|  |
| --- |
|  |
| GITLAB |
| **Continuous Integration (CI) – Runbook** |

**Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Ver** | **Updates** | **Author** | **Reviewer** | **Approver** |
| 20-Jun-22 | 0.1 | Initial draft | Binoy V | Prasanth P | Mathews |
| 29-Jun-22 | 0.2 | Changes specific to CI/CD | Binoy V | Prasanth P | Mathews |
| 09-Sep-22 | 0.3 | Created separate runbook for CI, layout and content changes | Binoy V | Ramana |  |
| 14-Sep-22 | 0.4 | Added Contents & comments | Prasanth P |  |  |
| 19-Sep-22 | 0.7 | Added Comments and Contents | Prasanth P |  |  |
| 27-sep-22 | 0.8 | Added MR approval process and contents | Binoy V |  |  |

**Contents**

[1 Introduction 3](#_Toc115204833)

[2 GitLab Architecture Overview 3](#_Toc115204834)

[2.1 Components 4](#_Toc115204835)

[3 GitLab (CI) Integration with JFrog and Xray 4](#_Toc115204875)

[4 JFrog Artifactory 5](#_Toc115204877)

[4.1 Artifactory Structure 6](#_Toc115204878)

[5 Continuous Integration 7](#_Toc115204885)

[5.1 Prerequisites 7](#_Toc115204886)

[5.1.1 GitLab Folder Structure 7](#_Toc115204887)

[5.1.2 Sonar Cloud Configuration and Tokens 7](#_Toc115204888)

[5.2 Environment Setup 8](#_Toc115204889)

[6 Proposed GitLab Folder Structure 9](#_Toc115204985)

[7 How to Configure Group Variables in GitLab 12](#_Toc115204986)

[8 Create/Build Continuous Integration (CI) Pipeline Script 14](#_Toc115204987)

[9 How to Configure Script in each Repository 19](#_Toc115204991)

[10 How to Run CI pipeline Scripts 20](#_Toc115204992)

[11 Scripts to Deploy Product Images 22](#_Toc115204993)

[12 How to Run CI Product Deployable Scripts 22](#_Toc115204994)

[13 MR Approval process 24](#_Toc115204995)

[13.1 Sonar check on Merge Request and fail the Merge if Sonar fails 30](#_Toc115204996)

[13.2 EMAIL Notification 31](#_Toc115204997)

[14 Pipeline Stages – End State 31](#_Toc115204998)

[15 APPENDIX 31](#_Toc115205007)

# Introduction

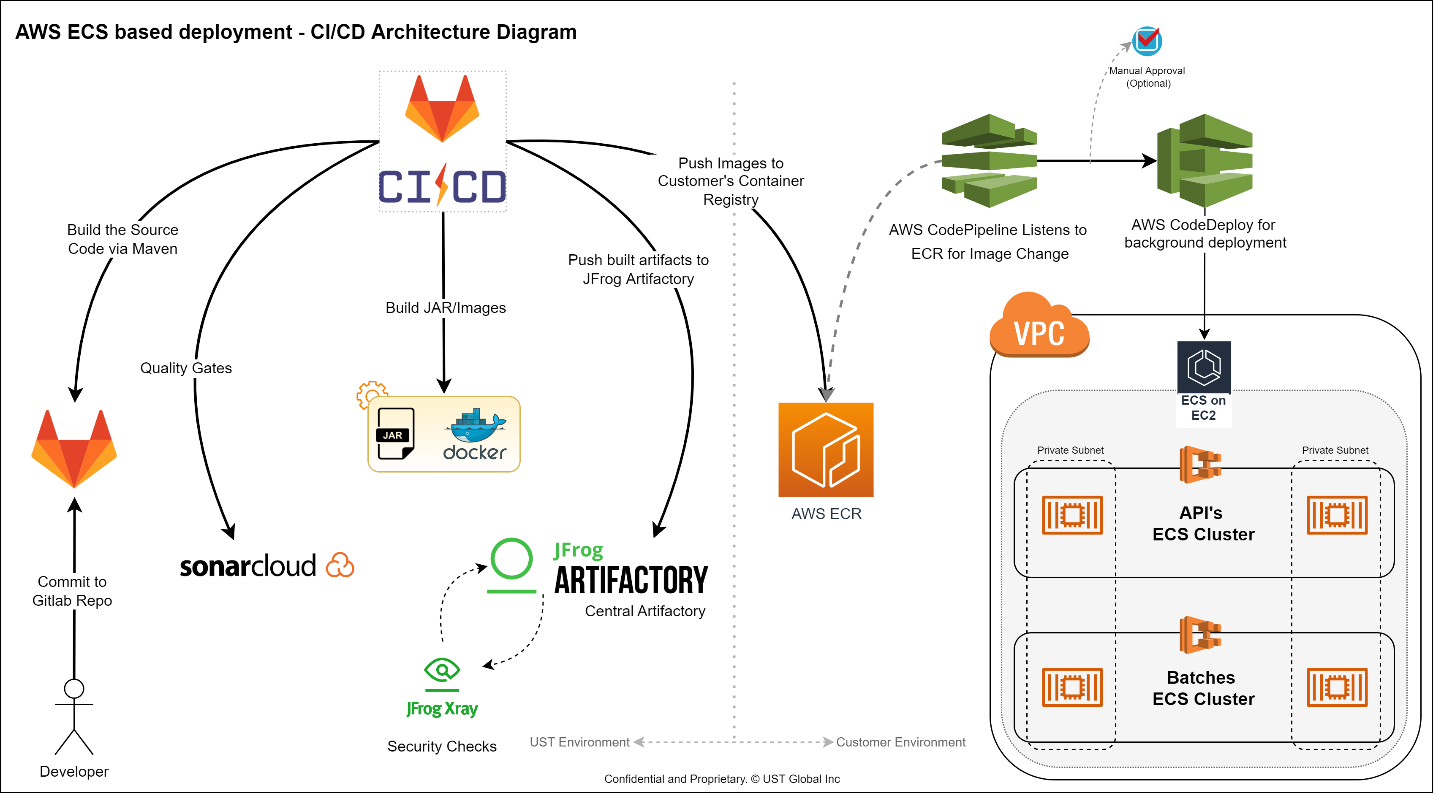
GitLab Continuous Integration (CI) Runbook helps users to configure pipeline based on USTHealthProof product specific standards and recommendations to automate builds.

* Runbook is designed by CIS DevOps/SRE and Enterprise Architect teams
* Runbook provides how Gitlab CI pipelines can be integrated to perform different stages of pipeline and finally pushing the produced artifact to a Container registry.

# GitLab Architecture Overview

Architecture diagram provides end-to-end workflow of Continuous Integration(CI) with GitLab and Continuous Deployment(CD) to cloud platforms or on-premises setups. Diagram below shows the orchestration of GitLab CI to produce the artifacts, do vulnerability and other violations checks and finally push the produced artifact to AWS cloud platform for deployment. The CD part is handled by the tools with in the Cloud or on-premise platform.

In the below diagram the left side deals with Continuous Integration (CI) and right-side deals with Continuous Deployment (CD). CI is owned and managed by USTHealthProof regardless of the implementation or hosting solutions. The CD part depends on the hosting solution and would be either owned and managed by USTHealthProof or the customer.



**(GitLab Integration with AWS – CICD Pipeline)**

## Components

* **GitLab Repo** – GitLab is used as the Code Repository
* **GitLab CI/Pipeline** – Continuous Integration configuration and execution will be through GitLab CI service
* **Sonar Cloud** – Quality Checks would be handled via Sonar Cloud
* **Maven** – Is used to build java-based artifacts
* **Docker** – Is used to containerize the produced java artifacts
* **JFrog Artifactory** – Used as the centralized artifactory for storing all the produced artifacts
* **Xray** – Used to identify security violations like, vulnerability, license compliance etc.



# GitLab (CI) Integration with JFrog and Xray

Architecture diagram provides GitLab CI integration with JFrog Artifactory, where it stores artifacts, images and Xray is used for security scanning & license compliance checks.



**(GitLab Integration with JFrog)**

The integration is based on below high-level steps -

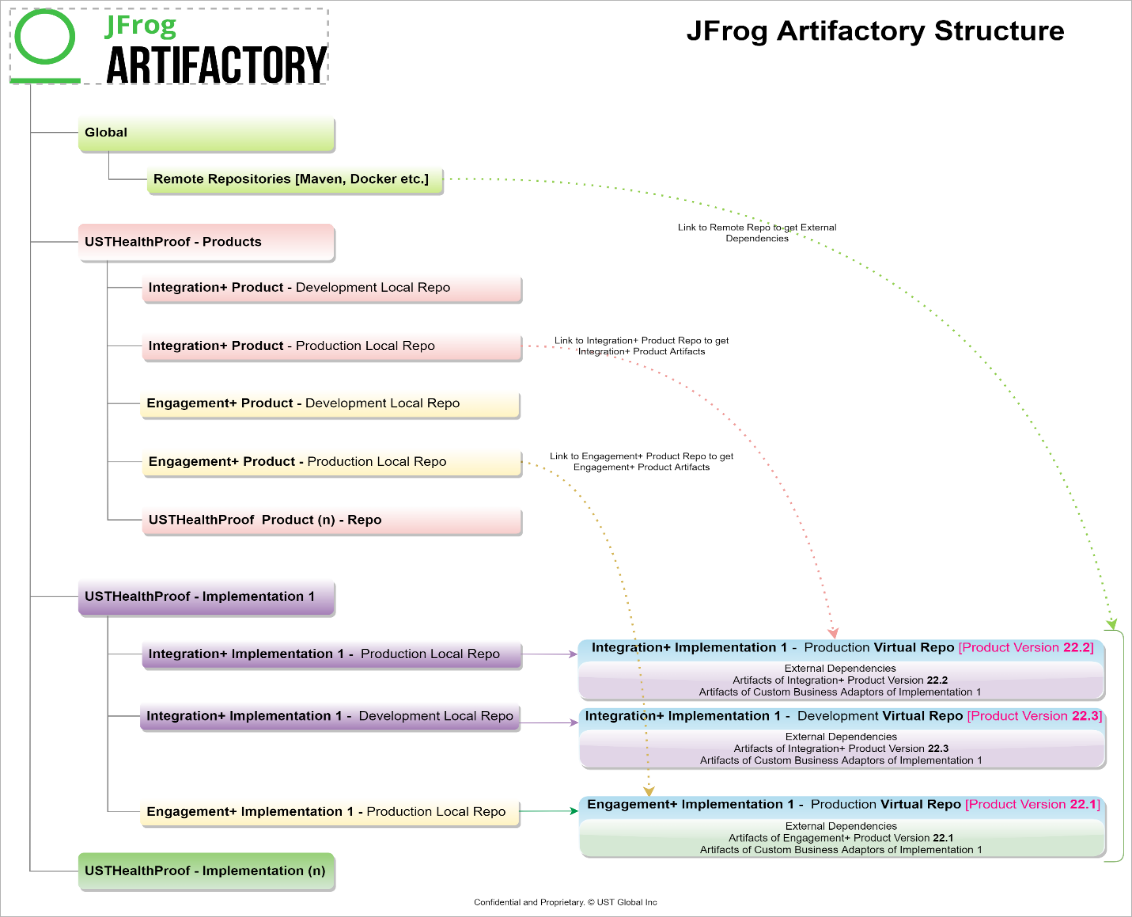
* Developer creates merge request and merges code to GitLab Repo
* GitLab Pipeline will get initialized on successful merge request
* Pipeline will start executing various stages, which includes.
  + Quality gates
  + Versioning
  + Build jar or docker image
  + Upload images/jars to JFrog Artifactory
  + Scan build/artifacts
  + Push produced Images to AWS ECR (when AWS as hosting solution)

# JFrog Artifactory

**JFrog Artifactory** is the single solution for housing and managing all the artifacts, binaries, packages, files, containers, and components for use throughout the DevOps cycle. JFrog Artifactory serves as the central hub for DevOps, integrating with the tools and processes to improve automation, increase integrity, and incorporate best practices along the way.

**JFrog Xray** is an application security SCA tool that integrates security directly into the DevOps workflows, enabling to deliver trusted software releases faster. JFrog Xray fortifies the software supply chain and scans the entire pipeline from your IDE, through your CI/CD Tools, and all the way through distribution to deployment.

## Artifactory Structure



**(JFrog Artifactory Structure)**

The JFrog Artifactory has three kinds of Repositories. Remote, Local and Virtual

* **Remote Repo** – Is used to configure the external repo from internet. JFrog will cache this internally. The same remote repo will be used globally for all Projects. E.g., Maven2 repo
* **Local** **Repo** – Is used to deploy produced artifacts and images
* **Virtual** **Repo** – VRs are the combination of both local and remote repos

The Product Development and various Implementations are grouped within JFrog as several JFrog Projects. At the Product Development Projects, each product will have its own artifactory. There will be repositories for production and non-production code. During the development cycle, non-production repositories are used, and after the code is baselined, the created artifacts are deployed to a production repository.

Every implementation would also be associated with a JFrog Project, and users would receive repositories for each product they chose to utilize. By mirroring or sharing Product's artifactory from the JFrog, Product will share the implementation's repositories with the production-quality images. This indicates that the corresponding implementations have access to all the necessary Product pictures via the implementation's repository. Implementations must decide which Product Images are required based on the release notes that the Product team has given.

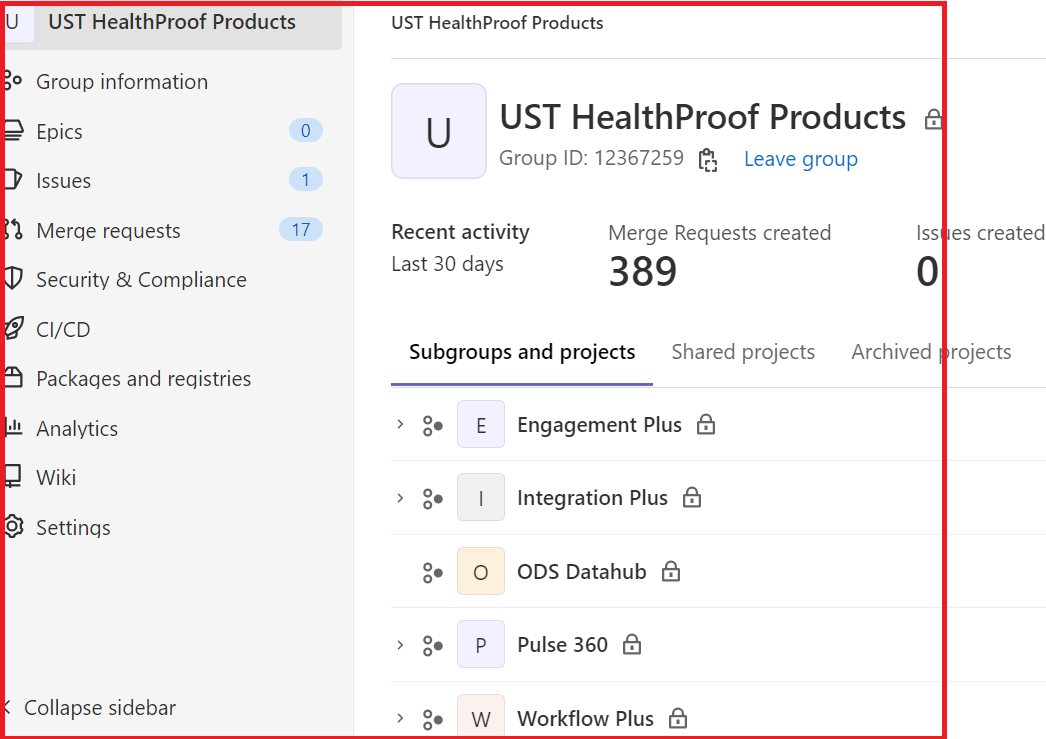
Similar to product development, creation of custom business components or customization of product components on top of implementations takes place in the repositories assigned to each implementation.

# Continuous Integration

## Prerequisites

### GitLab Folder Structure

We have four main Products (IntegrationPlus, Engagement-Plus, Workflow Plus , Plus-360) configured for CI Pipeline. Below folder structure is related to Integration Plus.



### Sonar Cloud Configuration and Tokens

To configure Sonar cloud, First step is to Provision the Sonar Cloud and create the project keys & token. Discuss this with Application developers and owner of the application and define in the Quality check stage along with maven command(Please refer config.yml file (Pls refer section 8 and Appendix section 15. 1.3)

### JFROG Configuration and Tokens

We are using SaaS based JFrog. Get the JFrog Artifactory URL, Server, Project token, keys from the Devops/Architect team(Please refer in Variables Section 5.2 Environment Set up).

