Experiment 03

Aim:

To implement Continuous Integration (CI) using Jenkins by automating the build and test process for a Java project with Maven, triggered on every code change via GitHub.

Theory:

In modern software development, speed, stability, and collaboration are crucial. This is where **DevOps** practices like **Continuous Integration (CI)** come into play.

Continuous Integration (CI) is a software development practice where developers frequently merge their code changes into a central repository. Each integration is verified by an automated build and test process, allowing teams to detect problems early and improve the software's quality and delivery speed.

Jenkins is an open-source CI tool written in Java. It acts as an automation server that supports building, testing, and deploying software projects. Jenkins supports plugins that integrate with virtually every tool in the DevOps toolchain, from version control systems like Git to build tools like Maven and Gradle.

Maven is a build automation and dependency management tool for Java projects. It uses a file named pom.xml (Project Object Model) to manage project dependencies, build configuration, testing, and packaging.

GitHub is a popular web-based hosting service for version control using Git. By integrating GitHub with Jenkins, every code push or pull request can automatically trigger Jenkins to:

- Pull the updated source code
- Build the application using Maven
- Run test cases
- Generate reports
- Deploy artifacts

This experiment explores how Jenkins connects with GitHub and Maven to automate the software development lifecycle. It eliminates manual processes, reduces integration issues, and improves code stability.

Advantages of Jenkins CI Integration:

- Early detection of bugs and integration conflicts
- Reduced manual intervention and errors

- Real-time feedback on code quality
- Faster development and deployment cycles
- Supports scalable and repeatable processes across teams

REQUIREMENTS

- Java JDK (version 11 or above)
- Apache Maven
- Jenkins WAR package
- Git and GitHub repository
- GitHub Personal Access Token (for private repositories)
- Web browser
- Optional: ngrok (to expose localhost for webhook)

Procedure:

Part A: Environment and Tool Setup

1. **Install Java JDK**
   - Verify Java: java -version

```
D:\Sem VII\DEVOPS\Experiment 03>java -version
java version "23.0.2" 2025-01-21
Java(TM) SE Runtime Environment (build 23.0.2+7-58)
Java HotSpot(TM) 64-Bit Server VM (build 23.0.2+7-58, mixed mode, sharing)
```

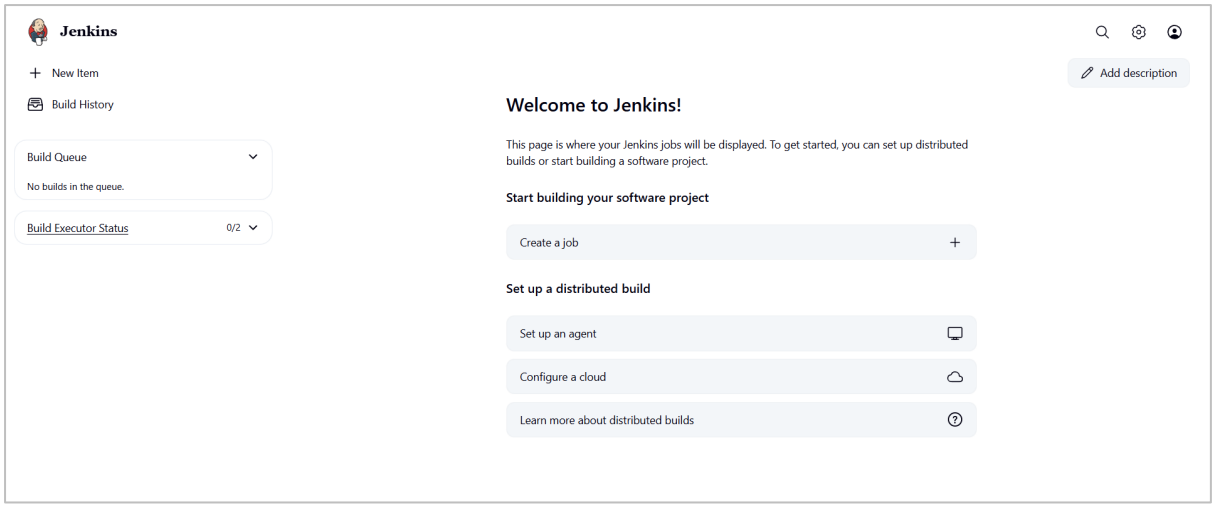
2. **Install Jenkins (WAR method)**
   - Download Jenkins WAR:

```
>> wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war
>> java -jar jenkins.war
```

   - Open in browser: http://localhost:8080
   - Unlock using initial password:

```
>> cat ~/.jenkins/secrets/initialAdminPassword
```

