REQUIREMENTS SPECIFICATION

Version 1.0

November 17, 2021

Fossilized Containers

Sponsor: Dr. Nicholas McKay

Mentor: Melissa D. Rose

Team Members: Jadon Fowler, Jeremy Klein,

Emily Ramirez, Mumbi Macheho-Mbuthia

Accepted as baseline requirements for the project:

Client Signature:	**Date:**
_____	_____
Team Signature:	**Date:**
_____	_____

Table of Contents

1. Introduction

Climate change immediately catches attention. The phrase demands space. The Earth is becoming increasingly inhabitable;but yet it remains, constantly worsening, constantly present. From the 19th century to now, the global average temperature has risen 1.18 degrees celsius [1]. While it appears to be negligible, this small change in temperature contributes to increased droughts, heatwaves, wildfires, and more extreme weather conditions. When the public discusses climate change, the focus stays on the present and the looming future. However, **paleoclimatology**, the study of past climates, takes a different approach. By understanding how Earth's climate has changed over the past several thousand years, scientists can predict and prepare for changes in the future.

Paleoclimatologists create paleoclimate reconstructions, **PRs**, or code that creates models of the past [2]. PRs or usually written in the programming languages R or Python. With **PReSto**, Paleoclimate Reconstruction Storehouse, Dr. McKay and collaborators can accept different datasets and models from different researchers and submit them to their system. With the thousands of types of datasets, however, it is difficult for researchers to submit their code to PReSto without an existing standardized way to review them.

Scientists use a multitude of programming languages, libraries, dependencies, and operating systems that are not guaranteed to work on other systems. **Containerization** is a way to package software so that it will be compatible across different host operating systems. It also allows users to test a model without having to install different libraries or dependencies. They simply need to build a container and run it to view the model.

The remainder of this document will define an issue and a solution for this area.

2. Problem Statement

Paleoclimatologists create models as their core business. These models can use different inputs to recreate historic climates. Unfortunately, there is no way to ensure that these models can work on every machine, so many of these models will not.

Every model requires different dependencies, e.g. different libraries that may only run on a specific operating system. It is not realistic to expect for paleoclimatologists to create cross-platform applications of their models. For these models to work on most systems or be cross-platform, there would need to be additional work to ensure that. Thankfully, containerization solves this cross-platform issue.

Containers are interactive snapshots of a system. These snapshots can be sent and used on any system. This is how our sponsor, Dr. McKay, envisions a solution to this problem. Specifically, the issues our client is facing are:

- Creating containers of paleoclimatologists' models
- Guiding paleoclimatologists through containerization of their models
- Scanning a model's source code for input, output, and parameters

PReSto containers, our proposed solution, will allow for scientists to package their models and run it on any system they wish. Right now, PReSto containers run on the assumption that LiPD files serve as input to PRs and NetCDF files will be outputted. LiPD files and NetCDF files are file formats relevant to climate data.

3. Solution Vision

To help our client solve this problem our solution will provide paleoclimatologists a simple way to containerize and collaborate with other climatologists. The climate reconstruction models will be stored in PReSto containers by using the tool that we are creating. The tool will provide the user with an interface and also give them the ability to run the containers on an external service.

The first part of our solution is the PReSto Container. This container will be created by the user with the help of the tool to ensure that all of the dependencies of the program are accounted for. This ensures that when a climatologist wants to share his code with others he can easily create a container that can be run on any machine. The container will also allow it to be stored on the PReSto website once it is complete so that any climatologist can access it.

Along with the PReSto Containers there will be a tool that will aid users in containerizing their climate reconstruction models. The tool has three main functionalities, creating, running, and managing PReSto containers, however it will also be language agnostic. This will allow for one local place for climatologists to convert their programs into something that can be shared and run by anyone that the user sends it to regardless of the language it was written in or the system it was built on.

Inside the tool mentioned above will be a user interface which will guide the user through all the necessary steps and information needed to build a container. The interface will provide documentation for its use on the interface as well as on the teams website. The documentation will include a description of each functionality provided as well as the technical internals in creating the container.

The last part of the solution is the ability to run the containers on an external service. This will allow for container(s) to be run on servers that have much higher computing power than their local machine. Which leads to multiple large models being constructed at the same time on large sets of data without straining the user's computer. This could also potentially allow for the containers to be stored and run on websites making access even easier.

