DocSecOps

Project Members

Laura Casals: lcasals2014@fau.edu

Daniel Ruess: druess2013@fau.edu

Omar Muniz: omuniz2021@fau.edu

Joseph Lee: josephlee2019@fau.edu

Jarold Sabillon: jsabillon2021@fau.edu

Advisor/Professor

Hanqi Zhuang: zhuang@fau.edu

Loi Nguyen: loi.t.nguyen.civ@us.navy.mil

John Latta: john.m.latta.civ@us.navy.mil

Florida Atlantic University

Department of Electrical Engineering and Computer Science

Fall 2022

Project Summary:

DevSecOps is a practice that is implemented to automate security protocols throughout the software development lifecycle (SDLC). Our project's purpose is to create the infrastructure of a DevSecOps ecosystem by utilizing Jenkins, which is an “open source automation server” [1], to verify and validate documents like plain text files, Word documents, PDFs, and PowerPoints. This environment will lead to developing custom plugins to check grammatical, spelling, and broken link errors.

Table of contents

1. Introduction

1.1. Problem Description

DevSecOps stands for **Dev**eloper, **Sec**urity, and **Op**erations. This practice is used to automate security protocols throughout the life cycle of software instead of implementing them at the end. Incorporating automated security features provides developers with insights of the code's viability by providing feedback when bugs or errors occur. Through DevSecOps, awareness of issues during each phase of development is highlighted so developers can address them earlier. By implementing security features, the amount issues towards the end of the software's lifecycle are lessened. This helps to promote a secure environment that facilitates continuous development and integration.

For our project, a DevSecOps pipeline will be configured to handle distinct types of documentation. The pipeline will support plain text files, Word documents, PDFs, and PowerPoints. The problem, however, is that current pipelines are optimized for source code control, syntax error checking, code formatting standards, etc. There is no existing pipeline software that takes documentation from a git repository and executes actions on it. Our group will be constructing the infrastructure of a DevSevOps ecosystem that allows the passing of documentation through a pipeline. Since there are no current methods to process documentation in this manner. Our first challenge will be developing and implementing a process to handle these documents as they are uploaded. To accomplish this, software will need to be created to handle preprocessing the text from the files and converting it into input strings. Plugins to process the input string data can then be integrated to the pipeline. Through plugins, future validity checks for the documents can be executed. For example, plugins for grammatical, spelling, or broken links may be integrated in the future. It is imperative that the pipeline that is configured be able to seamlessly support the addition of these future plugins. Dashboards may then be implemented to provide users with a visual representation of what is occurring within the document. The user will then be made aware of any issues and recommended to update the file prior to resubmitting them to the pipeline.

1.2. Significance of the Problem

Initially, security was not an imperative part of software development, but instead was implemented at the end of development. Not including security features until the end can cause many issues, especially now that software updates are being released more frequently. According to IBM, as developers work towards meeting the demands of software, adding security features at the end of the SDLC only creates more issues [2]. A major issue that can occur are bottlenecks, which is when software gets slowed down by an issue or bug and does not allow for other components to process. This is one of the reasons why DevSecOps became more prevalent in software development because it shifts security from the end of development to be incorporated into every phase. This ensures that as software is being developed issues are consistently being addressed at each stage and developers are aware of what is occurring. This transition can aid with increasing the development of software since running into a bunch of errors at the end of development can take time to fix as well as be an expense to the company. Then once the errors are fixed, the tests will need to be performed on it again.

For the scope of our project, we will be designing a DevSecOps environment that supports the validation and verification of documentation. This will be significant for our sponsors the Naval Education and Training Command (NETC) as this will improve the documentation used to recruit and train individuals for the United States Navy. The goal of the NETC is to train individuals in the art of warfare and develop the skills required to be a member of the Navy [3]. This training requires the distribution of various documents to help individuals learn necessary skills to be successful in their field. Proper documentation is vital as the NETC interacts with thousands of people who are interested in recruitment [3]. This automated pipeline will provide the Navy with the ability to seamlessly update and publish their training material. This is because handling documents like code would provide continuous updating, error checking, and publishing of documentation. Currently, there is no existing solution for this problem and will require our team to fabricate this environment. Our team will be creating this DevSecOps environment initially. This will allow for a starting point of the future creation of plugins that may be used to check for grammatical, spelling, or broken links. This will help the NETC to ensure that the documents they distribute for recruitment and training are accurate and up to date.

