

# Implementing DevSecOps Pipeline for an Enterprise Organization

Nilima Chavan<sup>1</sup>, Nikita Bharambe<sup>2</sup>, Srushti Deshmukh<sup>3</sup>, Dipali Ahire<sup>4</sup>, Prof. A. R. Jain<sup>5</sup>

Students, Department of Information Technology<sup>1,2,3,4</sup>

Guide, Department of Information Technology<sup>5</sup>

Pune Vidyarthi Griha's College of Engineering & S. S. Dhamankar Institute of Management, Nashik, India

**Abstract:** *DevSecOps is an organizational software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops). DevSecOps is an extension of DevOps, which is considered as a means to intertwine development, operation and security. The main characteristic of DevSecOps is to improve customer outcomes and mission value by automating, monitoring, and applying security at all phases of the software lifecycle: plan, develop, build, test, release, deliver, deploy, operate, and monitor. Continuous practices, i.e., continuous integration, delivery, and deployment, are the software development industry practices that enable organizations to frequently and reliably release new features and products. DevSecOps means thinking about application and infrastructure security from the start. With the increasing interest in the literature on continuous practices, it is important to systematically review and synthesize the approaches, tools, challenges, and practices reported for adopting and implementing continuous practices. This paper aimed at systematically reviewing the state of the art of continuous practices to classify approaches and tools, identify challenges and practices in this regard, and identify the gaps for future research. We used the systematic literature review method for reviewing the peer reviewed papers on continuous practices published between 2004 and June 1, 2016. We applied the thematic analysis method for analysing the data extracted from reviewing 69 papers selected using predefined criteria. Conclusion: Although DevSecOps is getting increasing attention by industry, it is still in its infancy and needs to be promoted by both academia and industry.*

**Keywords:** Container Orchestration and Resource Management Platform; DevSecOps; CI/CD Pipelines; Infrastructure as Code; Policy as Code; Observability As Code; GitOps; Workflow Models.

## I. INTRODUCTION

### 1.1 Overview

#### What is DevSecOps?

It stands for development, security and operation. Like DevOps, DevSecOps is a culture that is followed while developing software applications. It is short for development, security and operation. Involve Continuous development, Continuous Integration, Continuous deployment, Continuous testing, Continuous monitoring throughout its development life cycle. DevSecOps offers organizations with benefits of collaborative work culture along with complete software security.

#### What Does DevSecOps Do?

It deploys the appropriate automated application security testing tools. It is their responsibility to make users aware of how to make most of application security features. Software projects have become a complex mixture of different moving parts both human and machine.

#### What is DevSecOps Pipeline?

Put it simply, DevSecOps refers to integrating security into your software development life cycle. So, a DevSecOps pipeline is a set of security practices incorporated into your SDLC to build, test, and deploy secure software faster and easier. At the same time, adopting continuous practices is not a trivial task since organizational processes, practices, and tools may not be ready to support the highly complex and challenging nature of these practices. Practicing DevSecOps

provides demonstrable quality and security improvements over the traditional software lifecycle, which can be measured with these metrics: Mean-time to production: the average time it takes from when new software features are required until they are running in production. Average lead-time: how long it takes for a new requirement to be delivered and deployed. Deployment speed: how fast a new version of the application can be deployed into the production environment. Deployment frequency: how often a new release can be deployed into the production environment. Production failure rate: how often software fails during production. Mean-time to recovery: how long it takes applications in the production stage to recover from failure. In addition, DevSecOps practice enables: Fully automated risk characterization, monitoring, and mitigation across the application lifecycle. Software updates and patching at a pace that allows the addressing of security vulnerabilities and code weaknesses. DevSecOps practice enables application security, secure deployment, and secure operations in close alignment with mission objectives. In DevSecOps, testing and security are shifted to the left through automated unit, functional, integration, and security testing - this is a key DevSecOps differentiator since security and functional capabilities are tested and built simultaneously.

### **1.2 Motivation**

Due to the growing importance of continuous practices, an increasing amount of literature describing approaches, tools, practices, and challenges has been published through diverse venues. Evidence for this trend is the existence of five secondary studies on CI, rapid release, CDE and CD. These practices are highly correlated and intertwined, in which distinguishing these practices is sometimes hard and their meanings highly depends on how a given organization interprets and employs them. Whilst CI is considered the first step towards adopting CDE practice, truly implementing CDE practice is necessary to support automatically and continuously deploying software to production or customer environments (i.e., CD practice). We noticed that there was no dedicated effort to systematically analyse and rigorously synthesize the literature on continuous practices in an integrated manner. By integrated manner we mean simultaneously investigating approaches, tools, challenges, and practices of CI, CDE, and CD, which aims to explore and understand the relationship between them and what steps should be followed to successfully and smoothly move from one practice to another. This study aimed at filling that gap by conducting a Systematic Literature Review (SLR) of the approaches, tools, challenges and practices for adopting and implementing continuous practices.

### **1.3 Problem Definition & Objective**

The main purpose of this document is to provide a logical description of the key design components and processes to provide a repeatable reference design that can be used to instantiate a DoD DevSecOps software factory. The target audiences for this document include: DoD Enterprise DevSecOps capability providers who build DoD Enterprise DevSecOps hardened containers and provide a DevSecOps hardened container access service DoD organization DevSecOps teams who manage (instantiate and maintain) DevSecOps software factories and associated pipelines for its programs DoD program application teams who use DevSecOps software factories to develop, secure, and operate mission applications Authorizing Officials (AOs) The DoD Enterprise DevSecOps reference design leverages a set of hardened DevSecOps tools and deployment templates that enable DevSecOps teams to select the appropriate template for the program application capability to be developed. For example, these templates will be specialized around a specific programming language or around different types of capabilities such as web application, transactional, big data, or artificial intelligence (AI) capabilities. A program selects a DevSecOps template and toolset; the program then uses these to instantiate a DevSecOps software factory and the associated pipelines that enable Continuous Integration and Continuous Delivery (CI/CD) of the mission application.

The purpose and intent of DevSecOps is to build on the mind-set that everyone is responsible for security with the goal of safely distributing security decisions at speed and scale to those who hold the highest level of context without sacrificing the safety required," describes Shannon Lietz, co-author of the "DevSecOps Manifesto. A DevSecOps pipeline is a set of automated processes and tools that allows developers and operations professionals to collaborate on building and deploying code to a production environment.

### **1.4 Project Scope**

The main scope of this project is to build a secure CI/CD pipeline with DevSecOps. This system key or highlighting features would be, to be able to detect issues, bugs and vulnerability in code with automated processes. This process will element most of the human dependencies and automate the Continuous Integration and Continuous Deployment, which is if any code changes are done it will automatically Deploy the project to the specified environment.

The main objective of DevSecOps is to automate, monitor and apply security at all phases of the software lifecycle, i.e., plan, develop, build, test, release, deliver, deploy, operate and monitor. Is to enable teams to release a constant flow of software updates into production to quicken release cycles, lower costs, and reduce the risks associated with development.

## **II. BACKGROUND AND LITERATURE SURVEY**

### **2.1 Background**

Here we give an overview of continuous software engineering (e.g., continuous integration, continuous delivery, and continuous deployment) paradigm.

#### **Continuous Software Engineering:**

Is an emerging area of research and practice. It refers to developing, deploying and getting quick feedback from software and customers in a very rapid cycle. Continuous software engineering involves three phases: Business Strategy and Planning, Development and Operations. This study focuses on only three software development activities: continuous integration, continuous delivery and continuous deployment.

#### **CI/CD pipeline:**

CI/CD pipeline or continuous integration and continuous delivery is referred to as the backbone of devsecops approach. The pipeline is responsible for building codes, running tests, and deploying new software versions.

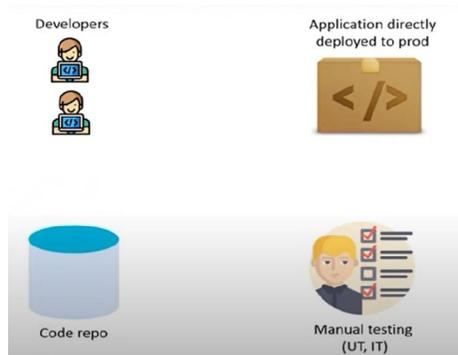
#### **Continuous Integration (CI):**

Is a widely established development practice in the software development industry, in which members of a team integrate and merge development work (e.g., code) frequently, for example multiple times per day. CI enables software companies to have shorter and frequent release cycles, improve software quality, and increase their teams' productivity. This practice includes automated software building and testing.

#### **Continuous Deployment (CD):**

Practice goes a step further and automatically and continuously deploys the application to production or customer environments. There is robust debate in academic and industrial circles about defining and distinguishing between continuous deployment and continuous delivery. What differentiates continuous deployment from continuous delivery is a production environment (i.e., actual customers): the goal of continuous deployment practice is to automatically and steadily deploy every change into the production environment. It is important to note that CD practice implies CDE practice but the converse is not true. Whilst the final deployment in CDE is a manual step, there should be no manual steps in CD, in which as soon as developers commit a change, the change is deployed to production through a deployment pipeline. DevSecOps is an organizational software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops). The main characteristic of DevSecOps is to improve customer outcomes and mission value by automating, monitoring, and applying security at all phases of the software lifecycle: plan, develop, build, test, release, deliver, deploy, operate, and monitor. Practicing DevSecOps provides demonstrable quality and security improvements over the traditional software lifecycle, which can be measured with these metrics: Mean-time to production: the average time it takes from when new software features are required until they are running in production.

- Average lead-time: how long it takes for a new requirement to be delivered and deployed.
- Deployment speed: how fast a new version of the application can be deployed into the production environment.
- Deployment frequency: how often a new release can be deployed into the production environment.
- Production failure rate: how often software fails during production.
- Mean-time to recovery: how long it takes applications in the production stage to recover from failure.



**Fig: 2.1 DevSecOps**

There is a developer who was committing the code repository. Now this repository will compile the code and send it to the quality assurance team. This team manually performs tests like unit test, integration test to check any logical error in the code once you have done facing the error directly deployed application to production.

CD Pipeline:



**Fig 2.2: CD pipeline**

In this case we see that the developers are directly committing the code into the continuous delivery pipeline. Now this pipeline has a few modules. First is commit in commit where the code and any changes to the code are committed or stored in the version control system. Now what is the version control system now it basically helps to track all the changes in the code over a period of time.

Why version control system ex- the developers who build the application and you were to update the application from the version one to version 2 and audit to that make a few changes and commit to the code in version control system now that we have started the application and version 2 it's a facing lot of issues to version 1, he easily does because the version control system has track and save all of the code changes is made.

Now look at the build module is implemented in continuous integration using tools like jenkins so what is continuous integration now every time a commit is made to the version control system that commits continuously pooled, built and tested using an automation tool. Instead of manually

Performing unit tests and integration tests we can automate using tools such as selenium test engines.

Now next is a Test Environment. This is a most important part of the continuous delivery pipeline because it makes sure the software is always in production that is the software is being saturated ready to release.

And in order to do this various test are carried out such as user acceptance test and low test. Low testing is a performance to check behaviour of the application under a certain load.

