C.ABDUL HAKEEM COLLEGE OF ENGINEERING & TECHNOLOGY

Hakeem Nagar, Melvisharam -632509,Vellore District,TamilNadu, India.

(Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai) (Regd. Under Sec 2(F) & 12(B) of the UGC Act 1956)

**Name of the Candidate:**

**Year:** III **Semester:** VI **Degree/Branch:** B.Tech./IT

**Subject Name:** DEVEOPS **Sub.Code:**CCS342

**University Register Number:**

# CERTIFICATE

Certified that this is the bonafide record of work done by the above student in **CCS342-DEVEOPS** during the year 2023 - 2024.

**Signature of Head of the Department Signature of Lab In-charge**

**Submitted for the University Practical Examination held on**

# EXAMINERS

|  |  |
| --- | --- |
| **Date:** | **Centre code:**5106 |
| **Internal:** | **External:** |

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**CCS342 DEVOPS LAB MANUAL**

**LIST OF PRACTICAL EXERCISES**

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

**TOTAL: 30 Periods**

|  |  |
| --- | --- |
| EX.NO:1 | CREATE MAVEN BUILD PIPELINE IN AZSURE |

1. **Create a New Project:**
   * Log in to your Azure DevOps account.
   * Create a new project or use an existing one.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# Triggers and Continuous Integration (CI) Setup:

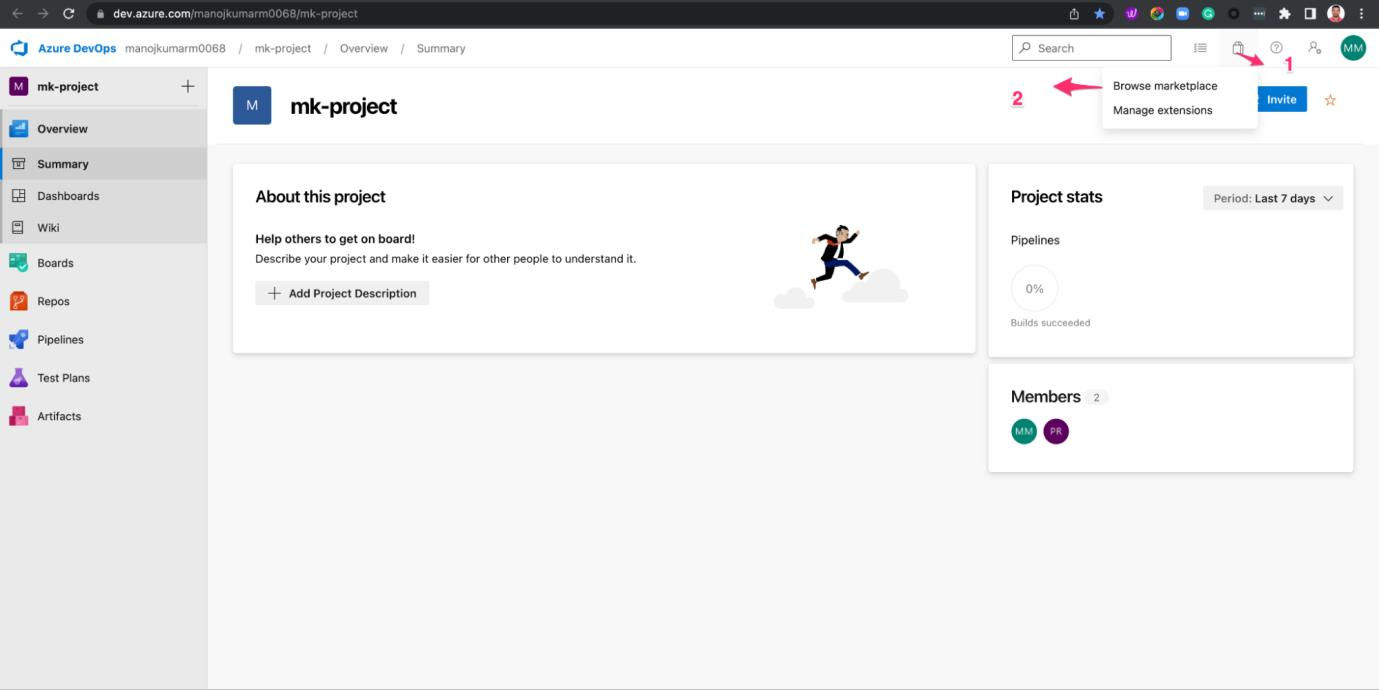
* + Configure triggers to automate pipeline runs. For example, you can trigger the pipeline whenever changes are pushed to specific branches.

# Environment-Specific Configuration (Optional):

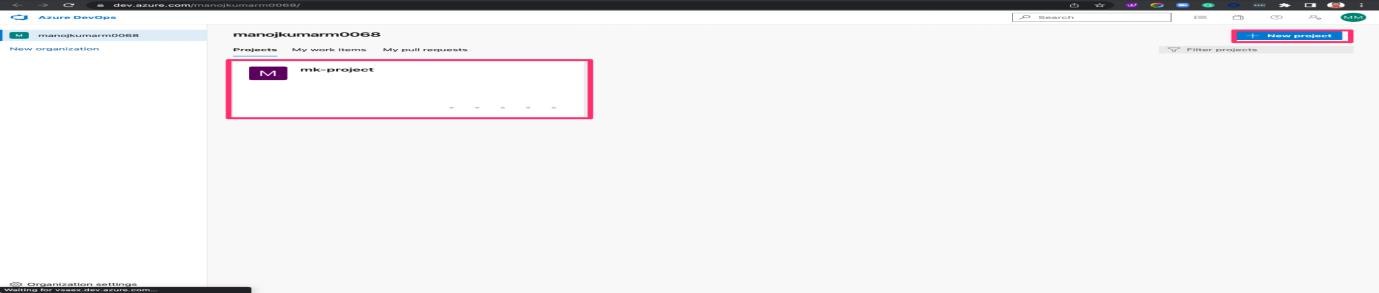
* + If you have different environments (like development, testing, production), you can set up multiple stages in the pipeline to deploy to these environments.

Remember that Azure DevOps pipelines can be highly customized to fit your project's needs. This outline provides a general idea of setting up a Maven build pipeline. The actual steps might vary based on your specific project structure and requirements. Always consult the most recent Azure DevOps documentation for the most accurate and up-to-date instructions.

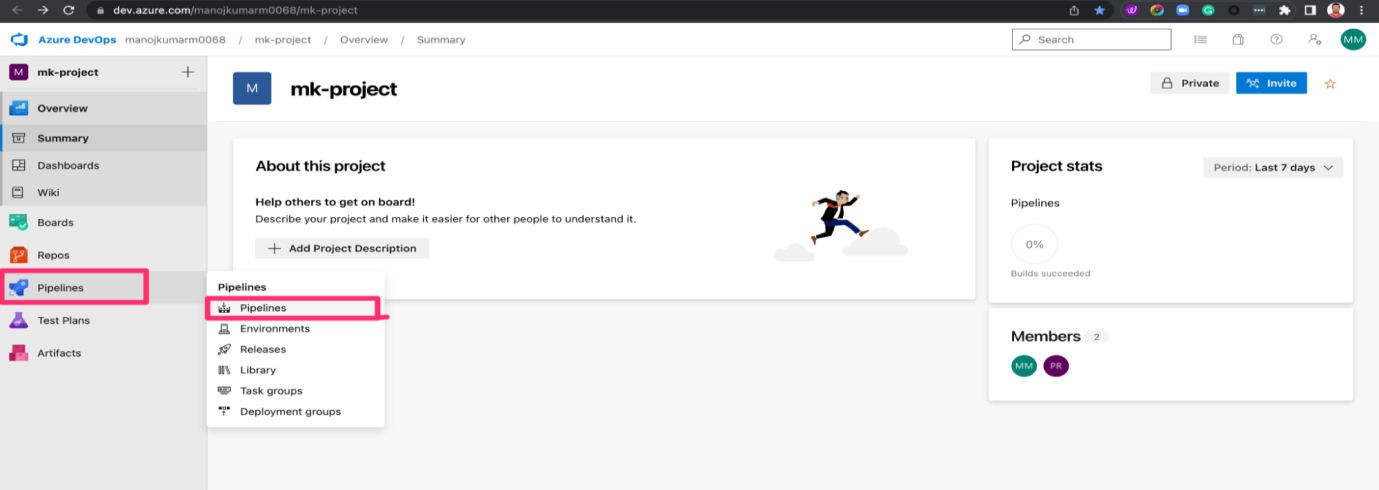
Step 1:Organization is created



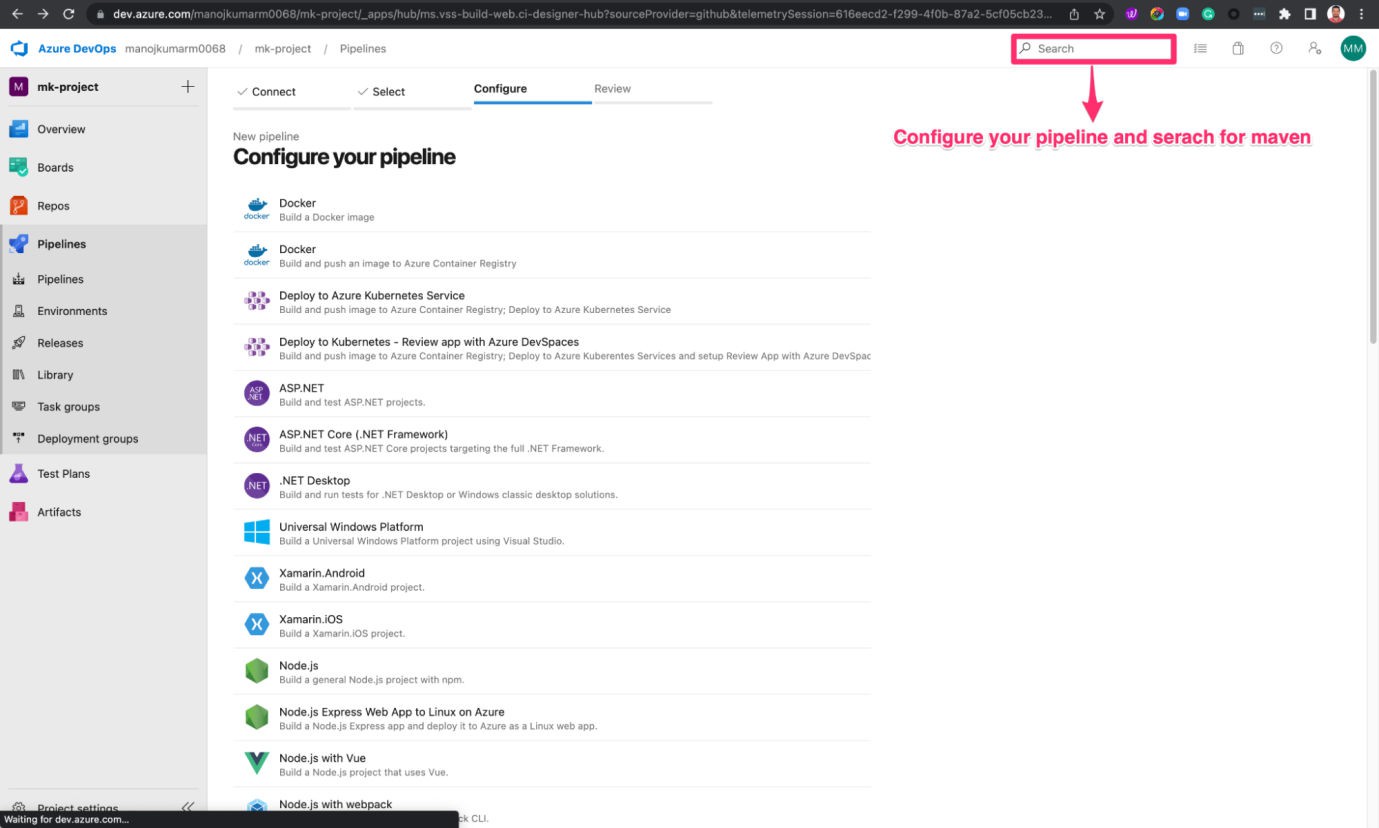
Step 2: Organization by clicking the new project button



Step3: Create Pipeline



Step4: Configure new project.



Results: Thus, the above programmes Create Maven Build Pipeline In Azsure

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | **EX.NO:2** | **To run regression tests using a Maven build pipeline in Azure** |   AIM: To run regression tests using a Maven build pipeline in Azure | |
|  | |
| **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.

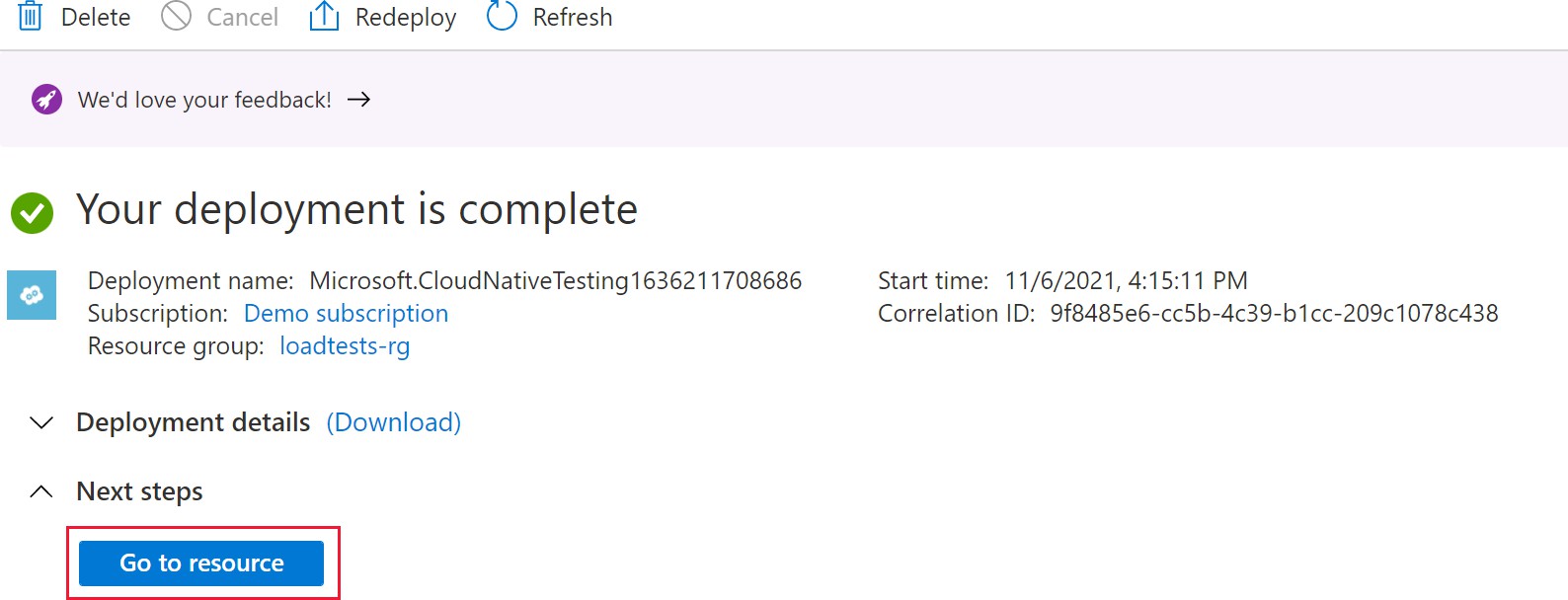
**Step 8: Set Up Triggers (Optional)**

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.

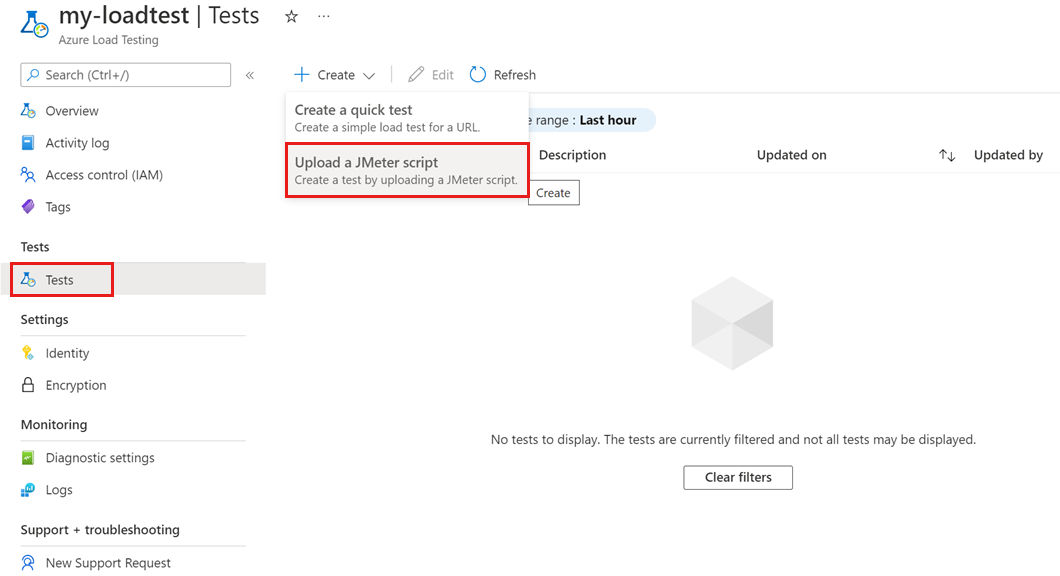
Remember that Azure DevOps provides a flexible environment, and you can customize your pipeline further to accommodate your specific project requirements.

Note: The steps and options might vary slightly depending on updates to Azure DevOps after my knowledge cutoff date in September 2021. Make sure to refer to the latest Azure DevOps documentation for accurate and up-to-date instructions

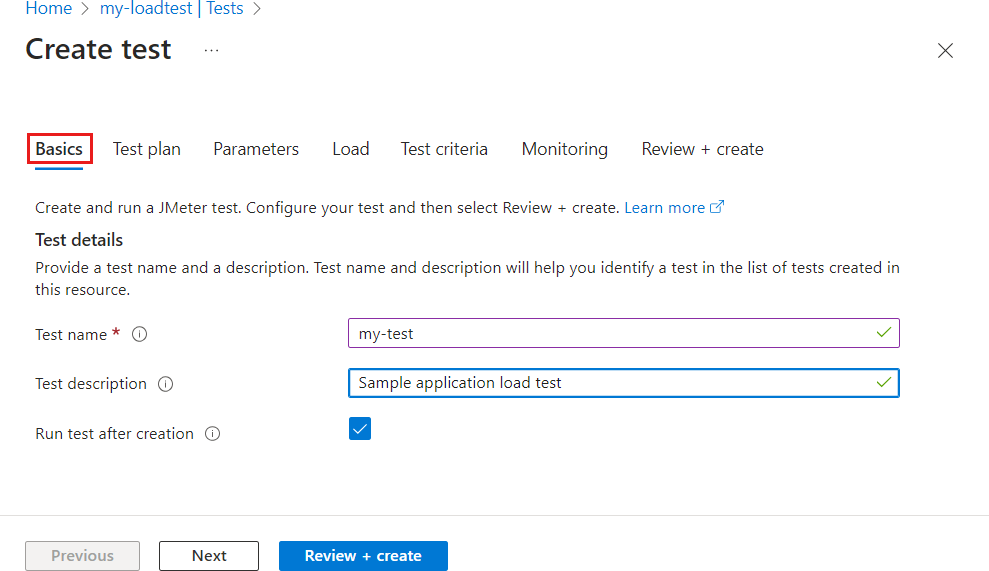
Step1: new resource, select go to resource**.**

****

Step2: Azure Load Testing resource, select Tests



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



Results: Thus the above programs completed run regression tests using a Maven build pipeline in Azure

|  |  |
| --- | --- |
| EX.NO:3 | To install Jenkins in a cloud environment |

# 

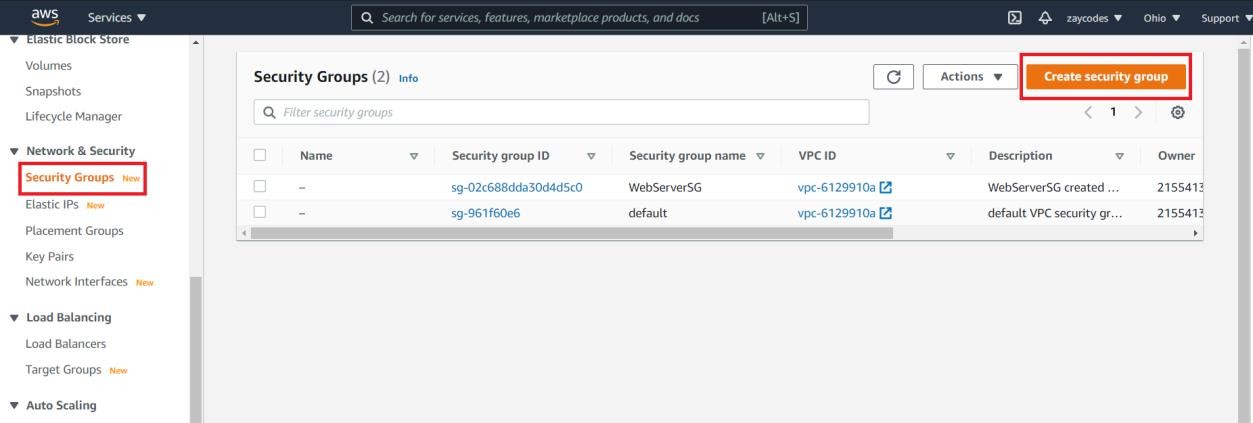
AIM: To install Jenkins in a cloud environment

|  |  |
| --- | --- |
| //To install Jenkins in a cloud environment, you'll typically need to choose a cloud provider (such as AWS, Azure, Google Cloud, etc.) and then deploy a virtual machine (VM) or container instance where Jenkins can run. Below, I'll provide a general outline of how to install Jenkins in a cloud environment using a virtual machine as an example. Keep in mind that the exact steps can vary depending on the cloud provider you choose.// | |
| **Example: Installing Jenkins on a Virtual Machine in AWS** |  |

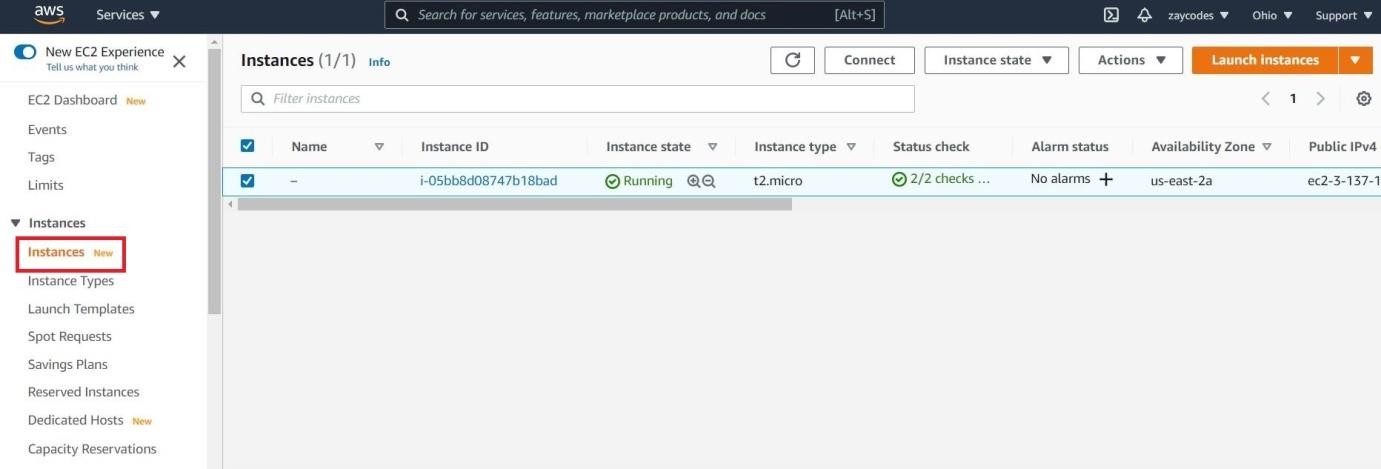
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | **Choose a Cloud Provider:** | | | | | | | Decide on a cloud provider that you want to use. For this | | |
| example, let's use Amazon Web Services (AWS). | | | | | | | | | | |
| 2. | **Create 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



Results: Thus, the above install Jenkins in a cloud environment and completed successfully completed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **EX.NO:4** | **Creating a Continuous Integration (CI) pipeline using Jenkins** |     AIM: To Creating a Continuous Integration (CI) pipeline using Jenkins | | |
|  | //Creating a Continuous Integration (CI) pipeline using Jenkins involves setting up a series of automated steps that facilitate the integration and testing of code changes as they are committed to a version control repository. Below, I'll provide you with a basic example of how to set up a CI pipeline using Jenkins:// | |
|  | **Prerequisites:** |  |

1. Jenkins installed and running.
2. A version control system (e.g., Git) repository containing your code.

**Steps to Create a CI Pipeline in Jenkins:**

# 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.

# 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.

1. **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 {

// Replace with your build commands (e.g., compiling, packaging) sh 'echo "Building"'

}

}

stage('Test') { steps {

* You can expand your pipeline to include additional stages such as deployment, code analysis, and notifications.

**Adding Additional Steps:**

6.

* 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.

**View Pipeline Results:**

5.

* Save your pipeline configuration.
* Click on "Build Now" to trigger the pipeline.
* Jenkins will run the pipeline steps defined in your script.

**Save and Run:**

4.

}

}

}

}

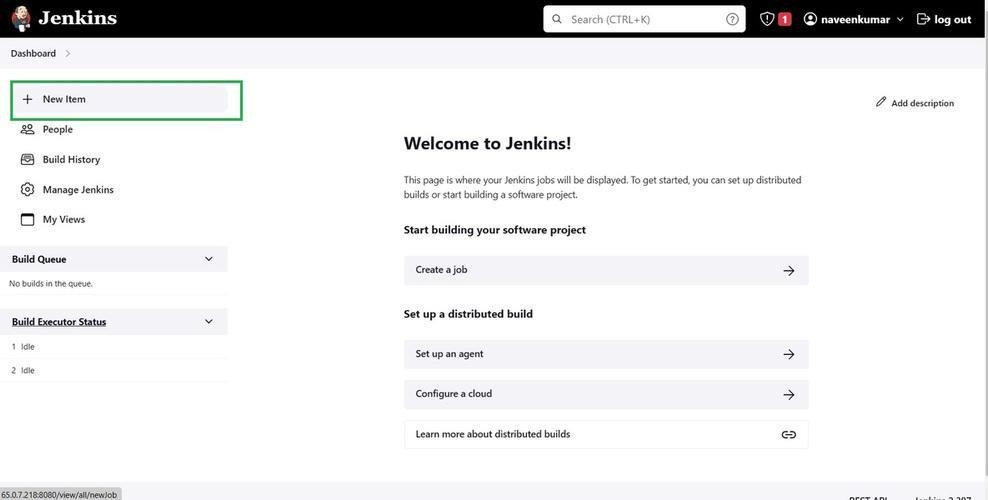
sh 'echo "Testing"'

