Project Report On

Secure CI/CD pipeline using Bitbucket and Docker registry

Submitted in partial fulfillment for the award of

Post Graduate Diploma in High Performance Computing System Administration from **C-DAC ACTS (Pune)**

Guided by

Mr. Roshan Gami

Presented By

Ms. Navodita Shrivastava	**PRN:230940127007**
Ms. Anuradha Chavan	**PRN:230940127031**
Mr. Tathagat Gaikwad	**PRN:230940127034**
Mr. Gourav Pavaiya	**PRN:230940127035**
Mr. Gautam Survade	**PRN:230940127050**

Centre of Development of Advanced Computing (C-DAC), Pune

TABLE OF CONTENTS

1. Abstract

This project aims to establish a secure Continuous Integration/Continuous Deployment (CI/CD) pipeline leveraging Bitbucket as the version control system and Docker registry for container image management. The project will be structured into several phases to ensure a comprehensive and robust implementation, while incorporating best practices to enhance security throughout the pipeline.

In the realm of modern software development, the implementation of a robust and secure Continuous Integration/Continuous Deployment (CI/CD) pipeline is imperative for ensuring efficient and reliable software delivery. This project endeavors to establish a Secure CI/CD Pipeline using Bitbucket and Docker Registry, orchestrated across three distinct virtual machine environments: Bitbucket, Jenkins, and Production. With Docker installed on both the Jenkins and Production machines, the pipeline integrates seamlessly, providing an automated workflow from code inception to production deployment.

The workflow initiates as developers contribute code from their local machines, which is subsequently integrated into a central Bitbucket repository. This repository serves as the cornerstone for version control, fostering collaboration and tracking changes across the development lifecycle. Upon code submission, Jenkins, hosted on a dedicated virtual machine, autonomously retrieves the latest code changes from Bitbucket to its local workspace, ensuring continuous monitoring and synchronization.

Jenkins, equipped with Docker capabilities, then proceeds to build Docker images encapsulating the application and its dependencies. This process occurs within the secure confines of the Jenkins machine, mitigating potential security vulnerabilities. Once the Docker image is successfully constructed, Jenkins securely communicates with Docker Hub, a centralized registry service, to push the image for storage and distribution. Docker Hub's robust features, including versioning and access control, enhance the integrity and manageability of the container images.

With the container image securely stored in Docker Hub, the pipeline advances to the deployment phase, targeting the designated Production environment. Leveraging passwordless SSH authentication, Jenkins orchestrates the deployment of the containerized application onto the Production machine. Docker,

deployed on the Production machine, facilitates the instantiation and execution of the container, ensuring seamless deployment and minimal downtime.

Upon successful deployment, stakeholders can access and interact with the application running on the Production machine, observing real-time outputs and feedback. This holistic approach to CI/CD not only accelerates the software delivery lifecycle but also fortifies security measures at each stage, safeguarding against potential threats and vulnerabilities.

Through the integration of Bitbucket, Jenkins, Docker, and Docker Hub, this Secure CI/CD Pipeline optimizes development workflows, enhances collaboration, and reinforces security standards, thereby empowering organizations to deliver high-quality software products with confidence and efficiency.

The expected outcome of the project is to provide a Secure CI/CD Pipeline Implementation with Bitbucket and Docker Registry process that meets the security requirements of the organization. The project will help reduce the risk of security breaches and ensure that the docker image is deployed securely.

2. Introduction

In today's fast-paced software development landscape, the need for efficient and secure Continuous Integration/Continuous Deployment (CI/CD) pipelines is paramount. Organizations are constantly seeking ways to streamline their development workflows while ensuring the reliability and security of their software deployments. To address these challenges, this project focuses on implementing a secure CI/CD pipeline utilizing Bitbucket, Jenkins, Docker, and Docker Hub.

The pipeline begins with developers working on their local machines, where they craft and refine code for their applications. Once the code is ready for integration, it is pushed to a central repository hosted on a Bitbucket server. Bitbucket serves as the version control system, providing a centralized location for collaboration and version tracking.

Upon code submission to Bitbucket, the CI/CD pipeline is triggered. Jenkins, running on a separate virtual machine, continuously monitors the Bitbucket repository for any changes. When new code is detected, Jenkins pulls the latest changes from Bitbucket to its local workspace for further processing.

Next, Jenkins leverages Docker, installed on the same machine, to build container images based on the application code. Docker facilitates the encapsulation of applications and their dependencies into lightweight, portable containers, ensuring consistency and reproducibility across different environments.

Once the container image is successfully built, Jenkins securely pushes it to Docker Hub, a cloud-based registry service for storing and distributing Docker images. Docker Hub serves as a central repository for managing container images, providing versioning, access control, and image scanning capabilities.

Subsequently, the CI/CD pipeline orchestrates the deployment of the containerized application onto the production environment. This environment consists of a separate virtual machine with Docker installed, designated for hosting production workloads. Jenkins initiates a passwordless SSH connection to the production machine, enabling

seamless communication and secure deployment.

Finally, the containerized application is launched on the production machine, allowing stakeholders to access and interact with the deployed application. Any output or feedback generated by the application can be observed directly on the production machine, facilitating real-time monitoring and troubleshooting.

By implementing this secure CI/CD pipeline, organizations can achieve faster time-to-market, improved software quality, and enhanced security posture. The integration of Bitbucket, Jenkins, Docker, and Docker Hub enables a seamless and automated workflow, empowering development teams to focus on innovation and delivering value to end-users.

3. Implementing Secure CI/CD pipeline using Bitbucket and Docker registry

---

Securing your CI/CD pipeline is essential for maintaining software integrity and stability. Integrating Bitbucket, Docker registry, and Jenkins offers a powerful and secure workflow. Here's a detailed guide with steps and considerations:

1. Setting Up:

- **Choose a Secure Docker Registry:** Opt for private options like Docker Private Registry, Azure ACR, or Google Container Registry. Utilize access control and security features unavailable in public registries.
- **Install and Configure Jenkins:** Securely install Jenkins and configure authentication/authorization using strong passwords and two-factor authentication.
- **Connect Bitbucket and Jenkins:** Establish secure communication using plugins like the "Bitbucket Branch Source Plugin" or Git SCM plugin with proper authentication methods.
- **Connect Jenkins and Docker Registry:** Utilize secure plugins like the "Docker Pipeline" plugin or dedicated registry plugins with secrets management for secure credential storage and access.

2. Build and Push Secure Images:

- **Create a Secure Jenkinsfile:** Implement security best practices within your Jenkinsfile. Utilize pipelines as code for transparency and version control.
- **Minimize Dependencies:** Reduce unnecessary dependencies and use trusted base images to minimize potential vulnerabilities.
- **Integrate Vulnerability Scanning:** Use plugins like "Docker Pipeline with Snyk Security" or "Aqua Security for Jenkins" to scan images for vulnerabilities before pushing.
- **Sign and Verify Images:** Implement Docker Content Trust (DCT) with plugins like "Docker Pipeline Sign and Verify" for image authenticity and integrity.

3. Additional Security Measures:

- **Secrets Management:** Never store sensitive information directly in Jenkins configurations. Use dedicated plugins like "Secret Management Plugin" for secure credential storage and access.
- **Network Security:** Secure communication between all components with firewalls, network segmentation, and other controls. Utilize plugins like "SSH Pipeline Plugin" for secure communication with remote servers.
- **Regular Security Audits:** Conduct regular penetration testing and security

audits of your pipeline and infrastructure to identify potential vulnerabilities and implement remediation measures.

Step-by-Step Breakdown:

1. **Secure communication:** Configure secure communication between Bitbucket, Jenkins, and the registry using secrets management or plugins.
2. **Secure Jenkinsfile:** Create a Jenkinsfile with vulnerability scanning, signing, and minimal dependencies.
3. **Logging & Monitoring:** Enable comprehensive logging and monitoring across all stages.
4. **Secrets Management:** Utilize dedicated plugins for secure credential storage and access.

4. System Requirement

User Interface

1. Ubuntu 22.0.4

Software Requirement

1. Bitbucket
2. Docker
3. Docker Registry
4. Jenkins
5. CI/CD Pipeline
6. CI/CD Using Jenkins
7. Open JDK 11

Hardware Requirement

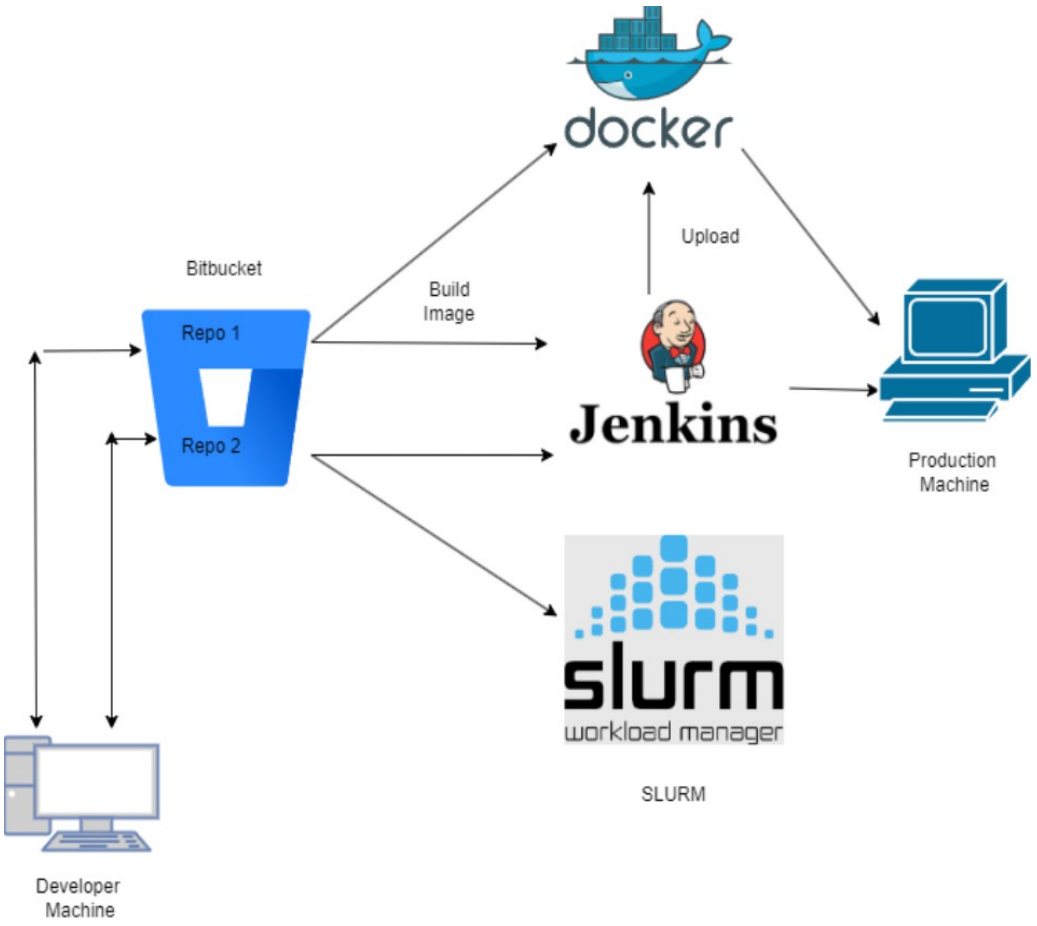
1. Linux Ubuntu
2. Ubuntu 22.0.4, 60GB , 4 GB RAM

Used Languages

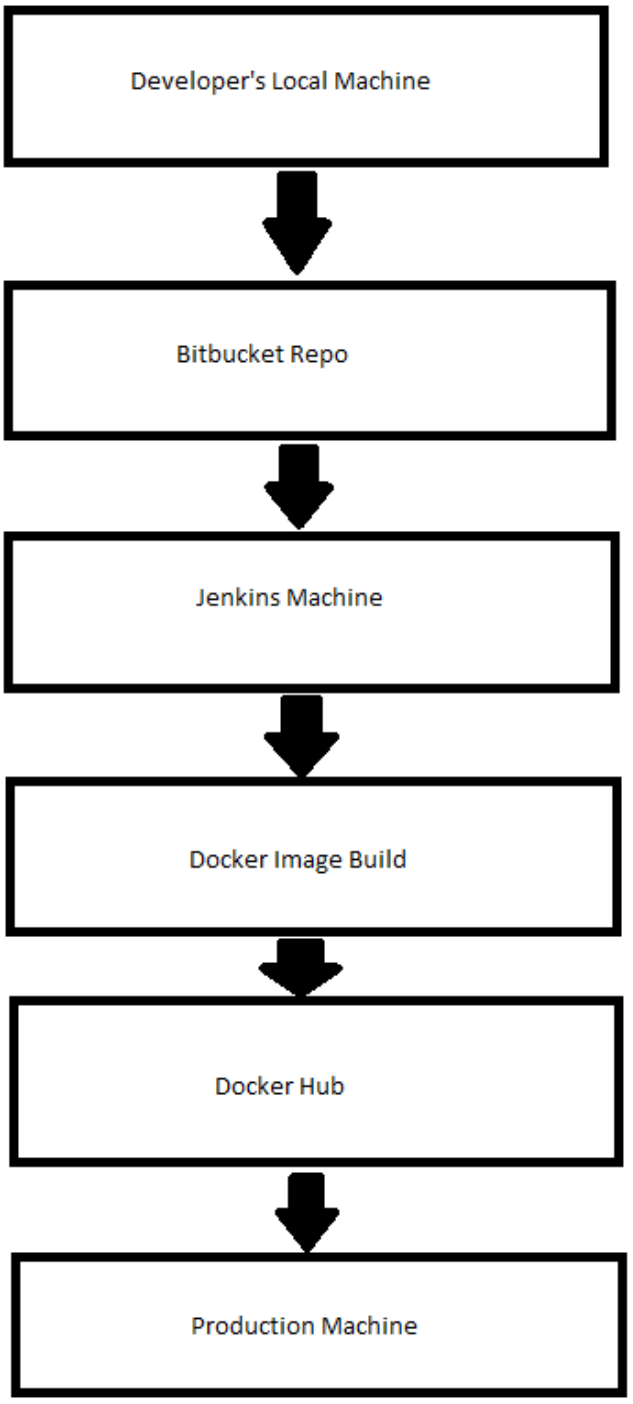
1. HTML
2. CSS
3. LINUX

5. System Architecture

Data Flow Diagram

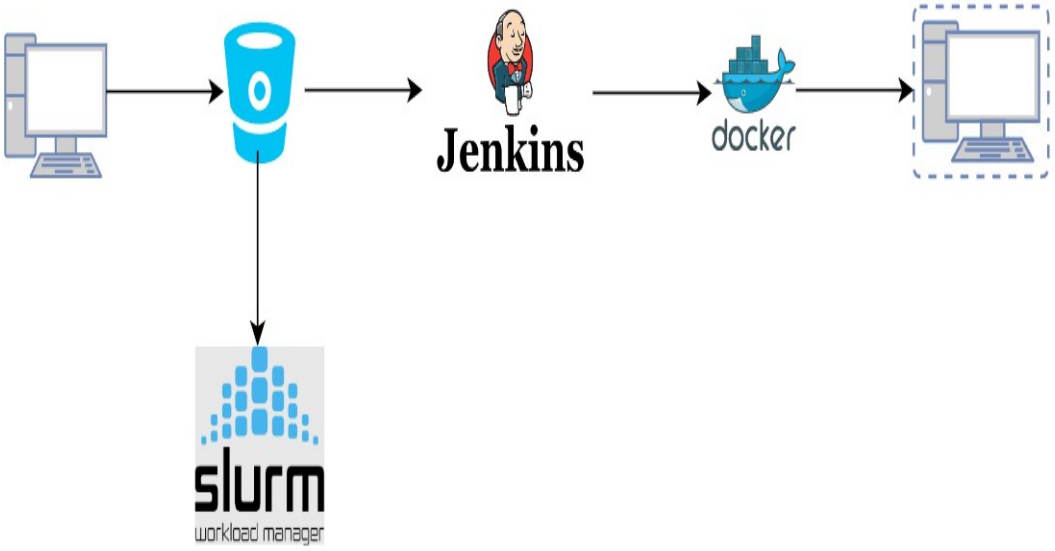


6. Activity Diagram



1. Developer adds code to their local machine.
2. The code is pushed to the Bitbucket repository.
3. Jenkins, running on its dedicated machine, pulls the code from the Bitbucket repository.
4. Jenkins builds the Docker image on its machine.
5. The built Docker image is pushed to Docker Hub.
6. The container is run on the Production machine.
7.Output and feedback from the containerized application can be observed on the Production machine.

5. Process Flow Diagram



6. Bitbucket Concepts

BitBucket is a cloud-based service that helps developers store and manage their code, as well as track and control the changes to their code. BitBucket provides a cloud-based Git repository hosting service. Its interface is user-friendly enough so even novice coders can take advantage of Git. We generally require a bit more technical knowledge and use of the command line to use Git alone. Additionally, BitBuckets provides a variety of services like it gives teams to collaborate and create projects, test and deploy the code.

To learn BitBucket, we need to have first-hand knowledge of:

- Version Control
- GIT

1. Version Control: It allows us to manage changes to files over time. It is also known as revision times. It's one of the important software configuration management. For ex: Atom Code Editor. Atom is one of the big open source project. If a developer wanted to make some changes to one specific part of the Atom codebase, it wouldn't be a good practice to have them directly merged to the official source code. Version Control lets developers safely and more efficiently work through **Branching** and **Merging**. A developer can copy/duplicate part of the source code(repository) with branching and can safely make changes to that part of the code without affecting the rest of the project. After making changes in the code, the developers can merge that code back into the main source code to make it official. All the changes can be tracked and can be reverted if required.

2. **GIT:** It is an open-source vision control system created by Linus Torvalds in the year 2005. Git is also known as a distributed version control system i.e all the codebase and history are available on every developer's computer, which allows for easy branching and merging.

2.1 Version Control Demystified:

Imagine working on a document, constantly revising and iterating. Without a version control system, tracking changes and reverting to previous versions becomes challenging. Git tackles this problem by offering a sophisticated framework for tracking code changes with meticulous detail.

Every modification, addition, or deletion is captured, allowing you to rewind or compare versions effortlessly. This empowers developers to experiment fearlessly, knowing they can always revert to a stable state if needed.

2.2 **The Power of Commits:**

The heart of Git lies in "commits," snapshots of your project's state at a specific point in time. Each commit encapsulates a set of changes, identified by a unique identifier and associated with a message outlining the modifications made. These messages serve as valuable documentation, providing context and insights into the evolution of your codebase. You can think of commits as milestones along your development journey, enabling you to revisit, compare, and understand the history of your project.

2.3 **Branching for Parallel Development:**

Imagine working on multiple features simultaneously without affecting the main codebase. This is where branching comes in. Git allows you to create branches, essentially independent lines of development diverging from the main codebase. This empowers you and your team to work on different features or bug fixes in parallel, without impacting each other's progress. Once satisfied with a branch, you can merge it back into the main codebase, integrating your changes seamlessly.

2.4 **Collaboration with Pull Requests:**

Git fosters collaboration by facilitating seamless integration of individual contributions. When you're ready to share your changes on a branch, you create a "pull request," a proposal to merge your branch into the main codebase. This triggers a review process where team members can assess your changes, provide feedback, and suggest modifications. This collaborative approach ensures code quality and consistency, promoting productive teamwork.

2.5 **Understanding Repositories:**

Think of a repository as the central hub for storing your project's codebase. It acts as a container for all your commits, branches, and tags, along with the complete version history. You can create multiple repositories for different projects, keeping your code organized and manageable. Version control platforms like GitLab, GitHub, and Bitbucket host these repositories, providing additional features and collaboration tools.

2.6 **Mastering Commands:**

While Git offers a graphical user interface, mastering its command-line interface unlocks its full potential. Core commands like git init, git add, git commit, and git push allow you to create, track, and share your code with precision. Learning a few essential commands significantly enhances your efficiency and control over your workflow.

2.7 **Practical Applications:**

Beyond the technical nuances, Git empowers developers in various scenarios:

- **Reverting Mistakes:** Accidentally introduce a bug? Simply revert to a previous stable commit with ease.
- **Collaborative Development:** Work on complex projects with a team, efficiently integrating contributions through pull requests.
- **Experimentation:** Fearlessly try new features or code improvements without jeopardizing the main codebase.
- **Open-Source Contributions:** Share your code with the world, allowing others to build upon your work and contribute back.

2.8 **Advanced Branching Strategies:**

Git goes beyond simple branching. We could explore advanced strategies like Gitflow, feature branching, and GitKraken merging, understanding their pros and cons for different scenarios.

2.9 **Integrating Git with Development Tools:**

Git seamlessly integrates with various tools like IDEs, build systems, and continuous integration/continuous delivery (CI/CD) pipelines. We could explore specific integrations and how they streamline your development process.

2.10 **Collaboration Best Practices:**

Effective collaboration plays a crucial role in Git workflows. We could discuss best practices for pull requests, code reviews, and conflict resolution, fostering productive teamwork.

2.11 **Security Considerations with Git** :

While Git empowers collaboration, security remains a key concern. We could explore best practices for access control, secret management, and protecting sensitive information within your repositories.

2.12 **Troubleshooting and Problem-Solving:**

- Even experienced users encounter Git issues. We could discuss common problems and troubleshooting steps, equipping you to handle errors and recover from mistakes confidently.

Advantages of Bitbucket

Version Control and Collaboration:

1. **Robust Git:** Powerful version control features for tracking changes, branching, and merging.
2. **Pull Requests:** Streamlined review and collaboration process with inline commenting and approvals.
3. **Forking and Branching:** Experiment and contribute easily with personal forks and dedicated branches.
4. **Team Access Control:** Granular control over user permissions for code access and visibility.
5. **Integration with Jira:** Manage projects seamlessly with integration into Jira's issue tracking system.
6. **Bitbucket Pipelines:** Build, test, and deploy code directly within Bitbucket using Docker containers.
7. **Integrations:** Connects with other tools like Trello, Slack, and CI/CD pipelines for efficient workflows.

Security and Reliability:

8. **SOC 2 compliance:** Ensures secure data storage and access control meeting industry standards.
9. **Two-factor authentication:** Adds an extra layer of security for user accounts.
10. **Code encryption:** Keeps your code confidential with optional at-rest encryption.
11. **Disaster recovery:** Secure backups and redundant systems minimize downtime and data loss.
12. **High availability:** Clustered architecture ensures uptime and scalability for large teams.

User Experience and Efficiency:

13. **Intuitive interface:** User-friendly web interface and mobile app for easy access and management.
14. **Keyboard shortcuts:** Navigates efficiently through functionalities with keyboard shortcuts.
15. **Git Large File Storage (LFS):** Efficiently store large files like assets and media without impacting code size.
16. **Code highlighting and syntax support:** Improves code readability and understanding.
17. **Customizable workflows:** Tailors to your team's needs and preferences with customisable workflows and plugins.

Open Source and Community:

18. **Free for Open Source projects:** Supports open-source development with free plans and features.
19. **Large and active community:** Access extensive documentation, tutorials, and forums for support and knowledge sharing.
20. **Contribution to Git development:** Contributes to the advancement of open-source Git technology.

Business Value and Scalability:

21. **Enterprise plans:** Scalable plans with additional features and support for large organizations.
22. **Advanced audit logs and data insights:** Monitor activity and gain valuable insights into your software development process.
23. **Compliance features:** Supports industry compliance needs with features like data residency and access control.
24. **Dedicated customer support:** Receive personalized support from Bitbucket's experienced team.

Additional Advantages:

25. **Code search and analysis:** Easily search your codebase and identify potential issues with analysis tools.
26. **Mercurial support:** Supports both Git and Mercurial version control systems.
27. **Bitbucket Server:** Self-hosted option for private repositories and complete control over your data.
28. **Bitbucket Data Center:** High-performance option for large enterprises with enhanced security and scalability.
29. **Integrations with Atlassian Marketplace:** Extends functionality with various add-ons and integrations.
30. **Active development and innovation:** Regular updates and new features ensure continued relevance and improvement.

Features & Applications of BitBucket

1. Robust Git: Bitbucket offers all the powerful features of Git, including tracking changes, branching, merging, and reverting.
2. **Pull Requests:** Streamline your review and collaboration process with Bitbucket's pull requests. Team members can leave inline comments, suggest edits, and approve or reject changes before they are merged into the main codebase.
3. **Forking and Branching:** Experiment and contribute easily with personal forks and dedicated branches. Fork a repository to create your own copy, make changes, and then submit a pull request to have your changes merged back into the original repository.
4. **Team Access Control:** Granular control over user permissions for code access and visibility. You can assign different levels of access to different users, such as read-only access, write access, or admin access.
5. **Integration with Jira:** Manage projects seamlessly with integration into Jira's issue tracking system. Link Bitbucket repositories to Jira issues to track progress, identify bugs, and assign tasks.
6. **Bitbucket Pipelines:** Build, test, and deploy code directly within Bitbucket using Docker containers. Bitbucket Pipelines automates your software development workflow, saving you time and effort.
7. **Integrations:** Bitbucket connects with other tools like Trello, Slack, and CI/CD pipelines for efficient workflows. You can automate tasks, share updates, and collaborate with your team using your favorite tools.

Security and Reliability:

7. **SOC 2 compliance** :
Ensures secure data storage and access control meeting industry standards.
8. **Two-factor authentication:** Adds an extra layer of security for user accounts.
9. **Code encryption:** Keeps your code confidential with optional at-rest encryption.
10. **Disaster recovery:** Secure backups and redundant systems minimize downtime and data loss.
11. **High availability:** Clustered architecture ensures uptime and scalability for large teams .

User Experience and Efficiency:

12. **Intuitive interface:** Bitbucket has a user-friendly web interface and mobile app for easy access and management.
13. **Keyboard shortcuts:** Navigates efficiently through functionalities with keyboard shortcuts.
14. **Git Large File Storage (LFS):** Efficiently store large files like assets and media without impacting code size.

15. **Code highlighting and syntax support:** Improves code readability and understanding.
16. **Customizable workflows:** Tailors to your team's needs and preferences with customisable workflows and plugins.

Open Source and Community:

17. **Free for Open Source projects:** Supports open-source development with free plans and features.
18. **Large and active community:** Access extensive documentation, tutorials, and forums for support and knowledge sharing.
19. **Contribution to Git development:** Contributes to the advancement of open-source Git technology.

Business Value and Scalability:

20. **Enterprise plans:** Scalable plans with additional features and support for large organizations.
21. **Advanced audit logs and data insights:** Monitor activity and gain valuable insights into your software development process.
22. **Compliance features:** Supports industry compliance needs with features like data residency and access control.
23. **Dedicated customer support:** Receive personalized support from Bitbucket's experienced team.
24. **Code search and analysis: Code search and analysis:** Easily search your codebase and identify potential issues with analysis tools.
25. **Mercurial support:** Supports both Git and Mercurial version control systems.
26. **Bitbucket Server:** Self-hosted option for private repositories and complete control over your data.
27. **Bitbucket Data Center:** High-performance option for large enterprises with enhanced security and scalability.
28. **Integrations with Atlassian Marketplace:** Extends functionality with various add-ons and integrations.
29. **Active development and innovation:** Regular updates and new features ensure continued relevance and improvement.

7. Difference between Bitbucket and Github

Parameters	Bitbucket	GitHub
Developed by	Bitbucket was developed by Jesper Noehr.	GitHub was developed by Chris Wanstrath, Tom Preston-Werner, P. J. Hyett, and Scott Chacon.
Version Control Systems	It supports Mercurial and Git.	It supports only Git.
Public Repository	It allows users to have multiple free repository.	It allows users to have unlimited free repository.
Private Repository	Bitbucket allows users to have free private repository but with maximum of five collaborators.	GitHub allows users to have free private repository but with maximum of three collaborators.
Navigation	Bitbucket has no feature for navigation.	GitHub allows user to navigate usability.
Project Analysis	Bitbucket allows developers to visualize the analysis with charts	GitHub doesn't have this feature yet but they can check the commit history.
Advantages	• Flexible with a variety of operating systems. • Authentication of social media support is created by Bitbucket.	• It helps us create an organized document for the project. • It is used for sharing the work in front of the public.
Disadvantages	• The maximum number of members can be 5 after that we need to pay for every additional member. • There is no stability when the process gets heavy and it results in slow down.	• There is a limited private repository. • It supports only Git version control.
Semantic Search	Supports Semantic Search features such as classes, and interfaces, etc thus saving a lot of time.	GitHub does not support Semantic Search features.

8. Docker Registry

A Docker registry is a system for storing and distributing Docker images with specific names. There may be several versions of the same image, each with its own set of tags. A Docker registry is separated into Docker repositories, each of which holds all image modifications. The registry may be used by Docker users to fetch images locally and to push new images to the registry (given adequate access permissions when applicable). The registry is a server-side application that stores and distributes Docker images. It is stateless and extremely scalable.

Uses of Docker Registry

1. Our images can be stored in the Docker registry.
2. We can automate the development.
3. With the help of a private docker registry, we can secure our image.

Why do We Use Docker Registry?

A Docker registry is an excellent way to supplement and integrate your CI/CD pipelines. Whenever there is a new commit in your source code or version control system, the CI workflow is triggered which then deploys the image to your registry if the CI workflow completion was successful. A signal from the Registry would then initiate a staging environment deployment or alert other systems to the availability of a new image.

So essentially we can say:

1. Docker Registry aids in development automation. The docker registry allows you to automate building, testing, and deployment. Docker registry may be used to create faster CI/CD pipelines, which helps to reduce build and deployment time.
2. Docker Registry is useful if you want complete control over where your images are kept. A private Docker registry can be used. You gain total control over your applications by doing so. In addition to controlling who may access your Docker images, you can determine who can view them.
3. Docker Registry can provide you with information about any issue you may encounter. You may also completely rely on it for container deployment and access it at any moment.

How does Docker Registry work?

1. Docker Registry provides a storage and distribution platform for Docker images.

2. Users can upload their Docker images to the registry, and these images can be tagged with a version number and a name.

3. Other users can then search for and download these images from the registry.

4. Docker Registry can be self-hosted or used as a cloud-based service.

Why is Docker Registry important?

1. Docker Registry is important because it makes it easy to share and distribute Docker images.

2. It simplifies the process of managing and deploying Docker containers, which can save time and resources

3. Docker Registry is a key component of the Docker ecosystem and is widely used by developers and organizations of all sizes.

Key Concepts:

1. Image :

Think of it as a portable software package encapsulating everything your application needs to run: code, libraries, runtime, and configuration.

Imagine a zip file containing all the ingredients for a recipe, ready to be used in any kitchen (container environment). Docker images are standardized, making them interchangeable and platform-agnostic.

2. Repository :

Visualize it as a bookshelf storing different versions of your application.

Each book on the shelf represents an image, identified by a unique tag (like book titles).

Repositories organize related images, allowing you to track changes, compare versions, and choose the right one for your needs.

3. Tag :

Think of it as a label attached to an image, specifying its version or purpose.

Tags differentiate between "latest", "stable", "development", or specific versions like "v1.2.3".

Using tags provides flexibility and clarity when selecting the right image for your environment.

4. Push & Pull:

Pushing is like uploading a book to the library (registry) so others can access it.

Pulling is like borrowing a book from the library, downloading the image to your local environment.

These actions enable sharing and collaboration, making containerized applications readily available.

Types of Docker Registries:

1. Public Registries:

Free to use, like Docker Hub.

Host millions of images from various users and organizations.

Useful for finding pre-built images like popular databases or development tools.

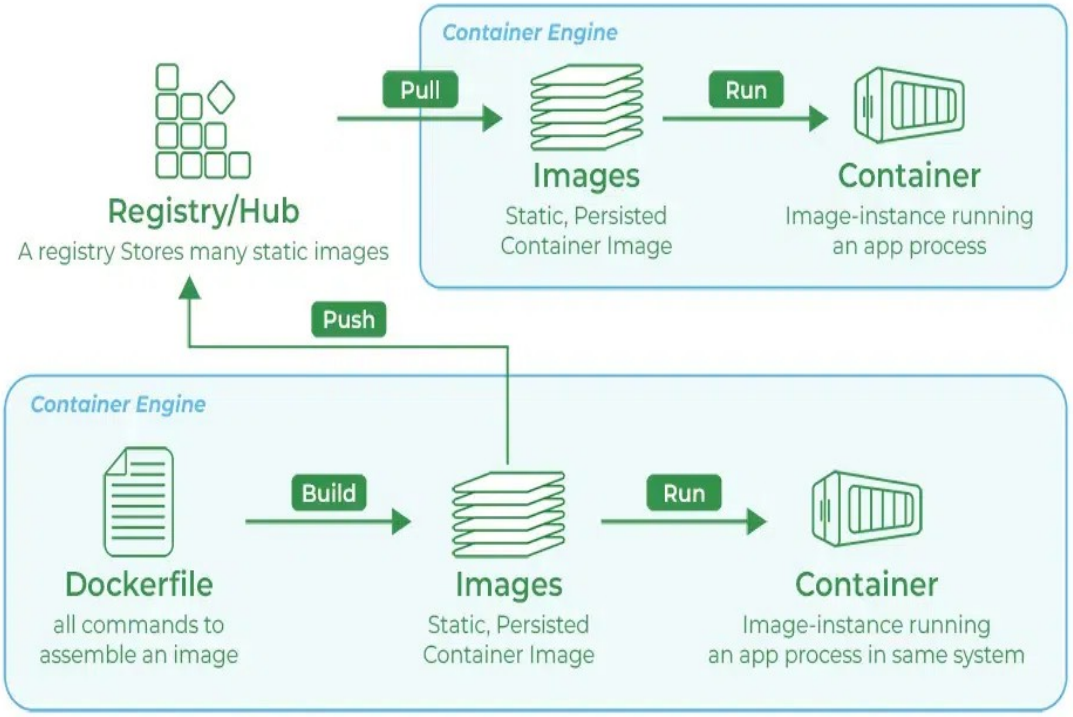
2. Private Registries:

Hosted on your own infrastructure or cloud platform.

Provide increased control and security for sensitive images.

Popular options include Docker Private Registry, Amazon ECR, Azure ACR, and Google Container Registry.

DOCKER RESISTORY DIAGRAM:



9. The Benefits of Docker Registry

---

1. A centralized location for storing container images: Centralized storage helps businesses to keep track of the container images they use in an efficient manner, without having to manage multiple images spread across different repositories.

2. A central location where users can find container images: Whether your users are employees of your business who need to access apps that you develop internally, external customers, or both, Docker Registry provides an easy-to-access place where they can search for and download container images.

3. **Access controls**: Most registries provide access control features that allow teams to manage who can access which container images. For example, in most cases, you can allow some images to be downloaded by the public at large while making others available only to certain authenticated users.

4. Image versioning: Docker Registry allows users to specify a version when downloading images and running them with Docker, Kubernetes, or a similar tool. Version management is useful in situations where some users need a particular version of an app (which may not be the latest version available), or when developers want to do things like make beta or alpha versions of their apps available while simultaneously offering stable releases.

5. Docker and Kubernetes integration: Docker Registry is designed to integrate natively with Docker's other tools and with Kubernetes, such that you can download and run container images from a registry using a single command. This integration saves users from having to download images, then upload them into a Docker or Kubernetes environment in order to use them. Most of the process is automated once you run a container to start a container based on a particular image.

CI/CD PIPELINE:

CI And CD is the practice of automating the integration of code changes from multiple developers into a single codebase. It is a software development practice where the developers commit their work frequently to the central code repository (Github or Stash). Then there are automated tools that build the newly committed code and do a code review, etc as required upon integration.

The key goals of Continuous Integration are to find and address bugs quicker, make the process of integrating code across a team of developers easier, improve software quality, and reduce the time it takes to release new feature updates. Some popular CI tools are Jenkins, TeamCity, and Bamboo.

Continuous Integration

There could be scenarios when developers in a team, work in isolation for an extended period and only merge their changes to the master branch once their work is completed. This not only makes the merging of code very difficult, prone to conflicts, and time-consuming but also results in bugs accumulating for a long time which are only identified in later stages of development. These factors make it harder to deliver updates to customers quickly.

With Continuous Integration, developers frequently commit to a shared common repository using a version control system such as Git. A continuous integration pipeline can automatically run builds, store the artifacts, run unit tests, and even conduct code reviews using tools like Sonar. We can configure the CI pipeline to be triggered every time there is a commit/merge in the codebase.

Continuous Delivery

Continuous delivery helps developers test their code in a production-similar environment, hence preventing any last-moment or post-production surprises. These tests may include UI testing, load testing, integration testing, etc. It helps developers discover and resolve bugs preemptively. By automating the software release process, CD contributes to low-risk releases, lower costs, better software quality, improved productivity levels, and most importantly, it helps us deliver updates to customers faster and more frequently. If Continuous Delivery is implemented properly, we will always have a deployment-ready code that has passed through a standardized test process.

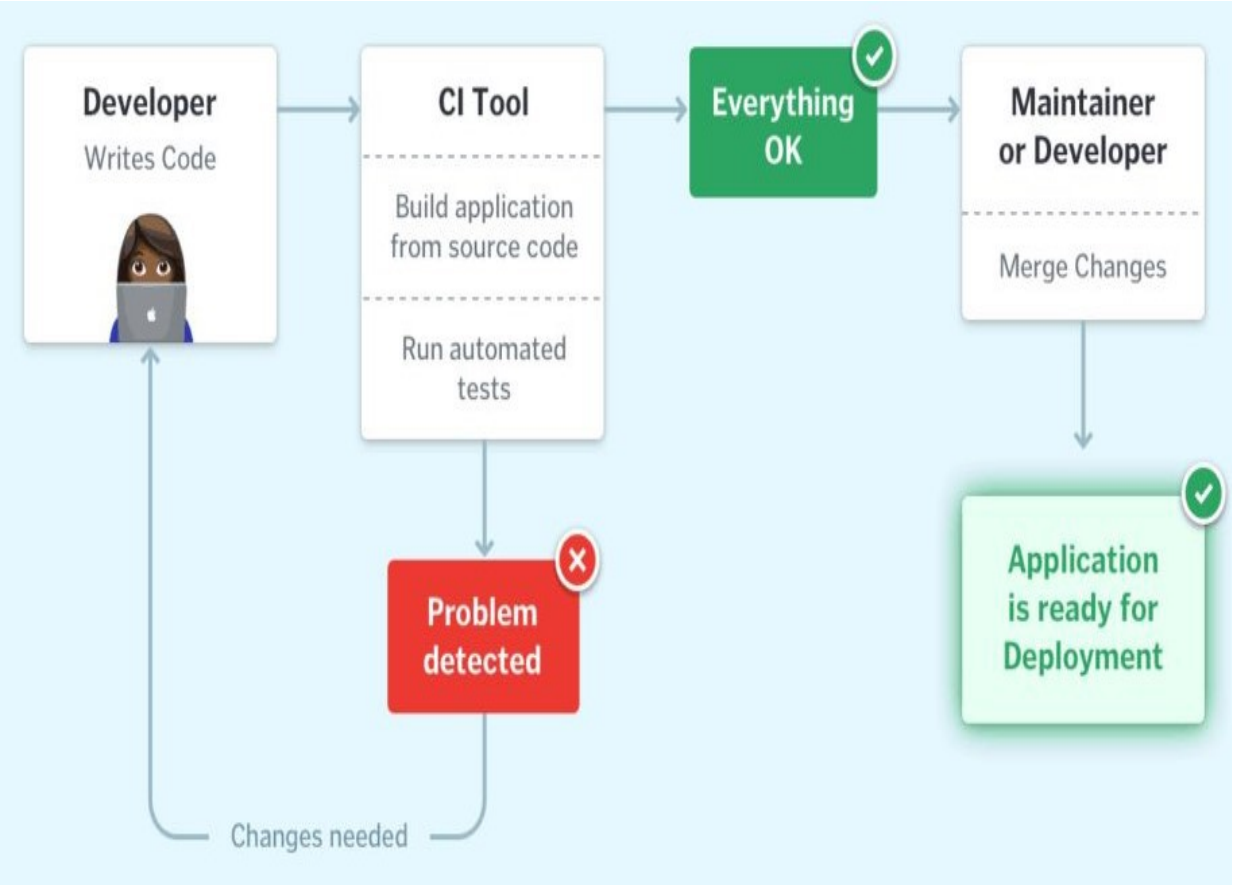
CD or **Continuous Delivery** is carried out after Continuous Integration to make sure that we can release new changes to our customers quickly in an error-free way. This includes running integration and regression tests in the staging area (similar to the production environment) so that the final release is not broken in production. It ensures automation of the release process so that we have a release-ready product at all times and we can deploy our application at any point in time. Continuous Delivery automates the entire software release process. The final decision to deploy to a live production environment can be triggered by the developer/project lead as required. Some popular CD tools are AWS CodeDeploy, Jenkins, and GitLab.

Continuous Deployment

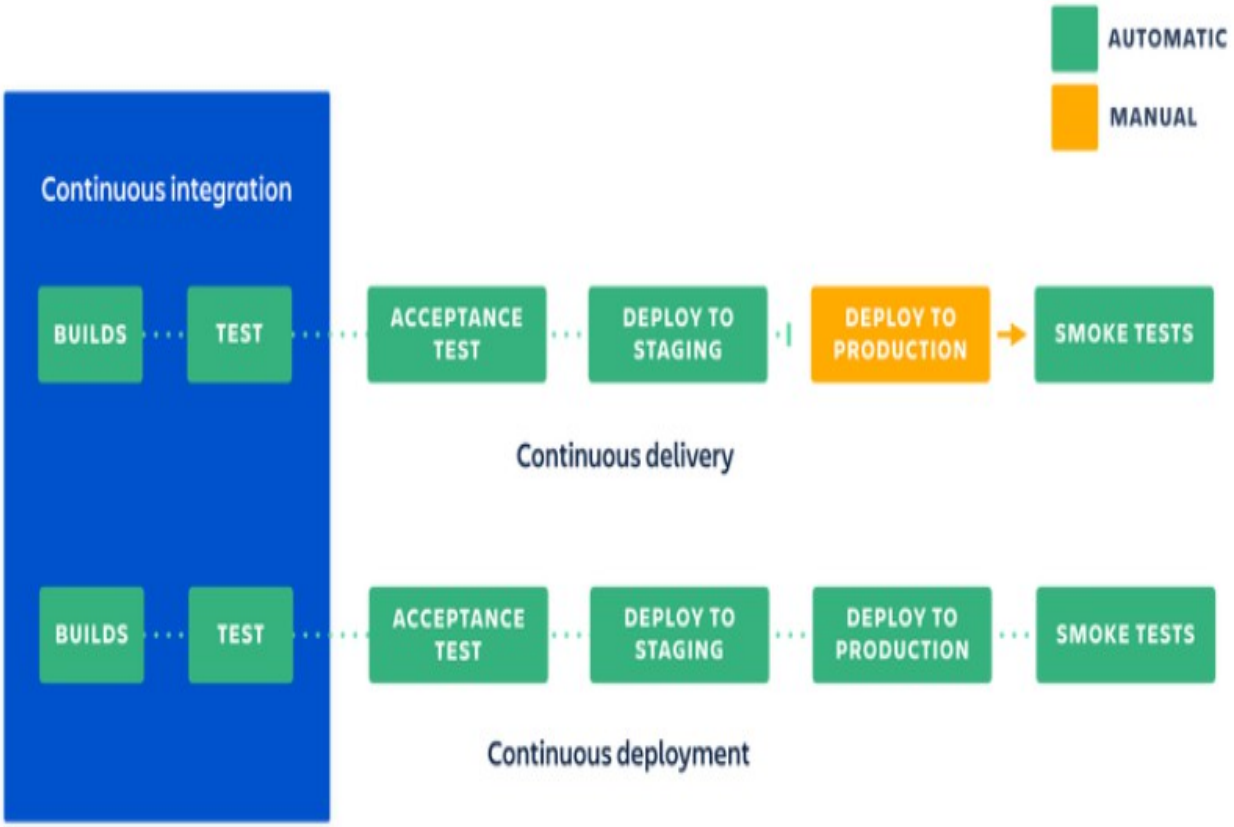
the final stage of CI and CD will be continuous deployment. It's an extension of continuous delivery, which automate the proper code to the code repository, continuous deployment will automate the related app for production purpose because there is not having any manual gate at the stage of the pipeline before production, continuous deployment relies on high automation.

in simple language, it is a change of application that goes through the cloud which is carried by the developer and it will live within a few minutes of writing pass with the automated testing.

CI Workflow:



CI and CD Workflow:



What are the Benefits of CI/CD?

Automated testing enables continuous delivery that ensures software quality and safety and increases code profitability in production. CI/ CD pipelines enable a much shorter time-to-market for new product features, resulting in happier customers and reducing the burden on development.
The significant increase in overall delivery speed enabled by CI/CD pipelines improves a company's competitive advantage.
Automation allows team members to focus on what they do best, resulting in the best end products. Companies with a successful CI/CD pipeline can attract outstanding talent. By moving away from traditional waterfall methods, engineers and developers are no longer engaged in repetitive activities that are often highly dependent on completing other tasks.

10. Jenkins

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.
It also allows us to continuously deliver our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.
With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

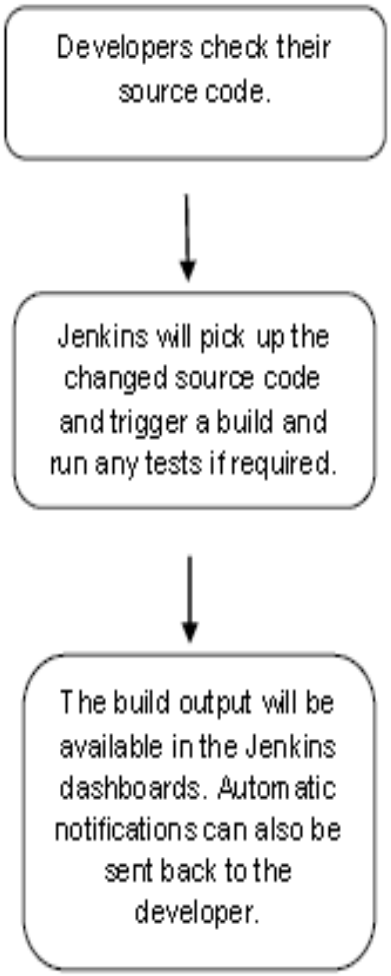
For example: If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Pipelines automate testing and reporting on isolated changes in a larger code base in real time and facilitates the integration of disparate branches of the code into a main branch. They also rapidly detect defects in a code base, build the software, automate testing of their builds, prepare the code base for deployment (delivery), and ultimately deploy code to containers and virtual machines, as well as bare metal and cloud servers. There are several commercial versions of Jenkins. This definition only describes the upstream Open Source project.

Jenkins History

Kohsuke Kawaguchi first developed Hudson in 2004 while working at Sun Microsystems. When Oracle acquired Sun Microsystems in 2010, there was a dispute between Oracle and the Hudson community with respect to the infrastructure used. There was a call for votes to change the project name from Hudson to Jenkins, which was overwhelmingly approved by the Hudson community on January 29, 2011, thereby creating the first “Jenkins” project.

Work Flow:



11. Jenkins Architecture

Here's how Jenkins elements are put together and interact:

- Developers commit changes to the source code, found in the repository.
- The Jenkins CI server checks the repository at regular intervals and pulls any newly available code.
- The Build Server builds the code into an executable file. In case the build fails, feedback is sent to the developers.
- Jenkins deploys the build application to the test server. If the test fails, the developers are alerted.
- If the code is error-free, the tested application is deployed on the production server.

The files can contain different code and be very large, requiring multiple builds. However, a single Jenkins server cannot handle multiple files and builds simultaneously; for that, a distributed Jenkins architecture is necessary.

Why Jenkins?

Jenkins is one of the top DevOps tools because it is free, open-source and modular, and can integrate with pretty much every other DevOps tool out there. There are over a thousand plugins that you can use to extend Jenkins' capabilities and make it more user-specific. All of these plugins and extensions are developed in Java. This means that Jenkins can also be installed on any operating system that runs on Java

12. Jenkins Benefits

1. **Faster development cycle**. Builds and tests on every commit create a fast-paced environment for getting rid of bugs. New features and releases reach the end-user faster and with fewer er

2. **Less time to integrate code**. Before Jenkins, code integration was a manual process, and debugging code was complex. Reaching a working version would require going through various commits and analyzing problems. When using Jenkins, integrating after each commit assures that the program functionality is always available and operational.

3. **Quick feedback for developers**. Whenever a build breaks, developers stay informed. The feedback system helps the dev team quickly address the issues instead of debugging numerous commits.

4. **Automated pipeline workflow**. Automated testing for each commit integrates into the Pipeline directly.

5. **Open source and free:** Jenkins is an open-source tool, which means that it is free to use and modify. This makes it a great option for organizations of all sizes, as there are no licensing fees involved.

6. **Highly extensible:** Jenkins has a large and active community of developers who have created over 1,700 plugins for the platform. These plugins allow you to extend Jenkins to meet the specific needs of your organization. For example, there are plugins for popular build tools, testing frameworks, and deployment tools.

7. **Easy to use:** Jenkins has a web-based interface that is easy to use, even for users with no prior experience with continuous integration. This makes it a great option for teams that are just getting started with CI/CD.

8. **Platform independent:** Jenkins runs on all major operating systems, including Windows, Linux, and macOS. This makes it a great option

for organizations that use a variety of different platforms.

9. **Scalable:** Jenkins can be scaled to meet the needs of small and large organizations alike. It can be deployed on a single server or on a cluster of servers.

10. **Reliable:** Jenkins is a mature and stable platform that has been used by millions of users around the world. It is a reliable choice for organizations that need a continuous integration tool that they can count on.

11. **Improves software quality:** By automating the build and test process, Jenkins can help to improve the quality of your software. This is because it can catch bugs early in the development process, before they are deployed to production.

12. **Reduces time to market:** By automating the deployment process, Jenkins can help you to get your software to market faster. This is because it can automate tasks such as building, testing, and deploying your software.

13. **Increases developer productivity:** By automating tasks such as building and testing, Jenkins can help developers to be more productive. This is because they can focus on writing code and fixing bugs, rather than spending time on manual tasks.

15. Docker Configuration

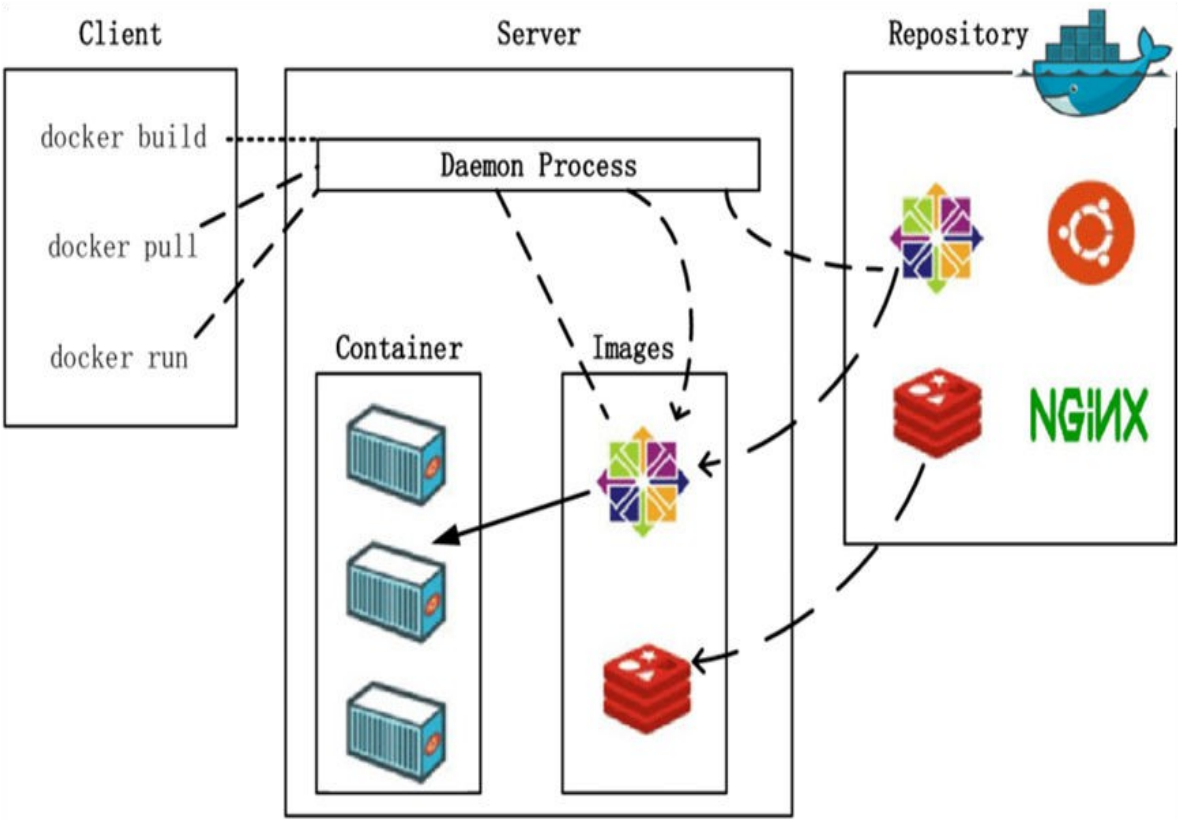
Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers are lightweight and portable units that can run on any machine with Docker installed, providing a consistent environment for applications to run in.

Here are the basic steps to configure Docker on your machine:

1. **Install Docker**: Docker provides installers for different operating systems. You can download the appropriate installer for your operating system from the Docker website and follow the instructions to install it on your machine.
2. **Verify the installation**: Once you have installed Docker, you can verify the installation by running the docker --version command in your terminal or command prompt. This command should return the version number of Docker that you have installed.
3. **Create a Dockerfile**: A Dockerfile is a text file that contains instructions to build a Docker image. You can create a Dockerfile in the root directory of your application by specifying the base image, copying files into the image, and running any commands to set up your application.
4. **Build the Docker image**: Once you have created a Dockerfile, you can use the docker build

command to build the Docker image. The basic syntax of the command is:

docker build -t <image name>:<tag>



Images and containers

A container is launched by running an image. An image is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.

A container is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, docker pc, just as you would in Linux.

After you run a docker image, it creates a docker container. All the applications and their environment run inside this container. You can use Docker API or CLI to start, stop, delete a docker container.

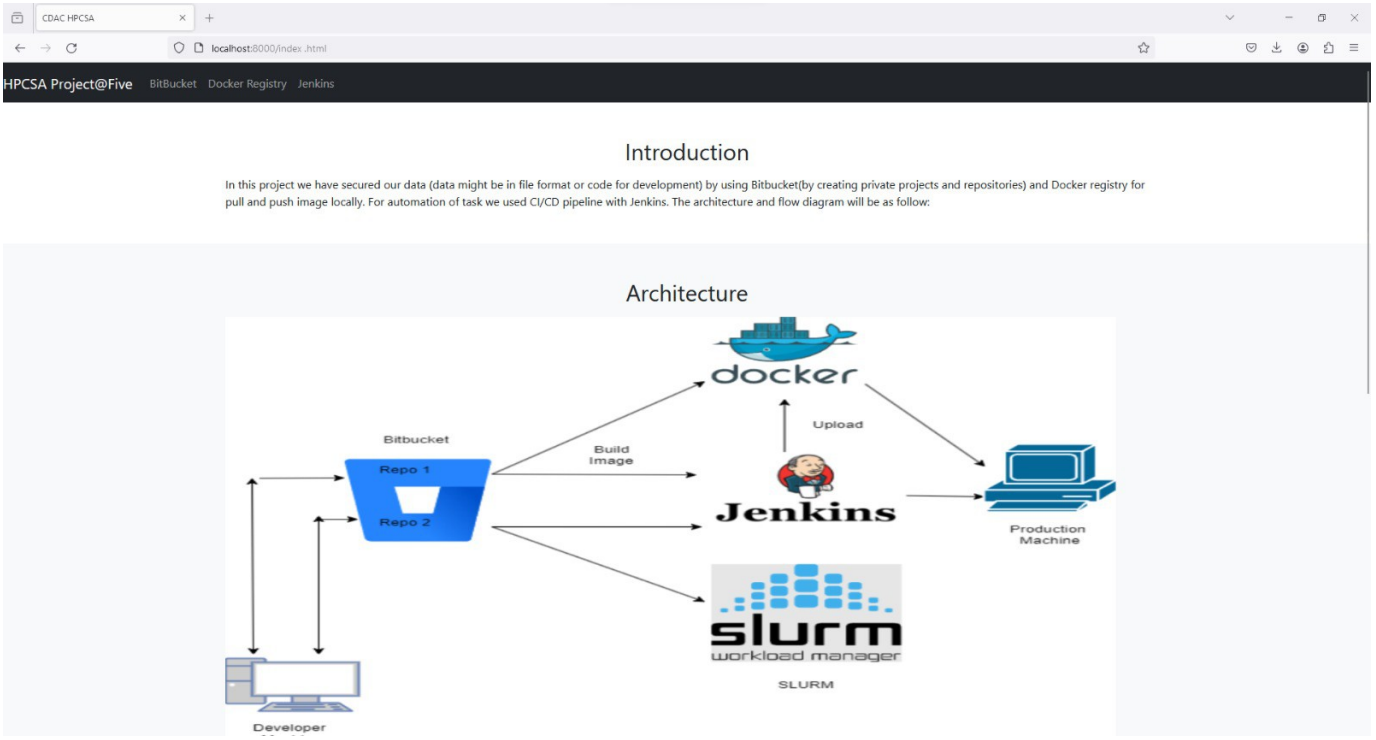
19 . CODE

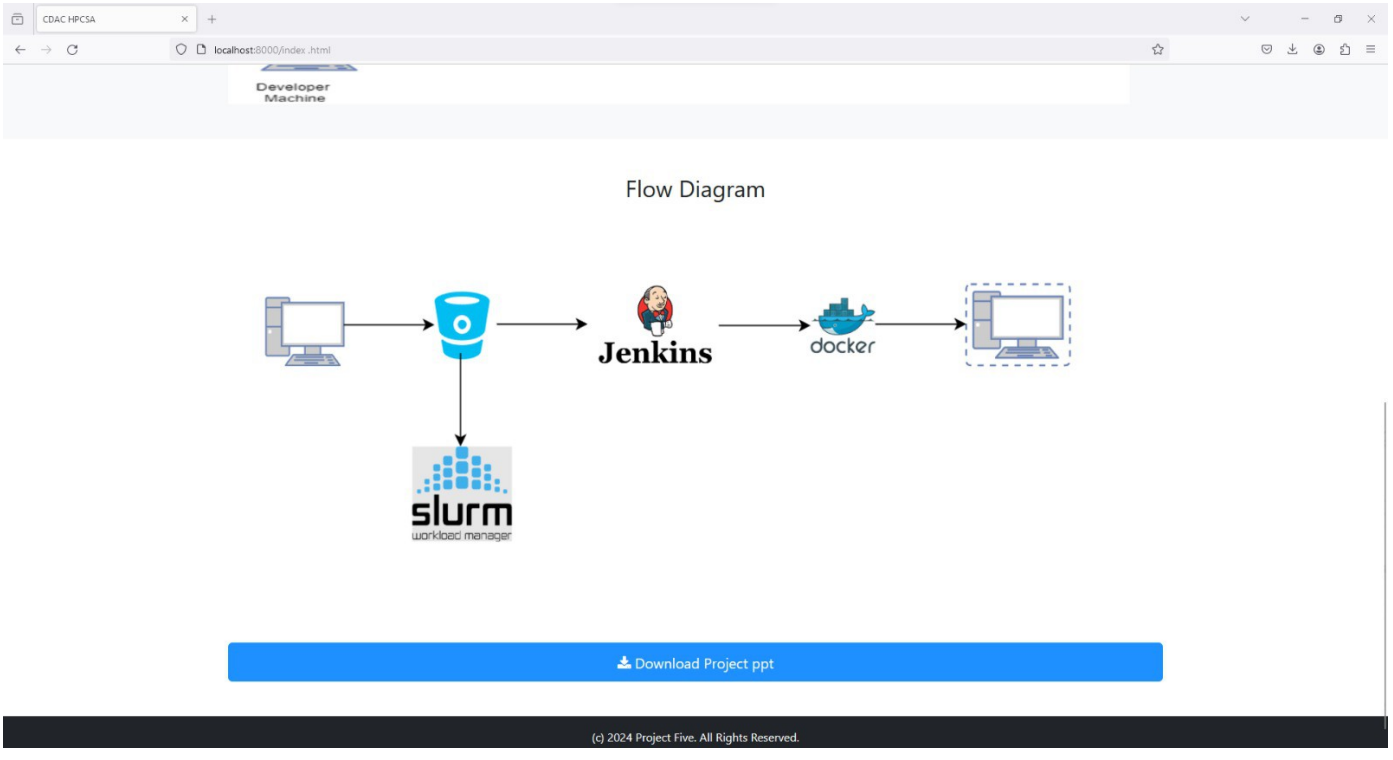
```
pipeline {
agent any
environment {
 DOCKERHUB_CREDENTIALS= credentials('docker_registry')
}
stages {
 stage("Fetch-code"){
  steps{
   sh '''
      cd /jenkins-demo/
      sudo rm -rf batch05
      sudo git clone http://bit:bit@192.168.144.239:7990/scm/hpcsa/batch05.git
      cd batch05
   '''
  }
 }
 stage("Build Docker Image"){
  steps{
   sh '''
      cd /jenkins-demo/batch05
      sudo docker build -t gourav1401/my-task .
      sudo echo 'Build Image Completed'
   '''
  }
 }
 stage('Login to Docker Hub') {
  steps{
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | sudo docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin'
   echo 'Login Completed'
```

```
        }
      }
      stage('Push Image to Docker Hub') {
        steps{
                sh '''
          sudo docker push gourav1401/my-task:latest
          echo 'Push Image Completed'
         '''
        }
      }
      stage('create-service') {
        steps{
                sh '''
      ssh -T master@192.168.144.242 "echo root | sudo -S docker stop bitbucket-app"
      ssh -T master@192.168.144.242 "echo root | sudo -S docker rm bitbucket-app"
      ssh -T master@192.168.144.242 "echo root | sudo -S docker rmi gourav1401/my-task:latest"
       ssh -T master@192.168.144.242 "echo root | sudo -S docker run --name bitbucket-app -p 8000:80 -d gourav1401/my-task:latest"
          echo 'Container created'
         '''
        }
      }
    } //stages
    post{
        always {
                sh 'docker logout'
        }
        }
  } //pipeline
```

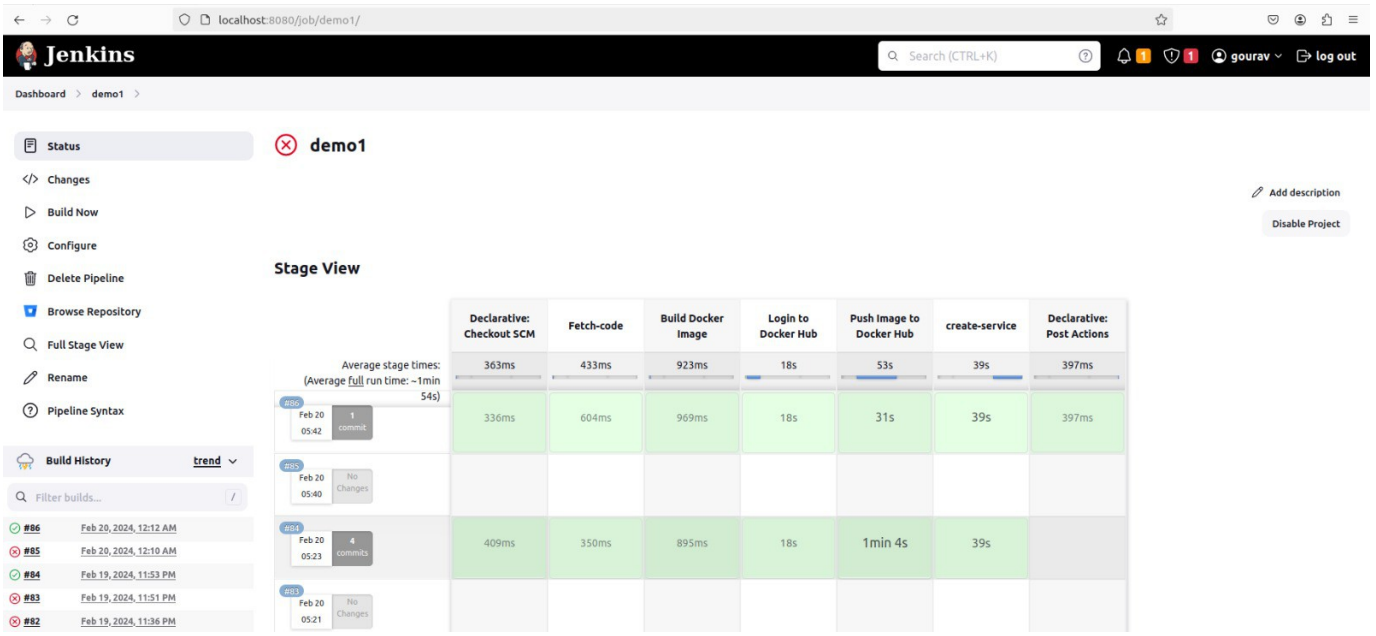
20. Result

- **Production Machine Output**

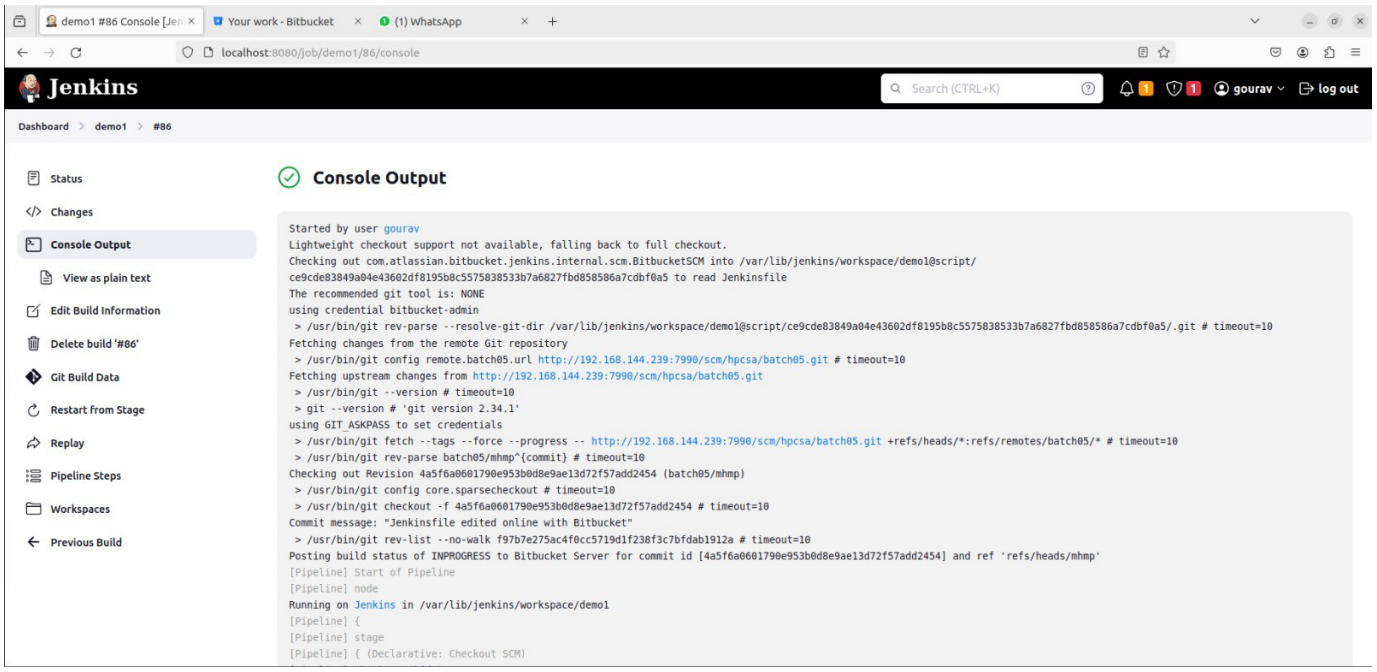


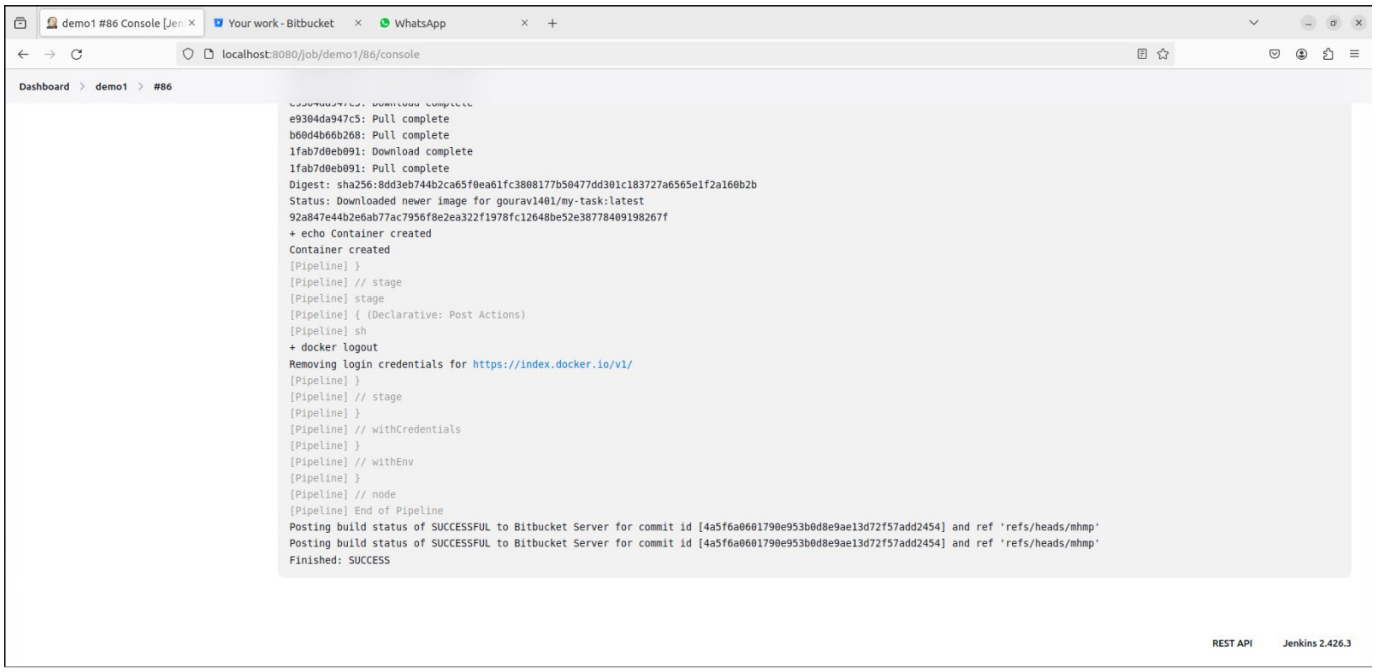


- **Pipeline**



Console output:




demo1 #86 Console [Jen
Your work - Bitbucket
WhatsApp
localhost:8080/job/demo1/86/console
Dashboard
demo1
#86
e9304da947c5: Pull complete
b60d4b66b268: Pull complete
1fab7d0eb091: Download complete
1fab7d0eb091: Pull complete
Digest: sha256:8dd3eb744b2ca65f0ea61fc3808177b50477dd301c183727a6565e1f2a160b2b
Status: Downloaded newer image for gourav1401/my-task:latest
92a847e44b2e6ab77ac7956f8e2ea322f1978fc12648be52e38778409198267f
+ echo Container created
Container created
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] sh
+ docker logout
Removing login credentials for https://index.docker.io/v1/
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Posting build status of SUCCESSFUL to Bitbucket Server for commit id [4a5f6a0601790e953b0d8e9ae13d72f57add2454] and ref 'refs/heads/mhmp'
Posting build status of SUCCESSFUL to Bitbucket Server for commit id [4a5f6a0601790e953b0d8e9ae13d72f57add2454] and ref 'refs/heads/mhmp'
Finished: SUCCESS
REST API
Jenkins 2.426.3

21. Conclusion

Implementing a secure CI/CD pipeline with Bitbucket and Docker registry provides numerous benefits for your software development process.

The implementation of a Secure CI/CD Pipeline with Bitbucket and Docker Registry, orchestrated across three distinct virtual machine environments, represents a significant advancement in streamlining software development workflows while prioritizing security and reliability. By following the outlined workflow, which involves developers contributing code, Bitbucket serving as the version control system, Jenkins managing the CI/CD processes, and Docker facilitating containerization and deployment, organizations can realize numerous benefits.

Firstly, the segregation of development, CI/CD orchestration, and production environments into dedicated virtual machines ensures compartmentalization and minimizes the risk of unauthorized access or data breaches. Furthermore, the utilization of Docker on both the Jenkins and Production machines promotes consistency and portability, enabling seamless deployment of containerized applications across different environments.

The establishment of passwordless SSH authentication between the Jenkins and Production machines enhances security by mitigating the risk of password-related vulnerabilities while facilitating automated deployment processes. This authentication mechanism ensures that only authorized entities can initiate deployment actions, safeguarding against unauthorized access to critical production infrastructure.

Additionally, the integration of Docker Hub as a centralized registry service enhances the manageability and scalability of the CI/CD pipeline. Docker Hub provides robust features such as versioning, access control, and image scanning, bolstering the integrity and security of container images throughout their lifecycle.

By adhering to the outlined workflow, organizations can achieve faster time-to-market, improved software quality, and enhanced security posture. The automated nature of the pipeline reduces manual intervention, thereby minimizing human error and accelerating the software delivery lifecycle. Furthermore, the visibility provided by real-time outputs on the Production machine enables stakeholders to promptly identify and address any issues or feedback, fostering continuous improvement and

iteration.

In conclusion, the Secure CI/CD Pipeline Implementation with Bitbucket and Docker Registry represents a comprehensive approach to software development and deployment, prioritizing security, reliability, and efficiency. By leveraging the strengths of Bitbucket, Jenkins, Docker, and Docker Hub, organizations can navigate the complexities of modern software development with confidence and agility, ultimately delivering value to end-users with unparalleled speed and resilience.

At this point our “Secure CI/CD pipeline using Bitbucket and Docker registry” project is being configured successfully.

22. References

1. **Bitbucket Pipelines Security:** https://www.atlassian.com/trust/security/security-practices

2. **Docker Security:** https://docs.docker.com/engine/security/

3. **Docker Private Registry Documentation:** https://docs.docker.com/registry/

4. **Docker Pipeline Plugin for Jenkins:** https://plugins.jenkins.io/docker-plugin

5. **Git SCM Plugin for Jenkins:** https://plugins.jenkins.io/git

6. **Docker Pipeline Sign and Verify Plugin:** https://marketplace.atlassian.com/

7. **Docker Pipeline Sign and Verify Plugin:** https://marketplace.atlassian.com/

Install Java : https://www.linode.com/docs/guides/how-to-install-openjdk-on-ubuntu-20-04/

Install Jenkins:
https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Ubuntu

Install Docker Regisrty:
https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-22-04

Activate Jenkins from /var/lib/Jenkins/secrets/initialAdminPassword.