# Dev. Ops & Agile Programming Lab

**Git:** Version control system that helps to track changes in code

**Github**: Website that allows developers to store and manage their code using git

**How to install Git**

In order to use Git, we have to install it on wer computer. To do this, we can download the latest version on the official website. we can download for wer operating system from the options given.

**Create a GitHub account**

To be able to use GitHub, we will have to create an account first. we can do that on their website.

**How to configure Git**

I will assume that at this point we have installed Git. To verify this, we can run this command on the command line: git --version. This shows we the current version installed on we PC. The next thing we'll need to do is to set wer username and email address. Git will use this information to identify who made specific changes to files

To set wer username, type and execute these commands:

**git config --global user.name "YOUR_USERNAME**" and

**git config --global user.email "YOUR_EMAIL"**.

git config –list

Just make sure to replace "YOUR_USERNAME" and "YOUR_EMAIL" with the values we choose.

# Experiment no: 1

## Exploring Git Commands through Collaborative Coding.

**Experiment Steps:**

**Step 1: Setting Up Git Repository**

- Create a new folder
- Open the vs code teminal on your computer.
- Navigate to the directory where you want to create your Git repository.
- Run the following commands:

**Command 1: git init**

This initialises a new Git repository in the current directory

**Step 2: Creating and Committing Changes**

- Create a new text file named "example.txt" using any text editor.
- Add some content to the "example.txt" file.
- In the command-line interface, run the following commands:

**Command 2: git status**

This command shows the status of your working directory, highlighting untracked files

**Command 3: git add example.txt**

This stages the changes of the "example.txt" file for commit.

**Command 4: git reset example.txt**

Reverts changes in the working directory and/or staging area to a previous commit.

**git status**

**git add . (or) git add example.txt**

**Command 5: git commit -m "Add content to example.txt"**

This commits the staged changes with a descriptive message

**Step 3: Exploring History**

Modify the content of "example.txt."

Run the following commands:

**git status**

Notice the modified file is shown as "modified

**Command 6: git diff**

This displays the differences between the working directory and the last commit.

**Command 7: git log**

This displays a chronological history of commits.

**Step 4: Branching and Merging**

Create a new branch named "feature" and switch to it:

**Command 8: git branch feature**

**Command 9: git checkout feature**

or shorthand: **git checkout -b feature**

Make changes to the "example.txt" file in the "feature" branch.

Commit the changes in the "feature" branch.

Switch back to the "master" branch:

**Command 10: git checkout master**

Merge the changes from the "feature" branch into the "master" branch:

**Command 11: git merge feature**

**Step 5: Collaborating with Remote Repositories**

- Create an account on a Git hosting service like GitHub (https://github.com/).
- Create a new repository on GitHub.
- Link your local repository to the remote repository:

**Command 11: git remote add origin <repository_url>**

Push your local commits to the remote repository:

**Command 12: git push origin master**

# Experiment No. 2

## Implement GitHub Operations using Git

**Experiment Steps:**

**Step 1: Cloning a Repository**

- Sign in to your GitHub account.
- Find a repository to clone (you can use a repository of your own or any public repository).
- Click the "Code" button and copy the repository URL.
- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Run the following command:

**Command 1: git clone <repository_url>**

Replace <repository_url> with the URL you copied from GitHub.

This will clone the repository to your local machine.

**Step 2: Making Changes and Creating a Branch**

- Navigate into the cloned repository:

  **cd <repository_name>**
- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

  **Command 2: git status**
- Stage the changes for commit:

  **Command 3: git add example.txt**
- Commit the changes with a descriptive message:

  **Command 4: git commit -m "Add content to example.txt"**
- Create a new branch named "feature":

  **Command 5: git branch feature**
- Switch to the "feature" branch:

  **Command 6: git checkout feature**

**Step 3: Pushing Changes to GitHub**

- Add Repository URL in a variable

  **Command 7: git remote add origin <repository_url>**

- Replace <repository_url> with the URL you copied from GitHub.
- Verifies remote repository URLs associated with your local repository.

  **Command 8: git remote –v**

- Push the "feature" branch to GitHub:

  **Command 9: git push origin feature**

  Check your GitHub repository to confirm that the new branch "feature" is available.

**Step 4: Collaborating through Pull Requests**

- Create a pull request on GitHub:
- Go to the repository on GitHub.
- Click on "Pull Requests" and then "New Pull Request."
- Choose the base branch (usually "main" or "master") and the compare branch
- ("feature").
- Review the changes and click "Create Pull Request."
- Review and merge the pull request:
- Add a title and description for the pull request.
- Assign reviewers if needed.
- Once the pull request is approved, merge it into the base branch.

**Step 5: Syncing Changes**

- After the pull request is merged, update your local repository:

  **Command 10: git checkout main**

  **Command 11: git pull origin main**

**Step 6: Fetching changes**

**Command 12: git branch –r**

- The command git branch -r is used to list all remote branches in a Git repository
- You can see the list of branches in git hub
- Open github
- Create new branch in github
- Branch name :feature2

**Command 13: git fetch –all**

- It downloads all the latest commits, branches, and tags from all remotes configured in your repository.
- After fetching again see the list of branches in git hub
- **git branch –r**

**Step 7: Deleting branches**

**Command 14 : git push -d origin feature2**

- Deletes the branch feature 2 in github (remote)

**Command 15 :** git branch -d feature2

Deletes a specified branch locally

**git branch –r**

- The command git branch -r is used to list all remote branches in a Git repository
- You can see the list of branches in git hub

# Experiment No. 3

## Implement GitLab Operations using Git.

**Experiment Steps:**

**Step 1: Creating a Repository**

- Sign in to your GitLab account.
- Click the "New" button to create a new project.
- Choose a project name, visibility level (public, private), and other settings.
- Click "Create project."

**Step 2: Cloning a Repository**

- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Copy the repository URL from GitLab.
- Run the following command:

  **Command 1: git clone <repository_url>**
- Replace <repository_url> with the URL you copied from GitLab.
- This will clone the repository to your local machine.

**Step 3: Making Changes and Creating a Branch**

- Navigate into the cloned repository:
- **cd <repository_name>**
- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

  **Command 2: git status**
- Stage the changes for commit:
- **Command 3: git add example.txt**
- Commit the changes with a descriptive message:

  **Command 4: git commit -m "Add content to example.txt"**

- Create a new branch named "feature":

  **Command 5: git branch feature**

- Switch to the "feature" branch:

  **Command 6: git checkout feature**

**Step 4: Pushing Changes to GitLab**

- Add Repository URL in a variable

  **Command 7: git remote add origin <repository_url>**

- Replace <repository_url> with the URL you copied from GitLab.
- Push the "feature" branch to GitLab:

  **Command 8: git push origin feature**

- Check your GitLab repository to confirm that the new branch "feature" is available.

**Step 5: Collaborating through Merge Requests**

1. Create a merge request on GitLab:
   - Go to the repository on GitLab.
   - Click on "Merge Requests" and then "New Merge Request."
   - Choose the source branch ("feature") and the target branch ("main" or "master").
   - Review the changes and click "Submit merge request."
2. Review and merge the merge request:
   - Add a title and description for the merge request.
   - Assign reviewers if needed.
   - Once the merge request is approved, merge it into the target branch.

**Step 6: Syncing Changes**

After the merge request is merged, update your local repository:

**Command 9: git checkout main**

**Command 10: git pull origin main**

# Experiment No. 4

## Title: Implement BitBucket Operations using Git

**Step 1: Creating a Repository**

- Sign in to your Bitbucket account.
- Click the "Create" button to create a new repository.
- Choose a repository name, visibility (public or private), and other settings.
- Click "Create repository."

**Step 2: Cloning a Repository**

- Open your terminal or command prompt.
- Navigate to the directory where you want to clone the repository.
- Copy the repository URL from BitBucket.
- Run the following command:

  **git clone <repository_url>**
- Replace <repository_url> with the URL you copied from Bitbucket.
- This will clone the repository to your local machine.

**Step 3: Making Changes and Creating a Branch**

- Navigate into the cloned repository:
- cd <repository_name>
- Create a new text file named "example.txt" using a text editor.
- Add some content to the "example.txt" file.
- Save the file and return to the command line.
- Check the status of the repository:

  **git status**
- Stage the changes for commit:
- **git add example.txt**
- Commit the changes with a descriptive message:
- **git commit-m "Add content to example.txt"**
- Create a new branch named "feature":
- **git branch feature**

- Switch to the "feature" branch:
- **git checkout feature**

**Step 4: Pushing Changes to Bitbucket**

- Add Repository URL in a variable

  **git remote add origin <repository_url>**
- Replace <repository_url> with the URL you copied from Bitbucket.
- Push the "feature" branch to Bitbucket:

  **git push origin feature**
- Check your Bitbucket repository to confirm that the new branch "feature" is available.

**Step 5: Collaborating through Pull Requests**

1. Create a pull request on Bitbucket:
    - Go to the repository on Bitbucket.
    - Click on "Create pull request."
    - Choose the source branch ("feature") and the target branch ("main" or
    - "master").
    - Review the changes and click "Create pull request."

2. Review and merge the pull request:

- Add a title and description for the pull request.
- Assign reviewers if needed.
- Once the pull request is approved, merge it into the target branch.

**Step 6: Syncing Changes**

- After the pull request is merged, update your local repository:

  **git checkout main**

  **git pull origin main**

# Experiment No: 5

## Title: Implement BitBucket Operations using Git

**Experiment Steps:**

**Step 1: Setting Up Bitbucket Repository**

- Create a new folder
- Open the gitbash on your computer.
- Navigate to the directory where you want to create your Git repository.
- Run the following commands:

**git init**

This initialises a new Git repository in the current directory

**Step 2: Creating and Committing Changes**

- Create a new text file named "example.txt" using any text editor.
- Add some content to the "example.txt" file.
- In the command-line interface, run the following commands:

**git status**

This command shows the status of your working directory, highlighting untracked files

**git add example.txt**

This stages the changes of the "example.txt" file for commit.

**git commit -m "Add content to example.txt"**

This commits the staged changes with a descriptive message

**Step 3: Exploring History**

Modify the content of "example.txt."

Run the following commands:

**git status**

Notice the modified file is shown as "modified

**git diff**

This displays the differences between the working directory and the last commit.

**git log**

This displays a chronological history of commits.

**Step 4: Branching and Merging**

Create a new branch named "feature" and switch to it:

**git branch feature**

**git checkout feature**

or shorthand: **git checkout -b feature**

Make changes to the "example.txt" file in the "feature" branch.

Commit the changes in the "feature" branch.

