opium Documentation

Release 0.1.1

Kolokotronis Panagiotis

Oct 26, 2020

CONTENTS:

CHAPTER ONE

OPIUM

OpenShift Pod Independent Usage Metrics (OPIUM) pulling & aggregating metrics from multiple pods made easy!

- Free software: MIT license
- Documentation: https://opium.readthedocs.io.

Aggregating metrics from pods through OKD's router can be a challenge. OPIUM offers the next best thing. While it doesn't (yet) aggregate the varius metrics it gathers the metrics from the defined Deployment Configs on a specific OKD project and serves them to a single /metrics endpoint. This way it allows for easier gathering of per-pod metrics.

Of course, given no real aggregation is happening, it makes sense that your metrics should contain some unique label value to differentiate the same measurements from different pods. The contents of the HOSTNAME env variable are usually a good fit for the value of such a label since they will be unique (within a project). This way 'true' aggregation can happen at a later stage (i.e. with a sum across all instances while graphing the values).

1.1 Configuration

OPIUM is configured through environment variables and specifically the following:

- OPIUM_OKD_URL => The url of the OKD instance's master, including the scheme and without a trailing /
- OPIUM_OKD_TOKEN => An access token for a service account with *view* and *edit* permissions on the desired project
- OPIUM_PROJECT => Project to be exported
- OPIUM_DEPLOYMENT_CONFIGS => Comma separated (no spaces) list of Deployment Configs to export

1.2 OKD Preparation

In your OKD admin CLI you will need to run the following

```
# create a service account
oc create serviceaccount <account>
# Retrieve the service account's access token (set this to OPIUM_OKD_TOKEN)
oc serviceaccounts get-token <account>
# Give the service account the required permissions to the desired project
```

(continues on next page)

(continued from previous page)

```
oc policy add-role-to-user view system:serviceaccount:<project>:<account>
oc policy add-role-to-user edit system:serviceaccount:<project>:<account>
```

1.3 Execution

To start OPIUM (after you've set the configuration environment variables appropriately) simly run:

```
opium
```

This will spawn an HTTP server listening on your system's public interface and on port 8080.

Configuration for the listening interface, as well as a containerized version of OPIUM will follow in later versions.

1.4 Features

- Gather the response of */metrics* from all the pods of the specified *deployment_config*
- Serve them as one response

1.5 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

This repository uses the following as a preview image (under CC BY-SA 2.0): "July 19th. Opium Poppy" by amandabhslater is licensed with CC BY-SA 2.0. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/2.0/

CHAPTER TWO

INSTALLATION

2.1 Stable release

To install opium, run this command in your terminal:

```
$ pip install opium
```

This is the preferred method to install opium, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for opium can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/panagiks/opium
```

Or download the tarball:

```
$ curl -OJL https://github.com/panagiks/opium/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use opium in a project:

```
import opium
```

CHAPTER FOUR

OPIUM

4.1 opium package

4.1.1 Submodules

4.1.2 opium.business module

opium.business.**gen_metrics_jobs**(*settings*, *session*, *dc_pods*)

async opium.business.**get_annotated_pod_metrics**(**args*, *pod*, ***kwargs*)

4.1.3 opium.config module

4.1.4 opium.okd module

async opium.okd.**get_pod_proxy**(*session*, *base_url*, *project*, *name*, *path=''*)

async opium.okd.**get_pods**(*session*, *base_url*, *project*)

4.1.5 opium.opium module

4.1.6 Module contents

Top-level package for opium.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/panagiks/opium/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

opium could always use more documentation, whether as part of the official opium docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/panagiks/opium/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *opium* for local development.

1. Fork the *opium* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/opium.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv opium
$ cd opium/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 opium tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/panagiks/opium/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_opium
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER SIX

CREDITS

6.1 Development Lead

- Kolokotronis Panagiotis <panagiks@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.0 (2020-10-25)

- First release on PyPI.

CHAPTER EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX