

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:

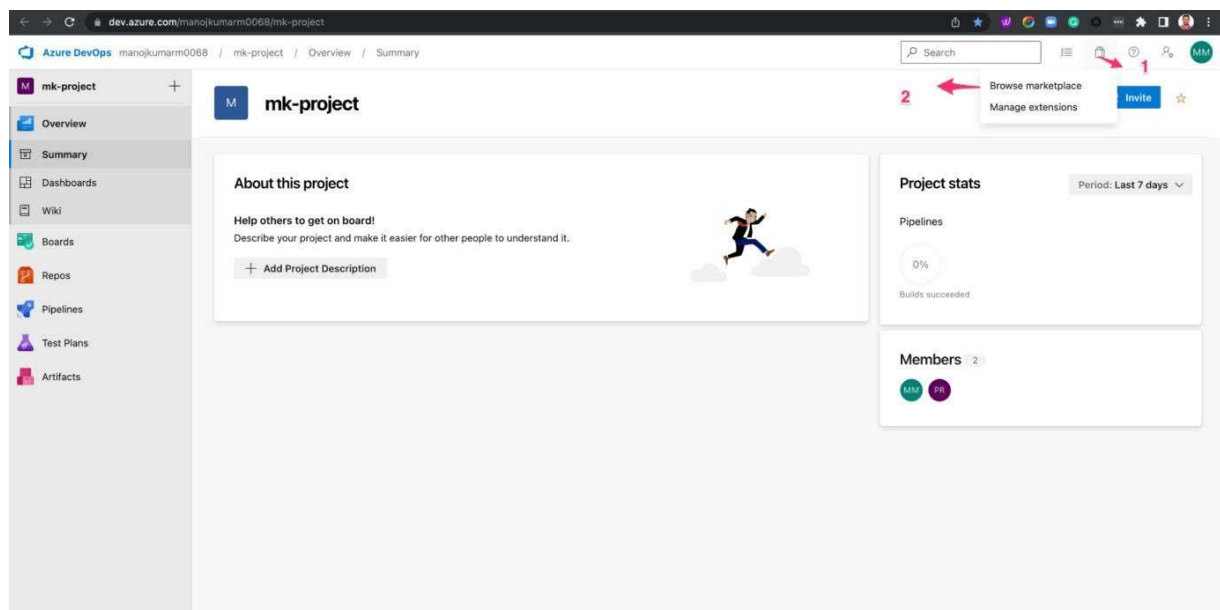
- Configure triggers to automate pipeline runs. For example, you can trigger the pipeline whenever changes are pushed to specific branches.

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

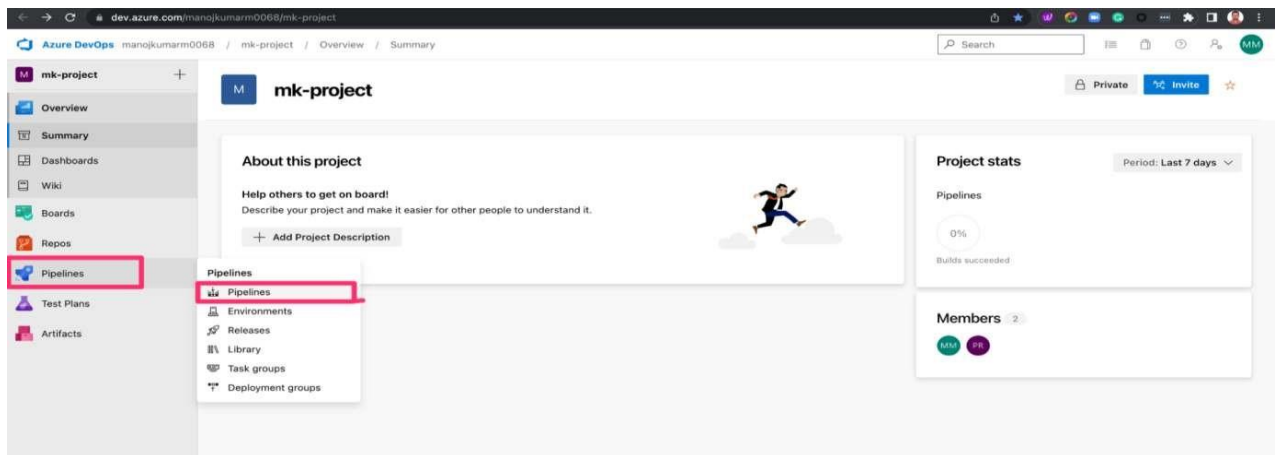
Step 1: Organization is created



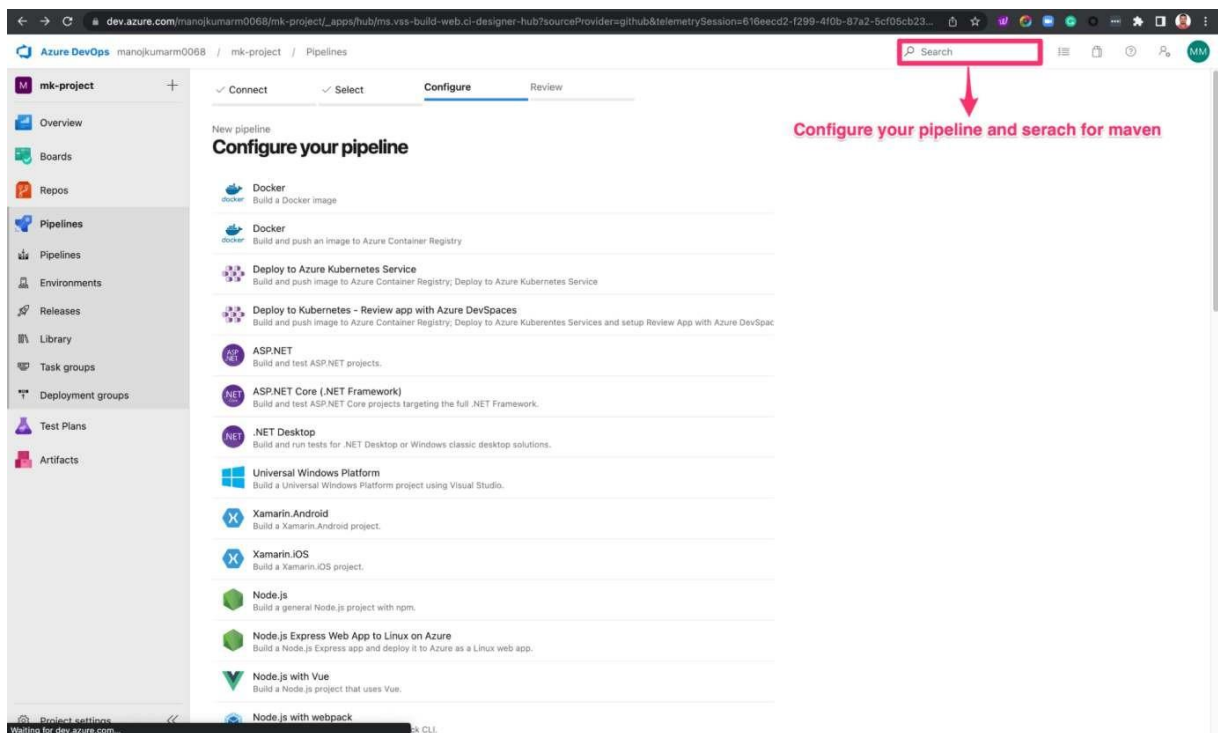
Step 2: Organization by clicking the new project button



Step3: Create Pipeline



Step4: Configure new project.



Results:

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

RUN REGRESSION TESTS USING MAVEN BUILD PIPELINE IN AZURE

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.

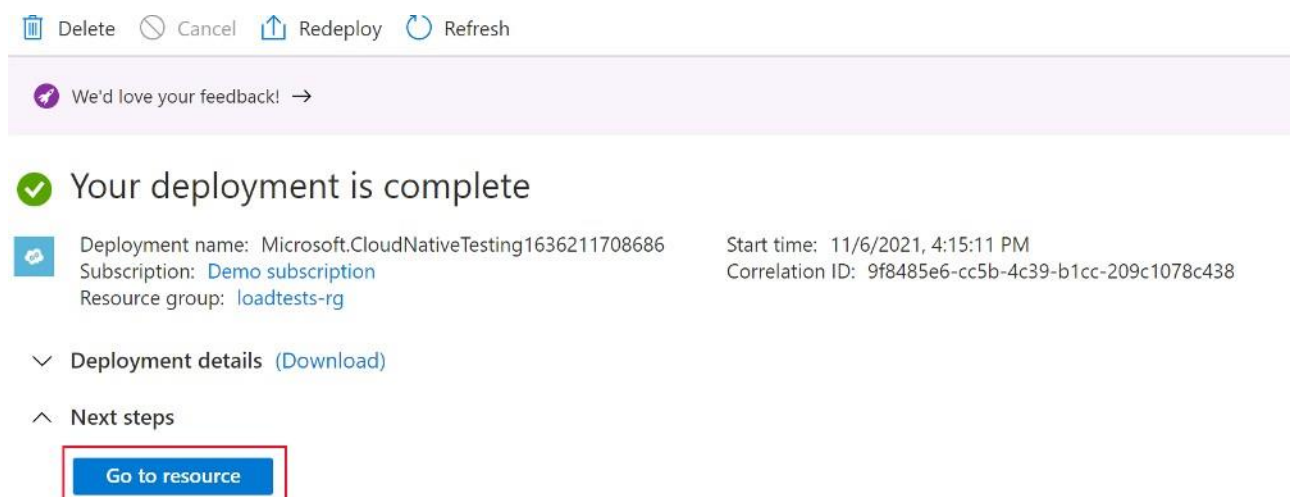
Step 7: Monitor the Build

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)

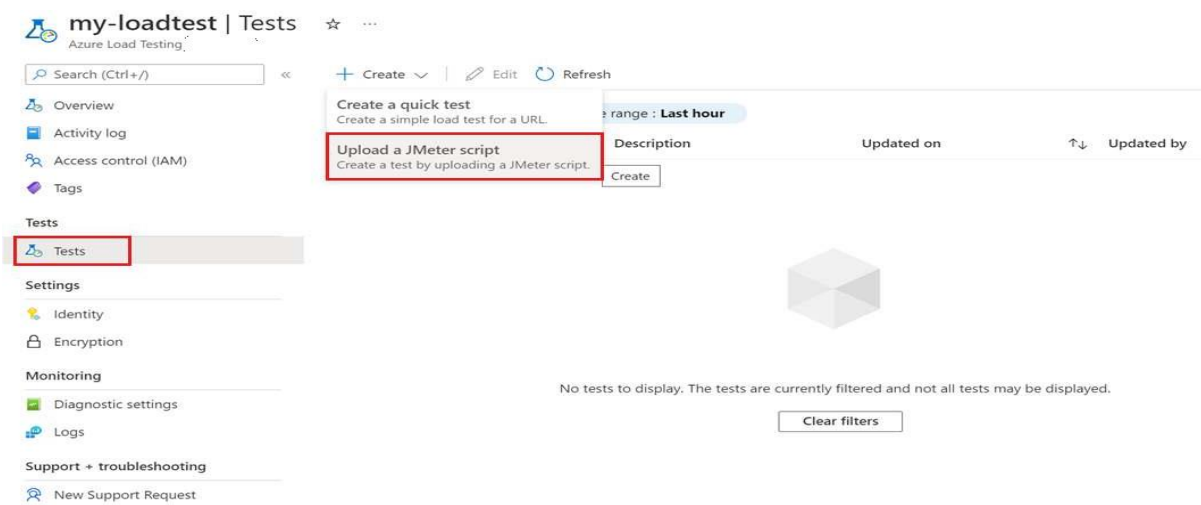
1. You can set up triggers to automatically run the build pipeline when changes are pushed to specific branches.
2. Configure branch filters and triggers according to your needs.

Step1: new resource, select go to resource.



The screenshot shows the Azure DevOps deployment interface. At the top, there are action buttons: Delete, Cancel, Redeploy, and Refresh. Below these is a feedback banner that says "We'd love your feedback! →". The main status is "Your deployment is complete" with a green checkmark icon. Below this, deployment details are listed: Deployment name: Microsoft.CloudNativeTesting1636211708686, Subscription: Demo subscription, and Resource group: loadtests-rg. To the right, the Start time is 11/6/2021, 4:15:11 PM and the Correlation ID is 9f8485e6-cc5b-4c39-b1cc-209c1078c438. There are two expandable sections: "Deployment details (Download)" and "Next steps". Under "Next steps", there is a blue button labeled "Go to resource" which is highlighted with a red rectangle.

Step2: Azure Load Testing resource, select Tests



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

Home > my-loadtest | Tests >

Create test

Basics Test plan Parameters Load Test criteria Monitoring Review + create

Create and run a JMeter test. Configure your test and then select Review + create. [Learn more](#)

Test details

Provide a test name and a description. Test name and description will help you identify a test in the list of tests created in this resource.

Test name *

Test description

Run test after creation ☒

[Previous](#) [Next](#) [Review + create](#)

Result:

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

INSTALL JENKINS IN CLOUD

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:

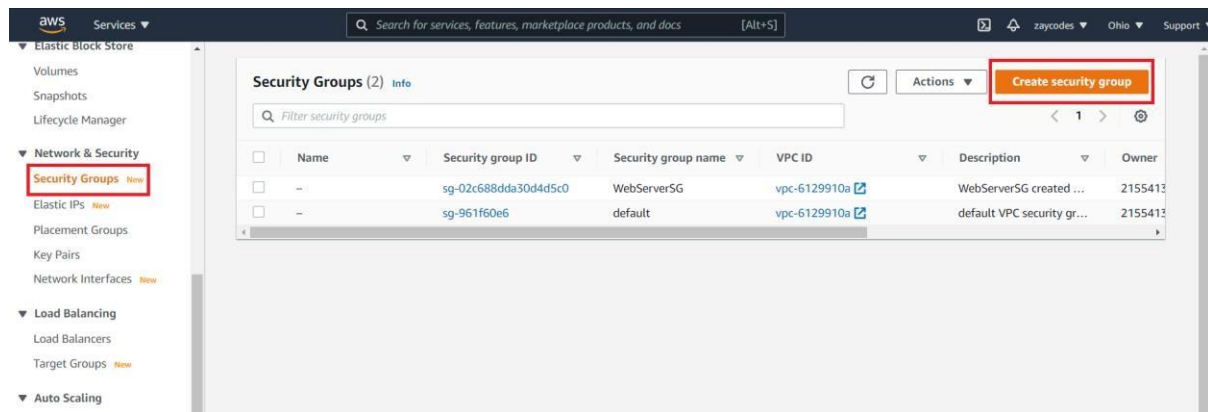
- Configure Jenkins settings according to your preferences.
- Implement security measures, such as setting up authentication, authorization, and HTTPS.

