most important to query but varies

Truncated shows if the log entry was too large

```
{
  t: <Datetime>, // timestamp
  s: <String>, // severity
  c: <String>, // component
  ctx: <String>, // context
  id: <String>, // unique identifier
  msg: <String>, // message body
  attr: <Object> // additional attributes (optional)
  tags: <Array of strings> // tags (optional)
  truncated: <Object> // truncation info (if truncated)
  size: <Integer> // original size of entry (if truncated)
}
```


In previous versions (versions < 4.2), MongoDB log messages specified D for all debug verbosity levels.

Log file output format

Components let you filter specific areas of functionality or messages

ACCESS	Messages related to access control, such as authentication
COMMAND	Messages related to database commands such as count
WRITE	Messages related to the write operations, such as update commands
INDEX	Messages related to indexing operations, such as creating indexes
JOURNAL	Messages related specifically to storage journaling activities
NETWORK	Messages related to network activities, such as accepting connections
QUERY	Messages related to queries, including query planner activities

Full set of components are - CONTROL, ELECTION, FTDC, GEO, INDEX, INITSYNC, JOURNAL, NETWORK, QUERY, RECOVERY, REPL, REPL_HB, ROLLBACK, SHARDING, STORAGE, TXN, WRITE, ACCESS

Database Profiler

Database profiler records information about slow operations:

- Definition of slow is configurable and can mean all operations
- The size of profiling data is capped at 10MB
- Recorded in a collection called system.profile
- Can be enabled globally or for any database
- Creates extra writes
  - Enabling profiler for every operation, creates extra write
  - Always turn off Profiling unless required - Affects performance

- The database profiler collects detailed information about Database commands executed against a running MongoD instance.
- This includes CRUD operations as well as configuration and administration commands.
- The profiler writes all the data it collects to the system.profile collection, a capped collection of 10 MB in the database being profiled
- The profiler is off by default, and you can enable the profiler on a per-database or per-instance basis

The setProfilingLevel command

`db.setProfilingLevel(level, slowms)`

level: 0, 1, or 2

0 - Profiler is off

1 - Store only slow queries

2 - Store all queries

slowms: threshold in milliseconds for slow operations

```
> db.setProfilingLevel(1,5)
  {"was" : 0,"slowms" : 100,"sampleRate" : 1,"ok" : 1, ... }
> use sample_airbnb
> db.listingsAndReviews.find({amenities:"Snooker"})
> use admin
> var loglines = db.runCommand({getLog:"global"})
> printjson(loglines)
...
"2021-07-09T11:49:00.237+0000 I  COMMAND  [conn214] command
sample_airbnb.listingsAndReviews appName: \"MongoDB Shell\" command:
find { find: \"listingsAndReviews\", filter: { amenities:
\"Snooker\" }, lsid: { id: UUID(\"6496eb58-cea3-4c05-b237-
0d04e2ffc60b\") }, $clusterTime: { clusterTime: Timestamp(1594295320,
1), signature: { hash: BinData(0,
E2DB7AAFBFE43F8A424F60EA04033378C1B73B89), keyId: 6847444150736912387 }
}, $db: \"sample_airbnb\" } planSummary: COLLSCAN keysExamined:0
docsExamined:5555 cursorExhausted:1 numYields:43 nreturned:0
queryHash:0AEEE9D2 planCacheKey:0AEEE9D2 reslen:246 locks:
{ ReplicationStateTransition: { acquireCount: { w: 44 } }, Global:
{ acquireCount: { r: 44 } }, Database: { acquireCount: { r: 44 } },
...
```


The db profiler can also be used to find slow queries - slowms sets the threshold for 'slow'
samplerate is what proportion 0-1(1 is 100%) of slow queries to log
Note: The setProfilingLevel() command is not allowed on Atlas Shared Tiers.

Note: The queries are stored in a collection system.profile.

Query Profiler Data

The **`show profile`** helper in the mongo shell displays the five most recent operations that match the **setProfilingLevel** filters.

We can also query or aggregate the **system.profile** collection in any database

```
> show profile
query sample_airbnb.listingsAndReviews 5ms Wed Dec 09
2020 16:39:04
command:{
      find : "listingsAndReviews",
      limit : NumberLong(21),
      maxTimeMS : NumberLong(15000),
      skip : NumberLong(0),
      $db : sample_airbnb,
      $readPreference : {
            mode : "primaryPreferred"
      }
}
keysExamined:0
docsExamined:21
cursorExhausted
numYield:0
nreturned:21
locks:{
      ReplicationStateTransition : { ...


> //Find the slowest ops
> db.system.profile.find().sort( { millis : -1 })
```



```
use sample_airbnb
```

And run above commands

db.serverStatus()

Returns a document that provides an overview of the state of the MongoDB instance (mongod or mongos)