## 2.2 Existing Literature Reviews

### Reference Paper

During this review, we also found five papers that have reported reviews on different aspects of continuous software engineering - two studies have investigated continuous integration in the literature, two papers have explored continuous delivery and deployment. We summarize the key aspects of these studies. Ståhl and Bosch have presented a SLR on different attributes or characteristics of CI practice. That review has explored the disparity in implementations of CI practice in the literature. Based on 46 primary studies, the study had extracted 22 clusters of descriptive statements for implementing CI.

A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC).

In this paper, 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC) Continuous Integration (CI) and Continuous Delivery (CD) have become a well-known practice in DevOps to ensure fast delivery of new features. This is achieved by automatically testing and releasing new software versions. Identify unique research/technology challenges the DevSecOps communities will face and propose preliminary solutions to these challenges. Our findings will enable informed decisions when employing DevSecOps practices in agile enterprise applications engineering processes and enterprise security.

Design and Implementation of Security Test Pipeline based on DevSecOps 021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)

In this paper, in recent years, a variety of information security incidents emerge endlessly, with different types. Security vulnerability is an important factor leading to the security risk of information systems, and is the most common and urgent security risk in information systems. The research goal of this paper is to seamlessly integrate the security testing process and the integration process of software construction, deployment, operation and maintenance. Through the management platform, the security testing results are uniformly managed and displayed in reports, and the project management system is introduced to develop.

### **III. SYSTEM AND REQUIREMENTS AND SPECIFICATIONS**

In this chapter we are going to have an overview on- problem definition; system requirements like software requirements, hardware requirements; and system features like text representation, machine learning algorithm.

#### **3.1 Problem Definition**

Every organization has to prioritize its activities, and DevSecOps may not be everyone's top priority. In some cases, organizations may not be able to integrate security into their DevOps process because they are dependent on some environment and script changes. Or the team may not have the capacity to take on these changes due to other priorities.

The DevSecOps is the theory, or we can say the philosophy of adopting security practices with the DevOps process. It is also used to describe a continuous delivery, security-focused software development life cycle (SDLC). It is often seen that the Security into DevOps is treated as the secondary system. InfoSec often comes at the end of the Software development life cycle (SDLC). It can be very frustrating to discover the security vulnerabilities at the end of the SDLC. DevSecOps promotes security engagement to a significant or active part of the Software development life cycle (SDLC). The General DevOps have introduced processes like Continuous Integration and Continuous Delivery, also known as the CI/CD. The Continuous Integration and Continuous Delivery process ensure continuous testing and verification of the code correctness during the agile process development.

#### **3.2 Requirement Specification**

##### **3.2.1 Hardware Requirements**

- 10 GB HDD
- 8 GB RAM

##### **3.2.2 Software Requirements**

- Ansible
- Jenkins
- Kubernetes
- Docker
- MySQL/Mongo dB

### **IV. SYSTEM DESIGN**



DevSecOps describes an organization’s culture and practices enabling organizations to bridge the gap between developers, security team, and operations team; improve processes through collaborative and agile workflows; drive for faster and more secure software delivery via technology; and achieve consistent governance and control. There is no uniform DevSecOps practice. Each DoD organization needs to tailor its culture and its DevSecOps practices to its own unique processes, products, security requirements, and operational procedures. Embracing DevSecOps requires organizations to shift their culture, evolve existing processes, adopt new technologies, and strengthen governance.

4.1 System Architecture

When approaching a complex DevSecOps implementation, it is often useful to consider Architecture as a starting point. As illustrated in Figure, the automation activities can be broken up into three major areas: Continuous Integration (CI), Continuous Deployment (CD) and Continuous Compliance (CC). Each of these areas encompasses a separate target for DevSecOps implementation. For example, a team can automate the build, test, and security scan aspects of CI without fully implementing automated deployment. However, to realize the full benefit of the architecture, it is necessary to have all areas built out and functioning as a unit. This includes proper integration across the selected pipeline of tools.

There are many tools available in the marketplace to enable DevSecOps automated delivery. For example, many automated pipelines are managed by Cloud Bees Jenkins workflow automation. Source control is often fulfilled by a variant of ‘git’, such as Bit Bucket, GitLab, or GitHub. Automated code scanning can be accomplished with a variety of open-source and commercial products, such as SonarQube, Veracode, Sonatype-Nexus IQ, or Blackduck. Xebialabs (provider of deployment tools) offers a very nice summary page for most of the currently available tooling.

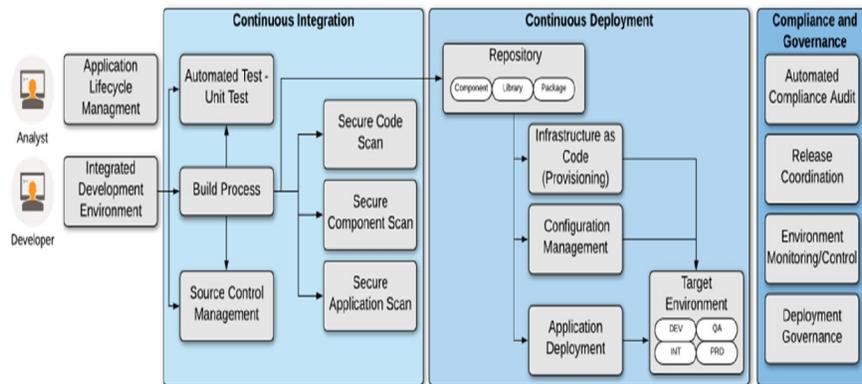


Figure 4.1 DevSecOps Pipeline

Continuous Integration (CI)

Continuous Integration automates the repetitive aspects of system building, packaging, unit testing, and security scanning. This allows development teams to focus energy on fulfilling feature requests rather than on the mechanics of system building. Moreover, this approach addresses the dreaded “it works on my laptop” problem where code changes break when combined with contributions from other team members. Building a full system on every approved submission (e.g. merge to the Development or Master code line) allows for verification of unit test coverage, early detection of security policy violations, integration failures, and collisions with other developer’s code.

Source Control

Source control is a key aspect of CI automation. The selection of the proper SCM tool facilitates integration with the pipeline workflow such that key events can trigger pipeline activities. For example, a product such as BitBucket can integrate directly with Jenkins to initiate the CI pipeline build process upon specific events (such as a code merge or a pull request approval). By selectively choosing these code-level events, the CI pipeline can be leveraged to dramatically speed development.

Build Management

Many, if not most, software systems must be compiled, linked, or otherwise converted from raw code into an executable product. These steps typically include the resolution of dependencies, such as third-party components or libraries. The end result of the build phase is a versioned “deployable unit” as the basis of automated deployment.

### **Automated Unit Test and Test-Driven Development**

After the system builds, it is common in DevSecOps pipelines to perform some level of automated unit test. Teams who leverage TTD (test-driven development) are best able to leverage automated unit testing, given the unit tests are the first code artifacts created for any given feature.

### **Security and Compliance Assurance**

Automation of secure code verification is a core best practice for DevSecOps automation. There are three automated scans performed for each CI build: secure code practices, third-party component use, and application penetration testing. The first two scans are often considered “static” code scans as they refer to the code base before compilation. The third test looks at the end product (such as a web-application) and verifies resistance against common attacks (e.g., OWASP Top Ten vulnerabilities).

### **Continuous Deployment (CD)**

In contrast to CI, Continuous Deployment manages the consistent delivery of deployable units into target environments. This element of DevSecOps automation manages system parameters, connection details, encryption, secrets, and other ‘run-time’ requirements. Deployment tooling simplifies the management of these values.

### **Repository Implementation**

All deployments should be made from a verified deployable unit. This is to ensure that each promotion to subsequent environments (e.g., Dev to Test to Production) always leverages the exact same build products. The use of a repository is therefore critical to ensuring versioning of these deployable units. Moreover, repositories can manage libraries, components, and other build-time dependencies.

### **Infrastructure as Code**

A goal of DevSecOps automation is to reduce or eliminate repetitive tasks from the delivery pipeline. The target environment for a particular system build should always be in a predictable state. This avoids time-consuming deployment problems when one environment configuration drifts from another. By encoding the expected system, secondary installed components, and network configuration this uniformity can be enforced at deploy-time.

### **Configuration Management**

Along with establishing and maintaining a target environment configuration, the associated system parameters and connection information must be maintained. As noted above, this is the purpose of configuration and secret management tools. Decoupling environment configuration information from the deployable unit greatly reduces the likelihood of a deployment error breaking the automation.

### **Continuous Compliance (CC) and Governance**

The final DevSecOps reference architecture section covers compliance and governance. For many, compliance with regulatory, industry, or customer demands requires that the organization spend valuable time verifying and approving adherence to security policy. To the greatest extent possible, this compliance audit function should be automated.

### **Automated Compliance Audit**

It is important to verify the configuration state of any target environment to ensure the consistency of servers, networks, and data access points. Likewise, the security hardening of operating systems, network policy/restrictions, firewall/load balancer configuration, and other system security aspects should be verified before system deployment. This is particularly necessary for production deployment, but should not be neglected for lower environments.

### **Environment Monitoring and Control**



Support for operations and on-going system monitoring is the final aspect of the DevSecOps reference architecture. This will enable the operations team to better respond to problems before they occur.

4.2 UML Diagrams

This Section content nine UML Diagram, which clearly specify the exact functionality of the prototype and they are as follows,

1. Class Diagram
2. Object Diagram
3. Workflow Diagram
4. State Diagram
5. Use Case Diagram
6. Activity Diagram
7. Sequence Diagram
8. Component Diagram
9. Deployment Diagram

4.2.1 Class Diagram

Mainly there are three classes DevSecOps, Pipeline and Database. DevSecOps will manage all security as code features operations and improves the agility of software. With the help of devsecops the Pipeline will be built. Pipeline will contain all the phases for development like plan, code, build, deploy, manage and it will perform operations like tracking of work, pairing, scanning, finding errors. Database will store the data which is developed.

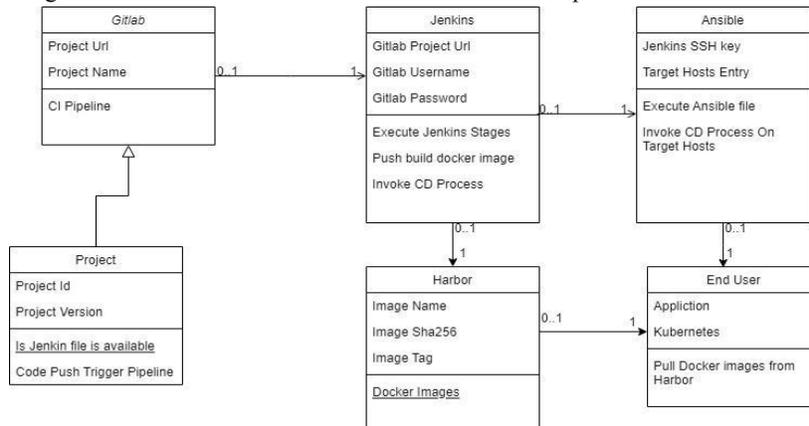


Figure 4.2 Class Diagram

4.2.2 Object Diagram

In this object diagram the multiple objects of DevSecOps user are present.

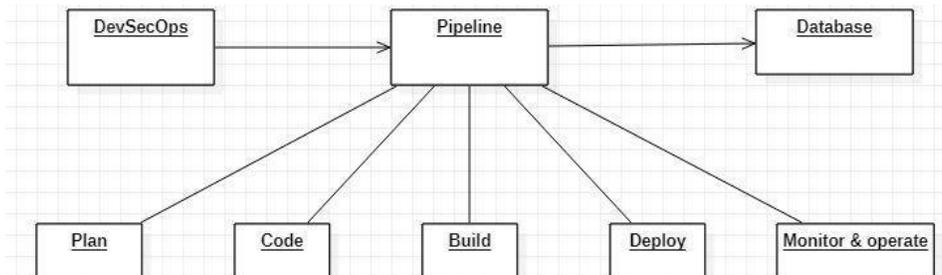


Figure 4.3 Object Diagram

4.2.3 Workflow Diagram



In workflow Diagram the private Registries, Version Control, Branch Strategy Host Inventory.

In JU While creating the project add the Jenkins CI user as maintainer in Gitlab. Jenkins CI user created in Jenkins and jobs are run in through it. GU Gitlab 360 Users has been configured in Jenkins. Ansible Server has been configured in Jenkins for SSH.

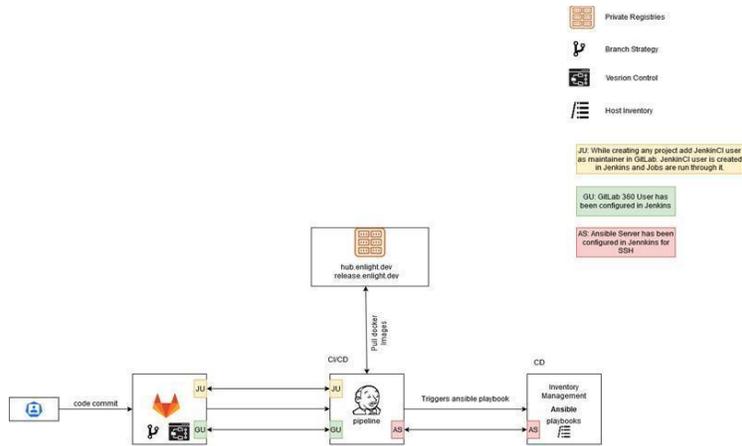


Figure 4.4 Workflow Diagram

### 4.2.4 State Diagram

In this diagram the state of the DevSecOps process is defined.

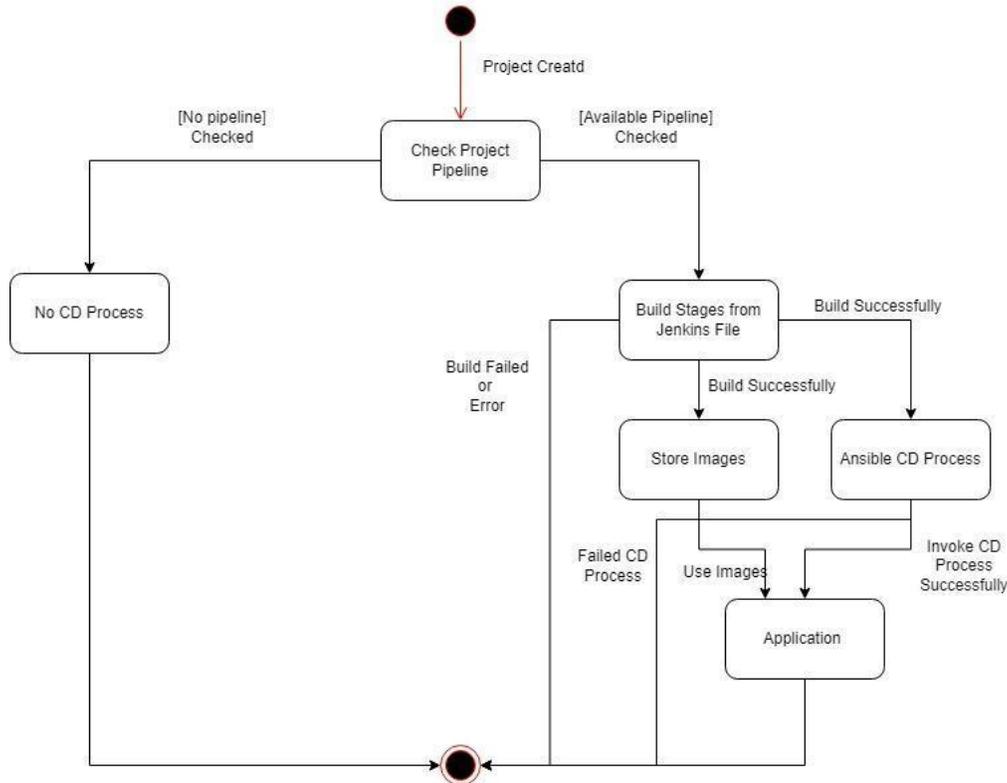


Figure 4.5 State Diagram



4.2.5 Use Case Diagram

In this system there are five main actors. The product owner will first login to the module then he can manage the project, manage phases and manage requirements. The system has to be first login and to manage the associations. Security master will control all security related features. Tester will login and will execute and manage the test cases. Developers will manage all the Developments related to the pipeline.

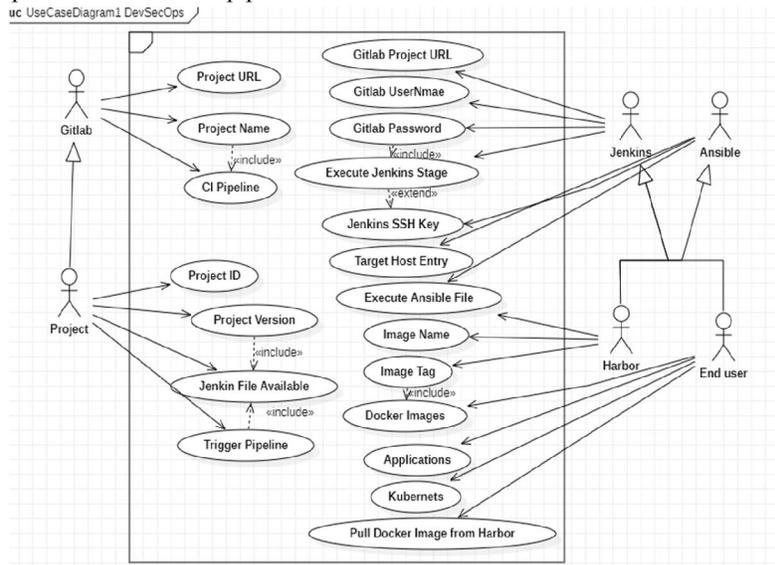


Figure 4.6 Use Case Diagram

4.2.6 Activity Diagram

This activity diagram shows how the activity is performed and changed as per the condition. The activity performed the DevSecOps Pipeline general conditional flow. This diagram helps in easy decisions in the implementation phase.

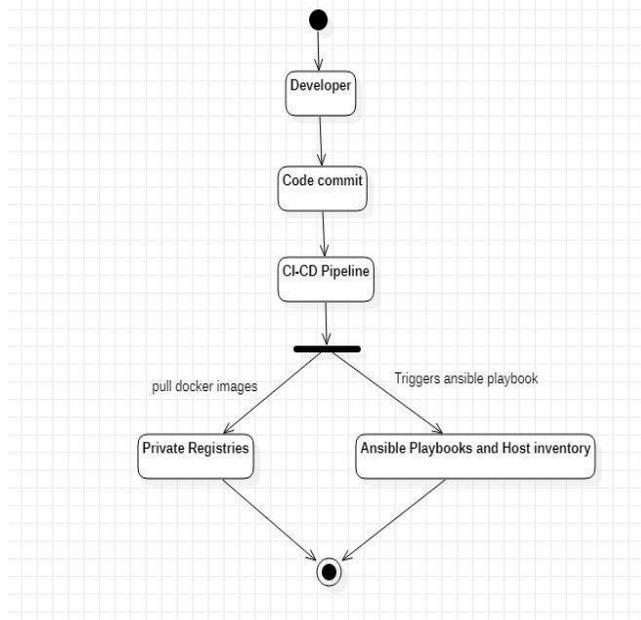


Figure 4.7 Activity Diagram



4.2.7 Sequence Diagram

This is a sequence diagram for DevSecOps Pipeline for an enterprise organization.

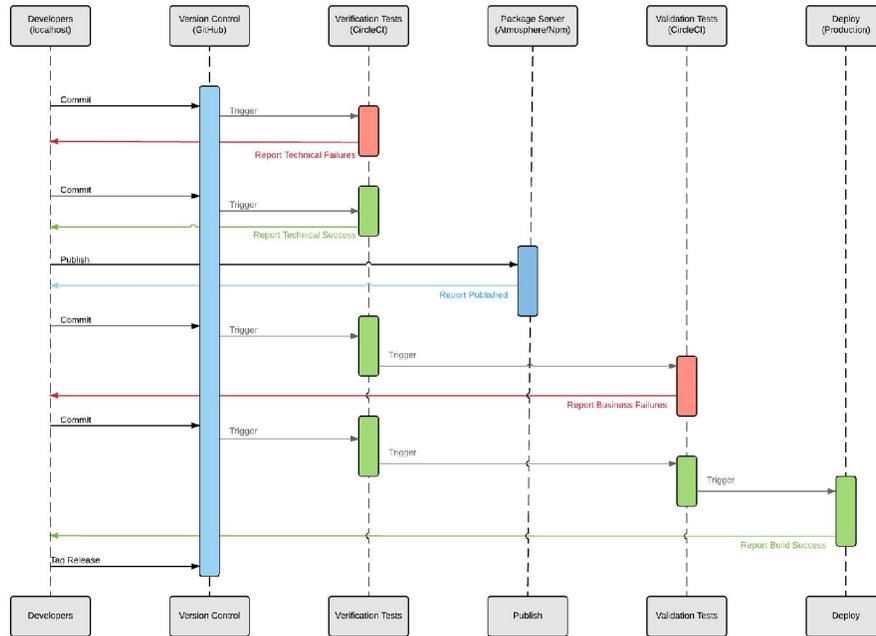


Figure 4.8 Sequence Diagram

4.2.8 Component Diagram

In the Component Diagram there are three main components DevSecOps, Pipeline and Database. The making software components which are used in developing the prototype are shown in the diagram. The relation and the data flow between the components are figured out using this diagram.

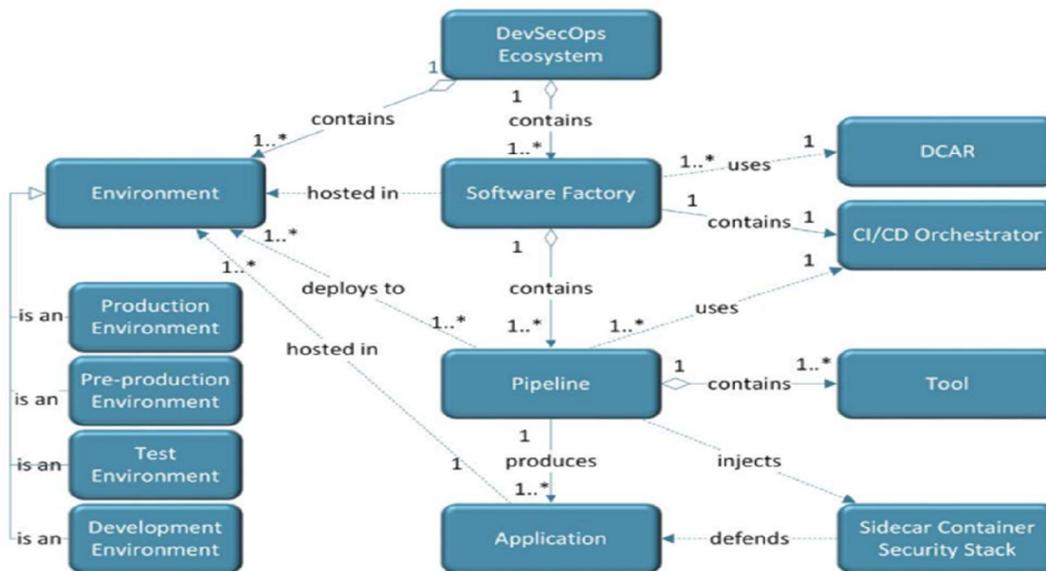


Figure 4.9 Component Diagram



In the Deployment diagram there are four nodes namely server, client, prototype, and database. All the clients are connected to the database. This deployment diagram shows the hardware requirement of the prototype. How this prototype is working with hardware is the main logic behind the deployment diagram.

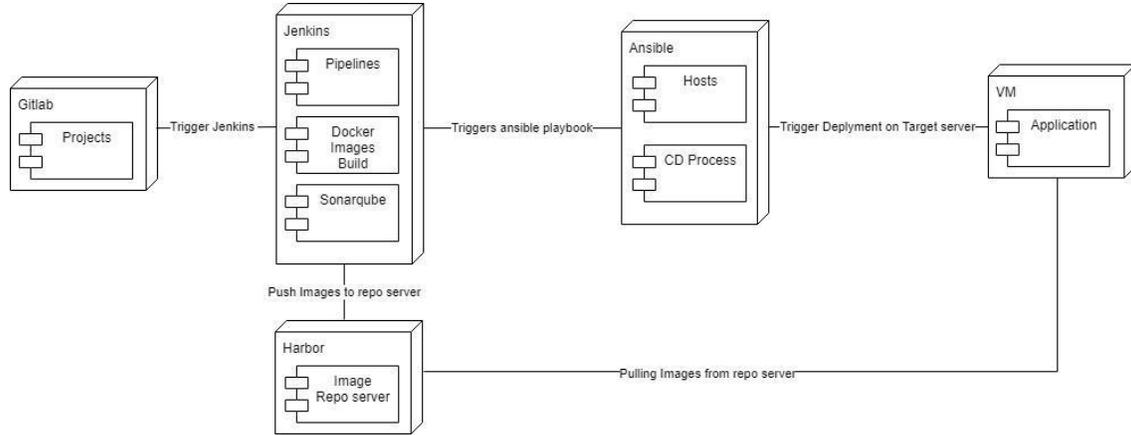


Figure 4.10 Deployment Diagram

V. Project Plan

In this chapter we are going to have an overview about how much time does it took to complete each task like- Preliminary Survey Introduction and Problem Statement, Literature Survey, Project Statement, Software Requirement and Specification, System Design, Partial Report Submission, Architecture Design, Implementation, Deployment, Testing, Paper Publish, Report Submission and etcetera.

This chapter also focuses on the stakeholder list which gives information about project type, customer of the proposed system, user and project member who developed the system.

5.1 Stakeholder List

The Stakeholder list shows the persons who are interacting with the prototype in various roles.

Sr. No.	Stakeholder	DevSecOps pipeline for an enterprise organization
1	Project Type	Implementing work
2	Customer	CD/CI
3	User	DevSecOps
4	Project Team Member	1. Chavan Nilima N. 2. Bharambe Nikita D. 3. Ahire Dipali R. 4. Deshmukh Shruti.

Table 5.1: Stakeholder List

5.2 System Implementation Plan

The System Implementation plan table, shows the overall schedule of tasks completion and time duration required for each task.

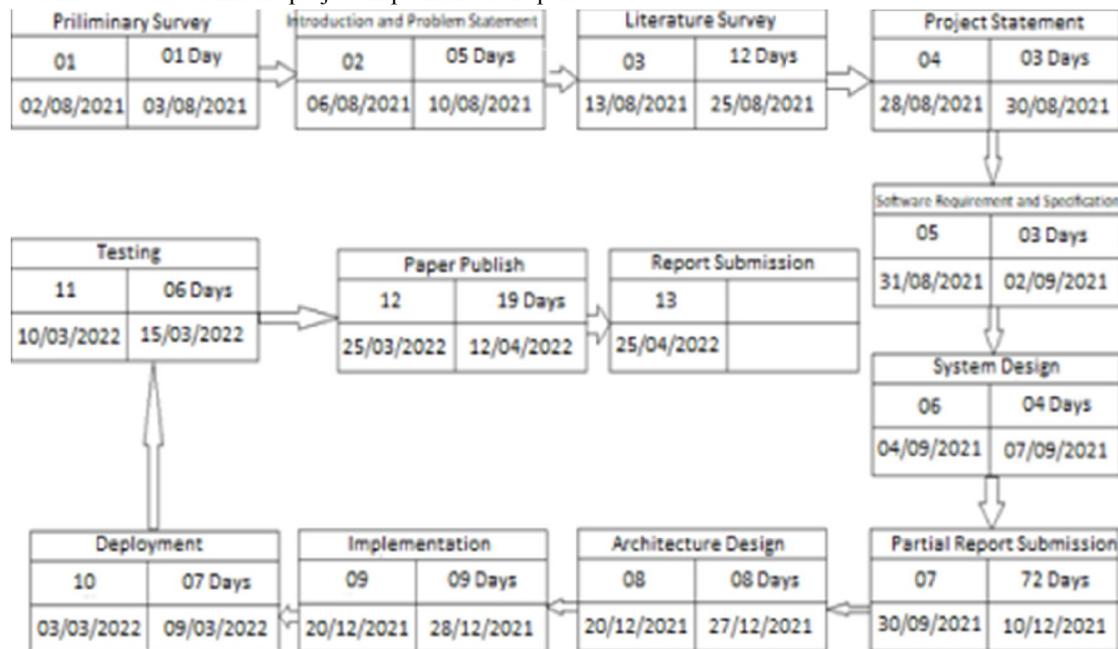
Task No.	Task Name	Start Date	End Date
----------	-----------	------------	----------

1	Preliminary Survey	02-08-2021	03-08-2021
2	Introduction and Problem Statement	06-08-2021	10-08-2021
3	Literature Survey	13-08-2021	25-08-2021
4	Project Statement	28-08-2021	30-08-2021
5	Software Requirement and Specification	31-08-2021	02-09-2021
6	System Design	04-09-2021	07-09-2021
7	Partial Report Sub-Mission	30-09-2021	10-12-2021
8	Architecture Design	20-12-2021	27-12-2021
9	Implementation	20-12-2021	28-02-2022
10	Deployment	03-03-2022	09-03-2022
11	Testing	10-03-2022	15-03-2022
12	Paper Publish	25-03-2022	12-04-2022
13	Report Submission	9-05-2022	-

**Table 5.2:** System Implementation Plan

### 5.3 PERT Chart

The PERT Chart Stands for program evaluation review technique. This PERT Chart of the prototype shows the schedule and coordinated tasks within the project implementation span.



**Figure 5.3:** PERT chart



In this chapter methodology, algorithms, and modules are explained.

6.1 Overview of Project model

DevSecOps is a way of approaching IT security with an “everyone is responsible for security” mind-set. It involves injecting security practices into an organization’s DevOps pipeline. The goal is to incorporate security into all stages of the software development workflow. That’s contradictory to its predecessor development models—DevSecOps means you’re not saving security for the final stages of the SDLC. If your company already does DevOps, then it’s a good idea to consider shifting toward DevSecOps. At its core, DevSecOps is based on the principle of DevOps, which will help your case for making the switch. And doing so will enable you to bring together proficient individuals from across different technical disciplines to enhance your existing security processes.

DevOps is a methodology meant to improve work throughout the software development lifecycle. You can visualize a DevOps process as an infinite loop, comprising these steps: plan, code, build, test, release, deploy, operate, monitor and -- through feedback -- plan, which resets the loop.

6.2 Software Factory Tools and Activities

CI/CD Orchestrator: -

The CI/CD Orchestrator is the central automation engine of the CI/CD pipeline. It manages pipeline creation, modification, execution, and termination. The DevSecOps team creates a pipeline workflow in the Orchestrator by specifying a set of stages, stage conditions, stage entrance and exit control rules, and stage activities.

Tool	Features	Benefits	Inputs	Outputs	Baseline
CI/CD Orchestrator	Create pipeline workflow	Customizable pipeline solution	Human input about: <ul style="list-style-type: none"> <li>• A set of stages</li> <li>• A set of event triggers</li> <li>• Each stage entrance and exit control gate</li> <li>• Activities in each stage</li> </ul>	Pipeline workflow configuration	MVP
	Orchestrate pipeline workflow execution by coordinating other plugin tools or scripts.	Automate the CI/CD tasks; Auditable trail of activities	Event triggers (such as code commit, test results, human input, etc.); Artifacts from the artifact repository	Pipeline workflow execution results (such as control gate validation, stage transition, activity execution, etc.); Event and activity audit logs	

Figure 6.1 CI/CD Orchestrator

6.2.1 Develop

The Develop phase uses tools to support the development activities that convert requirements into source code. The source code includes application code, test scripts, Infrastructure as Code, Security as Code, DevSecOps workflow scripts, etc. The development team may rely on a single modern integrated development environment (IDE) for multiple programming language support. The IDE code assistance feature aids developers with code completion, semantic coloring, and library management to improve coding speed and quality. The integrated compiler, interpreter, lint tools, and static code analysis plugins can catch code mistakes and suggest fixes before developers check code into the source code repository. Source code peer review or pair programming are other ways to ensure code quality control. All the code generated during development must be committed to the source code repository and thus version controlled. Committed code that breaks the build should be checked in on a branch and not merged into the trunk until it is fixed.