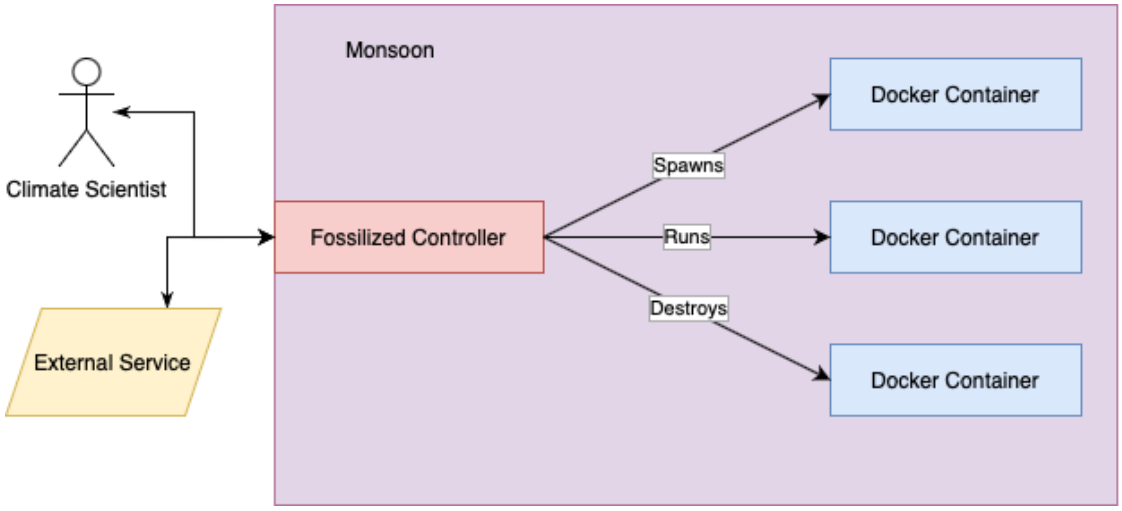


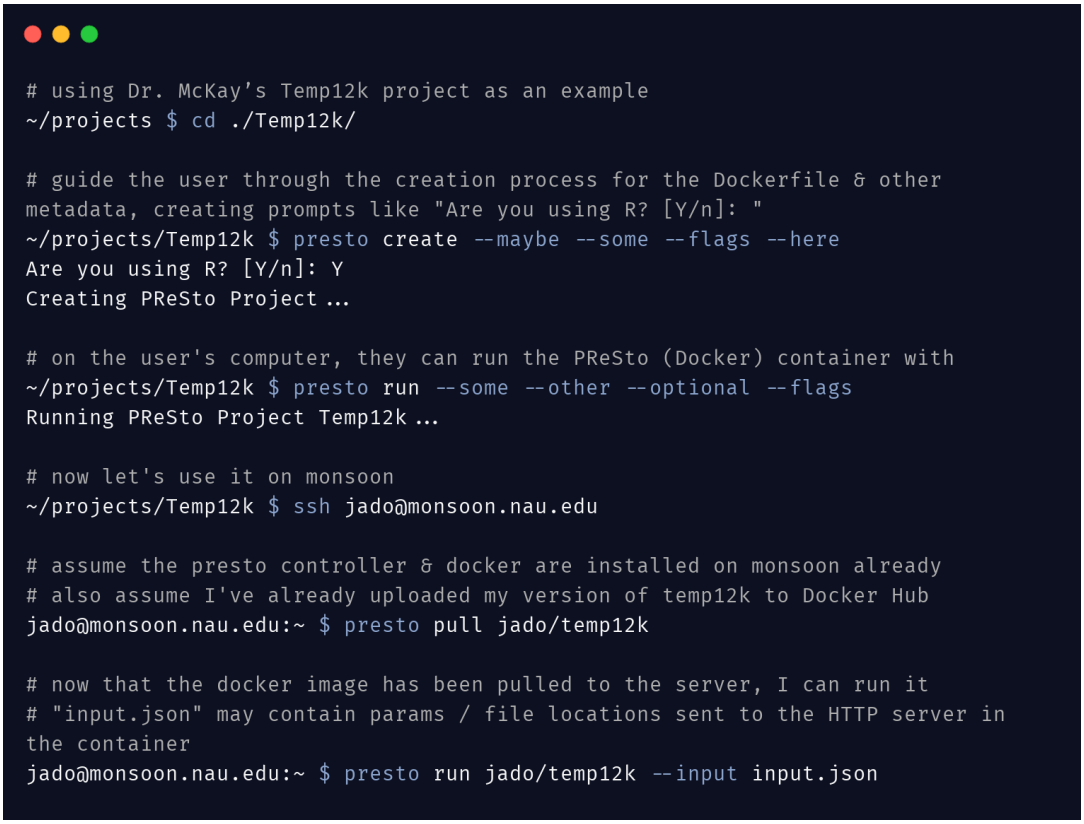
Figure 1: Fully realized Fossilized Controller running on NAU's Monsoon and interacting with Users and External Services