Switch back to the "master" branch:

**git checkout master**

Merge the changes from the "feature" branch into the "master" branch: git merge feature

**Step 5: Collaborating with Remote Repositories**

- Create an account on a Git hosting service like GitHub (https://github.com/).
- Create a new repository on GitHub.
- Link your local repository to the remote repository:

  **git remote add origin <repository_url>**

  Push your local commits to the remote repository:

  **git push origin master**

**Step 6: Merging Conflicts**

- This usually happens when changes are made to the same lines of a file or when one person deletes a file that another person edits.
- List our current branches

  **git branch –vv**
- Create new branch and switch to new branch

  **git checkout –b branch1**
- Switch to master branch

  **git checkout master**
- Create new branch and switch to new branch

  **git checkout –b branch2**
- see the list of branches

  **git branch -vv**

- switch to branch1

  **git checkout branch1**
- make changes in branch1
- add and commit changes made in branch1

  **git add .**

  **git commit –m "commit changes"**
- push branch 1 to github

  **git push origin branch1**
- switch to branch2

  **git checkout branch2**
- Make the changes in branch 2 similar to those you have done in branch 1
- Add and commit changes made in branch 2

  **git add .**

  **git commit -m "save changes made in branch 2"**
- push changes to git hub

  **git push origin branch2**
- switch to master branch

  **git checkout master**
- merge master with branch1

  **git merge branch1**

  **git status**

  **git push**
- merge master with branch2

  **git merge branch2**
- now merge conflict arrises
- now make the correct changes in code
- add and the changes and commit

  **git add .**

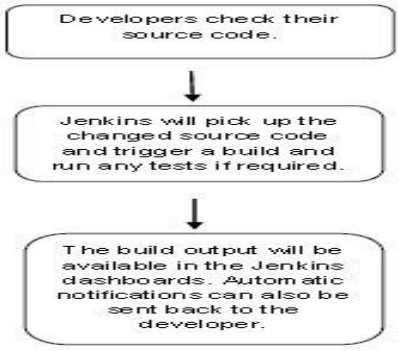
  **git commit –m "add changes"**

  **git push**

# JENKINS

## WHAT IS JENKINS?

- Jenkins is an open source automation tool written in Java programming language that allows continuous integration.
- Jenkins builds and tests our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.
- It also allows us to continuously deliver our software by integrating with a large number of testing and deployment technologies.
- Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.
- With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.
- Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.
- For example: If any organization is developing a project, then Jenkins will continuously test your project builds and show you the errors in early stages of your development.


Developers check their source code.
Jenkins will pick up the changed source code and trigger a build and run any tests if required.
The build output will be available in the Jenkins dashboards. Automatic notifications can also be sent back to the developer.

# Experiment No: 6

## To install and configure Jenkins to test and deploy Java or Web Applications.

To install Jenkins following software packages are required

1) GIT (git-scm.com)

2) Latest Java development kit (JDK)

3) Jenkins

**Process of installing Jenkins**

**Step 1**: Open any web browser

**Step 2**: Search for Jenkins

**Step 3**: Select download and deploy option (https://www.jenkins.io/download/)

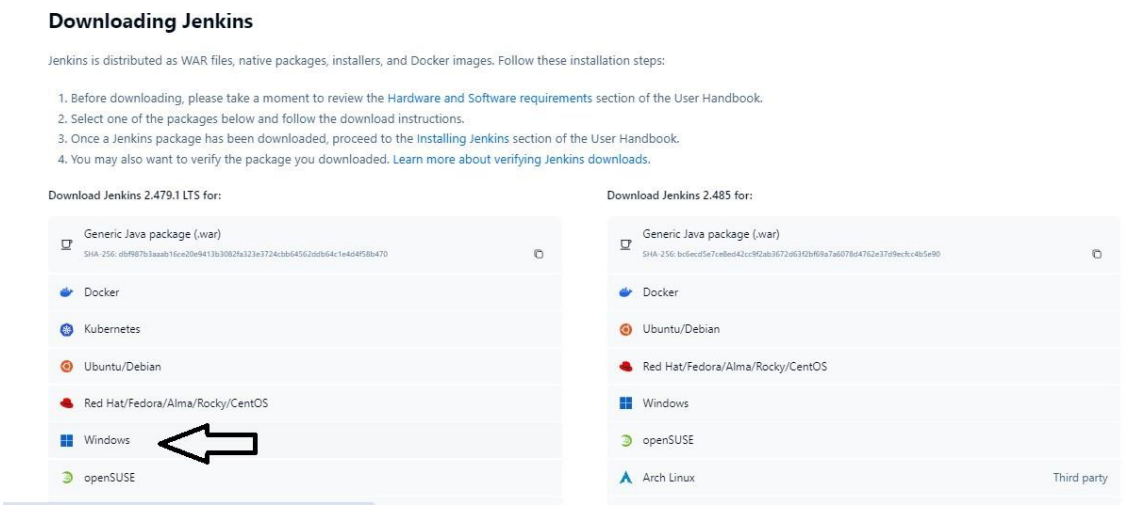
The Jenkins project produces two release lines: Stable (LTS) and weekly. Depending on your organization's needs, one may be preferred over the other.

**Step 4**: in the stable(LTS) select windows to download Jenkins software

This is called the Jenkins **Long-Term Support** release, or **LTS**.

**Stable (LTS)**

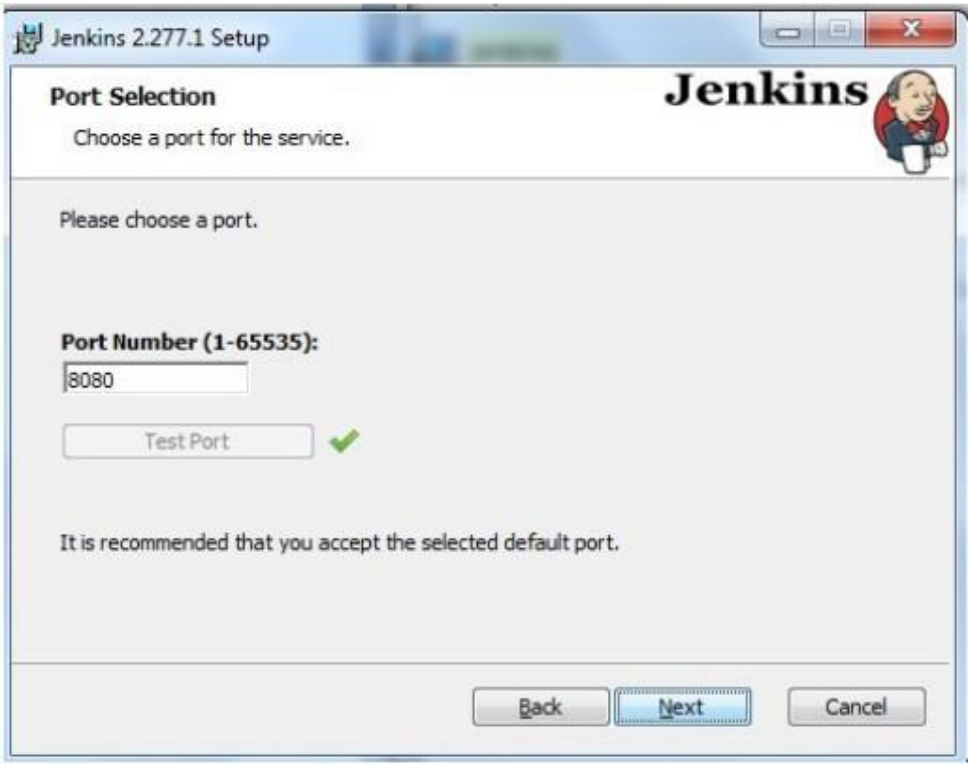
Long-Term Support (LTS) release baselines are chosen every 12 weeks from the stream of regular releases. Every 4 weeks we release stable releases which include bug and security fix backports



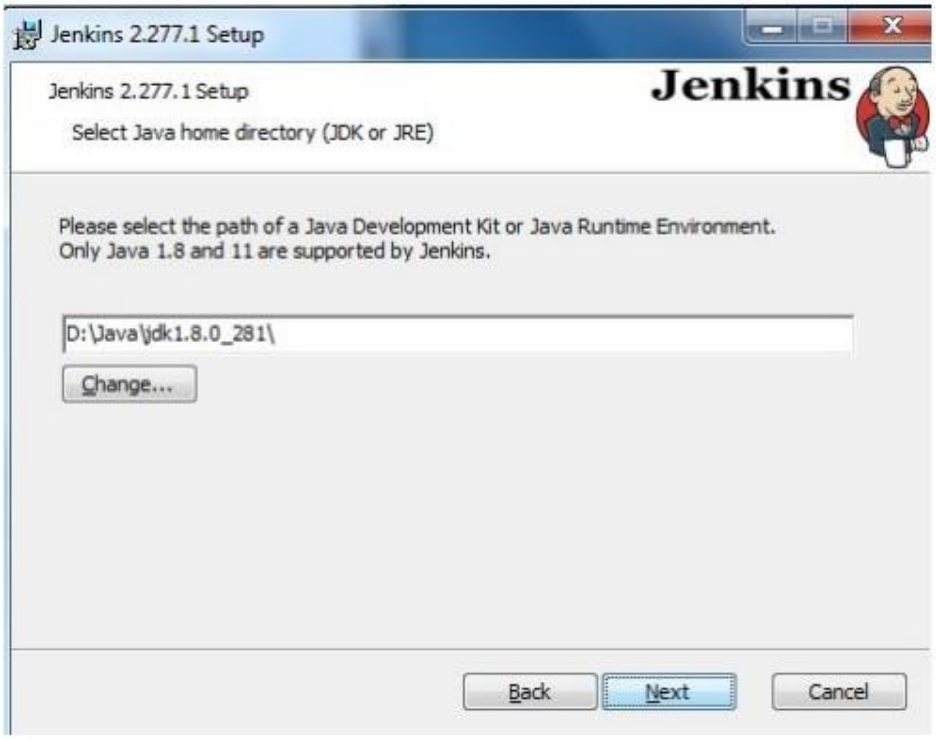

**Step5:** After downloading Jenkins →Open downloads folder →right click on Jenkins →select Run as Administrator → A setup box is opened →click next button.

**Step 4:** Install Jenkins


**Step6:** Check the port number, if it shows that port is used by some other software, then change the port number (ex:8090,8081 etc).

