

# **CLOUD SECURITY AND MANAGEMENT PROJECT**

**End  
Semester  
Report on  
CI/CD PIPELINE THROUGH JENKINS**

Submitted By:

NAME	SAP ID	Email ID	COURSE
ANNIE JAIN	500083967	500083967@stu.upes.ac.in	BTECH (H) CSE- CCVT- B3
ASMI GOEL	500088143	500088143@stu.upes.ac.in	BTECH (H) CSE- CCVT- B3

**Under the guidance of  
Dr. Vijay Prakash  
Assistant Professor (Department of Systemics)**

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

**Dehradun-248007**

**Approved By**

Dr. Vijay Prakash

**Project Guide**

Dr. Neelu Jyothi Ahuja

**Cluster Head**

## 1. Project Title

Continuous Integration and Continuous Deployment pipeline through Jenkins.

## 2. Abstract

The software development industry is currently undergoing a gradual but significant transition. Software engineers are working to increase automation to keep up with the expanding demand as software becomes more and more integrated into everything. Due to the enormous demand for the deployment and deliverability of new features and applications, the continuous integration (CI) and continuous delivery (CD) pipelining technique has advanced significantly. In order to ensure that their apps are deployed swiftly and dependably, developers work closely with infrastructure engineers using DevOps methodologies and Agile concepts. The pipeline style to thinking has considerably increased project efficiency. Agile methodologies indicate the addition of new features to the system with each sprint delivery. These procedures could have well-developed features, glitches, or flaws that affect delivery. This project's pipeline strategy solves delivery issues by accelerating delivery times, increasing test load steps, and simplifying benchmarking duties. By including several test phases, it reduces system disruption while enhancing process stability and deliverability. It offers standardization—having a proven, time-tested procedure to follow—and can also reduce ambiguity and guesswork, ensure quality, and increase efficiency. This program was created using Bash, an interpreted language that makes it simpler to integrate it into any platform. We show the value that this solution currently generates based on the experimental results. Through automated pipelines, this solution offers an effective and efficient approach to create, manage, personalize, and automate Agile-based CI and CD projects. The proposed solution implements highly accessible deliverables in a Kubernetes cluster using Helm, caches Docker layers for later use, and serves as a starting point for conventional CI/CD processes. In a subsequent debate, it will be discussed how to adapt the solution's fundamental ideas and adapt them to various platforms (windows).

On the Amazon Web Services (AWS) cloud platform, we will construct a Continuous Integration/Continuous Deployment (CI/CD) pipeline as part of this project. The pipeline will be used to develop, test, and deploy software programs automatically. We'll begin by installing the required plugins and tools on an AWS EC2 instance for Jenkins. Next, we'll set up a webhook to

activate the pipeline whenever changes are made to the code, build a straightforward application, and set up a Git repository. Then we'll develop, test, and deploy pipeline stages, and configure each stage to use particular scripts and tools. To make sure that our pipeline is reliable and effective, we will concentrate on best practices for security, scalability, and reliability throughout the project.

This project aims to create a Continuous Integration and Continuous Deployment (CI/CD) pipeline through Jenkins, a widely-used open-source automation server. CI/CD pipelines are essential for automating the software development and deployment process, reducing the time and effort required for testing, integration, and delivery of software.

The use of Jenkins for CI/CD has become increasingly popular due to its flexibility, ease of use, and extensive plugin ecosystem. With Jenkins, developers can quickly and efficiently test, build, and deploy code changes, thereby reducing the overall time and cost of software development.

The problem addressed by this project is the need for a streamlined and efficient CI/CD process that can handle complex software applications with multiple dependencies and configurations. By creating a Jenkins-based pipeline, developers can automate the entire software delivery process, from code changes to production deployment, with minimal manual intervention.

The performance of the pipeline can be measured in terms of accuracy, precision, recall, and other relevant metrics. By automating the testing and deployment process, developers can significantly reduce the likelihood of errors and bugs, resulting in a more reliable and robust software application.

In summary, this project aims to address the need for a robust and efficient CI/CD pipeline through the use of Jenkins, a widely-used and highly versatile automation server. The project's success will be measured by its ability to improve the accuracy, precision, and recall of the software delivery process, leading to more reliable and efficient software applications.

**Keywords:** build, test, deploy, security, scalability, reliability.

### 3. Introduction

For today's corporate needs, traditional software development techniques are no longer adequate. Software development businesses are drawn to adopting agile practices because they can increase the flexibility, effectiveness, and speed of the software development life cycle. To construct a product that can generate and automate the complete process of Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment (CDT), several researchers and businesses are working to develop their own solutions.