4. Project Requirements

4.1 Functional Requirements

4.1.1 Guiding Users Through Containerization

A number of requirements need to be met to fulfill the team's vision for a solution. The Fossilized Controller needs to guide inexperienced users through making a Docker Container out of their climate reconstruction code and must manage the created containers. The guiding process must be simple to follow and usable for any new or existing projects. The management of these containers entails starting the containers, making it run the code inside, and collecting the outputs. Together, the Fossilized Controller and containers can be used to make climate reconstruction programs more reliable and reproducible.



```
# using Dr. McKay's Temp12k project as an example
~/projects $ cd ./Temp12k/

# guide the user through the creation process for the Dockerfile & other
metadata, creating prompts like "Are you using R? [Y/n]: "
~/projects/Temp12k $ presto create --maybe --some --flags --here
Are you using R? [Y/n]: Y
Creating PReSto Project ...

# on the user's computer, they can run the PReSto (Docker) container with
~/projects/Temp12k $ presto run --some --other --optional --flags
Running PReSto Project Temp12k ...

# now let's use it on monsoon
~/projects/Temp12k $ ssh jado@monsoon.nau.edu

# assume the presto controller & docker are installed on monsoon already
# also assume I've already uploaded my version of temp12k to Docker Hub
jado@monsoon.nau.edu:~ $ presto pull jado/temp12k

# now that the docker image has been pulled to the server, I can run it
# "input.json" may contain params / file locations sent to the HTTP server in
the container
jado@monsoon.nau.edu:~ $ presto run jado/temp12k --input input.json
```

Figure 2: An example run using the Fossilized Controller to run a PReSto Container

4.1.2 Command Line Interface

The Fossilized Controller will be realized by a Command Line Interface (CLI) that users can interact with to initiate the process of building a Docker container. This involves generating a Dockerfile that includes what Operating System the container contains, and what libraries and languages need to be added to run the climate reconstruction code. The CLI will also have functionality to run containers with supplied climate model parameters, input LiPD files, and any other metadata needed to run the reconstruction.

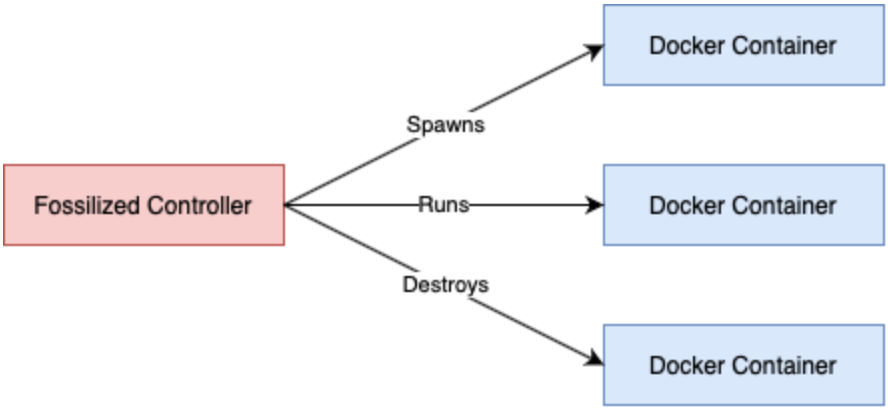


Figure 3: Fossilized Controller managing many Docker Containers.

4.1.3 Communicating With Containers

To communicate with containers, the Fossilized Controller will use HTTP. The Adapter Libraries used within the container will receive these files over HTTP to give to the climate reconstruction program. The Controller will also use the Docker API to expose the HTTP server within the container. This channel of communication will be used for running the programs from external sources, such as a user or miscellaneous tools.