   - Install Suggested Plugins
   - Create an admin user



3. Install Apache Maven

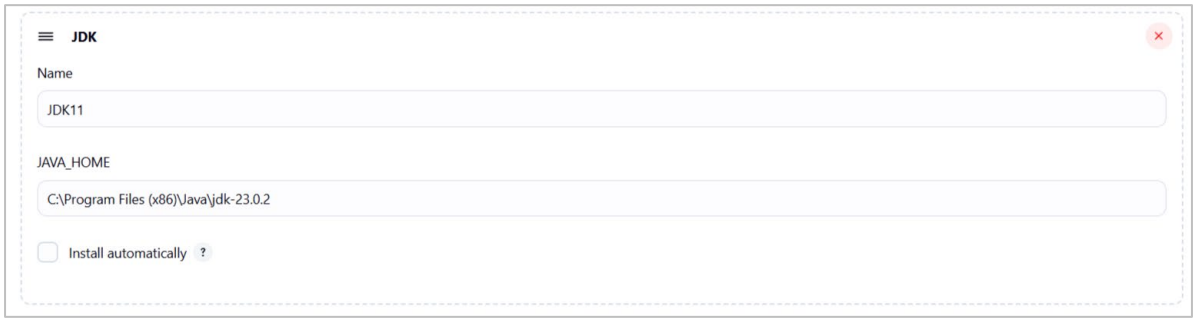
- Download and unzip from: https://maven.apache.org/download.cgi
- Set MAVEN_HOME and update system PATH

```
>> mvn -v
```

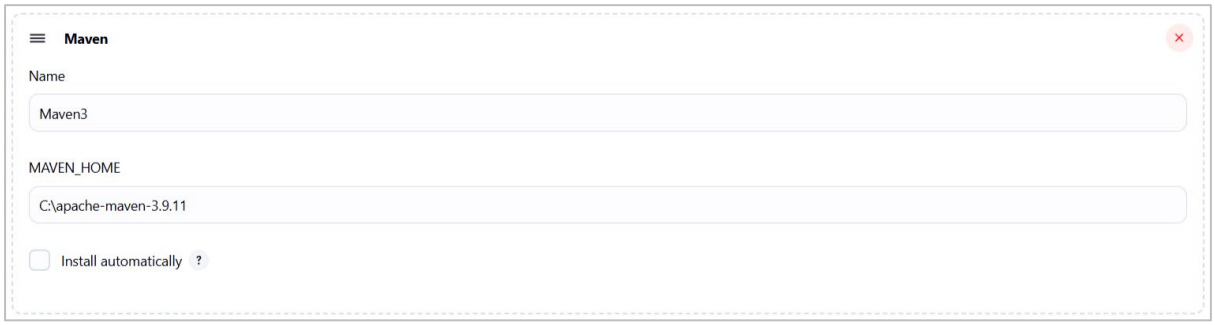
Part B: Configure Jenkins Tools

1. Global Tool Configuration

- Jenkins Dashboard → Manage Jenkins → Global Tool Configuration
- Add Java:
  - Name: JDK11
  - Uncheck “Install automatically”
  - Provide local JDK path



- Add Maven:
  - Name: Maven3
  - Uncheck “Install automatically”
  - Provide path from which mvn or Maven folder



2. Install Required Plugins

- Go to: Manage Jenkins → Plugin Manager → Available
- Install the following plugins:
  - Git Plugin
  - Maven Integration Plugin
  - GitHub Integration Plugin
  - JUnit Plugin
  - Pipeline Plugin (optional)
- Restart Jenkins if prompted

