HAProxy log analyzer Documentation

Release 0.1

Gil Forcada

Jul 26, 2017

Contents

Contents:

CHAPTER 1

HAProxy log analyzer

This Python package is a HAProxy log parser. It analyzes HAProxy log files in multiple ways (see commands section below).

Note: Currently only the HTTP log format is supported.

Tests and coverage

No project is trustworthy if does not have tests and a decent coverage!

Documentation

See the documentation and API at ReadTheDocs.

Command-line interface

The current `--help` looks like this:

```
usage: haproxy_log_analysis [-h] [-l LOG] [-s START] [-d DELTA] [-c COMMAND]
                            [-f FILTER] [-n] [--list-commands]
                            [--list-filters]

Analyze HAProxy log files and outputs statistics about it

optional arguments:
  -h, --help            show this help message and exit
  -l LOG, --log LOG     HAProxy log file to analyze
```

```
  -s START, --start START
                        Process log entries starting at this time, in HAProxy
                        date format (e.g. 11/Dec/2013 or
                        11/Dec/2013:19:31:41). At least provide the
                        day/month/year. Values not specified will use their
                        base value (e.g. 00 for hour). Use in conjunction with
                        -d to limit the number of entries to process.
  -d DELTA, --delta DELTA
                        Limit the number of entries to process. Express the
                        time delta as a number and a time unit, e.g.: 1s, 10m,
                        3h or 4d (for 1 second, 10 minutes, 3 hours or 4
                        days). Use in conjunction with -s to only analyze
                        certain time delta. If no start time is given, the
                        time on the first line will be used instead.
  -c COMMAND, --command COMMAND
                        List of commands, comma separated, to run on the log
                        file. See --list-commands to get a full list of them.
  -f FILTER, --filter FILTER
                        List of filters to apply on the log file. Passed as
                        comma separated and parameters within square brackets,
                        e.g ip[192.168.1.1],ssl,path[/some/path]. See --list-
                        filters to get a full list of them.
  -n, --negate-filter   Make filters passed with -f work the other way around,
                          i.e. ifthe ``ssl`` filter is passed instead of showing
                        only ssl requests it will show non-ssl traffic. If the
                        ``ip`` filter isused, then all but that ip passed to
                        the filter will be used.
  --list-commands       Lists all commands available.
  --list-filters        Lists all filters available.
```

Commands

Commands are small purpose specific programs in themselves that report specific statistics about the log file being analyzed. See the `--help` (or the section above) to know how to run them.

`counter` Reports how many log lines could be parsed.

`counter_invalid` Reports how many log lines could *not* be parsed.

`http_methods` Reports a breakdown of how many requests have been made per HTTP method (GET, POST...).

`ip_counter` Reports a breakdown of how many requests have been made per IP. Note that for this to work you need to configure HAProxy to capture the header that has the IP on it (usually the X-Forwarded-For header). Something like: `capture request header X-Forwarded-For len 20`

`top_ips` Reports the 10 IPs with most requests (and the amount of requests).

`status_codes_counter` Reports a breakdown of how many requests per HTTP status code (404, 500, 200, 301..) are on the log file.

`request_path_counter` Reports a breakdown of how many requests per path (/rss, /, /another/path).

`top_request_paths` Reports the 10 paths with most requests.

`slow_requests` Reports a list of requests that downstream servers took more than 1 second to response.

`counter_slow_requests` Reports the amount of requests that downstream servers took more than 1 second to response.

average_response_time Reports the average time (in milliseconds) servers spend to answer requests. .. note:: Aborted requests are not considered.

average_waiting_time Reports the average time (in milliseconds) requests spend waiting on the various HAProxy queues.

server_load Reports a breakdown of how many requests were processed by each downstream server. Note that currently it does not take into account the backend the server is configured on.

queue_peaks Reports a list of queue peaks. A queue peak is defined by the biggest value on the backend queue on a series of log lines that are between log lines without being queued.

connection_type Reports on how many requests were made on SSL and how many on plain HTTP. This command only works if the default port for SSL (443) appears on the path.

requests_per_minute Reports on how many requests were made per minute. It works best when used with `-s` and `-d` command line arguments, as the output can be huge.

print Prints the raw lines. This can be useful to trim down a file (with `-s` and `-d` for example) so that later runs are faster.

