**MAVEN INSTALLATION**

1. \*\*Download Maven\*\*:

- Go to the [Apache Maven website](https://maven.apache.org/download.cgi) and download the latest version of Maven.

- Choose the binary zip archive (e.g., `apache-maven-3.8.6-bin.zip`), which includes all the essential files.

2. \*\*Install Maven\*\*:

- Extract the downloaded zip file to a directory of your choice (e.g., `C:\Program Files\Apache\Maven`).

- Once extracted, open the folder and note the path. This path is what you’ll use when configuring Maven in Jenkins.

3. \*\*Set Up Environment Variables (for Windows)\*\*:

- Go to \*\*Control Panel\*\* > \*\*System and Security\*\* > \*\*System\*\* > \*\*Advanced system settings\*\* > \*\*Environment Variables\*\*.

- Under \*\*System Variables\*\*, click \*\*New\*\* and add:

- \*\*Variable Name\*\*: `MAVEN\_HOME`

- \*\*Variable Value\*\*: The path where you extracted Maven (e.g., `C:\Program Files\Apache\Maven`)

- Find the \*\*Path\*\* variable in \*\*System Variables\*\*, select it, click \*\*Edit\*\*, and add `%MAVEN\_HOME%\bin` to the list. This lets your system recognize Maven commands.

4. \*\*Verify Maven Installation\*\*:

- Open Command Prompt (or terminal) and type:

```bash

mvn -version

```

- You should see the Maven version info if it’s correctly installed.

### Part 1: Understanding Continuous Integration (CI)

\*\*Purpose\*\*: Continuous Integration helps catch bugs early by automatically building and testing the code each time a developer makes changes. This means that each update to the code triggers a process that checks if everything is working, and this is essential for quality and fast delivery.

### Part 2: Setting Up Jenkins

\*\*Purpose\*\*: Jenkins is an automation server, meaning it’s designed to automate processes like building and testing code. Installing Jenkins will allow you to automate the steps that CI requires.

---

**INSTALL JENKINS**

#### Step 1: Installing Jenkins

1. \*\*Download Jenkins\*\*:

- \*\*What to do\*\*: Go to the [Jenkins website](https://www.jenkins.io/download/), and download the package for your operating system (Windows, macOS, or Linux).

- \*\*Why\*\*: Jenkins is a software application, so you need to download the installation file that’s compatible with your system.

2. \*\*Install Jenkins\*\*:

- \*\*What to do\*\*: Run the downloaded file and follow the installation instructions (like you would with any software).

- \*\*Why\*\*: This step installs Jenkins on your computer, setting it up to run as a server.

3. \*\*Start Jenkins\*\*:

- \*\*What to do\*\*: After installation, start the Jenkins service. Open a web browser and go to `http://localhost:8080`.

- \*\*Why\*\*: Jenkins needs to run as a service. Visiting `http://localhost:8080` opens the Jenkins interface.

4. \*\*Unlock Jenkins\*\*:

- \*\*What to do\*\*: Jenkins will ask you for an initial admin password. Find the password in a file typically located at `C:\Program Files\Jenkins\secrets\initialAdminPassword` on Windows or `/var/lib/jenkins/secrets/initialAdminPassword` on Linux/Mac.

- \*\*Why\*\*: This step ensures only authorized users can set up Jenkins.

5. \*\*Install Suggested Plugins\*\*:

- \*\*What to do\*\*: Jenkins will prompt you to install plugins; choose the “Install Suggested Plugins” option.

- \*\*Why\*\*: Plugins add features to Jenkins, like integration with version control systems (Git) and other tools. Suggested plugins cover the basics most users need.

#### Step 2: Setting Up Essential Plugins

Now that Jenkins is installed, we’ll add plugins for Python and Maven. These plugins allow Jenkins to run Python scripts and build Java projects.

