

## TABLE OF CONTENTS

<b>Ex No</b>	<b>Date</b>	<b>Title</b>	<b>Page No</b>	<b>Signature</b>
01		Create Maven Build Pipeline in Azure		
02		Run regression tests using Maven Build pipeline in Azure		
03		Install Jenkins in Cloud		
04		Create CI pipeline using Jenkins		
05		Build a simple application using Gradle		
06		Install Ansible and Configure Ansible Roles and to write playbooks		
07		Create a CD pipeline in Jenkins and deploy in Cloud		
08		Create an Ansible playbook for a simple web application infrastructure		

**Ex.No:1**

# Create Maven Build Pipeline in Azure

**Date:**

**Aim:**

To create maven build pipeline in Azure

**Procedure:**

Step 1: Install Apache Maven Zip file

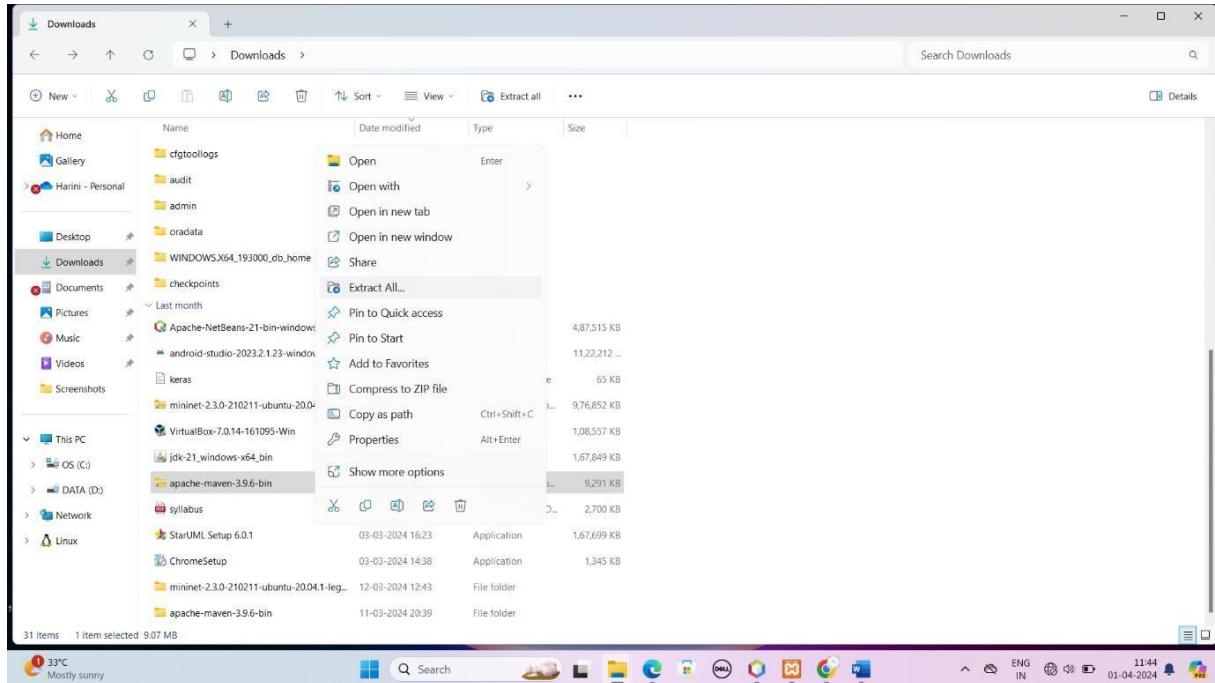
The screenshot shows the Apache Maven download page. It includes sections for Memory, Disk, and Operating System requirements. A table lists four Maven releases (3.9.6) with their respective links, checksums, and signatures. Below the table, there's a section for "Other Releases" with a note about using the latest version.

Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.9.6-bin.tar.gz	apache-maven-3.9.6-bin.tar.gz.asc
Binary zip archive	apache-maven-3.9.6-bin.zip	apache-maven-3.9.6-bin.zip.asc
Source tar.gz archive	apache-maven-3.9.6-src.tar.gz	apache-maven-3.9.6-src.tar.gz.asc
Source zip archive	apache-maven-3.9.6-src.zip	apache-maven-3.9.6-src.zip.asc

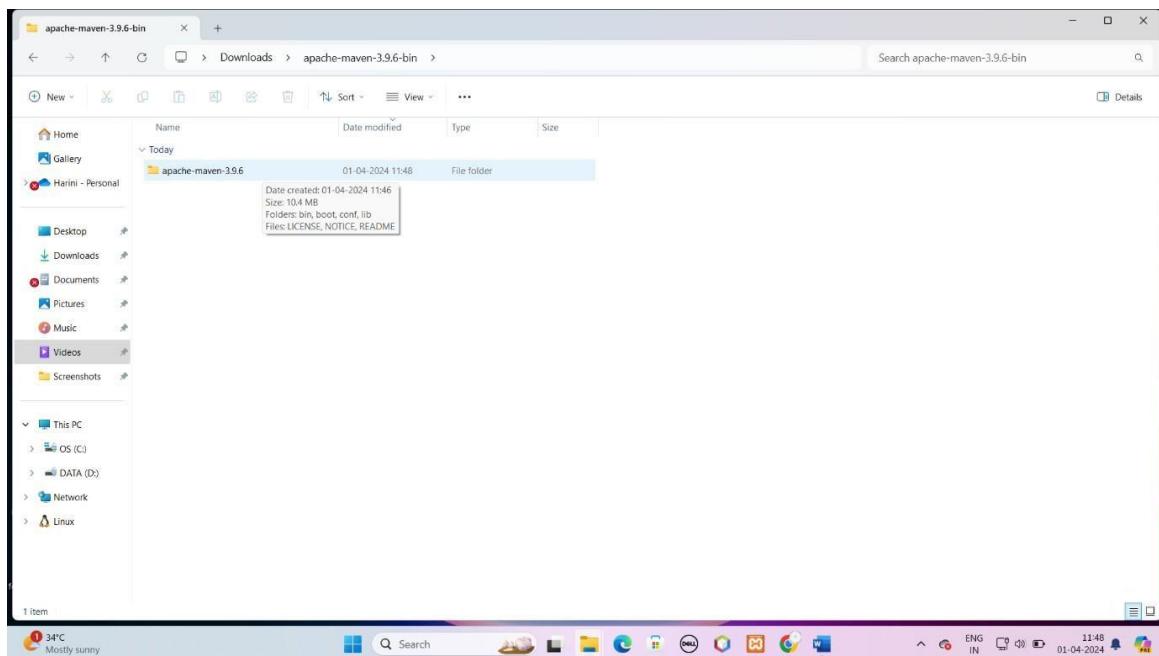
Step 2: Go to folder where apache maven downloaded

The screenshot shows a Windows File Explorer window with the 'Downloads' folder selected. The folder contains several files and folders, including 'Apache-NetBeans-21-bin-windows-x64', 'android-studio-2023.2.1-23-windows', 'keras', 'mininet-2.3.0-210211-ubuntu-20.04.1-leg...', 'VirtualBox-7.0.14-161095-Win', 'jdk-21-windows-x64\_bin', 'apache-maven-3.9.6-bin', 'syllabus', 'StarUML Setup 6.0.1', 'ChromeSetup', 'mininet-2.3.0-210211-ubuntu-20.04.1-leg...', and 'apache-maven-3.9.6-bin'. The 'apache-maven-3.9.6-bin' folder is highlighted.

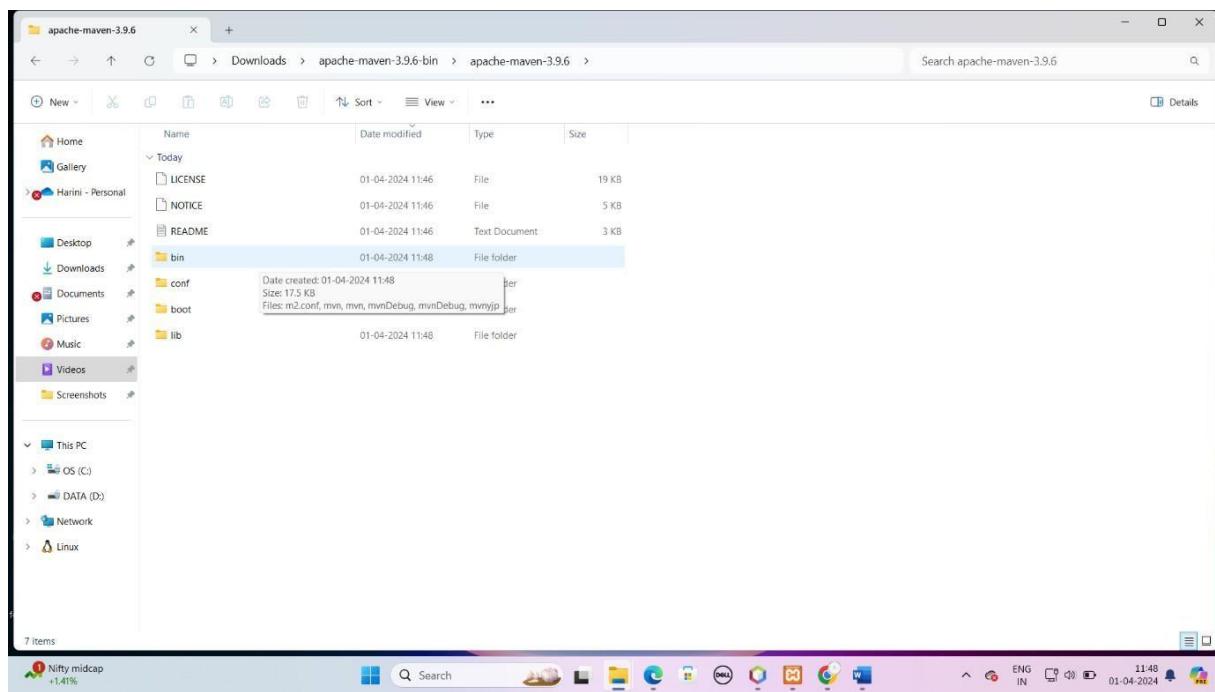
### Step 3: Right Click & Extract the Zip file



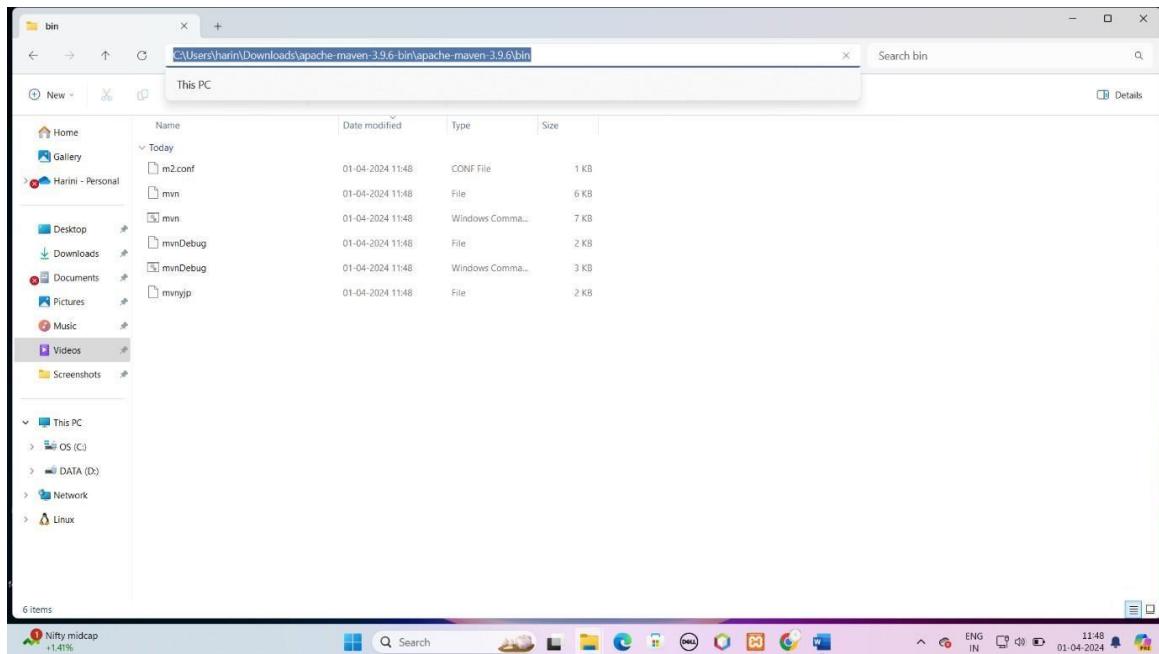
### Step 4: Click the Extracted file



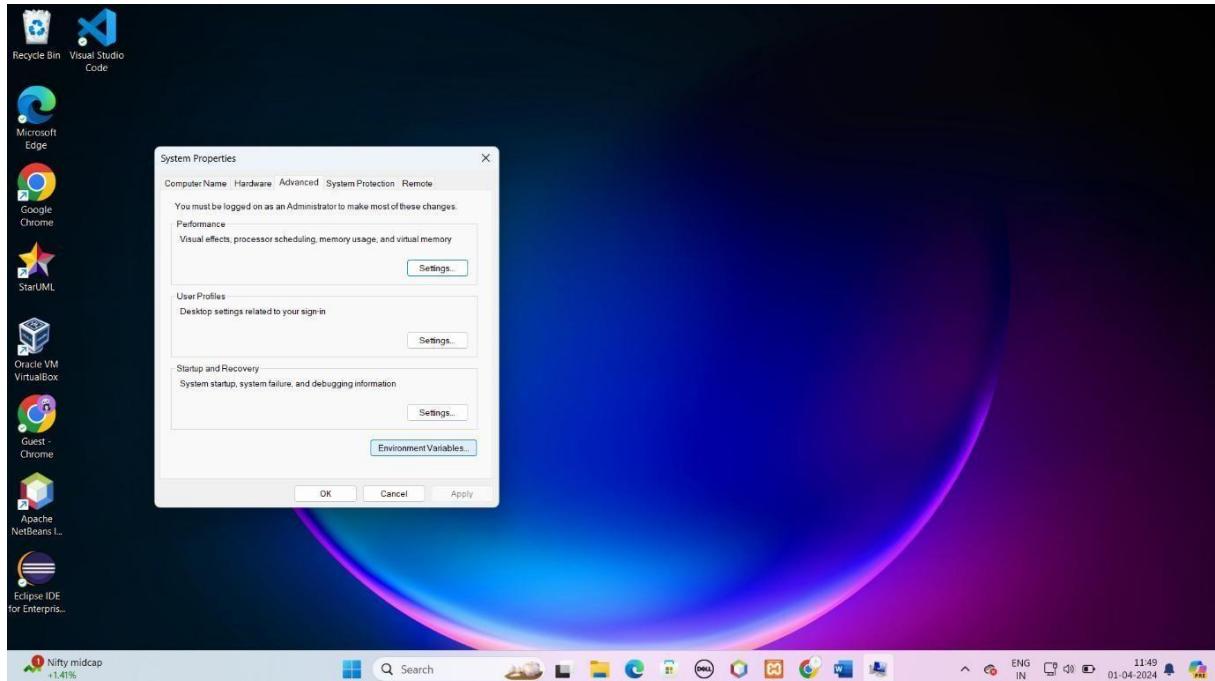
## Step 5: Select bin folder



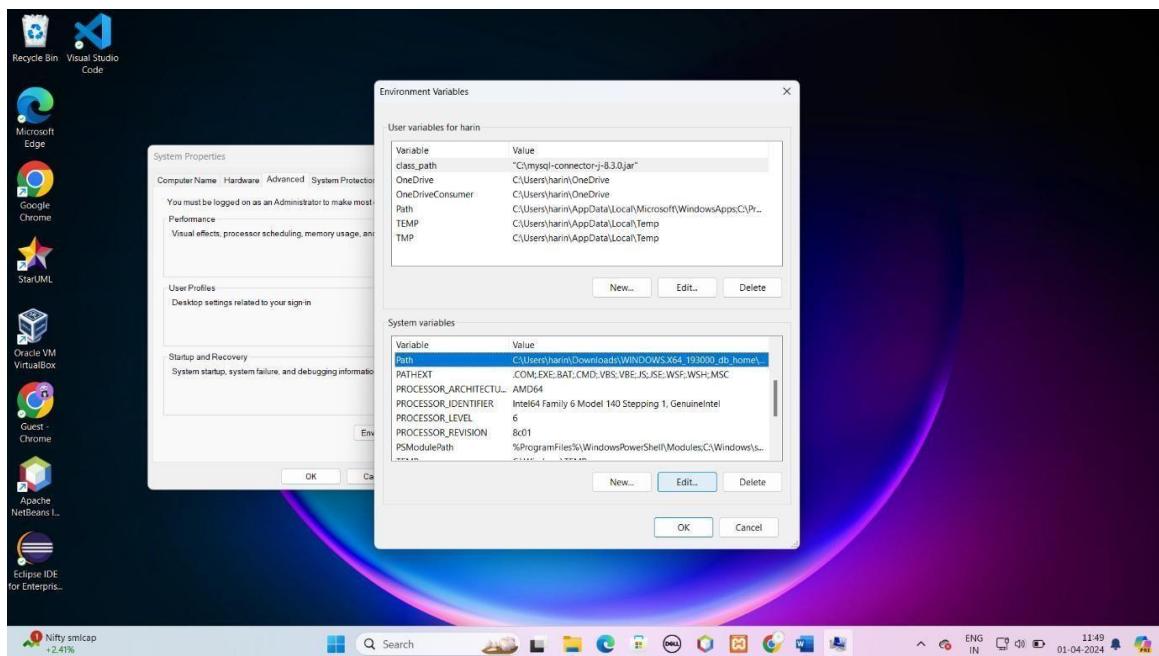
## Step 6: Copy bin folder path



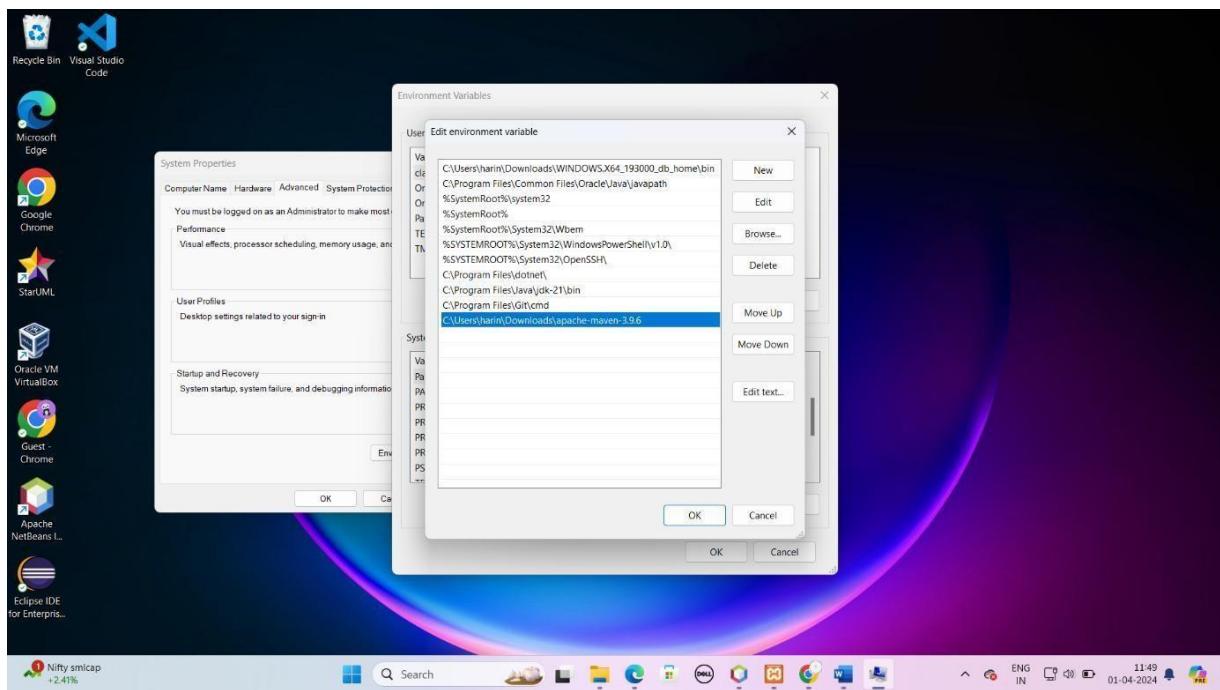
## Step 7: Open Environmental Variable



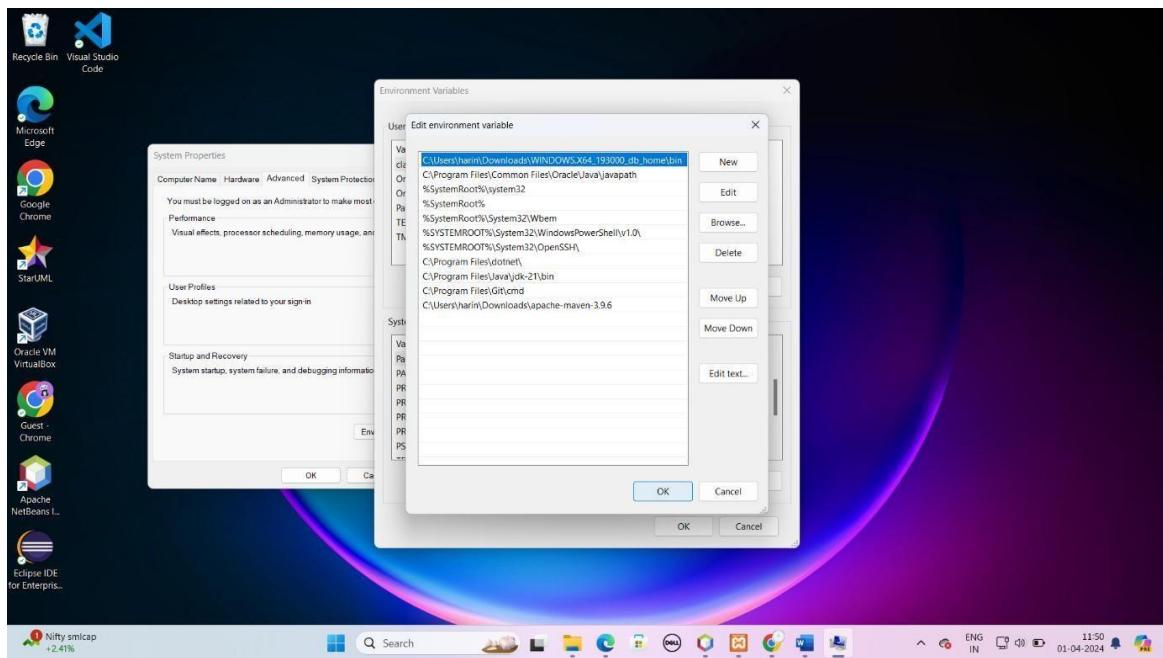
## Step 8: Select path and click edit



## Step 9: Click New and paste the copied path

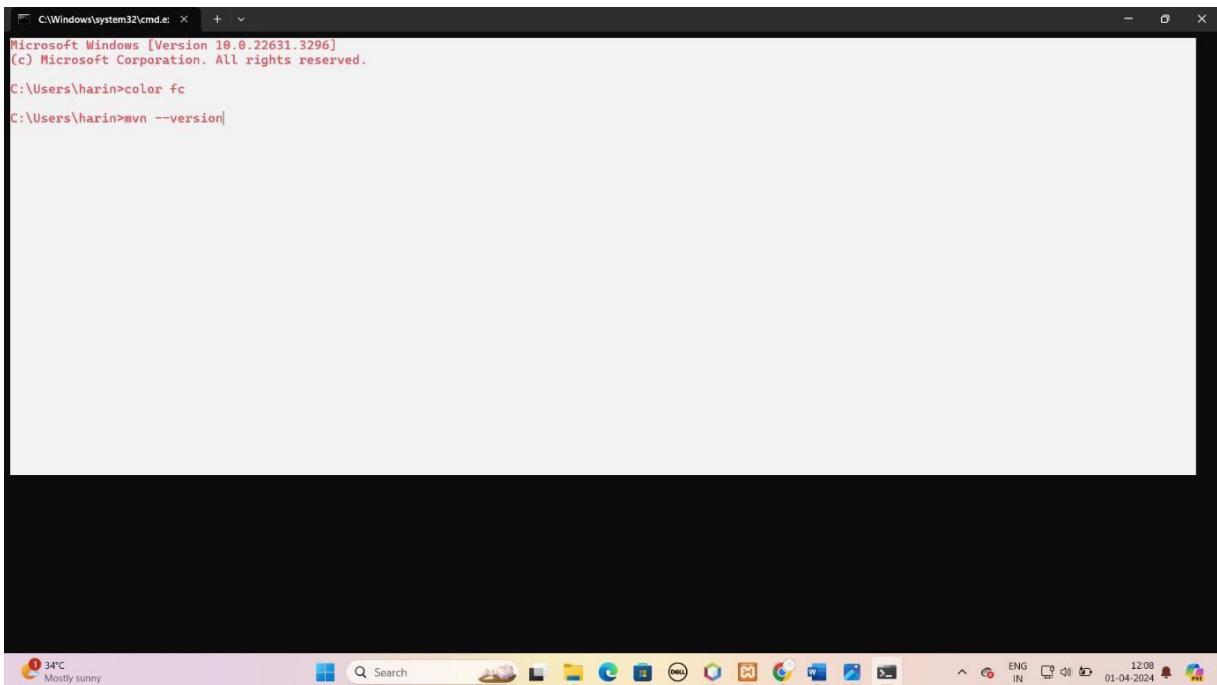


## Step 10: Click ok



## Step 11: Open Command Prompt

>> mvn --version

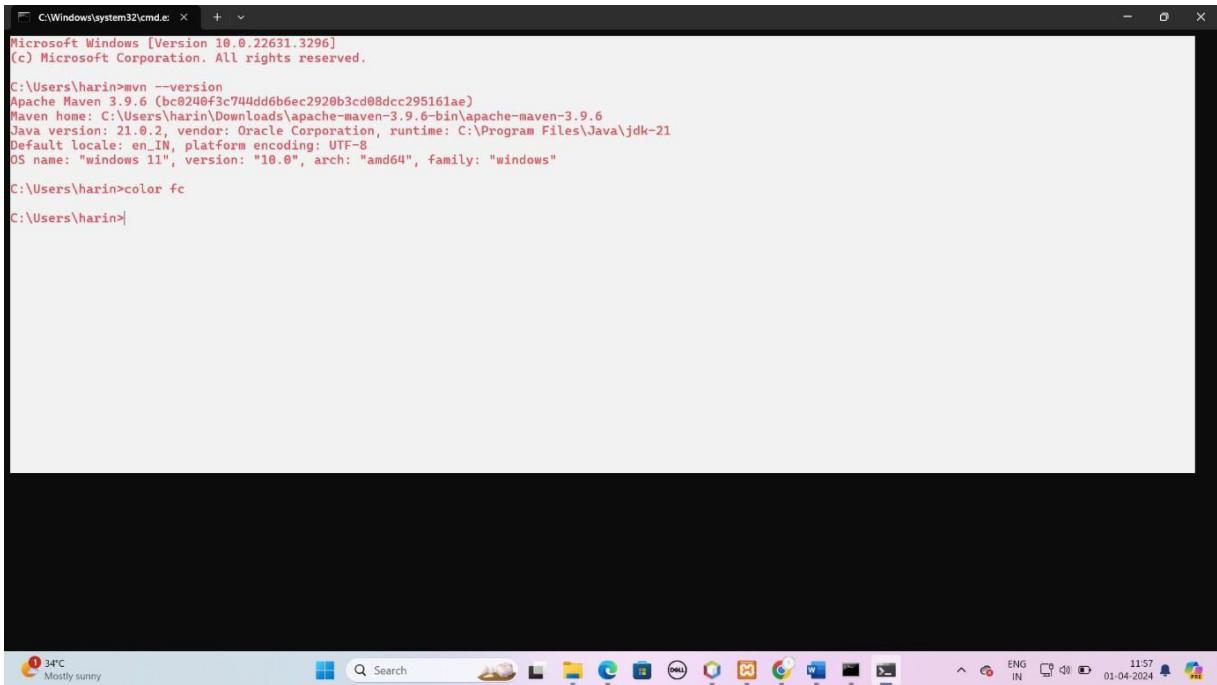


```
C:\Windows\system32\cmd.exe + 
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\harin>color fc
C:\Users\harin>mvn --version|
```

The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command 'mvn --version' has been entered and is being processed. The system tray at the bottom indicates it's 34°C and mostly sunny. The taskbar includes icons for various applications like File Explorer, Edge, and Control Panel.

## Step 12: Maven Verified



```
C:\Windows\system32\cmd.exe + 
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\harin>mvn --version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: C:\Users\harin\Downloads\apache-maven-3.9.6-bin\apache-maven-3.9.6
Java version: 21.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-21
Default locale: en_IN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\harin>color fc
C:\Users\harin>
```

This screenshot shows the same Windows Command Prompt window as the previous one, but the output of 'mvn --version' is now fully displayed. It provides detailed information about the Maven version, Java environment, and operating system. The system tray and taskbar are identical to the first screenshot.

## Result:

Thus, creating maven build pipeline in Azure done successfully.

**Ex.No: 2**

## Run regression tests using Maven Build pipeline in Azure

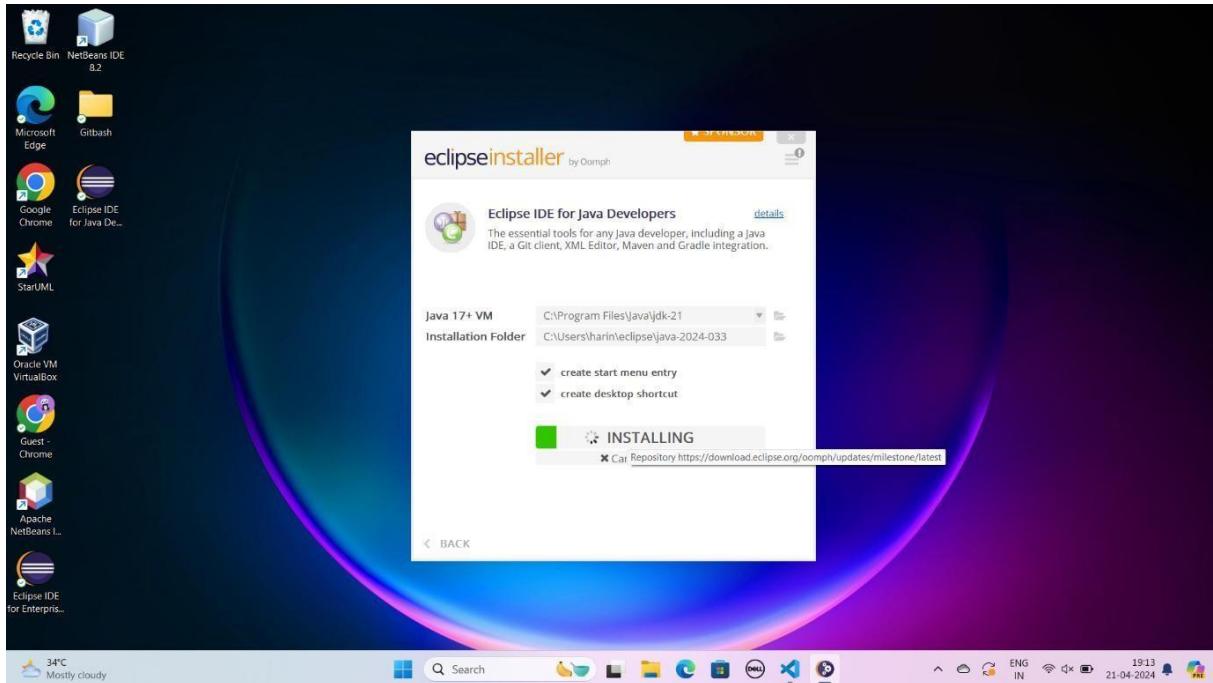
**Date:**

**Aim:**

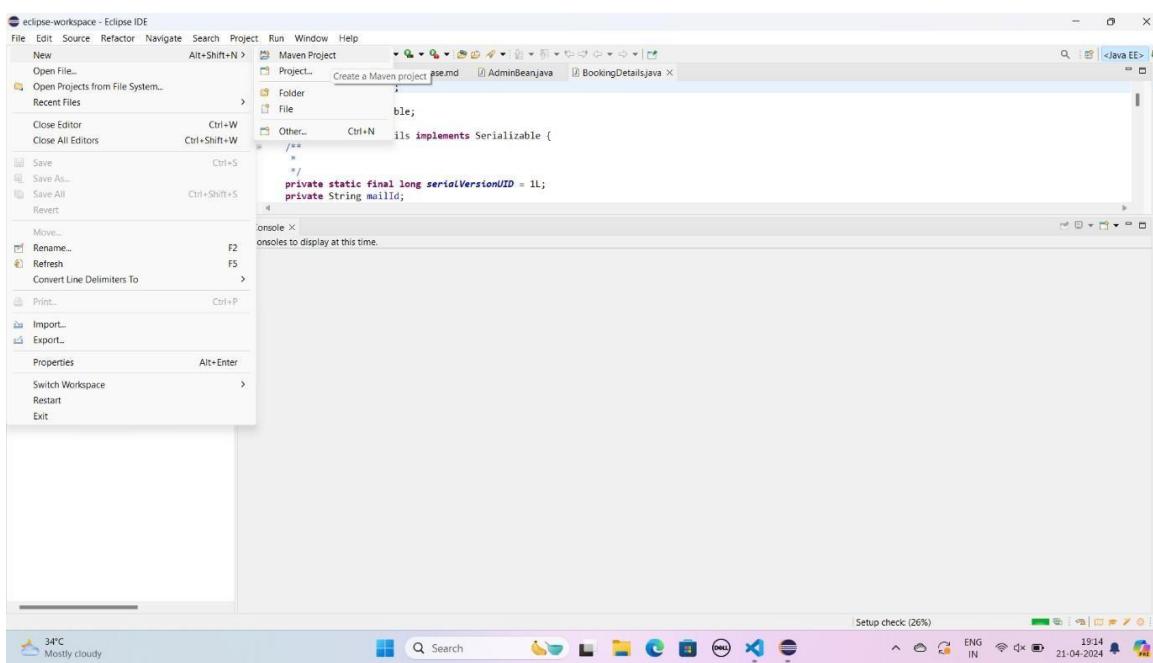
To run regression tests using Maven Build pipeline in Azure.

