Technological Feasibility Analysis

November 4th, 2019

Canopy - Team 11

Sponsor: Dr. Patrick Jantz
Mentor: Scooter Nowak
Team Members:
Robert Plueger
Nicholas Lopez
Dongyu Xia
Maria Granroth

Table of Contents

1 Introduction

We are Canopy, a 2019 - 2020 Northern Arizona University Computer Science capstone team. Our team members are Robert Plueger (team lead), Maria Granroth, Nicholas Lopez, and Dongyu Xia. Our project is to develop an application for characterizing 3D vegetation structure in tropical ecosystems. We will analyze the data of the three dimensional distribution of stems, branches, and leaves obtained from the Global Ecosystem Dynamics Investigation (GEDI). GEDI is funded by NASA, an Earth Ventures mission that started in December 2018. GEDI will acquire billions of vegetation profiles across the Earth's temperate and tropical ecosystems in two years.

The sponsor of this project is Dr. Patrick Jantz. He is a member of the Vegetation Structure as an Essential Biodiversity Variable (VSEBV) project based at NAU. He aims to use GEDI data to develop a vegetation structure essential biodiversity variables (EBVs) that can be used by policy makers and scientists to improve land use decisions and guide priorities for conservation of biodiversity in tropical landscapes. The reason for launching this project is that current workflow is clumsy and requires a lot of manual work. He hopes an application can help improve processing speed and accuracy while reducing repetition and manual steps. We envision an application that supports two different end users and usage scenarios: Data Ingestion and Analysis, and Data Viewing and Access. Data ingestion and analysis can provide a set of graphical interfaces that streamline the data acquisition and integration process. Data viewing and access can provide clear interfaces for browsing, selecting, graphing, and otherwise visualizing the growing data available in the system.

The purpose of this document is to ensure that the tools and technical ideas we develop in the design phase will be appropriate and function in the final product. The goal of the application is to integrate GEDI data access, data analysis, and data presentation steps, generating easy-to-interpret statistical and visual summaries for users. In *Technological Challenges*, we list the main technical barriers that we found. In the following section, *Technology Analysis*, we will analyze the difficulties we have encountered and our initial ideas to solve these problems. We will then introduce integration challenges in *Technology Integration*. Following, we will conclude our technological feasibility

analysis. In this part, we will summarize the challenges and solutions we have encountered and state our confidence level in implementing our project design.

2 Technological Challenges

2.1 Introduction

This section will outline the seven major technological hurdles we have identified through client and team meetings. This will not include a full design of the project or any minor challenges that we may encounter later. The challenges listed here can be considered high level requirements, and will be summarized by three overall choices that need to be made: programming language, web framework, and web hosting service. The programming language and web hosting service challenges fall under both the Data Ingestion and Data Viewing categories, while the web framework is categorized by Data Viewing.

2.2 Challenges

- Programming language
    - We will need a programming language than can work with large amounts of data quickly.
    - We will need a programming language that can be implemented in a website.
- Web framework
    - We will need a web framework that can help design websites using these languages.
    - We will need a web framework that is efficient and can be easily learned.

- Web hosting service
    - We will need a web hosting service to share our project publicly.
    - We will need a web hosting service that can adaptively work with large amounts of data.
    - We will need a web hosting service that is affordable.

3 Technology Analysis

3.1 Introduction

The challenges introduced in the previous section were summarized by three choices that need to be made: programming language, web framework, and web hosting service. This section will discuss each challenge and describe the thought process behind choosing the solution. Each discussion will be concluded by a plan for proving feasibility.

- Programming language
    - We will need a programming language than can work with large amounts of data quickly.
    - We will need a programming language that can be implemented in a website.
- Web framework
    - We will need a web framework that can help design websites using these languages.
    - We will need a web framework that is efficient and can be easily learned.
- Web hosting service
    - We will need a web hosting service to share our project publicly.
    - We will need a web hosting service that can adaptively work with large amounts of data.
    - We will need a web hosting service that is affordable.

3.2 Selecting a Programming Language

A fundamental challenge is determining the language we will use to develop our project. Our client has expressed his desire to use either Python or R, as he is comfortable with them and wants to be able to maintain the project after our collaboration. This significantly narrows down our potential candidate languages.

3.2.1 Python

Python is one of the most popular languages among developers. According to Github it is listed as the first most popular language with 29.49% usage. According to the TIOBE Index, it is listed as the third most popular language with 9.84% usage. Python is notably used for machine learning, big data, and artificial intelligence. It is a language that all of us on Team Canopy have been exposed to. There is an abundance of Python libraries available to use that could assist us with our project, such as the Fiona, Rasterio, and GDAL/OGR libraries.

