

# Visualizing Evolving Service Performance in Python

Michael Shell  
School of Electrical and  
Computer Engineering  
Georgia Institute of Technology  
Atlanta, Georgia 30332-0250

Email: <http://www.michaelshell.org/contact.html>

Homer Simpson  
Twentieth Century Fox  
Springfield, USA

Email: [homer@thesimpsons.com](mailto:homer@thesimpsons.com) San Francisco, California 96678-2391

James Kirk  
and Montgomery Scott  
Starfleet Academy

Telephone: (800) 555-1212

Fax: (888) 555-1212

**Abstract**—The abstract goes here.

## I. INTRODUCTION

Every system is a distributed system nowadays [].

Python is currently one of the most popular programming languages for API web service development. It was the *Todo: nth* most popular programming language in the year *Todo: 2017/16?*

Flask is one of the two most popular web frameworks for Python. It is a lightweight alternative to web site and service development. This is why it is advertised as a *micro-framework*. The popularity of Flask is hinted at also by the large number *ML: Thijs, Patrick: can we find an estimate on GH?* of open source projects that are hosted on GitHub which are based on it.

However, there is no dedicated solution for monitoring the performance of Flask web-applications. Thus, every one of those *Todo: NNNN* has one of the following options when confronted with the need of gathering insight into the runtime behavior of their service:

- 1) Use a heavyweight professional API monitoring setup they require setting up a different server *ML: Thijs, Patrick: can we find a few examples of professional but overkill tools? ideally they require setting up a bunch of servers, and writing configs in XML!*
- 2) Implement their own analytics tool
- 3) Live without analytics insight into their services <sup>1</sup>

For projects which are done on a budget (e.g. research projects) the first and the second options are often not available due to time and financial constraints.

To avoid these projects ending up in the third situation, in this paper we present a low-effort, lightweight service monitoring API for Flask and Python web-services.

To start using our Python library for service visualization solution one needs to add exactly three lines of code that connect their Flask application object with the dashboard:

*ML: Vasilios: do you know how to style the code env to make it look nicer?* *ML: see below/above* *ML: Thijs, Patrick: fix this example pls*

And a small configuration file:

<sup>1</sup>This is very real option: and is exactly what happened to the API that will be presented in this case study for many months.

Listing 1. TBA

```
import dashboard
dashboard.config.from_file('d.cfg')
dashboard....
```

*ML: Thijs, Patrick: example of minimal (the most minimal possible, we'll introduce later extra configs) config file. Or, if we can we do completely w/o config that would look even better. And we'll introduce the extra stuff later*

*ML: Thijs, Patrick: a small description of how the dashboard automatically intercepts the calls to the various API calls*

*ML: Thijs, Patrick: mention also that the dashboard automatically is available at /dashboard endpoint*

*ML: Thijs, Patrick: a small screenshot of how the dashboard allows one to select the interesting*

## II. CASE STUDY

Zeeguu case study description to be used as running example throughout the rest of the paper [1]

Architecture: series of web and mobile applications built around a core web service implemented in Python and Flask which provides:

- contextual translations
- reading recommendations
- exercises

We have this system for helping learners read texts that they like, and enable them to practice with exercises generated on their past readings.

*ML: we should consider adding also one section in which the architecture/implementation and main features of the dashboard are presented before going on with discussing them in more depth in the following sections*

## III. OVERALL ENDPOINT UTILIZATION

The most basic insights that a service maintainer must answer regard the utilization of their service. Dashboard presents two perspectives that provide insight into this question.

Figure 1 shows the usage of the services on a daily basis while highlighting the contribution of the individual endpoints<sup>2</sup>.

One can see that the service at its peak has about 2500 hits per day. Moreover, one can see the way the users are

<sup>2</sup>We recommend obtaining a color version of this paper for better readability

using the platform since the different endpoints are indicators of different activity types, e.g.:

- `api.get_possible_translations` is an indicator of the amount of reading the users are doing
- `api.report_exercise_outcome` is an indicator of the amount of vocabulary practice the users are doing

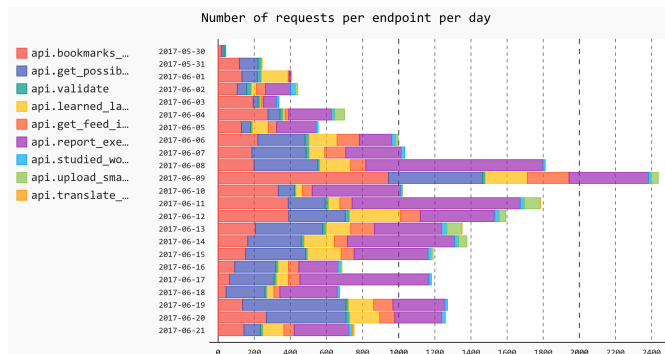


Fig. 1. Caption here

ML: Thijs, Patrick: Could we regenerte this graph with the same colors as in the previous one? Otherwise the Vissoft people will surely complain

Besides showing which endpoints are used, this view is informative for the maintainer since it provides insight into which endpoints are used. In our case study, the maintainer of the API has decided to not remove several endpoints when they saw that that were being used <sup>3</sup>



A second question that an API maintainer must answer regards cyclic patterns of usage.