**Procedure:**

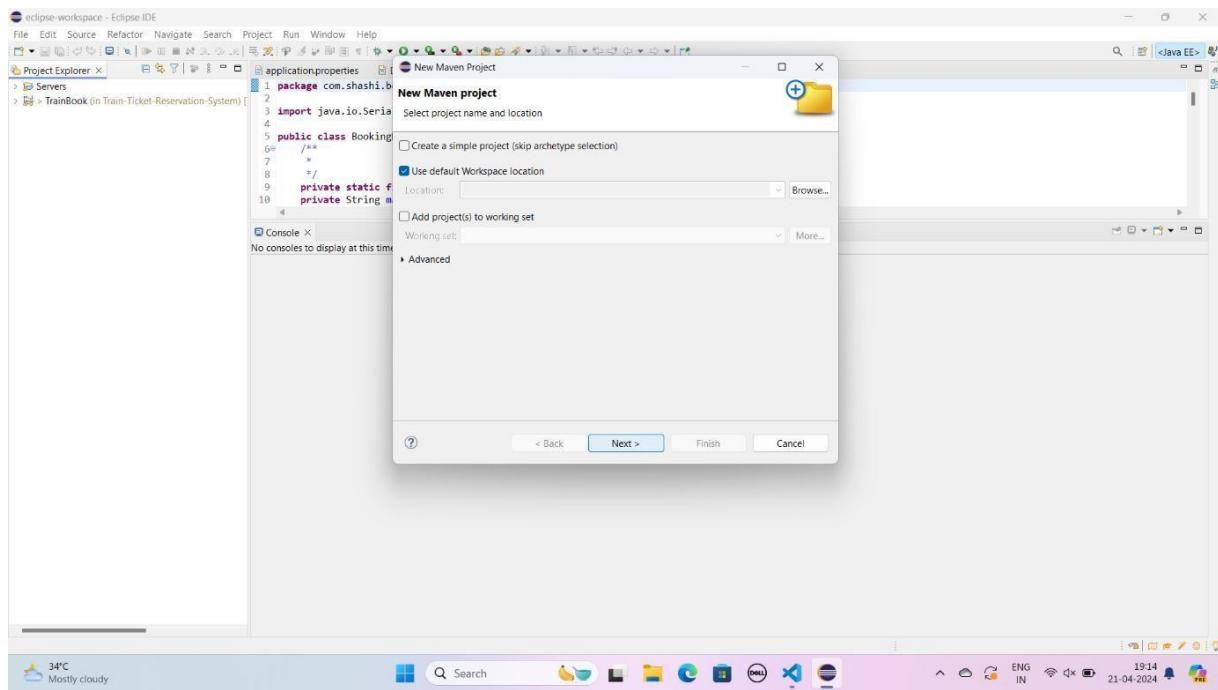
Step 1: Download Eclipse and install Eclipse IDE for java Developers



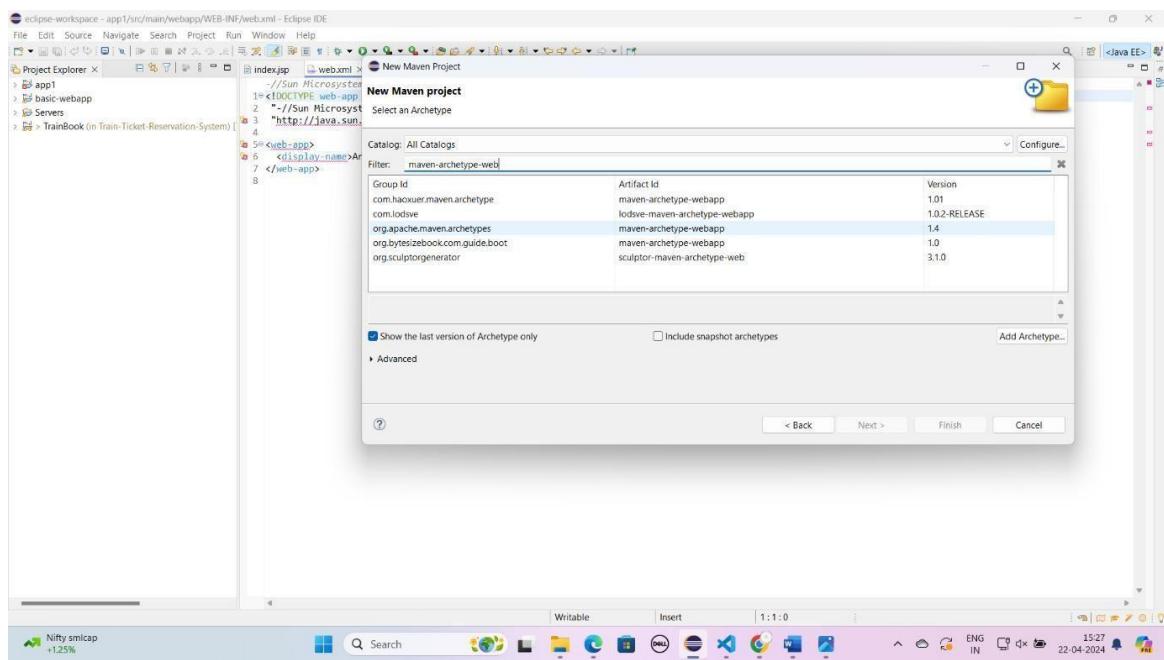
Step 2: Open file select new and create maven project



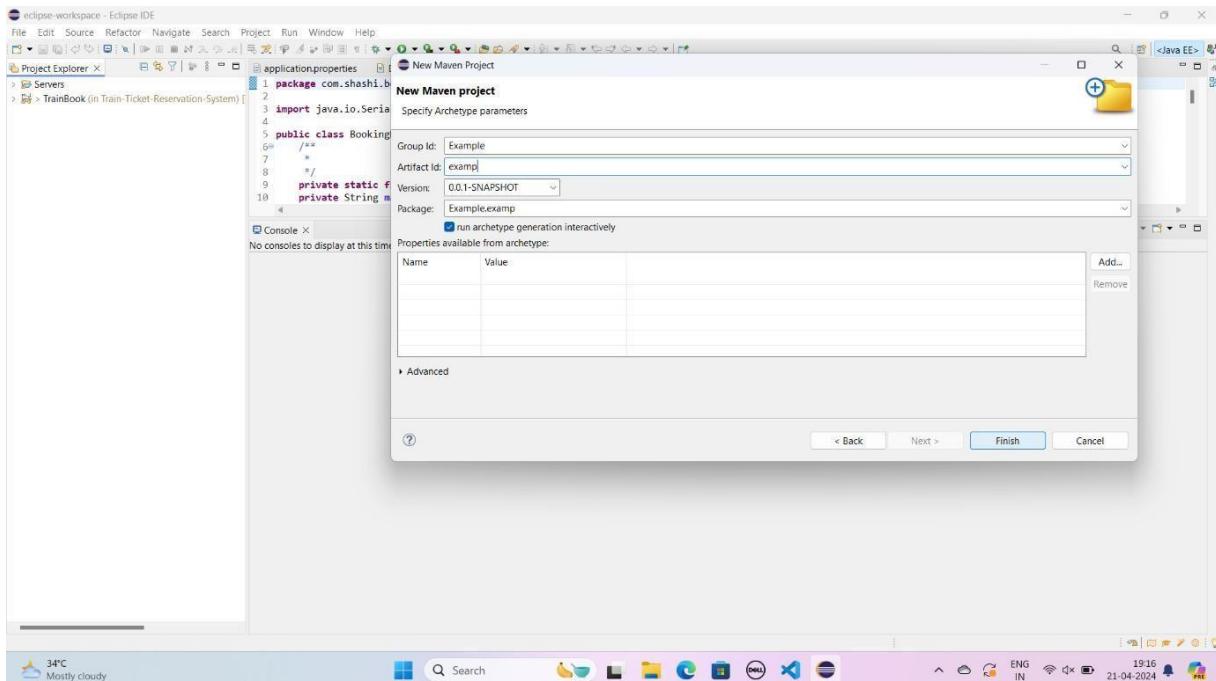
### Step 3: Just Click next



### Step 4: In New Maven project select Maven-archetype-webapp Version 1.4 and Click next



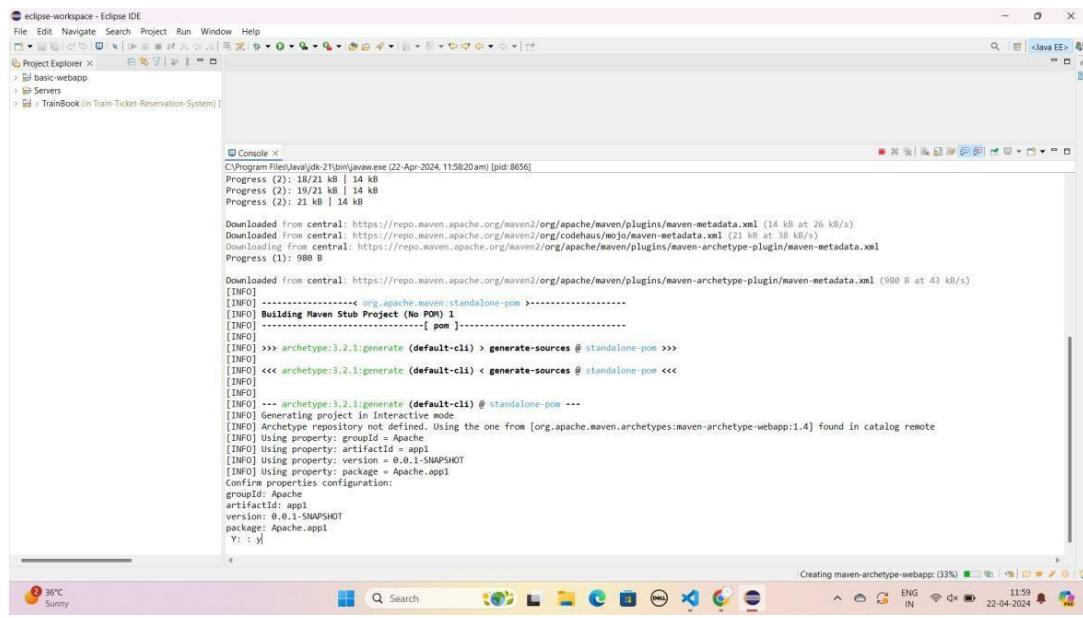
## Step 5: Provide the Group Id and Artifact Id then click finish



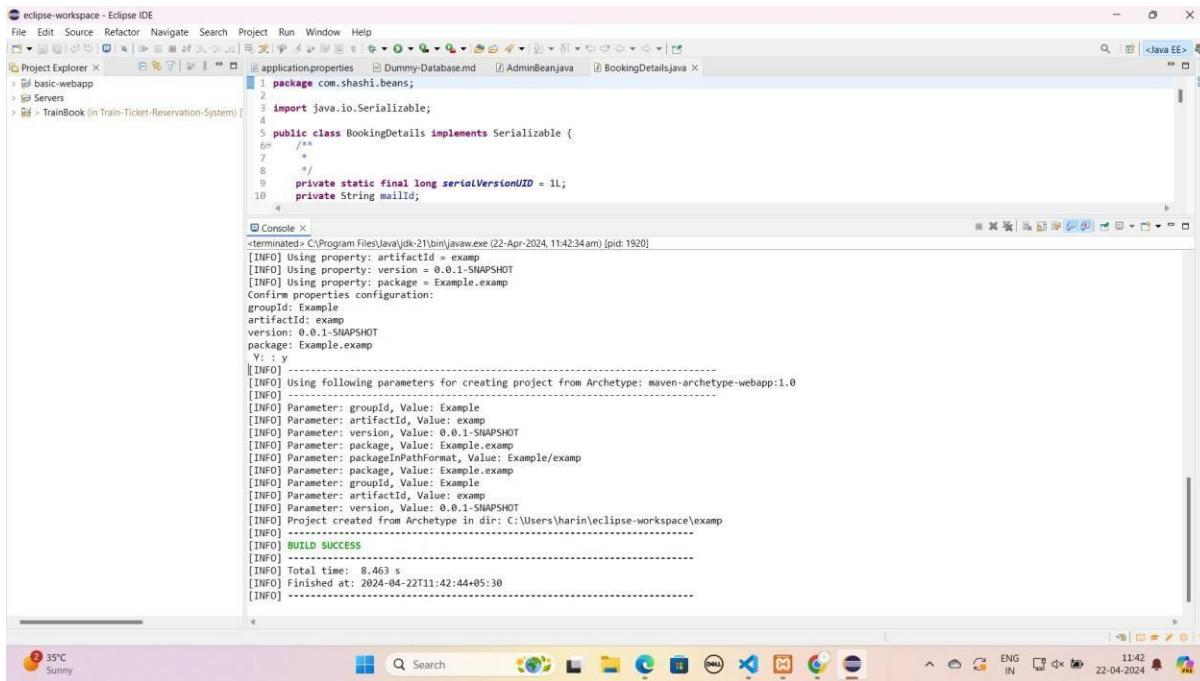
## Step 6: Creating maven-archetype-webapp

At 33% completion it will ask Y :: y

type y and it will continue the process



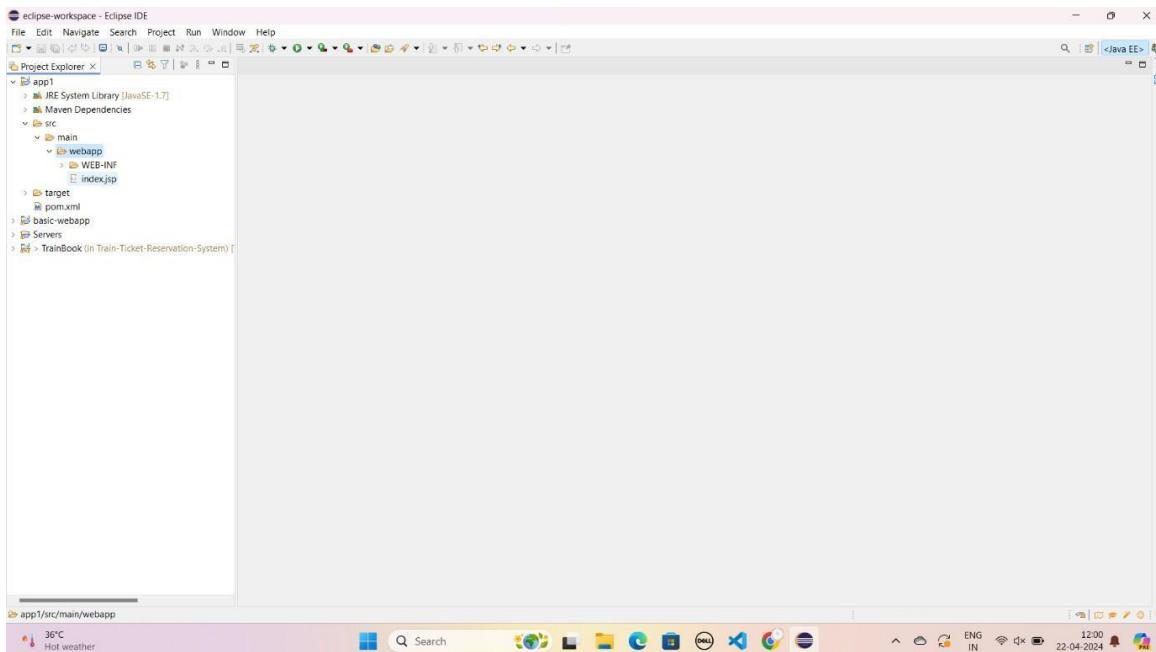
## Step 7: It Shows Build Success



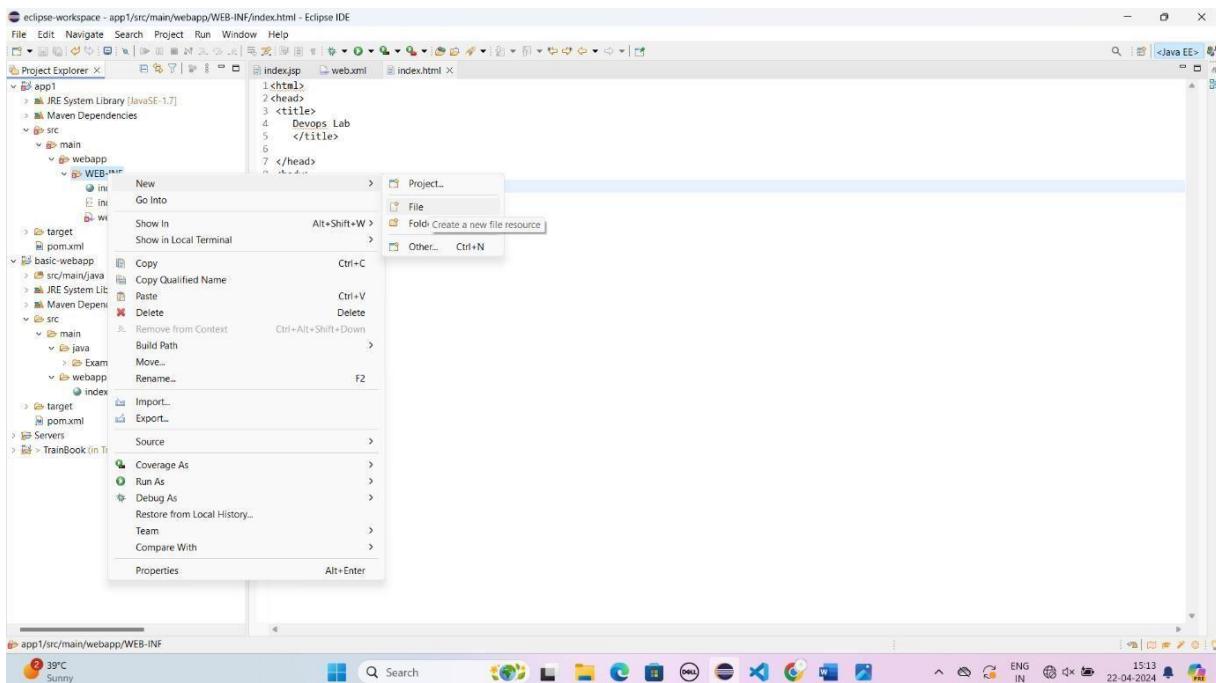
The screenshot shows the Eclipse IDE interface with the title "eclipse-workspace - Eclipse IDE". The Project Explorer view on the left shows a project named "TrainBook (in Train-Ticket-Reservation-System)". In the center, there is a code editor window displaying Java code for a class named "BookingDetails". Below the code editor is a "Console" window showing the output of a Maven build command. The console output indicates a "BUILD SUCCESS" message. The status bar at the bottom shows system information like weather (35°C, Sunny), date (22-04-2024), and time (11:42).

```
application.properties Dummy-Database.md AdminBean.java BookingDetails.java
1 package com.shashi.beans;
2
3 import java.io.Serializable;
4
5 public class BookingDetails implements Serializable {
6     /**
7      */
8     private static final long serialVersionUID = 1L;
9     private String mailId;
10
<terminated> C:\Program Files\Java\jdk-21\bin\javaw.exe (22-Apr-2024, 11:42:34 am) [pid: 1920]
[INFO] Using property: artifactId = examp
[INFO] Using property: version = 0.0.1-SNAPSHOT
[INFO] Using property: package = Example.examp
Confirm properties configuration:
groupId: Example
artifactId: examp
version: 0.0.1-SNAPSHOT
package: Example.examp
V:
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-webapp:1.0
[INFO] -----
[INFO] Parameter: groupId, Value: Example
[INFO] Parameter: artifactId, Value: examp
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[INFO] Parameter: package, Value: Example.example
[INFO] Parameter: packageInPathFormat, Value: Example/examp
[INFO] Parameter: package, Value: Example.example
[INFO] Parameter: artifactId, Value: Example
[INFO] Parameter: version, Value: examp
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[INFO] Project created from Archetype in dir: C:\Users\harin\eclipse-workspace\examp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.463 s
[INFO] Finished at: 2024-04-22T11:42:44+05:30
[INFO] -----
```

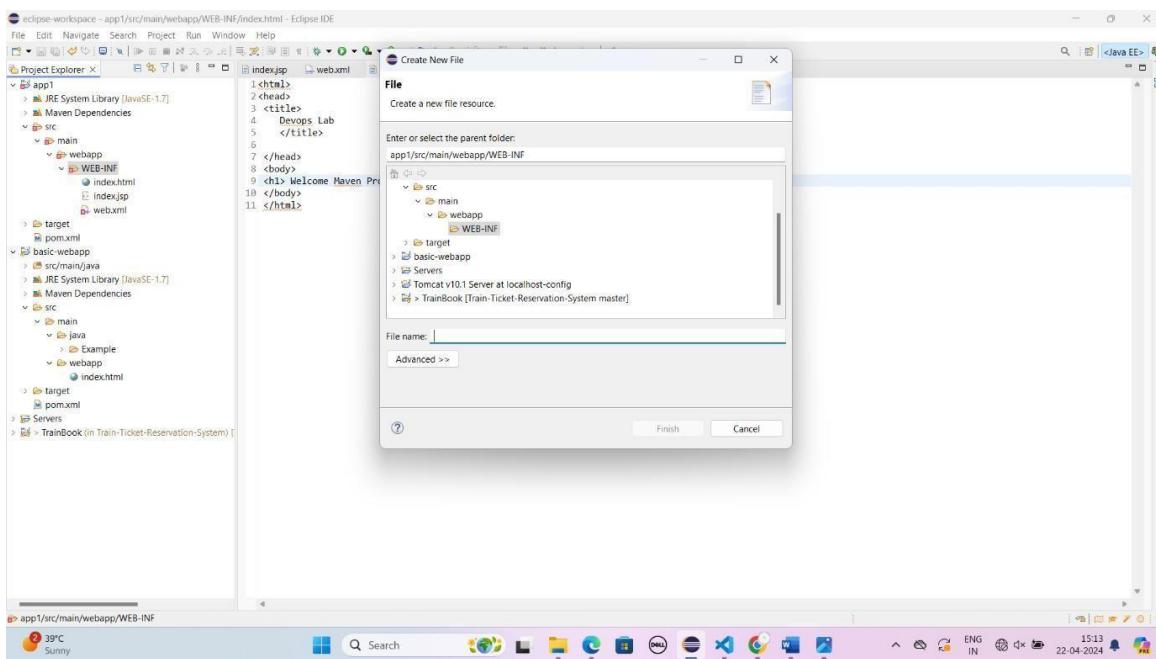
## Step 8: Click app1 -> src -> main -> webapp -> WEB-INF



## Step 9: Right click on WEB-INF select new and click on file



## Step 10: create new file and click finish



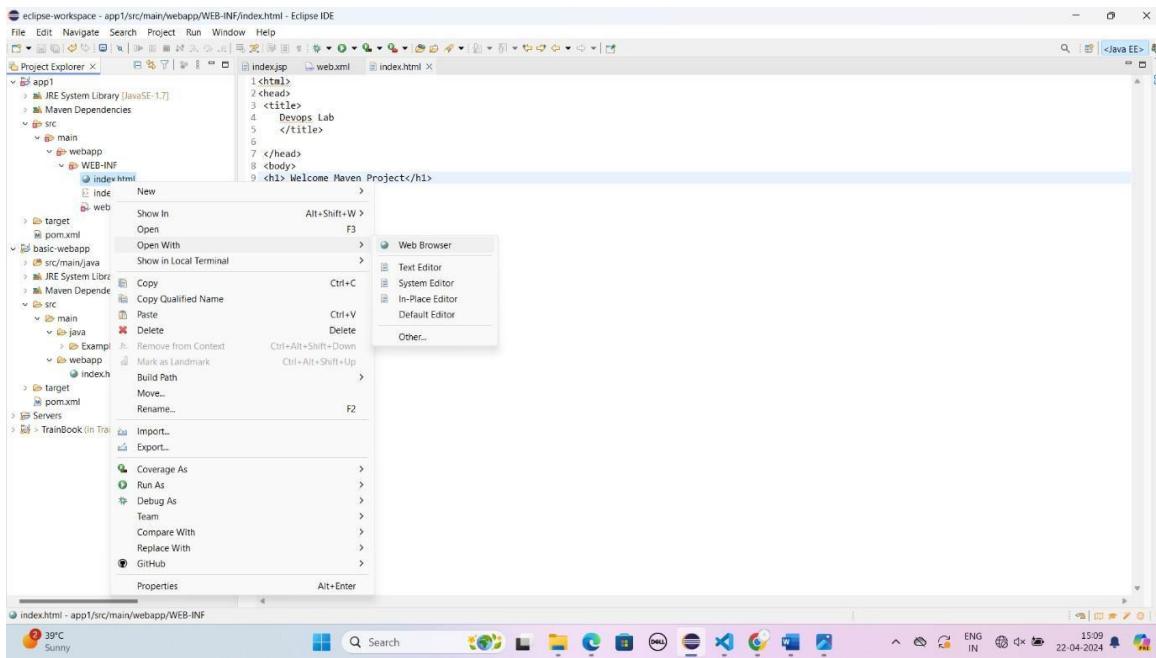
## Step 11: Create a simple HTML program

The screenshot shows the Eclipse IDE interface. The Project Explorer view on the left displays a Maven project named 'app1'. Inside 'src/main/webapp/WEB-INF', there is an 'index.html' file. The code in the editor is:

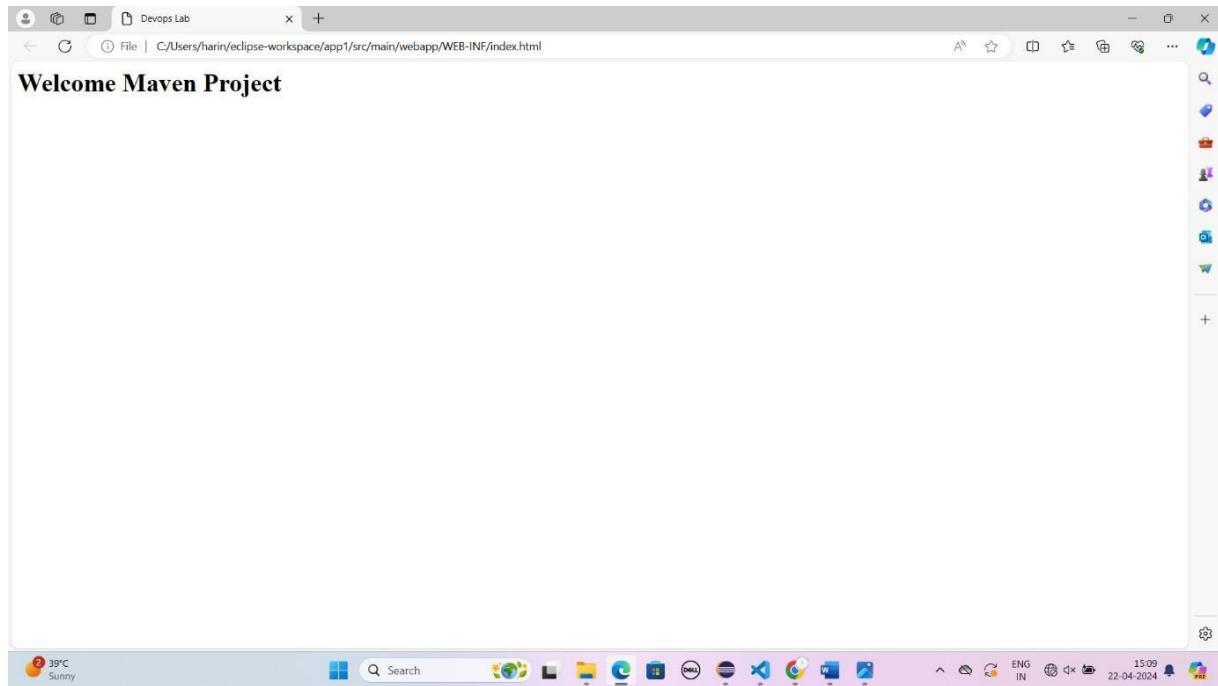
```
<html>
<head>
<title>
    Devops Lab
</title>
</head>
<body>
<h1>Welcome Maven Project</h1>
</body>
</html>
```

The status bar at the bottom shows the date and time as 22-04-2024 15:13.

## Step 12: Run html program in web browser



## Step 13: Output Verified



### Result:

Thus regression tests using Maven Build pipeline in Azure done successfully.

**Ex.No:3**

## Install Jenkins in Cloud

**Date:**

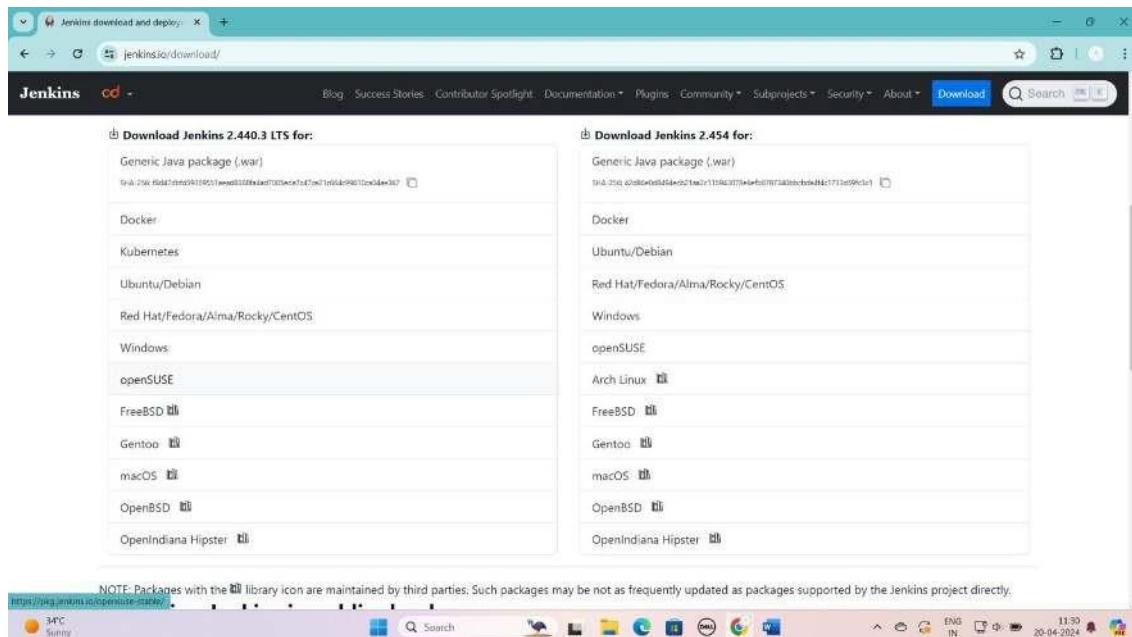
**Aim:**

To install Jenkins in Cloud.

