CI/CD PIPELINE FOR WEB APPLICATION

A Project Work Synopsis

Submitted in the partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE WITH SPECIALIZATION DEVOPS.

Submitted by:

21CDO1033 – NIKHIL SINGH

21CDO1043 – DURGESH PATEL

21CDO1058 – ANKIT RAJ

21CDO1062 – SHUBHAM

Under the Supervision of:

Ms. SUKHMEET KAUR (E13420)

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413

PUNJAB

ABSTRACT

Keywords: Continuous Integration, Continuous Deployment, CI/CD Pipeline, Web Applications, Automated Testing, Deployment Automation, Quality Assurance, Monitoring and Logging, Continuous Improvement

Modern organisation's rapid pace of software development necessitates teams delivering high-quality software products faster than ever before. In order to accomplish this, software development teams are incorporating continuous integration and continuous deployment (CI/CD) methodologies into their software development processes. CI/CD pipelines are a set of practises and tools that allow teams to efficiently and reliably automate the development, testing, and deployment of software products. CI/CD pipelines have become an essential tool for teams delivering web applications at a faster pace while ensuring scalability, security, and performance in the context of web applications.

This paper provides an overview of the best practises and tools for constructing a successful CI/CD pipeline for web applications. Version control, continuous integration, automated testing, deployment automation, monitoring, and logging are among the key steps covered in the paper. The paper also discusses the advantages and disadvantages of CI/CD pipelines, such as increased productivity, shorter time-to-market, fewer manual errors, and better collaboration between development and operations teams.

Several case studies are included in the paper to demonstrate the effectiveness of CI/CD pipelines in web application development. The case studies cover a variety of web applications, such as e-commerce websites, social media platforms, and healthcare apps. Each case study provides practical insights into CI/CD pipeline implementation, including the tools and technologies used, the benefits realised, and the challenges encountered. The case studies also emphasise the importance of culture and collaboration in CI/CD pipeline implementation success.

The paper also discusses the key tools and technologies used in web application CI/CD pipelines, such as Git, Jenkins, Docker, Kubernetes, and AWS. The paper provides an

overview of these tools as well as their role in various stages of the CI/CD pipeline. The paper also discusses the importance of security in CI/CD pipelines, as well as an overview of the key security practises that must be implemented.

In conclusion, this paper provides a thorough overview of the best practises and tools for developing an effective CI/CD pipeline for web applications. The paper discusses the key steps in creating a CI/CD pipeline, the benefits and challenges of CI/CD pipelines, and several case studies demonstrating the effectiveness of CI/CD pipelines in web application development. In addition, the paper discusses the key tools and technologies used in CI/CD pipelines, as well as the significance of security in CI/CD pipelines. Finally, the paper discusses the future directions of web application CI/CD pipelines and how they will evolve to meet the changing needs of software development teams.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Problem Definition

The CI/CD Pipeline (Continuous Integration/Continuous Deployment) methodology involves the automation of software delivery processes, from development to testing and deployment. This methodology aims to provide a dependable and efficient approach to software development, resulting in the delivery of high-quality software products in a shorter period of time. A CI/CD pipeline for web applications is designed to automate the entire software development lifecycle, from code changes to deployment, ensuring that code is consistently tested, built, and deployed.

The problem definition for a CI/CD pipeline for web applications includes the following elements:

- The first step in the CI/CD pipeline is to build the application and run unit and integration tests to ensure that the code is error-free and meets the acceptance criteria. The challenge is to automate this process in order to ensure consistency and repeatability.

- Code quality: Another critical aspect to consider when developing a CI/CD pipeline is code quality. The pipeline should include a mechanism for automatically checking code quality, such as code reviews and static analysis, to ensure that the code adheres to coding standards and best practises.

- Deployment: Once the code has been tested and validated, the application will be deployed to the production environment. The challenge is to automate and standardise the deployment process across multiple environments.

- Continuous monitoring of the application is a critical component of the CI/CD pipeline. When something goes wrong in the production environment, the pipeline should detect it and notify the team.

- Security: When developing web applications, security is a major concern. The CI/CD pipeline should include a mechanism for ensuring that the code is secure and free of vulnerabilities.

