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		
Coding bootcamps	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

Maven

Step 1: Install OpenJDK

1. Update the system's package repository index: *sudo apt update*

2. Install the latest OpenJDK version by running: sudo apt install default-jdk -y

3. Verify the installation by checking the current OpenJDK version: *java -version*

Step 2: Download and Install Maven

1. Visit the Maven download page.

2. Right-click the version of Maven you want to install and copy the link.

3. Download the Maven installation file to the */tmp* directory using the wget command. The syntax is: *wget [link] -P /tmp*

4. Once the download is complete, extract the installation file to the */opt* directory: *sudo tar xf /tmp/apache-maven-*.tar.gz -C /opt*

5. Create a symbolic link leading to the Maven installation directory: sudo ln -s /opt/apache-maven-3.9.9 /opt/maven

Step 3: Set Up Environment Variables

1. Use a text editor like Nano to create and open the *maven.sh* script file in the */etc/profile.d/* directory: sudo nano /etc/profile.d/maven.sh

2. Add the following lines to the *maven.sh* file:

export JAVA_HOME=/usr/lib/jvm/default-java

export M2_HOME=/opt/maven

export MAVEN_HOME=/opt/maven

export PATH=${M2_HOME}/bin:${PATH}

3. Use the chmod command to make the *maven.sh* file executable: sudo chmod +x /etc/profile.d/maven.sh

4. Execute the *maven.sh* script file with the source command to set up the new environment variables: source /etc/profile.d/maven.sh

Step 4: Verify Maven Installation

mvn -version

https://services.gradle.org/distributions/gradle-8.13-bin.zip

Gradle

1. Update the server's package index.

   sudo apt update

2. Install the required Java Development Kit (JDK) package.

   sudo apt install default-jdk -y

3. View the installed Java version.

java --version

4. Visit the Gradle release page, identify the latest version, and copy its binary direct download link. Then, use wget to download the file. For example, download the Gradle 8.12 version.
   wget https://services.gradle.org/distributions/gradle-8.12-bin.zip
5. Extract the Gradle ZIP archive contents to a system-wide directory *such as /opt*
   sudo unzip -d /opt/gradle gradle-8.12-bin.zip

Setup Environment Variables

1. Create a new gradle.sh script in the /etc/profile.d directory.
   sudo nano /etc/profile.d/gradle.sh
2. Add the following directives to the gradle.sh file.
   Save and close the file.
   export GRADLE_HOME=/opt/gradle/gradle-8.12
   export PATH=${GRADLE_HOME}/bin:${PATH}
3. Enable execute permissions on the script.
   sudo chmod +x /etc/profile.d/gradle.sh
4. Load the Gradle environment configuration in your active shell.
   source /etc/profile.d/gradle.sh
5. Verify the installed Gradle version.
   gradle --version

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

Working with Maven Project

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.
- Let's use the Apache Commons Lang library as a dependency (which provides utilities for working with strings, numbers, etc.). We will use this in a simple Java program to work with strings.

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

Step 2: Open The pom.xml File

- You can manually navigate the **project folder** named call **myapp** 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 myapp** – 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>myapp</artifactId>

        <version>1.0-SNAPSHOT</version>


        <dependencies>

                <!-- JUnit Dependency for Testing -->

                <dependency>

                        <groupId>junit</groupId>

                        <artifactId>junit</artifactId>

                        <version>4.13.2</version>

                        <scope>test</scope>

                </dependency>
```

```
    </dependencies>


    <build>
        <plugins>
            <!-- Maven Surefire Plugin for running tests -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.2</version>
                <configuration>
            <redirectTestOutputToFile>false</redirectTestOutputToFile>
                <useSystemOut>true</useSystemOut>
                </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 int add(int a, int b) {
        return a + b;
    }
```

```
    public static void main(String[] args) {
        App app = new App();

        int result = app.add(2, 3);
        System.out.println("2 + 3 = " + result);

        System.out.println("Application executed successfully!");
    }
}
```

Step 4: Open Java Code (`AppTest.java`) File

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

```
package com.example;

import org.junit.Assert;
import org.junit.Test;

public class AppTest {

        @Test
        public void testAdd() {
        App app = new App();
        int result = app.add(2, 3);

        System.out.println("Running test: 2 + 3 = " + result);