## Environment Setup

The following pre-requisites and configurations are required to create environment setup and configure Continuous Integration pipelines: -

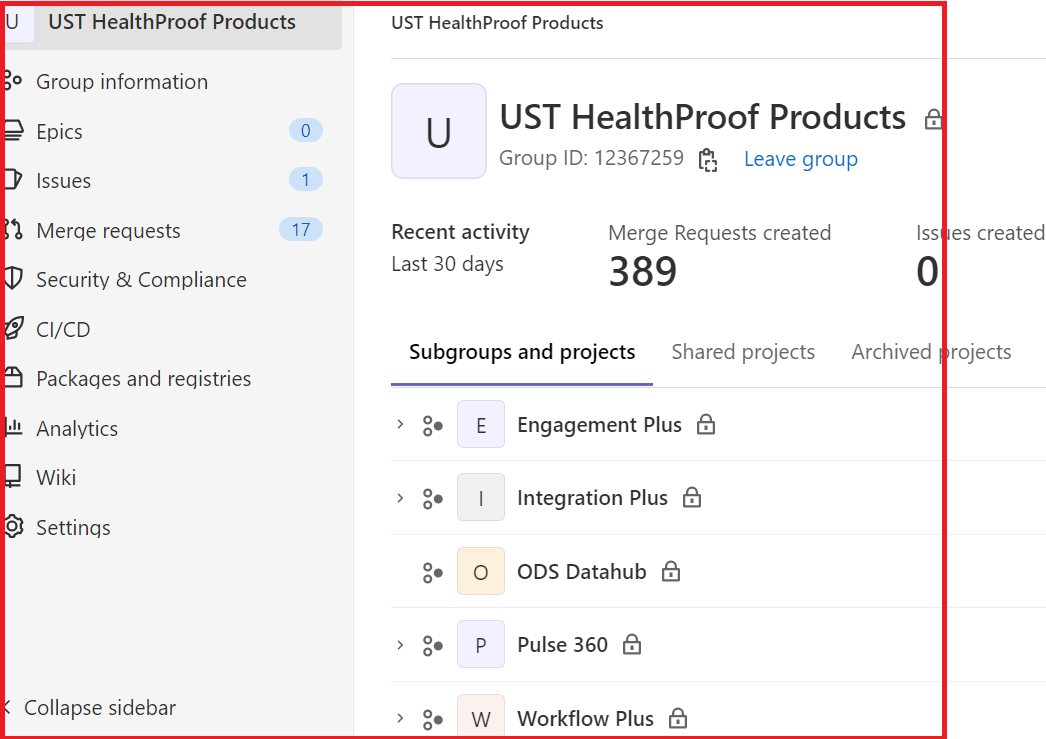
* Understand the high level architect and flow(plz refer section 2, 3 and 4)
* JFrog Projects, Repositories and URLS to be integrated and configured with GitLab
* Discuss and finalize the Gitlab Branching Strategy with the development team
* Discuss with development about standalone application , build versions ,build strategy to be implemented in CI Script and design architecture to create end-to-end flow of CICD
* Discuss about containerized applications build strategy using Docker to create pipelines and for the builds
* Discuss and define stages specific to testing, 3rd party integrations, approval process to be included in the pipelines
* Ensure and get the Programmatic Token access for JFrog, AWS account, Gitlab access in place - (Token access and passwords need to set in Group Variables). Please refer CI group variables in below table.
* Gather thecdevelopment team to svfors below table for more details)
* Gather the Global Variables from DevOps/Architect team to setup product/project specific repos
* Identify the group variables and values to be configured; recommended to set all variables at group /subgroups level, so that variables can be reused for all repos
* Configure the tools with group variables specific to JFrog, Docker and AWS ECR

|  |  |  |
| --- | --- | --- |
| **Variables used in the project implementation – Complete list** | | |
| **KEY** | **VALUE** | **REMARKS** |
| ARTIFACTORY\_CLI | ${ARTIFACTORY\_URL}artifactory/remote-repo-jfrog-cli/v2-jf/scripts/getCli.sh | JFROG ARTIFACTORY URL |
| ARTIFACTORY\_PASS | \*\*\* |  |
| ARTIFACTORY\_PASS\_BASE64 | \*\*\* | JFROG ARTIFACTORY USERNAME: PASSWORD Encoded Base |
| ARTIFACTORY\_PROJECT\_KEY | Usthps | JFROG PROJECT KEY |
| ARTIFACTORY\_SERVER | Usthealthproof | JFROG ARTIFACTORY SERVER |
| ARTIFACTORY\_URL | https://$ARTIFACTORY\_SERVER.jfrog.io/ | JFROG ARTIFACTORY URL |
| ARTIFACTORY\_USER | usthps\_intplus\_cicd\_user | JFROG ARTIFACTORY USER |
| AWS\_ACCESS\_KEY\_ID | \*\*\* | AWS ACCESS KEY ID FOR CICD USER |
| AWS\_DEFAULT\_REGION | us-east-2 | AWS DEFAULT REGION |
| AWS\_ECR\_REGISRTY | 581303157279.dkr.ecr.$AWS\_DEFAULT\_REGION.amazonaws.com | AWS ECR REGISTRY |
| AWS\_SECRET\_ACCESS\_KEY | \*\*\* | AWS SECRET ACCESS KEY |
| DOCKER\_AUTH\_CONFIG | {  "auths": {  "$ARTIFACTORY\_SERVER.jfrog.io": {  "auth": "$ARTIFACTORY\_PASS\_BASE64"  }  } } | AUTHENTICATION FOR DOCKER PRIVATE REGISTRY |
| DOCKER\_FILE | FROM eclipse-temurin:11.0.16\_8-jre MAINTAINER UST HealthProof COPY target/\*.jar /app/app.jar VOLUME /app/vols WORKDIR /app ENTRYPOINT ["sh", "-c", "java $JAVA\_OPTS -jar app.jar"] | COMMON DOCKER FILE |
| DOCKER\_IMAGE\_TAG | docker:latest | DOCKER IMAGE TAG |
| DOCKER\_NONPROD\_REPO | usthps-intplus-nonprod-docker-vr | DOCKER NONPROD REPO(VIRTUAL) |
| DOCKER\_PROD\_REPO | usthps-intplus-prod-docker-vr | DOCKER PROD REPO(VIRTUAL) |
| DOCKER\_REGISTRY | $ARTIFACTORY\_SERVER.jfrog.io | JFROG DOCKER REGISTRY (PRIVATE) |
| MAVEN\_IMAGE\_TAG | maven:3.8.6-eclipse-temurin-11-alpine | MAVEN IMAGE TAG |
| MAVEN\_NONPROD\_REPO | usthps-intplus-nonprod-maven2-vr | MAVEN NONPROD REPO(VIRTUAL) |
| MAVEN\_PROD\_REPO | usthps-intplus-prod-maven2-vr | MAVEN PROD REPO(VIRTUAL) |
| SONAR\_HOST\_URL | [https://sonarcloud.io](https://sonarcloud.io/) | SONAR URL |
| SONAR\_TOKEN | \*\*\* | SONAR TOKEN |



# Proposed GitLab Folder Structure of Product and Implementations

We have four main Products (IntegrationPlus, Engagement-Plus, Workflow Plus , Plus-360) configured for CI Pipeline. Below folder structure is related to Integration Plus.

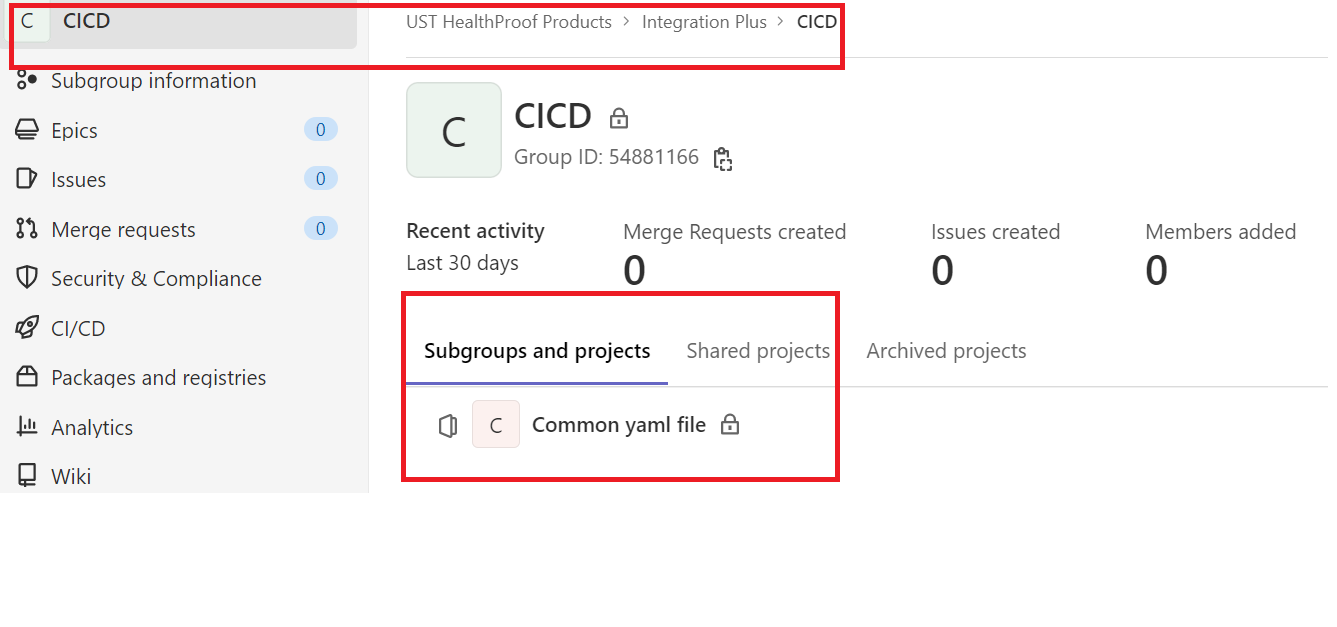


All the CICD Scripts are in the Product (Integration Plus) 🡪 CICD🡪 Common Yaml File .

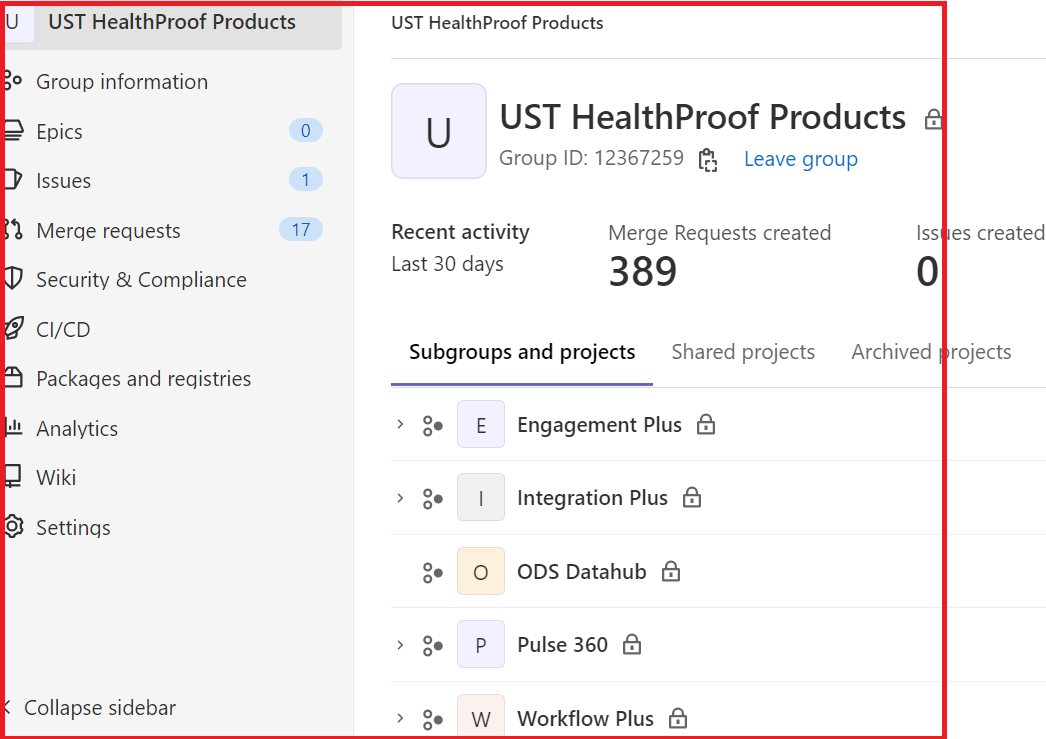
**Integration Plus**

* + CICD
    - Common CI
      * Customization Components
        + config-ci.yml>> Full Script to build and export images, along with quality checks and other stages.
      * Product
        + config-ci.yml>> Script to deploy Product Images

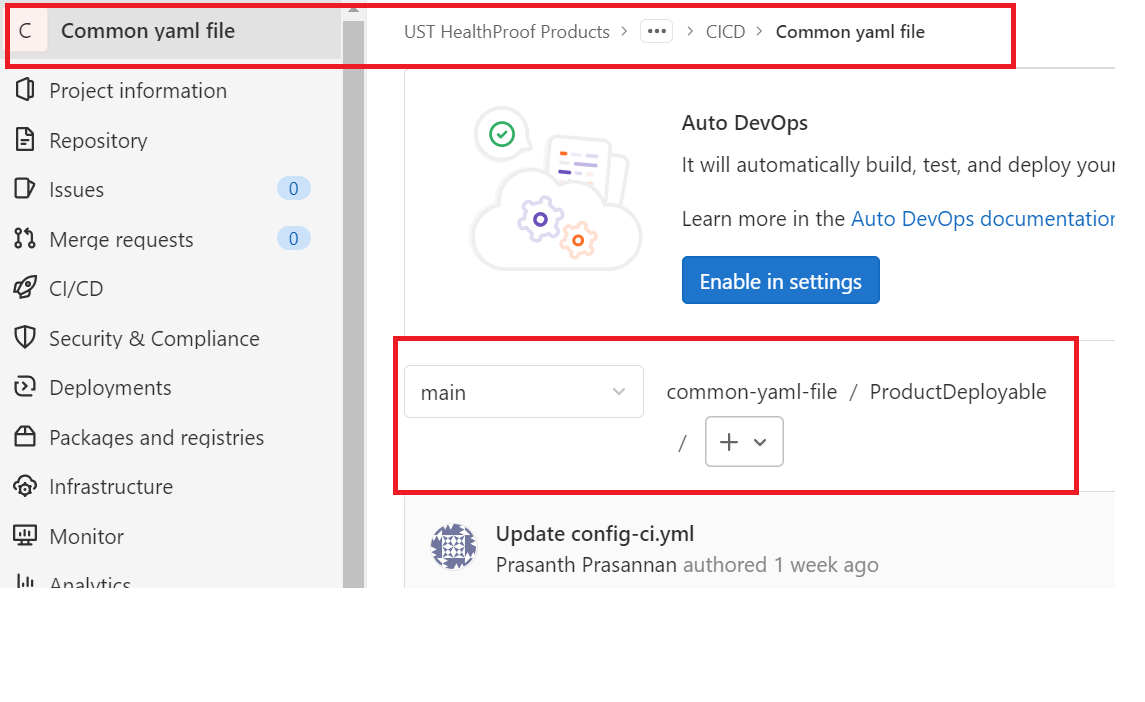
The purpose of CICD is keep all the CICD scripts in a Central location. Later we can point this file and branch which is in Global can configure in each repository where we need to run the pipeline.



* + Business Components



* + Deployables
    - Images.txt

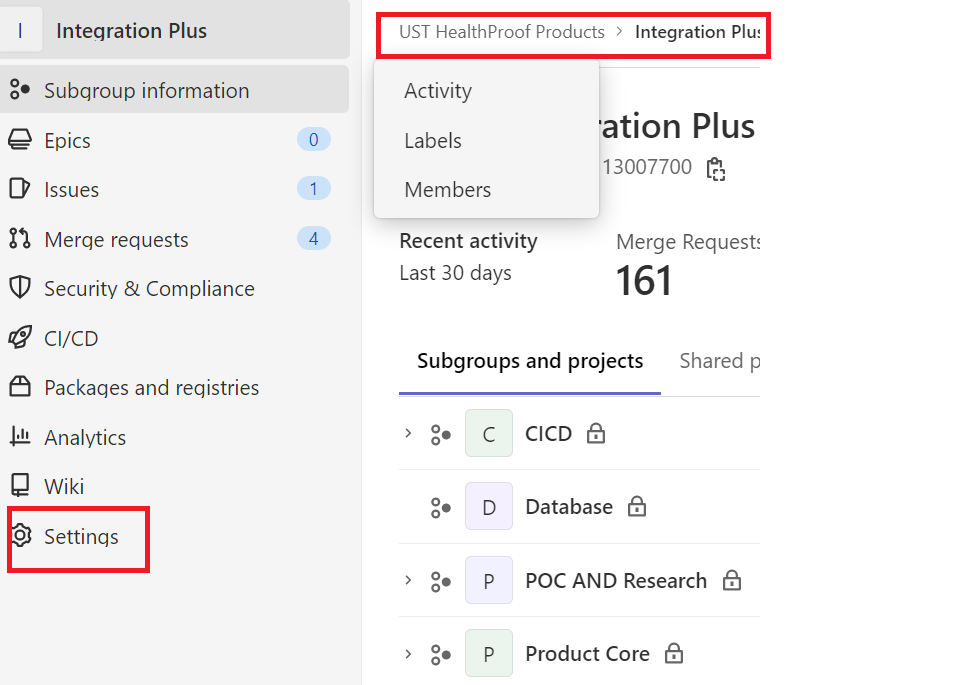


In In Deployable Repo where we need to create a Images.txt file where need to define all the the product images which has to pull from Jfrog and push to ECR.