- Scalability: The CI/CD pipeline should be scalable to support the application's and team's growth. It should be adaptable enough to accommodate multiple developers working on the same codebase at the same time.

Overall, the challenge is to create a dependable, efficient, and automated CI/CD pipeline for web applications that reduces time-to-market, improves code quality, and ensures application security and scalability.

1.2 Problem Overview

A CI/CD pipeline (Continuous Integration/Continuous Deployment) is a set of automated steps designed to help web application developers deliver new code changes or updates to their applications more efficiently and with fewer errors. The following steps are typically included in the CI/CD pipeline for web applications:

Code Development entails writing, testing, and committing code changes to a shared repository.

Continuous Integration (CI): When code changes are committed to the repository, the CI system builds and tests the application to detect errors.

After the code changes pass the initial build and test stage, the pipeline runs a series of automated tests to ensure that the application behaves as expected.

Continuous Deployment: Once the code changes have been tested and approved, the pipeline deploys the new version of the application to a staging environment automatically.

User Acceptance Testing (UAT): In the staging environment, users or testers review the new version of the application to ensure it works properly.

Production Deployment: Once the new version of the application is approved, it is deployed to the production environment by the pipeline.

Monitoring and Feedback: The pipeline monitors the performance of the application and provides feedback to the development team to help them identify and resolve issues as quickly as possible.

The CI/CD pipeline can assist developers in delivering high-quality software more quickly and with fewer bugs and errors. By automating the testing and deployment process, the risk of human error is reduced, and the time it takes to get new features and updates into production is reduced.

1.3 Hardware Specification

The hardware specifications for a CI/CD pipeline for web applications can vary depending on the size and complexity of the application, the number of developers working on the project, and the frequency of code changes being pushed to the pipeline. However, here are some general hardware recommendations:

- CPU: At least a quad-core processor with a clock speed of 2.5 GHz or higher. A higher core count and clock speed will help the CI/CD pipeline to run builds and tests faster.

- RAM: At least 8 GB of RAM, but 16 GB or more is recommended for larger applications.

- Storage: At least 256 GB of solid-state drive (SSD) storage is recommended, as it can handle the read and write operations faster than traditional hard disk drives (HDD).

- Network: A reliable and high-speed internet connection is necessary to ensure fast data transfer and enable seamless communication between the different components of the pipeline.

- Virtualization: If the CI/CD pipeline uses virtualization technologies such as containers or virtual machines, the hardware specifications should be increased accordingly to ensure sufficient resources are available.

- Scalability: The CI/CD pipeline should be designed to scale horizontally by adding more nodes to the cluster or vertically by adding more resources to the existing nodes. This will ensure that the pipeline can handle increased workloads as the application grows.

Overall, the hardware specifications for a CI/CD pipeline should be carefully evaluated and scaled based on the specific needs of the web application being developed.

1.4 Software Specification

The software specifications for a CI/CD pipeline for web applications can also vary depending on the application's specific needs and the tools used. Here are some general software recommendations, however:

- Version Control System (VCS): To manage the codebase and track changes, the pipeline should integrate with a VCS such as Git, SVN, or Mercurial.
- Continuous Integration (CI) Server: To automate the build and test processes, the pipeline should use a CI server such as Jenkins, CircleCI, or Travis CI.
- Configuration Management: To automate the setup and configuration of the pipeline's various components, the pipeline should use configuration management tools such as Ansible, Chef, or Puppet.
- Containerization: Containerization tools like Docker or Kubernetes can be used to package the application and its dependencies into containers that can be easily deployed across multiple environments.
- Continuous Delivery/Deployment (CD) Tools: CD tools such as AWS CodeDeploy, Octopus Deploy, or Azure DevOps can be used to automate application deployment across multiple environments.
- Testing Frameworks: To automate application testing, the pipeline should include testing frameworks such as Selenium, JUnit, or pytest.
- Monitoring and logging tools such as New Relic, Splunk, or the ELK stack should be used in the pipeline to monitor application performance and provide feedback to the development team.
- Collaboration Tools: Communication between the various stakeholders involved in the development process can be facilitated using collaboration tools such as Slack, Microsoft Teams, or Trello.

Overall, the software specifications for a CI/CD pipeline should be carefully evaluated and chosen in accordance with the specific requirements of the web application being developed.

2. LITERATURE SURVEY

2.1 Existing System

Existing systems for implementing a CI/CD pipeline for web applications are numerous. Here are a couple of examples:

- Jenkins: Jenkins is an open-source CI/CD automation server for building, testing, and deploying web applications. It has a large plugin ecosystem and integrations with other tools.

- CircleCI: CircleCI is a cloud-based continuous integration and delivery platform that supports multiple programming languages and integrates with a variety of tools and services, including GitHub, Docker, and AWS.

- Travis CI is a hosted continuous integration and delivery platform that integrates with GitHub and can be used to build, test, and deploy web applications. It supports multiple programming languages and includes Docker support.

- GitLab CI/CD: GitLab CI/CD is an open-source Git-based platform for managing the software development process that is part of the GitLab platform. It has many features, such as source code management, CI/CD pipelines, and issue tracking.

- AWS CodePipeline is a fully managed CI/CD service for building, testing, and deploying web applications on AWS. It works with other AWS services like AWS CodeBuild, AWS CodeDeploy, and AWS CloudFormation.

- Microsoft Azure DevOps is a cloud-based platform that offers a variety of tools and services for managing the software development process, such as CI/CD pipelines, source control, and project management**.**

These existing systems can be customised and configured to meet the specific requirements of a web application, and they can be integrated with a variety of other tools and services to form a comprehensive CI/CD pipeline.

2.2 Proposed System

Typically, a proposed CI/CD pipeline for web applications would include several components that work together to automate the build, testing, and deployment processes. A high-level overview of a proposed system for a CI/CD pipeline for web applications is provided below:

- Version Control System (VCS): To manage the codebase and track changes, a version control system such as Git would be used. Git would be used by developers to commit changes to the codebase, and Git would be used by the pipeline to retrieve the most recent version of the code.

- Containerization: The application and its dependencies would be packaged into containers using Docker. The build process would produce a Docker image, which would be uploaded to a container registry like Docker Hub.

- Continuous Deployment (CD) Tools: A CD tool, such as AWS CodeDeploy, would be used to automate the application's deployment across multiple environments. The CD tool would be set up to retrieve the Docker image from the container registry and deploy it to a staging environment for additional testing. The CD tool would automatically promote the image to the production environment once the application passed all tests in the staging environment.

- Testing Frameworks: Testing frameworks like Selenium would be used to automate application testing. A suite of automated tests would be run during the build process and deployment to the staging environment as part of the pipeline. This ensures that any problems are identified and resolved before the application is deployed to production.

- Monitoring and logging tools, such as New Relic, would be used to monitor the application's performance and provide feedback to the development team. The monitoring tools would be set up to notify the team if any problems arose in production, allowing them to quickly identify and resolve problems.

Overall, the proposed CI/CD pipeline for web applications would streamline and automate the process of developing, testing, and deploying applications. It would reduce the likelihood of errors and accelerate development and deployment, resulting in a more efficient and effective development process.

2.3 LITERATURE REVIEW SUMMARY

Year and Citation	Article/ Author	Tools/ Software	Technique	Source	Evaluation Parameter
1st edition in 2010. & 2nd edition in 2020.	"Continuous Integration and Delivery for Modern Web Applications " by David Farley and Jez Humble.	Jenkins, Docker, Kubernetes, Ansible, Git, CircleCI, Docker.	Automated Testing, Continuous Integration, Infrastructure as Code, Containerization .	O'Reilly Online Learning, Amazon Kindle, and other eBook platforms .	Software development, Practicality, Scalability, Automation, Testing, Deployment, Infrastructure as code.
1st edition published in 2008.	"Building a Continuous Integration Pipeline for a Web Application" by Andrew Glover	Jenkins, Git, GitHub, Maven, Selenium.	Setting up Jenkins Server, Creating a Git Repository, Configuring a build job in Jenkins.	IBM Developer Works Website.	Continuous Integration, Pipeline, Code Snippets, Deployment Testing, Open-Source
1st edition published in 2015.	"Continuous Delivery for Web Applications with Jenkins and Docker" by Rafal Leszko.	Jenkins, Docker, SonarQube and Ansible.	Building a Docker Image, pushing a Docker Image to a registry, Setting up production environment.	Official Jenkins Website.	Jenkins, Docker, Automation, Practicallity.
1st edition published in 2020.	"Continuous Integration and Deployment for Web Applications	GitLab, Git, Docker, Ansible and Nginx.	Setting up GitLab, Configuring GitLab CI/CD, Deploying the application	Online tutorial published on the GitLab website.	GitLab CI/CD, Continuous Integration, Continuous Deployment

	with GitLab CI/CD" by Andrei Dascalu.		and Monitoring Application.		
1st edition published in 2012.	"Automating Web Application Deployment with Jenkins" by Jayway Team	Jenkins, Git, Maven, Tomcat, Docker And ansible.	Creating Docker Container, Deploying the Container and Managing the application.	Article published on the Jayway website	Continuous Integration, Continuous Delivery, Accessibility.
1st edition published in 2016.	"Continuous Integration and Deployment for Web Applications with CircleCI" by Andrew Powell-Morse.	CircleCI, Docker And AWS Elastic Beanstalk.	Setting Up CircleCI, Creating a Docker Container, And Testing the application.	Published on the SitePoint website.	CircleCI, Continuous Deployment, Web applications .
1st edition published in 2018.	"Building a CI/CD Pipeline for Web Applications with AWS CodePipeline" by Mark Richman.	AWS CodePipeline , AWS CodeBuild, AWS Elastic Beanstalk and Amazon S3.	Setting up code pipeline, Configuring code build, And creating a rollback mechanism.	AWS Website.	AWS CodePipeline, Web application Development And Continuous integration.

.

3. PROBLEM FORMULATION

The problem formulation for the CI/CD pipeline for web applications is as follows:

Manual processes for building, testing, and deploying web applications that are inefficient and error-prone slow down development and increase the risk of errors and downtime.

Traditional software development practises for building, testing, and deploying web applications rely on manual processes. This method is slow and error-prone, and it can result in long lead times, missed deadlines, and application downtime. Complex build processes, limited testing, and manual deployment procedures may cause delays and errors for developers.

Solution: Set up a continuous integration/continuous delivery pipeline for web applications to automate the build, testing, and deployment processes. This method entails using automation tools and technologies to streamline the development process, reduce the risk of errors, and accelerate time to market. Developers can focus on building features and improving the application rather than manual processes by automating the process.

Objectives:

Automate the build, testing, and deployment processes to increase development speed.
Increase the application's reliability and quality by running automated tests at each stage of the pipeline.
By automating the deployment process, you can reduce the risk of errors and downtime.
By providing a standardised pipeline for development and deployment, you can improve collaboration and transparency among development teams.

Constraints:

The ability to implement and maintain a CI/CD pipeline may be hampered by a lack of resources (e.g., hardware, software, and personnel).
The pipeline's design and implementation may be influenced by security and compliance requirements.
Integration of legacy systems or applications with the pipeline may necessitate additional effort.

Metrics:

Build time: the amount of time required to build the application and generate a deployable artefact.
Test coverage: the percentage of the application that has been tested automatically.
Deployment frequency: the frequency with which new changes to the application are deployed.
The average time it takes to recover from an application failure is referred to as the mean time to recovery (MTTR).

4. OBJECTIVES

The following are the goals of implementing a CI/CD pipeline for web applications:

- Increase development speed: By automating the build, testing, and deployment processes, developers can concentrate on adding new features and improving the application. The CI/CD pipeline speeds up the development process, reducing time to market.

- Improve application quality: The CI/CD pipeline ensures that each stage of development is thoroughly tested. Automated tests detect errors and prevent bugs from reaching the production environment, resulting in higher-quality applications.

- Reduce the possibility of errors and downtime: Automation of the deployment process reduces the possibility of human error and downtime. The pipeline also allows for rapid failure recovery, ensuring that the application is always available.

- Increase collaboration and transparency: The CI/CD pipeline provides a standardised pipeline for development and deployment, allowing development teams to collaborate and communicate more effectively. Everyone in the organisation has access to the same pipeline, which ensures consistency.