Tool	Features	Benefits	Inputs	Outputs	Baseline
Integrated development environment (IDE)	Source code editor Intelligent code completion Compiler or interpreter Debugger Build automation (integration with a build tool)	Visual representation Increase efficiency Faster coding with less effort Improved bug fixing speed Reproducible builds via scripts	Developer coding input	Source code	MVP
Integrated development environment (IDE) security plugins	Scan and analyze the code as the developer writes it, notify developer of potential code weakness and may suggest remediation	Address source code weaknesses and aid developers to improve secure coding skills	Source code; known weaknesses	source code weakness findings	Objective
Source code repository	Source code version control Branching and merging Collaborative code review	Compare files, identify differences, and merge the changes if needed before committing. Keep track of application builds	Source code Infrastructure as code	Version controlled source code	MVP
Source code repository security plugin	Check the changes for suspicious content such as Secure Shell (SSH) keys, authorization tokens, passwords and other sensitive information before pushing the changes to the main repository. If it finds suspicious content, it notifies the developer and blocks the commit.	Helps prevent passwords and other sensitive data from being committed into a version control repository	Locally committed source code	Security findings and warnings	Objective
Code quality review tool	View code changes, identify defects, reject or approve the changes, and make comments on specific lines. Sets review rules and automatic notifications to ensure that reviews are completed on time.	Automates the review process which in turn minimizes the task of reviewing the code.	Source code	Review results (reject or accept), code comments	Objective

Figure 6.2 Develop Phase Tools

### 6.2.3 Build

The build tools perform the tasks of building and packaging applications, services, and micro services into artifacts. For languages like C++, building starts with compiling and linking. The former is the act of turning source code into object code and the latter is the act of combining object code with libraries to create an executable file. For Java Virtual Machine (JVM) based languages, building starts with compiling to class files, then building a compressed file such as a jar, war or ear file, which includes some metadata, and may include other files such as icon images. For interpreted languages, such as Python or JavaScript, there is no need to compile, but lint tools help to check for some potential errors such as syntax errors. Building should also include generating documentation, such as Javadoc, copying files like libraries or icons to appropriate locations, and creating a distributable file such as a tar or zip file. The build script should also include targets for running automated unit tests.

### 6.2.4 Test

Test tools support continuous testing across the software development lifecycle. Test activities may include, but are not limited to, unit test, functional test, integration test, system test, regression test, acceptance test, performance test, and variety of security tests. Mission programs can select their own test activities and merge several tests together based on the nature of their software and environment. All tests start with test development, which develops detailed test procedures, test scenarios, test scripts, and test data. Automated tests can be executed by running a set of test scripts or running a set of test

scenarios on the specific test tool without human intervention. If full automation is not possible, the highest percentage of automation is desired. It is highly recommended to leverage emulation and simulation to test proper integration between components such as micro services and various sensors/systems, so integration testing can be automated as much as possible. Automation will help achieve high test coverage and make continuous ATO practicable, as well as significantly increase the quality of delivered software.

### 6.2.5 Modules

Modules for proposed system are as follows,

1. GitLab: GitLab is known for industry-leading Source Code Management (SCM) and Continuous Integration (CI). Developers want to use GitLab.
2. Jenkins: Jenkins is a free and open-source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.
3. SonarQube: SonarQube is a Code Quality Assurance tool that collects and analyses source code, and provides reports for the code quality of your project.

### 1. Jenkins

#### Jenkins Pipeline:

Pipeline is a series or collection of jobs or events. Which are linked with each other in sequence with the help of CI/CD.

Key Features of Jenkins Pipeline: -

- Pipeline as a code
- Code can be checked into a version control system
- Incorporates user input
- Restart from saved checkpoint
- Integrate with other plugins
- Allow conditional loops (for, when)

#### Jenkins Pipeline:



**Fig 6.3 Jenkins Pipeline**

Pipeline concepts:

```

pipeline {
  agent any
  stages {
    stage ('Build') {
      ...
    }
    stage ('Test') {
      ...
    }
    stage ('QA') {
      ...
    }
    stage ('Deploy') {
      ...
    }
    stage ('Monitor') {
      ...
    }
  }
}

```

Steps - steps are carried out in a sequence to execute a stage. Build the test and run the simple echo command

```
pipeline {
  agent any
  stages {
    stage ('Build') {
      steps {
        echo 'Running build phase...'
      }
    }
  }
}
```

Pseudo code for Jenkins pipeline

```
pipeline {
  agent any
  stages {
    stage('Clean existing Built Containers'){
      when { expression { (env.BRANCH_NAME =~ /(?!)(.*development.*[qa-.*][release])/) } }
      steps {
        sh 'docker rmi -f imageurl:$BRANCH_NAME || true'
        sh 'docker rmi -f imageurl:$BRANCH_NAME || true'
      }
    }
    stage('Download Repo'){
      when { expression { (env.BRANCH_NAME =~ /(?!)(.*development.*[qa-.*][release])/) } }
      steps {
        git branch: '$BRANCH_NAME', credentialsId: 'jenkin-git-user', url:
        'http://gitlab360.enlightcloud.com/rajnesh.kumar/reporting.git'
      }
    }
    stage('Encrypt Code'){
      when { expression { ! (env.BRANCH_NAME =~ /(?!)(.*development.*[qa-.*][release])/) } }
      steps {
        sh 'sh -x reporting_enc.sh'
      }
    }
    stage('Build container Image'){
      when { expression { (env.BRANCH_NAME =~ /(?!)(.*development.*[qa-.*][release])/) } }
      steps {
        sh 'docker build -t imageurl:$BRANCH_NAME laravel/laravel/'
        sh 'docker build -t imageurl:$BRANCH_NAME laravel/nginx/'
      }
    }
    stage('Push Image'){
      when { expression { (env.BRANCH_NAME =~ /(?!)(.*development.*[qa-.*][release])/) } }
      steps {
        withDockerRegistry(credentialsId: 'enlight360-dev-hub', url: 'https://hub.enlight.dev') {
          sh 'docker push imageurl:$BRANCH_NAME'
          sh 'docker push imageurl:$BRANCH_NAME'
        }
      }
    }
    stage('Invoked CD process') {
      steps {
        script {
          if (env.BRANCH_NAME =~ /(?!)(.*development.*[qa-.*][release])/) {
            build job: 'dev-modulename-deployment', parameters: [
              string(name: 'param1', value: 'value1')
            ]
            echo 'I execute development environment'
          } else if (env.BRANCH_NAME =~ /(?!)(qa-.*[release])/) {
            build job: 'qa-modulename-deployment', parameters: [
              string(name: 'param2', value: 'value2')
            ]
            echo 'I execute QA environment'
          } else if (env.BRANCH_NAME =~ /(?!)(dev-no-source-code)/) {
            build job: 'dev-no-source-code-modulename', parameters: [
              string(name: 'param3', value: 'value3')
            ]
            echo 'I execute dev-no-source-code environment'
          } else if (env.BRANCH_NAME =~ /(?!)(release)/) {
            build job: 'release-modulename', parameters: [
              string(name: 'param4', value: 'value4')
            ]
            echo 'I execute release environment'
          } else if (env.BRANCH_NAME =~ /(?!)(v.*)/) {
            build job: 'edu-uat-poc-modulename', parameters: [
              string(name: 'param5', value: 'value5')
            ]
            echo 'I execute edu-uat-poc-reporting environment'
          } else {
            echo 'NO Deployment is called'
          }
        }
      }
    }
  }
}
```

## 2. Gitlab:

Integrating security into your DevOps lifecycle is easy with GitLab. Security and compliance are built-in, out of the box, giving you the visibility and control necessary to protect the integrity of your software. GitLab is known for industry-leading Source Code Management (SCM) and Continuous Integration (CI). Developers want to use GitLab. We make it easy for them to develop more secure and compliant software. The GitLab DevOps platform shifts both security and compliance earlier in the development process with consistent pipelines that automate scanning and policies. Uniting developers and security pros within one platform streamlines vulnerability management for both and improves collaboration.

Configuring GitLab Instance

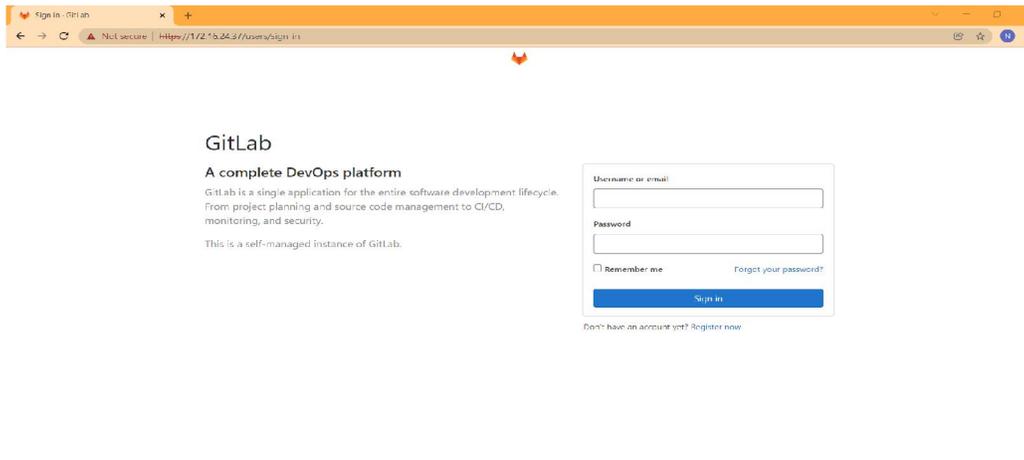
```
docker pull gitlab/gitlab-ce:latest
```

```
docker run --detach --hostname devops-cd-vm.enlight.dev --publish 443:443 --publish 80:80 --name gitlab-ce-ssl --restart always --volume $GITLAB_HOME/config:/etc/gitlab --volume $GITLAB_HOME/logs:/var/log/gitlab --volume $GITLAB_HOME/data:/var/opt/gitlab gitlab/gitlab-ce:latest
```

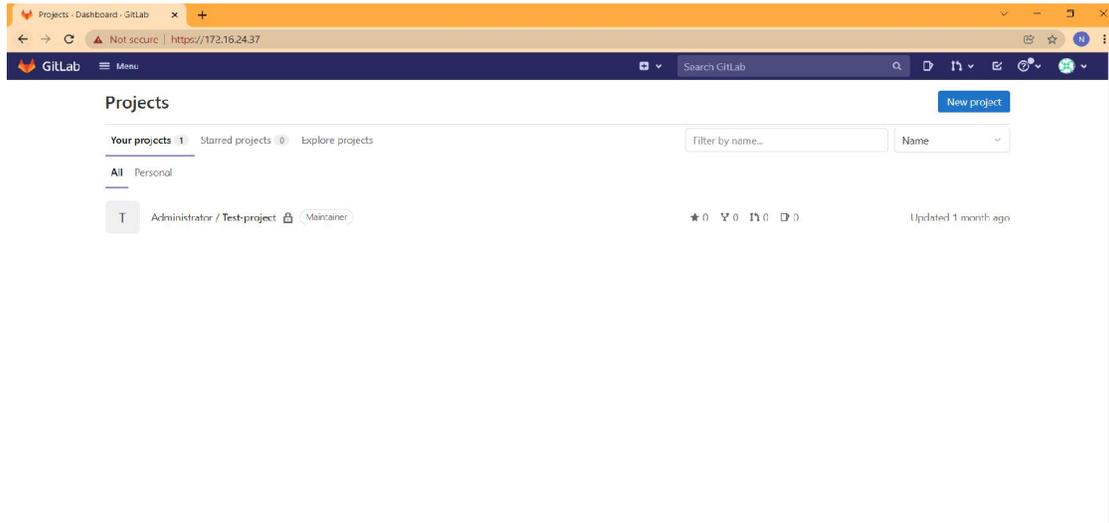
### Accessing GitLab Root Password

```
docker exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password
```

