Automation Tools for Invenio

Tool that generates report on the status of inveniosoftware repositories and suggests suitable actions

July - August 2019

AUTHOR:
Foteini Panagiotidou

SUPERVISOR:
Alexandros Ioannidis

ABSTRACT

Invenio is an open source framework, initially developed at CERN, but with many external users and contributors at this moment and prospects of growing even more in the future. Its nature as a digital library for large scale repositories renders it a very useful tool in other software projects, inside and out of CERN alike.

The maintenance of Invenio's software falls under the responsibility of the IT-CDA-DR section. The big workload that the section has to manage combined with the currently inefficient work management when it comes to Invenio, results in Invenio falling behind, which affects badly not only the section, which makes great use of Invenio in various other projects, but also the Invenio community, which stays inactive.

The goal of my two month internship at CERN was to fix this problem by developing a bot that would send reports to the maintainers of Invenio, concerning the Invenio related tasks they have to fulfil, thus helping them to organize their work and do it more efficiently.

The bot was developed using the python programming language and various contemporary technologies, such as git, github and gitter APIs, pytest, travis CI, docker and sphinx.

There currently exist two commands for the bot, *autobot report show*, that shows the global report for all maintainers and *autobot report send*, that sends the personalised reports to each corresponding maintainer.

The end result was a basic prototype for the bot and a lot of work remains to be done in the future. Some ideas for the future work are:

- Adjust the features to the users' needs and desires.
- Improve the implementation resource-wise (e.g. cache the reports).
- Include customization features (e.g. sorting and filtering of the reports).
- Add synchronization with GitHub.

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of this report is to present the content and the progress of my project as an openlab summer student at CERN during the summer of 2019.

The length of my internship was 9 weeks (1 July - 30 August), during which I worked in the IT-CDA-DR section with the Invenio team.

Invenio is an open source framework, initially developed at CERN and mainly used for storing a big amount and broad variety of files along with their metadata. As such, it serves as a digital library for large scale repositories and provides tools for the management of institutional repositories. Its maintenance falls under the responsibility of the IT-CDA-DR section, which also makes great use of its services for the rest of its ongoing projects, such as Zenodo and CDS, to name the primary ones. That practically means that most of the people in the section work on Invenio alongside their other responsibilities. Moreover, even though Invenio has its roots at CERN, as an open source software, it also has many external individual and organisational users and contributors, with prospect of growing even more in the future.

My project's objective was to improve Invenio's task management system by developing a bot that would send personalised reports to each individual in the section, responsible for the maintenance of the organization's software (for the purposes of the present report these people will be referenced as maintainers). These reports would serve not only as a regular reminder to the maintainers of the tasks that are due and await their actions, but also as a means of organizing their tasks so that they can manage them and work on them more efficiently. That fact makes the need for the reports to be personalised crucial, especially since every maintainer is responsible for different tasks and parts of inveniosoftware.

After two months of work, at the time that this report is being written, the first prototype for the bot has been implemented. As a first trial version, it includes only the very basic features that the mentioned above concept entails. Currently, the bot can be used to send personalised reports to each maintainer via gitter. The report for a specific maintainer contains a section for every *inveniosoftware* repository the maintainer is responsible for, that urges the maintainer to act on tasks concerning open issues and pull requests this repository might have (e.g. merge an open pull request that has passed all tests and has at least one approving review, comment on an open issue that has last comment from a non maintainer, etc.).

It only remains that the section tests the bot and after a trial period share their thoughts on it, so that it is clear what things were helpful and need to stay as they are, what didn't fit people's needs and comfort and has to be removed or changed and what is missing. This feedback will help determine the needed improvements, adjustments and feature extensions and give guidelines for possible future work.

The project's repository can be found here.

The project's documentation can be found here.

Closing this short introduction, I would like to thank the whole section for being so open and making my first time working a truly unique experience. I can only be happy that my project aims to improve and help their working process. Even more special thanks to my mentor, Alex, who has been kind and helpful and overall amazing beyond any expectation.

2. BACKGROUND

As mentioned in the introduction, Invenio is an open source software, used widely not only inside CERN, but also by external individuals and organisations as well. Its nature as a digital library for large scale repositories renders Invenio a very useful tool for the development of other software projects, inside and out of CERN alike. As a result, Invenio has a very broad community of users and contributors, which entails a big amount of repositories tied to it. That fact raises the question of how Invenio's software is going to be developed and maintained. As the founder of the Invenio organization, CERN undertook this responsibility and assigned it to the IT-CDA-DR section, which maintains, at this moment, 126 repositories hosted on Github under the inveniosoftware owner.