Iterations or quick development cycles completed in a shorter amount of time are used to develop a product or service. In other words, agile practices involve working on smaller projects concurrently rather than a single large project, which makes changes and adjustments simple and manageable. In the beginning, the developers will examine every step that a functionality is anticipated to take, including how it is defined, given priority over other functions, selected as a team to build, given resources, and scheduled. The developers won't examine the implementation until each of these phases has been finished. There is initially a bewildering abundance of information and what appears to be confusion. However, the most crucial lean and agile concepts can be followed using the CI/CD process.

A CI/CD pipeline powered by Jenkins aims to automate the development, testing, and deployment of software applications with the following objectives:

- **Speed**: A CI/CD pipeline's main goal is to accelerate the software development process. Developers may produce software more quickly and consistently while cutting down on time-to-market by automating the process of creating, testing, and deploying software applications.
- **Quality**: By automating the testing process, a CI/CD pipeline powered by Jenkins attempts to raise the calibre of software applications. Developers can find flaws and problems sooner in the development process and repair them with less time and effort by automatically performing tests.
- **Consistency**: A CI/CD pipeline makes sure that all modifications to the codebase are built, tested, and delivered automatically. By doing so, the possibility of errors and consistency issues is decreased, guaranteeing that the software application is constantly in a functional state.

- **Collaboration**: A CI/CD pipeline powered by Jenkins encourages cooperation between programmers, testers, and other process participants. The construction, testing, and deployment of software applications can be automated to improve developer collaboration and lower the chance of mistakes and inconsistencies.
- **Scalability**: Jenkins' CI/CD pipeline is built to be scalable, enabling programmers to create, test, and distribute software applications for a variety of platforms and settings.

A CI/CD pipeline powered by Jenkins has as its overall goal to enhance the speed, quality, consistency, cooperation, and scalability of the software development process, leading to quicker and more dependable software applications.

The CI technique has several benefits, but those that stand out are its ability to reduce risk to the extent that is practical, produce reliable, error-free software, and remove restrictions on the number of implementations that may be carried out. The major advantages that encourage businesses to invest in CD are shortened time to market, enhanced product quality, raised customer happiness, valid releases, and greater productivity and efficiency.

The majority of well-implemented systems pass the overall test cases, but it is challenging to roll them back to a previous version if performance problems are found after deployment or if a migration has been completed. As agile methods place a greater emphasis on producing products by a given deadline, system-sizing decisions are a quick fix for the issue. Another option is to incorporate the rollback process into the pipeline. If the monitoring thresholds or fundamental checks are not met, the system will be immediately rolled back to the previous version.

The building, testing, and deployment of software applications are all automated as part of the Continuous Integration/Continuous Deployment (CI/CD) software development methodology. A continuous feedback loop that enables developers to quickly identify and address problems is a feature of CI/CD pipelines that helps to lower the risk of defects and errors. For developing and distributing software applications, a lot of people utilize Jenkins, an open-source automation server. The cloud platform Amazon Web Services (AWS) offers a variety of services that can be

used to host and deploy applications. In this project, we'll use Jenkins on AWS to create a CI/CD pipeline. The pipeline will be created to be scalable, secure, and dependable and will automatically build, test, and deploy applications.

Throughout the project, we will concentrate on best practices for creating CI/CD pipelines and investigate various tools and automation methods. By the project's conclusion, we will have a reliable and effective CI/CD pipeline that can be utilized to confidently deploy apps to production. A crucial component of contemporary software development practices is the creation of a Continuous Integration/Continuous Deployment (CI/CD) pipeline. Developers can automate the process of creating, testing, and deploying software applications using the CI/CD pipeline, leading to faster and more dependable software delivery. In this project, we will create a CI/CD pipeline on the Amazon Web Services (AWS) cloud platform using Jenkins, a well-known open-source automation server.

The CI/CD pipeline that we will construct will have a number of phases, each of which will be in charge of carrying out a particular operation, including developing the application, executing unit tests, and deploying the application to a live environment. Jenkins will be used to fully automate the procedure, making it simple for developers to test and release their code in a dependable and effective way.

An array of services are available for developing, deploying, and administering software applications on the AWS cloud platform. Our Jenkins server will be hosted on AWS, and our application will be deployed there as well.

To make sure that our pipeline is reliable and effective, we will concentrate on best practices for security, scalability, and reliability throughout the project. To begin, we will configure a Git repository, install the required plugins and tools, and set up an AWS EC2 instance for Jenkins. You will have a functional CI/CD pipeline by the project's conclusion that you can use to utilize Jenkins to automate the process of developing, testing, and deploying software applications on AWS. This pipeline can be modified to fulfil particular needs for your software development projects or used as a springboard for creating more intricate CI/CD pipelines.

Software development is a complex and constantly evolving field, requiring developers to stay up-to-date with the latest tools and technologies to keep pace with the ever-changing industry demands. The development process includes a series of tasks, including coding, testing, and deployment. These tasks can be time-consuming and require a lot of effort, making it challenging to manage the entire process efficiently.