**Procedure:**

Step:1 Download Jenkins from jenkins official page

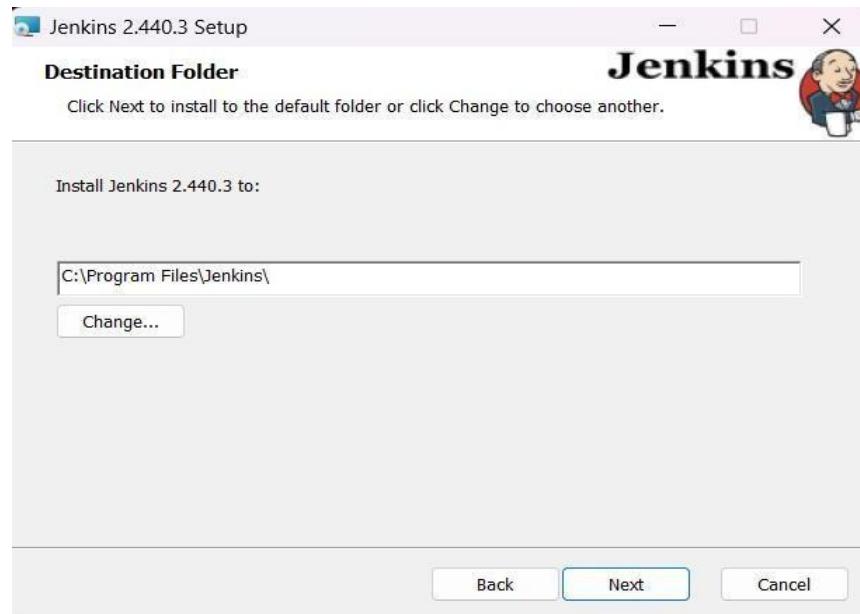
**Note:** Java JDK version must be 11 / 17/ 21 to install Jenkins on windows



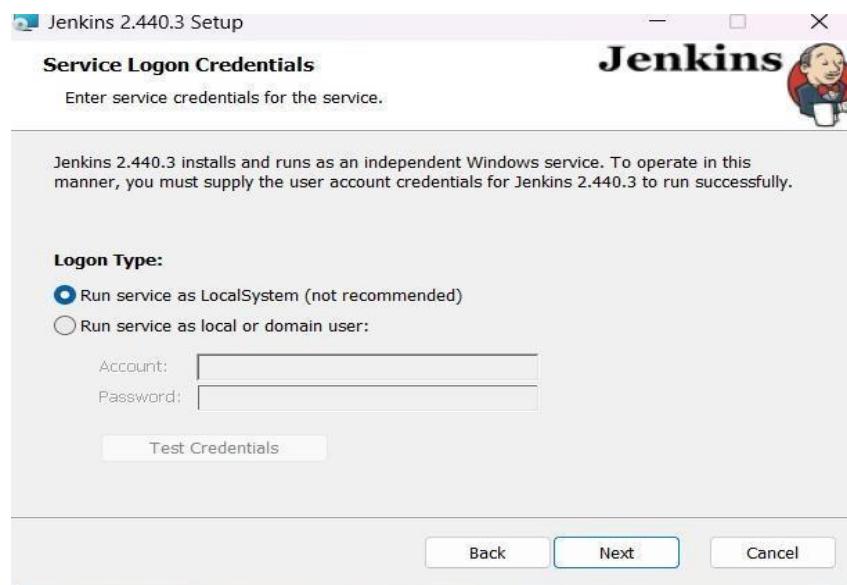
Step:2 Click the downloaded file and run as administrator



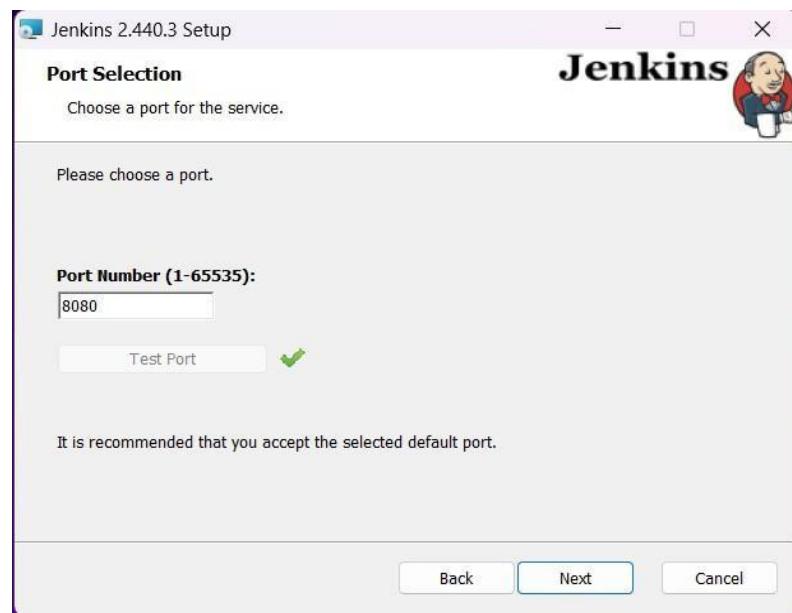
### Step:3 Click next to install



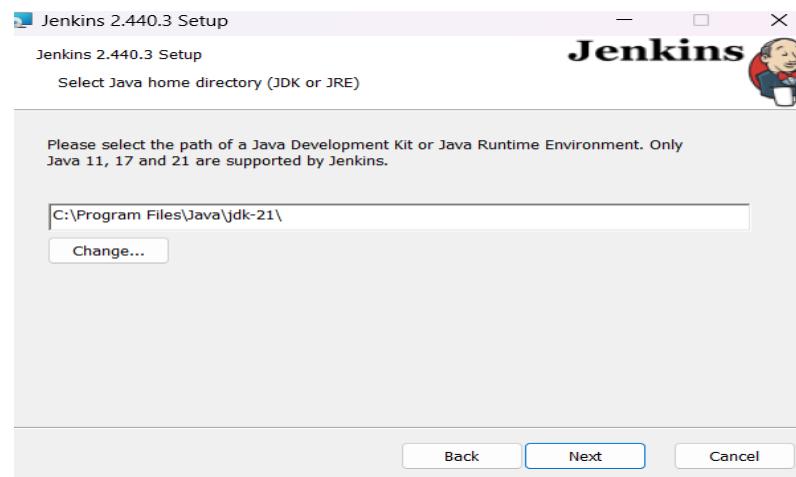
### Step:4 Enter service credentials for the service and click next



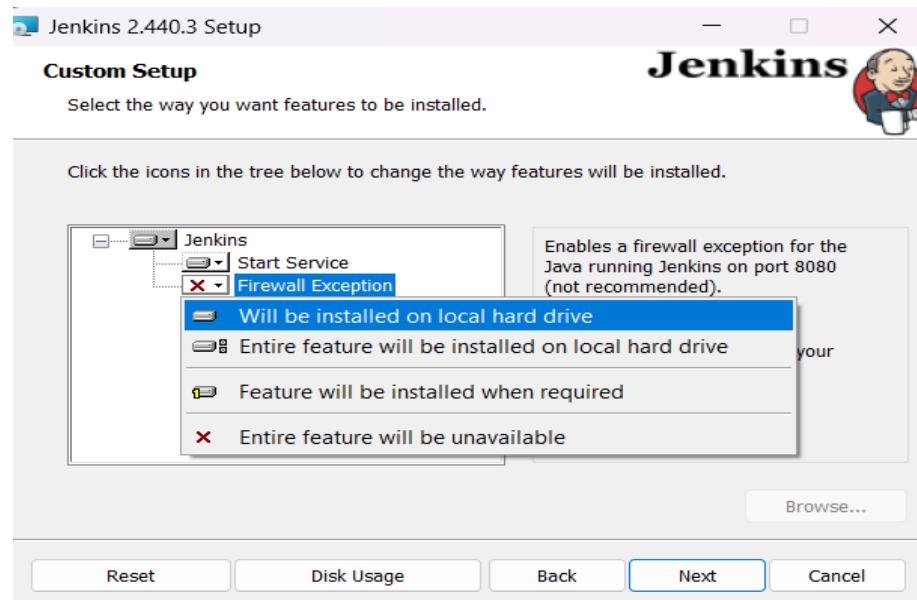
Step:5 Choose a port for the service and click next



Step:6 Select java home directory and click next



Step:7 Select the features to be installed and click next



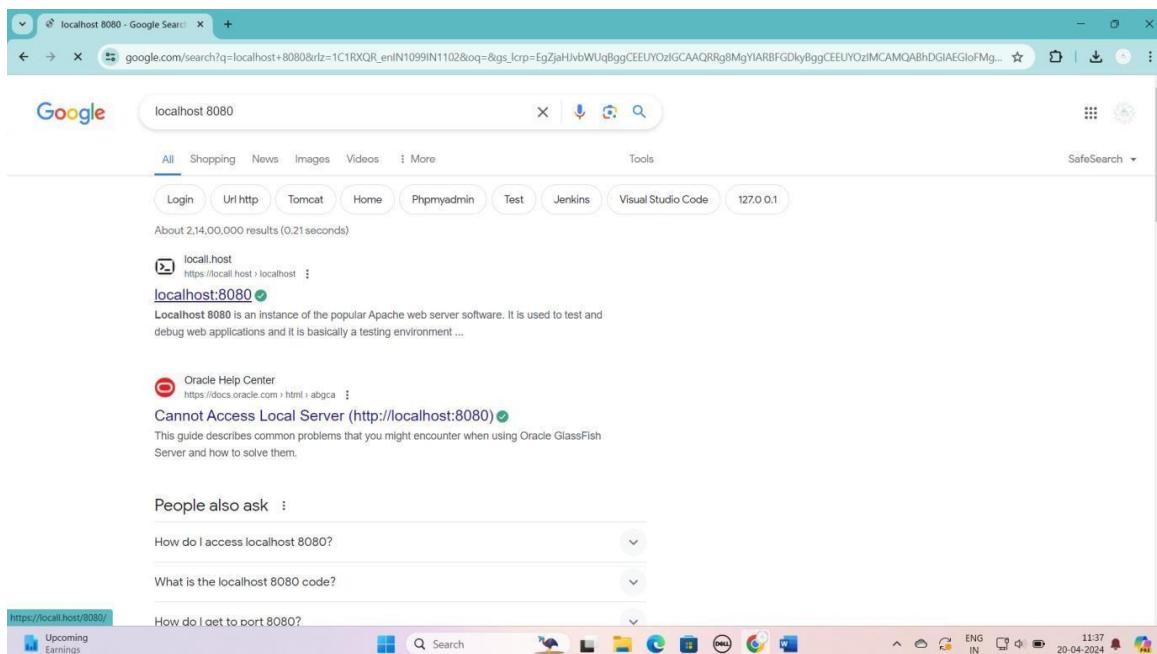
Step:8 Click install to begin the installation



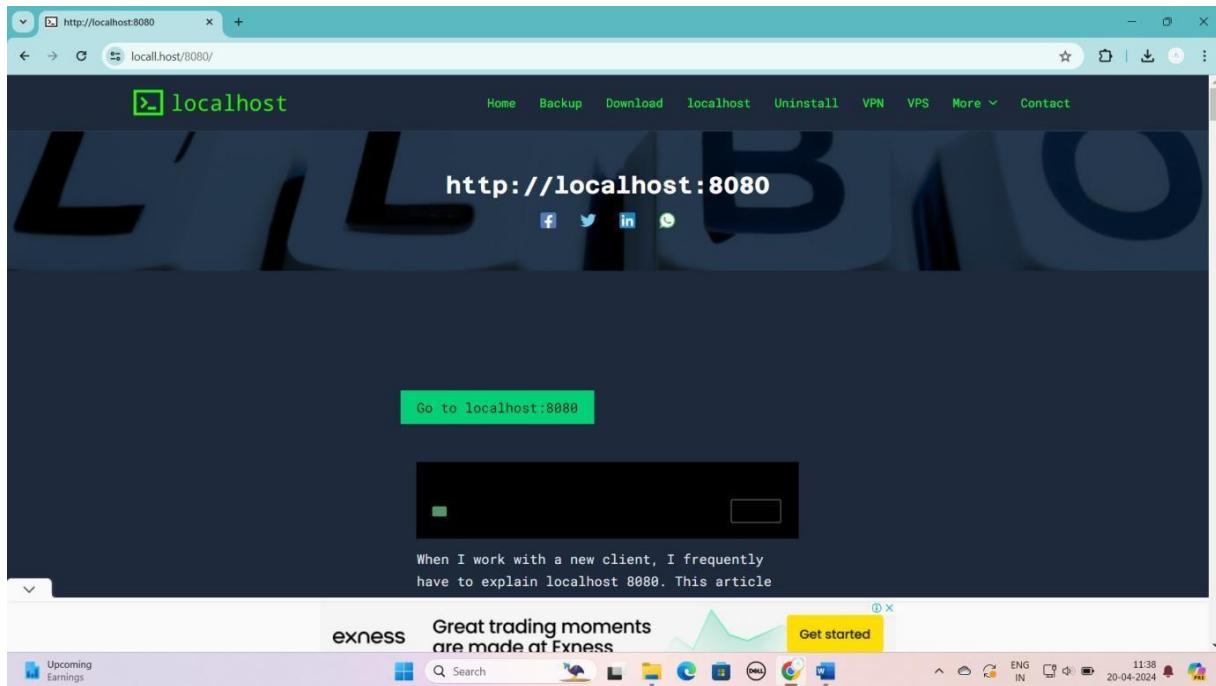
Step:9 Click finish the button to exit the setup



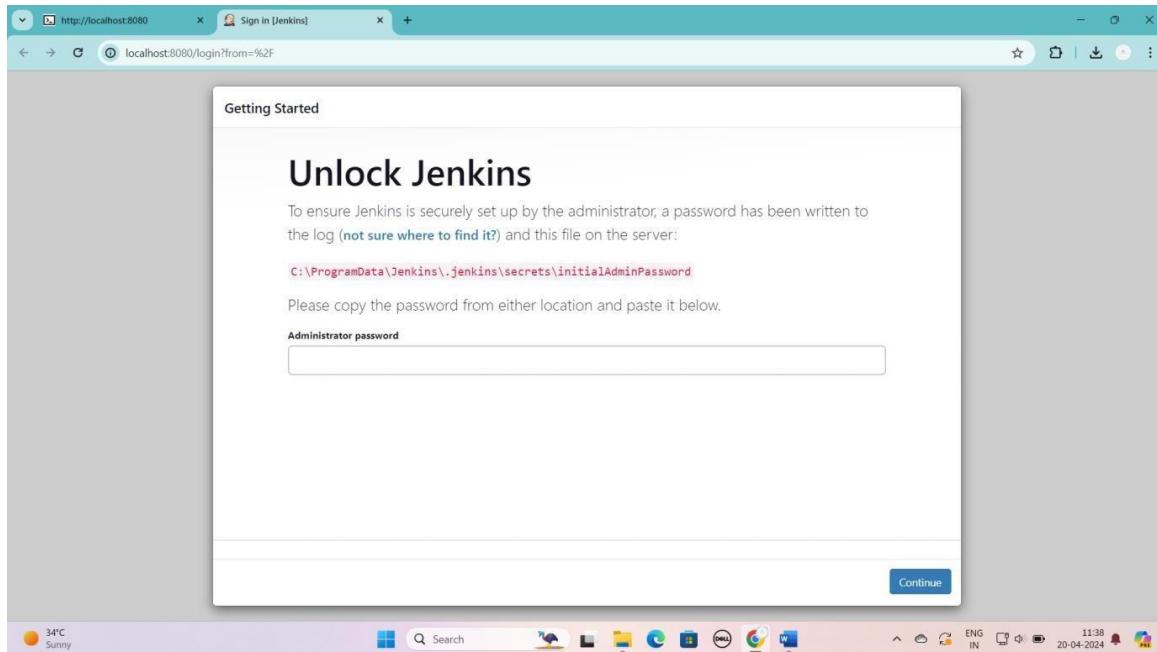
Step:10 Goto google and search localhost 8080



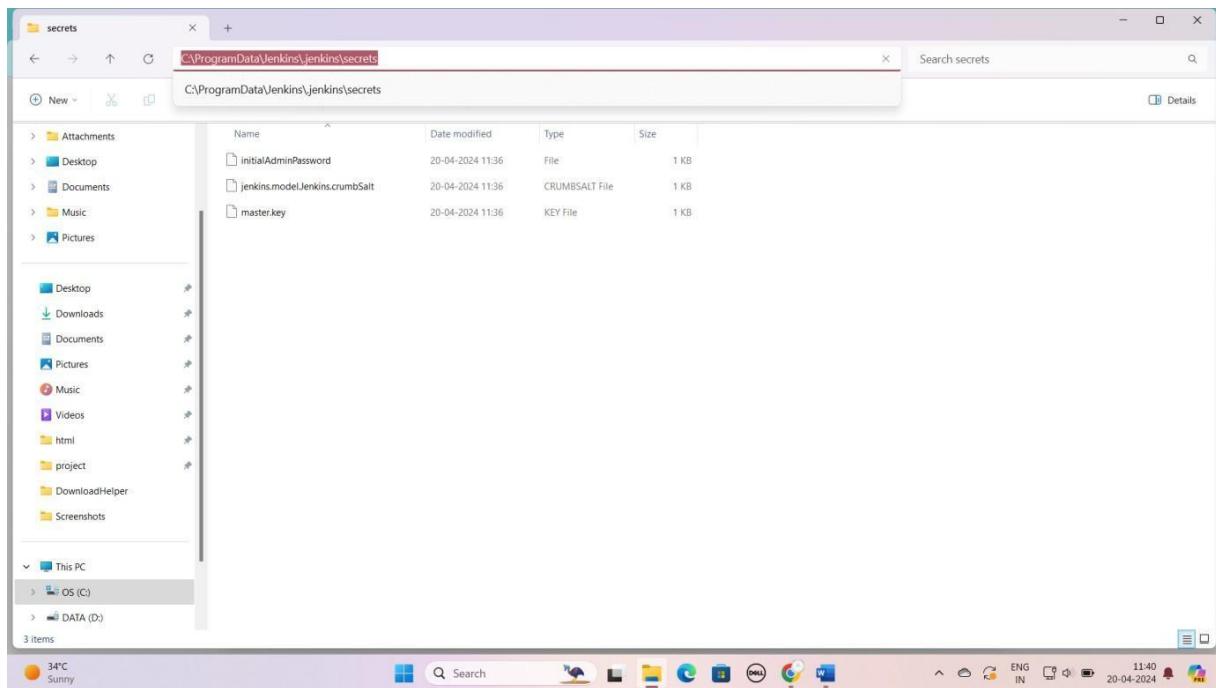
## Step:11 Opening localhost 8080



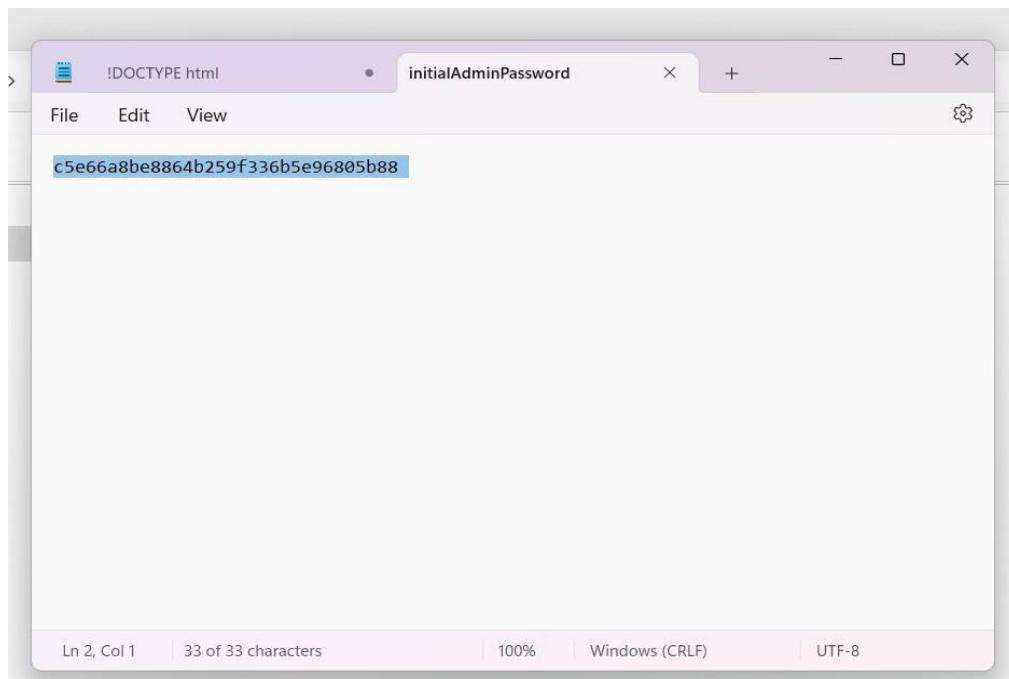
## Step:12 Unlock jenkins



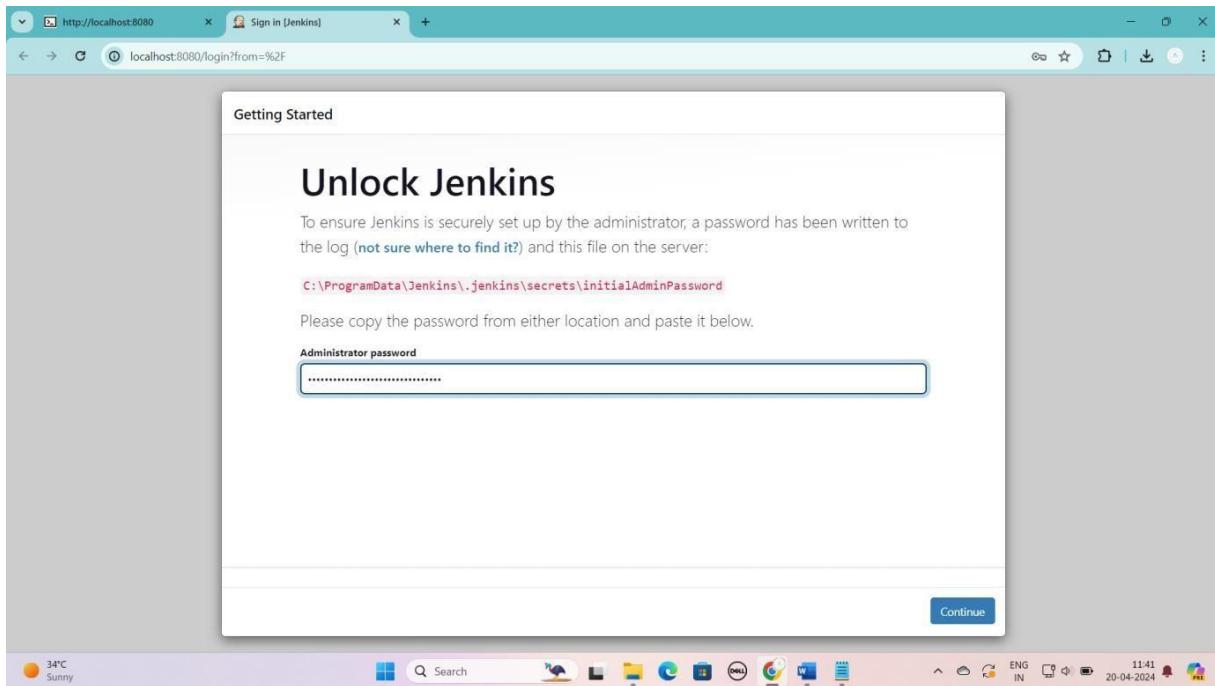
## Step:13 Goto jenkins folder and search for intial admin password



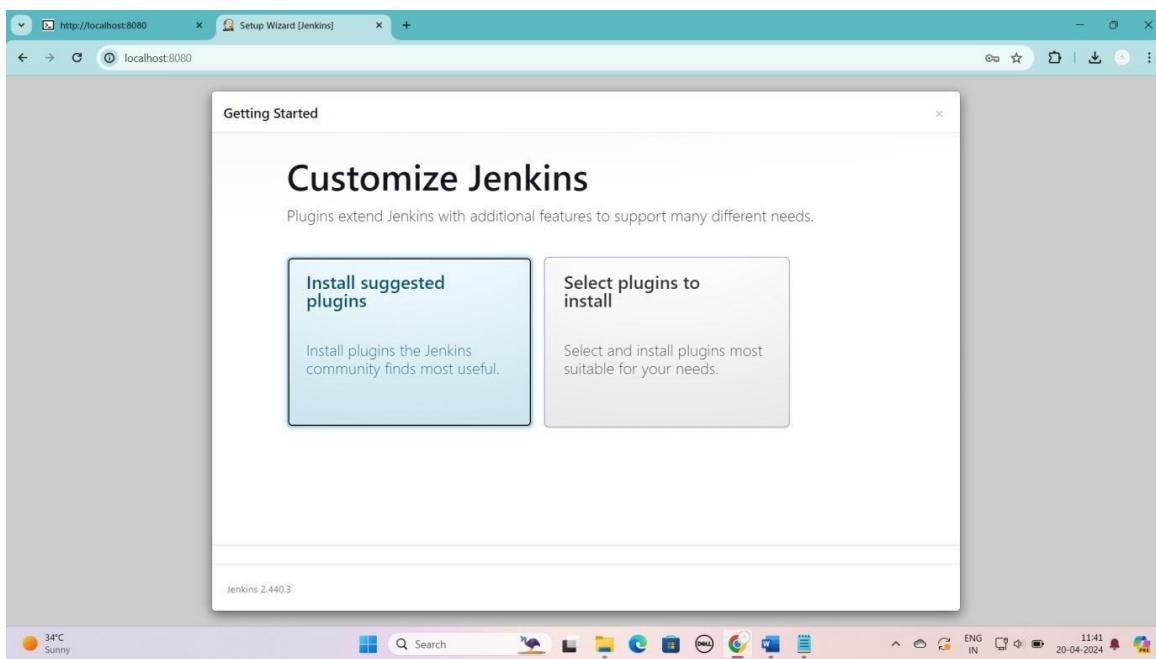
## Step:14 Copy the password



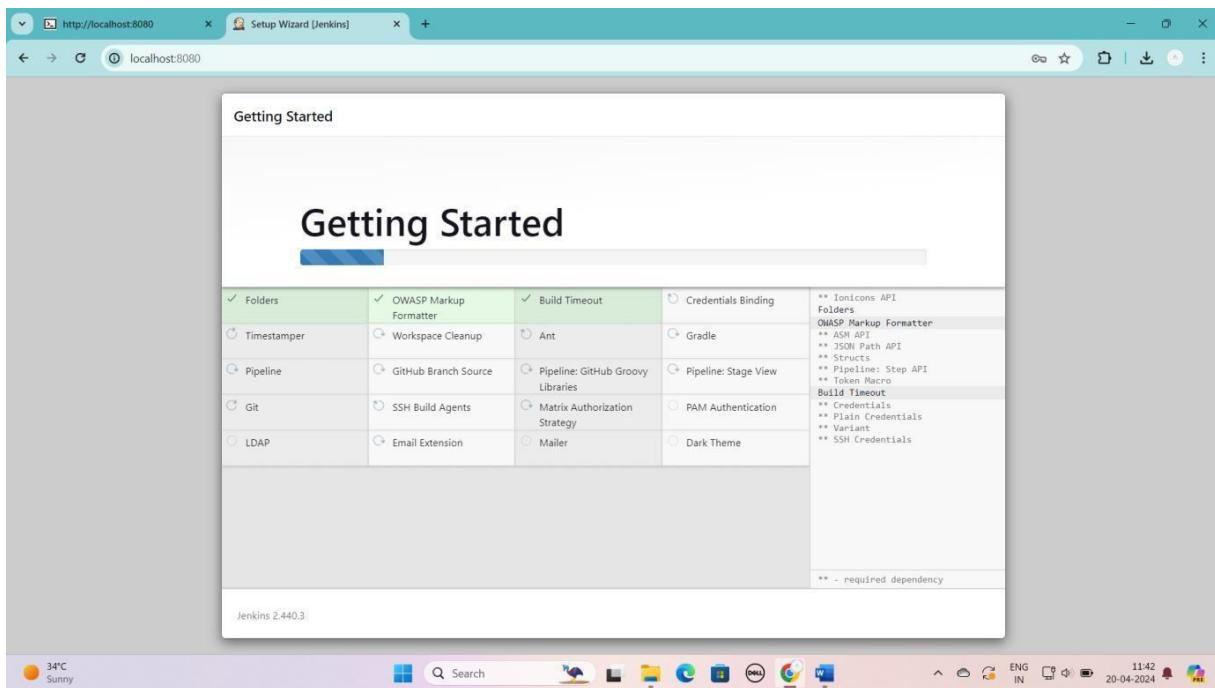
Step:15 To unlock jenkins paste the password and click continue



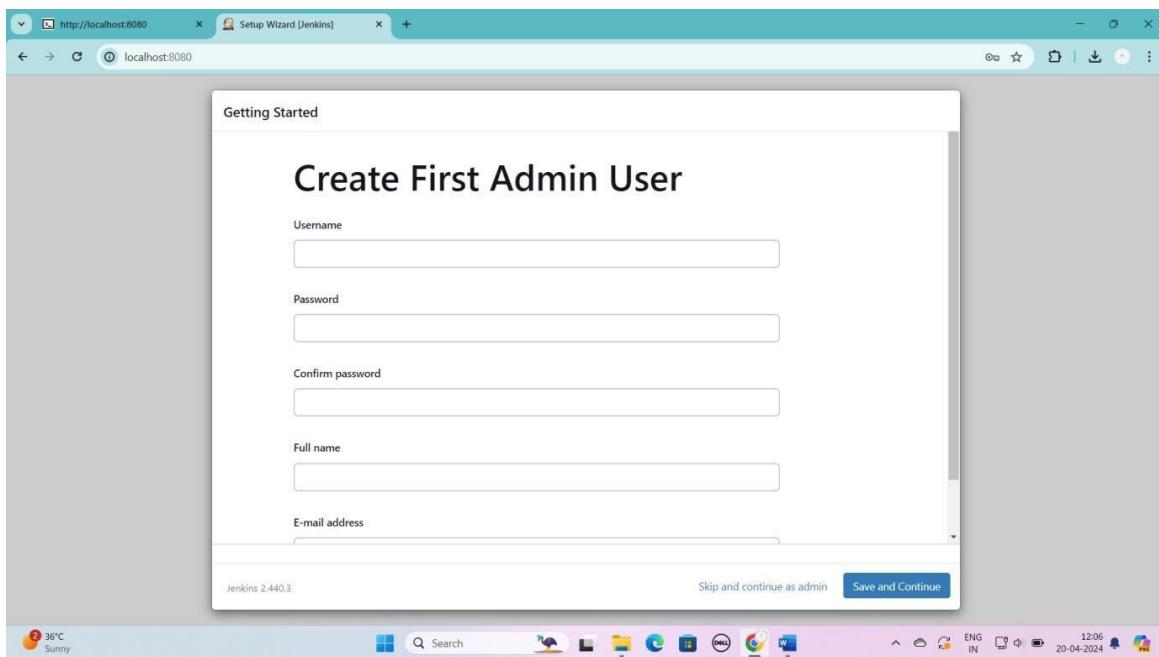
Step:16 Click install suggested plugins



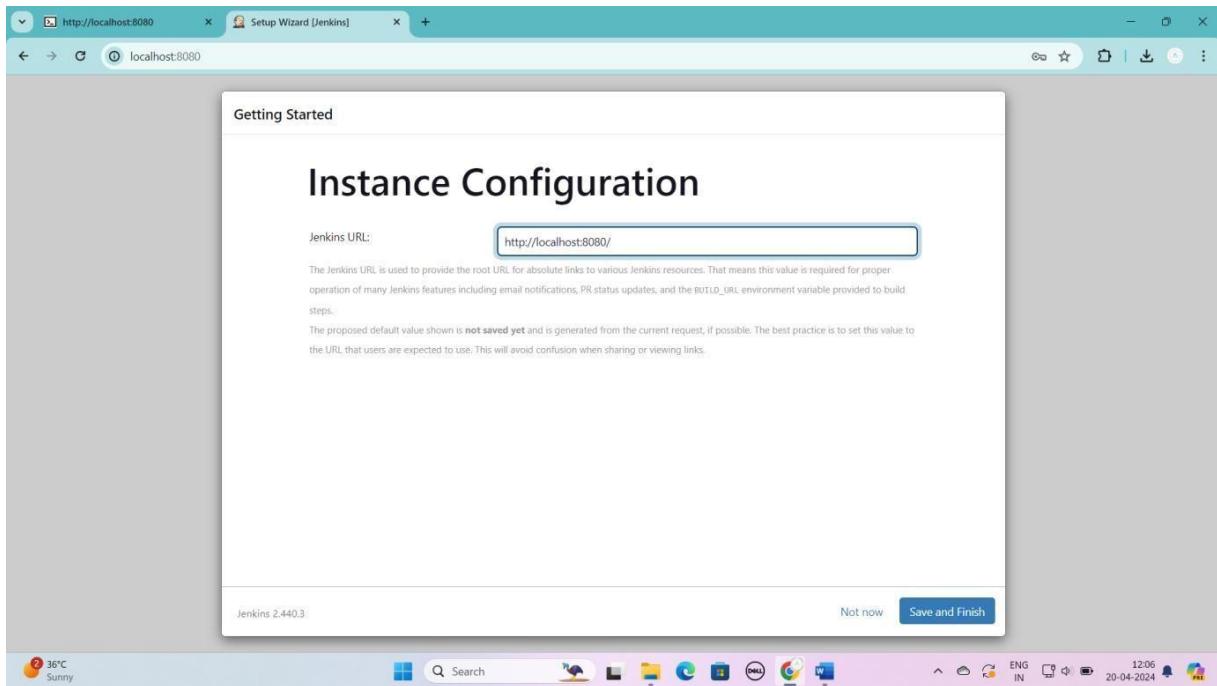
## Step:17 plugins installation



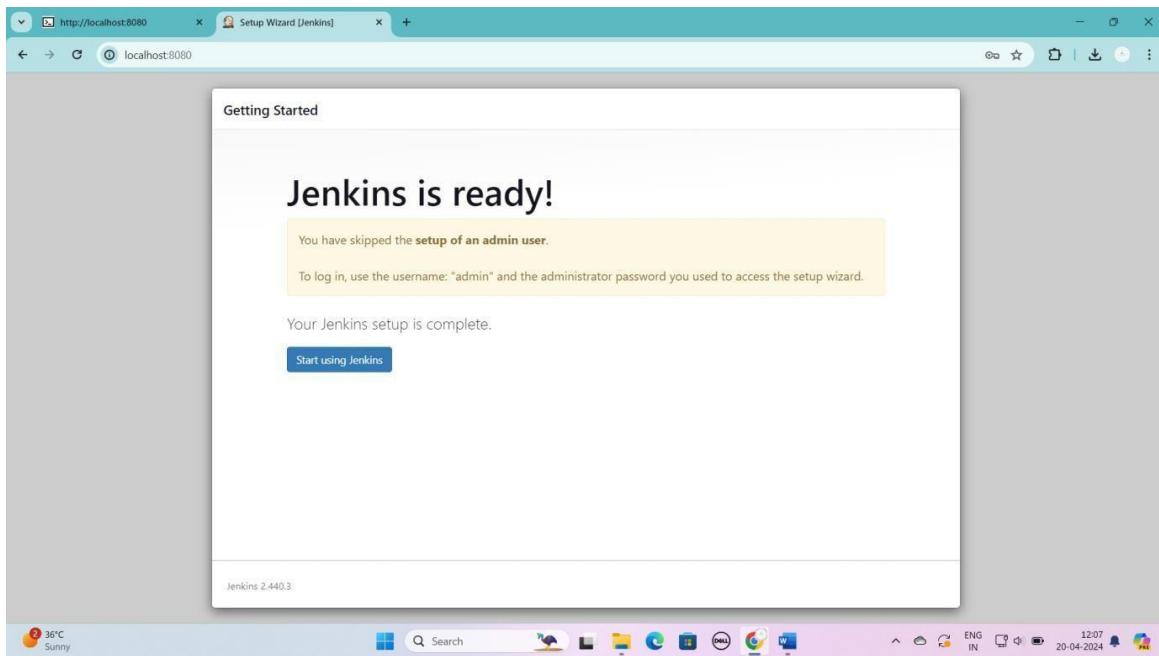
## Step:18 Create Admin User & click save and continue



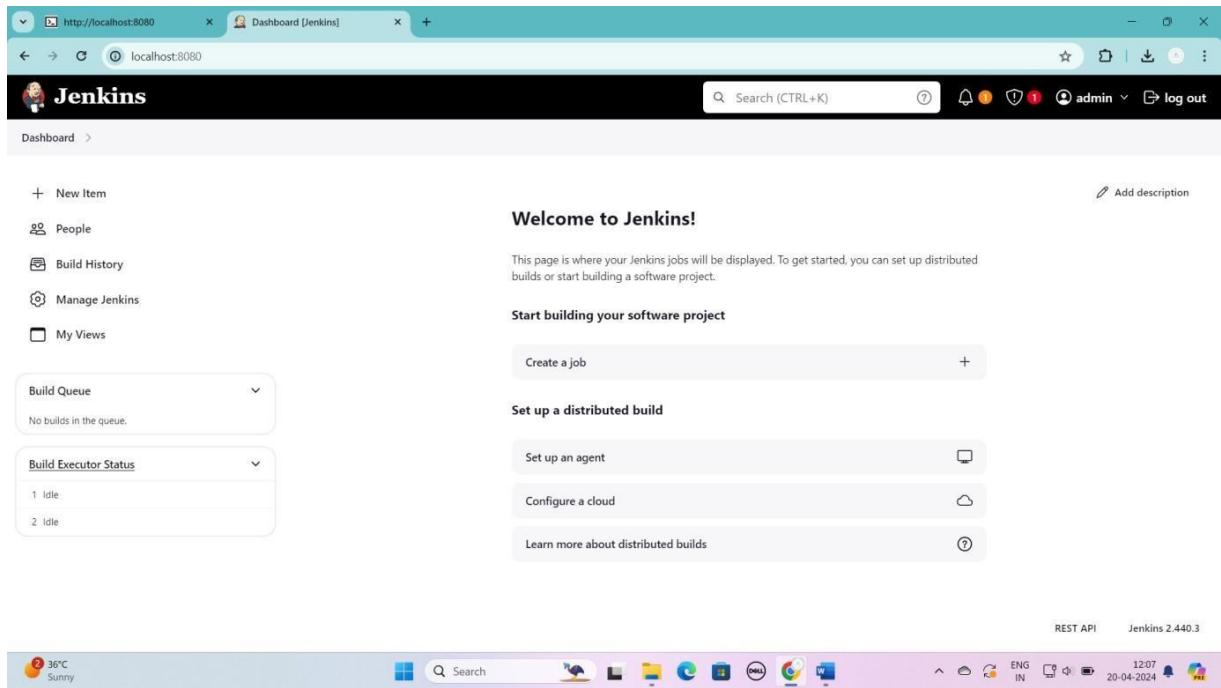
## Step:19 Configure jenkins & click save and continue



## Step:20 Jenkins setup is complete & click start using jenkins



## Step:21 Welcome to jenkins page



## Result:

Thus installing Jenkins in Cloud done successfully.