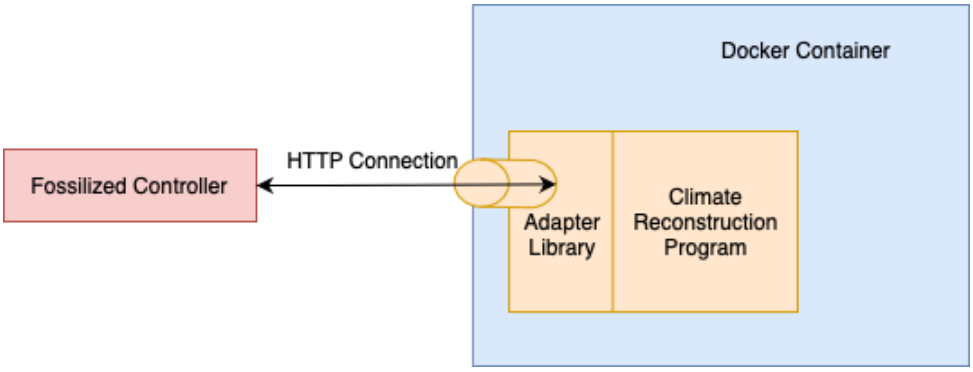


Figure 4: Connection between Fossilized Controller and Docker Container

4.1.4 Adapter Libraries

The Adapter Libraries must also be capable of reading LiPD files and climate model parameters to give to the climate reconstruction program, and will receive the resulting NetCDF files. This output is passed back to the Fossilized Controller.

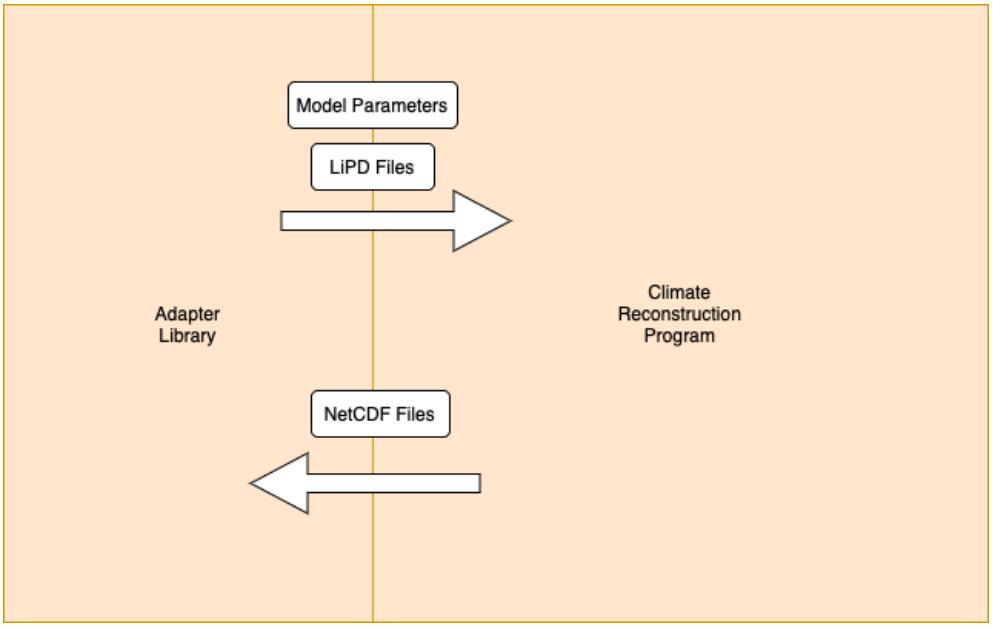


Figure 5: Interactions between the Adapter Library and Climate Reconstruction Program

4.2 Performance Requirements

This section of the document outlines and describes the Performance requirements for the tool that is currently being developed. These measurements are used to measure how the controller is expected to perform when performing tasks. The two performance requirements that are covered in this document are the ability to send and receive large files and the ability to wait for a response from the container as computation can take a long time.

4.2.1 Send Large Files

The first performance requirement is the controller and containers capability to send large files through the HTTP Server. This is because the climate models are often built using several LiPD files that vary in size. For our purposes the controller must be able to handle files of at least 5735 kilobytes, since that is the size of the largest set of LiPD files available. However, the http server and controller will be tested against files of between five thousand kilobytes and potentially to 10 gigabytes.