Step1: Enter the IP Address in url:103.43.200.163

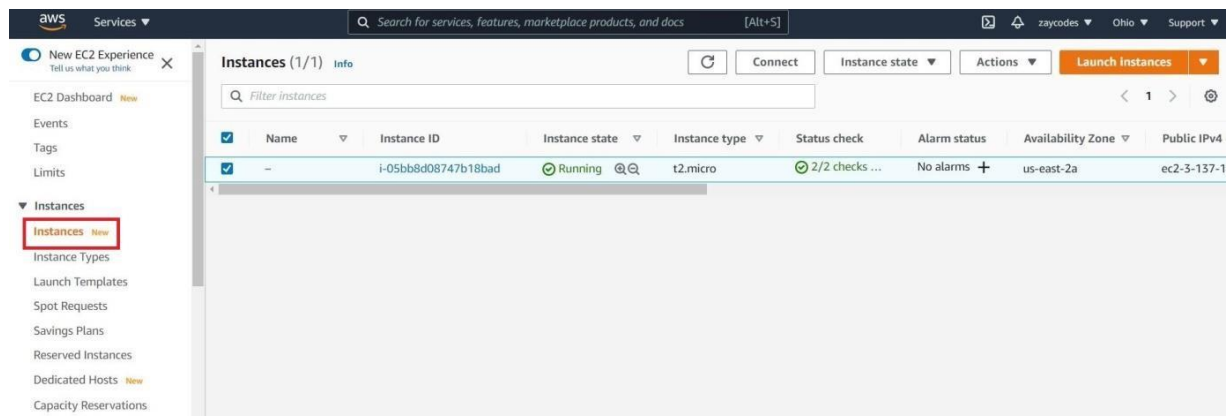
Step2: Sign in to the [AWS Management Console](#).

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

Date:

CREATE CI PIPELINE USING JENKINS

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 {  
                build  
            }  
        }  
    }  
}
```

```

    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.

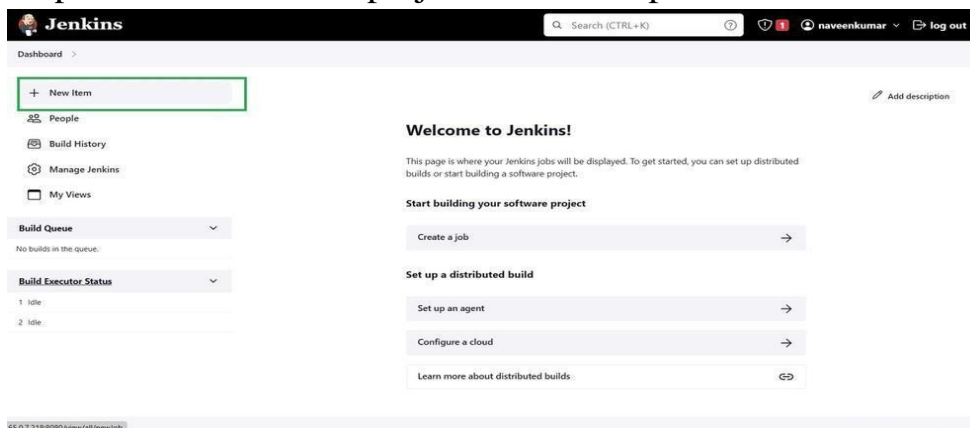
6. Adding Additional Steps:

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

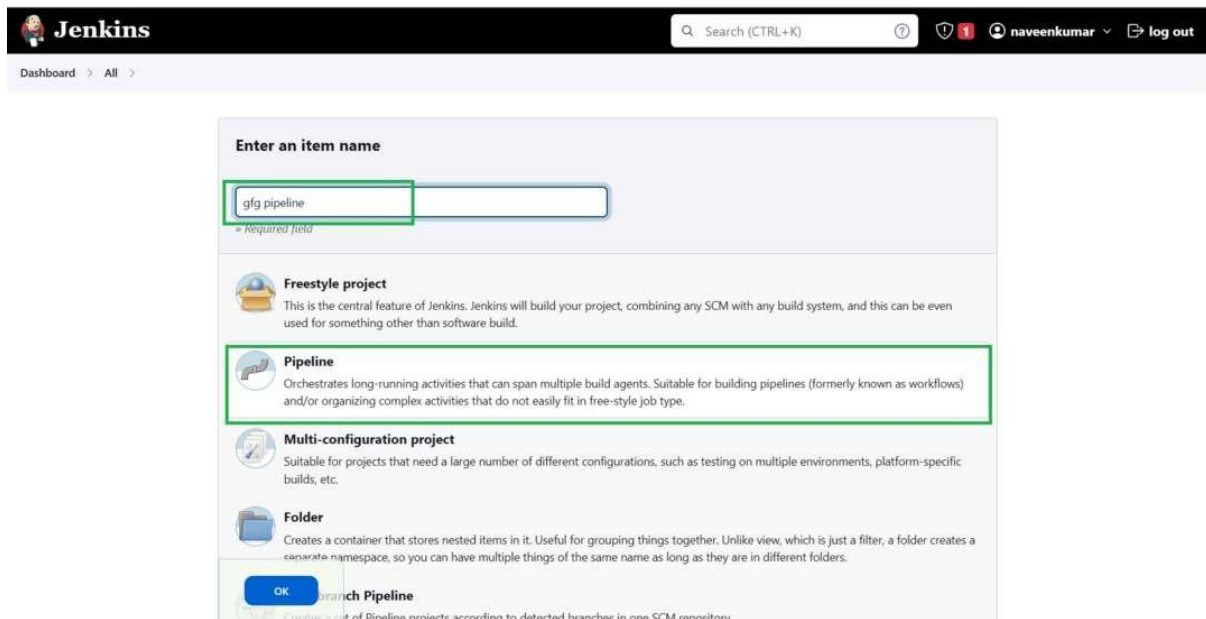
Step 1: Login into your Jenkins account as shown below.