**Ex.No:4**

## Create CI pipeline using Jenkins

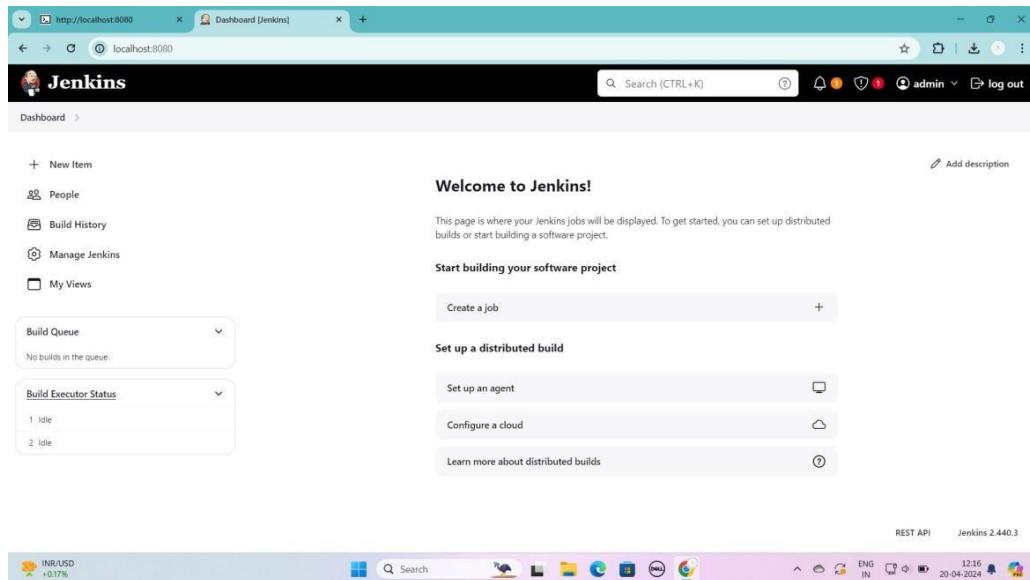
**Date:**

**Aim:**

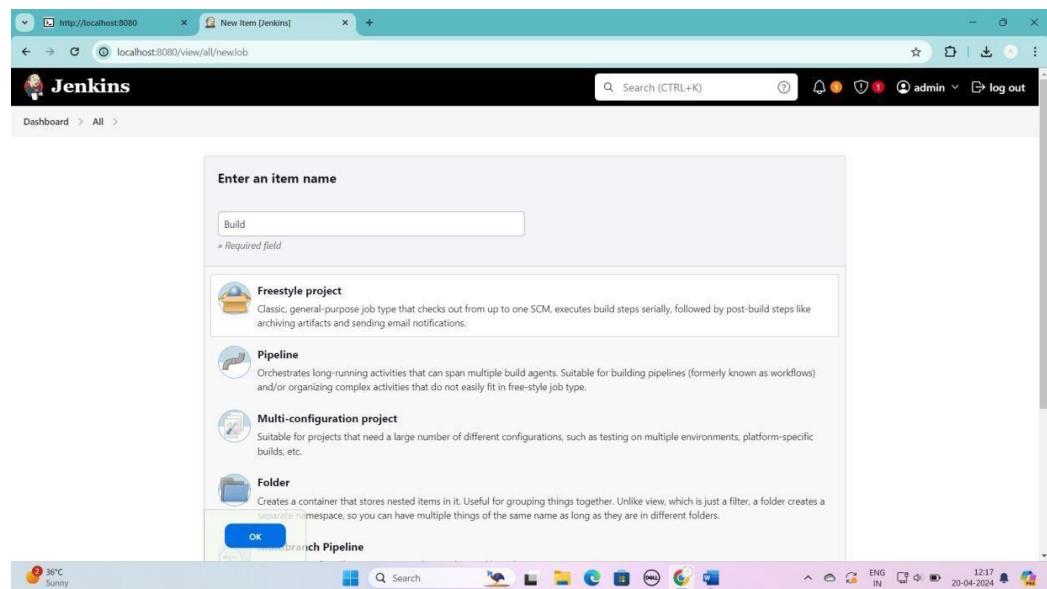
To create CI pipeline using Jenkins.

**Procedure:**

Step1: Go to jenkins page using localhost 8080



Step 2: Enter an item name as Build and Click Freestyle project & click ok.



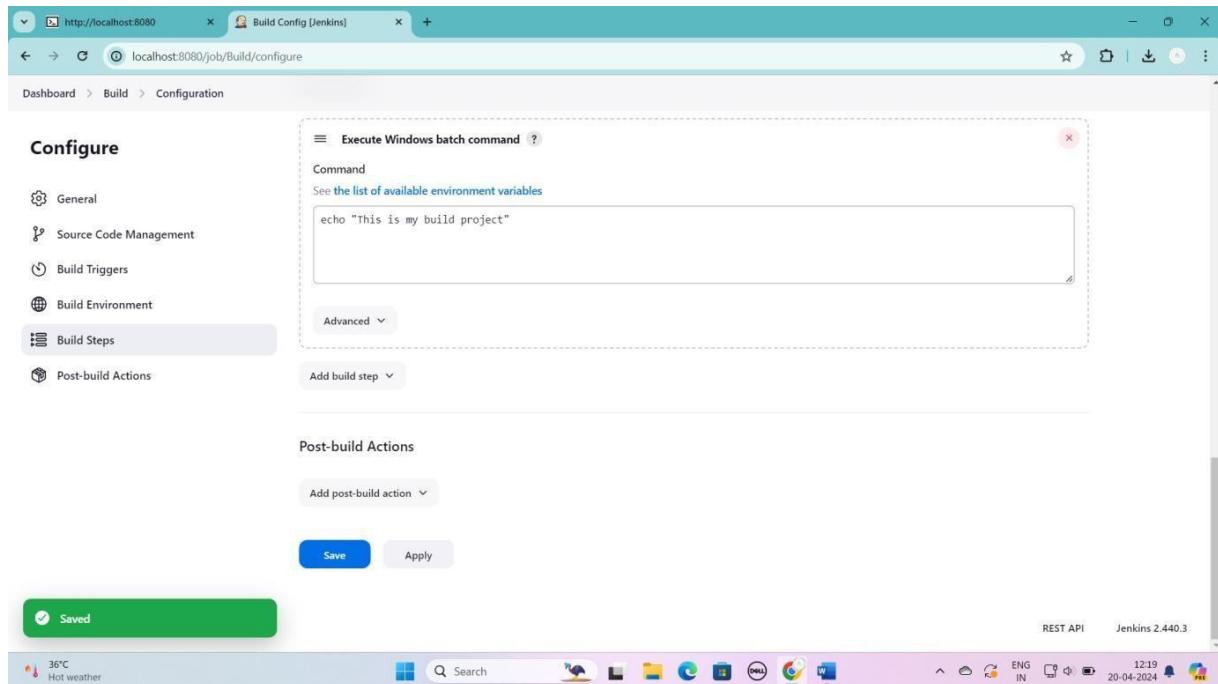
Step 3: From Configure click Build steps and scroll down the page.

The screenshot shows the Jenkins 'General' configuration page. On the left, a sidebar menu lists 'Configure' sections: General, Source Code Management, Build Triggers, Build Environment, Build Steps (which is currently selected), and Post-build Actions. The main content area is titled 'General' and contains a 'Description' field with a large empty text box. Below it are several checkboxes for build triggers: 'Discard old builds', 'GitHub project', 'This project is parameterized', 'Throttle builds', and 'Execute concurrent builds if necessary'. At the bottom of the page are 'Save' and 'Apply' buttons.

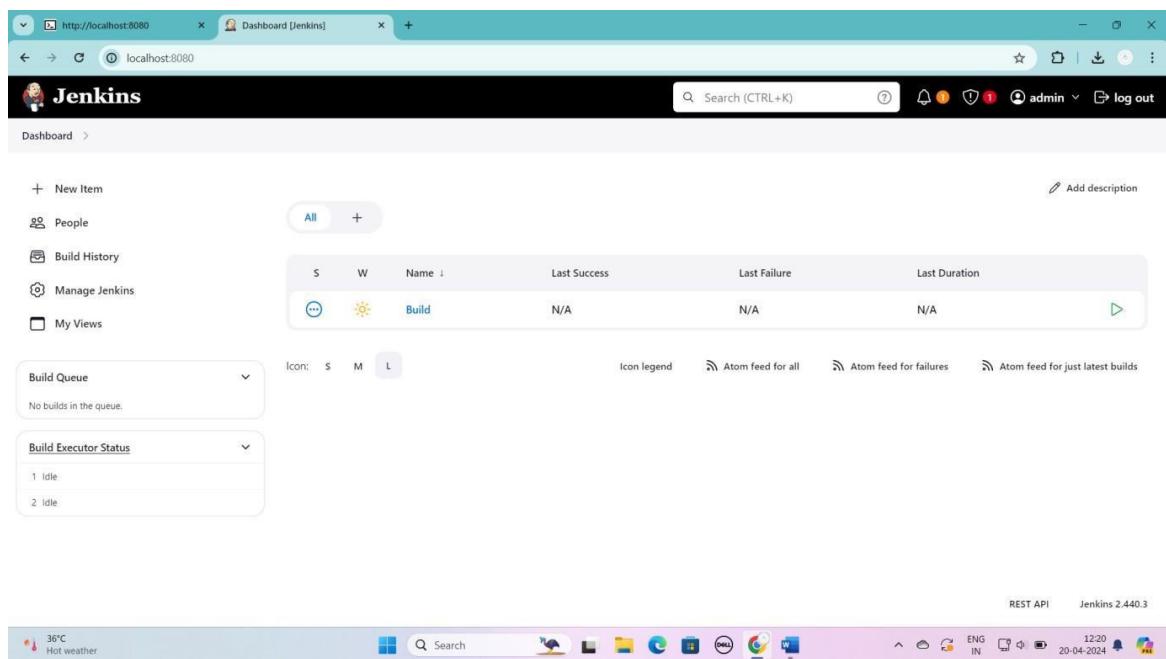
Step 4: Click Execute windows batch command as shown below

The screenshot shows the Jenkins 'Build Environment' configuration page. The 'Build Environment' section is active in the sidebar. In the main area, there are checkboxes for 'Delete workspace before build starts', 'Use secret text(s) or file(s)', 'Add timestamps to the Console Output', 'Terminate a build if it's stuck', and 'With Ant'. Below this, the 'Build Steps' section is visible. A dropdown menu titled 'Add build step' is open, showing options: 'Execute Windows batch command' (which is highlighted in blue), 'Execute shell', 'Invoke Ant', 'Invoke top-level Maven targets', 'Run with timeout', and 'Set build status to "pending" on GitHub commit'. The system tray at the bottom shows the date as 20-04-2024 and the time as 12:17.

Step 5: In command window give a message using echo and click apply and save.

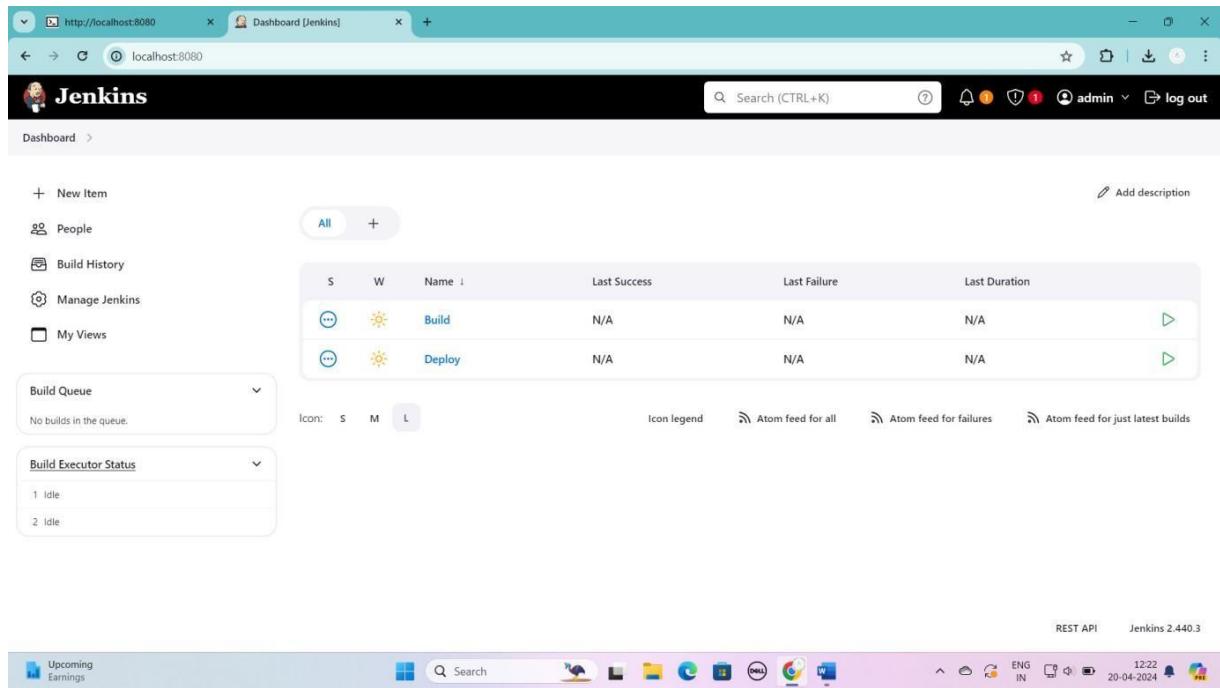


Step 6: The below page will be generated after apply and save.



Step 7: The same procedure will be followed for creating Deploy [refer step 2 to step 6].

After creating Deploy the below page will be generated and click on the new item.

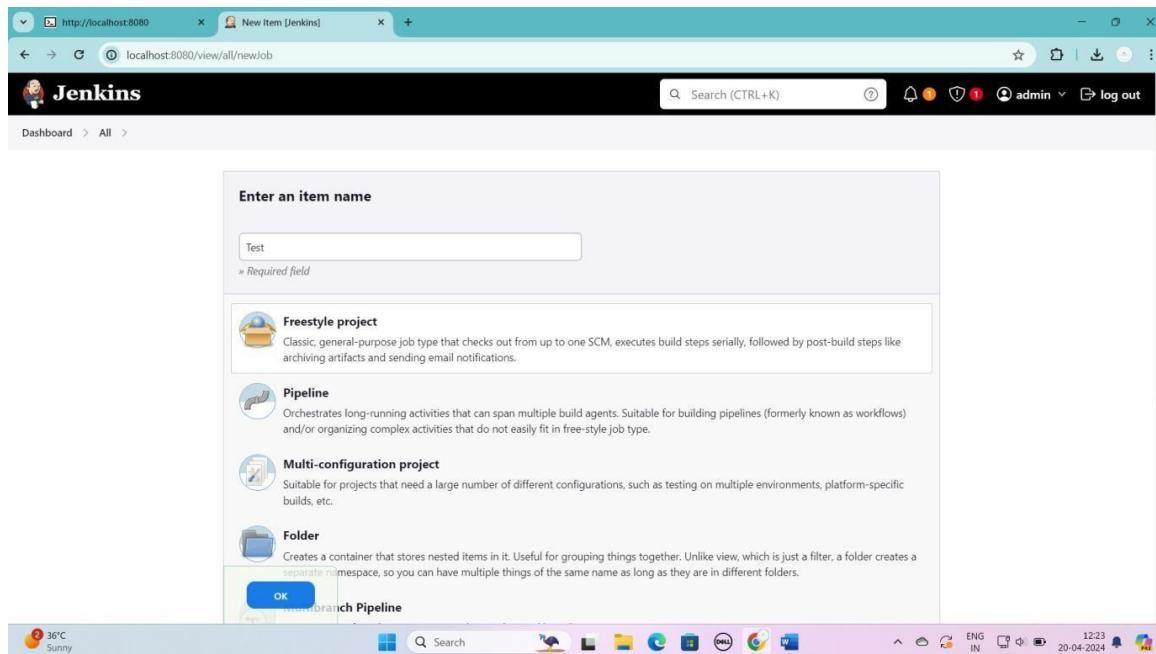


The screenshot shows the Jenkins dashboard at <http://localhost:8080>. The main content area displays a table of build items:

S	W	Name	Last Success	Last Failure	Last Duration
...	...	Build	N/A	N/A	N/A
...	...	Deploy	N/A	N/A	N/A

Below the table, there are sections for 'Build Queue' (empty), 'Build Executor Status' (2 idle), and 'Icon legend' with options S, M, L. At the bottom right, there are links for 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. The status bar at the bottom shows 'REST API Jenkins 2.440.3'.

Step 8: Enter an item name as Test &click on freestyle project and click ok.

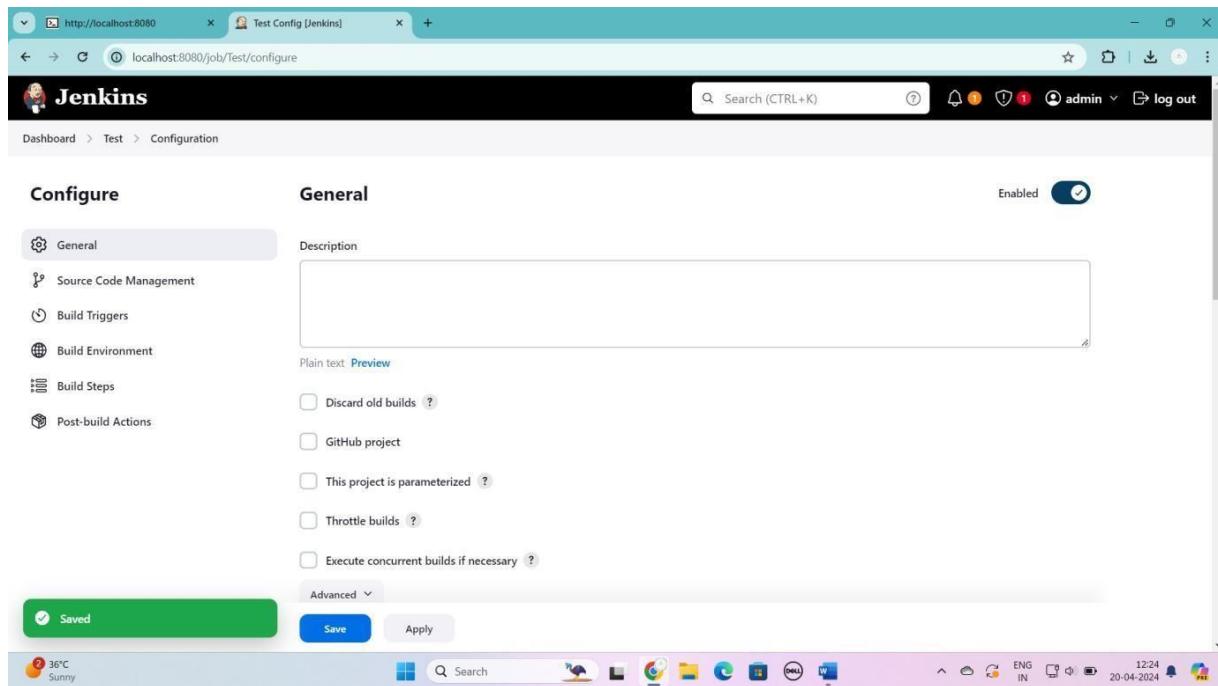


The screenshot shows the 'New Item' creation dialog at <http://localhost:8080/view/all/new/job>. The 'Enter an item name' field contains 'Test'. Below it, there are four project types listed:

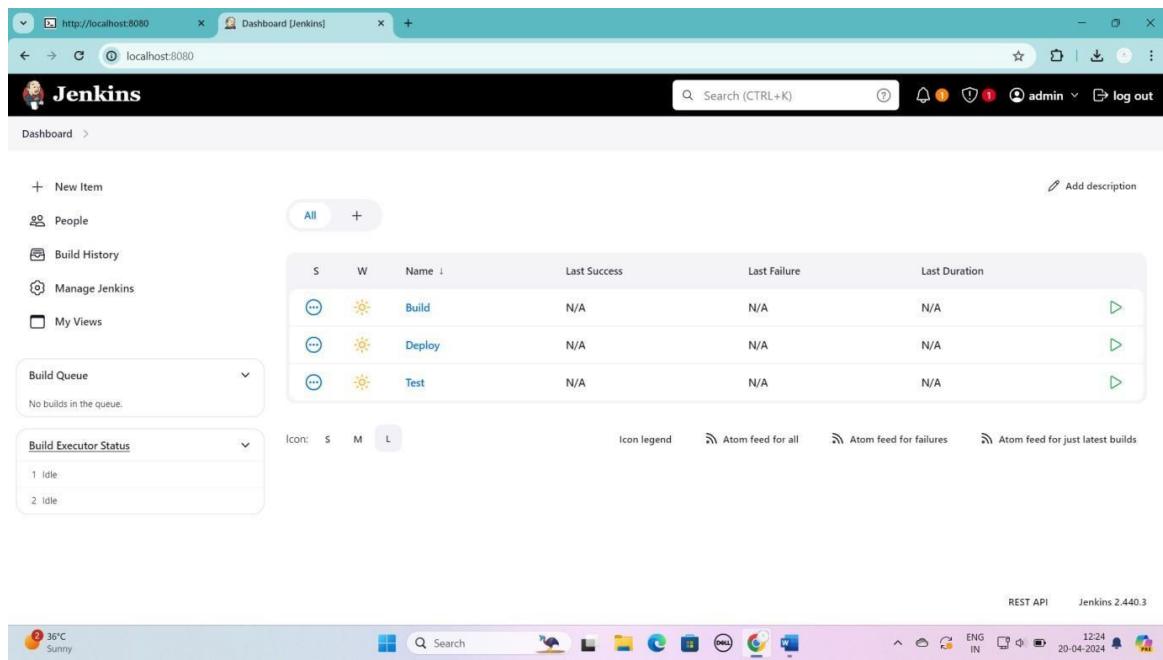
- Freestyle project**: Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

A blue 'OK' button is visible at the bottom left of the dialog. The status bar at the bottom shows '36°C Sunny'.

## Step 9: Click apply and save.

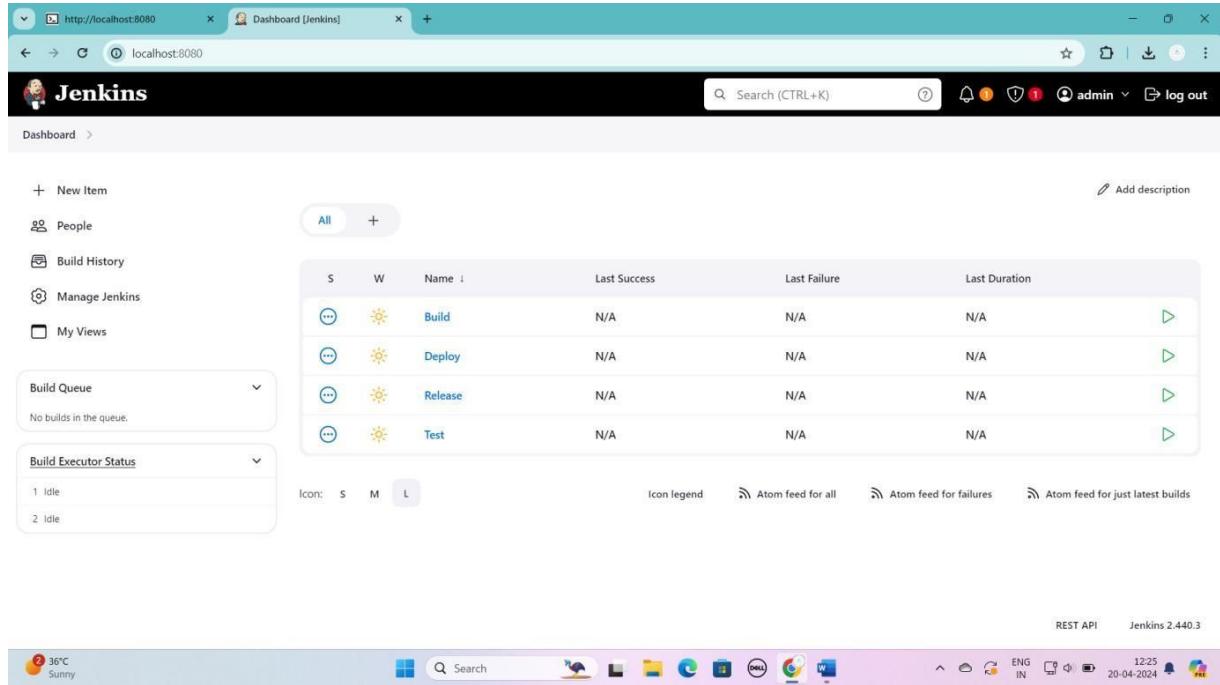


## Step 10: The below page will be generated after apply and save



Step 11: The same procedure will be followed for creating Release [refer test step 9, 10].

After creating Deploy the below page will be generated and click on the new item

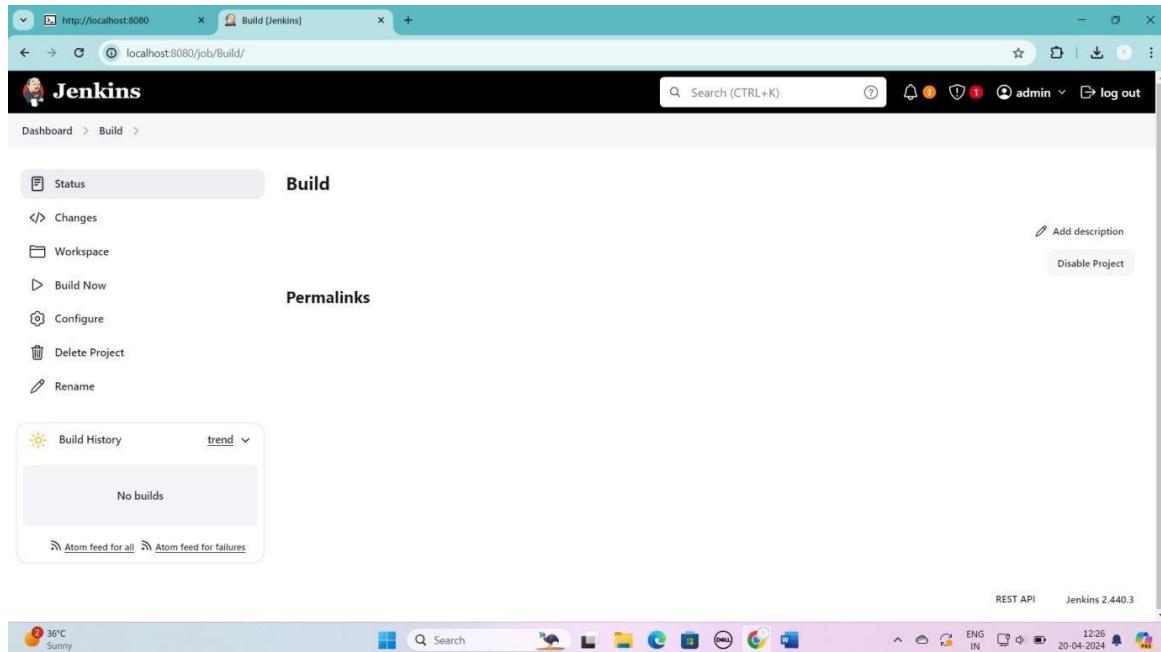


The screenshot shows the Jenkins Dashboard at <http://localhost:8080>. The left sidebar includes links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Under 'Build Queue', it says 'No builds in the queue.' Below that is a 'Build Executor Status' section with 1 idle and 2 idle executors. The main content area displays a table of build items:

S	W	Name	Last Success	Last Failure	Last Duration
...	...	Build	N/A	N/A	N/A
...	...	Deploy	N/A	N/A	N/A
...	...	Release	N/A	N/A	N/A
...	...	Test	N/A	N/A	N/A

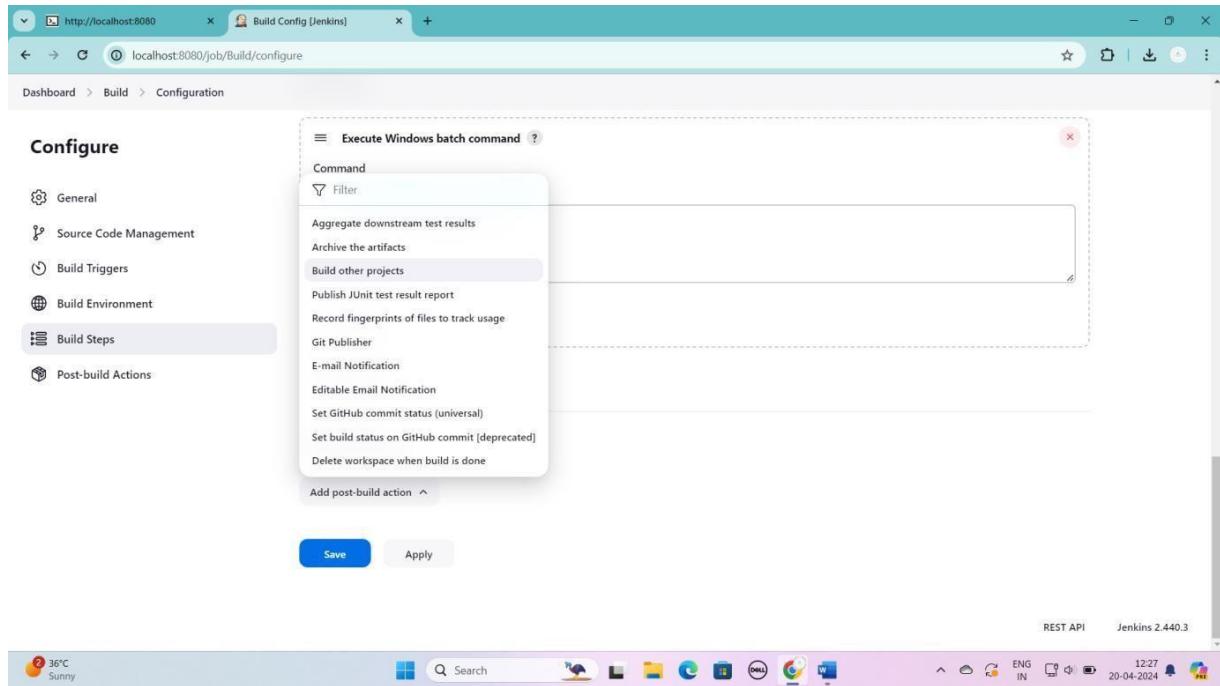
Icons for sorting by status (S), workspace (W), name (A), last success (L), and last failure (F) are shown above the table. At the bottom, there are links for 'Icon legend', 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. The status bar at the bottom right shows 'REST API Jenkins 2.440.3'.