In order to keep track of this vast amount of work, the section divided the responsibilities with respect to the repositories, resulting in each individual in the section undertaking the maintenance of certain repositories, thus earning the role of maintainer for them. One repository can have more than one maintainers, which is often the case, but the efforts are of course oriented towards making the project management as simple as possible, which means that most of the repositories don't have more than 3 maintainers. This distribution of the workload is referenced in a yaml file, called repositories.yml, where the repositories accompanied by some key information can be viewed in a simplistic hierarchical format.

The presented above project management plan for Invenio may sound good in theory, but experience showed that it doesn't have satisfying results. The big Invenio related workload, which is only small part of the goals the section has to accomplish, combined with the fact that the efforts to tackle this work creates many shared responsibilities that render the management confusing and inefficient, often results in Invenio related tasks to get left behind or undone. As a consequence, Invenio's software is being outdated, which is of course unwanted, not only for the sake of the section, which uses Invenio as a valuable tool in its other projects, but also for Invenio itself, as an open source community and a CERN product with aspirations of growing in users and contributors, that is currently inactive and badly maintained and gives off an overall unpolished image.

Thus, the idea for the project I undertook during my summer internship was born. The plan was to keep the project management as it is, but enhance it, so as to make the maintainers more aware of the tasks they need to finish, prioritize their work and all in all work more efficiently. In order to do that, I was called to implement a bot, that using the information stored in the *repositories.yml* file, would send a report to each maintainer, advising them on the tasks they have to do and in that way helping them manage their work and follow through with it.

3. TECHNOLOGIES

The programming language that was used for the development of the bot was python. In order to work on the project and run my code, I created a python virtual environment with all the required dependencies and this is what I would advise for anyone that wants to install and run this bot service.

Even though I was already familiar with the python language, my involvement with this project gave me the opportunity to expand my knowledge on it and learn how to use it more efficiently, as well as how to use it with many contemporary technologies, python oriented and widely used alike.

This part of the report is dedicated to giving a short introduction to these technologies, as well as to presenting how I used them in my project and what I learned in the process.

3.1 Git

Git is a version control system that is widely used in the programming community for managing the different versions of the source code of projects. That way, the developer can go back to an old version of the code if he/she finds that the changes made are not for some reason desired after all. Moreover, git gives the ability to store different versions of the code at a certain time, which makes it easy for a team of developers to work on the same project concurrently without interfering with each others work. That fact has made git very popular as a project management system too.

Git was used in this project for both objectives; to keep track of the different versions of the code, as well as for the management of the project in general. Again, I was already familiar with git before this internship, but I had only used it in individual projects to keep track of their history. During this project, I had the opportunity not only to greatly expand my git skills on tracking and managing the different versions of my code, but also to familiarise myself with the process of managing a section project using git. Concerning the latter, we used the GitHub host in order to organize our workflow as follows:

- My supervisor created issues describing the tasks that needed to be done.
- I worked on one issue at a time and when I was done I created a pull request tied to it.
- My supervisor reviewed my pull request and I made adjustments following his comments.
- We repeated the last step until we were satisfied with the result of the work, when he could merge the pull request and close the corresponding issue.

3.2 github3.py - GitHub API

github3.py is a python library that provides the developer with methods to interact with the GitHub API. It is basically a wrapper that helps the python developer make easy use of the GitHub API without actually having to have knowledge on it and use it directly. Using this library, it's possible to have most of the normal interactions between a user and GitHub, such as viewing information concerning a repository, creating comments, issues, pull requests, etc.

A primary task in my project was to fetch the information needed for building the reports for the maintainers. For that purpose, I had to make use of the GitHub API, since Invenio's software is hosted on GitHub (under the inveniosoftware account). Due to the nature of the project, I only had to make requests to get information. First, I used github3.py to login as a user with a GitHub token, so as to have all the required rights and accesses. After that, I could use the library's methods to fetch the information on the target repositories. I found this library very helpful and easy to use and I believe it saved me a lot of time and made my code neat and easy to understand.

3.3 Gitter API

Gitter is an instant messaging chat room, used by GitHub users and developers, in order to communicate on code related issues, shared projects and repositories etc. Gitter gives the ability to create a room and invite people to it, in order to discuss on a common topic/project.

