LATHA MATHAVAN ENGINEERING COLLEGE

KIDARIPATTI,ALAGARKOIL,MADURAI-625301.

CCS342 - Devops Laboratory

Regulation : 2021

Branch : B.E. – CSE

Year :III - YEAR

Semester :VI- Semester

LATHA MATHAVAN ENGINEERING COLLEGE

KIDARIPATTI, ALAGARKOIL, MELUR TALUK, MADURAI - 625301

Department of _____ _____Engineering Laboratory Record

NAME: _____CLASS_____

REGISTER NO: _____

Certified that this is bona fide record of work done by the above student of the **CCS342-Devops Lab** during the year _____

Signature of Lab-in-Charge

Signature of Head of the Department

Submitted for the Practical Examination Held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

EXP.NO	DATE	CONTENTS	PAGE NO	MARKS AWARDED	STAFF INITIAL

EXP.NO	DATE	CONTENTS	PAGE NO	MARKS AWARDED	STAFF INITIAL

EXP NO: **DATE: ___/___/_____**

Create Maven Build Pipeline In Azure

AIM:

To set up a Maven build pipeline in Azure DevOps to automate the build process for a Maven project.

PROCEDURE:

1. **Create a New Project**
    a. **In Azure DevOps:**
        i. Navigate to Azure DevOps.
        ii. Click on "New Project" and fill in the required details.
        iii. Click "Create" to initialize the project.
    b. **In Jenkins (for comparison):**
        i. Open Jenkins.
        ii. Click on "New Item".
        iii. Select "Freestyle project" and provide a name like "Maven-Project".
        iv. Click "OK" to create the project.
2. **Create a New Pipeline**
    a. **In Azure DevOps:**
        i. Navigate to the project which created.
        ii. Click on "Pipelines" in the left sidebar.
        iii. Select "New pipeline" to initiate pipeline creation.
    b. **In Jenkins:**
        i. Inside the created project, scroll to the "Build" section.
        ii. Choose "Add build step".
        iii. Select "Invoke top-level Maven targets".

3. **Select a Repository**
    a. **In Azure DevOps:**
        i. Choose the source where the Maven project repository is stored (Azure Repos Git, GitHub, or another Git service).
        ii. Authenticate and select the repository containing the Maven project.
    b. **In Jenkins:**
        i. Scroll to the Source Code Management section and select Git.
        ii. Enter the repository URL:

`https://github.com/MADHAVAN-BE-2003/Maven_Project_1.git`

        iii. Change Branch Specifier:

`master ---> main`

4. **Define the Pipeline Configuration**
    a. **In Azure DevOps (YAML):**
        i. Azure DevOps will guide through setting up a YAML file or need to create one manually. Below is a basic configuration for a Maven build pipeline:

```yaml
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    goals: 'clean package'
    options: '-X'
  displayName: 'Run Maven Package'
```

    b. **In Jenkins (Manual Configuration):**
        i. In the "Build" section, under "Goals", enter the following Maven goals:

`clean package -X`

        ii. Ensure the Advanced, "POM" file is set to `pom.xml`.
        iii. Save the project.

5. **Run the Pipeline**
    a. **In Azure DevOps:**
        i. Once the pipeline is configured, commit the azure-pipelines.yml file to the repository.
        ii. Azure DevOps will detect the YAML file and trigger the pipeline automatically based on the branch triggers, or it can manually run it by clicking "Run Pipeline".
    b. **In Jenkins:**
        i. Go back to the Jenkins project dashboard.
        ii. Click "Build Now" to start the build process.
6. **Monitor the Pipeline**
    a. **In Azure DevOps:**
        i. Go to the "Pipelines" section to monitor the build and view logs, check progress, and see the final results.
    b. **In Jenkins:**
        i. Go to the "Build History" section of the Jenkins project.
        ii. Click on the build number to view logs and monitor progress.

CONSOLE OUTPUT:

```
[Maven-Project] $ cmd.exe /C "mvn -f pom.xml clean package -X && exit %%ERRORLEVEL%%"
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\ProgramFiles\Maven\apache-maven-3.9.9
Java version: 17.0.11, vendor: Oracle Corporation, runtime: D:\Program files\Java\jdk-17
Default locale: en_IN, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- clean:3.1.0:clean (default-clean) @ crud-app ---
[INFO] Deleting target directory and files...
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] Changes detected - recompiling the module!
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] Changes detected - recompiling the module!
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[DEBUG] Determined Maven Process ID 15608
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.045 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- jar:3.0.2:jar (default-jar) @ crud-app ---
[INFO] Building jar: C:\ProgramData\Jenkins\.jenkins\workspace\Maven-Project\target\crud-app-1.0-
SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.912 s
[INFO] Finished at: 2024-11-11T21:14:02+05:30
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

RESULT:

Thus, The Maven build pipeline will be successfully created in Azure DevOps, automating the Maven build process. The steps outlined can also be replicated in Jenkins for a similar pipeline setup.

EXP NO: **DATE: ___/___/_____**

Run Regression Tests Using Maven Build Pipeline In Azure

AIM:

To extend the existing Maven build pipeline to include steps for running regression tests and reporting results.

PROCEDURE:

1. **Update the `azure-pipelines.yml` File**
   a. **In Azure DevOps:**
      i. To add regression tests to the Maven pipeline, modify the YAML configuration to include steps that run tests and publish the results. Here's the updated `azure-pipelines.yml` file:

```yaml
# azure-pipelines.yml
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    goals: 'clean verify'  # Use 'verify' to include tests
    options: '-X'
  displayName: 'Run Maven Tests'

- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/target/surefire-reports/*.xml'
    testRunTitle: 'Maven Test Results'
  condition: succeededOrFailed()
```

   b. **In Jenkins (for comparison):**
      i. In the Jenkins project configuration, scroll to the "Build" section.
      ii. Add another build step: select "Invoke top-level Maven targets".
      iii. In the "Goals" field, enter:

```
clean verify
```

2. **Explanation of YAML Configuration**
    - **trigger:** Specifies the branch that will trigger the pipeline.
    - **pool:** Defines the agent pool; ubuntu-latest is commonly used.
    - **steps:** Lists the steps of the pipeline.

1st Step: Running Maven Tests

a. **In Azure DevOps:**
    i. **task**: Maven@3: Uses the Maven task provided by Azure DevOps.
    ii. **goals**: clean verify: Cleans up previous builds and runs tests (verify includes both build and test phases).
    iii. **options**: -X: Provides detailed Maven output.

b. **In Jenkins:**
    i. The same goal (clean verify) is specified to execute Maven tests during the build process.

2nd Step: Publishing Test Results

a. **In Azure DevOps:**
    i. **task**: PublishTestResults@2: Publishes the results of the tests in Azure DevOps.
    ii. **testResultsFiles**: Specifies the pattern to locate test result files, typically found in target/surefire-reports.
    iii. **testRunTitle**: A title for the test run.
    iv. **condition**: succeededOrFailed(): Publishes results even if some tests fail.

b. **In Jenkins:**
    i. Add post-build action by selecting "Publish JUnit test result report".
    ii. In the "Test report XMLs" field, enter the path to the test reports, usually:

```
**/target/surefire-reports/*.xml
```

3. **Ensure Proper Configuration for Test Reports**
    a. **In Azure DevOps:**
        i. Ensure the Maven project is configured to produce test reports in a compatible format. The Maven Surefire Plugin generates XML reports that Azure DevOps can consume. The pom.xml should contain the following configuration:

```
<build>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-surefire-plugin</artifactId>
         <version>2.22.2</version>
         <!-- Ensure this version or newer -->
         <configuration>
            <includes>
               <include>**/*Test.java</include>
            </includes>
         </configuration>
      </plugin>
   </plugins>
</build>
```

    b. **In Jenkins:**
        i. Ensure the Maven Surefire Plugin is configured similarly to generate reports compatible with Jenkins.
4. **Commit Changes and Trigger Pipeline**
    a. **In Azure DevOps (YAML):**
        i. Save and commit the changes to azure-pipelines.yml in the repository.
        ii. Azure DevOps will automatically trigger the pipeline if configured to do so, or it can be manually run from the Azure DevOps UI.
    b. **In Jenkins (Manual Configuration):**
        i. Save the configuration changes in the Jenkins job and click "Build Now" to run the job.
        ii. Ensure that the Jenkins job is set to pull from the repository containing the updated configuration.

5. **Monitor Test Results**
    a. **In Azure DevOps:**
        i. After the pipeline runs, navigate to the "Pipelines" section.
        ii. Click on the pipeline run to view detailed logs and test results.
        iii. Go to the "Tests" tab to see the results of the regression tests, including any failures or errors.
    b. **In Jenkins:**
        i. Click on the build number in the "Build History" section to view the console output.
        ii. Check the "Test Results" tab for a summary of the regression tests, including any failures.

CONSOLE OUTPUT:

```
[Maven-Project] $ cmd.exe /C "mvn -f pom.xml clean package -X && exit %%ERRORLEVEL%%"
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\ProgramFiles\Maven\apache-maven-3.9.9
Java version: 17.0.11, vendor: Oracle Corporation, runtime: D:\Program files\Java\jdk-17
Default locale: en_IN, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- clean:3.1.0:clean (default-clean) @ crud-app ---
[INFO] Deleting target directory and files...
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[DEBUG] Determined Maven Process ID 19676
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.033 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- jar:3.0.2:jar (default-jar) @ crud-app ---
[INFO] Building jar: C:\ProgramData\Jenkins\.jenkins\workspace\Maven-Project\target\crud-app-1.0-
SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.502 s
[INFO] Finished at: 2024-11-11T22:39:17+05:30
[INFO] ------------------------------------------------------------------------

