SENG 371 Project 1

Prestige Worldwide
Julian Rocha, Nicholas Roethel, Geoffrey Harper, Ryan Russell, Ben Mazerolle
February 25th 2019

1. One Machine to Distributed Computing

1.1 Overview

This section will be a case study on the transfer of code from a single machine to a batch cluster. In particular, it will discuss the process and changes made to transfer code from the One Machine to the Compute Canada code base. First, the One Machine's code will be examined on its own to analyze its inefficiencies. Afterwards, the Compute Canada code will be analyzed. Finally, the two code bases will be compared and a high level diagram will be presented [1].

1.2 Analyzing One Machine Code

From looking at the readme document a few things became immediately apparent: The One Machine code base was originally only supported on Linux and OSx systems, and in order to access data, the user requires an a EUMETSAT EO-portal account. Moreover, it is only able to download Sentinel-3 marina data located on CODA. However, if necessary, the code may be modified to download Sentinel-2 and Sentinel-1 data from the ESA Sentinel online portal, or any Sentinel-3 data from the CODA-processing portal.

The first file committed to the One Machine is titled test1.py, which is a program written in Python. It begins by importing a series of values and creating a variable called time, which is initialized with the current processing time. Then the program is directed to the /polymer-4.8 folder on the local machine. Once there, the program asks the user for the filename. It then proceeds to search through the Sentinel-3 data. Once it has searched through the data, it prints each file name and their processing times. Upon completion, the program also prints the total processing time.

The next file committed to the One Machine is called loop_folders_unzip.sh. This is a shell script, which, as indicated by the name, loops through folders in the specified directory and unzips them if they match the regex expressions within them.

The last file provided is another shell script titled get_data_CODA_Fernanda.sh. Out of the three files provided, this file is by far the easiest to analyze, since it has the most complete documentation. This script first begins by initializing the user's operating system. Next, it gets the command path, the root command, and the user's login information. The program then continues to ask the user to edit various parameters such as the number of download retries, sensors types, start and end dates, the number of concurrent downloads, the amount of products to download per day, the type of data, the download area (what gets downloaded), and a variety of other specifiers. Once the user has specified the required download parameters, the rest of the code which loops through the dates creates directories when necessary, launches the command for the date, and reconstructs the dates when possible.

1.3 Inefficiencies

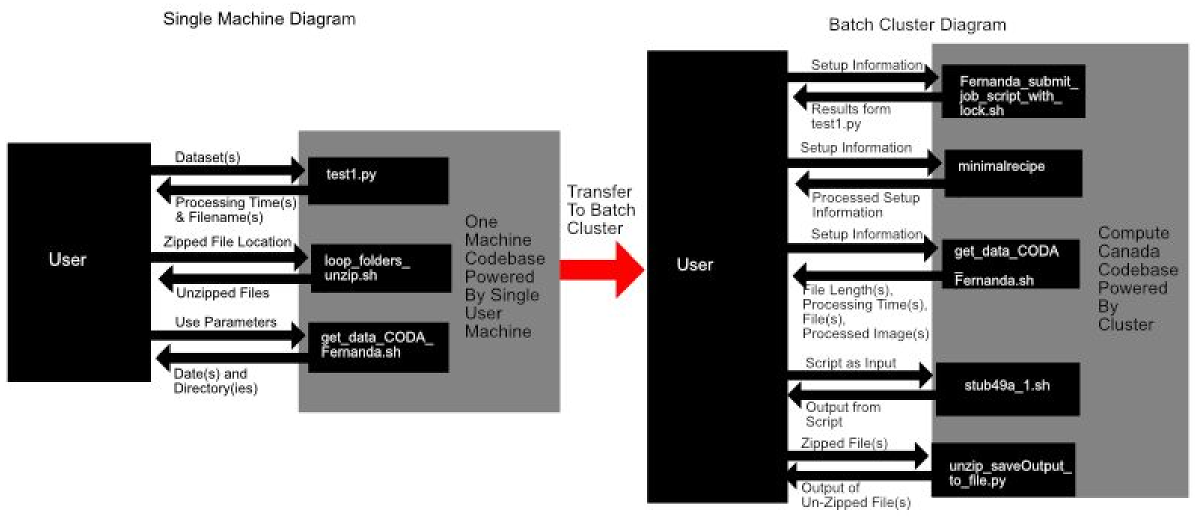
Upon analyzing the One Machine code, several inefficiencies were detected. The first inefficiency detected was the use of Python. Although Python is very readable and easy to use, its poor efficiency can often be detrimental when compared to other languages, such as C++. Although this may be a non-issue for small programs, if this program's complexity were to be expanded into a larger project, the use of a different language could make a big difference in run time. Another big problem within the code is that, for the most part, there is a significant lack of code comments and documentation. This made the code difficult to analyze and reverse engineer. It is very likely that this lack of documentation made the process of transferring from a single machine to batch cluster more difficult than it needed to be. Moreover, if this were to be transferred to a multi-cloud system, this would inevitably cause lots of problems and confusion for the developers transferring the old system.

1.4 Analyzing Compute Canada Code

The first script committed to the Compute Canada code base that will be analyzed is a shell script called Fernanda_submit_job_script_with_lock.sh. Upon inspection, it seems as if the purpose of this script is to set up how the job will be run (name, memory and nodes allocated, run time, email for notifications, etc.) and then to load and start the singularity. The program can be modified to run different programs, however, the program is currently set up to run the test1.py file (the python script from the original One Machine database). The next file is a script called minimalrecipe, which appears to load in the files and take care of a portion of the setup. The script_2018_one_folder.py is the script which stores the majority of the functionality of the codebase. It is able to load in files, compute processing times, calculate file lengths, search for files, edit files, and process images. Another shell script provided is the stub49a_1.sh, which simply opens and runs a python script. The last script provided is another python script called unzip_saveOutput_to_file.py. This file, as is indicated by the names, unzips files. The input files are unzipped in a recursive method, and the outputs are printed to a text file.

1.5 High-Level Diagram & Comparison

Although the functionality between the One Machine and batch-cluster code has remained similar to a large extent, there are some key differences that distinguish the two. The largest difference is that the Compute Canada code is much more modular and automated than the single machine code. This makes it easier to transfer to other machines in the future. Moreover, the code on the Compute Canada system has more documentation, even though the code itself is longer and more complex (which can be seen in Figure 1). We believe these concepts of enhanced modularity, well-documented and more transferrable code will need to be further implemented when transferring to a batch cluster system or cloud (and multi-cloud) environment.