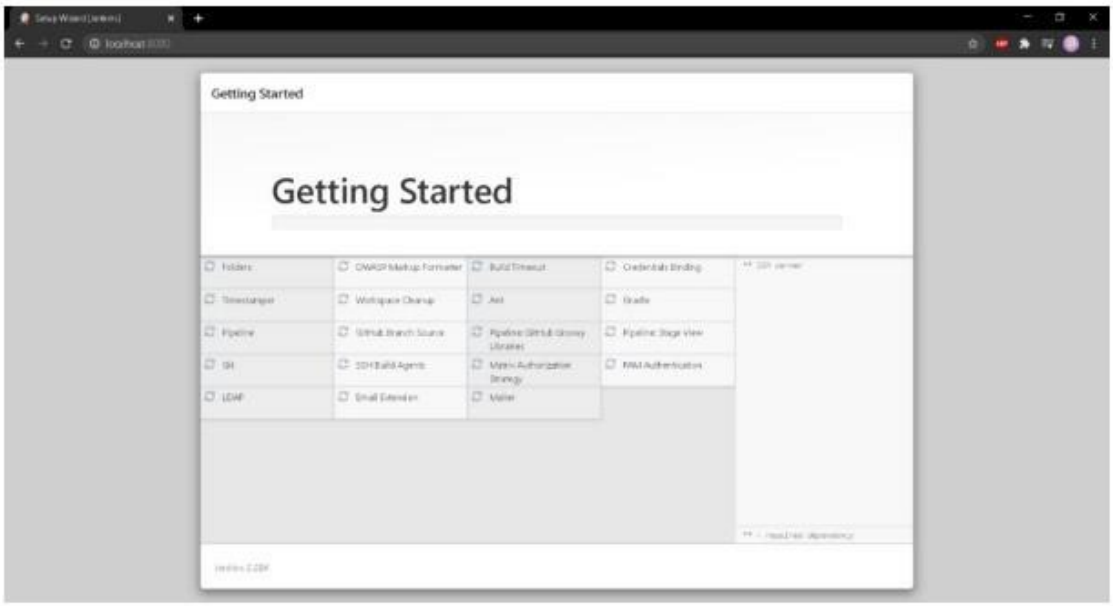



**Step7:** Check the path of JDK weather it is correct or not.



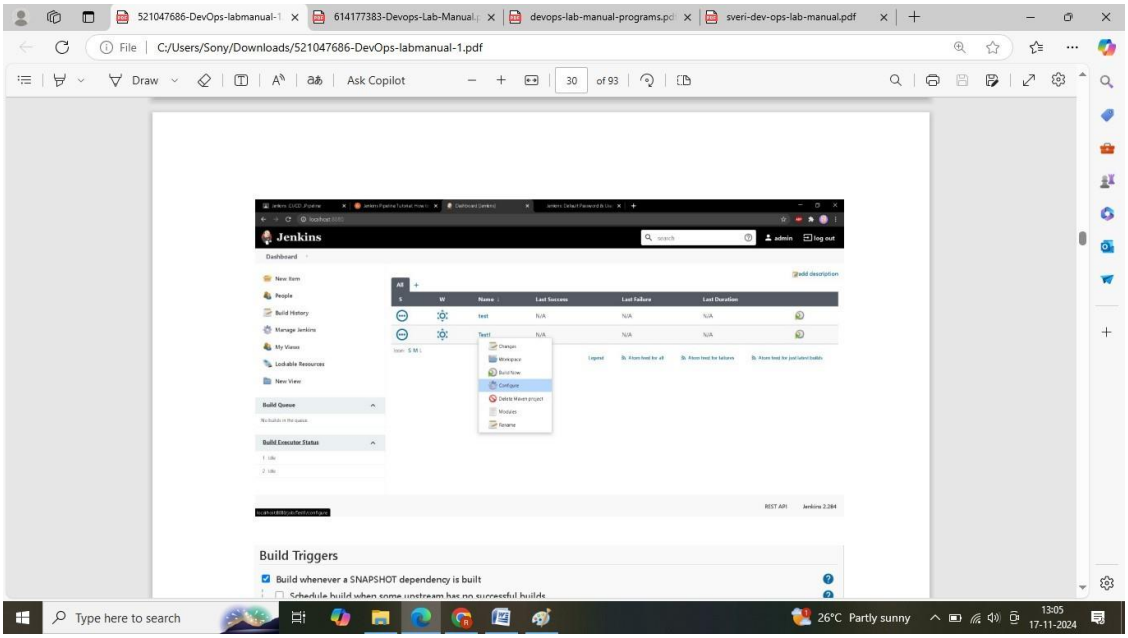
**Step8:** Click install button to install the Jenkins


**Step9:**



**Step10:** Jenkins dashboard is opened

Now we are ready to use Jenkins



# Experiment No: 7

## Building a Java Application from GitHub using Jenkins

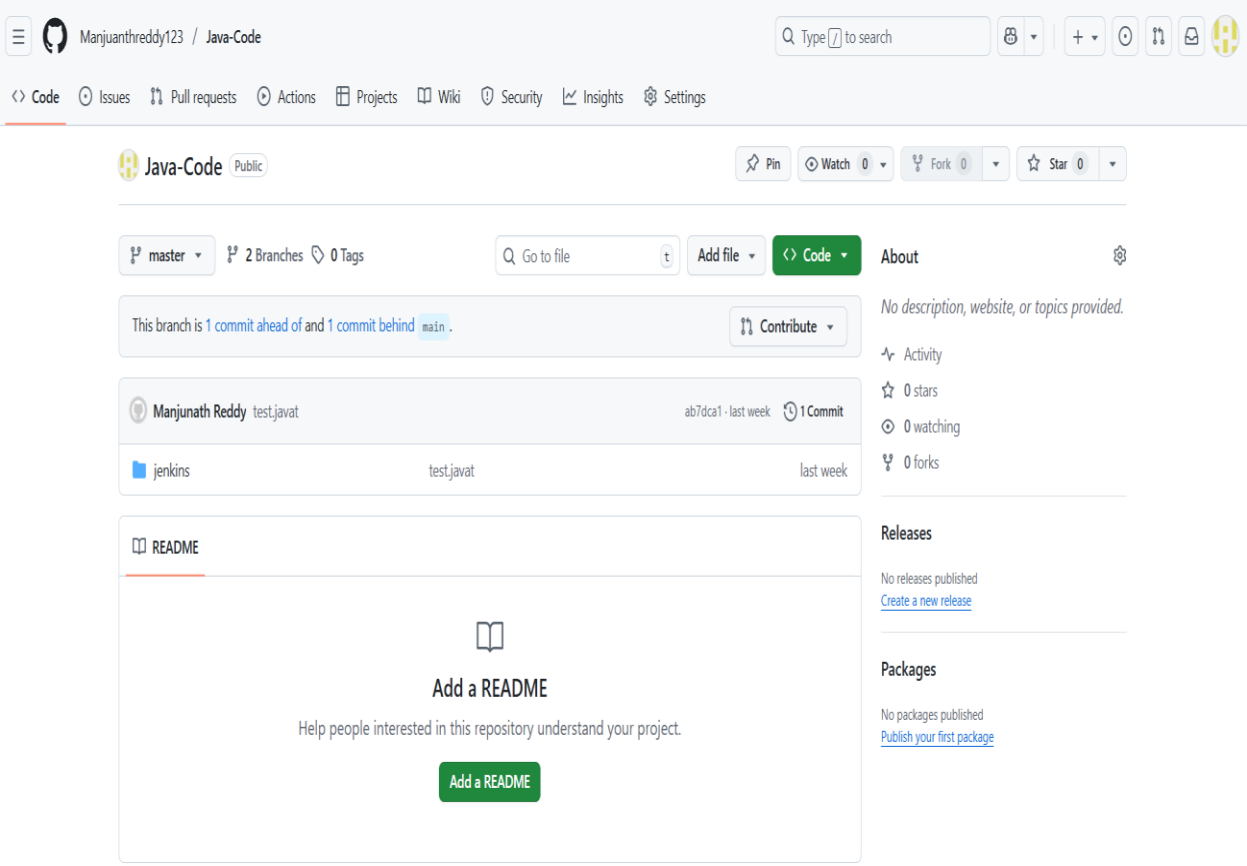
Building a Java application using Jenkins and GitHub involves setting up a continuous integration (CI) pipeline that fetches the source code from a GitHub repository, builds the application, runs tests, and possibly deploys it. Here are the general steps to achieve this:

**Step1:** Build a java application using any IDE and push the code git repository

Create New folder → open the new folder in VS code

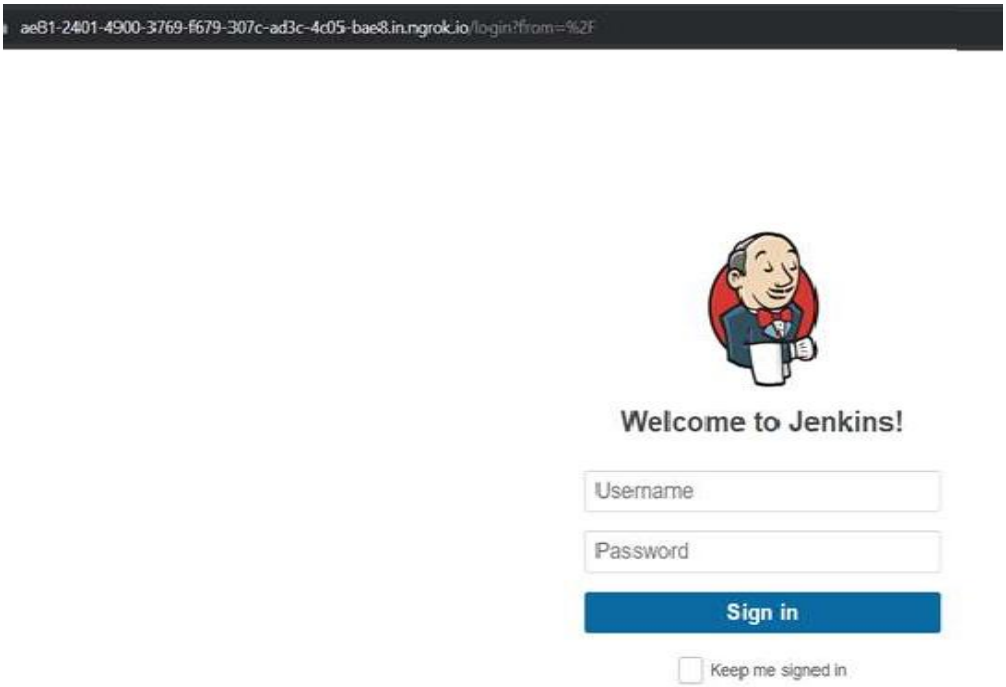
Create a java file in new folder by name test.java

