Best practices for documenting a scientific Python project

Gavin M. Wiggins, Gregory Cage, Robert Smith, Seth Hitefield, Marshall McDonnell,
Lance Drane, Jesse McGaha, Michael Brim, Mark Abraham, Richard Archibald
and Addi Malviya-Thakur
Oak Ridge National Laboratory, Oak Ridge, TN USA

Abstract—**Documentation is a crucial component of software development that helps users with installation and usage of the software. Documentation also helps onboard new developers to a software project with contributing guidelines and API information. Without the right tools, generating documentation can be laborious and distract from code development. The INTERSECT SDK project is an open federated hardware/software library to facilitate the development of autonomous laboratories. A documentation strategy using Sphinx has been utilized to help developers contribute to the source code and help users understand the INTERSECT SDK Python interface. Docstrings as well as reStructuredText files are used by Sphinx to generate HTML and PDF files, which can be hosted online as API documentation and user guides. The resulting documentation website is automatically built and deployed using GitLab runners to create Docker containers with NGINX servers. The approach discussed in this paper to automatically deploy documentation for a Python project can improve the user and developer experience for many scientific projects.**

Index Terms—**python, documentation, style, formatting, linting**

I. Introduction

Documentation is a crucial component of software development and is considered good practice for scientific computing [1]. For users, it communicates the purpose of the software, as well as instructions on how to install and use it. Documentation can help developers onboard new developers by providing contributing guidelines and API information. However, generating and maintaining documentation can be a burdensome task for developers and distract from code development.

In this paper, we develop a strategy for providing continuous deployment of documentation for the INTERSECT SDK Python library to help alleviate tedious documentation tasks. The INTERSECT SDK project [3], part of Oak Ridge National Laboratory's Interconnected Science Initiative [2], is an open hardware/software library for autonomous laboratories. It utilizes a system of systems and microservices architecture [4], facilitating seamless collaboration and efficient data sharing among labs. Clear and easily accessed documentation of its Application Programming Interface (API) is a necessity for scientific users. We describe a set of coding practices which provide up-to-date documentation via continuous deployment to automatically build websites based on source code and docstrings. The goal of this approach is to allow developers to focus on code development instead of writing and maintaining documentation while adhering to best practices for Python programming.

II. Coding style

Style guides, linters, and formatters help developers maintain a coherent code base that can be easily documented. Also, "clean code" can be enforced via automatic application of these tools via continuous integration workflows. The sections below describe the tools utilized by INTERSECT SDK for code development and documentation.

A. General style

The PEP 8 Style Guide is adhered to for general Python coding conventions [5]. A style guide defines naming conventions, line length, indentation, and other recommendations for code quality and readability. By following the best practices outlined in the PEP 8 style guide, developers can more easily maintain code because it is readable and well commented.

B. Linting and formatting

To enforce a coding style such as PEP 8, the *flake8* linter is used via command-line or as an extension/plug-in for code editors and IDEs [6]. A linter improves code quality by issuing warnings and errors for unused imports and variables, misaligned indentations, missing docstrings, and deviations from the style guide. In addition to the linter, a formatter tool named *black* ensures consistent code formatting throughout the project [7]. The linter and formatter tools are implemented in Continuous Integration (CI) workflows to enforce well-written production code; for example, the CI uses the linter and formatter results to check a pull request for code style and formatting before it can be merged to a production branch.

C. Docstrings

Docstrings are comments in Python code that help users and developers with documentation. The NumPy style guide [11] is often used for scientific projects for formatting docstrings in classes, functions, and modules to provide clear communication of the usage, effects, inputs, and outputs of the code. The

Notice: This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive, paid up, irrevocable, world-wide license to publish or reproduce the published form of the manuscript, or allow others to do so, for U.S. Government purposes. The DOE will provide public access to these results in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Presented at USRSE2023, Chicago, IL, October 16-18, 2023.

Google style guide [12] is another format commonly used for Python docstrings. The INTERSECT SDK developers chose the Google style because of its more compact design compared to the NumPy style. The use of a standardized format ensures that the library's documentation is consistent across the entire API. The Sphinx tool uses these docstrings to automatically generate HTML and PDF documentation.

III. Generating documentation

Tools such as Doxygen [13], pdoc [14], and Sphinx [15] can generate documentation for Python projects. However, Sphinx is preferred because it is the most comprehensive (and recommended) documentation generator for Python, it supports the NumPy and Google docstring styles, and has many professional HTML themes. Sphinx uses reStructuredText (rst) or Markdown (md) files to automatically generate HTML to publish documentation online. While Sphinx can be used for self-hosted documentation, services such as Read the Docs [16] will automatically build and host the documentation. The INTERSECT SDK project uses Sphinx to automatically create API documentation from Python docstrings which allows developers to write documentation directly in the source code. Without a tool like Sphinx, developers would have to write API documentation twice: as docstrings in the code and as external documentation pages. Such a manual approach is time intensive and a burden to maintain.

The tree diagram shown in Figure 1 demonstrates a typical project structure for a Python package. The source code for the package is located in the `src` directory, while the documentation files are in the `docs` directory. Sphinx uses the files in the `docs` directory to build the documentation related to the Python package. Notice the `func.rst` file which is used by Sphinx to automatically generate the API documentation for the code contained in `func.py`. This project structure is utilized by INTERSECT SDK but it should be applicable to many scientific code bases.

```
my_project/
|-- docs/
|   |-- index.rst
|   |-- conf.py
|   |-- func.rst
|-- src/
|   |-- my_package/
|       |-- __init__.py
|       |-- func.py
|-- tests/
|-- pyproject.toml
|-- LICENSE.md
|-- README.md
```

Figure 1. Basic structure of a Python project with Sphinx documentation.