Single Machine Diagram
User
Dataset(s)
test1.py
Processing Time(s) & Filename(s)
Zipped File Location
loop_folders_unzip.sh
Unzipped Files
Use Parameters
get_data_CODA_Fernanda.sh
Date(s) and Directory(ies)
One Machine Codebase Powered By Single User Machine
Transfer To Batch Cluster
Batch Cluster Diagram
User
Setup Information
Fernanda_submit_job_script_with_lock.sh
Results form test1.py
Setup Information
minimalrecipe
Processed Setup Information
Setup Information
get_data_CODA_Fernanda.sh
File Length(s), Processing Time(s), File(s), Processed Image(s)
Script as Input
stub49a_1.sh
Output from Script
Zipped File(s)
unzip_saveOutput_to_file.py
Output of Un-Zipped File(s)
Compute Canada Codebase Powered By Cluster

Figure 1: Conversion From One Machine to Compute Canada

2. Reverse Engineering Compute Canada Code

2.1 Compute Canada Code Structure

The Compute Canada codebase is divided into 4 different files – minimalrecipe, script_2018_one_folder.py, stub49a_1.sh, unzip_saveOutput_to_file.py – that assist One Machine users perform data processing techniques on the Compute Canada server. The minimalrecipe script runs a variety of bash shell commands to help set up the user's cluster with the proper packages so that they can communicate and use Compute Canada's clusters. Stub49a_1.sh is used to tell the cluster where to run the python scripts within the server. The script_2018_one_folder.py file is where most of the functionality of the Compute Canada codebase in contained. The script locates and loads user data files. In addition, it has the ability to compute processing times, calculate file lengths, and apply basic image processing. Finally, unzip_saveOutput_to_file.py is used to unpack stored zip files within the user's Compute Canada cluster by using a recursive method and then outputting files as text files in order to be processed by Compute Canada scripts such as script_2018_one_folder.py

In essence, the Supercluster code base can be seen as being divided into preprocessing scripts and processing scripts. Scripts like minimalrecipe, stub49a_1, and unzip_saveOutput_to_file.py act as preprocessing scripts that do cluster specific jobs in order for the Compute Canada python scripts to be able to process data. Then, processing scripts such as script_2018_one_folder.py compute the preprocessed data for the user.

2.2 Compute Canada Code as a Cloud-Based System

The structure of a Compute Canada code base as a cloud-based system will be based on the usage of AWS cloud computing features. Users are able to access their Compute Canada account through AWS Identity and Access Management. Now with their individual clusters, users can upload and store data using AWS S3 architecture and keep backups using AWS Glacier. AWS Lambda allows users to run their own functions and data processing in conjunction with AWS Aurora. AWS redshift helps query the required data. Along with Aurora's ability to run user-uploaded functions on Lambda, Aurora can also store Compute Canada's codebase functionality. This ability to store a codebase using Aurora allows the script_2018_one_folder.py methods to be accessed by the user via Lambda. AWS Elastic Loading and Balancing helps optimize the cloud system and AWS EC2 set with High Performance Cloud Computing Environment that allows big data computations to be very time and cost efficient. Note that there is no longer need for minmalrecipe bash scripts to initialize the user's cluster so it can run Compute Canada functions. Instead, the pre-stored Compute Canada functions in the Aurora database can be accessed without any user specific recipes set up, thus making recipes obsolete. The diagram in Figure 2 provides visualization for the cloud architecture.

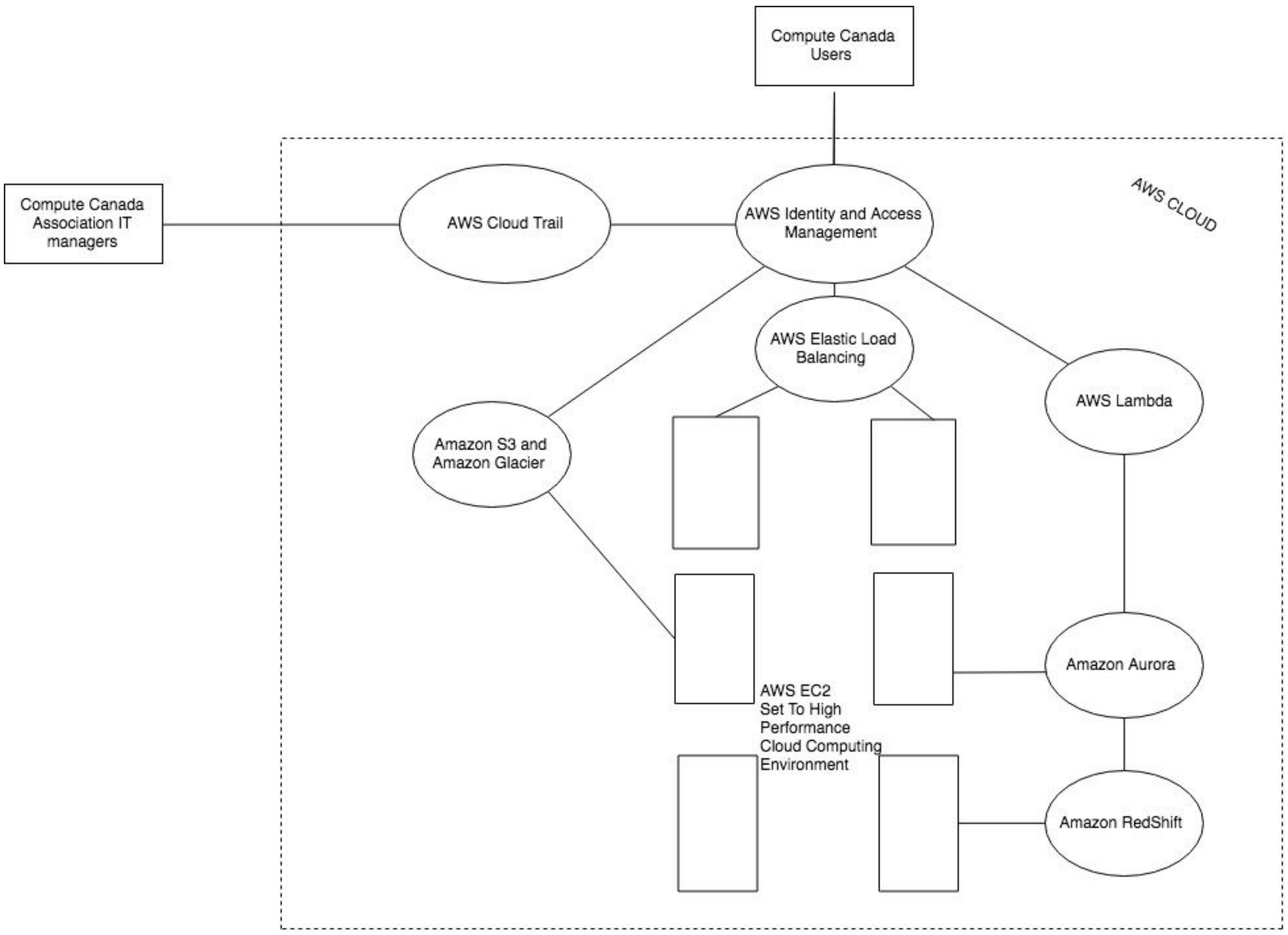

Compute Canada Users
Compute Canada Association IT managers
AWS Cloud Trail
AWS Identity and Access Management
AWS CLOUD
AWS Elastic Load Balancing
AWS Lambda
Amazon S3 and Amazon Glacier
Amazon Aurora
AWS EC2 Set To High Performance Cloud Computing Environment
Amazon RedShift

Figure 2: AWS Cloud Architecture Visualization

2.3 Converting Compute Canada Code Base to the Cloud

There are three components that need to be transferred to the AWS Cloud architecture:

1. The stored data in the Compute Canada clusters
2. The individual Compute Canada user accounts
3. The functional components of the Compute Canada code base (such as the script_2018_one_folder.py code)

Both stored data and their relational accounts can be transferred using the AWS Server Migration Service. Compute Canada can use tools like AWS snowball Edge to help move large amounts of data into AWS S3 storage. Transferring Compute Canada functionality code like the script_2018_one_folder.py to Amazon Aurora can also be transferred using the AWS Server Migration Service [9].

