



# SRI VENKATESHWARA COLLEGE OF ENGINEERING



## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

### **LAB MANUAL**

### **DEVOPS – BCS657D**

#### **Sri Venkateshwara College of Engineering (SVCE)**

Vidyanagar, Kempegowda International Airport Road,

Bettahalasoor Post, Chikkajala

Bengaluru, Karnataka – 562157.

# CONTENTS

1. Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup.
2. Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins.
3. Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation. Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation.
4. Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle.
5. Introduction to Jenkins: What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use.
6. Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests.
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.
8. Practical Exercise: Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins.
9. Introduction to Azure DevOps: Overview of Azure DevOps Services, Setting Up an Azure DevOps Account and Project.
10. Creating Build Pipelines: Building a Maven/Gradle Project with Azure Pipelines, Integrating Code Repositories (e.g., GitHub, Azure Repos), Running Unit Tests and Generating Reports.
11. Creating Release Pipelines: Deploying Applications to Azure App Services, Managing Secrets and Configuration with Azure Key Vault, Hands-On: Continuous Deployment with Azure Pipelines.
12. Practical Exercise and Wrap-Up: Build and Deploy a Complete DevOps Pipeline, Discussion on Best Practices and Q&A.

# Program 1

## 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
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:

- **Download Maven:**
  - Go to the Maven Download Page and download the latest binary ZIP file.
- **Extract the ZIP File:**
  - Right-click the downloaded ZIP file and select Extract All... or use any extraction tool like WinRAR or 7-Zip.
- **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\.
- **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).
- **Set Environment Variables:**
  - Open the Start Menu, search for Environment Variables, and select Edit the system environment variables.
  - Click Environment Variables.
  - Under System Variables:
    1. Find the path, double click on it and click New.
    2. Paste the full path to the bin folder of your Maven directory (e.g., C:\Program Files\apache-maven-x.x.x\bin).
- **Save the Changes:**
  - Click OK to close the windows and save your changes.
- **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

- **Download Gradle:**
  - Visit the Gradle Downloads Page and download the latest binary ZIP file.
- **Extract the ZIP File:**
  - Right-click the downloaded ZIP file and select Extract All... or use any extraction tool like WinRAR or 7-Zip.
- **Move the Folder:**
  - After extraction, move the extracted Gradle folder (usually named `gradle-x.x.x`) to a convenient directory like `C:\Program Files\`.
- **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.x\bin`).
- **Set Environment Variables:**
  - Open the Start Menu, search for Environment Variables, and select Edit the system environment variables.
  - Click Environment Variables.
  - Under System Variables:
    1. Find the path, double click on it and click New.
    2. Paste the full path to the bin folder of your Gradle directory (e.g., `C:\Program Files\gradle-x.x.x\bin`).
- **Save the Changes:**
  - Click OK to close the windows and save your changes.
- **Verify the Installation:**
  - Open a terminal or Command Prompt and run: `gradle -v` If it shows the Gradle version, the setup is complete.

# Program 2

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

### 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
```

- **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.

### 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:

A basic **pom.xml** structure looks like this:



# Program 3

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:**

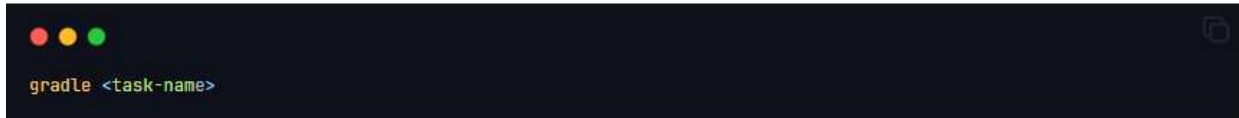
```
task printMessage {  
    doLast {  
        println 'This is a custom task automation'  
    }  
}
```

- **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:

A screenshot of a terminal window with a dark background. It shows the command `gradle <task-name>` in a light-colored font. The terminal has standard macOS window control buttons (red, yellow, green) in the top-left corner and a copy icon in the top-right corner.

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'
    }
}
```

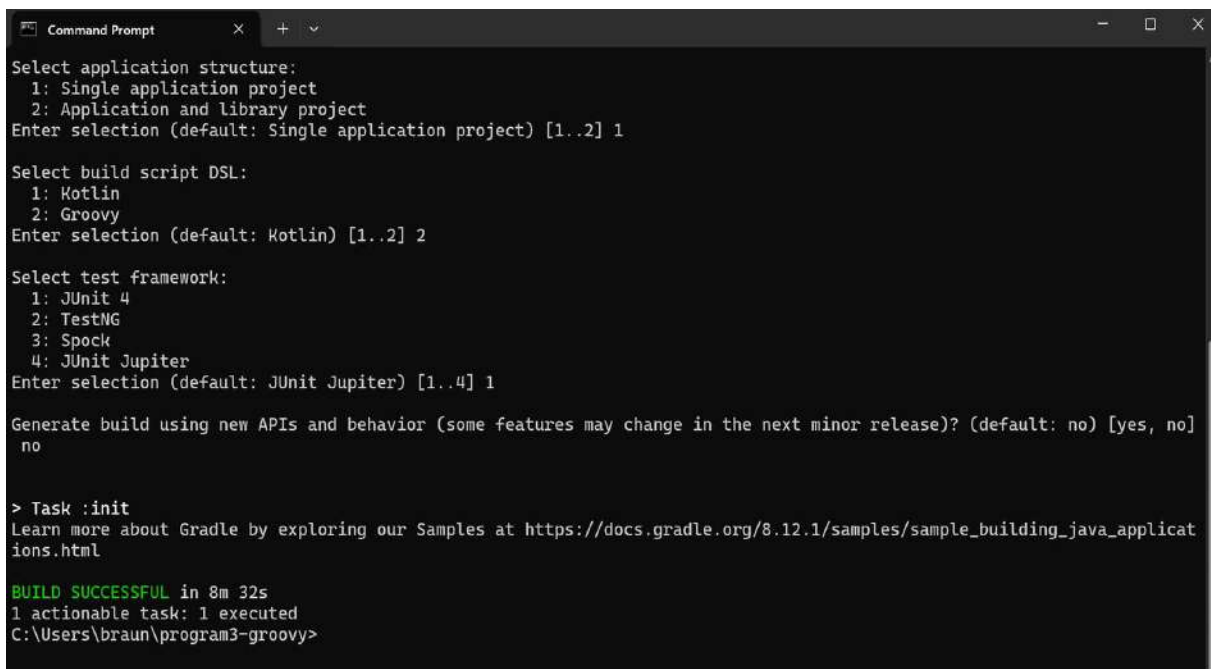
In this case, **secondTask** will depend on the completion of **firstTask** before it runs.