Write the code and push the code to git hub repository

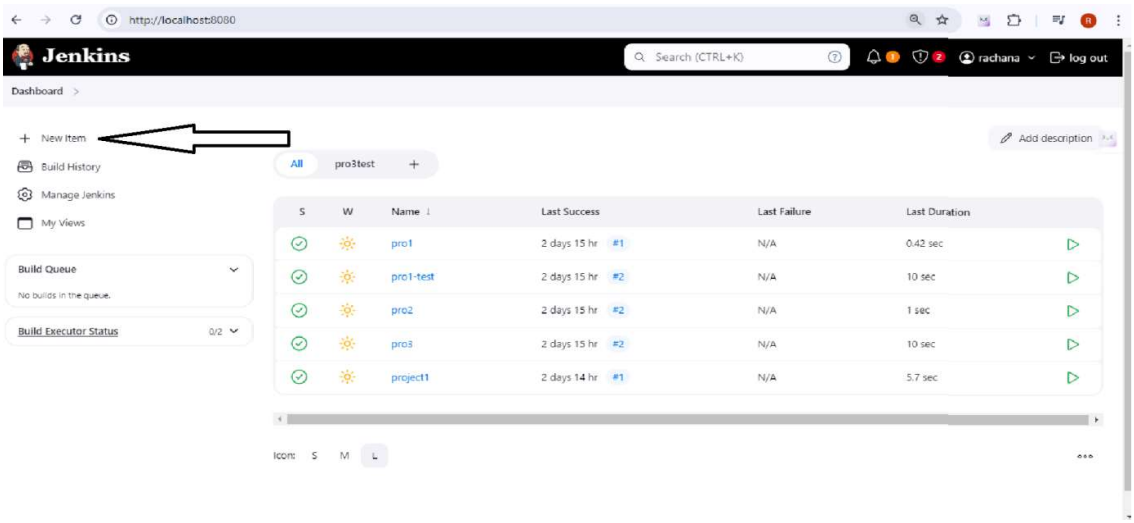


**Step2:** open Jenkins (loacalhost:<port number>)

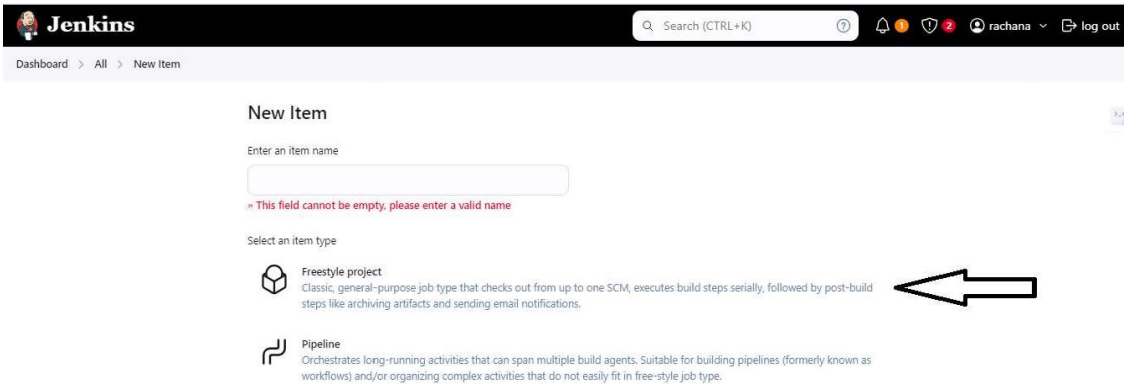
Login to jenkins



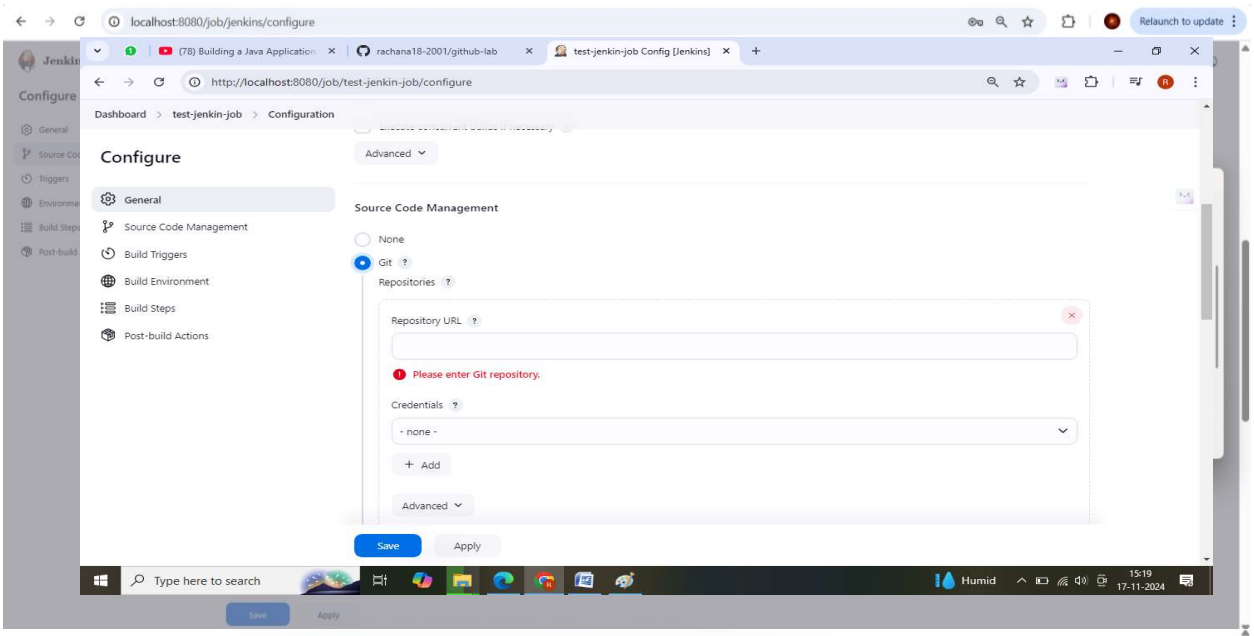

Create new Jenkins job

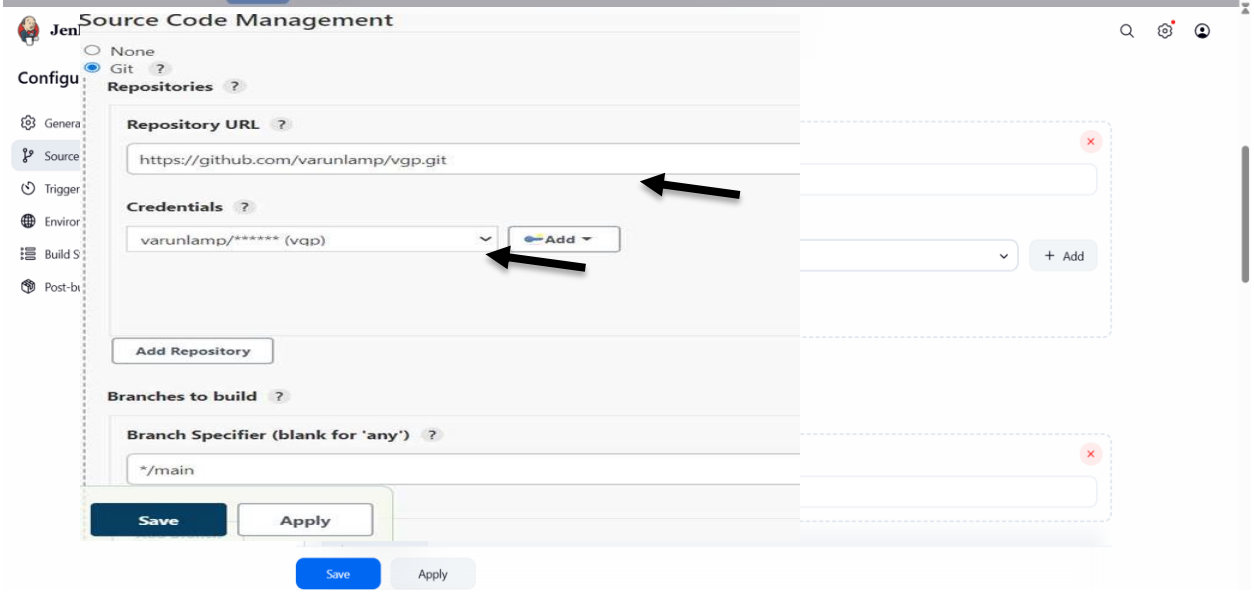


**Step3:** Enter the name of new item and Select Free Style project




**Step4:** Click on git and add git repository URL that has java application.

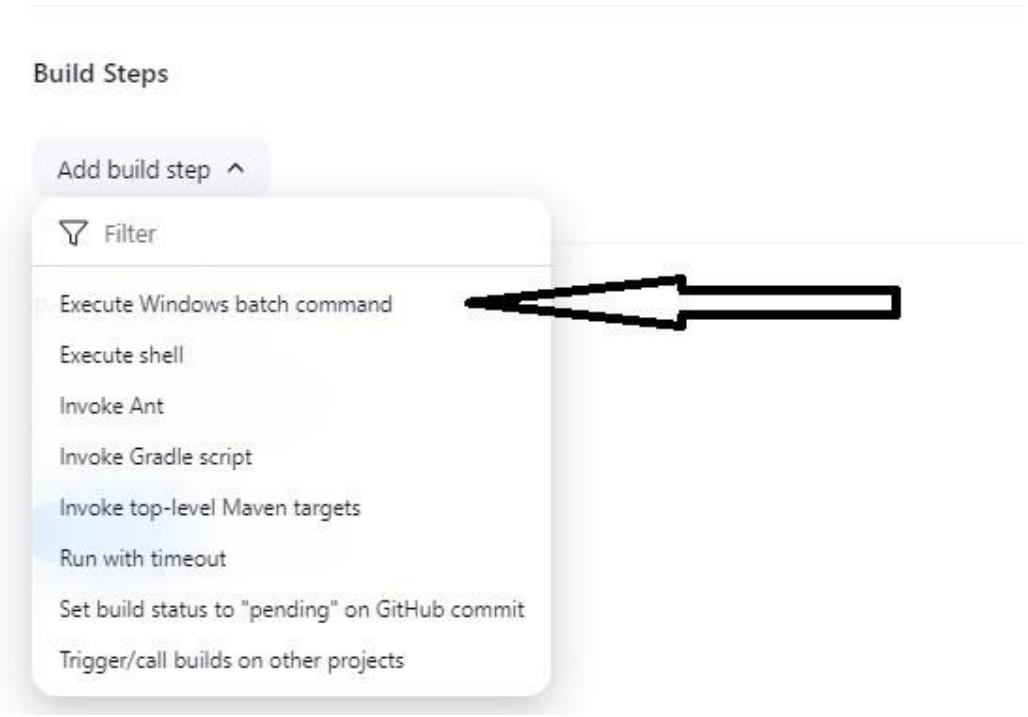


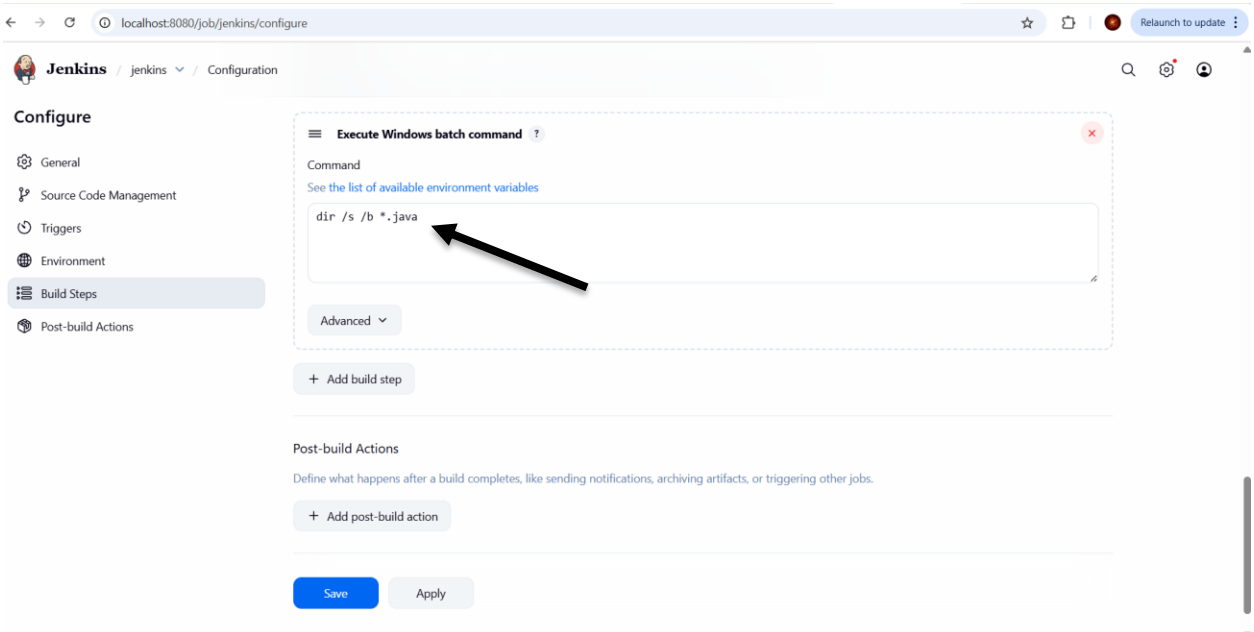
**Step5:** Now scroll down and select Add bulid step → select **Execute Windows batch command**