Filters

Filters, contrary to commands, are a way to reduce the amount of log lines to be processed.

Note: The `-n` command line argument allows to reverse filters output.

This helps when looking for specific traces, like a certain IP, a path...

---

ip Filters log lines by the given IP.

ip_range Filters log lines by the given IP range (all IPs that begin with the same prefix).

path Filters log lines by the given string.

ssl Filters log lines that are from SSL connections. See :method::*.HaproxyLogLine.is_https* for its limitations.

slow_requests Filters log lines that take at least the given time to get answered (in milliseconds).

time_frame This is an implicit filter that is used when `--start`, and optionally, `--delta` are used. Do not use this filter on the command line, use `--start` and `--delta` instead.

status_code Filters log lines that match the given HTTP status code (i.e. 404, 200...).

status_code_family Filters log lines that match the given HTTP status code family (i.e. 4 for all 4xx status codes, 5 for 5xx status codes...).

http_method Filters log lines by the HTTP method used (GET, POST...).

backend Filters log lines by the HAProxy backend the connection was handled with.

frontend Filters log lines by the HAProxy frontend the connection arrived from.

server Filters log lines by the downstream server that handled the connection.

response_size Filters log lines by the response size (in bytes). Specially useful when looking for big file downloads.

wait_on_queues Filters log lines by the amount of time the request had to wait on HAProxy queues. If a request waited less than the given amount of time is accepted.

Installation

After installation you will have a console script *haproxy_log_analysis*:

```
$ python setup.py install
```

TODO

- add more commands: *(help appreciated)*
  - reports on servers connection time
  - reports on termination state
  - reports around connections (active, frontend, backend, server)
  - *your ideas here*
- think of a way to show the commands output in a meaningful way
- be able to specify an output format. For any command that makes sense (slow requests for example) output the given fields for each log line (i.e. acceptance date, path, downstream server, load at that time...)
- *your ideas*

CHAPTER 2

Haproxy Modules

Log

class haproxy.logfile.**Log**(*logfile=None*)

_is_pickle_valid()
Logic to decide if the file should be processed or just needs to be loaded from its pickle data.

_load()
Load data from a pickle file.

_save()
Save the attributes defined on _pickle_attributes in a pickle file.

This improves a lot the nth run as the log file does not need to be processed every time.

static _sort_and_trim(*data*, *reverse=False*)
Sorts a dictionary with at least two fields on each of them sorting by the second element.

> **Warning:** Right now is hardcoded to 10 elements, improve the command line interface to allow to send parameters to each command or globally.

_sort_lines()
Haproxy writes its logs after having gathered all information related to each specific connection. A simple request can be really quick but others can be really slow, thus even if one connection is logged later, it could have been accepted before others that are already processed and logged.

This method sorts all valid log lines by their acceptance date, providing the real order in which connections where made to the server.

cmd_average_response_time()
Returns the average response time of all, non aborted, requests.

cmd_average_waiting_time()
Returns the average queue time of all, non aborted, requests.

cmd_connection_type()
Generates statistics on how many requests are made via HTTP and how many are made via SSL.

Note: This only works if the request path contains the default port for SSL (443).

Warning: The ports are hardcoded, they should be configurable.

cmd_counter()
Returns the number of valid lines.

cmd_counter_invalid()
Returns the number of invalid lines.

cmd_counter_slow_requests()
Counts all requests that took a certain amount of time to be processed.

Warning: By now hardcoded to 1 second (1000 milliseconds), improve the command line interface to allow to send parameters to each command or globally.

cmd_http_methods()
Reports a breakdown of how many requests have been made per HTTP method (GET, POST...).

cmd_ip_counter()
Reports a breakdown of how many requests have been made per IP.

Note: To enable this command requests need to provide a header with the forwarded IP (usually X-Forwarded-For) and be it the only header being captured.

cmd_print()
Returns the raw lines to be printed.

cmd_queue_peaks()
Generate a list of the requests peaks on the queue.

A queue peak is defined by the biggest value on the backend queue on a series of log lines that are between log lines without being queued.

