RAJALAKSHMI INSTITUTE OF TECHNOLOGY
KUTHAMBAKKAM, CHENNAI - 600 124

rit | RAJALAKSHMI INSTITUTE OF TECHNOLOGY

BELIEVE IN THE POSSIBILITIES

AN AUTONOMOUS INSTITUTION

Laboratory Record Note Book

NAME .....

SUBJECT .....

BRANCH .....

REGISTER No .....

SEMESTER .....

ACADEMIC YEAR.....

RAJALAKSHMI INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION
KUTHAMBAKKAM, CHENNAI - 600 124

BONAFIDE CERTIFICATE

NAME .....

ACADEMIC YEAR.....YEAR..... BRANCH .....

REGISTER No.

Certified that this is the bonafide record of work done by the above student in the

.....Laboratory during the year **20** **- 20**

Signature
Faculty - in - Charge

Signature
H.O.D

Submitted for the Practical Examination held on.....

Internal Examiner **External Examiner**

VISION OF THE INSTITUTION

- To set a benchmark in the field of engineering education by providing quality technical education that fosters the spirit of learning, research and globally competent professionalism

MISSION OF THE INSTITUTION

- To impart education that caters to the growing challenges of the industry and social needs of our nation.
- To constantly upgrade the standards of teaching and learning in the field of engineering and technology while promoting a healthy research atmosphere.
- To foster a healthy symbiosis with the industry through meaningful and dynamic interactions.

VISION OF THE DEPARTMENT

- To initiate high quality technical education and to nurture young minds towards creative thinking that inspires them to undertake innovations in the field of Electronics and Communication Engineering (ECE) and be competent in the global arena.
- To Emphasize on the student body to carry out research for the service of our Nation and to the Society at large.

MISSION OF THE DEPARTMENT

- To constantly upgrade engineering pedagogy that caters to the growing challenges of the Industries.
- To develop conceptual learning that leads towards critical and innovative thinking.
- To establish good harmony with industry that fills the gap between academia and the real world by enabling the students to prepare for the diverse and competitive career paths.
- To endorse higher studies and pursue research in the ECE discipline with sensitivity towards societal requirements.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Graduates can

PEO1: Pursue research or have a successful career in academia or industries associated with Electronics and Communication Engineering, or as entrepreneurs.

PEO2: Provide strong foundational concepts and advanced techniques and tools in order to enable them to build solutions or systems of varying complexities**.**

PEO3: Prepare the students to critically analyse the existing literature in the area of specialization and ethically develop innovative and research-oriented methodologies to solve the identified problems**.**

IV PROGRAM OUTCOMES (POs)

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics, responsibilities, and norms of the engineering practice.

9. Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

V PROGRAM SPECIFIC OUTCOMES (PSOs)

The Students will be able to

- Analyze, design and develop solutions by applying foundational concepts of electronics and communication engineering.
- Apply design principles and best practices for developing quality products for scientific and business applications.
- Adapt to emerging information and communication technologies (ICT) to innovate ideas and solutions for the existing/novel problems.

CCS342	DEVOPS	L T P C
		2 0 2 3

COURSE OBJECTIVES:

- To introduce DevOps terminology, definition & concepts
- To understand the different Version control tools like Git, Mercurial
- To understand the concepts of Continuous Integration/ Continuous Testing/ Continuous Deployment)
- To understand Configuration management using Ansible
- Illustrate the benefits and drive the adoption of cloud-based Devops tools to solve real world problems

UNIT I INTRODUCTION TO DEVOPS **6**

Devops Essentials - Introduction To AWS, GCP, Azure - Version control systems: Git and Github.

UNIT II COMPILE AND BUILD USING MAVEN & GRADLE **6**

Introduction, Installation of Maven, POM files, Maven Build lifecycle, Build phases(compile build, test, package) Maven Profiles, Maven repositories(local, central, global),Maven plugins, Maven create and build Artificats, Dependency management, Installation of Gradle, Understand build using Gradle

UNIT III CONTINUOUS INTEGRATION USING JENKINS **6**

Install & Configure Jenkins, Jenkins Architecture Overview, Creating a Jenkins Job, Configuring a Jenkins job, Introduction to Plugins, Adding Plugins to Jenkins, Commonly used plugins (Git Plugin, Parameter Plugin, HTML Publisher, Copy Artifact and Extended choice parameters). Configuring Jenkins to work with java, Git and Maven, Creating a Jenkins Build and Jenkins workspace.

UNIT IV CONFIGURATION MANAGEMENT USING ANSIBLE **6**

Ansible Introduction, Installation, Ansible master/slave configuration, YAML basics, Ansible modules, Ansible Inventory files, Ansible playbooks, Ansible Roles, adhoc commands in ansible

UNIT V BUILDING DEVOPS PIPELINES USING AZURE **6**

Create Github Account, Create Repository, Create Azure Organization, Create a new pipeline, Build a sample code, Modify azure-pipelines.yaml file

COURSE OUTCOMES:

CO1: Understand different actions performed through Version control tools like Git.
CO2: Perform Continuous Integration and Continuous Testing and Continuous Deployment using Jenkins by building and automating test cases using Maven & Gradle.
CO3: Ability to Perform Automated Continuous Deployment
CO4: Ability to do configuration management using Ansible
CO5: Understand to leverage Cloud-based DevOps tools using Azure DevOps

30 PERIODS

PRACTICAL EXERCISES: **30 PERIODS**

1. Create Maven Build pipeline in Azure
2. Run regression tests using Maven Build pipeline in Azure
3. Install Jenkins in Cloud
4. Create CI pipeline using Jenkins
5. Create a CD pipeline in Jenkins and deploy in Cloud
6. Create an Ansible playbook for a simple web application infrastructure
7. Build a simple application using Gradle
8. Install Ansible and configure ansible roles and to write playbooks

TEXT BOOKS

1. Roberto Vormittag, “A Practical Guide to Git and GitHub for Windows Users: From Beginner to Expert in Easy Step-By-Step Exercises”, Second Edition, Kindle Edition, 2016.
2. Jason Cannon, “Linux for Beginners: An Introduction to the Linux Operating System and Command Line”, Kindle Edition, 2014

