# To understand Continuous Integration by installing and configuring Jenkins with Maven to set up a build job."

* Set up an environment to support automated builds and testing.
* Connect a version control system (GitHub) with Jenkins.
* Set up Jenkins jobs that automatically respond to code changes.

**Part A: Environment and Tool Setup**

**Step 1: Install Java JDK**

**Why?**

Jenkins, Maven, and Gradle are Java-based tools and require the JDK to run.

**Check Java Installation**

bash

java -version

This command checks if Java is already installed and prints the current version (should be 11 or higher for compatibility).

**🔸 If Not Installed**

**Windows:**

1. Download from Oracle:  
   <https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>
2. After installation, **set the JAVA\_HOME** environment variable:
   * Go to **System Properties → Environment Variables**
   * Add a new **System Variable**:
     + Name: JAVA\_HOME
     + Value: path to your JDK installation (e.g., C:\Program Files\Java\jdk-11)
   * Add %JAVA\_HOME%\bin to the PATH variable.

**Step 2: Install Jenkins**

**Why?**

Jenkins is the automation server that performs builds, tests, and integrates code changes automatically.

**Recommended Method: WAR File**

bash

wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war

java -jar jenkins.war

This is the easiest setup for students without requiring system-level services.

**🔸 Access Jenkins in Browser:**

Go to:  
http://localhost:8080

**🔸 Unlock Jenkins**

bash

cat ~/.jenkins/secrets/initialAdminPassword

Paste this password into the browser setup wizard.

**🔸 Install Plugins (Suggested Plugins)**

* Jenkins automatically recommends essential plugins (Git, Maven, Pipeline, etc.)
* Select **Install Suggested Plugins** to continue.

**🔸 Create First Admin User**

Fill out:

* Username
* Password
* Full Name
* Email

This account will be used to access the Jenkins dashboard.

**Step 3: Install Build Tools (Maven/Gradle)**

**Why?**

These are **build automation tools** for Java:

* **Maven** uses XML (pom.xml

**🔸 A. Maven Installation**

**Windows:**

1. Download from <https://maven.apache.org/download.cgi>
2. Unzip and set MAVEN\_HOME and add to PATH.

**🔸 Validate Installation**

bash

java -version

mvn -v

Make sure all commands return valid version info. If not, installation is incomplete.

**Part B: Configure Jenkins Tools**

**Step 1: Configure Global Tools**

Go to:  
**Jenkins Dashboard → Manage Jenkins → Global Tool Configuration**

**🔸 Java Configuration**

* Name: JDK11
* Uncheck “Install automatically”
* Provide full JDK path (e.g., /usr/lib/jvm/java-11-openjdk)

Avoid auto-install for consistent environment across machines.

**🔸 Maven Configuration**

* Name: Maven3
* Uncheck “Install automatically”
* Provide Maven path (e.g., /usr/share/maven or output of which mvn)

**Step 2: Install Essential Plugins**

Go to:  
**Manage Jenkins → Plugin Manager → Available Tab**

Search and install:

* Git Plugin (to clone GitHub repos)
* Maven Integration Plugin
* GitHub Integration Plugin (for webhook & SCM link)
* JUnit Plugin (for test reports)
* Pipeline Plugin (optional, for advanced builds)

Restart Jenkins if prompted.

**Part C: Connect Jenkins to GitHub Repository**

**Step 1: Create or Fork Java Project**

**Option A: Use Sample Maven Project**

[e.g., Spring PetClinic](https://github.com/spring-projects/spring-petclinic)

**Option B: Create your own Maven project**

(Ensure you have pom.xml and sample test files)

**Step 2: Configure GitHub Webhook**

Go to your repository on GitHub → **Settings → Webhooks**

1. **Click Add Webhook**
2. Set:
   * **Payload URL**: http://<your-local-ip>:8080/github-webhook/
   * **Content type**: application/json
   * **Events**: Select “Just the push event”
3. Click **Add Webhook**

Use ngrok if Jenkins is running locally and GitHub needs external access:

bash

ngrok http 8080

**Step 3: Add GitHub Credentials in Jenkins**

**(Only needed if the repo is private)**

1. Jenkins → **Manage Jenkins → Credentials → (global) → Add Credentials**
2. Choose:
   * **Kind**: Username with password
   * **Username**: your GitHub username
   * **Password**: your GitHub Personal Access Token (PAT)

**How to Create a PAT**

1. Go to GitHub → Profile → **Settings → Developer Settings → Tokens**
2. Generate a classic token with scopes:
   * repo – full repo access
   * workflow – required for GitHub Actions access
3. Copy and paste the token into Jenkins credentials

**Part D: Create Jenkins Freestyle Project**

**Step 1: Create Job**

1. Go to Jenkins Dashboard
2. Click **New Item**
3. Enter name: MyJavaCIJob
4. Select **Freestyle Project**
5. Click OK

**Step 2: Configure Source Code Management**

Under **Source Code Management**:

* Choose: Git
* Enter the **GitHub repository URL** (e.g., https://github.com/your-name/spring-petclinic.git)
* If repo is private → select credentials from dropdown

**Step 3: Configure Build Triggers**

Check:  
**GitHub hook trigger for GITScm polling**

This allows Jenkins to automatically build the job every time code is pushed to GitHub (via webhook)

**Step 4: Configure Build Environment**

Optionally check:  
 **Delete workspace before build starts**

Ensures that each build starts fresh (no leftover artifacts)

**Step 5: Add Build Steps**

**🔸 A. If using Maven:**

bash

mvn clean install

* clean: removes previous builds
* install: compiles code, runs tests, installs package to local repo

You can also add:

bash

mvn test

To run only unit tests.

**Result:**

* Jenkins pulls latest code on every push
* Compiles and tests the project
* Creates .jar or .war file
* Optionally, displays test reports (if configured)