LOVELY
PROFESSIONAL
UNIVERSITY

DevOps Project REPORT

ON

Jenkins CI/CD Deployement Hub

SUBMITTED BY:

NAME: POTHANAPALLI SAKSHI

REGISTRATION NO: 12219299

Section: KO022

Roll No: 33

SUBMMITED TO:

Mr. Rakesh Gagneja

1.Introduction

In today's rapidly evolving software landscape, the ability to deliver high-quality applications quickly and efficiently is essential. Traditional software development practices often involve manual code integration, testing, and deployment steps that are prone to errors, delays, and inefficiencies. To overcome these challenges, the adoption of Continuous Integration and Continuous Deployment (CI/CD) methodologies has become a cornerstone in DevOps practices. This project, titled *Jenkins CI/CD Deployment Hub*, is aimed at automating the software delivery lifecycle using Jenkins as the central automation server. It demonstrates how modern tools and technologies can streamline development workflows, ensure consistency, and facilitate faster software releases.

2.Objective

The primary objective of this project is to design and implement a fully functional CI/CD pipeline using Jenkins that integrates seamlessly with GitHub and Docker. This automation enables teams to detect issues early, deploy changes quickly, and maintain a consistent and reliable build environment. By setting up this pipeline, the project seeks to eliminate manual interventions, minimize human error, and promote a culture of continuous improvement in software development. It empowers developers to push code with confidence, knowing that it will be automatically built, tested, and deployed in a controlled and repeatable manner.

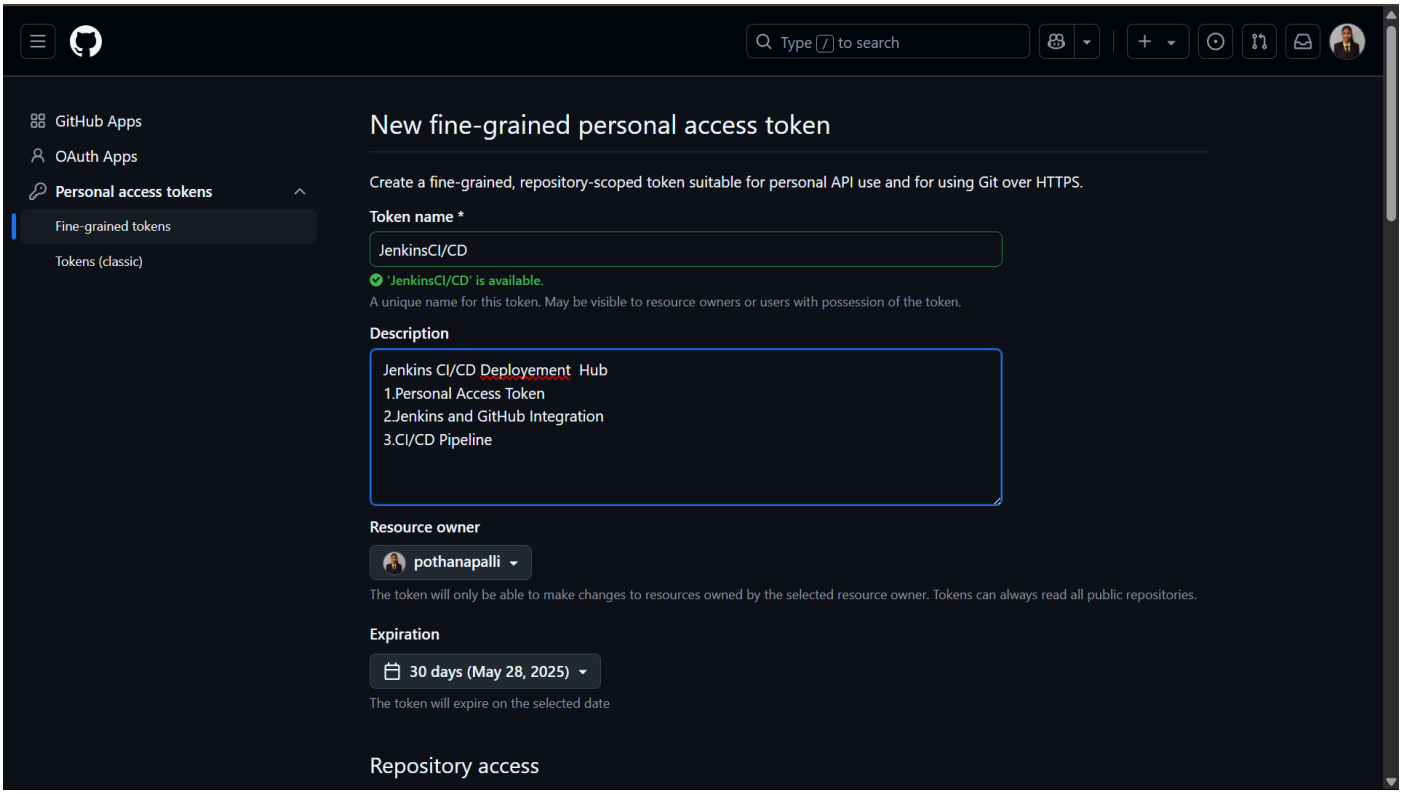
3.Project Components

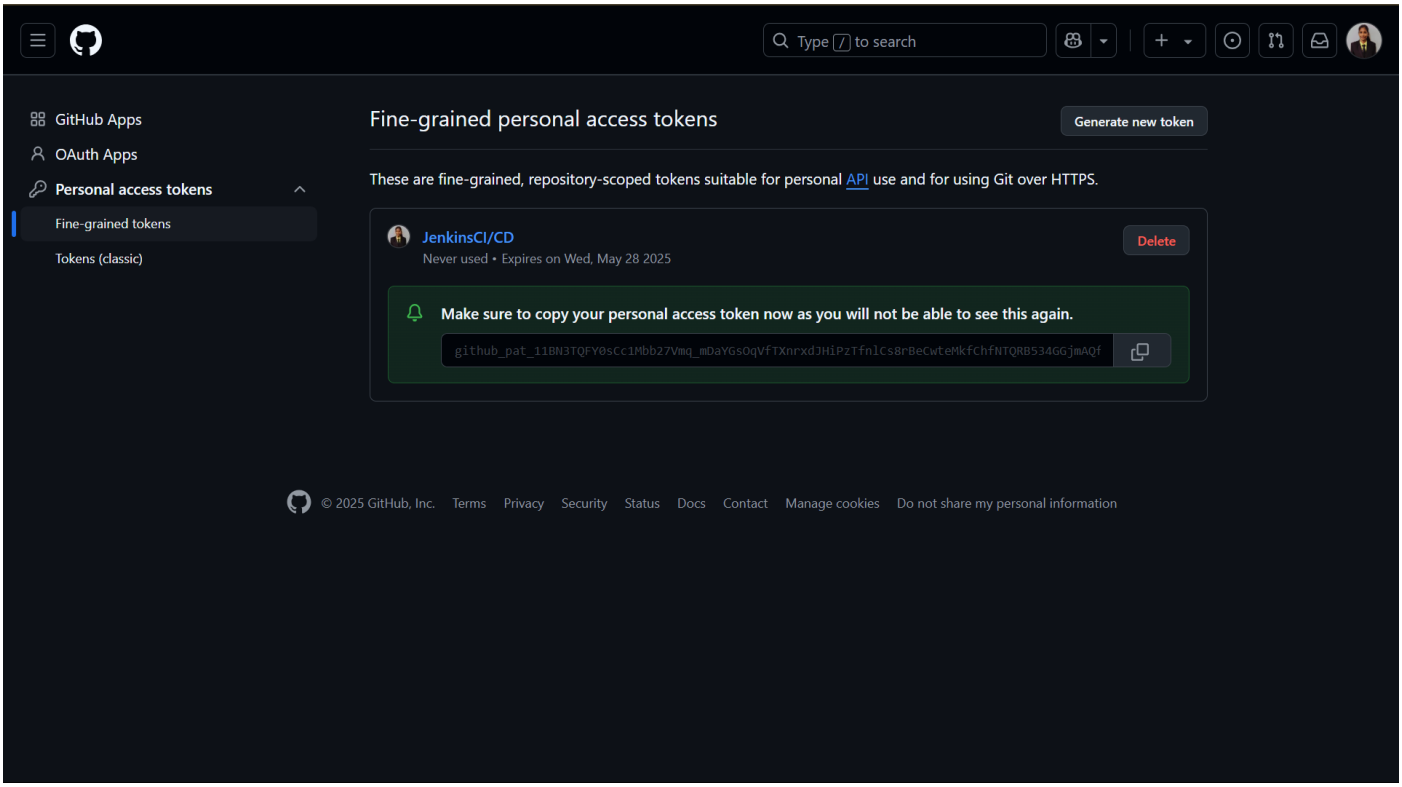
The Jenkins CI/CD Deployment Hub is comprised of three main phases that are integral to the successful implementation of automated

software delivery. Each phase plays a distinct role in establishing a secure, reliable, and efficient CI/CD pipeline.

3.1. Creation of Personal Access Token (PAT)

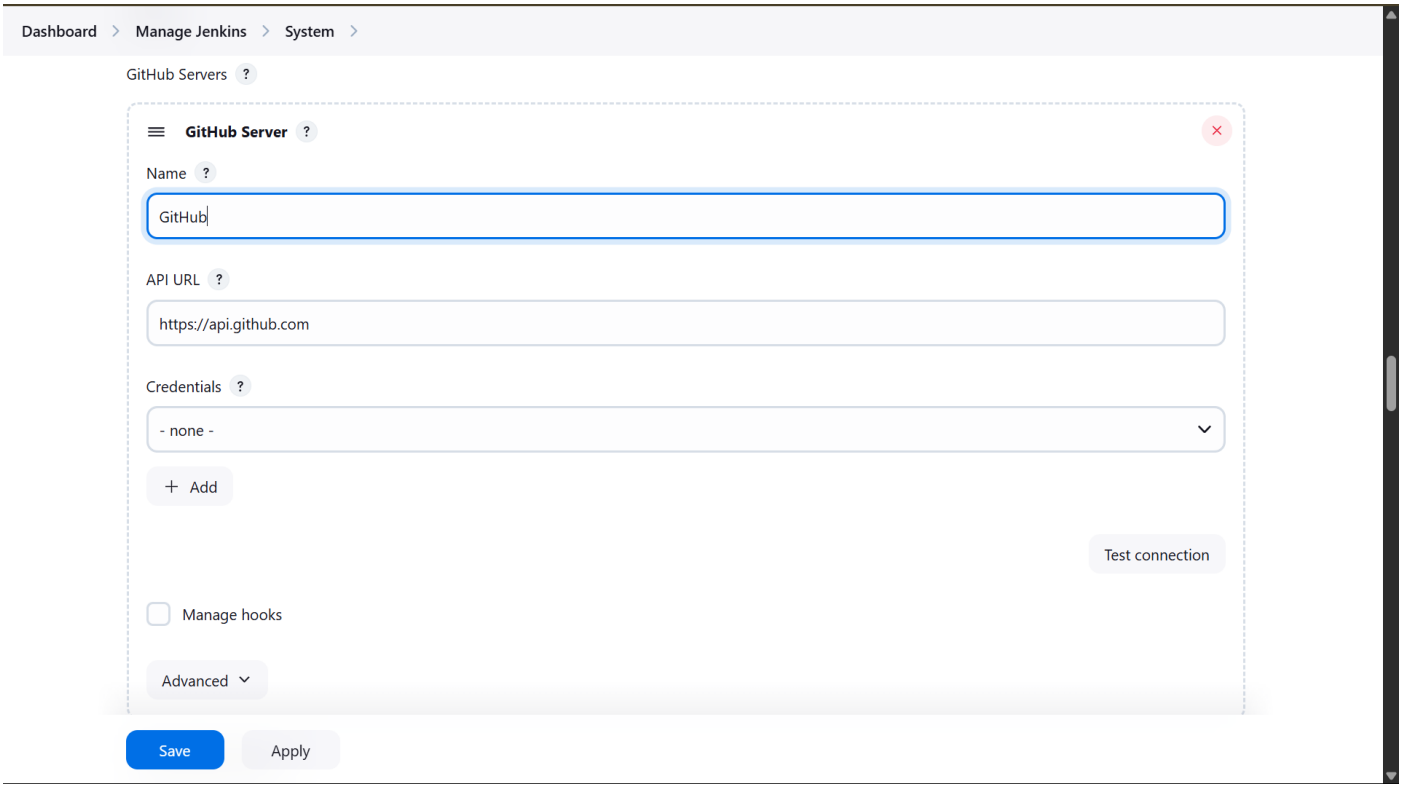
A Personal Access Token (PAT) serves as a secure alternative to passwords when connecting third-party applications like Jenkins to GitHub. With recent changes in GitHub's authentication policies, the use of passwords for integrations has been deprecated, and PATs have become the standard method for access control. In this project, the PAT acts as an authentication key that allows Jenkins to interact with GitHub repositories without exposing sensitive login credentials. To create the PAT, one must navigate to GitHub's developer settings and generate a token with appropriate scopes, such as repo access and workflow permissions. This token is then stored securely in Jenkins to enable GitHub integration. Using PATs enhances security, simplifies access management, and provides more control over the actions permitted by Jenkins within the repository.

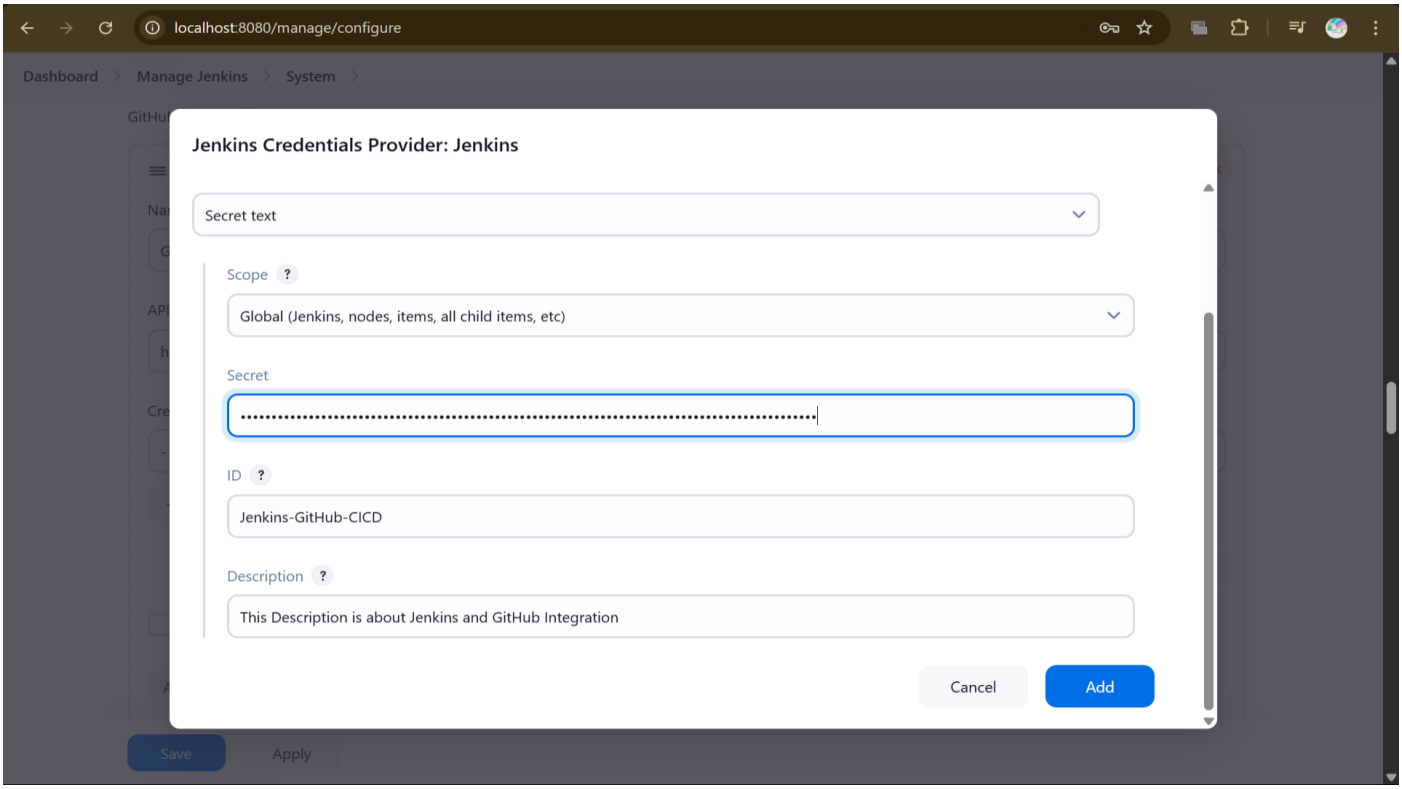


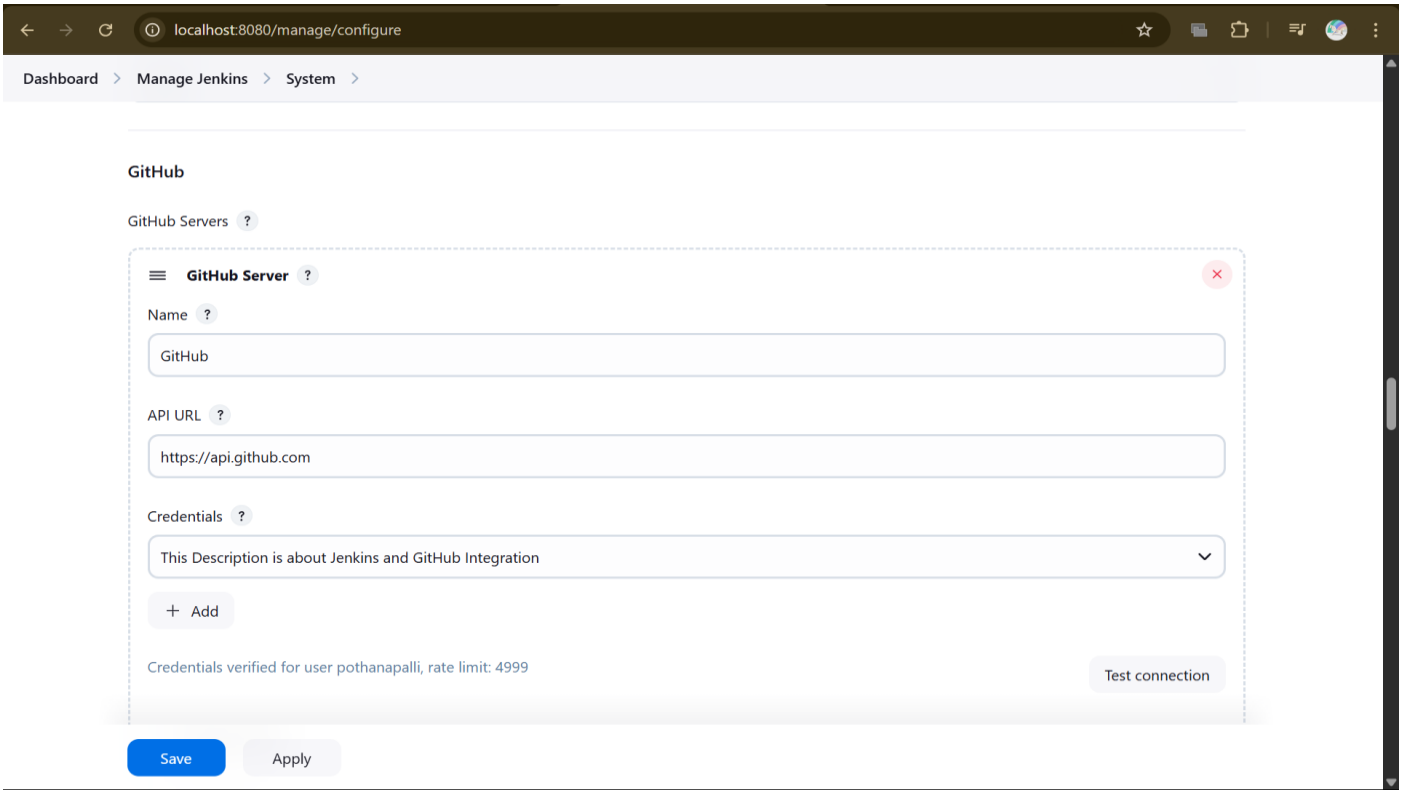


3.2. Integration of Jenkins and GitHub

Integrating Jenkins with GitHub is a critical step that allows automated communication between the source control system and the CI/CD server. Once the PAT is created, it is configured in Jenkins through the "Manage Credentials" section. After securely storing the token, a connection is established between Jenkins and the GitHub repository. This setup enables Jenkins to continuously monitor the repository for changes such as code commits or pull requests. Whenever a new change is detected, Jenkins can trigger automated builds, tests, and deployments based on the defined pipeline configuration. This tight integration ensures a continuous feedback loop, where developers receive immediate insights into the health and readiness of their codebase. It fosters a proactive development environment where issues are identified and resolved early in the development cycle.


Dashboard > Manage Jenkins > System >
GitHub Servers
GitHub Server
Name
GitHub
API URL
https://api.github.com
Credentials
- none -
Add
Test connection
Manage hooks
Advanced
Save
Apply


localhost:8080/manage/configure
Dashboard > Manage Jenkins > System >
Jenkins Credentials Provider: Jenkins
Secret text
Scope
Global (Jenkins, nodes, items, all child items, etc)
Secret
ID
Jenkins-GitHub-CICD
Description
This Description is about Jenkins and GitHub Integration
Cancel
Add
Save
Apply



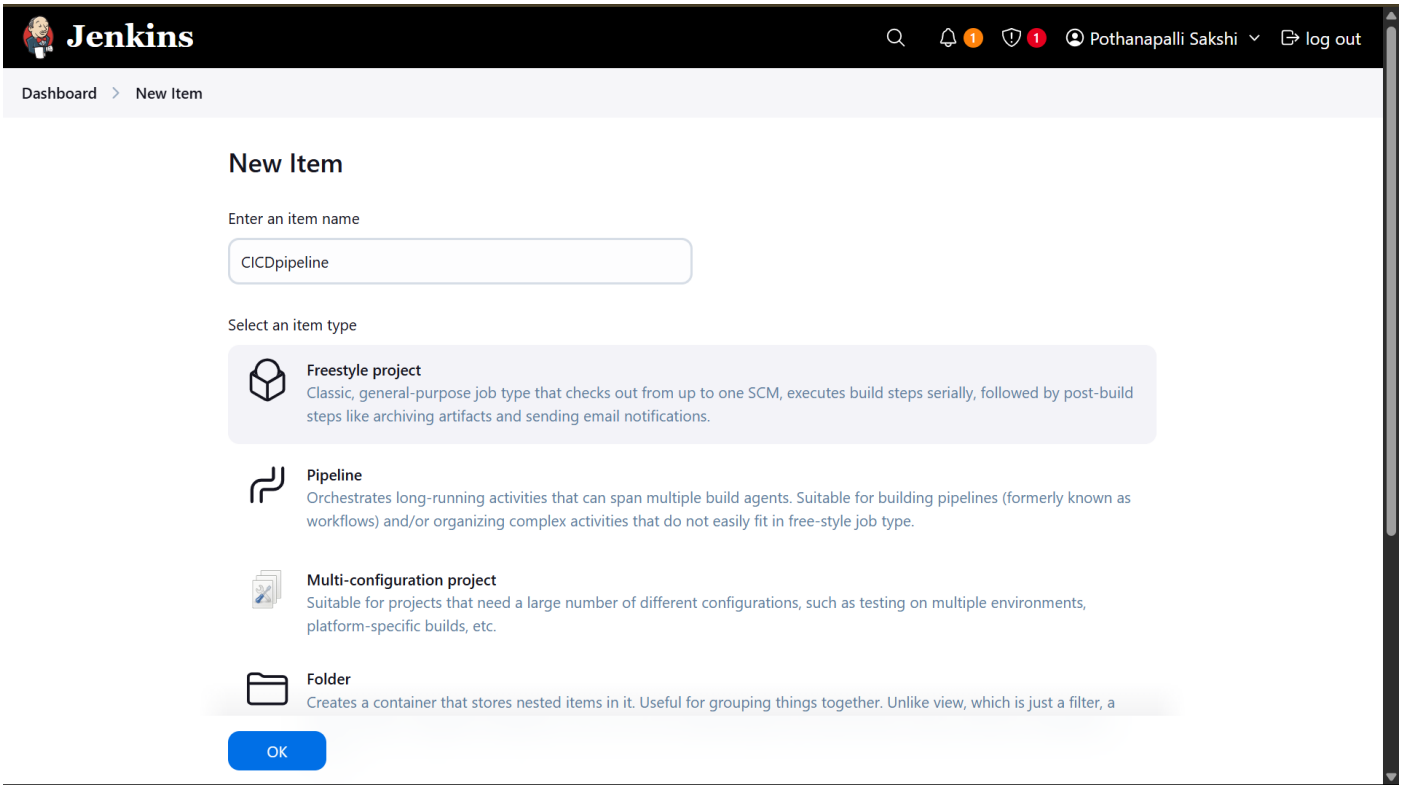
3.3. CI/CD Pipeline

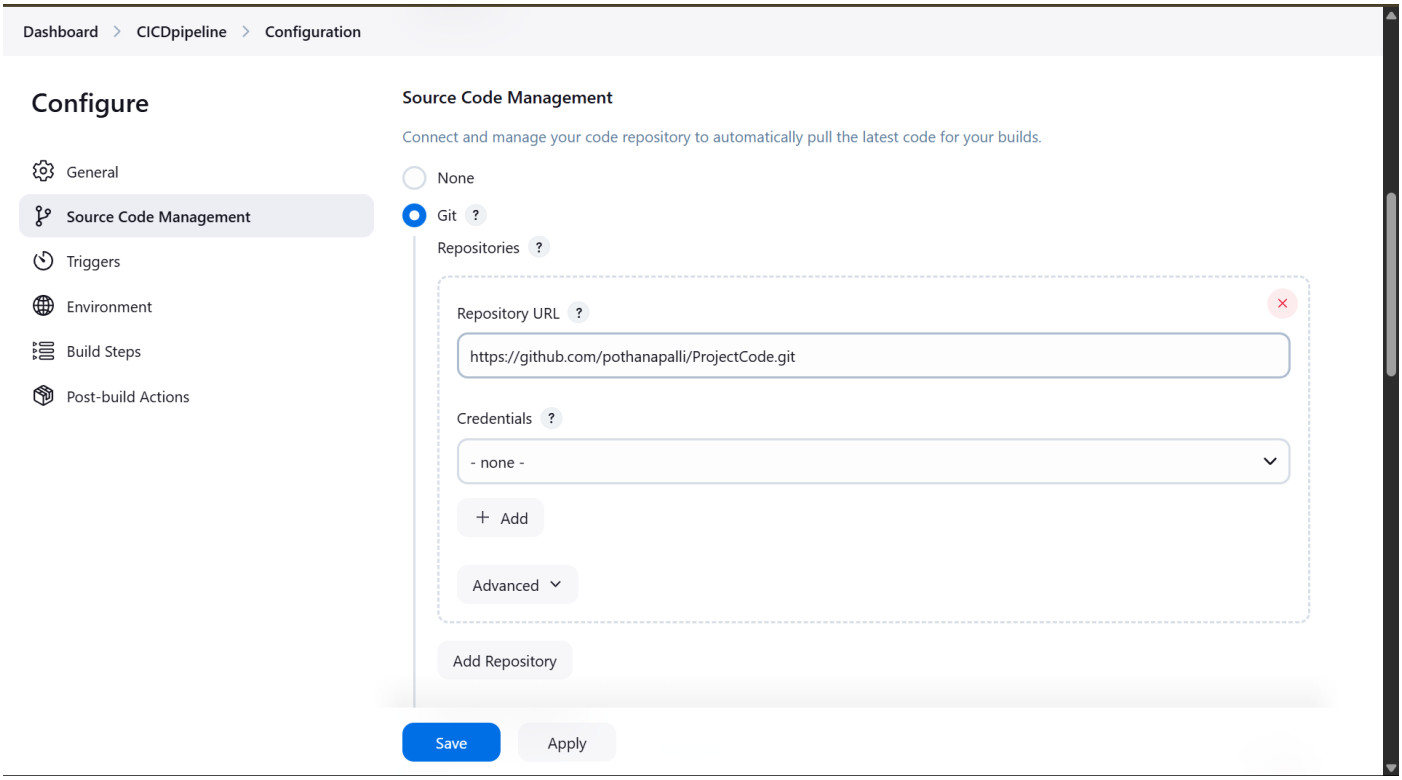
The CI/CD pipeline is the heart of the automation process. It defines a sequence of tasks that Jenkins executes automatically to ensure that software changes are thoroughly tested and deployed without manual intervention. The pipeline typically consists of several stages, including **Checkout**, **Build**, **Test**, and **Deploy**.