3. STAC Specification and AWS Cloud Platform

3.1 Overview

This section will analyze a specific implementation of the STAC specification built using AWS infrastructure. It will first give an overview of what the STAC specification is, before proceeding to analyze how STAC was implemented using AWS for the CEBRS geospatial imagery. The pros and cons of the STAC spec will be weighed, and finally, the section will conclude by explaining how the STAC specification could help solve some of the problems the EDS is facing.

3.2 What is the STAC Specification?

The SpatioTemporal Asset Catalog (STAC) specification is an effort to standardize the way geospatial assets are exposed and queried online [3]. One main focus of STAC is the ability to search for geospatial imagery based on location and timeframe. The idea behind a large-scale specification like STAC is to ensure all parties are formatting their data in the same way, therefore increasing collaboration across applications that make use of this data. The specification is still in an early version at the time of writing, so the STAC spec is still a work in progress. At a high level, the specification works by enforcing developers to implement a collection of JSON objects with links to other JSON objects. These objects are used as standardized metadata to search for geospatial assets of interest. See Figure 3 for a UML diagram of the STAC model.

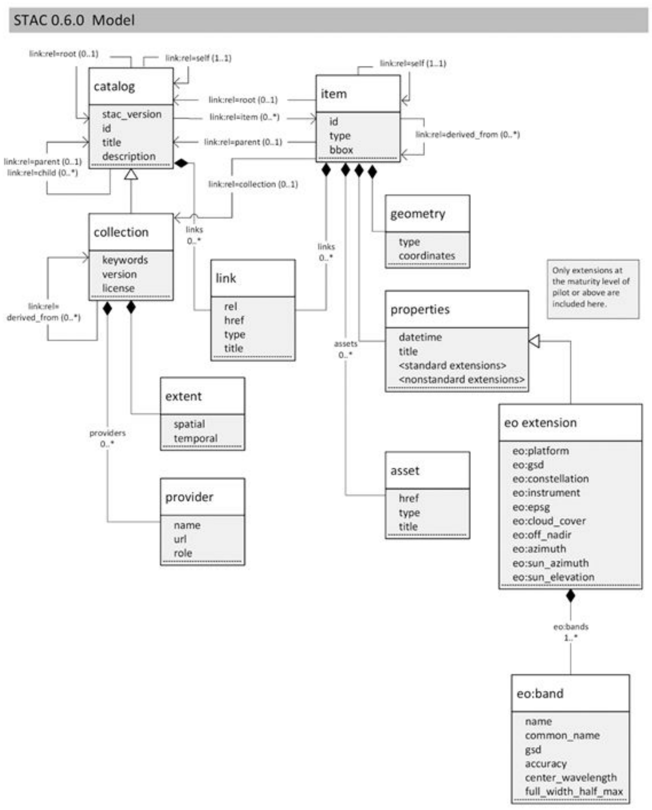


Figure 3: UML diagram of the STAC model [4]

Catalogues and collections are used like roadmaps, to crawl the data sets. With links to other catalogues and collections, eventually, a catalogue might link to an item. Items are “the core granular entit[ies] in a STAC,” providing all the metadata related to a particular geospatial asset [5]. Items are GeoJSON Features, which means they can encode geographic data structures [6]. In order for STAC to be implemented, a series of catalogues/collections must first be implemented and maintained. Using these catalogues/collections, new items can be linked when new data from the satellites comes in.

3.3 How was STAC Used for the CEBRS Data?

The China-Brazil Earth Resources Satellites (CBERS) consist of 5 satellites launched since 1988. The imagery collected by these satellites has been opened to be accessed by the public. This archive of satellite imagery has been stored on AWS using the STAC specification as a standardized way to expose and query the data.

The main challenge with implementing the STAC specification is keeping catalogues up to date. It must be updated whenever: new imagery comes in, a new STAC metadata version needs to be supported, or when the existing catalogue needs to be reconciled with the original contents of the S3 bucket [7]. See Figure 4 for a high level diagram of the AWS infrastructure CBERS used to keep their STAC specification up to date.

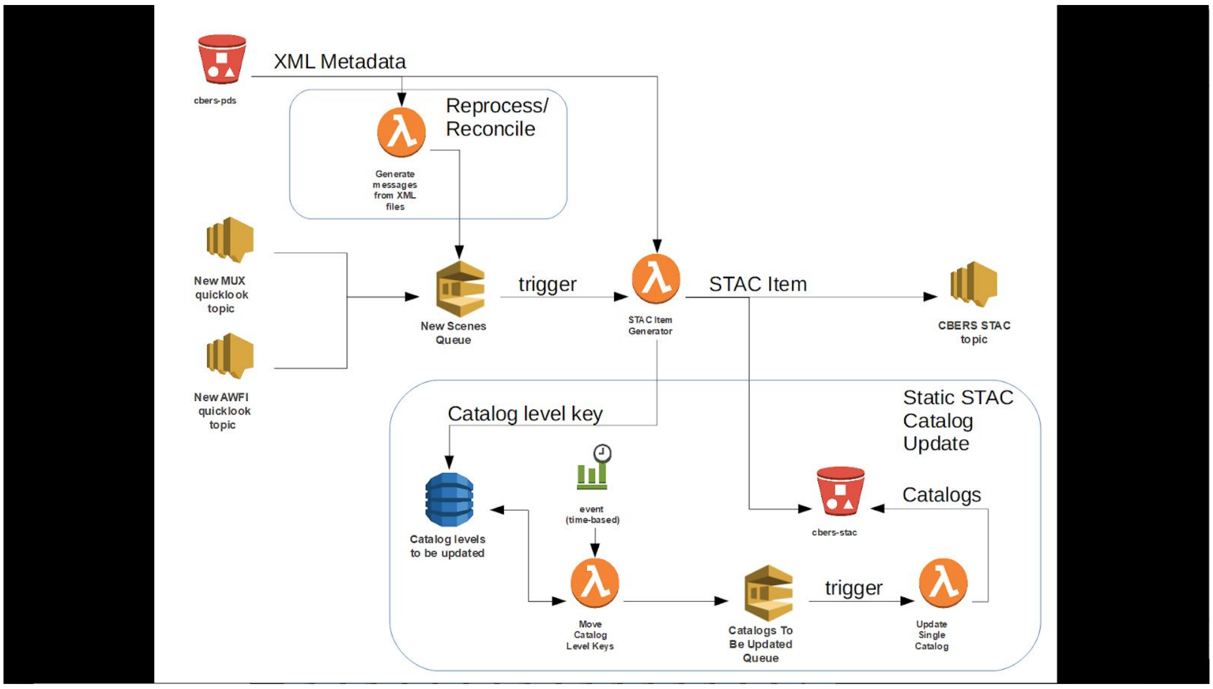


Figure 4: STAC Specification Update Diagram [7]

In order to understand Figure 4, let us simplify by walking through the sequence of events for different use cases. Say a new image is taken by a satellite. This image undergoes processing known as CBERS Scene Ingestion. Scene Ingestion produces XML metadata of the image and stores it in the publically available S3 bucket “cbers-pds”. XML Metadata is data which describes the image taken. Metadata is also commonly referred to as: data about data. The last step of Scene Ingestion is the generation of a thumbnail for the image. This last step publishes to an SNS topic, indicating that the scene ingestion is complete. Examples of these topics are labelled in Figure 4 as “New MUX quicklook topic” and “New AWFI quicklook topic”. The “New Scenes Queue” is an SQS subscribed to these topics and triggers the “STAC Item Generator” Lambda function as the topics come in. This is the point where the the generation of STAC Item actually begins.