1.3. **Goals and Objectives**

For this pipeline to properly process distinct types of files, an environment will be created to facilitate the passing of plain text files, Word Documents, PDFs, and PowerPoints. For this DevSecOps pipeline to be successful a strict list of core requirements needs to be followed. It is essential that our ecosystem can handle the passing of the previously mentioned data types. This may lead to the development of future plugins that will be used as security protocol before being stored in an artifact repository. Below is the list of requirements we deemed necessary, followed by the team members who will be responsible for completing them:

I. Implement a DevSecOps pipeline that will be connected to a git repository. This repository will act as the location where the user will upload files that need to enter the pipeline. The user shall only need to upload the file. This will be handled by Omar Muniz.

II. Design and implement a Python script that will detect which types of files are entering the pipeline to differentiate them between plain text files, word documents, PDFs, and PowerPoints. This will be necessary for the future implementation of plugins that will perform different tasks depending on the file type. This will be handled by Jarold Sabillon and Joseph Lee.

III. Design and implement Python scripts that will grab the information from the files and convert the text into a string. This will be necessary for the development of future plugins that will be needed to check the documents for any present errors. This will be handled by Joseph Lee and Jarold Sabillon.

IV. Configure the pipeline to work with the documents and premade plugins that will provide feedback to the user that the file was successfully grabbed from the repository, the text was grabbed and converted to a string, then the files are output to antifactory. This will be handled by Daniel Ruess and Omar Muniz.

V. Implement an artifact repository that will store the files that are entered into the pipeline once they have successfully passed all the necessary tests. This will be where the user can access the files. This will be handled by Laura Casals.

1.4. **Literature Survey**

Currently there are no solutions that provide the functionality to meet our requirements. There is software that provides features like versioning and live error checking for documents. For example, Microsoft Word has spelling and grammar checking, but it's only usable while a user is editing a document. Word also has versioning which provides a history of a document's changes which is like a git repository's version control [4].

Simuldocs is another software solution that provides versioning. This software sets itself apart from the built in Word implementation by making it simpler for multiple people to edit a document. It has a simple GUI that allows users to merge changes and view all changes made by different people [6].

Another piece of software that we can look at is SonarQube. SonarQube is a static analysis tool for code. It runs quality checks such as scanning for potential bugs, code duplication, complexity, and lack of test coverage on code. It also easily integrates with a pipeline [5]. Unfortunately, SonarQube is only used for code and none of the quality checks can run on document file types.

What is missing from existing solutions is the ability to upload a document of any file type, run automated checks on it, and upload it to a repository. Existing solutions also don't have the element of continuous deployment which a pipeline provides.

2. Proposed Design

2.1. Project Requirements

The following outlines the major requirements for the DevSecOps system. They are organized into three main sections: Functional, Usability, and Safety Requirements.

2.1.1. Functional Requirements

F.1. The system shall take input files from a git repository

F.2. The system shall allow for plain text files, Word Documents, PDFs, and PowerPoints to be entered into the pipeline.

F.3. The system shall detect which file types are entered in the pipeline

F.4. The system shall parse through the entirety of a file and analyze its text content

F.5. The system shall have the ability to add and manage plugins

F.6. The system shall produce a report file on the analysis of the files

F.7. The system shall provide the user the ability to review the analysis report files

F.8. The system shall store processed documents in an artifact repository

F.9. The system shall allow the user to log onto a Jenkins controller

F.10. The system shall allow the user to log onto a JFrog artifactory

2.1.2. Usability Requirements

U.1. The system shall require minimal interaction with the user.

U.2. The system shall notify through email when documents are added to the pipeline

U.3. The system shall allow the user to view what is occurring in the pipeline for each build

U.4. The system shall display insight into vulnerabilities and issues

2.1.3. Safety Requirements

S.1. The system shall display a warning by email or in the pipeline if a secret, such as credentials or other sensitive information, is discovered in the input files.

2.2. Product Design

The DevSecOps system will consist of three main components: a Git repository, a Jenkins CI/CD Engine, and an Artifact Repository.

The entry point of the system is the Git repository. Once a new file is uploaded to this repository, the Jenkins pipeline will be triggered and complete its actions.

