pgBadger - A fast PostgreSQL Log Analyzer

pgBadger - A fast PostgreSQL Log Analyzer

NAME

pgBadger - a fast PostgreSQL log analysis report

SYNOPSIS

Usage: pgbadger [options] logfile [...]

Arguments:

Options:

pgBadger is able to parse a remote log file using a passwordless ssh connection. Use -r or –remote-host to set the host IP address or hostname. There are also some additional options to fully control the ssh connection.

Log file to parse can also be specified using an URI, supported protocols are http[s] and [s]ftp. The curl command will be used to download the file, and the file will be parsed during download. The ssh protocol is also supported and will use the ssh command like with the remote host use. See examples bellow.

Return codes:

Examples:

Use URI notation for remote log file:

You can use together a local PostgreSQL log and a remote pgbouncer log file to parse:

Reporting errors every week by cron job:

Generate report every week using incremental behavior:

This supposes that your log file and HTML report are also rotated every week.

Or better, use the auto-generated incremental reports:

will generate a report per day and per week.

In incremental mode, you can also specify the number of weeks to keep in the reports:

If you have a pg_dump at 23:00 and 13:00 each day during half an hour, you can use pgBadger as follow to exclude these periods from the report:

This will help avoid having COPY statements, as generated by pg_dump, on top of the list of slowest queries. You can also use –exclude-appname "pg_dump" to solve this problem in a simpler way.

You can also parse journalctl output just as if it was a log file:

or worst, call it from a remote host:

you don't need to specify any log file at command line, but if you have other PostgreSQL log files to parse, you can add them as usual.

To rebuild all incremental html reports after, proceed as follow:

it will also update all resource files (JS and CSS). Use -E or –explode if the reports were built using this option.

pgBadger also supports Heroku PostgreSQL logs using logplex format:

this will stream Heroku PostgreSQL log to pgbadger through stdin.

pgBadger can auto detect RDS and cloudwatch PostgreSQL logs using rds format:

Each CloudSQL Postgresql log is a fairly normal PostgreSQL log, but encapsulated in JSON format. It is autodetected by pgBadger but in case you need to force the log format use `jsonlog`:

This is the same as with the jsonlog extension, the json format is different but pgBadger can parse both formats.

To create a cumulative report over a month use command:

this will add a link to the month name into the calendar view in incremental reports to look at report for month 2019 May. Use -E or –explode if the reports were built using this option.

DESCRIPTION

pgBadger is a PostgreSQL log analyzer built for speed providing fully detailed reports based on your PostgreSQL log files. It's a small standalone Perl script that outperforms any other PostgreSQL log analyzer.

It is written in pure Perl and uses a JavaScript library (flotr2) to draw graphs so that you don't need to install any additional Perl modules or other packages. Furthermore, this library gives us more features such as zooming. pgBadger also uses the Bootstrap JavaScript library and the FontAwesome webfont for better design. Everything is embedded.

pgBadger is able to autodetect your log file format (syslog, stderr, csvlog or jsonlog) if the file is long enough. It is designed to parse huge log files as well as compressed files. Supported compressed formats are gzip, bzip2, lz4, xz, zip

and zstd. For the xz format you must have an xz version higher than 5.05 that supports the –robot option. lz4 files must be compressed with the –content-size option for pgbadger to determine the uncompressed file size. For the complete list of features, see below.

All charts are zoomable and can be saved as PNG images.

You can also limit pgBadger to only report errors or remove any part of the report using command-line options.

pgBadger supports any custom format set in the log_line_prefix directive of your postgresql.conf file as long as it at least specifies the %t and %p patterns.

pgBadger allows parallel processing of a single log file or multiple files through the use of the -j option specifying the number of CPUs.

If you want to save system performance you can also use log_duration instead of log_min_duration_statement to have reports on duration and number of queries only.

FEATURE

pgBadger reports everything about your SQL queries:

The following reports are also available with hourly charts divided into periods of five minutes:

There are also some pie charts about distribution of:

All charts are zoomable and can be saved as PNG images. SQL queries reported are highlighted and beautified automatically.

pgBadger is also able to parse PgBouncer log files and to create the following reports:

You can also have incremental reports with one report per day and a cumulative report per week. Two multiprocess modes are available to speed up log parsing, one using one core per log file, and the second using multiple cores to parse a single file. These modes can be combined.

