CENTER FOR RESEARCH IN OPEN SOURCE SOFTWARE

SkyhookDM - Port wiki to ReadTheDocs or another documentation platform

Proposal for Google Summer of Code 2021 by Yash Jipkate

—

Introduction

Problem Statement

Skyhook Data Management is programmable storage for databases. Every software needs documentation that is easily accessible and easy to use/refer to. Being Open Source means that developers from anywhere are welcome to contribute. Strong and easy-to-use documentation encourages developers to start contributing right away. The documentation also needs to be in line with the Arrow documentation since we want to upstream it. The current documentation is hosted on GitHub wiki, which is not very intuitive to use. We want to port the documentation to Arrow's ReadTheDocs to have a better user experience and interface. There are some sections that might need some revising to make it better. The documentation structure also would need to be reorganized in light of the migration.

Current Status of the Project

Part of the current documentation of the project is hosted on the GitHub wiki of the outdated skyhookdm-ceph repository. A portion of the documentation is also present in the arrow-rados-cls/ directory. Some part of it is also present in the SkyhookDM website.

Brief Description of my Solution

I would first finalize the content and structure of the documentation with the community. The documentation would need to be in reStructuredText since Apache Arrow uses it. This can include some documentation from the wiki that is refactored to reStructuredText from Markdown. The documentation must be comprehensive and easy to understand for new contributors. Docstrings would also be added to the files so that the docs can generate the API references. These docstrings also provide granular support to the developer for understanding the lines of code.

Project goals

Project objectives

- Finalize the structure and contents of the documentation.
- Port the documentation to ReadTheDocs platform.

Expected Deliverables

- The contents and structure of the documentation are finalized.
- The documentation is written in reStructuredText in line with Arrow's documentation and ported to Arrow's ReadTheDocs documentation ready to be upstreamed.
- The docstrings of various APIs are documented.

Future work based on Project

Future work on this project can include updates to the documentation based on the changes in the code.

Implementation

Project methodology

- The first part requires that the documentation content is finalized. The documentation must conform to the Apache Arrow documentation style. The structure of the content also needs to be finalized as it will show up in the Sphinx documentation and will impact how the developers navigate through the documentation and hence needs to be logically structured.

- After deciding upon the contents and structure of the documentation, it needs to be written in reStructuredText and integrated with Arrow's documentation.
- Finally, docstrings would be added which will complement the documentation and make it more comprehensive.

Project technical elements

Apache Arrow's documentation uses the reStructuredText format to build its Sphinx documentation in ReadTheDocs, so all of our documentation will have to be in reStructuredText, involving heavy usage of the format.

- **reStructuredText** is a plaintext markup syntax and parser system and can be used as an alternative to the markdown syntax. The main advantage of RST over markdown is that we can use RST to document code in-line, which can be easily shown up in API docs.
- Another useful tool is **Sphinx** which is a great tool to build HTML webpages using RST files with a beautiful design, allowing the developers to concentrate more on the documentation rather than its design.
- **ReadTheDocs** is an amazing platform for hosting Sphinx-generated webpages. ReadTheDocs is specifically made for technical documentation and is used by many software projects. It has various features like versioning and documentation search provided right out of the box.

The documentation will have to be structured in a way so as to be integrated with the upstream Arrow ReadTheDocs. The code can be categorized into three parts - the client-side C++, client-side Python, and the storage side. The corresponding documentation needs to be placed under appropriate sections of the Arrow docs. The sections need more details to be able to be understood easily by a new contributor. Following are some additions to the documentation that would greatly improve the coverage of the docs and make SkyhookDM easy to use for a new contributor:

- The client-side Python documentation is missing the *_rados.pyx* file, the *rados.py* file section only describes what is contained in the file, but the constituent `SplittedParquetWriter` class isn't documented, as any of its member functions.

- The client-side C++ documentation misses the *.cc* files. The other files have only a brief description of what they consist of. Full API documentation needs to be created and all the constituent functions and classes need to be properly commented on.

- The storage side section gives brief information about *cls_arrow.cc* but the constituent classes and member functions aren't documented.

These additions would increase the coverage of the documentation and give the developer a thorough understanding of the SkyhookDM code. In addition to this, the docstrings in the code are automatically picked up by Sphinx which then shows up in the generated documentation and is shown by the Arrow docs in the *API Reference* section of the corresponding language section.