Step 12: Click the Build item and Click on the Configure.

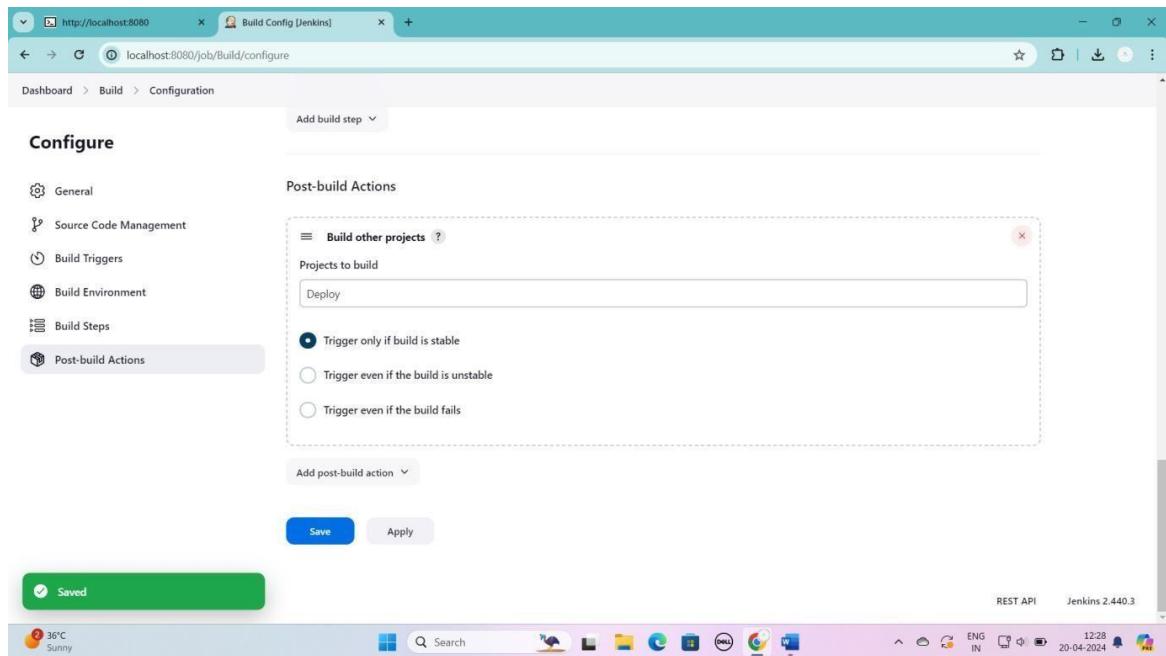


The screenshot shows the Jenkins Build configuration page at <http://localhost:8080/job/Build/>. The left sidebar has links for 'Status', 'Changes', 'Workspace', 'Build Now', 'Configure', 'Delete Project', and 'Rename'. The main content area is titled 'Build' and contains a 'Permalinks' section. A 'Build History' panel shows 'No builds' and links for 'Atom feed for all' and 'Atom feed for failures'. The status bar at the bottom right shows 'REST API Jenkins 2.440.3'.

Step 13: Click the Build Steps in Execute Window batch command and click the Build other Projects on drop down list.



Step 14: Click on the Post build actions in Build other project enter the Deploy as in next item name to connect here and click the Trigger only if build is stable, Click apply and save.



Step 15: In Build item -> Downstream project -> Click Deploy.

The screenshot shows the Jenkins 'Build' page for a project named 'Build'. On the left, there's a sidebar with options like Status, Changes, Workspace, Build Now, Configure, Delete Project, and Rename. The main area is titled 'Downstream Projects' and contains a 'Deploy' dropdown menu. Below it is a 'Permalinks' section. At the bottom, there's a 'Build History' panel showing 'No builds' and links for 'Atom feed for all' and 'Atom feed for failures'. The status bar at the bottom indicates 'localhost:8080/job/Deploy/' and shows system information including weather (36°C, Sunny), date (20-04-2024), and time (12:28).

Step 16: After clicking on the Deploy the below page will be displayed Upstream projects as Build.

The screenshot shows the Jenkins 'Deploy' page for a project named 'Deploy'. The sidebar and main area are identical to the 'Build' page in Step 15. The main area features a 'Upstream Projects' section with a 'Build' dropdown menu. Below it is a 'Permalinks' section. At the bottom, there's a 'Build History' panel showing 'No builds' and links for 'Atom feed for all' and 'Atom feed for failures'. The status bar at the bottom indicates 'localhost:8080/job/Deploy/' and shows system information including weather (36°C, Sunny), date (20-04-2024), and time (12:29).

Step 17: The same procedure will be followed for connecting the Test [refer step 12 to 16]. After creating Deploy the below page will be generated and click on the new item, connecting the below page is displayed. It shows Upstream project as Build and Downstream project as Test.

The screenshot shows the Jenkins Deploy page. On the left, there's a sidebar with links like Status, Changes, Workspace, Build Now, Configure, Delete Project, and Rename. The main area has tabs for Deploy, Status, and Changes. Under Deploy, there are sections for Upstream Projects (Build) and Downstream Projects (Test). Below these are sections for Build History (No builds), Permalinks, and feeds. At the bottom, there's a status bar with weather information (36°C, Sunny), system icons, and system status (ENG IN, 12:30, 20-04-2024).

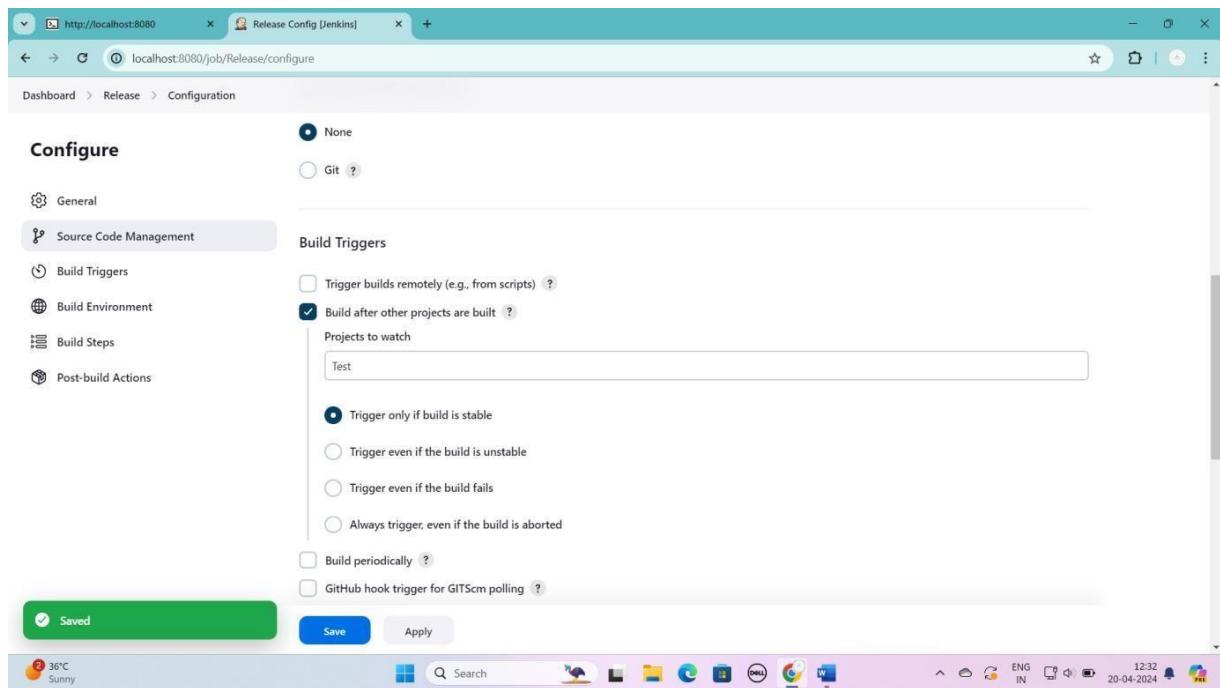
Step 18: After connecting all these items with each other the below page is display. Click the Release item.

The screenshot shows the Jenkins Dashboard. On the left, there are links for New Item, People, Build History, Manage Jenkins, and My Views. A Build Queue section shows 'No builds in the queue.' A Build Executor Status section shows 1 idle and 2 idle executors. The main area displays a table of items in the queue:

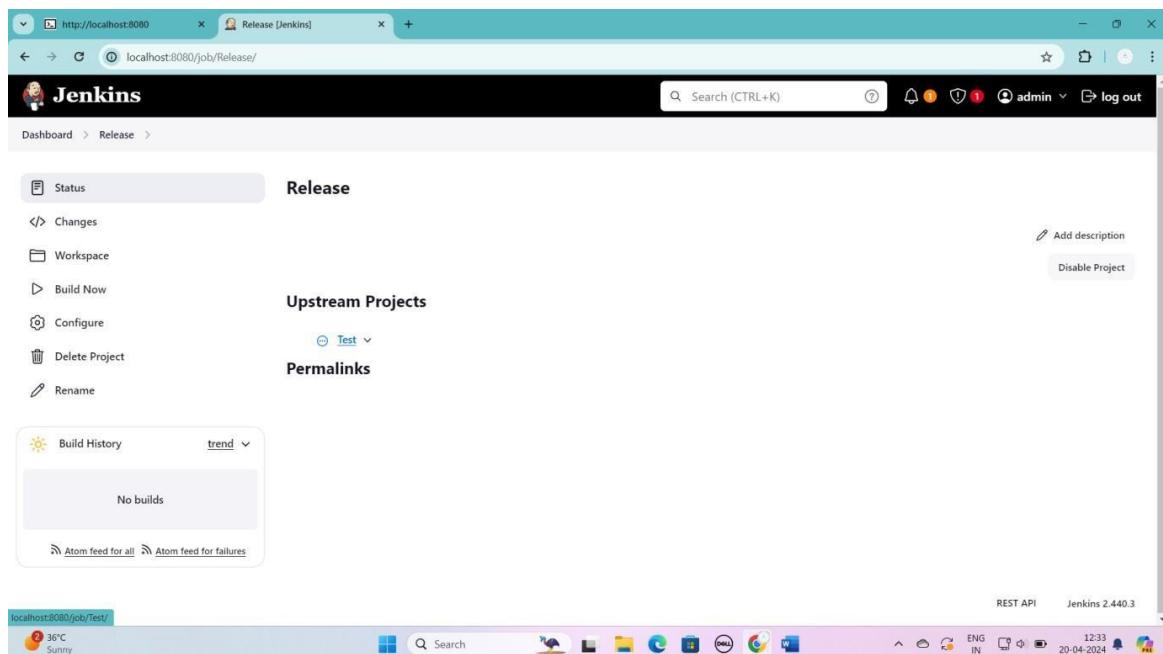
S	W	Name	Last Success	Last Failure	Last Duration
...	...	Build	N/A	N/A	N/A
...	...	Deploy	N/A	N/A	N/A
...	...	Release	N/A	N/A	N/A
...	...	Test	N/A	N/A	N/A

At the bottom, there's an icon legend, feeds for all, failures, and latest builds, and a status bar with weather information (36°C, Sunny), system icons, and system status (ENG IN, 12:31, 20-04-2024).

Step 19: Go to configure & click the Build Triggers ->Build after projects are built, Now in “project to match” give the Test and click on the “Trigger only if build is stable”, Click apply and save.



Step 20: After clicking the below page will be displayed the Test as upstream project. Click on the Test.



Step 21: After connecting all the items together below image will display. If we run the Build item the connecting Deploy item will display on the Build Queue.

The screenshot shows the Jenkins dashboard with the following details:

- Build Queue (1):** Deploy
- Build Executor Status:** 1 idle, 2 idle
- Build Queue Table:**

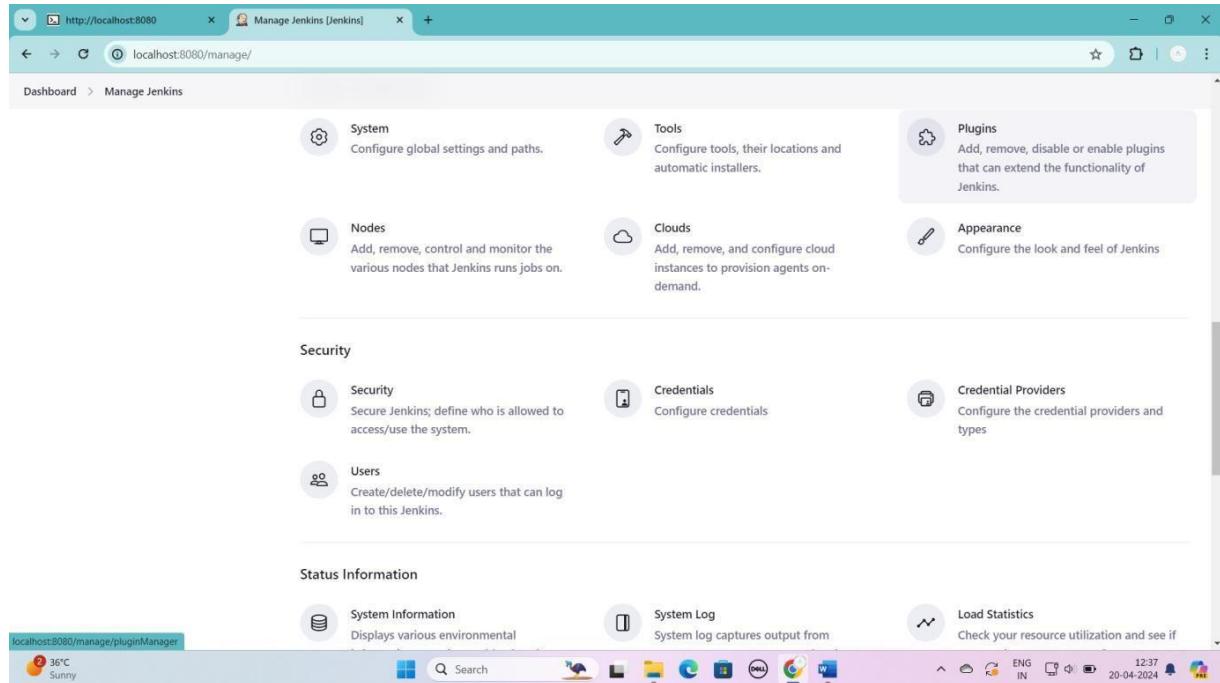
S	W	Name	Last Success	Last Failure	Last Duration
...	...	Build	N/A	N/A	N/A
...	...	Deploy	N/A	N/A	N/A
...	...	Release	N/A	N/A	N/A
...	...	Test	N/A	N/A	N/A
- Icon Legend:** S (Small), M (Medium), L (Large)
- Atom Feeds:** Atom feed for all, Atom feed for failures, Atom feed for just latest builds
- System Information:** REST API, Jenkins 2.440.3, Weather: 36°C Sunny, System tray icons (Cloud, ENG IN, 12:34, 20-04-2024, etc.)

The screenshot shows the Jenkins dashboard with the following details:

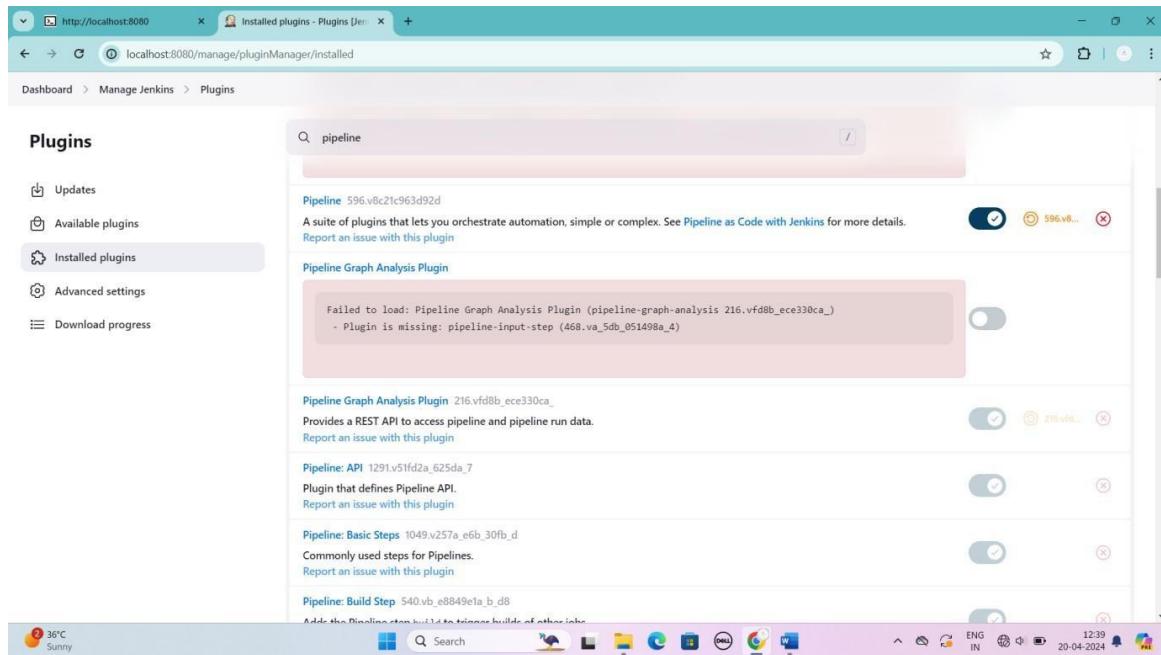
- Build Queue (1):** No builds in the queue.
- Build Executor Status:** 1 idle, 2 idle
- Build Queue Table:**

S	W	Name	Last Success	Last Failure	Last Duration
✓	...	Build	12 sec #2	N/A	0.21 sec
✓	...	Deploy	6.6 sec #2	N/A	0.19 sec
✓	...	Release	31 sec #1	N/A	0.13 sec
✓	...	Test	41 sec #1	N/A	0.14 sec
- Icon Legend:** S (Small), M (Medium), L (Large)
- Atom Feeds:** Atom feed for all, Atom feed for failures, Atom feed for just latest builds
- System Information:** REST API, Jenkins 2.440.3, Weather: 36°C Sunny, System tray icons (Cloud, ENG IN, 12:36, 20-04-2024, etc.)

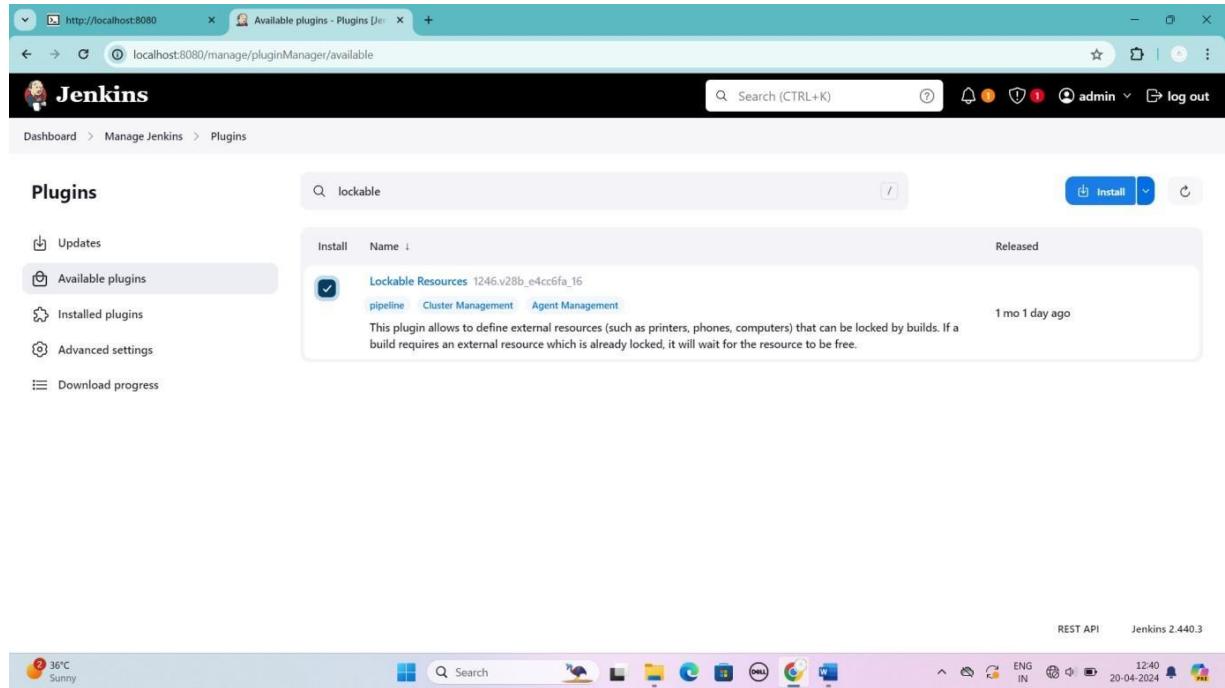
Step 22: Go to Mange Jenkins scroll down the page and click on plugins.



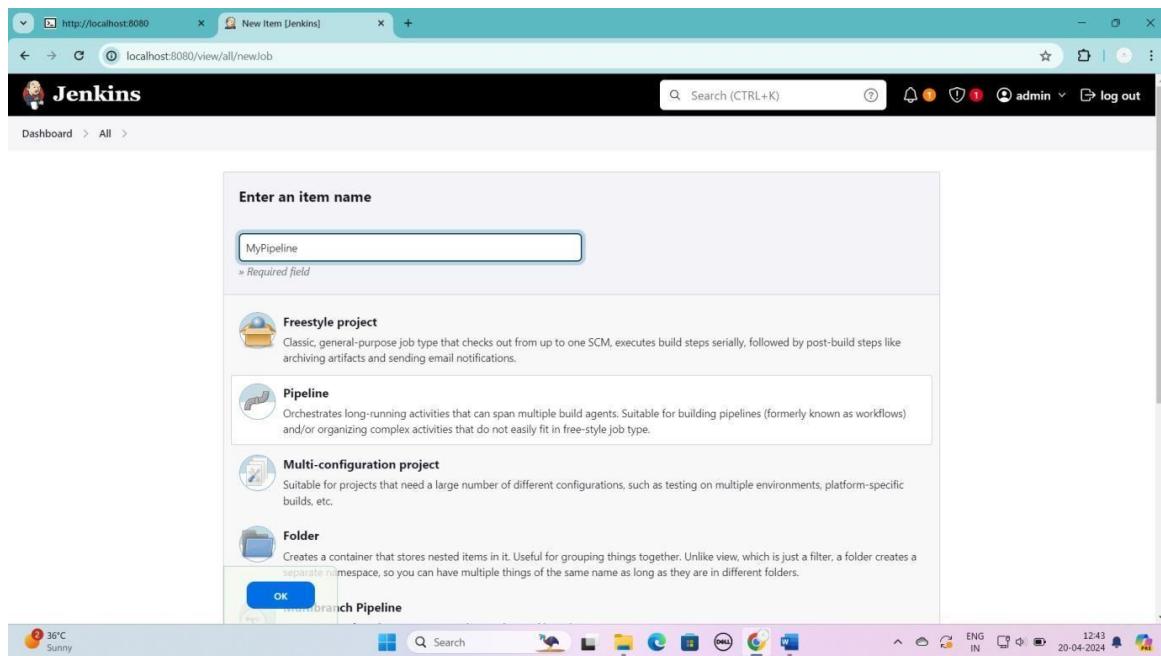
Step 23: Click on the Installed plugins and check the Pipeline and enable it.



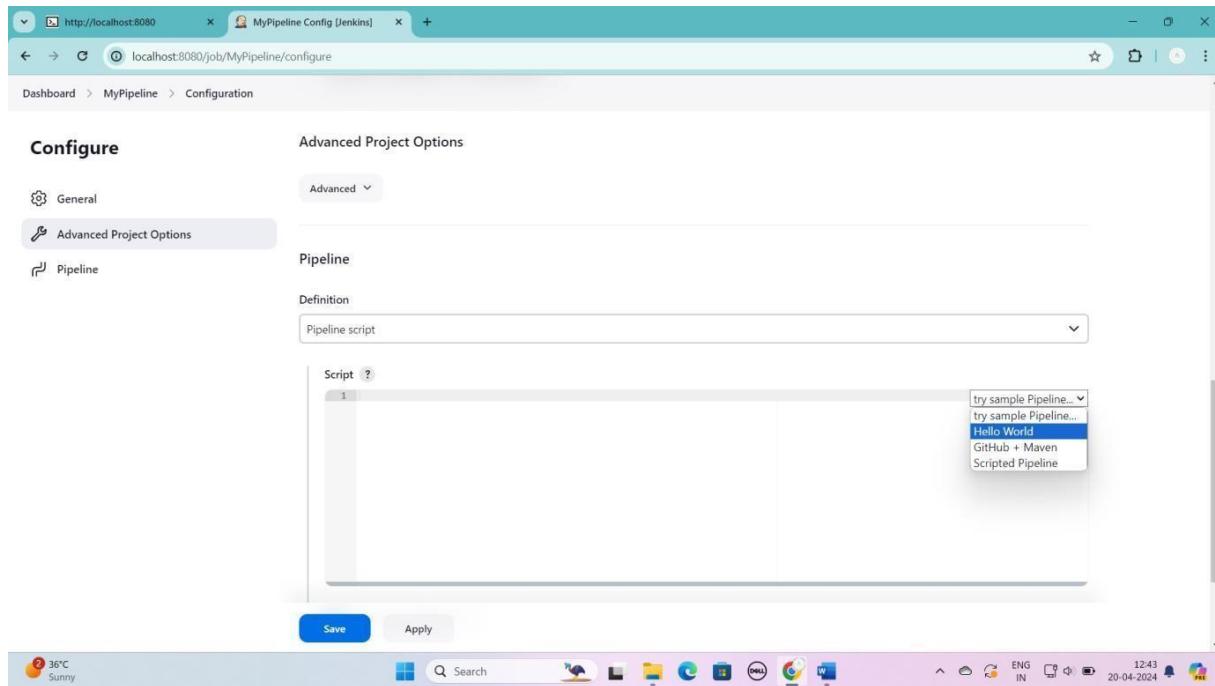
## Step 24: Go to Available plugins and install the Lockable Resources.



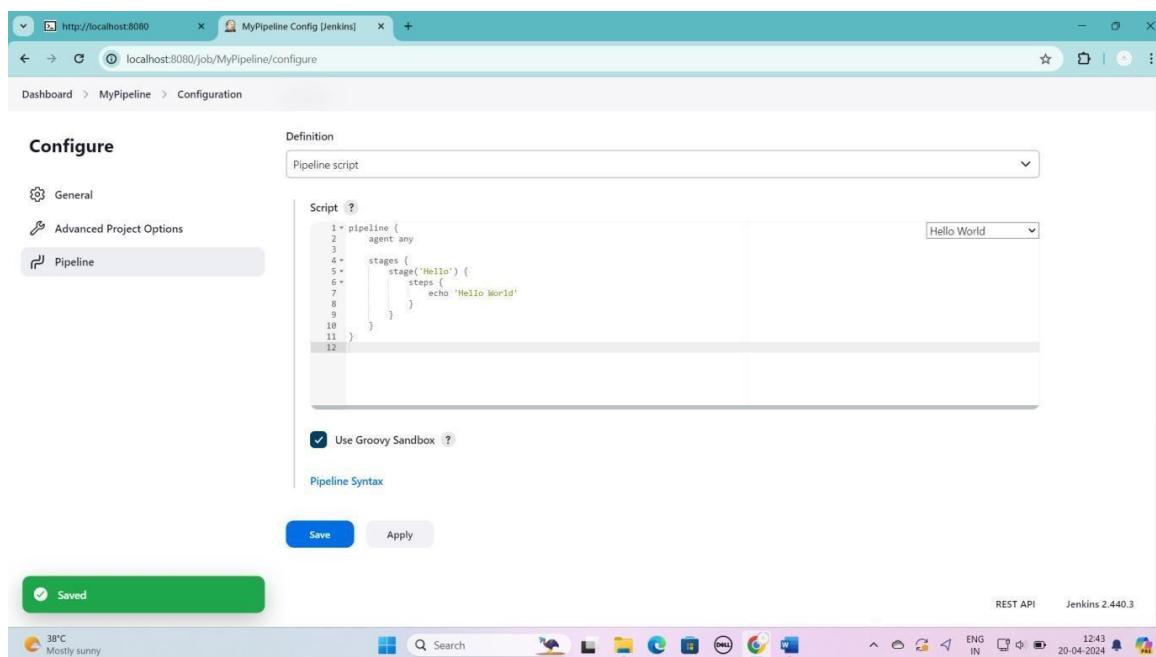
Step 25: After installing, the Pipeline Click on New item and Enter an item name as MyPipeline & click on the Pipeline and click ok to finish.



Step 26 : After Creating the below page will displayed and click on the Pipeline in side bar and Choose the Pipeline Script as Definition and Choose the Script default select the Hello world Program in the drop down menu.



Step 27: After Clicking it the below page is display, Click on the Use Groovy Sandbox and Click Apply and Save.



Step 28: After creating the Pipeline the below page is display and click on the run button to run the program and open it.

The screenshot shows the Jenkins dashboard at <http://localhost:8080>. The main content area displays a table of build jobs:

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	Build	9 min 57 sec #2	N/A	0.21 sec
✓	☀️	Deploy	9 min 51 sec #2	N/A	0.19 sec
...	☀️	MyPipeline	N/A	N/A	N/A
✓	☀️	Release	9 min 31 sec #2	N/A	0.16 sec
✓	☀️	Test	9 min 41 sec #2	N/A	0.17 sec

On the left sidebar, there are links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Below these are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle). At the bottom of the sidebar are icons for 'Icon: S M L', 'Icon legend', and three 'Atom feed' links. The status bar at the bottom shows 'localhost:8080/view/all/new/job' and 'REST API Jenkins 2.440.3'.

Step 29: After opening the page the below page will display and Click on the Hello.

The screenshot shows the Jenkins job details for 'MyPipeline' at <http://localhost:8080/job/MyPipeline/>. The left sidebar contains options like 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. It also includes a 'Build History' section with a 'trend' dropdown, a 'Filter...' button, and links for 'Atom feed for all' and 'Atom feed for failures'. The main content area is titled 'Stage View' and shows a single stage named 'Hello' with a duration of 192ms. A tooltip indicates 'Average stage times: (Average full run time: ~4s)'. Below the stage view is a 'Permalinks' section with a list of recent builds:

- Last build (#1), 15 sec ago
- Last stable build (#1), 15 sec ago
- Last successful build (#1), 15 sec ago
- Last completed build (#1), 15 sec ago

The status bar at the bottom shows 'localhost:8080/job/MyPipeline/' and 'INR/USD +0.17%'.

Step 30: After Clicking Hello the below page will displayed and click on the Logs.

The screenshot shows the Jenkins interface for the 'MyPipeline' project. On the left, there's a sidebar with options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, and Pipeline Syntax. Below that is a 'Build History' section with a 'trend' dropdown and a search bar. A 'Permalinks' section lists recent builds. The main area is titled 'Stage View' and shows a single stage named 'Hello' which has just succeeded. It displays an average stage time of 192ms. Below the stage view is a 'Permalinks' section with a list of recent builds. At the bottom right, it says 'REST API Jenkins 2.440.3'. The taskbar at the bottom shows various application icons and the date/time as 20-Apr-2024, 12:45 pm.

Step 31: By clicking on the Logs the output will display on the top of the screen like below image.

This screenshot shows the same Jenkins interface as above, but with a modal window titled 'Stage Logs (Hello)' overlaid. The modal contains the log message 'Hello World' and a 'Print Message -- Hello World (self time 16ms)' button. The background page remains mostly visible, showing the Stage View and Permalinks sections. The taskbar at the bottom is identical to the previous screenshot.

## Result:

Thus, Creating CI pipelines using Jenkins done successfully.