**Write command: javac test.java**

**Java test**

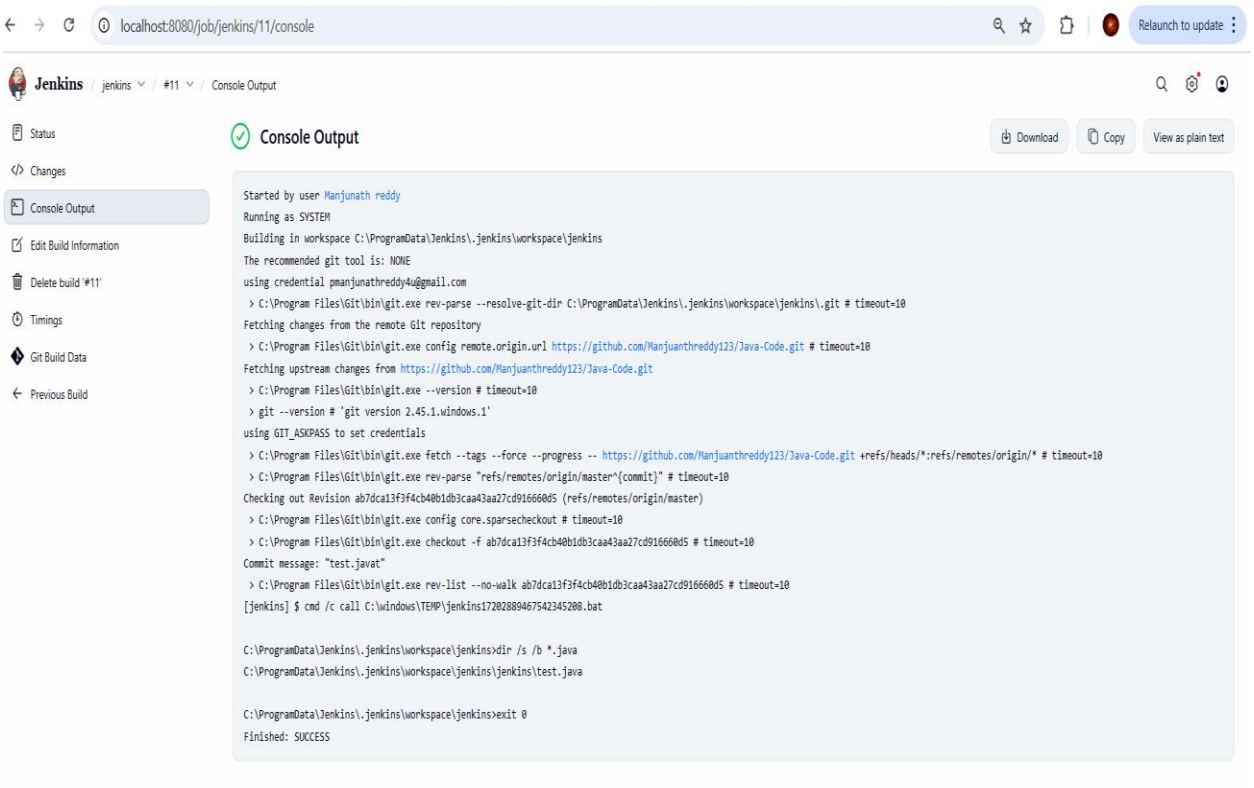



**Step6:**



**Step7:** click on Apply → save → Build Now

**Step8:** see the final output in the console

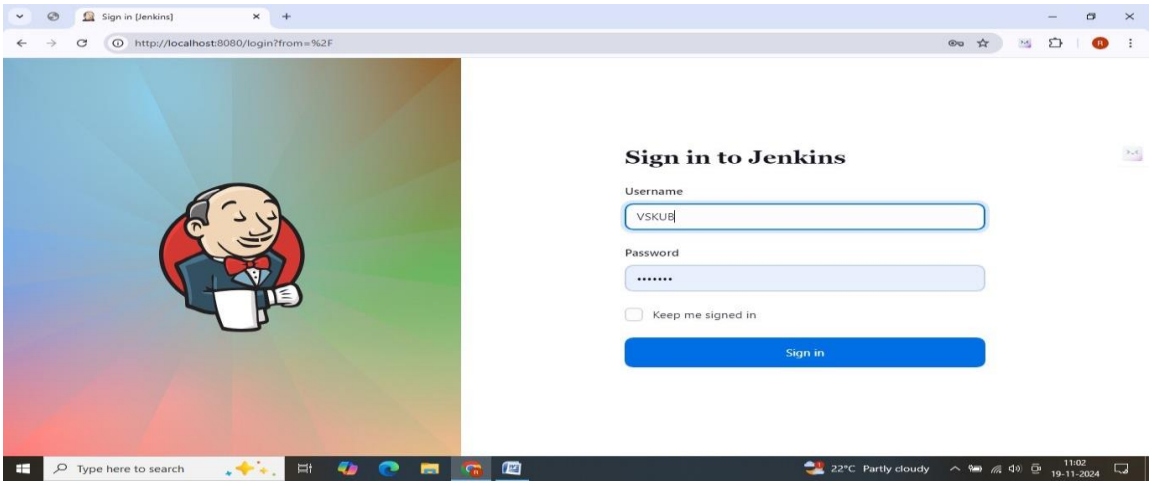


# Experiment no: 8

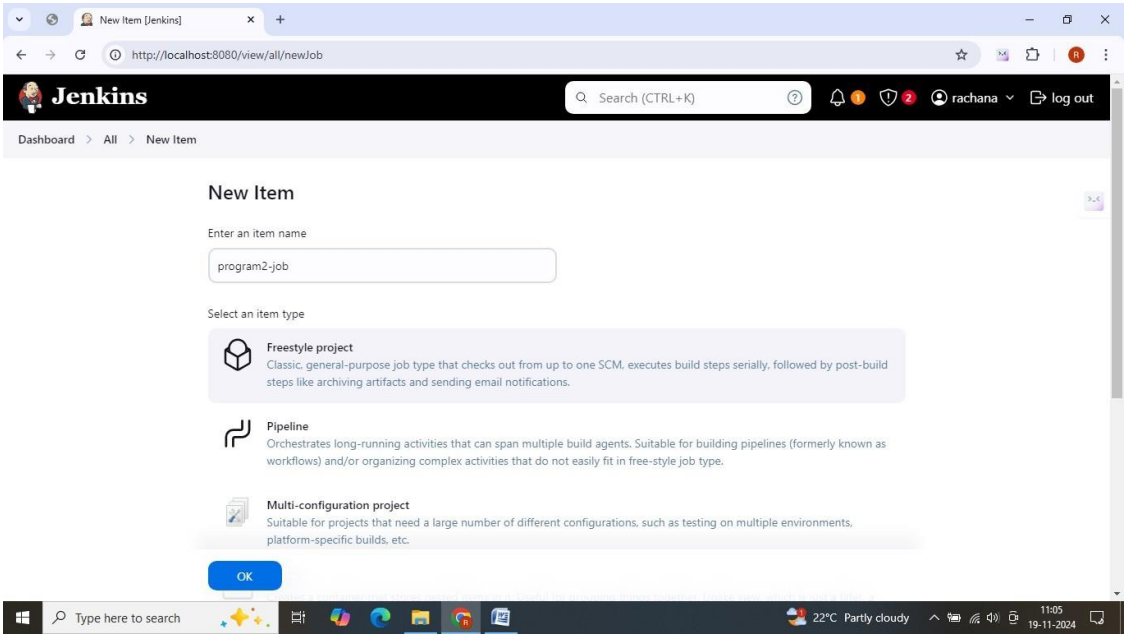
## Implement jenkin program using freestyle and pipeline