REFERENCES

1. Hands-On Azure Devops: Cicd Implementation For Mobile, Hybrid, And Web Applications Using Azure Devops And Microsoft Azure: CICD Implementation for ... DevOps and Microsoft Azure (English Edition) Paperback – 1 January 2020 by Mitesh Soni
2. Jeff Geerling, “Ansible for DevOps: Server and configuration management for humans”, First Edition, 2015.
3. David Johnson, “Ansible for DevOps: Everything You Need to Know to Use Ansible for DevOps”, Second Edition, 2016.
4. Mariot Tsitoara, “Ansible 6. Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer”, Second Edition, 2019.
5. https://www.jenkins.io/user-handbook.pdf
6. https://maven.apache.org/guides/getting-started/

CO’s- PO’s & PSO’s MAPPING

CO’s	PO’s												PSO’s		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1	3	3	3	2	3	-	-	-	-	-	-	-	2	2	2
2	3	3	3	2	3	-	-	-	-	-	-	-	2	2	2
3	3	3	3	2	3	-	-	-	-	-	-	-	2	2	2
4	3	3	3	2	3	-	-	-	-	-	-	-	2	2	2
5	3	3	3	2	3	-	-	-	-	-	-	-	2	2	2
AVg.	3	3	3	2	3	-	-	-	-	-	-	-	2	2	2

1 - low, 2 - medium, 3 - high, ‘-' - no correlation

INDEX

Ex. No: 1	CREATE MAVEN BUILD PIPELINE IN AZURE
Date:	

Aim:

To create Maven build pipeline in Azure

Procedure:

1. **Create a New Project:**
    - Log in to your Azure DevOps account.
    - Create a new project or use an existing one.
2. **Create a New Pipeline:**
    - In your project, navigate to the "Pipelines" section.
    - Click on the "New Pipeline" button.
    - Select the repository where your Maven project is stored.
3. **Configure Pipeline:**
    - Choose a template. You can start with an "Empty Job" if there's no specific template for your technology stack.
    - You'll be directed to the pipeline configuration page.
4. **Configure the Build Steps:**
    - Define your agent pool, VM image, and other basic settings.
    - Under "Agent Job 1," click on the "+" button to add a new task.
    - Search for "Maven" and add the "Maven" task.
5. **Configure the Maven Task:**
    - In the Maven task configuration, you'll need to provide the path to your **pom.xml** file and select the Maven version you want to use.

- Choose the goals you want to run (e.g., **clean install**, **package**, etc.).
- Configure other options like Advanced options and Java options if needed.

6. **Save and Queue:**

   - Save your pipeline configuration.
   - Click on "Save & queue" to initiate the pipeline run.
   - Review the pipeline run logs to ensure everything is running as expected.

7. **Artifact Publishing (Optional):**

   - If your project generates an artifact (like a JAR or WAR file), you might want to publish it as an artifact that can be used in subsequent stages or releases.

8. **Triggers and Continuous Integration (CI) Setup:**

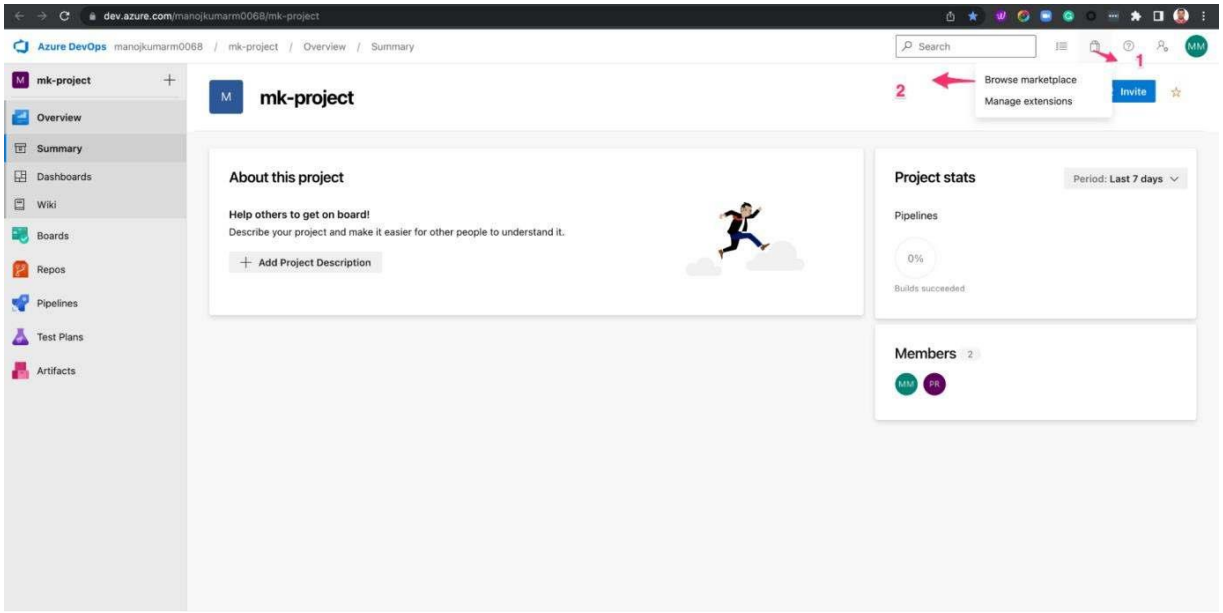
   - Configure triggers to automate pipeline runs. For example, you can trigger the pipeline whenever changes are pushed to specific branches.

9. **Environment-Specific Configuration (Optional):**

   - If you have different environments (like development, testing, production), you can set up multiple stages in the pipeline to deploy to these environments.