# How to Configure Group Variables in GitLab

Identify and set the Group variables and each group will have its own Unique ID. Here group is UST Health Proof Products 🡪 subgroup is Integration Plus:-

* Click on **Settings** in GitLab tool



* Select **CI/CD** section 🡪 variables and configure



# Create/Build Continuous Integration (CI) Pipeline Script

The CI pipeline script is developed using YAML language and find below standard script *“config.yml”* can be created using any text editor (notepad/++ or MS Source visual studio code) for all stages along with pre-defined group variables.

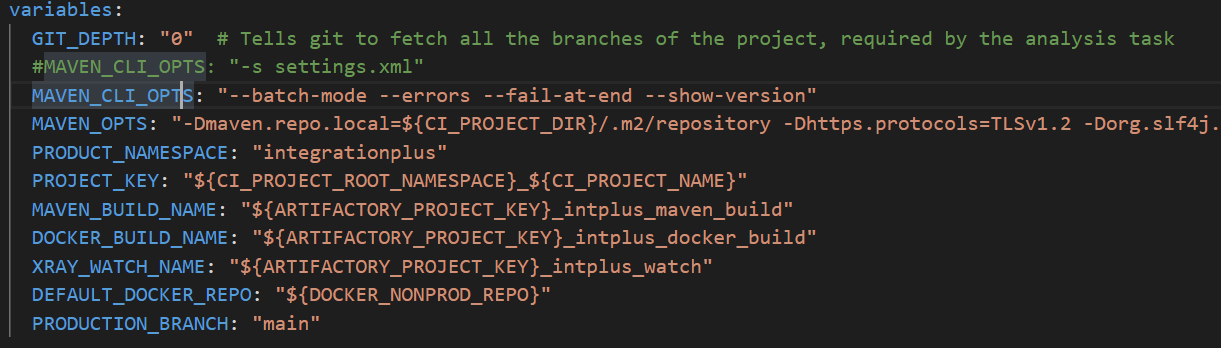
The sample script file (complete code in Appendix) consists of different sections are listed in below table: -

|  |  |
| --- | --- |
| **Section** | **Notes** |
| Variables | Variables Holds the Value |
| Stages | Stages are pre-defined Pipeline Stages |
| QualityGatesCheck | Sonar Checks |
| Versioning | Maven IMAGE Tagging |
| MavenBuild | Build the Source Code |
| DockerBuild | Build the image from DockerFile |
| Export Images | Push images to AWS ECR |

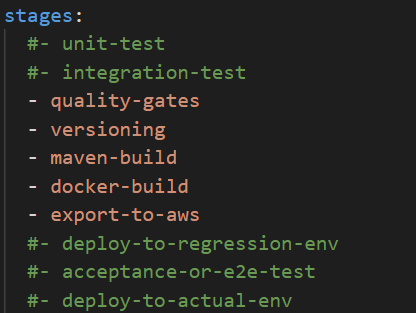
1. Start creating the script file using any text editor or available GUI interface and name it is as “config.yml”
2. Create yml file with below sections or stages as per the project specification and requirement, you can include all stages in a single file or create individual files for each stage



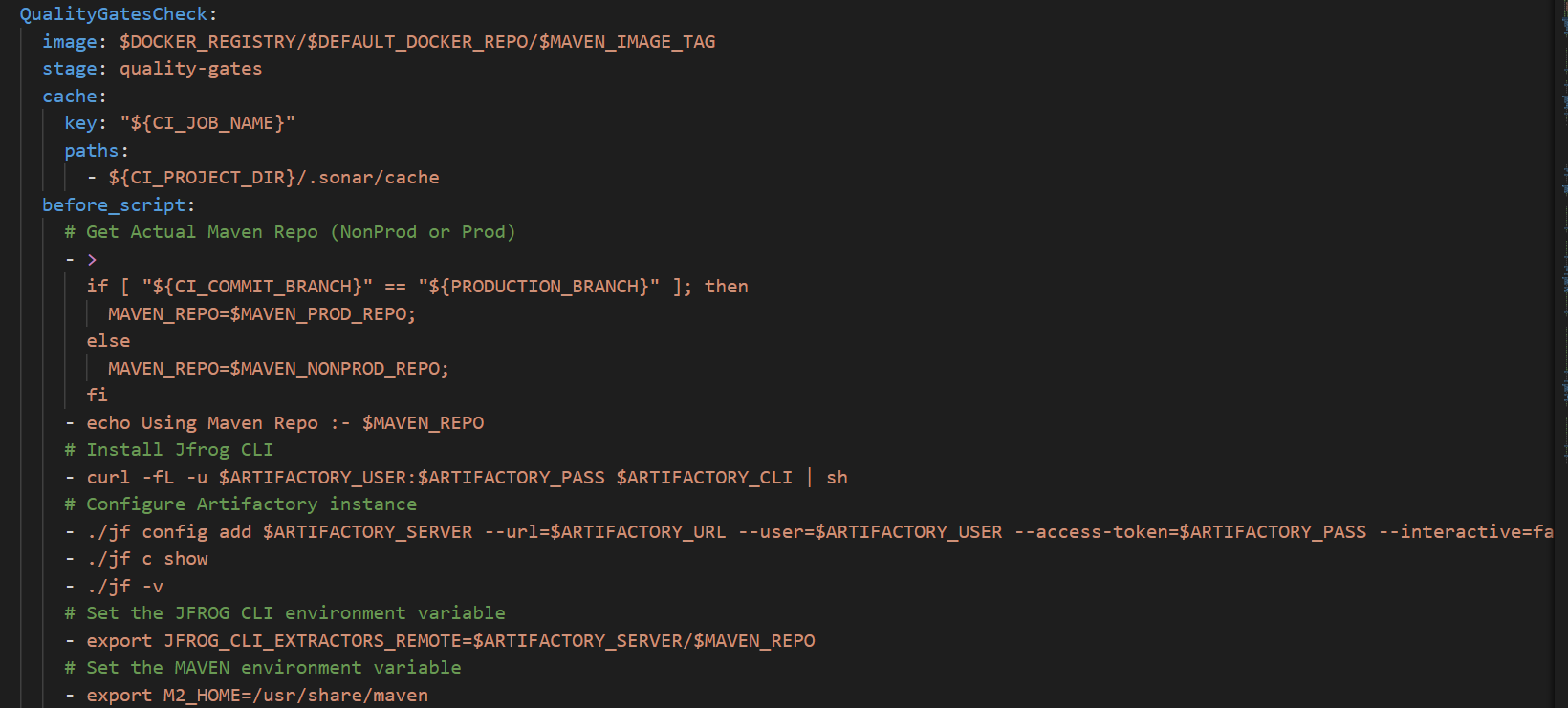
1. Start with variables declaration. The recommended way and best practices in configuring variable is Group variables which has to configure in Group level, so that it can inherit to child repositories as mentioned in the Environment Setup section(Pls refer 5.2)

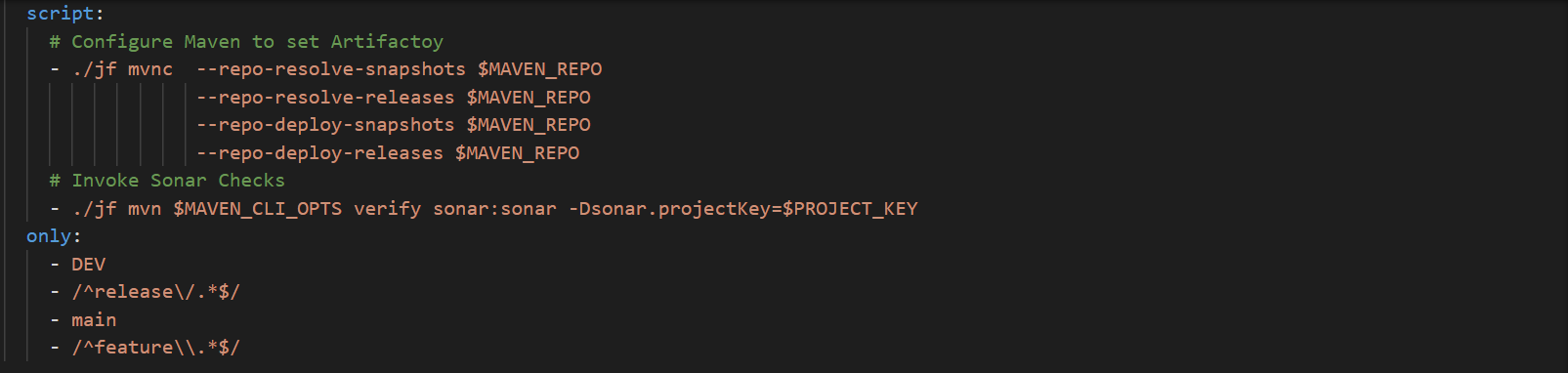


1. Create different Standard Stages as shown in below which starts from Sonar quality checks, versioning , Maven build , docker build and export-to-aws

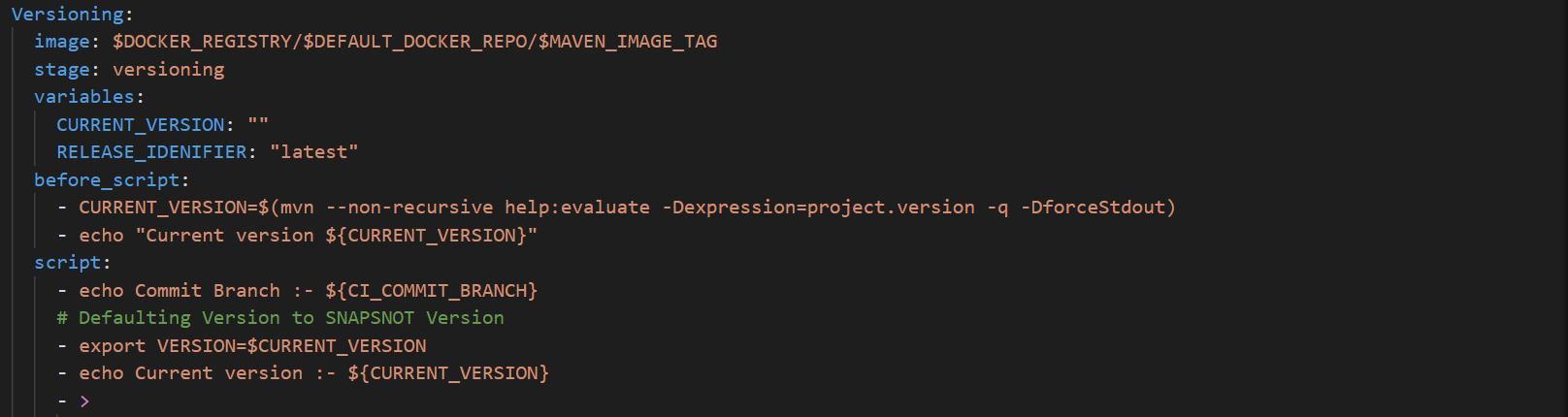


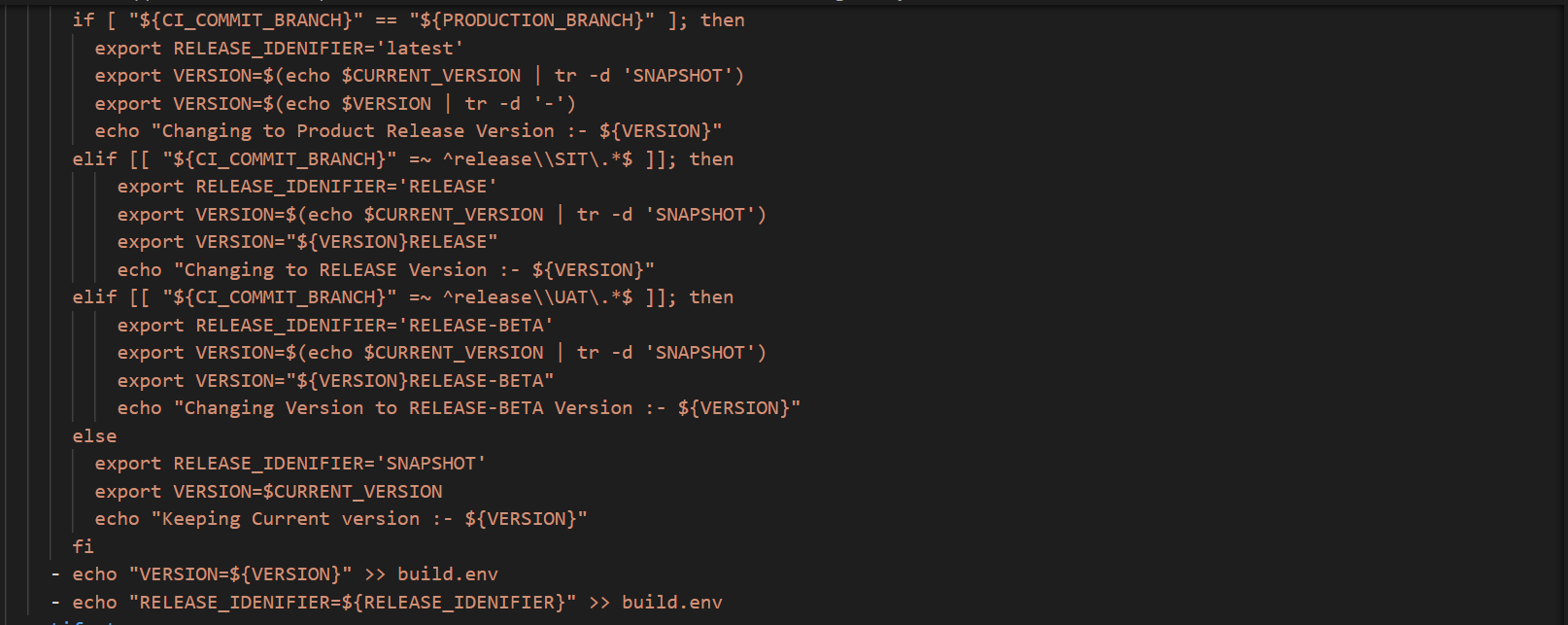
1. Define QualityGatesCheck stage where it runs sonar code vulnerabilities checks.





