# KPIs Dashboard for Invenio-Related Services

IT-CDA-DR

Tuesday 18th September, 2018

**AUTHOR:**

Niklas Persson

**SUPERVISORS:**

Lars Holm Nielsen
Jose Benito Gonzalez Lopez

CERN openlab

# Abstract

The purpose of this report is to document the project I was working on for nine weeks during the summer of 2018. As part of the CERN openlab Summer Student Program 2018 I had the opportunity to work with the Digital Repositories (IT-CDA-DR) section at CERN on developing a harvester in Python. The harvester's job was to collect key performance indicators from different services that the IT-CDA-DR section is running. The key performance indicators collected were selected based on how much value they would be to the section in order to evaluate how well the services were running.

Developing the harvester involved learning how to use the task scheduler Celery and developing clean, extendable and testable code. The final objective was to learn how deployments with containers work by using Docker with Kubernetes. The harvester was later deployed to the OpenShift platform. The collected data was visualized using the Grafana platform to give an intuitive and easy-to-use interface to the collected data.

# Acknowledgments

I would like to express my sincere gratitude to my supervisors Lars Holm Nielsen and Jose Benito Gonzalez Lopez for giving me this insightful project, the great feedback I received during the project as well as making me feel welcomed throughout the entire summer. I would also like to thank the entire IT-CDA-DR section for making me feel welcomed at CERN and the office and being a part of the team. Thank you to the CERN openlab team for organizing this entire summer and giving me the opportunity to work at CERN and learn from some of the best people in the industry. And finally a big thanks to the other students in the openlab program for the insightful discussions and the exciting adventures.

# Contents

# List of Figures

# Chapter 1

# Introduction

The initial description of the project was the following:

> Invenio is a Digital Library Framework used by many institutions around the world to provide digital repository services. At CERN there are around 8 different Invenio-powered services, and in total, there are 60+ world-wide installations using Invenio. The project consists of two parts, both related to gathering Key Performance Indicators: 1) the student would develop a crawler that will collect several public statistics (e.g. country, number of records, invenio version, etc) from all the known Invenio installations and will nicely display them using relevant (e.g. D3.js) charts directly in the project website: http://inveniosoftware.org/showcase 2) for all the Invenio-based services operated by the IT-CDA-DR section, the student will develop a crawler to gather different statistics (availability, number of records, number of files, total size for storage, etc) and will implement a dashboard to display them. This position requires both skills for front-end (HTML, JavaScript) and back-end (Python).

The focus of this project has been on developing the harvester that collected both public and private data from some of the services that are run by the IT-CDA-DR section at CERN.

## 1.1 Digital Repositories

The Digital Repositories section at CERN run several services that provide digital repositories for data produced at CERN. Some of those services will be discussed further in section 1.2 as well as the framework they are based on.

## 1.2 Invenio

Invenio is a framework written in Python used for large-scale digital repositories. CERN is the original author and Invenio is open source software [4]. CERN is running multiple services based on Invenio including Zenodo, CDS Videos and CERN OpenData.

Figure 1.1: The Invenio-based services run by CERN that were used for harvesting data for this project.

Zenodo is a platform run by CERN that is used to collect and provide research output. Publicly available research uploaded to Zenodo will receive a unique Digital Object Identifier (DOI) which can be used to find or cite specific research materials [5]. Zenodo is built on top of Invenio to distribute its digital resources.

Collections of documents related to CERN are uploaded to the *CERN Document Server* (CDS). Documents related to HEP, multimedia and others are handled by CDS while videos will be managed by *CDS Videos*. CDS Videos is currently running Invenio v3 [2].

Data produced from research at CERN are openly and publicly shared using the CERN Open Data (COD) platform. Similar to Zenodo, the data uploaded to COD are given DOIs in order to cite research data [3]. COD is another of CERN's project based on the Invenio framework.

# Chapter 2

# Harvester

Collecting the interesting performance indicators involves retrieving data from different sources then gathering them in a structured manner. The goal of the harvester is to collect the interesting KPIs then send them to CERN's Grafana instance.

## 2.1 Key Performance Indicators

Evaluating the progress of a project can be difficult and one way to analyze the performance is to use key performance indicators (KPIs). KPIs are a type of performance measurement used to evaluate whether a project or organization is working well or not [8]. There are two categories of performance indicators: quantitative and qualitative. Quantitative indicators are numeric measurements that are not subjective while qualitative indicators are based on interpretation and influenced by personal feelings.

