**Experiment 3: Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation  
Objective:** To learn how to set up a simple Gradle project, understand the build script (build.gradle), manage dependencies, and automate tasks using Gradle on Ubuntu. **Introduction to Gradle: What is Gradle?**  
Gradle is a **flexible and modern build automation tool** used for Java, Groovy, Kotlin, and other languages. It allows developers to define build logic using a **domain-specific language (DSL)** in Groovy or Kotlin instead of XML. **Key Features of Gradle: >** Uses **incremental builds** and **task caching** for speed. **>** Build scripts are written in **Groovy or Kotlin**, not XML. **>** Supports **multi-language** and **multi-module** projects. **>** Works well with **Maven repositories**. **>** Integrates seamlessly with **CI/CD tools** like Jenkins. **Step 1: Create Project Directory**Open your terminal and execute:mkdir sample\_projectcd sample\_project **Step 2: Initialize Gradle Project**Use Gradle’s init command to generate a Java application project:gradle init --type java-application **Step 3: Create Application Code**If not already created, make the directory:mkdir -p src/main/javaNow create the App.java file:nano src/main/java/App.javaPaste the following content:public class App {

public static void main(String[] args) {

System.out.println("Greetings from CSD!");

}

}  
**Step 4: Review and Edit build.gradle**  
Open the Gradle build file: nano build.gradle  
Make sure the file includes the following content:  
plugins {

id 'application'

}

application {

mainClass = 'App'

}

jar {

manifest {

attributes(

dependencies {

// Add dependencies here if needed

}

**Step 5: Build the Application**  
To compile and package the project, run: gradle build  
Gradle compiles the source code and packages it into a JAR file in: build/libs/sample\_project.jar  
**Step 6: Run the Application Using Gradle**  
gradle run  
This runs the main method in App.java. Output: Greetings from CSD!

'Main-Class': 'App'

)

}

}

repositories {

mavenCentral()

}

**Experiment 4: Practical Exercise: Build and Run a Java Application with Maven, Then Migrate the Same Application to Gradle  
Objective:>** Create a Java project using **Maven**. **>** Build and run the application using Maven. **>** Migrate the same project to **Gradle**. **>** Understand how both tools handle project builds. **Part A: Build and Run Java Application Using Maven  
Step 1: Create Maven Project**Open terminal and run:mvn archetype:generate -DgroupId=com.example -DartifactId=maven-example -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false **Step 2: Edit pom.xml**Navigate to the project directory:cd maven-exampleOpen pom.xml:nano pom.xmlReplace its contents with the following:<?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: Write Application Code (App.java)**Navigate to the source folder: cd src/main/java/com/example/Edit App.java: nano App.javaPaste the code:package com.example;public class App {

public static void main(String[] args) {

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

System.out.println("This is the simple real-world 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: Build and Run with Maven**  
Back in the project root directory: cd ~/maven-example

**1. Build the Project**mvn clean install  
**2. Run the App**mvn exec:java -Dexec.mainClass="com.example.App"Make sure you have the exec-maven-plugin configured. If not, add this to pom.xml inside the <plugins> section:

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.example.App</mainClass>

</configuration>

</plugin>

Then re-run:  
mvn compile exec:java

**Part B: Migrate Maven Project to Gradle**

**Step 1: Initialize Gradle in the Same Project Directory**

Make sure you’re still inside the maven-example directory.

Run: gradle init

You’ll be prompted: Found a Maven build. Generate a Gradle build from this?  
→ Type: yes  
Select build script DSL:  
→ Type: 2 (for Groovy)  
Generate build using new APIs and behavior?  
→ Type: no  
Gradle will generate build.gradle, settings.gradle, and related files.