ML: Vasilios: can we provide some support for this claim? a reference maybe?

ML: Thijs, Patrick: can we add vertical lines that highlight the begining of a new week (e.g. before Sunday):

Figure 2 shows only one clear pattern: the users of our API do not practice foreign languages as night. Besides that, the API is used around the clock with usually a few hundred hits per hour. So the load currently is not that high.

#### IV. VISUALIZING SERVICE PERFORMANCE

Another type of information that Dashboard can collect regards endpoint performance. Figure 3 shows a summary of the response times for various endpoints by using boxplot graphs.

After investigating this view it became clear to the maintainer that three of the endpoints had very large variation in performance. One of the most important endpoints that they decided to improve the performance of is the second from the top in Figure 3: the endpoint that returns translations. This is particularly critical since it is part of a live interaction loop.



<sup>3</sup>Usage information can also be used to increase the confidence of the maintainer that a given endpoint is not used

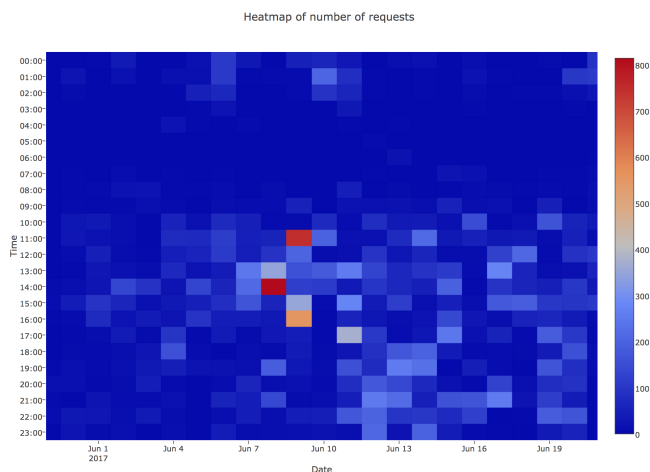


Fig. 2. Caption here

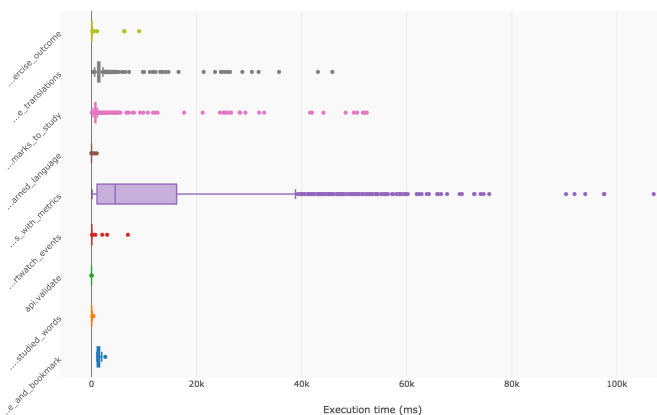


Fig. 3. ...

Once they increased the performance of the endpoint, the maintainer realized that this graph was not that useful anymore. Because they could not see the improvements.

For such situations, Dashboard provides an extra configuration directive:

ML: Thijs, Patrick: Add extra config line here

If the directive is enabled, the performance data is grouped by commit and commits are detected by analyzing the .git folder on the machine.

Figure 4 shows the performance of the give endpoint

This is important since only this way can confirm that our performance improves.