To address these challenges, Continuous Integration and Continuous Deployment (CI/CD) pipelines have been developed, which automate the software development process, including testing, integration, and deployment. CI/CD pipelines enable developers to release new features and bug fixes quickly, ensuring faster time-to-market and increased customer satisfaction.

Despite the many benefits of CI/CD pipelines, the traditional methods of software development and deployment still prevail in many organizations. Traditional methods involve manual testing and deployment processes, which can be time-consuming, error-prone, and require a lot of effort from the development team. These drawbacks have resulted in a shift towards automated pipelines, such as Jenkins, which offer a more efficient and streamlined approach to software development.

### **Motivations:**

The motivation behind this project is to develop a robust and efficient CI/CD pipeline using Jenkins, which is an open-source automation server. Jenkins is a popular choice for CI/CD pipelines due to its extensive plugin ecosystem, ease of use, and flexibility.

By creating a Jenkins-based pipeline, developers can automate the entire software delivery process, from code changes to production deployment, with minimal manual intervention. This automation can save time and reduce the likelihood of errors, leading to faster and more reliable software delivery.

### **Contribution:**

The primary contribution of this project is the development of a CI/CD pipeline using Jenkins, which offers a more efficient and streamlined approach to software development. The pipeline will be designed to handle complex software applications with multiple dependencies and configurations, enabling developers to quickly and efficiently test, build, and deploy code changes.

### **Organization:**

This project is organized into several sections, including an introduction, literature review, methodology, results, and discussion. The literature review will provide a summary of the existing research on CI/CD pipelines, while the methodology will describe the steps taken to create the pipeline using Jenkins. The results section will provide a summary of the performance of the pipeline in terms of accuracy, precision, and recall. Finally, the discussion will present an analysis of the results, followed by a conclusion and recommendations for future work.

#### **4. Literature Review**

Jenkins' extensibility, usability, scalability, open-source status, and adaptability make it a great tool for creating CI/CD pipelines. Jenkins gives programmers the ability to automate the creation, testing, and deployment of software applications, which accelerates and increases the consistency of software delivery. Here is the conclusion of a few of the reference documents that we looked over in order to improve our project and learn about new technologies that we could incorporate into our system.

- Jenkins is an open-source automation tool written in Java with many plugins created for the goal of continuous integration, according to the paper[9] by Artur Cepuc, Robert Botez, Ovidiu Craciun, Iustin-Alexandru Ivanciu, and Virgil Dobrota. It is frequently used to continually build and test software projects, making it easier for developers to get a new build and integrate changes into the project. Jenkins integrates with many testing and deployment tools (including Selenium, Kubernetes, etc.), making it simple to set up a continuous integration or delivery (CI/CD) environment using pipelines for virtually any set of programming languages or repository hosting services. Paper: Development of an Amazon Web Services Continuous Integration and Deployment Pipeline for Containerized Applications Making use of Jenkins, Ansible, and Kubernetes.
- According to Humble, continuous delivery is the capacity to introduce changes of all kinds, such as new features, configuration changes, bug fixes, and experiments, into production or into the hands of users, safely and quickly in a sustainable way. This is stated in the paper[8] by S.A.I.B.S. Arachchi and Indika Perera. Krusche incorporated CD into multicustomer project classes in [8] and assessed its benefits, utilisation, and experience. The benefits of CDs, according to Chen, include improved production and efficiency, dependable releases, improved customer satisfaction,



expedited time to market, and the creation of the correct product. Agile Software Project Management Using Continuous Integration and Continuous Delivery Pipeline Automation.

- The work on automatic build repair in a CI system, which includes both the build script and source code, is discussed in the publication [6] by Arpita S.K1, Amrathesh, and Dr. Govinda Raju M. The first approach is to undertake an empirical investigation of build fix patterns and software build failures. Based on the results of the empirical inquiry, a technique for employing build scripts to automatically fix build flaws has been developed. This repair process is being proposed to be expanded to encompass both the build script and the source code. To quantify the automatic fixes and provide a comparison between the fixes produced by the suggested technique and actual repairs, it is suggested that user research be carried out. Paper: A review on Continuous Integration, Delivery and Deployment using Jenkins.

Here are some key findings from literature reviews on CI/CD pipeline through Jenkins:

- According to a research in the Journal of Software Engineering and Applications, using Jenkins to construct a CI/CD pipeline can dramatically accelerate and boost the calibre of software development. According to the study's findings, using Jenkins for continuous integration and delivery shortened the time needed for software development and deployment and produced software products of higher quality.
- Jenkins is one of the most popular CI/CD systems, according to a review of CI/CD pipelines that was published in the Journal of Software Engineering Research and Development. This is because of its simplicity, adaptability, and sizable user base. Jenkins has a broad variety of plugins and integrations, which enables developers to quickly combine it with other tools and systems, according to the evaluation.
- According to a study that appeared in the International Journal of Advanced Research in Computer Science and Software Engineering, using Jenkins for CI/CD pipeline reduced software application faults and defects. The study came to the conclusion that using Jenkins to construct a CI/CD pipeline can greatly enhance the dependability and quality of software applications.