The Jenkins Pipeline is the component that will process the input document files. Software will need to be created to detect what type of file was uploaded, and parse through the file to scan its text contents. This can be done by developing custom Jenkins plugins or by calling a script in a pipeline Step. We will implement functionality that will show the flow of data in and display errors in the pipeline output and the Jenkins dashboard.

To check the text data for errors, we will possibly need to integrate software for language processing. If there are any errors detected, the pipeline build will fail and output a report to the Artifact repository. If there are no errors detected, it will successfully go through the pipeline.

The last part of the system is the Artifact Repository. We plan on using Artifactory from JFrog. This component will store a report of any errors that were found. It will also store the input files once they have successfully gone through the pipeline with no errors.

2.3. Block Diagram

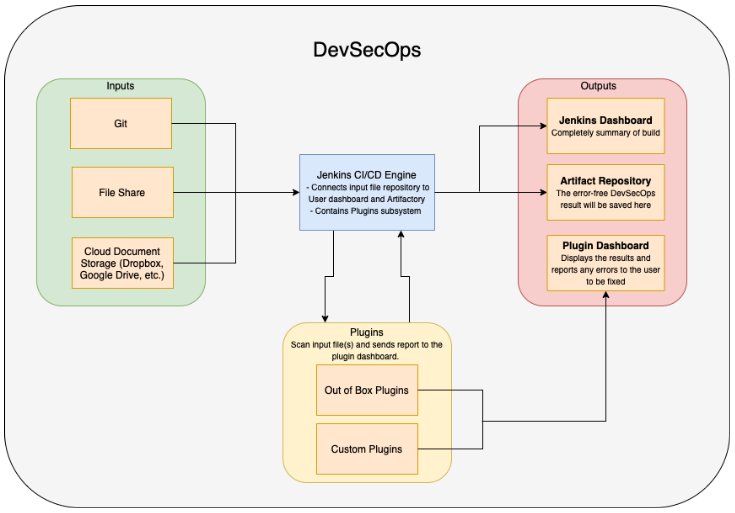

DevSecOps
Inputs
Git
File Share
Cloud Document Storage (Dropbox, Google Drive, etc.)
Jenkins CI/CD Engine
- Connects input file repository to User dashboard and Artifactory
- Contains Plugins subsystem
Outputs
Jenkins Dashboard
Completely summary of build
Artifact Repository
The error-free DevSecOps result will be saved here
Plugin Dashboard
Displays the results and reports any errors to the user to be fixed
Plugins
Scan input file(s) and sends report to the plugin dashboard.
Out of Box Plugins
Custom Plugins

Figure 1: Block Diagram of DevSecOps ecosystem

This block diagram goes over the three main aspects of our project: where the files will be inputted, the Jenkins pipeline, and where the files will be housed. For our project, we are currently utilizing a GitHub repository as our main location to handle input values. The user will upload the file, and this will trigger our pipeline to build. This occurs by GitHub sending a web hook to Jenkins and verifying that the repository defined in the SCM is the same one that sent the web hook. Then it will check if any changes have occurred in the repository. Then the files will travel through the pipeline and are verified by standard plugins. If the file has met all the requirements and passed through the pipeline successfully, it will be deployed to an artifact repository to be housed.

2.4. State Diagram

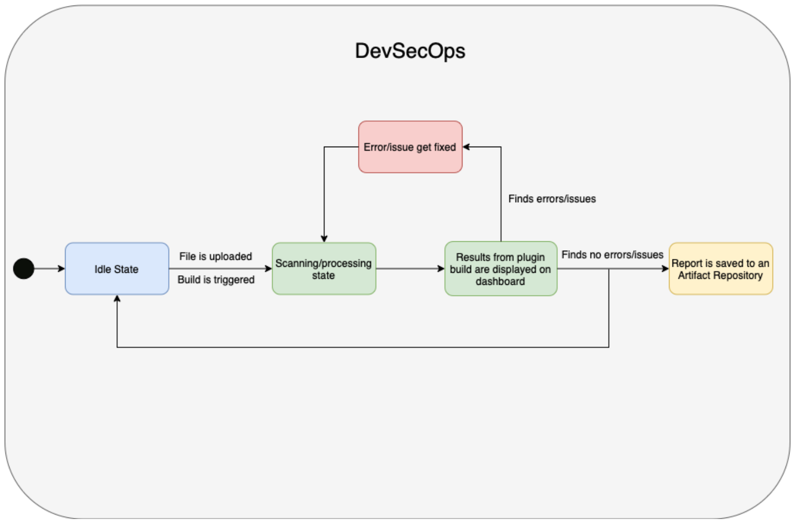

DevSecOps
Error/issue get fixed
Finds errors/issues
Idle State
File is uploaded
Build is triggered
Scanning/processing state
Results from plugin build are displayed on dashboard
Finds no errors/issues
Report is saved to an Artifact Repository