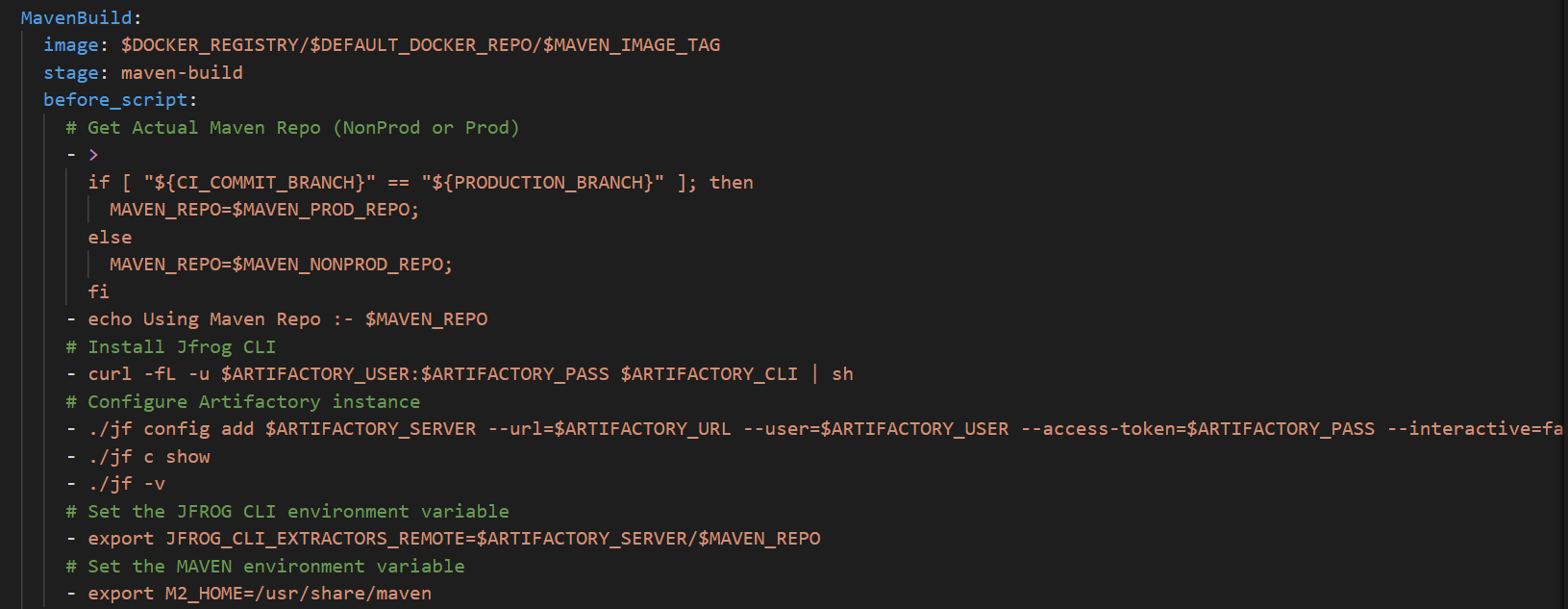
1. Define versioning standards as defined in the Naming Convention. Discuss this with Application Developers/Devops/Architect. Naming convention is standard across the environments from Dev, SIT, UAT & PROD.

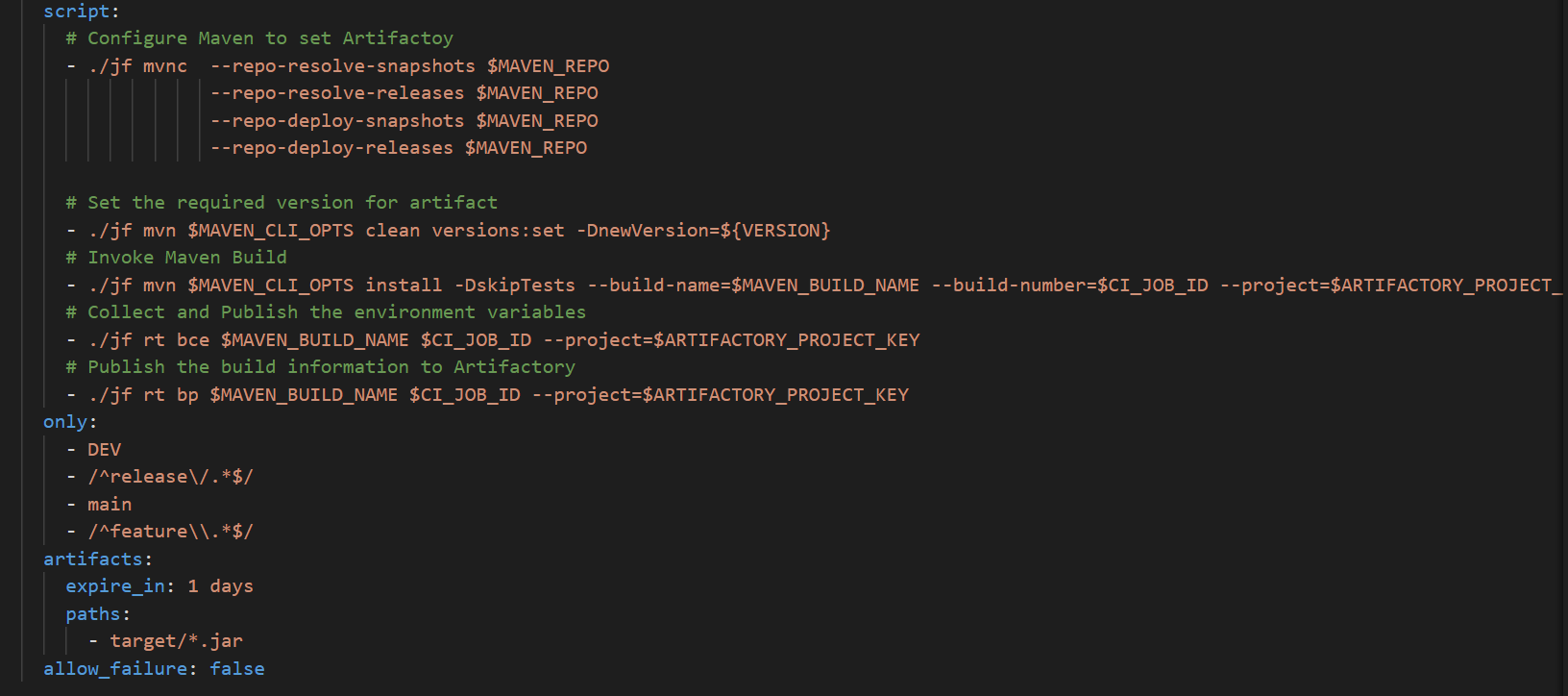




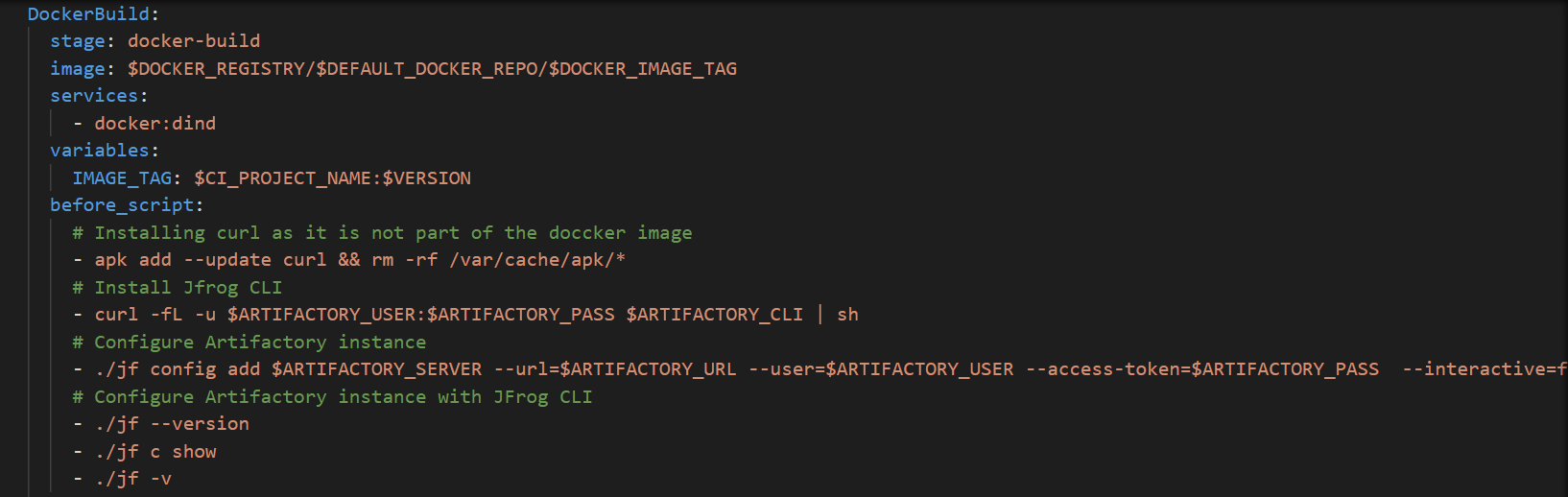


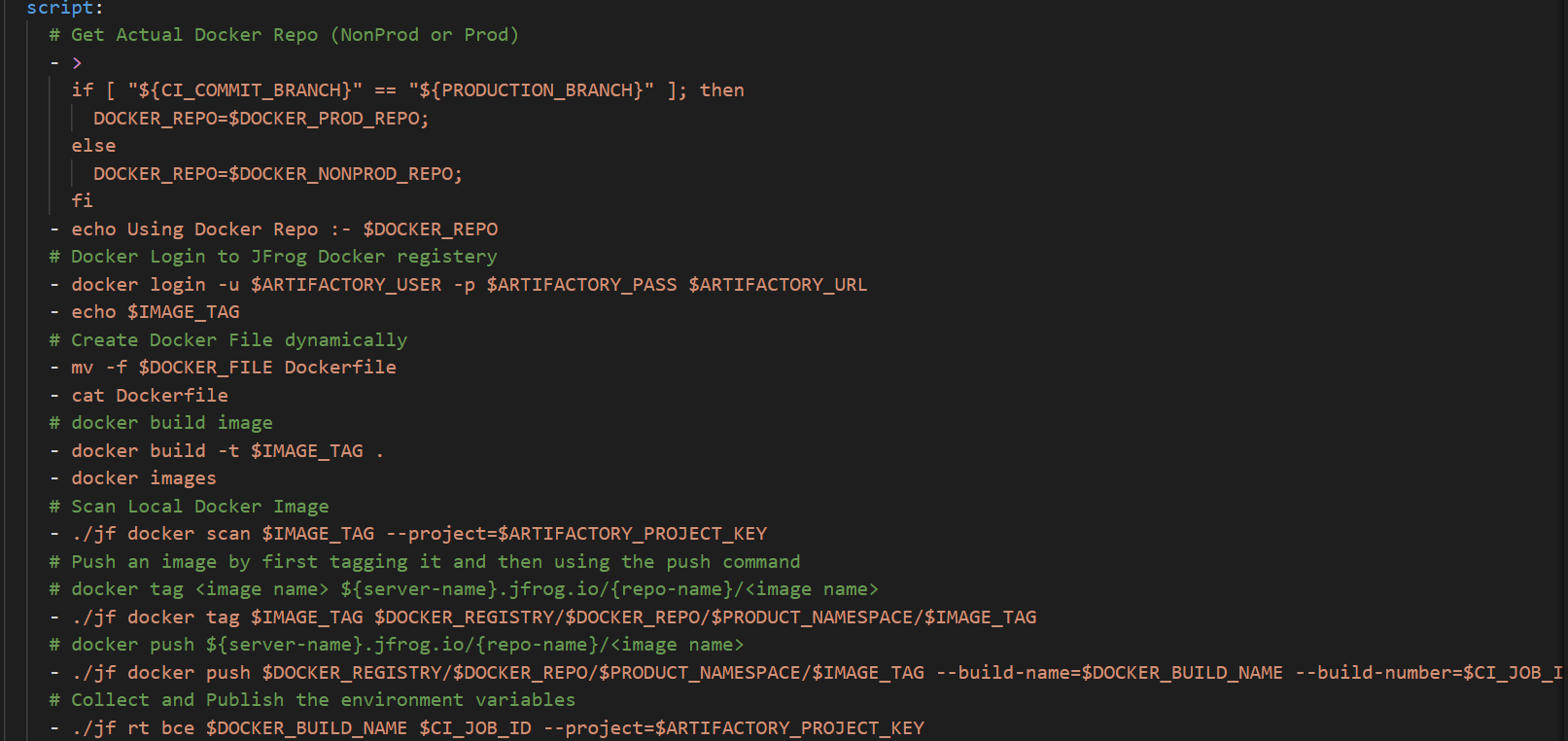
1. Define (Mavenbuild) stage where it build source code using maven tool and produce the artifacts usinb jfrog client.

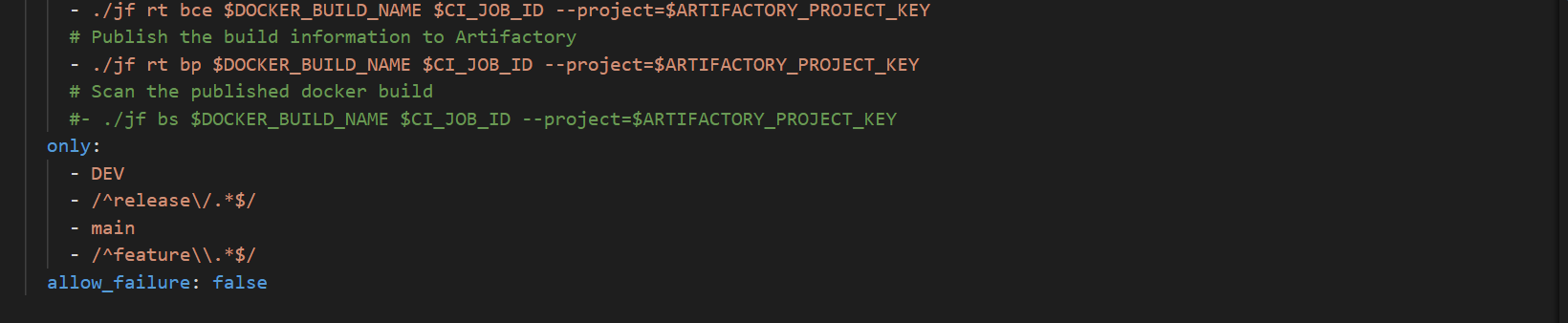




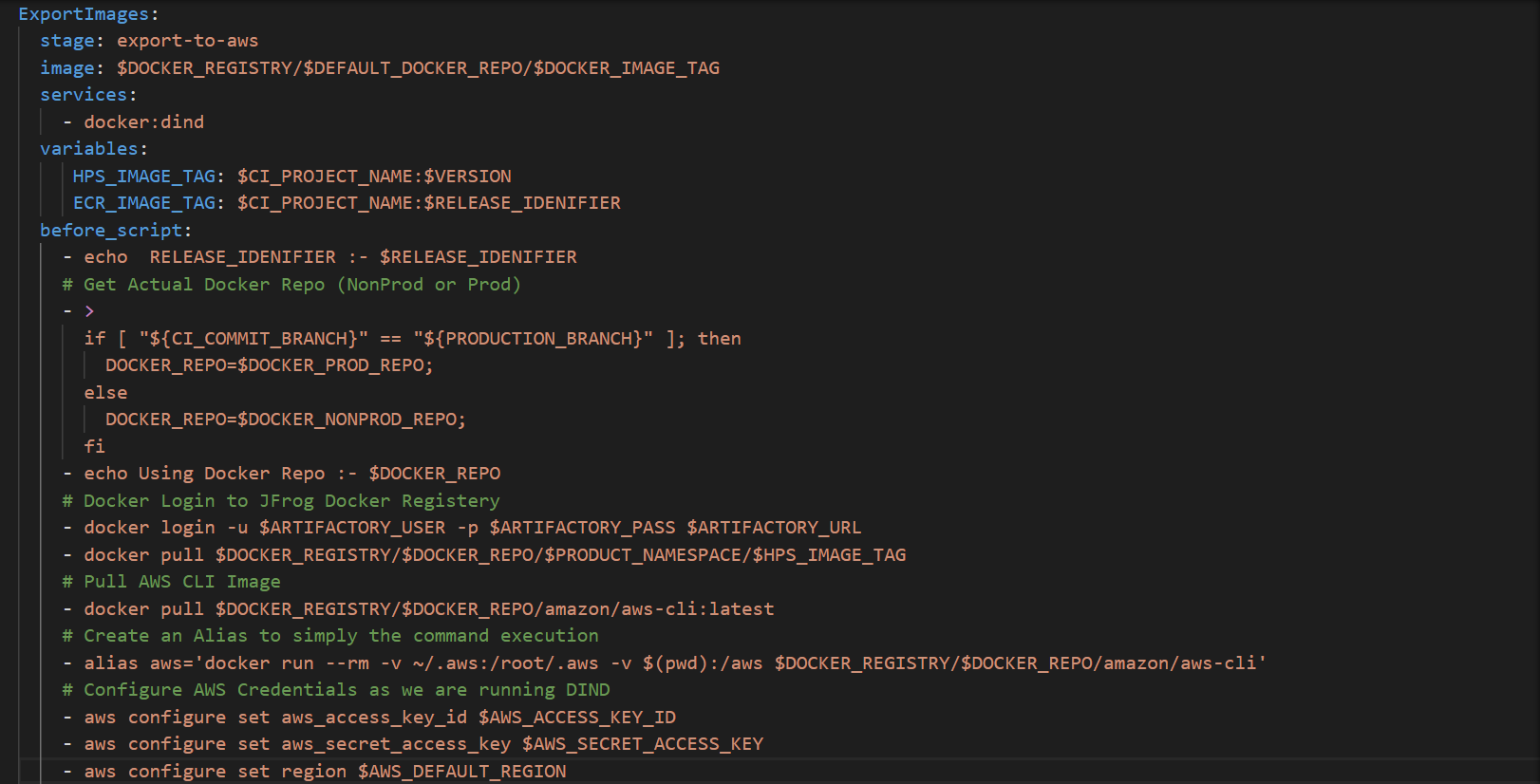
1. Define build (Dockerbuild) stage where it creates docker image of the application which is defined in the DockerFile and it will tag the image and scan the docker images.