- Allow for faster feedback loops: The CI/CD pipeline allows developers to receive feedback on code changes more quickly. Rapid feedback loops are enabled by automated testing and deployment, allowing developers to make improvements quickly and efficiently.

- Allow for more frequent releases: The CI/CD pipeline allows for the more frequent release of new features and improvements. New features can be released quickly and efficiently with automated testing and deployment, reducing time to market.

- Increase efficiency: By automating the build, testing, and deployment processes, developers can focus on developing new features and improving the application. The pipeline also saves time and effort on manual testing and deployment.

5. METHODOLOGY

The steps for implementing a CI/CD pipeline for web applications are generally as follows:

Plan and design: The pipeline's first step is to plan and design it. This includes defining the pipeline stages, identifying the requirements, selecting the tools and technologies to be used, and defining the deployment strategy.

The pipeline must be developed and tested in the second step. This includes constructing the pipeline as well as writing the necessary scripts and configurations. Testing the pipeline is also necessary to ensure that it is operational and meets the requirements.

Integrate and automate: The pipeline must be integrated and automated in the third step. This entails connecting the pipeline to the source code management system, automating the build, test, and deployment processes, and ensuring the pipeline's robustness and scalability.

Deploy and monitor: The fourth step is to deploy and monitor the pipeline's performance. This includes putting the pipeline into production, monitoring it for errors or failures, and fine-tuning it as needed.

The final step is to improve the pipeline on a continuous basis. This includes monitoring the pipeline's performance, gathering user feedback, and improving the pipeline based on the feedback.

Implementing a CI/CD pipeline for web applications should be done in an iterative and incremental manner. Before moving on to the next iteration, each iteration should focus on delivering a small set of features or improvements that can be tested and validated. The pipeline should also be monitored and improved on a regular basis to ensure that it is meeting the needs of the users and the organisation.

6. EXPERIMENTAL SETUP

The following components are typically included in the experimental setup for a CI/CD pipeline for web applications:

- Source code management system: This is the system where the web application's source code is stored, versioned, and managed. Git, SVN, and Mercurial are examples of source code management systems.

- The continuous integration server is in charge of building and testing the application whenever new code changes are pushed to the source code management system. Jenkins, Travis CI, and CircleCI are examples of continuous integration servers.

- Continuous deployment server: This is the server in charge of deploying the application to production whenever new changes are pushed to the source code management system and the application has been successfully built and tested by the continuous integration server. Ansible, Puppet, and Chef are examples of continuous deployment servers.

- The testing framework is used to test the application during the build and deployment process. JUnit, Selenium, and TestNG are examples of testing frameworks.

- Cloud infrastructure: This is the infrastructure that hosts the application as well as the various servers that are used in the CI/CD pipeline. Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform are examples of cloud infrastructure (GCP).

- Monitoring and logging tools: These are tools for monitoring application and pipeline performance, as well as logging and analysing errors and failures. Nagios, Grafana, and the ELK stack are examples of monitoring and logging tools.

The experimental setup should be designed to simulate a real-world scenario, with a production-like environment representative of the expected application usage. The pipeline's performance should be monitored and analysed to identify areas for improvement, and user feedback should be collected to assess the pipeline's effectiveness in meeting their needs.

7.CONCLUSION

Continuous Integration/Continuous Deployment (CI/CD) is a development methodology that allows developers to automate the process of creating, testing, and deploying software applications. The CI/CD pipeline is a collection of automated processes that assist developers in ensuring that code changes are constantly tested and integrated into the application without errors or issues.

In the world of software development, CI/CD is becoming increasingly popular, particularly for web applications. This is because web applications are typically more complex than other types of software applications, necessitating more testing and integration to ensure proper operation. This article will go over the CI/CD pipeline for web applications and how it can help developers.

The Web Application CI/CD Pipeline

The CI/CD pipeline for web applications is divided into several stages that developers must complete in order to ensure that their applications are continuously integrated, tested, and deployed. The main stages of the CI/CD pipeline for web applications are as follows:

- Commit Code

The code commit stage is the first stage in the CI/CD pipeline for web applications. Developers commit their changes to a code repository, such as GitHub or Bitbucket, at this stage. The code repository serves as a hub for developers to collaborate and share code changes.