Figure 2: State Diagram of DevSecOps ecosystem

State	Name	Description
1	Idle	The Jenkins Pipeline will continuously be in an idle state, until it gets triggered by a file upload. The Jenkins pipeline will be configured to check the GitHub repository location and check if any files have been added.
2	Scanning/Processing	Once the Jenkins pipeline connects to the repository, Python scripts will detect what type of file has been uploaded. Then will context the contents of the data file into a string to be processed in future plugins.
3	Display	A report will be displayed to the user that will identify if the file was successful taken from the GitHub repository and pushed to Artifactory. If any errors occurred during this process, the build would display that it failed.
4	Error/Issue Found	If an error occurs the document will not be stored in an artifact repository. The user will be notified of the error, to be fixed. Once the error is fixed, the pipeline will begin its automated testing again.
5	Artifactory	If the document has passed all necessary steps, it will be transferred to an artifact repository to be stored. This Is the location where our customer will be able to collect all the documents that passed through the pipeline.

3. Implementation

3.1. Platform / Infrastructural Stack

3.1.1. Virtualization and Container Orchestration Stack

Hardware will be a fully abstracted layer in this implementation. We will be using dedicated hosted infrastructure with a self-deployed bare-metal Kubernetes cluster implementation, leveraging Rancher for K8s management, Longhorn for persistent storage provisioning, MetalLB for bare-metal load balancing and Calico for pod networking. The physical host will be provisioned with Proxmox VE 7 as the hypervisor for virtual machine management.

The physical hypervisor server will be provisioned with an 8 core and16 thread CPU, 64GB of RAM and 3x8TB hard drives, provisioned in a software MDRAID-5 configuration for redundancy with maximized storage capacity. The base operating system will be Debian 11 Linux.

3.1.2. Virtual Machines

For this implementation, five Debian 11 Linux virtual machines will be used as Kubernetes nodes, four as worker nodes and one as a management node. The worker nodes will be given a standard 4 vCPU and 8GB of RAM configuration along with 64GB operating system disks as well as a 2TB data disk for the Longhorn storage backend. The management node will be provisioned with 2 vCPU and 4GB of RAM along with a 64GB operating system disk.

3.1.3. Storage

In order to seamlessly simulate a cloud native storage environment, the choice was made to go with Longhorn as the backend storage provider. The agent runs on every worker node and provides a highly available storage platform, with triple replication per stored block, meaning that the cluster’s storage backend will remain available so long as 2+ nodes are online.

3.1.4. Networking

Due to the nature of this as a non-cloud deployment, additional care needs to be given to the networking stack in order to mimic a more traditional cloud environment. At the highest level, pfSense is being leveraged as the primary firewall and router for the entire deployment – all ingress and egress traffic flows through pfSense. For the Kubernetes stack, Calico is being used for pod-to-pod communication, MetalLB for

Layer 2 load balancing to enable external connectivity to inside the cluster on a bare metal deployment and Ingress NGINX as the primary HTTP/S load balancer for handling service exposure through MetalLB.