4.2.2 Appropriate Response Length

The second requirement in this section is increasing the wait time for a response to the appropriate length. After the code receives the LiPD objects it can take tens of minutes for a file to run depending one the size of the files and compute time. Therefore, the controller will wait for a response from the container for 30 minutes before the connection is automatically closed. This allows large containers ample time to run the code and produce a climate model and if a file is received before the wait ends then it is ended automatically, closing the connection.

4.3 Environmental Requirements

The following paragraphs in this section of the document will cover the Environmental Requirements, or non-functional requirements that are required. Along with listing the requirements this section will cover the reasoning behind them as well as the different constraints that they have on the actual project.

4.3.1 Integrating Software With Platforms

One of these environmental requirements is integrating the software with platforms such as Github or DockerHub on individual accounts so that Climate Scientists can update their code or containers easily. This requirement allows for the scientists containerizing their code to be able to store them locally on their machine and

update the code or container at any point in time. Although this allows for easy use of the containers it requires the software to use the Docker API and a language that has a docker API such as Python or Go. Since our team does not have formal experience with Go, we are forced to use Python since the other APIs do not offer good documentation.

4.3.2 Language Agnostic

Another environmental requirement for our project is that it must be language agnostic, which means that the controller must be able to containerize any programming language. Therefore, regardless of what language is being used the tool will still be able to containerize the code and read and write files to it. This requirement causes constraints because it is incredibly difficult to account for all languages and possible dependencies that each language has. It would also require that each possible language has an adaptor library that can start the http server and can read and write the LiPD files.

4.3.3 LiPD Library

The third requirement that constrained the ability for the tool to be agnostic was the use of the LiPD library in order to read and write the file objects to the actual code. Because the code in the container can only receive LiPD files it must use the library provided, which is only available in three languages. Therefore, in order for our tool to be language agnostic there must also be a LiPD library for every language wanting to be containerized. The tool will therefore allow for an adaptor library to be added in the future if a language is being used and is wanting to be uploaded to PReSto.

4.3.4 Compatible Across Different Systems

The last environmental requirement for the tool is that it must be able to be run on different types of machines. The three types of machines that are required and generally the standard are Windows, Mac, and Linux, which means our CLI will need to

be written in a language that can be run on any of those operating systems. This made it a difficult choice in picking a language since it also needed to be able to communicate with docker to create containers.

5. Potential Risks

The CLI produced by the team does not have many risks overall. The main functionality is for scientists to create containers and send files to their climate model.

Risk	Likelihood	Effect
User Installation Issues	**Moderate**	**Moderate**
Corrupted Files	**Low**	**High**
Poor handling of large amount of files	**Low**	**Low**
Deprecated Libraries	**Moderate**	**High**
Poor tool maintenance	**Low**	**High**

Figure 6: Table showing the general overview of the risks involved

5.1 User Installation Issues

Likelihood: Moderate

Effect: Moderate

One of the most important components of the CLI is the ability to create containers with Docker. Unlike other programs used in Linux distributions, Docker is not installed fully from a package manager. There are manual steps that the user needs to take before Docker is up and running. Proper installation is needed considering that Docker is needed for creating containers. If the user can not install Docker properly then they can not use the tool. Such a scenario is detrimental for the tool, however, there are many troubleshooting tips online and multiple installation alternatives. Installation is only a one time thing so users would not have to continuously go through the troubleshooting process. Luckily, the users can easily access the solution for installation issues with the abundance of resources available online for Docker. The

CLI can also direct users to common sources to avoid having the user scour the internet for too long.

Another installation users can run into is using pip, a package manager used with Python, to install the CLI. It is possible that the pip installation can fail on various Linux distributions for unknown reasons at the moment. There is no current indicator if the tool will install properly on different distributions. If a user finds they are having issues, then the team can investigate the issue and deploy a fix.

5.2 Corrupted Files

Likelihood: Low

Effect: High

One of the main features in the CLI is the ability to send files between the Fossilized Controller and container. The controller sends LiPD files while the container sends output NetCDF files in return. It is possible that the files can arrive to either entity corrupted. Chances of such a situation happening are low but should be considered by the time. If a file sent to the container is completely corrupted, then the model will not work properly since it was given an invalid file. Similarly, if a file is corrupted in such a way that the data was only slightly modified then the model would produce incorrect results. Even though there are functions in LiPD libraries to validate, the latter scenario is not something caught in the validators. The effect of corrupted NetCDF compared to corrupt LiPD files is smaller because the user will clearly see if it is corrupt.