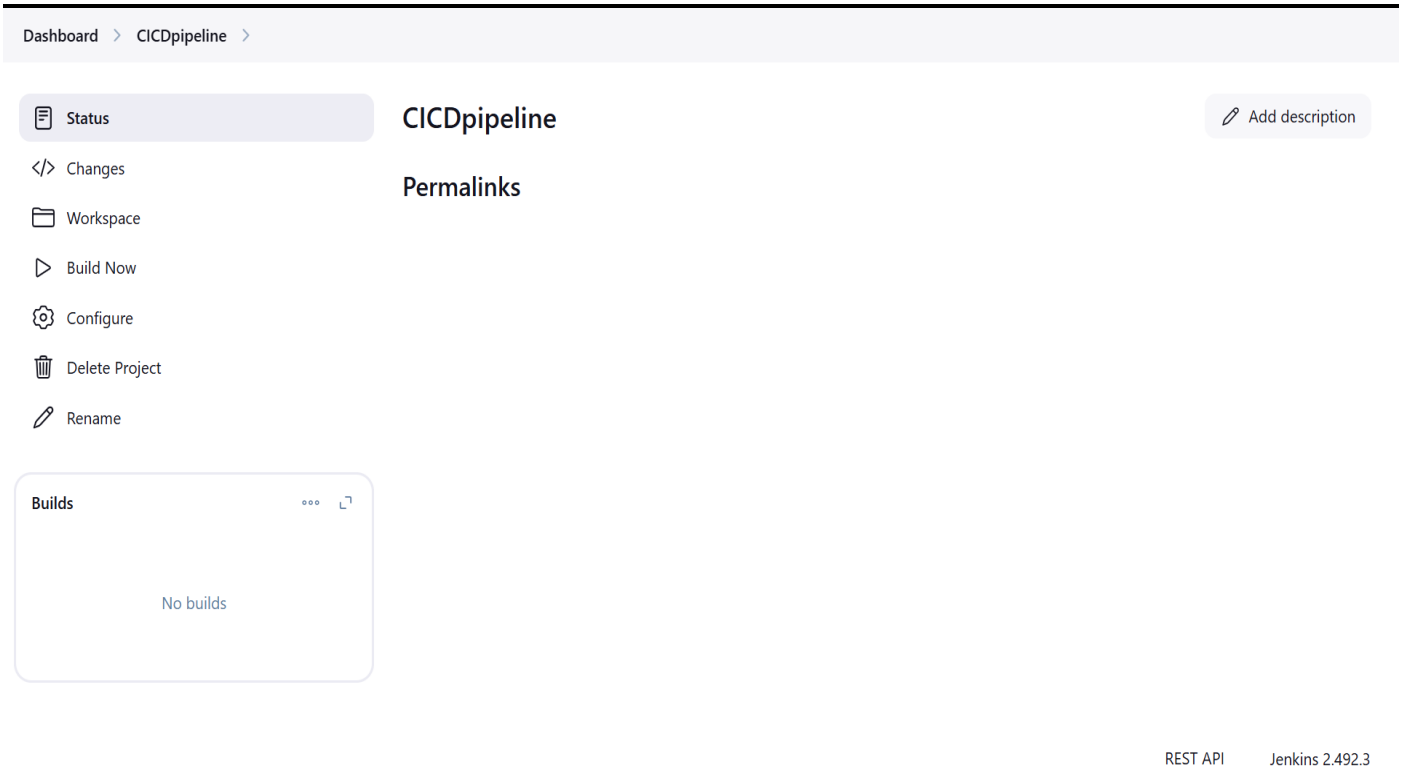
- In the **Checkout** stage, Jenkins pulls the latest code from the specified GitHub repository.
- During the **Build** stage, the application is compiled or packaged using the appropriate tools and dependencies.
- The **Test** stage involves running automated unit tests, integration tests, or other quality checks to validate the code.
- Finally, in the **Deploy** stage, the application is containerized using Docker and deployed to a staging or production environment.