        Assert.assertEquals(5, result);
        }
}
```

Step 4: Building the Project

To build and run this project, follow these steps:

1. Compile the Project

```
mvn compile
```

2. Run the Unit Tests

```
mvn test
```

3. Package the project into a JAR
mvn package
4. Run the application (using JAR)
java -cp target/myapp-1.0-SNAPSHOT.jar com.example.App

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

- Create a new sample_project directory to use with Gradle: mkdir sample_project
- Switch to the sample_project directory: gradle init --type java-application
- Create the src/main/java directory to store your application files: mkdir -p src/main/java
- Create a new App.java Java application file in the src/main/java directory using a text editor such as nano: nano src/main/java/App.java
- Add following contents to the file

```
public class App {
    public static void main(String[] args) {
        System.out.println("Greetings from CSD!");
    }
}
```

- Open the build.gradle file to ensure it includes the Java plugin and specifies the main class: nano build.gradle

```
plugins {
    id 'application'
}

application {
    // Define the main class
    mainClass = 'App'
}

jar {
    manifest {
        attributes(
            'Main-Class': 'App'
        )
    }
```

```
}

repositories {
    mavenCentral()
}

dependencies {
    // Add dependencies here if needed
}
```

- Build and compile the Java application into a single file using Gradle: gradle build
- Run the application using Gradle: gradle run
- Run the bundled .jar file using Java.: java -jar build/libs/sample_project.jar App

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;
    }
}
```

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"

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

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
}
```

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

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

1. What Is Jenkins?

Definition and Overview

Jenkins is an open-source automation server written in Java. It is widely used to facilitate **Continuous Integration (CI)** and **Continuous Deployment/Delivery (CD)** pipelines in software development. Jenkins automates repetitive tasks related to building, testing, and deploying software, helping teams to integrate changes continuously and deliver high-quality applications faster.

Key Functionalities

- **Continuous Integration/Delivery:** Automatically builds, tests, and deploys code after each commit.
- **Extensible Plugin Ecosystem:** Over 1,500 plugins allow integration with numerous tools such as Git, Maven, Gradle, Docker, and many cloud providers.
- **Automated Builds:** Jenkins can poll version control systems, trigger builds on code changes, and even schedule builds.
- **Pipeline as Code:** With the introduction of Jenkins Pipeline (using either scripted or declarative syntax), you can define your entire build process in a code file (Jenkinsfile) and store it alongside your code.
- **Distributed Builds:** Jenkins can distribute workloads across multiple machines, helping to run tests and builds faster.
- **Monitoring and Reporting:** Provides detailed logs, test reports

and build history.

Why Use Jenkins?

- **Improved Efficiency:** Automates mundane tasks to free up developer time. • **Rapid Feedback:** Quickly identifies

integration issues with automated tests. • **Scalability:**

Supports distributed builds and parallel testing.

• **Customization:** A highly configurable system that can integrate with nearly every tool in the software development lifecycle.

2. Installing Jenkins on a Local Machine (Ubuntu)

Below are detailed step-by-step instructions for installing Jenkins on an Ubuntu machine.

Step 1: Update Your System

Open your terminal and update your system repositories:

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

Step 2: Install Java

Jenkins requires Java to run. It is recommended to use Java 11 or later.
Install OpenJDK 11:

```
sudo apt install openjdk-21-jdk -y
```

Step 3: Add the Jenkins Repository Key

Download and add the repository key so that your system trusts the Jenkins packages:

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt key add -
```

Step 4: Add the Jenkins Repository

Add the Jenkins repository to your system's sources list:

```
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

Step 5: Update the Repository and Install Jenkins

Update the package list and install Jenkins:

```
sudo apt update
sudo apt install jenkins -y
```

Step 6: Start and Enable the Jenkins Service

Start Jenkins and configure it to start automatically on boot:

```
sudo systemctl start jenkins
sudo systemctl status jenkins
```

- **What it does:**
  - `start jenkins` launches the Jenkins service.
  - `status jenkins` confirms Jenkins is running.
- *Expected Output:* A status message indicating Jenkins is active (running).

Step 7: Access Jenkins on the Local

Machine

1. **Open a Web Browser:** Navigate to:
2. http://localhost:8080
3. **Unlock Jenkins:** The initial Jenkins screen will ask for an **administrator password**. Retrieve it by running:
4. sudo cat /var/lib/jenkins/secrets/initialAdminPassword o
   - **What it does:** Displays the auto-generated admin password.
   - o *Screenshot Tip:* Capture the terminal output showing the password and the Jenkins unlock screen.
5. **Follow the Setup Wizard:**
   - o **Install Suggested Plugins:** Click on "Install suggested plugins" for a typical setup.
   - o **Create an Admin User:** Follow prompts to create your first admin user.
   - o **Finalize Configuration:** Complete the remaining setup steps (e.g., instance configuration).
   - o *Screenshot Tip:* Capture each major step (unlocking Jenkins, plugin installation, and admin user creation).

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

Setting Up a CI Pipeline with Jenkins (Freestyle Project)

This section explains how to create a CI pipeline as a Freestyle project that integrates with a Maven or Gradle project.

Step 1: Create a New Jenkins Job

1. **Log into Jenkins:**
   - Open your web browser and navigate to your Jenkins URL (e.g., http://localhost:8080 or your cloud instance URL).
   - Log in with your admin credentials.
2. **Create a New Job:**
   - On the Jenkins dashboard, click on **"New Item"**.
   - **Enter an Item Name:** For example, Maven-CI (or Gradle-CI if you prefer Gradle).
   - **Select "Freestyle project"**.
   - Click **"OK"**.

Step 2: Configure Source Code Management (SCM)

1. **Select SCM:**
   - In the job configuration page, scroll down to the **"Source Code Management"** section.
   - Select **"Git"** (if using Git for version control).
2. **Enter Repository Details:**
   - **Repository URL:** Enter the URL of your Git repository (for example, https://github.com/yourusername/your-maven-project.git).
   - **Credentials:** If your repository is private, click **"Add"** to provide the necessary credentials.
   - Optionally, specify the **Branch Specifier** (e.g., */main).

Step 3: Add Build Steps

A. For a Maven Project

1. **Add Maven Build Step:**
   - Scroll down to **"Build"** and click on **"Add build step"**.
   - Select **"Invoke top-level Maven targets"**.
   - **Goals:** In the Goals field, enter:
   - clean package

     This command instructs Maven to clean the previous build artifacts, compile the code, run tests, and package the

application into a JAR/WAR file.

- Optionally, set the **POM File** location if it is not in the default location

B. For a Gradle Project

1. **Add Gradle Build Step:**
   - If you are integrating a Gradle project, click on **"Add build step"** and choose **"Invoke Gradle script"** (this option might be available if you have installed a Gradle plugin in Jenkins).
   - **Tasks:** In the tasks field, enter:
   - clean build

     This instructs Gradle to clean previous build outputs and then build the project, running tests along the way.
   - **Switches:** If needed, you can add additional flags (for example, --info or -- stacktrace for more detailed output).

Step 4: Configure Post-build Actions

1. **Publish Test Results:**
   - Scroll down to the **"Post-build Actions"** section.
   - Click **"Add post-build action"** and select **"Publish JUnit test result report"**.
   - **Test Report XMLs:** In the field, enter the pattern that matches your test report files. For example:
   - **/target/surefire-reports/*.xml

     for Maven, or a similar path for Gradle projects.

Step 5: Save and Run the Job

1. **Save the Configuration:**
   - Click **"Save"** at the bottom of the job configuration page.
2. **Trigger a Build:**
   - On the job's main page, click **"Build Now"**.
   - The build will be added to the build history on the left side.
3. **Monitor Build Output:**
   - Verify that Jenkins successfully checks out the code, runs the build commands (Maven or Gradle), and executes tests.
   - Look for **"BUILD SUCCESS"** or the equivalent output to confirm that the build and tests passed.

Setting Up a CI Pipeline with Jenkins (Pipeline as Code)

For greater flexibility and version-controlled CI configuration, you can use a **Jenkins Pipeline** defined in a `Jenkinsfile`.

Step 1: Create a Pipeline Job

1. **Log into Jenkins** and click on **"New Item"**.
2. **Enter an Item Name:** For example, `Pipeline-CI`.
3. **Select "Pipeline"** and click **"OK"**.

Step 2: Define the Pipeline Script

1. **Configure the Pipeline:**
   - In the job configuration page, scroll to the **"Pipeline"** section.
   - Choose **"Pipeline script"** (or "Pipeline script from SCM" if you want to load the script from your repository).
2. **Enter the Pipeline Script:**

   Below are sample pipeline scripts for Maven and Gradle projects.

   Example for a Maven Project:

```
pipeline {
    agent any
    stages {
        stage('Clone and Build') {
            steps {
                //
```

```
                }
            }
        stage('Test') {
            steps {
                //
                }
            }
        stage('Deploy') {
            steps {
                //
                }
            }
    }
```

3. **Save the Pipeline Script:**
    - After entering your pipeline script, click **"Save"**.

Step 3: Run the Pipeline

1. **Trigger the Build:**
    - On the Pipeline job's main page, click **"Build Now"**.
    - Monitor the build progress through the Pipeline visualization or by clicking

on the build number and then **"Console Output"**.

2. **Verify the Results:**

   - Confirm that each stage (Checkout, Build, Test) executes successfully.
   - Review the archived test reports to verify that tests have run and passed.

Experiment 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

1. Introduction to Ansible

What Is Ansible?

Ansible is an open-source IT automation and configuration management tool. It allows you to manage multiple servers and perform tasks such as:

- **Configuration Management:** Automate the configuration of servers.
- **Application Deployment:** Deploy applications consistently.
- **Orchestration:** Coordinate complex IT workflows and processes.

Key Concepts in Ansible

- **Inventory:**

  An inventory is a file (usually in INI or YAML format) that lists the hosts (or groups of hosts) you want to manage. It tells Ansible which machines to target.

- **Playbook:**

  A playbook is a YAML file that defines a set of tasks to be executed on your target hosts. It is the heart of Ansible automation. In a playbook, you specify:

  - **Hosts:** The target machines (or groups) on which the tasks should run.

- **Tasks:** A list of actions (using modules) that should be executed.
- **Modules:** Reusable, standalone scripts that perform specific actions (e.g., installing packages, copying files, configuring services).

- **Modules:**
  Ansible comes with a large collection of built-in modules (such as apt, yum, copy, service, etc.). These modules perform specific tasks on target hosts. You can also write custom modules if needed.

Why Use Ansible?

- **Agentless:** Ansible uses SSH to communicate with target hosts, so no agent needs to be installed on them.
- **Simplicity:** Playbooks use simple YAML syntax, making them easy to write and understand.
- **Idempotence:** Ansible tasks are idempotent, meaning running the same playbook multiple times yields the same result, ensuring consistency.
- **Scalability:** Ansible can manage a small number of servers to large infrastructures with hundreds or thousands of nodes.

2. Installing Ansible on Ubuntu

Before writing a playbook, you need to install Ansible on your control machine (your local Ubuntu system).

Step 1: Update Your System

Open your terminal and run:

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

Step 2: Install Ansible

Install Ansible using apt:

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

Verify the installation by checking the version:

```
ansible --version
```

3. Creating an Ansible Inventory

An inventory file lists the hosts you want to manage. For this experiment, you can use the local host.

Step 1: Create an Inventory File

1. Open your text editor to create a file called hosts.ini:
nano hosts.ini
2. Add the following content to define the local host:

```
[local]
localhost ansible_connection=local
```

- **Explanation:**
  - [local] is a group name.
  - localhost is the target host.
  - ansible_connection=local tells Ansible to execute commands on the local machine without SSH.

3. Save the file by pressing **Ctrl+O** then **Enter**, and exit with **Ctrl+X**.

5. Writing a Basic Ansible Playbook

You will now create a simple playbook that performs two common tasks:

- **Updating the apt cache**
- **Installing a package (e.g., curl)**

Step 1: Create the Playbook File

1. Open your text editor to create a file called setup.yml: nano setup.yml
2. Add the following YAML content:

```
---
- name: Basic Server Setup
  hosts: local
  become: yes # Use privilege escalation (sudo)
  tasks:
    - name: Update apt cache
      apt:
      update_cache: yes

    - name: Install curl
      apt:
      name: curl
      state: present
```

- **Explanation:**
  - name: Provides a descriptive name for the play.
  - hosts: Specifies the group or hosts from the inventory file.
  - become: yes: Uses sudo to perform tasks that require elevated privileges.
  - **Tasks Section:**
    - **Update apt cache:** Uses the apt module to update the package cache.
    - **Install curl:** Uses the apt module to install the curl package if it isn't already installed.

3. Save and exit the file.

6. Running the Ansible Playbook

Step 1: Execute the Playbook

In your terminal, run the following command:

```
ansible-playbook -i hosts.ini setup.yml
```

- **Explanation:**
  - `ansible-playbook`: The command to run an Ansible playbook.
  - `-i hosts.ini`: Specifies the inventory file.
  - `setup.yml`: The playbook file you just created.

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

1. Overview

In this experiment, you will:

- Set up a Jenkins job to automatically build a Maven project from source control.
- Archive the build artifact (a JAR file) produced by Maven.
- Integrate an Ansible deployment step within Jenkins (using a post-build action) to deploy the artifact to a target location.
- Verify that the artifact is deployed successfully.

This exercise demonstrates how Continuous Integration (CI) and automated configuration management can work together to streamline the build-and-deploy process.

2. Prerequisites

Before you begin, ensure that:

- **Jenkins** is installed, running, and accessible (locally or on the cloud).
- **Maven Project:** You have a Maven project available in a Git repository (or stored locally). For this example, we will assume you are using the "HelloMaven" project generated in Experiment 2/4.
- **Git Repository:** The Maven project is committed to a Git repository (e.g., on GitHub) so Jenkins can pull the latest code.
- **Ansible Installed:** Ansible is installed on your control machine (or the Jenkins server) and you have created a basic inventory file (e.g., `hosts.ini`).

3. Step 1: Preparing the Maven Project

1. **Ensure Your Maven Project Is in Version Control:** If your "HelloMaven" project is not already in a Git repository, navigate to its root and initialize Git:

```
2. cd path/to/HelloMaven
3. git init
4. git add .
5. git commit -m "Initial commit of HelloMaven project"
```

Then push it to your Github repository.

4. Step 2: Configuring Jenkins to Build the Maven

Project A. Create a New Jenkins Job (Freestyle Project)

1. **Log into Jenkins:** Open your browser and navigate to your Jenkins URL (e.g., `http://localhost:8080`).
2. **Create a New Job:**
   - Click **"New Item"** on the Jenkins dashboard.
   - **Enter an Item Name:** e.g., `HelloMaven-CI`.
   - **Select "Freestyle project"** and click **"OK"**.

B. Configure Source Code Management (SCM)

1. **Scroll to the "Source Code Management" Section:**
   - Select **"Git"**.
   - **Repository URL:** Enter your repository URL (e.g., `https://github.com/yourusername/HelloMaven.git`).
   - **Credentials:** If the repository is private, click **"Add"** and provide the necessary credentials.
   - **Branch Specifier:** (e.g., `*/main` or `*/master`).

C. Add a Maven Build Step

1. **Scroll Down to the "Build" Section:**
   - Click **"Add build step"** and select **"Invoke top-level Maven targets"**. 2. **Configure the Maven Build:**
   - **Goals:** Type:
   - `compile test package`

     This command cleans any previous builds, compiles the code, runs tests, and packages the application into a JAR file.
   - **POM File:** (Leave it as default if your `pom.xml` is in the root directory).

5. Step 3: Archiving the Artifact

After the Maven build completes, you need to archive the generated artifact so that it can be used later by the deployment process.

1. **Scroll Down to the "Post-build Actions" Section:**
   - Click **"Add post-build action"** and select **"Archive the artifacts"**. 2. **Configure Artifact Archiving:**
   - **Files to Archive:** Type:
   - `target/*.jar`

     This pattern tells Jenkins to archive any JAR file found in the `target` directory.

6. Step 4: Integrating Ansible Deployment in Jenkins

Now, integrate an Ansible deployment step into the Jenkins job. You can do this as a post-build action that executes a shell command.

1. **Add Another Post-build Action:**
   - Click **"Add post-build action"** and select **"Execute shell"**.
2. **Configure the Shell Command:**
   - In the command box, add a command to trigger your Ansible playbook. For example:
   - `ansible-playbook -i /path/to/hosts.ini /path/to/deploy.yml`

```
hosts.ini

[local]
localhost ansible_connection=local



nano deploy.yml


---
- name: Deploy Maven Artifact
  hosts: local
  become: yes
  tasks:
   - name: Copy the artifact to the deployment directory 10.
     copy:
       src: "/var/lib/jenkins/workspace/HelloMaven                  CI/target/HelloMaven-
1.0-SNAPSHOT.jar"
       dest: "/opt/deployment/HelloMaven.jar"
```