The “STAC Item Generator” function pulls the XML metadata from “cbers-pds” for the appropriate image and creates a STAC item for that image. It stores the STAC item in the publically available S3 bucket “cbers-stac” and publishes to the SNS topic “CBERS STAC topic” indicating that a new STAC item was generated.

The function also records any changes necessary to the catalogue. For example, the new image may belong to multiple catalogues/collections and these all must now link to the new STAC item. These catalogues are recorded in the DynamoDB “Catalog levels to be updated”. “Move Catalog Level Keys” is a Lambda function that finds any changes made in the DynamoDB and turns them into catalogue-level specific messages to be published to the SQS “Catalogs To Be Updated Queue”. This Lambda function is triggered periodically in batch-processing style using Amazon Cloudwatch. The messages in the SQS are consumed by the Lambda function “Update Single Catalog”, which determines the necessary changes in “cbers-stac” for that catalog and makes them. The image now has an appropriate STAC item and is linked to by the appropriate catalogs.

The infrastructure also has the ability to reprocess and reconcile catalogs and items by manually triggering the Lambda function “Generate messages from XML files”. Depending on the STAC objects the user wishes to reconstruct, they can use this function to push messages onto the “New Scenes Queue” which will force the “STAC Item Generator” to reconstruct the objects of interest using original XML metadata from “cbers-pds”.

3.4 Pros of Adopting STAC

Conforming to a data spec such as STAC is very beneficial when considering an evolvable system like the EDS. If adopted by multiple parties, it has the possibility of increasing cross-organization collaboration with geospatial data. Smart catalog design also means the data being collected will be more accessible than without a specification.

Looking at the CBERS implementation, if STAC is implemented with thoughtful design, it has many reliability and parallelization benefits. The “STAC Item Generator” Lambda function consumes messages from the “New Scenes Queue” directly. This means, no queue polling is required and when the queue grows with messages, as a result of many images coming in at once, the number of Lambda functions scales appropriately to handle the load in parallel. Moreover, if messages fail in the Lambda function, they are returned to the SQS instead of getting lost. This part of the system will likely scale well should new sensors be added in the future, or if image frequency increases with existing sensors.

3.5 Cons of Adopting STAC

STAC is still a work in progress and its specifications are always subject to change. Best practices of implementations are still being determined, although once there is a critical mass of extensions, STAC versions will be locked down to 1.0 [5]. An example of room for improvement in best practices can be seen by a weakness in the CEBRS implementation. It avoids redundant processing of catalog keys by consuming them in a batch processing style using Amazon Cloudwatch. While this is surely a benefit in reducing computing costs (as outlined in the blog), it creates a more significant time delay between the creation of a STAC item, and the updating of the catalog to reflect the addition of that item. A user who just received the “CBERS STAC topic” for an item they are interested will have to wait a variable amount of time before the catalog is updated with the new item. This time delay could very well be insignificant in some cases, but it is possible that future applications to be built off this infrastructure will have some kind of hard time requirements in order to adequately automate.

3.6 Will STAC Help Solve EDS problems?

If new data sources can conform to the STAC spec, it will be much easier to actually find and make use of the data being collected. STAC also provides a form of evolvable documentation for when new apps or data sources are being added to the EDS. Finally, implementing STAC today allows for feedback to be incorporated into STAC before its final design is locked down. This means the EDS has the opportunity to be one of the pioneers for STAC. If EDS developers collaborate appropriately with STAC developers, the EDS has the chance to shape STAC to suit its needs.

4. Proposed Cloud-Based System

This section will serve as a blueprint for data scientists and those familiar with the One Machine approach that was analyzed and extended to a distributed computing format with Compute Canada in Section 1. This section will also outline the design for a cloud-based platform that allows for multiple algorithms to be launched from a single cloud location.

4.1 Discussion of Tools and Architecture

Before the design for the cloud-based platform can be realized, a comparison and selection of software tools and architecture design must be completed. The tools we will be focusing on for this design will be selected based on their ability to assist with the rapidly changing nature of this design in the early stages. We expect the design of the cloud-based platform will require iterative updates to allow it to function with more algorithms with updated technology, and ideally to integrate with other initiatives from Canada’s Digital Technology Supercluster in the future. Therefore, the three types of tools we will compare and contrast are Continuous Integration, Configuration Management, and Containerization tools.

4.1.1 Continuous Integration Comparative Analysis

Continuous Integration (CI) “is the process of automating the build and testing of code every time a team member commits changes to version control” [8]. Usually, when code is committed, an automated build system is activated that runs unit tests against the master branch to ensure the new code changes do not break specific functionality of the system.

In terms of this project, the idea for the proposed cloud-based platform is that data scientists will be able to bring their own algorithm and seamlessly integrate them into the system, allowing their algorithm to be run using the same architecture as the original Polymer algorithm. Therefore, a Continuous Integration tool is the perfect candidate to ensure that each new algorithm is, indeed, seamlessly integrated. The selected CI tool should be directly linked to the platform to ensure consistency within the platform and that the newly added algorithm does not introduce unintended errors into the system.

CI is especially useful with the proliferation of software developers working remotely, as it expedites the process of merging changes without requiring lengthy meetings. This point is especially important for this project, as the cloud-based platform will essentially allow users to contribute to the software in the form of their own algorithms.

The two CI tools that will be compared in this section are Jenkins and Travis CI. Jenkins is a widely used, open source CI tool written in Java that boasts over 147,000 active installations and over 1 million active users [8]. It integrates with existing development, testing, and deployment technologies through the use of plugins. Some of these technologies are Maven, Git, GitHub, and Selenium, and various Configuration Management tools. Jenkins can also integrate efficiently and run with AWS, as we have done in the lab for this course, SENG 371.

In the integration phase of CI, developers commit their changes and expect an automated process to begin. This is where where Jenkins becomes useful. Once a commit has been made, Jenkins automatically pulls and tests these new changes. Jenkins runs on a server and periodically polls the git repository for new commits. Once it pulls the new commit, it can run a suite of unit tests to validate it before building the new module. If any of the unit tests fail, the build fails. Therefore, a developer must fix the system with a new commit. Once the build is complete, Jenkins saves an artifact to an external repository.

As for deployment of software, once Jenkins has completed the build, it can automatically deploy it to a server. For example, it can deploy the build to a test environment. These test environments can be set up to automatically deploy to production environment once integration testing has been completed, adding in continuous deployment to the process. This is important for the eventual deployment of an algorithm after it has been integration tested.

Developers can also check that their changes have taken effect, as well as being able to track if builds and tests have failed. Jenkins has functionality to track metrics and send alerts. For example, if a build is marked unstable by JUnit because a unit test failed, an email can automatically be sent to the developer or quality assurance analyst in assigned to that test suite.

On the other hand, Travis CI is another CI tool which is free to host on your own server [9]. Travis CI has very similar functionality to Jenkins for the integration and deployment of software, which has been described above. Travis CI differs in its setup efficiency: simply adding a travis.yml file to the root of a GitHub project, then creating an account on travis-ci.com will activate Travis CI on a project. Therefore, Travis CI is much easier to initially add to a project, and as a result is an attractive option for small applications with few developers.

Another difference between Travis CI and Jenkins is the lack of plugins available for Travis CI. While Jenkins can integrate with systems like JUnit, Mailer, and Git for unit testing, emails, and version control respectively, Travis CI severely lacks this additional functionality.