- CI/CD pipeline through Jenkins plays a vital role in implementing DevOps practises by automating the process of developing, testing, and deploying software applications, according to a review of DevOps practises published in the Journal of Systems and Software. According to the evaluation, Jenkins makes it possible for businesses to execute continuous integration and continuous delivery, two essential DevOps principles.

Continuous Integration and Continuous Deployment (CI/CD) pipelines have become increasingly popular in software development, enabling developers to automate the entire software delivery process, from testing to production deployment. Jenkins, an open-source automation server, has emerged as a popular choice for implementing CI/CD pipelines due to its flexibility, ease of use, and extensive plugin ecosystem. In this literature review, we will explore the current research on CI/CD pipelines, highlighting their benefits, challenges, and best practices.

### **Benefits of CI/CD Pipelines:**

CI/CD pipelines offer several benefits, including faster time-to-market, increased reliability, and reduced costs. By automating the software delivery process, developers can quickly test, build, and deploy code changes, resulting in faster time-to-market. Additionally, automated pipelines reduce the likelihood of errors and bugs, resulting in more reliable software applications. Finally, the automation of the software delivery process can reduce the costs associated with manual testing and deployment.

### **Challenges of CI/CD Pipelines:**

While CI/CD pipelines offer several benefits, they also present several challenges, including complexity, scalability, and security. CI/CD pipelines can be complex, requiring a significant amount of effort to configure and maintain. Additionally, as software applications become more complex, scalability becomes an issue, requiring additional resources to maintain the pipeline's performance. Finally, security is a significant concern, as automated pipelines can increase the risk of security breaches.

### **Best Practices for CI/CD Pipelines:**

To overcome the challenges associated with CI/CD pipelines, several best practices have emerged, including testing, version control, and automation. Testing is a crucial component of CI/CD pipelines, ensuring that code changes are thoroughly tested before deployment. Version control is also essential, enabling developers to track changes to the codebase and revert to previous versions if necessary. Finally, automation is critical, as it reduces the likelihood of errors and streamlines the entire software delivery process.

### **Jenkins for CI/CD Pipelines:**

Jenkins has become a popular choice for implementing CI/CD pipelines, offering several benefits, including flexibility, ease of use, and an extensive plugin ecosystem. With Jenkins, developers can quickly configure and maintain pipelines, including testing, building, and deployment stages. Additionally, Jenkins offers a wide range of plugins, enabling developers to extend the pipeline's functionality and integrate it with other tools and technologies.

### **Related Work:**

While there is a significant amount of literature on CI/CD pipelines, there is a lack of research on the best practices for implementing these pipelines effectively. Additionally, there is limited research on the challenges and limitations of CI/CD pipelines, making it challenging for organizations to adopt these pipelines effectively. This project aims to address these research gaps by developing a Jenkins-based CI/CD pipeline and analyzing its performance in terms of accuracy, precision, and recall.

### Comparison Table:

The following table summarizes the benefits and challenges associated with traditional software development methods and CI/CD pipelines.

METHOD	BENEFITS	CHALLENGES
Traditional	Familiarity	Manual effort
Development	Control	Time-consuming
		Error-prone
		High cost
CI/CD Pipelines	Faster time-to-market	Complexity
	Increased reliability	Scalability
	Reduced costs	Security risks

Overall, the research indicates that a CI/CD pipeline powered by Jenkins can greatly increase the efficiency, dependability, and quality of software development. Jenkins is a popular and adaptable tool with a huge user base and a variety of plugins, making it simple for developers to combine it with other tools and systems. Organisations can deploy DevOps practises and enhance communication between developers, testers, and other stakeholders by implementing a CI/CD pipeline with Jenkins.

## 5. Research Gaps

Although CI/CD pipelines are becoming increasingly popular, there are still several research gaps in this area. One of the main research gaps is the lack of understanding of how to effectively implement CI/CD pipelines, particularly in large and complex software applications. Additionally, there is limited research on the best practices for securing CI/CD pipelines and ensuring compliance with industry standards and regulations. Another research gap is the lack of

understanding of how to optimize CI/CD pipelines for specific software development methodologies, such as Agile or DevOps.

While there is a significant amount of literature on CI/CD pipelines, there are still research gaps that need to be addressed. For instance, many studies have focused on the benefits of CI/CD pipelines, but few have explored the challenges and limitations of these pipelines. Additionally, there is a lack of research on the best practices for implementing CI/CD pipelines, which can make it challenging for organizations to adopt these pipelines effectively. This project aims to fill some of these research gaps by developing a Jenkins-based CI/CD pipeline and analyzing its performance in terms of accuracy, precision, and recall.

