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 help developers streamline the process of building, testing, and deploying software projects. They take care of repetitive tasks like compiling code, managing dependencies, and packaging applications, which makes development more efficient and error-free.

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

Aspect	**Maven**	**Gradle**
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

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 -v** 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 -v** 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.

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

1. **Create a maven project using command line**

   mvn archetype:generate -DgroupId=com.example -DartifactId=myapp -

   DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

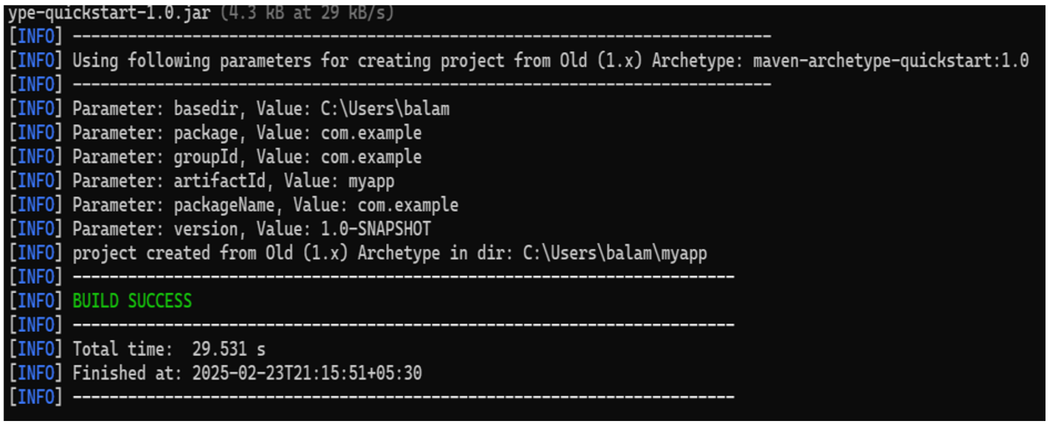
- **groupId:** A unique identifier for the group (usually the domain name).
- **artifactId:** A unique name for the project artifact (your project).
- **archetypeArtifactId:** The template you want to use for the project.
- **DinteractiveMode=false:** Disables prompts during project generation.

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

2. **Using IDEs**

Most modern IDEs (like IntelliJ IDEA or Eclipse) provide wizards to generate Maven projects. For example, in IntelliJ IDEA:

1. Go to **File > New Project**.
2. Choose **Maven** from the list of project types.
3. Provide the **groupId** and **artifactId** for your project.

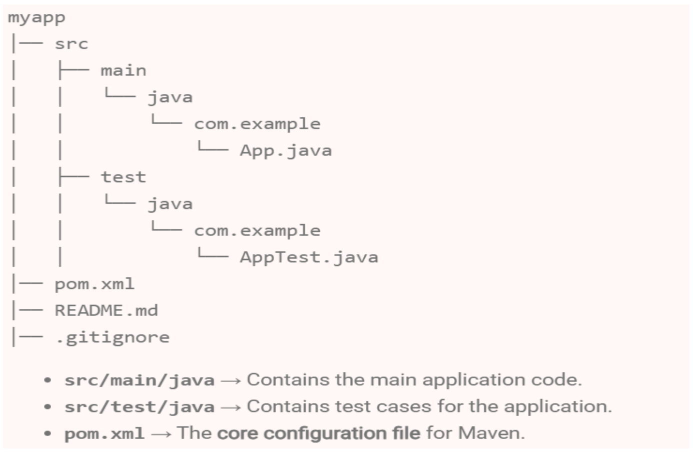


```
ype-quickstart-1.0.jar (4.3 kB at 29 kB/s)
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: C:\Users\balam
[INFO] Parameter: package, Value: com.example
[INFO] Parameter: groupId, Value: com.example
[INFO] Parameter: artifactId, Value: myapp
[INFO] Parameter: packageName, Value: com.example
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\balam\myapp
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  29.531 s
[INFO] Finished at: 2025-02-23T21:15:51+05:30
[INFO] ------------------------------------------------------------------------
```

Navigate to the Project Folder

cd myapp

Verify the Project Structure

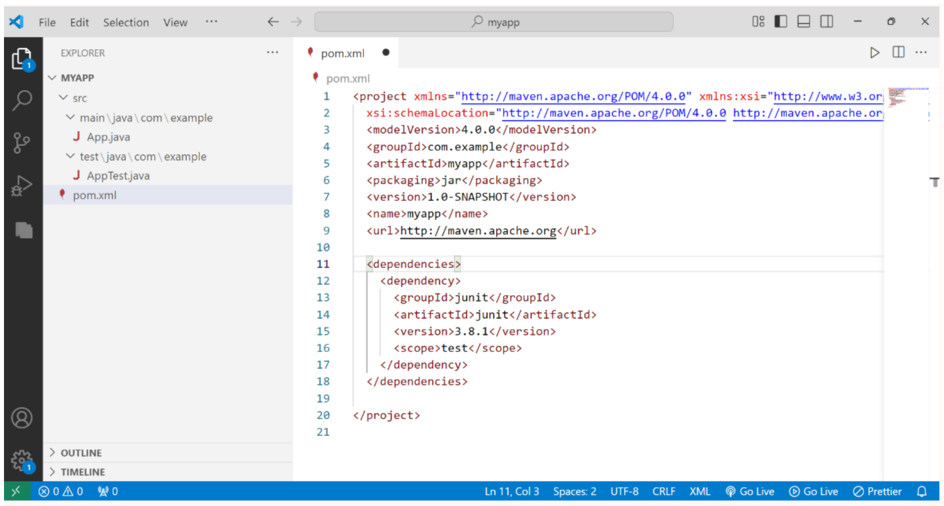


```
myapp
|── src
|   ├── main
|   |   └── java
|   |       └── com.example
|   |           └── App.java
|   ├── test
|   |   └── java
|   |       └── com.example
|   |           └── AppTest.java
|── pom.xml
|── README.md
|── .gitignore
```

- `src/main/java` → Contains the main application code.
- `src/test/java` → Contains test cases for the application.
- `pom.xml` → The **core configuration file** for Maven.

3: 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:



pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.or
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.or
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>myapp</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>myapp</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

- **<version> -** Version of the project (e.g., 1.0-SNAPSHOT).
- **<dependencies>** - Lists external libraries required for the project.
- **<build> -** Specifies plugins for compiling, packaging, and testing.

- **Managing dependencies in Maven**
  - To use Spring Boot, add the following inside the <dependencies> section:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
```

```
    <version>2.7.0</version>

</dependency>
```

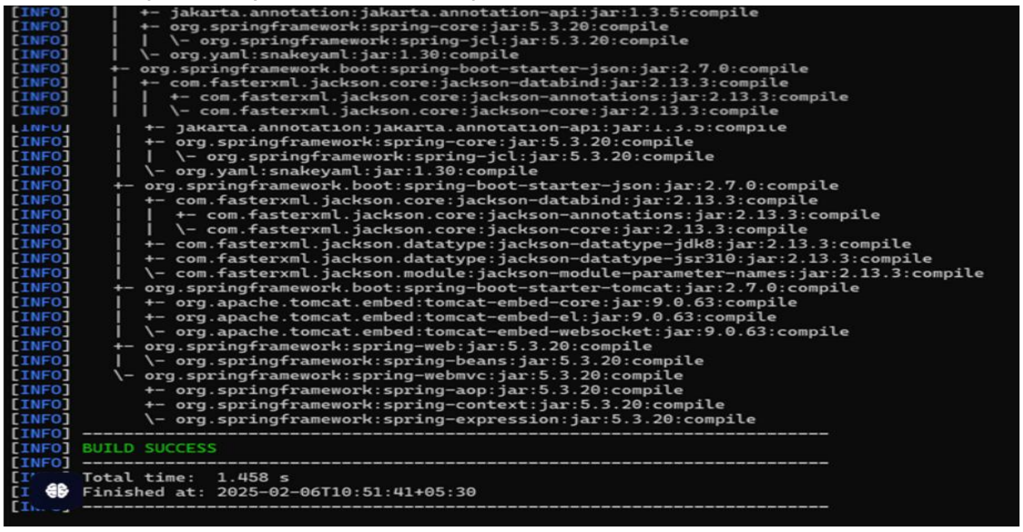
- After adding dependencies, run:

```
mvn clean install
```

Installs required .jar files

Archetype ID	Description
`maven-archetype-quickstart`	Creates a **basic Java project** with a `pom.xml` file, `src/main/java`, and `src/test/java`.
`maven-archetype-webapp`	Generates a **Java web application** structure with `WEB-INF/web.xml`.
`maven-archetype-j2ee-simple`	Creates a **basic J2EE project** (JSP, Servlets) with minimal setup.
`maven-archetype-site`	Generates a **Maven site project**, used for documentation generation (`mvn site`).
`maven-archetype-plugin`	Creates a **custom Maven plugin project** with necessary configurations.
`maven-archetype-portlet`	Generates a **Java Portlet project** for use in portals like Liferay.
`maven-archetype-archetype`	Used to **create a new custom archetype** (template for new projects).

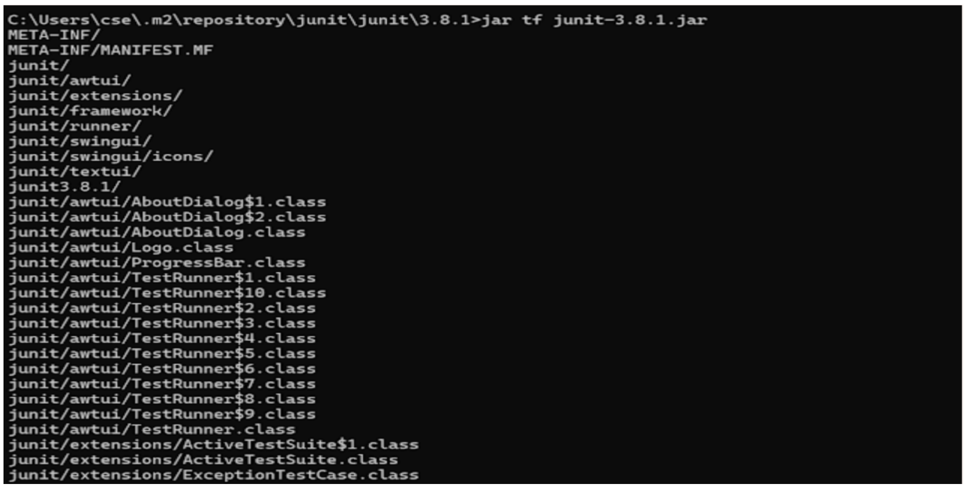
- **Use mvn dependency:tree to list all dependencies**



