Flask-MonitoringDashboard Documentation

Release 1.10.5

Patrick Vogel & Thijs Klooster

Mar 06, 2018

Contents

Dashboard for automatic monitoring of Flask web services.

CHAPTER 1

What is the Flask-Monitoring-Dashboard?

The Flask Monitoring Dashboard is designed to monitor your existing Flask application. You can find a brief overview of the functionality *here*.

Or you can watch the video below:

CHAPTER 2

Functionality

The Flask Monitoring Dashboard is an extension that offers 4 main functionalities with little effort from the Flask developer:

- **Monitor the Flask application:** Our Dashboard allows you to see which endpoints process a lot of request and how fast. Additionally, it provides information about the evolving performance of an endpoint throughout different versions if you're using git.
- **Monitor your test coverage:** The dashboard allows you to find out which endpoints are covered by unit tests, allowing also for integration with Travis for automation purposes. For more information, see this file.
- **Collect extra information about outliers:** Outliers are requests that take much longer to process than regular requests. The dashboard automatically detects that a request is an outlier and stores extra information about it (stack trace, request values, Request headers, Request environment).
- **Visualize the collected data in a number useful graphs:** The dashboard is automatically added to your existing Flask application. You can view the results by default using the default endpoint (this can be configured to another route): dashboard.

For a more advanced documentation, take a look at the information on this page.

CHAPTER 3

User's Guide

If you are interested in the Flask Monitoring Dashboard, you can find more information in the links below:

3.1 Installation

This is the complete overview of installing the Flask Monitoring Dashboard. It starts from the very basic, but it is likely that you can directly go to *Installing the Flask Monitoring Dashboard Package*.

3.1.1 Install Python

You can check if you have Python installed by opening a terminal and execution the following command:

```
python3 --version
```

It should return something like 'Python 3.6.3', if not, you probably see something like 'bash: python3: command not found'. In the former case, you're ok. In the latter, you can follow this link to install Python.

3.1.2 Installing a Virtual Environment (Optional)

Although you don't need a Virtual Environment, it is highly recommend. See this page to install a Virtual Environment.

Configuring the Virtual Environment (Optional)

If you have skipped the previous section, you can also skip this one (since it's optional). Once you've installed the Virtual Environment, you need to configure it. This can be done by the following command:

```
virtualenv ENV
```

Or using the following command for Python3:

```
virtualenv --python=python3 ENV
```

Activate the Virtual Environment (Optional)

This is the last part of the configuring the virtual environment. You should do this before you want to execute any python script/program. It is (again) one simple command:

```
source bin/activate
```

3.1.3 Installing the Flask Monitoring Dashboard Package

You can install the Flask Monitoring Dashboard using the command below:

```
pip install flask_monitoringdashboard
```

Alternatively, you can install the Flask Monitoring Dashboard from Github:

```
git clone https://github.com/flask-dashboard/Flask-MonitoringDashboard.git
cd Flask-MonitoringDashboard
python setup.py install
```

3.1.4 Setup the Flask Monitoring Dashboard

After you've successfully installed the package, you can use it in your code. Suppose that you've already a Flask application that looks like this:

```
from flask import Flask
app = Flask(__name__)

...

@app.route('/')
def index():
    return 'Hello World!'


if __name__ == '__main__':
  app.run(debug=True)
```

You can add the extension to your Flask application with only two lines of code:

```
...
import flask_monitoringdashboard as dashboard
dashboard.bind(app)
```

Together, it becomes:

```
from flask import Flask
import flask_monitoringdashboard as dashboard

app = Flask(__name__)
dashboard.bind(app)
```

```
...

@app.route('/')
def index():
    return 'Hello World!'

if __name__ == '__main__':
  app.run(debug=True)
```

3.1.5 Further configuration

You are now ready for using the Flask Monitoring Dashboard, and you can already view the dashboard at: dashboard.

However, the Dashboard offers many functionality which have to be configured. This is explained on the configuration page.

3.2 Configuration

Once you have successfully installed the Flask Monitoring Dashboard from this page, you can use the advanced features by correctly configuration the dashboard.

