mongoreplay

On this page

Synopsis

New in version 3.4.

`mongoreplay` is a traffic capture and replay tool for MongoDB that you can use to inspect and record commands sent to a MongoDB instance, and then replay those commands back onto another host at a later time.

`mongoreplay` can help you preview how your MongoDB deployment will perform a production workload under a different environment, such as with a different storage engine, on different hardware, or with a different operating system configuration. `mongoreplay` can also help reproduce and investigate an issue by recording and replaying the operations that trigger the issue. Finally, `mongoreplay` serves as a more flexible version of the legacy `mongosniff` tool to help you investigate database activity.

Required Access

`mongoreplay` requires access to the network interface that the `record` or `monitor` commands will listen on. You may need to run `mongoreplay` with root privileges to access the network device.

WARNING:

Only use root privileges when connecting to trusted sources.

If you are using `play` to connect to a MongoDB deployment that enforces access control, you must connect as a user with the required privileges to execute the recorded operations. Include the user's credentials in the `--host` MongoDB connection string.

Options

`mongoreplay`

`--verbosity, -v`

Increases the amount of internal reporting returned on standard output or in log files. Increase the verbosity with the `-v` form by including the option multiple times, (e.g. `-vvvvv`.)

`--debug, -d`

Increases the amount of detail about `mongoreplay` operations and errors recorded in log files. Increase the debugging detail with the `-d` form by including the option multiple times, (e.g. `-ddd`.)

`--silent, -s`

When set, `mongoreplay` does not produce any log output.

`--help`

Returns information on the options and use of `mongoreplay`.

Commands

`mongoreplay` includes the following *commands* to record, play back, and monitor MongoDB network traffic.

`record`

`record` produces a playback file based on network traffic. `record` supports collecting network traffic directly or can accept a pcap file to produce the playback file. The playback file contains a list of all requests sent to the `mongod` instance during the recording as well as all responses transmitted to the client during the recording. The playback file also records metadata for each request, such as the connection identifier and timestamp.

The following prototype uses `mongoreplay` to record data on the loopback network interface and creates a playback file located at `~/recordings/playback`.

```
mongoreplay record -i eth0 -e "port 27017" -p ~/recordings/playback
```

Similarly, the following prototype uses `mongoreplay` to produce a playback file from an existing pcap file:

```
mongoreplay record -f traffic.pcap -p ~/recordings/playback
```

`record` supports the following options:

`record`

`-f <path>`

Specifies the path to a pcap file that `record` should read to produce a playback file.

Use `-f` as an alternative to capturing network traffic using `-i`. You must specify *either* `-f` or `-i`. If you include both options, `mongoreplay record` produces an error.

`-b <number>`

Size of heap used to merge separate streams together.

`--expr <filter expression>, -e <filter expression>`

An expression in Berkeley Packet Filter (BPF) syntax to apply to incoming traffic to record. Required if you are capturing traffic from a network interface using `-i`.

For example, to capture traffic from a MongoDB instance running on port 27017, you would specify `-e 'port 27017'`.

`-i <interface>`

Specifies the network interface that `record` should listen on to capture network traffic.

Use with `-e`.

Use `-i` as an alternative to reading an existing pcap file with `-i`. You must specify *either* `-f` or `-i`. If you include both options, `mongoreplay record` produces an error.

`--gzip <boolean>`

If specified, `record` compresses the playback file with gzip.

`--playback-file <path>, -p <path>`

Specifies the path to which to write the playback file.

The produced playback file is a BSON file.

SEE:

Use record for examples of using `mongoreplay` with the `record` command.

play

`play` replays a playback file created with `record` against a `mongod` instance.

The following prototype uses `mongoreplay` to replay the `~/recordings/playback` playback file to the `mongod` instance running on `192.168.0.4:27018`:

```
mongoreplay play -p ~/recordings/playback --report ~/reports/replay_stats.json --host mongodb
```

`play` supports the following options:

`play`

`--collect <json|format|none>`

Default: format

Specifies the output format for the collected statistics.

- `json`: outputs stat information as json
- `format`: uses the formatting specified in the `--format` option to produce the output file.
- `none`: does not provide any output

`--report <path>`

Specifies the path to which to write an execution report. Use `--collect` to specify the output format for the report.