```
[INFO]     |  +- jakarta.annotation:jakarta.annotation-api:jar:1.3.5:compile
[INFO]     |  +- org.springframework:spring-core:jar:5.3.20:compile
[INFO]     |  |  \- org.springframework:spring-jcl:jar:5.3.20:compile
[INFO]     |  \- org.yaml:snakeyaml:jar:1.30:compile
[INFO]     +- org.springframework.boot:spring-boot-starter-json:jar:2.7.0:compile
[INFO]     |  +- com.fasterxml.jackson.core:jackson-databind:jar:2.13.3:compile
[INFO]     |  |  +- com.fasterxml.jackson.core:jackson-annotations:jar:2.13.3:compile
[INFO]     |  |  \- com.fasterxml.jackson.core:jackson-core:jar:2.13.3:compile
[INFO]     |  +- jakarta.annotation:jakarta.annotation-api:jar:1.3.5:compile
[INFO]     |  +- org.springframework:spring-core:jar:5.3.20:compile
[INFO]     |  |  \- org.springframework:spring-jcl:jar:5.3.20:compile
[INFO]     |  \- org.yaml:snakeyaml:jar:1.30:compile
[INFO]     +- org.springframework.boot:spring-boot-starter-json:jar:2.7.0:compile
[INFO]     |  +- com.fasterxml.jackson.core:jackson-databind:jar:2.13.3:compile
[INFO]     |  |  +- com.fasterxml.jackson.core:jackson-annotations:jar:2.13.3:compile
[INFO]     |  |  \- com.fasterxml.jackson.core:jackson-core:jar:2.13.3:compile
[INFO]     |  +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.13.3:compile
[INFO]     |  +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.13.3:compile
[INFO]     |  \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.13.3:compile
[INFO]     +- org.springframework.boot:spring-boot-starter-tomcat:jar:2.7.0:compile
[INFO]     |  +- org.apache.tomcat.embed:tomcat-embed-core:jar:9.0.63:compile
[INFO]     |  +- org.apache.tomcat.embed:tomcat-embed-el:jar:9.0.63:compile
[INFO]     |  \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:9.0.63:compile
[INFO]     +- org.springframework:spring-web:jar:5.3.20:compile
[INFO]     |  \- org.springframework:spring-beans:jar:5.3.20:compile
[INFO]     \- org.springframework:spring-webmvc:jar:5.3.20:compile
[INFO]        +- org.springframework:spring-aop:jar:5.3.20:compile
[INFO]        +- org.springframework:spring-context:jar:5.3.20:compile
[INFO]        \- org.springframework:spring-expression:jar:5.3.20:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[I     Total time:  1.458 s
[I     Finished at: 2025-02-06T10:51:41+05:30
[I     ------------------------------------------------------------------------
```

- **open .jar files using an archive tool or the jar command**

  jar tf junit-3.8.1.jar



```
C:\Users\cse\.m2\repository\junit\junit\3.8.1>jar tf junit-3.8.1.jar
META-INF/
META-INF/MANIFEST.MF
junit/
junit/awtui/
junit/extensions/
junit/framework/
junit/runner/
junit/swingui/
junit/swingui/icons/
junit/textui/
junit3.8.1/
junit/awtui/AboutDialog$1.class
junit/awtui/AboutDialog$2.class
junit/awtui/AboutDialog.class
junit/awtui/Logo.class
junit/awtui/ProgressBar.class
junit/awtui/TestRunner$1.class
junit/awtui/TestRunner$10.class
junit/awtui/TestRunner$2.class
junit/awtui/TestRunner$3.class
junit/awtui/TestRunner$4.class
junit/awtui/TestRunner$5.class
junit/awtui/TestRunner$6.class
junit/awtui/TestRunner$7.class
junit/awtui/TestRunner$8.class
junit/awtui/TestRunner$9.class
junit/awtui/TestRunner.class
junit/extensions/ActiveTestSuite$1.class
junit/extensions/ActiveTestSuite.class
junit/extensions/ExceptionTestCase.class
```

- **Maven Plugins**
  Maven plugins automate tasks like compiling code, running tests, and packaging the application

Plugin	Purpose
maven-compiler-plugin	Compiles Java code
maven-surefire-plugin	Runs unit tests
maven-jar-plugin	Packages the project as a JAR file

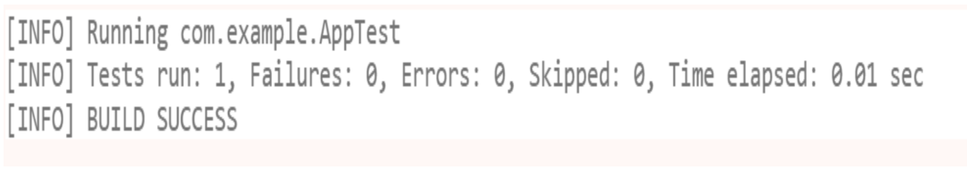
- **Writing a simple java program**

```java
package com.example;

public class App {
    public static String greet() {
        return "Hello, DevOps!";
    }

    public static void main(String[] args) {
        System.out.println(greet());
    }
}
```

- **Modifying a Unit Test – AppTest.java**

```java
package com.example;

import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class AppTest {
    @Test
    public void testGreet() {
        assertEquals("Hello, DevOps!", App.greet());
    }
}
```

- **Running Maven Commands**
  - **mvn compile**
  - **mvn test**



```
[INFO] Running com.example.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.01 sec
[INFO] BUILD SUCCESS
```

  - **mvn package**
  - **java -jar target/myapp-1.0-SNAPSHOT.jar**

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 Program 1 **click here**
- **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.

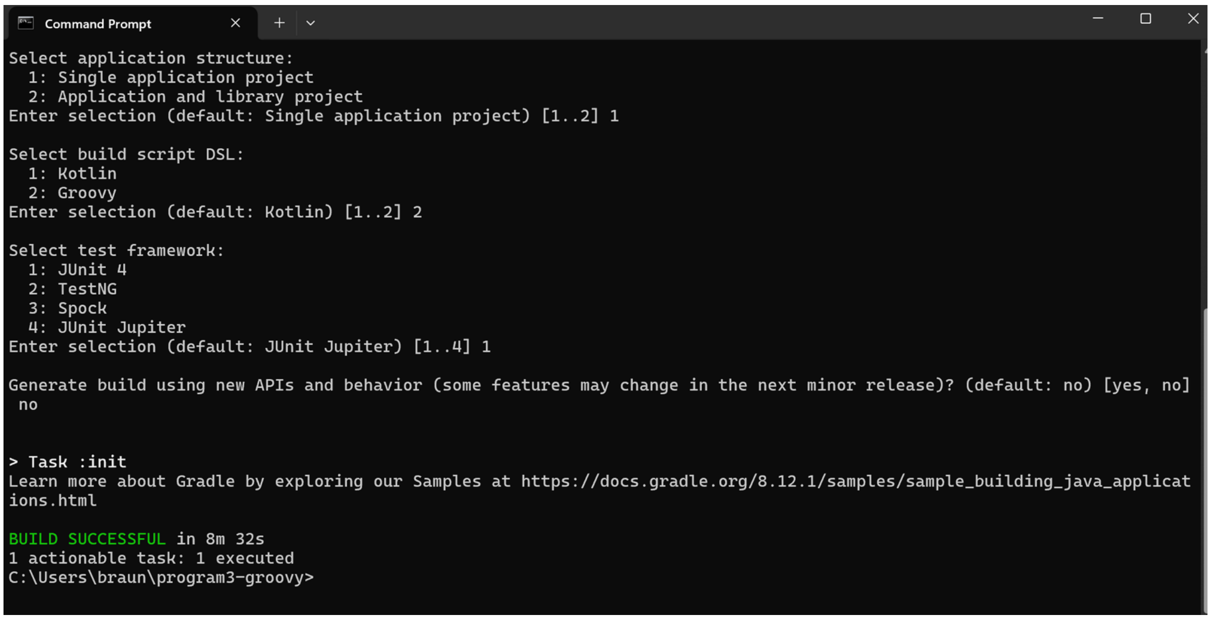
Working with Gradle Project (Groovy DSL):

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



```
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


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

BUILD SUCCESSFUL in 8m 32s
1 actionable task: 1 executed
C:\Users\braun\program3-groovy>
```

Step 2: build.gradle (Groovy DSL)

```
plugins {
   id 'application'
}
application {
   mainClass = 'com.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 com.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.

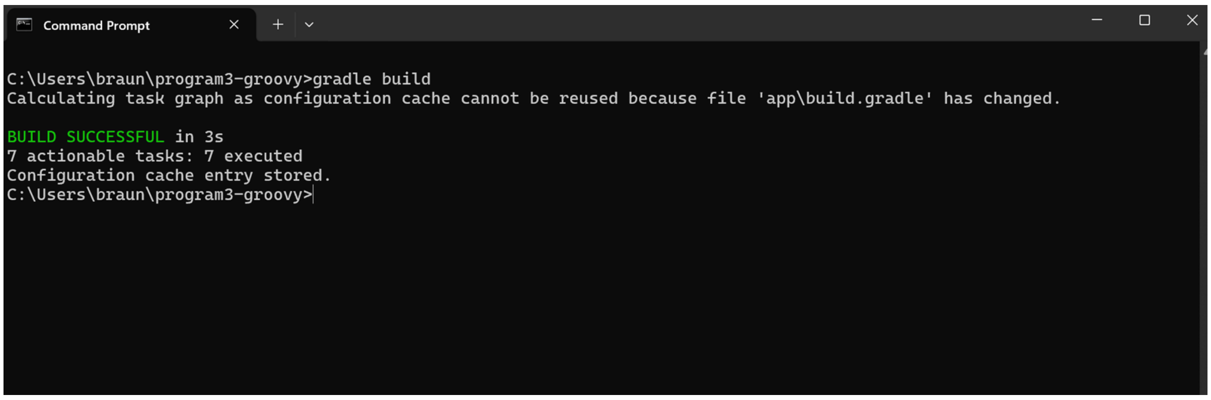
```
package com.example;
import org.junit.Test;
import static org.junit.Assert.*;
public class AdditionOperationTest {
  @Test
  public void testAddition() {
    double num1 = 5;
```

```
            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:

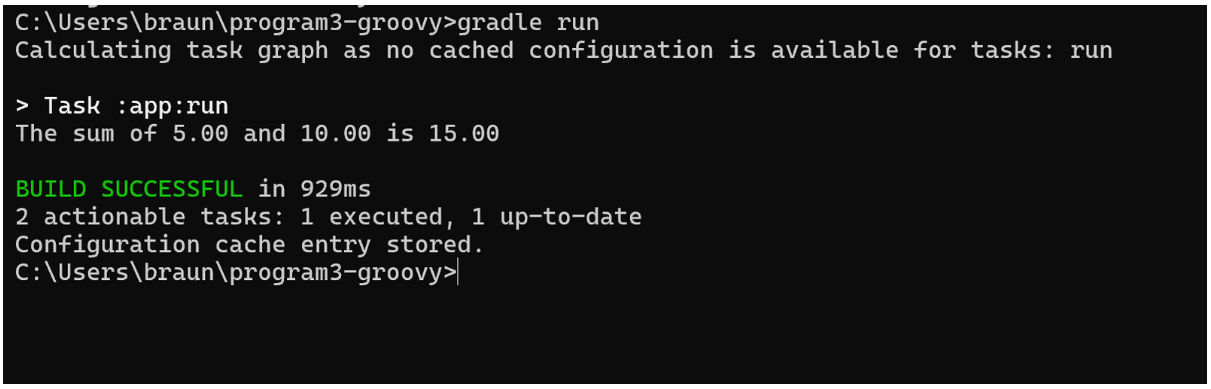
```
gradle build
```



```
C:\Users\braun\program3-groovy>gradle build
Calculating task graph as configuration cache cannot be reused because file 'app\build.gradle' has changed.