3.2.2 R

R is a popular language for statistical computation. According to Github, it is the 7th most popular programming language with 3.82% usage. According to the TIOBE Index, it is the 16th most popular language with 0.98% usage. R is used for statistical computation, graphing, and data analysis. It is the language our client's project is currently developed in. However, our team has limited to no experience with R.

	Familiarity	Supported Packages	Server Compatibility	Total
Python	7/10	8/10	10/10	25/30
R	3/10	10/10	6/10	19/30

Table 1 - Rankings of each language

3.2.3 Metrics

- Familiarity - our team's knowledge of the language
- Supported Packages - the availability of public packages relevant to our project
- Server Compatibility - the ease of connecting our project to a web server

3.2.4 Chosen Approach for Selecting a Programming Language

We have chosen to work with Python for our project. Overall, we ranked Python higher than R. Table 1 shows the rankings of each language in these categories.

Python received a 7 in familiarity, as it was something we have all seen before and worked with, but may not have worked with it enough to receive an excelling score.
R received a 3 in familiarity, as it was something only one of our team members has worked with before. Many of the team have not seen R syntax previously.

Python received an 8 in supported packages. This is due to there being a large amount of packages available, and several that we have found that can directly help with this project.
R received a 10 on supported packages. As the original project is written in R, it also uses supported packages that we can use that are guaranteed to work.

Python received a 10 in server compatibility. Python is compatible with each of our considered web hosting services. Additionally, we will be able to host our web application without the need of a virtual machine.
R received a 6/10 in server compatibility. Although R is compatible with each of our considered web hosting services, it requires the setup of a Linux virtual machine.

3.2.5 Proving Feasibility

We plan to utilize the available Python packages to create an analysis of a small set of GEDI data to prove our project will work on a small scale.

3.3 Selecting a Web Framework

Another challenge is finding frameworks that allow us to program websites. These frameworks must work with the language that we choose. Thus, we will look into Python and R frameworks.

3.3.1 Django

Django is a web framework for Python. According to Github, it is the 7th most popular framework. According to Stack Overflow, it is the 3rd most popular framework. It claims to be a fast and reliable way to develop websites using Python. It is free to use and works with each of our considered web hosting services.

3.3.2 Flask

Flask is also a free web framework for Python. According to Github, it is tied with Django at the 7th most popular framework. According to Stack Overflow, it is the 15th most popular framework. It was based on Django and made adjustments to make their version more beginner friendly. It has many guides and tutorials, and insists that it is heavily focused on being beginner friendly. Flask also works with each of our considered web hosting services.

3.3.3 Shiny

Shiny is an open source framework for R. It is not listed on any of the lists of popular frameworks. Similar to Flask, it has many guides and tutorials. It works with the web servers we have considered, but, as mentioned previously, R is more difficult to set up and host.

	Ease of Use	Affordability	Speed	Total
Django	6/10	10/10	7/10	23/30
Flask	8/10	10/10	8/10	26/30
Shiny	6/10	10/10	9/10	25/30

Table 2 - Rankings of each framework

3.3.4 Metrics

- Ease of Use - the level of beginner friendliness of the framework and the shallowness of its learning curve
- Affordability - the lack of cost of the framework
- Speed - how quickly functions or a program can be built using the framework

3.3.5 Chosen Approach for Selecting a Web Framework

We have chosen to use Flask for our project. Flask received the highest total score of our ranking system. Table 2 shows the rankings for each framework in these categories.

Django received a 6 in ease of use. It has a lot of boilerplate code that makes it more difficult to learn and understand. However, once that code has been learned, it is simple to work with.
Flask received an 8/10 on ease of use. It has taken away much of the boilerplate code that Django utilizes. This creates a more shallow learning curve. Additionally, there are many tutorials the developers provide to help with new users.
Shiny received a 6 on ease of use. There are functions that combine multi-step processes into a single function. However, this requires learning the R language as well as the framework, which makes the overall process more difficult.

Django, Flask, and Shiny all received a 10 in affordability. They are all free to use.

Django received a 7 in speed. The boilerplate language slows down starting a program substantially. However, once that code has been created, programming is fairly quick.
Flask received an 8 in speed. Flask is similar to Django, but reduces the amount of boilerplate code, which increases efficiency.
Shiny received a 9 in speed. It has the ability to combine complex tasks, such as modeling data into graphs, into single functions, eliciting a high score.

3.3.6 Proving Feasibility

We plan to use Flask to create a website outline that will resemble the interface that will be in our final product. This will allow us not only to test our project, but to show our client a potential website prototype.

3.4 Selecting a Cloud Service

Due to the requirements of this project — specifically the need for large, on-demand processing —a cloud computing service of some kind is mandatory. Again, the cloud service must be able to run the chosen language. In addition, some of these companies provide both batch and container-based options. Please note that batch services require a separate management service that adds complexity.