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

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

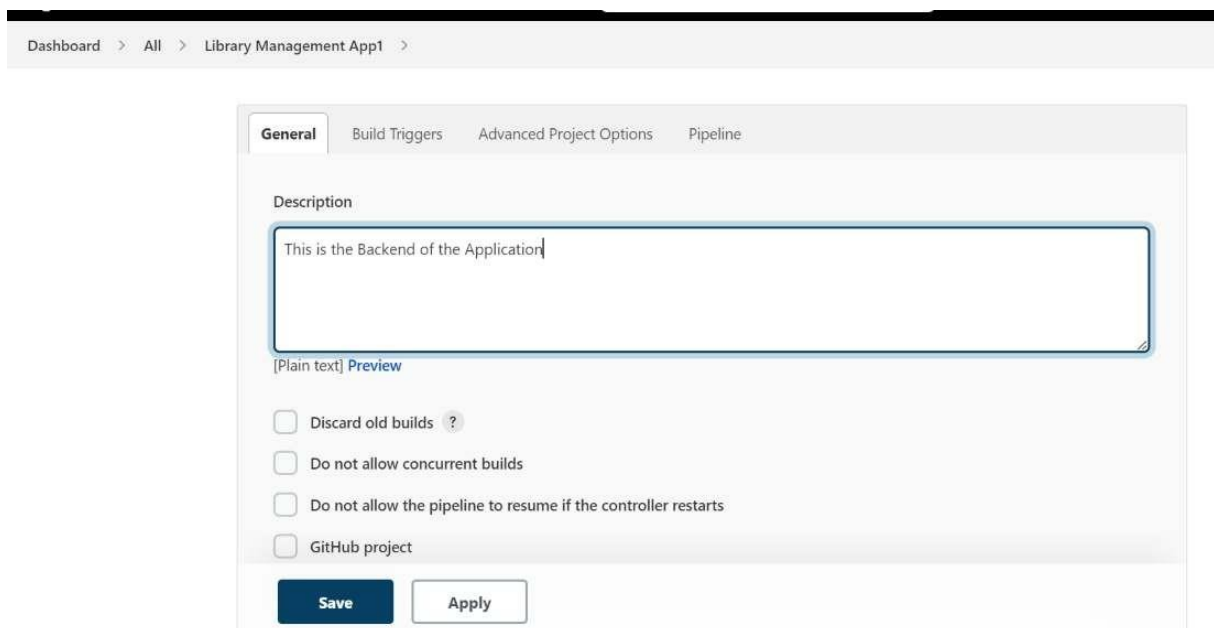


Step 4: Now a list of options will be visible on the screen.



The image shows the Jenkins 'Enter an item name' dialog box. At the top, there's a search bar with the text 'Search (CTRL+K)'. Below it, the breadcrumb 'Dashboard > All >' is visible. The main section is titled 'Enter an item name' and contains a text input field with 'gfg pipeline' entered. Below the input field, there's a red asterisk and the text 'Required field'. Underneath, there are four options: 'Freestyle project', 'Pipeline', 'Multi-configuration project', and 'Folder'. The 'Pipeline' option is highlighted with a green border. At the bottom, there's a blue 'OK' button and a link to 'branch Pipeline'.

