

## Index Page

Sl. No.	Date	Experiment Name	Page No.	Mark	Signature
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.		Build a simple application using Gradle			
7.		Create an Ansible playbook for a simple web application infrastructure			
8.		Install Ansible and configure ansible roles and to write playbooks			

**Date:**

## **Ex No: 1**

### **Create Maven Build pipeline in Azure**

#### **Aim**

To create a maven project and build a pipeline in Azure

#### **Procedure**

##### **1. Install Java and Maven and set the system environment variables**

- a. Download maven and extract it
- b. Add JAVA\_HOME and MAVEN\_HOME in environment variable
  - i. Right click on **MyComputer** -> **properties** -> **Advanced System Settings** -> **Environment variables** -> **click new button.**
  - ii. Add **MAVEN\_HOME** in variable name and path of maven in variable value. click on **OK** button. Similarly create **JAVA\_HOME**
- c. Add maven and java path in environment variable
  - i. Click on new tab and then set the path of maven and java till bin directory. If it is set, edit the path and append the path of maven and java
- d. Verify Maven and Java
  - open the command prompt and write:
    - i. `mvn -version`
    - ii. `java -version`

##### **2. Create a github account**

1. Go to <https://github.com/join> in a web browser
2. Enter your personal details.
3. Click Verify to start the verification puzzle.
4. Click the green Create account button.
5. Verify your email by entering the code.
6. Select your preferences and click Continue.
7. Note the types of plans offered by GitHub.
8. Select the free plan.

##### **3. Create an Azure student login and enter into Azure DevOps**

###### **Create an Azure Student Account:**

- Visit the Azure for Students page.
- Sign up using your university or school email. You'll need to renew this account each year while you're a student.
- You'll receive a USD 100 credit to use with Azure services. No credit card is required, and you get 12 months of free Azure services<sup>1</sup>.

###### **Sign Up for Azure DevOps:**

- Go to Azure DevOps.
- Choose either a Microsoft account or a GitHub account to sign up.
- If you don't have a Microsoft account, create one.
- Select Start free and enter your account credentials.
- Azure DevOps will create an organization for you.
- It will also create a project named after your newly created Microsoft account.
- If you signed up with an existing Microsoft account, you'll need to create a project next.

###### **Access Azure DevOps:**

- Sign in to your organization anytime at `dev.azure.com{Your_Organization}`.

#### **4. Install a Java IDE (Eclipse, NetBeans, IntelliJ IDEA, VS code)**

#### **5. Develop a Java Maven Project and build the application in the local server**

##### **Create a New Maven Project:**

- Open VS Code.
- Install the Maven for Java extension by searching for it in the Extensions view (Ctrl+Shift+X).
- Create a new Maven project using one of the following methods:
- Use the command palette (Ctrl+Shift+P) and type “Maven: Create Maven Project.”
- Right-click on a folder and select “Generate from Maven Archetype.”
- Choose the “maven-archetype-quickstart” archetype.
- Modify the pom.xml file to specify your project details and dependencies.

##### **Write Your Java Code:**

- Create your Java classes in the src/main/java directory.
- Define your application logic, classes, and methods.

##### **Build the Project:**

- Open the integrated terminal in VS Code (Ctrl+`).
- Run the following command to build the project:  
mvn clean install
- This will compile your code, run tests, and package the application into a JAR file.

##### **Run the Application Locally:**

- After successful build, use the following command to start a local server:  
java -jar target/your-artifact-id-version.jar
- Replace your-artifact-id-version with the actual name of your JAR file.

##### **Access Your Application:**

- Open a web browser and navigate to <http://localhost:8080> (or the port specified in your application).
- You should see your Java application running locally.

#### **6. Push the java maven project into Git Repository**

##### **Initialize Git Repository:**

- Open a terminal or command prompt.
- Navigate to the root directory of your Maven project using the cd command.
- Initialize a new Git repository:  
git init

##### **Add and Commit Your Files:**

- Use the following commands to stage all files and commit them:  
git add .  
git commit -m "Initial commit"

##### **Create a New Repository on GitHub:**

- Go to GitHub and log in (or sign up if you don't have an account).
- Click the “+” icon in the top right corner and select “New repository.”
- Give your repository a name, choose visibility (public or private), and create it.
- Link Your Local Repository to GitHub:
- Copy the HTTPS URL of your newly created GitHub repository.
- In your terminal, add the remote origin (replace <me> with your GitHub username and <myrepo> with your repository name):  
git remote add origin <https://github.com/<me>/<myrepo>.git>

##### **Push your code to the master branch on GitHub:**

- git push -u origin master

#### **7. Create Azure DevOps organization and import or clone the git repo of the java maven project**

##### **Sign in to Azure DevOps:**

- Log in to your Azure DevOps organization.
- Select Your Project:
- Choose the project where your Java Maven project resides.
- Create a New Git Repo in Your Project:
- Open the Repos page in your project by browsing to `dev.azure.com/OrganizationName`.
- Hover over the name of your project and select the Repos icon.
- From the repo drop-down, select New repository.
- Verify that Git is the repository type and enter a name for your new repository.
- Optionally, add a README and create a .gitignore file.
- Select Create.

## 8. Create a pipeline in Azure and build the project

### Create a New Pipeline:

- Navigate to Pipelines (usually on the left side menu).
- Click on New pipeline.
- Select Azure Repos Git as the source location.

### Choose Your Repository:

- Select the Git repository where your Java Maven project is hosted.
- Configure the Pipeline:

### You'll be guided through a wizard:

- First, choose the Maven, Gradle, or Ant template based on your build preference.
- Save your configuration.
- Commit an `azure-pipelines.yml` file to your repo.
- Save and run the pipeline.

### Watch Your Pipeline in Action:

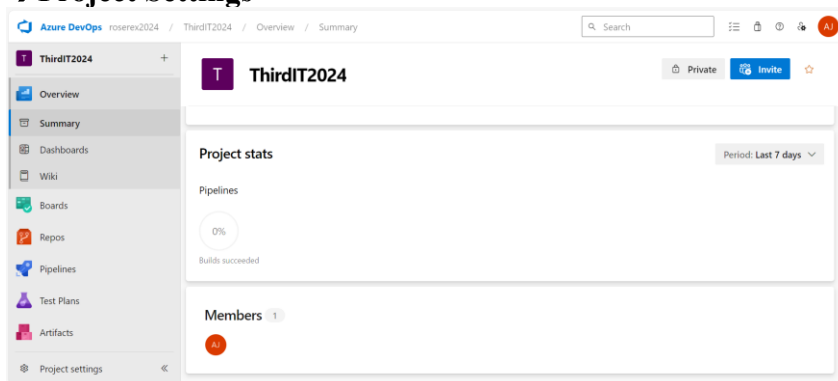
- If you want to see your pipeline executing, select the build job.

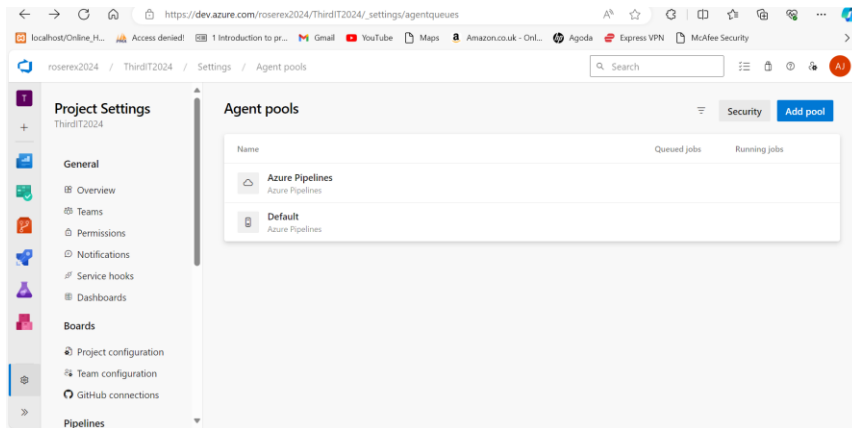
### Customize Your Pipeline:

- When you're ready to make changes, go to the Pipelines page, select your pipeline, and edit the `azure-pipelines.yml` file.

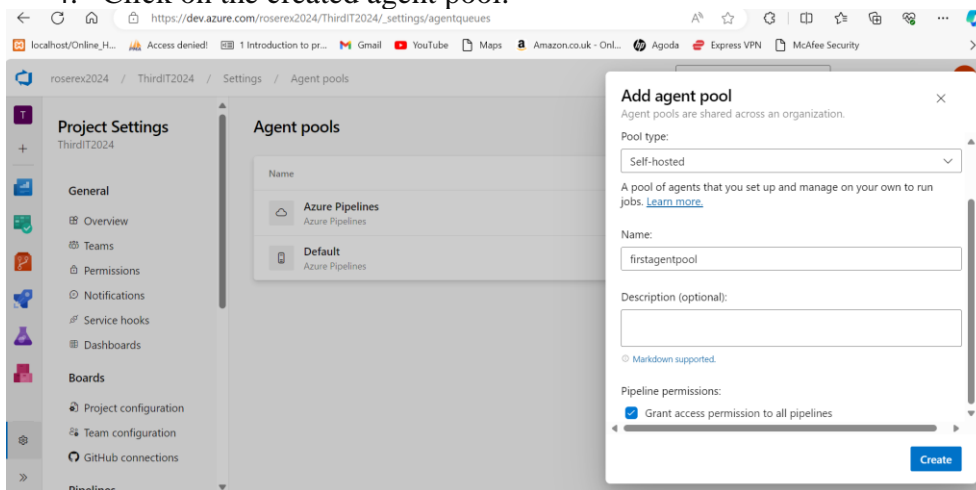
## Output

### →Project Settings





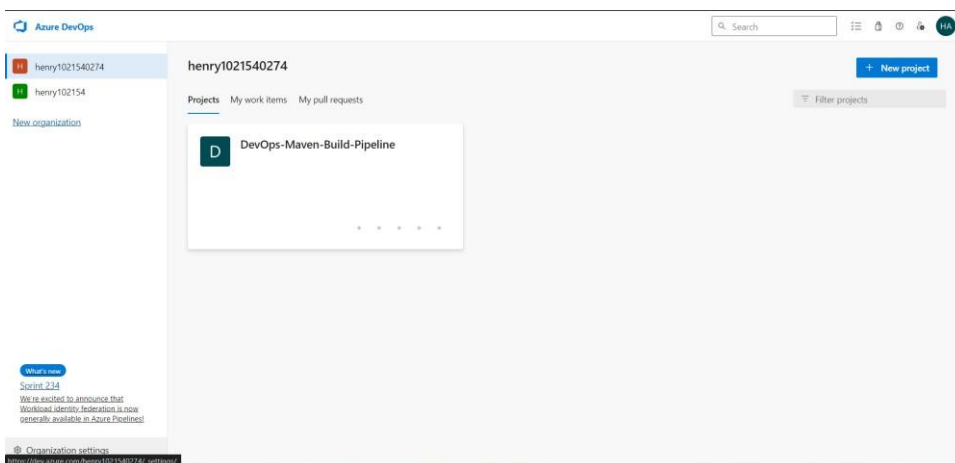
1. Choose Agent pools.
2. Click on Add pool.
3. Select Pool-type as Self-hosted, give it a name and click on Create.
4. Click on the created agent pool.



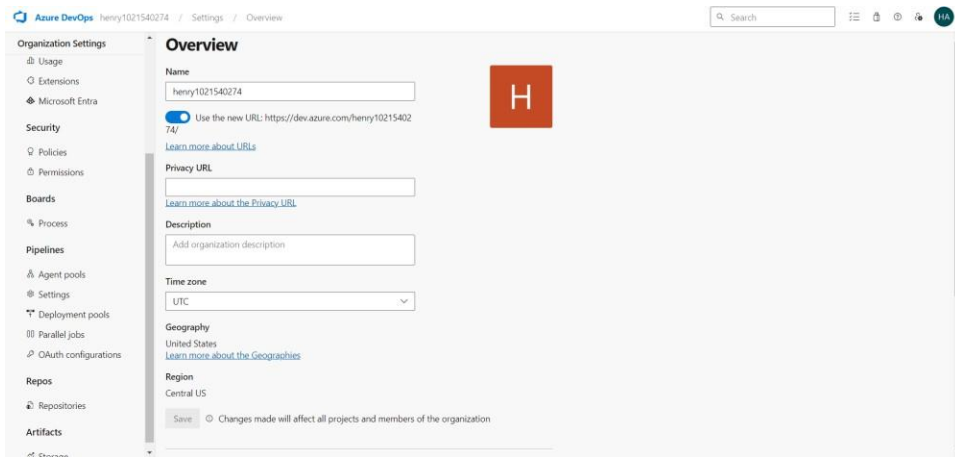
## Conclusion

To create an Agent:

- Click on organization setting



## - Create a new pool in Agent Pools



The screenshot shows the 'Overview' page of the 'Organization Settings' in Azure DevOps for the organization 'henry1021540274'. The left sidebar contains a navigation menu with categories: Organization Settings (Usage, Extensions, Microsoft Entra), Security (Policies, Permissions), Boards, Pipelines (Agent pools, Settings, Deployment pools, Parallel jobs, OAuth configurations), Repos (Repositories), and Artifacts. The main content area is titled 'Overview' and features a large orange profile picture with the letter 'H'. Below the profile picture, there are several configuration sections: 'Name' (text input with 'henry1021540274'), 'Use the new URL' (toggle switch, currently on), 'Privacy URL' (text input), 'Description' (text input with placeholder 'Add organization description'), 'Time zone' (dropdown menu showing 'UTC'), 'Geography' (text input with 'United States'), and 'Region' (text input with 'Central US'). At the bottom of the main content area, there is a 'Save' button and a warning message: 'Changes made will affect all projects and members of the organization'.

