NxPsr : Simple Nginx log parser (CFL log format)

Technical Desciption

The goal was to design a simple parser to store log and console print pertinent informations

- Configurable parsing
- See requests by section
- Ability to trigger an alert if request by seconds averaged over 2 min pass a limit

Design

Python3 was used, as it is practical for that type of application and fit well with the time limit imposed. Manipulating strings and high level helper allowed a faster development. The program was structured as :

- app.py : main and console management.
- dataplane.py : data ingestion and integration, powored by sqlite.
- wrapper.py : a middle ware allowing the query to the dataplane to forward filtered information to the console printing
- log_demo.py : log generator for demonstration purpose

The dateplane use pattern matching to get faster result and use a default 5000 batch entriess to push to sqlite. The program is multithreaded but could subdue further improvements, indeed only the update is threaded from the main console management. If more time were allowed, ingestion should also be multiprocessed. Sadly it would require architecture changes to implement a threading lock on the sqlite commit since we would have concurrent push to the db and a flow in breath first for the processing of the log.

Technical dependencies and import

NxPsr uses a number of projects to work properly:

- [Rich] - Enhanced console printing
    - Sqlite3 Python wrapper to use sqlite
- [Pytest] - Python test suit

it uses the following imports:

```
import argparse
import calendar
import copy
import datetime
import itertools
import logging
import re
import rich
import signal
import sqlite3
import sys
import threading
import time
```

Installation

Install the dependencies by using the script provided

```
1.install.sh
2.test.sh
3.1.run_demo_clf.sh
3.2.run_demo_nginx.sh
4.run.sh
```

Format support

It currently support 2 formats but could be easily extended (default CLF, -n for nginx) CLF log type :

- config :

  ```
  '$remote_addr - $user [$timestamp] "$request" $response_code $response_size'
  ```

- example :
```sh

127.0.0.1 - james [09/May/2018:16:00:39 +0000] "GET /report HTTP/1.0" 200 123
127.0.0.1 - jill [09/May/2018:16:00:41 +0000] "GET /api/user HTTP/1.0" 200 234
127.0.0.1 - frank [09/May/2018:16:00:42 +0000] "POST /api/user HTTP/1.0" 200 34
127.0.0.1 - mary [09/May/2018:16:00:42 +0000] "POST /api/user HTTP/1.0" 503 12

```
Nginx log type :
- config :
```sh
'$remote_addr - $user [$timestamp] "$request" $response_code $response_size "$referer"
"$user_agent"'
```

- example : ```sh

10.10.14.5 - - [29/Jun/2020:16:57:59 +0200] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:69.0) Gecko/20100101 Firefox/69.0" 10.10.14.5 - - [29/Jun/2020:16:58:30 +0200] "GET /info.php HTTP/1.1" 404 153 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:69.0) Gecko/20100101 Firefox/69.0" 10.10.14.5 - - [29/Jun/2020:16:59:21 +0200] "GET /info.php HTTP/1.1" 200 20 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:69.0) Gecko/20100101 Firefox/69.0" 10.10.14.5 - - [29/Jun/2020:17:11:35 +0200] "GET /info.php HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:69.0) Gecko/20100101 Firefox/69.0"

```


## Potential improvements :

The program has some shortcomings that could be improved :
- Implement a proper loggin process
```

```
- Multi processing on ingestion (multi threading won't imporve much because we are cpu
bound).
- Use panda or numpy to operate date convertion from dateutil, because the parser is
currently the weakest link.
- Complete use of pattern to get faster ingestion from logs (including request).
- Better planification: currently the ingestion and the queries are done every 10
seconds :They could be decoupled and the            ingestion run multiple times
between the status update (10s intervals).
- Better resiliency to error : the parser expect well formated log, and doesn't
correct time slides if it takes more times to process.
- Splitting the console management to a dedicated class.
- Providing a compiled version.
- Migrate to CPython to get better performances (pypy didn't improve performances, it
got even worse)



## Sum up

It was a good and interesting problem. I am not currently satisfied with the
performances. Ingestion runs at 10k log/secondes. It will not be enough in case of a
full production environment. The migration to a more robust library is necessary. The
test suit runs a 50k ingestion and update in 4,5 secondes on my local machine.
To use panda it needs to be refactored, indeed it seems we only get marginal gains
with single parsing.
To use numpy we get to cast dates at input and output. It gets faster on the
conversion but raises complexity and lower margin for consistency.
Also, new functionalities could be added to detect security patterns for example.

I would love to hear more about your point of view and expertise.

All the best
Philippe ARMANDO


[//]: #

   [Rich]: <https://github.com/willmcgugan/rich>
   [Pytest]: <https://github.com/pytest-dev/pytest>
=
```