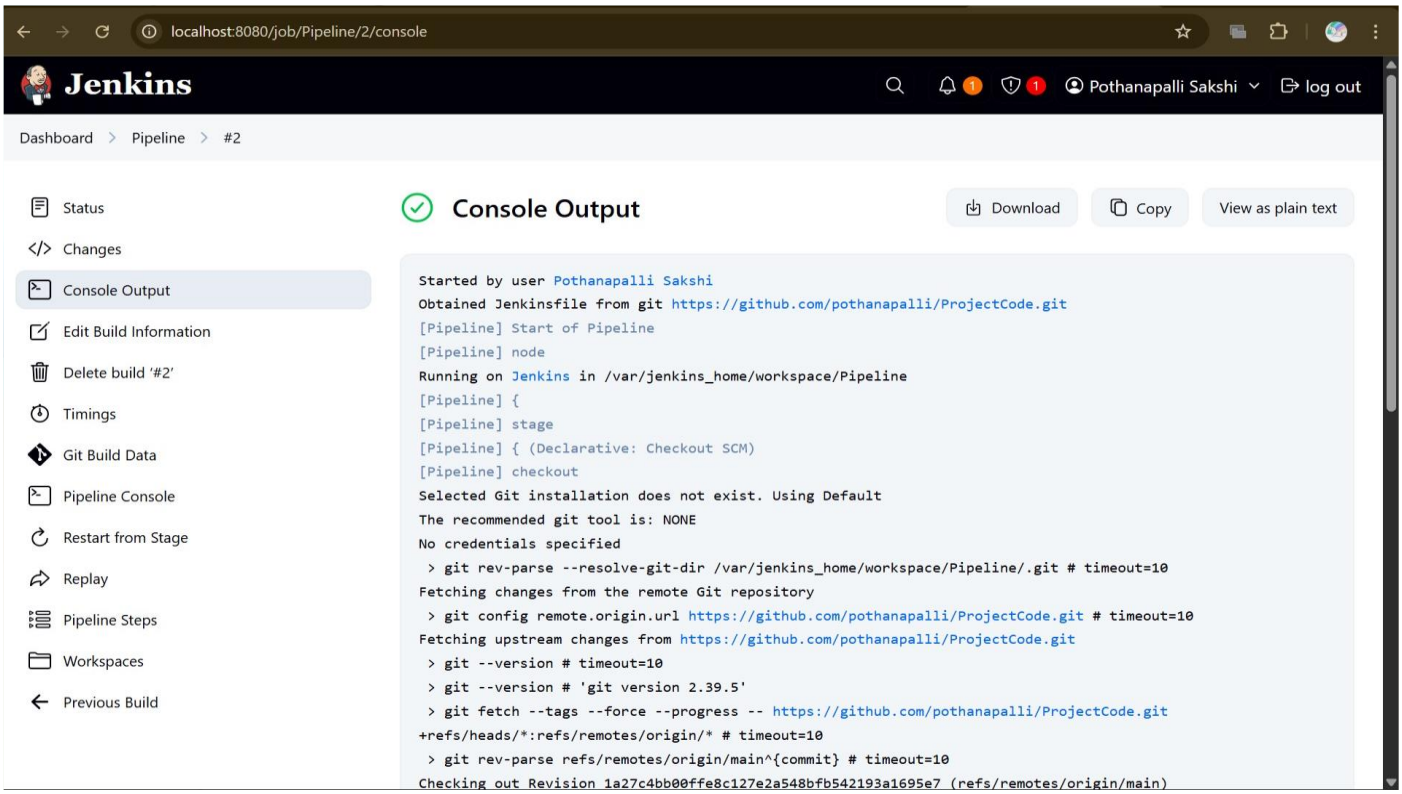
This structured approach provides consistency across development, testing, and production environments. It reduces manual overhead,

shortens feedback loops, and ensures that every change is delivered in a reliable and repeatable manner.









4. Tools Used

This project leverages several powerful tools that are widely used in modern DevOps workflows:

- GitHub: A distributed version control system used for managing source code, tracking changes, and collaborating with other developers.
- Jenkins: An open-source automation server that orchestrates the CI/CD workflow. It supports plugins and configurations for a wide variety of use cases, including GitHub integration and Docker builds.
- Docker: A containerization platform that packages applications and their dependencies into portable containers. Docker ensures consistent behavior across environments, making deployment easier and more efficient.

Each tool plays a crucial role in building a robust and scalable software delivery pipeline, and their integration provides a seamless end-to-end DevOps experience.

5. Final Summary

The Jenkins CI/CD Deployment Hub project demonstrates the power and efficiency of automating software development workflows. By creating a secure and seamless connection between GitHub and Jenkins through Personal Access Tokens, the project lays a strong foundation for continuous integration. The integration allows Jenkins to monitor repositories and trigger automated workflows in response to code changes. With the implementation of a CI/CD pipeline, the process of building, testing, and deploying code becomes hands-free and reliable. This not only accelerates software delivery but also improves code quality and developer productivity. The use of Docker further enhances the deployment process by ensuring that applications run consistently across various environments. Overall, the project

showcases how modern DevOps tools and practices can revolutionize the way software is built and delivered.

6.Conclusion

Through the development and deployment of this Jenkins CI/CD hub, I gained a deep understanding of key DevOps concepts and tools. The hands-on experience of setting up Jenkins, configuring GitHub integrations, managing secure tokens, and building a CI/CD pipeline has significantly enhanced my technical skill set. This project highlights the practical benefits of automation in software engineering, such as improved efficiency, reduced errors, and faster delivery cycles. It also reinforces the importance of secure and scalable integration between development and deployment systems. I believe that these skills will be highly valuable in real-world projects and will support my future growth as a software engineer or DevOps practitioner.