// Replace with your test commands (e.g., unit tests, integration tests)

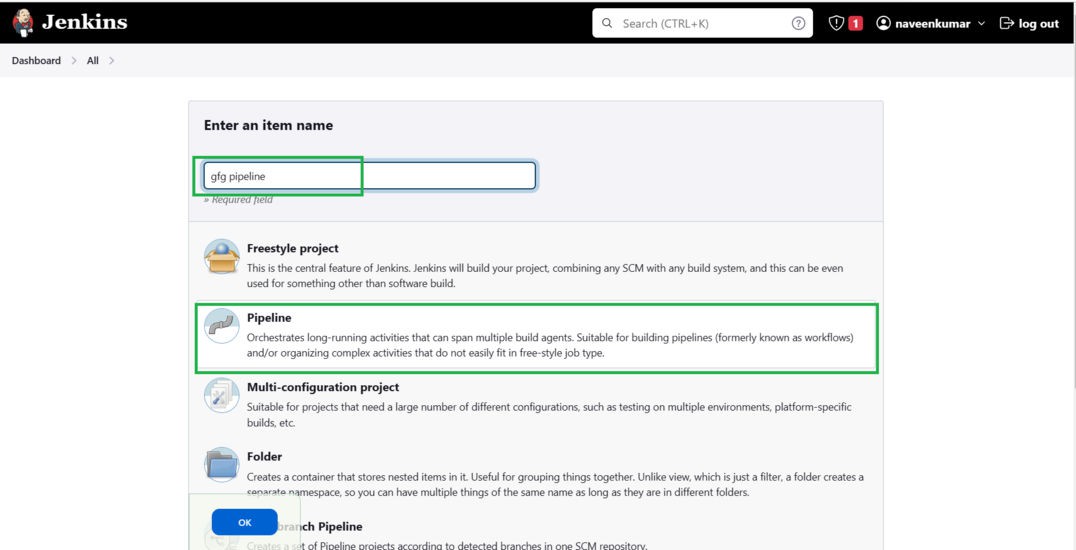
Remember that this is a basic example, and CI pipelines can become more complex depending on your project's requirements. You might want to incorporate tools like unit testing frameworks, code quality analysis tools, and deployment to various environments. Additionally, using a Jenkinsfile within your version control repository allows you to manage your pipeline script alongside your code.

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

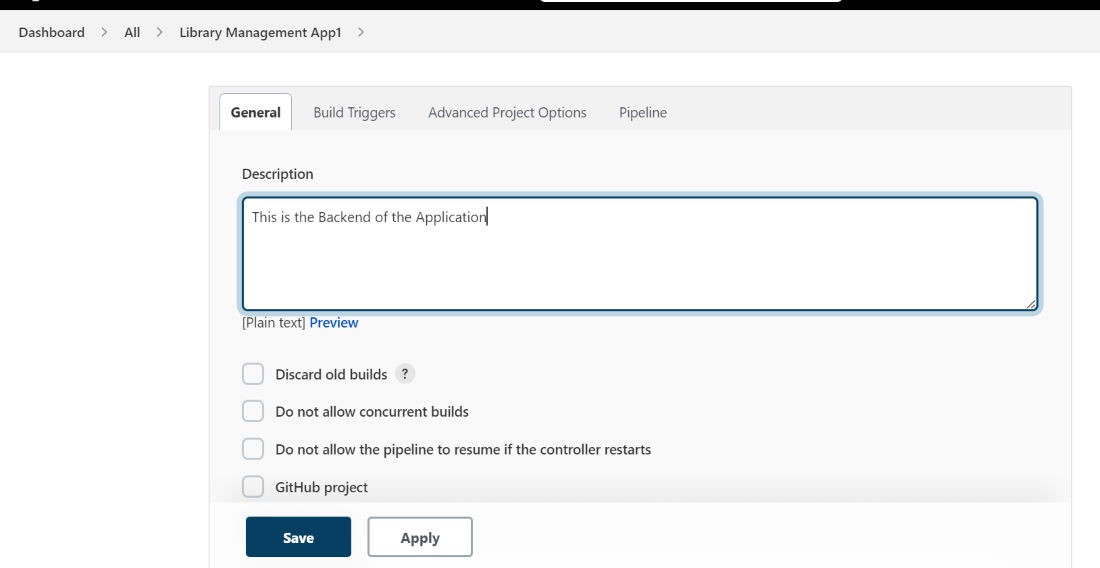
Step 2: Once logged in, the user will be redirected to the Jenkins console, Step 3: To create a new project select the option.



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



Step 5: Once redirected.