[Maven-Project] $ cmd.exe /C "mvn clean verify && exit %%ERRORLEVEL%%"
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- clean:3.1.0:clean (default-clean) @ crud-app ---
[INFO] Deleting C:\ProgramData\Jenkins\.jenkins\workspace\Maven-Project\target
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\ProgramData\Jenkins\.jenkins\workspace\Maven-
Project\src\main\resources
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\ProgramData\Jenkins\.jenkins\workspace\Maven-
Project\target\classes
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] Changes detected - recompiling the module!
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.028 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- jar:3.0.2:jar (default-jar) @ crud-app ---
[INFO] Building jar: C:\ProgramData\Jenkins\.jenkins\workspace\Maven-Project\target\crud-app-1.0-
SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.595 s
[INFO] Finished at: 2024-11-11T22:39:23+05:30
[INFO] ------------------------------------------------------------------------
Recording test results
[Checks API] No suitable checks publisher found.
Finished: SUCCESS
```

RESULT:

Thus, The Maven pipeline is successfully extended to run regression tests and publish results in Azure DevOps. This setup can also be mirrored in Jenkins, ensuring that the build process incorporates automated testing, which is crucial for maintaining code quality.

EXP NO: **DATE: ___/___/_____**

Install Jenkins In Cloud And Local Windows Environment

AIM:

To install Jenkins in various cloud environments (AWS, Azure, Google Cloud) and on a local Windows machine to facilitate Continuous Integration (CI) processes.

PROCEDURE:

Install Jenkins on AWS (Amazon Web Services)

1. **Create an EC2 Instance**
    a. **Log in to AWS Management Console:** Navigate to the AWS Management Console.
    b. **Launch a New EC2 Instance**:
        i. Go to the EC2 Dashboard.
        ii. Click on "Launch Instance".
        iii. Choose an Amazon Machine Image (AMI), such as Ubuntu Server.
        iv. Select Instance Type (e.g., t2.micro for small workloads).
        v. Configure Instance Details, Add Storage, and Tags as needed.
        vi. **Configure Security Group**:
            1. Add rules to allow inbound traffic on ports 22 (SSH) and 8080 (Jenkins default port).
    c. **Review and Launch:**
        i. Review settings and click "Launch".
        ii. Create or select a key pair for SSH access and click "Launch Instances".
2. **Connect to the Instance**
    a. Once the instance is running, connect using SSH.
3. **Install Jenkins**
    a. **Update package list and install Java**:

```
sudo apt update
sudo apt install openjdk-11-jdk
```

b. **Add Jenkins repository and install Jenkins:**

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary' >
/etc/apt/sources.list.d/jenkins.list
sudo apt update
sudo apt install jenkins
```

c. **Start Jenkins and enable on boot:**

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

4. **Access Jenkins**
   a. Open a web browser and go to http://<your-ec2-public-ip>:8080.
   b. Retrieve the Jenkins unlock key:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

c. Enter this key in the Jenkins setup wizard to complete the installation.

Install Jenkins on Azure

1. **Create a Virtual Machine**
   a. **Log in to Azure Portal:** Navigate to the Azure Portal.
   b. **Create a New Virtual Machine**:
      i. Click on "Create a resource" and select "Virtual Machine".
      ii. Choose an image like "Ubuntu Server".
      iii. Select an instance size and configure the admin username and SSH public key.
      iv. **Configure Networking**:
         1. Open ports 22 (SSH) and 8080 (Jenkins).
   c. **Review and Create**: Review the configuration and click "Create".
2. **Connect to the VM**
   a. Once the VM is deployed, connect via SSH.
3. **Install Jenkins**
   a. **Update package list and install Java**:

```
sudo apt update
sudo apt install openjdk-11-jdk
```

b. **Add Jenkins repository and install Jenkins:**

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary' >
/etc/apt/sources.list.d/jenkins.list
sudo apt update
sudo apt install jenkins
```

c. **Start Jenkins and enable on boot:**

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

5. **Access Jenkins**
   a. Open a web browser and go to http://<your-ec2-public-ip>:8080.
   b. Retrieve the Jenkins unlock key:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

c. Enter this key in the Jenkins setup wizard to complete the installation.

Install Jenkins on Google Cloud Platform (GCP)

1. **Create a Compute Engine Instance**
   a. **Log in to Google Cloud Console:** Navigate to the Google Cloud Console.
   b. **Create a New VM Instance:**
      i. Go to "Compute Engine" > "VM instances".
      ii. Click "Create Instance".
      iii. Choose a machine type and select an OS like "Ubuntu".
   c. **Configure Network Settings:**
      i. Allow HTTP traffic and specify firewall rules for ports 22 and 8080.
   d. **Create and Connect**: Create the instance and connect via SSH.
2. **Install Jenkins**
   a. **Update package list and install Java**:

```
sudo apt update
sudo apt install openjdk-11-jdk
```

b. **Add Jenkins repository and install Jenkins:**

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary' >
/etc/apt/sources.list.d/jenkins.list
sudo apt update
sudo apt install jenkins
```

c. **Start Jenkins and enable on boot:**

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

6. **Access Jenkins**
   a. Open a web browser and go to http://<your-ec2-public-ip>:8080.
   b. Retrieve the Jenkins unlock key:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

c. Complete the setup wizard with this key.

