cookiecutter-ds-docker

Release 0.9.2

Aug 08, 2020

Contents:

A Docker-based Data Science cookiecutter (for myself)

`cookiecutter-ds-docker` is a personalized, Docker-based cookiecutter template repo for Data Science projects. It aims to standardize the common decisions (repo structure, setup, integrations, etc.), which I need to consider for each new project, and hence minimize the (overtly dull) start-up effort for future work.

CHAPTER 1

Quickstart

In a terminal, run the following:

```
cd {base_folder}
cookiecutter gh:sertansenturk/cookiecutter-ds-docker
# follow the on-screen instructions to cut the project
# ...
cd {{ cookiecutter.repo_slug}} # replace repo_slug with what you entered earlier
make
# once the docker stack is running, click the URL starting with
# http://127.0.0.1:8888/?token=... to access JupyterLab
#
# mlflow UI is at http://localhost:5000/
```

1.1 1 Cookiecutter Template

1.1.1 1.1 Setup

`cookiecutter-ds-docker` requires these tools as a prerequisite:

- **docker**
- **homebrew** (Optional for *Mac OSX*)
- **Python cookiecutter**

1.1.1 Installing Docker

Please follow the instructions in the official Docker documentation.

Note: We suggest using *Docker 18.09* or higher.

1.1.2 (Optional) Installing homebrew in Mac OSX

homebrew is the easiest way to install *cookiecutter* in Mac OSX (see below). Please follow the instructions in the official website, if you wish to install *homebrew*.

1.1.3 Installing cookiecutter

cookiecutter is quite straightforward to install in many modern systems. For example, you can install *cookiecutter* in *Debian-based Linux* distributions (e.g., *Ubuntu*) and *Mac OSX* by:

OS	Command
Debian-based Linux	`sudo apt install cookiecutter`
Mac OSX	`brew install cookiecutter`

Please refer to the official cookiecutter documentation for alternatives.

1.1.2 1.2 Cutting a New Project

To "cut" a new project from the template, run on the terminal:

```
cd /{ base_folder }
cookiecutter https://github.com/sertansenturk/cookiecutter-ds-docker
```

cookiecutter will ask you to fill a few variables, namely:

Variable	Explanation	Modifies
repo_name	Name of the repository	Header of `README.md`
repo_slug	Slug of the repository name	Repository folder name, GitHub URL, explanations in the documentation
package_name	Name of the Python package in the project	Python package name, `setup.py`, `tox.ini`, unittests, docker image names, explanations in the documentation
author_name	Name of the authoring person/team/organization	Author name in `setup.py` and the documentation
author_email	E-mail to contact the author	Author e-mail in `setup.py`, `CODE_OF_CONDUCT.md` and the documentation
github_username	Github username	GitHub URL, URLs in `setup.py`, docker image names, explanations in the documentation
description	A short description of the project	Explanations in `setup.py` and the documentation

Afterward, the project will be created in `/{ base_folder }/{{ cookiecutter.repo_slug }}`.

For additional command-line options, please refer to the advanced options in the official cookiecutter documentation.

1.1.3 1.3 Local Usage

Attention: You should clone the repo if you would like to use *cookiecutter-ds-docker* locally or modify the template:

```
git clone https://github.com/sertansenturk/cookiecutter-ds-docker.git
cd cookiecutter-ds-docker
```

Below, we introduce some useful `Makefile` commands to interact with `cookiecutter-ds-docker`. For all available commands, please refer to the help by running:

```
make help
```

1.3.1 Cutting a New Project Locally

You can cut a project by running:

```
make
```

and entering the variables, as *explained above*. The project will be created at `../{{ cookiecutter.repo_slug }}` relative to the `./cookiecutter-ds-docker` folder.

1.3.2 Documentation

The documentation is hosted online at Read the Docs. *Read the Docs* automatically publishes and updates a version for the *master* branch, *dev* branch, and each release in *Github*.

If you would like to build the documentation locally, you need to run:

```
make sphinx-html
```

The above command builds a docker image called `sertansenturk/sphinx`. It then runs a container from the image and renders the documentation using *Sphinx*.

Afterward, you can access the documentation by opening `./docs/_build/html/index.html` on your browser.

To validate the documentation without building, run:

```
make sphinx-html-test
```

1.3.3 Running Tests Locally

You can run the tests with a single command by:

```
make test
```

The above command:

1. Cuts a dummy project and runs all tests inside (See Project Testing)
2. Validates the Sphinx documentation (See *above*)

1.1.4 1.4 Tests in Travis CI

`cookiecutter-ds-docker` has *Travis CI* integration (link), where all of the tests above are run automatically after each push.

Travis CI also generates code coverage reports for the starter Python package (see Python Tests in the Project), which can be viewed on *codecov* (link).