As a result of this comparison, we have determined that Jenkins is the optimal Continuous Integration tool for this cloud-based platform due to the following factors:

- Jenkins works seamlessly with AWS
- Jenkins provides a better plugin system and a greater selection of plugins
- The functionality of both tools is almost identical

Jenkins will therefore be selected as the CI tool for this proposed system.

4.1.2 Configuration Management Comparative Analysis

Software Configuration Management (SCM) allows for the organization, control, and management of software and documentation during the lifecycle of software development [10]. If implemented correctly, it can reduce mistakes and increase productivity, while mitigating development cost. Current day software development typically involves at least several developers, various budgets, multiple operating systems and platforms, and consistently updating schedules among a large number of ever-changing variables. A well-implemented SCM will further mitigate these issues by producing single, harmonic methodology that defines solutions to predicted issues.

For this project specifically, the usefulness of SCM tools is similar to that of Continuous Integration tools: in order to have the algorithms from many data scientists be successfully integrated into the platform, there must be some sort of management to standardize documentation, the correct process for integration, and the limitations on the algorithms that can be used with the system.

The tools that will be compared in this section are Puppet and GitHub, which are both tools that we have explored in SENG 371 up to this point. Puppet is a tool that is directly focused on Software Configuration Management, while GitHub is primarily used for version control and source code hosting, but also has a variety of useful and lesser known features. SCM capabilities are included in these lesser known features.

“Puppet is a Configuration Management tool that is used for deploying, configuring, and managing servers” [11]. GitHub, on the other hand, excels in version control and source code management. GitHub can also be used for configuration management through the use of some useful functions, primarily the streamlined sharing services and secure storage that the platform provides [12]. In addition, GitHub does not require manual backups.

GitHub is effective as configuration management software if a git client is available on the Config_Server that is hosted on GitHub. Furthermore, part of GitHub’s usefulness as a configuration management tool comes from the fact that it is, inherently, version control software. By storing files on GitHub, they can be quickly updated, the changes can be tracked, and previous versions can be restored [12].

Puppet “[defines] distinct configurations for each and every host, and continuously checking and confirming whether the required configuration is in place and is not altered” [11]. Moreover, Puppet is designed to be beneficial in all stages of the software development process. Often in real-world software projects, requirements change from prototype to implementation, and Puppet can maintain a system’s integrity as the implementation shifts.

In addition, Puppet is easily integrable with AWS, which makes it a very good option for the proposed cloud-based platform.

In conclusion, while GitHub can be useful for configuration management, it cannot compete with the functionality of Puppet and it’s dedication to configuration management functionality as well as its ability to integrate with AWS. Therefore, Puppet is the selected Software Configuration Management tool.

4.1.3 Containerization Tool Comparative Analysis

A container “is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another" [13]. Containers are designed to be similar to virtual machines in that they simulate an environment with a specific operating system, but they also serve as lightweight, standalone executables that can be run anywhere. Essentially, "Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware" [13].

Containers solve the widespread issue in software development of an application working on one computer but not another, due to a multitude of dependencies in software projects. Because of this, containerization tools like Docker have become extremely popular in recent years.

In terms of this project, containers are a perfect fit. Because the platform needs to run algorithms for many data scientists and users, the system needs to operate in many environments. Every user will have a slightly different, if not a vastly different setup. Containers are the ideal choice for a project with these requirements. The two tools that will be compared in this section are Docker and Vagrant.

Vagrant is a tool that "enables quick configuration and provisioning of [virtual-machine-like] containers that help to isolate the application in its own development environment" [14]. The difference between containers and the isolated environments that Vagrant enables is that Vagrant works on top of real hardware. However, it can still emulate the virtual architecture required by a developer. Vagrant provides no built-in security measures, so developers must set up required security firewalls on their own. Furthermore, Vagrant containers "cost a certain amount of resources to sustain themselves" because the run on the host operating system [14].

Docker, on the other hand, "is an open-source platform that allows isolating the apps within containers. Docker moved from LXC to *containerd* [as its container manager] to enable industry-wide standardization" [14]. Docker provides an extremely consistent application experience in all stages of the software development process. It can also be integrated with technology such as Kubernetes to manage many containers in an organized fashion, Jenkins, and various other Continuous Integration and Continuous Deployment tools.

Docker containers utilize the system kernel that they run on, so there is a risk involved where code run on a Docker container could affect the host operating system along with other containers running on the system. “This risk can be mitigated by adding built-in security checks to the container” [14]. The launch speed of Docker containers is significantly faster than Vagrant, consistently launching in under a minute. Docker containers also “boast 300% increased efficiency as compared to running virtual machines” [14]. The only notable limitation of Docker is that its containers cannot currently be run on Mac OS, which Vagrant can run on without full functionality.

As a result of this comparison, it has been determined that Docker is the optimal containerization software to use for the proposed cloud-based platform. This is due to Docker’s increased efficiency and launch speed, the consistent experience for users and developers on all platforms, the lightweight nature of its containers, and the integration capabilities with other technology that has been selected in the previous sections, such as Jenkins.

4.2 Bring Your Own Algorithm

4.2.1 Multiple Algorithms for One Dataset

One of the most prominent benefits of an easily-accessible Earth Data Store is the increase in knowledge applied to and gleaned from the dataset in the form of additional algorithms to the current Polymer algorithm. Using the tools described in Section 4.1, the transition from single computer to the EDS for the existing polymer algorithm is relatively simple: just put the algorithm in a Docker container, have it run when requested, and output the results at the end using the tools described below in Section 4.2. The process of incorporating more algorithms into this methodology is a similar process, albeit with a few more steps.

4.2.1.1 Adding a new algorithm

The EDS should contain the functionality to add new algorithms via a request-based system. This can be accomplished using an input form (described in greater detail in Section 4.2.2) that allows for the submittal of the description and the language it is written in. Alongside these manual entries, EDS users should attach their algorithmic code and a pre-configured docker image containing all the dependencies necessary for running the algorithm in a completely isolated environment [15].

Then, the submitted algorithm must pass the EDS' approval process. The algorithm should be submitted to the EDS approval team via a merge request orchestrated by Puppet. To further utilize the power of crowd-sourced knowledge, the EDS approval team should be consisting of vetted, regularly-contributing individuals, from whom the decision to adopt an algorithm or not should be sourced. Since the numbers of contributors are unknown at this point, no defined number of approvers can be set, however the EDS should employ a structure that requires a minimum number of approvals to deploy the algorithm.

Puppet allows for automated testing, as well as the Git merge request approval process, so new algorithms should be run using automated testing that leverages the EDS data set to ensure the algorithm produces an error-free result. Then, the approval team can review the new algorithm, its testing results, and provide final approval on its addition to the EDS.

Finally, once the approval process is completed and the algorithm is approved, the new algorithm's docker container should be deployed for use to the EDS.

4.2.2.2 Running multiple algorithms

Once integrated into the EDS, users should be able to select which algorithms they want to run on the dataset. Using the pre-configured docker container from the algorithm adding stage, the EDS can run each algorithm simultaneously in an isolated environment specifically configured to that algorithm's language and dependencies. Depending on the space constraints associated with the EDS, each individual algorithm should have a set number of docker containers pre-configured for use. When first added, the algorithm should be allocated with a single docker container for use. However, as time progresses, and the usage statistics of the algorithm become more well-documented, the EDS should reconfigure the number of containers allocated to each algorithm to reflect their actual usage. Our proposed balancing methodology is to initially give the polymer algorithm all of the available docker containers, removing one at a time as other algorithms are added. When a new algorithm is added, it should undergo a one-week probationary period where its usage is tracked but only one docker container is made available. After that week, it’s number of uses will be contrasted with the usage of all the other algorithms in the previous week, and it will be allocated the corresponding proportional number of containers. This process will then repeat as new algorithms pass their one week probation, producing a container allocation procedure that best represents actual usage.