Warning: Allow to configure up to which peak can be ignored. Currently set to 1.

cmd_request_path_counter()
Generate statistics about HTTP requests' path.

cmd_requests_per_minute()
Generates statistics on how many requests were made per minute.

Note: Try to combine it with time constrains (`-s` and `-d`) as this command output can be huge otherwise.

cmd_server_load()
Generate statistics regarding how many requests were processed by each downstream server.

cmd_slow_requests()
List all requests that took a certain amount of time to be processed.

> **Warning:** By now hardcoded to 1 second (1000 milliseconds), improve the command line interface to allow to send parameters to each command or globally.

cmd_status_codes_counter()
Generate statistics about HTTP status codes. 404, 500 and so on.

cmd_top_ips()
Returns the top most frequent IPs.

> **Note:** See *Log._sort_and_trim()* for its current limitations.

cmd_top_request_paths()
Returns the top most frequent paths.

> **Note:** See *Log._sort_and_trim()* for its current limitations.

classmethod commands()
Returns a list of all methods that start with `cmd_`.

filter(*filter_func*, *reverse=False*)
Filter current log lines by a given filter function.

This allows to drill down data out of the log file by filtering the relevant log lines to analyze.

For example, filter by a given IP so only log lines for that IP are further processed with commands (top paths, http status counter...).

Parameters

- **filter_func** (*function*) – [required] Filter method, see filters.py for all available filters.
- **reverse** (*boolean*) – negate the filter (so accept all log lines that return `False`).

Returns a new instance of Log containing only log lines that passed the filter function.

Return type *Log*

parse_data(*logfile*)
Parse data from data stream and replace object lines.

Parameters **logfile** (*str*) – [required] Log file data stream.

Line

class haproxy.line.**Line**(*line*)
For a precise and more detailed description of every field see: http://cbonte.github.io/haproxy-dconv/configuration-1.4.html#8.2.3

accept_date = None
datetime object with the exact date when the connection to HAProxy was made.

backend_name = None
HAProxy backend that the connection was sent to.

bytes_read = None
Total number of bytes send back to the client.

client_ip = None
IP of the upstream server that made the connection to HAProxy.

client_port = None
Port used by the upstream server that made the connection to HAProxy.

connections_active = None
Total number of concurrent connections on the process when the session was logged (`actconn` in HAProxy documentation).

connections_backend = None
Total number of concurrent connections handled by the backend when the session was logged (`beconn` in HAProxy documentation).

connections_frontend = None
Total number of concurrent connections on the frontend when the session was logged (`feconn` in HAProxy documentation).

connections_server = None
Total number of concurrent connections still active on the server when the session was logged (`srv_conn` in HAProxy documentation).

frontend_name = None
HAProxy frontend that received the connection.

get_ip()
Returns the IP provided on the log line.

http_request_method = None
HTTP method (GET, POST...) used on this request.

http_request_path = None
Requested HTTP path.

http_request_protocol = None
HTTP version used on this request.

is_https()
Returns True if the log line is a SSL connection. False otherwise.

queue_backend = None
Total number of requests which were processed before this one in the backend's global queue (`backend_queue` in HAProxy documentation).

queue_server = None
Total number of requests which were processed before this one in the server queue (`srv_queue` in HAProxy documentation).

retries = None
Number of connection retries experienced by this session when

server_name = None
Downstream server that HAProxy send the connection to.

status_code = None
HTTP status code returned to the client.

time_connect_server = None
Time in milliseconds to connect to the final server (`Tc` in HAProxy documentation).

time_wait_queues = None
Time in milliseconds that the request spend on HAProxy queues (`Tw` in HAProxy documentation).

time_wait_request = None
Time in milliseconds waiting the client to send the full HTTP request (`Tq` in HAProxy documentation).

time_wait_response = None
Time in milliseconds waiting the downstream server to send the full HTTP response (`Tr` in HAProxy documentation).

total_time = None
Total time in milliseconds between accepting the HTTP request and sending back the HTTP response (`Tt` in HAProxy documentation).

CHAPTER 3

Filters

haproxy.filters.**filter_backend**(*backend_name*)
Filter *Line* objects by the HAProxy backend name they were processed with.

Parameters **backend_name** (*string*) – Name of the HAProxy backend section to investigate.

