Spices Documentation

Release 0.0.3

Luis Alejandro Martínez Faneyth

May 02, 2022

CONTENTS

Spices is an application that generates a Module Index from the Python Package Index (PyPI) and also from various versions of the Python Standard Library.

Spices generates a configurable index written in `JSON` format that serves as a database for applications like spices. It can be configured to process only a range of packages (by initial letter) and to have memory, time or log size limits. It basically aims to mimic what the Contents file means for a Debian based package repository, but for the Python Package Index.

This repository stores the application. The actual index lives in a different repository and is rebuilt weekly via Github Actions.

- Free software: GPL-3
- Documentation: https://spices.readthedocs.org

CHAPTER ONE

INSTALLATION

1.1 Stable release

To install Spices, run this command in your terminal:

```
$ pip install spices
```

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

1.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/LuisAlejandro/spices
```

Or download the tarball:

```
$ curl  -OL https://github.com/LuisAlejandro/spices/tarball/master
```

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

```
$ pip install .
```

CHAPTER

TWO

USING THE APPLICATION

Spices is divided in several commands.

2.1 spices pypi

This command generates a JSON module index with information from PyPI. Read below for more information on how to use it:

```
$ spices pypi --help

usage: spices pypi [options]

General Options:
  -V, --version         Print version and exit.
  -h, --help            Show this help message and exit.

Pypi Options:
  -l <level>, --loglevel <level>
                        Logger verbosity level (default: INFO). Must be one
                        of: DEBUG, INFO, WARNING, ERROR or CRITICAL.
  -f <path>, --logfile <path>
                        A path pointing to a file to be used to store logs.
  -o <path>, --outputfile <path>
                        A path pointing to a file that will be used to store
                        the JSON Module Index (required).
  -R <letter/number>, --letter-range <letter/number>
                        An expression representing an alphanumeric range to be
                        used to filter packages from PyPI (default: 0-z). You
                        can use a single alphanumeric character like "0" to
                        process only packages beginning with "0". You can use
                        commas use as a list o dashes to use as an interval.
  -L <size>, --limit-log-size <size>
                        Stop processing if log size exceeds <size> (default:
                        3M).
  -M <size>, --limit-mem <size>
                        Stop processing if process memory exceeds <size>
                        (default: 2G).
  -T <sec>, --limit-time <sec>
                        Stop processing if process time exceeds <sec>
                        (default: 2100).
```

2.2 spices stdlib

This command generates a JSON Module Index from the Python Standard Library. Read below for more information on how to use it:

```
$ spices stdlib --help

usage: spices stdlib [options]

General Options:
  -V, --version         Print version and exit.
  -h, --help            Show this help message and exit.

Stdlib Options:
  -o <path>, --outputfile <path>
                        A path pointing to a file that will be used to store
                        the JSON Module Index (required).
  -p <version>, --pyver <version>
                        Python version to be used for the Standard Library
                        (default: 2.7).
```

2.3 spices stats

This command gathers statistics from the logs generated by the `pypi` command. Read below for more information on how to use it:

```
$ spices stats --help

usage: spices stats [options]

General Options:
  -V, --version         Print version and exit.
  -h, --help            Show this help message and exit.

Stats Options:
  -i <path>, --inputdir <path>
                        A path pointing to a directory containing JSON files
                        generated by the pypi command (required).
  -o <path>, --outputfile <path>
                        A path pointing to a file that will be used to store
                        the statistics (required).
```

2.4 spices errors

This command summarizes errors found in the logs generated by the `pypi` command. Read below for more information on how to use it:

```
$ spices errors --help

usage: spices errors [options]

General Options:
  -V, --version         Print version and exit.
  -h, --help            Show this help message and exit.

Errors Options:
  -i <path>, --inputdir <path>
                        A path pointing to a directory containing JSON files
                        generated by the pypi command (required).
  -o <path>, --outputfile <path>
                        A path pointing to a file that will be used to store
                        the errors (required).
```

2.5 spices merge

This command searches for JSON files generated by the `pypi` or `stdlib` commands and combines them into one. Read below for more information on how to use it:

```
$ spices merge --help

usage: spices merge [options]

General Options:
  -V, --version         Print version and exit.
  -h, --help            Show this help message and exit.