Another crucial point in my project was sending the reports to the maintainers, once they were built. For that purpose, I used Gitter as the default report dispatch resource, since it's so tightly related to GitHub and Invenio is a GitHub based organisation. The bot makes brief use of the Gitter API, in order to send the reports; first, it makes a post request in order to get the room it shares with the maintainer by providing the maintainer's username and after, it makes a post request again, in order to send the report to this room.

3.4 pytest

pytest is a python library that assists the developer in implementing tests for his/her python code, in order to determine if it works in every valid possible scenario. It is fit for writing small tests that are meant to check the functionality of snippets of code. That way, it is easier for the developer to realize where the problem in the code lies and come up with the appropriate solution. The library also allows for scaling the tests, in order to have more complex functional testing, meant for whole modules.

For this project, I used pytest only for unit test implementation. The nature of the project as an external service, that makes requests to GitHub and Gitter APIs alike, rendered another library for testing, the mock library, very useful. In more detail, when the bot runs normally, it makes requests to external APIs. These requests are resource expensive and redundant when the bot runs in test mode, so I used the mock library in order to imitate the behavior of the github3.py library classes and methods without actually making the requests to the GitHub API. That way, it was possible to check the functionality of my code without wasting time and resources.

Through developing these tests for my project's code, not only did I make the debugging and the problem resolving process easier and simpler for me (or anyone who is going to contribute to this project in the future), but I also ensured that the end product of my work will be reliable. Last but not least, I learned a lot about the professional procedures of developing a project with proper tests.

3.5 Travis CI

Travis CI is a continuous integration tool used for building and testing projects hosted at GitHub. In order to configure the way that Travis runs, you just need to add a *.travis.yml* file to your project. This file determines, among others, the programming language the project uses and the building and testing environment along with the dependencies that are required. By adding your project's repository in your Travis repositories, it is possible to ensure that Travis builds and tests the project remotely every time you push a feature to it.

We used Travis for this project in order to automate the testing procedures and improve the project's overall management. For that purpose, I added the *.travis.yml* to the project, specifying that the language used is python and that whenever something is pushed, three tests have to be run, each in a different environment; lowest, release and devel. It is also specified that the devel tests are allowed to fail. The tests run in each of these environments are defined in a *run-tests.sh* file. Except for the unit tests mentioned before, there are also tests concerning the format of the files (we followed the black format and isort conventions), as well as tests for the docs.

3.6 Docker

Docker is a virtualization software that aims to improve the procedures of running a certain project in different machines and environments by simplifying the dependencies organization and the requirements installation process. Using operating-system-level virtualization, it offers a set of platform as a service (PaaS) tools, that enable the developer to receive and deliver software in well defined packages, called containers, that are closely tied to their libraries and requirements. The communication between containers is also very clean and simple. As a result, taking care to put only the necessary requirements in a container, meaning only the requirements that are needed for it to run independently, docker makes it easy to build a container system that allows the developer to run lightweight, independent software tasks in any machine. It is easy to configure how docker will be run for someone's project by adding a *Dockerfile* in it.

We used docker in order to deploy the work done for this project. In our *Dockerfile* we determined that the base for the docker image will be the standard python3 docker image, we installed cron, as well as the requirements the project needs in order to run and of course the source code. After creating the image, we pushed it to openshift, where we could configure the environment (e.g. adding GitHub and Gitter tokens as secrets) and run the service as a cron job that will send the reports to the maintainers periodically (e.g. once per month). Using docker helped me familiarise with a cutting-edge technology that I am probably going to be using a lot in the future, but also simplified the deployment process of the project and ensured that it can run as an efficient independent service on openshift, as well as be downloaded and installed easily by any individual who wants to make local use of it.

3.7 Sphinx

Sphinx is a python library that gives the ability to the programmer to create beautiful documentation for his/her project at the same time that it is being developed and without putting much effort, since it is based on an automated process.

4. EXAMPLES

The result of my two month internship was a basic prototype for a bot that fetches target information for the repositories specified in a predefined yaml file and sends reports to the corresponding maintainers (also specified in the same yaml file) via Gitter.

