



L OVELY  
P ROFESSIONAL  
U NIVERSITY

## **DevOps Project REPORT**

**ON**

### **Jenkins CI/CD Deployment Hub**

**SUBMITTED BY:**

**NAME: POTHANAPALLI SAKSHI**

**REGISTRATION NO: 12219299**

**Section: KO022**

**Roll No: 33**

**SUBMITTED 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.

The screenshot shows the GitHub developer settings page for creating a new personal access token. The left sidebar has 'Personal access tokens' selected under 'Fine-grained tokens'. The main area is titled 'New fine-grained personal access token' and contains fields for 'Token name' (set to 'JenkinsCI/CD'), 'Description' (containing 'Jenkins CI/CD Deployment Hub' and a numbered list: 1. Personal Access Token, 2. Jenkins and GitHub Integration, 3. CI/CD Pipeline), 'Resource owner' (set to 'pothanapalli'), and 'Expiration' (set to '30 days (May 28, 2025)'). A note at the bottom states 'The token will only be able to make changes to resources owned by the selected resource owner. Tokens can always read all public repositories.'

The screenshot shows the GitHub 'Personal access tokens' page. On the left sidebar, under 'Personal access tokens', 'Fine-grained tokens' is selected. A single token named 'JenkinsCI/CD' is listed. It was never used and expires on May 28, 2025. A prominent green callout box contains the message: 'Make sure to copy your personal access token now as you will not be able to see this again.' Below the token details is a code block containing the token value: `github_pat_118BN3TQFY0scCc1Mbb27Vmq_mDaYGs0qVFTxnrxdJHiPzTfn1cs8rBeCwteMkfChfNTQRB534GGjmAQF`. There is also a copy icon next to the code block.

## 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

This screenshot shows the 'GitHub Server' configuration page in Jenkins. The 'Name' field is set to 'GitHub'. The 'API URL' field contains 'https://api.github.com'. The 'Credentials' dropdown is currently empty ('- none -'). There is a 'Manage hooks' checkbox and an 'Advanced' dropdown. At the bottom are 'Save' and 'Apply' buttons.

localhost:8080/manage/configure

Dashboard > Manage Jenkins > System >

Jenkins Credentials Provider: Jenkins

Secret text

Scope ? Global (Jenkins, nodes, items, all child items, etc)