Proposed solutions

- The docstrings for the files can be added by following the style of the Arrow docstrings so the generated *API reference* docs are consistent with the existing ones, by consulting the community and the mentors on it. Similarly, the documentation of the files can be decided upon. A draft content will be proposed by me and the mentors can give their feedback on the technical details and correctness of the documentation. The documentation will need to be in reStructuredText format as is specified by the Arrow docs styles. Some improvements that I'll incorporate in the documentation are:
    - The existing documentation about the files is very brief. More details would be added.
        - *cpp/src/arrow/dataset/file_rados_parquet.h*: This file contains definitions of 3 APIs which are RadosCluster, DirectObjectAccess, and the RadosParquetFileFormat API. Dedicated subsections for these APIs would be very helpful, containing an explanation of the APIs with some sample usage examples to help understand their functions and usage in a more beginner-friendly way.
        - *cpp/src/arrow/dataset/rados.h*: Although we assume the developer already knows about the 'librados SDK', having a link to its documentation or having a brief introduction of it in a sidenote would be helpful. This would also help to give a background context of the methods exposed by the wrapper.
        - *cpp/src/arrow/dataset/rados_utils.h*: In a similar fashion, here we can provide a link to the '`ceph::bufferlist`' and add more details of the utility functions provided and some sample use cases for some of them.
        - *python/pyarrow/_rados.pyx/_rados.pxd*: Since this would be part of the *Python bindings* section, separate from the *C++ implementation* section which contains the `RadosParquetFileFormat` C++ API, we could provide a link to the API, to give context to a Python user.

        - *python/pyarrow/rados.py*: This could use some more details about the features and functions of `SplittedParquetWriter`, and possibly some sample use cases.

        - *cpp/src/arrow/adapters/arrow-rados-cls/cls_arrow.cc*: I will be adding more details about the objclass functions and the APIs implemented in the file and the RandomAccessObject API. I would organize them in subsections for better readability.

    - There are files that do not have a mention in the existing docs such as the *_rados.pyx* file. Documentation of such files would be written from scratch.

    - Docstrings are missing on almost all of the files. Docstrings would be added, with a brief description of the method/class and the input parameters and return values and their expected types. This would give the developer a thorough understanding of what a method does and what should be provided to it in order to make use of it. Any variable used or complex logic would be accompanied by appropriate comments.

    - Another improvement would be to make the README self-contained. The existing README will be utilized to provide installation and setup instructions, features of SkyhookDM, and a brief introduction to the SkyhookDM project and its community. The 'Installation Instructions' section of the README contains instructions on external links. This can be a bad experience for a developer. The necessary instructions would be extracted to this README itself so that the instructions are centralized. I'll still leave the links in case the developer wants to have more context.

- After the documentation is completed, the documentation goes into relevant sections. The client-side Python documentation goes in the *Python Bindings* section, the client-side C++ and the storage side documentation goes in the *C++ Implementation* section.

- Other documentation like the introduction to SkyhookDM, salient features, and setup instructions would remain in the *arrow-rados-cls* directory, which would serve as the home of SkyhookDM in the Arrow project.

Project Timeline

Project plan and deliverables schedule

1. Milestone 1

Community Bonding - May 17 - June 7

Deliverables	Completed By
Created a draft document with the existing documentation	May 19
Added documentation for the missing files in the draft. Changed some existing content as needed to enhance readability.	May 22
Made changes to the draft document according to the feedback received by mentors	May 26
Converted the final documentation to RST format and added the documentation to their appropriate places in the main Arrow docs	June 7

2. Milestone 2

First Evaluation - June 7 - July 12

Deliverables	Completed By
Added docstrings to the client-side C++ files and made changes suggested by mentors	June 22
Added docstrings to the client-side Python files and made changes suggested by mentors	June 27
Added docstrings to the storage side files and made changes suggested by mentors	July 12

3. Milestone 3

Final Evaluation - July 16 - August 16

This period would serve as a buffer period. Manual testing for quality assurances will be carried out. Feedback from the community if any, and improving based on the feedback. Leftover work, if any, is completed in this period. Additional documentation, if required, can also be added during this period. The project report and a blog describing my work on this project would also be written.

Commitments outside project that might impact work

I don't have my college during the summers, so I won't be having any kind of commitments outside GSoC.

Areas of project you expect to be most/least challenging

The most challenging part of the project will be to decide the structure of the documentation and the content of it, some of which would be imported from the old Wiki and some to be added anew. This would require collaboration with the community and the mentors.