**Organization Settings** henry1021540274 / Settings / Overview

**Overview**

**Name**  
henry1021540274

☒ Use the new URL: <https://dev.azure.com/henry1021540274/>  
[Learn more about URLs](#)

**Privacy URL**

[Learn more about the Privacy URL](#)

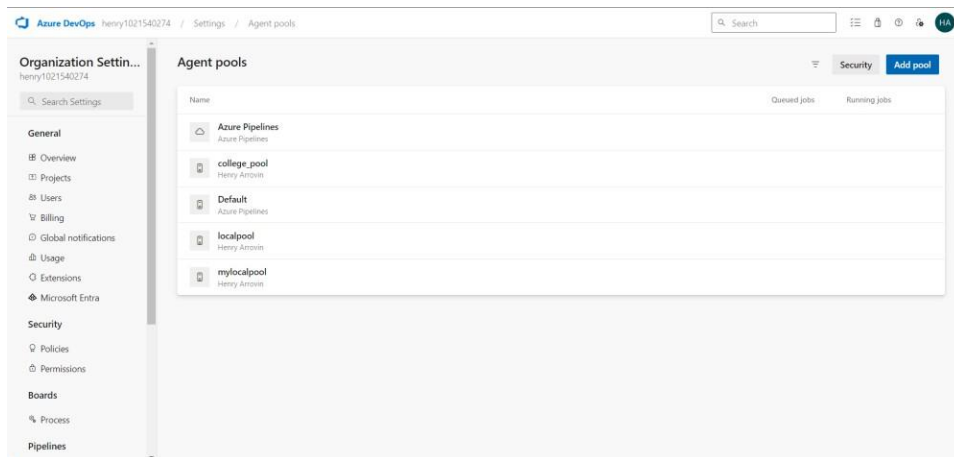
**Description**

**Time zone**  
UTC

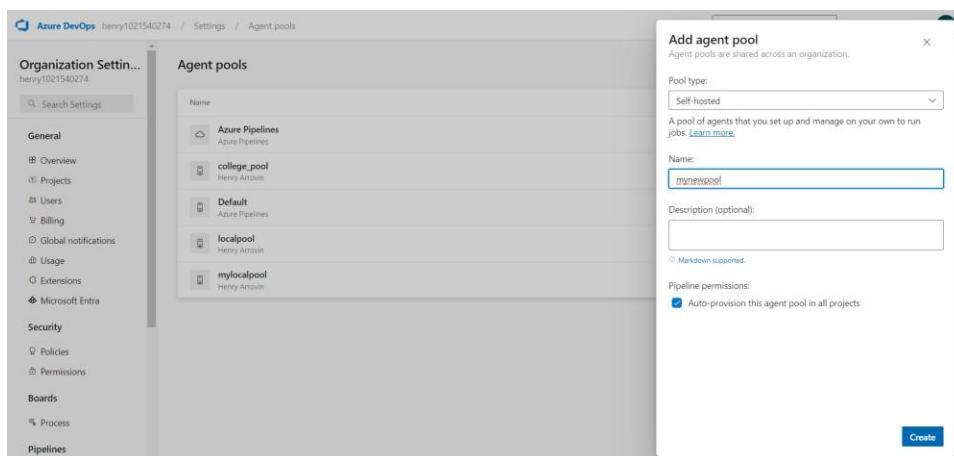
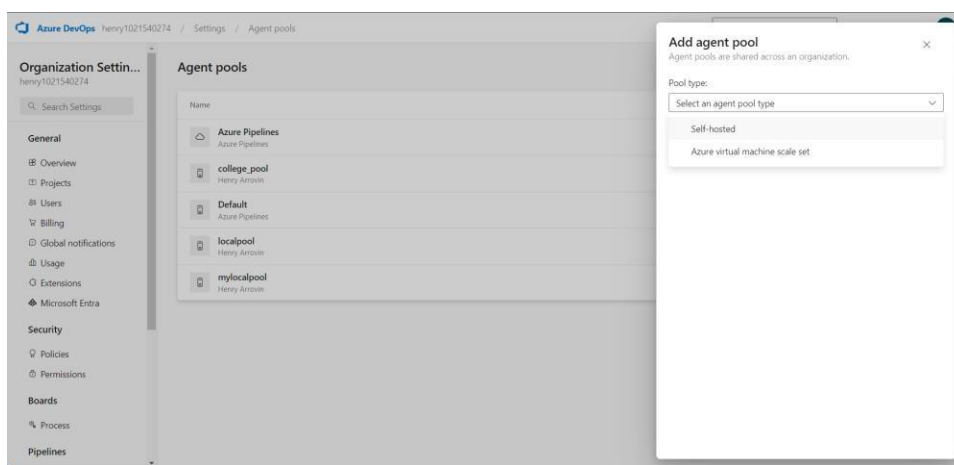
**Geography**  
United States  
[Learn more about the Geographies](#)

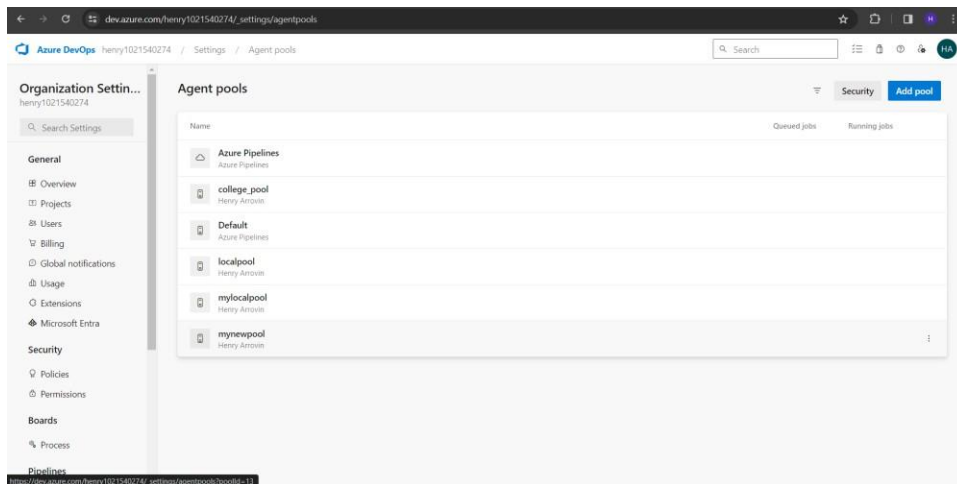
**Region**  
Central US

**Save** ☐ Changes made will affect all projects and members of the organization

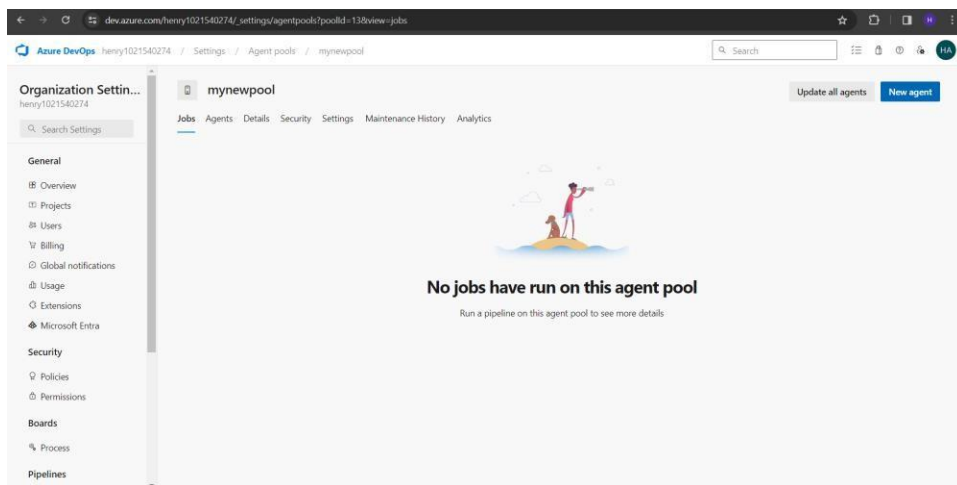


## - Create a new Agent

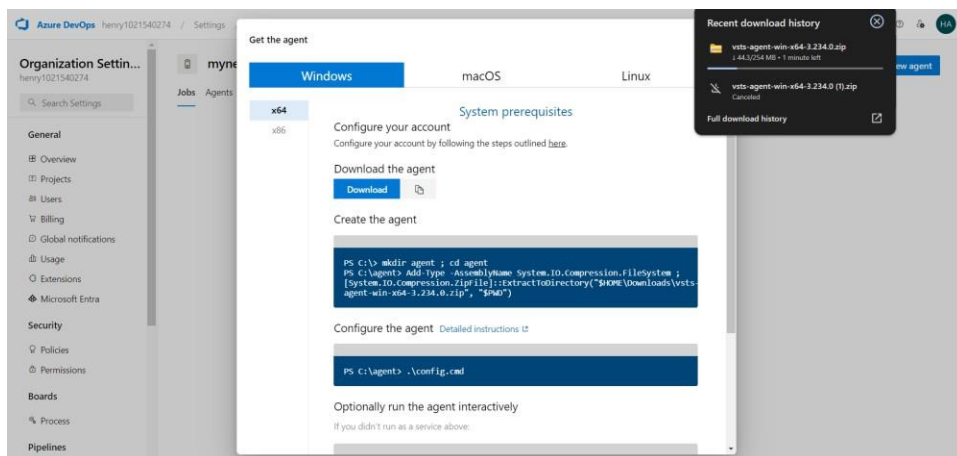




- Go inside the pool created and create a new agent



- Follow the steps shown in the screen





```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

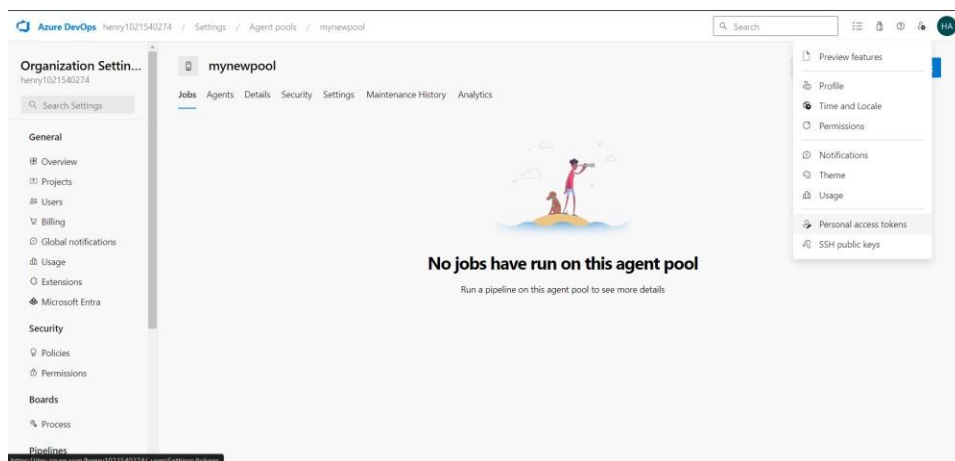
PS C:\WINDOWS\system32> cd ..\..
PS C:\> mkdir agent

Directory: C:\

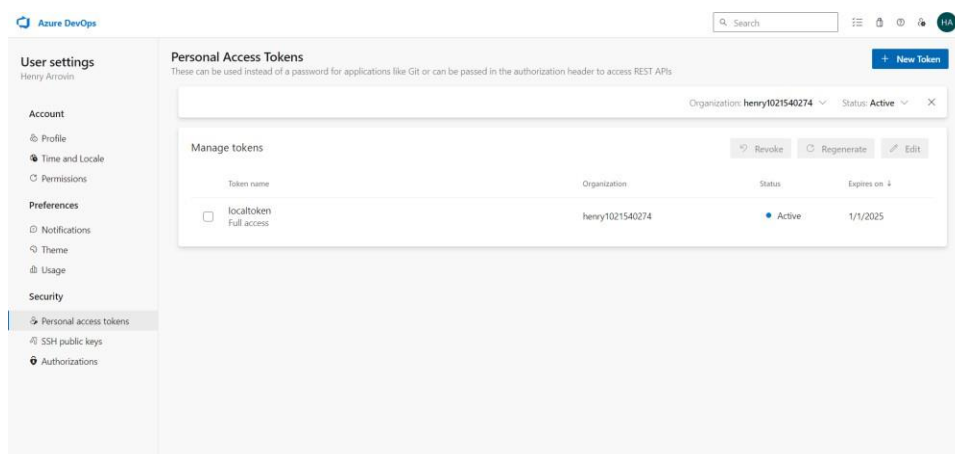
Mode                LastWriteTime         Length Name
----                -
d-----          3/4/2024 10:45 AM             agent

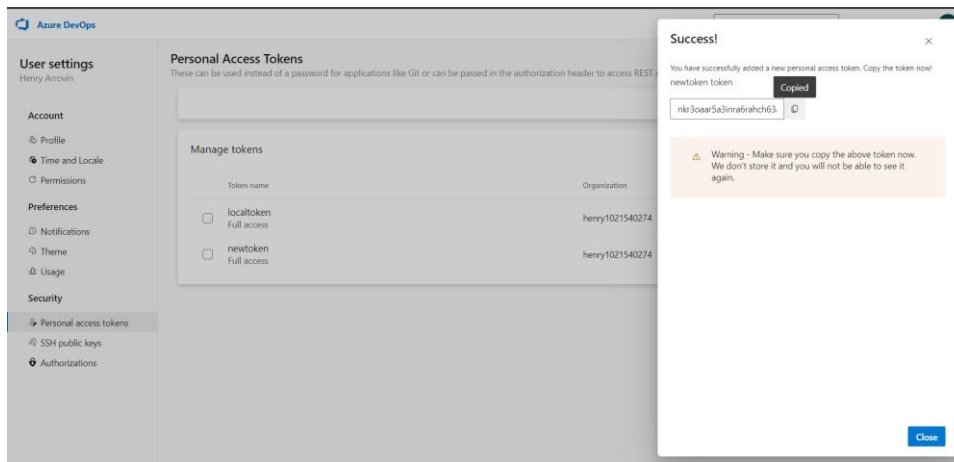
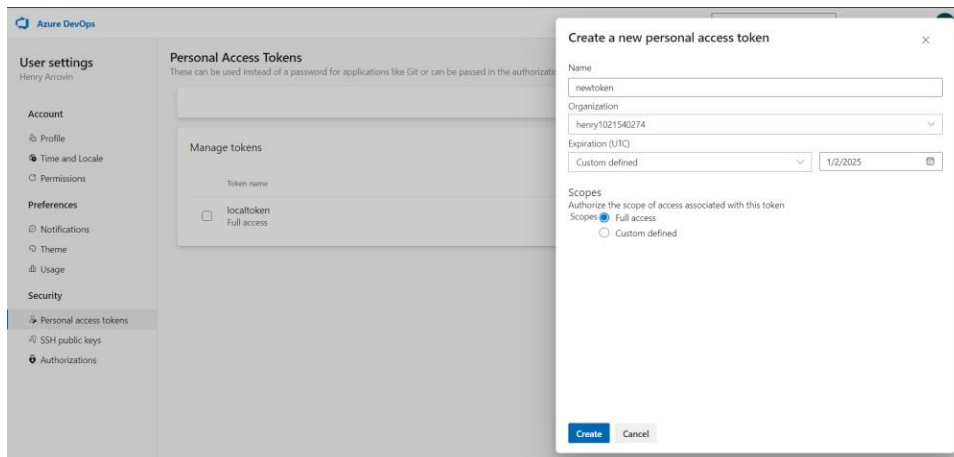
PS C:\> cd agent
PS C:\agent> Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compression.ZipFile]::ExtractToDirectory("$HOME\Downloads\vsts-agent-win-x64-3.234.0.zip", "$PWD")
PS C:\agent>
```

- Give the URL till organization name
  - For PAT follow the below steps
- Click on Personal Access Tokens



- Create a new token





```

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> cd ..\..
PS C:\> mkdir agent

Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----          3/4/2024 10:45 AM             agent

PS C:\> cd agent
PS C:\agent> Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compression.ZipFile]::ExtractToDirectory("$env:VARIABLES\agent-win-x64-3.234.0.zip", ".")
PS C:\agent> .\config.cmd

  agent v3.234.0 (commit 21ca259)

-- Connect:
Enter server URL > https://dev.azure.com/henry1021540274
Enter authentication type (press enter for PAT) >
Enter personal access token > *****
Connecting to server ...

-- Register Agent:
Enter agent pool (press enter for default) > mynewpool
Enter agent name (press enter for L9P70P-3AJ4S221) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2024-03-04 05:20:15Z: Settings Saved.
Enter run agent as service? (Y/N) (press enter for N) > n
Enter configure autologon and run agent on startup? (Y/N) (press enter for N) > n
PS C:\agent>
  