In order for CERN to better understand how services like Zenodo, CDS Videos, and CERN Open Data are performing, multiple KPIs were collected from various sources. Having access to statistics about the services allows CERN to analyze how things are working and if anything looks out of the ordinary. If any of the services are showing potential issues appropriate actions can be taken.

## 2.2 System Architecture

The harvester is designed to be extensible and testable in order to be able to collect various types of data. Three base models make up the core of the harvester: metrics, providers and publishers. Providers retrieve data from external sources like external APIs or websites which are later stored in metrics instances. A metric is a collection of data for a specific performance indicator and the metric model simplifies updating and accessing the collected data. The publisher is in charge of publishing the collected metrics to the chosen publisher. In this project, the publisher sends the data to CERNs Grafana instance where the data is visualized. The interaction between the components can be seen in Figure 2.1.

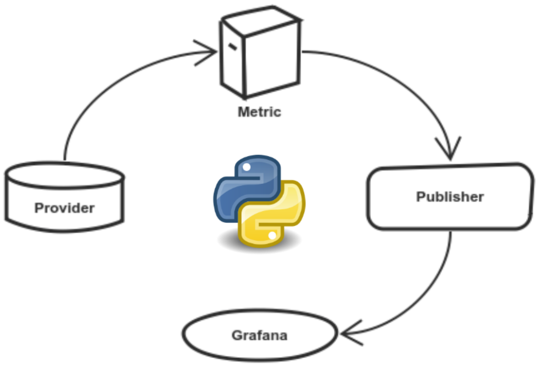


Figure 2.1: The way the different components interact with each other. The Provider collects the data which is stored as a Metric. The Metric object is sent to publisher which uploads the data to CERN's Grafana instance.

Unit tests were developed to ensure that the code continued to work despite both minor and major changes to the code. More information about the unit testing can be found in Appendix A.

## 2.3 Tasks

The harvester's tasks are executed with different frequencies and sometimes multiple tasks need to be done simultaneously so a task queue was used for managing the tasks. Celery is an asynchronous task queue which is based on distributed message passing and has support for both real-time processing and scheduling [1]. Celery performs two tasks: collect data and store it as metrics and publish the data. The collection phase and the publishing phase are chained together so the publishing happens after the data was successfully collected.

## 2.4 Sources

The harvester collects data from multiple public and private sources. Uptime and response time data is retrieved from the service UptimeRobot that collects the statistics from the website, search engine and file download/upload server of Zenodo and CDS Videos. The data is then available using a private access token through their JSON REST API.

DataCite provides statistics about the DOI related KPIs but does not have an easy-to-use API. However, the data is available on the website so the HTML source code was analyzed then parsed to retrieve the relevant data.

ServiceNow is the platform used at CERN to handle support tickets and the data was retrieved using their API. ServiceNow's API has multiple options that can be used to build the query to control the returned results and in order to keep most of that flexibility a query builder was developed. The query builder builds a URL by chaining options, for example it is possible to build queries like this: `ServiceNowQuery('incident').where(test='hello').and_(name='hello').limit(5).`

Piwik (now called Matomo) is an analytics platform we used to gather visitor statistics. To access the Piwik statistics a Single Sign-On cookie needs to be sent with the API request. The cookie can only be retrieved with a valid Kerberos ticket which made authentication more of an issue compared to the other data sources. Once the cookie is retrieved, requests can be made to Piwik using their API.

## 2.5 Deployment

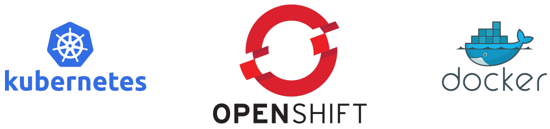


Figure 2.2: The technologies used in the deployment of the KPI harvester.

The harvester is hosted on OpenShift which is a cloud service running Docker containers. The Docker containers are executed and managed by Kubernetes which allows communication between the containers. The entire harvester deployment consists of two smaller deployments, one for the Celery worker and one for the Redis database. Each deployment has a Kubernetes pod which acts as a host for the Docker containers. Communication between the two pods are done with a Kubernetes service that manages and exposes the Redis server.