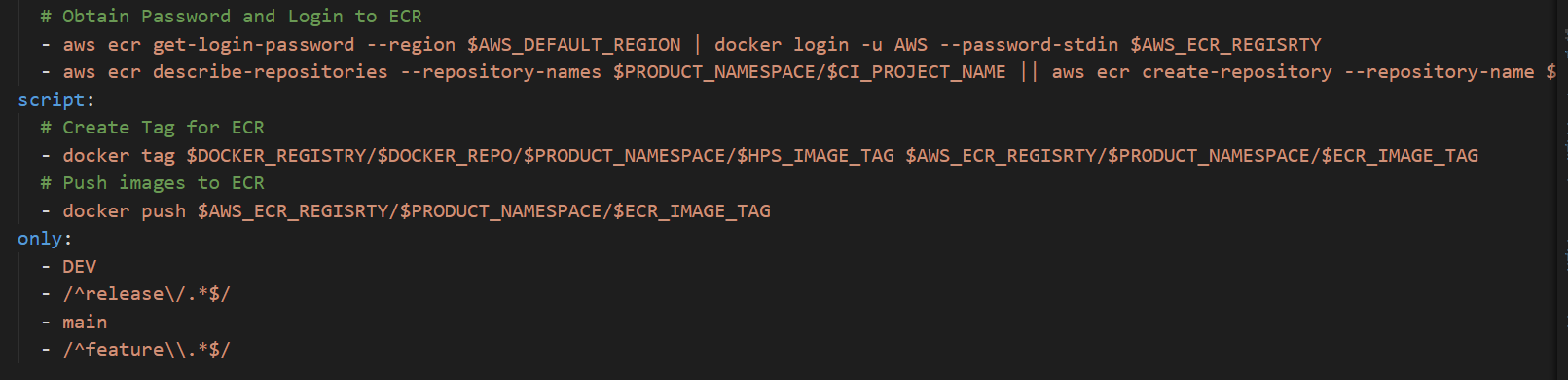






1. Define ExportImages stage where it will pull the images from jfrog and push to AWS ECR registry.



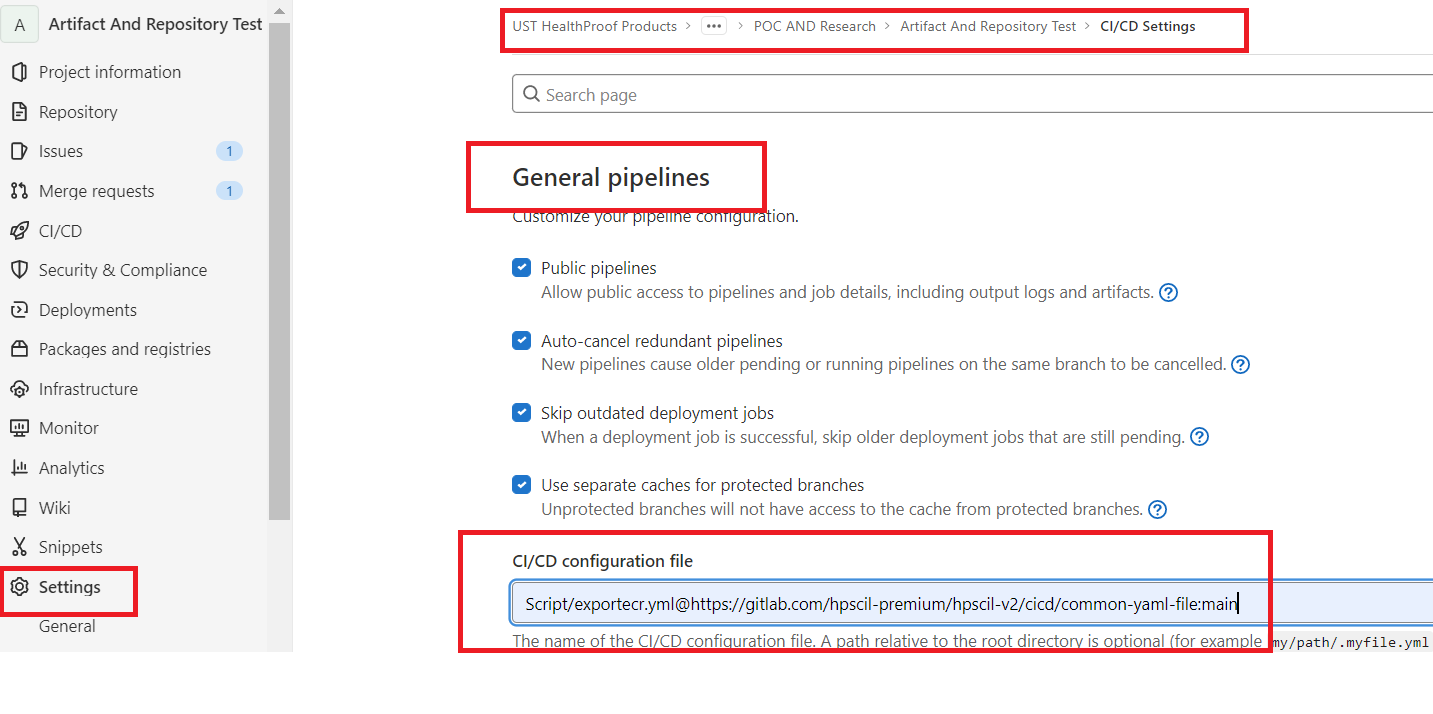


*Note: - Sample configuration file in appendix section (1.2 Continuous Integration (CI) Pipeline repository)*

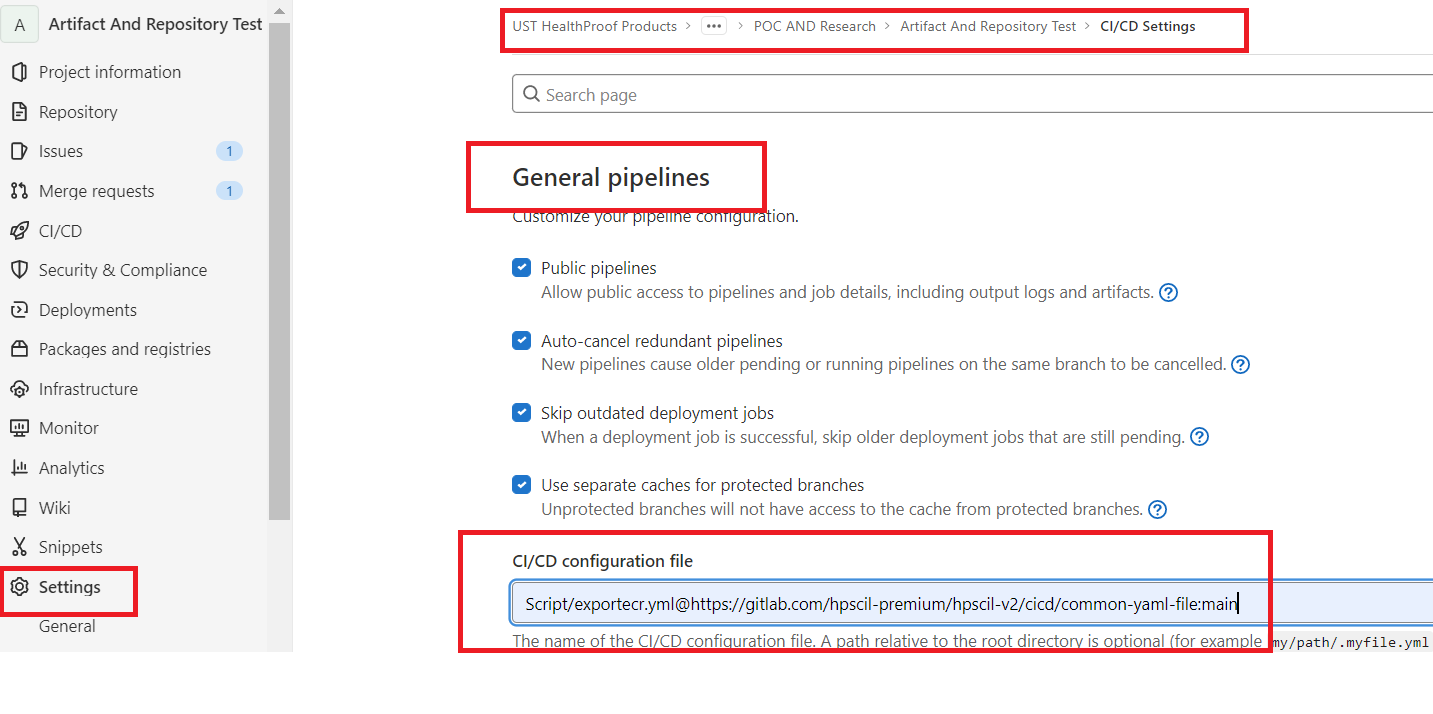
# How to Configure Script in each Repository

Find below the steps to configure script files in each repository -

1. Open the respective repository where the script file need to be configured



1. In GitLab tool click on settings 🡪 general pipelines 🡪 CICD configuration files



Script File Path - Script/Config-ci.yml

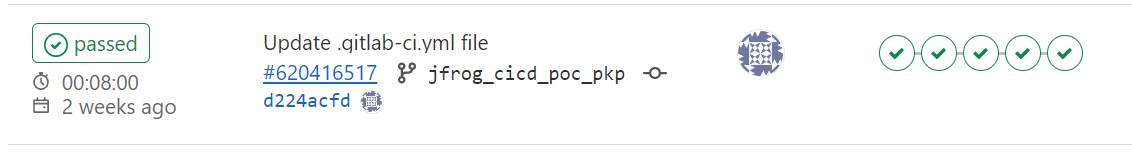
Repo Path - @hpscil-premium/hpscil-v2/cicd/common-yaml-file –

Branch name - Main

# How to Run CI pipeline Scripts

Find below the steps to Run pipeline script files -

1. To Run the pipeline, go to project repository of CICD 🡪 click on pipelines and also please note as part of normal merge request , pipeline will get triggered automatically.
2. To verify pipeline execution status, all stages in pipeline should be in success status



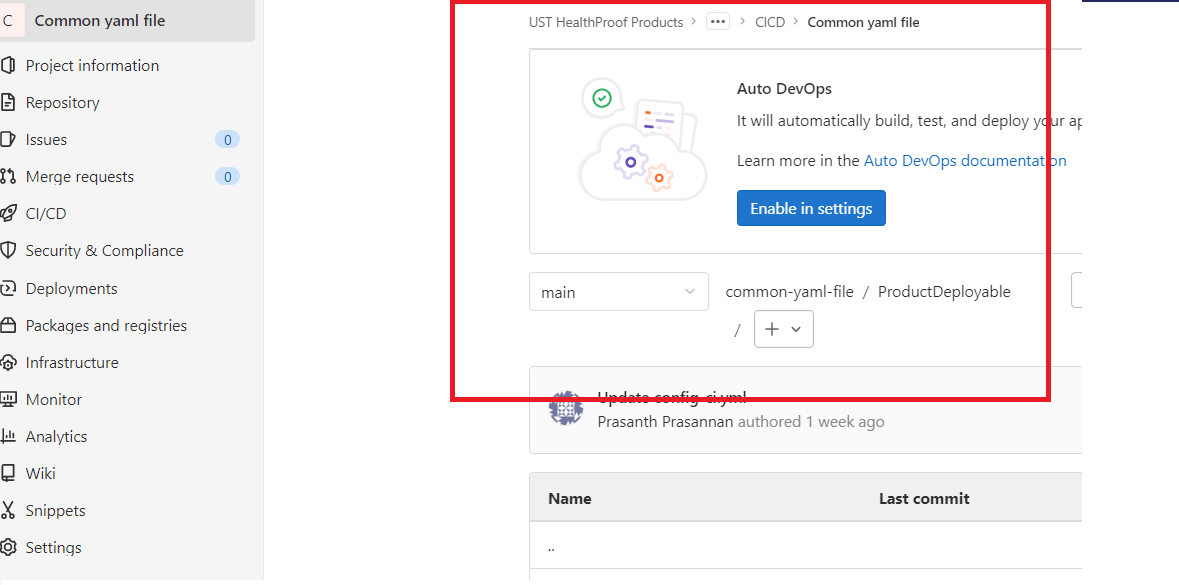
1. If any error what to do

Go to Gitlab CICD🡪 Pipelines-🡪Click Pipeline failed id and click respective stages which is failed and analyse the logs.



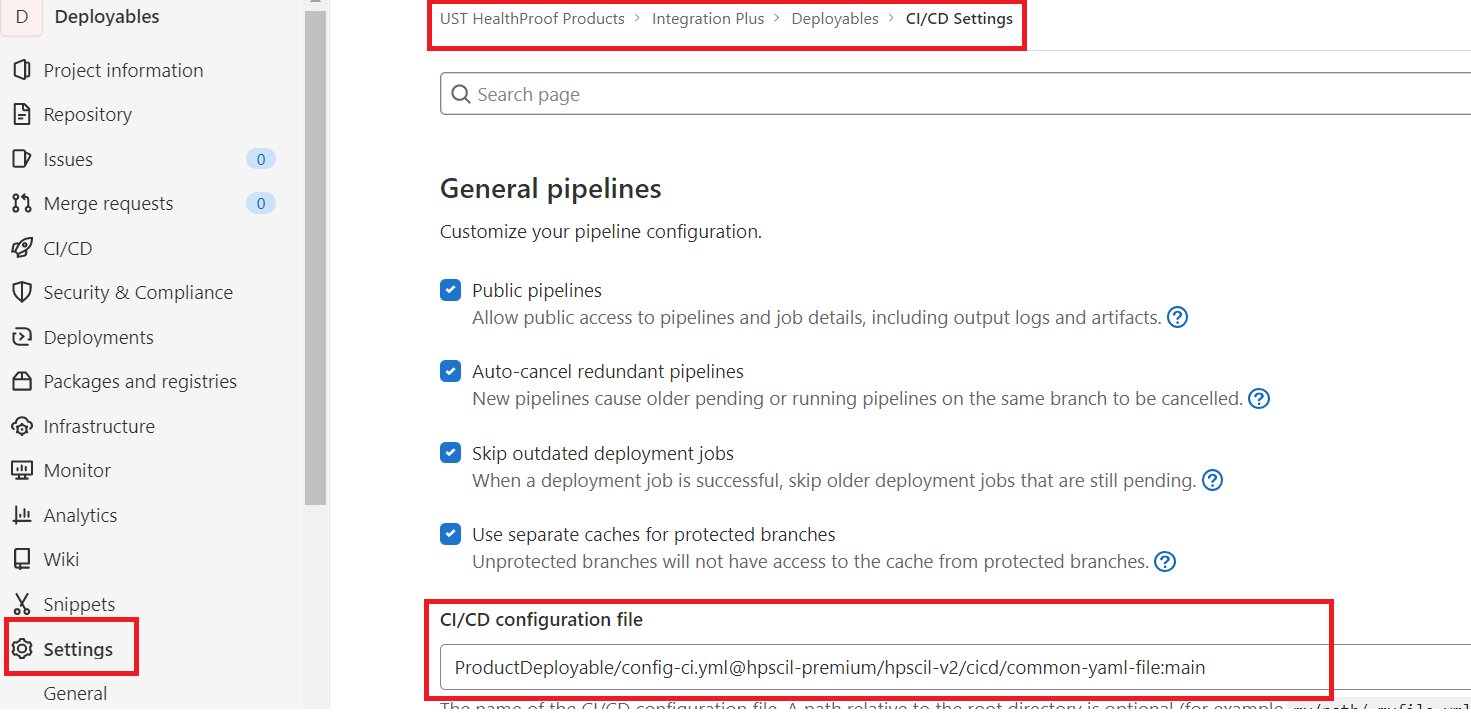
# Scripts to Deploy Product Images

Product Deployable Script Config.yml in CICD-> Common Yaml File 🡪 ProductDeployable



# How to Run CI Product Deployable Scripts

Please note as part of the Normal Merge Request, Pipeline will get triggered automatically



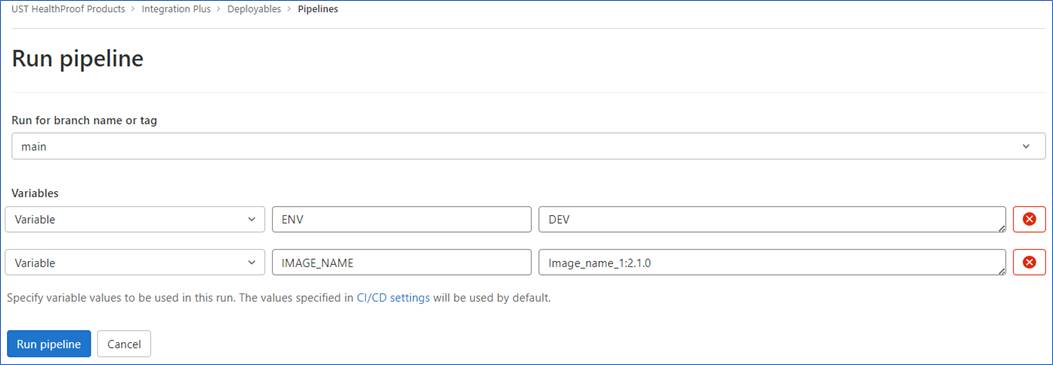
Deployables Repo🡪 settings 🡪 CICD/Configuration file and put below location

Script location : [ProductDeployable/config-ci.yml@hpscil-premium/hpscil-v2/cicd/common-yaml-file:main](mailto:ProductDeployable/config-ci.yml@hpscil-premium/hpscil-v2/cicd/common-yaml-file:main)

* To deploy the Product Images, manually run the Pipeline for Deployables Project. Specify the environment to which the images should be deployed as an input variable. This is to identity the AWS subscription details.
* Variable Key      : ENV
* Variable Value   : DEV, SIT, UAT, PROD etc. (logic inside the CI script has to be enhanced based on this value, if this is NOT defined already)
* The CI script will read the Images.txt file and will pull the listed images from JFrog and push to AWS ECR one by one. Note that this will deploy all the images mentioned in the text file.
* Ideally the images shared from the Product will NOT have any RELEASE IDENTIFIERS and these images would get tagged as “latest” while pushing to ECR. For those images having RELEASE IDENTIFIERS (currently few Product shares SNAPSHOT Images), it would get tagged based on the respective RELEASE IDENTIFIERS.
* e.g.