There currently exist two commands that can be run via the command line*:*

- ***autobot report show***: fetches the information on the repositories and builds a ***global*** report, containing the tasks for every repository and its corresponding maintainers

```
(env) fpanagio@pcuds51:~$ autobot report show -h
Usage: autobot report show [OPTIONS]

  Autobot report show CLI.

Options:
  -o, --owner TEXT          The repo owner/organization the service is offered
                            to.
  -r, --repo TEXT           The repositories to check for notifications.
  -m, --maintainer TEXT     The maintainers to notify.
  -e, --dotenv_config PATH  The .env file to load configurations.
  -i, --ini_config PATH     The .ini file to load configurations.
  -f, --format TEXT         The report format.
  -h, --help                Show this message and exit.
```

- ***autobot report send***: fetches the information on the repositories, builds a ***personalised*** report for each maintainer containing the tasks he/she has to do and sends these individual reports

```
(env) fpanagio@pcuds51:~$ autobot report send -h
Usage: autobot report send [OPTIONS]

  Autobot report send CLI.

Options:
  -o, --owner TEXT          The repo owner/organization the service is offered
                            to.
  -r, --repo TEXT           The repositories to check for notifications.
  -m, --maintainer TEXT     The maintainers to notify.
  -e, --dotenv_config PATH  The .env file to load configurations.
  -i, --ini_config PATH     The .ini file to load configurations.
  -v, --via TEXT            Resource used for notification dispatch.
  -h, --help                Show this message and exit.
```

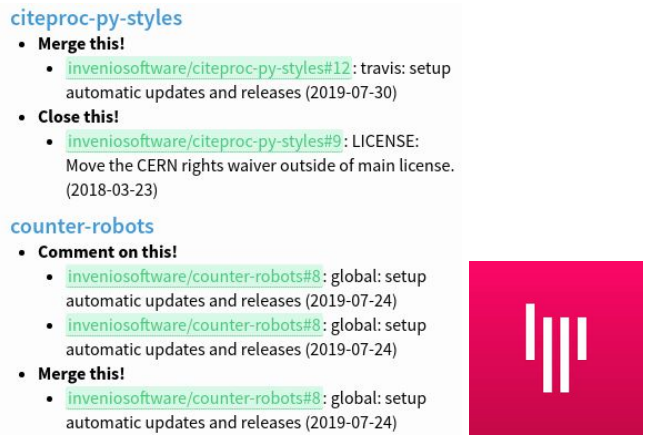
As shown in the screenshots above, the two available commands share some options that they offer to the user. Some important shared options are the ones that specify the target repositories and maintainers the application is going to fetch the information for.

- With the ***o*** option it is possible to specify the ***organization*** the reports are being generated for (in our case for Invenio).
- With the ***r*** option it is possible to specify the target ***repositories*** the report is being generated for.
- With the ***m*** option it is possible to specify the target ***maintainers*** the report is being generated for.
- With the ***e, i*** options it is possible to specify some ***configuration paths***.

Moreover, there are some offered option specific to each individual command.

- For the *autobot report show* command, it is possible to specify the ***format*** the global report is going to be shown in (e.g. json, yaml, markdown etc.) through the ***f*** option.
- For the *autobot report send* command, it is possible to specify the ***resource*** via which the personal report is going to be sent (e.g. gitter, mattermost, CERN email etc.) through the ***v*** option.

The following screenshot is an example of a personal report sent via Gitter.



In the cases that the report is too large and can't be sent directly and whole via Gitter (more than 4000 characters), a link to a GitHub gist that contains the full report is sent.

5. FUTURE WORK

There is still a lot of room left for progress and plenty material for future work. The feedback that we are going to get from the people that are going to use this first prototype are going to be valuable and critical for deciding what the course of this future work should be. The purpose of this section of the report is to present some ideas and suggestions for the work to come in the future concerning this project.

- Make adjustments in the offered features in order to fit the clients' needs and desires, as they will be revealed by the feedback. One very important feature to be determined is the nature of the actions that the bot proposes according to the repositories status, if they will need to be expanded or limited or changed in content.
- Make improvements in the implementation in order to fetch the information from GitHub faster and more efficiently in general, resource-wise. One suggested improvement would be to cache the reports, so that when the bot is called with the same parameters and nothing has changed, it won't be necessary to waste time and resources fetching the same information.
- Add features that will help the maintainers customize their report in order to manage their tasks better, such as sorting and filtering, customizing the format of the report and the resource used for receiving it (e.g. Mattermost instead of Gitter), as well as how often it will be sent.
- Add the option of being synchronized with your GitHub account and of requesting the report locally without the need of a GitHub token, but by being redirected to the GitHub login page and adding the corresponding credentials, if not already logged in.