Histogram granularity can be adjusted using the -A command-line option. By default, they will report the mean of each top queries/errors occurring per hour, but you can specify the granularity down to the minute.

pgBadger can also be used in a central place to parse remote log files using a passwordless SSH connection. This mode can be used with compressed files and in the multiprocess per file mode (-J), but cannot be used with the CSV log format.

Examples of reports can be found here: https://pgbadger.darold.net/#reports

REQUIREMENT

pgBadger comes as a single Perl script - you do not need anything other than a modern Perl distribution. Charts are rendered using a JavaScript library, so you don't need anything other than a web browser. Your browser will do all the work.

If you plan to parse PostgreSQL CSV log files, you might need some Perl Modules:

This module is optional, if you don't have PostgreSQL log in the CSV format, you don't need to install it.

If you want to export statistics as JSON file, you need an additional Perl module:

This module is optional, if you don't select the json output format, you don't need to install it. You can install it on a Debian-like system using:

and on RPM-like system using:

Compressed log file format is autodetected from the file extension. If pgBadger finds a gz extension, it will use the zcat utility; with a bz2 extension, it will use bzcat; with lz4, it will use lz4cat; with zst, it will use zstdcat; if the file extension is zip or xz, then the unzip or xz utility will be used.

If those utilities are not found in the PATH environment variable, then use the –zcat command-line option to change this path. For example:

By default, pgBadger will use the zcat, bzcat, lz4cat, zstdcat and unzip utilities following the file extension. If you use the default autodetection of compression format, you can mix gz, bz2, lz4, xz, zip or zstd files. Specifying a custom value of –zcat option will remove the possibility of mixed compression format.

Note that multiprocessing cannot be used with compressed files or CSV files as well as under Windows platform.

INSTALLATION

Download the tarball from GitHub and unpack the archive as follow:

This will copy the Perl script pgbadger to /usr/local/bin/pgbadger by default and the man page into /usr/local/share/man/man1/pgbadger.1. Those are the default installation directories for 'site' install.

If you want to install all under /usr/ location, use INSTALLDIRS='perl' as an argument of Makefile.PL. The script will be installed into /usr/bin/pgbadger and the manpage into /usr/share/man/man1/pgbadger.1.

For example, to install everything just like Debian does, proceed as follows:

By default, INSTALLDIRS is set to site.

POSTGRESQL CONFIGURATION

You must enable and set some configuration directives in your postgresql.conf before starting.

You must first enable SQL query logging to have something to parse:

Here every statement will be logged, on a busy server you may want to increase this value to only log queries with a longer duration. Note that if you have log_statement set to 'all', nothing will be logged through the log_min_duration_statement directive. See the next chapter for more information.

pgBadger supports any custom format set in the log_line_prefix directive of your postgresql.conf file as long as it at least specifies a time escape sequence (%t, %m or %n) and a process-related escape sequence (%p or %c).

For example, with 'stderr' log format, log_line_prefix must be at least:

Log line prefix could add user, database name, application name and client ip address as follows:

or for syslog log file format:

Log line prefix for stderr output could also be:

or for syslog output:

You need to enable other parameters in postgresql.conf to get more information from your log files:

Do not enable log_statement as its log format will not be parsed by pgBadger.

Of course your log messages should be in English with or without locale support:

pgBadger parser does not support other locales, like 'fr_FR.UTF-8' for example.

LOG STATEMENTS

Considerations about log_min_duration_statement, log_duration and log_statement configuration directives.

If you want the query statistics to include the actual query strings, you must set log_min_duration_statement to 0 or more milliseconds.

If you just want to report duration and number of queries and don't want all details about queries, set log_min_duration_statement to -1 to disable it and enable log_duration in your postgresql.conf file. If you want to add the most common query report, you can either choose to set log_min_duration_statement to a higher value or to enable log_statement.

Enabling log_min_duration_statement will add reports about slowest queries and queries that took up the most time. Take care that if you have log_statement set to 'all', nothing will be logged with log_min_duration_statement.