Monitoring applications can run this command at a regular interval to collect statistics about the instance

Includes up to 1,500 parameters

```
> db.serverStatus()

{
      host : "atlas-10yszs-shard-00-
01.g79ii.mongodb.net",
      version : 6.0.6,
      process : "mongod",
      pid : NumberLong(16421),
      uptime : 340215,
      uptimeMillis : NumberLong(340215520),
      uptimeEstimate : NumberLong(340215),
      localTime : ISODate("2020-12-08T06:43:21.953Z"),
      asserts : {
            regular : 0,
            warning : 0,
            msg : 0,
            user : 3576,
            rollovers : 0
      },connections : {
current : 36,
available : 1464,
totalCreated : 41747,
active : 5,
}

     ...
```


The db.serverStatus() command returns a large amount of data. To return a specific object or field from the output, append the object or field name to the command.

`db.runCommand({ serverStatus: 1}).metrics` or `db.serverStatus().metrics`
`db.runCommand({ serverStatus: 1}).metrics.commands`
`db.runCommand({ serverStatus: 1}).metrics.commands.update`

Admin should use `db.serverStatus()`

`db.serverStatus()` and `db.runCommand({ serverStatus: 1})` both return the same values.

db.serverStatus()

To exclude fields that are not wanted, use the same syntax as projection

You can then see specific parts using the MongoDB shell

```
> db.serverStatus( { repl: 0,  metrics: 0, locks: 0 } )

> db.serverStatus().connections
{
  current: 83,
  available: 1417,
  totalCreated: 1227,
  rejected: 0,
  active: 27,
  threaded: 83,
  limitExempt: 67,
  exhaustIsMaster: 3,
  exhaustHello: 16,
  awaitingTopologyChanges: 19
}


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

Some causes of slow operations

- Missing indexes - leads to lots of Disk I/O and CPU usage
- Writes to the Cache outstripping disk write capability
  - Pauses waiting for cache to flush/checkpoint
- Not enough Cache allocated - leads to paging into cache
- Locking - you can see in logs what locks things take
  - Most things don't block others - but a few admin things do
  - One admin task could block production
- Excessive CPU usage
  - Constantly authenticating
  - Inappropriately large arrays in schemas
  - Running code in the database
  - Sorting without index
  - Usage of JS on the server

- These are many of the most common causes of performance problems.
- Often they come from bad design or a lack of understanding - fixing most of these is
- not something an Operations person can do without a developer.

- JS on the server has been deprecated as of MongoDB version 8.0 and should be moved away from, or even disabled on the server.

Slow Ops output