4.2.2 User Interface

A simple, no-frills UI for use by the scientists and enthusiasts who would leverage the Earth Data Store is a critical part to the ultimate success of converting to cloud-based computing. By making the “bring your own algorithm” process as straightforward as possible, the EDS will not only leverage the improved speed of cloud computing, but also the potential for crowdsourced knowledge that would result from easily-accessible algorithm application.

4.2.2.1 Algorithm Selection and Runtime

Algorithms and datasets should be easily chosen and added to by the user. Since there is a limited number of algorithms for use on the data, a searchable dropdown menu of all possible algorithms and dataset restrictions should act as a filter for the operations. These dropdowns could be easily incorporated using Vue.js’s built-in libraries, and their information could be stored in a small SQL database as shown in Figure 5 [16].

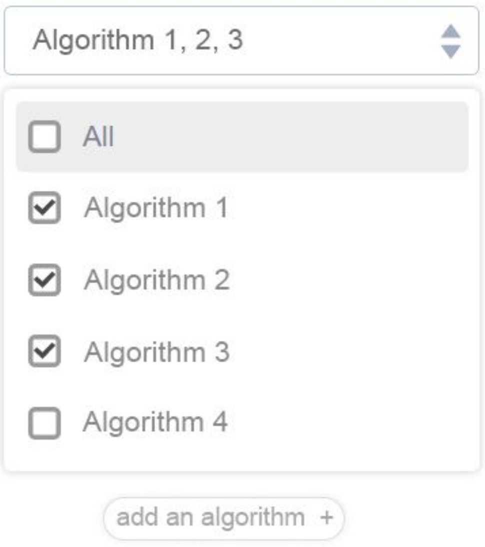


Figure 5: Algorithm Dropdown List

4.2.2.2 Result Display

The resultant graphs from the algorithms can also be visualized using Vue.js's extensive chart libraries, as displayed in Figure 6.

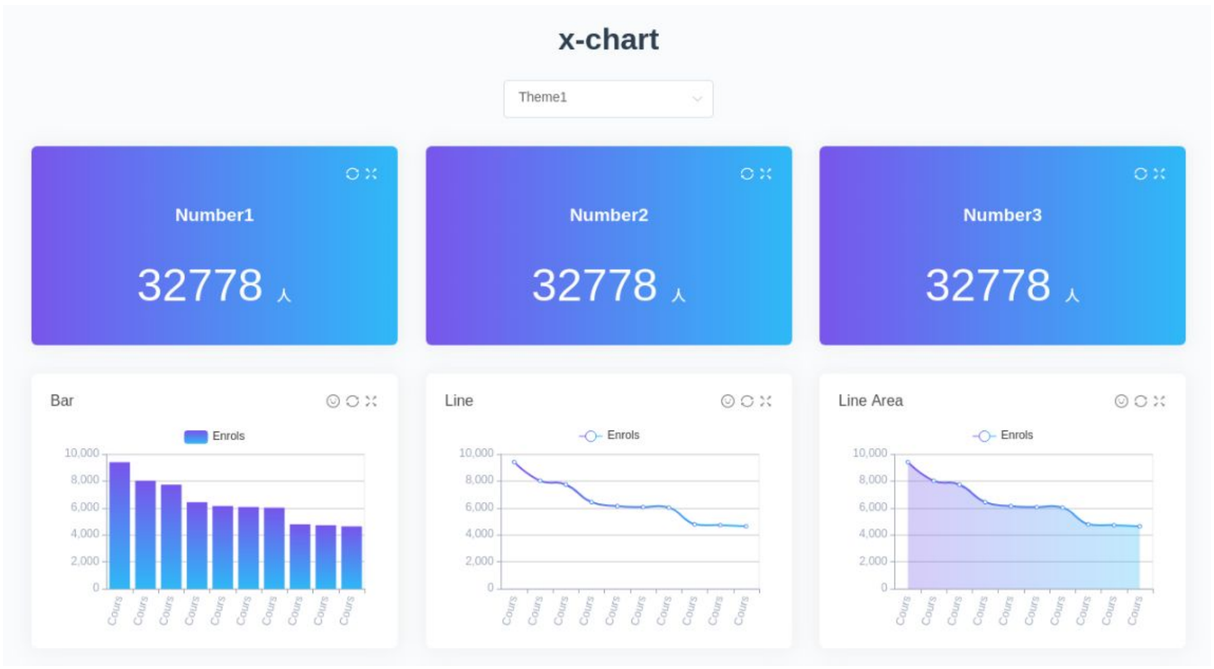


Figure 6: X-Chart Examples

The above X-chart Library allows for dynamic graph scaling and drag-and-drop charts, meaning that large numbers of output graphs could be easily formatted by the user to display in a format they prefer [17].

4.2.2.3 Adding to the Database

Algorithms can be added to the Earth Data Store using the Vue.js auto-formatted entry field libraries [18]. The user can attach their algorithmic code and docker image, the language it's written in, and a brief description of their algorithm, which can then be formatted into a pull request that is sent to the approval team via the Puppet-enabled configuration management process. Then, once the algorithm has passed the approval process, it will be added to the list of possible algorithms in 4.2.2.1's dropdown menu.

5. Future Work

5.1 Definition of the Subsystem

The EDS is a clearly a large project with many moving parts. This section will focus on a deployment plan for a prototype of a small subsystem of the EDS. The substem is the the Bring Your Own Algorithm (BYOA) feature. This feature will allow for scientists to deploy their algorithm to the EDS and run it on the available data sets. Other users will then be able to use and view the output of other algorithms through a UI. This section will discuss the tool setup, the approval process, and finally the business plan to design this prototype.

5.2 Continuous Integration Setup

In Section 4.1 various tools were analyzed and compared for their usage in the proposed cloud-based system. The selected tools for Continuous Integration, Software Configuration Management, and Containerization were Jenkins, Puppet, and Docker respectively. However, for this scope of future work that is being explored in Section 5, we have decided to only require the setup of Continuous

Integration, as it will directly impact the efficiency and quality of the development on the proposed cloud-based system from the first commit to the project.

Now that the Continuous Integration tool has been selected, the required steps to set Jenkins up to work in conjunction with the cloud-based system can be explained.

Setting Jenkins with the project will be a fairly straightforward process. Firstly, we must sign in or create a Jenkins account and install the desired plugins [19]. Once the plugins are installed, a project can be created: click 'New Item' on the Dashboard. Then, "enter a name for the new project … and select 'Freestyle project' as the project type" [19]. The next step is to select the 'Execute shell' build option for the project, as shown in Figure 7.