Secret

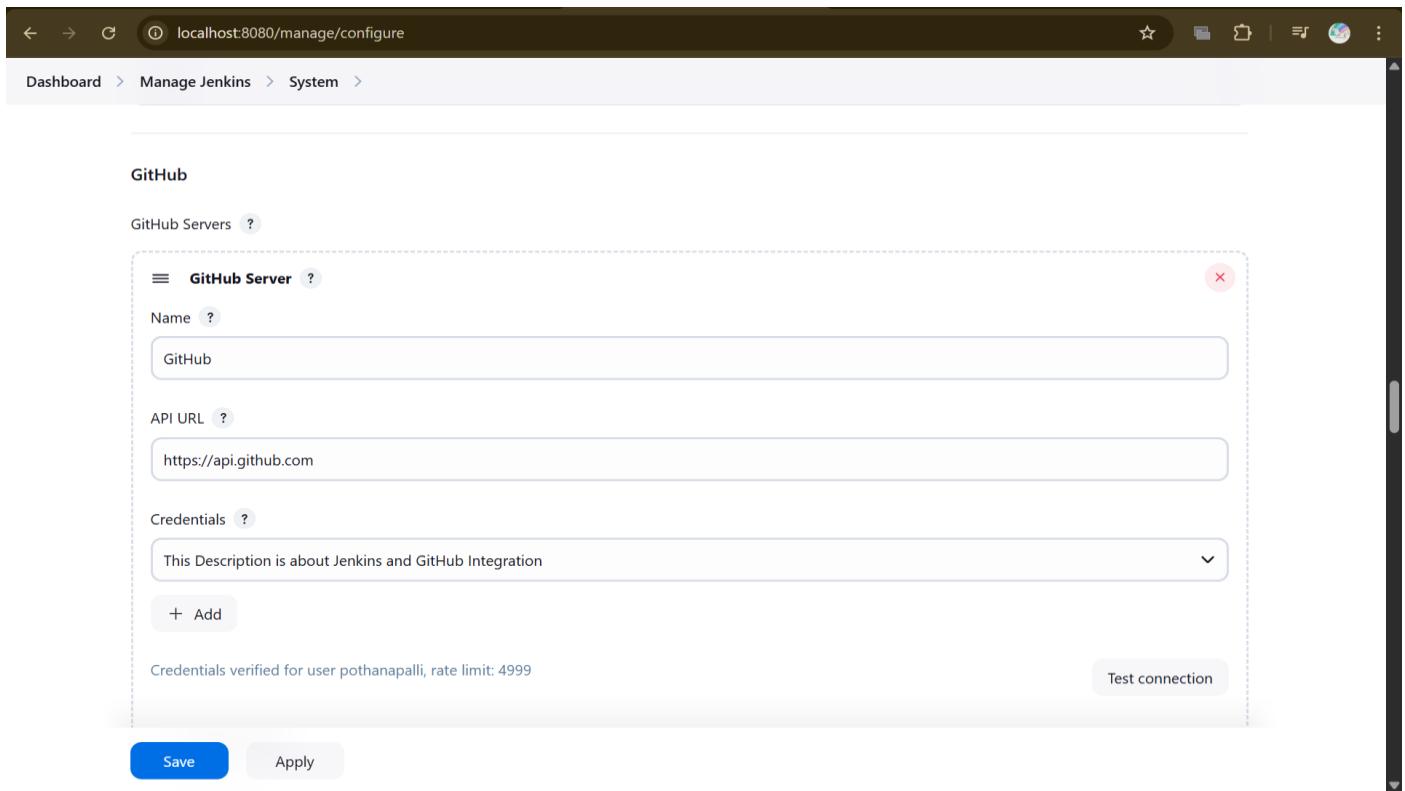
ID ? Jenkins-GitHub-CICD

Description ? This Description is about Jenkins and GitHub Integration

Cancel Add

Save Apply

This screenshot shows the 'Jenkins Credentials Provider: Jenkins' dialog. It includes fields for 'Secret text', 'Scope' (set to 'Global'), 'Secret' (containing redacted text), 'ID' (set to 'Jenkins-GitHub-CICD'), and 'Description' (set to 'This Description is about Jenkins and GitHub Integration'). At the bottom are 'Cancel' and 'Add' buttons.



### 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.

The screenshot shows the Jenkins 'New Item' creation interface. At the top, there is a navigation bar with a user icon, the word 'Jenkins', and a search bar. Below the navigation bar, the path 'Dashboard > New Item' is shown. The main title 'New Item' is centered above a form. The first field is 'Enter an item name', which contains 'CICDpipeline'. The next section is 'Select an item type', which lists four options: 'Freestyle project', 'Pipeline', 'Multi-configuration project', and 'Folder'. The 'Freestyle project' option is highlighted with a light gray background. Below each option is a brief description. At the bottom of this section is a blue 'OK' button. A vertical scrollbar is visible on the right side of the page.

The screenshot shows the Jenkins configuration interface for the 'CICDpipeline' item. The top navigation bar shows the path 'Dashboard > CICDpipeline > Configuration'. On the left, there is a sidebar with several tabs: 'General', 'Source Code Management' (which is selected and highlighted in blue), 'Triggers', 'Environment', 'Build Steps', and 'Post-build Actions'. The main content area is titled 'Source Code Management' and contains instructions: 'Connect and manage your code repository to automatically pull the latest code for your builds.' Below this, there are two radio button options: 'None' and 'Git'. The 'Git' option is selected and has a question mark icon next to it. Under the 'Git' section, there is a 'Repositories' section with a 'Repository URL' input field containing 'https://github.com/pothanapalli/ProjectCode.git'. There is also a 'Credentials' dropdown menu set to '- none -' and a '+ Add' button. At the bottom of the 'Repositories' section is an 'Advanced' dropdown menu. At the very bottom of the configuration page are 'Save' and 'Apply' buttons.

Status **CICDpipeline** Add description

Changes Build Now

Configure Delete Project

Rename

**Builds**

No builds

REST API Jenkins 2.492.3

localhost:8080/job/Pipeline/2/console

**Jenkins** Pothanapalli Sakshi log out

Dashboard > Pipeline > #2

Status Console Output Download Copy View as plain text

Changes Edit Build Information Delete build '#2' Timings Git Build Data Pipeline Console Restart from Stage Replay Pipeline Steps Workspaces Previous Build

**Console Output**

```

Started by user Pothanapalli Sakshi
Obtained Jenkinsfile from git https://github.com/pothanapalli/ProjectCode.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/jenkins_home/workspace/Pipeline/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/pothanapalli/ProjectCode.git # timeout=10
Fetching upstream changes from https://github.com/pothanapalli/ProjectCode.git
> git --version # timeout=10
> git --version # 'git version 2.39.5'
> git fetch --tags --force --progress -- https://github.com/pothanapalli/ProjectCode.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 1a27c4bb0ffe8c127e2a548bfb542193a1695e7 (refs/remotes/origin/main)

```

## 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.