```

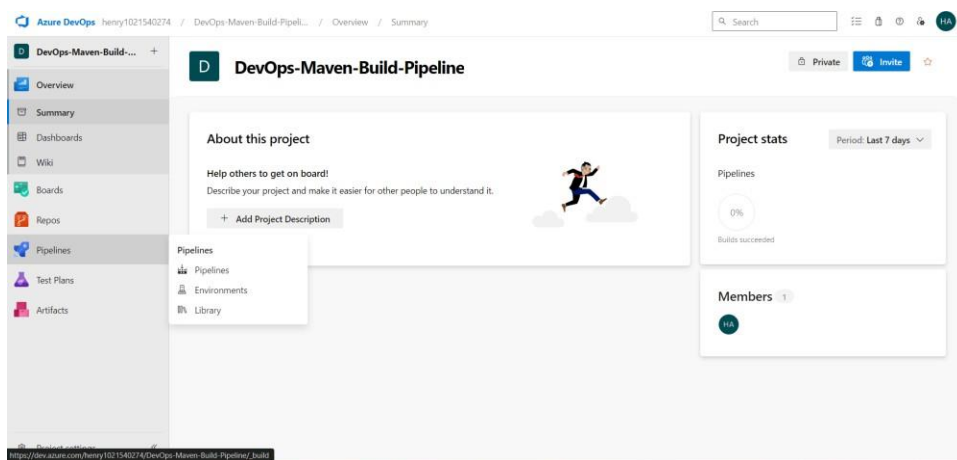
- Copy the token and paste it

Name	Date modified	Type	Size
agent	3/4/2024 10:58 AM	File folder	

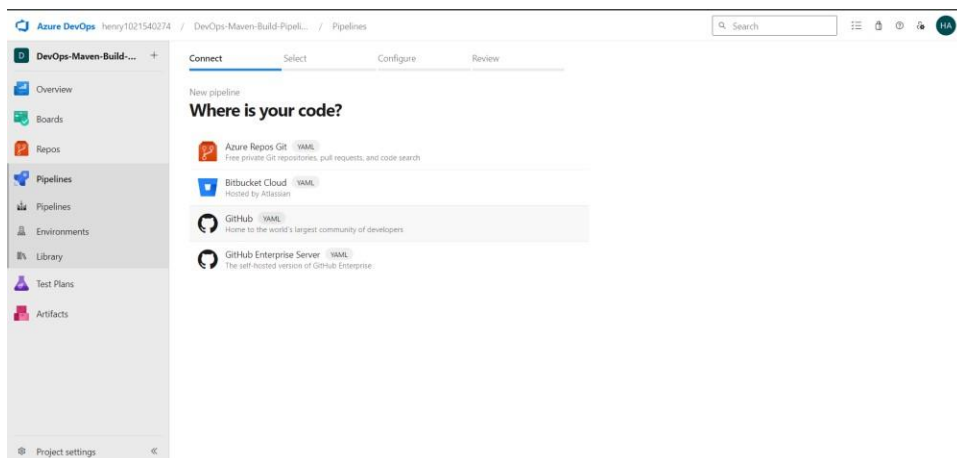
Name	Date modified	Type	Size
📁 _diag	3/4/2024 10:58 AM	File folder	
📁 _work	3/4/2024 10:58 AM	File folder	
📁 bin	3/4/2024 10:46 AM	File folder	
📁 externals	3/4/2024 10:46 AM	File folder	
📄 config.cmd	2/8/2024 1:25 PM	Windows Comma...	3 KB
📄 run.cmd	2/8/2024 1:25 PM	Windows Comma...	4 KB

Now to create a pipeline

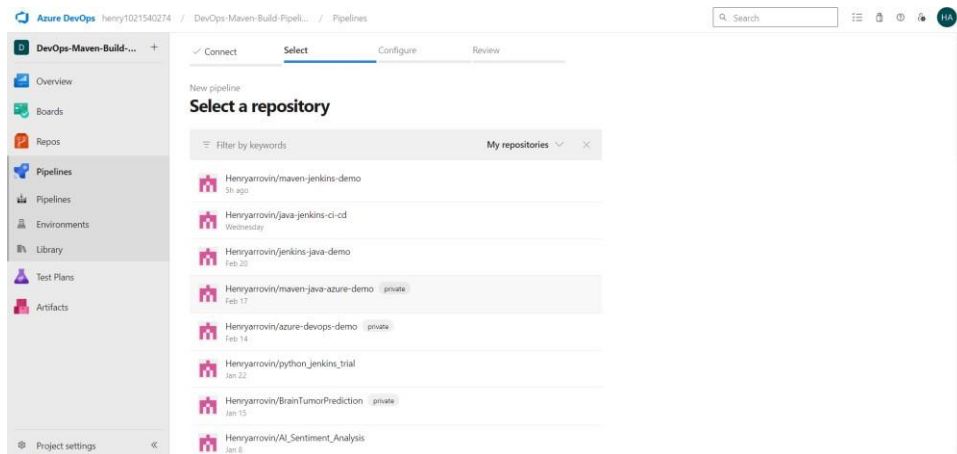
- Go to pipelines in Azure DevOps



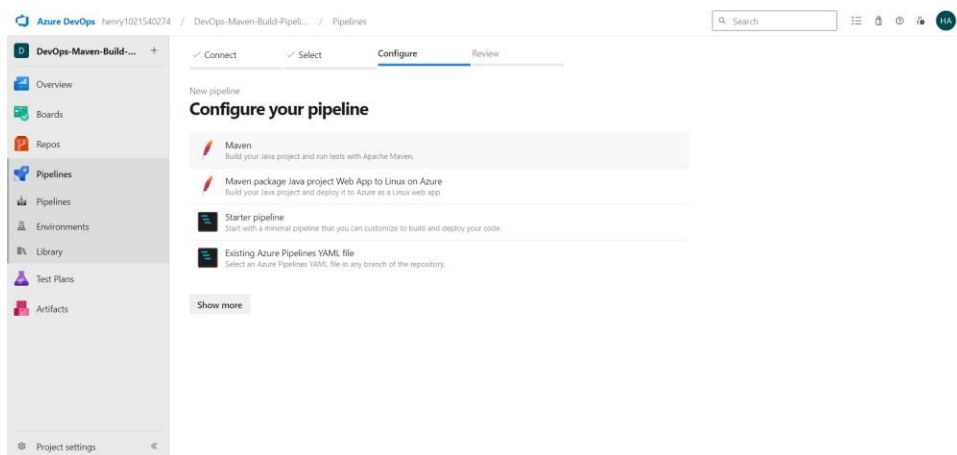
- Click GitHub or where your code is stored
- GitHub is used in this Demo



- Choose your source code



- Make sure it is a maven project by verifying whether there is pom.xml in that project



- Since maven project click on Maven to Generate a sample YAML script for basic test and build

```
steps:
Settings
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    JDK version | on: 'JDKVersion'
    jdkVersionOption: '1.8'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    goals: 'package'
```

- Since we want to create an artifact (jar/war) change the YAML script as given below

trigger:

- master  
pool: mynewpool

### **Conclusion**

Thus, a maven project was created and build a pipeline in Azure

**Date:**

## **Ex No: 2**

### **Run regression tests using Maven Build pipeline in Azure**

#### **Aim**

To create a maven project and build a pipeline in Azure and create test cases to run regression test

#### **Procedure**

##### **Create a Maven Project:**

If you haven't already, create a basic Java project with the following structure:

- package main:
  - Main class
  - Calculator class
  - Add method
- package test:
  - CalculatorTest class
  - addTest method
- Ensure that your tests use the JUnit framework.

##### **Azure DevOps Pipeline Configuration:**

- In your Azure DevOps project, import your Java project from GitHub.
- Create a new pipeline using the starter template.
- Modify the pipeline YAML file to include the necessary tasks.

##### **Configure the Pipeline YAML:**

- Define the trigger (e.g., on each commit to the master branch).
- Specify the VM image (e.g., 'Ubuntu-16.04').
- Add tasks to build, test, and deploy your project. For testing, you'll need to run your JUnit tests.

##### **Run Tests Automatically:**

- To run your tests automatically after each commit, follow these steps:
- Open the Test Plans web portal.
- Select your test plan.
- Choose a test suite containing your automated tests.
- Select the specific tests you want to run.
- Click "Run test."
- Ensure that the test binaries are available in the build artifacts generated by your build pipeline.

##### **View Results:**

- You'll see the test results (including code coverage) in the pipeline summary.
- For code coverage, consider integrating tools like JaCoCo or Cobertura.

##### **Publish Build Artifacts:**

- To publish build artifacts, add the following task to your pipeline YAML:
  - task: PublishBuildArtifacts@1

Note that the warning "Directory '/home/vsts/work/1/a' is empty" means that no files were found to include in the artifact. Make sure your build generates the necessary output files.

## Output

steps:

- script: echo "Stage 1 - Setting up Java version."

- task: Maven@3

inputs:

mavenPomFile: 'pom.xml' mavenOptions: '-

Xmx3072m' javaHomeOption: 'JDKVersion'

jdkArchitectureOption: 'x64'

publishJUnitResults: true

testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml' goals:

'package'

- script: echo "Stage 2 - Maven build and test completed successfully."

- task: PublishBuildArtifacts@1

inputs:

PathToPublish: '\$(System.DefaultWorkingDirectory)/target' ArtifactName: 'my-java-artifacts'

publishLocation: 'container'

- script: echo "Stage 3 - Artifacts published successfully."

- task: PublishTestResults@2

inputs:

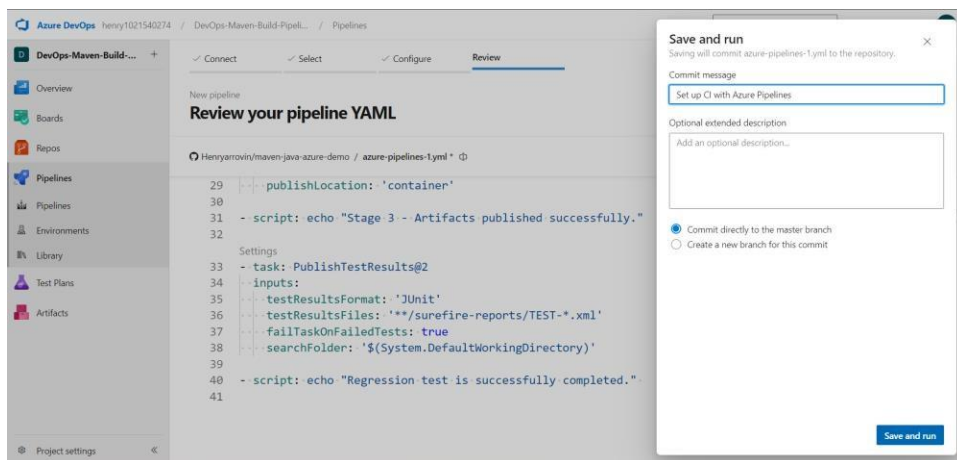
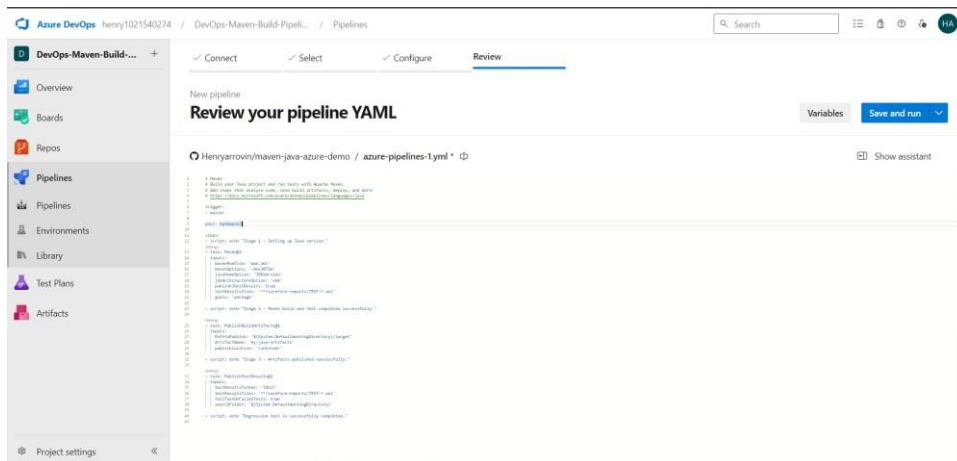
testResultsFormat: 'JUnit'

testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'

failTaskOnFailedTests: true

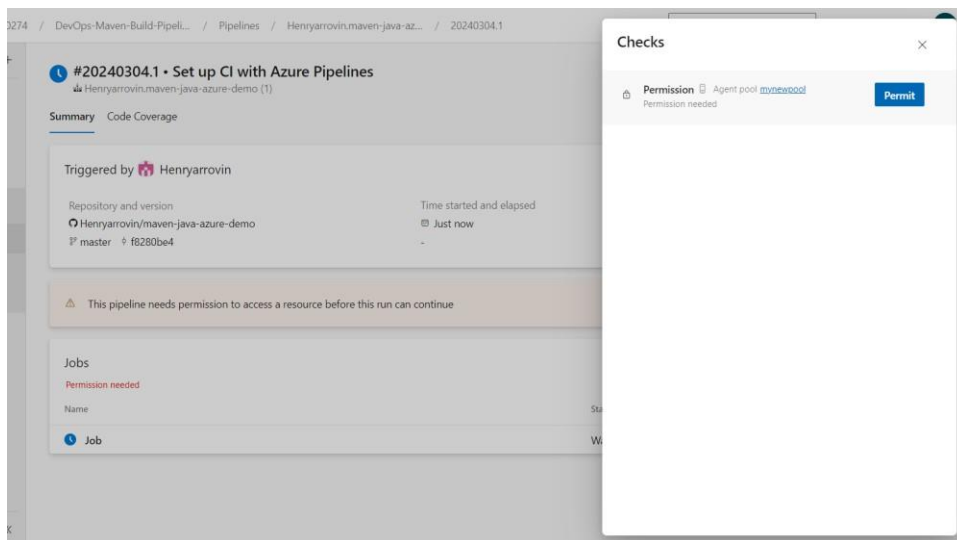
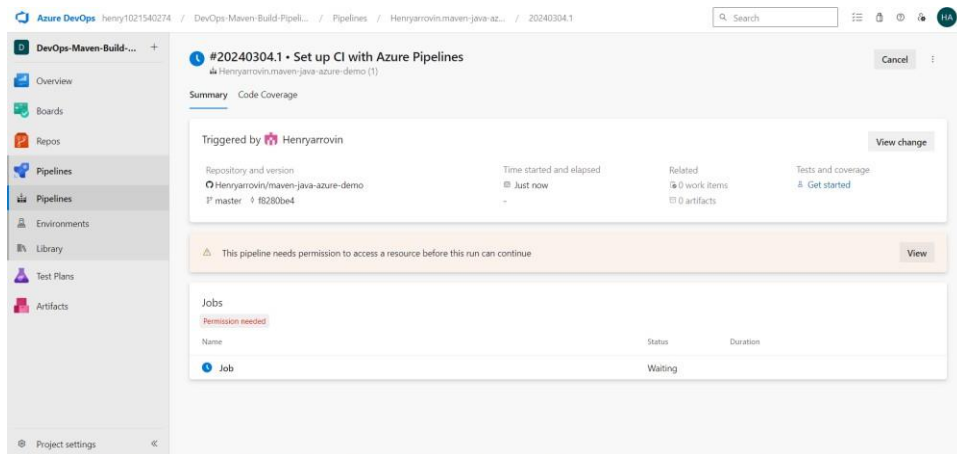
searchFolder: '\$(System.DefaultWorkingDirectory)'

- script: echo "Regression test is successfully completed."



- Commit the YAML script to the main branch directly or to a newly created branch and merge it to the main branch.
- If the pipeline has no access, then give permission by clicking on to “Permission Needed”.