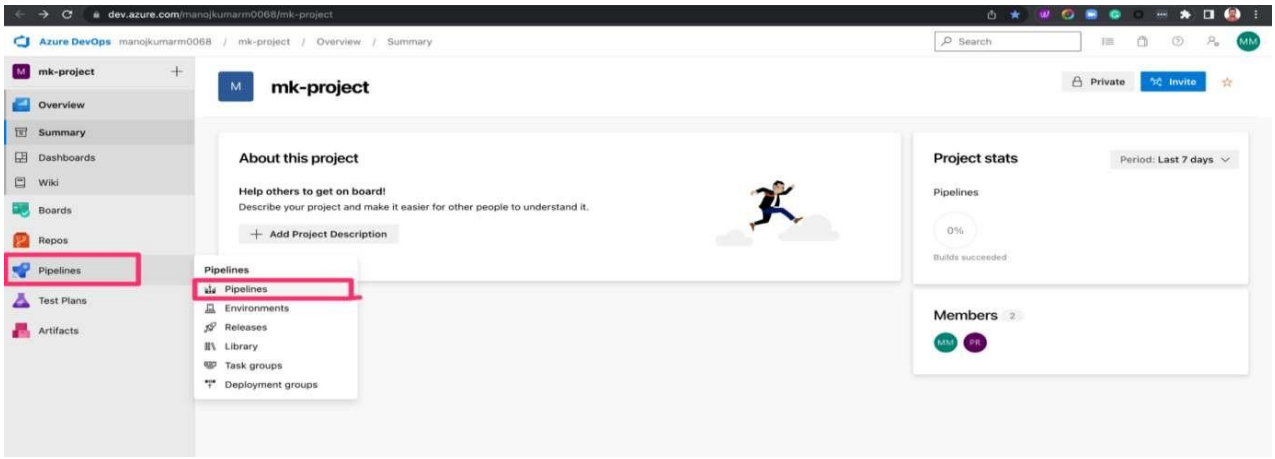
Screen Shots

Step 1: Organization is created

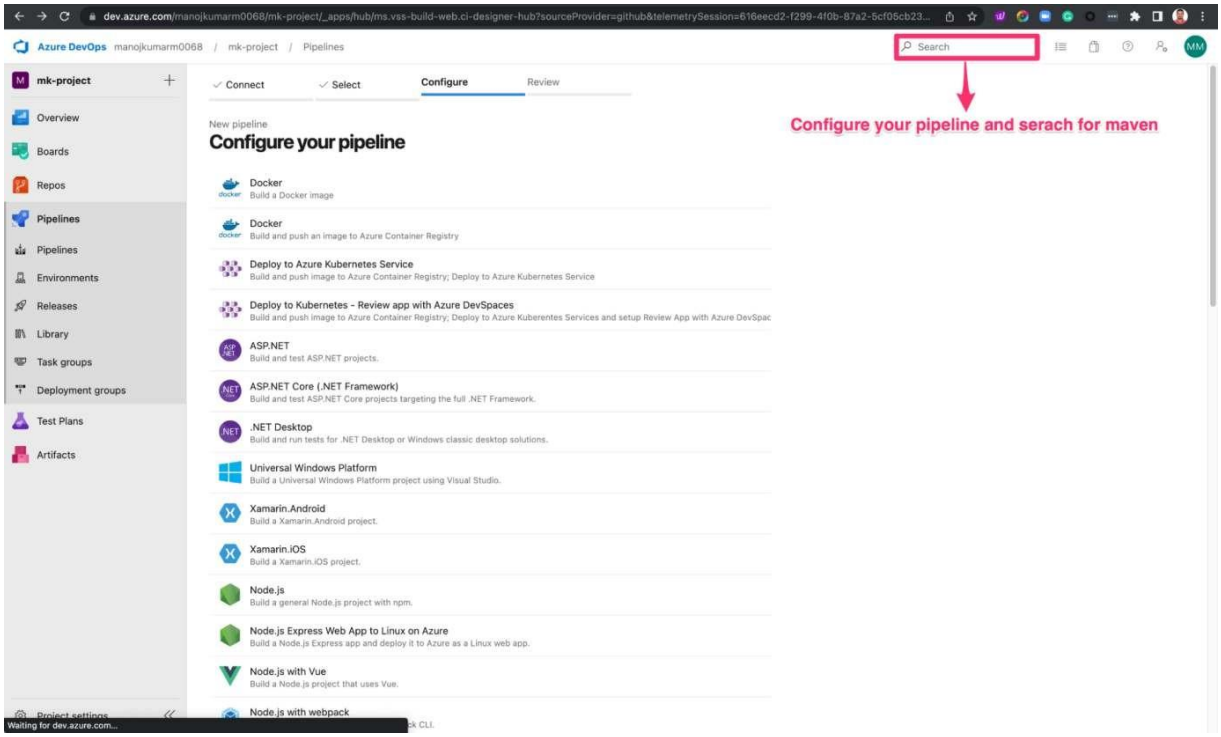


Step 2: Organization by clicking the new project button

Step3: Create Pipeline



Step4: Configure new project.



Results:

Thus, the above programme to Create Maven Build Pipeline in Azure is executed and verified successfully.

Ex. No: 2	RUN REGRESSION TESTS USING MAVEN BUILD PIPELINE IN AZURE
Date:	

Aim:

To run regression tests using a Maven build pipeline in Azure

Procedure:

Step 1: Create a New Build Pipeline

1. Open your Azure DevOps project.
2. Navigate to Pipelines > Builds.
3. Click on the "+ New" button to create a new build pipeline.

Step 2: Configure the Source Repository

1. Select the repository where your Maven project is stored.
2. Choose the branch you want to build.

Step 3: Configure the Build Pipeline

1. Choose a template: Select "Maven" as the template for your pipeline.

Step 4: Configure the Maven Task

1. In the pipeline editor, you will see a task named "Maven" (or similar). Click on it to configure the Maven task.
2. In the "Goals" field, specify the Maven goals needed to run your regression tests. For example, if your regression tests are part of the "integration-test" phase, you can enter: **clean integration-test**.
3. Set other Maven-related configurations as needed (e.g., POM file, options).

Step 5: Add Test Reporting (Optional)

1. If your regression tests produce test reports, you might want to publish them in Azure DevOps.

2. After the Maven task, add a task to publish the test reports. For example, if your tests generate Surefire or Failsafe reports, you can use the "Publish Test Results" task and specify the path to the test report XML files.