**Ex.No:4**

## Build a simple application using Gradle

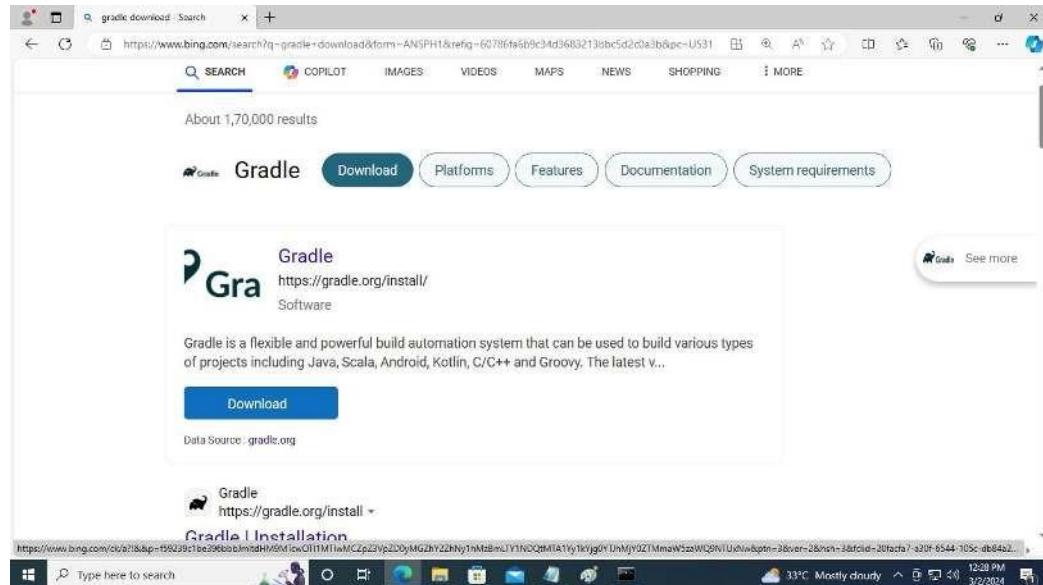
**Date:**

**Aim:**

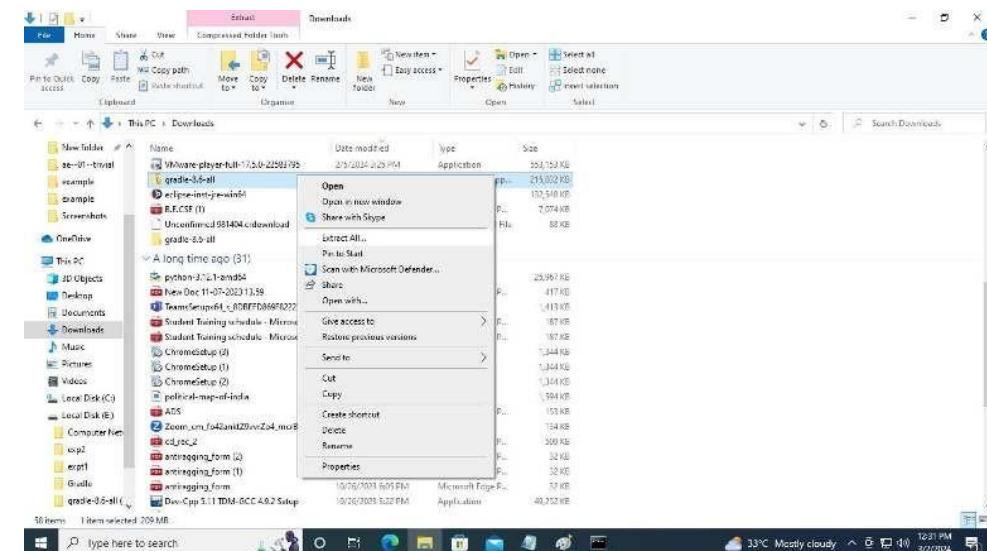
Build a simple application using Gradle

**Procedure:**

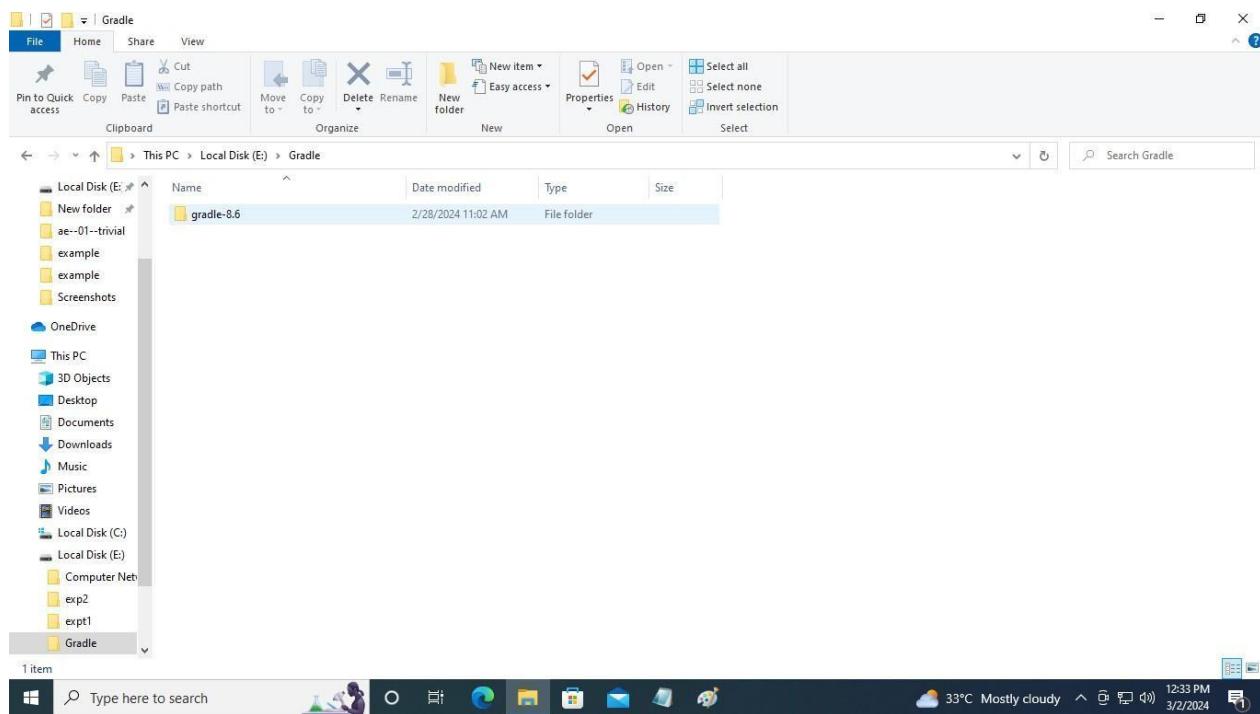
Step 1: Download gradle from official website



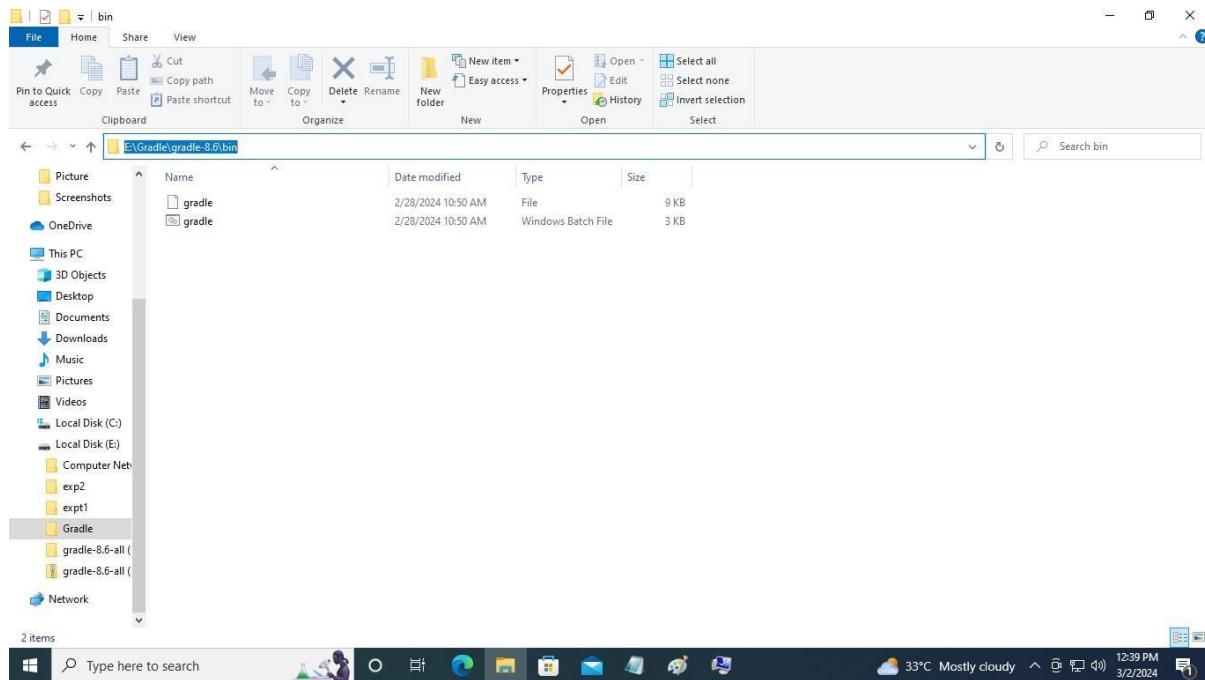
Step 2: Go to downloads and extract the gradle folder



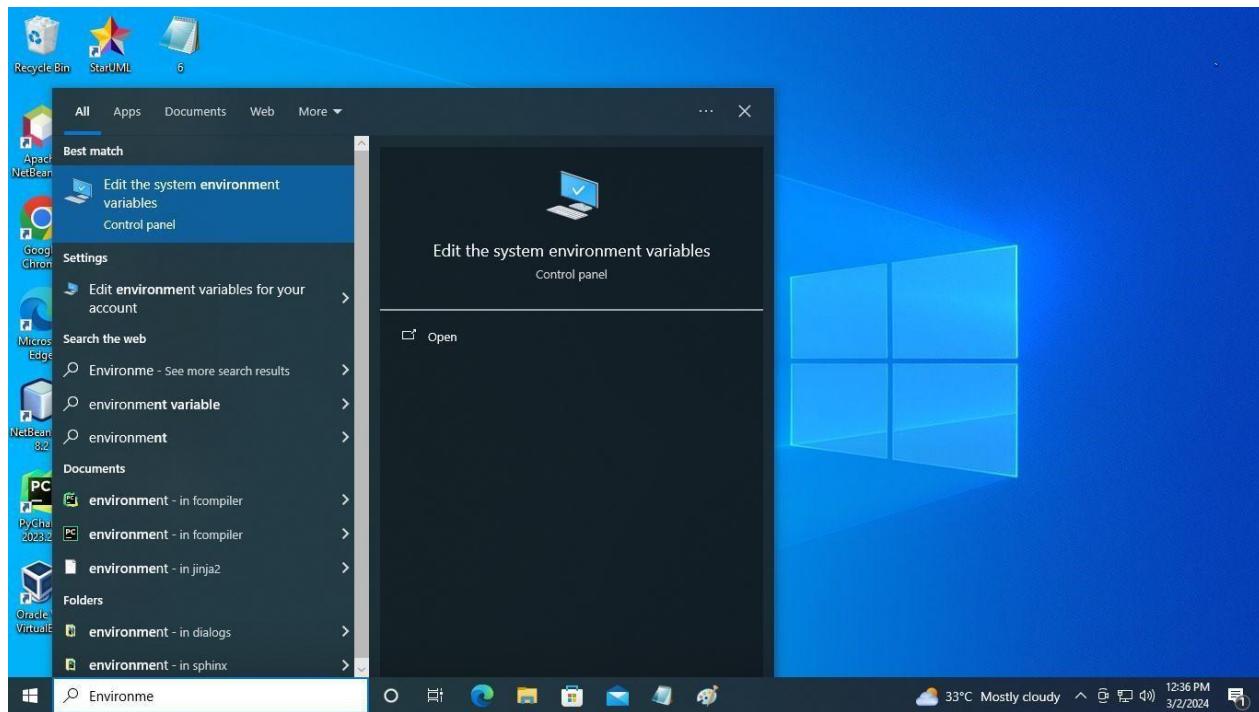
### Step 3: Extracted gradle folder



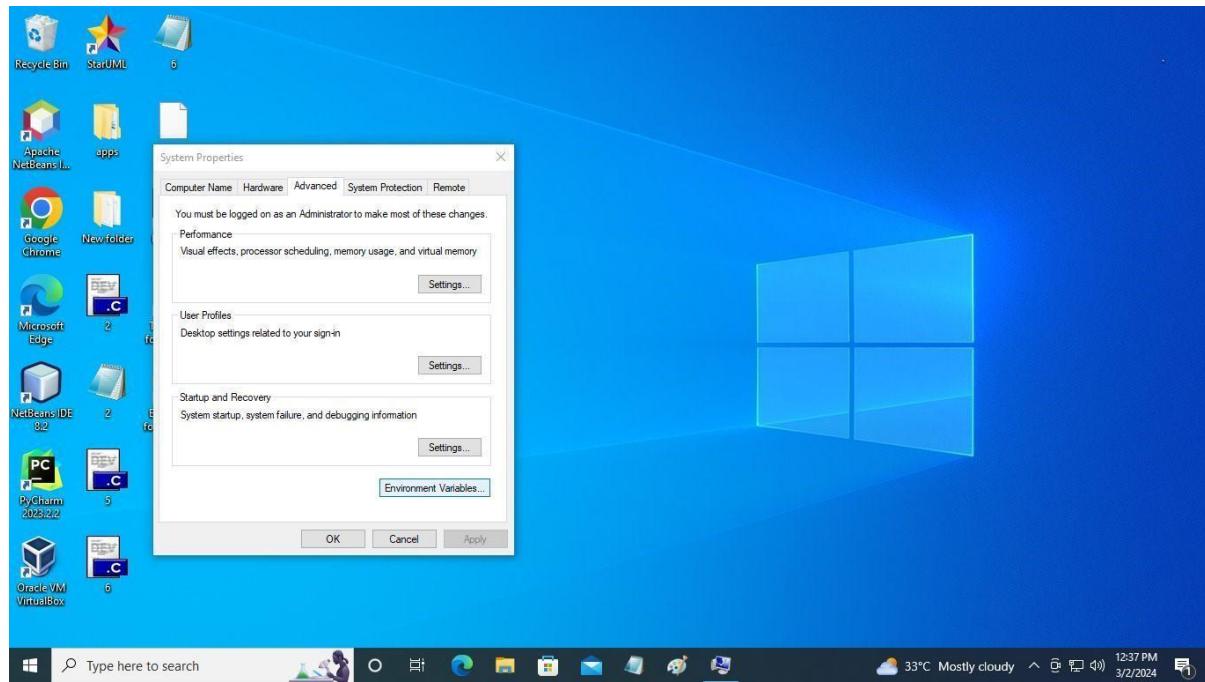
### Step 4: Copy the bin path from gradle folder



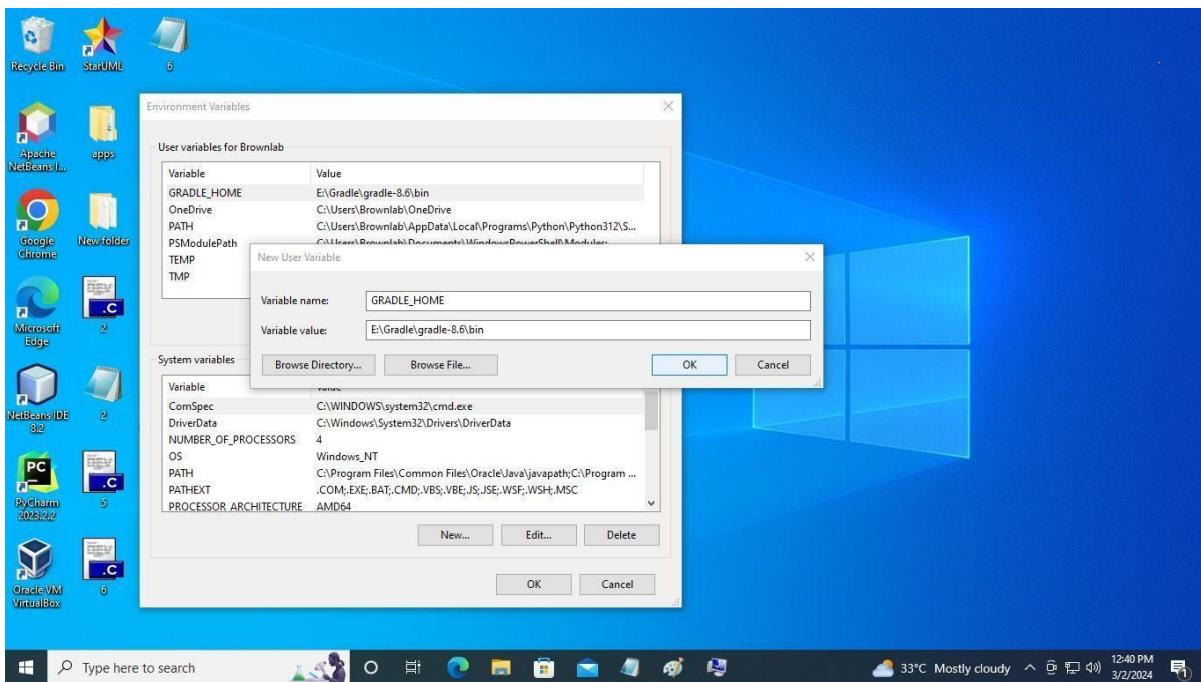
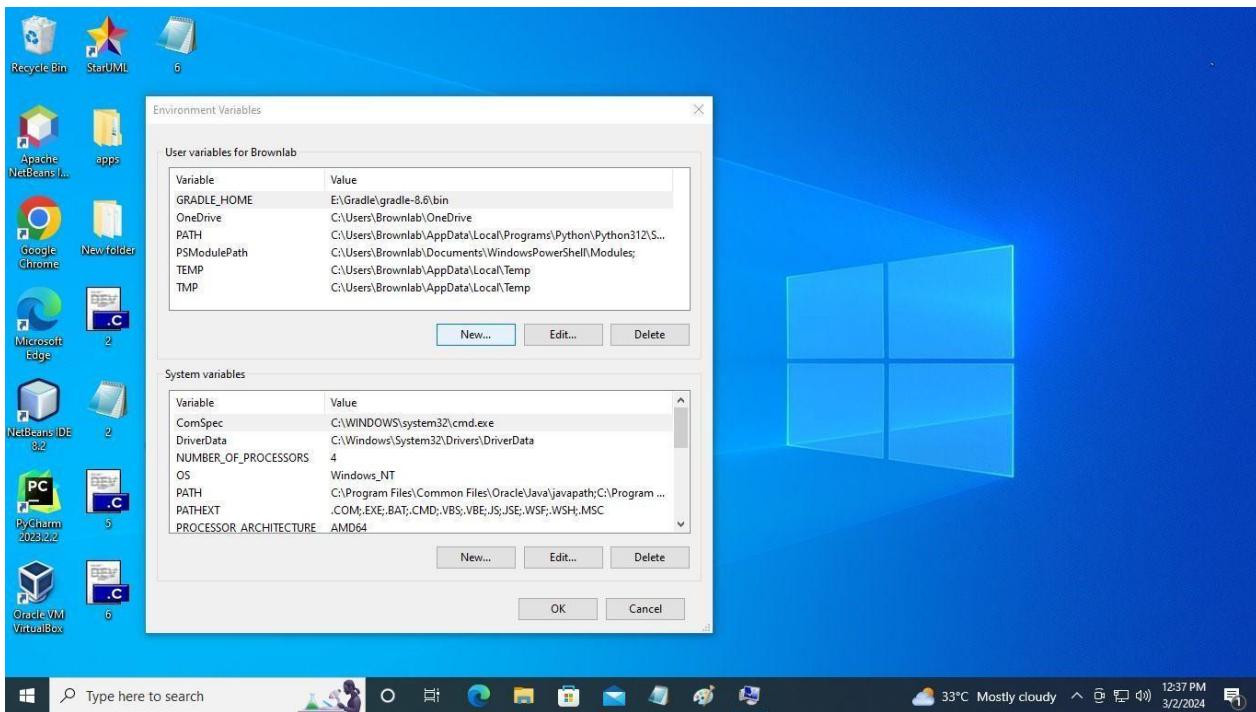
## Step 5: Search environment variable to edit the path



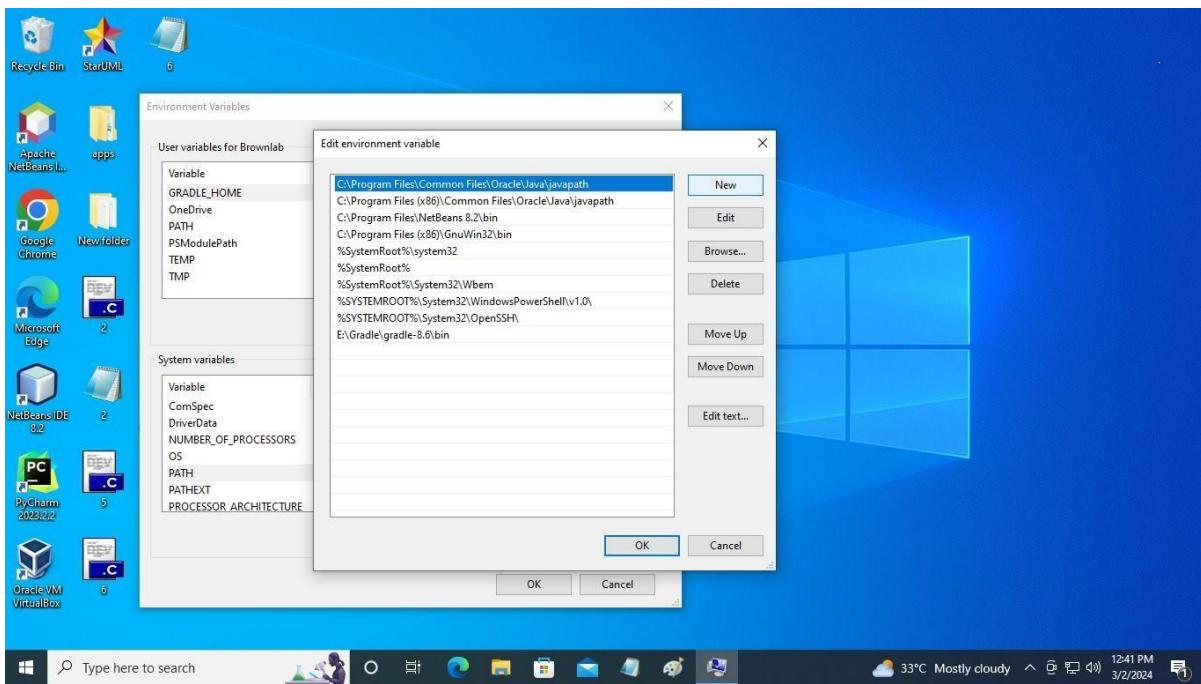
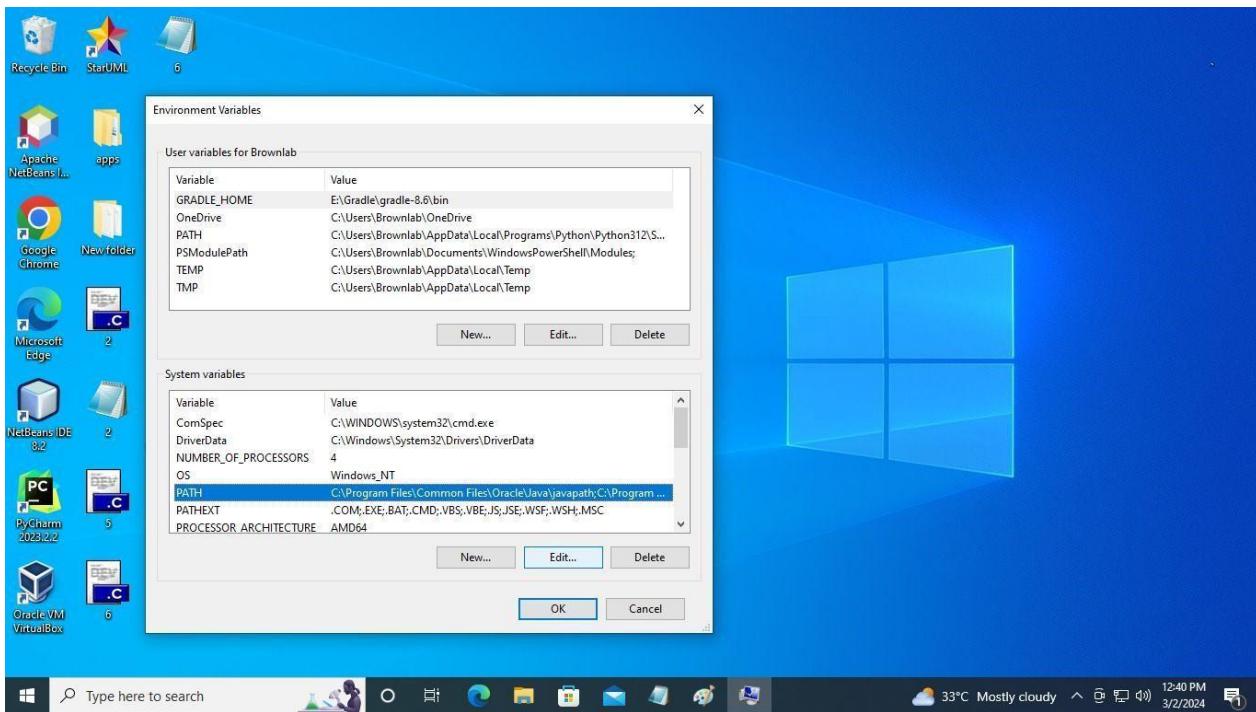
## Step 6: Click environment variable



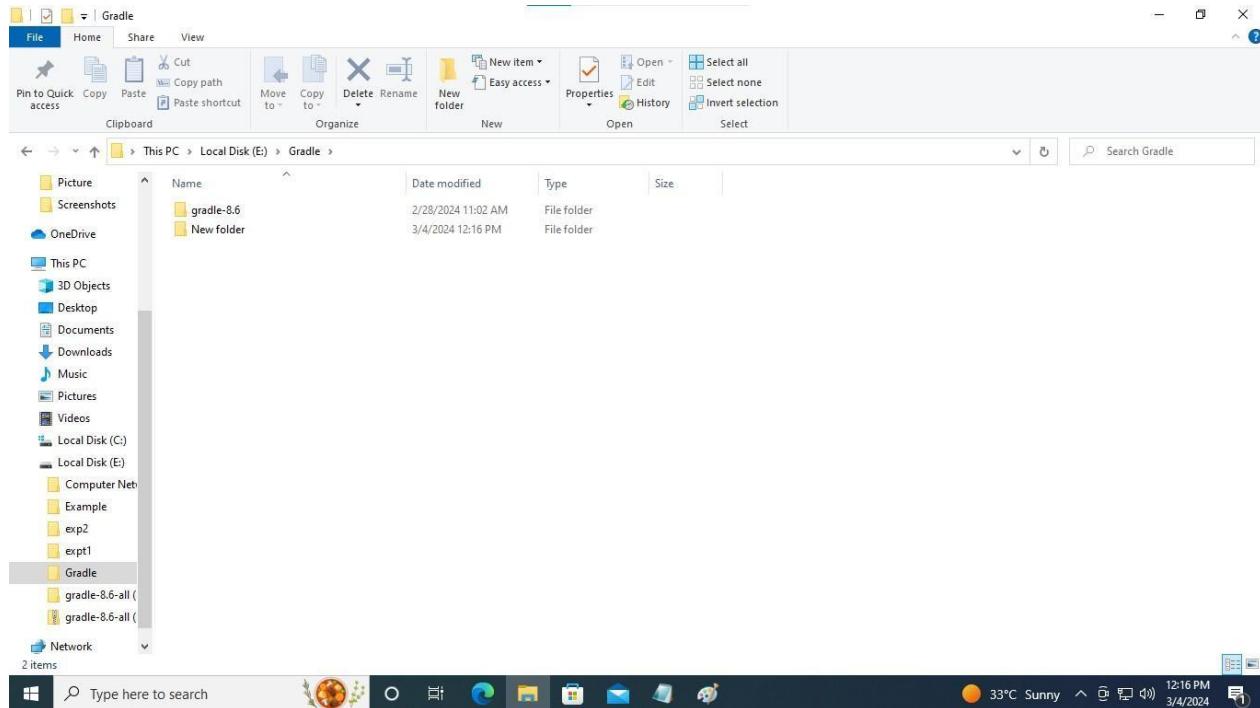
## Step 7: Click new and paste the variable name & variable value as below



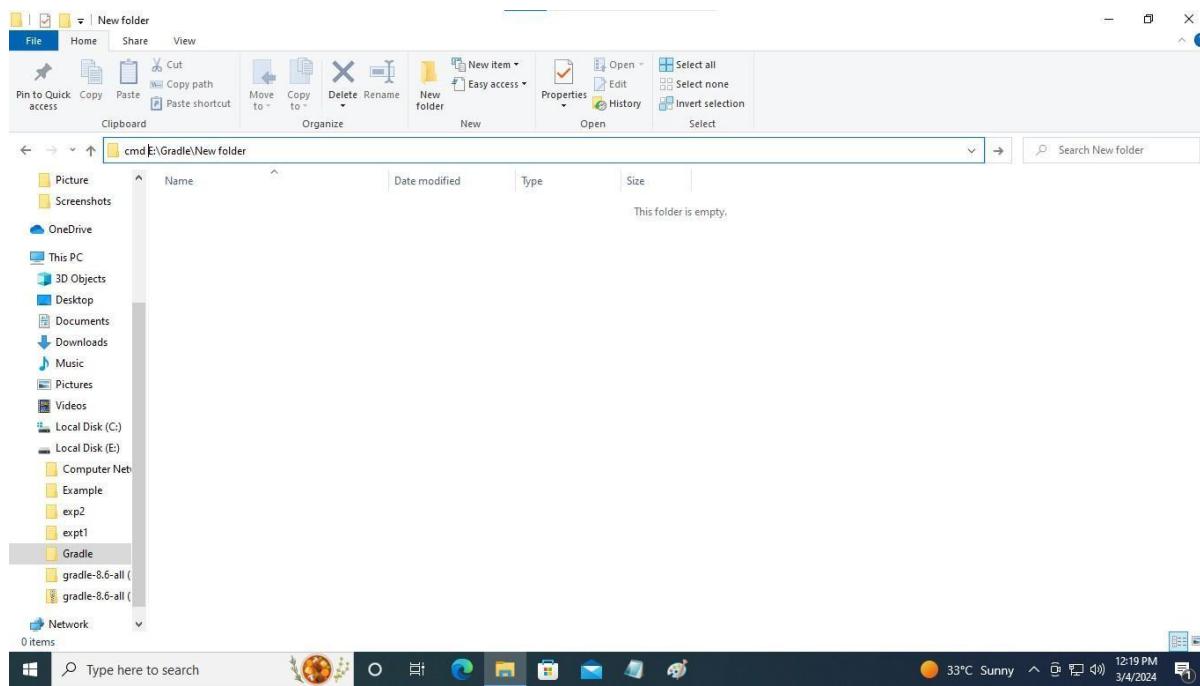
## Step 8: In path click edit and paste gradle path



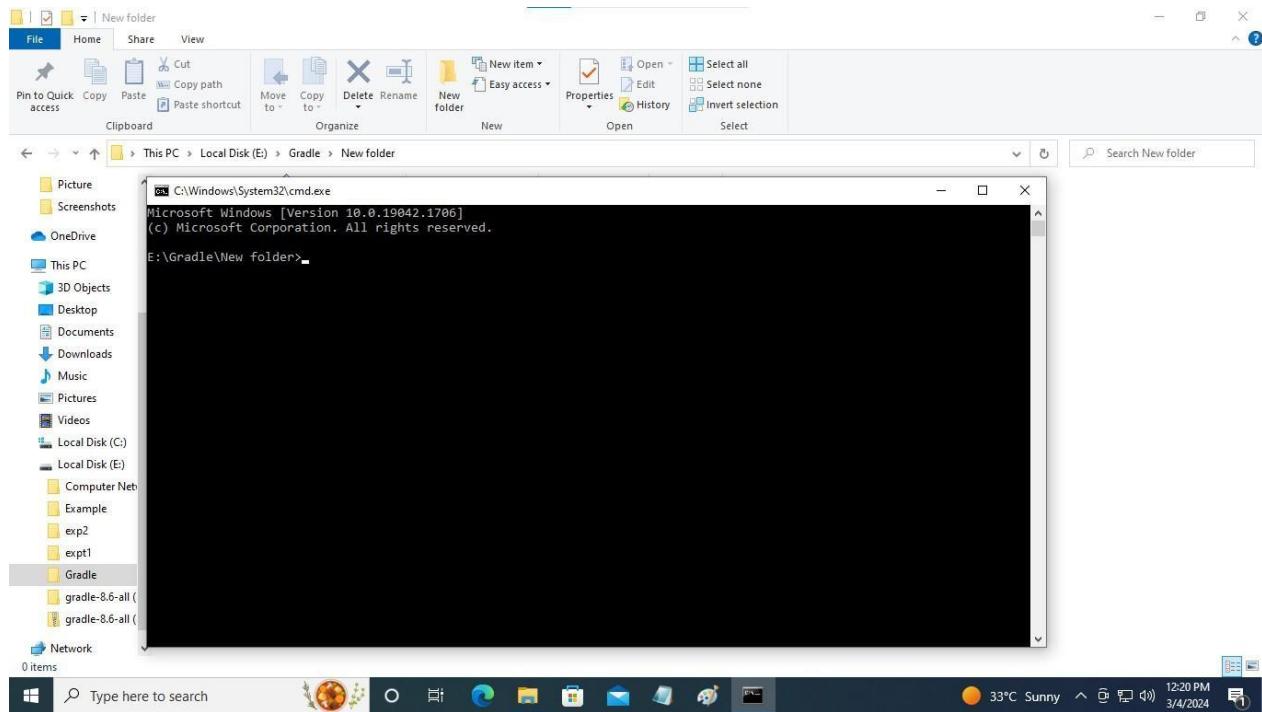
## Step 9: Create a new folder in gradle