If you do not specify `--report`, `play` writes to STDOUT.

`--no-truncate`

If specified, disables truncation of large reply payload data in the `play` log output.

`--format`

Default: `%F{blue}%t%f %F{cyan}(Connection: %o:%i)%f %F{yellow}%l%f %F{red}%T %c%f %F{white}%n%f %F{green}%Q{Request:}%f%q%F{green}%R{Response:}%f%r)`

Specifies the format for terminal output. You can specify arguments immediately after the format 'verbs' by wrapping them in curly braces, as in `%Q{<arg>}`.

If you specify `--format`, also specify `format` as the value for the `--collect` option.

`--format` supports the following verbs:

- `%n`: namespace
- `%l`: latency
- `%t`: time. You may optionally specify the date layout using the Go Programming Language's time formatting. Go uses `Mon Jan 2 15:04:05 MST 2006` as its reference time. You must specify the time format using the reference time. Thus, if you wanted to print the date in format `yyyy-mm-dd hh:mm`, you would specify `%t{2006-01-02 15:04}`. Refer to the Go time formatting documentation for more information.
- `%T`: op time
- `%c`: command
- `%o`: number of connections
- `%i`: request ID
- `%q`: request. You may optinally specified a dot-delimited field within the JSON structure, as in, `%q{command_args.documents}`.
- `%r`: response. You may optinally specified a dot-delimited field within the JSON structure, as in, `%q{command_args.documents}`.
- `%Q{<arg>}`: display `<arg>` on presence of request data
- `%R{<arg>}`: display `<arg>` on presence of response data

In addition, `--format` supports the following start/end ANSI escape sequences:

- `%B`/`%b`: bold
- `%U`/`%u`: underline
- `%S`/`%s`: standout
- `%F`/`%f`: text color (required arg – word or number, 8-color)
- `%K`/`%k`: background color (required arg – same as %F/%f)

`--no-colors`

When set, removes colors from the `default format`.

`--playback-file <path>, -p <path>`

Specifies the path from which to read the playback file.

If the playback file was created using the `--gzip` option, you must also specify `--gzip` when running `play`.

`--speed number`

Default: 1.0

Specifies a multiplier to adjust playback speed. `--speed 1.0` processes the playback file in real time; `--speed 0.5` at half speed; `--speed 3.0` at triple speed.

The specified speed is a *target* speed. If `mongoreplay play` encounters a bottleneck, playback may be slower than the specified multiplier.

`--host <hostname><:port>, -h <hostname><:port>`

Default: localhost:27017

Specifies a MongoDB connection string for the MongoDB deployment to which to play back the captured network traffic.

By default, `play` attempts to connect to a `mongod` instance running on the localhost on port number `27017`.

`--repeat number`

Default: 1

Specifies the number of times to play the playback file.

`--queueTime number`

Default: 15

Specifies the maximum time, in seconds, to queue operations in advance of transmitting them.

`--no-preprocess`

When set, `play` does not preprocess the input file to pre-map data such as MongoDB cursor IDs.

`--gzip <boolean>`

If specified, `play` decompresses the playback file with gzip.

SEE:

Use play for examples of using `mongoreplay` with the `play` command.

monitor

`monitor` inspects live or pre-recorded MongoDB network traffic.

The following prototype uses `mongoreplay` to produce a JSON report based on the `playback.bson` playback file in the `~/recordings` directory:

```
mongoreplay monitor --collect json --report ~/reports/monitor-report.json -p ~/recordings/pla
```

`monitor` supports the following options:

`monitor`

`--collect <json|format|none>`

Default: format

Specifies the output format for the collected statistics.

- `json`: outputs stat information as json
- `format`: uses the formatting specified in the `--format` option to produce the output file.
- `none`: does not provide any output

`--report <path>`

Specifies the path to which to write an execution report. Use `--collect` to specify the output format for the report.

If you do not specify `--report`, `monitor` writes to STDOUT.

`--no-truncate`

If specified, disables truncation of large reply payload data in the `monitor` log output.

