**MongoDB Audit Log: Why and How**

This blog post is another in the series on the Percona Server for MongoDB 3.4 bundle release. In this blog post, we’ll talk about the MongoDB audit log.

Percona’s development team has always invested in the open-source community a priority – especially for MongoDB. As part of this commitment, Percona continues to build MongoDB Enterprise Server features into our free, alternative, open-source [Percona Server for MongoDB](https://www.percona.com/software/mongo-database/percona-server-for-mongodb" \t "_blank). One of the key features that we have added to Percona Server for MongoDB is audit logging. Auditing your MongoDB environment strengthens your security and helps you keep track of who did what in your database.

In this blog post, we will show how to enable this functionality, what general actions can be logged, and how you can filter only the information that is important for your use-case.

Enable Audit Log

Audit messages can be logged into syslog, console or file ([JSON](https://en.wikipedia.org/wiki/JSON) or [BSON](https://en.wikipedia.org/wiki/BSON) format). In most cases, it’s preferable to log to the file in BSON format (the performance impact is smaller than JSON). In the last section, you can find some simple examples of how to further query this type of file.

Enable the audit log in the command line or the config file with:



|  |  |
| --- | --- |
| 1 | mongod --dbpath /var/lib/mongodb --auditDestination file --auditFormat BSON --auditPath /var/lib/mongodb/auditLog.bson |



|  |  |
| --- | --- |
| 1  2  3  4 | auditLog:     destination: file     format: BSON     path: /var/lib/mongodb/auditLog.bson |

Just note that until [this bug](https://jira.percona.com/browse/PSMDB-121) is fixed and released, if you’re using Percona Server for MongoDB and the --fork option while starting the mongod instance you’ll have to provide an absolute path for audit log file instead of relative path.

Actions logged

Generally speaking, the following actions can be logged:

* Authentication and authorization
* Cluster operations
* Read and write operations (logged under authCheck event and require auditAuthorizationSuccessparameter to be enabled)
* Schema operations
* Custom application messages (logged under applicationMessage event if the client/app issues a logApplicationMessage command,  the user needs to have clusterAdmin role or the one that inherits from it to issue this command)

You can see the whole list of actions logged [here](https://docs.mongodb.com/manual/reference/audit-message/#audit-event-actions-details-and-results).

By default, MongoDB doesn’t log all the read and write operations. So if you want to track those, you’ll have to enable the auditAuthorizationSuccess parameter. They then will be logged under the authCheck event. Note that this can have a serious performance impact.

Also, this parameter can be enabled dynamically on an already running instance with the audit log setup, while some other things can’t be changed once setup.

Enable logging of CRUD operations in the command line or config file:



|  |  |
| --- | --- |
| 1 | mongod --dbpath /var/lib/mongodb --setParameter auditAuthorizationSuccess=true --auditDestination file --auditFormat BSON --auditPath /var/lib/mongodb/auditLog.bson |



|  |  |
| --- | --- |
| 1  2  3  4  5 | auditLog:    destination: file    format: BSON    path: /var/lib/mongodb/auditLog.bson  setParameter: { auditAuthorizationSuccess: true } |

Or to enable it on the running instance, issue this command in the client:



|  |  |
| --- | --- |
| 1 | db.adminCommand( { setParameter: 1, auditAuthorizationSuccess: true } ) |

Filtering

If you don’t want to track all the events MongoDB is logging by default, you can specify filters in the command line or the config file. Filters need to be valid JSON queries on the audit log message (format available [here](https://docs.mongodb.com/manual/reference/audit-message/#audit-message)). In the filters, you can use standard query selectors ($eq, $in, $gt, $lt, $ne, …) as well as regex. Note that you can’t change the filters dynamically after the start.

Also, Percona Server for MongoDB 3.2 and 3.4 have slightly different message formats. 3.2 uses a “params” field, and 3.4 uses “param” just like MongoDB. When filtering on those fields, you might want to check for the difference.

Filter only events from one user:



|  |  |
| --- | --- |
| 1 | mongod --dbpath /var/lib/mongodb --auditDestination file --auditFormat BSON --auditPath /var/lib/mongodb/auditLog.bson --auditFilter '{ "users.user": "prod\_app" }' |



|  |  |
| --- | --- |
| 1  2  3  4  5 | auditLog:    destination: file    format: BSON    path: /var/lib/mongodb/auditLog.bson    filter: '{ "users.user": "prod\_app" }' |

Filter events from several users based on username prefix (using regex):



|  |  |
| --- | --- |
| 1 | mongod --dbpath /var/lib/mongodb --auditDestination file --auditFormat BSON --auditPath /var/lib/mongodb/auditLog.bson --auditFilter '{ "users.user": /^prod\_app/ }' |



|  |  |
| --- | --- |
| 1  2  3  4  5 | auditLog:    destination: file    format: BSON    path: /var/lib/mongodb/auditLog.bson    filter: '{ "users.user": /^prod\_app/ }' |