3.1.5. Infrastructure Diagram

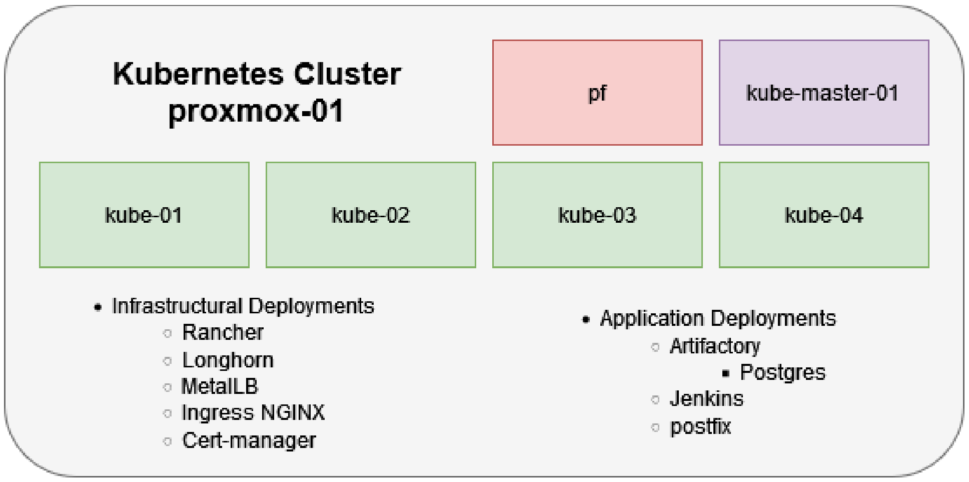


Figure 3: Kubernetes Cluster

3.1.6. Network / Ingress Diagram

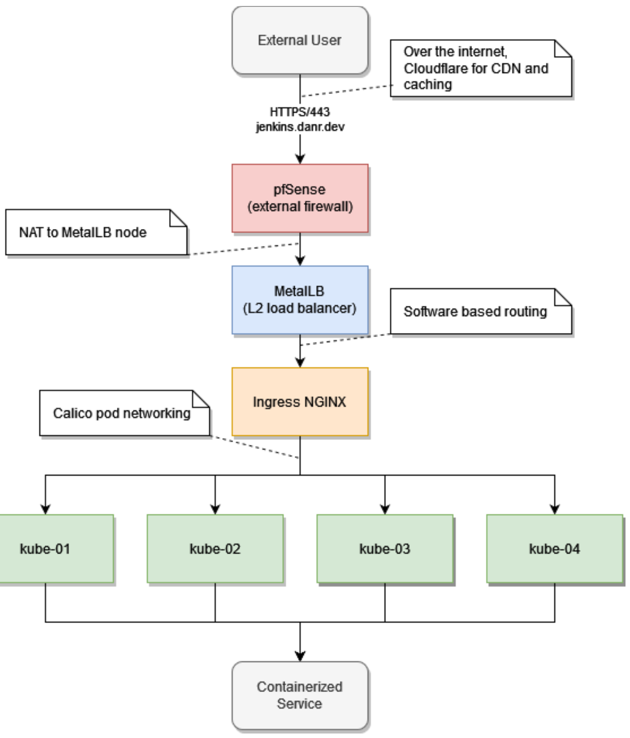


Figure 4: Network/Ingress Diagram

3.2. **Pipeline**

3.2.1. **User Interface and Data communications**

GitHub will provide us with a secure location to drop files. The simple GitHub GUI allows the Navy to really utilize the full features of Git. Jenkins also provides a GUI to help see the pipeline's health, which will be used by the Navy so they can operate it after we leave. Atrifactory also has a great GUI that will help them retrieve their processed data. We then use custom plugins to help transfer data from one parser to another. This allows data communications to move smoothly. Using these custom plugins, we will have to create python scripts which will pass data from one script to another.

3.2.2. **Testing**

Great testing starts with a great test environment, by us using VMs we can easily recreate our environment, but by using docker we will be able to redeploy our environments accurately and securely. We will then use these redeployed test environments and run unit tests on them to try and find certain faults as well as test outliers. We then create a certain outcome as baseline and test against that line validating the results.

4. Development plan and Schedule

4.1. Outline of plan

1.

Our top priority in this project is to set up the pipeline ecosystem. This consists of a way to input files, to process these files with the specified plugins designed for the file type and output the results of the process. The output will contain information about the process that occurred such as the error it came across and where it occurred.

2.

After the Jenkins server's completion, a GitHub repository will be created and connected to it. This will also include the detection of a file being uploaded, which Jenkins will automatically pick it up and begin processing it.

3.

Once the input has been completed, the output will be configured with artifactory. Once Jenkins has completed the processing of the file, it will spit it out into artifactory with details of the process.

4.

The development of plugins will be handled at the very end to handle documents of several types.

5.

Finally, the pipeline will be tested with a variety of document types by the entire team.

4.2. Work breakdown and key Milestones

By the end of the first semester, the pipeline ecosystem will have been completed. The server will be running, and Jenkins will detect a file uploaded from the GitHub repository. It will then be taking this file and simply output it to artifactory. This will show the flow of data within the pipeline ecosystem. The Jenkins server is developed and hosted by Daniel Reuss.