BUILD SUCCESSFUL in 3s
7 actionable tasks: 7 executed
Configuration cache entry stored.
C:\Users\braun\program3-groovy>
```

- To **run** the project:

```
gradle run
```



```
C:\Users\braun\program3-groovy>gradle run
Calculating task graph as no cached configuration is available for tasks: run

> Task :app:run
The sum of 5.00 and 10.00 is 15.00

BUILD SUCCESSFUL in 929ms
2 actionable tasks: 1 executed, 1 up-to-date
Configuration cache entry stored.
C:\Users\braun\program3-groovy>
```

- To **test** the project:
  - gradle test

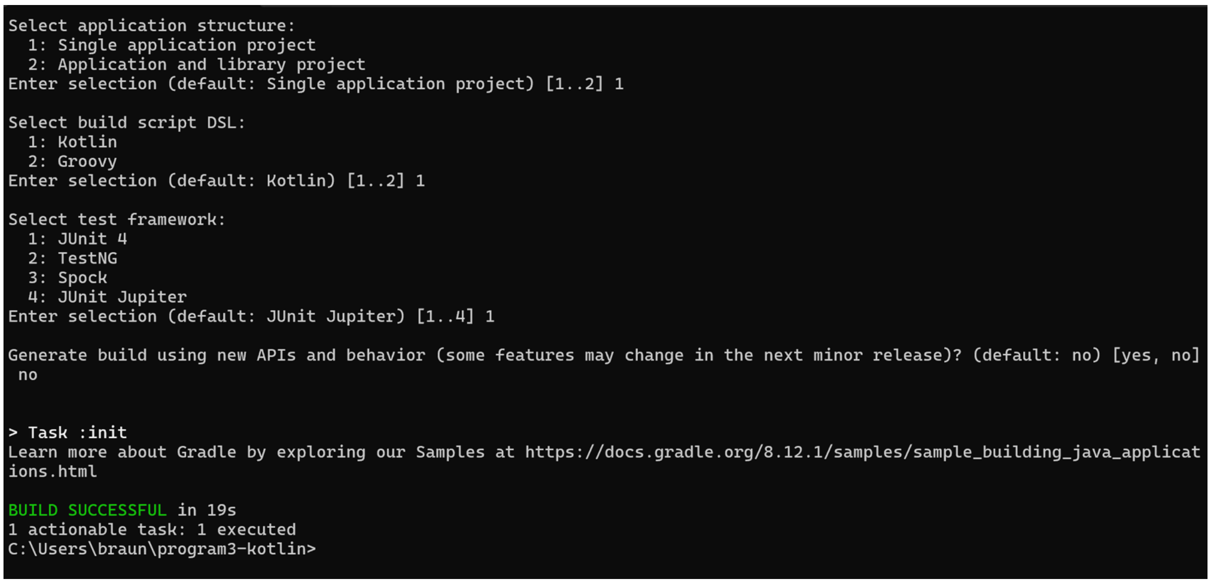
Working with Gradle Project (Kotlin DSL):

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_applicat
ions.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("com.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 com.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")
}
```

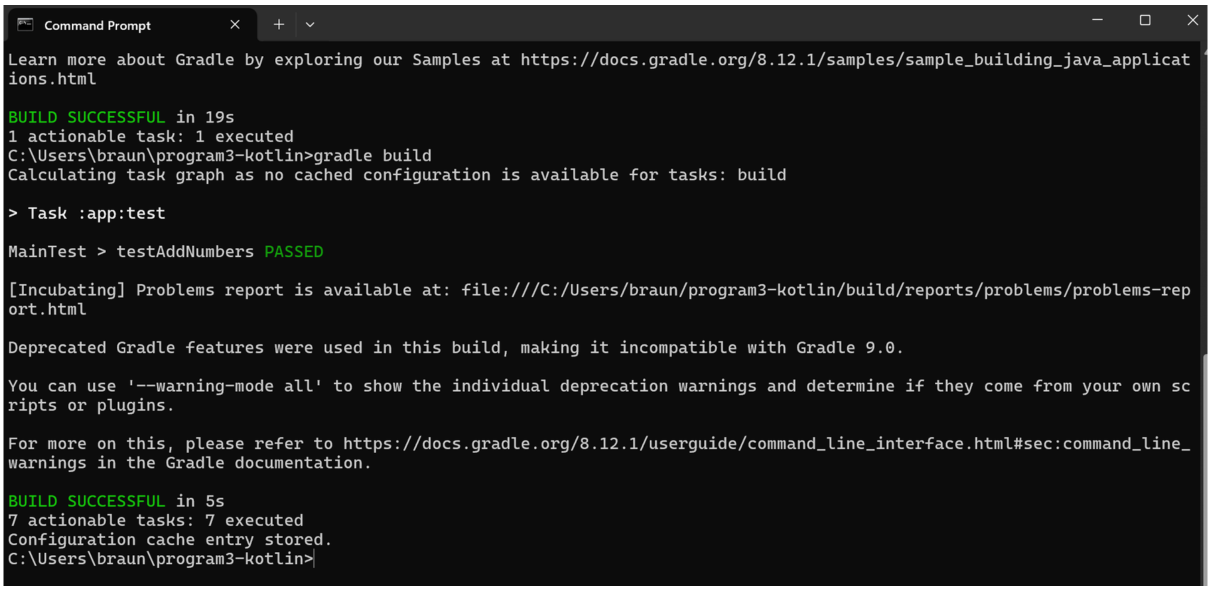
Step 4: MainTest.kt (JUnit Test) (Change file name and update below code)

- 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 com.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

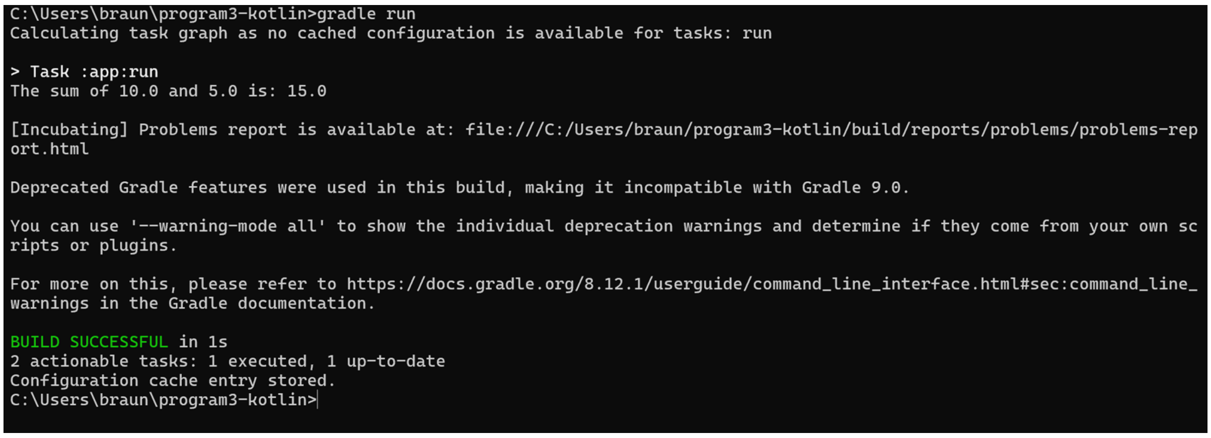
- To **build** the project: gradle build



```
Learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.12.1/samples/sample_building_java_applicat
ions.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-rep
ort.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 sc
ripts 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:

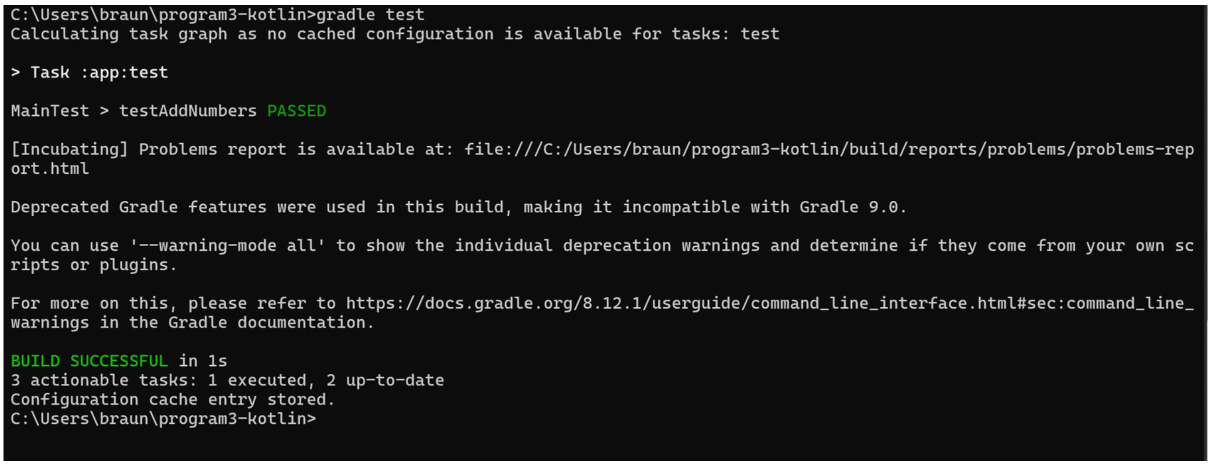
  gradle run



```
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-rep
ort.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 sc
ripts 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



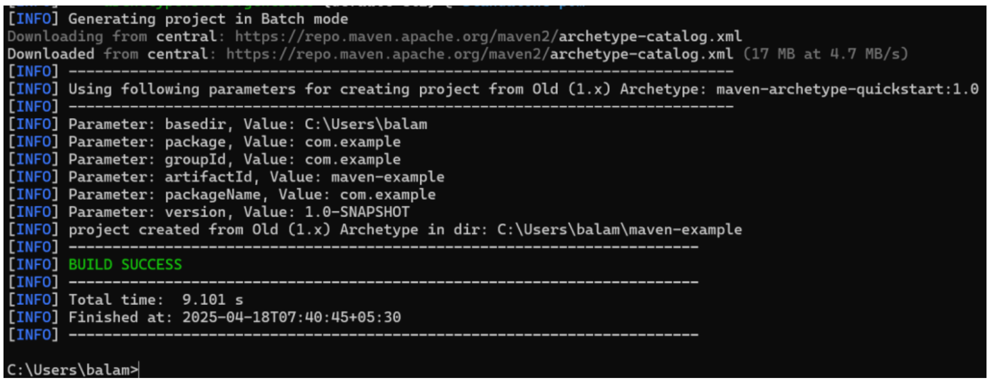
```
C:\Users\braun\program3-kotlin>gradle test
Calculating task graph as no cached configuration is available for tasks: test

> Task :app:test

MainTest > testAddNumbers PASSED

[Incubating] Problems report is available at: file:///C:/Users/braun/program3-kotlin/build/reports/problems/problems-rep
ort.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 sc
ripts 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>
```