1. \*\*Navigate to Manage Jenkins\*\*:

- \*\*What to do\*\*: From the Jenkins dashboard (the main page after logging in), click on “Manage Jenkins.”

- \*\*Why\*\*: This section lets you configure Jenkins settings, including installing plugins.

2. \*\*Install the Python Plugin\*\*:

- \*\*What to do\*\*: Go to “Manage Plugins” > “Available” tab, then search for “Python” and click Install.

- \*\*Why\*\*: The Python plugin enables Jenkins to run Python scripts and manage Python environments. This is useful if your project has scripts written in Python.

3. \*\*Install the Maven Plugin\*\*:

- \*\*What to do\*\*: In the same “Manage Plugins” section, search for “Maven Integration” and install it.

- \*\*Why\*\*: Maven is a build automation tool for Java projects. This plugin lets Jenkins handle Maven projects, making it ideal for Java-based CI setups.

4. \*\*Configure Maven\*\*:

- \*\*What to do\*\*: Go back to “Manage Jenkins” > “Global Tool Configuration.” Under Maven, specify the version and installation path of Maven.

- \*\*Why\*\*: This setup tells Jenkins where to find Maven so it can use it for building your projects.

**Or  
Alright, let’s get that Maven set up for you in Jenkins! Here’s what to do:**

**1. \*\*Go to Manage Jenkins\*\*:**

**- From the Jenkins dashboard, click on \*\*Manage Jenkins\*\* in the left sidebar.**

**- Select \*\*Global Tool Configuration\*\* (or "Tools" like in your screenshots).**

**2. \*\*Find the Maven Installation Section\*\*:**

**- Scroll down until you see \*\*Maven installations\*\*.**

**3. \*\*Add Maven\*\*:**

**- Click \*\*Add Maven\*\* to create a new Maven installation.**

**- Give it a name (like "Maven 3.8.1" or whichever version you're using).**

**4. \*\*Set the Maven Version\*\*:**

**- You can either \*\*install automatically\*\* or \*\*provide the path\*\* to an existing Maven installation.**

**- If you choose to install automatically, Jenkins will download the specified version from the official repository, so no need to manually install it.**

**5. \*\*Configure the Settings and Global Settings\*\* (Optional):**

**- The \*\*Default settings provider\*\* and \*\*Default global settings provider\*\* fields allow you to specify custom Maven settings files if needed (e.g., if you have specific repository settings or proxies). By default, Jenkins uses the default Maven settings, which should be fine unless your organization requires custom settings.**

****

**6. \*\*Save\*\*:**

**- Once you've filled in the details, click \*\*Save\*\* at the bottom of the page.**

**7. \*\*Use Maven in Your Job\*\*:**

**- When creating or configuring a Jenkins job (pipeline or freestyle), you’ll now see the option to select this Maven installation under \*\*Build\*\* > \*\*Invoke top-level Maven targets\*\* or in a pipeline script.**

**If you run into issues with the automatic installation or need to specify a local path, let me know!**

#### Step 3: Create a Build Job

With Jenkins and your plugins set up, you’re ready to create a build job. This is the actual process Jenkins will use to build and test your code whenever there’s a new change.

1. \*\*Create a New Job\*\*:

- \*\*What to do\*\*: In the Jenkins dashboard, click on “New Item.”

- \*\*Why\*\*: Each job in Jenkins represents a CI process for a specific project. Naming the job makes it easy to organize and locate.

2. \*\*Select Job Type\*\*:

- \*\*What to do\*\*: Choose “Freestyle project” (for simpler projects) or “Pipeline” (for more complex CI/CD setups).

- \*\*Why\*\*: These types are templates for different levels of automation. "Freestyle" works well for simpler builds, while "Pipeline" offers more control and flexibility.

CHOOSE PIPELINE

3. \*\*Configure Source Code Management\*\*:

- \*\*What to do\*\*: Under “Source Code Management,” choose the version control system you’re using, like Git. Enter the repository URL.

https://github.com/Apoorva-Practical/Try-1

- \*\*Why\*\*: This links your project’s code repository to Jenkins so that Jenkins can monitor changes in the code and trigger builds.