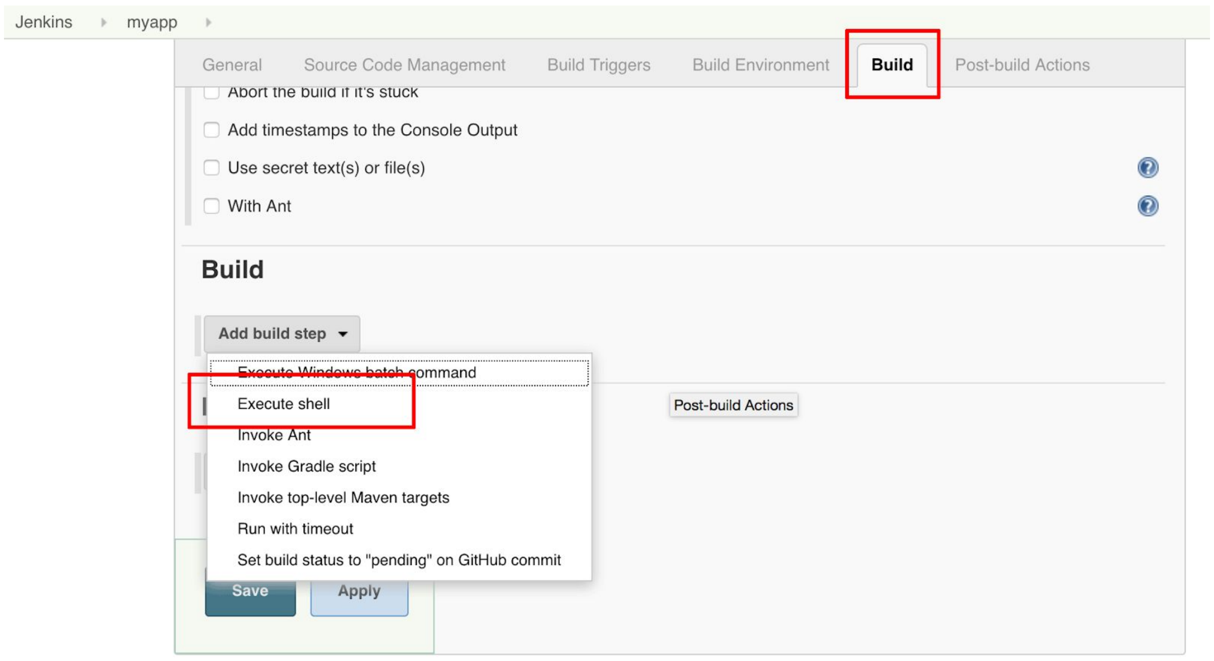
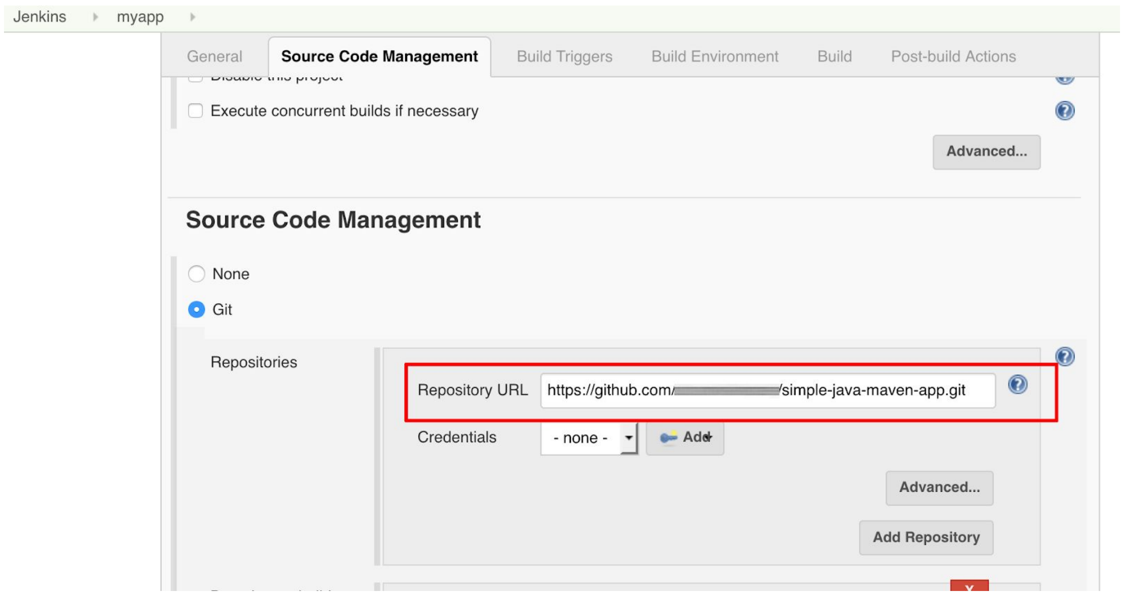


Figure 7: Build Options [19]

After this step, the project must be linked to Jenkins (the next step assumes that the project will be hosted on GitHub as the current One Machine code is). "On the project configuration page, in the "Source Code Management" section, select "Git" as the option and enter the clone URL for your GitHub repository" [19]. This step is displayed in Figure 8.



Execute concurrent builds if necessary

Source Code Management

None

Git

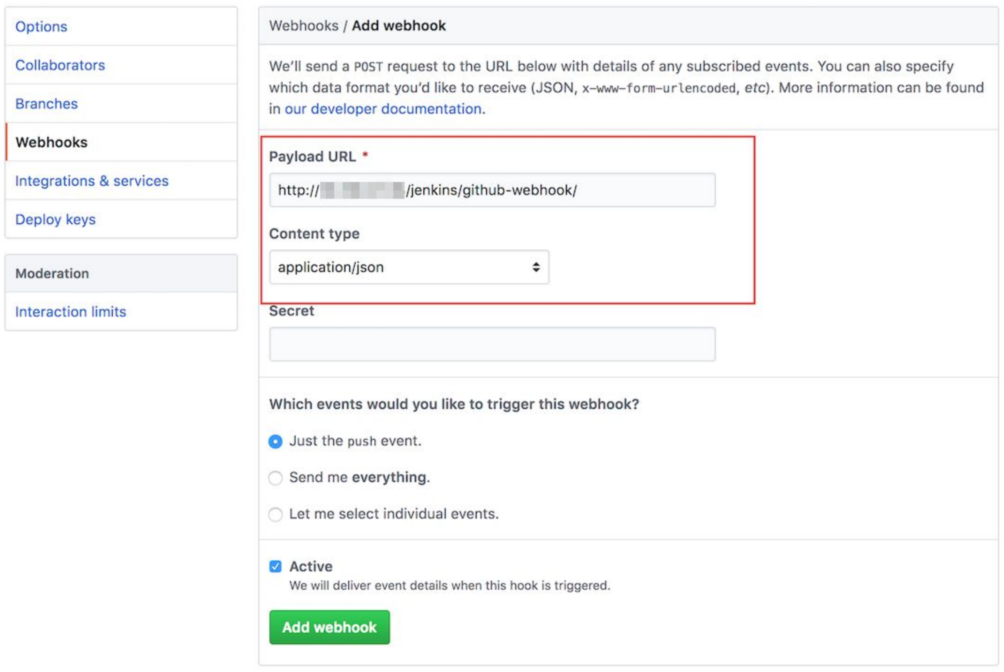
Repositories

Repository URL https://github.com/[redacted]/simple-java-maven-app.git

Credentials - none - Add

Figure 8: Source Code Link [19]

Now, Jenkins is linked to the GitHub project. After this step, a Continuous Integration pipeline can be set up in GitHub, which will trigger a build in Jenkins each time code is committed to the GitHub project. On the GitHub repository, some information must be added to the 'Webhooks' page. Specifically, the Payload URL must be set to the URL for the local Jenkins installation with '/github-webhook/' appended. In addition, the content type must be set to 'application/json'. This step is shown in Figure 9.