Install Jenkins on Windows Local Machine

1. **Install Java**
   a. **Download Java JDK:** Go to the Oracle website and download the JDK installer.
   b. **Install Java:** Run the installer and follow the installation instructions.
   c. **Set JAVA_HOME Environment Variable:**
      i. Open Control Panel > System and Security > System > Advanced system settings > Environment Variables.
      ii. Under "System variables", click "New" and add JAVA_HOME with the path to your JDK installation (e.g., C:\Program Files\Java\jdk-11.0.x).
2. **Install Jenkins**
   a. **Download Jenkins:** Go to the Jenkins website and download the Windows installer.
   b. **Run the Installer:** Follow the installation instructions to install Jenkins as a Windows service.
   c. **Start Jenkins:** Jenkins should start automatically as a service; if not, start it via the Services management console.

3. **Access Jenkins**
    a. Open a web browser and go to http://localhost:8080.
    b. Retrieve the Jenkins unlock key from the specified path (default: C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword) and enter it in the setup wizard.

RESULT:

Thus, Jenkins was successfully installed across AWS, Azure, Google Cloud, and a local Windows environment, enabling automated building, testing, and deployment of code, which enhances the software development lifecycle's efficiency and reliability.

EXP NO: **DATE: ___/___/_____**

Create CI Pipeline Using Jenkins

AIM:

To set up a Continuous Integration (CI) pipeline in Jenkins to automate the build process for a Maven project.

PREREQUISITES:

- **Jenkins Installed:** Ensure Jenkins is installed and accessible.
- **Jenkins Plugins:** Install necessary plugins (e.g., Git, Maven).
- **Source Code Repository:** Code should be stored in a version control system like GitHub, GitLab, or Bitbucket. Here:

  `https://github.com/MADHAVAN-BE-2003/Maven_Project_Test.git`

PROCEDURE:

1. **Set Up Jenkins**
   a. **Log in to Jenkins**: Open your Jenkins instance in a web browser

      `http://localhost:8080/`

   b. **Install Required Plugins**:
      i. Navigate to Manage Jenkins > Manage Plugins.
      ii. Under the Available tab, search for and install plugins such as:
         1. **Git Plugin** (default installed)
         2. **Pipeline** (default installed)
         3. **Maven Integration**
      iii. Restart Jenkins if required.
   c. Check Maven Path:
      i. Go to Manage Jenkins > Global Tool Configuration.
      ii. Ensure that the Maven section is configured with Maven installation.

2. **Create a New Pipeline Job**
    a. **Create a New Item**:
        i. Click "New Item" on the Jenkins dashboard.
        ii. Enter a name for your pipeline (e.g., My-CI-Pipeline) and select Pipeline. Click OK.
    b. **Configure the Pipeline**:
        i. General Tab: Optionally, add a description.
        ii. Pipeline Tab:
            1. Choose Pipeline script from SCM if you are using the Jenkinsfile stored in the repository.
3. **Configure Source Code Management (SCM)**

    If using a Jenkinsfile stored in SCM:
    a. **Pipeline Tab**:
        i. Select Pipeline script from SCM.
        ii. Choose Git (or your SCM type) and enter the repository URL:

```
https://github.com/MADHAVAN-BE-2003/Maven_Project_Test.git
```

        iii. Change the Branch Specifier, from master to main.
        iv. Specify the Script Path (usually Jenkinsfile).
4. **Save and Run the Pipeline**
    a. Click "Save" to create the pipeline.
    b. Click "Build Now" to run the pipeline manually or set up triggers to run automatically.
5. **Monitor Pipeline Execution**
    a. Go to the pipeline job page to monitor the build process, view logs, test results, and build artifacts.

CONSOLE OUTPUT:

```
Started by user Madhavan V
Obtained Jenkinsfile from git https://github.com/MADHAVAN-BE-2003/Maven_Project_Test.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline

C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline>mvn clean install
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- clean:3.1.0:clean (default-clean) @ crud-app ---
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.042 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- jar:3.0.2:jar (default-jar) @ crud-app ---
[INFO] Building jar: C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline\target\crud-app-1.0-
SNAPSHOT.jar
[INFO] --- install:2.5.2:install (default-install) @ crud-app ---
[INFO] Installing C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline\target\crud-app-1.0-
SNAPSHOT.jar to C:\WINDOWS\system32\config\systemprofile\.m2\repository\com\example\crudapp\crud-
app\1.0-SNAPSHOT\crud-app-1.0-SNAPSHOT.jar
[INFO] Installing C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline\pom.xml to
C:\WINDOWS\system32\config\systemprofile\.m2\repository\com\example\crudapp\crud-app\1.0-
SNAPSHOT\crud-app-1.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.775 s
[INFO] Finished at: 2024-11-12T03:03:08+05:30
[INFO] ------------------------------------------------------------------------
[Pipeline] echo
Build completed successfully.

C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline>mvn test
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.059 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.153 s
[INFO] Finished at: 2024-11-12T03:03:14+05:30
[INFO] ------------------------------------------------------------------------
[Pipeline] echo
Tests executed successfully.
[Pipeline] echo
Cleaning up workspace post-build.
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\My-CI-Pipeline@2
Build and test stages completed successfully.
[Pipeline] End of Pipeline
Finished: SUCCESS
```

RESULT:

Thus, the CI pipeline was successfully established in Jenkins, automating the Maven build process and enabling efficient management of code integration and deployment within the software development lifecycle.