Part C: Connect Jenkins to GitHub

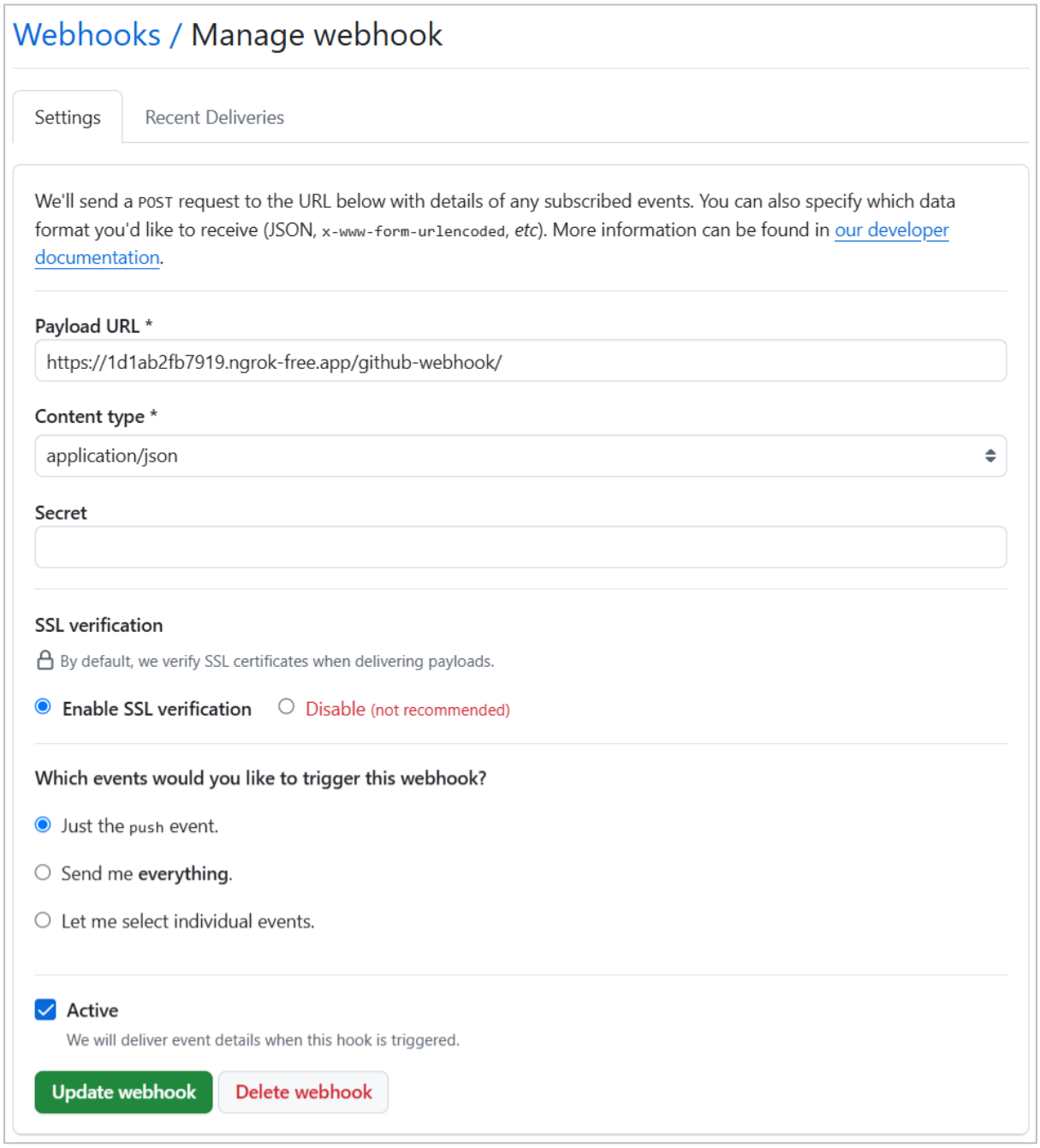
1. Fork or Create Maven Project

- Use a sample project like Spring PetClinic or create your own Java project with:
  - pom.xml
  - Source and test files

2. Set Up GitHub Webhook

- GitHub → Repository Settings → Webhooks → Add Webhook
  - Payload URL: http://<your-ip>:8080/github-webhook/
  - Content type: application/json
  - Event: Push events
- If Jenkins is hosted locally, use ngrok to expose:

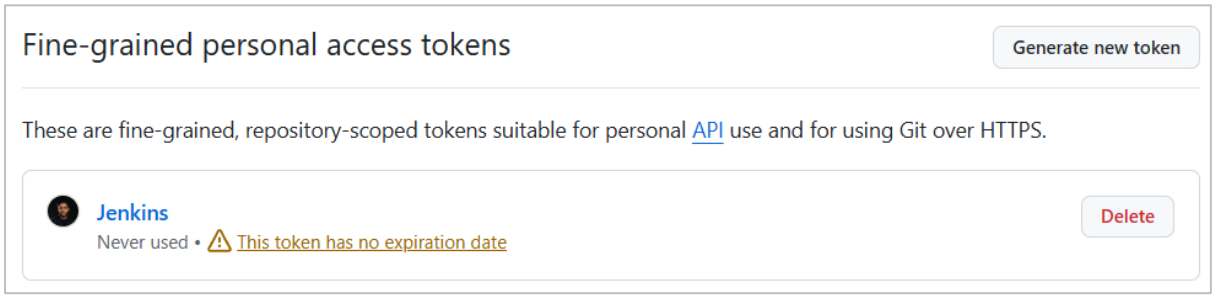
```
>> ngrok http 8080
```

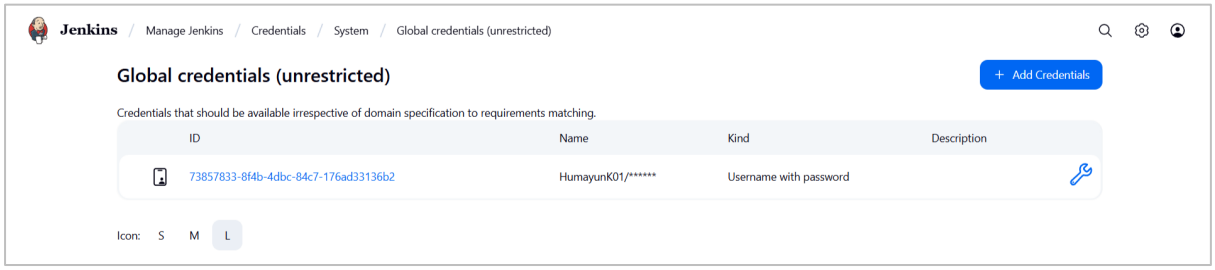


3. Add GitHub Credentials in Jenkins

- Jenkins Dashboard → Manage Jenkins → Credentials → (Global) → Add Credentials
- Type: Username with Password
  - Username: GitHub username
  - Password: GitHub Personal Access Token
- Generate a token from:
  - GitHub → Settings → Developer Settings → Personal Access Tokens