Step 6: Save and Queue the Pipeline

1. After configuring the pipeline, click on "Save & Queue" to save your changes and trigger a build.

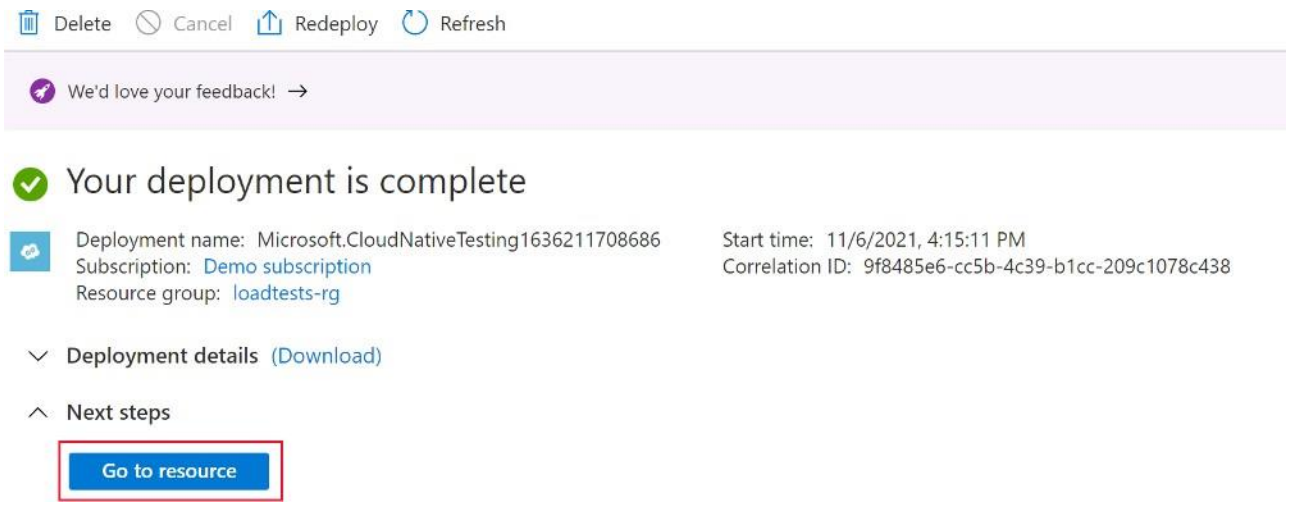
Step 7: Monitor the Build

1. Once the pipeline is triggered, it will run the Maven build, including the regression tests.
2. You can monitor the progress and results in the Azure DevOps pipeline interface.

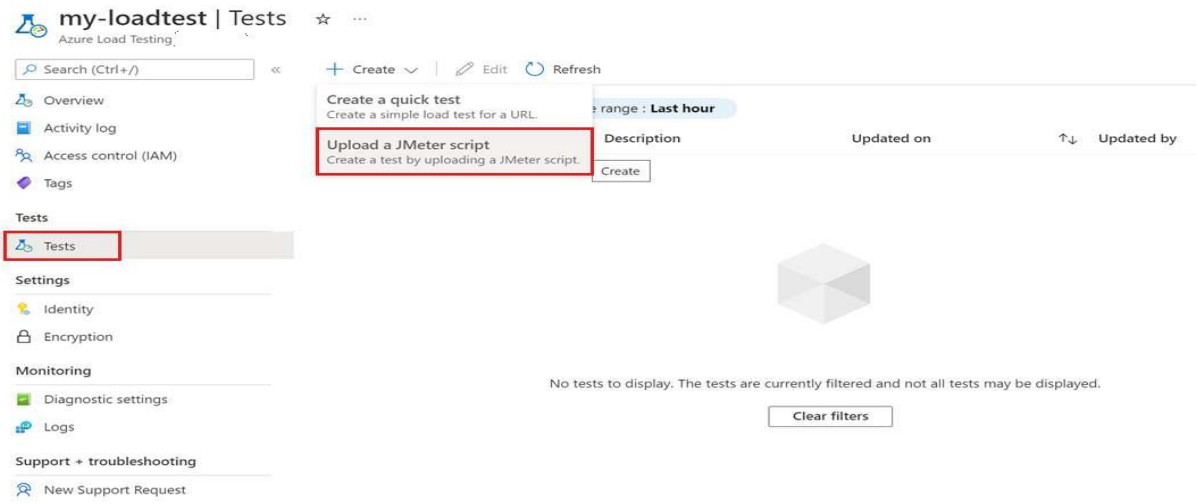
Step 8: Set Up Triggers (Optional)

1. You can set up triggers to automatically run the build pipeline when changes are pushed to specific branches.
2. Configure branch filters and triggers according to your needs.

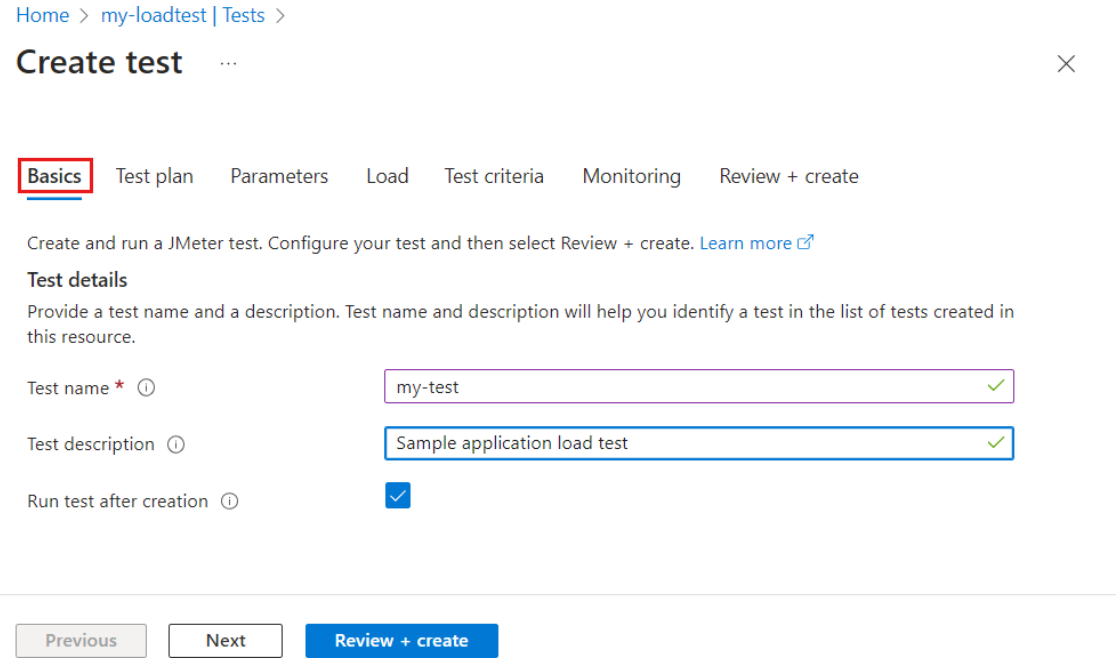
Step1: new resource, select go to resource**.**



Step2: Azure Load Testing resource, select Tests



Step3: On the Basics tab, enter the Test name and Test description information.



Result:

Thus the above programs to run regression tests using a Maven build pipeline in Azure is executed and verified successfully.