- Don't forget to run the agent created using the run command so that our agent can be used to run the job.

```
2024-03-04 05:20:15Z: Settin
Enter run agent as service?
Enter configure autologon an
PS C:\agent> .\run.cmd
```

- Now if we run pipeline or commit any changes in our code the job will get executed.

Azure DevOps interface showing the 'Jobs in run #20240304.1' for the 'DevOps-Maven-Build-Pipeline'.

**Jobs in run #20240304.1**  
Henryaravin.maven-java-azure-demo [1]

**Jobs**

Job	Duration
Initialize job	29s
Checkout Henryaravin/...	8s
CmdLine	1s
Maven	13s
CmdLine	<1s
PublishBuildArtifacts	10s
CmdLine	1s
PublishTestResults	1s
CmdLine	<1s
Post-job: Checkout He...	<1s
Finalize Job	<1s

**Job**

```
1 Pool: mavenpool
2 Agent: LAPTOP-3A142521
3 Started: Just now
4 Duration: 1m 3s
5
6 * Job preparation parameters
7 1 Artifact produced
8 Job live console data:
9 Async Command Start: DetectDockerContainer
10 Async Command End: DetectDockerContainer
11 Async Command Start: DetectDockerContainer
12 Async Command End: DetectDockerContainer
13 Async Command Start: DetectDockerContainer
14 Async Command End: DetectDockerContainer
15 Finishing: Job
```

Azure DevOps interface showing the 'Published artifacts' for the 'DevOps-Maven-Build-Pipeline'.

**Artifacts**

**Published**

Name	Size
my-java-artifacts	9 KB
classes	4 KB
maven-archiver	77 B
maven-java-azure-demo-1.0-SNAPSHOT.jar	5 KB
maven-status	280 B

Windows File Explorer showing the contents of the 'agent > \_work > 1 > s > target' directory.

Name	Date modified	Type	Size
classes	3/4/2024 10:59 AM	File folder	
generated-sources	3/4/2024 10:59 AM	File folder	
maven-archiver	3/4/2024 10:59 AM	File folder	
maven-status	3/4/2024 10:59 AM	File folder	
maven-java-azure-demo-1.0-SNAPSHOT...	3/4/2024 10:59 AM	Executable Jar File	5 KB

## Conclusion

Thus a regression test cases are executed using Maven Build pipeline in Azure

**Date:**

**Ex No: 3**

## **Install Jenkins in Cloud**

### **Aim**

To install Jenkins in Cloud

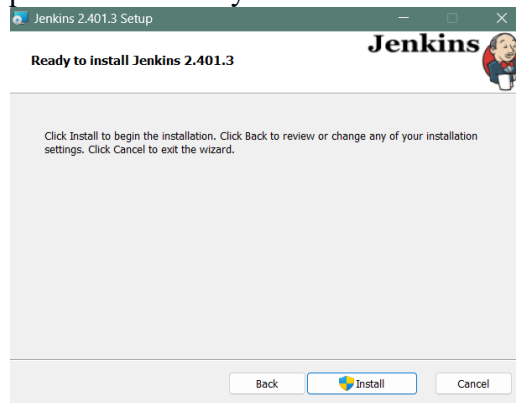
### **Procedure**

#### **Windows**

Downloading Jenkins from the link <https://www.jenkins.io/download/> stable LTS release of the Windows installer. After the download completes, open the Windows installer and follow the steps below to install Jenkins.

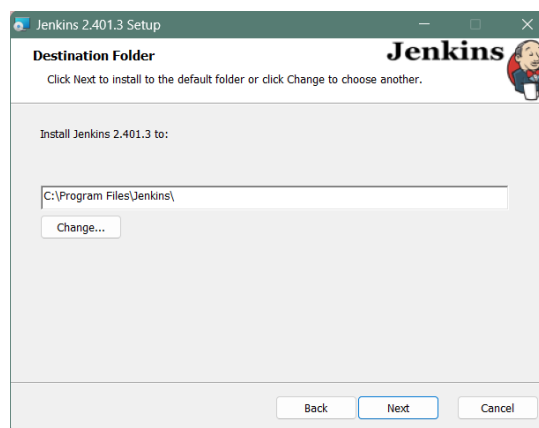
#### **Step 1: Setup wizard**

On opening the Windows Installer, an **Installation Setup Wizard** appears, Click **Next** on the Setup Wizard to start your installation.



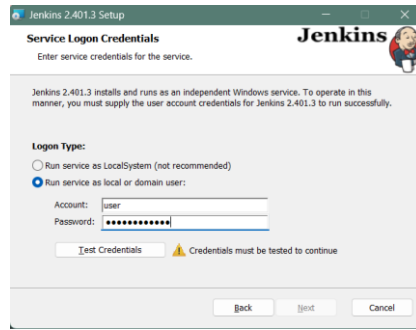
#### **Step 2: Select destination folder**

Select the destination folder to store your Jenkins Installation and click **Next** to continue.



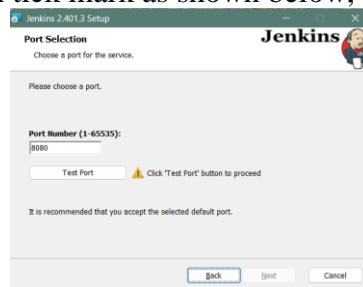
#### **Step 3: Service logon credentials**

When Installing Jenkins, it is recommended to install Jenkins using **LocalSystem(Windows equivalent of root)** which will grant Jenkins full access to your machine and services. click on **Test Credentials** to test your domain credentials and click on **Next**.



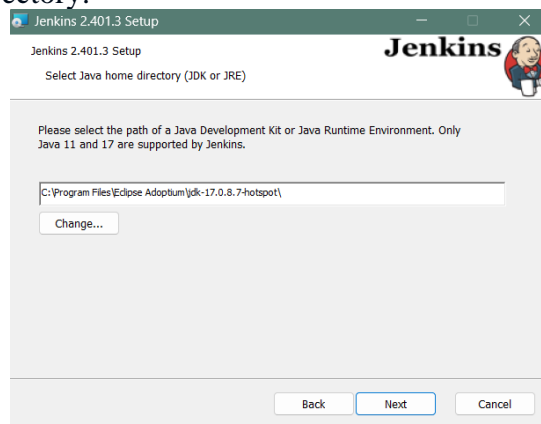
#### Step 4: Port selection

Specify the port on which Jenkins will be running, **Test Port** button to validate whether the specified port is free on your machine or not. Consequently, if the port is free, it will show a green tick mark as shown below, then click on **Next**.



#### Step 5: Select Java home directory

The installation process checks for Java on your machine and prefills the dialog with the Java home directory.



#### Step 6: Install Jenkins

Click on the **Install** button to start the installation of Jenkins.



#### Step 7: Finish Jenkins installation

Once the installation completes, click on **Finish** to complete the installation.



## Post-installation setup wizard

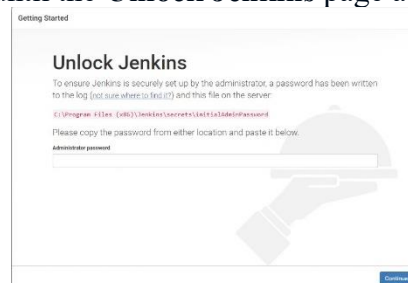
After downloading, installing and running Jenkins, the post-installation setup wizard begins.

## Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically-generated password.

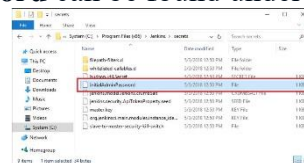
### Step 1

Browse to <http://localhost:8080> (or whichever port you configured for Jenkins when installing it) and wait until the **Unlock Jenkins** page appears.



### Step 2

The initial Administrator password should be found under the Jenkins installation path. For default installation location to C:\Program Files\Jenkins, a file called **initialAdminPassword** can be found under C:\Program Files\Jenkins\secrets.



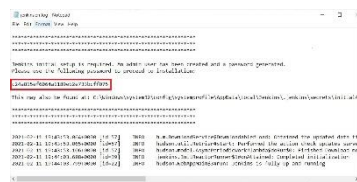
### Step 3

Open the highlighted file and copy the content of the **initialAdminPassword** file.



### Step 4

On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**.



This password must be entered in the setup wizard on new Jenkins installations before you can access Jenkins's main UI.

## Customizing Jenkins with plugins

After [unlocking Jenkins](#), the **Customize Jenkins** by installing any number of useful plugins as part of your initial setup.

Click one of the two options shown:

- **Install suggested plugins** - to install the recommended set of plugins, which are based on most common use cases.

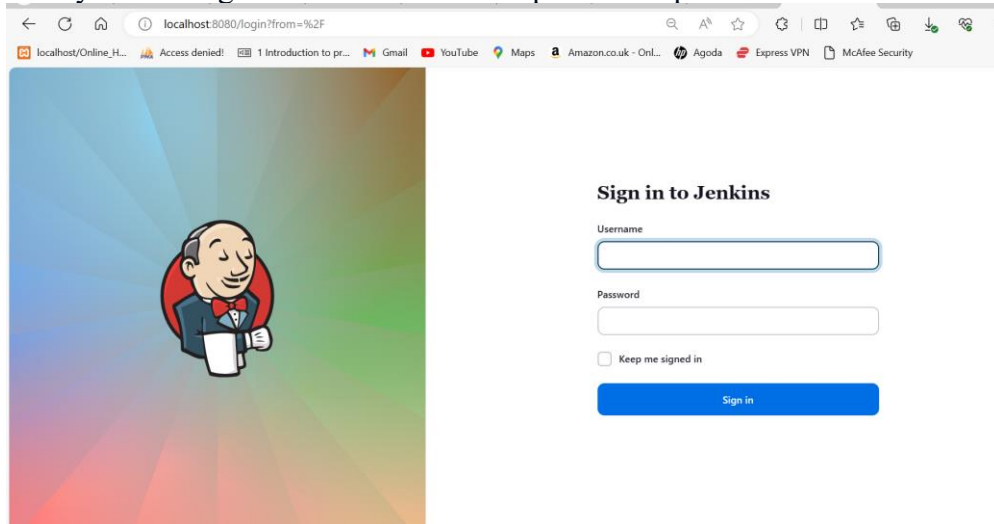
- **Select plugins to install** - to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.

### Creating the first administrator user

Finally, after [customizing Jenkins with plugins](#), Jenkins asks you to create your first administrator user.

1. When the **Create First Admin User** page appears, specify the details for your administrator user in the respective fields and click **Save and Finish**.
2. When the **Jenkins is ready** page appears, click **Start using Jenkins**.

Re login every time using the local host with the port no. <http://localhost:8080/>



### Google Cloud Platform (GCP):

- Ensure you have a Google Cloud account. If not, you can create one.
- Follow the tutorial on Jenkins on Google Cloud to set up Jenkins on Google Compute Engine. This guide assumes familiarity with Packer (for creating images) and Jenkins itself.

### Amazon Web Services (AWS) EC2:

- Launch an EC2 instance.
- In the security group of the instance, open port 8080.
- SSH into the EC2 instance using EC2 Instance Connect.
- Install Jenkins on the EC2 instance. You can follow the steps outlined in this tutorial<sup>2</sup>.

### Running CloudFormation from GitHub using Jenkins on EC2:

- Install the CloudFormation plugin in Jenkins.
- Run a CloudFormation template from a GitHub repository using the plugin.
- Optionally, you can also install the AWS CLI without using keys by leveraging SSM Session Manager

Note that the warning “Directory ‘/home/vsts/work/1/a’ is empty” means that no files were found to include in the artifact. Make sure your build generates the necessary output files.

### Azure

#### Configure Your Environment:

- Ensure you have an Azure subscription. If not, create a free account.
- Open Azure Cloud Shell (you can skip this step if you already have a session open).

#### Create a Virtual Machine:

- In Cloud Shell, create a test directory (let’s call it jenkins-get-started).
- Switch to the test directory.
- Create a file named cloud-init-jenkins.txt.

Paste the following code into the new file:

```
#cloud-config
package_upgrade: true
```

runcmd:

```
- sudo apt install openjdk-11-jre -y
- curl -fsSL [^1^][5] | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
- echo 'deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] [^2^][6] binary/' | sudo tee
/etc/apt/sources.list.d/jenkins.list > /dev/null
- sudo apt-get update && sudo apt-get install jenkins -y
- sudo service jenkins restart
```

Run the following commands:

```
az group create --name jenkins-get-started-rg --location eastus
```

```
az vm create \
  --resource-group jenkins-get-started-rg \
  --name jenkins-get-started-vm \
  --image UbuntuLTS \
  --admin-username "azureuser" \
  --generate-ssh-keys \
  --public-ip-sku Standard \
  --custom-data cloud-init-jenkins.txt
```

Verify the creation of the new virtual machine:

```
az vm list -d -o table --query "[?name=='jenkins-get-started-vm']"
```

Configure Jenkins:

Retrieve the public IP address of the virtual machine:

```
az vm show --resource-group jenkins-get-started-rg --name jenkins-get-started-vm -d --query
"[publicIps]" --output tsv
```

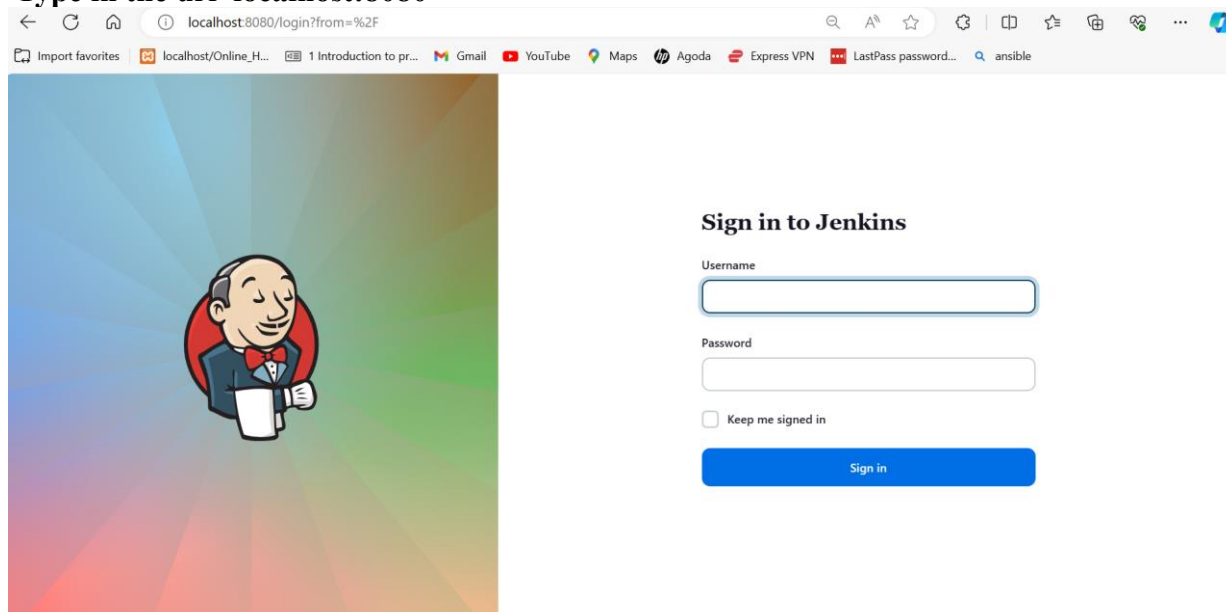
SSH into the virtual machine using the IP address:

```
ssh azureuser@<ip_address>
```

Upon successful connection, you'll see the prompt: azureuser@jenkins-get-started-vm.