- Scope: repo, workflow

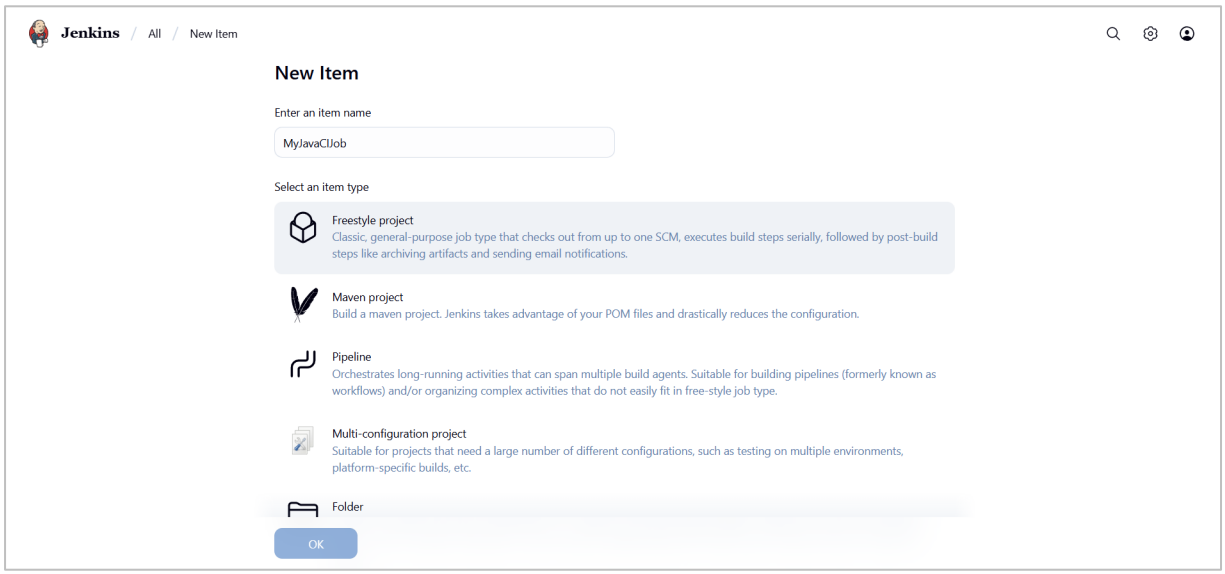




Part D: Create Jenkins Build Job

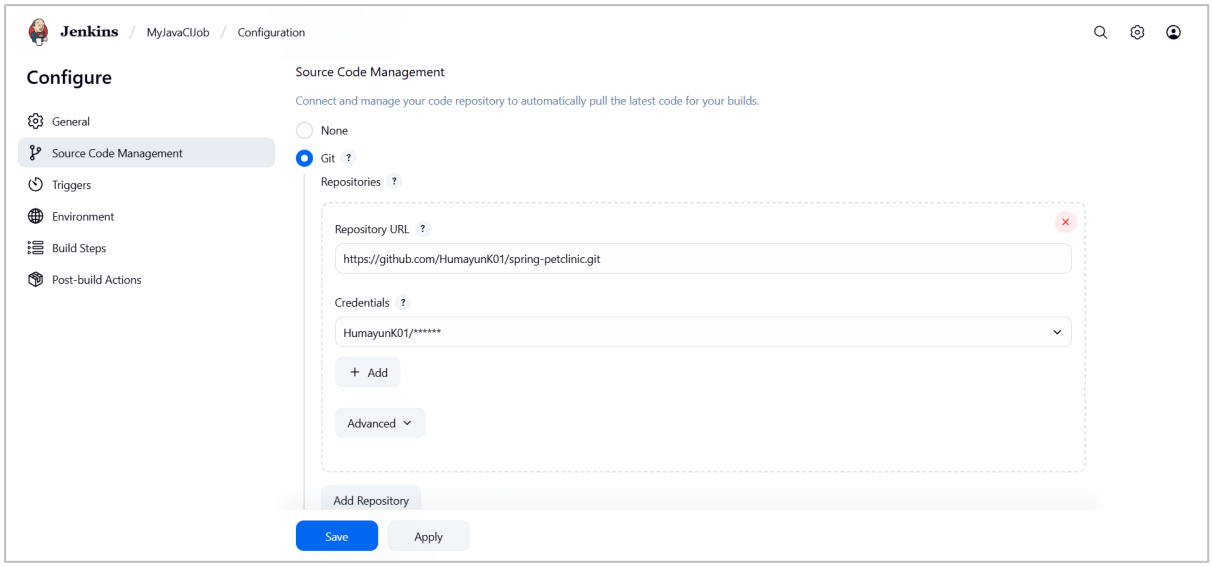
1. Create Freestyle Project

- Jenkins Dashboard → New Item → Name: MyJavaCIJob
- Select: Freestyle Project → OK