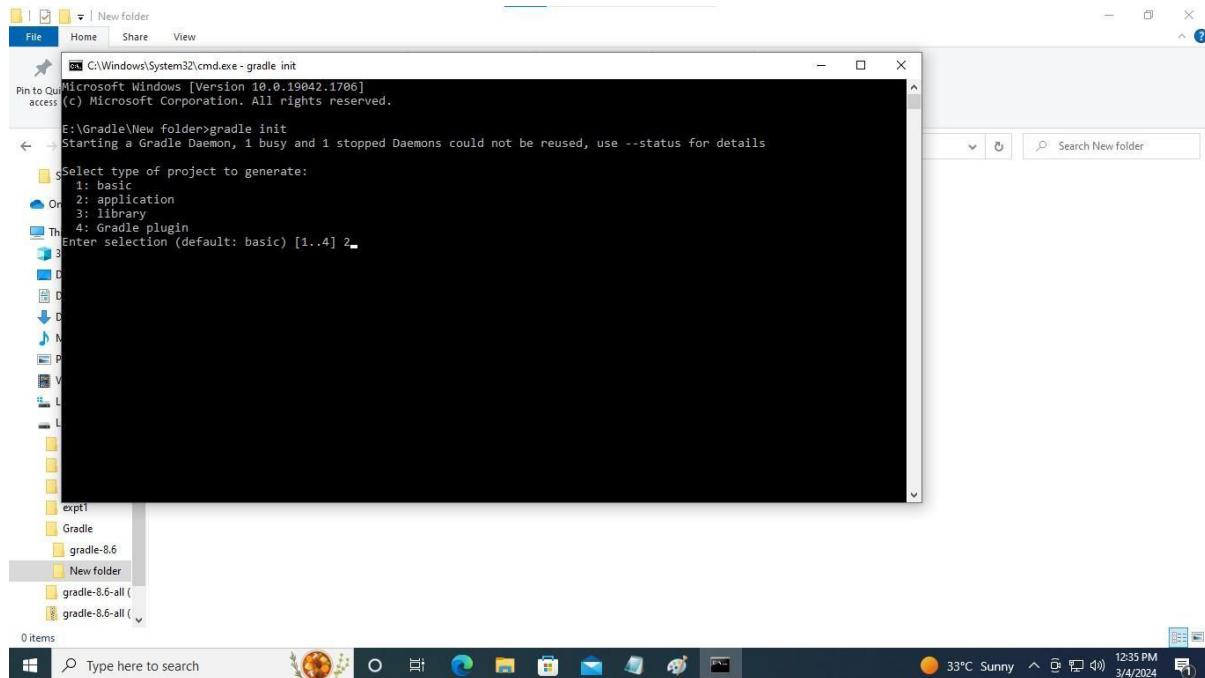
## Step 10: Open command prompt in that folder refer below picture



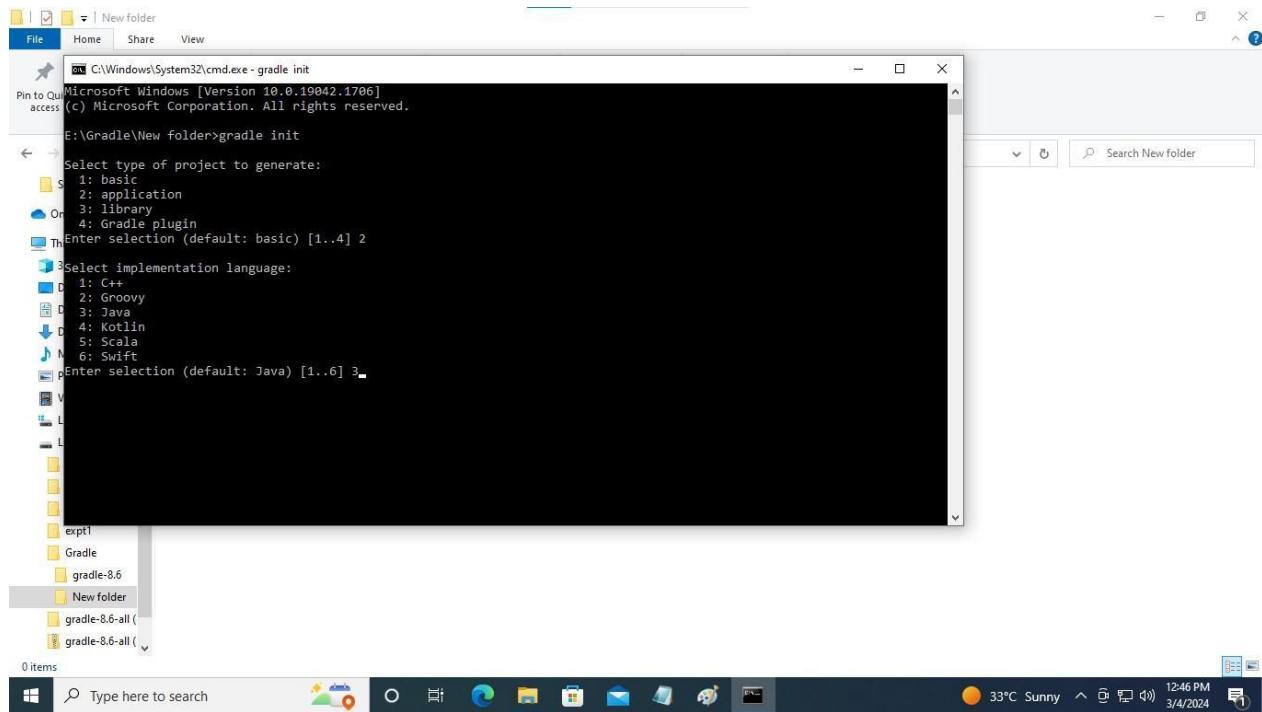
## Step 11: Command prompt opened



Step 12: Type the command gradle init and press enter .Now Select the type of project to generate and we choose number 2 [application]



## Step 13: Select the implementation language and we choose number 3 [Java]

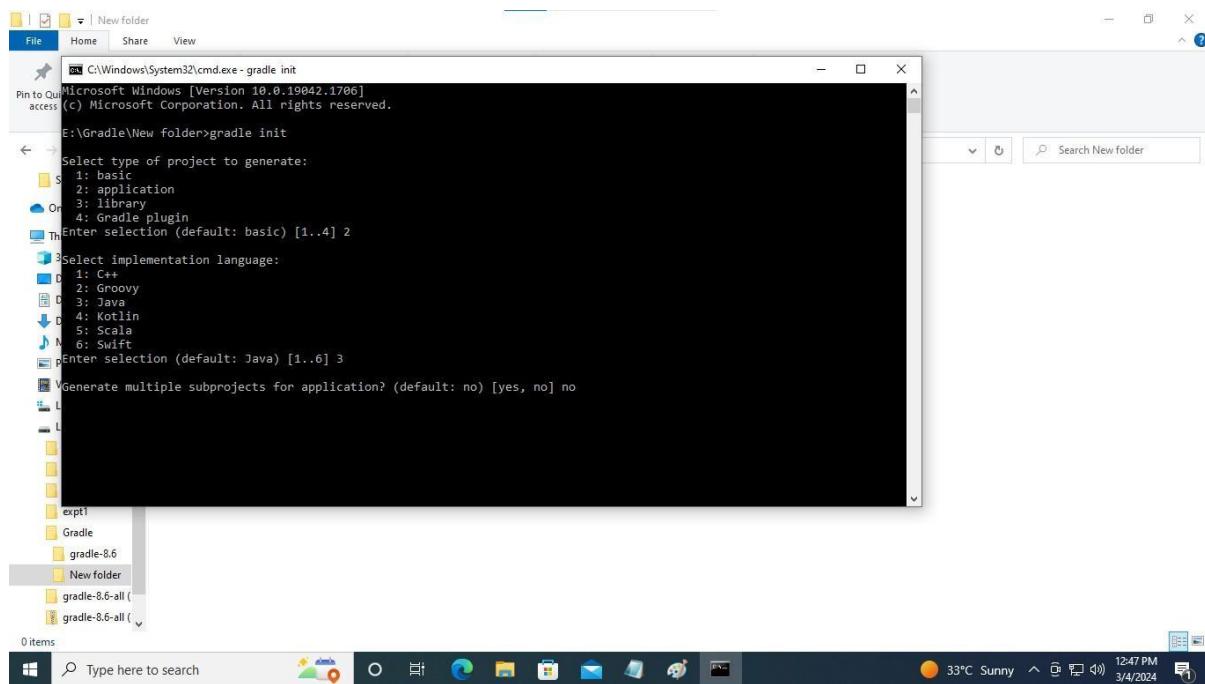


```
cmd C:\Windows\System32\cmd.exe - gradle init
Microsoft Windows [Version 10.0.19042.1706]
(c) Microsoft Corporation. All rights reserved.

E:\Gradle\New folder>gradle init
← → Select type of project to generate:
S 1: basic
2: application
O 3: library
4: Gradle plugin
T Enter selection (default: basic) [1..4] 2

PSelect implementation language:
D 1: C++
2: Groovy
G 3: Java
4: Kotlin
L 5: Scala
M 6: Swift
Enter selection (default: Java) [1..6] 3
```

## Step 14: Type NO to generate multiple subprojects for application



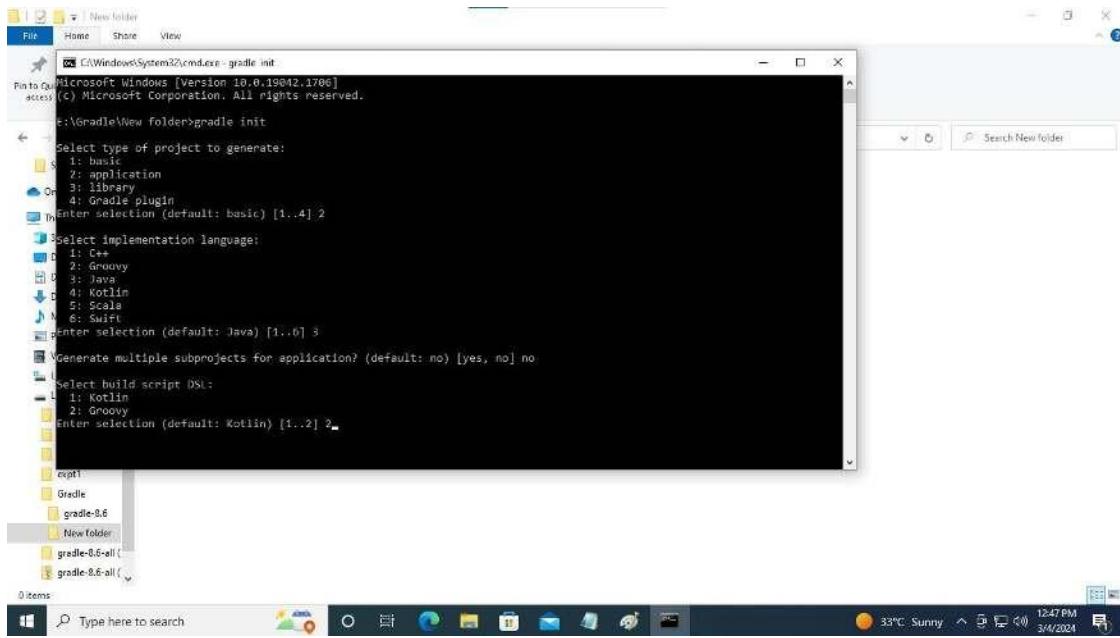
```
cmd C:\Windows\System32\cmd.exe - gradle init
Microsoft Windows [Version 10.0.19042.1706]
(c) Microsoft Corporation. All rights reserved.

E:\Gradle\New folder>gradle init
← → Select type of project to generate:
S 1: basic
2: application
O 3: library
4: Gradle plugin
T Enter selection (default: basic) [1..4] 2

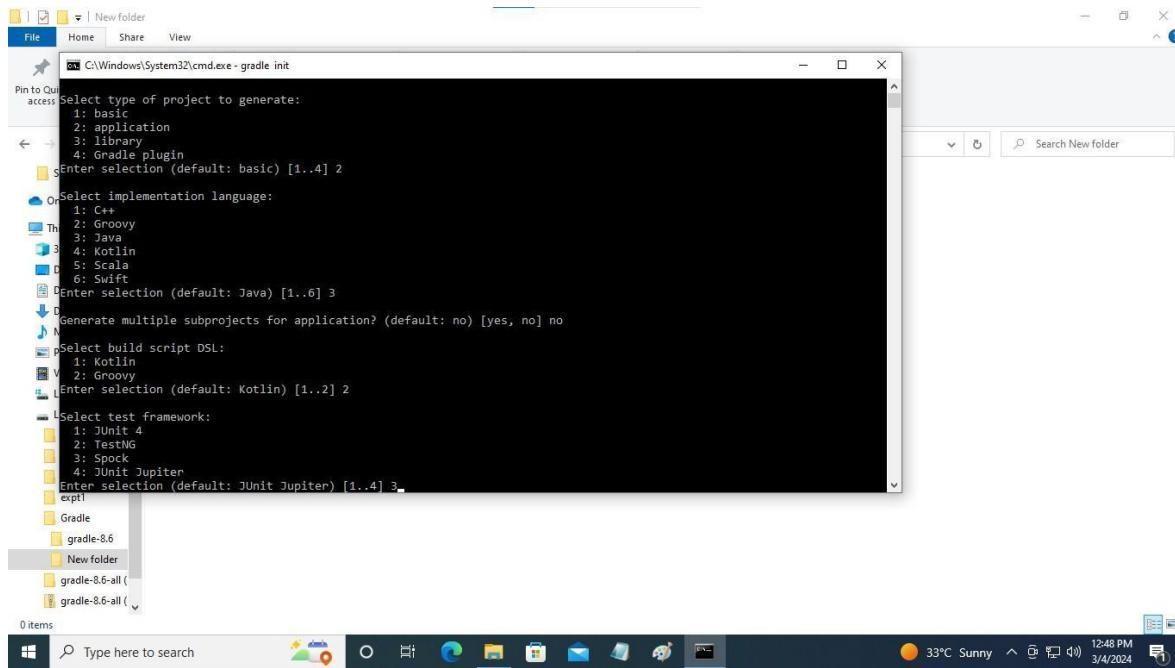
PSelect implementation language:
D 1: C++
2: Groovy
G 3: Java
4: Kotlin
L 5: Scala
M 6: Swift
Enter selection (default: Java) [1..6] 3

Generate multiple subprojects for application? (default: no) [yes, no] no
```

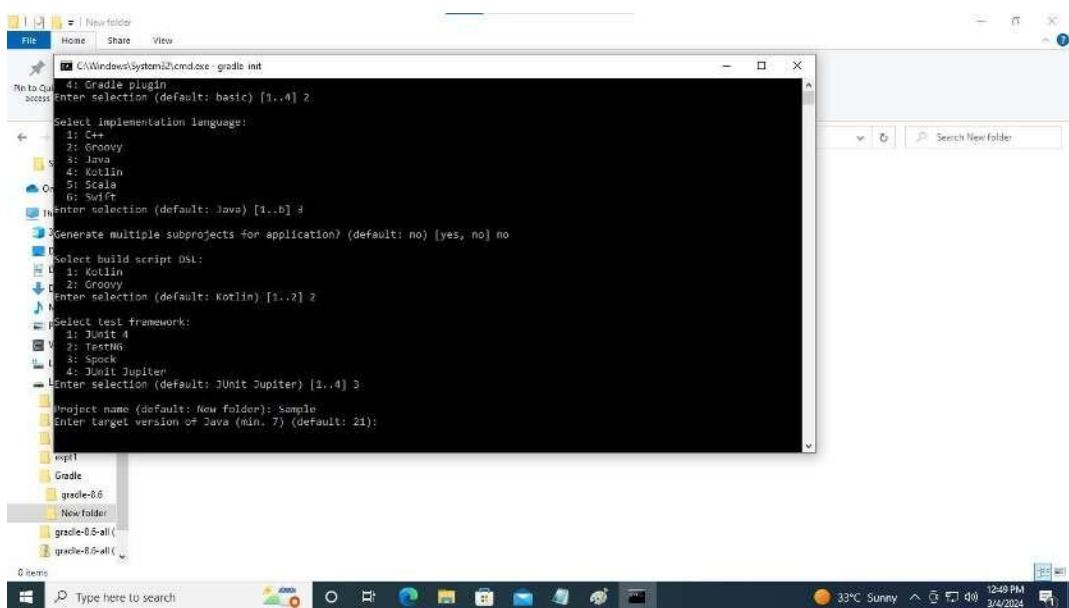
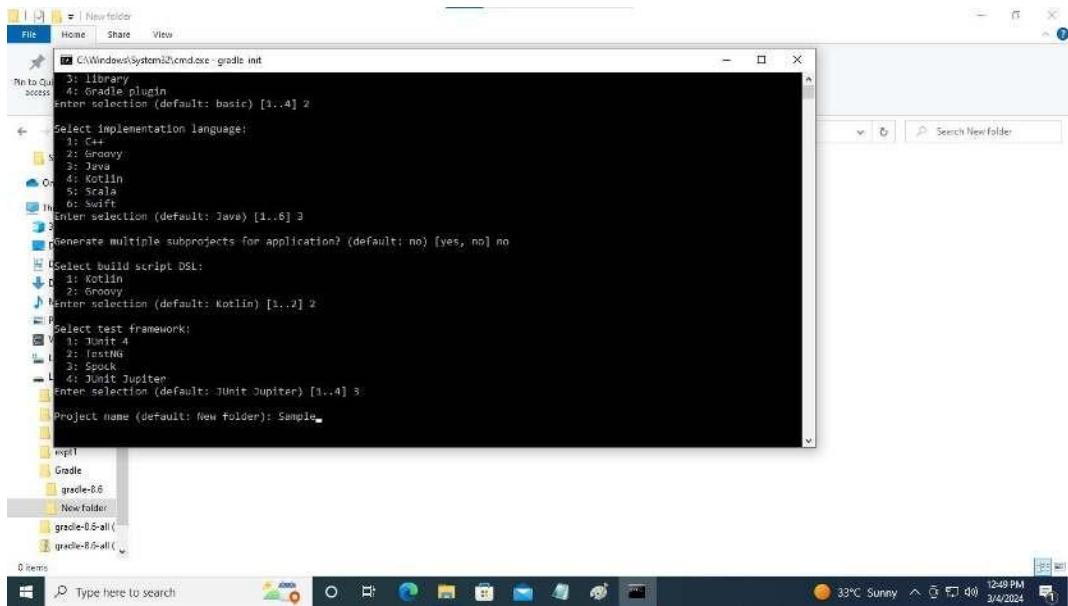
## Step 15: Select build script DSL and we choose 2 [Groovy]



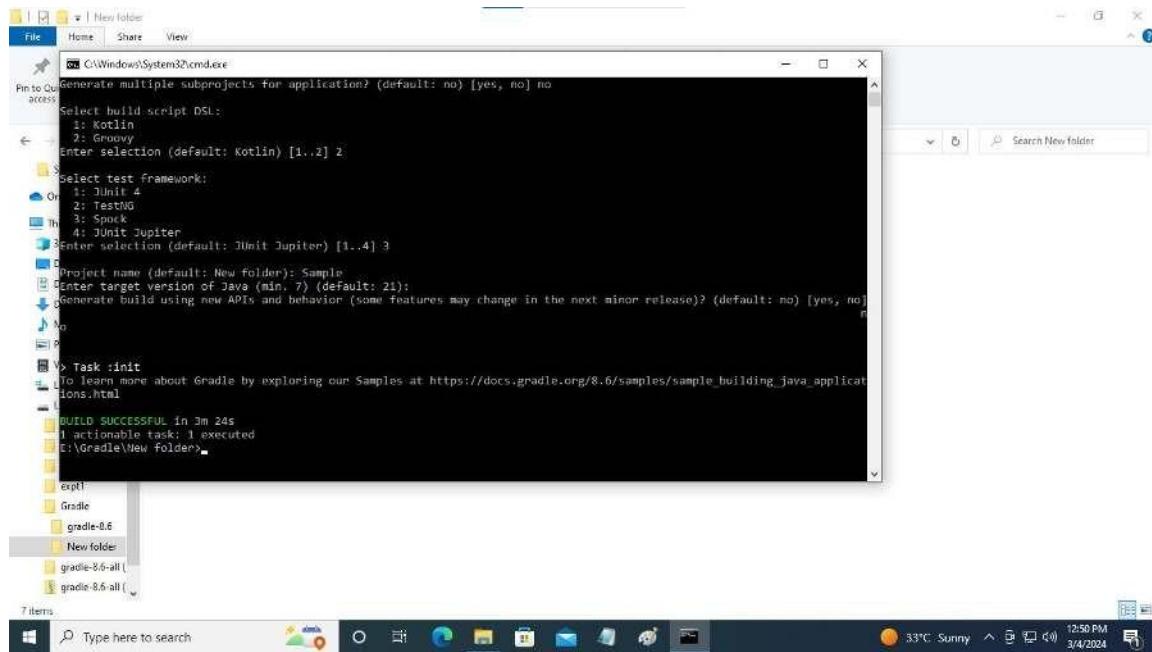
## Step16: Select test framework and we choose number 3 [Spock]



## Step 17: Give project name as sample and enter



Step 18: Type NO to generate build using new APIs and it gives the Task: init and display the message as BUILD SUCCESSFUL

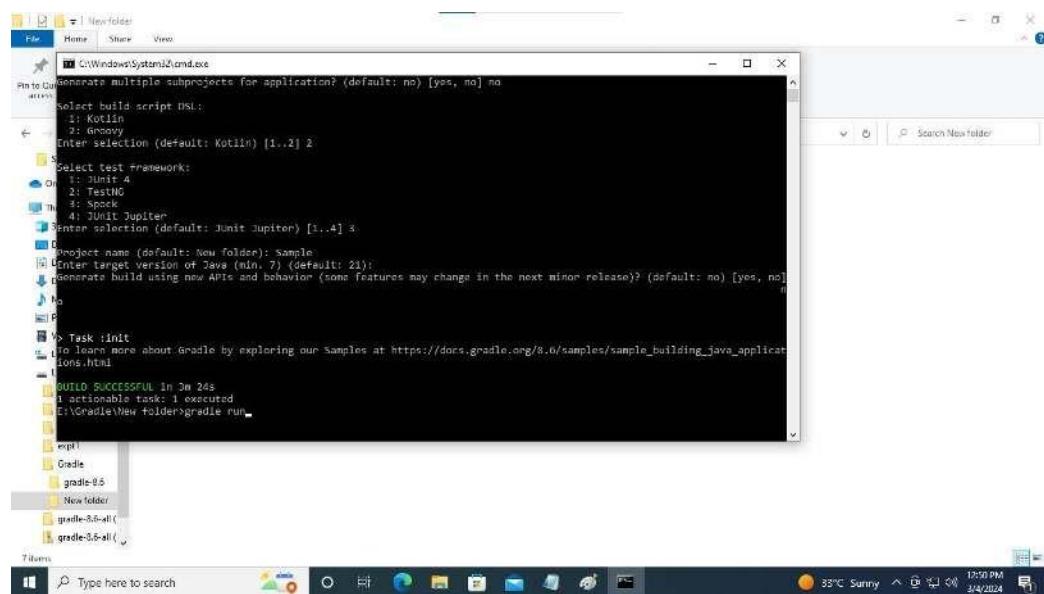


```
C:\Windows\System32\cmd.exe
Generate multiple subprojects for application? (default: no) [yes, no] no
Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2] 2
Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 3
Project name (default: New Folder): Sample
Enter target version of Java (min. 7) (default: 21):
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
No
>
> Task :init
To learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.6/samples/sample_building_java_applications.html
BUILD SUCCESSFUL in 3m 24s
1 actionable task: 1 executed
E:\Gradle\New Folder\

ext\ Gradle\ grade-8.6\ New folder\ gradle-8.6-all\ gradle-8.6-all\

7 items
Type here to search 33°C Sunny 12:50 PM 3/4/2024
```

Step 19: Now run gradle using the command gradle run

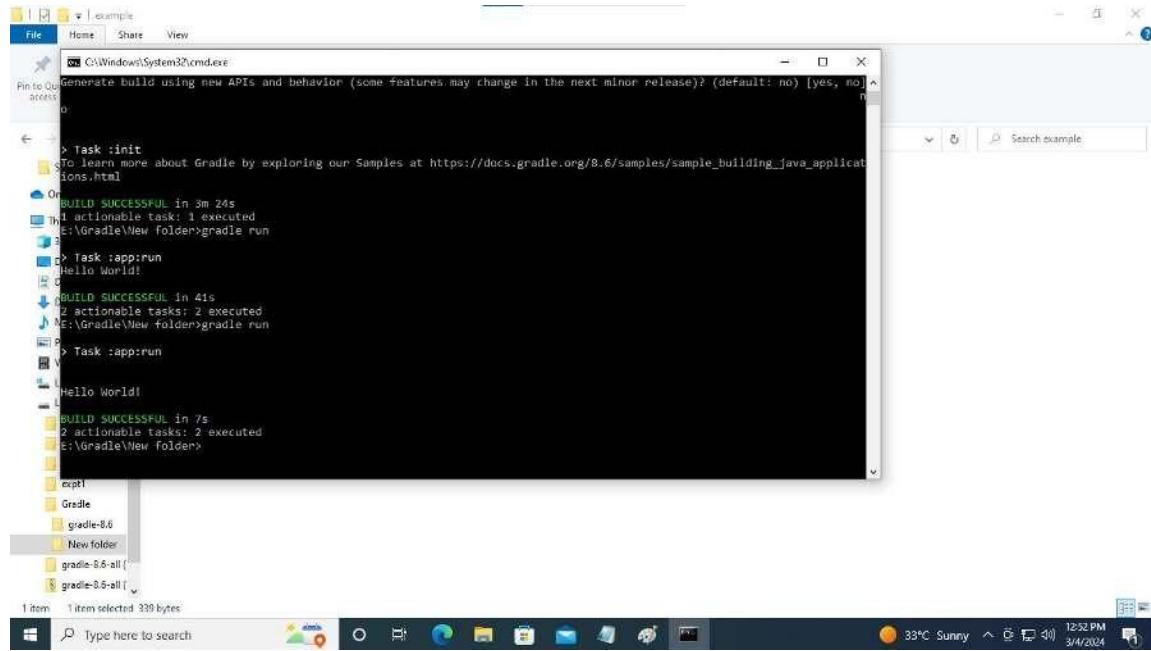


```
C:\Windows\System32\cmd.exe
Generate multiple subprojects for application? (default: no) [yes, no] no
Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2] 2
Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 3
Project name (default: New Folder): Sample
Enter target version of Java (min. 7) (default: 21):
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
No
>
> Task :init
To learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.6/samples/sample_building_java_applications.html
BUILD SUCCESSFUL in 3m 24s
1 actionable task: 1 executed
E:\Gradle\New folder\gradle run

ext\ Gradle\ grade-8.6\ New folder\ gradle-8.6-all\ gradle-8.6-all\

7 items
Type here to search 33°C Sunny 12:50 PM 3/4/2024
```

## Step 20: It gives the output as Hello World!



The screenshot shows a Windows Command Prompt window titled 'cmd.exe' running in the background of a desktop environment. The command prompt displays the following text:

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

```
<--> Task :init
  to learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.6/samples/sample_building_java_applications.html
  On
  BUILD SUCCESSFUL in 3m 24s
  1 actionable task: 1 executed
  E:\Gradle\New folder>gradle run
  E:\Gradle\New folder>
  <--> Task :app:run
  Hello World!
  E:\Gradle\New folder>gradle run
  E:\Gradle\New folder>
  <--> Task :app:run
  Hello World!
  E:\Gradle\New folder>
  BUILD SUCCESSFUL in 7s
  2 actionable tasks: 2 executed
  E:\Gradle\New folder>
```

The desktop taskbar at the bottom shows various icons for system functions like battery status, network, and date/time (3/4/2024, 12:32 PM). The system tray also displays weather information (33°C, Sunny).

## Result

Thus, a simple application using Gradle was Build successfully.