Warning: Do not enable both log_min_duration_statement, log_duration and log_statement all together, this will result in wrong counter values. Note that this will also increase drastically the size of your log. log_min_duration_statement should always be preferred.

PARALLEL PROCESSING

To enable parallel processing you just have to use the -j N option where N is the number of cores you want to use.

pgBadger will then proceed as follow:

With that method, at start/end of chunks pgBadger may truncate or omit a maximum of N queries per log file, which is an insignificant gap if you have millions of queries in your log file. The chance that the query that you were looking for is lost is near 0, this is why I think this gap is livable. Most of the time the query is counted twice but truncated.

When you have many small log files and many CPUs, it is speedier to dedicate one core to one log file at a time. To enable this behavior, you have to use option -J N instead. With 200 log files of 10MB each, the use of the -J option starts being really interesting with 8 cores. Using this method you will be sure not to lose any queries in the reports.

Here is a benchmark done on a server with 8 CPUs and a single file of 9.5GB.

With 200 log files of 10MB each, so 2GB in total, the results are slightly different:

So it is recommended to use -j unless you have hundreds of small log files and can use at least 8 CPUs.

IMPORTANT: when you are using parallel parsing, pgBadger will generate a lot of temporary files in the /tmp directory and will remove them at the end, so do not remove those files unless pgBadger is not running. They are all named with the following template tmp_pgbadgerXXXX.bin so they can be easily identified.

INCREMENTAL REPORTS

pgBadger includes an automatic incremental report mode using option -I or –incremental. When running in this mode, pgBadger will generate one report per day and a cumulative report per week. Output is first done in binary format into the mandatory output directory (see option -O or –outdir), then in HTML format for daily and weekly reports with a main index file.

The main index file will show a dropdown menu per week with a link to each week report and links to daily reports of each week.

For example, if you run pgBadger as follows based on a daily rotated file:

you will have all daily and weekly reports for the full running period.

In this mode, pgBadger will create an automatic incremental file in the output directory, so you don't have to use the -l option unless you want to change the path of that file. This means that you can run pgBadger in this mode each day on a log file rotated each week, and it will not count the log entries twice.

To save disk space, you may want to use the -X or –extra-files command-line option to force pgBadger to write JavaScript and CSS to separate files in the

output directory. The resources will then be loaded using script and link tags.

Rebuilding reports

Incremental reports can be rebuilt after a pgbadger report fix or a new feature to update all HTML reports. To rebuild all reports where a binary file is still present, proceed as follow:

it will also update all resource files (JS and CSS). Use -E or –explode if the reports were built using this option.

Monthly reports

By default, pgBadger in incremental mode only computes daily and weekly reports. If you want monthly cumulative reports, you will have to use a separate command to specify the report to build. For example, to build a report for August 2019:

this will add a link to the month name into the calendar view of incremental reports to look at monthly report. The report for a current month can be run every day, it is entirely rebuilt each time. The monthly report is not built by default because it could take a lot of time following the amount of data.

If reports were built with the per-database option (-E | –explode), it must be used too when calling pgbadger to build monthly report:

This is the same when using the rebuild option (-R | –rebuild).

BINARY FORMAT

Using the binary format it is possible to create custom incremental and cumulative reports. For example, if you want to refresh a pgBadger report each hour from a daily PostgreSQL log file, you can proceed by running the following commands each hour:

to generate the incremental data files in binary format. And to generate the fresh HTML report from that binary file:

Or as another example, if you generate one log file per hour and you want reports to be rebuilt each time the log file is rotated, proceed as follows:

When you want to refresh the HTML report, for example, each time after a new binary file is generated, just do the following:

Adjust the commands to suit your particular needs.

JSON FORMAT

JSON format is good for sharing data with other languages, which makes it easy to integrate pgBadger result into other monitoring tools, like Cacti or Graphite.

AUTHORS

pgBadger is an original work from Gilles Darold.

The pgBadger logo is an original creation of Damien Cazeils.

The pgBadger v4.x design comes from the "Art is code" company.

This web site is a work of Gilles Darold.

pgBadger is maintained by Gilles Darold and everyone who wants to contribute.

Many people have contributed to pgBadger, they are all quoted in the Changelog file.

LICENSE