## 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



```
Command Prompt
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_applications.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;

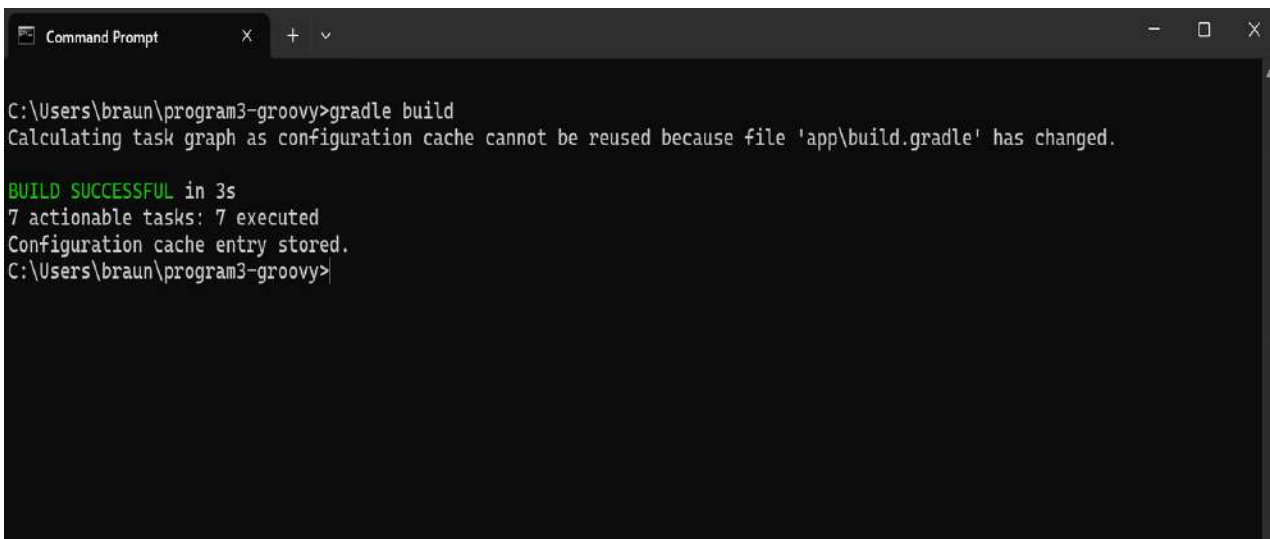
        double actualSum = num1 + num2;

        assertEquals(expectedSum, actualSum, 0.01);
    }
}
```

### Step 5: Run Gradle Commands

- To **build** the project:

```
gradle build
```



```
Command Prompt
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
```