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

Step 1: Creating a Maven Project

- **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



```
[INFO] Generating project in Batch mode
Downloading from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml
Downloaded from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml (17 MB at 4.7 MB/s)
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: C:\Users\balam
[INFO] Parameter: package, Value: com.example
[INFO] Parameter: groupId, Value: com.example
[INFO] Parameter: artifactId, Value: maven-example
[INFO] Parameter: packageName, Value: com.example
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\balam\maven-example
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.101 s
[INFO] Finished at: 2025-04-18T07:40:45+05:30
[INFO] ------------------------------------------------------------------------

C:\Users\balam>
```

Step 2: Open The pom.xml File

- **You can manually navigate the project folder named call maven-example "C:\Users\balam\maven-example" and open the file pom.xml and copy the below code and paste it then save it.**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.2.2</version>  <!-- Use latest version -->
      <configuration>
        <archive>
          <manifest>
            <mainClass>com.example.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
    </plugins>
</build>
```

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

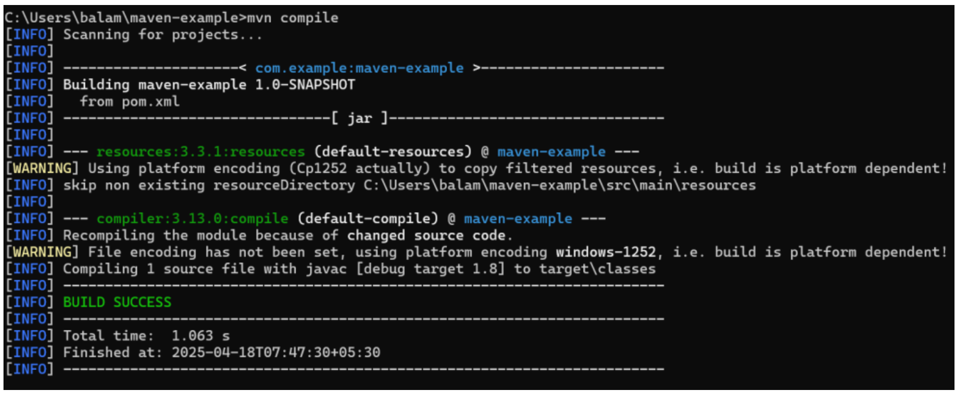
- **Open a file App.java inside the src/main/java/com/example/ directory.**
- **After opening the App.java copy the below code and paste it in that file then save it.**

```java
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.

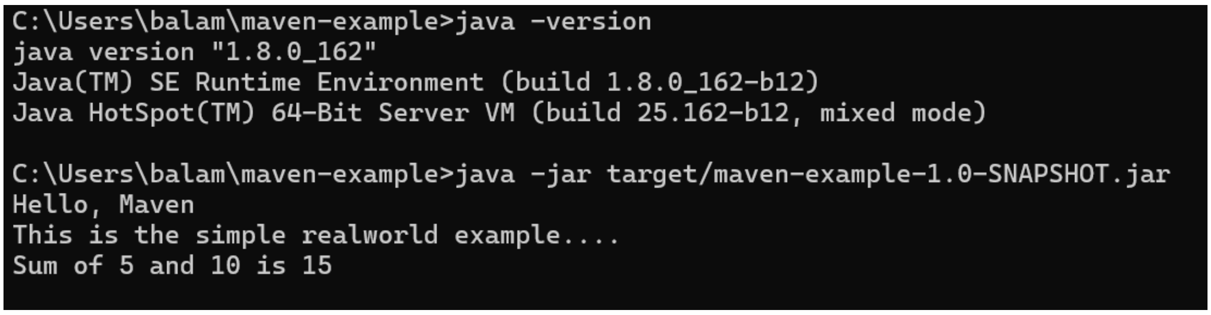
Step 4: Run the Project

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.**
- 

```
C:\Users\balam\maven-example>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< com.example:maven-example >---------------------
[INFO] Building maven-example 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ maven-example ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\Users\balam\maven-example\src\main\resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ maven-example ---
[INFO] Recompiling the module because of changed source code.
[WARNING] File encoding has not been set, using platform encoding windows-1252, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 1.8] to target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.063 s
[INFO] Finished at: 2025-04-18T07:47:30+05:30
[INFO] ------------------------------------------------------------------------
```

- **Execute mvn package**
- **And then java -jar target/maven-example-1.0-SNAPSHOT.jar**



```
C:\Users\balam\maven-example>java -version
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)

C:\Users\balam\maven-example>java -jar target/maven-example-1.0-SNAPSHOT.jar
Hello, Maven
This is the simple realworld example....
Sum of 5 and 10 is 15
```

Step 5: Migrate the Maven Project to Gradle

1. **Initialize Gradle: Navigate to the project directory (maven-example) and run:**

   gradle init –type pom

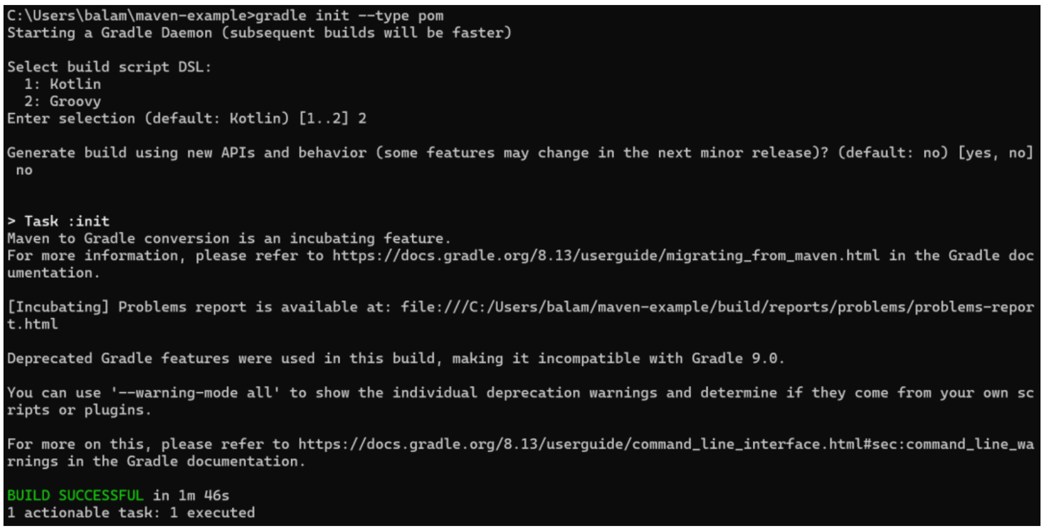
- **Select build script DSL:**

  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\balam\maven-example>gradle init --type pom
Starting a Gradle Daemon (subsequent builds will be faster)

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/8.13/userguide/migrating_from_maven.html in the Gradle doc
umentation.

[Incubating] Problems report is available at: file:///C:/Users/balam/maven-example/build/reports/problems/problems-repor
t.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 sc
ripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.13/userguide/command_line_interface.html#sec:command_line_wa
rnings in the Gradle documentation.

BUILD SUCCESSFUL in 1m 46s
1 actionable task: 1 executed
```

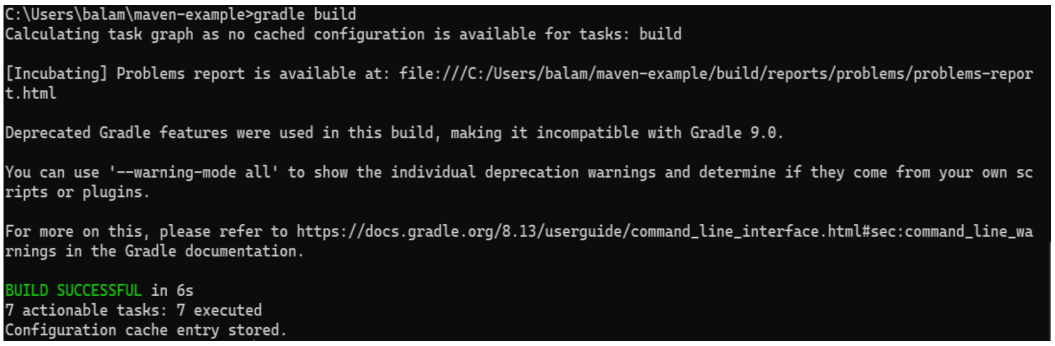
2. **Navigate the project folder and open build.gradle file "C:\Users\balam\maven-example\build.gradle" then add the below code and save it.**

```
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

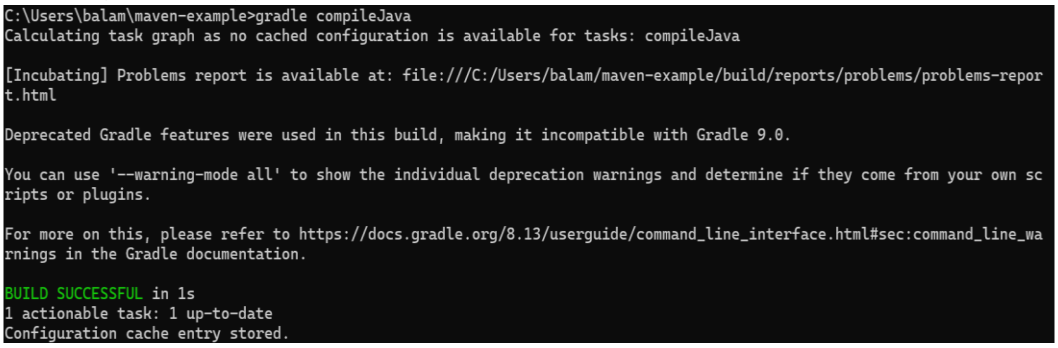
- **Build the Project: In the project directory (maven-example), run the below command to build the project:**

```
gradle build
```



```
C:\Users\balam\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/balam/maven-example/build/reports/problems/problems-repor
t.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 sc
ripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.13/userguide/command_line_interface.html#sec:command_line_wa
rnings in the Gradle documentation.

BUILD SUCCESSFUL in 6s
7 actionable tasks: 7 executed
Configuration cache entry stored.
```