Options
Collaborators
Branches
Webhooks
Integrations & services
Deploy keys

Moderation
Interaction limits

Webhooks / **Add webhook**

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *

http://[redacted]/jenkins/github-webhook/

Content type

application/json

Secret

Which events would you like to trigger this webhook?

- [✓ Just the `push` event.]
- Send me **everything**.
- Let me select individual events.

[✓ **Active**]
We will deliver event details when this hook is triggered.

Add webhook

Figure 9: GitHub Jenkins Webhook [19]

The final step is to simply enable the "Github hook trigger for GITScm polling" button in the Build Triggers tab of the project back on Jenkins. After that, Jenkins has successfully been set up for the project and a Continuous Integration pipeline has been activated. Therefore a build will be triggered on each commit and integration tests will be run to determine if the most recent commit breaks any unintended functions of the system.

5.3 Approval Process

The approval process for the BYOA process will consist of two major steps - an automated testing procedure to ensure the algorithm can run, followed by a manual review that verifies that the algorithm runs properly.

The automated testing phase is intended to test "best case" usage, as well as edge cases that may occur to provide a general overview of the algorithm's success. Using Jenkins' automatic build capabilities, a new Docker container should be initialized using the provided image file. Then, the algorithm will be automatically run on the following data set ranges, assuming n entries:

- [0, n]
- [n/2, n]
- [0, n/2]
- [n, n+1]
- [-1, 0]

Should the algorithm initialize without error from its Docker image, produce error-free results from the first three tests, and catch the error conditions produced by the last two tests, the resultant data should be ported to the Vue.js graph builders to produce a final result that can be viewed by the manual testing team.

The manual testing team should provide final approval on each algorithm. Given the provided description, they will ensure that the algorithm is producing the correct output. Although EDS contributor numbers are unknown at this point, an ideal approval process would involve a team of 5 or more reviewers, from which at least half must approve. As well, the contributor should also review their results and approve of them before the algorithm is approved and given a container on the EDS.

5.4 Business Plan

Once the proposed solution has been prototyped, it will be necessary to propose it to the investors and stakeholders in the Canadian Digital Technology Supercluster. The Canadian Digital Technology Supercluster consists of companies, intellectuals, and government officials. It will be important to consult a variety of stakeholders, since all (or most) parties will need to agree that our proposed solution is the ideal solution. This is a very important step in the process, because if the investors and stakeholders do not approve the solution, it will not be feasible to implement it within the final product.

The way in which we will pitch the system must be remote, since most of the stakeholders are not located within our vicinity. Therefore, we believe that the best way to pitch the idea is through a screen sharing demo of the prototype during an online Skype or Google Hangouts meeting with the stakeholders, developers, and investors. We will begin the meeting first by giving a small powerpoint presentation for context and a brief introduction, then we’ll move onto a live demo of the protype, and finally we’ll open up for questions and discussion.

The powerpoint presentation will be mainly focused on why moving to the newly proposed subsystem will be beneficial to scientists, the Government of Canada, and the Canadian people as a whole. In particular, the introduction will highlight the improved efficiency of using the new proposed subsystem, and how the improved UI improves accessibility and usability for users. We also will provide some basic cost benefit analysis in order to persuade investors on why the new subsystem should be adopted financially. To further persuade investor groups the prototype demo will be centred around setting up a user account and accessing the user UI, then applying a custom algorithm to some data, and finally displaying the processed data with the new visualization tooling.

We believe that it is important to keep the introduction and demo fairly high level so to not bog down the investors with complicated technical information. If the investors have any technical questions this will be saved for question and discussion section. We plan on taking Section 4 of the report and sending it to the investor group before the presentation for them to read so they can understanding the tooling that will be needed to create the subsystem. If they have any questions about the technical side it will be saved until this final section of the presentation.

References

[1] "paulagent/Yvonne_class", GitHub, 2019. [Online]. Available: https://github.com/paulagent/Yvonne_class. [Accessed: February 25th, 2019].

[2] Amazon.com. “AWS Database Migration Service,” [Online]. Available: https://aws.amazon.com/dms/. [Accessed: February 20th, 2019].

[3] “radianearth/stac-spec”, GitHub, 2019. [Online]. Available: https://github.com/radiantearth/stac-spec. [Accessed: February 24th 2019]

[4] “radianearth/stac-spec/blob/master/STAC-060-uml”, GitHub, 2019. [Online]. Available: https://github.com/radiantearth/stac-spec/blob/master/STAC-060-uml.pdf. [Accessed: February 24th 2019]

[5] “radianearth/stac-spec/blob/master/item-spec/item-spec.md”, GitHub, 2019. [Online]. Available: https://github.com/radiantearth/stac-spec/blob/master/item-spec/item-spec.md. [Accessed: February 24th 2019]

[6] GeoJSON. “The GeoJSON Specification”, [Online]. Available: http://geojson.org. [Accessed: February 24th 2019]

[7] AWS Blogs. “Keeping a SpatioTemporal Asset Catalog (STAC) Up To Date with SNS/SQS”, [Online]. Available: https://aws.amazon.com/blogs/publicsector/keeping-a-spatiotemporal-asset-catalog-stac-up-to-date-with-sns-sqs/. [Accessed: February 24th 2019]

[8] Edureka. “What is Jenkins,” [Online]. Available: https://www.edureka.co/blog/what-is-jenkins/ [Accessed: February 20th, 2019].

[9] Travis CI, GMBH. (2019). “Travis CI” [Online]. Available: https://travis-ci.com/. [Accessed: February 20th, 2019].

[10] Guru 99. (2019). “Software Configuration Management Tutorial” [Online]. Available: https://www.guru99.com/software-configuration-management-tutorial.html\. [Accessed: February 20th, 2019].

[11] Edureka. (2018, July 23). “What is puppet?,” edureka.co [Online]. Available: https://www.edureka.co/blog/what-is-puppet/ . [Accessed: February 20th, 2019].

[12] Github Inc. (2014, May 14). “Using Github for Configuration Management” [Online]. Available:

https://github.com/BAXTER-IT/yurconf/wiki/Using-github-for-Configuration-Management. [Accessed: February 20th, 2019].

[13] Docker. “What is a Container,” [Online]. Available: https://www.docker.com/resources/what-container. [Accessed: February 20th, 2019].

[14] Dzone.com. (2018, July 30). “Vagrant vs. Docker: Which is Better for Software Development Environments,” [Online]. Available: https://dzone.com/articles/vagrant-vs-docker-which-is-better-for-software-dev. [Accessed: February 20th, 2019].

[15] Docker. “About images, containers, and storage drivers,” [Online]. Available: https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/#sharing-promotes-smaller-images. [Accessed: February 21st, 2019].

[16] Vue.js Examples. “A prettier way to display select boxes,” [Online]. Available: https://vuejsexamples.com/a-prettier-way-to-display-select-boxes/. [Accessed: February 21st, 2019].

[17] Made with Vue.js. “Charts,” [Online]. Available: https://madewithvuejs.com/charts. [Accessed: February 21st, 2019].

[18] Vue.js. “Form Input Bindings,” [Online]. Available: https://vuejs.org/v2/guide/forms.html. [Accessed: February 21st, 2019].

[19] docs.bitnami.com. “Create A Continuous Integration Pipeline With Jenkins And GitHub On Oracle Jump Start,” [Online]. Available: https://docs.bitnami.com/oci/how-to/create-ci-pipeline-jenkins-oracle/. [Accessed: February 24th, 2019].