4. \*\*Add Build Steps\*\*:

- \*\*What to do\*\*: Scroll to the “Build” section and add a build step. Choose Maven, Ant, or Gradle, depending on your project needs. For Maven, you could add goals like `clean install`.

- \*\*Why\*\*: Build steps define what Jenkins does when a build is triggered. "Clean install" in Maven, for example, cleans previous builds and installs the project dependencies fresh.

1. PYTHON ON GIT:  
**1. Configuring a Job for a Python Project**

1. **Go to Jenkins Dashboard** and click **New Item**.
2. **Enter Job Name** (e.g., Python-Project-Build) and select **Freestyle Project**.
3. **Source Code Management**:
   * Choose **Git** and enter the repository URL if your Python project is in Git.
4. **Build Environment**:
   * If you have specific environment variables, you can configure them here.
5. **Build Steps**:
   * Select **Add build step** > **WINDOws smth wala option.**.
   * In the command box, enter:

bash

COPY CODE:

# Install dependencies

pip install -r requirements.txt

# Run Python script

python add\_num.py

1. **Post-build Actions** (optional):
   * You can add steps like **Archive the artifacts** to save any output files.
2. **Save and Build Now** to test if Jenkins runs the Python script.

2. Maven on eclipse:  
Building a Java project using Maven in Eclipse and then executing it with Jenkins involves several steps. Below, I’ll guide you through the entire process, from setting up your Java project in Eclipse to configuring Jenkins to build and run the project.

### Part 1: Building a Java and Maven Project in Eclipse

#### Step 1: Install Eclipse IDE

1. \*\*Download Eclipse:\*\*

- Go to the [Eclipse download page](https://www.eclipse.org/downloads/).

- Choose the \*\*Eclipse IDE for Java Developers\*\* and download the appropriate installer for your OS.

2. \*\*Install Eclipse:\*\*

- Follow the installation instructions for your operating system.

#### Step 2: Install Maven in Eclipse

1. \*\*Check Maven Installation:\*\*

- Eclipse usually comes with Maven integrated. To check, go to `Window` > `Preferences` > `Maven`.

- Ensure that Maven is installed. If not, you can install it via Eclipse Marketplace.

#### Step 3: Create a Maven Project

1. \*\*Create a New Maven Project:\*\*

- In Eclipse, go to `File` > `New` > `Other...`.

- Select `Maven` > `Maven Project`, then click `Next`.

2. \*\*Select Project Location:\*\*

- You can use the default workspace or specify a different location.

3. \*\*Select an Archetype:\*\*

- Choose an archetype (for example, `maven-archetype-quickstart`) and click `Next`.

4. \*\*Configure Project:\*\*

- Enter \*\*Group Id\*\* (e.g., `com.example`).

- Enter \*\*Artifact Id\*\* (e.g., `my-java-project`).

- Specify the \*\*Version\*\* (e.g., `1.0-SNAPSHOT`).

- Click `Finish`.

#### Step 4: Write Your Java Code

1. \*\*Open the Project:\*\*

- In the `Package Explorer`, navigate to `src/main/java/com/example`.

2. \*\*Create a Java Class:\*\*

- Right-click on the `com.example` package, select `New` > `Class`, and name it (e.g., `Main`).

3. \*\*Write Your Code:\*\*

- Implement your Java code in the newly created class.

4. \*\*Create a `pom.xml`:\*\*

- If not created automatically, you’ll find a `pom.xml` file in your project root. This file manages dependencies and build configurations.

#### Step 5: Build the Maven Project in Eclipse

1. \*\*Build the Project:\*\*

- Right-click on your project in the `Package Explorer`.

- Select `Run As` > `Maven build...`.

2. \*\*Enter Goals:\*\*

- In the dialog, enter `clean install` as goals and click `Run`.