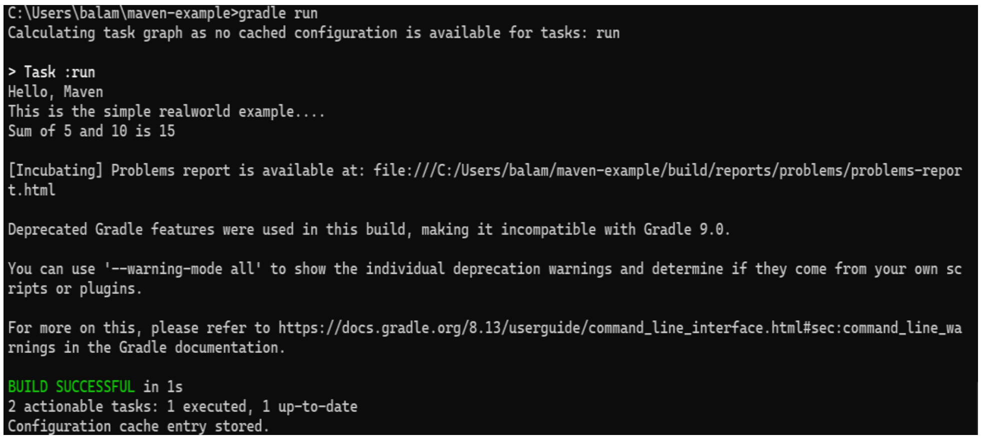
```
gradle compileJava
```



```
C:\Users\balam\maven-example>gradle compileJava
Calculating task graph as no cached configuration is available for tasks: compileJava

[Incubating] Problems report is available at: file:///C:/Users/balam/maven-example/build/reports/problems/problems-repor
t.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 sc
ripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.13/userguide/command_line_interface.html#sec:command_line_wa
rnings in the Gradle documentation.

BUILD SUCCESSFUL in 1s
1 actionable task: 1 up-to-date
Configuration cache entry stored.
```