Filtering multiple event types by using standard query selectors:



|  |  |
| --- | --- |
| 1 | mongod --dbpath /var/lib/mongodb --auditDestination file --auditFormat BSON --auditPath /var/lib/mongodb/auditLog.bson --auditFilter '{ atype: { $in: [ "dropCollection", "dropDatabase" ] } }' |



|  |  |
| --- | --- |
| 1  2  3  4  5 | auditLog:    destination: file    format: BSON    path: /var/lib/mongodb/auditLog.bson    filter: '{ atype: { $in: [ "dropCollection", "dropDatabase" ] } }' |

Filter read and write operations on all the collections in the test database (notice the double escape of dot in regex):



|  |  |
| --- | --- |
| 1 | mongod --dbpath /var/lib/mongodb --auditDestination file --auditFormat BSON --auditPath /var/lib/mongodb/auditLog.bson --setParameter auditAuthorizationSuccess=true --auditFilter '{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete", "update", "findandmodify" ] }, "param.ns": /^test\\./ } }' |



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | auditLog:    destination: file    format: BSON    path: /var/lib/mongodb/auditLog.bson    filter: '{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete", "update", "findandmodify" ] }, "param.ns": /^test\\./ } }'  setParameter: { auditAuthorizationSuccess: true } |

Example messages

Here are two example messages from an audit log file. The first one is from a failed client authentication, and the second one is where the user tried to insert a document into a collection for which he has no write authorization.



|  |  |
| --- | --- |
| 1  2 | > bsondump auditLog.bson  {"atype":"authenticate","ts":{"$date":"2017-02-14T14:11:29.975+0100"},"local":{"ip":"127.0.1.1","port":27017},"remote":{"ip":"127.0.0.1","port":42634},"users":[],"roles":[],"param":{"user":"root","db":"admin","mechanism":"SCRAM-SHA-1"},"result":18} |



|  |  |
| --- | --- |
| 1  2 | > bsondump auditLog.bson  {"atype":"authCheck","ts":{"$date":"2017-02-14T14:15:49.161+0100"},"local":{"ip":"127.0.1.1","port":27017},"remote":{"ip":"127.0.0.1","port":42636},"users":[{"user":"antun","db":"admin"}],"roles":[{"role":"read","db":"admin"}],"param":{"command":"insert","ns":"test.orders","args":{"insert":"orders","documents":[{"\_id":{"$oid":"58a3030507bd5e3486b1220d"},"id":1.0,"item":"paper clips"}],"ordered":true}},"result":13} |

Querying audit log for specific event

The audit log feature is now working, and we have some data in the BSON binary file. How do I query it to find some specific event that interests me? Obviously there are many simple or more complex ways to do that using different tools (Apache Drill or Elasticsearch come to mind), but for the purpose of this blog post, we’ll show two simple ways to do that.

The first way without exporting data anywhere is using the bsondump tool to convert BSON to JSON and pipe it into the [jq](https://stedolan.github.io/jq/" \t "_blank)tool (command-line JSON processor) to query JSON data. Install the jq tool in Ubuntu/Debian with:



|  |  |
| --- | --- |
| 1 | sudo apt-get install jq |

Or in Centos with:



|  |  |
| --- | --- |
| 1  2 | sudo yum install epel-release  sudo yum install jq |

Then, if we want to know who created a database with the name “prod” for example, we can use something like this (I’m sure you’ll find better ways to use the jq tool for querying this kind of data):



|  |  |
| --- | --- |
| 1  2 | > bsondump auditLog.bson | jq -c 'select(.atype == "createDatabase") | select(.param.ns == "prod")'  {"atype":"createDatabase","ts":{"$date":"2017-02-17T12:13:48.142+0100"},"local":{"ip":"127.0.1.1","port":27017},"remote":{"ip":"127.0.0.1","port":47896},"users":[{"user":"prod\_app","db":"admin"}],"roles":[{"role":"root","db":"admin"}],"param":{"ns":"prod"},"result":0} |

In the second example, we’ll use the mongorestore tool to import data into another instance of mongod, and then just query it like a normal collection:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | > mongorestore -d auditdb -c auditcol auditLog.bson  2017-02-17T12:28:56.756+0100    checking for collection data in auditLog.bson  2017-02-17T12:28:56.797+0100    restoring auditdb.auditcol from auditLog.bson  2017-02-17T12:28:56.858+0100    no indexes to restore  2017-02-17T12:28:56.858+0100    finished restoring auditdb.auditcol (142 documents)  2017-02-17T12:28:56.858+0100    done |

The import is done, and now we can query the collection for the same data from the MongoDB client:



|  |  |
| --- | --- |
| 1  2  3  4  5 | > use auditdb  switched to db auditdb    > db.auditcol.find({atype: "createDatabase", param: {ns: "prod"}})  { "\_id" : ObjectId("58a6de78bdf080b8e8982a4f"), "atype" : "createDatabase", "ts" : { "$date" : "2017-02-17T12:13:48.142+0100" }, "local" : { "ip" : "127.0.1.1", "port" : 27017 }, "remote" : { "ip" : "127.0.0.1", "port" : 47896 }, "users" : [ { "user" : "prod\_app", "db" : "admin" } ], "roles" : [ { "role" : "root", "db" : "admin" } ], "param" : { "ns" : "prod" }, "result" : 0 } |

It looks like the audit log in MongoDB/Percona Server for MongoDB is a solid feature. Setting up tracking for information that is valuable to you only depends on your use case.