1.2 2 Working with a Project

Attention: We assume you have already cut a project by following the instructions, and you are in the project directory, `/{ base_folder }/{{ cookiecutter.repo_slug }}`.

1.2.1 2.1 Overview

A project cut from `cookiecutter-ds-docker` consists of a docker-compose stack with the services below:

1. A customized Jupyter service with a starter Python package installed. It runs on *Python 3.7*.
2. An mlflow tracking server to log experiments.
3. A postgresql database, which stores *mlflow* tracking information.

We mount several folders from our host to these services:

- The project base folder, `./`, is mounted on the *Jupyter docker* container so that all modifications are synchronized immediately.
- The folder, `./data/artifacts`, where the artifacts logged by *mlflow* are stored by default, is mounted on the *Jupyter* and *mlflow* services.
- The *postgresql data folder*, `/var/lib/postgresql/data` inside the container, is mounted locally on `./data/db/` to keep the database intact, after stopping the stack.

Note: The project also includes these supplementary, standalone *Docker* images:

1. for building Sphinx documentation (See *Documentation*)
2. for testing *Python* code (See *Python Tests*)

2.1.1 Makefile

`Makefile` commands are used extensively to interact with the project. For a list of commands, please refer to the help by running on the terminal:

```
make help
```

2.1.2 Python development

The project comes with a Python starter package called `{{ cookiecutter.package_name }}`, which is located at `./src/`. The package is `pip` installed to the *Jupyter docker* service in **editable** mode, while *the Docker stack is being built*.

1.2.2 2.2 Setup

If you want to build the stack from the cut project without starting it, run:

```
make build
```

The above command will build these images:

Service	Image name
jupyter	`{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}/jupyter:0.1.0`
mlflow	`{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}/mlflow:0.1.0`
postgres	`{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}/postgres:0.1.0`

Note: The version tag of docker images in a new project starts from `0.1.0`, which is read from the `VERSION` variable in `.env` file.

If you need to make a clean start:

```
make clean-all
```

1.2.3 2.3 Running the Docker Stack

To build and run the Docker stack in a cut project, run:

```
make
```

For convenience, the above command stops running stacks (if exist), cleans, (re)builds, and starts the services.

Note: Accessing Jupyter UI

Once the stack is up and running, you will see a link on the terminal, e.g., `http://127.0.0.1:8888/?token=3c321...`, which you can follow to access the *JupyterLab* interface from your browser.

Note: Accessing mlflow UI

You can reach the *mlflow* UI at `http://localhost:5000`. For a simple example on how to track a run, please refer to notebooks/mlflow_example.ipynb.

For in-depth tutorials, please refer to the official mlflow documentation.

2.3.1 Additional Run Options

By default, the *Jupyter* service is based on the official scipy-notebook image. You can also build & run from tensorflow or pyspark notebooks by:

```
make tensorflow
make pyspark
```

If you want to use classic *Jupyter* notebooks, run instead:

```
make notebook
```

1.2.4 2.4 Documentation

The project comes with basic documentation, which is located at `{{ cookiecutter.repo_slug }}/docs`. You can use Sphinx to build the documentation locally by running:

```
make sphinx-html
```

The above command builds a docker image called `{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}/sphinx`. It then starts a container from the image and renders the documentation (including automatic Python API documentation from docstrings).

Afterward, you can access the documentation by opening `./docs/_build/html/index.html` on your browser.

Note: By default, `{{ cookiecutter.package_name }}` follows the numpy docstring style. If you would like to use Google style docstrings instead, please reverse the `napoleon_google_docstring` and `napoleon_numpy_docstring` variables inside `{{ cookiecutter.repo_slug }}/docs/conf.py`.

1.2.5 2.5 Testing

2.5.1 Python

Build, code style, linting checks and unittests of the starter Python package are automated using `tox` in a docker environment. You can run these tests by:

```
make tox
```

This command builds a *docker* image called `{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}/python-dev`. It then starts a container from the image and runs the Python tests.

2.5.2 Docker Stack

You can test the integration of the Docker services (e.g., sending log requests to *mlflow tracking server* from the *Jupyter* service) automatically by running the *docker-compose* stack in "test" mode by executing:

```
make test
```

2.5.3 Documentation

To validate the documentation without building, run:

```
make sphinx-html-test
```

1.2.6 2.6 Online Services

2.6.1 Github

Github is a popular code hosting platform with (git) version control (and many other complementary services).

To host the project in *Github*, follow the steps below:

1. Create an **empty** repository (**do not** initialize *readme*, *license*, or *.gitignore* files). See the official Github documentation for detailed instructions.

   Note: Your *Github Username* and *Repository Name* should match `{{ cookiecutter.github_username }}` and `{{ cookiecutter.repo_slug }}`, respectively.