- **Run the gradle application**

  gradle run



```
C:\Users\balam\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/balam/maven-example/build/reports/problems/problems-repor
t.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 sc
ripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.13/userguide/command_line_interface.html#sec:command_line_wa
rnings in the Gradle documentation.

BUILD SUCCESSFUL in 1s
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:

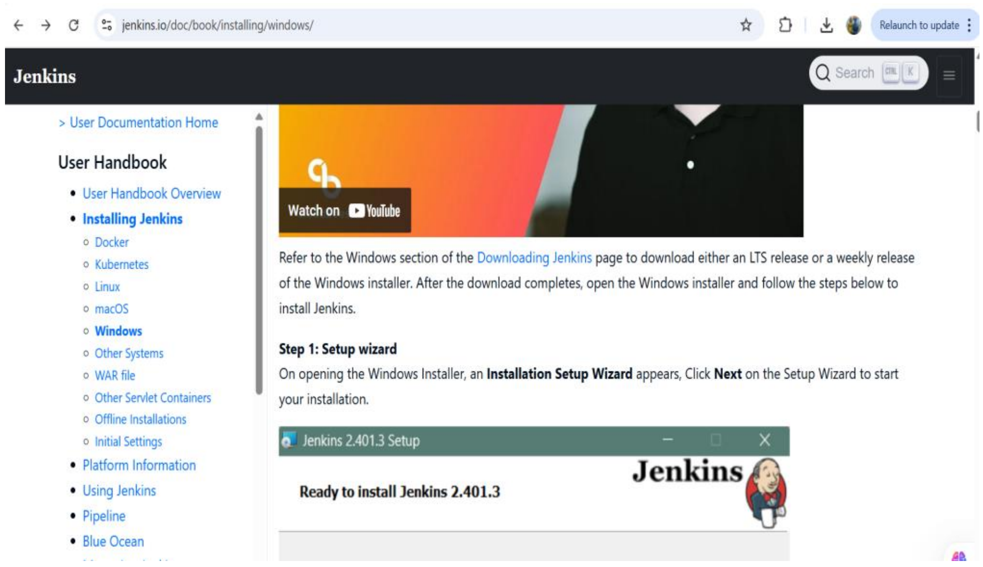
- CI/CD: Jenkins supports Continuous Integration and Continuous Deployment, allowing developers to integrate code changes frequently and automate the deployment of applications.
- Plugins: Jenkins has a vast library of plugins that can extend its capabilities. These plugins integrate Jenkins with version 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**

   https://www.jenkins.io/doc/book/installing/windows/

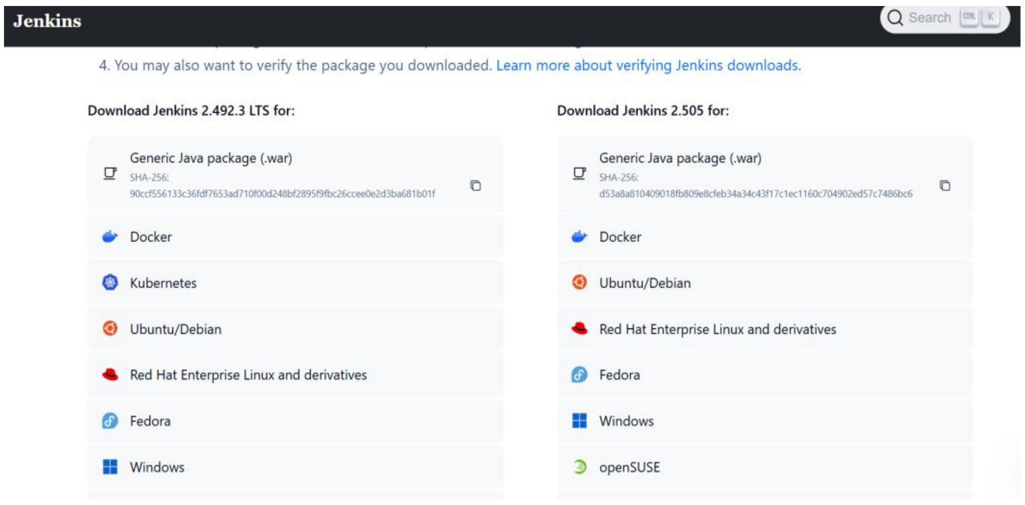


Refer to the Windows section of the Downloading Jenkins page to download either an LTS release or a weekly release of the Windows installer. After the download completes, open the Windows installer and follow the steps below to install Jenkins.

Step 1: Setup wizard
On opening the Windows Installer, an **Installation Setup Wizard** appears, Click **Next** on the Setup Wizard to start your installation.

Ready to install Jenkins 2.401.3

- Click Downloading Jenkins link



4. You may also want to verify the package you downloaded. Learn more about verifying Jenkins downloads.

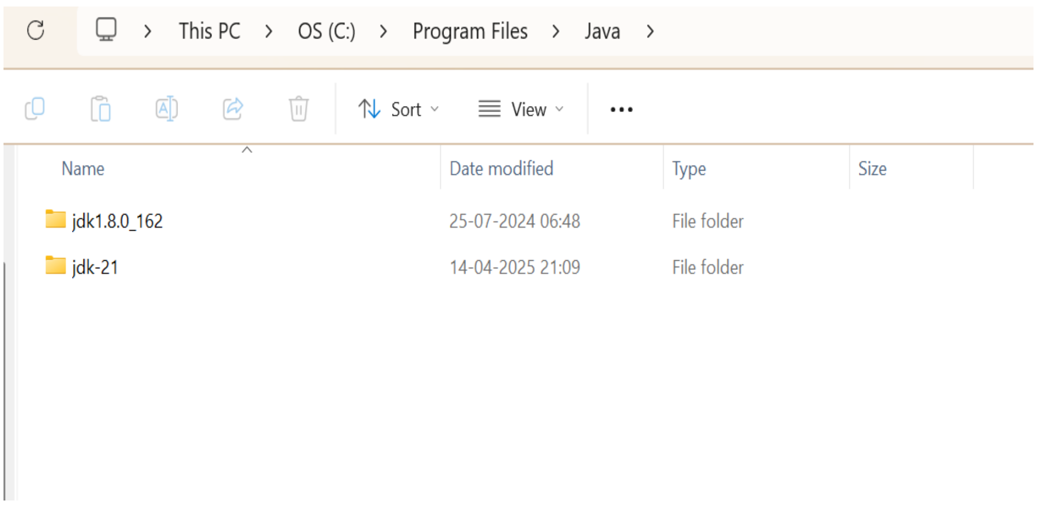
Download Jenkins 2.492.3 LTS for:

- Generic Java package (.war)
  SHA-256:
  90ccf556133c36fdf7653ad710f00d248bf2895f9fbc26ccee0e2d3ba681b01f
- Docker
- Kubernetes
- Ubuntu/Debian
- Red Hat Enterprise Linux and derivatives
- Fedora
- Windows

Download Jenkins 2.505 for:

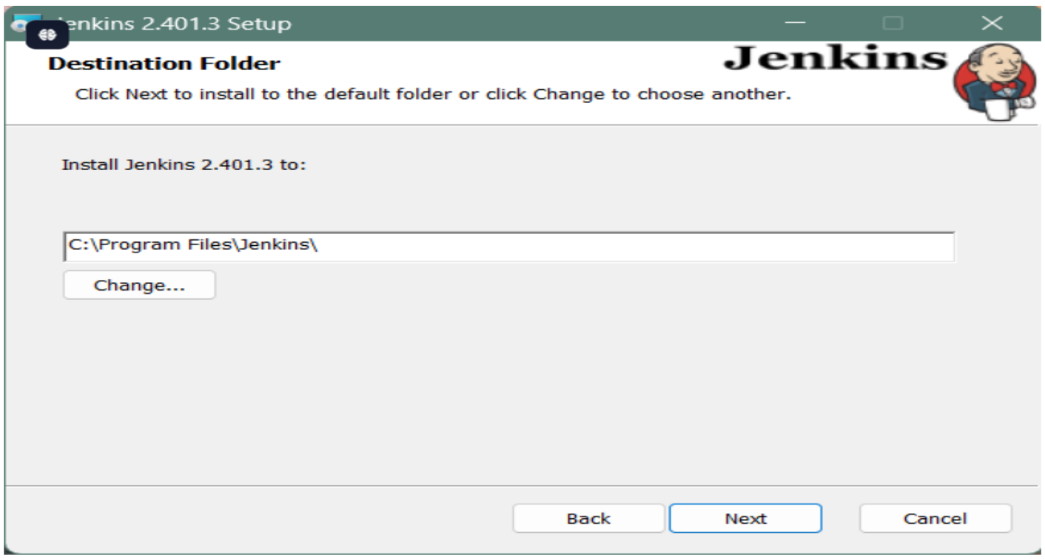
- Generic Java package (.war)
  SHA-256:
  d53a8a810409018fb809e8cfeb34a34c43f17c1ec1160c704902ed57c7486bc6
- Docker
- Ubuntu/Debian
- Red Hat Enterprise Linux and derivatives
- Fedora
- Windows
- openSUSE

- Choose windows option from LTS
- Zip file will get downloaded
- Make sure java 17 version or 21 version is installed in the local system



Name	Date modified	Type	Size
jdk1.8.0_162	25-07-2024 06:48	File folder	
jdk-21	14-04-2025 21:09	File folder	

- Double click on the Jenkins file – Download will start
- In setup wizard, choose the destination folder



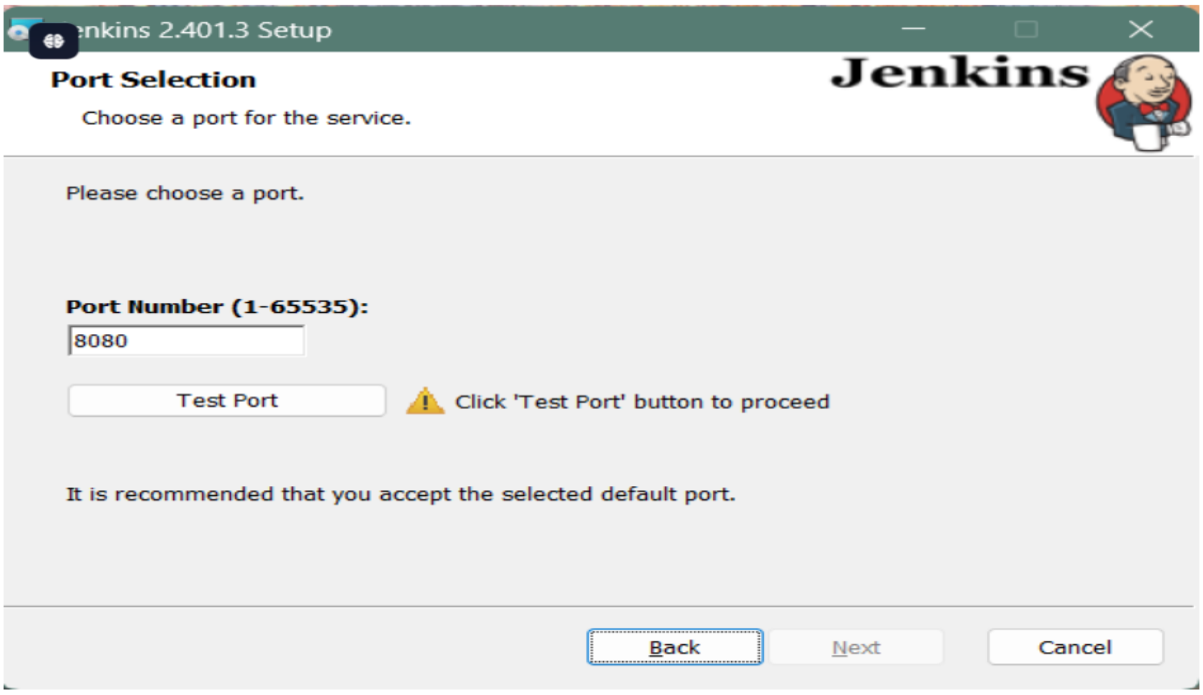
Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install Jenkins 2.401.3 to:

C:\Program Files\Jenkins\

- Choose Run Service as Local System
- Default Port No:8080



Jenkins 2.401.3 Setup

Port Selection

Choose a port for the service.

Please choose a port.

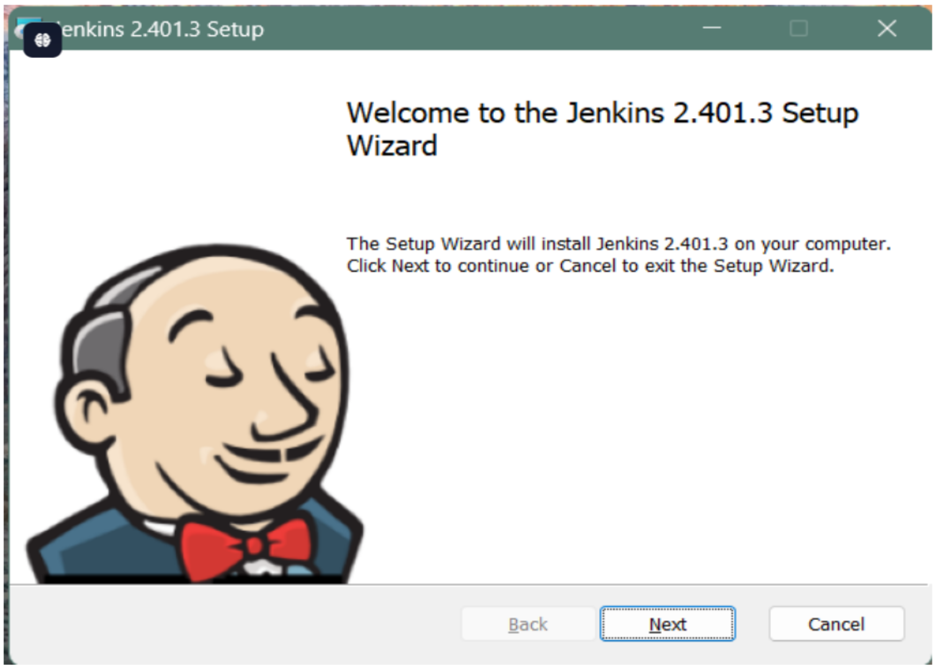
Port Number (1-65535):

8080

Click 'Test Port' button to proceed

It is recommended that you accept the selected default port.

- Select Java home directory
- Change the path based on java folder "C:\Program Files\Java\jdk-21"
- Select Start Service in the next window

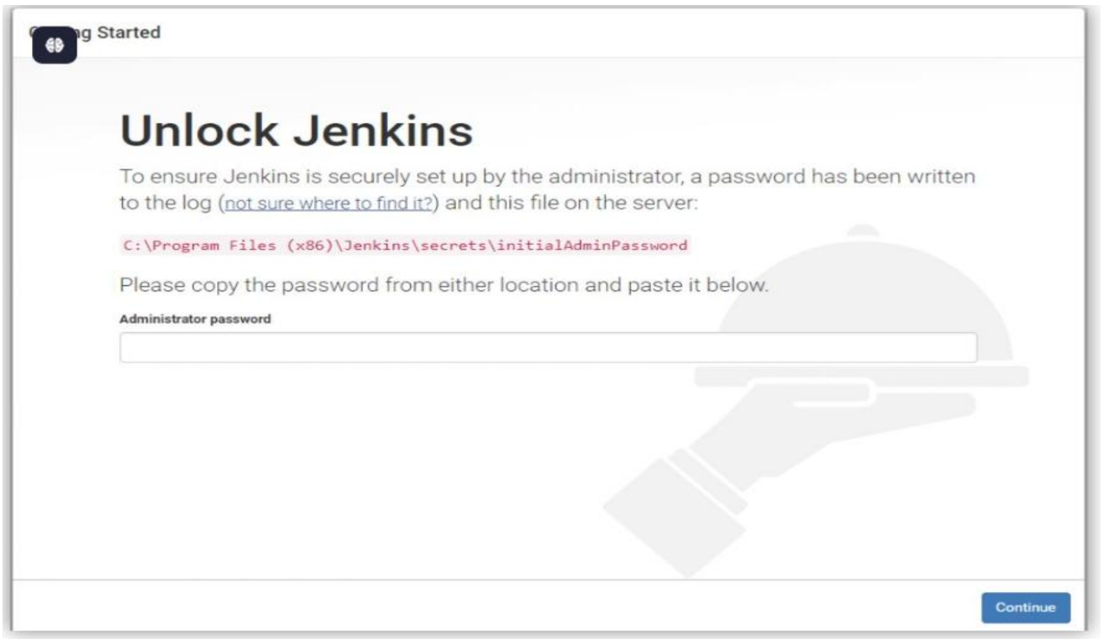


Jenkins 2.401.3 Setup

Welcome to the Jenkins 2.401.3 Setup Wizard

The Setup Wizard will install Jenkins 2.401.3 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

- Once installation is completed, type http://localhost:8080 in the web browser (Chrome or Edge).


Getting Started

Unlock Jenkins

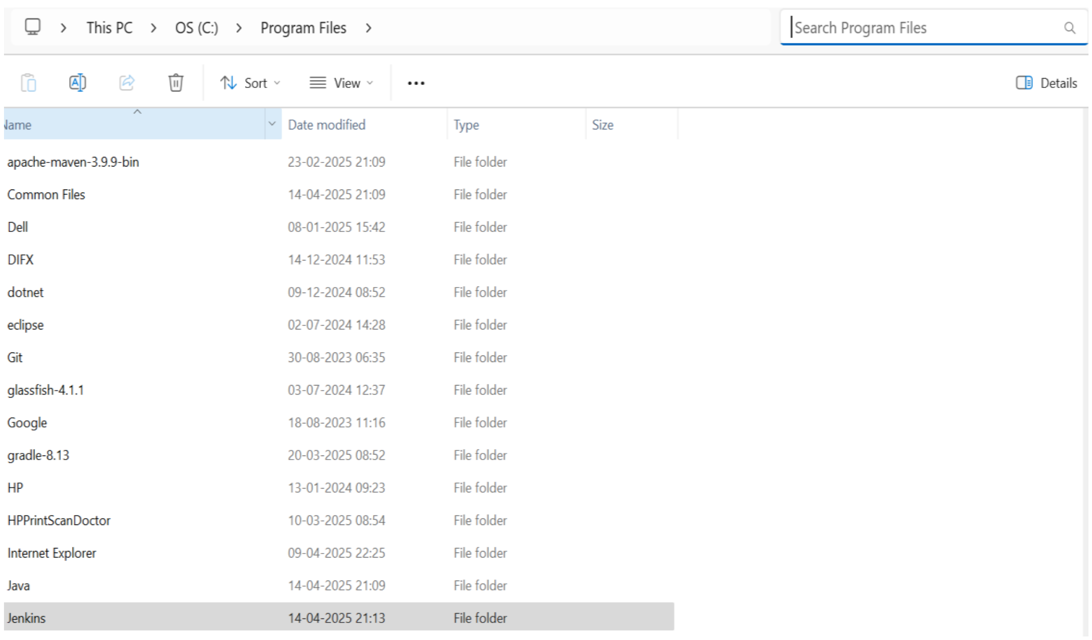
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword
```

Please copy the password from either location and paste it below.