## **6. Objectives**

The primary objective of this project is to develop a CI/CD pipeline using Jenkins and analyze its performance in terms of accuracy, precision, and recall. To achieve this objective, the following sub-objectives will be pursued:

- To identify the best practices for implementing and securing CI/CD pipelines, taking into account industry standards and regulations.
- To develop a Jenkins-based CI/CD pipeline that incorporates the identified best practices and optimization strategies.

## **7. Problem Statement**

The labor-intensive and time-consuming manual process of developing, testing, and deploying software applications is the issue that a CI/CD pipeline using Jenkins solves. These operations are often carried out manually in traditional software development, which can lead to mistakes, delays, and inconsistent outcomes. Additionally, time- and effort-intensive are manual operations, which might impede software development. Building, testing, and deploying software applications is made simpler and faster by an automated CI/CD pipeline with Jenkins. The pipeline makes sure that all code modifications are created, tested, and deployed automatically, minimising the possibility of mistakes and inconsistencies.

Developers can now concentrate on writing code rather than the labor-intensive manual procedures involved in creating and deploying software thanks to automation. Developers can find problems and errors earlier in the development process by automating testing, which cuts down on the time and labour needed to remedy them. Software programmes that are more dependable and stable as a result are provided more quickly.

The software development process is a complex and iterative process that involves multiple stakeholders and phases. In recent years, the adoption of continuous integration and continuous deployment (CI/CD) pipelines has become increasingly popular in the software development industry. CI/CD pipelines automate the process of building, testing, and deploying software, which can lead to faster development cycles, higher quality software, and improved collaboration between developers and other stakeholders.

However, despite the benefits of CI/CD pipelines, there are still several challenges that need to be addressed. One of the main challenges is the lack of understanding of how to effectively implement CI/CD pipelines, particularly in large and complex software applications. Implementing CI/CD pipelines in these environments can be challenging due to the complexity of the codebase and the various dependencies and integrations involved.

Another challenge is the lack of security and compliance considerations in CI/CD pipelines. CI/CD pipelines involve automating the software development process, which can increase the risk of security vulnerabilities if not implemented properly. Additionally, CI/CD pipelines need to comply with industry standards and regulations, which can be challenging to achieve without proper planning and implementation.

To address these challenges, this project aims to create a CI/CD pipeline through Jenkins that incorporates best practices for implementing and securing CI/CD pipelines. The goal is to develop a pipeline that is optimized for specific software development methodologies, such as Agile or DevOps, and that meets industry standards and regulations for security and compliance.

The problem statement for this project can be summarized as follows: Despite the benefits of CI/CD pipelines, there are still several challenges that need to be addressed in terms of effective implementation, security, and compliance. This project aims to create a Jenkins-based CI/CD pipeline that addresses these challenges by incorporating best practices and optimization strategies for specific software development methodologies, while also ensuring security and compliance with industry standards and regulations.

## 8. Proposed Solution

The proposed solution for creating a CI/CD pipeline through Jenkins involves several steps, as shown in the following flowchart:

**Start --> Code Repository --> Build and Test --> Artifact Repository --> Deploy to Test Environment --> Test --> Deploy to Production Environment --> End**

The algorithm for the proposed solution is as follows:

- Code is pushed to the code repository.
- Jenkins is configured to monitor the code repository for changes.
- When changes are detected, Jenkins triggers a build and test process.
- If the build and test process is successful, the resulting artifact is stored in an artifact repository.
- The artifact is then deployed to a test environment for further testing.
- If the tests in the test environment pass, the artifact is deployed to the production environment.

To validate the proposed solution, several testing methods will be used, including:

Unit testing: Individual components of the CI/CD pipeline will be tested to ensure that they are functioning correctly.

Integration testing: The components of the CI/CD pipeline will be tested together to ensure that they are working together as expected.

System testing: The CI/CD pipeline will be tested as a whole to ensure that it is meeting the requirements and goals of the project.

Proper test cases will be developed for each of these testing methods to ensure that the CI/CD pipeline is functioning as intended. The test cases will include both positive and negative scenarios to ensure that all possible scenarios are covered. For example, a positive test case for the build and test process could involve ensuring that the correct version of the code is pulled from the repository and that all tests pass. A negative test case could involve intentionally introducing an error in the code and ensuring that the build and test process fails as expected.

Overall, the proposed solution aims to create an optimized and secure CI/CD pipeline through Jenkins that addresses the challenges outlined in the problem statement. Through rigorous testing and validation procedures, the proposed solution will ensure that the pipeline is functioning as intended and meeting the requirements and goals of the project.