2. Initialize git and make the first commit, e.g.:

   ```
   git init
   git add .
   git commit -m "First commit"
   ```

3. Push the project to *Github*, e.g. using *https* connection:

   ```
   git remote add origin https://github.com/{{ cookiecutter.github_username }}/{{␣
   ↪cookiecutter.repo_slug }}.git
   git push -u origin master
   ```

For more information on the *Github ecosystem*, please refer to the official help and guides.

2.6.2 Travis CI

Travis CI is a continuous integration service to build and test projects hosted in *Github*. The project comes with a pre-made *Travis CI* configuration located at `.travis.yml`.

Important: You need to *host the project in Github* to use Travis CI.

Please follow the official Travis CI documentation for instructions to grant *Travis CI* access to the repository.

Once enabled, Travis CI runs *all of the tests mentioned above* automatically after each push. You can view the results at:

`https://travis-ci.com/github/{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}`

Travis CI also generates code coverage reports for the starter Python package, which can be viewed at *codecov*:

`https://codecov.io/gh/{{ cookiecutter.github_username }}/{{ cookiecutter.repo_slug }}`

Note: Please refer to the official guide to how to quick-start and use *codecov*.

2.6.3 Online Documentation

You may want to host the *Sphinx documentation* online, e.g. at Read the Docs or Github Pages. Typically, these services offer effortless integration with *Github*. Please refer to these services to learn how.

Note: We assume that you will host the documentation at `https://{{ cookiecutter.repo_slug}}.readthedocs.io`. Please modify the URLs in the project `README` and documentation, if you would like to host it elsewhere.

1.3 3 Licenses

- The source code is licensed under Affero GPL version 3.
- Any data (features, models, figures, results, documentation, etc.) are licensed under Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

1.4 4 Known Issues

1. Issue #56 (**no_fix**): In Linux, if `DATA_DIR` (defined in the {{ cookiecutter.repo_slug }}/.env) is in a drive formatted in NTFS, you might have permission issues when mounting the folder to the docker containers. We suggest you to cut the project into a drive, which is formatted in a native Linux filesystem.

1.5 5 Credits

1.5.1 5.1 Authors

Sertan Şentürk - contact@sertansenturk.com

1.5.2 5.2 Contributors

Halil Erdoğan - herdoganturkey@gmail.com

- Mac OSX integration (Issue #28, Pull Request #31)
- Animated GIF for Quickstart (PR #48)

jralfonsog - jralfonsog@gmail.com

- NTFS mount permission problem in Linux (Issue #56)

1.5.3 5.3 Contributor Code of Conduct

5.3.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

5.3.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

5.3.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

5.3.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

5.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at contact@sertansenturk.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately. Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

5.3.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html.

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

1.6 6 Changelog

1.6.1 v0.9.2

- Document permission issue when mounting NTFS drives in Linux (Pull Request #55)

1.6.2 v0.9.1

- Fix pip install Python package in editable mode
- *make clean-python* cleans *ipynb_checkpoints* folder(s)

1.6.3 v0.9.0

- Add Sphinx documentation (Pull Requests #36, #41, #45, #46, #49)
- Create online documentation at Read the Docs
- Create Sphinx docker image for local build and test (Pull Request #43)
- Create cookiecutter docker image for local development (Pull Request #39)
- Add VERSION file to the base folder
- Add LICENSE file to the base folder (Pull Request #33)
- Add `step` parameter to the `mlflow.log_metrics` test case and jupyter demo

1.6.4 v0.8.1

- Fix docker stack start-up failure in Mac OSX (Pull Request #31)

1.6.5 v0.8.0

- Increment `mlflow` version to 1.8.*
- Add Github issue and PR templates into the cookiecutter project
- Simplify Jupyter multi-stage builds
- Update `cookiecutter` installation instructions in the `README.md`

1.6.6 v0.7.0

- Convert the repo into a cookiecutter template
- Rename the repo from *ds-template* to cookiecutter-ds-docker
- Fix a bug in `make test` where `mlflow` and `postgres` containers do not stop after testing
- Add maintainer and description related fields to `setup.py`

1.6.7 v0.6.0

- Deprecate static Jupyter and no-cache builds
- Add git to python-dev docker image

1.6.8 v0.5.0

- Add `scipy`, `tensorflow` and `pyspark` base image options from Jupyter docker stack
- Add pull request template

1.6.9 v0.4.0

- Enable JupyterLab
- Pass username and id to the Jupyter service from the host
- Build and run improvements
- Add code of conduct

1.6.10 v0.3.0

- Add travis.ci and codecov integration

1.6.11 v0.2.0

- Add Python template repo and Python `tox` automation
- Share host user uid & gid with the Postgres user in docker

1.6.12 v0.1.0

- Create an initial docker-compose stack with `Jupyter`, `mlflow` and `postgresql` images