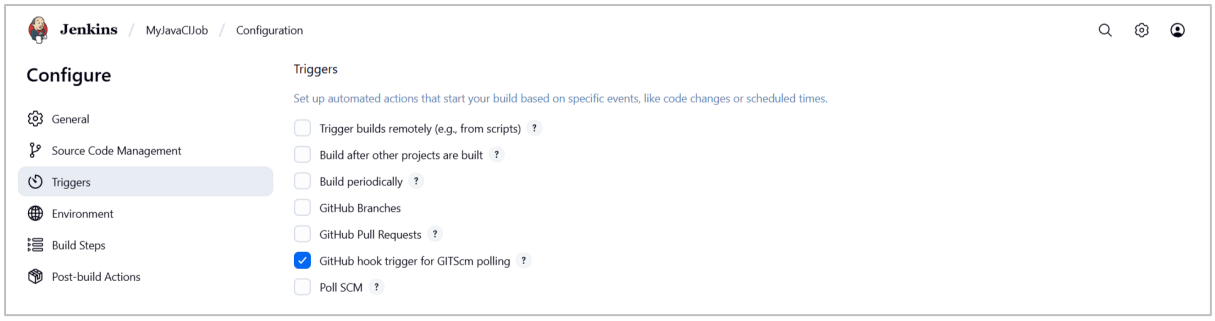
2. Configure Source Code Management

- Select Git
- Provide GitHub repository URL
- https://github.com/HumayunK01/spring-petclinic.git
- Add credentials (for private repos)



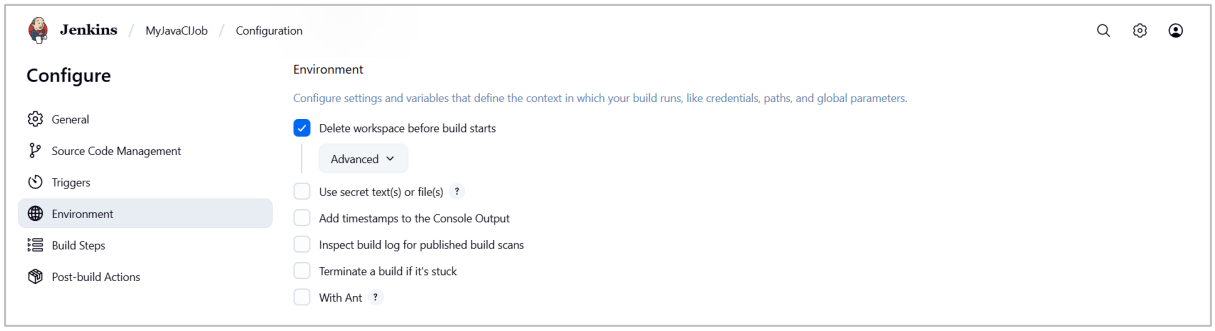
3. Set Build Trigger

- Enable: GitHub hook trigger for GITScm polling



4. Configure Build Environment

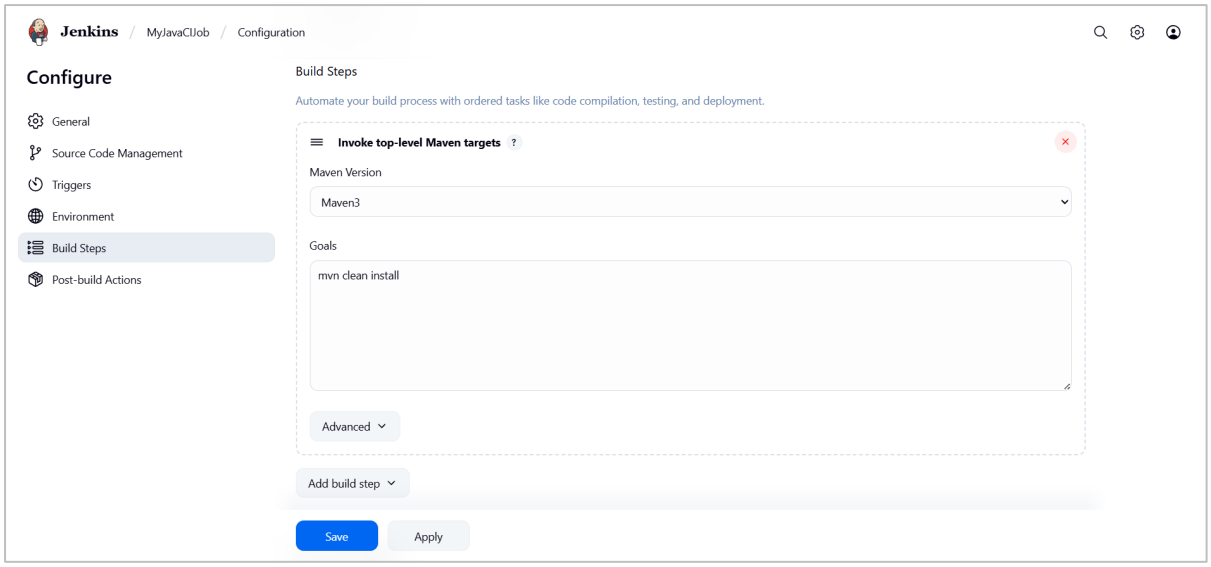
- Optionally enable: Delete workspace before build starts