Ex. No: 3	INSTALL JENKINS IN CLOUD
Date:	

Aim:

To install Jenkins in a cloud environment

Procedure:

1. **Choose a Cloud Provider:** Decide on a cloud provider that you want to use. For this example, let's use Amazon Web Services (AWS).
2. **Create a Virtual Machine:**
    - Log in to your AWS console.
    - Navigate to the EC2 service (Elastic Compute Cloud).
    - Click on "Launch Instances" to create a new virtual machine.
    - Choose an Amazon Machine Image (AMI) that supports your desired operating system (e.g., Amazon Linux, Ubuntu Server).
    - Configure the instance details (instance type, networking, etc.).
    - Add storage as needed.
3. **Security Group Configuration:**
    - Configure security groups to allow incoming traffic on the necessary ports. Jenkins typically uses port 8080 for web access and optionally other ports for agents.
4. **Connect to the Virtual Machine:**
    - Once the virtual machine is launched, connect to it using SSH.
    - You can use the public IP address or DNS name provided by AWS.
5. **Install Java:**
    - Jenkins requires Java to run. Install Java on the virtual machine using the package manager for your chosen operating system.
6. **Install Jenkins:**

- Add the Jenkins repository to your package manager's sources.
- Install Jenkins using the package manager.
- Start the Jenkins service.

7. **Access Jenkins Web Interface:**
   - Open a web browser and navigate to the public IP address or DNS name of your virtual machine, followed by port 8080.
   - You will need to enter the initial administrator password, which can be found on the virtual machine in a file.

8. **Complete Jenkins Setup:**
   - Follow the on-screen instructions to complete the Jenkins setup process.
   - Install recommended plugins or select the plugins you need.
   - Create an admin user account.

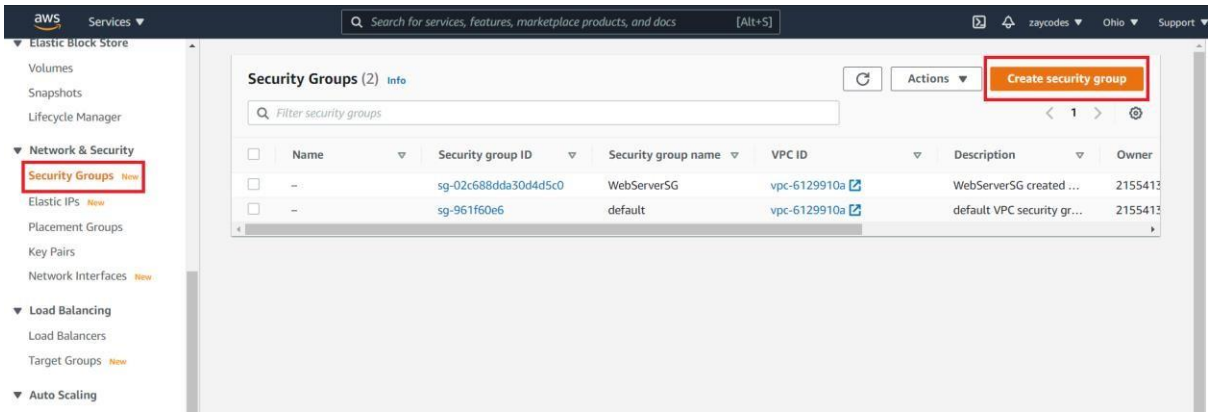
9. **Customize and Secure:**
   - Configure Jenkins settings according to your preferences.
   - Implement security measures, such as setting up authentication, authorization, and HTTPS.

Step1: Enter the IP Address in url:103.43.200.163

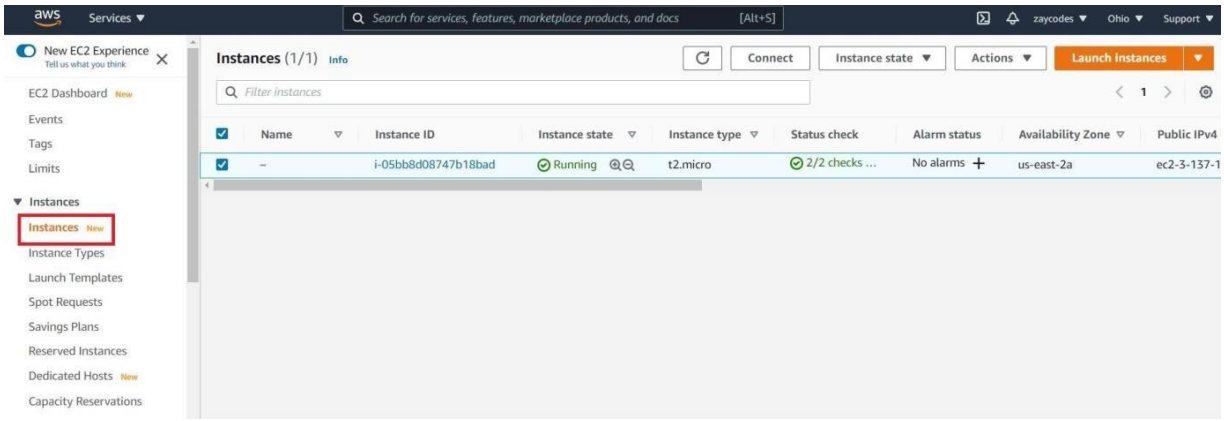
Step2: Sign in to the AWS Management Console.

Step3: Open the Amazon EC2 console

Step4: select Security Groups



Step5: choose Instances



Result:

Thus, the above experiment to install Jenkins in a cloud environment is completed successfully.

Ex. No: 4	CREATE CI PIPELINE USING JENKINS
Date:	

Aim:

To Creating a Continuous Integration (CI) pipeline using Jenkins

Procedure:

1. **Create a New Jenkins Job:**
   - Log in to your Jenkins dashboard.
   - Click on "New Item" to create a new job.
   - Enter a name for your job (e.g., "MyCIJob") and select the "Pipeline" option.
2. **Configure Pipeline:**
   - In the pipeline section, select "Pipeline script" from the Definition dropdown.
   - You can write your pipeline script directly in the Jenkins UI or load it from a Jenkinsfile located in your version control repository.
3. **Write the Pipeline Script:** Below is a basic example of a Jenkins pipeline script that checks out code, builds it, and runs tests. Customize it according to your project's needs.