3.2.1 Using a configuration file

You can use a configuration file for all options below. This is explained in the following section. In order to configure the dashboard with the configuration-file, you can use the following function:

```
dashboard.config.init_from(file='/<path to file>/config.cfg')
```

Thus, it becomes:

```
from flask import Flask
import flask_monitoringdashboard as dashboard

app = Flask(__name__)
dashboard.config.init_from(file='/<path to file>/config.cfg')
# Make sure that you first configure the dashboard, before binding it to your Flask␣
↪application
dashboard.bind(app)
...

@app.route('/')
def index():
    return 'Hello World!'

if __name__ == '__main__':
  app.run(debug=True)
```

Instead of having a hard-coded string containing the location of the config file in the code above, it is also possible to define an environment variable that specifies the location of this config file. The line should then be:

```
dashboard.config.init_from(envvar='DASHBOARD_CONFIG')
```

This will configure the dashboard based on the file provided in the environment variable called *DASHBOARD_CONFIG*.

3.2.2 The content of the configuration file

Once the setup is done, a configuration file ('config.cfg') should be set next to the python file that contains the entry point of the app. The following things can be configured:

```
[dashboard]
APP_VERSION=1.0
CUSTOM_LINK=dashboard
DATABASE=sqlite:////<path to your project>/dashboard.db
USERNAME=admin
PASSWORD=admin
GUEST_USERNAME=guest
GUEST_PASSWORD=['dashboardguest!', 'second_pw!']
GIT=/<path to your project>/.git/
OUTLIER_DETECTION_CONSTANT=2.5
TEST_DIR=/<path to your project>/tests/
N=5
SUBMIT_RESULTS_URL=http://0.0.0.0:5000/dashboard/submit-test-results
COLORS={'main':[0,97,255], 'static':[255,153,0]}
```

This might look a bit overwhelming, but the following list explains everything in detail:

- **APP_VERSION:** The version of the app that you use. Updating the version helps in showing differences in execution times of a function over a period of time.
- **CUSTOM_LINK:** The dashboard can be visited at localhost:5000/{{CUSTOM_LINK}}.
- **DATABASE:** Suppose you have multiple projects where you're working on and want to separate the results. Then you can specify different database_names, such that the result of each project is stored in its own database.
- **USERNAME** and **PASSWORD:** Must be used for logging into the dashboard. Thus both are required.
- **GUEST_USERNAME** and **GUEST_PASSWORD:** A guest can only see the results, but cannot configure/download any data.
- **GIT:** Since updating the version in the configuration-file when updating code isn't very useful, it is a better idea to provide the location of the git-folder. From the git-folder, The version is automatically retrieved by reading the commit-id (hashed value). The location is relative to the configuration-file.
- **OUTLIER_DETECTION_CONSTANT:** When the execution time is more than this $constant * average$, extra information is logged into the database. A default value for this variable is 2.5, but can be changed in the config-file.
- **TEST_DIR**, **N**, **SUBMIT_RESULTS_URL:** To enable Travis to run your unit tests and send the results to the dashboard, you have to set those values:
  - **TEST_DIR** specifies where the unit tests reside.
  - **SUBMIT_RESULTS_URL** specifies where Travis should upload the test results to. When left out, the results will not be sent anywhere, but the performance collection process will still run.
  - **N** specifies the number of times Travis should run each unit test.
- **COLORS:** The endpoints are automatically hashed into a color. However, if you want to specify a different color for an endpoint, you can set this variable. It must be a dictionary with the endpoint-name as a key, and a list of length 3 with the RGB-values. For example:

```
COLORS={'main':[0,97,255], 'static':[255,153,0]}
```

3.2.3 What have you configured?

A lot of configuration options, but you might wonder what functionality is now supported in your Flask Monitoring Dashboard? Have a look at this file to find the answer.

3.3 Detailed Functionality

The functionality of the dashboard is divided into 4 main components. You can find detailed information about every comopont below:

3.3.1 Endpoint Monitoring

The core functionality of the Dashboard is monitoring which Endpoints are heavily used and which are not. If you have successfully configured the dashboard from this page, then you are ready to use it. In order to monitor a number of endpoints, you have to do the following:

1. Log into the dashboard at: http://localhost:5000/dashboard/login
2. Go to the Rules-tab in the left menu: http://localhost:5000/dashboard/rules
3. Select the rules that you want to monitor.
4. Wait until a request to this endpoint is being made.
5. Go to the Measurements-tab in the left menu: http://localhost:5000/measurements/overview

As you can see on the last page (http://localhost:5000/measurements/overview) there are a number of tabs available. All tabs are explained in the *Visualisations-tab*.