`--format`

Default: `%F{blue}%t%f %F{cyan}(Connection: %o:%i)%f %F{yellow}%l%f %F{red}%T %c%f %F{white}%n%f %F{green}%Q{Request:}%f%q%F{green}%R{Response:}%f%r)`

Specifies the format for terminal output. You can specify arguments immediately after the format 'verbs' by wrapping them in curly braces, as in `%Q{<arg>}`.

If you specify `--format`, also specify `format` as the value for the `--collect` option.

`--format` supports the following verbs:

- `%n`: namespace
- `%l`: latency
- `%t`: time. You may optionally specify the date layout using the Go Programming Language's time formatting. Go uses `Mon Jan 2 15:04:05 MST 2006` as its reference time. You must specify the time format using the reference time. Thus, if you wanted to print the date in format `yyyy-mm-dd hh:mm`, you would specify `%t{2006-01-02 15:04}`. Refer to the Go time formatting documentation for more information.
- `%T`: op time
- `%c`: command
- `%o`: number of connections
- `%i`: request ID
- `%q`: request. You may optinally specified a dot-delimited field within the JSON structure, as in, `%q{command_args.documents}`.
- `%r`: response. You may optinally specified a dot-delimited field within the JSON structure, as in, `%q{command_args.documents}`.
- `%Q{<arg>}`: display `<arg>` on presence of request data
- `%R{<arg>}`: display `<arg>` on presence of response data

In addition, `--format` supports the following start/end ANSI escape sequences:

- `%B`/`%b`: bold
- `%U`/`%u`: underline
- `%S`/`%s`: standout
- `%F`/`%f`: text color (required arg – word or number, 8-color)
- `%K`/`%k`: background color (required arg – same as %F/%f)

`--no-colors`

When set, removes colors from the `default format`.

`-f <path>`

Specifies the path to a pcap file that `monitor` should read to produce a playback file.

Use `-f` as an alternative to capturing network traffic using `-i`. You must specify *either* `-f` or `-i`. If you include both options, `mongoreplay monitor` produces an error.

`-b <number>`

Size of heap used to merge separate streams together.

`--expr <filter expression>, -e <filter expression>`

An expression in Berkeley Packet Filter (BPF) syntax to apply to incoming traffic to record. Required if you are capturing traffic from a network interface using `-i`.

For example, to capture traffic from a MongoDB instance running on port 27017, you would specify `-e 'port 27017'`.

`-i <interface>`

Specifies the network interface that `monitor` should listen on to capture network traffic.

Use with `-e`.

Use `-i` as an alternative to reading an existing pcap file with `-i`. You must specify *either* `-f` or `-i`. If you include both options, `mongoreplay monitor` produces an error.

`--paired`

When specified, `monitor` outputs one line for each request/reply pair record.

`--gzip <boolean>`

If specified, `monitor` decompresses the playback file with gzip.

`--playback-file <path>, -p <path>`

Specifies the path from which to read the playback file.

SEE:

Use monitor for examples of using `mongoreplay` with the `monitor` command.

`mongoreplay` Report Format

`monitor` and `play` can produce reports based on a playback file when run with the `--report` option.

Sample Record

The following is an example record from a JSON-formatted `monitor` report:

```
{
     "order": 57,
     "op": "op_command",
     "command": "aggregate",
     "ns": "example",
     "request_data": {
             "command_args": {
                     "aggregate": "restaurants",
                     "cursor": {},
                     "pipeline": [{
                             "$match": {
                                     "borough": "Queens"
                             }
                     }, {
                             "$group": {
                                     "_id": "$cuisine",
                                     "count": {
                                             "$sum": 1.0
                                     }
                             }
                     }]
             },
             "input_docs": [],
             "metadata": {}
     },
     "reply_data": null,
     "connection_num": 0,
     "seen": "2016-09-08T10:21:48.072329-04:00",
     "request_id": 53
}
```


Fields

`mongoreplay` reports can include the following fields:

`order`

A monotonically increasing key indicating the order in which the operations were recorded and played back. This can be used to reconstruct the ordering of the series of operations executed on a connection, since the order in which they appear in the report file may not match the playback order.

`op`

The type of operation represented by the request: i.e. “query”, “insert”, “command”, “getmore”.