Step 5: Once redirected.



The image shows the Jenkins configuration page for 'Library Management App1'. The breadcrumb 'Dashboard > All > Library Management App1 >' is at the top. The page has four tabs: 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline'. The 'General' tab is active. It contains a 'Description' field with the text 'This is the Backend of the Application'. Below the description field, there's a '[Plain text] Preview' link. Underneath, there are four checkboxes: 'Discard old builds', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the controller restarts', and 'GitHub project'. At the bottom, there are two buttons: 'Save' and 'Apply'.

Result:

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

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>

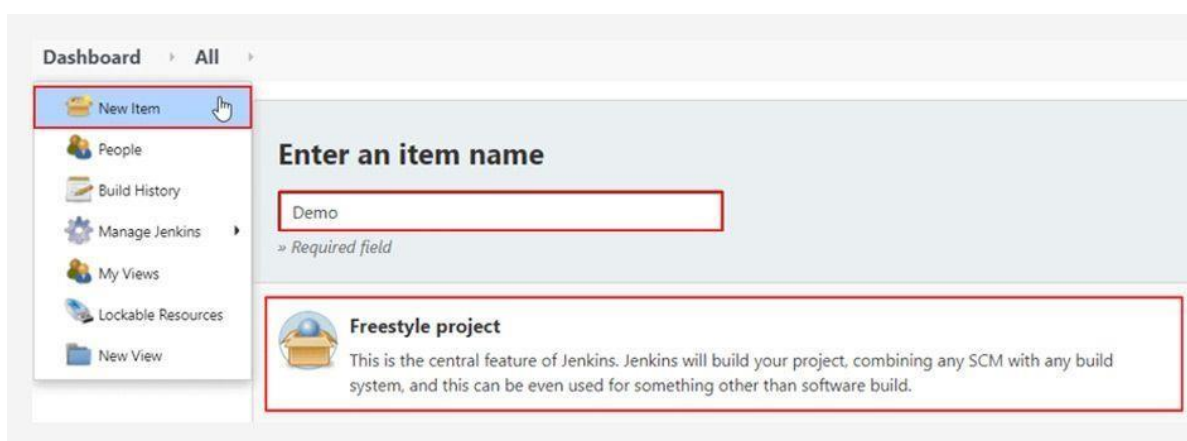
aws configure set aws_secret_access_key <your_secret_key>