### Implementing Jenkins using free style project

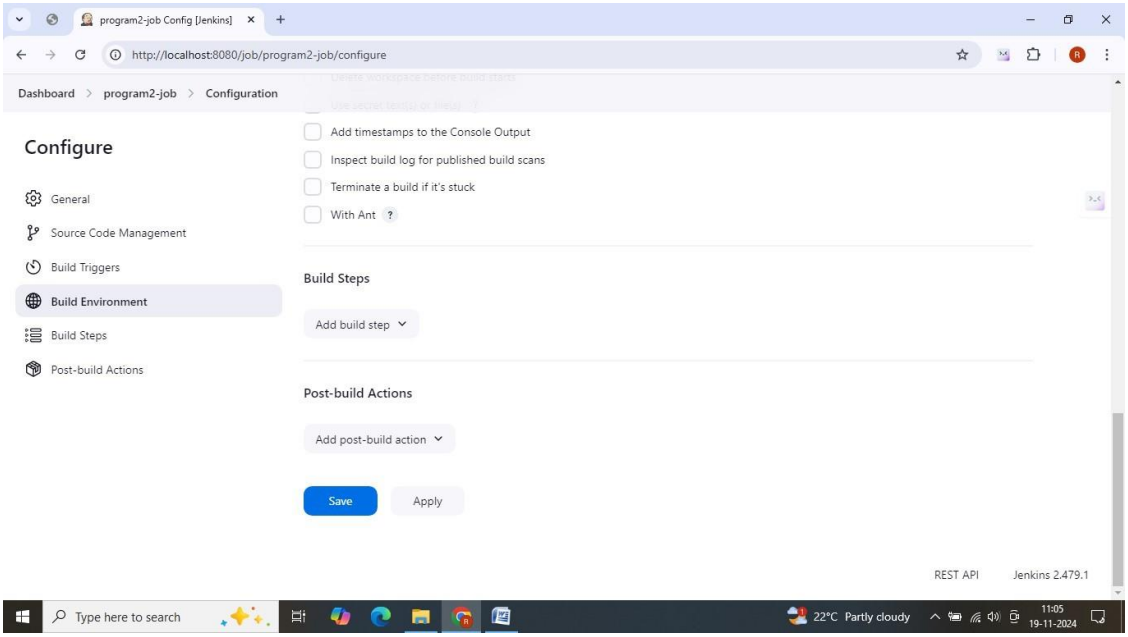
Login to jenkins



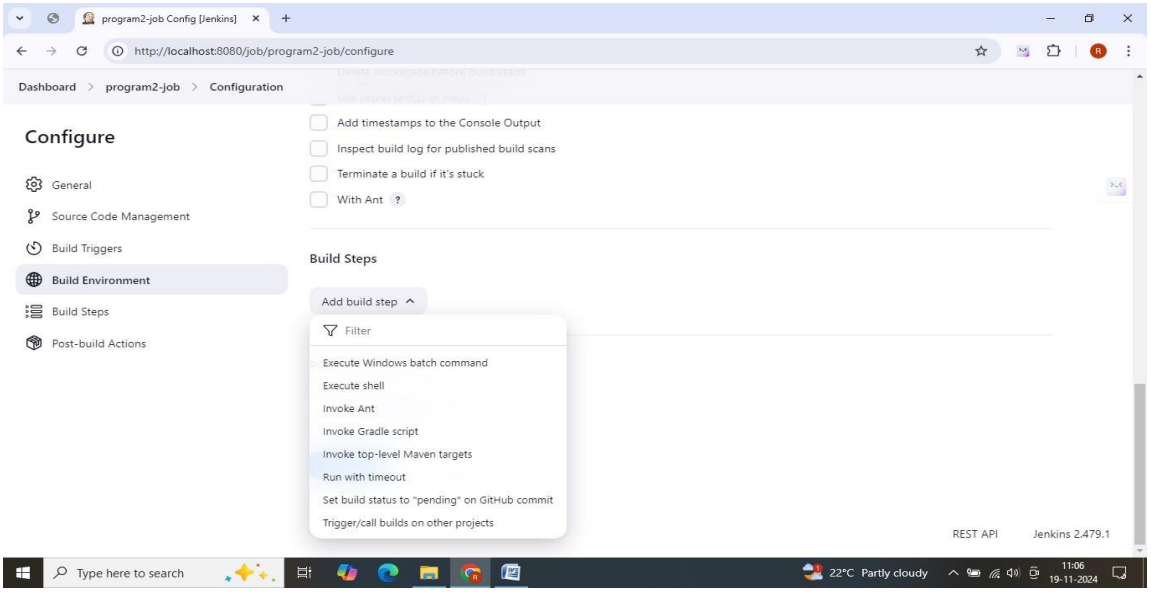

Enter new item name and select free style project



Select build steps

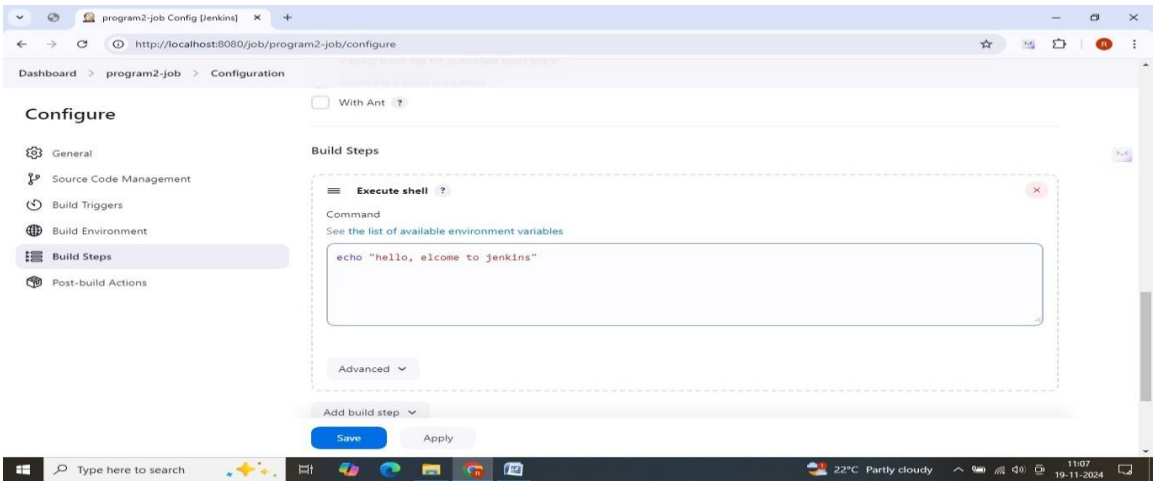



Click on execute shell

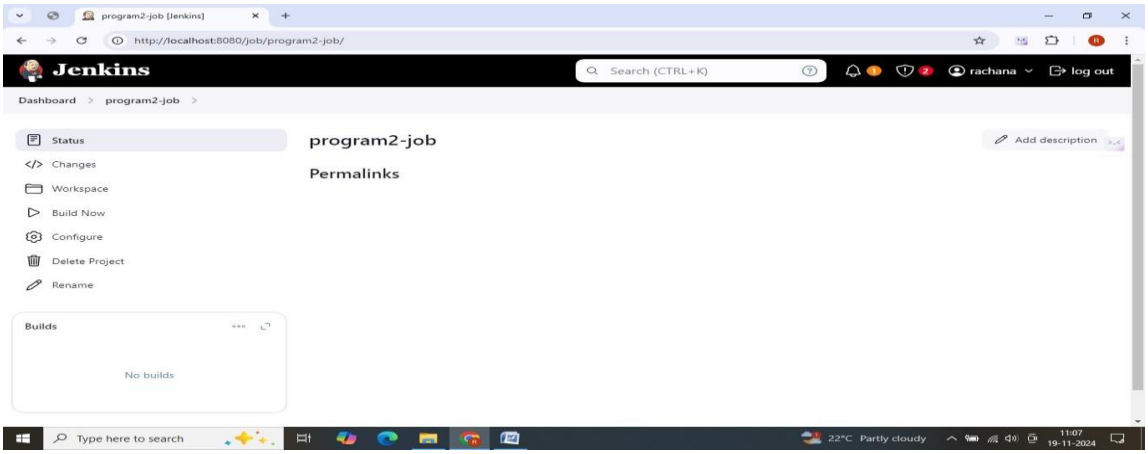



Write command

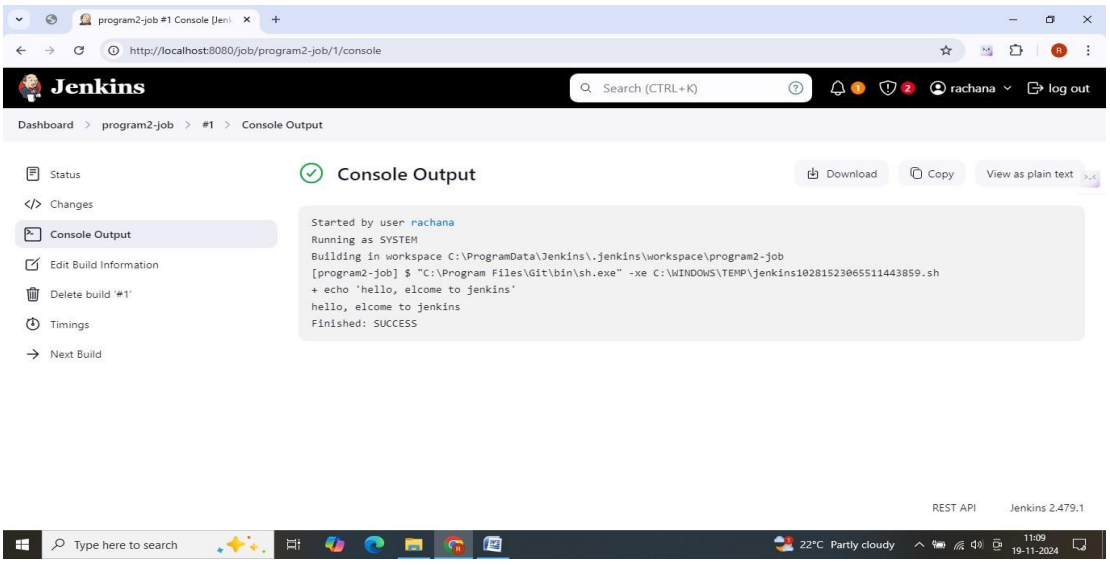
Save and apply




Click on bulid now and see the output



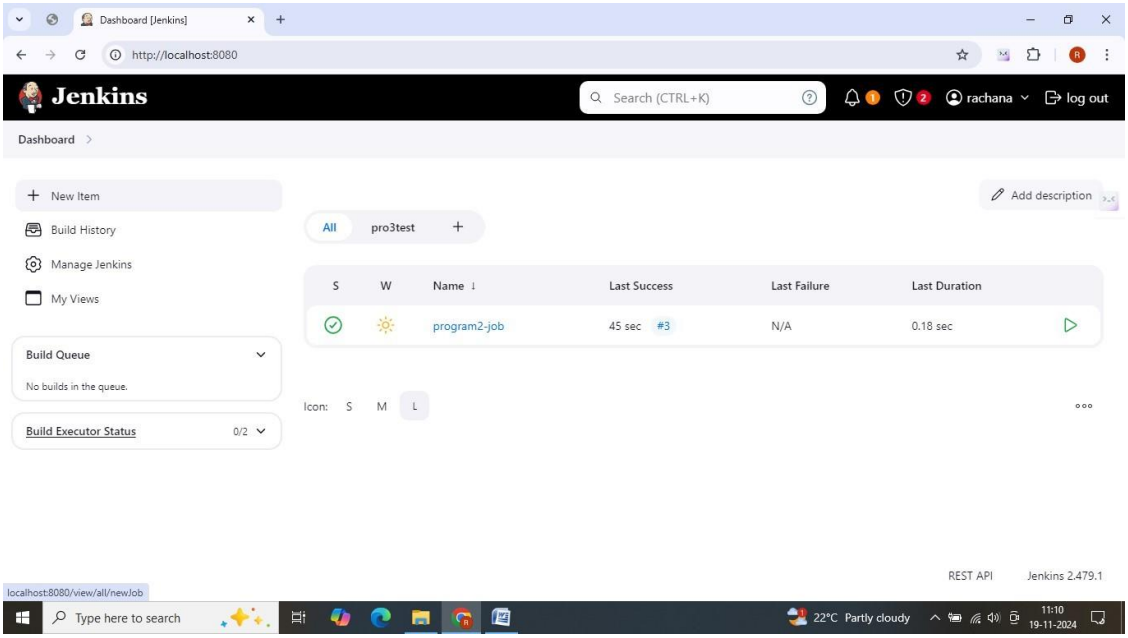

OUTPUT:



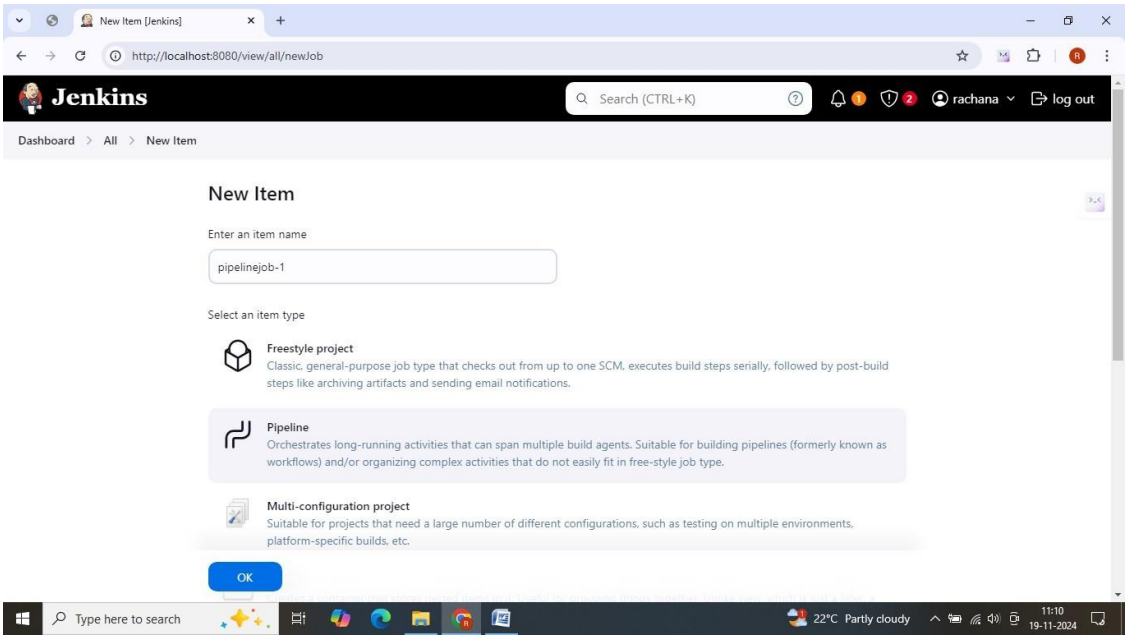
**Implemeting Jenkin program using pipeline**

