

## DevOps LAB

### 1. Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup.

#### Introduction to Maven and Gradle

##### Overview of Build Automation Tools

Build automation tools simplify the process of **compiling, testing, packaging, and deploying** software projects. They manage dependencies, execute tasks, and integrate seamlessly with CI/CD pipelines.

##### Why Use Build Automation?

- ✓ Ensures **consistency** across builds
- ✓ Handles **dependency management** automatically
- ✓ Reduces **manual errors** and increases efficiency
- ✓ Integrates with tools like **Jenkins & Azure DevOps**

##### Popular Build Automation Tools

**Apache Ant** → Script-based, manual dependency management

**Apache Maven** → XML-based, convention-over-configuration model

**Gradle** → Flexible, fast, and supports Groovy/Kotlin DSL

Two popular tools in the Java ecosystem are **Maven** and **Gradle**. Both are great for managing project builds and dependencies, but they have some key differences.

##### Maven

- **What is Maven?** Maven is a build automation tool primarily used for Java projects. It uses an XML configuration file called pom.xml (Project Object Model) to define project settings, dependencies, and build steps.
- **Main Features:**
  - Predefined project structure and lifecycle phases.
  - Automatic dependency management through Maven Central.
  - Wide range of plugins for things like testing and deployment.
  - Supports complex projects with multiple modules.

# Gradle

- **What is Gradle?** Gradle is a more modern and versatile build tool that supports multiple programming languages, including Java, Groovy, and Kotlin. It uses a domain-specific language (DSL) for build scripts, written in Groovy or Kotlin.
- **Main Features:**
  - Faster builds thanks to task caching and incremental builds.
  - Flexible and customizable build scripts.
  - Works with Maven repositories for dependency management.
  - Excellent support for multi-module and cross-language projects.
  - Integrates easily with CI/CD pipelines.

## Key Differences Between Maven and Gradle

Aspect	Maven	Gradle
Configuration	XML (pom.xml)	Groovy or Kotlin DSL
Performance	Slower	Faster due to caching
Flexibility	Less flexible	Highly customizable
Learning Curve	Easier to pick up	Slightly steeper
Script Size	Verbose	More concise
Dependency Management	Uses Maven Central	Compatible with Maven too
Plugin Support	Large ecosystem	Extensible and versatile

Maven is great for structured, enterprise-level projects.

Gradle is ideal for scalable and performance-driven applications.

## Installation and Setup

### How to Install Maven:

#### 1. Download Maven:

- Go to the [Maven Download Page](#) and download the latest binary ZIP file.

#### 2. Extract the ZIP File:

- Right-click the downloaded ZIP file and select **Extract All...** or use any extraction tool like WinRAR or 7-Zip.

#### 3. Move the Folder:

- After extraction, move the extracted **Maven folder** (usually named **apache-maven-x.x.x**) to a convenient directory like C:\Program Files\.

#### 4. Navigate to the bin Folder:

- Open the **Maven folder**, then navigate to the **bin** folder inside.
- Copy the path from the File Explorer address bar(e.g., **C:\Program Files\apache-maven-x.x.x\bin**).

#### 5. Set Environment Variables:

- Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
- Click **Environment Variables**.
- Under **System Variables**:
  - Find the **path**, double click on it and click **New**.
  - Paste the full path to the bin folder of your Maven directory (e.g., **C:\Program Files\apache-maven-x.x.x\bin**).

#### 6. Save the Changes:

- Click **OK** to close the windows and save your changes.

#### 7. Verify the Installation:

- Open Command Prompt and run: **mvn --version** If Maven is correctly installed, it will display the version number.

### How to install Gradle

#### 1. Download Gradle:

Visit the [Gradle Downloads Page](#) and download the latest binary ZIP file.

2. **Extract the ZIP File:**

- Right-click the downloaded ZIP file and select **Extract All...** or use any extraction tool like WinRAR or 7-Zip.

3. **Move the Folder:**

- After extraction, move the extracted **Gradle folder** (usually named **gradle-x.x.x**) to a convenient directory like C:\Program Files\.

4. **Navigate to the bin Folder:**

- Open the **Gradle folder**, then navigate to the **bin** folder inside.
- Copy the path from the File Explorer address bar (e.g., C:\Program Files\gradle-x.x\bin).

5. **Set Environment Variables:**

- Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
- Click **Environment Variables**.
- Under **System Variables**:
  - Find the **path**, double click on it and click **New**.
  - Paste the full path to the bin folder of your Gradle directory (e.g., C:\Program Files\gradle-x.x.x\bin).

6. **Save the Changes:**

- Click **OK** to close the windows and save your changes.

7. **Verify the Installation:**

- Open a terminal or Command Prompt and run: **gradle --version** If it shows the Gradle version, the setup is complete.

## 2. **Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins**

### **Maven: Basic Introduction**

**Maven** is a build automation tool primarily used for Java projects. It simplifies the build process, manages project dependencies, and supports plugins for different tasks. It uses XML files (called pom.xml ) for project configuration and dependency management.

#### **Key Benefits of Maven:**

1. Dependency management (automates downloading and including libraries).
2. Build automation (compiles code, runs tests, creates artifacts).
3. Consistent project structure (standardizes how Java projects are set up).
4. Integration with CI tools (like Jenkins).

#### **Key Features of Maven:**

1. **Project Management:** Handles project dependencies, configurations, and builds.
2. **Standard Directory Structure:** Encourages a consistent project layout across Java projects.
3. **Build Automation:** Automates tasks such as compilation, testing, packaging, and deployment.
4. **Dependency Management:** Downloads and manages libraries and dependencies from repositories (e.g.,
5. Maven Central).
6. **Plugins:** Supports many plugins for various tasks like code analysis, packaging, and deploying.

### **1: Install the Java JDK**

- If you haven't installed the Java JDK yet, you can follow the link below to download and install it. [Download Java JDK from Oracle](#)

Working with Maven is a key skill for managing Java-based projects, particularly in the areas of build automation, dependency management, and project configuration. Below is a guide on creating a Maven project, understanding the POM file, and using dependency management and plugins:

### **Overview of the Project**

### **2: Creating a Maven Project**

There are a few ways to create a Maven project, such as using the command line, IDEs like IntelliJ IDEA or Eclipse, or generating it via an archetype.

### 1. Using Command Line:

- To create a basic Maven project using the command line, you can use the following command:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=myapp -  
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Above is Maven command to generate a new Java project using a Maven archetype.

### Explanation of the command:

1. **mvn**: This is the command for invoking Maven, which is a build automation tool for Java projects.
2. **archetype:generate**: This tells Maven to generate a new project based on a specified archetype. An archetype is a template or a project prototype.
3. **-DgroupId=com.example**: This defines the **groupId** for your project. The **groupId** uniquely **identifies the group** (usually your organization's domain name) to which your project belongs. In this case, it's set to com.example.

### Examples:

org.apache.maven

org.apache.commons

com.google.guava

4. **-DartifactId=myapp**: This defines the **artifactId** of your project. The **artifactId** is the **name of the project**, and it's used to name the project's JAR file after it's built. Here, it's set to myapp.

**Note:** The identifiers should only consist of *lowercase* letters, digits, and hyphens.

### Examples:

commons-math

maven-clean-plugin

5. **-DarchetypeArtifactId=maven-archetype-quickstart**: This specifies the archetype to use for generating the project. In this case, it's using the maven-archetype-quickstart archetype, which is a simple Maven archetype for creating a **basic Java application** with a sample structure.
6. **-DinteractiveMode=false**: This option disables interactive mode, which means Maven won't prompt you for further inputs while creating the project. It will automatically accept the given parameters.

### What happens after running the command:

- Maven will create a new directory called **myapp** (based on the **artifactId**).
- Inside this directory, Maven will generate the following project structure:
  - A pom.xml file (which contains the project's build configuration).
  - A sample App.java file located in src/main/java/com/example/App.java.
  - A simple test file AppTest.java located in src/test/java/com/example/AppTest.java.

### Summary:

- groupId: A unique identifier for the group (usually the domain name).
- artifactId: A unique name for the project artifact (your project).
- archetypeArtifactId: The template you want to use for the project.
- DinteractiveMode=false: Disables prompts during project generation.

This will create a basic Maven project with the required directory structure and pom.xml file.