5. Add Build Steps

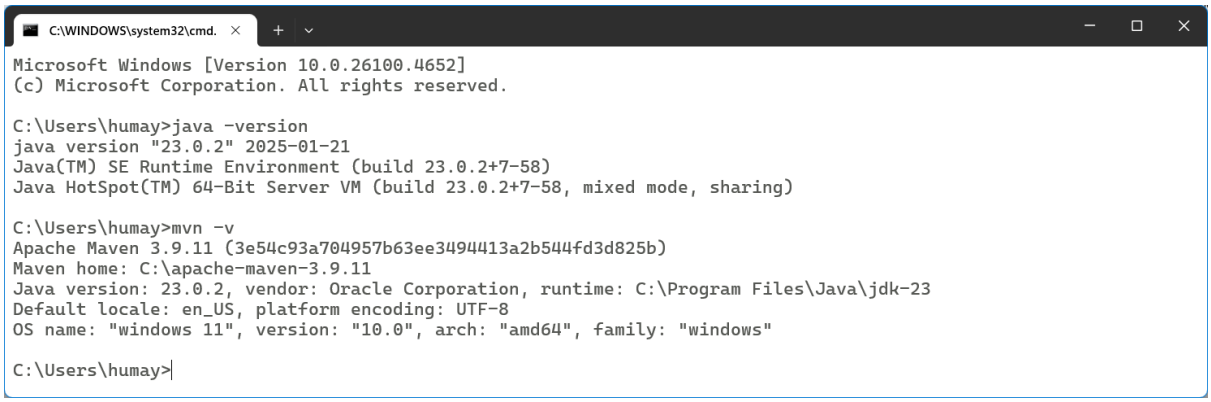
- Build Step: Invoke Maven Targets
  - Command:

>> mvn clean install



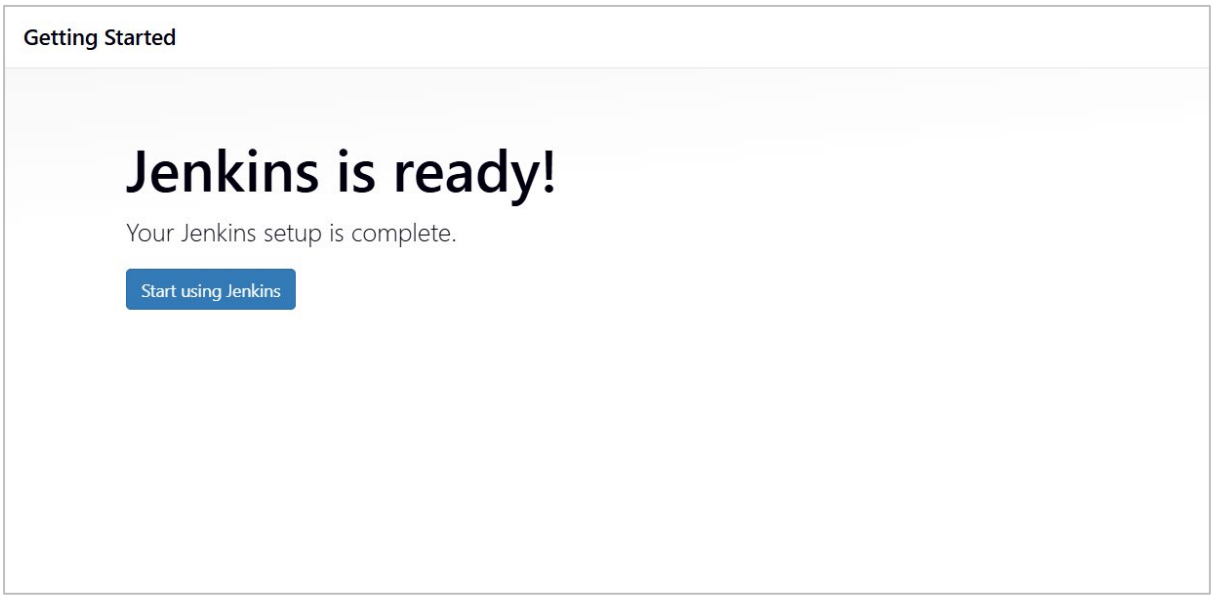
Output:

- Java and Maven installation

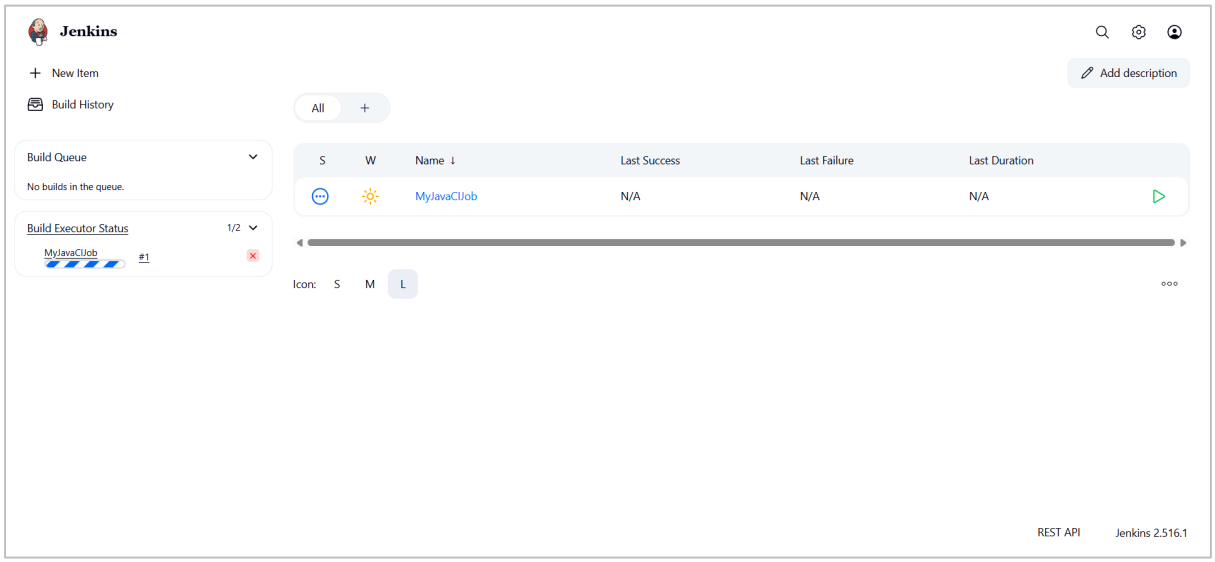


```
C:\WINDOWS\system32\cmd.
Microsoft Windows [Version 10.0.26100.4652]
(c) Microsoft Corporation. All rights reserved.

C:\Users\humay>java -version
java version "23.0.2" 2025-01-21
Java(TM) SE Runtime Environment (build 23.0.2+7-58)
Java HotSpot(TM) 64-Bit Server VM (build 23.0.2+7-58, mixed mode, sharing)

C:\Users\humay>mvn -v
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
Maven home: C:\apache-maven-3.9.11
Java version: 23.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-23
Default locale: en_US, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\humay>
```