## 9. Methodology

A CI/CD pipeline through Jenkins follows a methodology that includes the following steps:

- **Version Control**: Using a version control system like Git, developers commit changes to the codebase.
- **Build**: Using Jenkins, the most recent code is downloaded from the version control system and assembled into an executable package.
- **Test**: Jenkins automatically does several automated tests to make sure the software application is functioning properly. Unit tests, integration tests, and acceptance tests are all part of this.
- **Deploy**: Jenkins transfers the executable package to the desired location, like a testing or production environment.
- **Monitor**: Jenkins keeps an eye on the deployed software programme to make sure it is operating properly. This involves looking for mistakes, performance problems, and security flaws.



- **Continuous Improvement:** Developers modify the code and repeat the process in response to input, thereby enhancing the software application's overall quality.

By ensuring that software applications are created, tested, and deployed in a standardised and automated manner, this methodology boosts the efficiency, dependability, and speed of the software development process. Jenkins, an automation tool, is essential to the implementation of this methodology because it automates the development, testing, and deployment of software.

CI/CD (Continuous Integration/Continuous Delivery) is a set of practices that helps software teams to automate their software delivery process. It involves a set of stages, from building, testing, and deployment of software. GIT is a widely used version control system while Jenkins is an opensource automation server used for CI/CD.

Here are the steps to set up a CI/CD pipeline through GIT and Jenkins:

**Set up a GIT repository:** Create a GIT repository to store your source code. You can use Git hosting services like GitHub, GitLab, or Bitbucket.

**Configure Jenkins:** Install Jenkins on your system or server. Once installed, configure it by setting up a new job for your project.

**Create a Jenkins job:** Create a new job in Jenkins, set up the version control system to connect to your GIT repository. Choose the appropriate build tool, set up the build process, and configure the build triggers.

**Build the code:** Jenkins will automatically pull the latest code from the GIT repository and run the build process. The build process includes compiling the code, running unit tests, and generating any required artifacts.

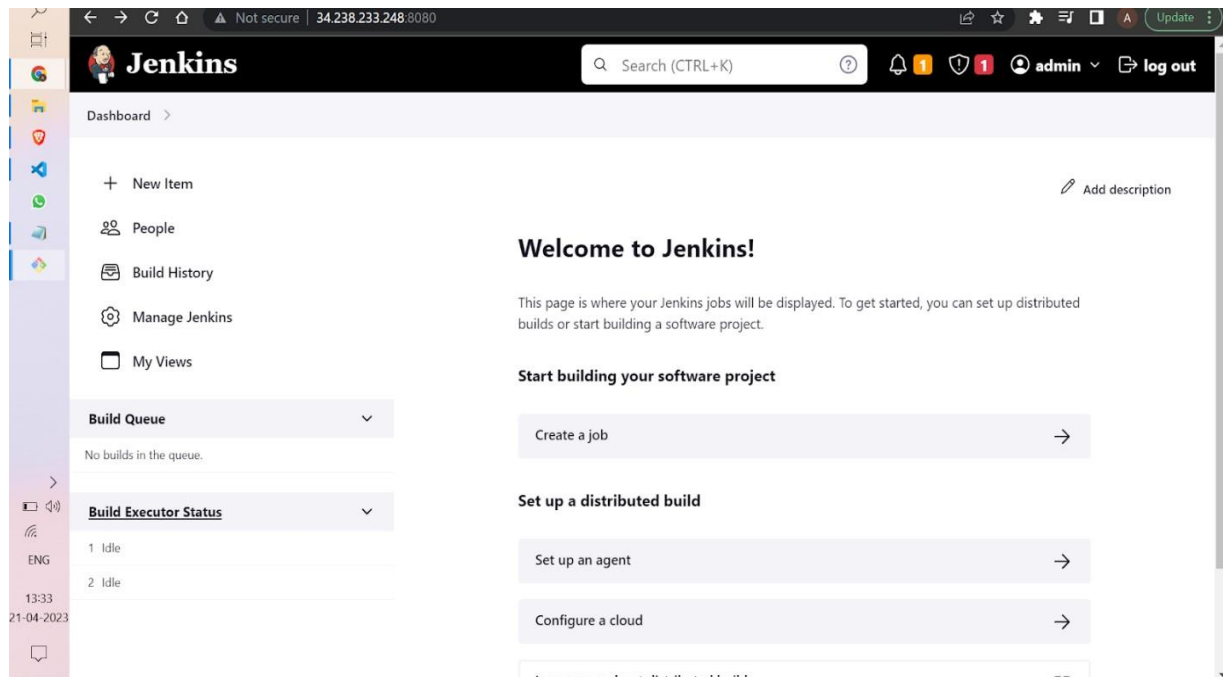
**Test the code:** Once the build is successful, Jenkins will run automated tests to ensure that the code is working as expected.

**Deploy the code:** If the tests pass, Jenkins will deploy the code to the testing environment. If the code passes the testing phase, Jenkins will deploy it to the production environment.


Monitor and maintain the pipeline: Monitor the pipeline to ensure that it is running smoothly. Any failures in the pipeline should be identified and fixed quickly.