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

> Task :app:test
AdditionOperationTest > testAddition PASSED

[Incubating] Problems report is available at: file:///C:/Users/braun/program3-groovy/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-groovy>
```



## 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("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 paste 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

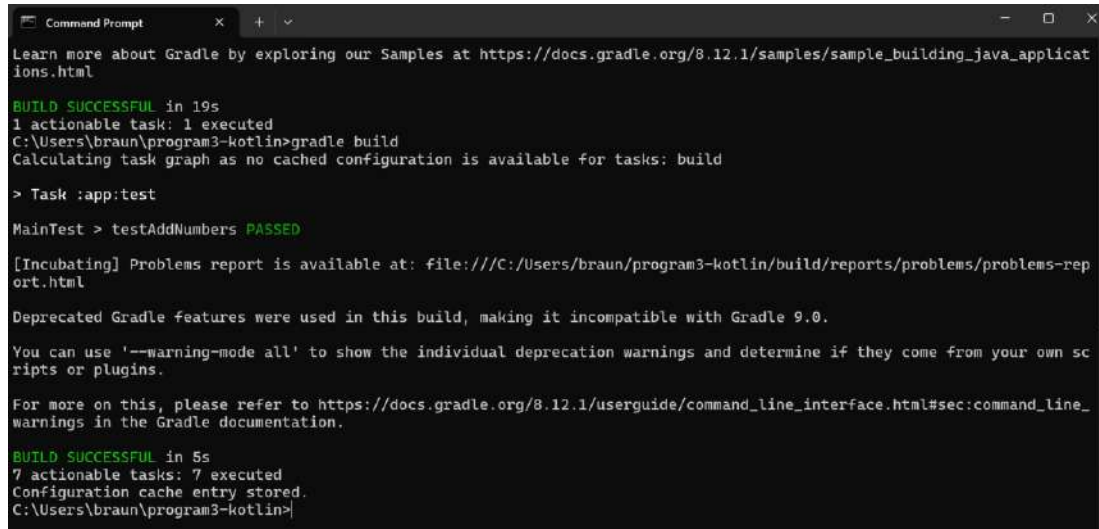
        val result = addNumbers(num1, num2)

        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
```



```
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.

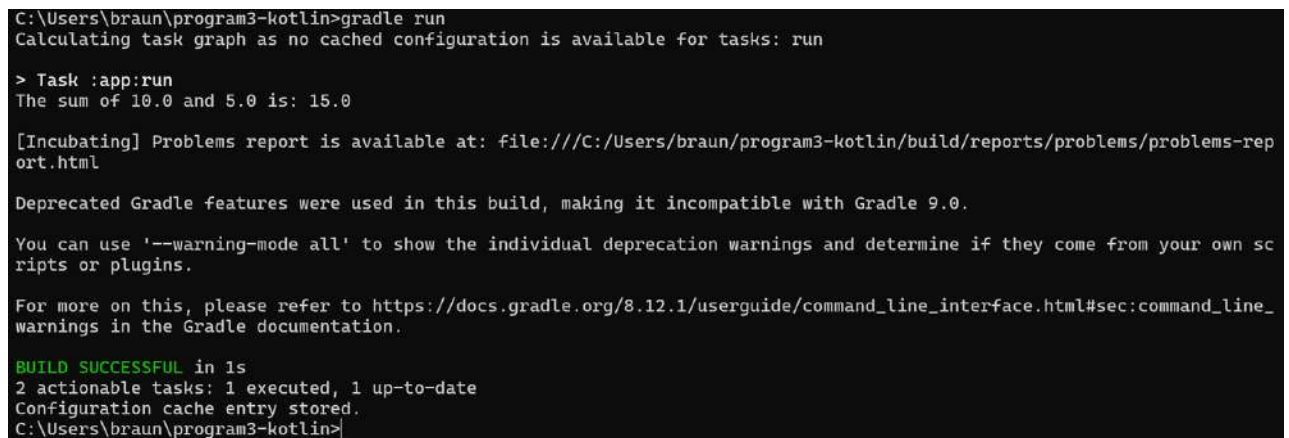
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

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>
```

# Program 4

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**.

- **I'm 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"
```

```
Command Prompt
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
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.4.0/commons-exec-1.4.0.jar
Downloading 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/apache/maven/resolver/maven-resolver-api/1.4.1/maven-resolver-api-1.4.1.jar (149 kB at 1.2 MB/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/apache/commons/commons-exec/1.4.0/commons-exec-1.4.0.jar (66 kB at 341 kB/s)
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 216 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm/9.7.1/asm-9.7.1.jar (126 kB at 515 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/ow2/asm/asm-commons/9.7.1/asm-commons-9.7.1.jar (73 kB at 297 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/4.0.2/plexus-utils-4.0.2.jar (192 kB at 694 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: 3.044 s
[INFO] Finished at: 2025-02-05T16:53:12+05:30
[INFO]
C:\Users\braun\maven-example>
```

## 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\braun\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 d
ocumentation.

BUILD SUCCESSFUL in 3m 56s
1 actionable task: 1 executed
C:\Users\braun\maven-example>

```

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:

```
gradlew build
```

```
C:\Users\braun\maven-example>gradlew build
Calculating task graph as no cached configuration is available for tasks: build

[Incubating] Problems report is available at: file:///C:/Users/braun/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
4 actionable tasks: 4 executed
Configuration cache entry stored.
C:\Users\braun\maven-example>
```

- **Run the Application:** Once the build is successful, run the application using below command:

```
gradlew run
```

```
C:\Users\braun\maven-example>gradlew 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/braun/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 1s
2 actionable tasks: 1 executed, 1 up-to-date
Configuration cache entry stored.
C:\Users\braun\maven-example>
```

## 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
```

# Program 5

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):

- Download the Jenkins Windows installer from the [official Jenkins website](#).
- Run the installer and follow the on-screen instructions. While installing choose **login system: run service as LocalSystem (not recommended)**.

- After then use **default port** or you can **configure you own port** like I'm using **port 3030** then **click on test** and **next**.
- After then **change the directory** and choose **java jdk-21** path look like **C:\Program Files\Java\jdk-21\**.
- After then **click next, next** and then it will **ask permission** click on **yes** and it will start installing.
- After successfully installed, Jenkins will be **running on port either default port** or **chosen port** like i choose **port 3030** by default (you can access it in your browser at **http://localhost:8080**) or **http://localhost:3030**.


## 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.
- 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 Jenkin.

← → ↻ 🌐 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

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

Homepage of jenkins

REST API Jenkins 2.492.1

## Program 6

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

This guide will walk you through setting up **Jenkins** on Windows, configuring a **CI pipeline**, integrating Jenkins with **Maven/Gradle**, and running **automated builds and tests**.

---

### Step 1: Install Java (JDK) on Windows

Jenkins requires **Java** to run. Install **Java 11 or later**.

#### 1.1 Download and Install Java

- Download the **Java JDK** from the official website:  
[Oracle JDK](#)  
or  
[OpenJDK](#)
- Install the JDK and note the installation path (e.g., C:\Program Files\Java\jdk-11).

#### 1.2 Set Up Environment Variables

- Open **System Properties** → **Advanced** → **Environment Variables**.
- Under **System Variables**, click **New** and add:
  - **Variable Name:** JAVA\_HOME
  - **Variable Value:** C:\Program Files\Java\jdk-11
- Add %JAVA\_HOME%\bin to the **Path** variable.

#### 1.3 Verify Java Installation

Open **Command Prompt (cmd)** and run:

```
java -version
```

Expected Output:

```
Nginx
```

```
java version "11.0.x"
```

---

## Step 2: Install Jenkins on Windows

Jenkins provides a **Windows installer** for easy setup.

### 2.1 Download Jenkins

- Download the **Jenkins Windows Installer** from:  
Jenkins official website

### 2.2 Install Jenkins

1. Run the downloaded `.msi` installer.
2. Follow the on-screen instructions:
  - Choose an installation directory (`C:\Program Files\Jenkins` recommended).
  - Select the **Run as a Service** option.
  - Click **Next** and install.

### 2.3 Start Jenkins

Once installed, Jenkins will start automatically. If not, start it manually:

```
cmd
```

```
net start jenkins
```

### 2.4 Unlock Jenkins

1. Open a browser and go to:

```
arduino
```

```
http://localhost:8080
```

2. Enter the **Administrator Password** (found in `C:\Program Files\Jenkins\secrets\initialAdminPassword`).
  3. Install recommended plugins.
  4. Create an **Admin User**.
-

## Step 3: Install Required Plugins

Go to **Manage Jenkins** → **Plugins** → **Available Plugins**, and install:

- **Maven Integration Plugin**
- **Gradle Plugin**
- **Git Plugin**
- **Pipeline Plugin**

After installation, **restart Jenkins**.

---

## Step 4: Configure Git and Build Tools

### 4.1 Install Git

- Download and install Git from [Git official site](#).
- Verify installation:

cmd

```
git --version
```

### 4.2 Install Maven

- Download Apache Maven from [Maven official site](#).
- Extract it to C:\Maven.
- Set environment variables:
  - **MAVEN\_HOME** → C:\Maven
  - Add %MAVEN\_HOME%\bin to **Path**.
- Verify installation:

cmd

```
mvn -version
```

### 4.3 Install Gradle (if needed)

- Download from Gradle official site.
- Extract it to C:\Gradle.
- Set environment variables:
  - **GRADLE\_HOME** → C:\Gradle
  - Add %GRADLE\_HOME%\bin to **Path**.
- Verify installation:



```
cmd
```

```
gradle -v
```

---

## Step 5: Set Up a CI Pipeline in Jenkins

### 5.1 Create a New Pipeline Job

1. Open **Jenkins Dashboard** → **New Item**.
  2. Enter a job name (e.g., `MyCI-Pipeline`).
  3. Select **Pipeline** and click **OK**.
- 

## Step 6: Integrate Jenkins with Maven/Gradle

### 6.1 Configure a Maven Project

1. Go to **Jenkins Dashboard** → **New Item** → **Freestyle Project**.
2. In **Source Code Management**, select **Git** and enter the repository URL.
3. In **Build**, add:

```
cmd
```

```
mvn clean install
```

4. Click **Save** and **Build Now**.

### 6.2 Configure a Gradle Project

1. Go to **Jenkins Dashboard** → **New Item** → **Freestyle Project**.
2. In **Build**, add:

```
cmd
```

```
gradle build
```

3. Click **Save** and **Build Now**.
-

## Step 7: Writing a Jenkinsfile for CI Pipeline

Jenkinsfile defines a pipeline for automated builds and tests.

### 7.1 Create a Jenkinsfile in Your Project Root

```
groovy

pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying application...'
            }
        }
    }
}
```

### 7.2 Configure Jenkins to Use Jenkinsfile

1. In **Jenkins Dashboard**, open the Pipeline Job.
2. Go to **Pipeline** → **Definition** → Select **Pipeline script from SCM**.
3. Choose **Git** and enter the repository URL.
4. Click **Save** and **Build Now**.

---

## Step 8: Running Automated Builds and Tests

- Click **Build Now** to start the pipeline.
  - Go to **Console Output** to check the progress.
-

## Step 9: Set Up Automatic Build Triggers

1. Go to **Project Settings** in Jenkins.
2. Under **Build Triggers**, enable:
  - **Poll SCM** (`H/5 * * * * *` to check every 5 minutes).
  - **GitHub Webhooks** (configure in GitHub for push triggers).

## **Program 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.**

Since Ansible primarily runs on Linux, you need to set up a Linux environment on your Windows machine using **WSL (Windows Subsystem for Linux)** or a **Linux VM**. This guide will walk you through setting up Ansible on Windows, managing configurations, and automating server setups.

---

## **Step 1: Install Windows Subsystem for Linux (WSL)**

### **1.1 Enable WSL**

Run the following command in **PowerShell (Admin Mode)**:

```
wsl --install
```

By default, this installs **Ubuntu**. If you want a different Linux distribution, use:

```
wsl --install -d Debian
```

Restart your PC after installation.

### **1.2 Verify Installation**

Open **Command Prompt (cmd)** or **PowerShell** and run:

```
wsl --list --verbose
```

You should see Ubuntu (or your chosen Linux distro) running.

---

## **Step 2: Install Ansible on WSL**

Once WSL is set up, open **Ubuntu** (or your chosen Linux distro) from the **Start Menu**.

### **2.1 Update and Install Ansible**

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

## 2.2 Verify Installation