3.4.1 Amazon Web Services

Amazon Web Services (AWS) is the largest cloud service by market size with roughly 41% of all web applications running through their servers. As a well established service, AWS has plenty of options for batch and container services as well as plethora of information available as a popular software choice. However, costs with them have the potential to become expensive quickly and their network structure leaves several shortcomings in security.

3.4.2 Google Cloud

The youngest of the cloud computing options, Google has made a name for themselves with their focus on security. They are the smallest cloud service on here with only 3% of web applications running on their servers. As a result, guides and advice are hard to come by. Despite that,

Google Cloud is noted to have the most complete Python tutorial on this list.

3.4.3 Microsoft Azure

While Microsoft Azure doesn't have the same market share as AWS, they have earned their place as the second largest cloud service. Occupying roughly 30% of cloud services, Microsoft has made sure they have an answer to every feature AWS has. They are noteworthy for being the only batch service to properly support R.

	Ease of Use	Affordability	Language Support	Total
AWS	5/10	5/10	5/10	15/30
Google Cloud	7/10	2/10	7/10	15/30
Microsoft Azure	6/10	6/10	10/10	22/30

Table 3 - Rankings of each cloud service

3.4.4 Metrics

- Ease of Use - the level of beginner friendliness of the service and the shallowness of its learning curve
- Affordability - the lack of cost of the framework
- Language Support- how well the service can handle the languages we need

3.4.5 Chosen Approach for Selecting a Cloud Service

We plan to use Microsoft Azure for the cloud computing service as its ranking is higher compared to the other services. Table 3 illustrates how the different choices ranked against each other.

Microsoft Azure appears to be more cost-effective and easier to implement than the container-based services. Being able to execute R code provides options should we encounter problems with Python as well as making testing easier. In addition, Azure is flexible enough to allow multiple different project architectures.

3.4.6 Proving Feasibility

We plan to use the trial time offered by Microsoft to build a virtual machine that can host a website backend. In addition, we will run the currently existing R programs on Azure with the initial data from the GEDI project. Both of these tasks can be presented as prototypes to our client.

4 Technology Integration

4.1 Introduction

Each individual design decision needs to be able to work with the other decisions. This section will outline the challenges in creating a coherent system with the solutions discovered in the previous section, and discuss how we plan to solve those challenges. Descriptions for how either a direct or an indirect architecture will look like with our system will be provided at the end.

4.2 Integration Challenges

The integration challenges to be outlined in this section are as follows:

- Will the web hosting service chosen support the language chosen?
- Will the web hosting service chosen support the framework chosen?

For the language, web framework, and web hosting service, we chose Python, Flask, and Azure Batch respectively. Microsoft Azure must be able to support Python. It is confirmed that it does on Microsoft documentation, and it also supports R. The support for R serves as a backup in case Python does not end up being the ideal language like we thought.

Microsoft Azure must also be able to support Flask. There are several tutorials specific to running Flask web applications on Microsoft Azure. In addition, the Microsoft Azure documentation has an official walkthrough for web application projects based on Bottle, Flask, or Django frameworks. It appears that Microsoft Azure will have no problems supporting the Flask framework.

Figures 1 and 2, pictured below, demonstrate how the system would connect for either direct or indirect architectures. Application for Extracting and Exploring Analysis Ready Samples (AppEEARS) is a NASA tool that provides geospatial datasets for users. This is the application that we will request GEDI data from.

With a direct architecture, shown in Figure 1, the website would send a request to the cloud-hosted program based on the region requested by the user. The cloud-hosted program would then fetch the necessary GEDI data from the AppEEARS database and run the computations requested. Once finished, it'll return the results to the website. In this case, Microsoft Azure will be continuously running. Delays due to overwhelming the program are a risk if several requests for data come in at once. In addition, this approach requires the cloud service to always be running, possibly increasing costs.

If we use an indirect architecture, shown in Figure 2, Microsoft Azure provides two independent systems instead of one. The management service is a middle-man between these components, receiving the requests from the website and fetching the GEDI data from AppEEARS. However, no data processing is done on the management service. The management service only holds this information to deliver it to the computational service. The computational service then processes the data and returns it to the management service, to be displayed for the user. With this approach, only the management service needs to be continuously running, which has much smaller computational needs. Instead, the computational service is started on-demand or when financially appropriate to process the data. While this approach is more cost effective, it is more complex and may be potentially slower due to activation of the computational service for each request. This approach is also far more resistant to becoming overwhelmed, as the management service can easily cut off or delay multiple requests, preventing the computational service from being overwhelmed.