5.3 Poor Handling of Large Amount of Files

Likelihood: Low

Effect: Low

As a reconstruction model progresses, a user will need to send a large amount of files to a container. The Fossilized Controller contains a client that will send files to a server on the model's container. As is the issue with any server, it is possible to overwhelm it and cause it to not function properly. Files that are sent to the container are not usually large, however, a scientist with a large amount of data could use the tool and encounter issues. It is also possible that the connection between the Fossilized Controller and the container gets throttled because of the high traffic volume. The likelihood of that happening is low because the team will ensure a

proper connection protocol, however, it can be difficult to test all instances without the correct amount of files. Users could split up the amount of files they send at once to avoid overwhelming the container's server.

5.4 Deprecated Libraries

Likelihood: Moderate

Effect: High

The team is using multiple libraries to implement the features of the Fossilized Controller. The most important ones are the Docker Python library and HTTP libraries in both Python and R. They provide the core functionality needed in order to help scientists create containers and communicate with them. If a library becomes deprecated or they create enough changes, the code developed by the team can break from the changes. If the changes are big enough, the entire Fossilized Controller needs to be changed to account for such changes.

5.5 Poor Maintainability

Likelihood: Low

Effect: High

Considering the project is going to stay open source, there are going to be people who will want to contribute and update the CLI that is meant to aid scientists. The client has expressed that he will want to add on to the tool as they come across new needs. New features can not be added if the project has poor maintainability. The team plans to practice proper coding standards and develop the tool making sure that features can easily be added in the future.

6. Project Plan

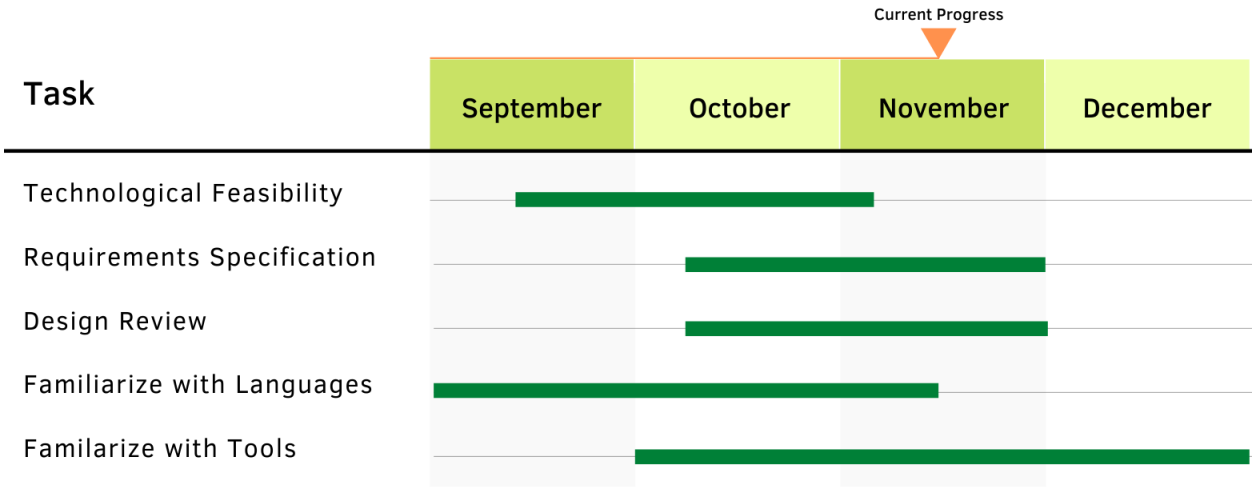


Figure 7: Gantt chart of the Fall 2021 semester

The team has accomplished the major documents and learning phases since the project has been started. While learning the languages and tools required for the project, the team has created a good plan for creating a tool to help scientists containerize their code.