Biographical information

Relevant experience

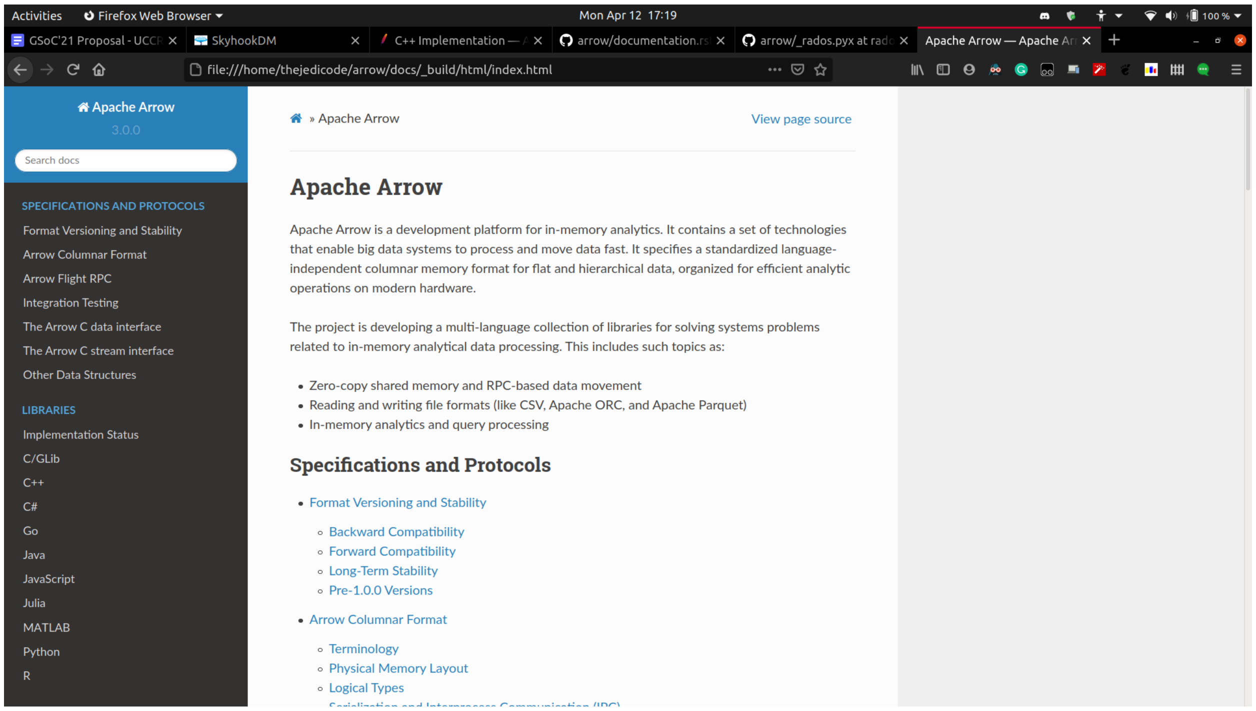
I have previously worked in the direction of automated documentation generation and Static Site Generators.

- My personal portfolio site is made in Next.js, a modern static site generator. It pulls its contents from markdown files residing in the GitHub repository.
- I have also contributed to a documentation repository, namely infusion-docs, where I had the task to migrate the deprecated DocPad framework to a newer 11ty framework (PR).

Apart from these, I have vast experience in Web Development. I have designed and implemented a DRF-powered React website from scratch for a budding startup. I have also developed an Angular portal for a college fest. I have also contributed to the Oppia open source project and have been a core member and project lead there. I also got to document the changes that I made in the code in Oppia. These experiences have made me accustomed to teamwork and quality development.

Previous work

I have successfully built the docs locally using the instructions provided in the Apache Arrow documentation. Here are some screenshots of the built Sphinx docs:



Apache Arrow
3.0.0
Search docs

SPECIFICATIONS AND PROTOCOLS
- Format Versioning and Stability
- Arrow Columnar Format
- Arrow Flight RPC
- Integration Testing
- The Arrow C data interface
- The Arrow C stream interface
- Other Data Structures

LIBRARIES
- Implementation Status
- C/GLib
- C++
- C#
- Go
- Java
- JavaScript
- Julia
- MATLAB
- Python
- R