## 2. Understanding the POM File

The POM (Project Object Model) file is the heart of a Maven project. It is an XML file that contains all the configuration details about the project. Below is an example of a simple POM file:

A basic pom.xml structure looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <dependencies>
    <!-- Dependencies go here -->
  </dependencies>

  <build>
    <plugins>
      <!-- Plugins go here -->
    </plugins>
  </build>
</project>
```

### Key element in pom.xml:

- `<groupId>`: The group or organization that the project belongs to.
- `<artifactId>`: The name of the project or artifact.
- `<version>`: The version of the project (often follows a format like 1.0-SNAPSHOT).
- `<packaging>`: Type of artifact, e.g., jar, war, pom, etc.
- `<dependencies>`: A list of dependencies the project requires.
- `<build>`: Specifies the build settings, such as plugins to use.

### 3. Dependency Management

- Maven uses the `<dependencies>` tag in the pom.xml to manage external libraries or dependencies that your project needs. When Maven builds the project, it will automatically download these dependencies from a repository (like Maven Central).



- **Example of adding a dependency:**

```
<dependencies>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
  </dependency>
</dependencies>
```

- **Transitive Dependencies**

- Maven automatically resolves transitive dependencies. For example, if you add a library that depends on other libraries, Maven will also download those.

- **Scopes**

- Dependencies can have different scopes that determine when they are available:
  - **compile (default):** Available in all build phases.
  - **provided:** Available during compilation but not at runtime (e.g., a web server container).
  - **runtime:** Needed only at runtime, not during compilation.
  - **test:** Required only for testing.

#### 4. Using Plugins

Maven plugins are used to perform tasks during the build lifecycle, such as compiling code, running tests, packaging, and deploying. You can specify plugins within the <build> section of your **pom.xml**.

- **Adding Plugins**

- You can add a plugin to your pom.xml like so:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

In this example, the maven-compiler-plugin is used to compile Java code and specify the source and target JDK versions.

## 1. Common Plugins

- **maven-compiler-plugin:** Compiles Java code.
- **maven-surefire-plugin:** Runs unit tests.
- **maven-jar-plugin:** Packages the project as a JAR file.
- **maven-clean-plugin:** Cleans up the target/ directory.

## 2. Plugin Goals Each plugin consists of goals, which are specific tasks to be executed.

For example:

- **mvn clean install:** This will clean the target directory and then install the package in the local repository.
- **mvn compile:** This will compile the source code.
- **mvn test:** This will run unit tests.

## 5. Dependency Versions and Repositories

### 1. Version Ranges

- You can specify a version range for dependencies, allowing Maven to choose a compatible version automatically. for example:

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>[30.0,)</version> <!-- Guava version 30.0 or higher -->
</dependency>
```

### 2. Repositories

- Maven primarily fetches dependencies from Maven Central, but you can also specify custom repositories. For example:

```
<repositories>
  <repository>
    <id>custom-repo</id>
    <url>https://repo.example.com/maven2</url>
  </repository>
</repositories>
```

Run the application (using JAR)

```
java -cp target/myapp-1.0-SNAPSHOT.jar com.example.App
```

The command `mvn exec:java -Dexec.mainClass="com.example.App"` is used to run a Java application from the command line using Apache Maven.

Here's what each part of the command does:

- `mvn`: This invokes Maven, a build automation tool.
- `exec:java`: This tells Maven to use the `exec-maven-plugin` to execute a Java program.
- `-Dexec.mainClass="com.example.App"`: This defines a system property (`exec.mainClass`) that specifies the fully qualified name of the main class (`com.example.App`) that contains the public static void `main(String[] args)` method to run.

**3. Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation.**

## Gradle Project Overview

### 1: Setting Up a Gradle Project

- Install Gradle (If you haven't already):
  - Follow Gradle installation steps
- Create a new Gradle project: You can set up a new Gradle project using the Gradle Wrapper or manually. Using the Gradle Wrapper is the preferred approach as it ensures your project will use the correct version of Gradle.
- To create a new Gradle project using the command line:

***gradle init --type java-application***

This command creates a new Java application project with a sample **build.gradle** file.

### 2: Understanding Build Scripts

Gradle uses a DSL (Domain-Specific Language) to define the build scripts.

Gradle supports two DSLs:

- Groovy DSL (default)
- Kotlin DSL (alternative)

Groovy DSL: This is the default language used for Gradle build scripts (build.gradle).

Example of a simple build.gradle file (Groovy DSL):

```
plugins {
```

```
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web:2.5.4'
}

task customTask {
    doLast {
        println 'This is a custom task'
    }
}
```

**Kotlin DSL:** Gradle also supports Kotlin for its build scripts (**build.gradle.kts**). Example of a simple **build.gradle.kts** file (Kotlin DSL):

```
plugins {
    kotlin("jvm") version "1.5.21"
}

repositories {
    mavenCentral()
}

dependencies {
    implementation("org.springframework.boot:spring-boot-starter-web:2.5.4")
}
```

```
}  
  
tasks.register("customTask") {  
  
    doLast {  
  
        println("This is a custom task")  
  
    }  
  
}
```

### **Difference between Groovy and Kotlin DSL:**

- **Syntax:** Groovy uses a more concise, dynamic syntax, while Kotlin offers a more structured, statically-typed approach.
- **Error handling:** Kotlin provides better error detection at compile time due to its static nature.

**Task Block:** Tasks define operations in Gradle, and they can be executed from the command line using `gradle <task-name>`.

### **In Groovy DSL:**

```
task hello {  
  
    doLast {  
  
        println 'Hello, Gradle!'  
  
    }  
  
}
```

### **In Kotlin DSL:**

```
tasks.register("hello") {  
  
    doLast {  
  
        println("Hello, Gradle!")  
  
    }  
  
}
```

### **3: Dependency Management**

Gradle provides a powerful dependency management system. You define your project's dependencies in the dependencies block.

#### **1. Adding dependencies:**

- Gradle supports various dependency scopes such as implementation, compileOnly, testImplementation, and others.

Example of adding a dependency in **build.gradle (Groovy DSL)**:

```
dependencies {  
  
    implementation 'com.google.guava:guava:30.1-jre'  
  
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.7.1'  
  
}
```

Example in **build.gradle.kts (Kotlin DSL)**:

```
dependencies {  
  
    implementation("com.google.guava:guava:30.1-jre")  
  
    testImplementation("org.junit.jupiter:junit-jupiter-api:5.7.1")  
  
}
```

2. **Declaring repositories:** To resolve dependencies, you need to specify repositories where Gradle should look for them. Typically, you'll use Maven Central or JCenter, but you can also configure private repositories.

**Example (Groovy):**

```
repositories {  
  
    mavenCentral()  
  
}
```

**Example (Kotlin):**

```
repositories {  
  
    mavenCentral()  
  
}
```

#### **4: Task Automation**

Gradle tasks automate various tasks in your project lifecycle, like compiling code, running tests, and creating builds.

1. **Using predefined tasks:** Gradle provides many predefined tasks for common activities, such as:
  - **build** – compiles the project, runs tests, and creates the build output.
  - **test** – runs tests.
  - **clean** – deletes the build output.

2. **Example of running the build task:**

```
gradle build
```

3. **Creating custom tasks:** You can define your own tasks to automate specific actions. For example, creating a custom task to print a message.

- **Example Groovy DSL:**



```
task printMessage {  
    doLast {  
        println 'This is a custom task automation'  
    }  
}
```

- **Example Kotlin DSL:**

```
tasks.register("printMessage") {  
    doLast {  
        println("This is a custom task automation")  
    }  
}
```

## 5: Running Gradle Tasks

To run a task, use the following command in the terminal:

```
gradle <task-name>
```

For example:

- To run the build task: **gradle build**
- To run a custom task: **gradle printMessage**

## 6: Advanced Automation

You can define task dependencies and configure tasks to run in a specific order. Example of task dependency:

```
task firstTask {  
    doLast {  
        println 'Running the first task'    }  
}
```

```

    }
}

task secondTask {

    dependsOn firstTask

    doLast {

        println 'Running the second task'

    }

}

```

In this case, **secondTask** will depend on the completion of **firstTask** before it runs.

### ❖ Working with Gradle Project (Groovy DSL):

#### Step 1: Create a new Project

```
gradle init --type java-application
```

- while creating project it will ask necessary requirement:
  - **Enter target Java version (min: 7, default: 21):** 17
  - **Project name (default: program3-groovy):** groovyProject
  - **Select application structure:**
    - 1: Single application project
    - 2: Application and library project
    - **Enter selection (default: Single application project) [1..2]** 1
  - **Select build script DSL:**
    - 1: Kotlin
    - 2: Groovy

- Enter selection (default: Kotlin) [1..2] 2
- Select test framework:
  - 1: JUnit 4
  - 2: TestNG
  - 3: Spock
  - 4: JUnit Jupiter
- Enter selection (default: JUnit Jupiter) [1..4] 1
- Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
  - no

```
C:\Users\cool>cd groovyproject
C:\Users\cool\groovyproject>gradle init --type java-application
Starting a Gradle Daemon (subsequent builds will be faster)
Enter target Java version (min: 7, default: 21):
Project name (default: groovyproject):
Select application structure:
  1: Single application project
  2: Application and library project
Enter selection (default: Single application project) [1..2] 1
Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2] 2
Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
no
```

Execute build ,test and run commands

```

> Task :init
Learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.12.1/samples/sample\_building\_java\_applications.html

BUILD SUCCESSFUL in 4m 52s
1 actionable task: 1 executed
C:\Users\cool\groovyproject>gradle build
Calculating task graph as no cached configuration is available for tasks: build

BUILD SUCCESSFUL in 13s
7 actionable tasks: 7 executed
Configuration cache entry stored.
C:\Users\cool\groovyproject>gradle test
Calculating task graph as no cached configuration is available for tasks: test

BUILD SUCCESSFUL in 2s
3 actionable tasks: 3 up-to-date
Configuration cache entry stored.
<-----> 0% WAITING
> IDLE
C:\Users\cool\groovyproject>gradle run
Calculating task graph as no cached configuration is available for tasks: run

> Task :app:run
Hello World!

BUILD SUCCESSFUL in 2s
2 actionable tasks: 1 executed, 1 up-to-date
Configuration cache entry stored.