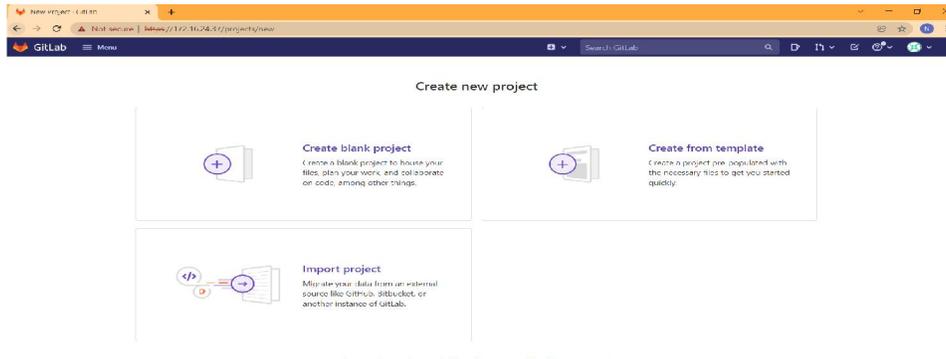
### GitLab Dashboard



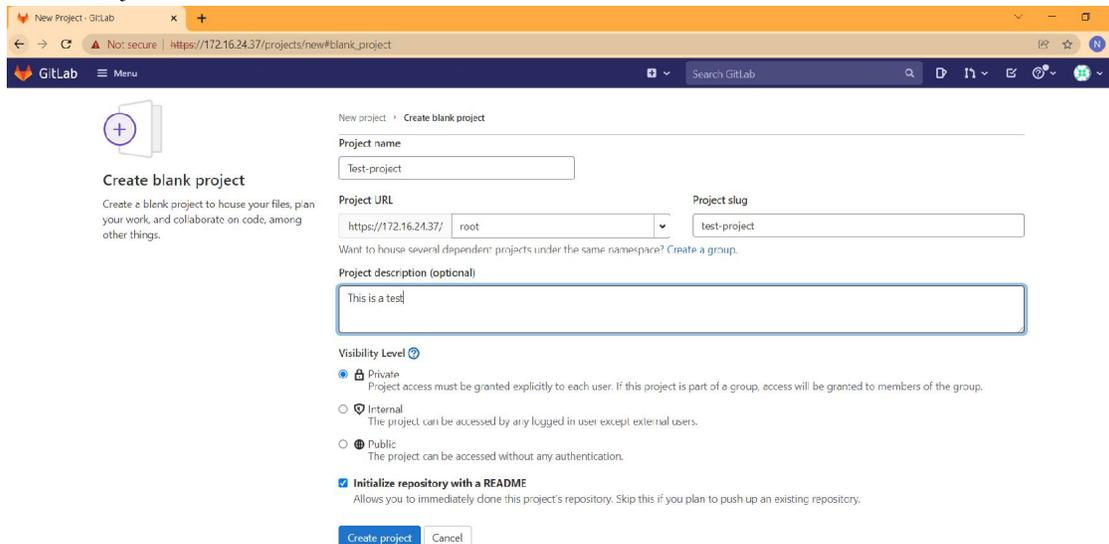
### GitLab Dashboard after Login



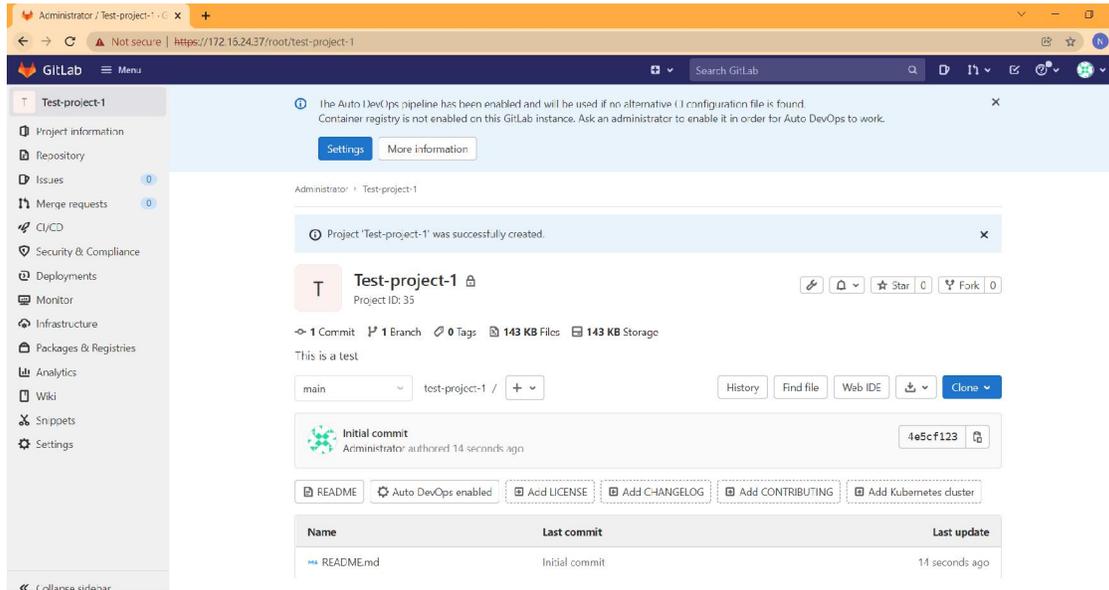
Create New Project



Create Blank Project



After Create Project  
Copyright to IJAR SCT  
www.ijarsct.co.in



### 3. SonarQube

#### SonarQube

SonarQube is a Code Quality Assurance tool that collects and analyses source code, and provides reports for the code quality of your project. It combines static and dynamic analysis tools and enables quality to be measured continually over time.

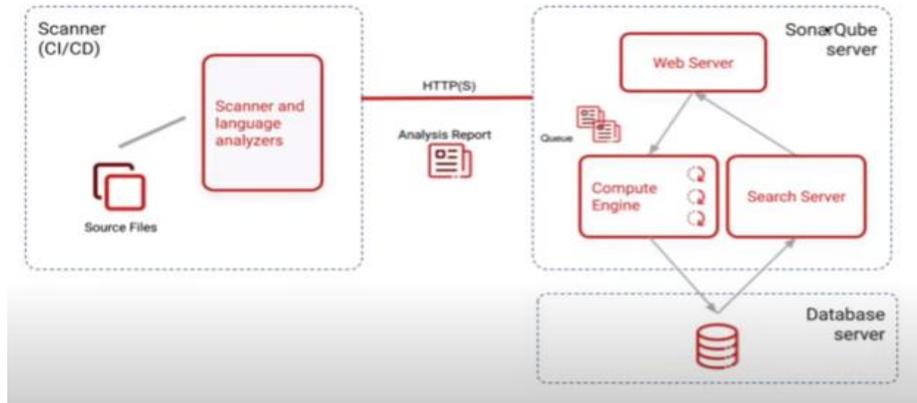
Some of the code quality checks are:

- Potential bugs: - Potential bugs is easy to understand
- Code defects to design inefficiencies: -Code defects to design inefficiencies basically means that when we write a code and it's not very design specific or it's not very compatible with the design or the structure of the application, we're going with then it can cause a lot of inefficiencies.
- Code duplication: -Code duplication takes a lot of memory to resolve that the tool helps us.
- Lack of test coverage: -Lack of test coverage which means that sometimes there aren't enough tests written for a specific code.

#### SonarQube Installation Steps:

### SonarQube Installation

1. Install Docker and pull SonarQube Image
2. Setup SonarQube on the Browser
3. Download Sample Code and Analyze it on Sonar



### 6.3 Algorithm

#### Round Robin Algorithm

The round robin algorithm sends traffic to a sequence of eligible pods in a predetermined order. For example, if you had five pods in a round robin configuration, the load balancer would send the first request to pod 1, the second request to pod 2, and so on down the line in a repeating cycle. The round robin algorithm is static, which means it will not account for variables such as the current load on a particular server. That's why round robin is typically preferred for testing environments and not for production traffic. Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get a fair share of CPU. One of the most commonly used techniques in CPU scheduling as a core.

## VII. SOFTWARE TESTING

In this chapter there is a relevant explanation on experimental setup, testing strategies used to test the system, and test cases.

### 7.1 Experimental Setup

#### Requirement for project

Sr. No.	Hardware Component	Component Description	Quantity
1	Personal Computer	Min 8 GB RAM, Min 10 GB Hard Disk	1

**Table 7.1:** Hardware Requirement

Sr.No.	Software Component	Component Description	Quantity
1	MySQL	Database	1
2	Proxy	Server	1
3	Docker, Kubernetes, Jenkins	Containers	1

**Table 7.2:** Software Requirement

Sr. No.	Client Component	Component Description	Quantity
1	Browser	Firefox, Google chrome	1

**Table 7.3:** Client Requirement

### 7.2 Testing Strategy



Testing Strategy used for testing the system are as follows,

1. Unit Testing
2. Integration Testing
3. Regression Testing

### 7.2.1 Unit Testing

In case of unit testing, each software component, software modules or software subsystem is tested independent of any other components involved in the whole software system.

That is individual software modules or software components are tested in unit testing. The main agenda behind unit testing is to verify and validate each and every unit of the software system by checking its working and performance and comparing it with the software specification. The significant control paths are tested and verified to discover errors within the boundary of the module and the component level design used for the same.

### 7.2.2 Integration Testing

Integration testing is a kind of testing meant for building the software architecture along with finding out the errors related with the interfacing. After successful execution of unit testing, software subsystems will be collected together and combined together in order to build the whole software system as it is specified and defined at high level design.

Integration testing is an efficient procedure for verification of the structure of a software system and validation of order of execution of software system while conducting tests to determine errors allied with interfacing.

### 7.2.3 Regression Testing

During the software development procedure, whenever the software system is modified by means of editing, removing, adding source code, software developers need to be sure that the new version of the software is as good as the earlier version. Tests that focus on the software modules that have been modified or altered and focus on overall functionality of the software system when the software functions are likely to be affected by the modifications or change.