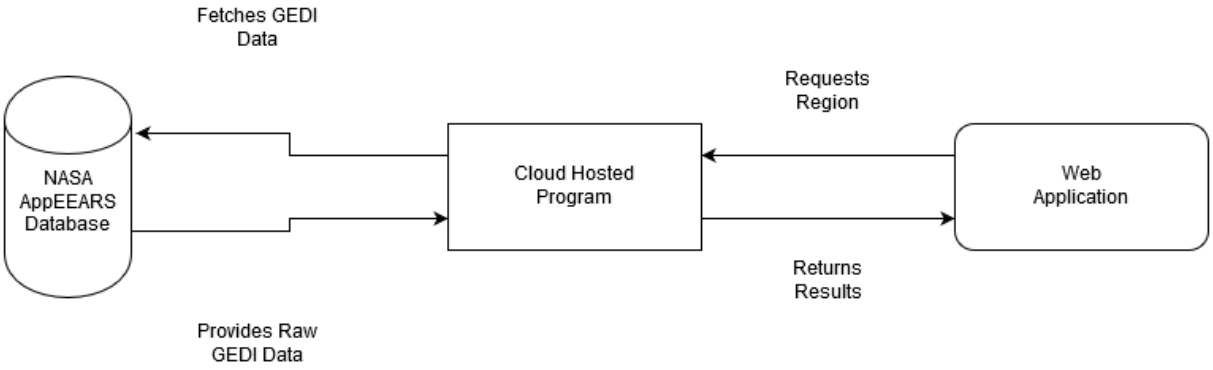

Figure 1 - Direct architecture layout

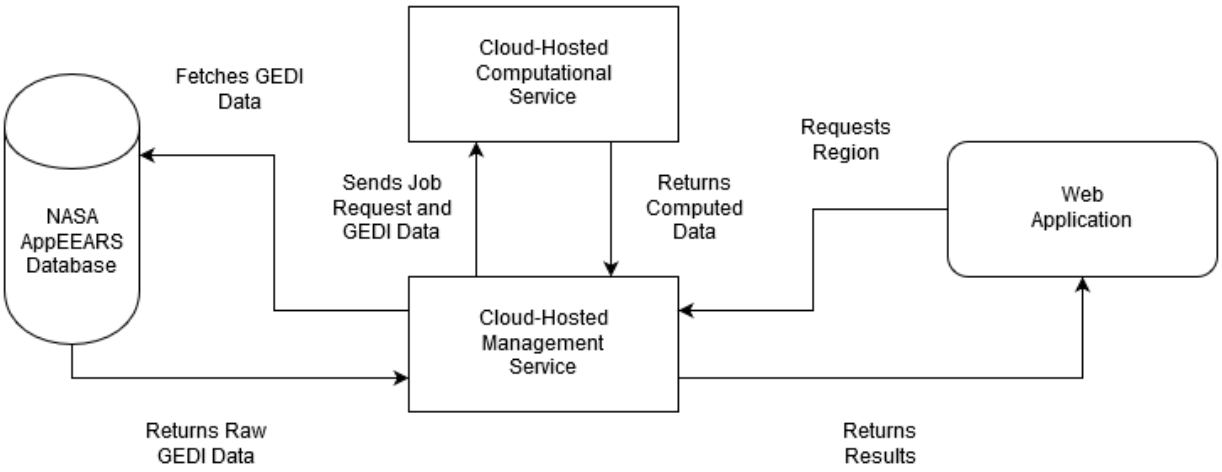

Figure 2 - Indirect architecture layout

5 Conclusion

The purpose of this project is to develop an application to solve the problem of an inefficient workflow that requires a large amount of manual work when processing GEDI files. This document is a technological feasibility analysis of the project. Through client and team discussion, we encountered three main problems. They are choosing the programming language, web framework and web hosting service. The following table will show the challenges, solutions we have decided upon, and the confidence level of our solutions.

Programming language (confidence level)	Web framework (confidence level)	Web hosting service (confidence level)
Python (25/30)	Django (23/30)	AWS (15/30)
R (19/30)	Flask (26/30)	Google Cloud (15/30)
	Shiny (25/30)	Microsoft Azure (22/30)

Table 4-Level of confidence in solution

Regarding the programming language, we considered the familiarity, supported packages and server compatibility, using a potential score of 10 points for each. Python got 25, and R got 19 (for details, see Table 1 on page 7), so we will choose python as our language. Regarding the web framework, we considered ease of use, affordability and speed, and scored 10 points for each. Django got 23, Flask got 26, Shiny got 25 (see Table 2 on page 9), so we will choose Flask as our web framework. For the web hosting service, we considered ease of use, affordability and language support, out of a score of 10 points for each. AWS got 15, Google Cloud got 15, Microsoft Azure got 22 (see Table 3 on pages 11 - 12), so we will choose Microsoft Azure as our web hosting service. We are very confident in this analysis. It not only considers the project itself, but

also considers the capabilities of the developers. Our sponsor has also confirmed the feasibility of our analysis. We believe that we can use these solutions to develop the application our project requires.