Postgres Query Analyser Documentation

Release 0.9

Rick van Hattem

Nov 13, 2017

Contents

Contents:

CHAPTER 1

pg_query_analyser – PostgreSQL Slow Query Log parser

1.1 Overview

pg_query_analyser is a C++ clone of the PgFouine log analyser.

Processing logs with millions of lines only takes a few minutes with this parser while PgFouine chokes long before that.

1.2 Example output

The normal overview:

The overview with the examples expanded:

1.3 Requirements

Ubuntu (tested with 13.10):

```
apt-get install qt4-dev-tools
```

Ubuntu (tested with 16.04.2 LTS)

```
apt-get install qt4-dev-tools qt4-qmake libqt4-dev
```

1.4 Install

```
qmake
make
make test
sudo make install
```

1.5 Usage

Set Postgres to use this *log_line_prefix*:

```
log_line_prefix = '%t [%p]: [%l-1] host=%h,user=%u,db=%d,tx=%x,vtx=%v '
```

After that we can start parsing data.

From stdin:

```
cat /var/log/postgresql/postgresql.log | head -n 100000 | ./pg_query_analyser -i -
```

From file:

```
./pg_query_analyser --input-file=/var/log/postgresql/postgresql.log
```

For a full build+analyze on Ubuntu:

```
cd examples
fab -H <remote-ubuntu-postgres-server> build log_and_analyse
```

1.6 Help

```
# ./pg_query_analyser -h
Usage: ./pg_query_analyser [flags]
Options:
  -h, --help=[false]
  -u, --users=[]
  -v, --verbose=[false]
  -i, --input-file=[/var/log/postgresql/postgresql-9.1-main.log]
  -d, --databases=[]
  -t, --query-types=[SELECT,UPDATE,INSERT,DELETE]
  -o, --output-file=[report.html]
  --top=[20]
```

CHAPTER 2

fabfile Module

`fabfile.`**`get_env`**`()`

Get the env with all variables parsed using Python string formatting

Example:

```
>>> from fabric import api
>>> api.env.foo = 'The value of Foo!'
>>> api.env.bar = 'The value of Bar and foo: %(foo)s'
>>> get_env().bar
'The value of Bar and foo: The value of Foo!'
```

The parser does `ENV_PARSE_PASSES` over the variables so nested variables are supported.

It is also possible to add host-specific configuration overrides using the `ENV_OVERRIDES`. Something like this will give you host specific support:

```
>>> from fabric import api
>>> ENV_OVERRIDES['host'] = {'my_special_host': {'foo': 'Special Foo!'}}
>>> api.env.foo = 'The value of Foo!'
>>> api.env.bar = 'The value of Bar and foo: %(foo)s'
>>> api.env.host = 'my_special_host'
>>> get_env().bar
'The value of Bar and foo: Special Foo!'
```

`fabfile.`**`wrap_environments`**`()`

Wrap the command with these environment files

By using this context you can add overrides for specific hosts in external files or just add global defaults.

To use the general overrides simply add your settings to one of the files in `ENVIRONMENT_FILES` and/or add extra files to that file. To use the host-specific configuration files the settings must be in one of the files mentioned in `AVAILABLE_ENVIRONMENT_FILES`.

`fabfile.`**`enable_logging`**`()`

Enables logging on the Postgres server

1. `Copy` *pg_query_analyser_log.conf* to the remote server

2. Add '**'**include pg_query_analyser_log.conf ' to the main postgres config file
3. Reload postgres

Including the config file is done by `appending` the include line to the main Postgres config or by *`uncommenting`* the line if already exists.

`fabfile.`**`comment`**(*file*, *line*)

Comment the given line in the given file using perl

This essentially does the same as the fabric `comment` method but because of weird escaping issues I couldn't get that one to work.

`fabfile.`**`uncomment`**(*file*, *line*)

Uncomment the given line in the given file using perl

This essentially does the same as the fabric `uncomment` method but because of weird escaping issues I couldn't get that one to work.

`fabfile.`**`disable_logging`**()

Disable logging on the Postgres server

1. *`Comment`* the include for the config file from the enable step above
2. Reload postgres

`fabfile.`**`wait`**()

A waiting task with a ETA indicator

This task waits `api.env.log_duration` seconds and tells you how much time is left.