The deployment process begins with building the docker image using the latest version of the harvester code. The newly built docker image is then tagged and pushed to OpenShift where the deployment will create a new pod using the new image.

# Chapter 3

# Grafana

In order to visualize the harvested data in an intuitive way, Grafana was used to generate real-time plots. Grafana is an open source project used for monitoring and analyzing time-series data [6].

## 3.1 Dashboard

The dashboard consists of several graphs (see Figure 3.1) displaying metrics organized by the type of data it represents. The plots are divided into three categories: storage, uptime and miscellaneous metrics related to the services. The uptime graphs include the percentage of time the web, search and file upload/download services has been online since the measurements started. The response time and uptime metrics are interesting when monitoring the services to ensure that they perform to the expected level. A more detailed view of the response time graph can be seen in Figure 3.2.

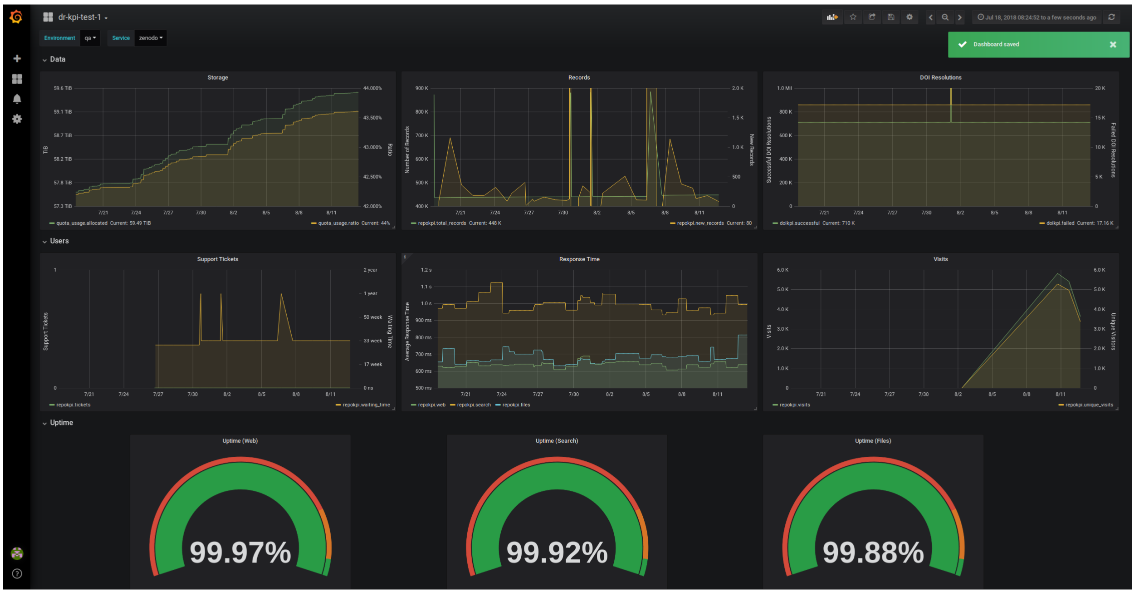


Figure 3.1: A view of the Grafana dashboard.

The graph of the number of records can be used to see the activity of each service. Both the total number of records and the new records since the last time data was collected are displayed in the graph. The number of new records could be used to see patterns of when new records are posted in order to optimize the service.

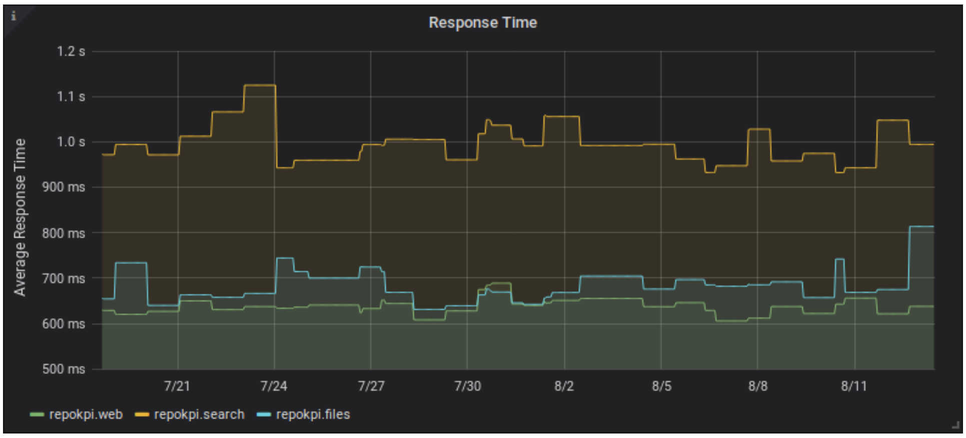


Figure 3.2: A detailed view of the response time graph showing the response time for the website, search and files services.