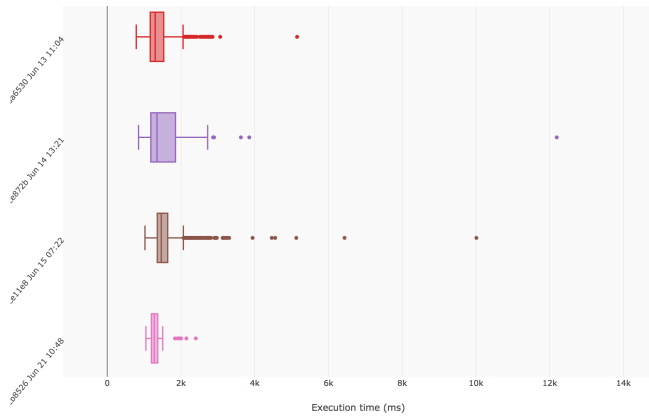


Fig. 4. Visualizing The Performance Evolution of the `api.get_possible_translations` endpoint (the second from the top in Fig. 3)

## V. USER CENTERED VISUALIZATION

Different users might have different experiences: - a user has 10K emails one has 10 emails - a user is subscribed to 20 feeds one to 2 feeds

The system will have different processing times. It is important for the DevOps-er to be able to understand the difference in performance on a per user basis.

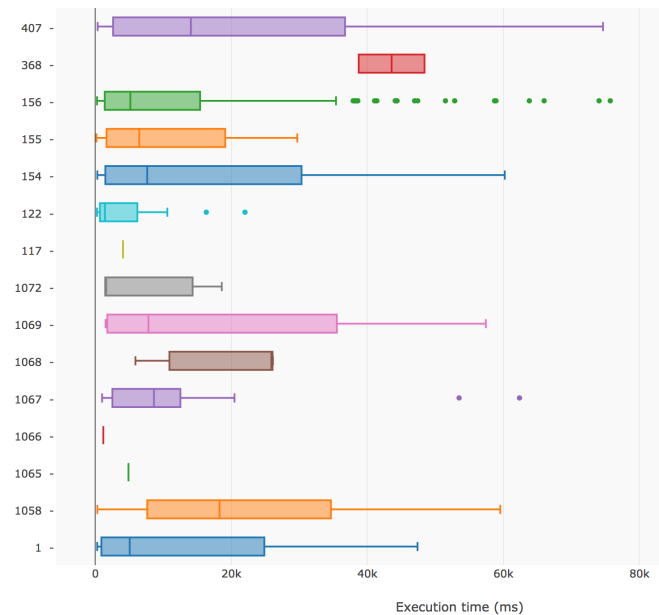


Fig. 5. One of the endpoints (`/get_feed_items_with_metrics`) shows a very high variability across users

If we try to show also per version:

## VI. TOOL AVAILABILITY

The code of Dashboard is available online at ...

The images in this paper are screenshots of the actual deployment of the tool which can be found at <https://zeeguu.unibe.ch/api/dashboard>. For the readers of this

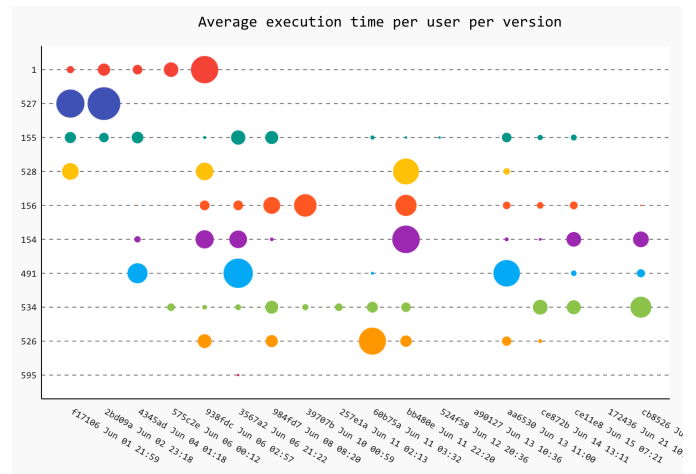


Fig. 6. Caption here

paper to be able to see the tool in action, they can login with the username and password: guest, dashboardguest!. ML: Thijs, Patrick: we should add a guest/guest username password which is allowed to only visualize things, but not modify anything, and not export anything!

## VII. RELATED WORK

Java Visualization [2]

## VIII. CONCLUSION

The conclusion goes here.

## ACKNOWLEDGMENT

The authors would like to thank...

## REFERENCES

- [1] M. F. Lungu, "Bootstrapping an ubiquitous monitoring ecosystem for accelerating vocabulary acquisition," in *Proceedings of the 10th European Conference on Software Architecture Workshops*, ser. ECSAW 2016. New York, NY, USA: ACM, 2016, pp. 28:1–28:4. [Online]. Available: <http://doi.acm.org/10.1145/2993412.3003389>
- [2] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. M. Vlissides, and J. Yang, "Visualizing the execution of java programs," in *Revised Lectures on Software Visualization, International Seminar*. London, UK: Springer-Verlag, 2002, pp. 151–162.