```
{ op : "query",
  ns : "sample_airbnb.large",
  command : {
    find : large,
    filter : {  amenities : "Snooker" },
    $db : "sample_airbnb" },
    keysExamined : 0,
    docsExamined : 277750,
    numYield : 868,
    nreturned : 0,
    queryHash : "0AEEE9D2",
    planCacheKey : "0AEEE9D2",
    locks : {
      Global : {
        acquireCount : {
          r : NumberLong(870) }},
      Database : {
        acquireCount : {
          r : NumberLong(869) }},
      Collection : {
        acquireCount : {
          r : NumberLong(869) }  },
      storage : {
        data : {
          bytesRead : Long(4744493955),
          timeReadingMicros:Long(14794471)},
          timeWaitingMicros : {
            cache :Long(3256)}},
  responseLength : 233,
  protocol : "op_msg",
  millis : 15640,
  planSummary : "COLLSCAN",
  execStats : {
    stage : "COLLSCAN",
    filter : {
      amenities : {
        $eq : "Snooker" } },
    nReturned : 0,
    executionTimeMillisEstimate : 15227,
    works : 277752,
    advanced : 0,
    needTime : 277751,
    saveState : 868,
    docsExamined : 277750},
  ts : ISODate(2020-12-09T16:41:22.563Z),
  client : 86.179.217.72,
  appName : "MongoDB Shell",
}
```


- This is output from the Log OR Profiling
- A few fields have been omitted.

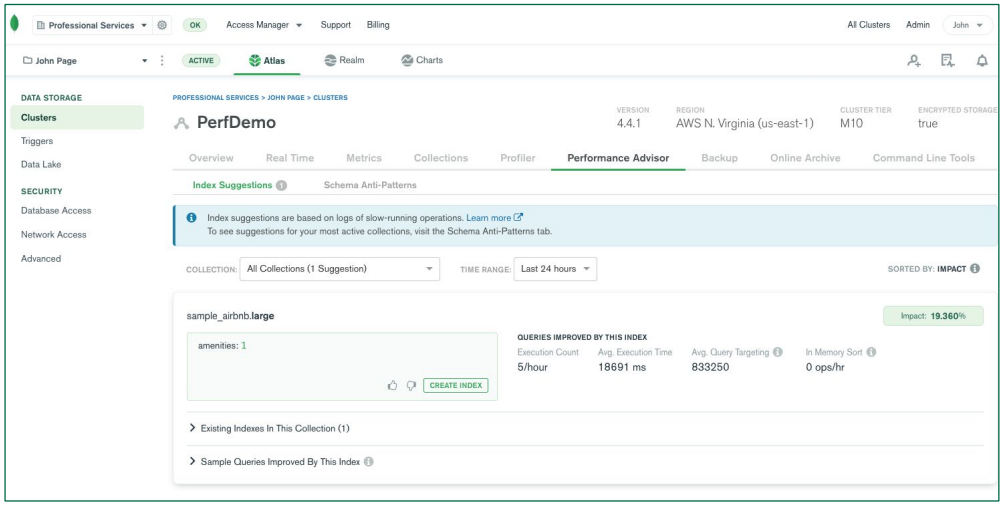
Finding slow operations in Atlas

There are various ways to analyze slow queries in Atlas or Cloud/Ops Manager:

- Performance Advisor
- Real-Time Performance Panel
- Query Profiler

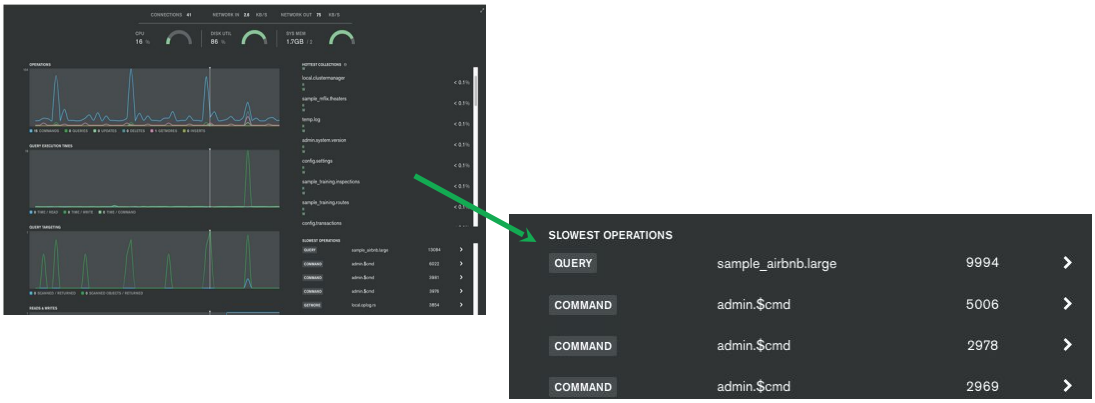
- Only available on M10+ and larger Clusters
- If required, the trainer can show the demo on M10+ Clusters.
- Details: https://docs.atlas.mongodb.com/analyze-slow-queries/

Atlas - Performance Advisor



- Click Collections.
- Click **Performance Advisor**.
- Select a collection from the Collections dropdown.
- Select a time period from the Time Range dropdown.

Atlas - RTPP / Compass

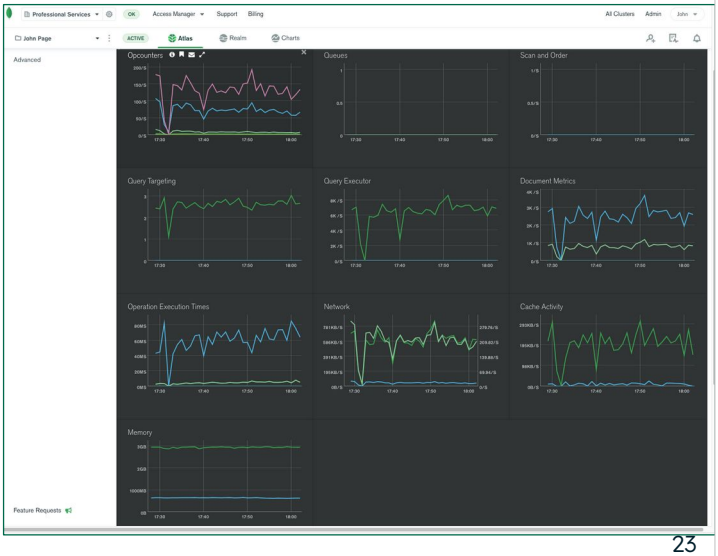


- Real-Time Performance Panel (RTPP) Only available on M10+ and larger Clusters
- If required, the trainer can show the demo on M10+ Clusters.
- Details : https://docs.atlas.mongodb.com/analyze-slow-queries/

Atlas - metrics

Atlas and Ops/Cloud Manager show metrics over time:

- Ops per second
- Queue lengths
- Scan and Order
- Targeting
- Query Executor
- Document Metrics
- Execution Times
- Network usage
- Cache activity
- Memory usage

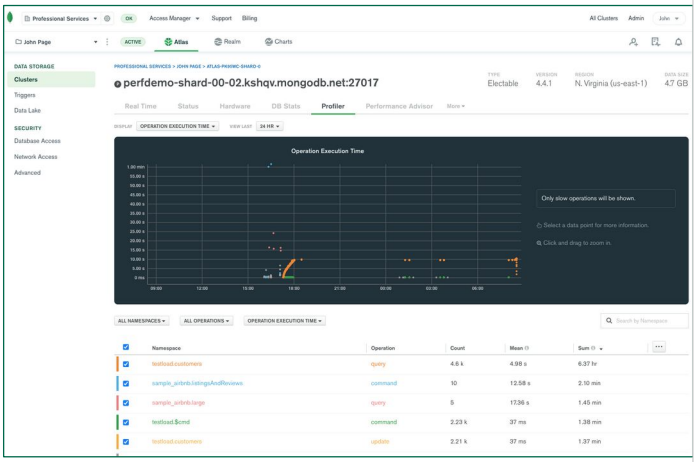


- Full metrics require an M10 or higher or Cloud Manager Standard/Premium
- The free version has more limited information
- This is a great tool for determining longer term problems.