If users require assistance with any of the services they can create a support ticket. The tickets will be available in Service Now and a graph dedicated to support tickets information was designed to follow how the number of tickets changes over time. Another interesting metric related to support tickets is the average time it takes to resolve an issue and the average waiting time before a ticket is assigned to someone in the support team.

Other metrics included in the graphs are storage usage and DOI resolutions. Alerts can be added to the storage space plot to give a warning if the available storage space needs to be increased. DOI resolutions are interesting as statistics because of the possibilities of analyzing the number of times, both unique and in total, a DOI was searched for and looked up.

# Chapter 4

# Discussion & Conclusion

The initial project description included developing a harvester in Python that collected KPIs for different Invenio-based services that the IT-CDA-DR section are responsible for. The harvester does collect most of the data that was requested from the initial project description and it also successfully publishes the data to CERN's Grafana instance. The must-have features for the harvester were implemented so I consider the project a success and the final result can be found in the official git repository [7].

There are several improvements that can be done to the harvester. A more robust way of handling errors and collect more KPIs from the different services. The second part of this project was not completed due to time constraints but involved developing a module to the Invenio framework that collects data from all the Invenio-based services that would want to be included in the statistics. This would have been an exciting project to learn more about the inner workings of Invenio.

# Bibliography

[1] Celery. Celery: Distributed Task Queue. http://www.celeryproject.org/. 4

[2] CERN. About CDS Videos. https://videos.cern.ch/about. 2

[3] CERN. CERN Open Data Portal. http://opendata.cern.ch/docs/about. 2

[4] CERN. Invenio – Open Source framework for large-scale digital repositories. https://invenio-software.org/. 1

[5] CERN. Research. Shared. https://about.zenodo.org/. 2

[6] Grafana. Grafana: The analytics platform for all your metrics. https://grafana.com/grafana. 6

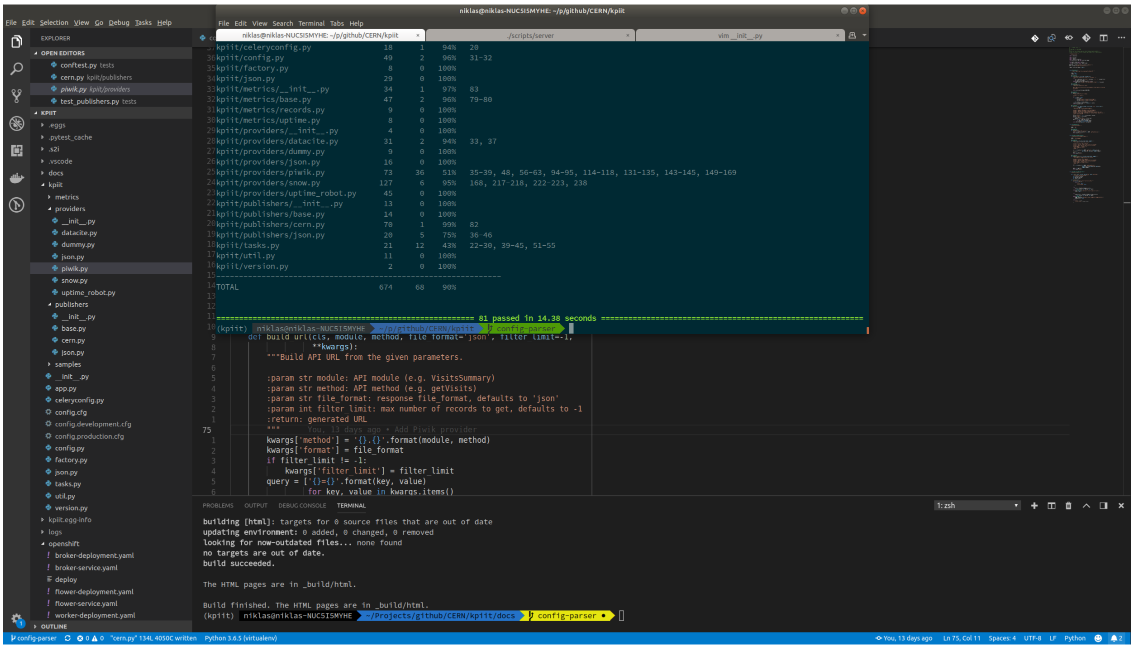
[7] Niklas Persson. Collect and publish KPIs. https://github.com/inveniosoftware-contrib/kpiit, 2018. 8