## Output

Type in the url localhost:8080



The screenshot shows the Jenkins Dashboard with a sidebar on the left containing links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'My Views'. The main area displays a table of pipeline jobs under the 'Pipelinejob' filter.

S	W	Name	Last Success	Last Failure	Last Duration
✗	☁	datedisplay	N/A	1 mo 6 days #2	0.4 sec
✗	☁	firstjavajob	N/A	1 mo 12 days #2	0.64 sec
...	☀	firstmaven	N/A	N/A	N/A
✗	☁	gitintegration	N/A	1 mo 4 days #3	1.7 sec
✗	☁	jav	N/A	1 mo 8 days #6	0.35 sec
✗	☁	javagitintegration	N/A	1 mo 1 day #7	0.34 sec
✓	☁	jenjava	1 mo 8 days #6	1 mo 8 days #4	0.79 sec
...	☀	pipelinecode	N/A	N/A	N/A

To use the installed Jenkins in Azure Cloud type the following link  
<http://52.190.63.117:8080/>

The screenshot shows the Jenkins login page with the Jenkins logo on the left. On the right, there is a 'Sign in to Jenkins' section with fields for 'Username' (containing 'AzureAnnrose') and 'Password' (masked with dots). There is a 'Keep me signed in' checkbox and a blue 'Sign in' button.

The screenshot shows the Jenkins Dashboard after successful login. The sidebar on the left is visible. The main area shows a table with one job, 'pipelinecode', which is successful.

S	W	Name	Last Success	Last Failure	Last Duration
✓	☁	pipelinecode	8 days 18 hr #3	8 days 18 hr #2	0.39 sec

## Conclusion

Thus, the Jenkins was successfully installed.



Date:

**Ex No: 4**

## **Create CI pipeline using Jenkins**

### **Aim**

To create CI pipeline using Jenkins

### **Procedure**

#### **Log in to Jenkins:**

- Access your Jenkins account.
- If you haven't installed Jenkins yet, follow the Jenkins installation guide.

#### **Create a New Jenkins Project:**

- Once logged in, you'll be redirected to the Jenkins console.
- Click on "New Item" in the dashboard.
- Choose a suitable name for your pipeline project.
- Select the "Pipeline" option.

#### **Configure Your Pipeline:**

- In the pipeline configuration, you can define your stages. A typical CI/CD pipeline includes stages like:
- Test code: Run tests to ensure code quality.
- Build Application: Compile and package your application.
- Push to Repository: Store artifacts in a version control system (e.g., Git).
- Deploy to Server: Deploy your application to different environments (e.g., Dev, Test, Production).

#### **Pipeline Script:**

- In the pipeline configuration, you'll find a section to write your pipeline script.
- You can use either declarative syntax or scripted syntax.

Here's a simple example of a declarative pipeline script:

```
pipeline {
  agent any
  stages {
    stage('Test code') {
      steps {
        // Run tests here
      }
    }
    stage('Build Application') {
      steps {
        // Compile and package your app
      }
    }
    stage('Push to Repository') {
      steps {
        // Push artifacts to Git
      }
    }
    stage('Deploy to Server') {
      steps {
        // Deploy to different environments
      }
    }
  }
}
```

Save and Run:

- Save your pipeline configuration.
- Click “Build Now” to trigger the pipeline execution.
- Jenkins will execute each stage sequentially.

## Output

### Step 1

Since I am using Docker to host Jenkins, I will start the Jenkins container by pulling it. Alternatively, Jenkins can also be hosted by installing it directly on a server or virtual machine, configuring it, and then starting the Jenkins service.

The top part of the image shows a Docker container for Jenkins. The container name is 'jenkins\_localhost' with ID 'd8a2f8f928de'. It is in a 'Running' state, using the 'jenkins/jenkins' image, and has a CPU usage of 0.15%. It was started 3 seconds ago.

The bottom part of the image shows the Jenkins web interface running on 'localhost:8001'. The dashboard displays a list of build items:

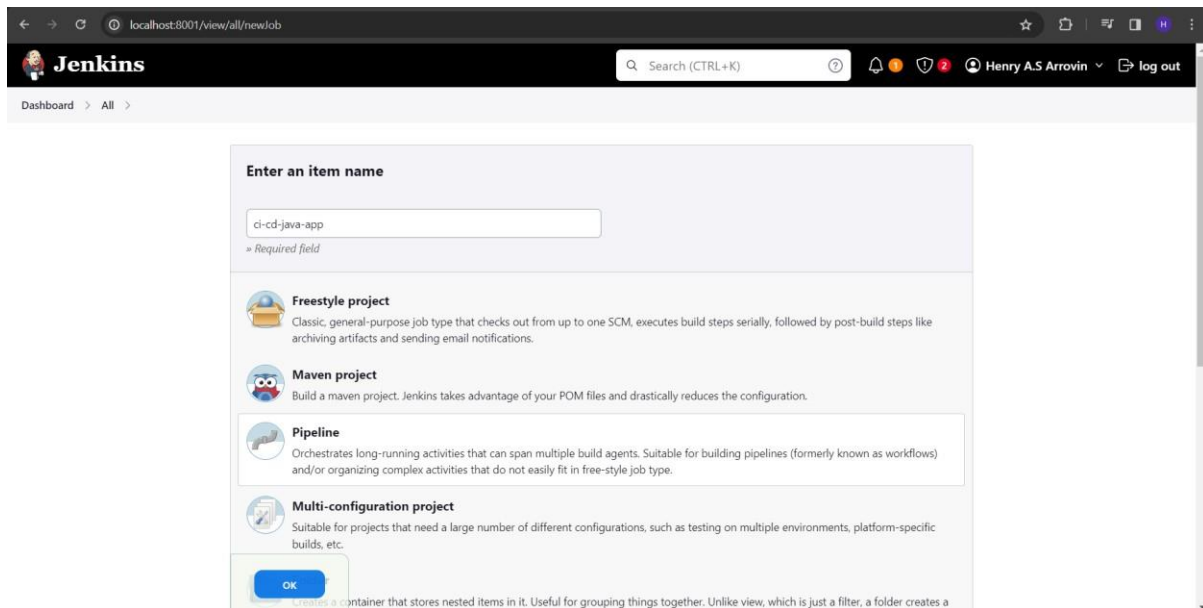
S	W	Name	Last Success	Last Failure	Last Duration
✓	☁	java-ci-cd-jenkinsfile	26 min #27	1 hr 7 min #25	1 min 15 sec
✓	☁	maven-plugin-pipeline	25 min #58	4 hr 4 min #55	1 min 22 sec

On the left sidebar, the 'Build Queue' is empty, and the 'Build Executor Status' shows 2 idle executors. The bottom status bar indicates 'Jenkins 2.441'.

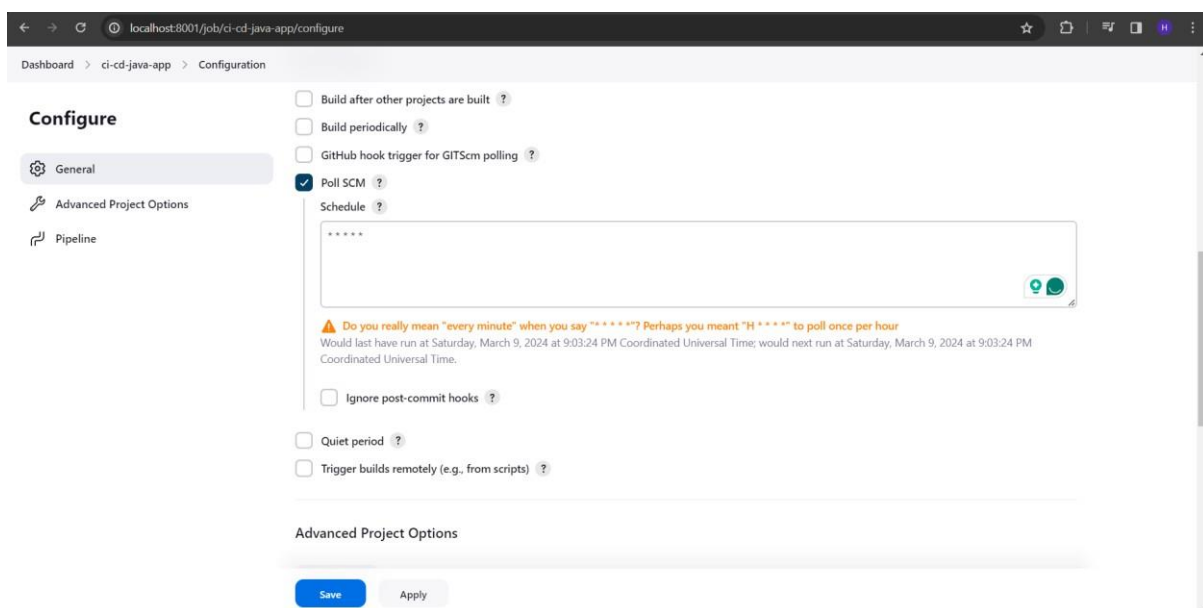
### Step 2

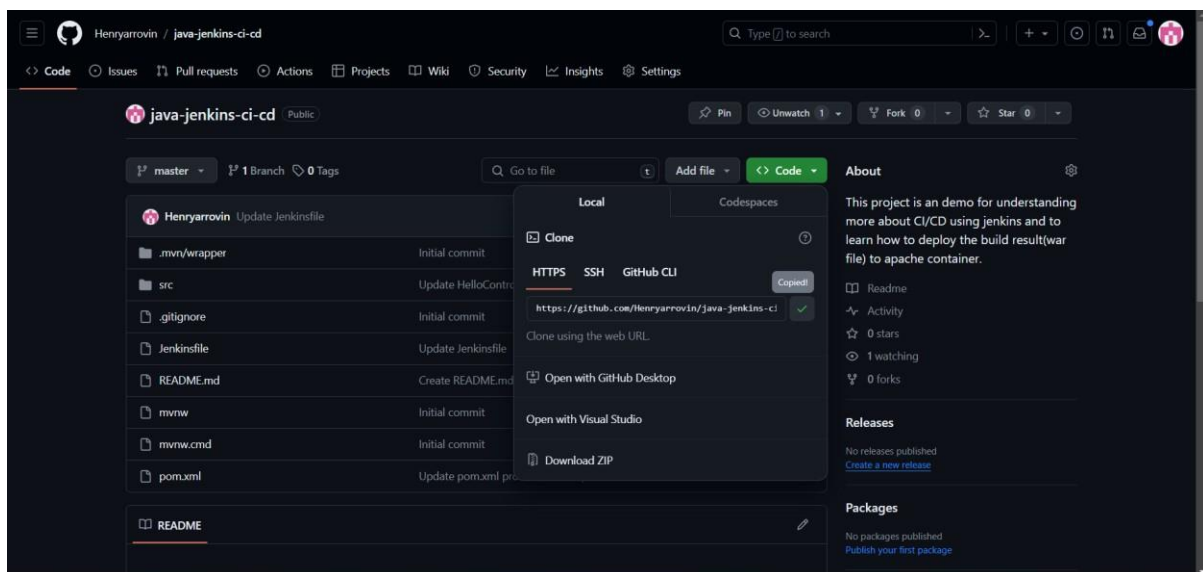
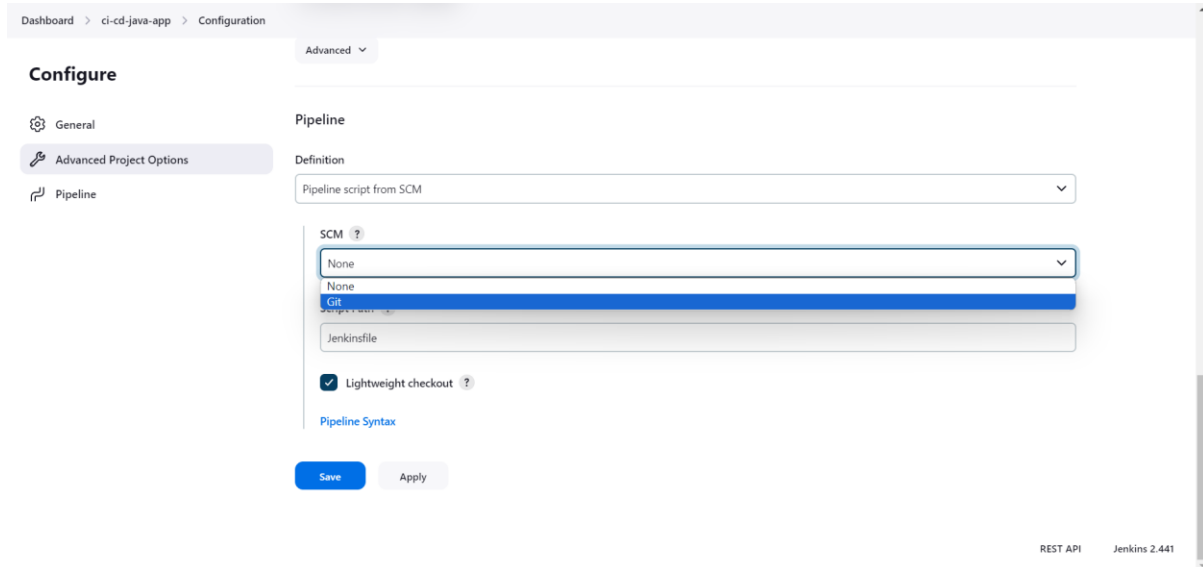
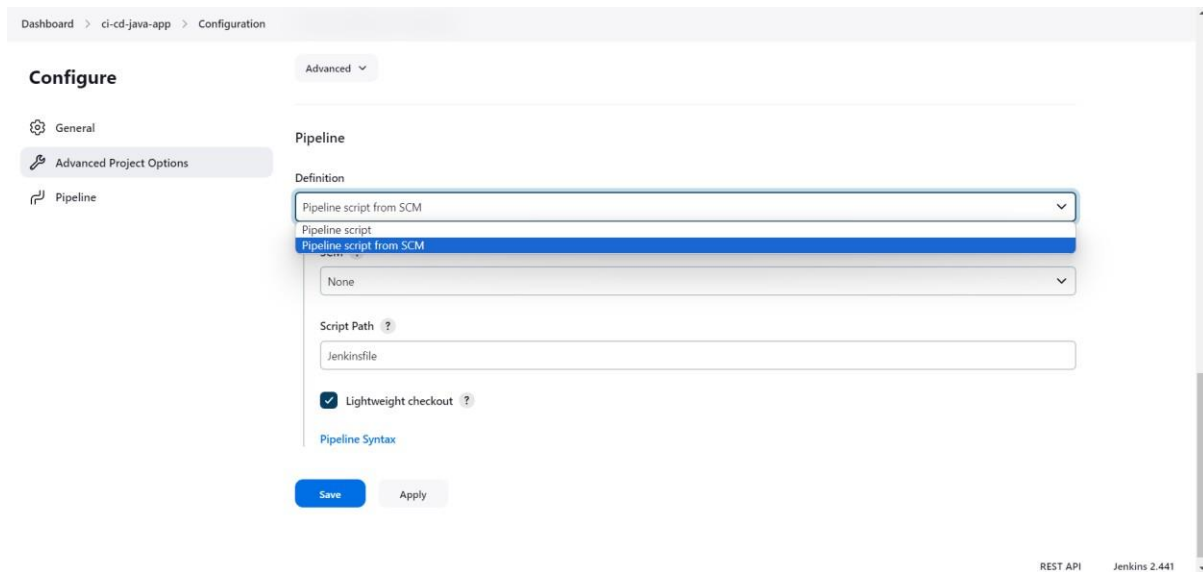
Click on new Item to create a new Pipeline project.