Now create new pipeline job

Click on new item



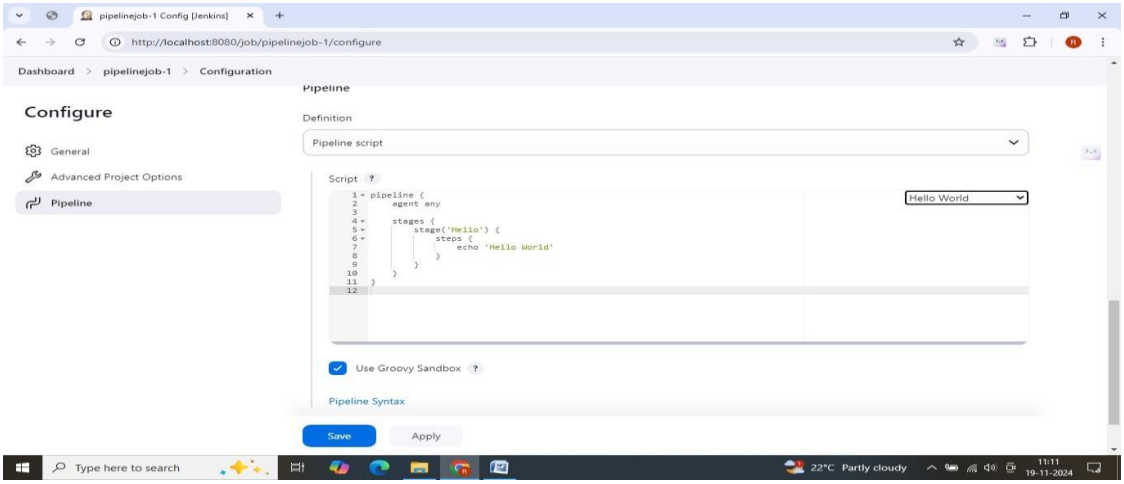

Name the job and click on pipeline



**Write descriptive code**

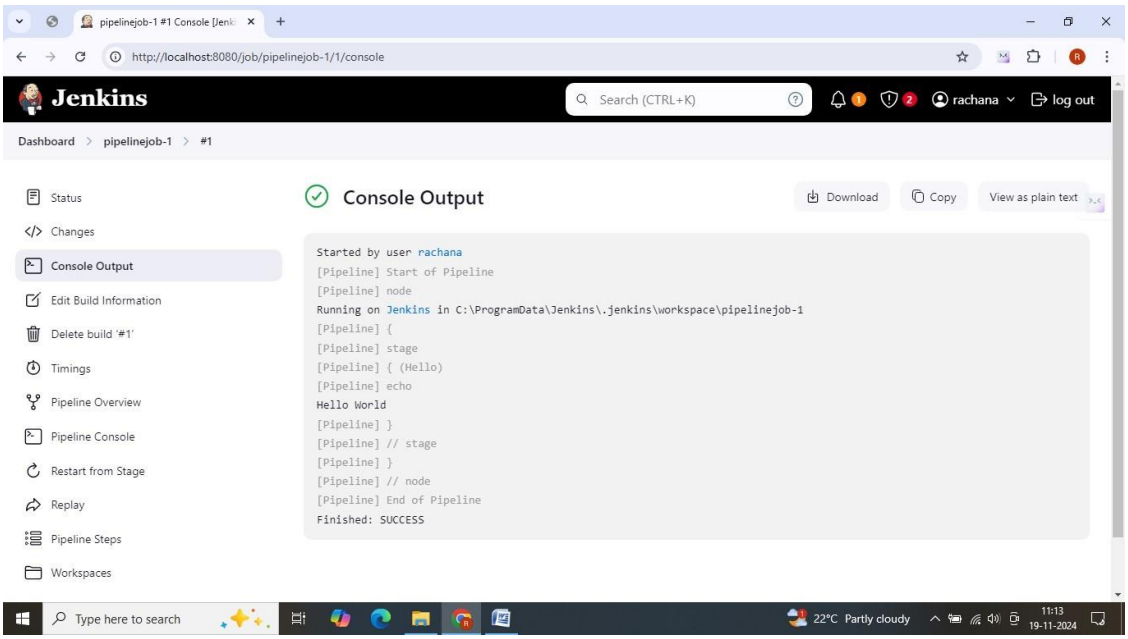
```
pipeline {
    agent any
    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

Click on apply and save




Click on build now

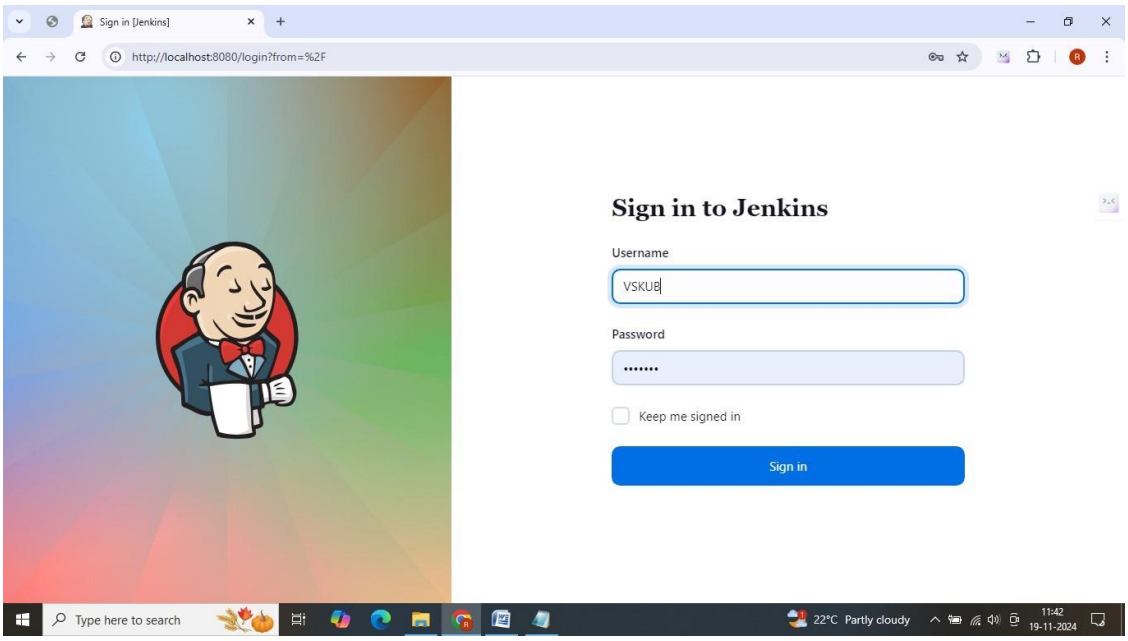
And see the output


pipelinejob-1 #1 Console [Jenk
http://localhost:8080/job/pipelinejob-1/1/console
Jenkins
Search (CTRL+K)
rachana
log out
Dashboard
pipelinejob-1
#1
Status
Changes
Console Output
Edit Build Information
Delete build '#1'
Timings
Pipeline Overview
Pipeline Console
Restart from Stage
Replay
Pipeline Steps
Workspaces
Console Output
Download
Copy