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                checkout scm}}
        stage('Build') {
            steps {
```

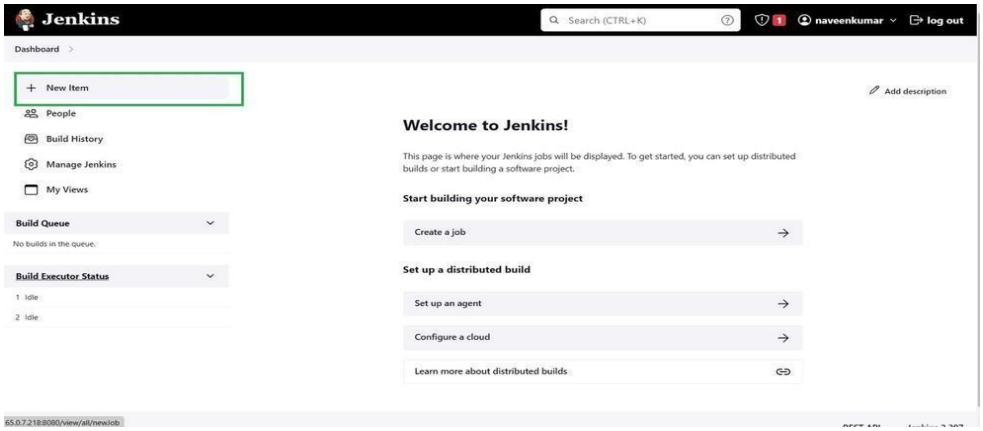
```
        sh 'echo "Building"'}}
    stage('Test') {
        steps {
            sh 'echo "Testing"'        }}}}
```

4. **Save and Run:**
    - Save your pipeline configuration.
    - Click on "Build Now" to trigger the pipeline.
    - Jenkins will run the pipeline steps defined in your script.
5. **View Pipeline Results:**
    - You can view the progress and results of your pipeline by clicking on the job name and selecting a specific build number.
    - Jenkins will display the console output and the status of each stage.
6. **Adding Additional Steps:**
    - You can expand your pipeline to include additional stages such as deployment, code analysis, and notifications.

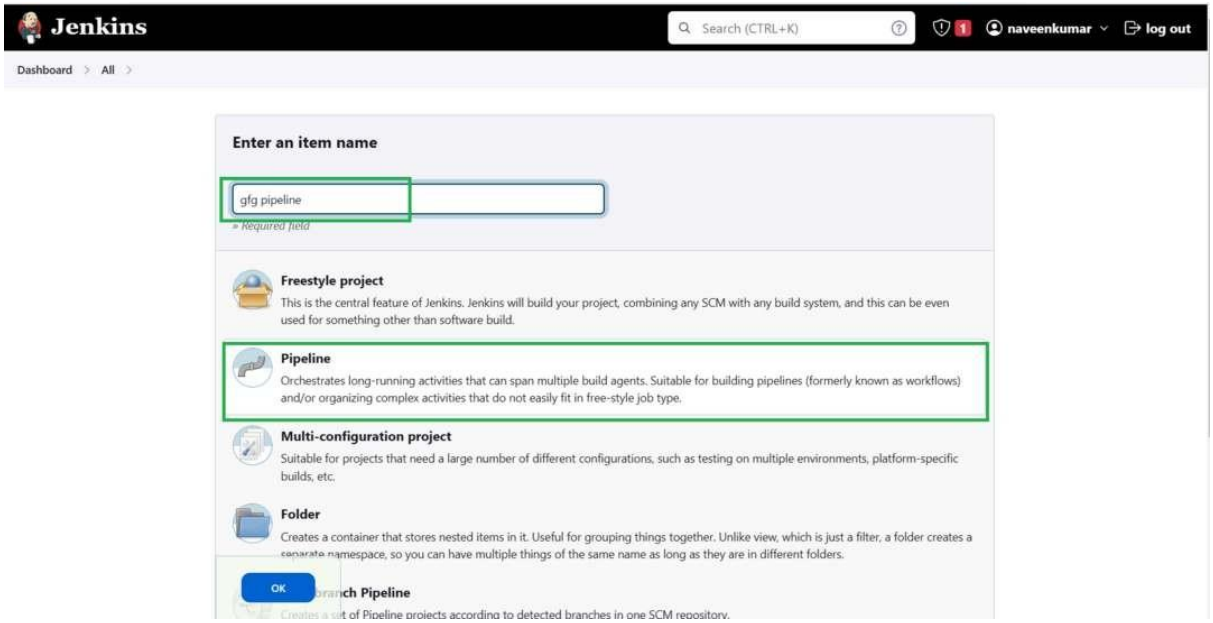
Step 1: Login into your Jenkins account as shown below.

Step 2: Once logged in, the user will be redirected to the Jenkins console,

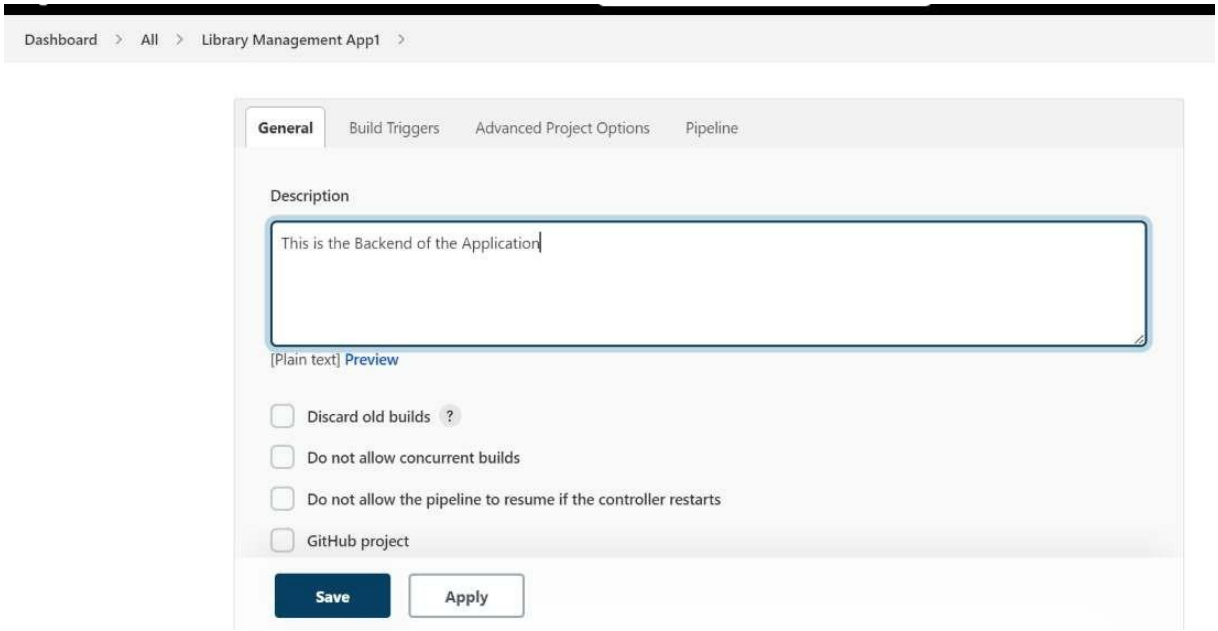
Step 3: To create a new project select the option.



Step 4: Now a list of options will be visible on the screen.



Step 5: Once redirected.



Result:

Thus, the above program to create a Continuous Integration (CI) pipeline using Jenkins is executed and verified successfully.

Ex. No: 5	CREATE A CD PIPELINE IN JENKINS AND DEPLOY IN CLOUD
Date:	

Aim:

To Create a CD pipeline in Jenkins and deploy in Cloud.

Prerequisites:

1. Jenkins installed and running.
2. AWS account and Elastic Beanstalk environment set up.
3. Your code hosted in a version control repository (e.g., GitHub).

Here's a high-level overview of the steps to create the pipeline:

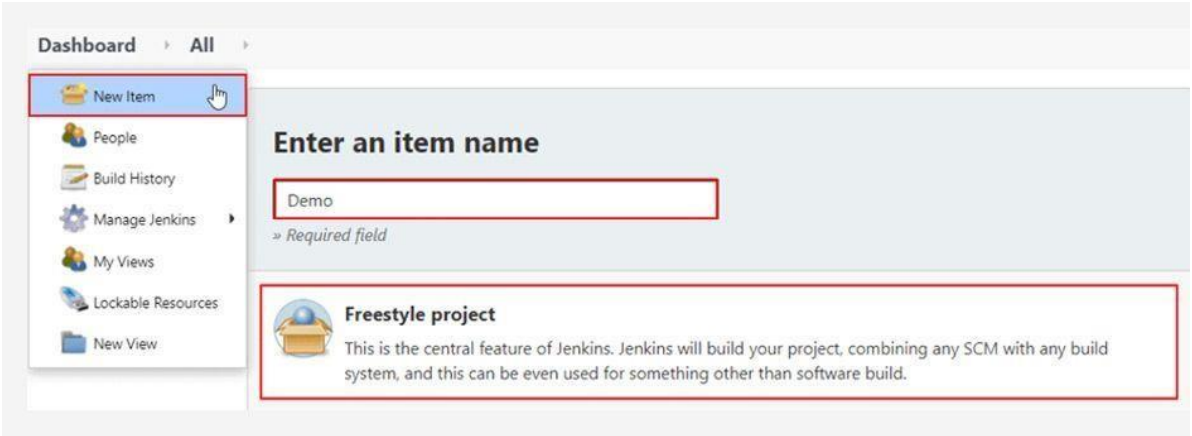
1. **Set Up Jenkins Job**:
   - Log in to your Jenkins instance.
   - Create a new pipeline job.
   - In the pipeline configuration, choose your version control system (e.g., Git) and provide the repository URL.
2. **Configure Jenkinsfile**: Create a **Jenkinsfile** in your repository's root directory. This file defines the pipeline stages and steps.