EXP NO: **DATE: ___/___/_____**

Create A CD Pipeline In Jenkins And Deploy In Cloud

AIM:

To create a Continuous Deployment (CD) pipeline in Jenkins that automates building, testing, and deploying code to a cloud environment.

PROCEDURE:

1. **Set Up Jenkins**
    a. Ensure Jenkins is running with required plugins for cloud deployment. Add plugins based on the cloud environment:
        i. **AWS Steps Plugin** (for AWS deployment)
        ii. **Azure CLI Plugin** (for Azure deployment)
        iii. **Google Cloud Plugin** (for Google Cloud deployment)
    b. **For Local Setup:**
        i. Download Jenkins from the official Jenkins site.
        ii. Follow your operating system's installation steps, then start Jenkins.
        iii. Access Jenkins via `http://localhost:8080`.

2. **Create or Update the Jenkins Pipeline**
    a. **Jenkinsfile for Continuous Deployment**
        i. Ensure the Jenkinsfile for your Maven project is configured with the necessary stages for building, testing, and deploying.
        ii. **Local Setup Adjustments:**
        To run the pipeline on a local machine, adjust the deployment stage to copy the artifact to a specified directory.

b. **Explanation of Jenkinsfile**

   i. **environment**: Sets environment variables (e.g., AWS region and S3 bucket).

   ii. **stages**:

      1. **Prepare Workspace:** Cleans up the workspace before the pipeline execution.
      2. **Checkout Code:** Retrieves the latest code from the specified Git repository.
      3. **Build:** Compiles the project.
      4. **Test:** Executes tests on the project.
      5. **Deploy:** Moves the built artifact to a designated directory.

3. **Configure Deployment**

   a. **AWS Deployment (S3 Example)**

      i. **Install AWS CLI**:

         1. Install and configure AWS CLI on your Jenkins server or local machine.

```
aws configure
```

      ii. **IAM Permissions**:

         1. Ensure IAM permissions allow S3:PutObject for the bucket.

   b. **Azure Deployment**

   For Azure, use the Azure CLI within Jenkins for deployment.

      i. **Install Azure CLI**:

         1. Install and authenticate using:

```
az login
```

      ii. **Deploy Using Azure CLI**:

         1. Set up deployment commands according to your application requirements.

c. **Google Cloud Deployment**

For GCP, use the Google Cloud SDK.

i. **Install Google Cloud SDK**:

1. Install and authenticate with:

```
gcloud auth login
```

ii. **Deploy with Google SDK**:

1. Use gcloud commands to manage your application deployment.

d. **Local Setup**

For local deployment, skip cloud-specific configurations and ensure that the deployment file is saved to a local directory or accessible service path.

4. **Configure Jenkins Job**

a. **Create a Pipeline Job**:

i. In Jenkins, navigate to New Item > Pipeline.

ii. Enter a name for your job (e.g., Maven-CD-Pipeline).

iii. If using a Jenkinsfile stored in version control, select Pipeline script from SCM and provide the following details:

1. **SCM**: Git
2. **Repository URL**:

```
https://github.com/MADHAVAN-BE-2003/Maven_Project_Deploy.git
```

3. **Branch Specifier**: */main

iv. If entering the Jenkinsfile directly, paste the script into the provided field.

b. **Run the Pipeline**:

i. Save the configuration by clicking Save.

ii. Click on Build Now in the left menu to trigger the pipeline execution.

5. **Monitor and Validate**

a. **Monitor Pipeline Execution**:

i. Click on the job name in Jenkins, then select the build number to view the console output.

ii. Ensure each stage completes without errors. Pay attention to build logs for any warnings or errors that may need addressing.

b. Deployment Validation:

i. Verify that the application has been deployed to the cloud environment by checking:

1. AWS S3: Log in to your AWS Management Console and navigate to S3. Check if the artifact exists in the specified bucket.

2. Azure: Use the Azure Portal to confirm deployment in the App Service or Resource Group.

3. Google Cloud: Check Google Cloud Console for deployment in the appropriate service.

ii. For local deployment, navigate to the specified local directory (e.g., D:\Deployed) to confirm that the artifact (crud-app-1.0-SNAPSHOT.jar) has been successfully copied.