Atlas - Query profiler

Query profiler shows slower operations graphically

Can drill down or order by different parameters



- Only available on M10+ and larger Clusters
- Details : https://docs.atlas.mongodb.com/analyze-slow-queries/

Quiz Time!

#1. The getLog command returns the last _____ log entries:

A	512	B	1024	C	4096
D	65535	E	24 hours of		

Answer in the next slide.

#1. The getLog command returns the last _____ log entries

A	512	B	1024	C	4096
D	65535	E	24 hours of		

The getLog command returns the last 1024 log entries in a rolling fashion. Other modes of retrieving logs may have more entries such as looking at the logs in the logPath of the server.

#2. What does the Data profiler do when is enabled with `setProfilingLevel(1,2)`?

A	Records all write operations	B	Puts operations taking more than 2ms in the profile database	C	Puts operations taking more than 2ms in the logfile
D	Puts all operations in the logfile	E	Causes large write load within the database cache		

Answer in the next slide.

#2. What does the Data profiler do when is enabled with `setProfilingLevel(1,2)`?

A	Records all write operations	B	Puts operations taking more than 2ms in the profile database	C	Puts operations taking more than 2ms in the logfile
D	Puts all operations in the logfile	E	Causes large write load within the database cache		

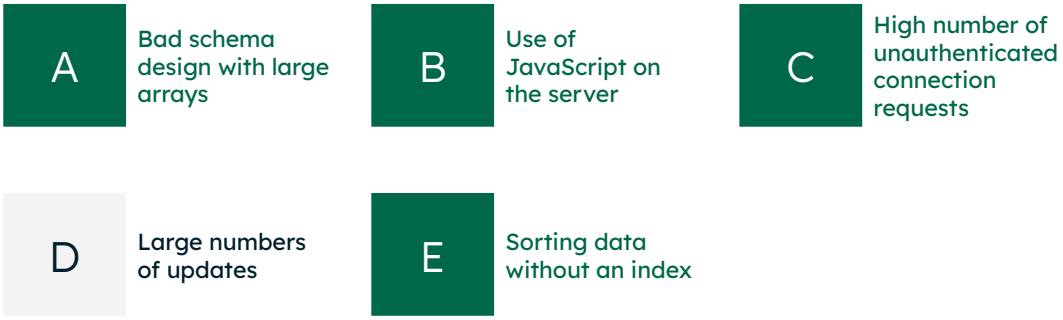
The setProfilingLevel() takes 2 arguments, the first one sets the level (0/1/2) and the second argument is what we define as slow in ms so here we are looking to record all operations taking longer than 2 ms to be recorded in the profiler, such a tiny threshold will lead to a large write load due to most operations (not all) not finishing that fast. B,C,E are valid.

#3. Which of these are likely causes of excessive CPU usage:

A	Bad schema design with large arrays
B	Use of JavaScript on the server
C	High number of unauthenticated connection requests
D	Large numbers of updates
E	Sorting data without an index

Answer in the next slide.

#3. Which of these are likely causes of excessive CPU usage:



Note: Large numbers of updates usually do not cause excessive CPU usage, but there are some edge cases, such as if the update doesn't hit any indexes and causes a collection scan. In the context of this question - if all things are normal in the DB, D would not be a valid choice whereas the others can cause issues.

Recap

Log of events hosted as a file in the Database server

getLog allows us to output the log into a collection and query it

With Atlas the log can be downloaded

Performance Advisor

Real-Time Performance Panel (RTPP)

Query Profiler