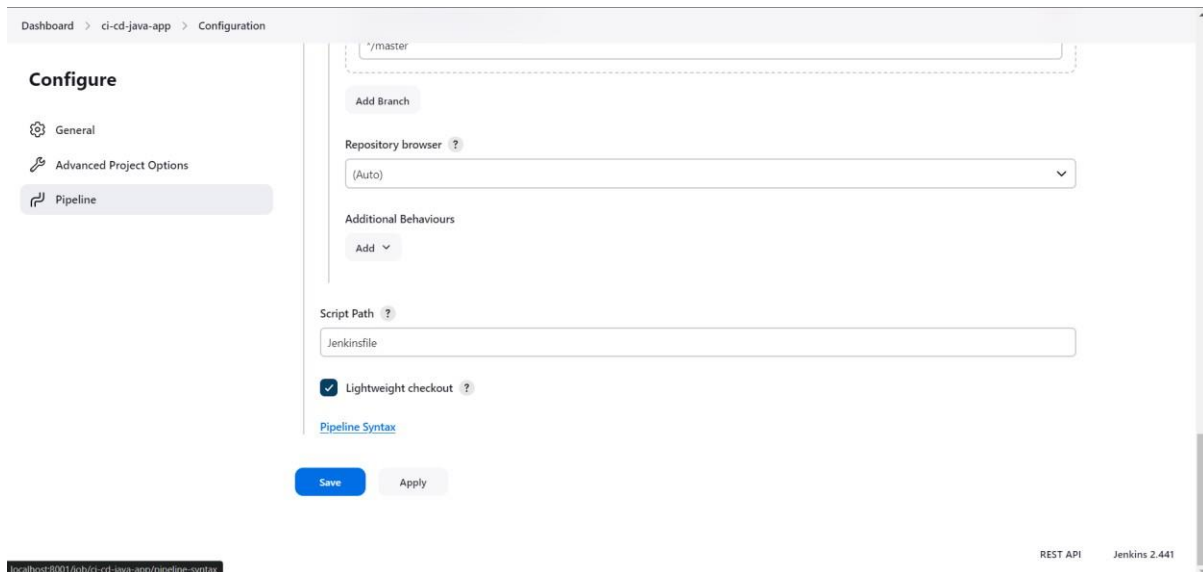
This screenshot shows the 'New Item' page in the Jenkins web interface. The 'New Item' button is highlighted in the left sidebar. The main content area shows the same build history table as in Step 1. At the bottom left, the browser address bar shows 'localhost:8001/view/all/new/job', indicating the user is navigating to create a new job.



Configure the pipeline to run a job every minute using the cron syntax '\* \* \* \* \*' and trigger the pipeline whenever a commit is made in GitHub.







Dashboard > ci-cd-java-app > Configuration

**Configure**

- General
- Advanced Project Options
- Pipeline**

\*/master

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

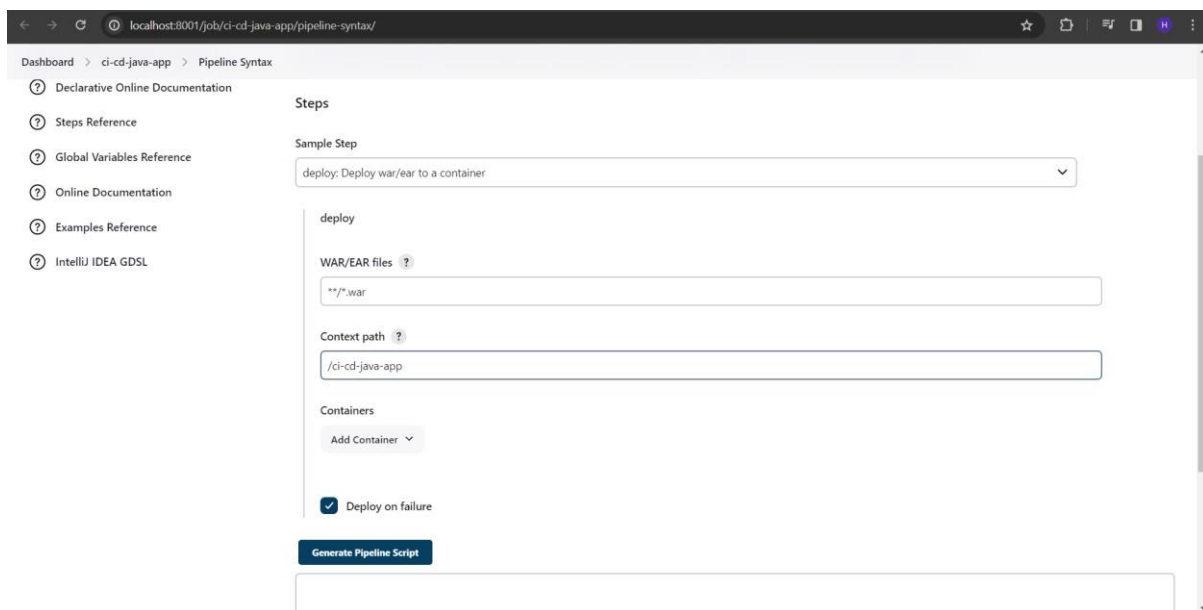
☒ Lightweight checkout ?

[Pipeline Syntax](#)

Save Apply

localhost:8001/job/ci-cd-java-app/pipeline-syntax/ REST API Jenkins 2.441

Generate the pipeline syntax for deploying the application's artifact to the Tomcatserver, specifying the context path as '/your-context-path'. This will automate the deployment process as part of your continuous integration pipeline.



localhost:8001/job/ci-cd-java-app/pipeline-syntax/

Dashboard > ci-cd-java-app > Pipeline Syntax

- Declarative Online Documentation
- Steps Reference
- Global Variables Reference
- Online Documentation
- Examples Reference
- IntelliJ IDEA GDLS

**Steps**

Sample Step

deploy; Deploy war/ear to a container

deploy

WAR/EAR files ?

\*\*/\*.war

Context path ?

/ci-cd-java-app

Containers

Add Container

☒ Deploy on failure

Generate Pipeline Script

Choose the installed Tomcat version on your system and provide the credentials for the Tomcat Manager application. Additionally, in the advanced manager context path, specify '/manager/text' as it is crucial for receiving an 'OK' message, ensuring the success of the deployment process. Follow the steps below for guidance.

Context path ?

/ci-cd-java-app

Containers

Add Container ^

Filter

- GlassFish 2.x
- GlassFish 3.x
- GlassFish 4.x
- JBoss AS 3.x
- JBoss AS 4.x
- JBoss AS 5.x
- JBoss AS 6.x
- JBoss AS 7.x
- Tomcat 4.x Remote
- Tomcat 5.x Remote
- Tomcat 6.x Remote
- Tomcat 7.x Remote
- Tomcat 8.x Remote
- Tomcat 9.x Remote

Global

There are some steps that are not steps. These are often exposed via global variables, which are not supported by

Refer

#### Jenkins Credentials Provider: Jenkins

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?

admin

☒ Treat username as secret ?

Password ?

.....

ID ?

Description ?

## Jenkins Credentials Provider: Jenkins

Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?

admin

☒ Treat username as secret ?

Password ?

.....

ID ?

Description ?

Cancel

Add

localhost:8001/job/ci-cd-java-app/pipeline-syntax/

Dashboard > ci-cd-java-app > Pipeline Syntax

Tomcat 9.x Remote

Credentials

Sd741c1c-b993-425b-a724-4f42d2defe28 ▼

+ Add

Tomcat URL ?

http://192.168.0.108:8082/

Advanced ^ ✎ Edited

Manager context path ?

/manager/text

Add Container ▼

☒ Deploy on failure

Generate Pipeline Script

Dashboard > ci-cd-java-app > Pipeline Syntax

Manager context path ?  
/manager/text

Add Container ▾

☒ Deploy on failure

Generate Pipeline Script

```

deploy adapters: [tomcat9(credentialsId: '5d741c1c-b993-425b-a724-4f42d2defe28', path: '/manager/text', url: 'http://192.168.0.108:8082/'), contextPath: '/ci-cd-java-app', war:
'**/*.war'

```

Global Variables

There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the [Global Variables Reference](#) for details.

Now paste the syntax generated in the deploy stage of the pipeline code(Jenkinsfile)

```

1 pipeline {
2   agent any
3   tools {
4     maven 'Maven'
5   }
6   stages {
7     stage('Build') {
8       steps {
9         script {
10           sh "mvn clean package"
11         }
12       }
13       post {
14         success {
15           echo "Archiving the Artifacts"
16           archiveArtifacts artifacts: '**/target/*.war'
17         }
18       }
19     }
20     stage('Deploy to Tomcat') {
21       steps {
22         script {
23           deploy adapters: [tomcat9(credentialsId: '5d741c1c-b993-425b-a724-4f42d2defe28', path: '/manager/text', url: 'http://192.168.0.108:8082/'), co
24         }
25       }
26     }
27   }
28 }
29

```

```

Code: pipeline {
  agent any
  tools {
    maven 'Maven'
  }
  stages {

```



```

stage('Build') {
    steps {
        script {
            sh "mvn clean package"

        }
    }
}

post {
    success {
        echo "Archiving the Artifacts" archiveArtifacts
        artifacts: '**/target/*.war'
    }
}

stage('Deploy to Tomcat') {
    steps {
        script {
            // Paste the code generated.

        }
    }
}
}

```


### Step 3







If we build the pipeline or commit changes, it will be triggered.

← → ↺ 🔍

localhost:8001/job/ci-cd-java-app/

☆ 📄 🗑️ 📄 📄 📄

 **Jenkins**

     Henry A.S Aravin  log out

Dashboard > ci-cd-java-app >

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

🔍 Full Stage View

✎ Rename

❓ Pipeline Syntax

📄 Git Polling Log

ci-cd-java-app

Add description

Disable Project

Stage View

No data available. This Pipeline has not yet run.

Permalinks

Build History trend

No builds

localhost:8001/job/ci-cd-java-app/build?delay=5sec

Dashboard > ci-cd-java-app >

Status

ci-ca-java-app

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

🔍 Full Stage View

✎ Rename

❓ Pipeline Syntax

📄 Git Polling Log

Add description

Disable Project

Stage View

No data available. This Pipeline has not yet run.

Permalinks

Build History trend

Filter...

#1

Mar 9, 2024, 9:10 PM

Atom feed for all Atom feed for failures

REST API Jenkins 2.441

Dashboard > ci-cd-java-app >

Status

ci-ca-java-app

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

🔍 Full Stage View

✎ Rename

❓ Pipeline Syntax

📄 Git Polling Log

Add description

Disable Project

Stage View

Average stage times: 1s

Declarative: Checkout SCM

#1

Mar 10 02:40

No Changes

1s

Permalinks

Build History trend

In progress > Console Output

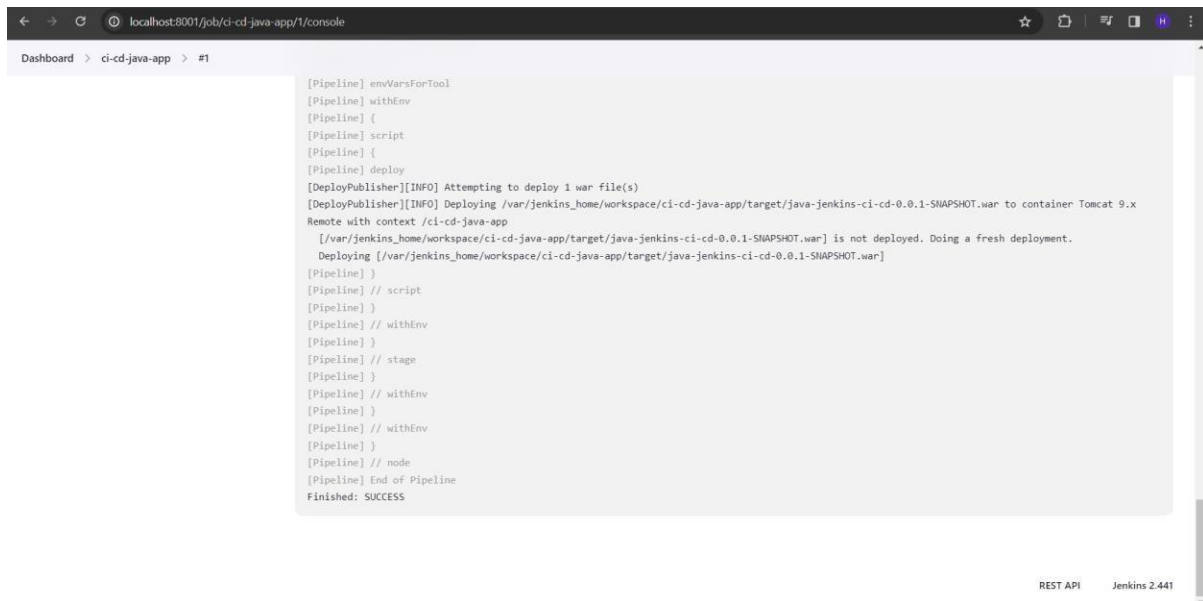
#1

Mar 9, 2024, 9:10 PM

Atom feed for all Atom feed for failures

REST API Jenkins 2.441

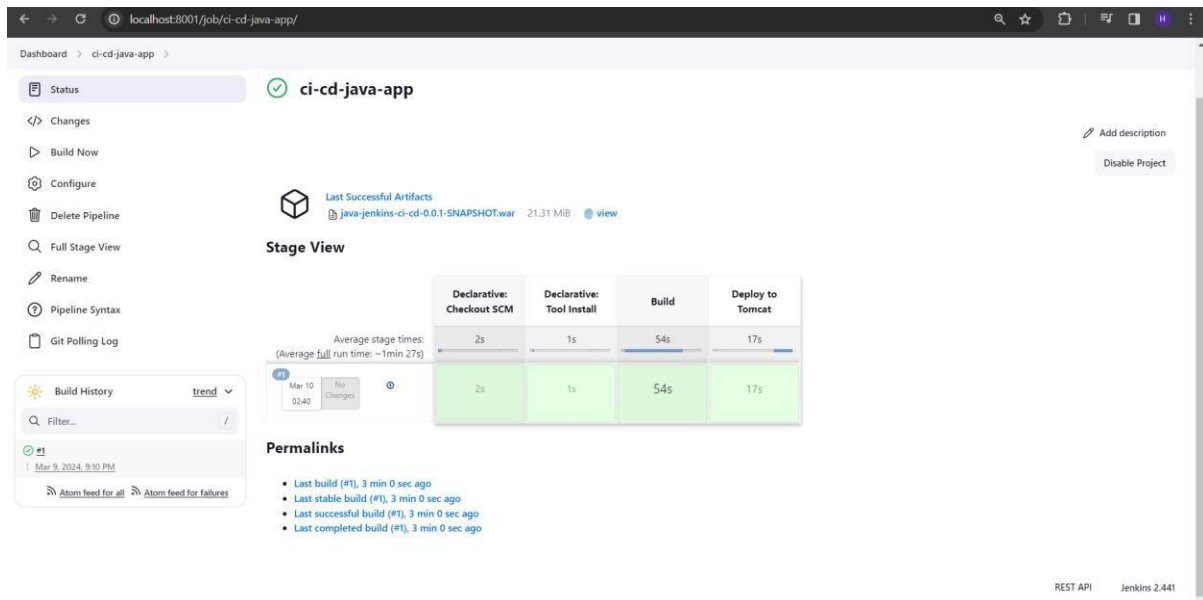
We can also check what is happening properly in the console.