CONSOLE OUTPUT:

```
Started by user Madhavan V
Obtained Jenkinsfile from git https://github.com/MADHAVAN-BE-2003/Maven_Project_Deploy.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline

[Pipeline] checkout
[Pipeline] timeout
Timeout set to expire in 1 hr 0 min
[Pipeline] echo
Workspace cleaned successfully.

[Pipeline] checkout
[Pipeline] echo
Code checkout successful.

C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline>mvn clean install
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- clean:3.1.0:clean (default-clean) @ crud-app ---
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.049 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- jar:3.0.2:jar (default-jar) @ crud-app ---
[INFO] --- install:2.5.2:install (default-install) @ crud-app ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.797 s
[INFO] Finished at: 2024-11-12T03:17:47+05:30
[INFO] ------------------------------------------------------------------------
[Pipeline] echo
Build completed successfully.

C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline>mvn test
[INFO] Scanning for projects...
[INFO] --------------------< com.example.crudapp:crud-app >--------------------
[INFO] Building crud-app 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] --- resources:3.0.2:resources (default-resources) @ crud-app ---
[INFO] --- compiler:3.8.1:compile (default-compile) @ crud-app ---
[INFO] --- resources:3.0.2:testResources (default-testResources) @ crud-app ---
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ crud-app ---
[INFO] --- surefire:2.22.1:test (default-test) @ crud-app ---
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.crudapp.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.045 s - in
com.example.crudapp.AppTest
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.011 s
[INFO] Finished at: 2024-11-12T03:17:52+05:30
[INFO] ------------------------------------------------------------------------
[Pipeline] echo
Tests executed successfully.
```

```
C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline>if not exist "D:\Deployed" mkdir
"D:\Deployed"
C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline>copy
"C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline\target\crud-app-1.0-SNAPSHOT.jar"
"D:\Deployed"
        1 file(s) copied.
[Pipeline] echo
Deployment completed successfully.
[Pipeline] echo
Cleaning up workspace post-build.
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CD-Pipeline@2
[Pipeline] echo
Build, test, and deploy stages completed successfully.
[Pipeline] End of Pipeline
Finished: SUCCESS
```

RESULT:

Thus, The configured CD pipeline in Jenkins now automatically builds, tests, and deploys the project to the target environment. Local machine adjustments enable consistent processes across cloud and local setups, verifying artifact presence in the defined directory or cloud storage.

EXP NO: **DATE: ___/___/_____**

Create Ansible Playbook for Simple Web Application Infrastructure

AIM:

To create an Ansible playbook for setting up a basic web application infrastructure using Nginx and deploying a simple HTML web page.

PROCEDURE:

1. **Install Ansible on Kali Linux WSL**
    a. **Ensure your environment is updated and Ansible is installed:**

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

2. **Create the Playbook File**
    a. **Open a text editor to create the playbook:**

```
nano webapp.yml
```

b. Copy the following playbook content into the file:

```
---
- name: Set up web application infrastructure
  hosts: webservers
  become: yes
  tasks:
    - name: Ensure Nginx is installed
      apt:
        name: nginx
        state: present
      notify:
        - restart nginx

    - name: Start Nginx service
      service:
        name: nginx
        state: started
        enabled: yes
    - name: Deploy the web application HTML file
      copy:
        dest: /var/www/html/index.html
        content: |
          <!DOCTYPE html>
          <html>
          <head>
              <title>Welcome to My Web App</title>
          </head>
          <body>
              <h1>Hello, world!</h1>
              <p>This is a simple web application.</p>
          </body>
          </html>
        owner: www-data
        group: www-data
        mode: '0644'
  handlers:
    - name: restart nginx
      service:
        name: nginx
        state: restarted
```

c. Save and exit by pressing CTRL + O and Enter to save and CTRL + X to exit.

3. Create the Inventory File

a. Create an inventory file to specify target hosts:

```
nano hosts
```

b. Add the following configuration to target `localhost`:

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

c. Save and exit.

4. **Run the Ansible Playbook**
    a. **Execute the playbook using the Ansible command:**

       ```
       ansible-playbook -i hosts webapp.yml
       ```

    b. **This command uses the -i flag to specify the inventory file (hosts) and runs the playbook (webapp.yml).**
5. **Verify the Setup**
    a. **After running the playbook, verify the web server setup by opening a browser and navigating to:**

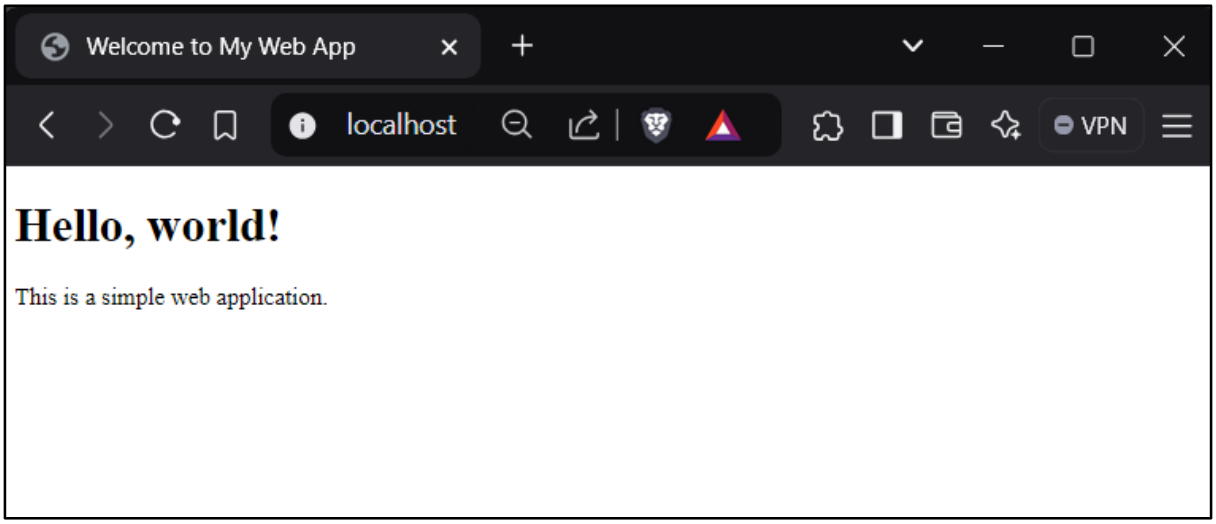
       ```
       http://localhost
       ```

    b. **You should see a web page displaying "Hello, world! This is a simple web application."**