`command`

The name of the database command performed, such as `isMaster` or `getLastError`. This field is left blank for operations that are not commands, such as queries and inserts.

`ns`

The namespace on which the request was executed.

`request_data`

The payload of the operation.

- Query operations: `request_data` contains the actual query that was issued.
- Insert operations: `request_data` contains the documents being inserted.
- Update operations: `request_data` contains the query selector and the update modifier.

`reply_data`

The payload of the reply to the request.

`nreturned`

The number of documents returned as a result of the operation.

`played_at`

The time at which the `play` command executed the operation.

`play_at`

The time at which the operation was supposed to be executed by the `play` command.

`playbacklag_us`

The difference in microseconds in time between `played_at` and `play_at`. Higher values generally indicate that the target server is not able to keep up with the rate at which requests need to be executed according to the playback file.

`connection_num`

A key that identifies the connection on which the request was executed. All requests/replies that executed on the same connection have the same value for `connection_num`.

The `connection_num` value does *not* match the connection ID logged on the server side.

`latency_us`

The time difference in microseconds between when the request was sent by the client and when a response from the server was received.

`errors`

Lists any errors returned from the server.

`msg`

Lists the error message returned from the server.

`seen`

The time at which the operation was originally captured.

`request_id`

The ID of the MongoDB operation. The `request_id` for a request operation is the same as the `response_id` for the corresponding reply.

Output Formatting with `--format`

`monitor` and `play` output to either the terminal or, when run with `--report`, to a file.

`--collect` specifies the format of the output: `--collect json` produces JSON output, while `--collect format` outputs the data based on the formatting specified by the `--format` option.

Use

Use `record`

Capture TCP data with `record`

To create a recording of traffic, use the `record` command. The following operation records traffic through port `27017` on the `eth0` network interface and writes the captured traffic to `~/recordings/recording.bson`:

```
mongoreplay record -i eth0 -e "port 27017" -p ~/recordings/recording.bson
```

The produced playback file contains everything needed to re-execute the workload on another system.

Record a Playback File from Existing pcap Data

If you do not wish to use `mongoreplay` to capture traffic, you can capture traffic using a utility such as `tcpdump` and then create a `mongoreplay` recording from the static pcap file.

> **WARNING:**
> Only use root privileges when connecting to a trusted source.

The following operation uses `tcpdump` to capture traffic through port `27017` on the `eth0` network interface and writes the captured data to a pcap file called `traffic.pcap`:

```
sudo tcpdump -i eth0 -n "port 27017" -w traffic.pcap
```

To create the `mongoreplay` playback file, you can use `record` with the `-f` option to specify the pcap file from which to create the playback file, as in the following:

```
mongoreplay record -f traffic.pcap -p ~/recordings/playback.bson
```

The produced playback file contains everything needed to re-execute the workload on another system.

Use `play`

Execute a Playback File Against a Target Host

`play` takes a playback file and executes the recorded operations against the `mongodb://example.com:27018` host. Since the `mongod` enforces authentication, the connection string specified to `--host` also includes the username, password, and authentication database so that `mongoreplay` can write to the database.

```
mongoreplay play -p ~/recordings/recording.bson --host mongodb://username:password@example.co
```

By default, `play` executes the playback file at the rate of the recording. `--speed` lets you modify the playback speed. For example, the following operation executes the operations in `recording.bson` at twice the recording speed:

```
mongoreplay play -p ~/recordings/recording.bson --speed=2.0 --host mongodb://username:passwor
```

Log Metrics About Execution Performance during Playback

`play` can produce a report with detailed metrics about the performance of each operation executed during playback.

The following example executes playback against the `mongodb://example.com:27017` host and produces a JSON report written to `~/reports/playback-report.json`

```
mongoreplay play -p ~/recordings/recording.bson --report ~/reports/playback-report.json --col
```