```
bash
```

```
ansible --version
```

Expected output:

```
java
```

```
ansible [core 2.x.x]  
python version = 3.x
```

---

## Step 3: Set Up Ansible Inventory

The **Inventory file** lists the servers Ansible will manage.

### 3.1 Create a Project Directory

```
mkdir ansible_project && cd ansible_project
```

### 3.2 Create an Inventory File

```
nano inventory.ini
```

Add the following content:

```
ini  
  
[web_servers]  
192.168.1.100  
192.168.1.101  
  
[db_servers]  
192.168.1.102
```

Replace the IPs with the actual server addresses.

---

## Step 4: Configure SSH Access

Ansible requires **passwordless SSH** access to manage remote servers.

## 4.1 Generate an SSH Key

```
ssh-keygen -t rsa
```

Press **Enter** to accept the default location.

## 4.2 Copy SSH Key to Managed Nodes

```
ssh-copy-id user@192.168.1.100  
ssh-copy-id user@192.168.1.101  
ssh-copy-id user@192.168.1.102
```

Replace `user` with the actual username of the remote machines.

---

## Step 5: Test Connection

Run the following command to check if Ansible can connect:

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

Expected output:

```
json  
192.168.1.100 | SUCCESS => {  
    "changed": false,  
    "ping": "pong"  
}
```

If successful, Ansible is able to communicate with the target servers.

---

## Step 6: Writing an Ansible Playbook

A **Playbook** is a YAML file that defines tasks for automation.

## 6.1 Create a Playbook

```
nano webserver_setup.yml
```

## 6.2 Add the following content

```
yaml
```

```
---
```

```
- name: Configure Web Servers
  hosts: web_servers
  become: yes
  tasks:
    - name: Install Apache
      ansible.builtin.apt:
        name: apache2
        state: present

    - name: Start and Enable Apache
      ansible.builtin.service:
        name: apache2
        state: started
        enabled: yes
```

### Explanation:

- Installs and starts **Apache** on all `web_servers`.

---

## Step 7: Run the Playbook

Execute the Playbook with:

```
ansible-playbook -i inventory.ini webserver_setup.yml
```

Once the playbook runs successfully, open a browser and visit:

```
cpp
```

```
http://192.168.1.100
```

---

## Step 8: Using Ansible Modules

Modules are used to execute specific tasks.

## 8.1 Example: Creating a User

yaml

```
- name: Create a New User
  hosts: all
  become: yes
  tasks:
    - name: Add a User
      ansible.builtin.user:
        name: ansible_user
        state: present
```

Run it with:

```
ansible-playbook -i inventory.ini user_setup.yml
```

## 8.2 Example: Copying a File

yaml

```
- name: Copy Index File
  hosts: web_servers
  become: yes
  tasks:
    - name: Copy HTML File
      ansible.builtin.copy:
        src: index.html
        dest: /var/www/html/index.html
```

Run it with:

```
ansible-playbook -i inventory.ini copy_file.yml
```

---



## Step 9: Automating Server Configurations

### Example: Installing and Configuring Nginx

yaml

```
- name: Install and Start Nginx
  hosts: web_servers
  become: yes
  tasks:
    - name: Install Nginx
      ansible.builtin.apt:
        name: nginx
        state: present

    - name: Start and Enable Nginx
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: yes
```

Run it with:

```
ansible-playbook -i inventory.ini nginx_setup.yml
```

Open a browser and visit:

cpp

<http://192.168.1.100>

---

## Step 10: Debugging and Logs

To check logs:

```
ansible-playbook -i inventory.ini nginx_setup.yml --check
```

To run with **detailed output**:

```
ansible-playbook -i inventory.ini nginx_setup.yml -vvv
```

## Program 8: Practical Exercise: Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins.

This guide will help you set up a **Jenkins CI pipeline** for a **Maven project** and use **Ansible** to deploy the generated artifacts.

---

### Step 1: Install Java, Maven, Git, and Jenkins on Windows

Before setting up Jenkins and Ansible, install the required dependencies.

#### 1.1 Install Java (JDK)

1. Download **JDK 11 or later** from:
  - [Oracle JDK](#)
  - [OpenJDK](#)
2. Install the JDK and set environment variables:
  - **JAVA\_HOME** → C:\Program Files\Java\jdk-11
  - Add %JAVA\_HOME%\bin to **Path**.
3. Verify installation:

```
cmd

java -version
```

#### 1.2 Install Maven

1. Download **Maven** from:
  - [Maven official site](#)
2. Extract it to C:\Maven.
3. Set environment variables:
  - **MAVEN\_HOME** → C:\Maven
  - Add %MAVEN\_HOME%\bin to **Path**.
4. Verify installation:

```
cmd

mvn -version
```

#### 1.3 Install Git

1. Download **Git** from:
  - [Git official site](#)
2. Install Git and verify:

```
cmd
```

```
git --version
```

## 1.4 Install Jenkins

1. Download **Jenkins for Windows** from:
  - o Jenkins official site
2. Run the .msi installer.
3. Follow the setup process and start Jenkins.
4. Open a browser and go to:

```
arduino
```

```
http://localhost:8080
```

5. Unlock Jenkins using the password from:

```
makefile
```

```
C:\Program Files\Jenkins\secrets\initialAdminPassword
```

6. Install recommended plugins and create an **admin user**.

---

## Step 2: Install and Configure Ansible on Windows

### 2.1 Install Windows Subsystem for Linux (WSL)

Run the following command in **PowerShell (Admin Mode)**:

```
powershell
```

```
wsl --install
```

Restart your PC after installation.

### 2.2 Install Ansible on WSL

1. Open **Ubuntu (WSL)**.
2. Run:

```
bash
```

```
sudo apt update
```

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

3. Verify installation:

```
bash

ansible --version
```

---

## Step 3: Set Up a Jenkins CI Pipeline for a Maven Project

### 3.1 Create a Jenkins Pipeline Job

1. Open **Jenkins Dashboard** → **New Item**.
2. Enter a name (e.g., `Maven-CI-Pipeline`).
3. Select **Pipeline** and click **OK**.

### 3.2 Configure Git Repository

1. In **Pipeline Settings**, select **Pipeline script from SCM**.
2. Choose **Git** and enter the **GitHub repository URL**.

### 3.3 Create a Jenkinsfile in Your Git Repository

Add a Jenkinsfile to your project root:

```
groovy

pipeline {
    agent any

    tools {
        maven 'Maven' // Ensure Maven is installed in Jenkins Global Tools
    }

    stages {
        stage('Checkout Code') {
            steps {
                git 'https://github.com/your-repo/maven-project.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }

        stage('Archive Artifact') {
            steps {
                archiveArtifacts artifacts: 'target/*.jar', fingerprint: true
            }
        }
    }
}
```

```
        stage('Deploy Using Ansible') {
            steps {
                sh 'ansible-playbook -i inventory.ini deploy.yml'
            }
        }
    }
}
```

---

## Step 4: Configure Jenkins to Run Maven

1. Go to **Manage Jenkins** → **Global Tool Configuration**.
  2. Under **Maven**, click **Add Maven** and name it `Maven`.
  3. Click **Save**.
- 

## Step 5: Run the Pipeline in Jenkins

1. Click **Build Now** in Jenkins.
  2. Monitor the Console Output.
- 

## Step 6: Set Up Ansible for Deployment

### 6.1 Create an Inventory File

```
bash
```

```
nano inventory.ini
```

Add:

```
ini
```

```
[servers]
192.168.1.100 ansible_user=your_user
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Replace `192.168.1.100` with your server IP.

### 6.2 Create a Playbook for Deployment

```
bash
```

```
nano deploy.yml
```

Add:

yaml

```
---
- name: Deploy Maven Artifact
  hosts: servers
  become: yes
  tasks:
    - name: Copy JAR File
      ansible.builtin.copy:
        src: target/myapp.jar
        dest: /opt/myapp.jar
        mode: '0755'

    - name: Restart Application
      ansible.builtin.systemd:
        name: myapp
        state: restarted
```

---

## Step 7: Run the Pipeline and Deploy

1. **Run the Jenkins Pipeline.**
2. **Ansible will deploy the artifact** to the remote server.

## **Program 9: Introduction to Azure DevOps: Overview of Azure DevOps Services, Setting Up an Azure DevOps Account and Project.**

### **Step 1: Understanding Azure DevOps Services**

Azure DevOps provides a suite of tools for software development and collaboration. The key services include:

- **Azure Repos** – Source control for managing code (Git and TFVC).
  - **Azure Pipelines** – CI/CD automation for building, testing, and deploying applications.
  - **Azure Boards** – Agile project management with Kanban, Scrum, and dashboards.
  - **Azure Test Plans** – Testing solutions for quality assurance.
  - **Azure Artifacts** – Package management for dependency management.
- 

### **Step 2: Setting Up an Azure DevOps Account**

1. **Open a Web Browser**
    - Launch **Google Chrome**, **Edge**, or any preferred browser.
  2. **Navigate to Azure DevOps**
    - Go to [Azure DevOps](#).
  3. **Sign In or Sign Up**
    - If you already have a Microsoft account, click **Sign in** and enter your credentials.
    - If you don't have an account, click **Start Free** and sign up using your Microsoft, GitHub, or work email account.
- 

### **Step 3: Creating an Azure DevOps Organization**

1. **Click on 'Create New Organization'**
    - After signing in, you'll be prompted to create an organization.
  2. **Provide Organization Name and Region**
    - Enter a unique name for your organization.
    - Choose a region closest to your location for better performance.
  3. **Click 'Continue'**
    - Azure DevOps will set up your organization.
- 

### **Step 4: Creating a New Project**

1. **Click 'New Project'**
  - After creating an organization, you'll be directed to the project creation page.
2. **Enter Project Name**

- Provide a meaningful name for your project.
  - 3. **Choose Visibility**
    - Select **Private** (for internal use) or **Public** (for open-source projects).
  - 4. **Choose Version Control and Work Item Process**
    - Select **Git** or **Team Foundation Version Control (TFVC)** for version control.
    - Choose a work item process: **Basic, Agile, Scrum, or CMMI**.
  - 5. **Click ‘Create’**
    - Azure DevOps will generate your project.
- 

## Step 5: Exploring Azure DevOps Services

1. **Navigate to the Project Dashboard**
  - Click on the project name to access services like **Repos, Pipelines, Boards, Test Plans, and Artifacts**.
2. **Configure Teams and Permissions**
  - Click on **Project Settings** → **Teams** to add members.
  - Assign roles and permissions as needed.