» Apache Arrow   View page source

Apache Arrow

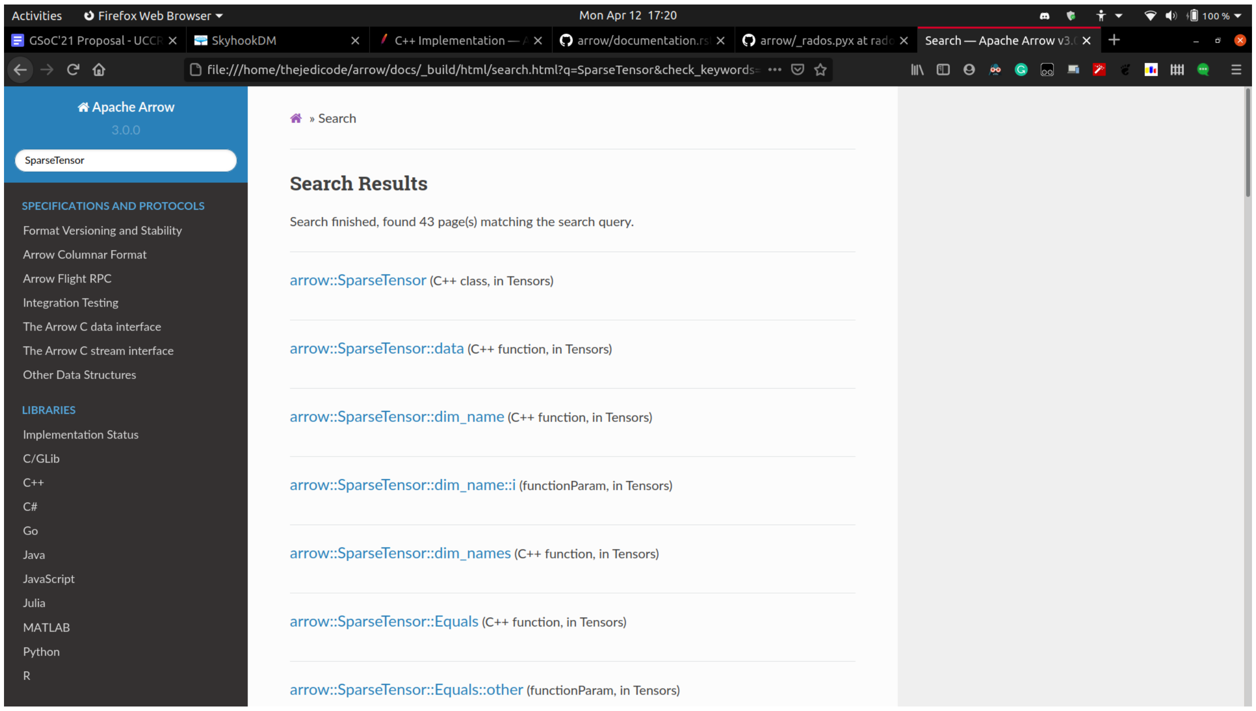
Apache Arrow is a development platform for in-memory analytics. It contains a set of technologies that enable big data systems to process and move data fast. It specifies a standardized language-independent columnar memory format for flat and hierarchical data, organized for efficient analytic operations on modern hardware.

The project is developing a multi-language collection of libraries for solving systems problems related to in-memory analytical data processing. This includes such topics as:

- Zero-copy shared memory and RPC-based data movement
- Reading and writing file formats (like CSV, Apache ORC, and Apache Parquet)
- In-memory analytics and query processing

Specifications and Protocols

- Format Versioning and Stability
  - Backward Compatibility
  - Forward Compatibility
  - Long-Term Stability
  - Pre-1.0.0 Versions
- Arrow Columnar Format
  - Terminology
  - Physical Memory Layout
  - Logical Types



Apache Arrow
3.0.0
SparseTensor

SPECIFICATIONS AND PROTOCOLS
- Format Versioning and Stability
- Arrow Columnar Format
- Arrow Flight RPC
- Integration Testing
- The Arrow C data interface
- The Arrow C stream interface
- Other Data Structures

LIBRARIES
- Implementation Status
- C/GLib
- C++
- C#
- Go
- Java
- JavaScript
- Julia
- MATLAB
- Python
- R

» Search

Search Results

Search finished, found 43 page(s) matching the search query.

arrow::SparseTensor (C++ class, in Tensors)

arrow::SparseTensor::data (C++ function, in Tensors)

arrow::SparseTensor::dim_name (C++ function, in Tensors)

arrow::SparseTensor::dim_name::i (functionParam, in Tensors)

arrow::SparseTensor::dim_names (C++ function, in Tensors)

arrow::SparseTensor::Equals (C++ function, in Tensors)

arrow::SparseTensor::Equals::other (functionParam, in Tensors)

Relevant education background

I am currently a final-year undergraduate at the Indian Institute of Technology (BHU), Varanasi in India with Mechanical Engineering as my major.

Programing interests and strengths

My interest in programming began by attending club workshops and have been attached to the programming club of my college for a long time. I am primarily interested and skilled in Web development, although I am keen to try other areas. I have worked with backend frameworks like DRF, frontend frameworks like Angular and React, and Static Site Generators like 11ty, HUGO, and Next.js. Some of my projects include my personal site built in Next.js (link), my work on the documentation framework migration in infusion-docs (link), my open-source contributions to Oppia (link).

I also have decent experience in competitive programming and hence have a good understanding of various data structures and algorithms. I have a fair experience working alone and solving problems on my own so I won't be bugging mentors too much for little problems that can be solved by a Google search. I also have good experience in git workflows and community activities like reviewing and testing others' code.

Contact information

I am open to any kind of communication medium that the mentors prefer. During the course of the GSoC period, I will be in India (UTC +5:30). Following are my contact details:

- GitHub: YashJipkate
- Mail: yashjipkate@gmail.com
- LinkedIn: https://www.linkedin.com/in/yashjipkate
- Discord: YashJipkate#9159