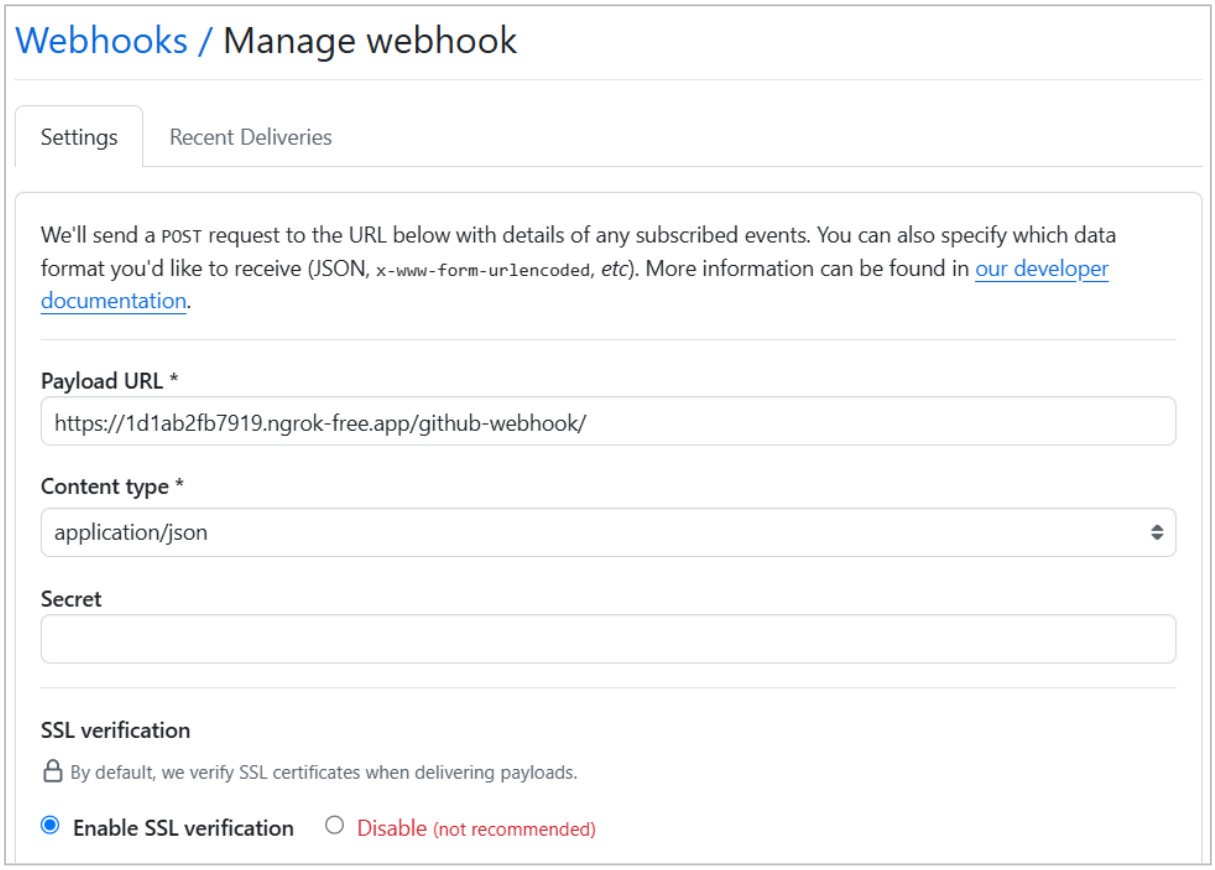
- Jenkins installation and startup

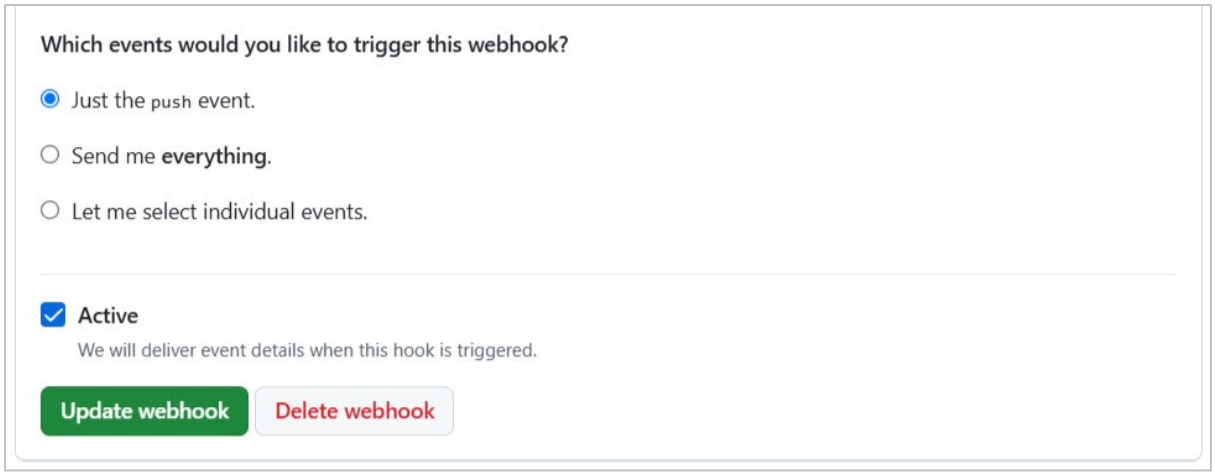


- Jenkins dashboard with configured job

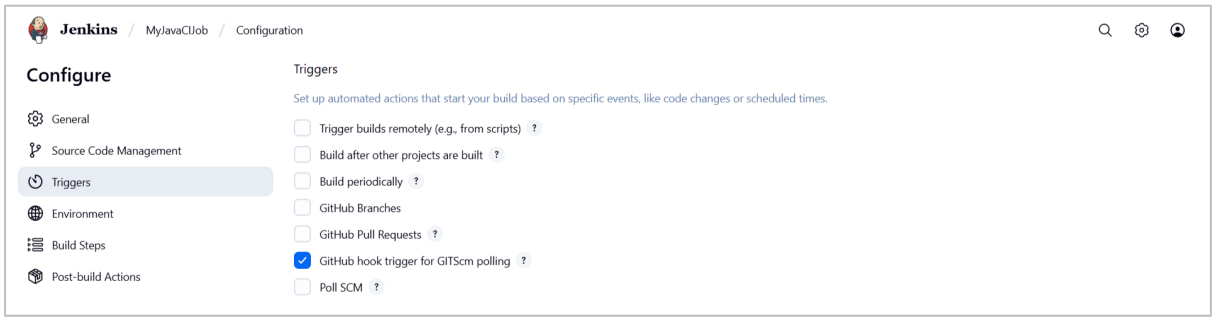


- GitHub webhook settings





- Build trigger and output logs



Conclusion:

This experiment successfully demonstrates the implementation of a continuous integration pipeline using Jenkins, Maven, and GitHub. By configuring Jenkins to automatically respond to code changes in a GitHub repository, developers can automate the build and testing process. This results in faster feedback, early bug detection, reduced integration issues, and improved development productivity. The integration of GitHub, Maven, and Jenkins embodies the core principles of DevOps in practice.