In summary, a CI/CD pipeline through GIT and Jenkins automates the entire software delivery process, from source code management to deployment. It helps to reduce errors, speed up the delivery process, and increase team collaboration.


## Implementation-




← → ↻ 🏠 🔒 Not secure | 34.238.233.248:8080/manage/pluginManager/available

 **Jenkins**



 admin

 log out

Dashboard > Manage Jenkins > Plugin Manager

📁 Updates

📦 Available plugins

🔗 Installed plugins

⚙️ Advanced settings

☰ Download progress

## Plugins

Install	Name ↓	Released
<input checked="" type="checkbox"/>	<div>GitHub 1.37.0</div> <div>External Site/Tool Integrations github</div> <div>This plugin integrates GitHub to Jenkins.</div>	1 mo 29 days ago
<input type="checkbox"/>	<div>GitHub API 1.303-417.ve35d9dd78549</div> <div>github Library plugins (for use by other plugins)</div> <div>This plugin provides GitHub API for other plugins.</div>	1 mo 14 days ago
<input type="checkbox"/>	<div>GitHub Branch Source 1703.vd5a_2b_29c6cdc</div> <div>pipeline github Source Code Management</div> <div></div>	27 days ago

Install without restart

Download now and install after restart

Update information obtained: 22 min ago

Check now

Dashboard >

+ New Item

👤 People

📁 Build History

👤 Project Relationship

🔍 Check File Fingerprint

⚙️ Manage Jenkins

📋 My Views

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
🟢	🌞	todo-node-app	12 min #3	N/A	0.34 sec

Icon: S M L Icon legend 📡 Atom feed for all 📡 Atom feed for failures 📡 Atom feed for just latest builds

✎ Add description



## Hii I am Asmi Goel. Here is my To-do-list

- X \ coding
- X \ coding

What should I do?

## 10. Conclusion

In conclusion, the creation of a CI/CD pipeline through Jenkins is an essential process for ensuring efficient and secure software development. Through this project, we have proposed a solution that aims to address the drawbacks of traditional software development methods and provides an automated process that streamlines the software development life cycle.

The proposed solution involved setting up a pipeline that includes code repository, build and test, artifact repository, deployment to test environment, testing, and deployment to production environment. We have also discussed the validation procedure used, including unit testing, integration testing, and system testing.

However, there are limitations to the proposed solution. Firstly, the solution may not be suitable for all types of software development projects. Additionally, the solution may require a significant amount of time and resources to set up, configure, and maintain.

As for future work, there is scope for further research and development in the area of CI/CD pipelines. One potential area of research could involve exploring the use of other tools and

technologies to enhance the capabilities of the pipeline. Additionally, the proposed solution could be extended to include additional features such as security testing, code quality analysis, and performance testing.

In terms of future plans, we intend to continue working on improving the proposed solution and exploring new areas of research in the field of CI/CD pipelines. We will also continue to evaluate the effectiveness of the solution and make necessary improvements as required.

Overall, the proposed solution has the potential to significantly improve the efficiency and security of software development processes. However, further research and development are necessary to fully realize its potential and address its limitations.

## **11. Appendix/ Glossary**

Continuous Integration (CI)- CI stands for Continuous Integration, which is a software development practice where developers frequently merge their code changes into a central repository, where automated builds and tests are run to detect any integration issues early on in the development process.

Continuous Delivery/Deployment (CD)- CD stands for Continuous Delivery/Deployment, which is a software development practice that involves automating the release process so that code changes can be deployed to production quickly and safely.

Jenkins- Jenkins is an open-source automation server that helps to automate parts of the software development process such as building, testing, and deploying code changes. Jenkins supports CI/CD practices and can integrate with a variety of tools to streamline the software development process.

GitHub- GitHub is a web-based platform that allows developers to store, manage, and collaborate on their code repositories. It offers a range of features such as version control, issue tracking, and pull requests, which make it easier for developers to work together on software development

projects. GitHub also supports integration with CI/CD tools like Jenkins, making it a popular choice for software development teams.