**Step 2: Modify build.gradle**Open the file:nano build.gradleEdit to match the following: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 3: Build the Gradle Project**

./gradlew build  
**Step 4: Run the Application**  
./gradlew run  
Output should display:  
Hello, Maven  
This is the simple real-world example....  
Sum of 5 and 10 is 15

**Experiment 5: Introduction to Jenkins: What is Jenkins? Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use  
Objective:>** Understand what Jenkins is and why it’s used. **>** Install Jenkins on an **Ubuntu machine**. **>** Perform the initial configuration and setup. **What is Jenkins: Jenkins** is an **open-source automation server** written in Java. It enables developers to build, test, and deploy code continuously through a process known as **CI/CD (Continuous Integration and Continuous Delivery/Deployment).  
Key Functionalities of Jenkins:> Continuous Integration (CI):** Automatically build and test code after each commit. **> Extensible Plugin Support:** Offers 1500+ plugins to integrate with tools like Git, Maven, Gradle, Docker, etc. **> Automated Builds:** Automatically triggers builds based on changes in the version control system. **> Pipeline as Code:** You can define and manage your CI/CD workflows in a code file called Jenkinsfile. **> Distributed Builds:** Jenkins can use multiple nodes to run jobs in parallel. **> Monitoring & Reporting:** Gives detailed logs, test results, and build history. **Why Use Jenkins? > Efficiency:** Automates repetitive and time-consuming development tasks.**> Rapid Feedback:** Instantly notifies developers if the build or tests fail. **> Scalability:** Easily handles large-scale build pipelines with multiple jobs. **> Flexibility:** Works with almost all languages and environments via plugins. **Step-by-Step: Installing Jenkins on Ubuntu  
Step 1: Update Your System**Open a terminal and run:sudo apt updatesudo apt upgrade -y

**Step 2: Install Java (OpenJDK 21)**Jenkins runs on Java, so install Java first:sudo apt install openjdk-21-jdk -yYou can verify the installation:java -version

**Step 3: Add Jenkins Repository Key**Download the Jenkins key and add it:wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -If you get a warning about apt-key being deprecated, you can use this method:curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | gpg --dearmor | sudo tee /usr/share/keyrings/jenkins-keyring.gpg > /dev/null **Step 4: Add Jenkins Repository**Now add Jenkins repository to APT sources:sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list' **Step 5: Install Jenkins**First update your package list:sudo apt updateThen install Jenkins:sudo apt install jenkins -y **Step 6: Start and Enable Jenkins Service**Start Jenkins and make it launch automatically on boot:sudo systemctl start Jenkinssudo systemctl enable JenkinsCheck the status:sudo systemctl status Jenkins **Step 7: Access Jenkins in Browser**Open your web browser and go to:<http://localhost:8080>You’ll be asked for an **Administrator password**.Get it by running:sudo cat /var/lib/jenkins/secrets/initialAdminPasswordCopy this password and paste it into the browser input box. **Step 8: Setup Jenkins**Once logged in: **Install Suggested Plugins**Click on “Install suggested plugins”.Jenkins will automatically download and install essential plugins. **Create First Admin User:** Enter a username, password, full name, and email. **Configure Instance URL:** Confirm or change the Jenkins URL (e.g., http://localhost:8080). **Save and Finish:** Click “Start using Jenkins” to go to the Jenkins Dashboard.