```

## Step 2: build.gradle (Groovy DSL)

```

plugins {

    id 'application'

}

application {

    mainClass = 'org.example.AdditionOperation'

}

repositories {

    mavenCentral()

}

dependencies {

    testImplementation 'junit:junit:4.13.2'

}

```

```

test {

    outputs.upToDateWhen { false }

    testLogging {

        events "passed", "failed", "skipped"

        exceptionFormat "full"

        showStandardStreams = true

    }

}

```

### **Step 3: AdditionOperation.java(Change file name and update below code)**

- After creating project change the file name.
- Manually navigate the folder path like src/main/java/org/example/
- Change the file name App.java to AdditionOperation.java
- After then open that file and copy the below code and past it, save it.

```

package org.example;

public class AdditionOperation {

    public static void main(String[] args) {

        double num1 = 5;

        double num2 = 10;

        double sum = num1 + num2;

        System.out.printf("The sum of %.2f and %.2f is %.2f\n", num1, num2, sum);

    }

}

```

#### Step 4: AdditionOperationTest.java (JUnit Test) (Change file name and update below code)

- After creating project change the file name.
- Manually navigate the folder path like src/test/java/org/example/
- Change the file name AppTest.java to AdditionOperationTest.java
- After then open that file and copy the below code and past it, save it.

```
package org.example;

import org.junit.Test;

import static org.junit.Assert.*;

public class AdditionOperationTest {

    @Test

    public void testAddition() {

        double num1 = 5;

        double num2 = 10;

        double expectedSum = num1 + num2;

        double actualSum = num1 + num2;

        assertEquals(expectedSum, actualSum, 0.01);

    }

}
```

#### Step 5: Run Gradle Commands

- To build the project:

gradle build

- To run the project:

gradle run

- To test the project:

gradle test

## ❖ **Working with Gradle Project (Kotlin DSL):**

### **Step 1: Create a new Project**

```
gradle init --type java-application
```

- **while creating project it will ask necessary requirement:**
  - Enter target Java version (min: 7, default: 21): 17
  - Project name (default: program3-kotlin): kotlinProject
  - Select application structure:
    - 1: Single application project
    - 2: Application and library project
      - Enter selection (default: Single application project) [1..2] 1
  - Select build script DSL:
    - 1: Kotlin
    - 2: Groovy
      - Enter selection (default: Kotlin) [1..2] 1
  - Select test framework:
    - 1: JUnit 4
    - 2: TestNG
    - 3: Spock
    - 4: JUnit Jupiter
      - Enter selection (default: JUnit Jupiter) [1..4] 1

- Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
- no

```

Select application structure:
 1: Single application project
 2: Application and library project
Enter selection (default: Single application project) [1..2] 1

Select build script DSL:
 1: Kotlin
 2: Groovy
Enter selection (default: Kotlin) [1..2] 1

Select test framework:
 1: JUnit 4
 2: TestNG
 3: Spock
 4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
no

> Task :init
Learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.12.1/samples/sample_building_java_applications.html

BUILD SUCCESSFUL in 19s
1 actionable task: 1 executed
C:\Users\braun\program3-kotlin>

```

## Step 2: build.gradle.kts (Kotlin DSL)

```

plugins {
    kotlin("jvm") version "1.8.21"
    application
}

repositories {
    mavenCentral()
}

dependencies {
    implementation(kotlin("stdlib"))
    testImplementation("junit:junit:4.13.2")
}

```



```

application {
    mainClass.set("org.example.MainKt")
}

tasks.test {
    useJUnit()

    testLogging {
        events("passed", "failed", "skipped")

        exceptionFormat = org.gradle.api.tasks.testing.logging.TestExceptionFormat.FULL

        showStandardStreams = true
    }

    outputs.upToDateWhen { false }
}

java {
    toolchain {
        languageVersion.set(JavaLanguageVersion.of(17))
    }
}

```

### **Step 3: Main.kt (Change file name and update below code)**

- After creating project change the file name.
- Manually navigate the folder path like src/main/java/org/example/
- Change the file name App.java to Main.kt
- After then open that file and copy the below code and past it, save it.

```

package org.example

fun addNumbers(num1: Double, num2: Double): Double {

    return num1 + num2

}

fun main() {

    val num1 = 10.0

    val num2 = 5.0

    val result = addNumbers(num1, num2)

    println("The sum of $num1 and $num2 is: $result")

}

```

#### **Step 4: MainTest.kt (JUnit Test) (Change file name and update below code)**

- After creating project change the file name.
- Manually navigate the folder path like src/test/java/org/com/example/
- Change the file name MainTest.java to MainTest.kt
- After then open that file and copy the below code and past it, save it.

```

package org.example

import org.junit.Assert.*

import org.junit.Test

class MainTest {

    @Test

    fun testAddNumbers() {

        val num1 = 10.0

        val num2 = 5.0

```

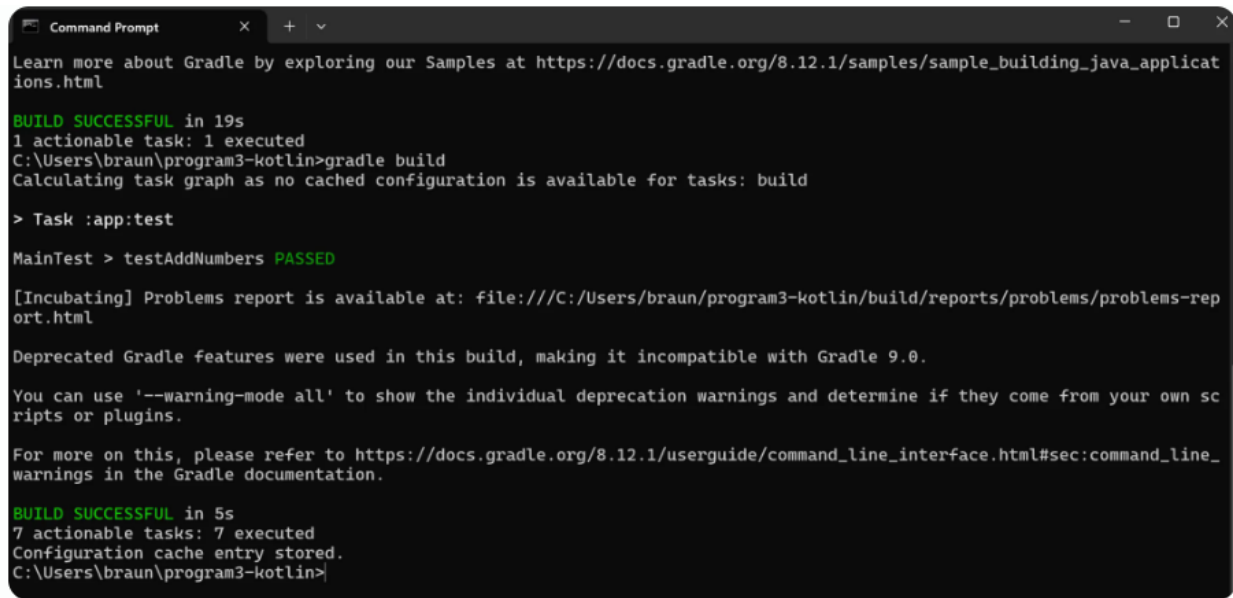
```
        val result = addNumbers(num1, num2)

        assertEquals("The sum of $num1 and $num2 should be 15.0", 15.0, result, 0.001)
    }
}
```

## Step 5: Run Gradle Commands

- To build the project:

gradle build



```
Command Prompt
Learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.12.1/samples/sample_building_java_applications.html

BUILD SUCCESSFUL in 19s
1 actionable task: 1 executed
C:\Users\braun\program3-kotlin>gradle build
Calculating task graph as no cached configuration is available for tasks: build

> Task :app:test

MainTest > testAddNumbers PASSED

[Incubating] Problems report is available at: file:///C:/Users/braun/program3-kotlin/build/reports/problems/problems-report.html

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.12.1/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 5s
7 actionable tasks: 7 executed
Configuration cache entry stored.
C:\Users\braun\program3-kotlin>
```

- To run the project:

gradle run

```

C:\Users\braun\program3-kotlin>gradle run
Calculating task graph as no cached configuration is available for tasks: run
> Task :app:run
The sum of 10.0 and 5.0 is: 15.0

[Incubating] Problems report is available at: file:///C:/Users/braun/program3-kotlin/build/reports/problems/problems-report.html

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.12.1/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 1s
2 actionable tasks: 1 executed, 1 up-to-date
Configuration cache entry stored.
C:\Users\braun\program3-kotlin>

```

- To test the project:

gradle test

```

C:\Users\braun\program3-kotlin>gradle test
Calculating task graph as no cached configuration is available for tasks: test
> Task :app:test

MainTest > testAddNumbers PASSED

[Incubating] Problems report is available at: file:///C:/Users/braun/program3-kotlin/build/reports/problems/problems-report.html

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.


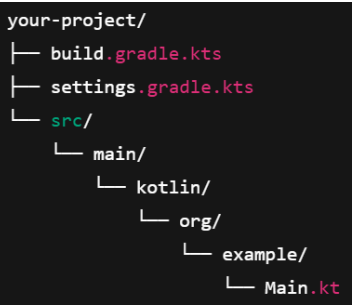
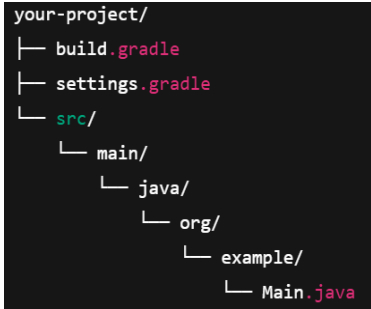
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.12.1/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 1s
3 actionable tasks: 1 executed, 2 up-to-date
Configuration cache entry stored.
C:\Users\braun\program3-kotlin>