The screenshot shows the Jenkins console output for a pipeline named 'ci-cd-java-app'. The output is as follows:

```
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] deploy
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /var/jenkins_home/workspace/ci-cd-java-app/target/java-jenkins-ci-cd-0.0.1-SNAPSHOT.war to container Tomcat 9.x
Remote with context /ci-cd-java-app
  [/var/jenkins_home/workspace/ci-cd-java-app/target/java-jenkins-ci-cd-0.0.1-SNAPSHOT.war] is not deployed. Doing a fresh deployment.
  Deploying [/var/jenkins_home/workspace/ci-cd-java-app/target/java-jenkins-ci-cd-0.0.1-SNAPSHOT.war]
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

REST API Jenkins 2.441



The screenshot shows the Jenkins job page for 'ci-cd-java-app'. The page includes a sidebar with navigation options, a main content area with a 'Stage View' table, and a 'Permalinks' section.

**Stage View**

	Declarative: Checkout SCM	Declarative: Tool Install	Build	Deploy to Tomcat
Average stage times: (Average full run time: ~1min 27s)	2s	1s	54s	17s
Mar 10 02:40 No Changes	2s	1s	54s	17s

**Permalinks**

- Last build (#1), 3 min 0 sec ago
- Last stable build (#1), 3 min 0 sec ago
- Last successful build (#1), 3 min 0 sec ago
- Last completed build (#1), 3 min 0 sec ago

REST API Jenkins 2.441

## Conclusion

Thus, CI pipeline was successfully created using Jenkins

**Date:**

**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

### **Procedure**

Install Jenkins:

- If you haven't already, install Jenkins on your preferred environment (local, cloud VM, or Docker container). Create a Jenkins Job:
- Log in to Jenkins.
- Click "New Item" to create a new job.
- Choose "Pipeline" as the job type.

Configure Your Pipeline:

- In the pipeline configuration, define your stages. A CD pipeline typically includes stages like:
- Checkout: Pull your code from the repository.
- Build: Compile, package, and create artifacts.
- Test: Run automated tests.
- Deploy: Deploy to the cloud.

Write Your Jenkinsfile:

- A Jenkinsfile defines your pipeline as code.
- Use either declarative syntax or scripted syntax.

Here's a simple example of a declarative Jenkinsfile:

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        checkout scm
      }
    }
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
      }
    }
    stage('Deploy to Cloud') {
      steps {
        // Deploy to your cloud provider (e.g., AWS, Azure, GCP)
        // Use appropriate tools (e.g., Terraform, Ansible, CloudFormation)
      }
    }
  }
}
```

### Configure Cloud Credentials:

- Add your cloud provider credentials (e.g., AWS access keys, Azure service principal) to Jenkins.
- Use the Credentials Plugin to securely manage secrets.

### Deploy to Cloud:

In the “Deploy to Cloud” stage, use tools like:

- Terraform: Define infrastructure as code (IaC) and provision resources.
- Ansible: Configure servers and services.
- CloudFormation: Define AWS infrastructure.
- Kubernetes: Deploy containers to a cluster.

### Post-Deployment Actions:

After successful deployment, consider additional steps:

- Notifications: Send email notifications or Slack messages.
- Monitoring: Set up monitoring and alerts.
- Rollbacks: Implement rollback strategies.

## Output

Now we can check whether our app is deployed in tomcat server in `<tomcat-url>/manager/html` page.



The screenshot shows the Tomcat Web Application Manager interface in a web browser. The address bar shows the URL `192.168.0.108:8082/manager/html`. The page title is "Tomcat Web Application Manager". Below the title, there is a message bar showing "Message: OK". The main content area is divided into several sections. The "Manager" section has links for "List Applications", "HTML Manager Help", "Manager Help", and "Server Status". The "Applications" section is a table listing deployed applications.

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/ci-cd-java-app	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/demo-api	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/java-app	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/java-jenkins-ci-cd-0.0.1-SNAPSHOT	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

It'll be in the name of our context path.

/ci-cd-java-app	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
-----------------	----------------	--	------	---	--

### Prerequisite:

Before proceeding, ensure that Tomcat is installed on your system as a prerequisite for this setup.

To start the Tomcat server, navigate to the 'bin' directory and run 'startup.bat' on Windows or './startup.sh' on Ubuntu. To stop the server, use 'shutdown.bat' on Windows or './shutdown.sh' on Ubuntu.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. All rights reserved.

C:\apache-tomcat-9.0.86\bin>startup.bat
Using CATALINA_BASE: "C:\apache-tomcat-9.0.86"
Using CATALINA_HOME: "C:\apache-tomcat-9.0.86"
Using CATALINA_TMPDIR: "C:\apache-tomcat-9.0.86\temp"
Using JRE_HOME: "C:\Users\Henry\Downloads\jdk-19_windows-x64_bin\jdk-19.0.1"
Using CLASSPATH: "C:\apache-tomcat-9.0.86\bin\bootstrap.jar;C:\apache-tomcat-9.0.86\bin\tomcat-juli.jar"
Using CATALINA_OPTS: ""
C:\apache-tomcat-9.0.86\bin>
```

And make sure that maven is configured in Jenkins.

Dashboard >


+ New Item

 People

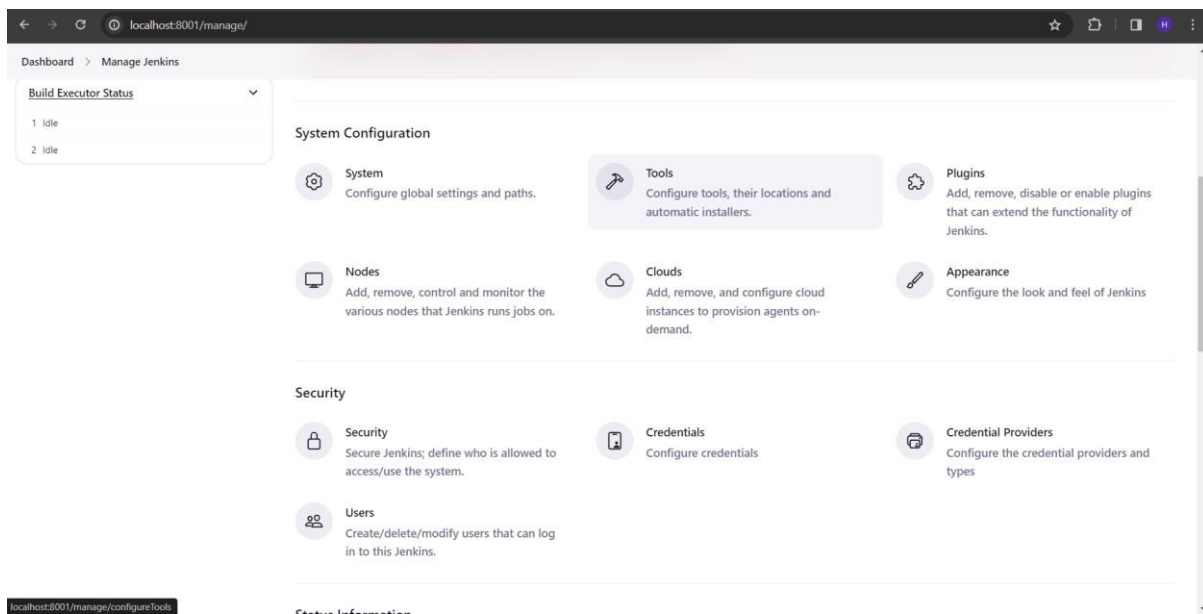
 Build History

 Project Relationship

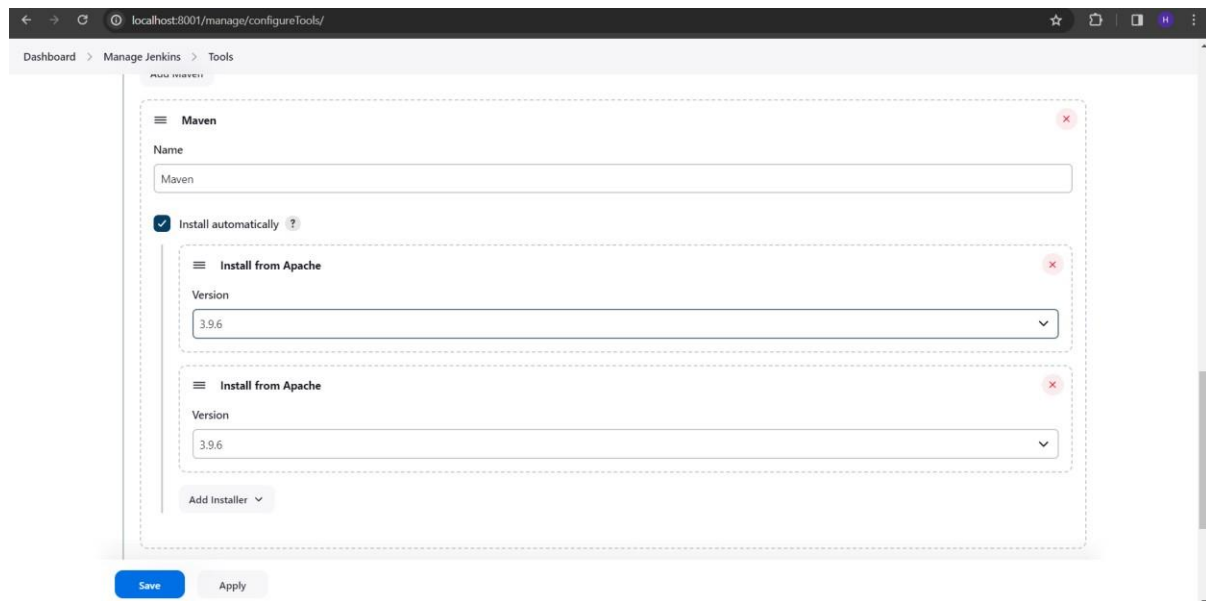
 Check File Fingerprint

 Manage Jenkins

 My Views



The screenshot shows the Jenkins Manage Jenkins dashboard in a web browser. The browser address bar shows 'localhost:8001/manage/'. The dashboard has a sidebar on the left with a 'Build Executor Status' dropdown showing two 'Idle' executors. The main content area is titled 'Dashboard > Manage Jenkins' and is divided into two main sections: 'System Configuration' and 'Security'. The 'System Configuration' section includes links for 'System' (Configure global settings and paths), 'Tools' (Configure tools, their locations and automatic installers), 'Nodes' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on), 'Clouds' (Add, remove, and configure cloud instances to provision agents on-demand), 'Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), and 'Appearance' (Configure the look and feel of Jenkins). The 'Security' section includes links for 'Security' (Secure Jenkins; define who is allowed to access/use the system), 'Users' (Create/delete/modify users that can log in to this Jenkins), 'Credentials' (Configure credentials), and 'Credential Providers' (Configure the credential providers and types). At the bottom of the page, there is a 'Footer Information' section.



## Conclusion

Thus, CD pipeline was successfully created using Jenkins



Date:

Ex No: 6

## Build a simple application using Gradle

### Aim

To build a simple application using Gradle

### Procedure

1. Install Gradle (if not already done):

- Make sure you have Gradle installed on your system. You can download it from the official Gradle website.



The image shows a guide for Gradle installation. At the top, a title box says "Gradle Installation". Below it, a yellow box states "There are four major steps to install Gradle:". Four steps are listed with hand icons: 1. Check if Java is installed (index finger), 2. Download Gradle (peace sign), 3. Set Environment variables (three fingers), and 4. Verify Gradle Installation (four fingers). Below the steps is a terminal screenshot showing the command `C:\Users\rose>gradle -version` and its output. The output includes a welcome message for Gradle 8.7, release highlights, and system details.

```
C:\Users\rose>gradle -version

Welcome to Gradle 8.7!

Here are the highlights of this release:
- Compiling and testing with Java 22
- Cacheable Groovy script compilation
- New methods in lazy collection properties

For more details see https://docs.gradle.org/8.7/release-notes.html

-----
Gradle 8.7
-----

Build time:   2024-03-22 15:52:46 UTC
Revision:     650af14d7653aa949fce5e886e685efc9cf97c10

Kotlin:      1.9.22
Groovy:      3.0.17
Ant:         Apache Ant(TM) version 1.10.13 compiled on January 4 2023
JVM:         1.8.0_144 (Oracle Corporation 25.144-b01)
OS:          Windows 10 10.0 amd64

C:\Users\rose>
```

2. Create a Project Directory:

- Open your terminal or command prompt and create a new directory for your project:

```
mkdir my-java-app
```

```
cd my-java-app
```

### 3. Create Java Classes:

- Inside your project folder, create a simple Java class for an Employee:

Java

```
// Employee.java
public class Employee {
    private String name;
    private String emailAddress;
    private int yearOfBirth;

    // Constructors, getters, and setters (if needed)
}
```

Next, create a main class that prints employee data:

Java

```
// EmployeeApp.java
public class EmployeeApp {
    public static void main(String[] args) {
        Employee employee = new Employee();
        employee.setName("John");
        employee.setEmailAddress("john@example.com");
        employee.setYearOfBirth(1990);

        System.out.println("Name: " + employee.getName());
        System.out.println("Email Address: " + employee.getEmailAddress());
        System.out.println("Year of Birth: " + employee.getYearOfBirth());
    }
}
```

### 4. Create a Gradle Build Script:

- In your project folder, create a build.gradle file with the following content:

```
plugins {
    id 'java'
}

repositories {
    jcenter()
}

dependencies {
    implementation 'org.slf4j:slf4j-api:1.7.32'
    testImplementation 'junit:junit:4.13.2'
```

```
}
```

## 5. Build Your Application:

- Run the following command to build your Java application:

```
gradle build
```

Gradle will compile your code, run tests, and create a JAR file in the build/libs directory.

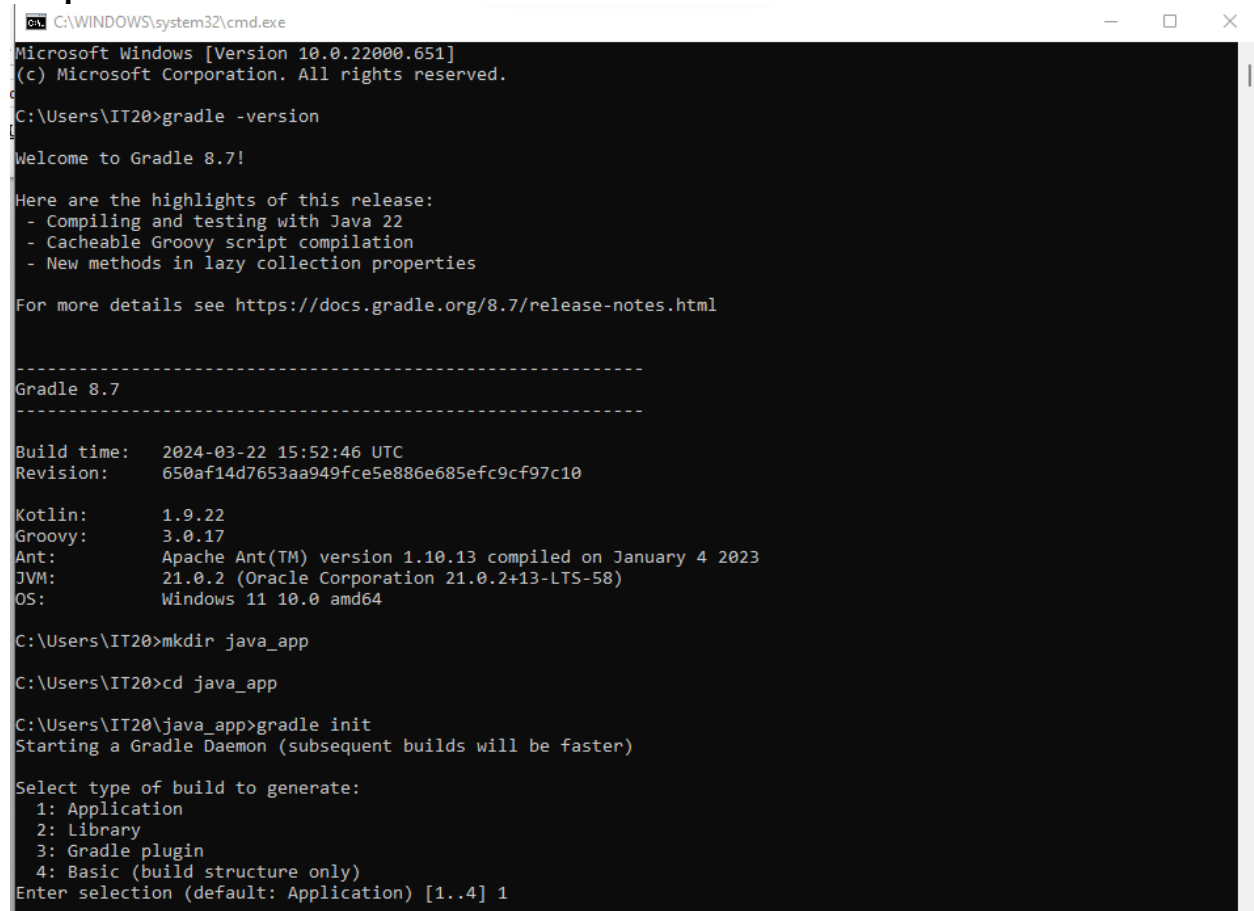
## 6. Run Your Application:

- Execute the following command to run your application:

```
gradle run
```

You'll see the output

### Output



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.22000.651]
(c) Microsoft Corporation. All rights reserved.

C:\Users\IT20>gradle -version

Welcome to Gradle 8.7!

Here are the highlights of this release:
- Compiling and testing with Java 22
- Cacheable Groovy script compilation
- New methods in lazy collection properties

For more details see https://docs.gradle.org/8.7/release-notes.html

-----
Gradle 8.7
-----

Build time:   2024-03-22 15:52:46 UTC
Revision:     650af14d7653aa949f5e886e685efc9cf97c10

Kotlin:       1.9.22
Groovy:       3.0.17
Ant:          Apache Ant(TM) version 1.10.13 compiled on January 4 2023
JVM:          21.0.2 (Oracle Corporation 21.0.2+13-LTS-58)
OS:           Windows 11 10.0 amd64

C:\Users\IT20>mkdir java_app

C:\Users\IT20>cd java_app

C:\Users\IT20\java_app>gradle init
Starting a Gradle Daemon (subsequent builds will be faster)

Select type of build to generate:
 1: Application
 2: Library
 3: Gradle plugin
 4: Basic (build structure only)
Enter selection (default: Application) [1..4] 1
```

```

Select implementation language:
 1: Java
 2: Kotlin
 3: Groovy
 4: Scala
 5: C++
 6: Swift
Enter selection (default: Java) [1..6] 1

Enter target Java version (min: 7, default: 21): 21

Project name (default: java_app): javagradle

Select application structure:
 1: Single application project
 2: Application and library project
Enter selection (default: Single application project) [1..2] 1

Select build script DSL:
 1: Kotlin
 2: Groovy
Enter selection (default: Kotlin) [1..2] 2

Select test framework:
 1: JUnit 4
 2: TestNG
 3: Spock
 4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] yes

> Task :init
To learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.7/samples/sample\_building\_java\_applications.html

BUILD SUCCESSFUL in 3m 51s
1 actionable task: 1 executed
C:\Users\IT20\java_app>gradlew build
Downloading https://services.gradle.org/distributions/gradle-8.7-bin.zip
.....10%.....20%.....30%.....40%.....50%.....60%.....70%.....80%.....
..90%.....100%

BUILD SUCCESSFUL in 3m 44s
7 actionable tasks: 7 executed
C:\Users\IT20\java_app>gradlew tasks

> Task :tasks

-----
Tasks runnable from root project 'javagradle'
-----

Application tasks
-----
run - Runs this project as a JVM application

Build tasks
-----
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.

```

```

jar - Assembles a jar archive containing the classes of the 'main' feature.
testClasses - Assembles test classes.

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Distribution tasks
-----
assembleDist - Assembles the main distributions
distTar - Bundles the project as a distribution.
distZip - Bundles the project as a distribution.
installDist - Installs the project as a distribution as-is.

Documentation tasks
-----
javadoc - Generates Javadoc API documentation for the 'main' feature.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root project 'javagradle'.
dependencies - Displays all dependencies declared in root project 'javagradle'.
dependencyInsight - Displays the insight into a specific dependency in root project 'javagradle'.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project 'javagradle'.
projects - Displays the sub-projects of root project 'javagradle'.
properties - Displays the properties of root project 'javagradle'.
resolvableConfigurations - Displays the configurations that can be resolved in root project 'javagradle'.
tasks - Displays the tasks runnable from root project 'javagradle' (some of the displayed tasks may belong to subprojects).

Verification tasks
-----
check - Runs all checks.
test - Runs the test suite.

To see all tasks and more detail, run gradlew tasks --all

To see more detail about a task, run gradlew help --task <task>

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
C:\Users\IT20\java_app>gradlew run

> Task :app:run
Hello World!

BUILD SUCCESSFUL in 2s
2 actionable tasks: 1 executed, 1 up-to-date

```

## Conclusion

Thus, a simple Gradle application was successfully created and build.

**Date:**

**Ex No: 7**

## **Create an Ansible playbook for a simple web application infrastructure**

### **Aim**

To create an Ansible playbook for a simple web application infrastructure

### **Procedure**

1. Install Ansible
2. Update the host file to add the managed host Ips
3. Create a playbook
4. Run the playbook using `$ansible-playbook playbookname.yml`

### **Output**

To check the installation

A terminal window with a dark background and light text. The prompt is 'root@it20:/home/student#'. The command 'ansible --version' has been executed, resulting in the following output: 'ansible [core 2.12.10]', 'config file = /etc/ansible/ansible.cfg', 'configured module search path = [\'/root/.ansible/plugins/modules\', \'/usr/share/ansible/plugins/modules\']', 'ansible python module location = /usr/lib/python3/dist-packages/ansible', 'ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections', 'executable location = /usr/bin/ansible', 'python version = 3.8.10 (default, Nov 22 2023, 10:22:35) [GCC 9.4.0]', 'jinja version = 2.10.1', 'libyaml = True', and the prompt 'root@it20:/home/student#' again.

```
root@it20:/home/student# ansible --version
ansible [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Nov 22 2023, 10:22:35) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
root@it20:/home/student#
```

Host file updation

`$gedit /etc/ansible/hosts`

```
root@it20: /etc/ansible

# Note that this file was always incomplete and lagging changes to configuration settings
# for example, for 2.9: https://github.com/ansible/ansible/blob/stable-2.9/examples/ansible.cfg
root@it20: /etc/ansible# cat hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups
[windows]
192.168.8.130
192.168.8.140
[windows:vars]
ansible_user= IT20
ansible_password= 1
ansible_port= 5986
ansible_connection= winrm
ansible_winrm_server_cert_validation= ignore

# Ex 1: Ungrouped hosts, specify before any group headers:

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

# If you have multiple hosts followed by a software version, you can specify
```

## \$ansible main.yml

```
main.yml
/etc/ansible/roles/mysql/tasks

1 ---
2 # tasks file for mysql
3 - name: My first play
4   hosts: all
5   tasks:
6     - name: Ping my hosts
7       ansible.builtin.win_ping: null
8     - name: Print message
9       ansible.builtin.debug:
10         msg: Hello world
11     - name: Start the MySQL service
12       ansible.builtin.service:
13         name: mysql
14         state: started

YAML Tab Width: 8 Ln 14, Col 22 INS
```

## Run the playbook

### \$ansible-playbook main.yml

```
root@it20:/etc/ansible/roles/mysql/tasks# ansible-playbook main.yml

PLAY [My first play] *****

TASK [Gathering Facts] *****
ok: [192.168.8.140]
fatal: [192.168.8.130]: UNREACHABLE! => {"changed": false, "msg": "ssl: HTTPConnectionPool(host='192.168.8.130', port=5986): Max retries exceeded with url: /wsman (Caused by ConnectTimeoutError(<urllib3.connection.VerifiedHTTPSConnection object at 0x7fb5421b0d30>, 'Connection to 192.168.8.130 timed out. (connect timeout=30)'))", "unreachable": true}

TASK [Ping my hosts] *****
ok: [192.168.8.140]

TASK [Print message] *****
ok: [192.168.8.140] => {
  "msg": "Hello world"
}

PLAY RECAP *****
192.168.8.130      : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0
192.168.8.140      : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

## Conclusion

Thus, an Ansible playbook for a simple application infrastructure was created and tested the code on managed hosts



**Date:**

**Ex No:8**

## **Install Ansible and configure ansible roles and to write playbooks**

### **Aim**

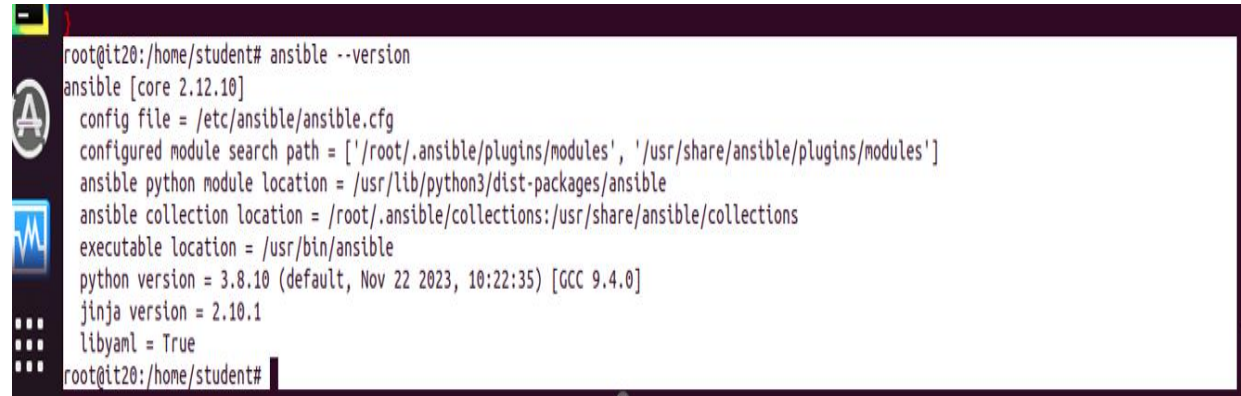
To install Ansible and configure ansible roles and to write playbooks

### **Procedure**

1. Install Ansible
2. Update the host file to add the managed hostIPs
3. Create new roles
4. Create a playbook under the roles
5. Run the playbook using \$ansible-playbook playbookname.yml

### **Output**

To check the installation

A terminal window with a dark background and light text. The prompt is 'root@it20:/home/student#'. The command 'ansible --version' has been executed, resulting in the following output: 'ansible [core 2.12.10]', 'config file = /etc/ansible/ansible.cfg', 'configured module search path = [\'/root/.ansible/plugins/modules\', \'/usr/share/ansible/plugins/modules\']', 'ansible python module location = /usr/lib/python3/dist-packages/ansible', 'ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections', 'executable location = /usr/bin/ansible', 'python version = 3.8.10 (default, Nov 22 2023, 10:22:35) [GCC 9.4.0]', 'jinja version = 2.10.1', and 'libyaml = True'. The prompt 'root@it20:/home/student#' is visible again at the bottom.

```
root@it20:/home/student# ansible --version
ansible [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = [\'/root/.ansible/plugins/modules\', \'/usr/share/ansible/plugins/modules\']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Nov 22 2023, 10:22:35) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
root@it20:/home/student#
```

Host file updation

\$gedit /etc/ansible/hosts

```
root@it20: /etc/ansible
# Note that this file was always incomplete and lagging changes to configuration settings
# for example, for 2.9: https://github.com/ansible/ansible/blob/stable-2.9/examples/ansible.cfg
root@it20: /etc/ansible# cat hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups
[windows]
192.168.8.130
192.168.8.140
[windows:vars]
ansible_user= IT20
ansible_password= 1
ansible_port= 5986
ansible_connection= winrm
ansible_winrm_server_cert_validation= ignore

# Ex 1: Ungrouped hosts, specify before any group headers:

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110
# If you have multiple hosts following a pattern, you can specify
```

## To view the files in ansible

```
>> cd /etc/ansible
```

```
>>ls
```

```
hosts ansible.cfg roles
```

```
>>roles
```

```
>>$ ansible-galaxy init apache
```

```
>>ls
```

```
>>tree apache
```

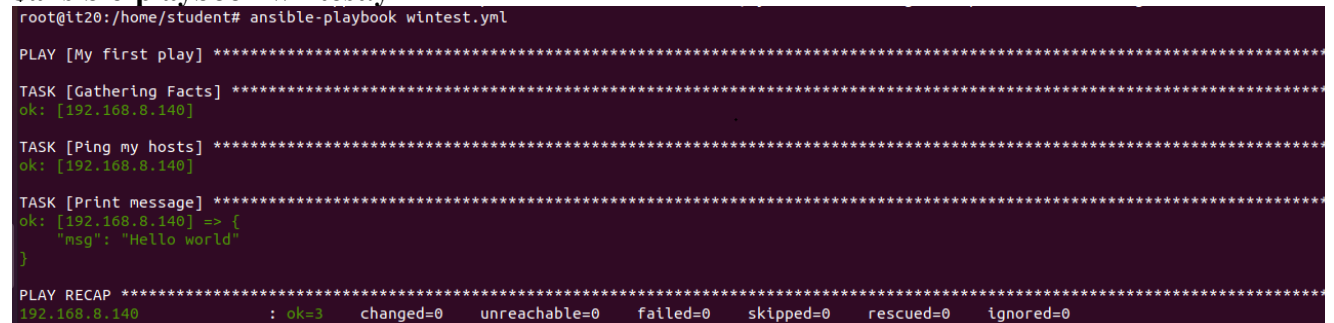
## \$ansible main.yml



```
1 ---
2 # tasks file for mysql
3 - name: My first play
4   hosts: all
5   tasks:
6     - name: Ping my hosts
7       ansible.builtin.win_ping: null
8     - name: Print message
9       ansible.builtin.debug:
10         msg: Hello world
11     - name: Start the MySQL service
12       ansible.builtin.service:
13         name: mysql
14       state: started
```

## Run the playbook

### \$ansible-playbook wintest.yml



```
root@it20:/home/student# ansible-playbook wintest.yml

PLAY [My first play] *****
TASK [Gathering Facts] *****
ok: [192.168.8.140]
TASK [Ping my hosts] *****
ok: [192.168.8.140]
TASK [Print message] *****
ok: [192.168.8.140] => {
  "msg": "Hello world"
}
PLAY RECAP *****
192.168.8.140 : ok=3  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

## Conclusion

Thus, an ansible roles are configured and new playbooks are created and executed successfully