CONSOLE OUTPUT:

```
┌──(root㉿BLACK-EVIL)-[/home/blackevil]
└─# ansible-playbook -i hosts webapp.yml

PLAY [Set up web application infrastructure]
**********************************

TASK [Gathering Facts]
*********************************************************
[WARNING]: Platform linux on host localhost is using the discovered Python
interpreter at /usr/bin/python3.11, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter
_discovery.html for more information.
ok: [localhost]

TASK [Ensure Nginx is installed]
*********************************************
ok: [localhost]

TASK [Start Nginx service]
*****************************************************
ok: [localhost]

TASK [Deploy the web application HTML file]
***********************************
ok: [localhost]

PLAY RECAP
*********************************************************************
localhost      : ok=4    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

BROWSER OUTPUT:



RESULT:

Thus, the Ansible playbook successfully creates a simple web application infrastructure using Nginx, automating the deployment of a basic HTML web page. This setup can be easily modified and expanded for more complex web applications.

EXP NO: **DATE: ___/___/_____**

Build a Simple Application using Gradle

AIM:

To set up a Gradle build pipeline for a simple Java application in Windows, automating the build and run process using Gradle.

PROCEDURE:

1. **Create a New Directory for the Project**
    a. **In Windows:**
        i. Open Command Prompt or PowerShell.
        ii. Navigate to the desired directory where you want to create your project (e.g., D:\Tem\DevOps\Gradle).
        iii. Run the following commands:

```
mkdir hello-world
cd hello-world
```

2. **Initialize the Gradle Project**
    a. **In Windows:**
        i. In the hello-world directory, run:

```
gradle init
```

        ii. Follow the prompts:
            1. Select project type: 1. Application
            2. Select language: 1. Java
            3. Java version: 17 (check the installed java version)
            4. Select application structure: 1. Single application project
            5. Select build script DSL: 2. Groovy
            6. Select test framework: 1. JUnit 4
            7. Choose to use new APIs and behaviour: yes

    This initializes the project with a basic structure and generates the necessary files.

3. **Edit the build.gradle File**
    a. **In Windows:**
        i. Open build.gradle in a text editor (e.g., Notepad, VSCode).
        ii. Replace its content with the following:

```
plugins {
    id 'java'
    id 'application'
}

group 'org.example'
version '1.0-SNAPSHOT'

sourceCompatibility = '17'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.13.2'
}

application {
    mainClassName = 'org.example.App'
}
```

4. **Write the Java Code**
    a. **In Windows:**
        i. Navigate to src/main/java/org/example/ and create a new file named App.java.
        ii. Add the following Java code to the file:

```
package org.example;

public class App {
    public String getGreeting() {
        return "Hello World!";
    }

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

5. Build the Project
    a. **In Windows:**
        i. Run the following command to build the project:

```
gradlew build
```

        ii. Gradle will compile the application and generate the build artifacts.

6. Run the Application
    a. **In Windows:**
        i. To run the application, execute:

        ```
        gradlew run
        ```

        ii. The output should display: Hello, World!.
7. Run Tests (Optional)
    a. In Windows:
        i. To execute the tests, run:

        ```
        gradlew test
        ```

        ii. Gradle will run the tests and output the results.

CONSOLE OUTPUT:

```
D:\Tem\DevOps\Gradle\hello-world>gradlew build
Calculating task graph as no cached configuration is available for tasks:
build
[Incubating] Problems report is available at:
file:///D:/Tem/DevOps/Gradle/hello-world/build/reports/problems/problems-
report.html

BUILD SUCCESSFUL in 2s
7 actionable tasks: 4 executed, 3 from cache
Configuration cache entry stored.


D:\Tem\DevOps\Gradle\hello-world>gradlew run
Calculating task graph as no cached configuration is available for tasks:
run

> Task :app:run
Hello World!
[Incubating] Problems report is available at:
file:///D:/Tem/DevOps/Gradle/hello-world/build/reports/problems/problems-
report.html

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


D:\Tem\DevOps\Gradle\hello-world>gradlew test
Calculating task graph as no cached configuration is available for tasks:
test
[Incubating] Problems report is available at:
file:///D:/Tem/DevOps/Gradle/hello-world/build/reports/problems/problems-
report.html

BUILD SUCCESSFUL in 1s
3 actionable tasks: 3 up-to-date
Configuration cache entry stored.
```

RESULT:

Thus, a simple Java application has been successfully set up and built using Gradle on Windows. The project was initialized, the necessary configuration was made in build.gradle, and the application was built, run, and tested through Gradle commands. This process automates the build, run, and test steps, streamlining the development workflow.

EXP NO: **DATE: ___/___/_____**

Install Ansible and Configure Ansible Roles and to Write Playbooks

AIM:

To install Ansible, configure it on Kali Linux running via Windows Subsystem for Linux (WSL), and create playbooks to automate the creation of users and installation of Nginx and MariaDB on web and database servers.