```

### Summary: Example 3:-

Groovy	Kotlin	Java
<b>Directory structure</b>  <pre>your-project/ ├─ build.gradle ├─ settings.gradle └─ src/     └─ main/         └─ groovy/             └─ org/                 └─ example/                     └─ Main.groovy</pre>	<b>Directory structure</b>  <pre>your-project/ ├─ build.gradle.kts ├─ settings.gradle.kts └─ src/     └─ main/         └─ kotlin/             └─ org/                 └─ example/                     └─ Main.kt</pre>	<b>Directory structure</b>  <pre>your-project/ ├─ build.gradle ├─ settings.gradle └─ src/     └─ main/         └─ java/             └─ org/                 └─ example/                     └─ Main.java</pre>
<b>Main.groovy</b> <pre>package org.example  class Main {     static void main(String[] args) {         int num1 = 8         int num2 = 6         int result = num1 * num2          println "Multiplication of \$num1 and \$num2 is \$result"     } }</pre>	<b>Main.kt</b> <pre>package org.example  fun main() {     val num1 = 8     val num2 = 6     val result = num1 * num2      println("Multiplication of \$num1 and \$num2 is \$result") }</pre>	<b>Main.java</b> <pre>package org.example;  public class Main {     public static void main(String[] args) {         int num1 = 8;         int num2 = 6;         int result = num1 * num2;          System.out.println("Multiplication of " + num1 + " and " + num2 + " is " + result);     } }</pre>
<b>build.gradle</b> <pre>plugins {     id 'groovy'     id 'application' }  repositories {     mavenCentral() }</pre>	<b>build.gradle.kts</b> <pre>plugins {     kotlin("jvm") version "1.9.0"     application }  repositories {     mavenCentral() }</pre>	<b>build.gradle</b> <pre>plugins {     id 'java'     id 'application' }  repositories {     mavenCentral() }</pre>

<pre>dependencies {     implementation 'org.codehaus.groovy:groovy-all:3.0.9' }  application {     mainClass = 'org.example.Main' }</pre>	<pre>dependencies {     implementation(kotlin("stdlib")) }  application {     mainClass.set("org.example.MainKt") }  Note: Kotlin puts top-level functions (like main) in a class named MainKt, so we reference it that way.</pre>	<pre>application {     mainClass = 'org.example.Main' }</pre>
<b>settings.gradle</b> <pre>rootProject.name = 'GroovyMultiplication'</pre>	<b>settings.gradle.kts</b> <pre>rootProject.name = "KotlinMultiplication"</pre>	<b>settings.gradle</b> <pre>rootProject.name = 'JavaMultiplication'</pre>
<p><b>gradle build</b></p> <p><b>gradle run</b></p> <p><b>gradle test</b></p>		
<p><b>Output:- Multiplication of 8 and 6 is 48</b></p>		

#### 4. Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle.

##### Step 1: Creating a Maven Project

You can create a Maven project using the mvn command (or through your IDE, as mentioned earlier). But here, I'll give you the essential pom.xml and Java code.

##### ❖ Using Command Line:

- To create a basic Maven project using the command line, you can use the following command:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=maven-example -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

##### Step 2: Open The pom.xml File

- You can manually navigate the project folder named call maven-example and open the file pom.xml and copy the below code and paste it then save it.
- In case if you not getting project folder then type command in your cmd.
  - cd maven-example – is use to navigate the project folder.
  - notepad pom.xml – is use to open pom file in notepad.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>


  <groupId>com.example</groupId>
```

```
<artifactId>maven-example</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

  <dependency>

    <groupId>junit</groupId>

    <artifactId>junit</artifactId>

    <version>4.12</version>

    <scope>test</scope>

  </dependency>

</dependencies>

<build>

  <plugins>

    <plugin>

      <groupId>org.apache.maven.plugins</groupId>

      <artifactId>maven-compiler-plugin</artifactId>

      <version>3.8.1</version>

      <configuration>

        <source>1.8</source>

        <target>1.8</target>

      </configuration>

    </plugin>

  </plugins>

</build>
```



</project>

### Step 3: Open Java Code (App.java) File

- Open a file App.java inside the src/main/java/com/example/ directory.
- After opening the App.java copy the below code and paste it in that file then save it.

```
package com.example;

public class App {

    public static void main(String[] args) {

        System.out.println("Hello, Maven");

        System.out.println("This is the simple realworld example....");

        int a = 5;

        int b = 10;

        System.out.println("Sum of " + a + " and " + b + " is " + sum(a, b));

    }

    public static int sum(int x, int y) {

        return x + y;

    }

}
```

Note: before building the project make sure you are in the project folder if not navigate the project folder type command in your command prompt cd maven-example.

## Step 4: Run the Project

To build and run this project, follow these steps:

- Open the terminal in the project directory and run the following command to build the project.
  - mvn clean install
- Run the program with below command:
  - mvn exec:java -Dexec.mainClass="com.example.App"

```
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- exec:3.0.0:java (default-cli) @ maven-example ---
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-util/1.4.1/maven-resolver-util-1.4.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-util/1.4.1/maven-resolver-util-1.4.1.pom (2.8 kB at 90 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver/1.4.1/maven-resolver-1.4.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver/1.4.1/maven-resolver-1.4.1.pom (18 kB at 387 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.pom (2.6 kB at 57 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.pom (13 kB at 413 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.4.0/commons-exec-1.4.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.4.0/commons-exec-1.4.0.pom (9.5 kB at 307 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm/9.7.1/asm-9.7.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm/9.7.1/asm-9.7.1.pom (2.4 kB at 50 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.pom (2.8 kB at 90 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/9.7.1/asm-tree-9.7.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/9.7.1/asm-tree-9.7.1.pom (2.6 kB at 81 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-util/1.4.1/maven-resolver-util-1.4.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-util/1.4.1/maven-resolver-util-1.4.1.jar (168 kB at 3.6 MB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm/9.7.1/asm-9.7.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm/9.7.1/asm-9.7.1.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.4.0/commons-exec-1.4.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.4.0/commons-exec-1.4.0.jar (66 kB at 270 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/9.7.1/asm-tree-9.7.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/9.7.1/asm-tree-9.7.1.jar (73 kB at 284 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.jar (126 kB at 489 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.jar (149 kB at 575 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.jar (193 kB at 743 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/9.7.1/asm-tree-9.7.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-tree/9.7.1/asm-tree-9.7.1.jar (52 kB at 201 kB/s)
Hello, Maven
This is the simple realworld example....
Sum of 5 and 10 is 15
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.650 s
[INFO] Finished at: 2025-04-08T17:46:44+05:30
[INFO] -----
```

## Step 5: Migrate the Maven Project to Gradle

1. Initialize Gradle: Navigate to the project directory (gradle-example) and run:
  - gradle init
- It will ask Found a Maven build. Generate a Gradle build from this? (default: yes) [yes, no]
  - Type Yes
- Select build script DSL:
  - 1: Kotlin

- 2: Groovy
- Enter selection (default: Kotlin) [1..2]
  - **Type 2**
- **Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]**
- **Type No**

```
C:\Users\cool\Devops\maven-example>gradle init
Found a Maven build. Generate a Gradle build from this? (default: yes) [yes, no] yes
Select build script DSL:
 1: Kotlin
 2: Groovy
Enter selection (default: Kotlin) [1..2] 2
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] no

> Task :init
Maven to Gradle conversion is an incubating feature.
For more information, please refer to https://docs.gradle.org/8.12.1/userguide/migrating_from_maven.html in the Gradle documentation

BUILD SUCCESSFUL in 42s
1 actionable task: 1 executed
```

2. **Navigate the project folder and open build.gradle file then add the below code and save it.**

```
plugins {
    id 'java'
}

group = 'com.example'

version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.12'
}
```

```
task run(type: JavaExec) {  
  
    main = 'com.example.App'  
  
    classpath = sourceSets.main.runtimeClasspath  
  
}
```

## Step 6: Run the Gradle Project

- **Build the Project:** In the project directory (gradle-example), run the below command to build the project:
  - `gradle build`
- **Run the Application:** Once the build is successful, run the application using below command:
  - `gradle run`

```
1 actionable task: 1 executed  
C:\Users\cool\Devops\maven-example>gradle build  
Calculating task graph as no cached configuration is available for tasks: build  
[Incubating] Problems report is available at: file:///C:/Users/cool/Devops/maven-example/build/reports/problems/problems-report.html  
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.  
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.  
For more on this, please refer to https://docs.gradle.org/8.12.1/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.  
  
BUILD SUCCESSFUL in 9s  
4 actionable tasks: 4 executed  
Configuration cache entry stored.  
C:\Users\cool\Devops\maven-example>gradle run  
Calculating task graph as no cached configuration is available for tasks: run  
  
> Task :run  
Hello, Maven  
This is the simple realworld example....  
Sum of 5 and 10 is 15  
[Incubating] Problems report is available at: file:///C:/Users/cool/Devops/maven-example/build/reports/problems/problems-report.html  
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.  
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.  
For more on this, please refer to https://docs.gradle.org/8.12.1/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.  
  