In the following semester, custom and predeveloped plugins for the pipeline will be implemented. But before specific plugins are added, the system will first be set up to detect the type of document being uploaded. If a plugin processes a document type that is not meant for it, it could potentially crash the system or lead to unexpected errors. This portion should take no more than a week to complete. Omar Muniz will connect the repository to the Jenkins server, while Jarold Sabillon and Joseph Lee will work on detecting the specific file type that was uploaded.

Once document type is specified, the document will be processed by specific plugins designed for it. For example, these plugins could be spelling/grammatical corrections, broken link detection, and inappropriate word usage in word documents. The choosing of these plugins will take one week, while the actual implementation of the plugins could take another two weeks to a month. A variety of plugins will be discussed by the entire team which will then be designed and implemented to Jenkins.

When document type detection is selected, the files uploaded will be passed through specific plugins designed for it. We will be designing plugins to work with pdfs, PowerPoint, and word to begin with. Jarold Sabillon and Joseph Lee will be creating the specified plugins selected by the team that will process the documents being passed through.

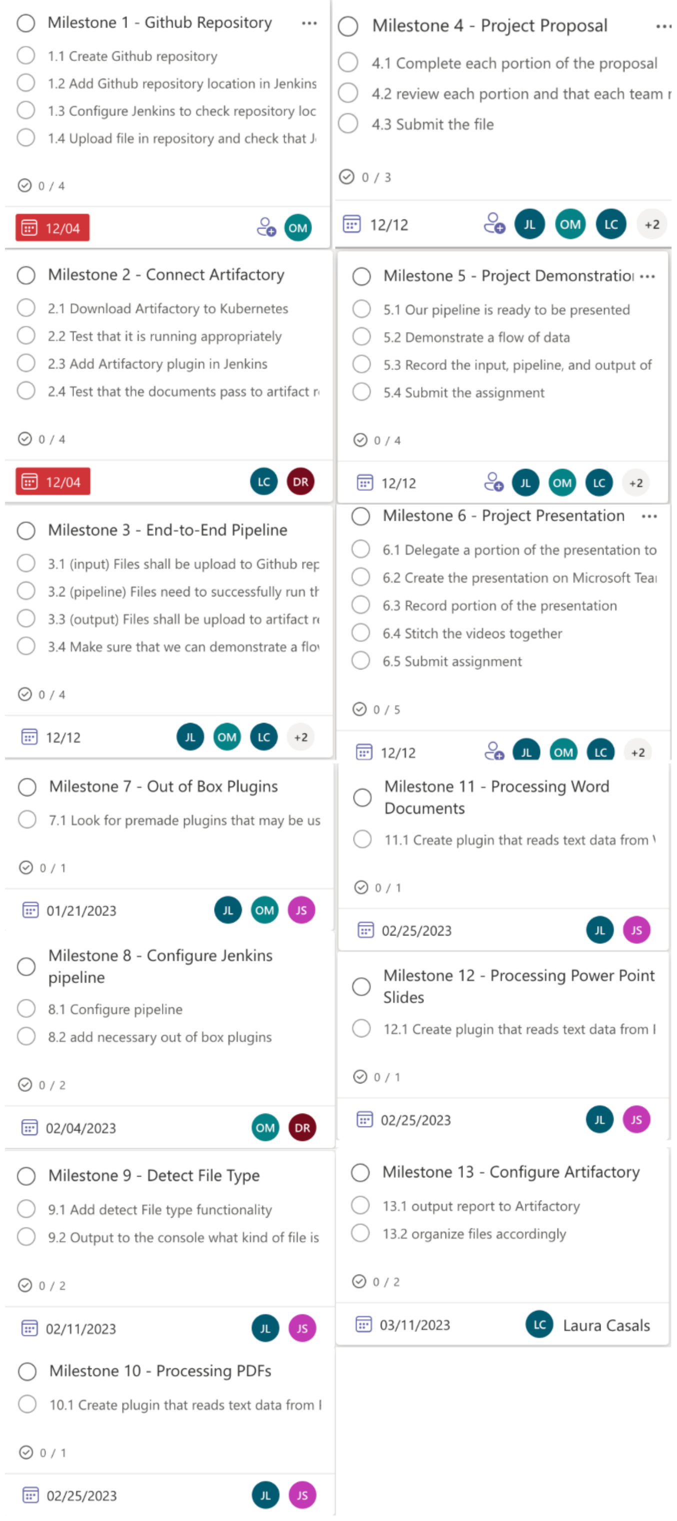
Once plugins are finished, Laura Casals, Daniel Reuss, and Omar Muniz will ensure that they are working properly within the Jenkins pipeline and artifactory. They will ensure all errors are included in the output. This will be followed by a series of tests to ensure the pipeline continues to work properly under different circumstances. Such as multiple files uploaded at once, difference sources, different file types, and how the pipeline will handle a file that has not been considered.

