# Experiment 03

**Aim:**

To setup Jenkins environment in installing required libraries including Maven.

---

**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)
```

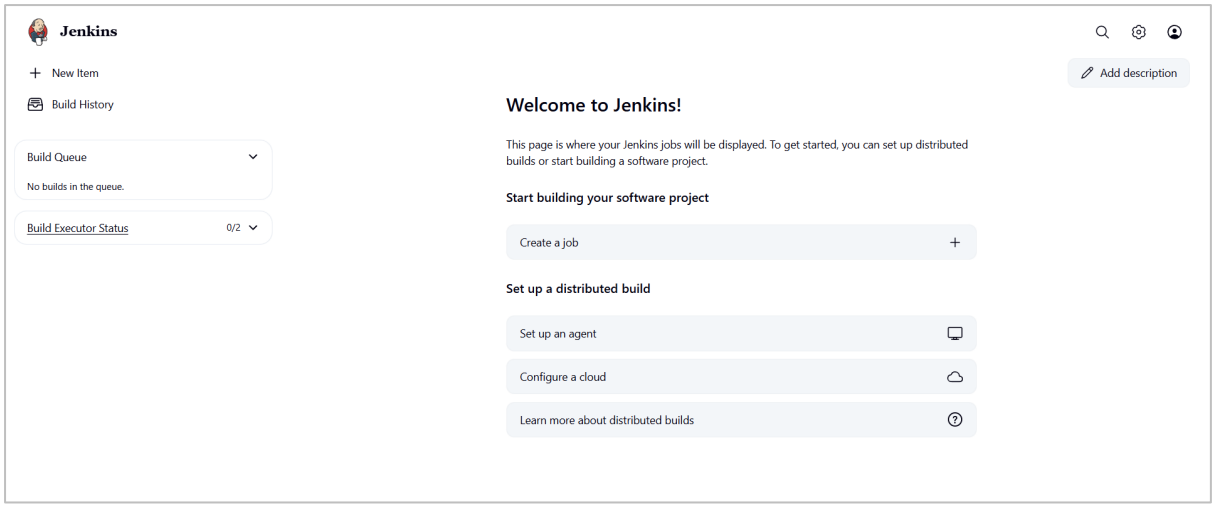
2. **Install Jenkins (WAR method)**
   - Download Jenkins WAR:

```
>> 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

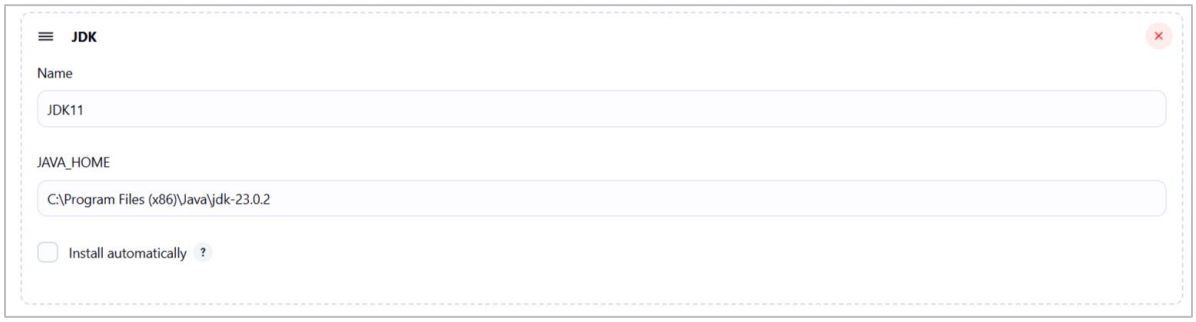
```
>> mvn -v
```


---

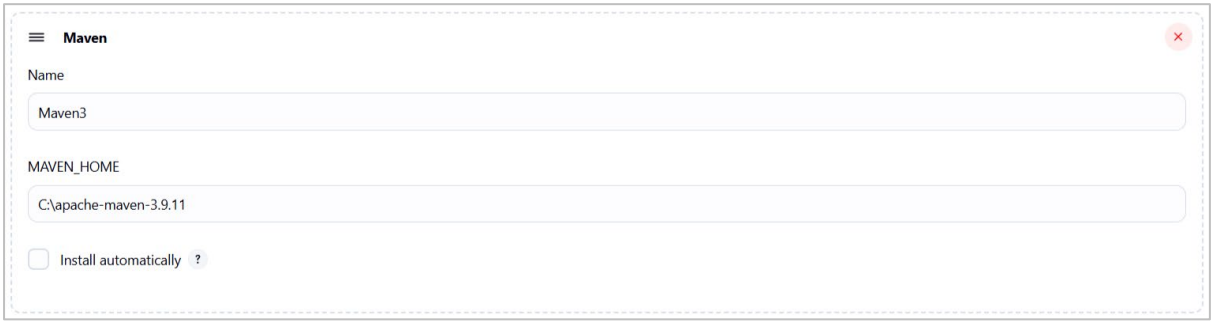
## Part B: Configure Jenkins Tools

### 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



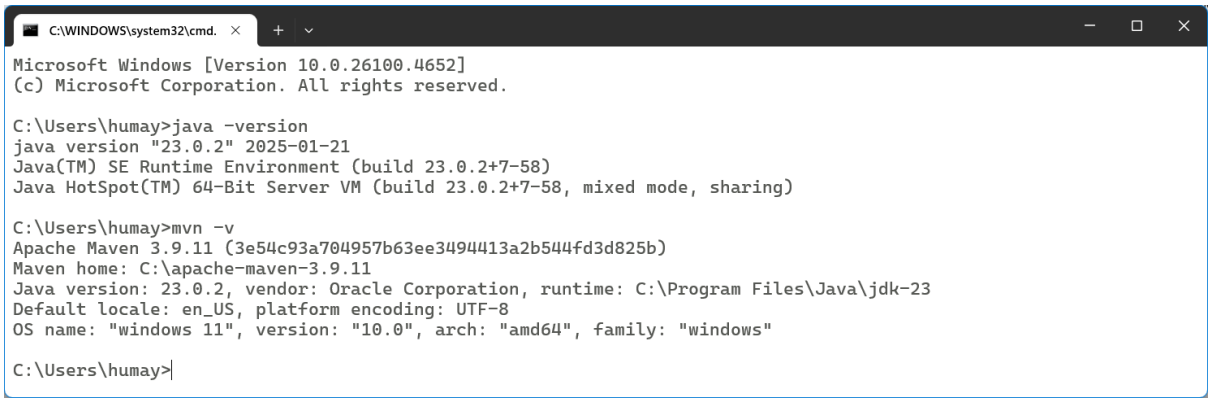

**2. Install Required Plugins**

- 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

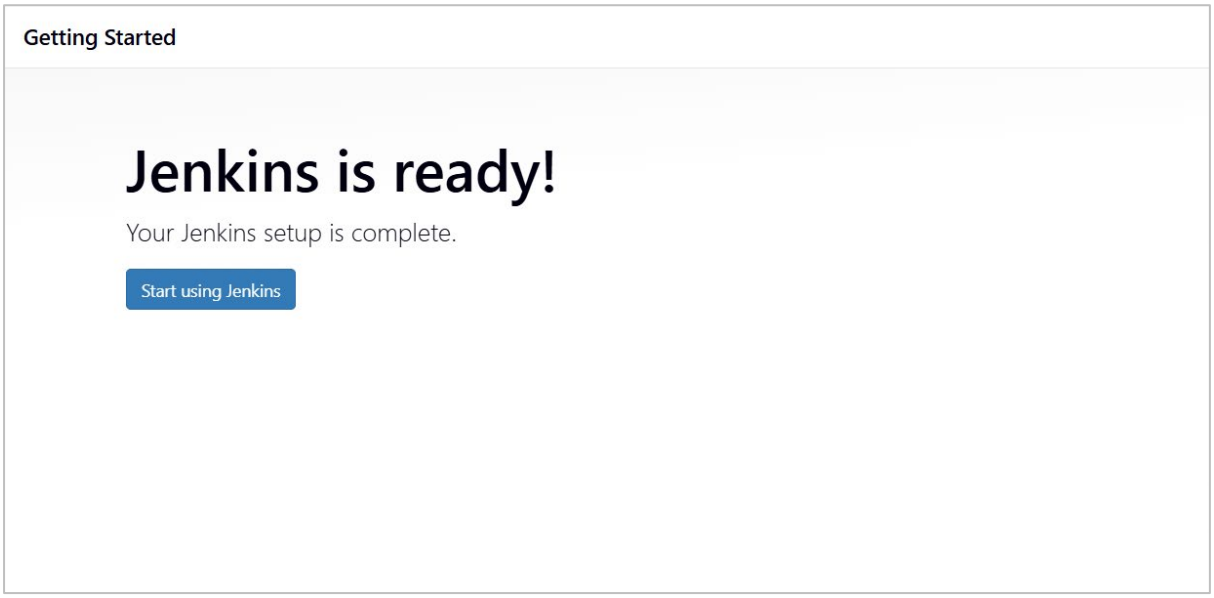
---

**Output:**

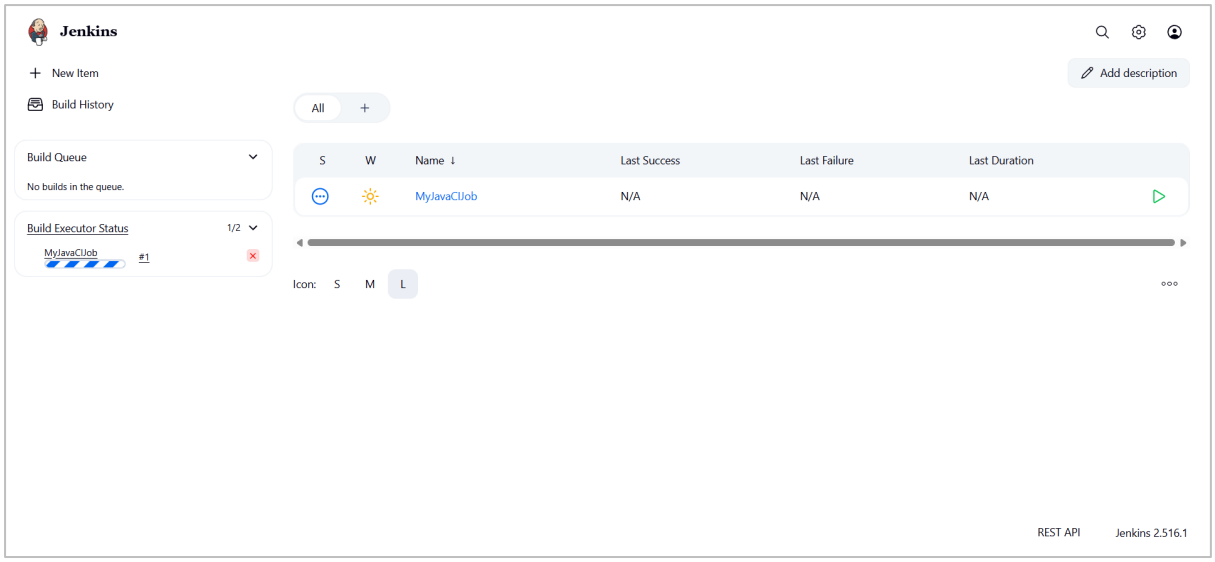
- Java and Maven installation




- Jenkins installation and startup




- Jenkins dashboard with configured job




**Conclusion:**

The setup of the Jenkins environment, along with the installation of required libraries including Maven, was successfully completed. This process established a reliable CI/CD foundation, enabling automated build, test, and deployment workflows. By integrating Maven, the environment now supports efficient project management, dependency handling, and build automation. Overall, the configured system ensures better scalability, maintainability, and faster development cycles for future projects.