[8] F. John Reh. What You Need to Know About Key Performance Indicators. https://www.thebalancecareers.com/key-performance-indicators-2275156, Aug 2018. 3

# Appendix A

# Unit Testing

Unit testing was an important part in ensuring the code continued working despite changes to the structure of the code. Figure A.1 shows the result of running the unit tests written with `pytest` as well as the test coverage. Some modules needed more tests including `piwik.py` and the Celery tasks.


niklas@niklas-NUC5I5MYHE: ~/p/github/CERN/kpiit
File Edit View Search Terminal Tabs Help
niklas@niklas-NUC5I5MYHE: ~/p/github/CERN/kpiit
./scripts/server
vim __init__.py
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
KPIIT
OUTLINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: zsh
config-parser
Python 3.6.5 (virtualenv)
Ln 75, Col 11
Spaces: 4
UTF-8
LF
Python

```
kpiit/celeryconfig.py                18      1    94%   20
kpiit/config.py                      49      2    96%   31-32
kpiit/factory.py                      8      0   100%
kpiit/json.py                        29      0   100%
kpiit/metrics/__init__.py            34      1    97%   83
kpiit/metrics/base.py                47      2    96%   79-80
kpiit/metrics/records.py              9      0   100%
kpiit/metrics/uptime.py               8      0   100%
kpiit/providers/__init__.py           4      0   100%
kpiit/providers/datacite.py          31      2    94%   33, 37
kpiit/providers/dummy.py              9      0   100%
kpiit/providers/json.py              16      0   100%
kpiit/providers/piwik.py             73     36    51%   35-39, 48, 56-63, 94-95, 114-118, 131-135, 143-145, 149-169
kpiit/providers/snow.py             127      6    95%   168, 217-218, 222-223, 238
kpiit/providers/uptime_robot.py      45      0   100%
kpiit/publishers/__init__.py         13      0   100%
kpiit/publishers/base.py             14      0   100%
kpiit/publishers/cern.py             70      1    99%   82
kpiit/publishers/json.py             20      5    75%   36-46
kpiit/tasks.py                       21     12    43%   22-30, 39-45, 51-55
kpiit/util.py                        11      0   100%
kpiit/version.py                      2      0   100%
---------------------------------------------------------------
TOTAL                               674     68    90%


=========================================================== 81 passed in 14.38 seconds ===========================================================
(kpiit) niklas@niklas-NUC5I5MYHE  ~/p/github/CERN/kpiit  config-parser
```

```python
    def build_url(cls, module, method, file_format='json', filter_limit=-1,
                  **kwargs):
        """Build API URL from the given parameters.

        :param str module: API module (e.g. VisitsSummary)
        :param str method: API method (e.g. getVisits)
        :param str file_format: response file_format, defaults to 'json'
        :param int filter_limit: max number of records to get, defaults to -1
        :return: generated URL
        """      You, 13 days ago • Add Piwik provider
        kwargs['method'] = '{}.{}'.format(module, method)
        kwargs['format'] = file_format
        if filter_limit != -1:
            kwargs['filter_limit'] = filter_limit
        query = ['{}={}'.format(key, value)
                 for key, value in kwargs.items()
```

```
building [html]: targets for 0 source files that are out of date
updating environment: 0 added, 0 changed, 0 removed
looking for now-outdated files... none found
no targets are out of date.
build succeeded.

The HTML pages are in _build/html.

Build finished. The HTML pages are in _build/html.
(kpiit) niklas@niklas-NUC5I5MYHE  ~/Projects/github/CERN/kpiit/docs  config-parser •
```

Figure A.1: Test result and test coverage when running the unit tests. A total of 81 tests were executed and passed (100%) with a test coverage of 90%.