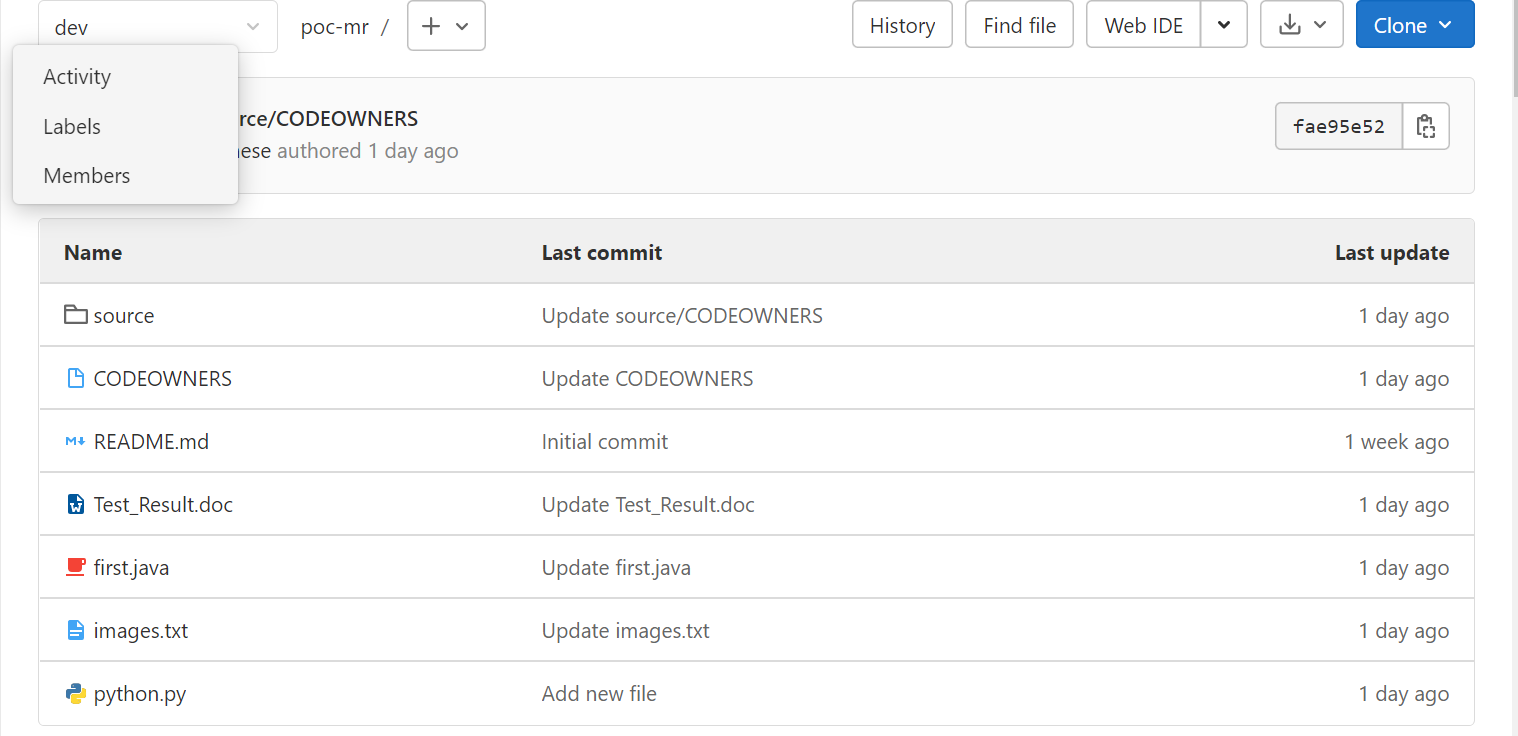
|  |  |  |
| --- | --- | --- |
| **Artifact Name** | **Artifact in JFrog** | **Image in ECR** |
| Image\_name\_1:2.1.0 | Image\_name\_1:2.1.0 | Image\_name\_1:latest |
| Image\_name\_2:2.0.0-SNAPSHOT | Image\_name\_2:2.0.0-SNAPSHOT | Image\_name\_2:SNAPSHOT |

* The same CI script can be used to deploy single images as well. This can be achieved by either updating the Images.txt file to the single image name, **or specify the image name as an input variable**. This will deploy the specified image alone.

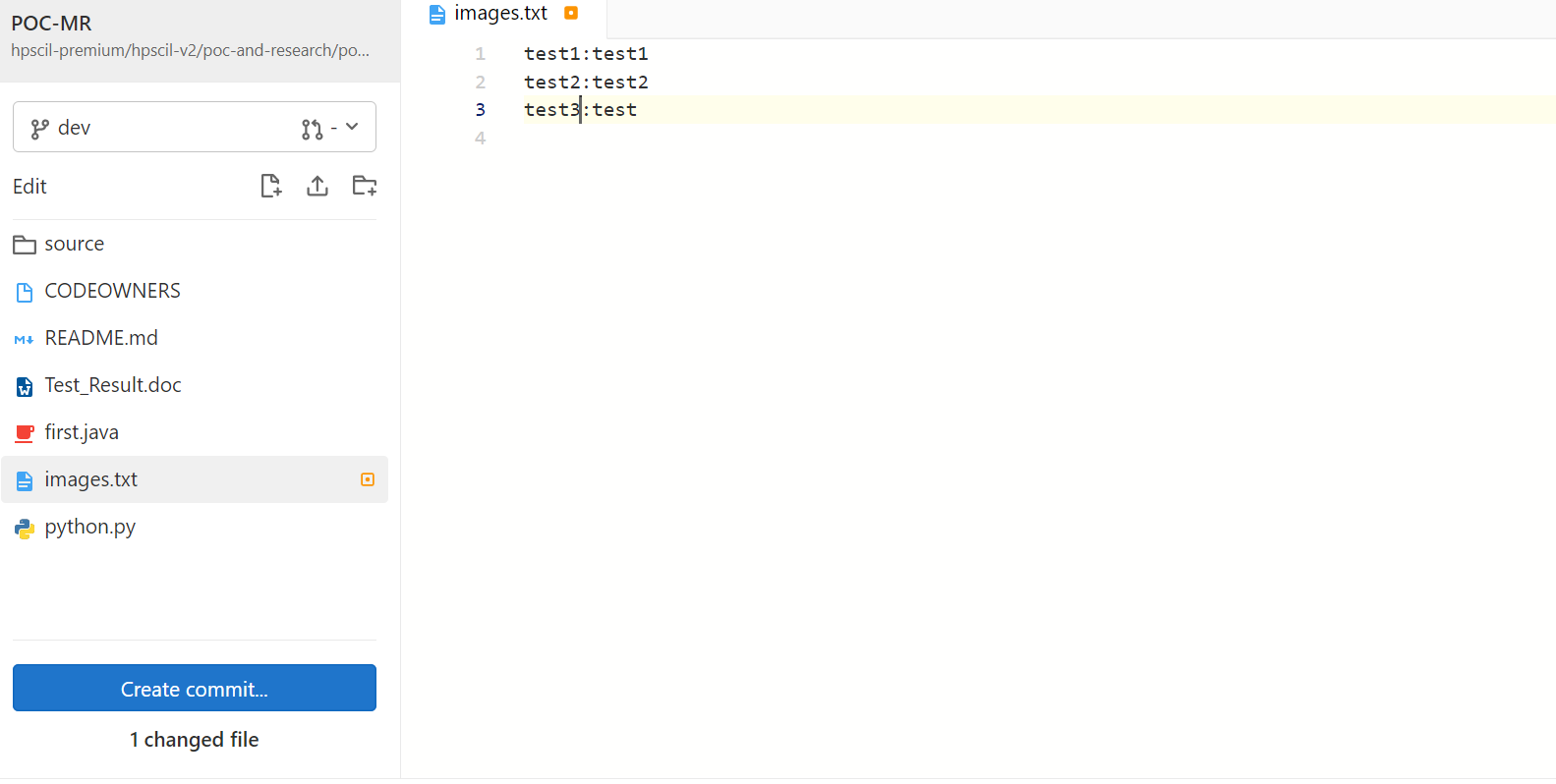


# MR Approval process

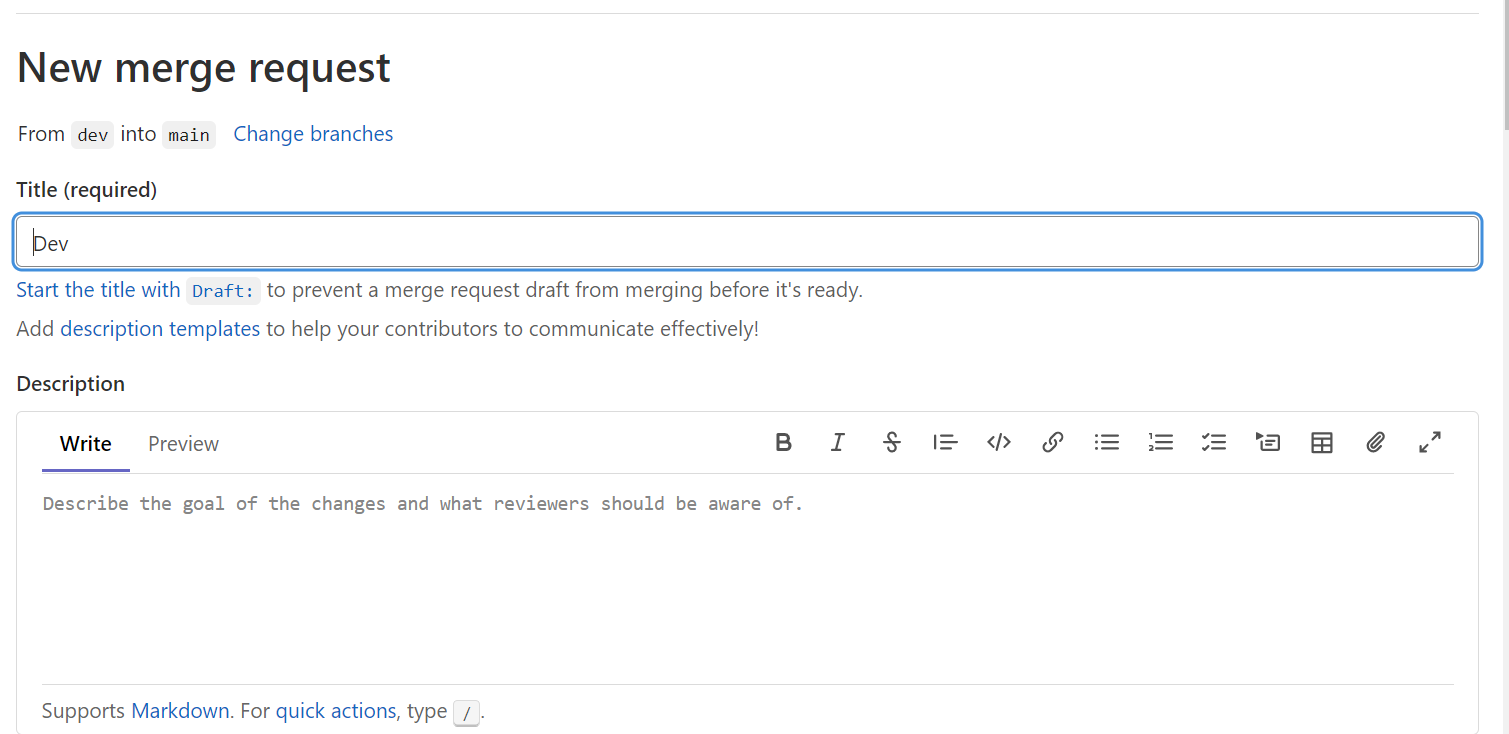
Added multiple files in dev branch. If any file changes then it will ask for approval as shown in below file, we have multiple types of files.



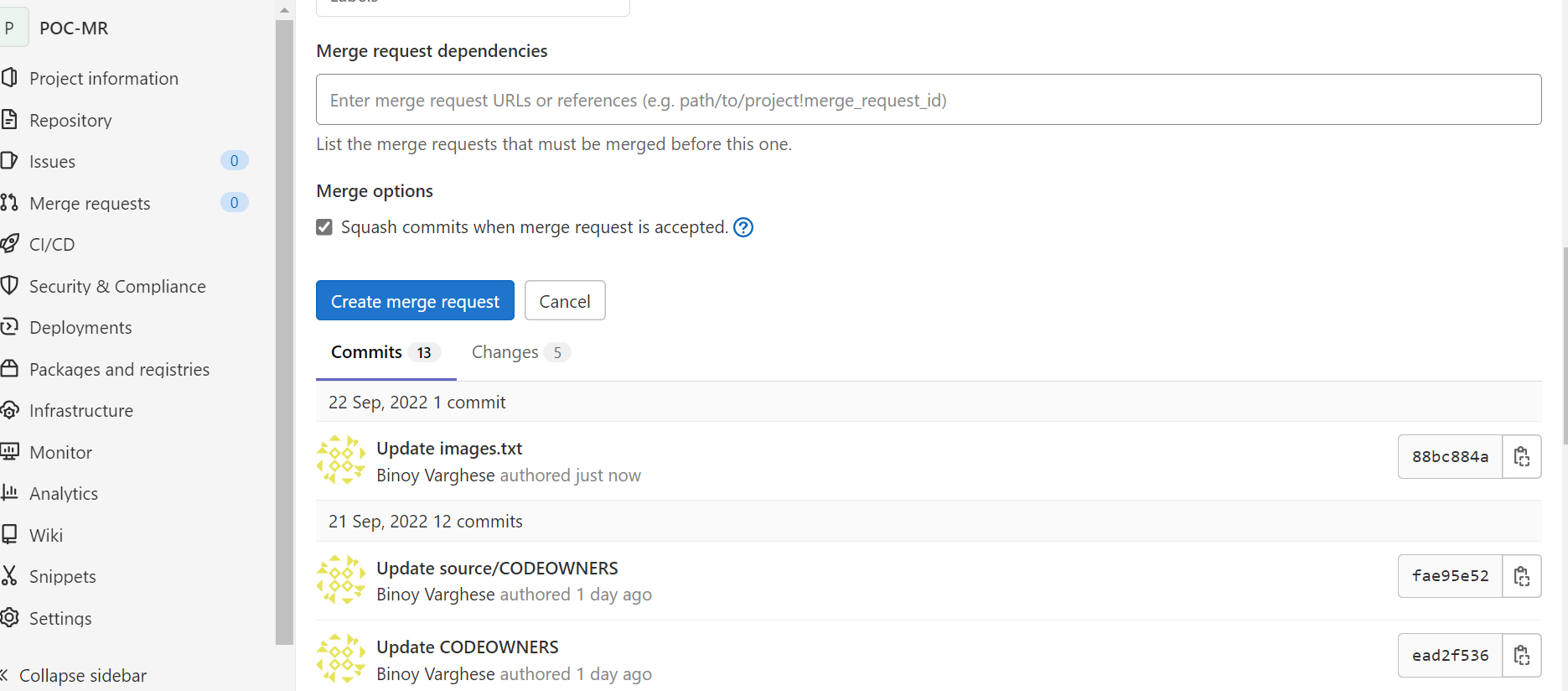
Made the changes in one of the file from the Dev branch as in screenshot



After the changes these changes needs to be pushed in main branch. So created new Merge Request.