Returns a function that filters by the given backend name.

Return type function

haproxy.filters.**filter_frontend**(*frontend_name*)
Filter *Line* objects by the HAProxy frontend name the connection arrived from.

Parameters **frontend_name** (*string*) – Name of the HAProxy frontend section to investigate.

Returns a function that filters by the given frontend name.

Return type function

haproxy.filters.**filter_http_method**(*http_method*)
Filter *Line* objects by their HTTP method used (i.e. GET, POST...).

Parameters **http_method** (*string*) – HTTP method (POST, GET...).

Returns a function that filters by the given HTTP method.

Return type function

haproxy.filters.**filter_ip**(*ip*)
Filter *Line* objects by IP.

Parameters **ip** (*string*) – IP that you want to filter to.

Returns a function that filters by the provided IP.

Return type function

haproxy.filters.**filter_ip_range**(*ip_range*)
Filter *Line* objects by IP range.

Both *192.168.1.203* and *192.168.1.10* are valid if the provided ip range is `192.168.1` whereas *192.168.2.103* is not valid (note the *.2.*).

Parameters **ip_range** (*string*) – IP range that you want to filter to.

Returns a function that filters by the provided IP range.

Return type function

haproxy.filters.**filter_path**(*path*)
Filter *Line* objects by their request path.

Parameters **path** (*string*) – part of a path that needs to be on the request path.

Returns a function that filters by the provided path.

Return type function

haproxy.filters.**filter_response_size**(*size*)
Filter *Line* objects by the response size (in bytes).

Specially useful when looking for big file downloads.

Parameters **size** (*string*) – Minimum amount of bytes a response body weighted.

Returns a function that filters by the response size.

Return type function

haproxy.filters.**filter_server**(*server_name*)
Filter *Line* objects by the downstream server that handled the connection.

Parameters **server_name** (*string*) – Name of the server HAProxy send the connection to.

Returns a function that filters by the given server name.

Return type function

haproxy.filters.**filter_slow_requests**(*slowness*)
Filter *Line* objects by their response time.

Parameters **slowness** (*string*) – minimum time, in milliseconds, a server needs to answer a request. If the server takes more time than that the log line is accepted.

Returns a function that filters by the server response time.

Return type function

haproxy.filters.**filter_ssl**(*ignore=True*)
Filter *Line* objects that from SSL connections.

Parameters **ignore** (*bool*) – parameter to be ignored just to conform to the rule that all filters need a parameter

Returns a function that filters SSL log lines.

Return type function

haproxy.filters.**filter_status_code**(*http_status*)
Filter *Line* objects by their HTTP status code.

Parameters **http_status** (*string*) – HTTP status code (200, 404, 502...) to filter lines with.

Returns a function that filters by HTTP status code.

Return type function

haproxy.filters.**filter_status_code_family**(*family_number*)
Filter *Line* objects by their family of HTTP status code, i.e. 2xx, 3xx, 4xx

Parameters **family_number** (*string*) – First digit of the HTTP status code family, i.e. 2 to get all the 2xx status codes, 4 for the client errors and so on.

Returns a function that filters by HTTP status code family.

Return type function

haproxy.filters.**filter_time_frame**(*start*, *delta*)
Filter *Line* objects by their connection time.

Parameters

- **start** (*string*) – a time expression (see -s argument on –help for its format) to filter log lines that are before this time.
- **delta** (*string*) – a relative time expression (see -s argument on –help for its format) to limit the amount of time log lines will be considered.

Returns a function that filters by the time a request is made.

Return type function

haproxy.filters.**filter_wait_on_queues**(*max_waiting*)
Filter *Line* objects by their queueing time in HAProxy.

Parameters **max_waiting** (*string*) – maximum time, in milliseconds, a request is waiting on HAProxy prior to be delivered to a backend server. If HAProxy takes less than that time the log line is counted.

Returns a function that filters by HAProxy queueing time.