Collected data

For each request that is being to a monitored-endpoint, the following data is recorded:

- **Exeuction time:** measured in ms.
- **Time:** the current timestamp of when the request is being made.
- **Version:** the version of the Flask-application. This can either be retrieved via the *CUSTOM_VERSION* value, or via the *GIT* value. If both are configured, the *GIT* value is used.
- **group_by:** An option to group the collected results. As most Flask applications have some kind of user management, this variable can be used to track the performance between different users. It is configured using the following command:

```
def get_user_id():
    return '1234'  # replace with a function to retrieve the id of the
                   # user within a request.

dashboard.config.get_group_by = get_session_id
# Note that the function-pointer is passed, not the function itself.
```

Thus, it becomes:

```
from flask import Flask
import flask_monitoringdashboard as dashboard

app = Flask(__name__)
dashboard.config.init_from(file='/<path to file>/config.cfg')

def get_user_id():
    return '1234'  # replace with a function to retrieve the id of the
                   # user within a request.

dashboard.config.get_group_by = get_session_id
dashboard.bind(app)

@app.route('/')
def index():
    return 'Hello World!'

if __name__ == '__main__':
    app.run(debug=True)
```

- **IP:** The IP-address from which the request is made.

Observations

Using the collected data, a number of observations can be made:

- Is there a difference in execution time between different versions of the application?
- Is there a difference in execution time between different users of the application?
- Is there a difference in execution time between different IP addresses? *As tracking the performance between different users requires more configuration, this can be a quick alternative.*
- On which moments of the day does the Flask application process the most requests?
- What are the users that produce the most requests?
- Do users experience different execution times in different version of the application?

3.3.2 Test-Coverage Monitoring

To enable Travis to run your unit tests and send the results to the dashboard, four steps have to be taken:

1. Update the config file ('config.cfg') to include three additional values, *TEST_DIR*, *SUBMIT_RESULTS_URL* and *N*.
    - **TEST_DIR** specifies where the unit tests reside.
    - **SUBMIT_RESULTS_URL** specifies where Travis should upload the test results to. When left out, the results will not be sent anywhere, but the performance collection process will still run.
    - **N** specifies the number of times Travis should run each unit test.
2. The installation requirement for the dashboard has to be added to the *setup.py* file of your app:

```
dependency_links=["https://github.com/flask-dashboard/Flask-
↪MonitoringDashboard/tarball/master#egg=flask_monitoringdashboard"]

install_requires=('flask_monitoringdashboard')
```

3. In your *.travis.yml* file, three script commands should be added:

```
export DASHBOARD_CONFIG=./config.cfg
export DASHBOARD_LOG_DIR=./logs/
python -m flask_monitoringdashboard.collect_performance
```

   The config environment variable specifies where the performance collection process can find the config file. The log directory environment variable specifies where the performance collection process should place the logs it uses. The third command will start the actual performance collection process.

4. A method that is executed after every request should be added to the blueprint of your app. This is done by the dashboard automatically when the blueprint is passed to the binding function like so:

```
dashboard.bind(app=app, blue_print=api)
```

   This extra method is needed for the logging, and without it, the unit test results cannot be grouped by endpoint that they test.

3.3.3 Outliers

It is always useful to investigate why certain requests take way longer to process than other requests. If this is the case, it is seen as an outlier. Mathematically an outlier is determined if the execution of the request is longer than:

$> average * constant$

Where *average* is the average execution time per endpoint, and *constant* is given in the configuration-file (otherwise its default value is 2.5).

When a request is an outlier, the dashboard stores more information, such as:

- The stacktrace in which it got stuck.
- The percentage of the CPU's that are in use.
- The current amount of memory that is used.
- Request values.
- Request headers.
- Request environment.

The data that is collected from outliers, can be seen by the following procedure:

1. Go to the Measurements-tab in the left menu: http://localhost:5000/measurements/overview
2. Click on the Details-button (on the right side) for which endpoint you want to see the outliers-information.
3. Go to the outliers-tab: http://localhost:5000/dashboard/<endpoint-name>/main/outliers

3.3.4 Visualisations

There are a number of visualisation generated to view the results that have been collected in (Endpoint-Monitoring) and (Test-Coverage Monitoring).

The main difference is that visualisations from (Endpoint-Monitoring) can be found in the tab 'Measurements' (in the left menu), while visualisations from (Test-Coverage Monitoring) can be found in the tab 'Test Monitor' (below the 'Measurements'-tab).