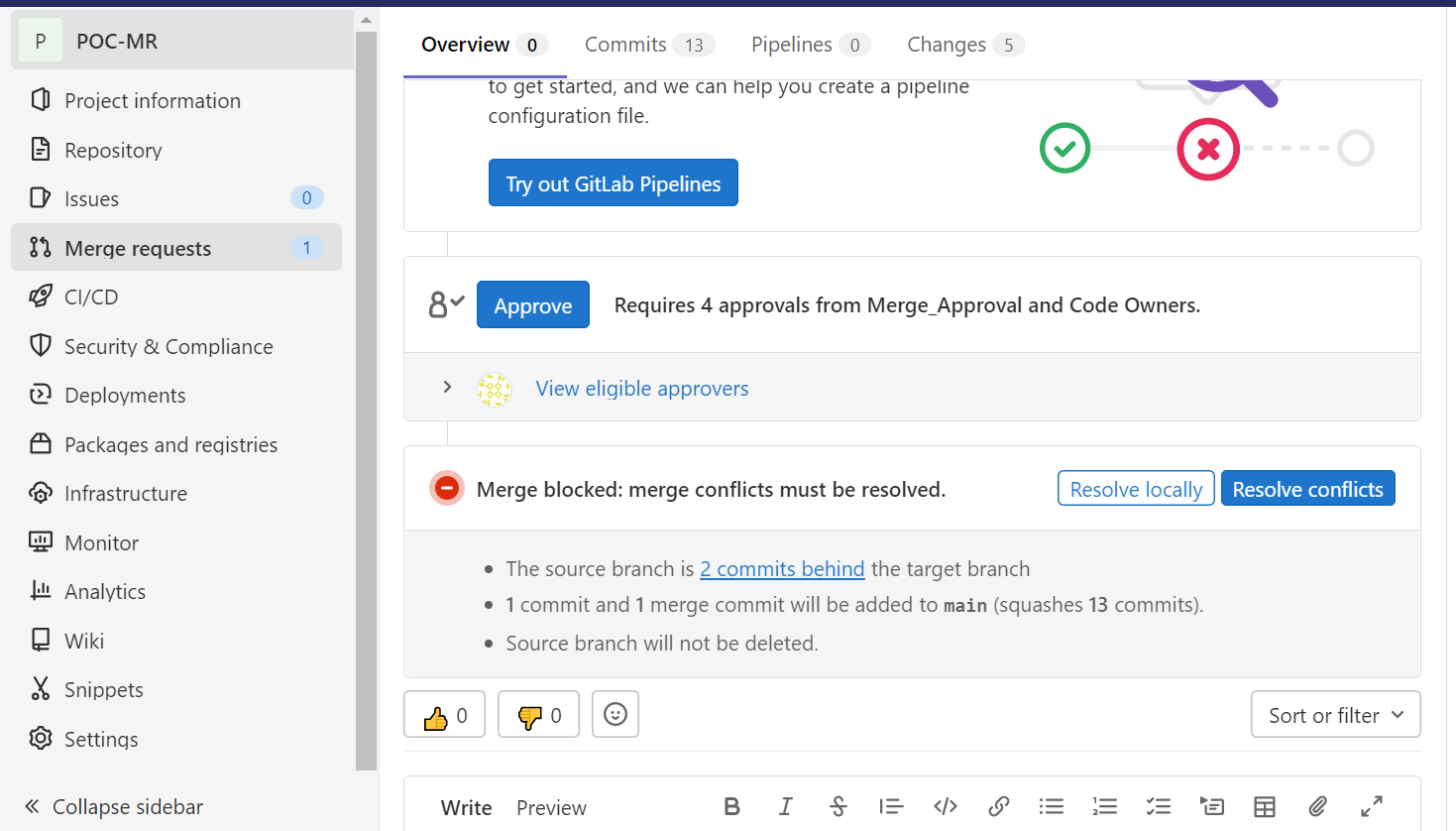


While committing the changes enabled squash commit as well. Squash commit will push the multiple changes in one go.

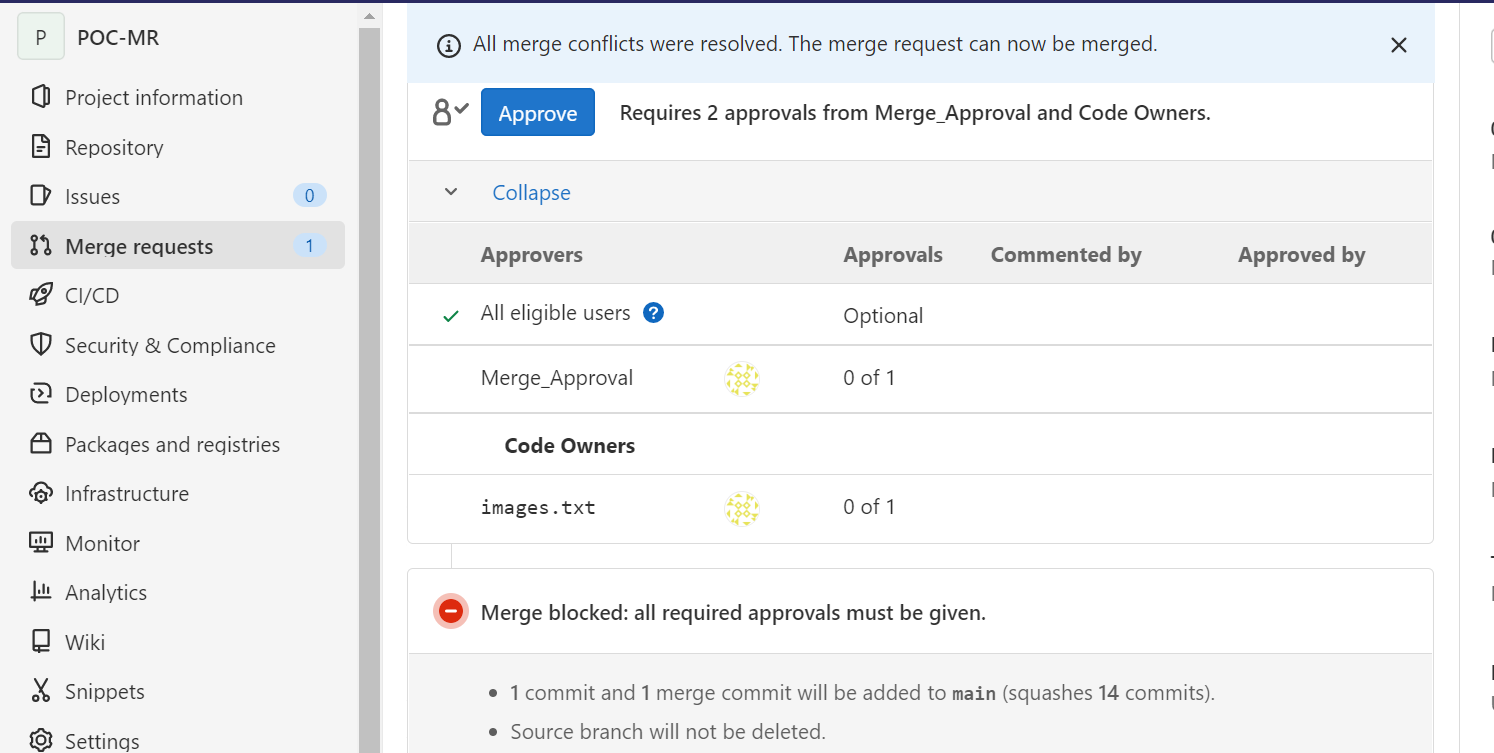


Applied approval rule to Merge request it is blocking us from merging the dev branch into main branch as it is waiting for Approval.

Direct commit is not happening as it is waiting for approval



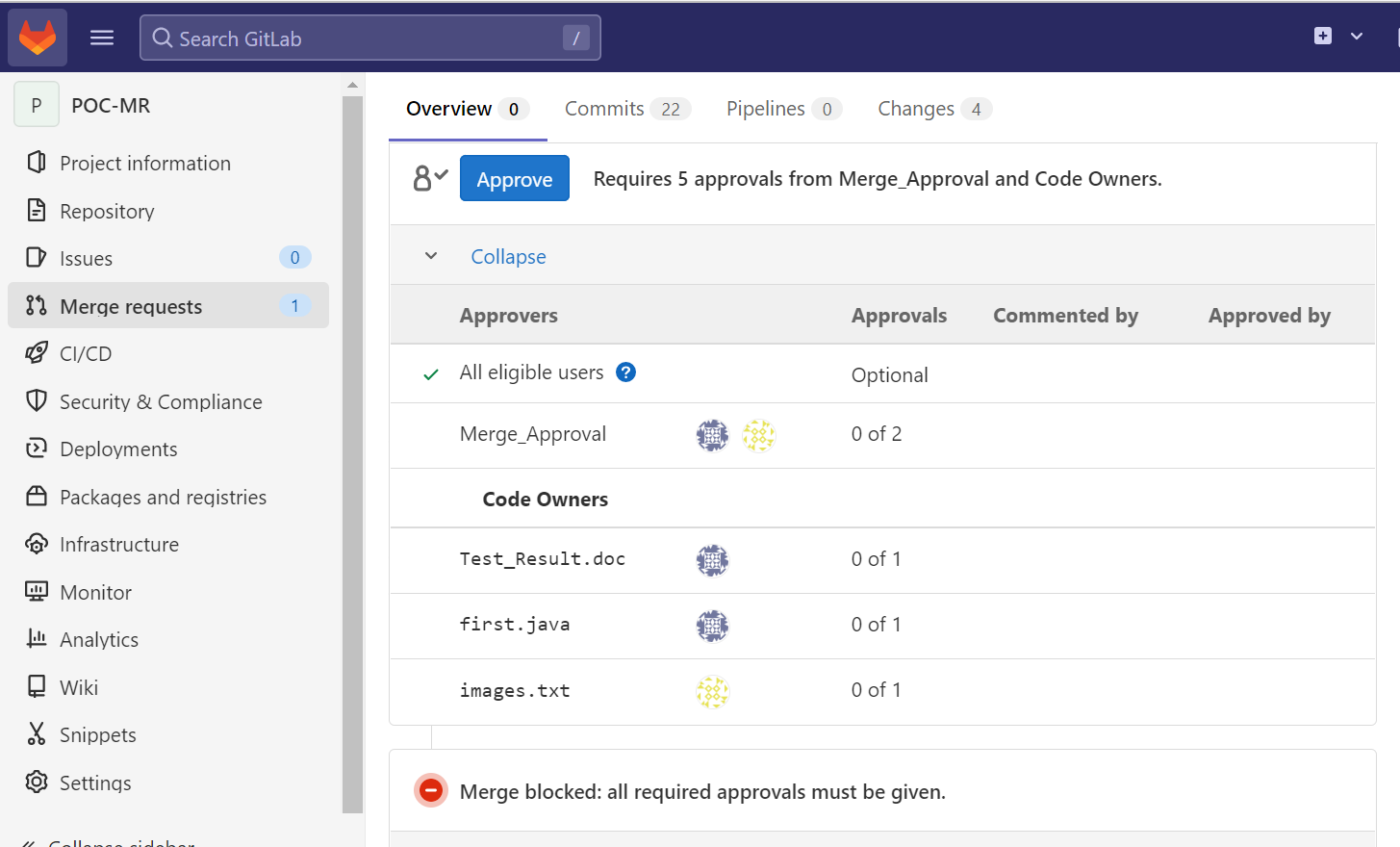
We can add multiple approvers in the approval rule. For the time being we have added one approver and It is showing number of approval required as 1.

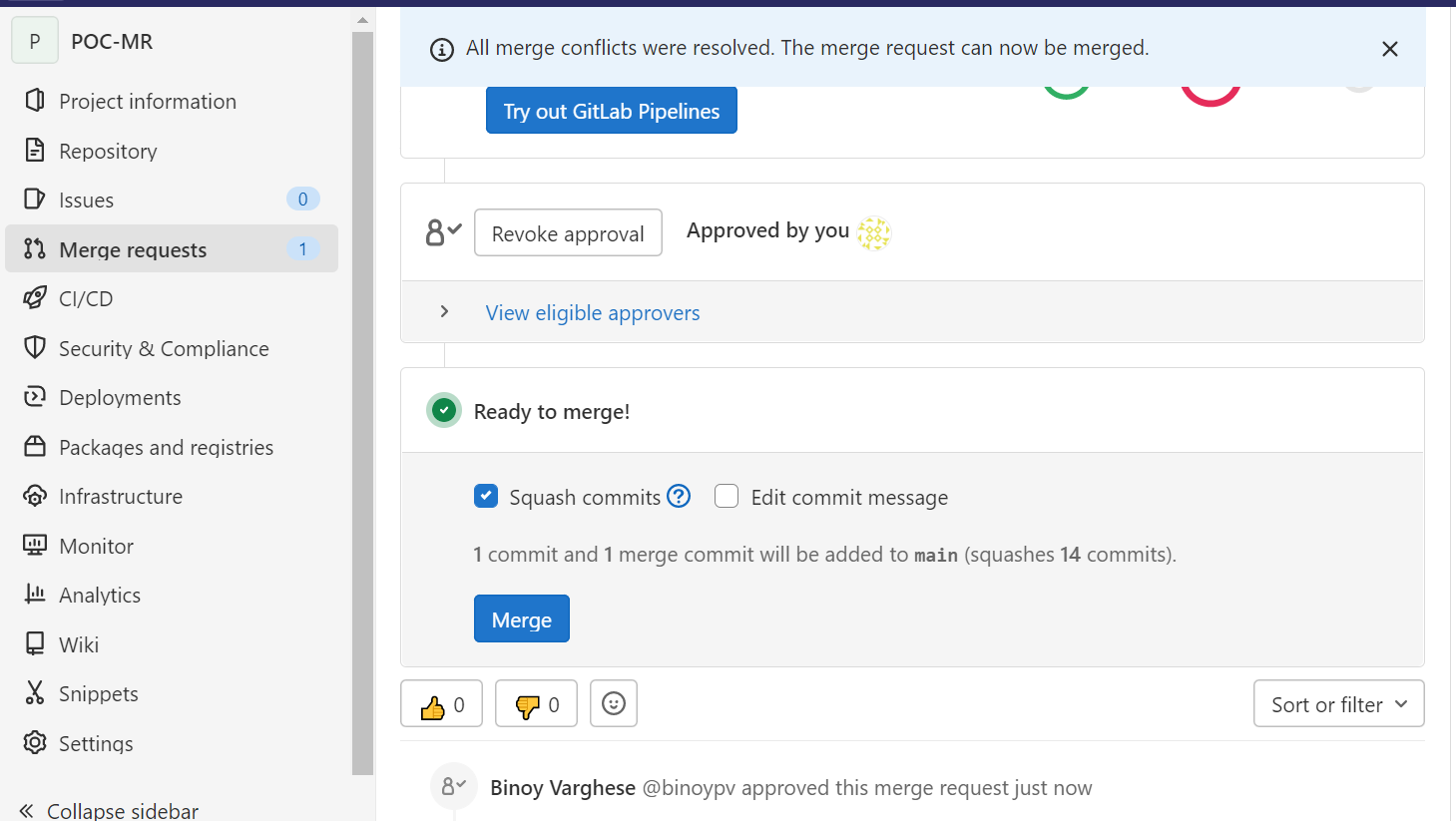


Once we approve this merge request dev branch successfully gets merged in main branch as shown in below screenshot

After approval it is ready for merge. Also note that squash commit is enables here by default