## 7.3 Test Cases & Test Result

Jenkins Test Cases:

Sr no.	Module	Expected Result	Actual Result	Status
1	Login	Users should be able to login into Jenkins Dashboard.	Users are able to login into Jenkins Dashboard.	Success
2	GitLab connectivity	GitLab should connect successfully with Jenkins.	GitLab is connected successfully with Jenkins.	Success
3	SonarQube connectivity	SonarQube should connect successfully with Jenkins.	SonarQube is connected successfully with Jenkins.	Success
4	Ansible server connectivity	Ansible should connect successfully with Jenkins.	Ansible is connected successfully with Jenkins.	Success
5	Credentials	Check if the credentials of all the modules are added in the Jenkins "Manage Credentials".	Credentials of all the modules are added in the Jenkins "Manage Credentials".	Success

Sonarqube Test Cases:

Sr no.	Module	Expected Result	Actual Result	Status
1	Login	Users should be able to login into SonarQube Dashboard.	Users are able to login into SonarQube Dashboard.	Success
2	Jenkins Sonar Scanner connectivity	Jenkins Sonar Scanners should connect successfully with SonarQube Server.	Jenkins Sonar Scanner is connected successfully with SonarQube Server.	Success



3	Generate security report	SonarQube should generate Graphs, Issues and Vulnerabilities reports.	SonarQube generates Graphs, Issues and Vulnerabilities reports.	Success
---	--------------------------	-----------------------------------------------------------------------	-----------------------------------------------------------------	---------

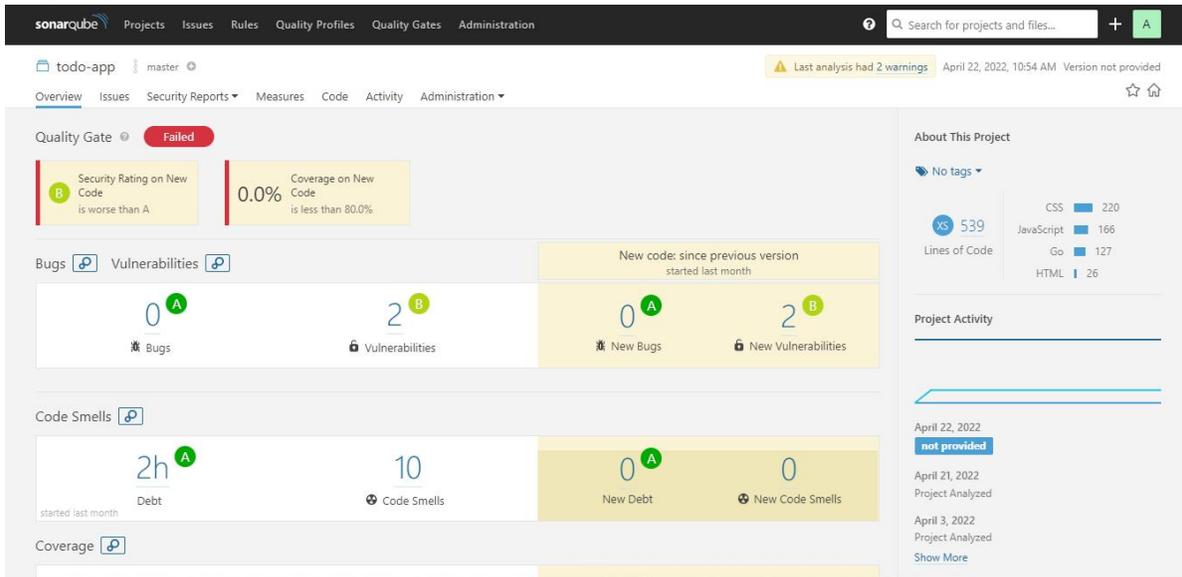
Harbor Test Cases:

Sr no.	Module	Expected Result	Actual Result	Status
1	Login	Users should be able to login into Harbor Dashboard.	Users are able to login into Harbor Dashboard.	Success
2	Image Stored	Pushed images from Jenkins should be stored.	Pushed image from Jenkins is getting stored.	Success
3	Pull Image	Image should get pulled while deploying the application.	Image is getting pulled while deploying the application.	Success
4	Update Image	If any code changes, the new image should update the previous version image.	Image is getting updated when there is new code with the new version.	Success

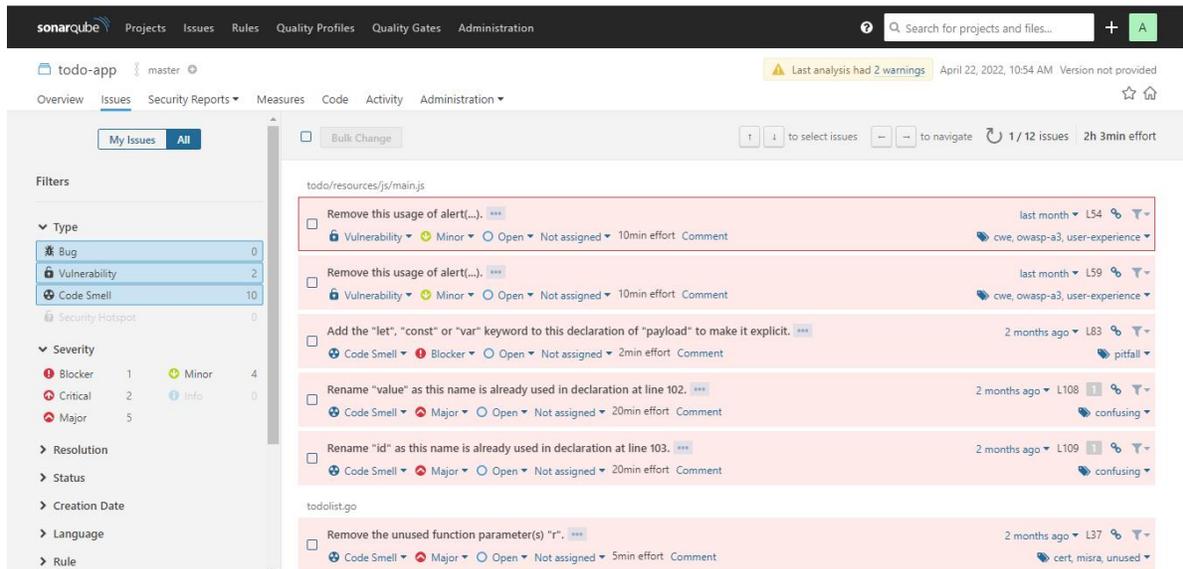
VIII. RESULT

8.1 Screenshots

SonarQube:



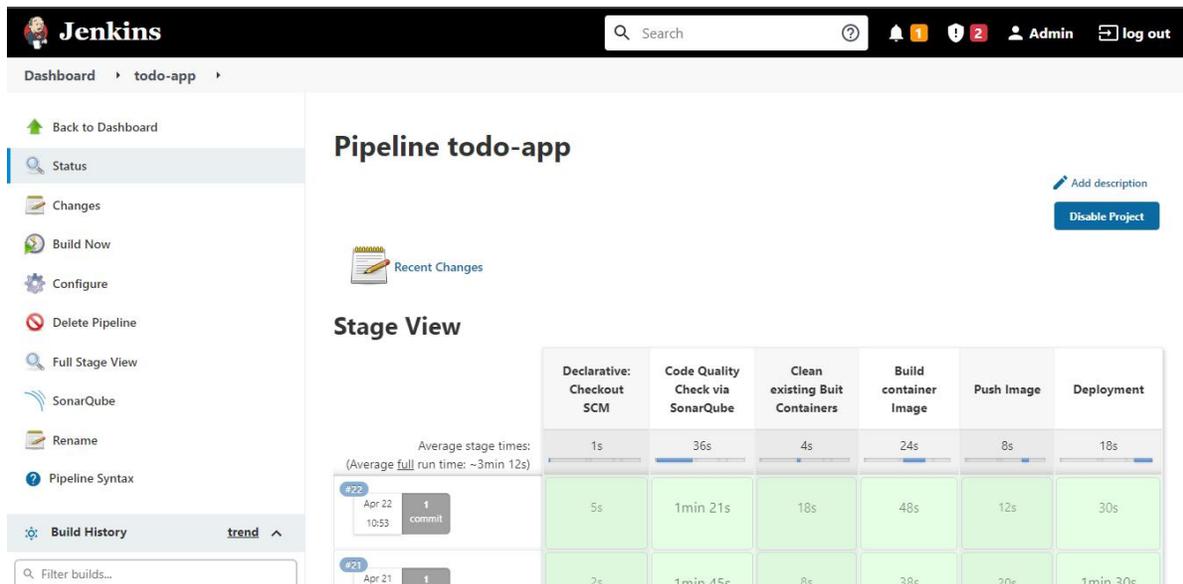
Project Report



The screenshot shows the SonarQube interface for a project named 'todo-app'. The left sidebar contains filters for Type (Bug, Vulnerability, Code Smell, Security Hotspot) and Severity (Blocker, Critical, Major, Minor, Info). The main area displays a list of issues, including vulnerabilities and code smells, with details such as severity, effort, and resolution status.

Project Vulnerabilities

Jenkins:

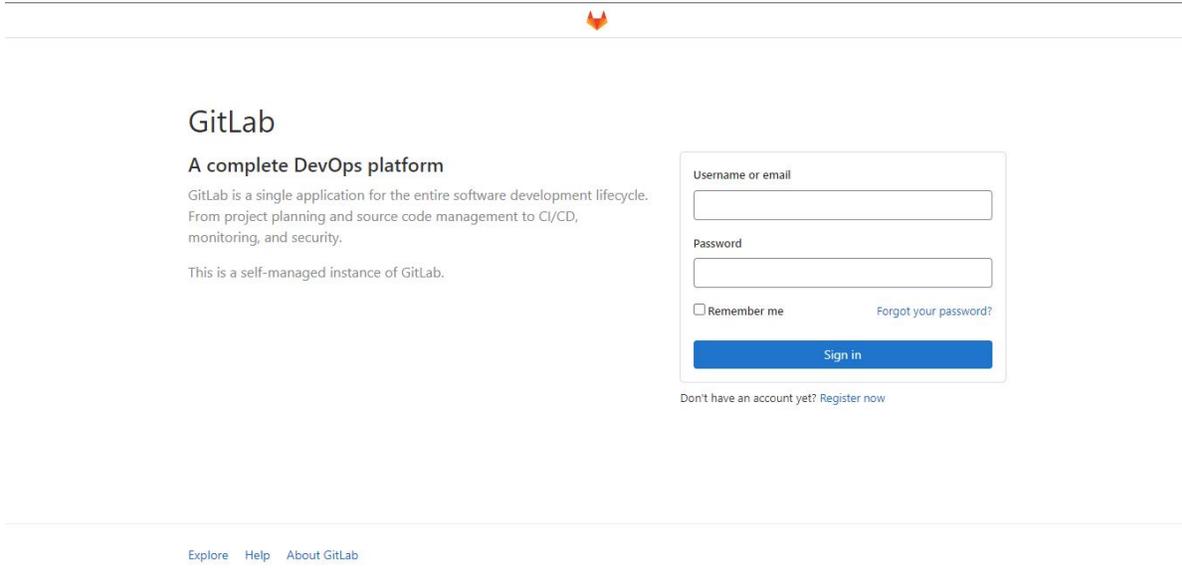


The screenshot shows the Jenkins Pipeline View for a project named 'todo-app'. The pipeline consists of several stages: Declarative: Checkout SCM, Code Quality Check via SonarQube, Clean existing Built Containers, Build container Image, Push Image, and Deployment. The Stage View shows the execution times for each stage across multiple builds.