PROCEDURE:

1. **Install Ansible on Kali Linux (WSL)**
    a. **On Kali Linux (WSL):**
        i. To install Ansible on Kali Linux (running on WSL), run the following commands:

```
sudo apt update
sudo apt install ansible
```

2. **Configure Ansible**

    After installing Ansible, configure it by creating a project-specific ansible.cfg file in your project directory. Here's an example of how to set up the configuration:

    a. **Create or edit the ansible.cfg file:**

```
[defaults]
inventory = ./hosts
remote_user = blackevil
private_key_file = ~/.ssh/id_rsa
```

        i. **inventory:** Specifies the location of the inventory file.
        ii. **remote_user:** The default SSH user for connecting to remote hosts.
        iii. **private_key_file:** Path to your SSH private key for authentication.

3. **Set Up Ansible Inventory**
    a. Create an inventory file named hosts in your project directory. This inventory file will define your servers and the necessary information for Ansible to communicate with them.

b. Example inventory (hosts):

```
[webservers]
webserver1 ansible_host=172.28.178.218 ansible_user=blackevil
webserver2 ansible_host=172.28.178.219 ansible_user=madhavan

[databases]
dbserver ansible_host=172.28.178.220 ansible_user=raja
```

i. The webservers group includes two servers: webserver1 and webserver2.

ii. The databases group includes dbserver.

iii. Each server is associated with an IP address and an SSH user.

4. Create Ansible Playbooks

a. Next, create a playbook that will automate the following tasks:

i. Create users user1, user2, and dbserver1 on the respective servers.

ii. Install Nginx on both web servers.

iii. Install MariaDB on the database server.

b. In this playbook:

i. Users webserver1, webserver2, and dbserver are created on their respective servers.

ii. Nginx is installed on webserver1 and webserver2.

iii. MariaDB is installed on dbserver.

c. Example playbook (site.yml):

```yaml
---
- name: Configure web servers
  hosts: webservers
  become: yes
  tasks:
    - name: Create user blackevil on webserver1
      user:
        name: blackevil
        state: present
        shell: /bin/bash
        groups: sudo
      when: ansible_host == "172.28.178.218"  # webserver1

    - name: Install Nginx on webserver1
      apt:
        name: nginx
        state: present
        update_cache: yes
      when: ansible_host == "172.28.178.218"  # webserver1

    - name: Create user madhavan on webserver2
      user:
        name: madhavan
        state: present
        shell: /bin/bash
        groups: sudo
      when: ansible_host == "172.28.178.219"  # webserver2

    - name: Install Nginx on webserver2
      apt:
        name: nginx
        state: present
        update_cache: yes
      when: ansible_host == "172.28.178.219"  # webserver2

- name: Configure database server
  hosts: databases
  become: yes
  tasks:
    - name: Create user raja on dbserver
      user:
        name: raja
        state: present
        shell: /bin/bash
        groups: sudo
      when: ansible_host == "172.28.178.220"  # dbserver

    - name: Install MariaDB on dbserver
      apt:
        name: mariadb-server
        state: present
        update_cache: yes
      when: ansible_host == "172.28.178.220"  # dbserver
```

5. **Run the Playbook**

   a. To execute the playbook and apply the changes, use the following command:

   ```
   ansible-playbook -i hosts site.yml --ask-become-pass
   ```

      i. -i hosts: Specifies the location of your inventory file.

      ii. site.yml: The playbook file you want to run.

      iii. --ask-become-pass: Prompts for the sudo password when needed.

CONSOLE OUTPUT:

```
┌──(blackevil㉿DESKTOP-BPQC4CV)-[~/ansible_project]
└─$ ansible-playbook -i hosts site.yml --ask-become-pass
BECOME password:

PLAY [Configure web servers]
*********************************************

TASK [Gathering Facts]
*****************************************************
ok: [webserver2]
ok: [webserver1]

TASK [Create user blackevil on webserver1]
*************************************************************************
ok: [webserver2]
ok: [webserver1]

TASK [Install Nginx on webserver1]
*************************************************************************
ok: [webserver2]
ok: [webserver1]

TASK [Create user madhavan on webserver2]
*************************************************************************
ok: [webserver1]
ok: [webserver2]

TASK [Install Nginx on webserver2]
*************************************************************************
ok: [webserver2]
ok: [webserver1]

PLAY [Configure database server]
*************************************************************************
TASK [Gathering Facts]
*************************************************************************
ok: [dbserver]

TASK [Create user raja on dbserver]
*************************************************************************
ok: [dbserver]

TASK [Install MariaDB on dbserver]
*************************************************************************
ok: [dbserver]

PLAY RECAP
*************************************************************************
dbserver   : ok=3    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
webserver1 : ok=5    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
webserver2 : ok=5    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

RESULT:

Thus, the Ansible playbook has successfully automated the configuration of the web and database servers. This includes creating users (webserver1, webserver2, and dbserver), installing Nginx on the web servers, and setting up MariaDB on the database server. The process simplifies server management, ensuring consistency and efficiency, while enabling easy scalability for multiple servers using Ansible in a WSL environment on Kali Linux.