```
pipeline {
   agent any
   stages {
      stage('Checkout') {
         steps {
            checkout scm}}
      stage('Build') {
      stage('Deploy') {
         steps {
```

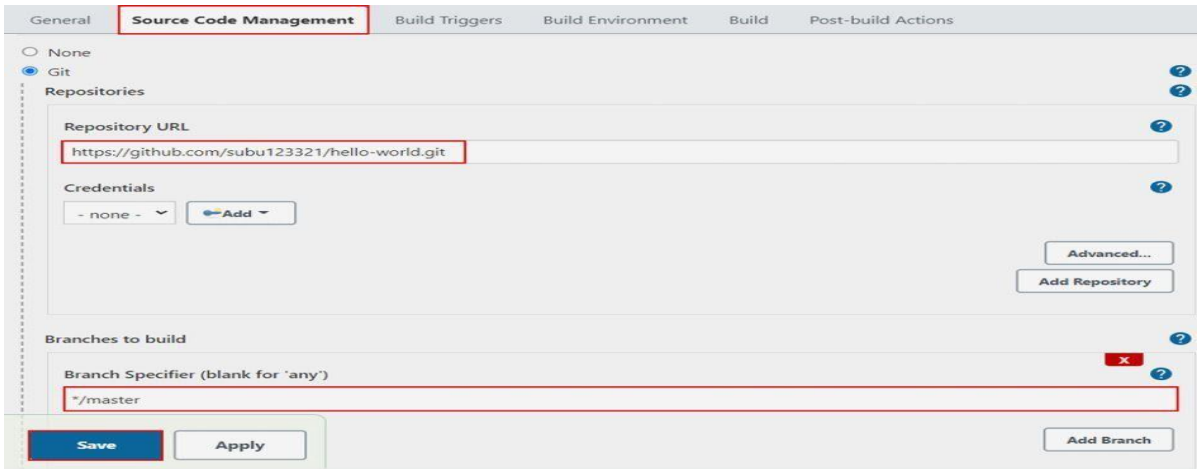
```
sh '"aws configure set aws_access_key_id <your_access_key>
    aws configure set aws_secret_access_key <your_secret_key>
    eb init <your_eb_app_name> -p <your_eb_platform>
    eb deploy"'}}}}
```

4. Replace placeholders (**<your_access_key>**, **<your_secret_key>**, **<your_eb_app_name>**, **<your_eb_platform>**) with your actual AWS credentials and Elastic Beanstalk information.
5. **Install Required Plugins**: Depending on your cloud provider and tools, you might need additional plugins in Jenkins. For AWS Elastic Beanstalk, you'd need the "AWS Elastic Beanstalk" plugin.
6. **Configure AWS Credentials in Jenkins**:
    1. Go to Jenkins Dashboard > Credentials > System.
    2. Add new AWS credentials with access key and secret key.
7. **Run the Pipeline**:
    1. Trigger the pipeline manually or configure webhooks to trigger it automatically on code changes.

Step1: Click on Freestyle project

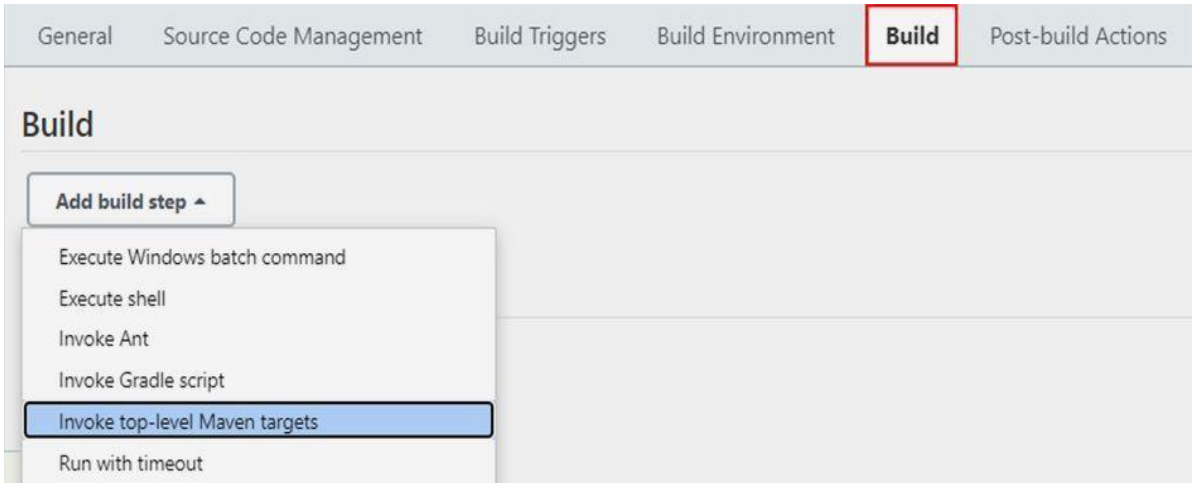


Step2: Source Code Management window.



Step3: Jenkins needs to fetch all the repository files

Step4: Go to the Build tab and add the build step



Step5: Drop- down file menu

Step 6: Build Now

Result:

Thus, the above program was successfully completed using CD pipeline in Jenkins and deploy in Cloud.

Ex. No: 6	CREATE AN ANSIBLE PLAYBOOK FOR A SIMPLE WEB APPLICATION INFRASTRUCTURE
Date:	

Aim:

To create simple Ansible playbook for setting up a basic web application

Program:

```
- name: Setup Web Application Infrastructure
  hosts: all
  become: yes
  tasks:
    - name: Update apt cache (Ubuntu) or yum cache (CentOS)
      apt:
        update_cache: yes
      when: ansible_os_family == "Debian"
    - name: Install required packages
      apt:
        name:
          - nginx
          - mysql-server
      state: present
      yum:
        name:
          - nginx
          - mysql-server
        state: present
```

```
    when: ansible_os_family == "RedHat"
  - name: Copy nginx configuration file
    copy:
      src: files/nginx.conf
      dest: /etc/nginx/nginx.conf
  - name: Ensure MySQL root password is set
    debconf:
      name: mysql-server
      question: mysql-server/root_password
      value: "{{ mysql_root_password }}"
      vtype: password
    become: yes
  - name: Create database for the web application
    mysql_db:
      name: mywebapp
    state: present
    become: yes
handlers:
      state: restarted
```

Result:

Thus, the above program to Create an Ansible playbook for a simple web application infrastructure is executed and verified successfully.

Ex. No: 7

Date:

BUILD A SIMPLE APPLICATION USING GRADLE

Aim:

To build a simple application using gradle

Procedure:

1. Create a new directory for your project and navigate to it in your terminal.
2. Create a file named **build.gradle** in the project directory and add the following content:

```
apply plugin: 'java'

repositories {
    jcenter()
}

dependencies {
    compile 'com.google.guava:guava:30.1-jre'
}

sourceSets {
    main {
        java {
            srcDir 'src'
        }
    }
}
```

3. Create a **src** directory in your project directory.
4. Inside the **src** directory, create a package directory structure that matches your package name, for example, **com.example**.
5. Inside your package directory, create a Java file named **Main.java** with the following content:

```
package com.example;

import com.google.common.base.Joiner;

public class Main {

   public static void main(String[] args) {

      String[] words = {"Hello", "Gradle"};

      String joinedWords = Joiner.on(" ").join(words);

      System.out.println(joinedWords);

   }

}
```

6. pen a terminal and navigate to your project directory.
7. Run the following command to build and run your application:

   ./gradlew run

Result:

Thus, the above programs to build a simple application using gradle is successfully completed.

Ex. No: 8	INSTALL ANSIBLE AND CONFIGURE ANSIBLE ROLES AND TO WRITE PLAYBOOKS
Date:	

Aim:

To Install Ansible and configure ansible roles and to write playbooks

Procedure:

Installing Ansible:

For Ubuntu/Debian:

sudo apt update

sudo apt install ansible

For CentOS/RHEL:

sudo yum install epel-release

sudo yum install ansible

For macOS:

brew install ansible

For Windows:

1. You can use Windows Subsystem for Linux (WSL) or install Ansible in a Linux virtual machine.

Ansible Directory Structure:

Create a directory structure for your Ansible project:

Plaintext

ansible_project/

|-- inventory/

| |-- hosts

|-- ansible.cfg

inventory: This directory contains your inventory file where you define the hosts you want to manage.

- **roles**: This directory will hold your Ansible roles.
- **ansible.cfg**: Ansible configuration file.

Writing an Ansible Role:

Inside the **roles** directory, create a new role:

ansible-galaxy init my_role

This will create a basic directory structure for your role.

Writing an Ansible Playbook:

Create a playbook inside the **playbooks** directory. For example, create a file named **my_playbook.yml**:

```
- name: My Ansible Playbook
  hosts: your_target_host_group
  roles:
    - my_role
```

Replace **your_target_host_group** with the group of hosts defined in your inventory.

Running the Ansible Playbook:

Run the playbook using the following command:

ansible-playbook -i inventory/hosts playbooks/my_playbook.yml

Result:

Thus, the above program to Install Ansible and configure ansible roles and to write playbooks is executed and verified successfully.