3. \*\*Verify Build Success:\*\*

- Check the `Console` view for build success messages.

### Part 2: Setting Up Jenkins to Build and Execute the Project

#### Step 1: Install Jenkins

1. \*\*Download Jenkins:\*\*

- Visit the [Jenkins download page](https://www.jenkins.io/download/) and download the installer for your operating system.

2. \*\*Install Jenkins:\*\*

- Follow the installation instructions provided for your OS.

3. \*\*Start Jenkins:\*\*

- After installation, start Jenkins. You can access it via `http://localhost:8080`.

#### Step 2: Configure Jenkins

1. \*\*Unlock Jenkins:\*\*

- During the first launch, you’ll be asked to unlock Jenkins using an initial admin password. Find this password in the specified file and enter it.

2. \*\*Install Suggested Plugins:\*\*

- Follow the on-screen instructions to install the suggested plugins.

3. \*\*Create Admin User:\*\*

- Set up your first admin user when prompted.

#### Step 3: Create a New Job in Jenkins

1. \*\*Create a New Item:\*\*

- On the Jenkins dashboard, click on \*\*New Item\*\*.

2. \*\*Enter Job Name:\*\*

- Enter a name for your job (e.g., `MyJavaProjectBuild`).

3. \*\*Select Freestyle Project:\*\*

- Choose \*\*Freestyle project\*\* and click \*\*OK\*\*.

#### Step 4: Configure Source Code Management

1. \*\*Configure SCM:\*\*

- In the job configuration page, scroll to \*\*Source Code Management\*\*.

- Select \*\*Git\*\* and enter the repository URL where your Maven project is hosted (e.g., GitHub).

2. \*\*Specify Branch:\*\*

- Enter the branch you want to build (e.g., `main`).

3. \*\*Add Credentials (If Necessary):\*\*

- If your repository is private, add credentials for accessing the repository.

#### Step 5: Configure Build Environment

1. \*\*Add Build Step:\*\*

- Scroll down to the \*\*Build\*\* section and click on \*\*Add build step\*\*.

2. \*\*Choose Invoke top-level Maven targets:\*\*

- Select \*\*Invoke top-level Maven targets\*\*.

3. \*\*Set Goals:\*\*

- In the \*\*Goals\*\* field, enter:

```

clean install

```

4. \*\*Set POM Location:\*\*

- If your `pom.xml` is not in the root directory, specify its location.

#### Step 6: Execute the Build

1. \*\*Add Post-Build Action (Optional):\*\*

- If you want to run a Java program after the build, you can add another build step.

- Choose \*\*Execute Windows batch command\*\* (or \*\*Execute shell\*\* if using a Unix-like environment).

- Enter the command to run your Java application:

```bash

java -cp target/my-java-project-1.0-SNAPSHOT.jar com.example.Main

```

- Adjust the classpath and class name according to your project structure.

#### Step 7: Save and Build

1. \*\*Save the Configuration:\*\*

- Scroll down and click on \*\*Save\*\*.

2. \*\*Build the Job:\*\*

- On the job page, click on \*\*Build Now\*\* to trigger a build.

3. \*\*Check Console Output:\*\*

- Click on the build number to see the console output and verify that the build and execution were successful.

---

### Conclusion: What Each Part Achieves

1. \*\*Understanding Continuous Integration\*\*: CI makes it easy to spot issues early by automatically building and testing code. You just make changes, commit them, and Jenkins handles the rest.

2. \*\*Jenkins as a CI Tool\*\*: Jenkins automates the CI process, saving time and reducing errors in manual testing and builds.

3. \*\*Using Plugins\*\*: Plugins like Python and Maven Integration expand Jenkins’s abilities, making it compatible with more programming languages and tools.

4. \*\*Automated Build Jobs\*\*: With your build job set up, Jenkins will continuously integrate your code changes, helping your team produce better software, faster.

---