**Ex.N0:6**

## Install Ansible and Configure Ansible Roles and to write playbooks

**Date:**

**Aim:**

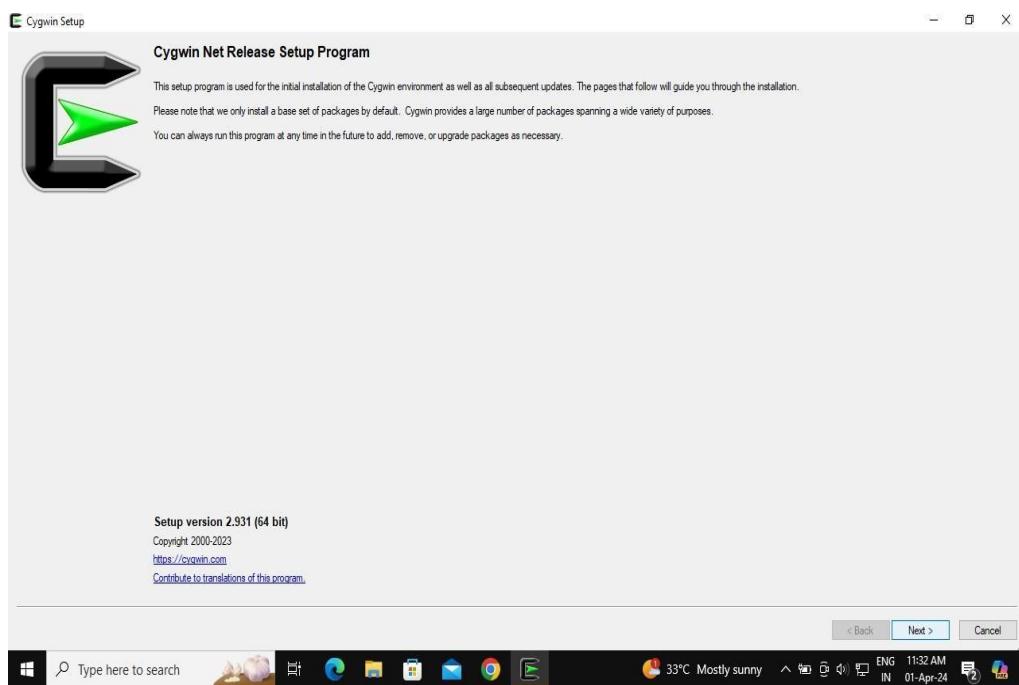
To install Ansible ,Configure Ansible roles and to write playbooks

**Procedure:**

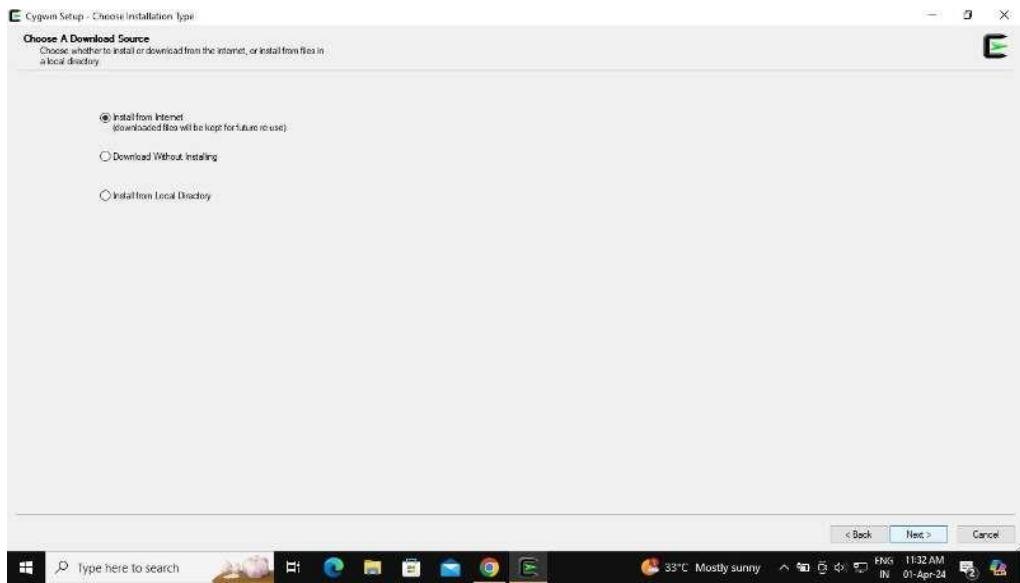
Step 1: Install Cygwin64 Terminal:



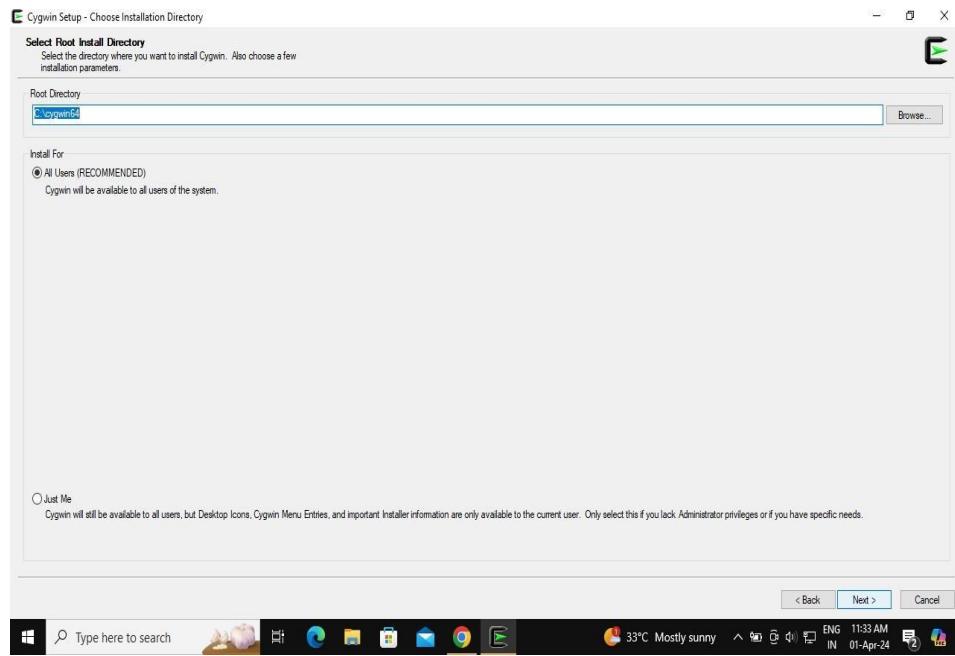
Step 2: Cygwin Net Release setup program



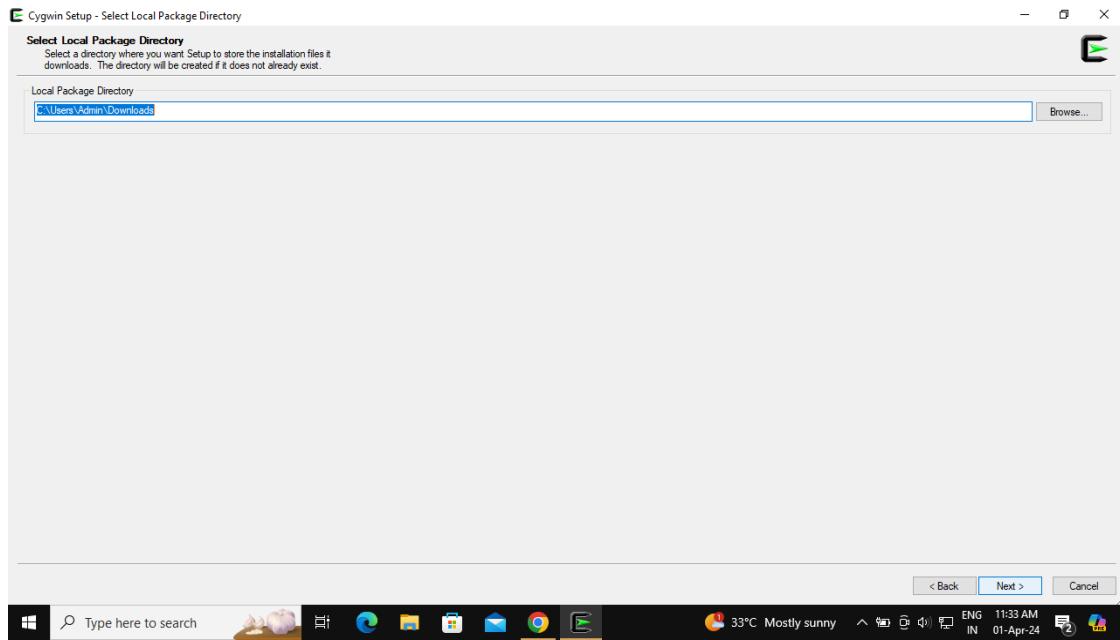
## Step 3: Here we choose the source



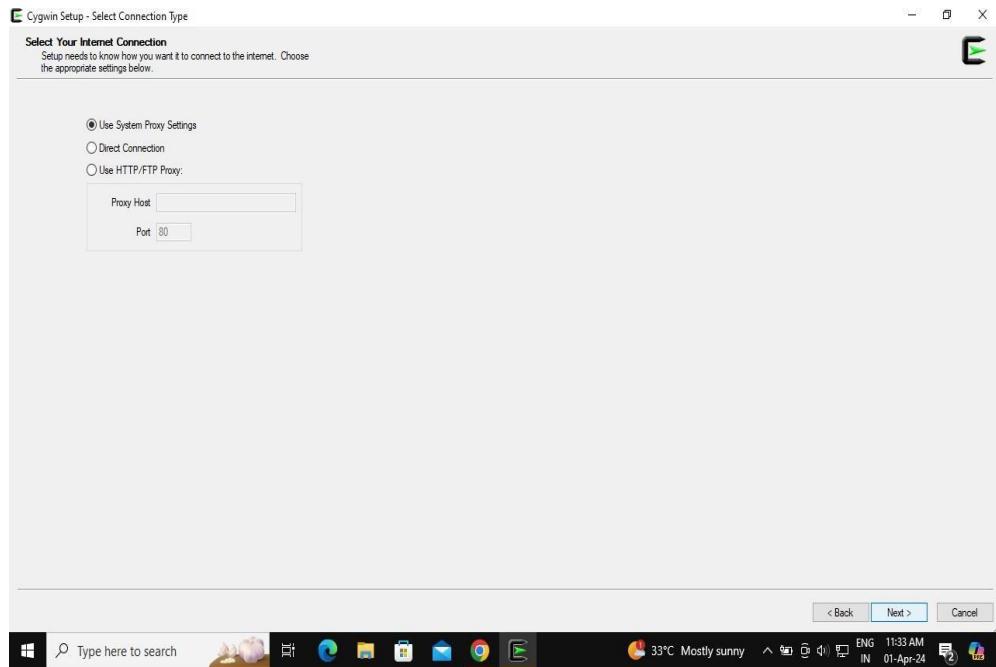
## Step 4: Select root to install directory



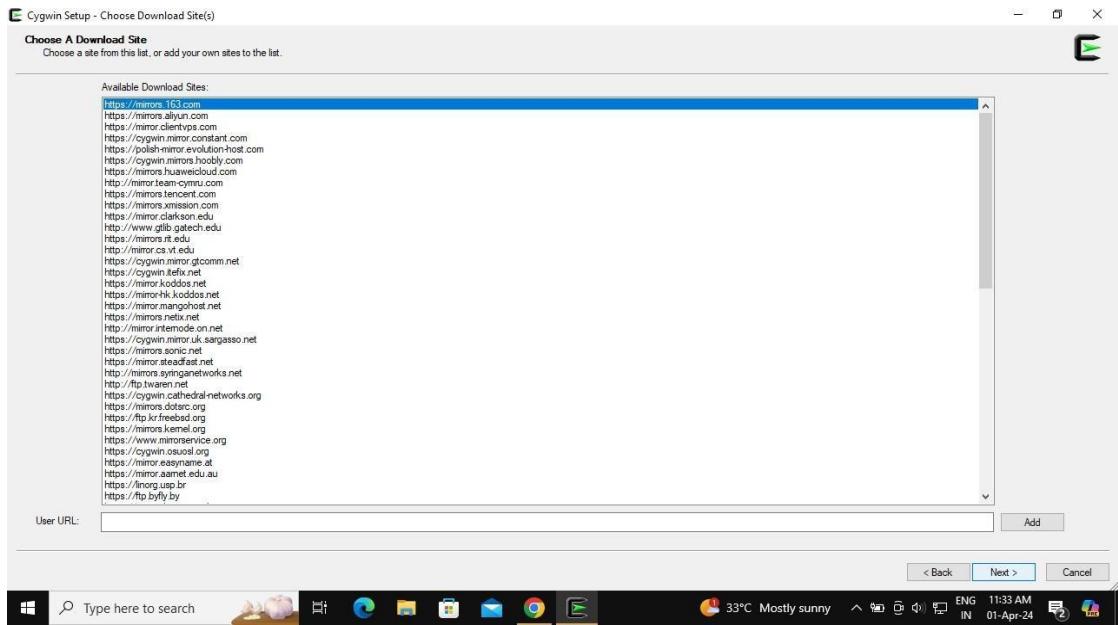
## Step 5: Select local package directory



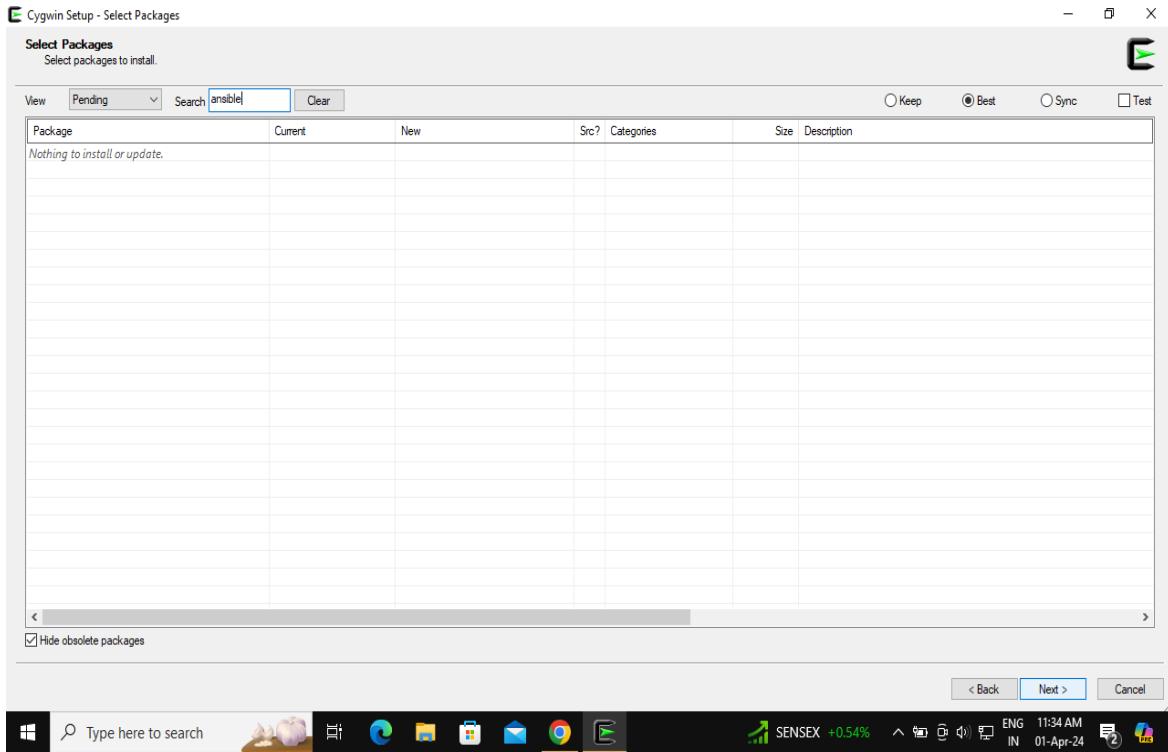
## Step 6: Here select your internet connection



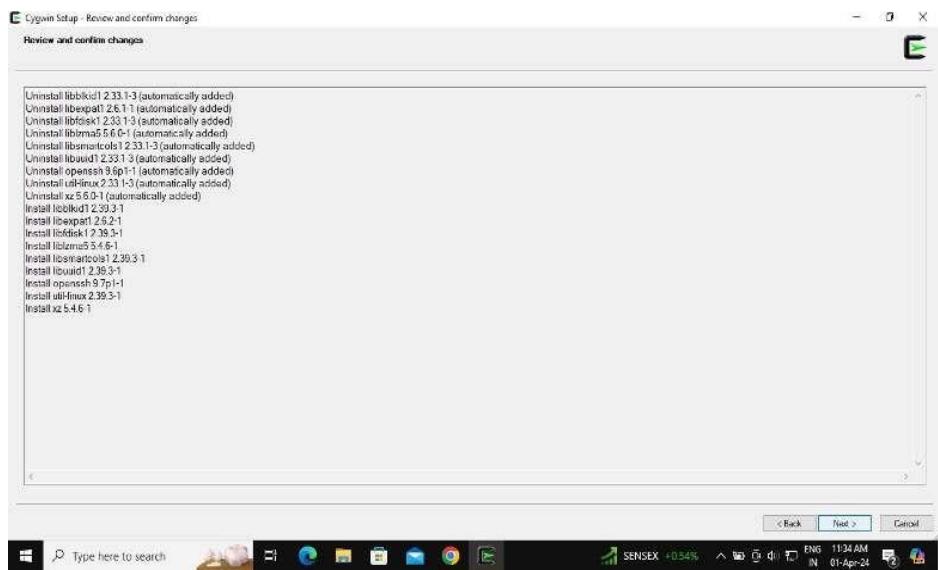
## Step 7: Choose a downloaded site



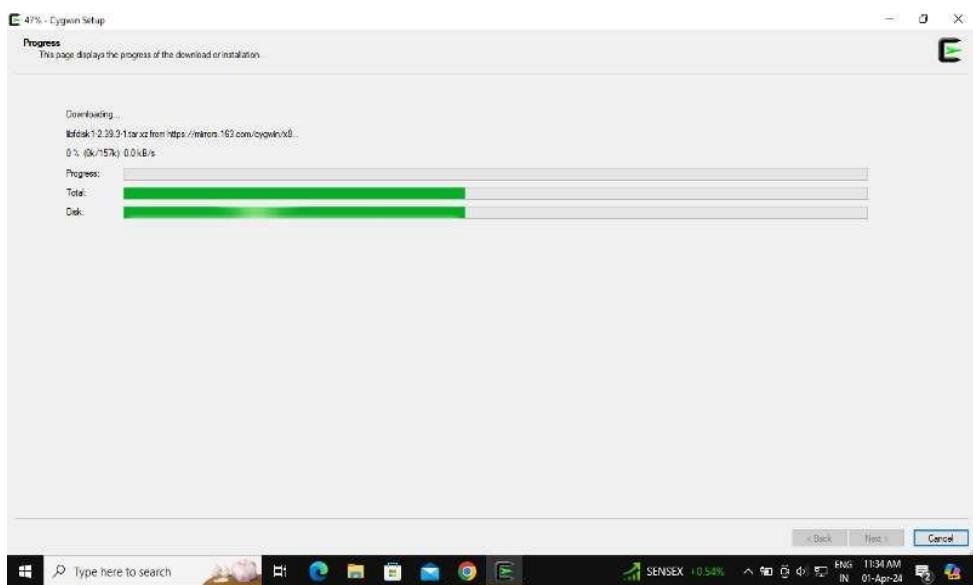
## Step 8: Select packages to install



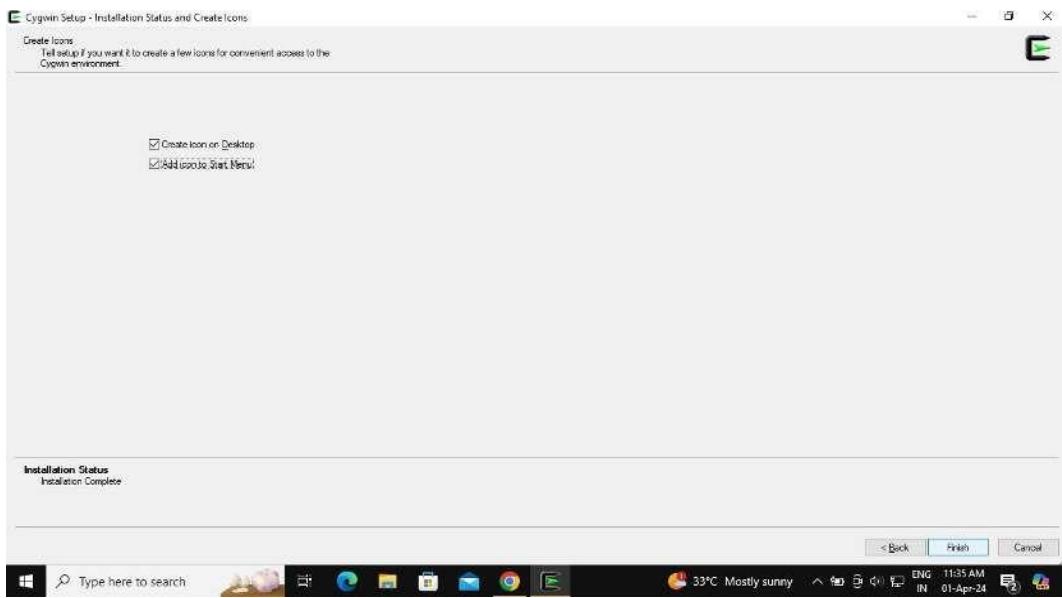
## Step 9: Review and confirm changes



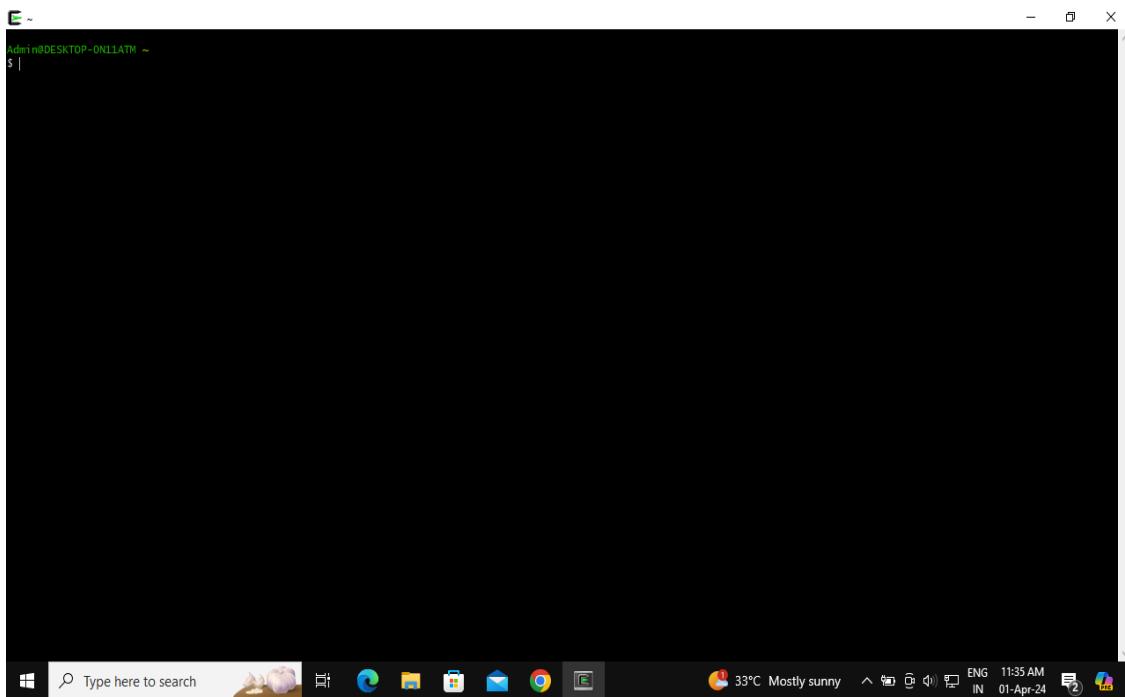
## Step 10: Downloading



## Step 11: installation complete

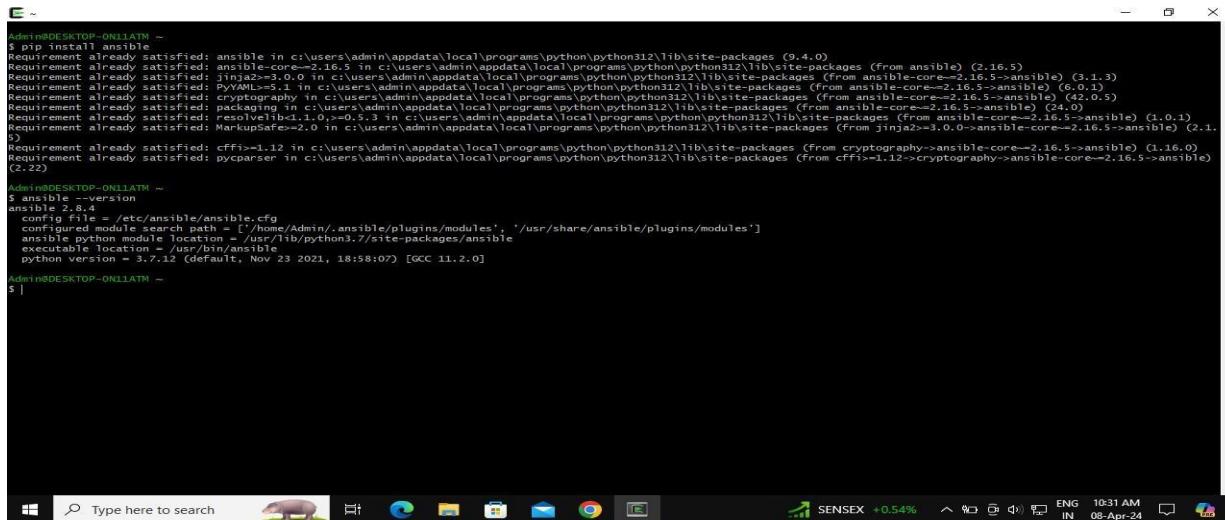


## Step 12: The cygwin was installed successfully.



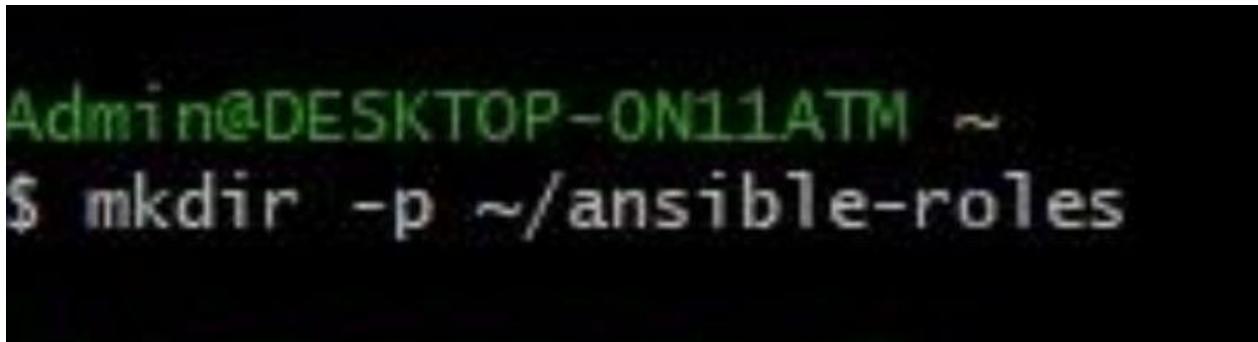
## INSTALL ANSIBLE:

Step 13: Install Ansible in cygwin command prompt as pip install ansible



```
Administrator:DESKTOP-ON11ATM ~
$ pip install ansible
Requirement already satisfied: ansible in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (9.4.0)
Requirement already satisfied: ansible-core==2.16.5 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from ansible) (2.16.5)
Requirement already satisfied: Jinja2<3.0.0,>=2.10.0 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from ansible) (3.1.3)
Requirement already satisfied: jinja2 >=2.10.0 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from ansible-core==2.16.5->ansible) (6.0.1)
Requirement already satisfied: cryptography in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from ansible-core==2.16.5->ansible) (42.0.5)
Requirement already satisfied: packaging in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from ansible-core==2.16.5->ansible) (24.0)
Requirement already satisfied: MarkupSafe<2.0 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from ansible-core==2.16.5->ansible) (<1.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from jinja2>=3.0.0->ansible-core==2.16.5->ansible) (2.1.5)
Requirement already satisfied: pyparser in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from cffi>=1.12>cryptography> ansible-core==2.16.5->ansible) (42.0.5)
Requirement already satisfied: cffi>=1.12 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from cryptography> ansible-core==2.16.5->ansible) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (from jinja2>=3.0.0->ansible-core==2.16.5->ansible) (2.1.5)
Administrator:DESKTOP-ON11ATM ~
$ |
```

Step 14: Create Ansible roles Directory



```
Administrator:DESKTOP-ON11ATM ~
$ mkdir -p ~/ansible-roles
```

Step 15: Using change directory command Create ansible roles



```
Administrator:DESKTOP-ON11ATM ~
$ cd ~/ansible-roles

Administrator:DESKTOP-ON11ATM ~/ansible-roles
$ mkdir common apache mysql
mkdir: cannot create directory 'common': File exists
mkdir: cannot create directory 'apache': File exists
mkdir: cannot create directory 'mysql': File exists
```

## Step 16: Organize the role Structure

```
Admin@DESKTOP-ON11ATM ~/ansible-roles
$ cd common

Admin@DESKTOP-ON11ATM ~/ansible-roles/common
$ mkdir tasks handlers defaults files
mkdir: cannot create directory 'tasks': File exists
mkdir: cannot create directory 'handlers': File exists
mkdir: cannot create directory 'defaults': File exists
mkdir: cannot create directory 'files': File exists
```

## Step 17: Write Tasks And Handlers

```
Admin@DESKTOP-ON11ATM ~/ansible-roles/common
$ mkdir -p ~/tasks

Admin@DESKTOP-ON11ATM ~/ansible-roles/common
$ cd ~/tasks
```

```
Admin@DESKTOP-ON11ATM ~/tasks
$ mkdir main.yml

Admin@DESKTOP-ON11ATM ~/tasks
$ cd main.yml

Admin@DESKTOP-ON11ATM ~/tasks/main.yml
$ |
```

## Step 18: Create Ansible Playbooks Directory

```
Admin@DESKTOP-ON11ATM ~
$ mkdir ~ansible-playbooks

Admin@DESKTOP-ON11ATM ~
$ cd ~ansible-playbooks

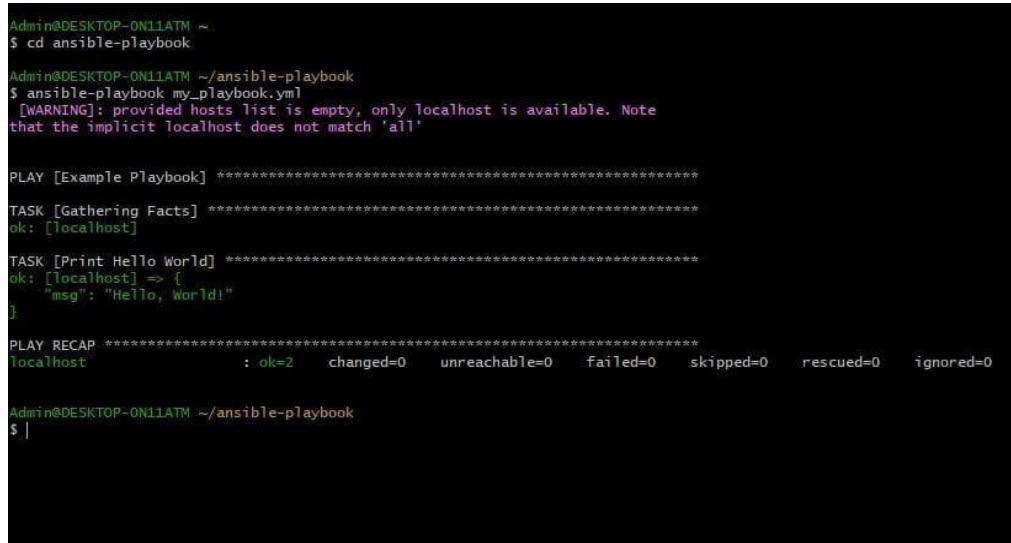
Admin@DESKTOP-ON11ATM ~/ansible-playbooks
$
```

Step 19: Open a notebook and type the program save it as my\_playbook.yml



```
my_playbook - Notepad
File Edit Format View Help
|---|
- name: Example Playbook
  hosts: localhost
  tasks:
    - name: Print Hello World
      debug:
        msg: "Hello, World!"
```

Step 20: Run the program and we get the output



```
Admin@DESKTOP-ON11ATM ~
$ cd ansible-playbook
Admin@DESKTOP-ON11ATM ~/ansible-playbook
$ ansible-playbook my_playbook.yml
[WARNING]: provided hosts list is empty, only localhost is available. Note
that the implicit localhost does not match 'all'

PLAY [Example Playbook] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [Print Hello World] ****
ok: [localhost] => {
    "msg": "Hello, World!"
}

PLAY RECAP ****
localhost          : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Admin@DESKTOP-ON11ATM ~/ansible-playbook
$ |
```

Result:

Thus, installing ansible, configure ansible roles and to write playbooks done successfully.

**EX.No:** Create a CD pipeline in Jenkins and deploy in Cloud

**Date:**

**Aim :**

To create a Continuous Deployment (CD) pipeline using Jenkins for automating the deployment of applications on a cloud platform, ensuring faster and reliable delivery of updates.

**Procedure :**

Step 1: Set Up Jenkins

- Install Jenkins on a server or use a cloud-hosted Jenkins instance.
- Configure Jenkins by installing necessary plugins (e.g., Git, Pipeline, Cloud-specific deployment plugins).
- Create a Jenkins user with appropriate permissions.

Step 2: Prepare the Cloud Environment

- Choose a cloud provider (e.g., AWS, Azure, GCP).
- Set up the infrastructure required for deployment (e.g., virtual machines, Kubernetes clusters, or app services).
- Generate and securely store the cloud credentials (API keys, access tokens).

Step 3: Create a Jenkins Pipeline

- Navigate to Jenkins and create a new pipeline project.
- Define the pipeline stages in a Jenkinsfile using the following typical structure:
  - Checkout Code: Pull code from the version control system (e.g., GitHub).
  - Build: Compile and build the application (e.g., Maven, Gradle).
  - Test: Run automated tests.
  - Deploy: Deploy the application to the cloud environment.

Step 4: Write the Jenkinsfile

Example structure:

```
pipeline {  
    agent any  
    stages {  
        stage('Checkout Code') {  
            steps {  
                git branch: 'main', url: 'https://github.com/username/repository.git'  
            }  
        }  
        stage('Build') {  
            steps {  
                sh 'mvn clean package'  
            }  
        }  
        stage('Test') {  
            steps {  
                sh 'mvn test'  
            }  
        }  
        stage('Deploy') {  
            steps {  
                sh 'aws deploy push --application-nameAppName --s3-location  
s3://bucket-name/'  
            }  
        }  
    }  
}
```

Step 5: Integrate Jenkins with Cloud

- Configure cloud provider plugins in Jenkins (e.g., AWS CLI, Azure DevOps).
- Store the cloud credentials in Jenkins using the Credentials Manager.
- Test connectivity between Jenkins and the cloud environment.

#### Step 6 : Trigger and Monitor the Pipeline

- Start the pipeline manually or configure it to trigger on code commits or schedule.
- Monitor each stage in the Jenkins dashboard for successful execution.

#### Step 7: Verify Deployment

- Access the deployed application in the cloud environment.
- Perform manual or automated tests to confirm the deployment was successful.

### **Result :**

The experiment successfully demonstrates the creation of a Continuous Deployment pipeline using Jenkins and its integration with a cloud platform.

**EX.No:** Create an Ansible playbook for a simple web application infrastructure

**Date:**

**Aim :**

To create an Ansible playbook for provisioning and configuring a basic web application infrastructure, including a web server and database server, ensuring consistency and automation in deployment.

**Procedure :**

**Step 1: Set Up Ansible**

- Install Ansible on a control node (e.g., Linux system).
- Configure SSH access to managed nodes (web server and database server) by setting up passwordless authentication using SSH keys.
- Define the inventory file (hosts) to list the target nodes.

**Step 2: Define the Infrastructure**

- Decide the components of the infrastructure:
  - A web server (e.g., Nginx or Apache).
  - A database server (e.g., MySQL).
- Identify required software packages and configurations.

**Step 3: Create an Inventory File**

- Example hosts file:

[web]

192.168.1.10

[db]

192.168.1.11

**Step 4: Write the Ansible Playbook**

- Create a YAML file (e.g., web\_app\_playbook.yml).
- Example playbook:

```
- name: Set up a simple web application infrastructure
hosts: all
become: yes
tasks:
  - name: Update the system
    apt:
      update_cache: yes
      upgrade: dist

  - name: Install web server
    when: "'web' in group_names"
    apt:
      name: apache2
      state: present

  - name: Copy website files
    when: "'web' in group_names"
    copy:
      src: /path/to/website/
      dest: /var/www/html/
      owner: www-data
      group: www-data
      mode: '0644'

  - name: Install database server
    when: "'db' in group_names"
    apt:
```

```
-name: mysql-server  
state: present
```

```
- name: Start services  
service:  
  name: "{{ item }}"  
  state: started  
  enabled: yes  
with_items:  
  - apache2  
  - mysql
```

Step 5:

### **Execute the Playbook**

- Run the playbook using the following command:

bash

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

### **Step 6: Verify the Infrastructure**

- Check the web server by accessing the application in a browser using the web server's IP address.
- Verify the database server by logging into MySQL and ensuring it is functional.

### **Step 7: Test and Validate**

- Test the web application for functionality.
- Confirm the infrastructure meets the requirements and is consistent across the nodes.

### **Result :**

The Ansible playbook successfully provisions and configures a simple web application infrastructure.