## **Program 10: Creating Build Pipelines: Building a Maven/Gradle Project with Azure Pipelines, Integrating Code Repositories (e.g., GitHub, Azure Repos), Running Unit Tests and Generating Reports.**

### **Step 1: Prerequisites**

Before starting, ensure you have: ✓An **Azure DevOps account** – [Sign up here](#)

✓A **GitHub/Azure Repos** repository with a **Maven/Gradle project**

✓The necessary **Azure DevOps permissions** to create pipelines

---

### **Step 2: Setting Up a New Build Pipeline**

1. **Sign in to Azure DevOps**
    - Open a browser and go to [Azure DevOps](#).
    - Select your organization and project.
  2. **Navigate to Pipelines**
    - Click on **Pipelines** (left sidebar).
    - Click **New Pipeline**.
  3. **Choose a Code Repository**
    - Select where your project code is stored:
      - **GitHub** (sign in and select your repository).
      - **Azure Repos Git** (select your project repository).
  4. **Choose a Pipeline Configuration**
    - Select **Starter Pipeline** or **Maven/Gradle template** (if available).
    - If using a starter pipeline, you'll configure the YAML manually.
- 

### **Step 3: Configuring the YAML Pipeline for a Maven/Gradle Build**

1. If using **Maven**, update the YAML with:

```
yaml

trigger:
- main

pool:
  vmImage: 'windows-latest'

steps:
- task: Maven@4
```

```
inputs:
  mavenPomFile: 'pom.xml'
  goals: 'clean install'
  publishJUnitResults: true
  testResultsFiles: '**/surefire-reports/TEST-*.xml'
```

2. If using **Gradle**, use:

```
yaml

trigger:
- main

pool:
  vmImage: 'windows-latest'

steps:
- task: Gradle@3
  inputs:
    gradleWrapperFile: 'gradlew'
    workingDirectory: ''
    gradleOptions: '-Xmx3072m'
    tasks: 'clean build'
    publishJUnitResults: true
    testResultsFiles: '**/TEST-*.xml'
```

3. Click **Save and Run** to trigger the build.

---

## Step 4: Running Unit Tests & Generating Reports

1. **Ensure Unit Tests Run in the Build**

- The **Maven** or **Gradle** configuration automatically runs tests.
- Test reports are generated in:
  - **Maven** → target/surefire-reports/
  - **Gradle** → build/reports/tests/test/

2. **Publish Test Results**

- Azure Pipelines automatically detects test results from JUnit and displays them under "**Tests**" in the pipeline summary.
- 

## Step 5: Viewing Build and Test Results

1. **Check the Pipeline Status**

- Navigate to **Pipelines > Your Pipeline** and click the latest run.
- View logs to troubleshoot errors if any.

2. **View Test Reports**

- Go to the "**Tests**" tab in the pipeline summary to see results.

# Program 11: Creating Release Pipelines: Deploying Applications to Azure App Services, Managing Secrets and Configuration with Azure Key Vault, Hands-On: Continuous Deployment with Azure Pipelines.

## Overview

- **Objective:** Automate deployment to **Azure App Services** using **Azure Pipelines**.
  - **Tools Used:** Azure DevOps, Azure App Service, Azure Key Vault.
  - **Key Features:** **Continuous Deployment (CD)**, **Secrets Management**, **Automated Deployments**.
- 

## Step 1: Prerequisites

Before setting up a release pipeline, ensure: ✓An **Azure DevOps account** – [Sign up here](#)

✓A **GitHub/Azure Repos** repository with your application code

✓An **Azure App Service** (Web App) created in [Azure Portal](#)

✓An **Azure Key Vault** for secret management (optional but recommended)

✓The **Azure DevOps Service Connection** is set up to access Azure resources.

---

## Step 2: Creating a Build Pipeline for Continuous Deployment

1. **Sign in to Azure DevOps**
  - Go to [Azure DevOps](#).
  - Select your project.
2. **Create a New Build Pipeline**
  - Navigate to **Pipelines > New Pipeline**.
  - Select your **code repository** (GitHub/Azure Repos).
  - Choose a template:
    - **For .NET applications:**

```
yaml
trigger:
- main

pool:
vmImage: 'windows-latest'

steps:
```

```

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
- task: DotNetCoreCLI@2
  inputs:
    command: 'publish'
    publishWebProjects: true
    arguments: '--configuration Release --output
$(Build.ArtifactStagingDirectory) '
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory) '
    artifactName: 'drop'

```

- Click **Save and Run**.

---

## Step 3: Creating a Release Pipeline for Deployment

1. **Go to Releases**
  - In **Azure DevOps**, navigate to **Pipelines > Releases**.
  - Click **New Release Pipeline**.
2. **Select Deployment Template**
  - Click **Add a Stage** and choose **Azure App Service deployment**.
3. **Connect to Azure App Service**
  - Click **Stage 1 → Tasks**.
  - Under **Azure subscription**, select your existing service connection.
  - Under **App Service Name**, choose the correct Web App.
  - Click **Save**.
4. **Select the Build Artifact**
  - Click **Add an Artifact**.
  - Choose **Build** as the source and select your **latest pipeline build**.
  - Click **Save**.

---

## Step 4: Managing Secrets with Azure Key Vault

1. **Go to Azure Portal**
  - Navigate to **Azure Key Vault**.
2. **Add Secrets**
  - Click **Secrets > Generate/Import**.
  - Add connection strings, API keys, and secrets.
3. **Configure Key Vault in Azure DevOps**
  - In **Azure DevOps**, go to **Library > Secure Files > Add Azure Key Vault**.
  - Link it to your Azure subscription and Key Vault.
  - In your **release pipeline**, add a new **Azure Key Vault** task:

yaml

```
- task: AzureKeyVault@1
  inputs:
    azureSubscription: '<Your Service Connection>'
    KeyVaultName: '<YourKeyVaultName>'
    SecretsFilter: '*'
    RunAsPreJob: false
```

- Save the pipeline.