The `play` report contents would resemble:

```
{
     "order": 2,
     "op": "op_command",
     "command": "listCollections",
     "ns": "dbs",
     "request_data": {
             "command_args": {
                     "filter": {},
                     "listCollections": 1.0
             },
             "input_docs": [],
             "metadata": {}
     },
     "reply_data": {
             "command_reply": {
                     "cursor": {
                             "firstBatch": [],
                             "id": {
                                     "$numberLong": "0"
                             },
                             "ns": "dbs.$cmd.listCollections"
                     },
                     "ok": 1.0
             },
             "metadata": {},
             "output_docs": null
     },
     "played_at": "2016-09-12T15:50:47.576511488-04:00",
     "play_at": "2016-09-12T15:50:47.568587106-04:00",
     "playbacklag_us": 7924,
     "connection_num": 1,
     "latency_us": 327,
     "seen": "2016-08-29T15:18:46.000445Z",
     "request_id": 20
}
{
     "order": 4,
     "op": "op_command",
     "command": "aggregate",
     "ns": "example",
     "request_data": {
             "command_args": {
                     "aggregate": "restaurants",
```

```
                    "cursor": {},
                    "pipeline": [{
                            "$match": {
                                    "borough": "Manhattan"
                            }
                    }, {
                            "$group": {
                                    "_id": "$cuisine"
                            }
                    }]
            },
            "input_docs": [],
            "metadata": {}
    },
    "reply_data": {
            "command_reply": {
                    "cursor": {
                            "firstBatch": [],
                            "id": {
                                    "$numberLong": "0"
                            },
                            "ns": "example.restaurants"
                    },
                    "ok": 1.0
            },
            "metadata": {},
            "output_docs": null
    },
    "played_at": "2016-09-12T15:51:09.775716-04:00",
    "play_at": "2016-09-12T15:51:09.775244206-04:00",
    "playbacklag_us": 471,
    "connection_num": 1,
    "latency_us": 407,
    "seen": "2016-08-29T15:22:28.067016Z",
    "request_id": 21
}
```


Refer to mongoreplay Report Format for a description of each field.

Use `monitor`

Inspect Recorded Operations

`monitor` can create a report based on the contents of a playback file. `monitor`'s report includes *all* operations and some metadata about each operation's execution.

The following operation uses `monitor` to create a JSON report based on the contents of the `recording.bson` playback file located in the `~/recordings` directory and write the report to `~/reports/monitoring-report.json`:

```
mongoreplay monitor -p ~/recordings/recording.bson --report ~/reports/monitoring-report.json
```

The report contents would resemble:

```
{
     "order": 0,
     "op": "command",
     "command": "isMaster",
     "ns": "admin.$cmd",
     "request_data": {
             "isMaster": 1
     },
     "reply_data": null,
     "connection_num": 0,
     "seen": "2016-08-26T14:45:59.387837Z",
     "request_id": 14
}
{
     "order": 1,
     "op": "reply",
     "request_data": null,
     "reply_data": {
             "ismaster": true,
             "localTime": {
                     "$date": "2016-08-26T14:45:59.388Z"
             },
             "maxBsonObjectSize": 16777216,
             "maxMessageSizeBytes": 48000000,
             "maxWireVersion": 4,
             "maxWriteBatchSize": 1000,
             "minWireVersion": 0,
             "ok": 1.0
     },
     "nreturned": 1,
     "connection_num": 0,
     "latency_us": 214,
     "seen": "2016-08-26T14:45:59.388051Z",
     "request_id": 14
}
{
     "order": 2,
     "op": "op_command",
     "command": "listCollections",
     "ns": "folkore",
     "request_data": {
             "command_args": {
                     "filter": {},
```

```
                    "listCollections": 1.0
            },
            "input_docs": [],
            "metadata": {}
    },
    "reply_data": null,
    "connection_num": 0,
    "seen": "2016-08-26T14:45:59.388186Z",
    "request_id": 15
}
```

Refer to mongoreplay Report Format for a description of each field.

Inspect Live MongoDB Traffic

`monitor` can also inspect live traffic and, optionally, create a report based on the observed operations.

To monitor traffic in real time in your terminal, omit the `--report` option, as in the following:

```
mongoreplay monitor -i eth0 -e 'port 27017' --collect json
```

Optionally, you can create a report based on the operations observed using `monitor`. The following example creates a JSON report written to `~/reports/monitor-live.json` based on the traffic through port `27017` on the `eth0` network interface:

```
mongoreplay monitor -i eth0 -e 'port 27017' --report ~/reports/monitor-live.json --collect js
```