proof_platform

Scalable Web Scrapping

User guide

Libor Polčák, Tomáš Kocman

TARZAN project VI20172020062 document
Faculty of Information Technology, Brno University of Technology

Last change: June 26, 2020

proof_platform — User guide

Libor Polčák, Tomáš Kocman

Faculty of Information Technology, Brno University of Technology, e-mail: `polcak@fit.vutbr.cz`

This tool allows web page content scrapping and exporting the content as a compressed archive. The web crawl is performed using user-supplied regular expressions that may represent for example Torrent file names, Bitcoin wallets or keywords. Collected data may be used for law enforcement and other entitites, such as searching for information about a specific product or personal archive of web pages. The tool is designed with scalability in mind. The crawling jobs can be distributed.

The aim of this document is to introduce the tool `proof_platform`[1] developed by *Integrated platform for analysis of digital data from security incidents* project.

1 Comparison of proof_platform and Winit

As the *Integrated platform for analysis of digital data from security incidents* project developed a similar tool — Winit, let us compare the two project.

Whereas Winit aims on Windows users that do want to let their computer all the work, proof_platform aims to be a fully scalable server-side option for web scrapping.

proof_platform goals are following:

– scalability,
– distributed environment,
– integrity.

2 Tool use case — What does a web page contains and what does it look like?

There are a lot of use cases during which an investigator is interested in a archive content of a web page. `proof_platform` aims to archive the exact look even if a part of the page is created by JavaScript.

Besides investigators, the tool can be handy to archive web pages for any purpose (provided that the archiver has legal grounds). Also, the tool is useful for researchers investigating trends in web development.

See the diploma thesis of Tomáš Kocman[2] for more use cases.

[1] `https://gitlab.com/tomaskocman/proof_platform/`
[2] `https://www.fit.vut.cz/study/thesis/21459/`

3 User Manual

3.1 Architecture overview

Currently, the application is managed through the REST API. See the `EXPOSE` port directive, by default 5000. By creating jobs through the API, the platform initiates Scrapy to crawl the web. Scrapy searches HTML for regex pattern. For each matched page, Scrapy saves related resources to the Redis database.

Lemmiwinks processes further process data obtained by Scrapy. Specifically, it creates MAFF archives. The archive contains `index.htm` and `index_files` directory containing resources for used by the `index.htm` file. The MAFF archive can be treated as a zip file which means one can easily unzip the archive and open the archived web page.

Information about the processed jobs is stored into the Postgre database, see Figure 2 for more details about the schema.

For example, you can use pgAdmin for this task. The structure of the database depicts Figure 1.

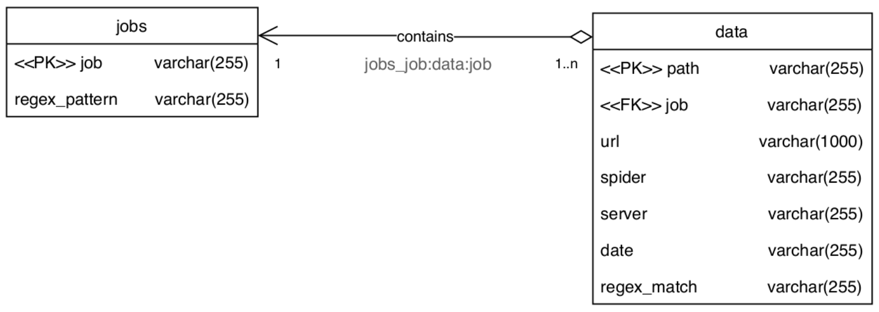

Fig. 1. Database schema used for information about the performed jobs.

The whole architecture is depicted in Figure 2.

3.2 Creating jobs

There are two endpoints in the REST API: `/jobs` (operations with all jobs) and `/jobs/<id>` (operations with a specific job).

Operations with a specific job Endpoint `/jobs/<id>` supports HTTP methods GET and DELETE.

GET method retrieves information about job with ID `<id>`. The return value contains three possible answers: job not found, job is in progress and the job finished. The return value is
`{"message": "description"}`, and HTTP_STATUS_CODE 404 (job not found) or 200 (OK).

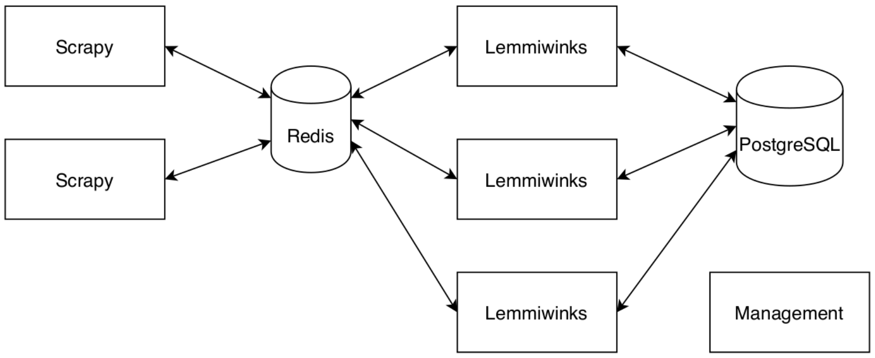

Fig. 2. The architecture of the platform.

DELETE method stops a process. The return HTTP STATUS CODE signals either that the process does not exist (404) or success (204).

Operations with all jobs Endpoint `/jobs` supports HTTP methods GET, POST, and DELETE.

GET method displays IDs of all known jobs. The return value is `{"jobNames": "array"}` with status code 200.

POST method creates a new job. The JSON has to contain all data structures necessary to configure the selected Scrapy spider. The user assigns the ID.

An examle of the input JSON: `{"robotstxtObey": "boolean", "dynamicJavaScript": "boo- lean", "job": "string", "spider": "string", "regexPattern": "string", "logLevel": "string", "logFile": "string", "loginUrl": "string", "formName": "string", "for- mLogin": "string", "formPassword": "string", "username": "string", "password": "string", "allowedDomains": "array", "startUrls": "array"}`

The return status code and body is one of the following:

```
409: {"message": "process <job> is already running"}
400: {"message": "some field is missing"}
400: {"message": "process <job> unknown spider"}
200: {}
```

For the whole list of parameters, go to the Scrapy settings manual page[3].

The platform currently supports spider basic and login. The basic spider does not requrire JSON attributes loginUrl, formName, formLogin, formPassword, username a password. These attributes are used for authentication performed by the login spider.

DELETE method allows removal of all jobs and is always successfull — HTTP status code 200.

[3] https://docs.scrapy.org/en/latest/topics/settings.html

3.3 Results

The resulting MAFF archives are created in the specified directory (see `docker-compose.yml`, services, lemmit, volumes).