All of the API generation is achieved through the Sphinx autodoc extension [17]. This extension requires a level of initial setup, but afterward automatically picks up changes within different files in the project it is being used in. For INTERSECT SDK, this setup was the creation of various rst files that correspond to the various Python files in the project. These rst files are placed inside a dedicated documentation directory for organization.

The rst files contain configurations telling the autodoc extension which class this file should document, the descriptions that will appear alongside the documentation, and statements that control exactly which parts of the Python code will be included. After all the rst files are created, by running Sphinx along with its autodoc extension, documentation pages are created for each code file. Since Sphinx can build the documentation locally, developers can easily review their changes before pushing their documentation to production. Users can provide feedback on published documentation by submitting labeled issues on the repository. By integrating Sphinx into our continuous deployment, we are able to automatically generate the documentation without manual intervention from developers.

IV. Continuous deployment

The INTERSECT SDK project uses an internal GitLab instance for managing the code repositories. GitLab runners start build jobs on each commit to the development branches. Docker [8] containers are built containing NGINX [9] web servers, and saved to the GitLab Container Registry. Sphinx is run to generate the HTML pages to be served by NGINX. Helm charts [18] are published and pushed to the GitLab Package Registry; they are configured in a dedicated `chart` directory. Once the containers and charts have been built, a GitLab trigger can start a job in a dedicated deployments repository. This deployments repository utilizes the Helm umbrella chart concept [19] to deploy an entire configurable INTERSECT instance made up of many individual Helm charts. These individual Helm Charts and the umbrella chart can be utilized on any Kubernetes [10] cluster. This allows for reuse across any cloud platform that supports managed Kubernetes or on "bare metal" installs of Kubernetes.

V. Conclusion

We have described a set of best practices that produce legible and up-to-date documentation for a scientific Python project. This strategy is easily generalized to other scientific libraries, as there is a one-time cost associated with creating the deployment pipeline to stand up the web server. Once that is completed, merely adhering to good code commenting and styling practices is sufficient to maintain the library's documentation. Thus, the strategy can be easily applied to any code base written in Python. The resulting documentation is easily accessible and readable by the API's users and developers. This approach also creates a PDF of the documentation for offline viewing; therefore, eliminating the need for developers to write separate documentation in Microsoft Word or LaTeX. Overall, the implementation of these best practices not only enhances the accessibility and readability of the documentation for a scientific Python project like INTERSECT SDK but also significantly impacts the wider adoption and long-term value of the software within the scientific community.

REFERENCES

[1] Greg Wilson et al. Good enough practices in scientific computing. PLOS Computational Biology, vol. 13, pp. 1-20, 2017. https://doi.org/10.1371/journal.pcbi.1005510

[2] Oak Ridge National Laboratory, "Interconnected science ecosystem website," https://www.ornl.gov/intersect, 2023, (Accessed on 05/02/2023).

[3] Thakur, A.M. et al. (2022). Towards a Software Development Framework for Interconnected Science Ecosystems. In: Accelerating Science and Engineering Discoveries Through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation. SMC 2022. https://doi.org/10.1007/978-3-031-23606-8_13

[4] Engelmann, C. et al.(2022). The intersect open federated architecture for the laboratory of the future," in Accelerating Science and Engineering Discoveries Through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation. SMC 2022.

[5] PEP 8 Style Guide for Python Code. https://peps.python.org/pep-0008/. Accessed May 3, 2023.

[6] Flake8: Your Tool For Style Guide Enforcement. Flake 8 v6.0 documentation. https://flake8.pycqa.org/en/latest/. Accessed May 3, 2023.

[7] Black: The Uncompromising Code Formatter. Black 23.3 documentation. https://black.readthedocs.io/en/stable/. Accessed May 8, 2023.

[8] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, vol. 239, pp. 2, March 2014.

[9] Nginx documentation. https://nginx.org/en/docs/. Accessed May 15, 2023.

[10] Kubernetes for automating deployment, scaling, and management of containerized applications. https://kubernetes.io. Accessed May 15, 2023.

[11] NumPy Style Guide. https://numpydoc.readthedocs.io/en/latest/format.html. Accessed June 30, 2023.

[12] Google Python Style Guide. https://google.github.io/styleguide/pyguide.html. Accessed April 27, 2023.

[13] Doxygen. https://www.doxygen.nl/index.html. Accessed June 30, 2023.

[14] Pdoc API documentation for Python. https://pdoc.dev. Accessed June 30, 2023.

[15] Sphinx Python Documentation Generator. https://www.sphinx-doc.org/en/master/. Accessed May 3, 2023.

[16] Read the Docs. https://readthedocs.org. Accessed May 3, 2023.

[17] Autodoc extension for the Sphinx documentation generator. https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html. Accessed May 4, 2023

[18] Helm Charts for Kubernetes Package Management. https://helm.sh/, Accessed May 9, 2023.

[19] Helm Umbrella Chart Concept from *"Tips and Tricks: Complex Charts with Many Dependencies"* Helm Website. https://helm.sh/docs/howto/charts_tips_and_tricks/#complex-charts-with-many-dependencies. Accessed May 9, 2023.