BUILD SUCCESSFUL in 2s  
2 actionable tasks: 1 executed, 1 up-to-date  
Configuration cache entry stored.
```

## Step 7: Verify the Migration

- **Compare the Output:** Make sure that both the Maven and Gradle builds produce the same output:
  - **Maven Output:**

Hello, Maven

This is the simple realworld example....

Sum of 5 and 10 is 15

- **Gradle Output:**

Hello, Maven

This is the simple realworld example....

Sum of 5 and 10 is 15

## **5.Introduction to Jenkins: What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use**

### **Introduction to Jenkins**

#### **What is Jenkins?**

Jenkins is an open-source automation server widely used in the field of Continuous Integration (CI) and Continuous Delivery (CD). It allows developers to automate the building, testing, and deployment of software projects, making the development process more efficient and reliable.

#### **Key features of Jenkins:**

- **CI/CD:** Jenkins supports Continuous Integration and Continuous Deployment, allowing developers to integrate code changes frequently and automate the deployment of applications.
- **Plugins:** Jenkins has a vast library of plugins that can extend its capabilities. These plugins integrate Jenkins with version control systems (like Git), build tools (like Maven or Gradle), testing frameworks, deployment tools, and much more.
- **Pipeline as Code:** Jenkins allows the creation of pipelines using Groovy-based DSL scripts or YAML files, enabling version-controlled and repeatable pipelines.
- **Cross-platform:** Jenkins can run on various platforms such as Windows, Linux, macOS, and others.

#### **Installing Jenkins**

Jenkins can be installed on local machines, on a cloud environment, or even in containers. Here's how you can install Jenkins in Window local System environments:

## 1. Installing Jenkins Locally

### Step-by-Step Guide (Window):

#### 1. Prerequisites:

- Ensure that **Java (JDK) is installed** on your system. **Jenkins requires Java 21**. If not then [click here](#).
- You can check if Java is installed by running **java -version** in the terminal.

#### 2. Install Jenkins on Window System:

Run the WAR file

The Jenkins Web application ARchive(WAR) file can be started from the command line like this:

- Download the Jenkins Windows installer from the [official Jenkins website](#).
- Save Downloaded WAR file in proper folder(create folder called Jenkins in C drive and save Jenkins.war file)
- After then use **default port** or you can **configure you own port like I'm using port 9090** using following command

Run the command

```
java -jar jenkins.war
```

(OR)

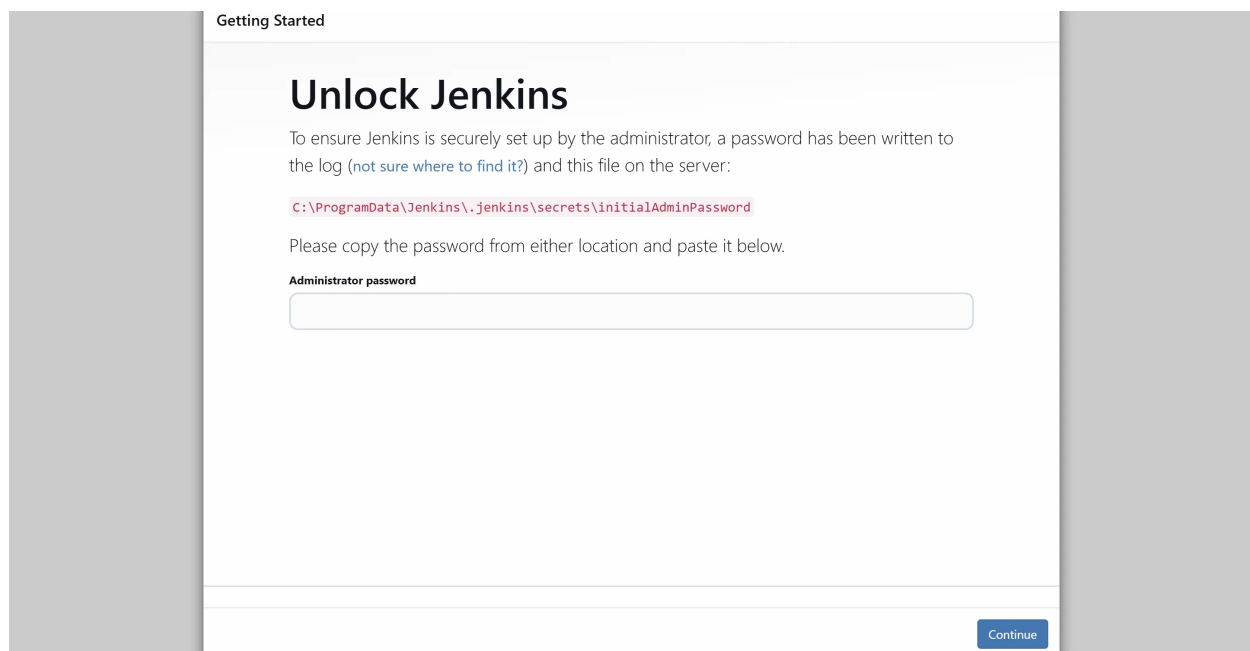
```
java -jar jenkins.war --httpPort=9090
```

- After successfully installed, Jenkins will be **running on port either default port or chosen port** like i choose **port 9090** by default (you can access it in your browser at **http://localhost:8080**) or

- 

## 2. Jenkins Setup in browser:

- After opening browser by visiting your local address the browser should look like below screenshot.



- It will ask **administrator password** so you have to **navigate the above highlighted path** and open that **initialAdminPassword** in **notepad** or any software to **see the password** or password displayed in the command prompt like bellow.

```

2025-04-20 07:46:11.914+0000 [id=41] INFO jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
2025-04-20 07:46:11.924+0000 [id=41] INFO jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs updated
2025-04-20 07:46:12.032+0000 [id=61] INFO hudson.util.Retrier#start: Attempt #1 to do the action check updates server
2025-04-20 07:46:12.694+0000 [id=42] INFO jenkins.install.SetupWizard#init:

*****
*****
*****

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

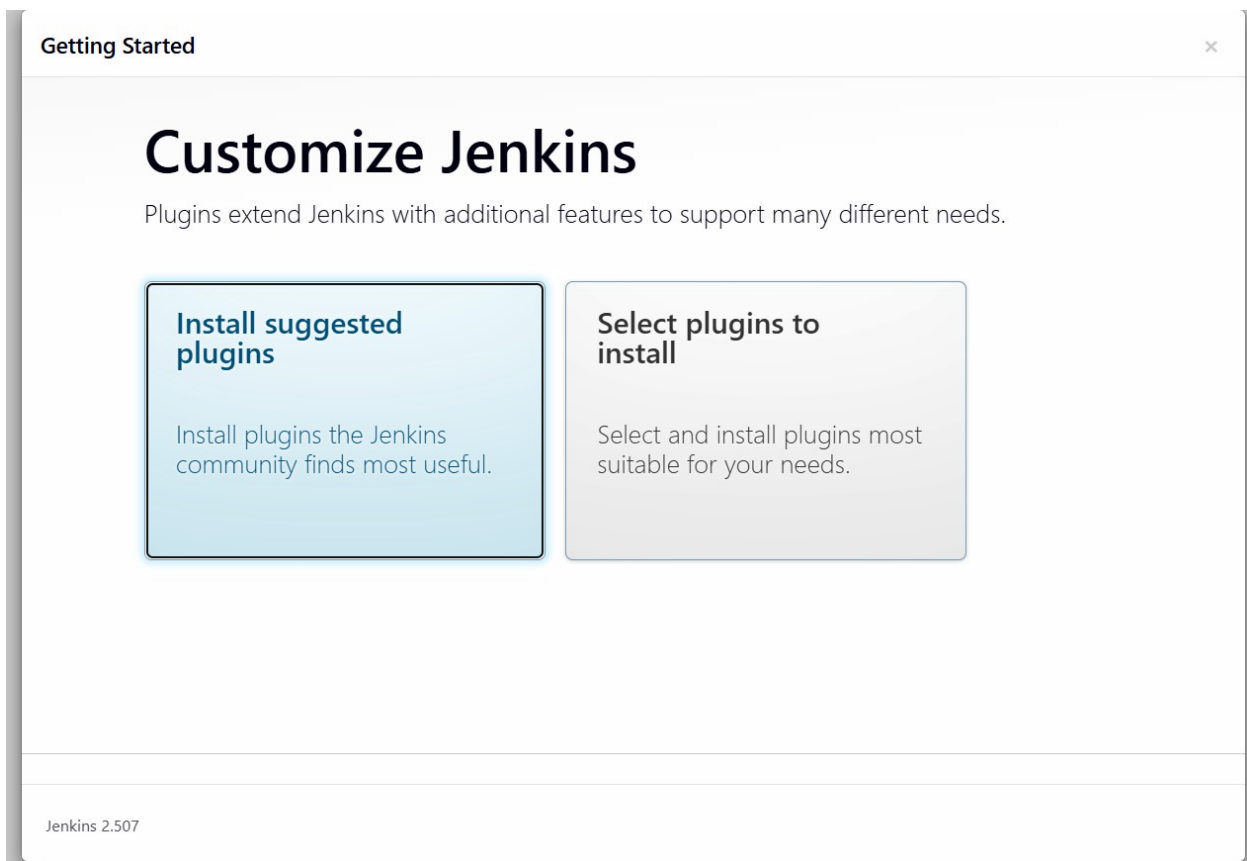
f6f3579d39e84a6b912dd1f5c111b4ec

This may also be found at: C:\Users\cool\.jenkins\secrets\initialAdminPassword

*****
*****
*****

```

- Just copy that password and paste it and click on continue.
- It will ask to **customize Jenkins** so click on **install suggested plugin** it will **automatically install all required plugin**.





- After then **create admin profile** by **filling all details** then **click on save and continue** after then **save and finish** after then **click on start using Jenkins**.

Getting Started

## Create First Admin User

Username

Password

Confirm password


Full name

E-mail address

Jenkins 2.507

Skip and continue as admin

Save and Continue


 Jenkins


vtucircle


log out

Dashboard >

+ New Item

 Build History

 Manage Jenkins

 My Views

Build Queue

No builds in the queue.

Build Executor Status

0/2

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

Set up a distributed build

Set up an agent

Configure a cloud

Learn more about distributed builds

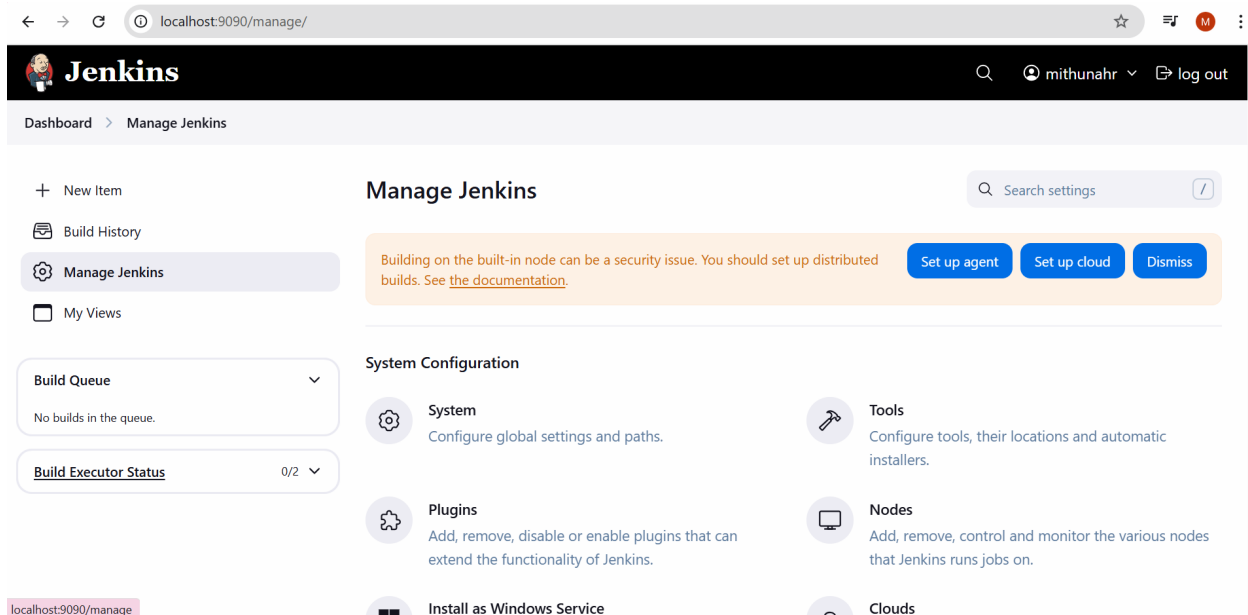
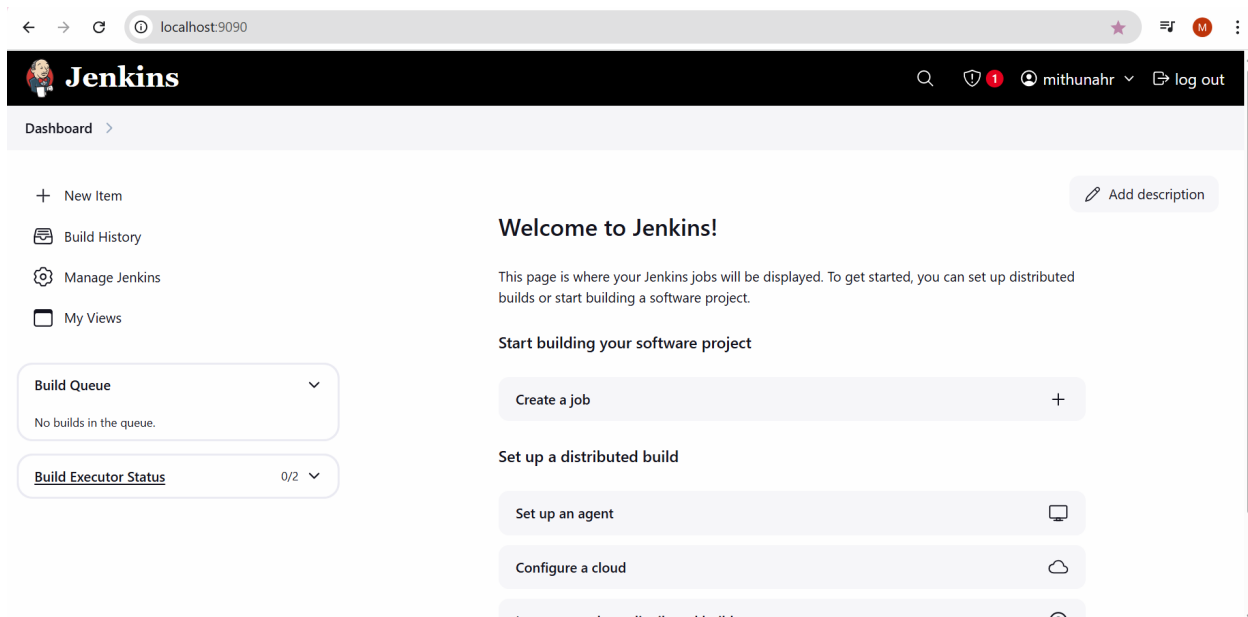
Add description

REST API

Jenkins 2.492.1

**Note:** Refer below link for Jenkins installation <https://www.jenkins.io/doc/>

## 6. Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests



The screenshot shows the Jenkins 'Plugins' page. The top navigation bar includes the Jenkins logo, a search icon, a shield icon with a red '1', the user 'mithunahr', and a 'log out' button. The breadcrumb trail is 'Dashboard > Manage Jenkins > Plugins'. On the left, a sidebar lists 'Updates', 'Available plugins' (highlighted), 'Installed plugins', 'Advanced settings', and 'Download progress'. The main content area has a search bar with 'maven integration' and an 'Install' button. Below the search bar is a table with columns 'Install', 'Name', and 'Released'. Two plugins are listed: 'Maven Integration 3.25' (checked for installation, released 2 months ago) and 'Pipeline Maven Integration 1508.v347c4b\_692202' (unchecked, released 1 month ago).

Install	Name	Released
<input checked="" type="checkbox"/>	<a href="#">Maven Integration</a> 3.25 <a href="#">Build Tools</a> This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTS as well as the automated configuration of various Jenkins publishers such as Junit.	2 mo 12 days ago
<input type="checkbox"/>	<a href="#">Pipeline Maven Integration</a> 1508.v347c4b_692202 <a href="#">pipeline</a> <a href="#">Maven</a> This plugin provides integration with Pipeline, configures maven environment to use within a pipeline job by calling sh mvn or bat mvn. The selected maven installation will be configured and prepended to the path.	1 mo 10 days ago

Click on install

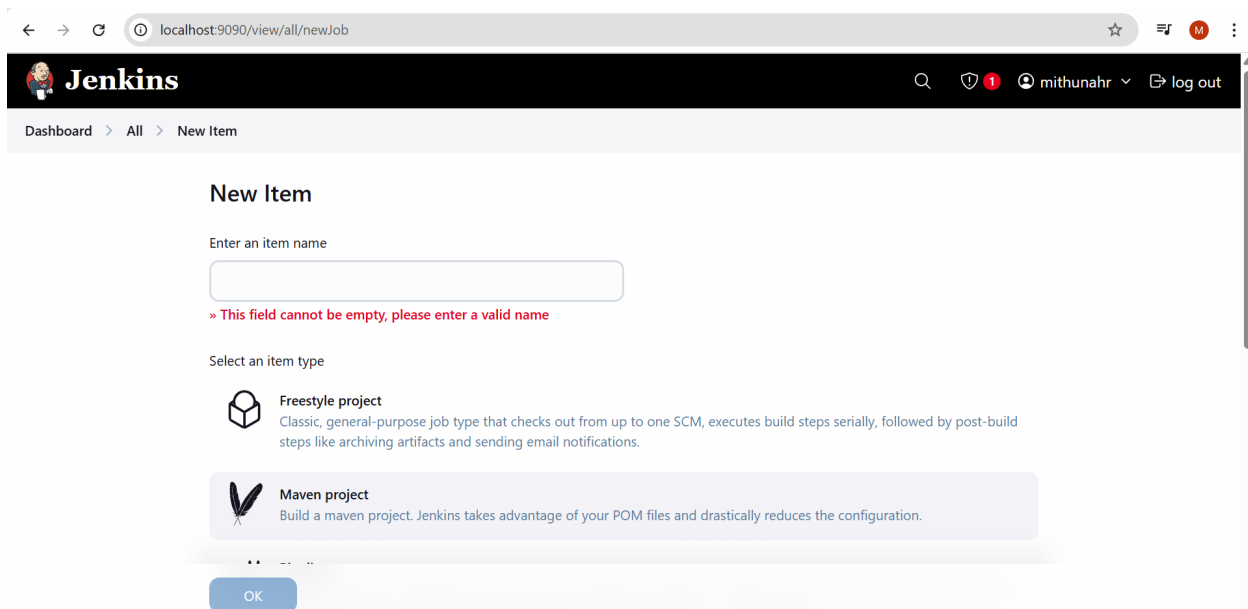
After the installation check in installed plugin

The screenshot shows the Jenkins 'Plugins' page after installation. The top navigation bar is the same. The breadcrumb trail is 'Dashboard > Manage Jenkins > Plugins'. The sidebar is the same, with 'Installed plugins' highlighted. The main content area has a search bar with 'maven'. Below the search bar is a table with columns 'Name' and 'Enabled'. One plugin is listed: 'Maven Integration plugin 3.25'. The 'Enabled' column shows a toggle switch turned on and a red 'X' icon.