`fabfile.`**`is_installed`**(*package*)

ubuntu/debian specific command to check if a package is currently installed

`fabfile.`**`install`**(*package*)

Install the package if not installed and returns whether it was installed or already existed

`fabfile.`**`uninstall`**(*package*)

Uninstall the package and purge the settings

WARNING: Since this purges the setting this should only be used if this command was the command that installed the package

`fabfile.`**`analyse`**()

Upload the query analyser, analyse the logs and download the report

1. Create a temporary directory to do the parsing (default /tmp/postgres)
2. Check if libqt4-sql is installed as this is a requirement to run the app. if it's not installed, it will be installed automatically.
3. Upload the pg_query_analyser binary for the platform. automatically uploads the version for this ubuntu version with this architecture (if you don't have the binary, use the build command)
4. Run pg_query_analyser over the current logfile
5. Copy the report to your local machine

`fabfile.`**`build`**()

Build the application on the remote system and download to the local pc

1. install *git*, *libqt4-dev* and *qt4-qmake* if needed
2. git clone the *pg_query_analyser* repository
3. run *qmake* and *make* in the cloned directory

4. download the generated binary into *pg_query_analyser_<os_version>_<architecture>*
5. remove the packages from step 1 only if they were installed by this step
6. remove the temporary directory which was created by the *git clone*

`fabfile.`**`log_and_analyse`**`()`

Do a full log and analyse cycle

1. `enable_logging()`
2. `wait()`
3. `analyse()`
4. `disable_logging()`

`fabfile.`**`analyse`**

Upload the query analyser, analyse the logs and download the report

1. Create a temporary directory to do the parsing (default /tmp/postgres)
2. Check if libqt4-sql is installed as this is a requirement to run the app. if it's not installed, it will be installed automatically.
3. Upload the pg_query_analyser binary for the platform. automatically uploads the version for this ubuntu version with this architecture (if you don't have the binary, use the build command)
4. Run pg_query_analyser over the current logfile
5. Copy the report to your local machine

`fabfile.`**`build`**

Build the application on the remote system and download to the local pc

1. install *git*, *libqt4-dev* and *qt4-qmake* if needed
2. git clone the *pg_query_analyser* repository
3. run *qmake* and *make* in the cloned directory
4. download the generated binary into *pg_query_analyser_<os_version>_<architecture>*
5. remove the packages from step 1 only if they were installed by this step
6. remove the temporary directory which was created by the *git clone*

`fabfile.`**`comment`**`(`*file*, *line*`)`

Comment the given line in the given file using perl

This essentially does the same as the fabric `comment` method but because of weird escaping issues I couldn't get that one to work.

`fabfile.`**`disable_logging`**

Disable logging on the Postgres server

1. `Comment` the include for the config file from the enable step above
2. Reload postgres

`fabfile.`**`enable_logging`**

Enables logging on the Postgres server

1. `Copy` *pg_query_analyser_log.conf* to the remote server
2. Add 'include pg_query_analyser_log.conf ' to the main postgres config file
3. Reload postgres

Including the config file is done by `appending` the include line to the main Postgres config or by *`uncommenting`* the line if already exists.

`fabfile.`**`get_env`**`()`

Get the env with all variables parsed using Python string formatting

Example:

```
>>> from fabric import api
>>> api.env.foo = 'The value of Foo!'
>>> api.env.bar = 'The value of Bar and foo: %(foo)s'
>>> get_env().bar
'The value of Bar and foo: The value of Foo!'
```

The parser does `ENV_PARSE_PASSES` over the variables so nested variables are supported.

It is also possible to add host-specific configuration overrides using the `ENV_OVERRIDES`. Something like this will give you host specific support:

```
>>> from fabric import api
>>> ENV_OVERRIDES['host'] = {'my_special_host': {'foo': 'Special Foo!'}}
>>> api.env.foo = 'The value of Foo!'
>>> api.env.bar = 'The value of Bar and foo: %(foo)s'
>>> api.env.host = 'my_special_host'
>>> get_env().bar
'The value of Bar and foo: Special Foo!'
```

`fabfile.`**`install`**`(`*package*`)`

Install the package if not installed and returns whether it was installed or already existed

`fabfile.`**`is_installed`**`(`*package*`)`

ubuntu/debian specific command to check if a package is currently installed

`fabfile.`**`log_and_analyse`**

Do a full log and analyse cycle

1. *`enable_logging()`*
2. *`wait()`*
3. *`analyse()`*
4. *`disable_logging()`*

`fabfile.`**`uncomment`**`(`*file*, *line*`)`

Uncomment the given line in the given file using perl

This essentially does the same as the fabric `uncomment` method but because of weird escaping issues I couldn't get that one to work.

`fabfile.`**`uninstall`**`(`*package*`)`

Uninstall the package and purge the settings

WARNING: Since this purges the setting this should only be used if this command was the command that installed the package

`fabfile.`**`wait`**

A waiting task with a ETA indicator

This task waits `api.env.log_duration` seconds and tells you how much time is left.

`fabfile.`**`wrap_environments`**`()`

Wrap the command with these environment files

By using this context you can add overrides for specific hosts in external files or just add global defaults.

To use the general overrides simply add your settings to one of the files in `ENVIRONMENT_FILES` and/or add extra files to that file. To use the host-specific configuration files the settings must be in one of the files mentioned in `AVAILABLE_ENVIRONMENT_FILES`.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index