**Experiment 6: Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests  
Objective:>** Create a **Jenkins job** to automatically build a project (Maven or Gradle). **>** Integrate **source code management (SCM)** like Git. **>** Run **automated builds and tests**. **>** View build results and test reports. **Part A: Set Up a CI Pipeline (Freestyle Project)  
Step 1: Create a New Jenkins Job**Open Jenkins in your browser:<http://localhost:8080>Click on **“New Item”** on the left side of the dashboard.Enter a name for the project, e.g., Maven-CI.Select **“Freestyle project”**.Click **OK**. **Step 2: Configure Source Code Management (SCM)**In the configuration page, scroll to **“Source Code Management”**.Select **“Git”**.Enter your repository URL, for example:<https://github.com/yourusername/your-maven-project.git>If the repo is private:Click **“Add”** next to Credentials.**> E**nter your GitHub username and token/password.In **Branch Specifier**, enter:\*/main **Step 3: Add Build Step**Depending on whether you’re using **Maven** or **Gradle**: **For Maven Project:**Scroll to the **Build** section.Click **“Add build step”** → Select **“Invoke top-level Maven targets”**.In **Goals**, enter:clean packageLeave the POM file field blank if it is in the root directory (pom.xml).  
**For Gradle Project:**Install **Gradle Plugin** in Jenkins (Manage Jenkins → Manage Plugins).After plugin is installed:Click **“Add build step”** → Choose **“Invoke Gradle script”**.In **Tasks**, enter:clean buildIn **Switches**, you may optionally add:--info –stacktrace **Step 4: Configure Post-build Actions  
Publish Test Results:** Scroll to **Post-build Actions**.Click **“Add post-build action”** → Choose **“Publish JUnit test result report”**.In the **Test Report XMLs** field:For Maven:\*\*/target/surefire-reports/\*.xmlFor Gradle:\*\*/build/test-results/test/\*.xml **Archive Artifacts (Optional):** Add another post-build action: **Archive the artifacts**.Enter:target/\*.jaror for Gradle:build/libs/\*.jar **Step 5: Save and Build**Click **Save**.On the project page, click **“Build Now”**.View the build progress and console output:Click on the build number in the **Build History**.Select **“Console Output”**. **Part B: Set Up CI Using Pipeline (Pipeline as Code)**  
**Step 1: Create a Pipeline Job**  
On the Jenkins dashboard, click **“New Item”**.  
Enter a name, e.g., Pipeline-CI.  
Choose **“Pipeline”** and click **OK**.  
**Step 2: Define the Pipeline Script**  
**Option 1: Enter Script Directly**  
Scroll to **Pipeline** section.  
Select **“Pipeline script”**.

Paste a pipeline script, e.g., for Maven:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git 'https://github.com/yourusername/your-maven-project.git'

}

}

stage('Build') {

steps {

sh 'mvn clean package'

}

}

stage('Test') {

steps {

junit '\*\*/target/surefire-reports/\*.xml'

}

}

stage('Archive') {

steps {

archiveArtifacts artifacts: 'target/\*.jar', fingerprint: true

}

}

}

}

**Option 2: Load from SCM**Instead of writing the script directly, choose **“Pipeline script from SCM”**.Select **Git**, and enter your repository URL.Make sure the repo contains a file named Jenkinsfile at the root with the pipeline code above. **Step 3: Run and Verify**Click **“Build Now”**.Watch the pipeline graph update.Click each stage to see logs.Confirm build/test success and artifact archiving.

**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  
Objective:>** Understand what **Ansible** is. **>** Learn about **inventory**, **playbooks**, and **modules**.**>** Automate simple server configuration tasks using a playbook. **>** Run Ansible on an Ubuntu system using the **localhost**. **What is Ansible? Ansible** is an **open-source IT automation tool** used for: **Configuration Management** – keeping server settings consistent. **Application Deployment** – installing and updating software. **Orchestration** – coordinating multiple systems and services.It uses **SSH**, so no agent is required on the target machines (agentless).

**Key Concepts in Ansible: 1. Inventory:** An inventory is a file (in INI or YAML format) listing the target hosts or groups of hosts you want to manage.Example:[local]localhost ansible\_connection=local **2. Playbook:** A YAML file that contains **plays**, and each play has **tasks** that run on specific hosts using **modules**. **3. Modules:** Reusable Ansible components that perform specific tasks like installing packages (apt), copying files (copy), starting services (service), etc.

**Why Use Ansible?  
Agentless:** Uses SSH; no need to install software on client machines. **Simple:** Written in YAML, human-readable. **Idempotent:** Running the same playbook multiple times doesn’t cause repeated changes. **Scalable:** Works on a single server or 1000+ nodes.