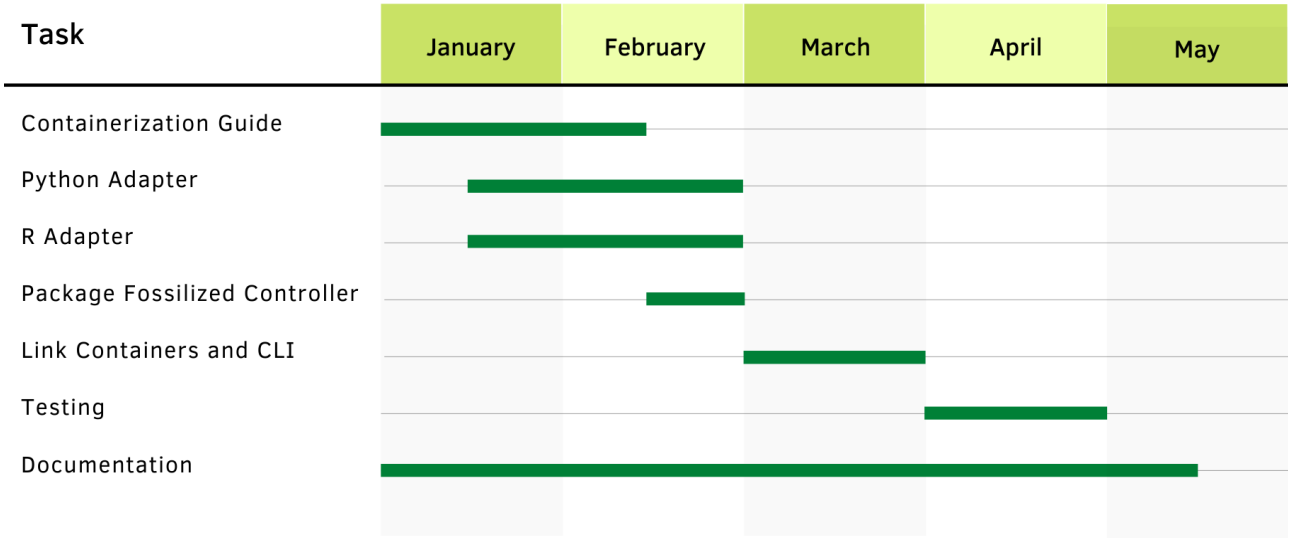


Figure 8: Gantt chart of the Spring 2022 semester

There are about 5 major milestones in the development of the Fossilized Controller as shown in the Gantt chart of Figure 8. Towards the end of the project development, time is allocated to test

and debug the Controller so that it is ready for a quality deployment. The team will also create The major milestones are described in more below:

1. Containerization Guide - The main features of the Fossilized Controller consist of guiding the user on how to containerize their code will start in January and last for about a month and a half. This will possibly be extended as its only dependency is packaging the Fossilized Controller.
2. Python Adapter & R Adapter- The two adapter libraries will begin and end in parallel. They are both separate milestones but are dependencies for linking containers and the Fossilized Controller
3. Package Fossilized Controller - The milestone consists of creating an installable tool that users can use. Development is short because there will be adequate testing in previous milestones. The Fossilized Controller needs to be installable before linking it with the containers with adapter libraries.
4. Link Containers and CLI - The final stage in development. At this point, the project will start reaching its end and testing will follow to ensure a quality product.

7. Conclusion

Paleoclimatologists build models as their main output which help to better understand and combat climate change. These models, code to create representations of the past, are not built with different systems in mind. Containers can help solve issues with using models on different systems. With containers, the problem becomes:

- How to create containers of paleoclimatologists' models?
- How to guide paleoclimatologists through containerization of their models?
- How to scan a model's source code for input, output, and parameters?

The requirements to satisfy these issues are:

- Guiding users through containerization
- A CLI to create and manage PReSto containers also known as the Fossilized Controller
- Communicating with containers via the Fossilized Controller to access the models within the containers

- Python and R adapter libraries to scan a model's source code for input, output, and parameters and to communicate with the model and the Fossilized Controller

The requirements specified in this document completely address all of the sponsor's concerns and initial requests. Fossilized Containers will move on to create a satisfactory demo of the application using the work here. As long as the demo and following minimal viable product satisfies the requirements laid out in this document, it is guaranteed that this project will be a success.

Glossary

Containerization: Packaging software so that it will be compatible across different host operating systems.

Containers:

Paleoclimatology: The study of climate of the past

PR: Paleoclimate reconstruction or code the creates models of the past

PReSto: Paleoclimate Reconstruction Storehouse, a system that allows for the storage of Paleoclimate Reconstructions or PRs

References

[1] "Climate change evidence: How do we know?," *NASA*, 12-Oct-2021. [Online]. Available: https://climate.nasa.gov/evidence/. (Accessed: 18-Oct-2021).

[2] PReSto A paleoclimate reconstruction storehouse. *USC Climate Dynamics*. [Online]. Available: https://climdyn.usc.edu/projects/5_presto/. (Accessed: 16-Nov-2021).