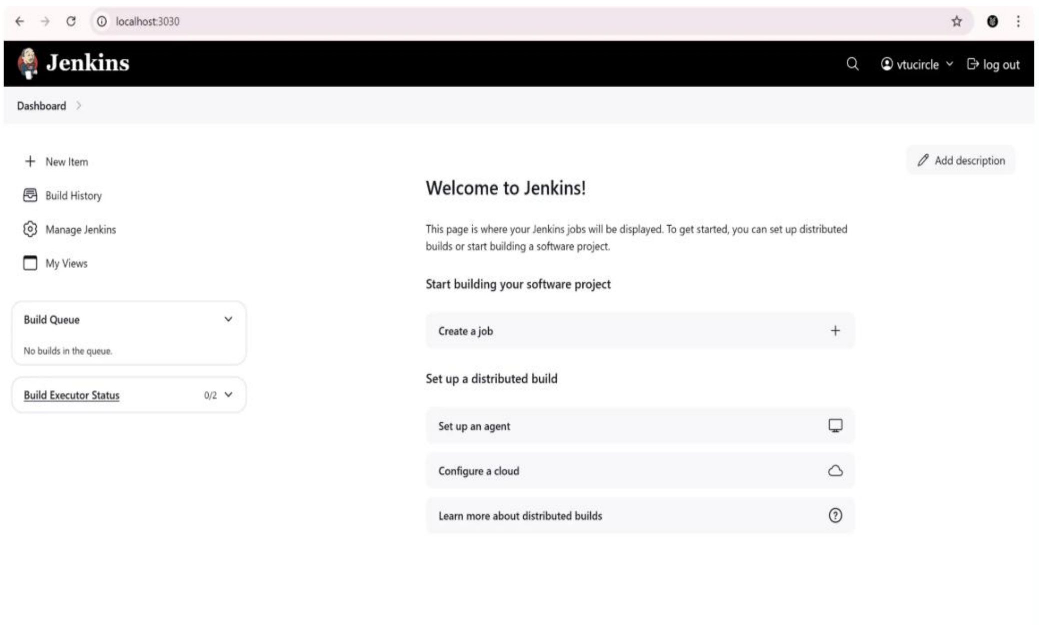
Administrator password

- Copy the link provided in the unlock Jenkins window to retrieve AdminPassword
- Paste it in the folder search

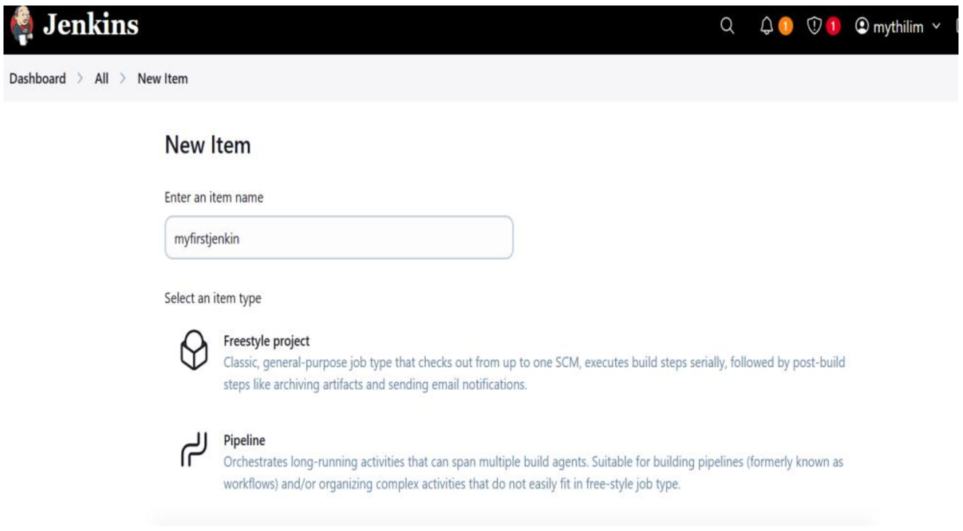


Name	Date modified	Type	Size
apache-maven-3.9.9-bin	23-02-2025 21:09	File folder	
Common Files	14-04-2025 21:09	File folder	
Dell	08-01-2025 15:42	File folder	
DIFX	14-12-2024 11:53	File folder	
dotnet	09-12-2024 08:52	File folder	
eclipse	02-07-2024 14:28	File folder	
Git	30-08-2023 06:35	File folder	
glassfish-4.1.1	03-07-2024 12:37	File folder	
Google	18-08-2023 11:16	File folder	
gradle-8.13	20-03-2025 08:52	File folder	
HP	13-01-2024 09:23	File folder	
HPPrintScanDoctor	10-03-2025 08:54	File folder	
Internet Explorer	09-04-2025 22:25	File folder	
Java	14-04-2025 21:09	File folder	
Jenkins	14-04-2025 21:13	File folder	

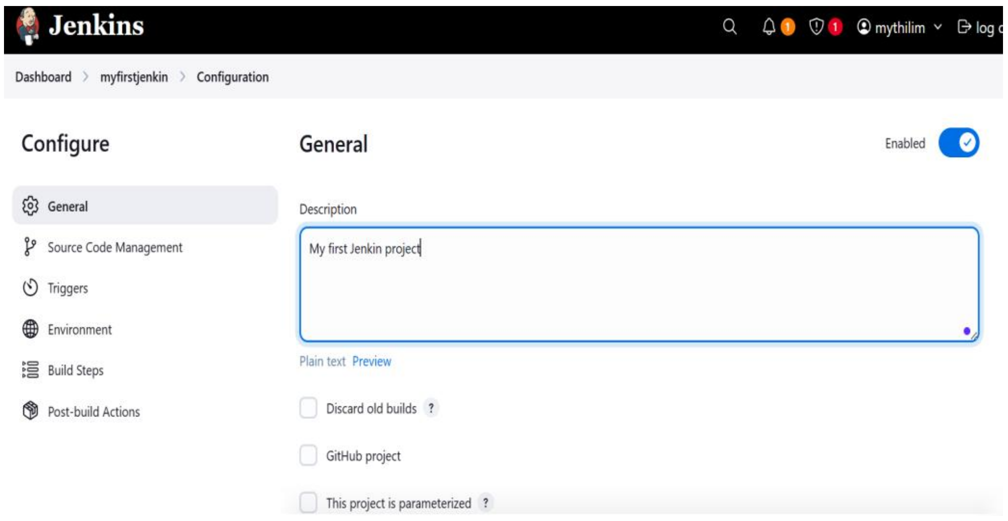
- Open the password document in notepad. Copy and paste the complete password in Unlocking Jenkins
- Install suggested plugins – All default plugins
- Create admin profile by giving a name and own password. Save and continue then finish.


localhost:3030
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
Add description
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

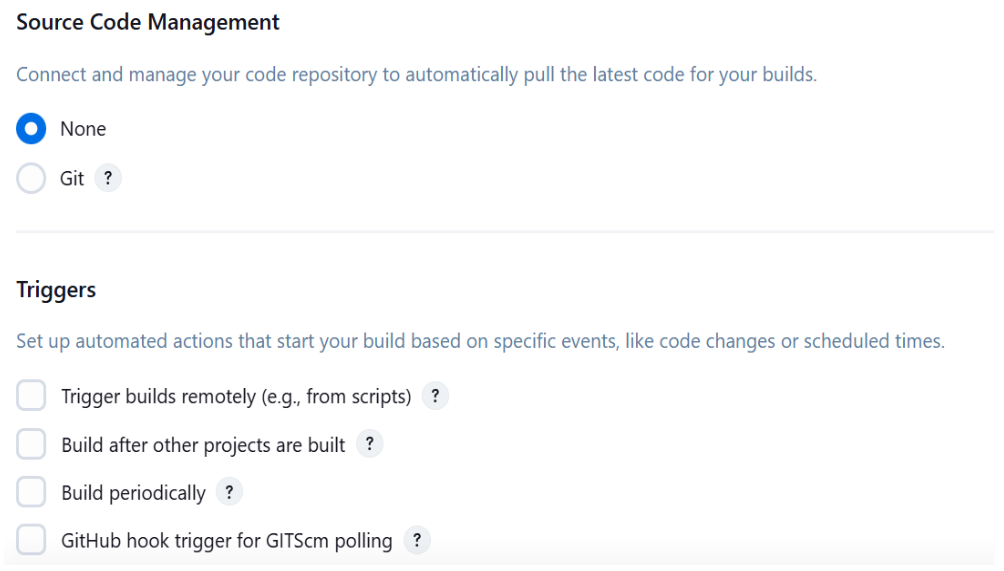
- Select new item to create a new build in Jenkins
- Specify item name


Jenkins
mythilim
Dashboard > All > New Item
New Item
Enter an item name
myfirstjenkin
Select an item type
Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

- Select freestyle project



- Add Description



Source Code Management

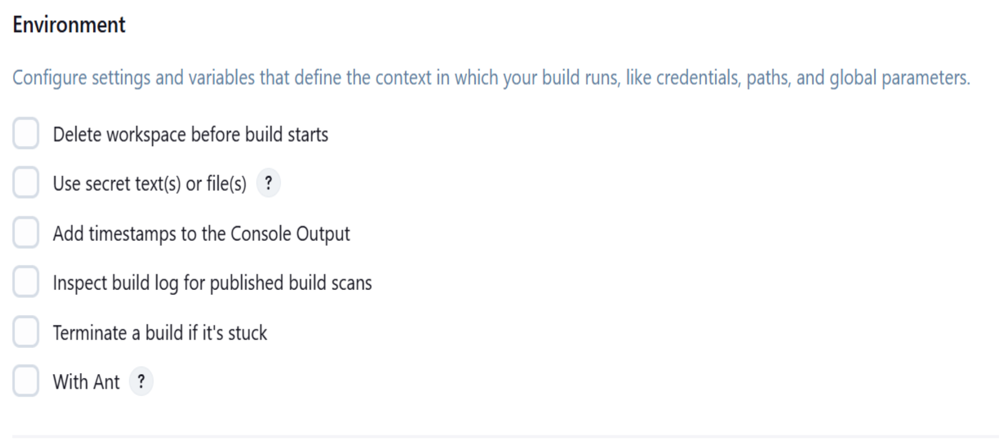
Connect and manage your code repository to automatically pull the latest code for your builds.

- [✓ None]
- Git ?

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- ☐ Trigger builds remotely (e.g., from scripts) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☐ GitHub hook trigger for GITScm polling ?

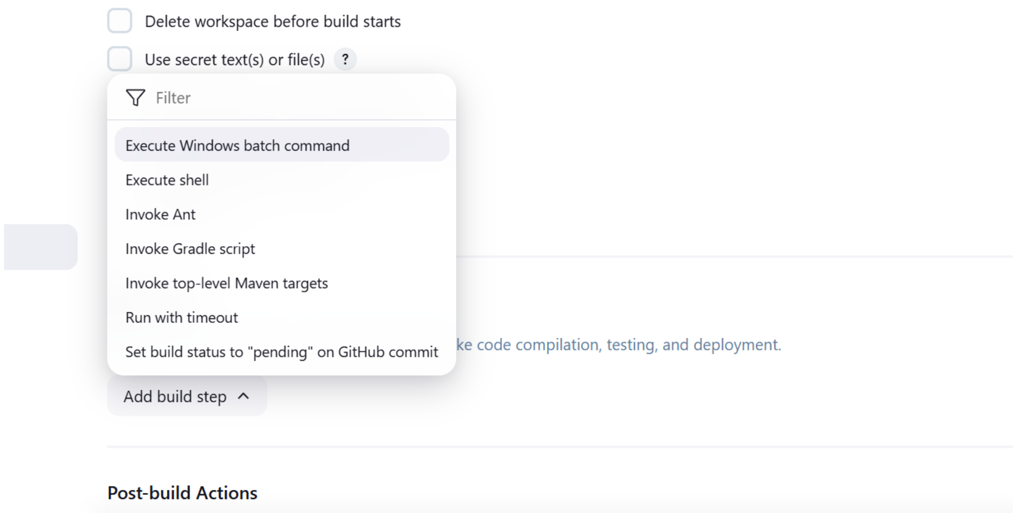


Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?
- ☐ Add timestamps to the Console Output
- ☐ Inspect build log for published build scans
- ☐ Terminate a build if it's stuck
- ☐ With Ant ?