This shows approval is required from different approver for different files

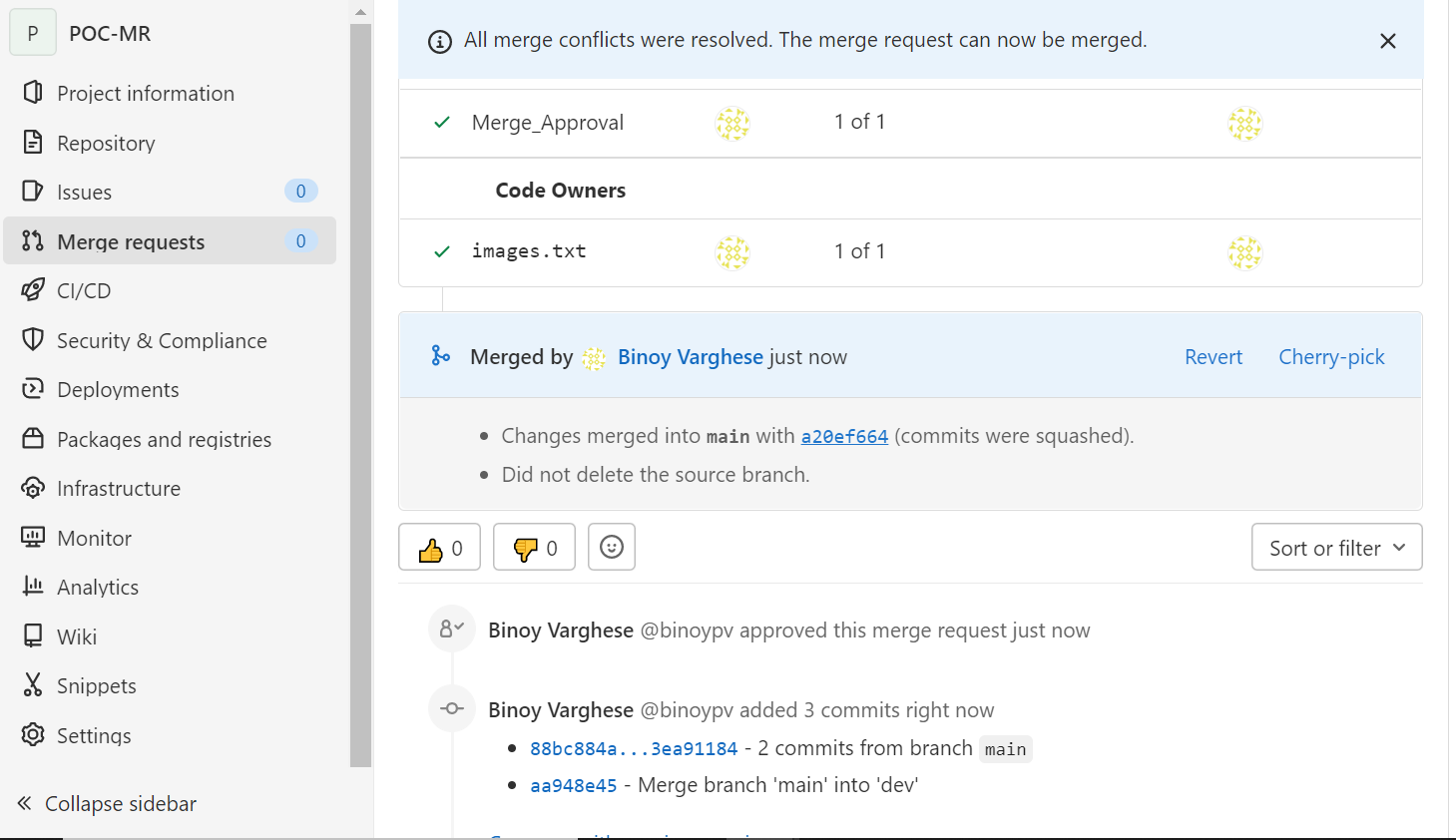




Here we have enabled Delete source Branch option by default which will not affect protected branches



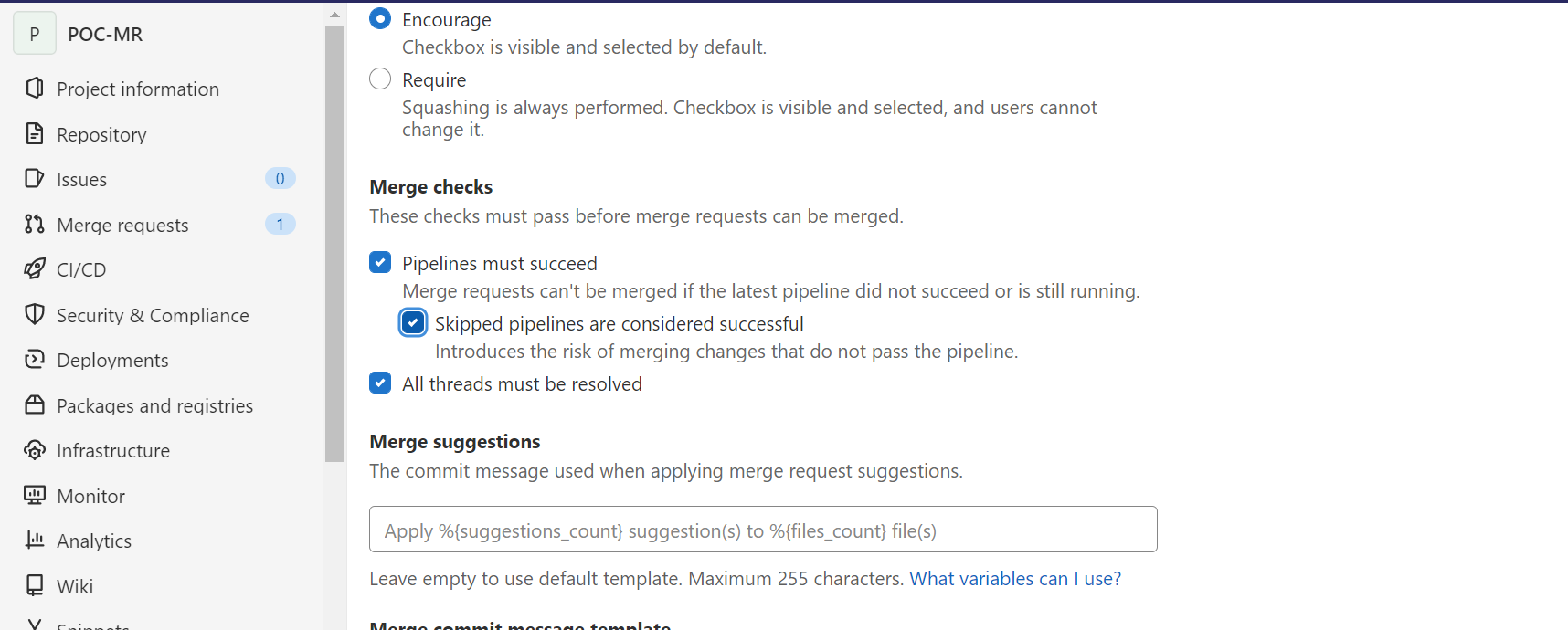
Dev code is merged to main branch. It also shows the file which has changes while merging the dev branch to main.



## Sonar check on Merge Request and fail the Merge if Sonar fails

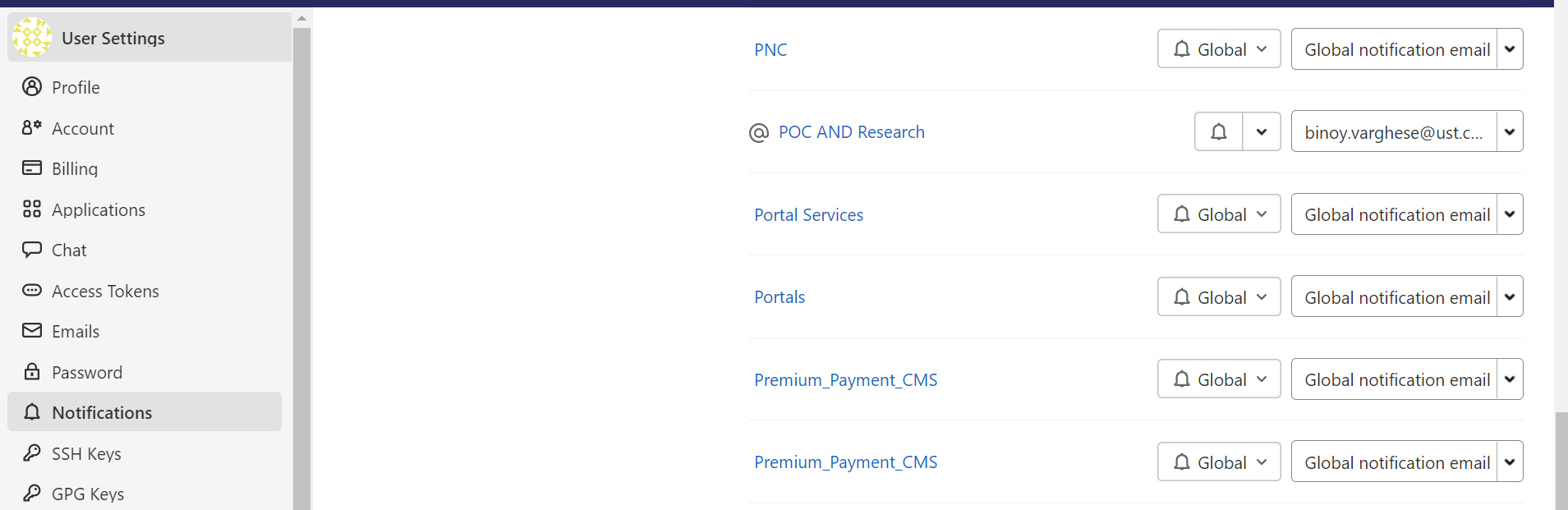
We can achieve if sonar check on Merge request and fail the Merge if the pipeline fails by enabling Merge checks under Merge requests (project settings)





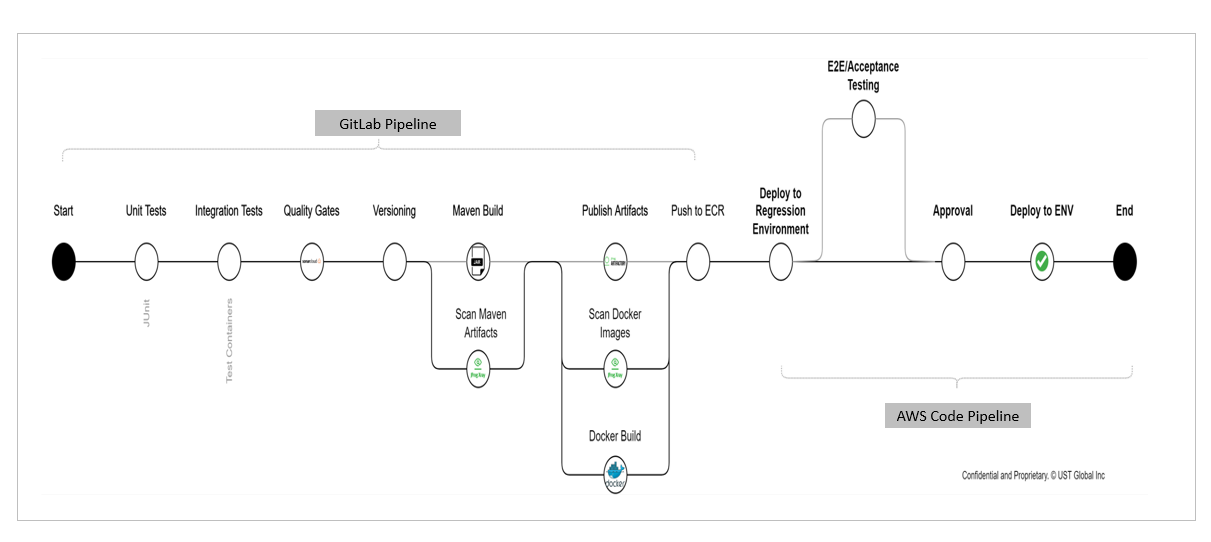
## EMAIL Notification

To get the notification of the activities happening in dev/main branch we can integrate this with Microsoft Teams or Outlook. Below series shows Gitlab integration with Microsoft Teams.



# Pipeline Stages – End State

The below diagram provides different stages that we are envisioning to be involved in CI pipeline.



# APPENDIX

* 1. **High-level overview of tools/products used in building CICD pipeline**

**GitLab -** SaaS based management platform for Git repositories used as the Continuous Integration-Continuous Deployment (CICD) tool which can integrate with any 3rd party tools like Maven, JFrog, Sonar, Docker and Cloud platforms. For more details visit official site – <https://about.gitlab.com>

**Sonar Cloud –** Cloud based code analysis service designed to detect code quality issues targeting 25+ different programming languages and ensuring maintainability, reliability, and security of the code. and Cloud platforms. For more details visit official site <https://sonarcloud.io>

Quality Gates can be implemented for the below: -

* Code coverage
* For new code (configurable)
* Branches
* Usage / Non usage of coding practices
* Common security issues with code
* Duplicate code
* Cyclometric complexity
* Code Quality, Code Smells etc.
* Security, Maintainability, Reliability ratings
* For Java 600+ rules available in sonar (configurable)

**Maven –** It’s a build automation tool used primarily for java-based projects and be used to build and manage projects written in C#, Ruby, Scala and other languages and the product has been developed by Apache Software Foundation. For more details visit official site <https://maven.com>

**Docker –** It’s an open-source containerization platform enable developers to package applications into images (executable components combining application source code, dependencies, and operating system libraries) and run images in an isolated environment called containers. The software that hosts the containers is call Docker Engine. For more details visit official site <https://www.docker.com>

**JFrog –** It is universal DevOps solution providing end-to-end automation and management of binaries and artifacts through the application delivery process that improves productivity across development ecosystem. It has integration with 25+ software build packages, all major CICD platforms, Artifactory is Kubernetes ready supporting containers, dockers and helm charts. For more details visit official site <https://www.jfrog.com> & <https://www.jfrog.com/confluence/display/JFROG/JFrog+Artifactory>

The following are the main features and functionality –

* Hybrid and Multi-cloud environments – Host Artifactory on own infrastructure, in the cloud or use the SaaS solution providing maximum flexibility
* Universal Binary Repository Manager – universal solution in supporting all major package formats including Alpine, Maven, Gradle, Docker, Cargo, Conda, Conan, Debian, Go, Helm, Vagrant, YUM, P2, Ivy, NuGet, PHP, NPM, RubyGems, PyPI, Bower, CocoaPods, GitLFS, Opkg, SBT, Swift, Terraform and more
* Extensive Metadata – provides complete metadata for all major package formats for both artifacts and folders; allows custom data format from users in searchable format
* Artifactory as Your Kubernetes Registry – allows to deploy containerized microservices to the Kubernetes cluster
* Massively Scalable - Supports a variety of enterprise-scale storage capabilities including [S3 Object Storage](https://www.jfrog.com/confluence/display/JFROG/S3+Object+Storage), [Google Cloud Storage](https://www.jfrog.com/confluence/display/JFROG/Google+Cloud+Storage), [Azure Blob Storage](https://www.jfrog.com/confluence/display/JFROG/Azure+Blob+Storage) and [Filestore Sharding](https://www.jfrog.com/confluence/display/JFROG/Filestore+Sharding) providing unlimited scalability, disaster recovery, and unmatched stability and reliability
* Replication – supports on-demand, on-schedule or event-based replication
* High Availability – Complete active/active HA solution with live failover and non-disruptive production upgrades
* Advanced CI Server integration with build tools
* Custom API-Driven Automation – It exposes an extensive [REST API](https://www.jfrog.com/confluence/display/JFROG/Artifactory+REST+API) that provides access to its features throughout development cycle. API allows to manage builds, repositories, and artifacts, perform searches, apply configurations, perform maintenance tasks and more
* Advanced Search with Artifactory Query Language – AQL gives you unprecedented flexibility in how you search for artifacts. It offers a simple way to formulate complex queries that specify any number of search criteria, filters, sorting options and output fields
* Artifactory Cloud with CDN Distribution - JFrog Artifactory Cloud with Amazon's CloudFront CDN solution allows Enterprise users to manage, control, and distribute high volumes of software distribution across multiple locations

**Amazon Elastic Container Register (ECR)** –CI can push to any Registry. ECR is the one of the registry which is using AWS. ECR is a private registry hosts and manage your container images and artifacts consisting of Docker and Open Container Initiative (OCI) in a highly available and scalable architecture.

* 1. **Sources or Reference to get more details**

|  |  |  |
| --- | --- | --- |
| **#** | **Tool/Repository** | **Reference Link** |
| 1 | GitLab | **https://docs.gitlab.com** |
| 2 | Jfrog | **https://jfrog.com** |
| 3 | AWS ECR | **https://docs.aws.amazon.com/AmazonECR/** |
| 4 | Docker | <https://www.docker.com> |
| 5 | Maven | <https://maven.com> |

|  |  |
| --- | --- |
| A pipeline with sample stages and actions. | * Automate release processes * Establish a consistent release process * Speed up delivery while improving quality * Allows integration of 3rd party tools * Pipeline progress and execution history |

*Source – docs.aws.amazon.com*



* 1. **Continuous Integration (CI) Pipeline repository**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Script File** | **Notes** | **Attachment** |
| 1 | Config.yml | Complete code for CI pipeline |  |
| 3 | Config.yml | ProductDeployable Script |  |