Return type function

CHAPTER 4

CHANGES

2.1 (2017-07-06)

- Enforce QA checks (flake8) on code. All code has been updated to follow it. [gforcada]
- Support Python 3.6. [gforcada]
- Support different syslog timestamps (at least NixOS). [gforcada]

2.0.2 (2016-11-17)

- Improve performance for `cmd_print`. [kevinjqiu]

2.0.1 (2016-10-29)

- Allow hostnames to have a dot in it. [gforcada]

2.0 (2016-07-06)

- Handle unparseable HTTP requests. [gforcada]
- Only test on python 2.7 and 3.5 [gforcada]

2.0b0 (2016-04-18)

- Check the divisor before doing a divison to not get `ZeroDivisionError` exceptions. [gforcada]

2.0a0 (2016-03-29)

- Major refactoring:

  # Rename modules and classes:

  - haproxy_logline -> line
  - haproxy_logfile -> logfile
  - HaproxyLogLine -> Line
  - HaproxyLogFile -> Log

  # Parse the log file on Log() creation (i.e. in its __init__)

  [gforcada]

1.3 (2016-03-29)

- New filter: `filter_wait_on_queues`. Get all requests that waited at maximum X amount of milliseconds on HAProxy queues. [gforcada]
- Code/docs cleanups and add code analysis. [gforcada]
- Avoid using eval. [gforcada]

1.2.1 (2016-02-23)

- Support -1 as a status_code [Christopher Baines]

1.2 (2015-12-07)

- Allow a hostname on the syslog part (not only IPs) [danny crasto]

1.1 (2015-04-19)

- Make syslog optional. Fixes issue https://github.com/gforcada/haproxy_log_analysis/issues/10. [gforcada]

1.0 (2015-03-24)

- Fix issue #9. log line on the syslog part was too strict, it was expecting the hostname to be a string and was failing if it was an IP. [gforcada]

0.0.3.post2 (2015-01-05)

- Finally really fixed issue #7. `namespace_packages` was not meant to be on setup.py at all. Silly copy&paste mistake. [gforcada]

0.0.3.post (2015-01-04)

- Fix release on PyPI. Solves GitHub issue #7. https://github.com/gforcada/haproxy_log_analysis/issues/7 [gforcada]

0.0.3 (2014-07-09)

- Fix release on PyPI (again). [gforcada]

0.0.2 (2014-07-09)

- Fix release on PyPI. [gforcada]

0.0.1 (2014-07-09)

- Pickle :class::.*HaproxyLogFile* data for faster performance. [gforcada]
- Add a way to negate the filters, so that instead of being able to filter by IP, it can output all but that IP information. [gforcada]
- Add lots of filters: ip, path, ssl, backend, frontend, server, status_code and so on. See `--list-filters` for a complete list of them. [gforcada]
- Add :method::.*HaproxyLogFile.parse_data* method to get data from data stream. It allows you use it as a library. [bogdangi]
- Add `--list-filters` argument on the command line interface. [gforcada]
- Add `--filter` argument on the command line interface, inspired by Bogdan's early design. [bogdangi] [gforcada]
- Create a new module :module::*haproxy.filters* that holds all available filters. [gforcada]
- Improve :method::.*HaproxyLogFile.cmd_queue_peaks* output to not only show peaks but also when requests started to queue and when they finsihed and the amount of requests that had been queued. [gforcada]
- Show help when no argument is given. [gforcada]
- Polish documentation and docstrings here and there. [gforcada]
- Add a `--list-commands` argument on the command line interface. [gforcada]
- Generate an API doc for `HaproxyLogLine` and `HaproxyLogFile`. [bogdangi]
- Create a `console_script` *haproxy_log_analysis* for ease of use. [bogdangi]
- Add Sphinx documentation system, still empty. [gforcada]
- Keep valid log lines sorted so that the exact order of connections is kept. [gforcada]
- Add quite a few commands, see README.rst for a complete list of them. [gforcada]
- Run commands passed as arguments (with -c flag). [gforcada]
- Add a requirements.txt file to keep track of dependencies and pin them. [gforcada]
- Add travis and coveralls support. See its badges on README.rst. [gforcada]

- Add argument parsing and custom validation logic for all arguments. [gforcada]
- Add regular expressions for haproxy log lines (HTTP format) and to parse HTTP requests path. Added tests to ensure they work as expected. [gforcada]
- Create distribution. [gforcada]

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

A

B

C

F

G

H

I

L

P

Q

R

S

T