Stage	Build #22 (Apr 22 10:53)	Build #21 (Apr 21)
Declarative: Checkout SCM	1s	2s
Code Quality Check via SonarQube	36s	1min 45s
Clean existing Built Containers	4s	8s
Build container Image	24s	38s
Push Image	8s	20s
Deployment	18s	1min 30s

Jenkins pipeline

Gitlab:



**GitLab**  
A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

Username or email

Password

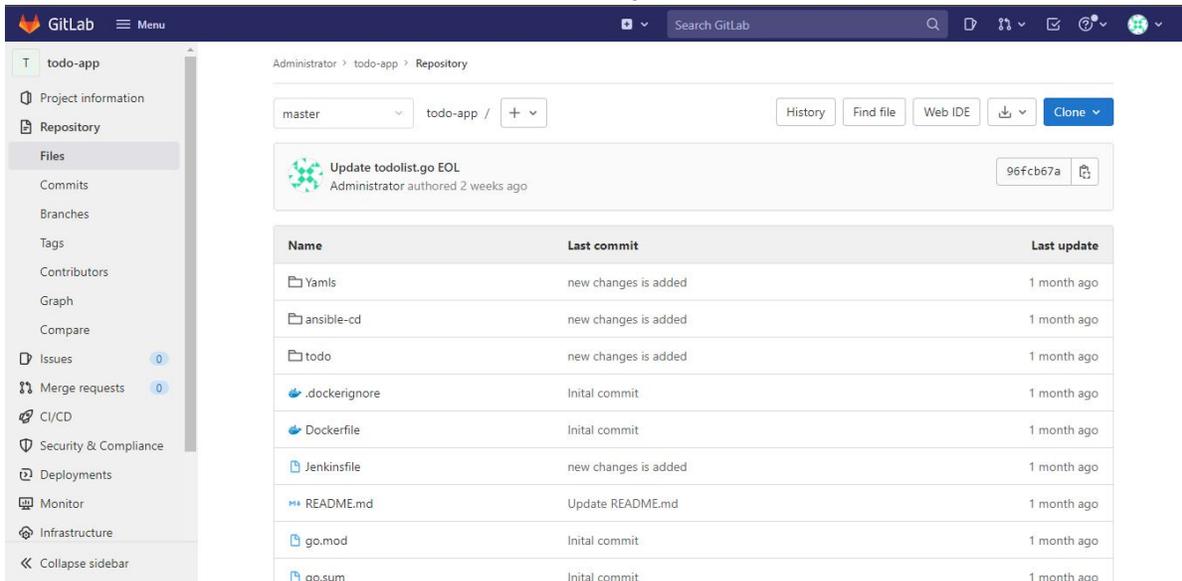
Remember me      [Forgot your password?](#)

**Sign in**

[Don't have an account yet? Register now](#)

[Explore](#) [Help](#) [About GitLab](#)

Gitlab Login



Administrator > todo-app > Repository

master    todo-app /    +    [History](#)    [Find file](#)    [Web IDE](#)    [Download](#)    [Clone](#)

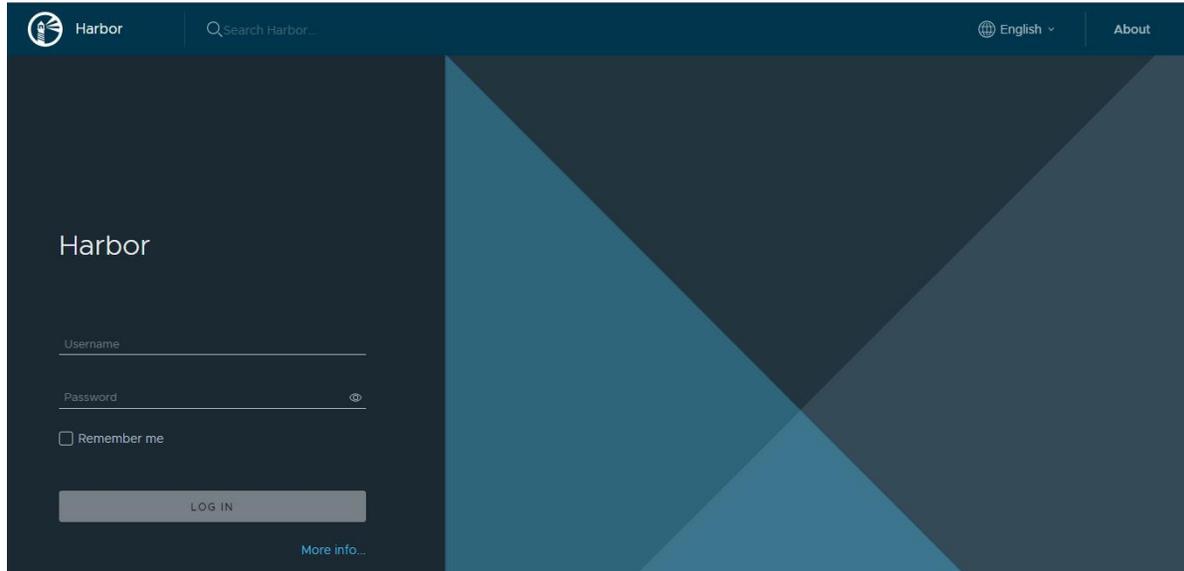
Update todoist.go EOL  
Administrator authored 2 weeks ago    96fcb67a

Name	Last commit	Last update
Yamls	new changes is added	1 month ago
ansible-cd	new changes is added	1 month ago
todo	new changes is added	1 month ago
.dockerignore	Initial commit	1 month ago
Dockerfile	Initial commit	1 month ago
Jenkinsfile	new changes is added	1 month ago
README.md	Update README.md	1 month ago
go.mod	Initial commit	1 month ago
go.sum	Initial commit	1 month ago

Gitlab Project with Pipeline setup



Harbor:



Harbor Login

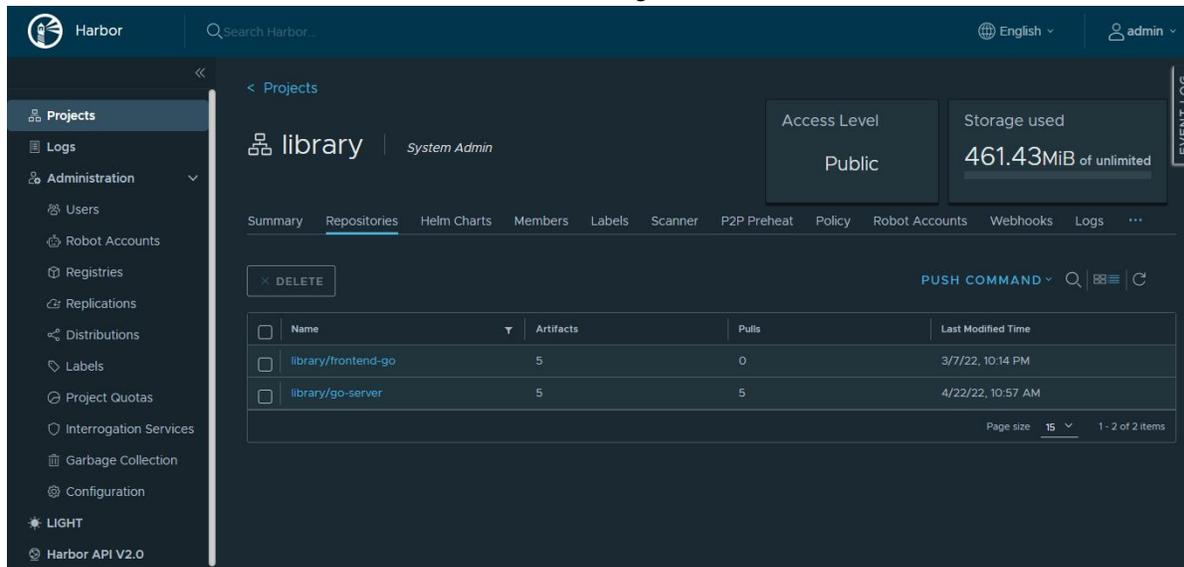


Image and its Details

### IX. CONCLUSION

DevSecOps implementation is a non-trivial event. Multiple integrated tools, processes, and policies must be aligned. Leveraging a consistent reference architecture will ensure that all aspects of the automation work closely together. In this post, I have detailed one such architecture. Thus, its deployment should be down to infractions and broken-down infractions, giving full attention to each step. We also remember that detecting vulnerabilities is just half of the job, and empowering developers can quickly fix the detected issues. DevSecOps is a new approach to security, and tools aimed explicitly should be widely adopted. Adopting DevSecOps principles in our continuous pipeline will lower the risk of security vulnerabilities, resulting in increased consumer trust towards the organization. We have introduced key DevSecOps concepts, described the DevSecOps Ecosystem, including the Software Factory and the Sidecar Container Security Stack, and indicated how the ecosystem should be set up and used.

### 9.1 Future Work

#### 1. DevSecOps in the Security Field:

The field of security is peculiar because the more you automate, the higher chances of automating problems too. So, all automation being done in this area to be intrinsically controlled, and this brings enormous scope for DevSecOps philosophy.

#### 2. AI/ML in the DevSecOps Framework:

The software development life cycle is revolutionized with the DevOps methodology, cloud-native approach, and micro services architecture. DevSecOps integrates testing and production environments, and developers get to see the problems before applications go live.

#### 3. Automation for Every Company:

In today's world, everything happens over the internet. Most companies are changing to be like an IT company that provides some particular services. For example, booking.com was a travel company which now functions as an IT company that provides travel services.

#### 4. Container Technology:

Container technology is evolving and emerging faster than before. Containers can be used in various ways to provide different benefits. Containers can be used to sandbox applications for security and resource constraints. Research is going around using containers per user or user session.

### 9.2 Applications

- **Detect Bugs:** Detecting tricky bugs or can raise issues on the pieces of code means that sometimes bugs that the coder can't understand early can be identified using sonarqube.
- **Security vulnerability:** Detect security issues that code may face if a developer forgets to close and open SQL database or if important details like username and passwords have been directly written in the code then sooner you can identify these things.
- **Integrated GitHub:** It can directly integrate your choice of version control software. For example, in our project we are using GitHub.

### REFERENCES

- [1]. J. Humble, and D. Farley, Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, 1st ed. Reading, MA, USA: Addison-Wesley, 2010.
- [2]. M. Fowler, Continuous Integration, accessed on Oct. 21, 2015. [Online]. Available: <http://martinfowler.com/articles/continuousIntegration.html>
- [3]. A. Phillips, M. Sens, A. de Jonge, and M. van Holsteijn, The IT Manager's Guide to Continuous Delivery: Delivering Business Value in Hours, XebiaLabs, Hilversum, The Netherlands, 2015.
- [4]. J. P. Reed. The Business Case for Continuous Delivery, accessed on Jul. 12, 2016. [Online]. Available: <https://www.atlassian.com/continuousdelivery/business-case-for-continuous-delivery>.
- [5]. J. Humble. Continuous Delivery vs Continuous Deployment, accessed on Mar. 1, 2016. [Online]. Available: <https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>
- [6]. P. Hammant, "Legacy application Strangulation: Case Studies," 14 July 2013. [Online]. Available: <https://paulhammant.com/2013/07/14/legacy-application-strangulation-casestudies/>. [Accessed 12 July 2019].
- [7]. N. M. Chaillan, "DOD Enterprise DevSecOps Initiative Hardening Containers," DRAFT, 2019.
- [8]. NIST, "Security and Privacy Controls for Federal Information Systems and Organizations," NIST SP 800-53 Revision 4, 2013.
- [9]. Y. Sundman. (2013). Continuous Delivery vs Continuous Deployment, accessed on Aug. 1, 2016. [Online]. Available: <http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>.