These plugins ensure that documents being sent out to students are free from errors and inappropriate content. Most importantly It will save time from having to reupload and redistribute these documents. It will save resources like time, paper, and ink by preventing these errors from occurring in the first place. Incorrect verbiage could also lead to confusion among students.

Some challenges we are expected to face is that Jenkins is not designed to take documents as input but designed for programming files. There is little if not any plugins available to handle common document types such as pdfs, word, PowerPoint, etc. So, most of these plugins will have to be designed by the team.

The last challenge will be having the server run within the Navys network. Security system restrictions could prevent the server from running or being able to interact with GitHub or artifactory, preventing any flow of data from occurring.

4.2.1. Milestones with deadlines



Milestone 1 - Github Repository
- 1.1 Create Github repository
- 1.2 Add Github repository location in Jenkins
- 1.3 Configure Jenkins to check repository loc
- 1.4 Upload file in repository and check that J

0 / 4

12/04 OM

Milestone 2 - Connect Artifactory
- 2.1 Download Artifactory to Kubernetes
- 2.2 Test that it is running appropriately
- 2.3 Add Artifactory plugin in Jenkins
- 2.4 Test that the documents pass to artifact r

0 / 4

12/04 LC DR

Milestone 3 - End-to-End Pipeline
- 3.1 (input) Files shall be upload to Github rep
- 3.2 (pipeline) Files need to successfully run th
- 3.3 (output) Files shall be upload to artifact re
- 3.4 Make sure that we can demonstrate a flov

0 / 4

12/12 JL OM LC +2

Milestone 4 - Project Proposal
- 4.1 Complete each portion of the proposal
- 4.2 review each portion and that each team r
- 4.3 Submit the file

0 / 3

12/12 JL OM LC +2

Milestone 5 - Project Demonstratio
- 5.1 Our pipeline is ready to be presented
- 5.2 Demonstrate a flow of data
- 5.3 Record the input, pipeline, and output of
- 5.4 Submit the assignment

0 / 4

12/12 JL OM LC +2

Milestone 6 - Project Presentation
- 6.1 Delegate a portion of the presentation to
- 6.2 Create the presentation on Microsoft Tea
- 6.3 Record portion of the presentation
- 6.4 Stitch the videos together
- 6.5 Submit assignment

0 / 5

12/12 JL OM LC +2

Milestone 7 - Out of Box Plugins
- 7.1 Look for premade plugins that may be us

0 / 1

01/21/2023 JL OM JS

Milestone 8 - Configure Jenkins pipeline
- 8.1 Configure pipeline
- 8.2 add necessary out of box plugins

0 / 2

02/04/2023 OM DR

Milestone 9 - Detect File Type
- 9.1 Add detect File type functionality
- 9.2 Output to the console what kind of file is

0 / 2

02/11/2023 JL JS

Milestone 10 - Processing PDFs
- 10.1 Create plugin that reads text data from I

0 / 1

02/25/2023 JL JS

Milestone 11 - Processing Word Documents
- 11.1 Create plugin that reads text data from \

0 / 1

02/25/2023 JL JS

Milestone 12 - Processing Power Point Slides
- 12.1 Create plugin that reads text data from I