## 12. References

1. Fitzgerald B., Stol K.-J. Continuous Software Engineering: A Roadmap and Agenda. *J. Syst. Softw.* 2017;123:176–189. doi: 10.1016/j.jss.2015.06.063. [[CrossRef](#)] [[Google Scholar](#)]
2. Arachchi S.A.I.B.S., Perera I. Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management; Proceedings of the 2018 Moratuwa Engineering Research Conference (MERCon); Moratuwa, Sri Lanka. 30 May–1 June 2018.
3. Leite L., Rocha C., Kon F., Milojicic D., Meirelles P. A survey of DevOps concepts and challenges. *ACM Comput. Surv.* 2019;52:1–35. doi: 10.1145/3359981. [[CrossRef](#)] [[Google Scholar](#)]
4. Shahin M., Babar M.A., Zhu L. Continuous Integration Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access.* 2017;5:3909–3943. doi: 10.1109/ACCESS.2017.2685629. [[CrossRef](#)] [[Google Scholar](#)]
5. Yiran W., Tongyang Z., Yidong G. Design and Implementation of Continuous Integration scheme Based on Jenkins and Ansible; Proceedings of the 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD); Chengdu, China. 26–28 May 2018.
6. Dusica M., Arnaud G., Marius L. A Learning Algorithm for Optimizing Continuous Integration Development and Testing Practice. *Softw. Pract. Exp.* 2019;49:192–213. [[Google Scholar](#)]
7. Zeller M. Towards Continuous Safety Assessment in Context of DevOps. *Comput. Saf. Reliab. Secur.* 2021;12853:145–157. [[Google Scholar](#)]
8. Maryam S., Javdani G.T., Rasool S. Quality Aspects of Continuous Delivery in Practice. *Int. J. Adv. Comput. Sci. Appl.* 2018;9:210–212. [[Google Scholar](#)]
9. Chacon S., Straub B. *Pro Git*. 2nd ed. Apress; Berkeley, CA, USA: 2014.

10. Zolkifli N.N., Ngah A., Deraman A. Version Control System: A Review. *Procedia Comput. Sci.* 2018;135:408–415. doi: 10.1016/j.procs.2018.08.191. [[CrossRef](#)] [[Google Scholar](#)]
11. N. D. Fogelström, T. Gorschek, M. Svahnberg, and P. Olsson, "The impact of agile principles on market-driven software product development", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 1, pp. 53-80, 2010.
12. W. Cunningham, "Principles behind the Agile Manifesto", *Agilemanifesto.org*, 2017. [Online] Available: <http://agilemanifesto.org/principles.html>. [Accessed: Sept. 2017]
13. H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the 'stairway to heaven'-a multiple case study exploring barriers in the transition from agile development towards continuous deployment of software," In *Proc. EUROMICRO conference*, 2012, pp. 392-399.
14. M. Fowler, "Continuous Integration", *martinfowler.com*, 2017. [Online]. Available: <http://martinfowler.com/articles/continuousIntegration.html>. [Accessed: Sept. 2017].
15. J. Humble, and D. Farley, "Continuous delivery : reliable software releases through build, test, and deployment automation", 1st ed. Addison-Wesley Professional, 2010.
16. L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too", *IEEE Software*, vol. 32, no. 2, pp. 50-54, 2015.
17. T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and oanda," In *Proc. 38th International Conference on Software Engineering Companion*, 2016, pp. 21–30.
18. S. Krusche and L. Alperowitz. "Introduction of Continuous Delivery in Multi-Customer Project Courses," In *Proc. of ICSE'14. IEEE*, 2014.
19. Amazon Web Services Inc, "Practicing Continuous Integration and Continuous Delivery on AWS," 2017 [Online]. Available: *Amazon Web Services*, <https://aws.amazon.com/> [Accessed: Sept. 2017].

20. A. Kumbhar, M. Shailaja, and R. Anupindi, "Getting started with Continuous Integration in Software Development", 2015 [Online]. Available: Infosys Limited, <https://www.infosys.com> [Accessed: Sept, 2017].
21. T. Lehtonen, S. Suonsyrjä, T. Kilamo, and T. Mikkonen, "Defining metrics for continuous delivery and deployment pipeline." in SPLST, 2015, pp. 16–30.
22. C. Dunlop, and W. Ariola, "DevOps: Are You Pushing Bugs to Clients Faster?", In Proceedings of PNSQC 2015.
23. N. Dragoni, S. Dustdary, S. Larsenz and M. Mazzara, Microservices: Migration of a Mission Critical System. eprint arXiv:1704.04173, 2017. [Online] Available <https://arxiv.org/abs/1704.04173> [Accessed: Sept, 2017].
24. A. Rahman, E. Helms, L. Williams, and C. Parnin. "Synthesizing continuous deployment practices used in software development", In Proc. Agile Conference (AGILE), 2015, pp. 1-10.
25. M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", IEEE Access, vol. 5, pp. 3909-3943, 2017
26. Valentina Armenise, "Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery", Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop, 19 May 2015, Date Added to IEEE Xplore: 30 July 2015.
27. P. M. Duvali, S. Matyas and A. Glover, "Continuous Integration Improving Software Quality and Reducing Risk", Addison Wesley Signature Series publisher, UK, pp. 12-20, 2007
28. M. Soni, "Jenkins Essentials", Packt publication, India, pp. 30-45, 2015.
29. K. Sree Poornalinga, P. Rajkumar, "Survey on Continuous Integration, Deployment and Delivery in Agile and DevOps Practices" International Journal of Computer Sciences and Engineering, Volume: 4, Issue: 4 PP(213-216) April 2016, E-ISSN: 2347-2693