eb init <your_eb_app_name> -p <your_eb_platform>

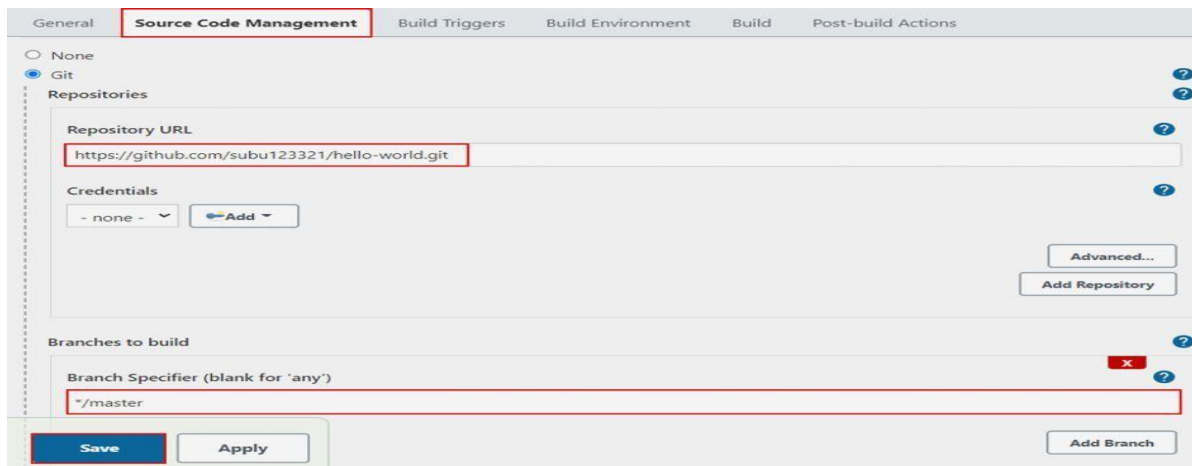
eb deploy"}}
```

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.

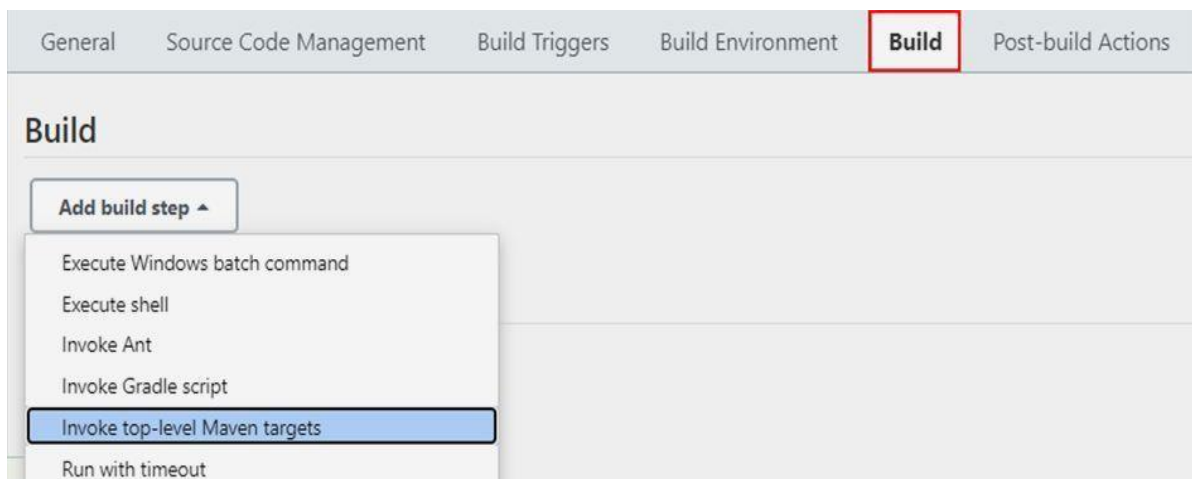


The screenshot shows the 'Source Code Management' tab in the Jenkins configuration interface. The 'Git' radio button is selected under the 'Repositories' section. The 'Repository URL' field contains 'https://github.com/subu123321/hello-world.git'. The 'Credentials' dropdown is set to '- none -'. Below this, the 'Branches to build' section has a 'Branch Specifier (blank for 'any')' field containing '*/master'. At the bottom, there are 'Save', 'Apply', and 'Add Branch' buttons.

Step3: Jenkins needs to fetch all the repository files



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



The screenshot shows the 'Build' tab in the Jenkins configuration interface. The 'Add build step ^' button is clicked, and a dropdown menu is open. The menu options are: 'Execute Windows batch command', 'Execute shell', 'Invoke Ant', 'Invoke Gradle script', 'Invoke top-level Maven targets' (which is highlighted with a blue box), and 'Run with timeout'.

Step5: Drop- down file menu

Invoke top-level Maven targets

Maven Version

maven_home

Goals

clean compile package

Advanced...

Step 6: Build Now

General Source Code Management Build Triggers Build Environment Build Post-build Actions 1

Post-build Actions

Archive the artifacts

Files to archive ?

3 **/*.war

Advanced...

Add post-build action 2

Save Apply

Result:

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

Ex. No: 6

Date:

CREATE AN ANSIBLE PLAYBOOK FOR A SIMPLE WEB APPLICATION INFRASTRUCTURE

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.

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

Date:

INSTALL ANSIBLE AND CONFIGURE ANSIBLE ROLES AND TO WRITE PLAYBOOKS

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.