---

## Step 5: Enable Continuous Deployment (CD)

1. **Go to Your Release Pipeline**
  - Click on **Triggers**.
  - Enable **Continuous Deployment Trigger**.
  - Select the build artifact and check **Enabled**.
2. **Deploy the Application**
  - Click **Create Release → Deploy**.

---

## Step 6: Monitor Deployment

1. **Check the Release Pipeline Logs**
  - Go to **Releases > Your Release > Logs**.
  - Verify that the deployment completed successfully.
2. **Verify in Azure App Service**
  - Open the **Azure Portal → App Services**.
  - Click your Web App and check if the latest deployment is active.

## Program 12: Practical Exercise and Wrap-Up: Build and Deploy a Complete DevOps Pipeline, Discussion on Best Practices and Q&A.

### Step 1: Prerequisites

Before starting, ensure you have:

- ✓ **Azure DevOps account** ([Sign up here](#))
  - ✓ **Azure App Service** created in **Azure Portal** ([portal.azure.com](https://portal.azure.com))
  - ✓ **Code repository** on **GitHub** or **Azure Repos**
  - ✓ **Azure Key Vault** (optional but recommended)
  - ✓ **Azure DevOps Service Connection** set up for deployment
- 

### Step 2: Creating a Build Pipeline (CI - Continuous Integration)

#### 1. Open Azure DevOps and Set Up a Build Pipeline

- Sign in to **Azure DevOps**
- Select your **Project**
- Go to **Pipelines > New Pipeline**
- Choose your **code repository** (GitHub/Azure Repos)

#### 2. Select a Build Configuration

Choose a predefined **Maven/Gradle/.NET/NPM** template or use a custom **YAML pipeline**.

##### *Example: YAML for a .NET Build Pipeline*

```
yaml
CopyEdit
trigger:
  - main

pool:
  vmImage: 'windows-latest'

steps:
  - task: UseDotNet@2
    inputs:
      packageType: 'sdk'
      version: '6.x'
      installationPath: $(Agent.ToolsDirectory)/dotnet

  - task: DotNetCoreCLI@2
    inputs:
      command: 'restore'
```

```

    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
    arguments: '--configuration Release'

- task: DotNetCoreCLI@2
  inputs:
    command: 'publish'
    publishWebProjects: true
    arguments: '--configuration Release --output
$(Build.ArtifactStagingDirectory) '

- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory) '
    artifactName: 'drop'

```

- **Save and Run** the pipeline.
- This will generate a **build artifact** named **drop**, which will be used in deployment.

---

## Step 3: Creating a Release Pipeline (CD - Continuous Deployment)

### 1. Navigate to Releases

- Go to **Azure DevOps** → **Pipelines** → **Releases**
- Click **New Release Pipeline**

### 2. Add Deployment Stages

- Click **Add a stage** → Select **Azure App Service deployment**
- Name the stage **Production Deployment**

### 3. Configure Azure App Service Deployment

- Click **Stage 1 > Tasks**
- Under **Azure subscription**, select your **Azure Service Connection**
- Select the **App Service** for deployment
- Under **Package or Folder**, select `$(System.DefaultWorkingDirectory)/drop`

### 4. Add Azure Key Vault for Secret Management (Optional but Recommended)

- In the **release pipeline**, click **+ Add Task**
- Choose **Azure Key Vault** and enter:

```
yaml
CopyEdit
- task: AzureKeyVault@1
  inputs:
    azureSubscription: '<YourServiceConnection>'
    KeyVaultName: '<YourKeyVaultName>'
    SecretsFilter: '*'
    RunAsPreJob: false
```

- **Save** the pipeline

## 5. Enable Continuous Deployment (CD)

- Click **Triggers** in the release pipeline
- Enable **Continuous Deployment Trigger**
- Select the **Build Artifact** → **Enabled**

## 6. Deploy the Application

- Click **Create Release** → **Deploy**
- Monitor deployment logs under **Releases > Logs**

---

## Step 4: Testing & Monitoring Deployment

1. **Check Azure App Service Deployment**
  - Open **Azure Portal** → **App Services**
  - Select your **Web App** and confirm the deployment
2. **Check Azure DevOps Logs**
  - Navigate to **Releases** → Click on the latest **release**
  - Review logs to ensure deployment succeeded

---

## Step 5: Best Practices for Azure DevOps Pipelines

- ✓ **Use YAML for Pipeline as Code** – Improves **version control** & collaboration
  - ✓ **Enable Pull Request (PR) Triggers** – Ensures **code quality** before merging
  - ✓ **Use Secrets from Azure Key Vault** – Securely manage API keys & credentials
  - ✓ **Implement CI/CD in Stages** – Use **staging environments** before production
  - ✓ **Monitor Pipelines with Azure Monitor & Application Insights** – Improves **observability**
  - ✓ **Use Infrastructure as Code (IaC)** – Automate infrastructure with **Terraform/Bicep**
-



## Step 6: Wrap-Up and Q&A

### Discussion Topics:

- What challenges did you face while setting up the pipeline?
- How can we improve performance in CI/CD?
- What additional security measures should be taken?