- Construction Stage

The build stage is the second stage in the CI/CD pipeline for web applications. During this stage, the source code is compiled and a deployable artefact is created. The build process should be automated and started by the code commit stage.

- Stage of testing

The test stage is the third stage in the CI/CD pipeline for web applications. This stage entails testing the code changes to ensure that no bugs or issues are introduced. This stage allows for the execution of a variety of tests, including unit tests, integration tests, and acceptance tests.

- Stage of Deployment

The deploy stage is the fourth stage in the CI/CD pipeline for web applications. At this stage, the code changes are deployed to the production environment. The deployment process should be automated and initiated by the successful completion of the test stage.

The Advantages of a CI/CD Pipeline for Web Applications

Using a CI/CD pipeline for web applications has several advantages, including:

- Reduced Time-to-Market

The web application CI/CD pipeline enables developers to release new features and updates more frequently, reducing time-to-market. This leads to increased customer satisfaction and revenue for the company.

- Better Quality

The CI/CD pipeline's automated testing ensures that any issues or bugs are discovered early in the development process. This results in higher-quality software and lowers the likelihood of errors and downtime.

- Improved Collaboration

The code repository and automated processes in the CI/CD pipeline encourage developer collaboration, allowing them to collaborate more efficiently and effectively.

- Increased agility

The CI/CD pipeline for web applications enables developers to respond quickly to market, customer, and business requirements changes. This allows businesses to stay competitive and adapt to changing conditions.

Conclusion

Finally, the CI/CD pipeline for web applications is a must-have tool for developers looking to improve their software development process. Developers can ensure that their applications are of high quality, released faster, and more responsive to changing requirements by automating the building, testing, and deployment processes. A CI/CD pipeline for web applications has numerous advantages, and it is a tool that all developers should consider using to improve their software development process.

8. TENTATIVE CHAPTER PLAN FOR THE PROPOSED WORK

CHAPTER 1: INTRODUCTION

- Background and motivation
- Problem statement and objectives
- Research questions
- Significance and scope of the project
- Outline of the thesis

CHAPTER 2: LITERATURE REVIEW

- Introduction to DevOps and CI/CD pipelines
- Related work in the field
- Analysis and comparison of existing systems
- Advantages and limitations of DevOps and CI/CD pipelines

CHAPTER 3: OBJECTIVE

- Requirements gathering and analysis
- System design and architecture
- Tools and techniques selection
- CI/CD pipeline design and configuration
- Security design and implementation

CHAPTER 4: METHODOLOGIES

- Development of the static website
- Implementation of the DevOps CI/CD pipeline
- Automated testing script development
- Containerization and deployment on cloud-based hosting service

CHAPTER 5: EXPERIMENTAL SETUP

- Hardware and software requirements
- Website development process
- Configuration and deployment of the CI/CD pipeline
- Automated testing and quality assurance
- Security measures implementation

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

- Performance evaluation of the CI/CD pipeline
- Quality assessment of the deployed website
- Security analysis and evaluation
- Comparison with existing systems

REFERENCES

1. Kim, G., Humble, J., & Debois, P. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press.

2. Hassani, S., Abdollahi, A., & Oroumchian, F. (2021). Continuous Deployment of Web Applications: A Comprehensive Review. Journal of Software Engineering and Applications, 14(02), 77-92.

3. O'Reilly, T., & Loukides, M. (2019). Modern Web Development on the JAMstack: The Complete Guide. O'Reilly Media.

4. Krivtsov, A., & Nedosekin, A. (2018). Continuous Integration, Delivery and Deployment of Mobile and Web Applications. Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 670-675.

5. Fowler, M. (2018). Continuous Integration. Retrieved from https://martinfowler.com/articles/continuousIntegration.html

6. Duvall, P. M., Matyas, S., & Glover, A. (2007). Continuous Integration: Improving Software Quality and Reducing Risk. Pearson Education.

7. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

8. Docker. (2021). Docker Documentation. Retrieved from https://docs.docker.com/

9. Jenkins. (2021). Jenkins Documentation. Retrieved from https://www.jenkins.io/doc/

10. Travis CI. (2021). Travis CI Documentation. Retrieved from https://docs.travis-ci.com/