**Step-by-Step Guide (on Ubuntu)**

**Step 1: Install Ansible**Update your system:sudo apt updatesudo apt upgrade -yInstall Ansible:sudo apt install ansible -yCheck version:ansible –version **Step 2: Create an Ansible Inventory File:** An inventory defines the machines Ansible should manage.Create the file:nano hosts.iniPaste the following:[local]localhost ansible\_connection=local **Step 3: Write a Basic Ansible Playbook:** A playbook is a YAML file that performs tasks.Create the file:nano setup.ymlPaste the following content:---- name: Basic Server Setup hosts: localbecome: yes

tasks:

- name: Update apt cache

apt:

update\_cache: yes

- name: Install curl

apt:

name: curl

state: present  
**Step 4: Run the Ansible Playbook**Execute the playbook using: ansible-playbook -i hosts.ini setup.yml

**Experiment 8: Set Up a Jenkins CI Pipeline for a Maven Project and Deploy Artifacts Using Ansible  
Objective: >** Set up Jenkins to automatically build a **Maven project** from source control. **> Archive the JAR artifact** produced by the build. **>** Use **Ansible** to deploy the artifact via Jenkins (post-build step). **>** Verify successful deployment.This combines **CI (Continuous Integration)** with **automated configuration management**, illustrating end-to-end DevOps workflow.

**Prerequisites:** Before starting:Jenkins is installed and running.You have a Maven project in a Git repo (like the HelloMaven from earlier).Ansible is installed and working.You’ve created a basic inventory (hosts.ini) for Ansible. **Step 1: Prepare the Maven Project**If your HelloMaven project is not yet in Git, do the following:cd ~/path/to/HelloMavengit initgit add .git commit -m "Initial commit of HelloMaven project"Push it to GitHub (replace URL with your own):git remote add origin <https://github.com/yourusername/HelloMaven.git>git branch -M maingit push -u origin main **Step 2: Configure Jenkins Job  
A. Create a New Jenkins Job**Open Jenkins (http://localhost:8080)Click **“New Item”**Enter job name: HelloMaven-CISelect **“Freestyle project”**Click **OK  
B. Configure Source Code Management (Git)**Scroll to **Source Code Management**Choose **Git**Enter repository URL:<https://github.com/yourusername/HelloMaven.git>If private, click **Add Credentials** and supply your GitHub username/token.Branch specifier:\*/main **C. Add Maven Build Step**Scroll to **Build** → Click **“Add build step”** → Select **“Invoke top-level Maven targets”**In **Goals**, enter:compile test package **Step 3: Archive the Artifact**Scroll to **Post-build Actions**Click **Add post-build action** → **Archive the artifacts**In **Files to archive**, enter:target/\*.jar **Step 4: Integrate Ansible Deployment in Jenkins**You’ll now tell Jenkins to run an Ansible playbook after the build. **A. Add Another Post-build Action**Scroll down again to **Post-build Actions**Click **Add post-build action** → Select **“Execute shell”  
B. Write the Shell Command**Paste the following into the shell box:ansible-playbook -i /path/to/hosts.ini /path/to/deploy.yml **C. Create hosts.ini File**Example content:[local]localhost ansible\_connection=localSave this as hosts.ini. **D. Create deploy.yml Ansible Playbook**Create a playbook that copies the JAR to a desired location.---- name: Deploy Maven Artifact

hosts: local

become: yes

tasks:

- name: Copy the artifact to the deployment directory

copy:

src: "/var/lib/jenkins/workspace/HelloMaven-CI/target/HelloMaven-1.0-SNAPSHOT.jar"

dest: "/opt/deployment/HelloMaven.jar"  
**Step 5: Run and Verify**  
Save the Jenkins job configuration.  
On the job dashboard, click **“Build Now”**.  
Watch build progress under **Build History** → Click build → **Console Output**.  
Look for: BUILD SUCCESS  
and output from Ansible:  
TASK [Copy the artifact to the deployment directory]  
changed: [localhost]  
Confirm that the JAR was copied:  
ls /opt/deployment/HelloMaven.jar