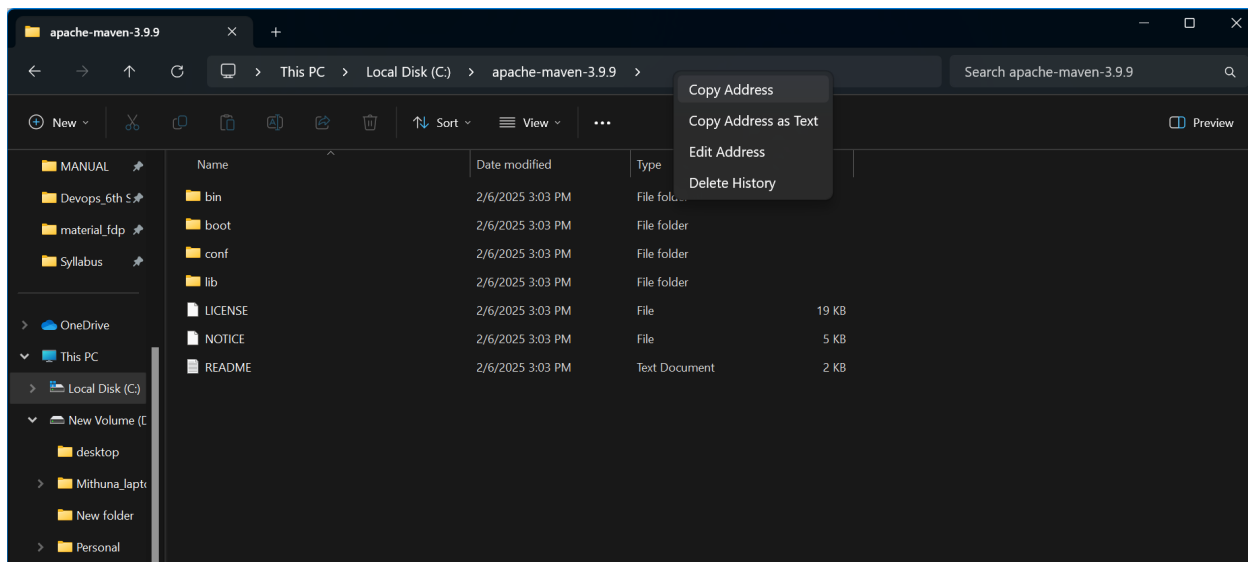
Name	Enabled
<a href="#">Maven Integration plugin</a> 3.25 This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTS as well as the automated configuration of various Jenkins publishers such as Junit. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>

Restart Jenkins (logout and login again)

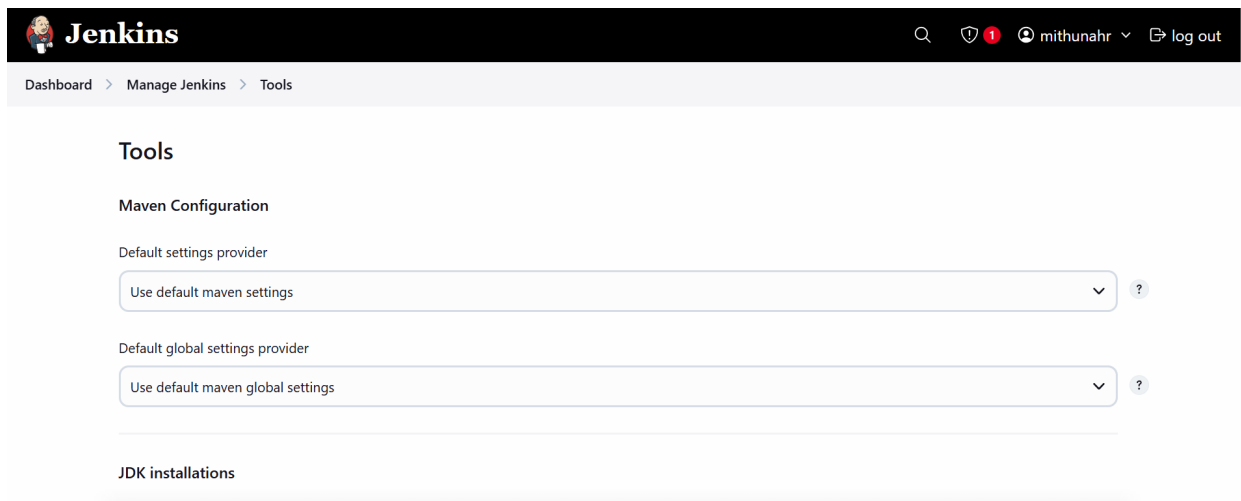
And after login click on New Item and find Maven project



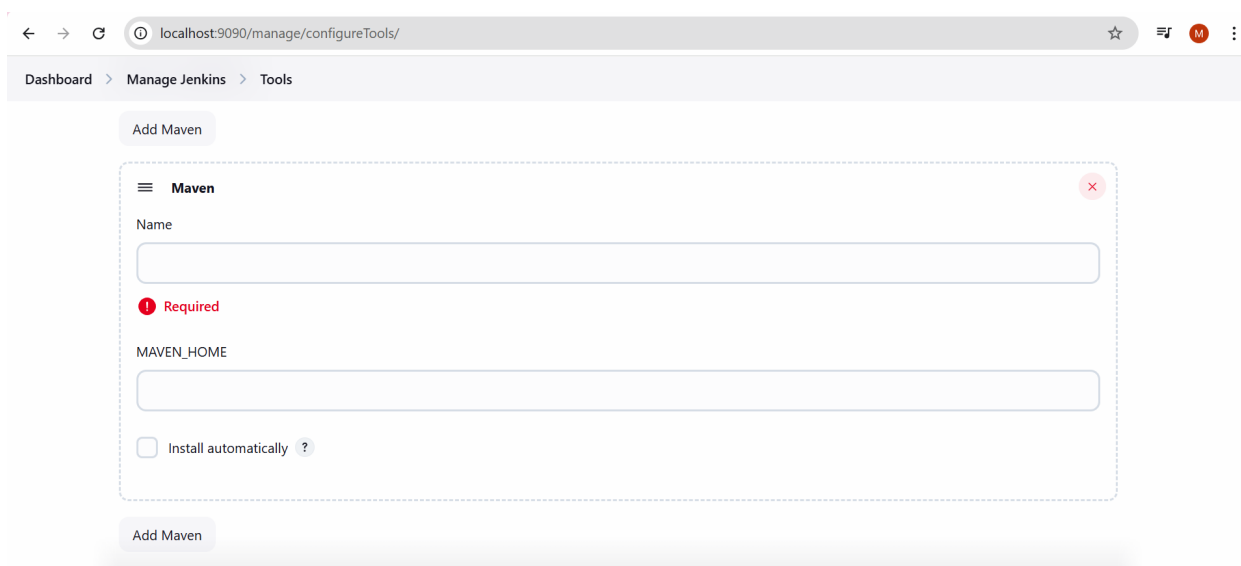
Navigate to C:\apache-maven-3.9.9 (installed path) copy the address



Now back to Jenkins and navigate to Manage Jenkins->Tools



Scroll down goto **Maven installation** and click on **Add Maven** unchecked **install automatically**(because we already installed maven)



And paste apache maven path in MAVEN\_HOME like below

localhost:9090/manage/configureTools/

Dashboard > Manage Jenkins > Tools

### JDK installations

JDK installations ^ Edited

Add JDK

**JDK**

Name

jdk-21

JAVA\_HOME

C:\Program Files\Java\jdk-21

☐ Install automatically ?

Save Apply

localhost:9090/manage/configureTools/

Dashboard > Manage Jenkins > Tools

Add Maven

**Maven**

Name

Maven-3.9.9

MAVEN\_HOME

C:\apache-maven-3.9.9

☐ Install automatically ?

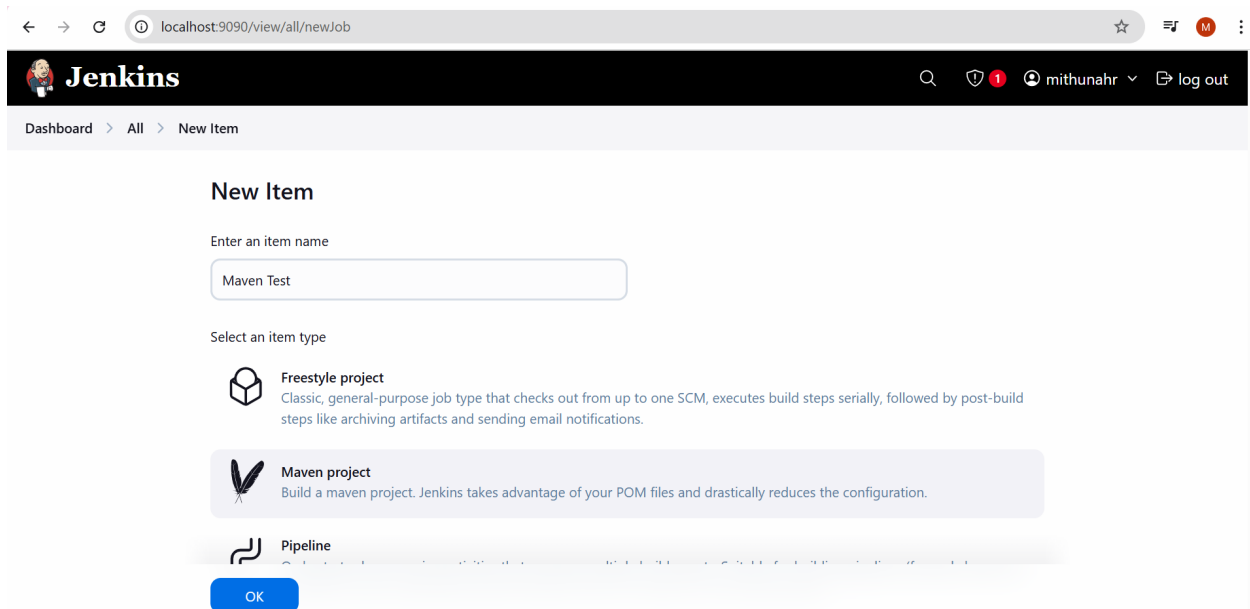
Add Maven

Click on Save

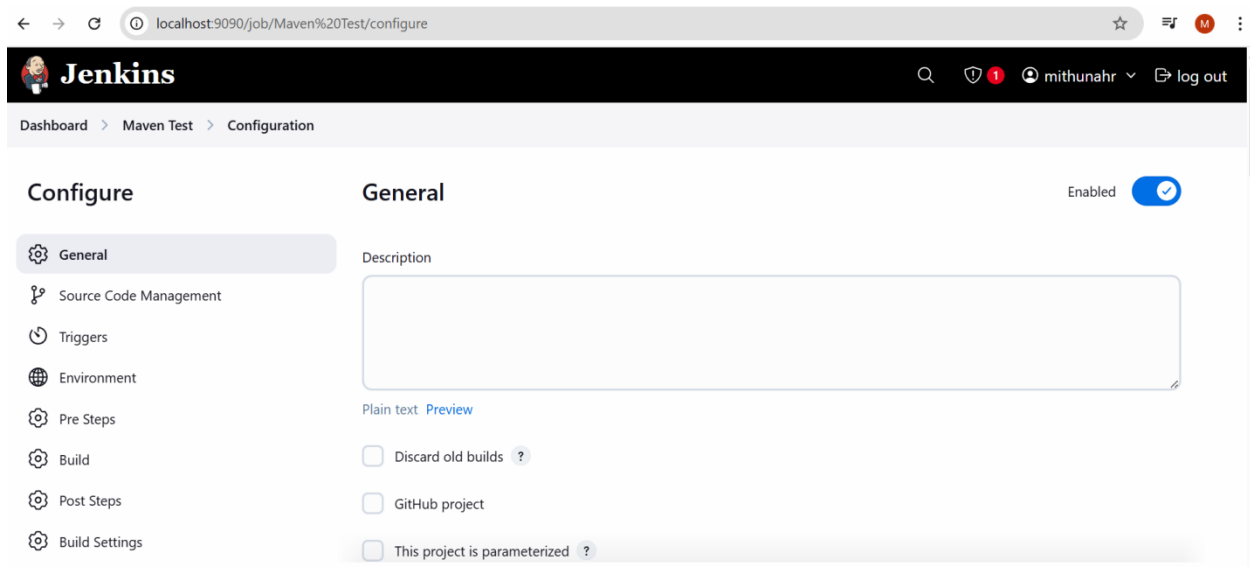
Now we can create Maven project with jenkins

Goto **New item**

Select **Maven Project** and give project name as **Maven Test**



Click on Ok



## Configure

- General
- Source Code Management**
- Triggers
- Environment
- Pre Steps
- Build
- Post Steps
- Build Settings
- Post-build Actions

Connect and manage your code repository to automatically pull the latest code for your builds.

☐ None

☒ Git ?

Repositories ?

Repository URL ?

C:\Users\cool\jenkins\workspace\Maven Test

Credentials ?

- none -

+ Add

Advanced ▾

## Configure

- General
- Source Code Management
- Triggers
- Environment
- Pre Steps
- Build**
- Post Steps
- Build Settings
- Post-build Actions

### Build

Root POM ?

pom.xml

Goals and options ?

install

Advanced ▾

### Post Steps

☐ Run only if build succeeds



7. Configuration Management with Ansible: Basics of Ansible: Inventory, Playbooks, and Modules, Automating Server Configurations with Playbooks, Hands-On: Writing and Running a Basic Playbook

## **Configuration Management with Ansible**

### **1. Basics of Ansible**

#### **What is Ansible?**

- An open-source IT automation tool.
- Used for configuration management, application deployment, orchestration, and task automation.
- Agentless – uses SSH to connect to target nodes.

#### **Why Ansible?**

- Simple YAML-based language.
- No agents required on client machines.
- Works on Linux/macOS and manages Windows via WinRM.

### **2. inventory in Ansible**

#### **What is an Inventory?**

- A file that lists the target hosts.
- Default: /etc/ansible/hosts, or custom using -i inventory.ini.

#### **Inventory File Example (inventory.ini)**