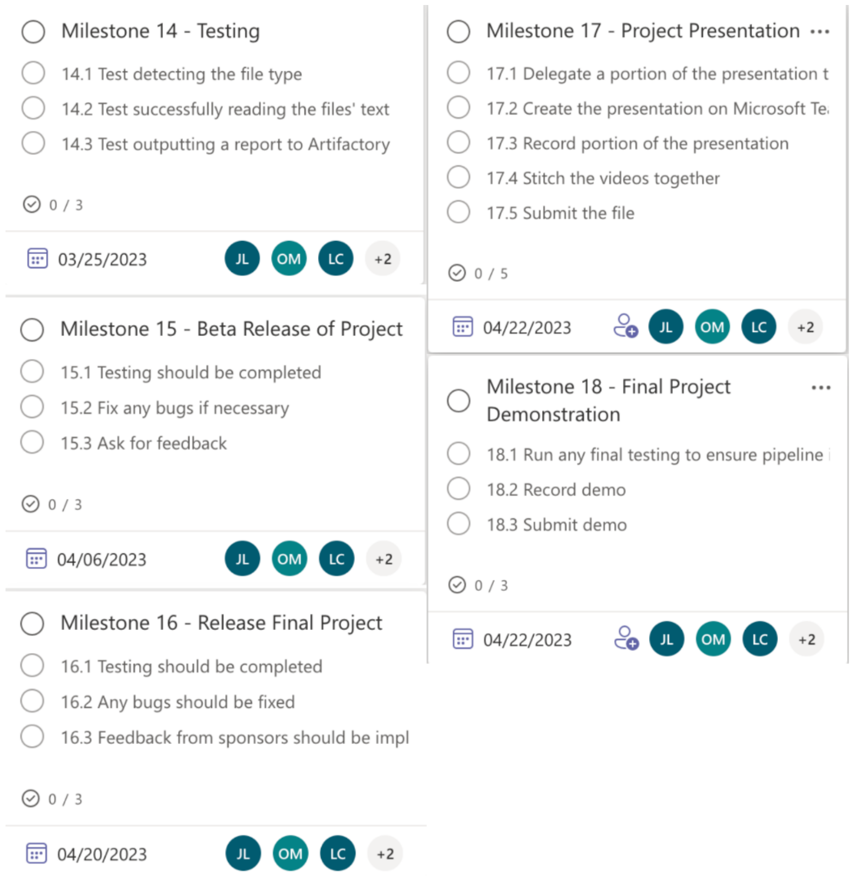
0 / 1

02/25/2023 JL JS

Milestone 13 - Configure Artifactory
- 13.1 output report to Artifactory
- 13.2 organize files accordingly

0 / 2

03/11/2023 LC Laura Casals


Milestone 14 - Testing

14.1 Test detecting the file type

14.2 Test successfully reading the files' text

14.3 Test outputting a report to Artifactory

0 / 3

03/25/2023 JL OM LC +2

Milestone 15 - Beta Release of Project

15.1 Testing should be completed

15.2 Fix any bugs if necessary

15.3 Ask for feedback

0 / 3

04/06/2023 JL OM LC +2

Milestone 16 - Release Final Project

16.1 Testing should be completed

16.2 Any bugs should be fixed

16.3 Feedback from sponsors should be impl

0 / 3

04/20/2023 JL OM LC +2

Milestone 17 - Project Presentation

17.1 Delegate a portion of the presentation t

17.2 Create the presentation on Microsoft Te

17.3 Record portion of the presentation

17.4 Stitch the videos together

17.5 Submit the file

0 / 5

04/22/2023 JL OM LC +2

Milestone 18 - Final Project Demonstration

18.1 Run any final testing to ensure pipeline

18.2 Record demo

18.3 Submit demo

0 / 3

04/22/2023 JL OM LC +2

4.3. Budget

We are using free open-source software to run our project. No expenses will be necessary. The server is being run and handled by a teammate with no charges.

References

[1] Author unknown. (Date Unkown). *Jenkins*. [Online]. Available: https://www.jenkins.io/

[2] IBM Cloud Education. (2020, July 30). *What is DevSecOps?*.IBM. [Online]. Available: https://www.ibm.com/cloud/learn/devsecops

[3] Author unknown. (Date Unknown). *Naval education and training command*. [Online] Available: https://www.netc.navy.mil/

[4] Author unknown. (Date Unknown). *Restore a previous version of an item or file in SharePoint. Microsoft Support*. [Online] Available: https://support.microsoft.com/en-us/office/restore-a-previous-version-of-a-file-without-unwanted-changes-bdb2cafa-d588-475c-97d7-20e8b9949b84

[5] Dissanayake, Kasun. (2020 June 20). *SonarQube(Part2)*. [Online] Available: https://medium.com/swlh/sonarqube-part-2-features-of-sonarqube-installation-and-some-practice-on-sonarqube-d523ae9a998a

[6] Author unknown. (2022 June 17). Three better ways to version control your word documents. Simul Docs. [Online] Available: https://www.simuldocs.com/blog/3-better-ways-to-version-control-your-word-documents