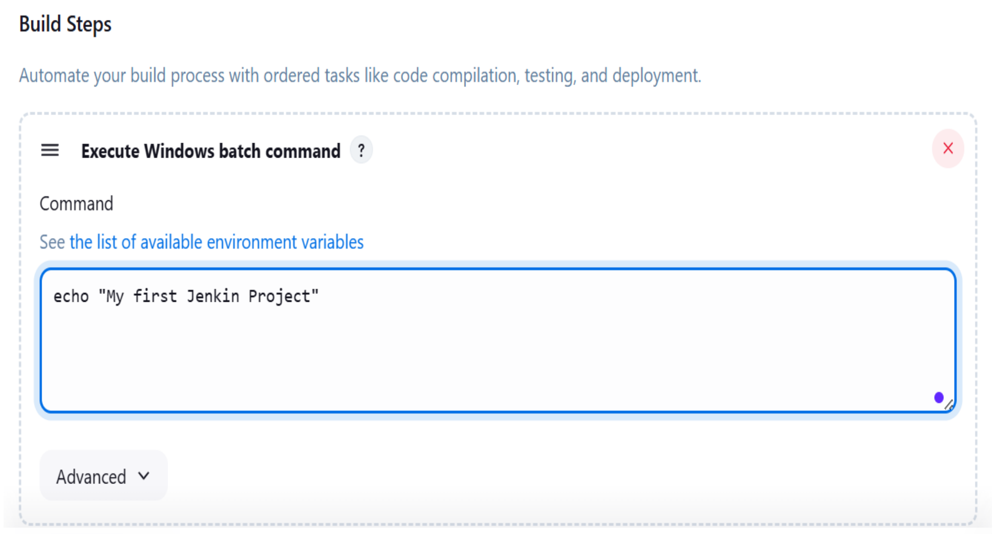
- Select AddBuildStep



- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?

Filter
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
ke code compilation, testing, and deployment.
Add build step
Post-build Actions

- Select Execute Windows Batch Command



Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

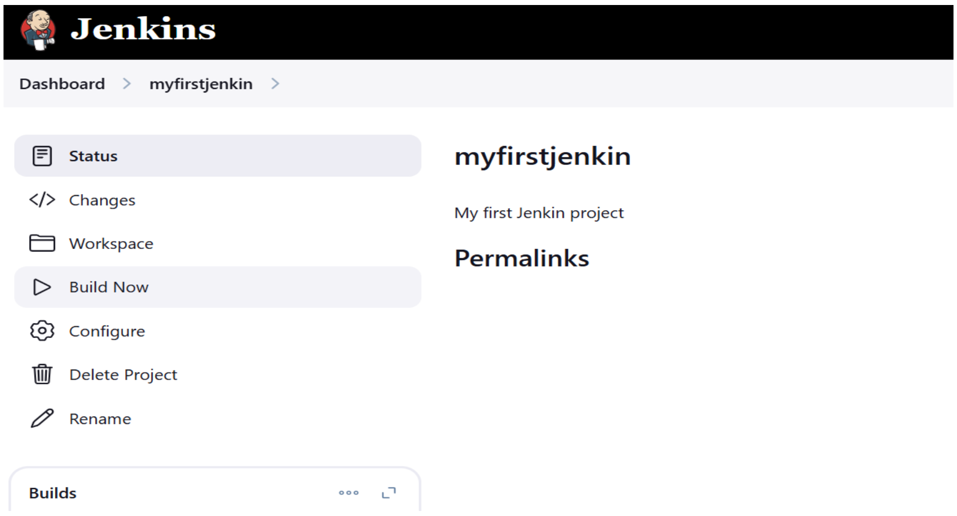
Execute Windows batch command ?

Command

See the list of available environment variables

```
echo "My first Jenkin Project"
```

- Click Save

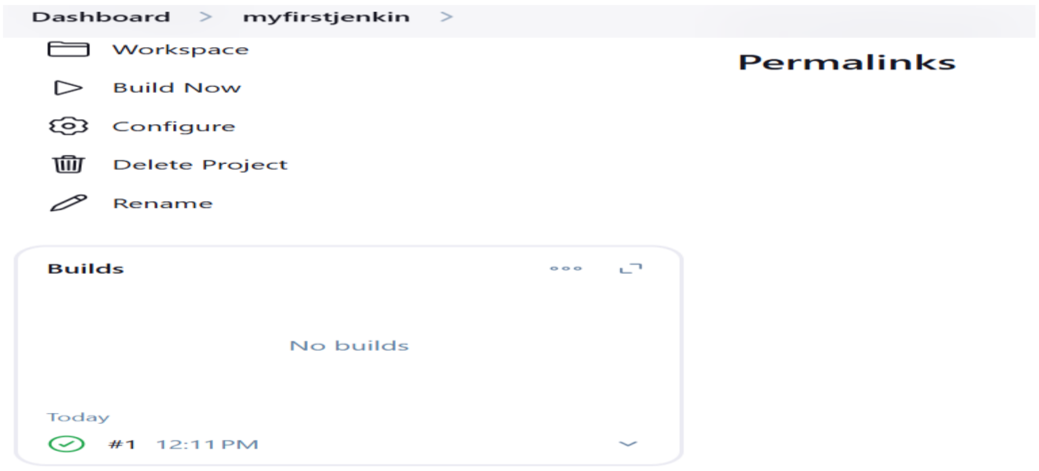


myfirstjenkin

My first Jenkin project

Permalinks

- Select Build Now

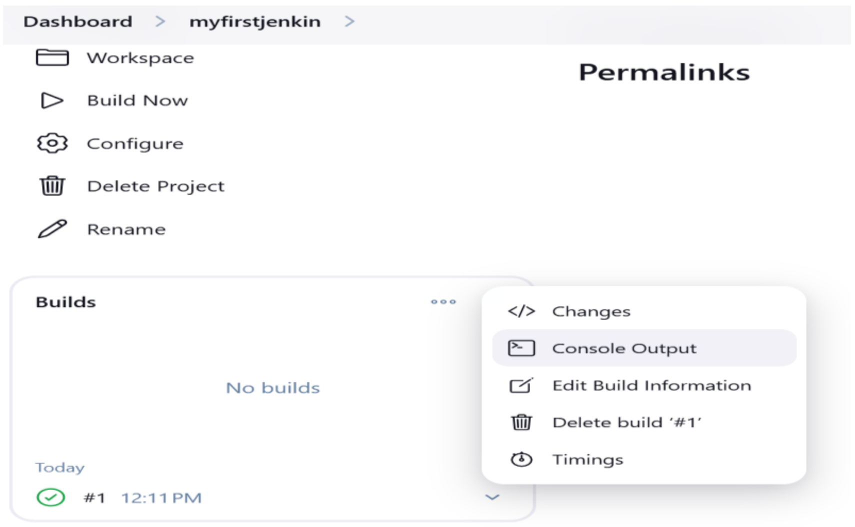


No builds

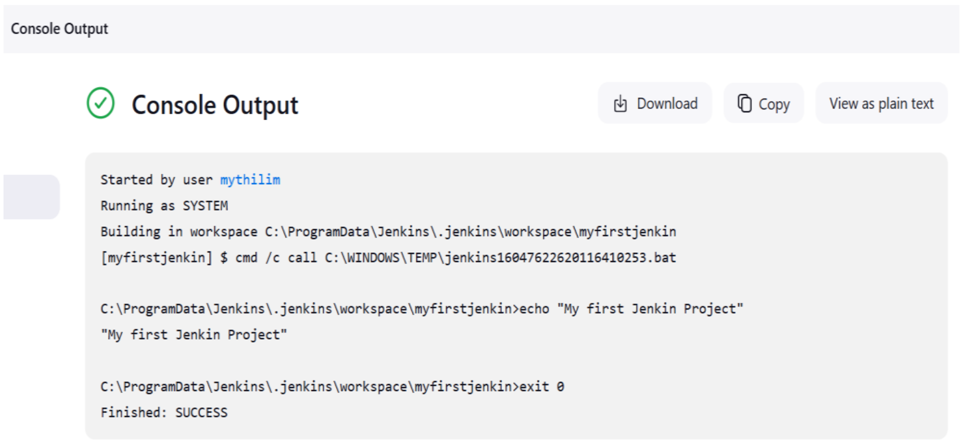
Today

#1 12:11 PM

- Click on the down arrow next to the time

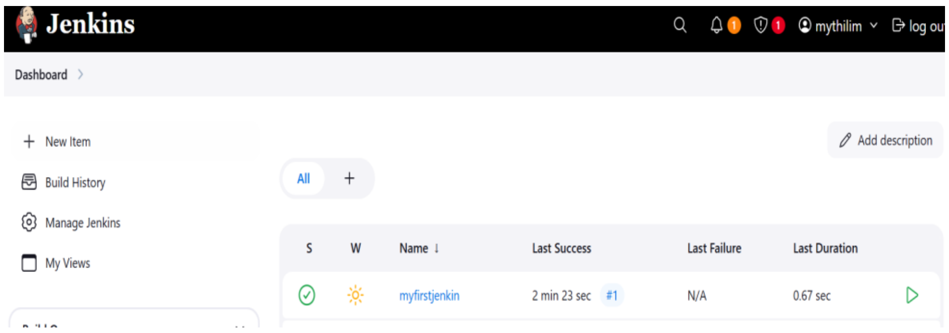


- Select console output



```
Started by user mythilim
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\.jenkins\workspace\myfirstjenkin
[myfirstjenkin] $ cmd /c call C:\WINDOWS\TEMP\jenkins16047622620116410253.bat

C:\ProgramData\Jenkins\.jenkins\workspace\myfirstjenkin>echo "My first Jenkin Project"
"My first Jenkin Project"

C:\ProgramData\Jenkins\.jenkins\workspace\myfirstjenkin>exit 0
Finished: SUCCESS
```

- To make changes, Goto Dashboard



S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀	myfirstjenkin	2 min 23 sec #1	N/A	0.67 sec

- Click on the name and choose the option “Configure”
- Change the build step to **echo “My first Jenkin Project on %date% at %time%”** and repeat to execute the command.