```
[web]
192.168.1.10

[db]
192.168.1.20

[local]
localhost ansible_connection=local
```

### 3. Ansible Modules

#### What are Modules?

- Building blocks of Ansible tasks.
- Do one job: install packages, copy files, start services, etc.

#### Commonly Used Modules

Module	Description
ping	Checks connection to a host
yum/apt	Installs packages
copy	Copies files to remote machine
command	Runs a shell command
service	Manages services (start/stop)

Example:

```
ansible localhost -m ping -i inventory.ini
```

## 4. Ansible Playbooks

### What is a Playbook?

- A YAML file that defines one or more tasks to be executed on target hosts.
- More powerful than ad-hoc commands.

### Playbook Structure:

```
---  
- name: Install and Start Nginx  
  hosts: web  
  become: yes  
  tasks:  
    - name: Install nginx  
      apt:  
        name: nginx  
        state: present  
        update_cache: yes  
  
    - name: Start nginx service  
      service:  
        name: nginx  
        state: started
```

## 5. Automating Server Configurations with Playbooks

### Use Cases:

- Install software (e.g., Apache, Java).
- Configure firewall rules.
- Set up users and permissions.

- Deploy applications.

#### **Best Practices:**

- Use variables and roles for modularity.
- Use handlers for conditional actions (e.g., restart service if config changes).
- Use become: yes to run as sudo.

## **6. Hands-On: Writing and Running a Basic Playbook**

❖ **Here are the steps to install Ansible on Ubuntu:**

### **✓ Step 1: Update the System**

```
sudo apt update  
sudo apt upgrade -y
```

### **✓ Step 2: Install Required Dependencies**

```
sudo apt install software-properties-common -y
```

### **✓ Step 3: Add the Ansible PPA Repository**

```
sudo add-apt-repository --yes --update ppa:ansible/ansible
```

### **✓ Step 4: Install Ansible**

```
sudo apt install ansible -y
```

## ✓ Step 5: Verify Ansible Installation

```
ansible --version
```

You should see output showing the Ansible version and location.

```
mithuna@DESKTOP-2ARMSRN:~$ ansible --version
ansible 2.10.8
  config file = None
  configured module search path = ['/home/mithuna/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
```

## ❖ Goal: Compile and Run Java Program

### 6.1 Inventory File (inventory.ini)

This defines your target system — in this case, it's the **local machine**:

```
sudo vi inventory.ini
```

```
[local]
```

```
localhost ansible_connection=local
```

### 6.2 HelloWorld.java:

Place this file in your **home directory** or where you'll run the playbook from:

```
sudo vi HelloWorld.java
```

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

### 6.3 Ansible Playbook (run-java.yml):

This playbook installs Java (if needed), compiles the Java program, runs it, and shows the output.

```
sudo vi run-java.yml
```

```
---  
- name: Compile and run HelloWorld.java in current directory  
  hosts: localhost  
  gather_facts: yes  
  tasks:  
    - name: Ensure Java is installed  
      apt:  
        name: default-jdk  
        state: present  
        become: yes  
  
    - name: Compile HelloWorld.java  
      command: javac HelloWorld.java  
      args:  
        chdir: "{{ ansible_env.PWD }}"  
  
    - name: Run HelloWorld
```

```
command: java HelloWorld
args:
  chdir: "{{ ansible_env.PWD }}"
register: output

- name: Display output
  debug:
    msg: "{{ output.stdout }}"
```

**Note: To check and correct YAML syntax errors online, follow these steps:**

✔ **Step 1: Choose an Online YAML Validator**

Here are some reliable online YAML validators:

- **YAML Lint:** Offers syntax validation and reformatting. [yamllint.com](https://yamllint.com)
- **YAML Checker:** Provides real-time error detection with line numbers. [YAML Checker - The Best YAML Validator](#)
- **YAML Validator by Code Beautify:** Supports URL-based validation and sharing. [Code BeautifyCode Beautify](#)

✔ **Step 2: Paste Your YAML Code**

Copy your YAML content and paste it into the input area of the chosen validator.

✔ **Step 3: Click "Go" or "Validate"**

Initiate the validation process by clicking the "Go" or "Validate" button. The tool will analyze your YAML for syntax errors. [Free Aspose Appsminifier.org](https://FreeAsposeAppsminifier.org)


✔ **Step 4: Review and Correct Errors**

If errors are found, the validator will highlight them with line numbers and descriptions. Make the necessary corrections in your YAML file.

### ✔ Step 5: Revalidate

After making corrections, paste the updated YAML into the validator and click "Go" or "Validate" again to ensure all errors are resolved.

Example:

 yamllint.com

## YAML Lint

Paste in your YAML and click "Go" - we'll tell you if it's valid or not, and give you a nice clean UTF-8 version of it.

```
1 ---
2 - name: nginx install & start services
3   hosts: all
4   become: true
5   tasks:
6     - name: install nginx
7       apt:
8         name: nginx
9         state: latest
10    - name: Start nginx manually (WSL workaround)
11      command: service nginx start
12      become: yes
13      warn: false
14
15
16
17
18
19
20
```

☒ Reformat (strips comments) ☒ Resolve aliases

Valid YAML!

**6.3 Run the playbook:** using following command(Ensure the files are all in the **same folder** and then run)

```
ansible-playbook -i inventory.ini run-java.yml
```



```
mithuna@DESKTOP-2ARMSRN:~$ ansible-playbook -i inventory.ini run-java.yml

PLAY [Compile and run HelloWorld.java in current directory] *****

TASK [Gathering Facts] *****
ok: [localhost]

TASK [Ensure Java is installed] *****
changed: [localhost]

TASK [Compile HelloWorld.java] *****
changed: [localhost]

TASK [Run HelloWorld] *****
changed: [localhost]

TASK [Display output] *****
ok: [localhost] => {
  "msg": "Hello, World!"
}

PLAY RECAP *****
localhost          : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```