**Result:** Thus, the above program Creating a Continuous Integration (CI) pipeline using Jenkins

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | **EX.NO:5** | **Create a CD pipeline in Jenkins and deploy in Cloud** |   **Aim:** To Create a CD pipeline in Jenkins and deploy in Cloud.  Creating a Jenkins pipeline for building a Continuous Delivery (CD) pipeline and deploying. | |
| //Creating a Jenkins pipeline for building a Continuous Delivery (CD) pipeline and deploying to a cloud environment involves several steps. In this example, I'll outline a basic pipeline for deploying a simple web application to a cloud platform like AWS Elastic Beanstalk. You can adapt this example to your specific use case and cloud provider.// | |
| **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. | | | | | | | |

stage('Build') {

stages {

stage('Checkout') { steps {

checkout scm

}

}

pipeline {

agent any

steps {

// Perform build steps (e.g., compile code, package artifacts)

}

}

stage('Deploy') {

steps {

// Deploy to cloud environment (e.g., AWS Elastic Beanstalk)

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

'''

}

}

}

}

1. Replace placeholders (**<your\_access\_key>**, **<your\_secret\_key>**,

**<your\_eb\_app\_name>**, **<your\_eb\_platform>**) with your actual AWS credentials and Elastic Beanstalk information.

1. **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.

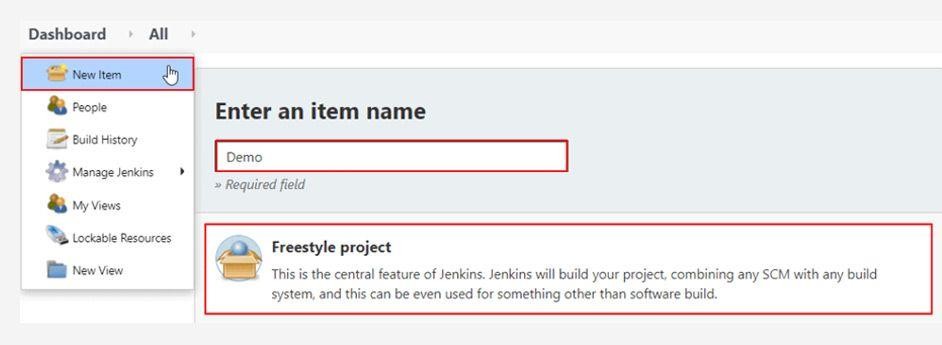
# Configure AWS Credentials in Jenkins:

1. Go to Jenkins Dashboard > Credentials > System.
2. Add new AWS credentials with access key and secret key.

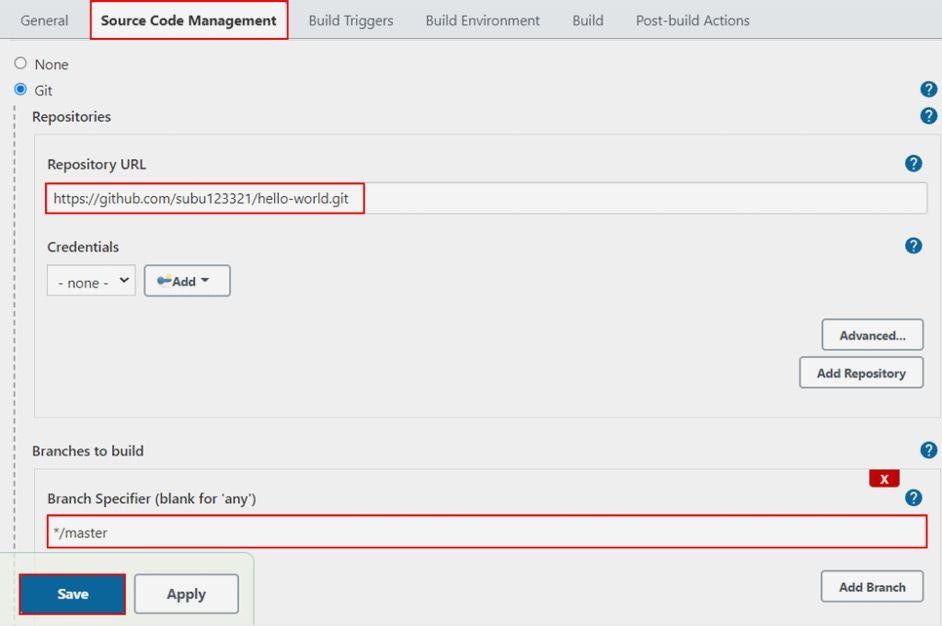
# Run the Pipeline:

1. Trigger the pipeline manually or configure webhooks to trigger it automatically on code changes.

Step1: Click on Freestyle project



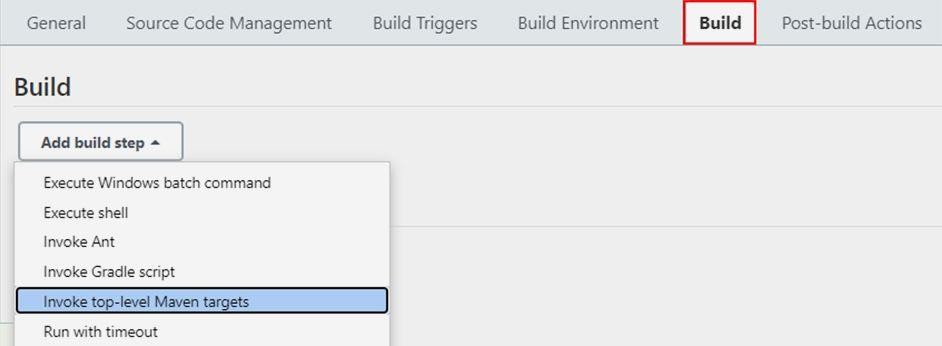
Step2: Source Code Management window.