Merge Options:
  -i <path>, --inputdir <path>
                        A path pointing to a directory containing JSON files
                        generated by pypi or stdlib commands (required).
  -o <path>, --outputfile <path>
                        A path pointing to a file that will be used to store
                        the merged JSON files (required).
```

CHAPTER THREE

INTERNAL API

3.1 spices package

`spices` is just black magic.

Spices is a package that studies the codebase of your project in search for internal and external imports. It then discards the imports that are satisfied with internal code or with the standard library and finally searches the PyPIContents index to list which packages satisfy your imports.

3.2 spices.api submodule

3.3 spices.config submodule

3.3.1 spices.config

3.4 spices.core submodule

The spices.common package contains all the basic functions and modules.

This package contains all the functions and modules that can be reused throughout the entire project.

CHAPTER

FOUR

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/LuisAlejandro/spices/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/LuisAlejandro/spices/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 :)

4.2 Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/spices.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 spices
   $ cd spices/
   $ 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 spices
   $ python3 -m unittest -v -f
   $ 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.

4.3 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. Check https://github.com/LuisAlejandro/spices/actions and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_core_logger
$ python -m unittest tests.test_core_utils
```

CHAPTER

FIVE

AUTHORS

5.1 Development Lead

- Luis Alejandro Martínez Faneyth <luis@collagelabs.org>

5.2 Contributors

None yet. Why not be the first?

CHAPTER

SIX

MAINTAINER GUIDE

If you are reading this, you probably forgot how to release a new version. Keep reading.

6.1 Making a new release

1. Start your project with a cookiecutter template.
2. Start your git flow workflow:

```
git flow init
```

3. Create a new milestone in GitHub. Plan the features of your new release. Assign existing bugs to your new milestone.
4. Start a new feature:

```
git flow feature start <feature name>
```

5. Code, code and code. More coding. Mess it up several times. Push to feature branch. Watch Travis go red. Write unit tests. Watch Travis go red again. Don't leave uncommitted changes. 6. Finish your feature:

```
git flow feature finish <feature name>
```

7. Repeat 4-6 for every other feature you have planned for this release.
8. When you're done with the features and ready to publish, start a new release:

```
git flow release start <release number>
```

9. Bump your version (check everything before next step):

```
bumpversion --no-commit <major, minor or patch>
```

10. Update your changelog (edit HISTORY.rst after to customize):

    gitchangelog > HISTORY.rst

11. Commit your changes to version files and changelog:

    git commit -aS -m "Updating Changelog and version."

12. Delete the tag made by bumpversion:

    git tag -d <release number>

13. Finish your release:

    git flow release finish -s -p <release number>

15. Draft a new release in GitHub (based on the new version tag) and include a description. Also pick a codename because it makes you cool.

16. Close the milestone in GitHub.
17. Publish your new version to PyPI:

    make release

18. Write about your new version in your blog. Tweet it, post it on facebook.

6.2 Making a new hotfix

1. Create a new milestone in GitHub. Assign existing bugs to your new milestone.
2. Start a new hotfix:

```
git flow hotfix start <new version>
```

3. Code your hotfix.
4. Bump your version (check everything before next step):

```
bumpversion --no-commit <major, minor or patch>
```

5. Update your changelog (edit HISTORY.rst after to customize):

```
gitchangelog > HISTORY.rst
```

6. Commit your changes to version files and changelog:

```
git commit -aS -m "Updating Changelog and version."
```

7. Delete the tag made by bumpversion:

```
git tag -d <new version>
```

8. Finish your hotfix:

```
git flow hotfix finish -s -p <new version>
```

10. Draft a new release in GitHub (based on the new version tag) and include a description. Don’t change the codename if it is a hotfix.

11. Close the milestone in GitHub.
12. Publish your new version to PyPI:

    make release

13. Write about your new version in your blog. Tweet it, post it on facebook.

CHAPTER

SEVEN

MADE WITH AND

Web luisalejandro.org · GitHub @LuisAlejandro · Twitter @LuisAlejandro

PYTHON MODULE INDEX

INDEX