The 'Measurements'-tab contains the following content:

1. **Overview:** A table with the all the endpoints that are being monitored (or have been monitored in the past). This table provides information about when the endpoint is last being requested, and the average execution time. Furthermore, it has a 'Details' button on the right. This is explained further in (6.).
2. **Heatmap of number of requests:** This graph provides information for each hour of the day about how often the endpoint is being requested. In this graph it is possible to detect popular hours during the day.
3. **Requests per endpoint:** This graph provides a row of information per day. In this graph, you can find whether the total number of requests grows over days.
4. **Time per version:** This graph provides a row of information per application-version. In this graph, you can find whether the execution time for all requests grows over the versions of the application.
5. **Time per endpoint:** This graph provides a row of information per endpoint. In that row, you can find all the requests for that endpoint. This provides information whether certain endpoints perform better (in terms of execution time) than other endpoints.
6. For each endpoint, there is a 'Details'-button. This provides the following information (thus, all information below is specific for a single endpoint):
    - **Heatmap:** The same heatmap as explained in (2.), but this time it is focused on the data of that particular endpoint only.
    - **Time per hour:** A vertical bar plot with the execution time (minimum, average and maximum) for each hour.
    - **Hits per hour:** A vertical bar plot with the number of requests for that endpoint.
    - **Time per version per user:** A circle plot with the average execution time per user per version. Thus, this graph consists of 3 dimensions (execution time, users, versions). A larger circle represents a higher execution time.
    - **Time per version per ip:** The same type of plot as (Time per version per user), but now that users are replaced by IP-addresses.
    - **Time per version:** A horizontal box plot with the execution times for a specific version. This graph is equivalent to (4.), but now it is focused on the data of that particular endpoint only.
    - **Time per user:** A horizontal box plot with the execution time per user. In this graph, it is possible to detect if there is a difference in the execution time between users.
    - **Outliers:** See Section (Outliers) above.

3.3.5 Need more information?

See the contact page to see how you can contribute on the project. Furthermore you can request this page for questions, bugs, or other information.

CHAPTER 4

Developer information

4.1 Contact

In this file, you can find information to find us.

4.1.1 Found a bug?

Post an issue on Github. You can use the template below for the right formatting:

Issue Template

- Expected Behavior

  Tell us what should happen

- Current Behavior

  Tell us what happens instead of the expected behavior

- Possible Solution

  Not obligatory, but suggest a fix/reason for the bug

- Steps to Reproduce

  Provide a link to a live example, or an unambiguous set of steps to reproduce this bug. Include code to reproduce, if relevant

  1.
  2.
  3.
  4.

- Context (Environment)

  How has this issue affected you? What are you trying to accomplish? Providing context helps us come up with a solution that is most useful in the real world

- Detailed Description

  Provide a detailed description of the change or addition you are proposing

- Possible Implementation

  Not obligatory, but suggest an idea for implementing addition or change

4.2 TODO List

All things that can improved in Flask Monitoring Dashboard are listed below.

4.2.1 Features to be implemented

- [ ] Pagination of results. See this page: http://flask.pocoo.org/snippets/44/

  This process is especially useful for the following results:

  - Page '/rules' - Max 20 endpoints per table.
  - Page '/measurements/overview' - Max 20 endpoints per table.
  - Page '/measurements/heatmap' - Heatmap for each week
  - Page '/measurements/requests' - Barplot for each week
  - Page '/measurements/versions' - Max 10 versions per plot
  - Page '/measurements/endpoints' - Max 10 endpoints per plot.
  - Page '/result/<endpoint>/heatmap - Heatmap for each week.
  - Page '/result/<endpoint>/time_per_hour' - barplot for each week.
  - Page '/result/<endpoint>/hits_per_hour' - barplot for each week.
  - Page '/result/<endpoint>/time_per_version' - Max 10 versions per plot.
  - Page '/result/<endpoint>/outliers' - Max 20 results per table.

4.2.2 Work in progress

Use this section if someone is already working on an item above.

4.3 Change Log

All notable changes to this project will be documented in this file. This project adheres to Semantic Versioning. Please note that the changes before version 1.10.0 have not been documented.

4.3.1 Unreleased

Changed

- Added TODO List
- Updated functionality to retrieve the stacktrace of an Outlier

4.3.2 v1.10.0

Changed

- Added security for automatic endpoint-data retrieval.
- Added test for export_data-endpoints
- Added MIT License.
- Added documentation