Step3: Jenkins needs to fetch all the repository files



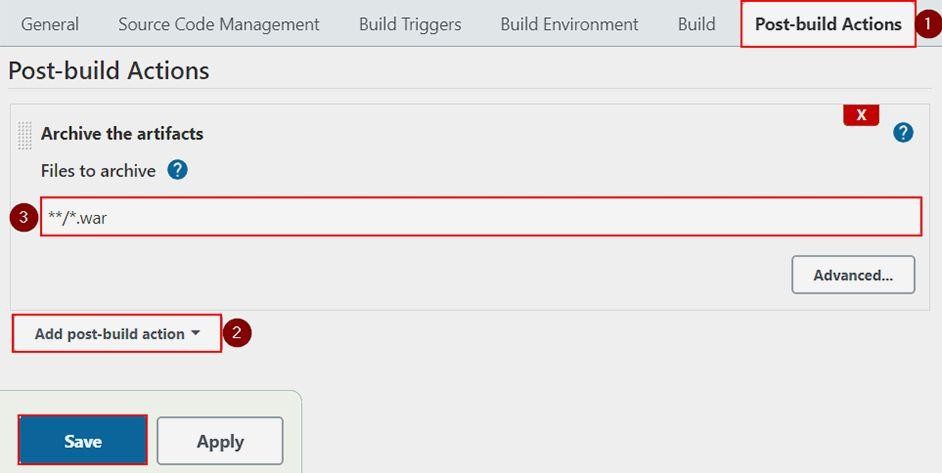
Step4: Go to the Build tab and add the build step



Step5: Drop- down file menu



Step 6: Build Now



Result: Thus, the above program was successfully completed CD pipeline in Jenkins and deploy in Cloud

|  |  |
| --- | --- |
| **EX.NO:6** | **Create an Ansible playbook for a simple web application infrastructure** |

Aim: simple Ansible playbook for setting up a basic web application

---

* 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

when: ansible\_os\_family == "Debian"

* + name: Install required packages yum:

name:

* + - nginx
    - mysql-server state: present

when: ansible\_os\_family == "RedHat"

* + name: Start and enable services service:

name:

* + - nginx
    - mysql state: started enabled: yes
  + name: Copy nginx configuration file copy:

src: files/nginx.conf

dest: /etc/nginx/nginx.conf notify:

* + - Restart Nginx
  + name: Ensure MySQL root password is set debconf:

handlers:

- name: Restart Nginx service:

name: nginx

state: restarted

- name: Create database for the web application mysql\_db:

name: mywebapp

state: present become: yes

name: mysql-server

question: mysql-server/root\_password value: "{{ mysql\_root\_password }}" vtype: password

become: yes

**Result** : Thus, the above program are successfully Create an Ansible playbook for a simple web application infrastructure

|  |  |
| --- | --- |
| **EX.NO:7** | **Build** **a** **simple** **application** **using** **gradle** |

Aim: To build a simple application using gradle

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

**build.gradle**

apply plugin: 'java' repositories {

jcenter()

}

dependencies {

compile 'com.google.guava:guava:30.1-jre'

}

sourceSets { main {

java {

srcDir 'src'

}

}

}

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

**Main.java**

**src**

**src**

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);

}

}

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

./gradlew run

Result: Thus, the above programs are build a simple application using gradle successfully completed.

|  |  |
| --- | --- |
| **EX.NO:8** | **Install Ansible and configure ansible roles and to write playbooks** |

**AIM**: To Install Ansible and configure ansible roles and to write playbooks

|  |  |  |
| --- | --- | --- |
| **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

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

For Windows:

**Ansible Directory Structure:**

Create a directory structure for your Ansible project:

|-- playbooks/

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

manage.

: This directory will hold your Ansible roles.

: This directory will contain your Ansible playbooks.

**ansible.cfg**: Ansible configuration file.

**playbooks**

**roles**







**inventory**

|-- ansible.cfg

|-- roles/

| |-- hosts

|-- inventory/

ansible\_project/

Plaintext

**Writing an Ansible Role:**

Inside the directory, create a new role: ansible-galaxy init my\_role

This will create a basic directory structure for your role.

**roles**

**Writing an Ansible Playbook:**

Create a playbook inside the

**my\_playbook.yml**:

directory. For example, create a file named

---

- name: My Ansible Playbook hosts: your\_target\_host\_group become: yes

roles:

- my\_role

Replace

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 Install Ansible and configure ansible roles and to write playbooks in suceessfully.

**your\_target\_host\_group**

**playbooks**

**VIVA QUESTIONS:**

**UNIT-1**

1. What is DevOps?
2. How is DevOps different from Agile?
3. Explain the key principles of DevOps.
4. What are some of the main challenges in implementing DevOps?
5. How do DevOps practices improve software delivery?
6. What is a version control system, and why is it important in DevOps?
7. Explain the difference between Git and SVN.
8. What are Git branching strategies?
9. How do you resolve merge conflicts in Git?
10. What is a pull request, and how is it reviewed in a DevOps process?
11. What is Continuous Integration, and why is it important?
12. Explain how Jenkins works in a CI pipeline.
13. What are some common CI tools apart from Jenkins?
14. How do you handle build failures during CI?
15. What are webhooks in the context of CI/CD?
16. What is the difference between Continuous Delivery and Continuous Deployment?
17. What are the key stages of a CD pipeline?

18.How do you implement rollbacks in a CD pipeline?

19.Explain the concept of canary deployment.

20.What are blue-green deployments?