View as plain text
Started by user rachana
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\pipelinejob-1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
Type here to search
22°C Partly cloudy
11:13
19-11-2024

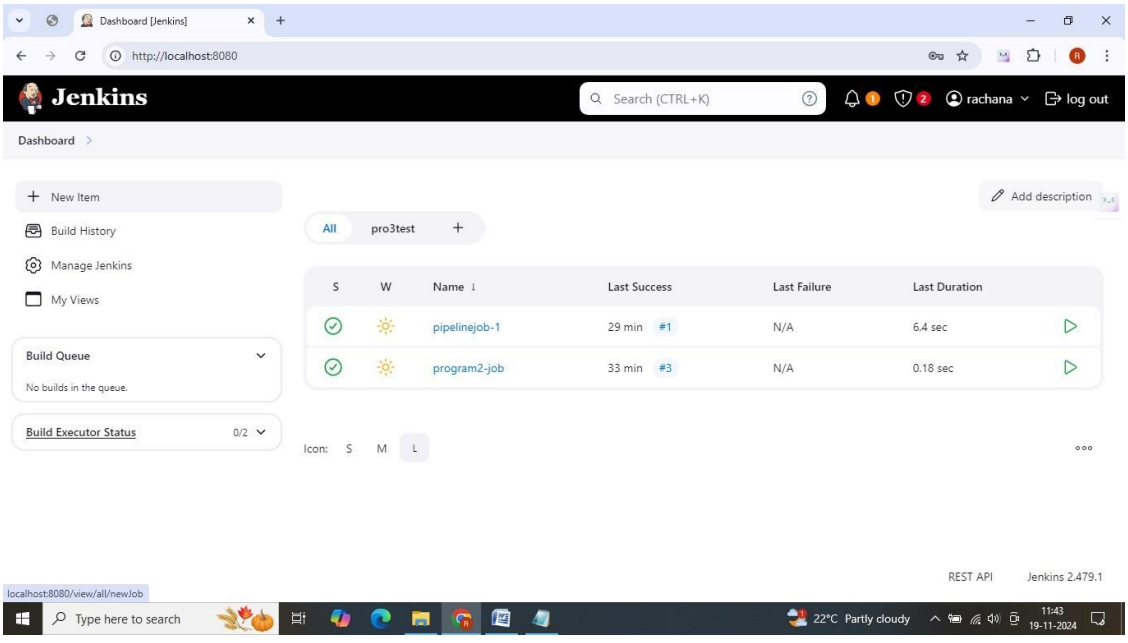
# Experiment 9

## Create first Jenkins Pipeline using Build Pipeline

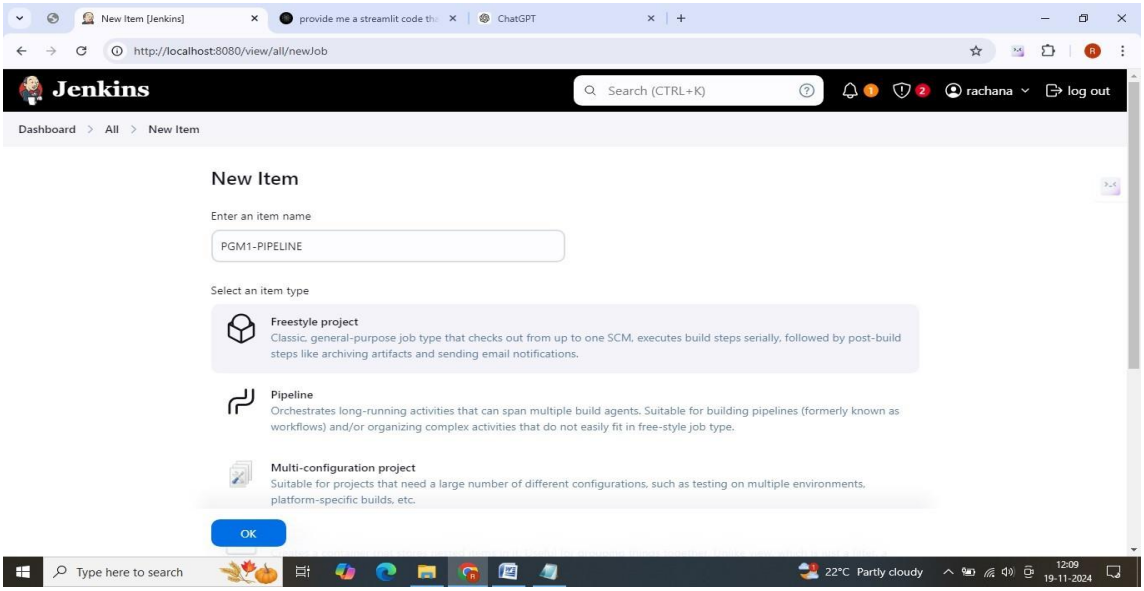
Login to jenkins



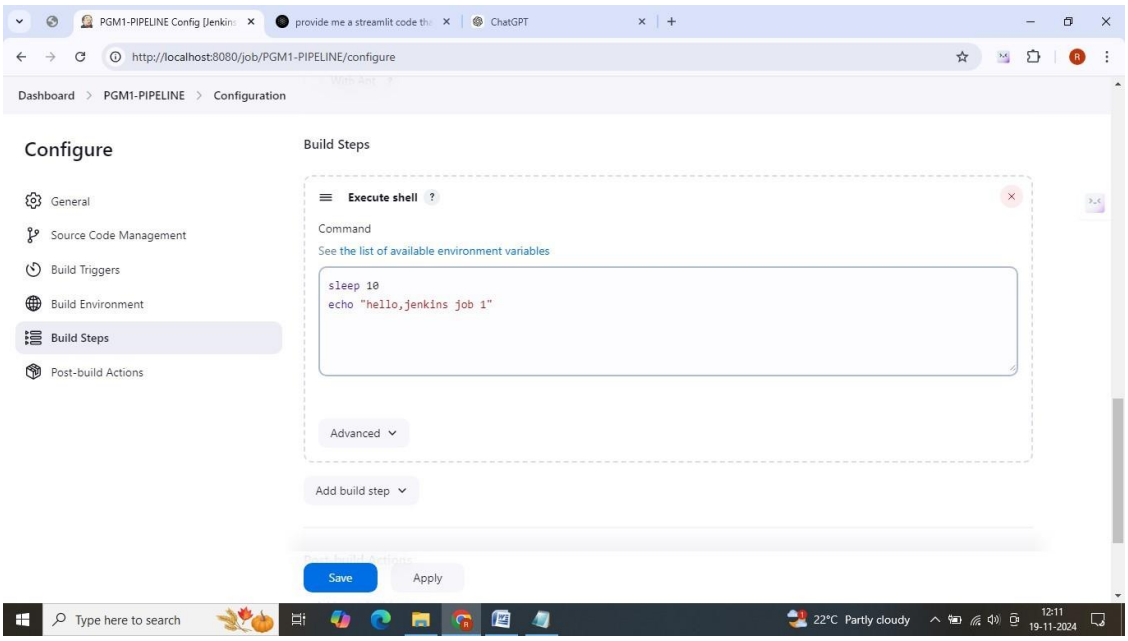

Create new job



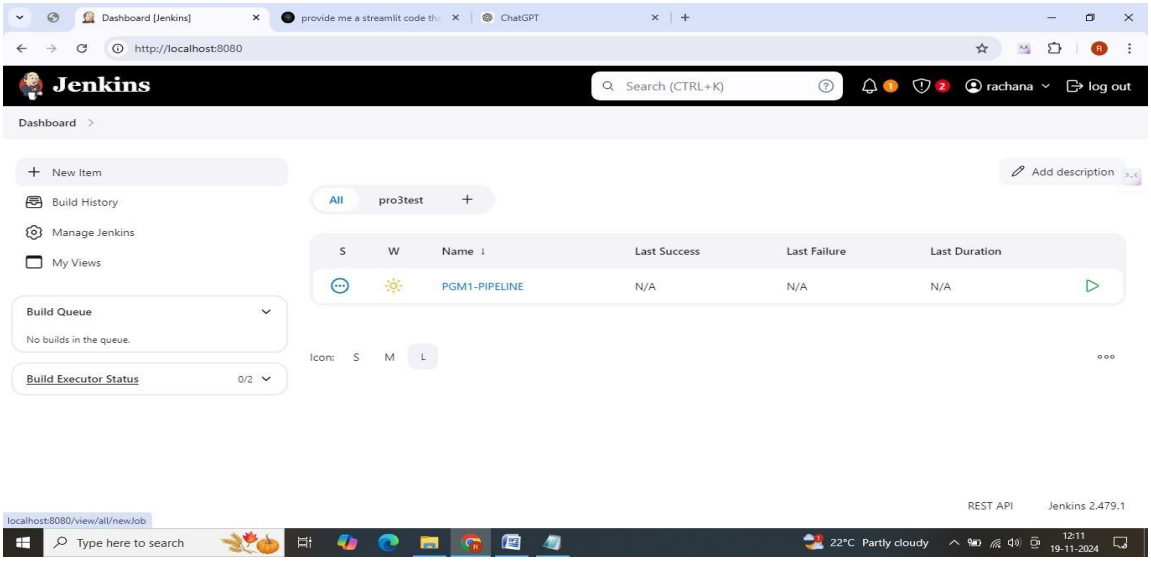
Write the job name and click on free style project



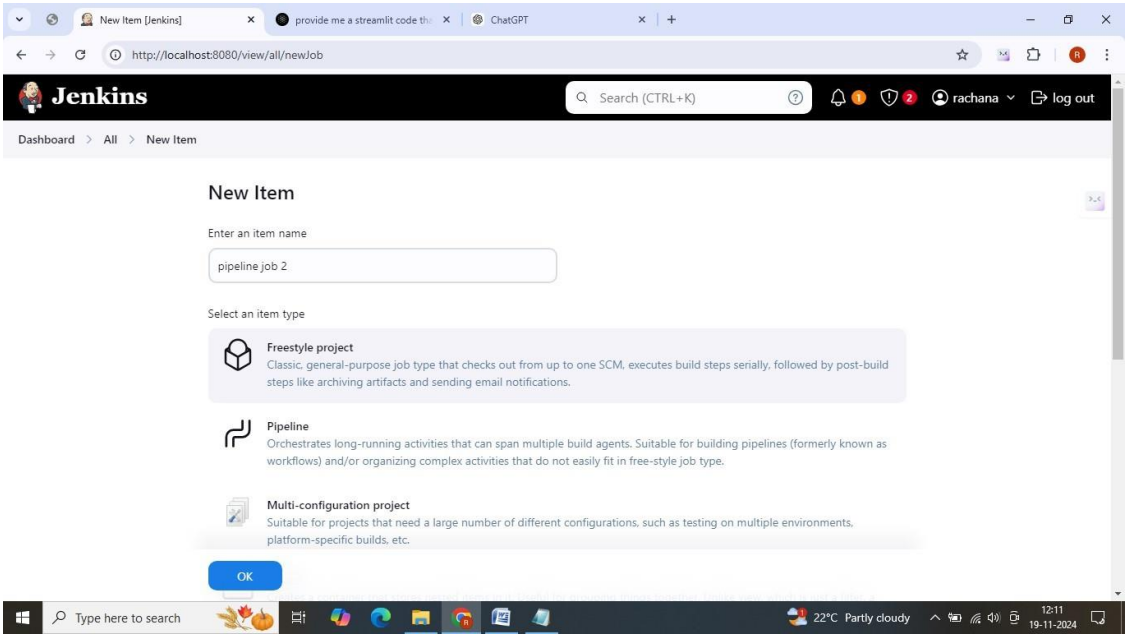

Click on build step →select execute shell → write the shell script→save and apply



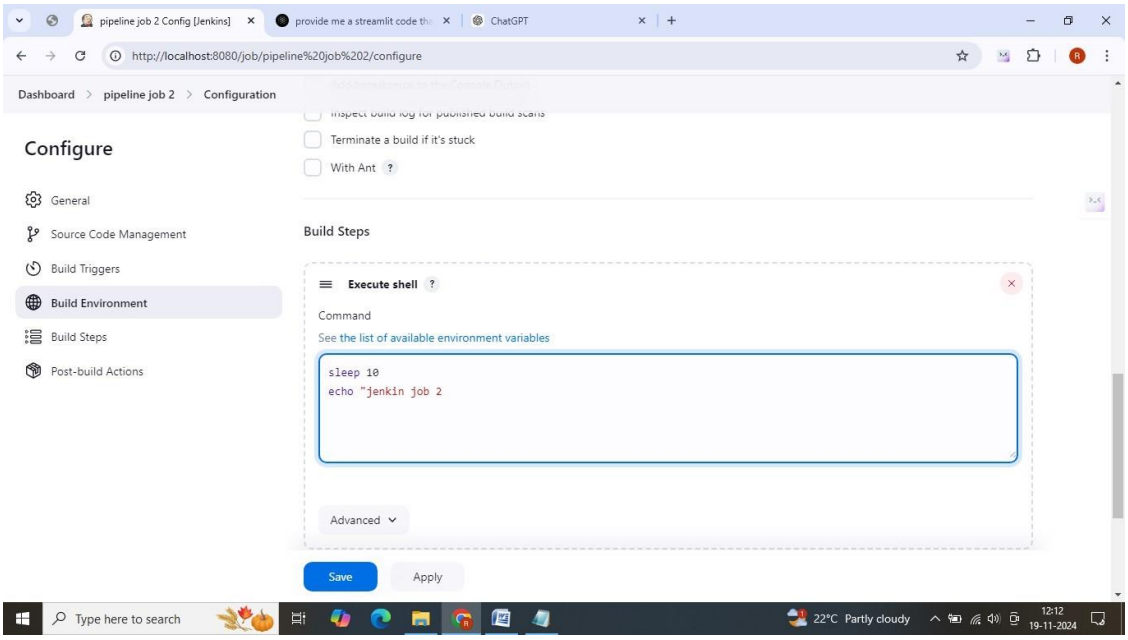
Create another new item



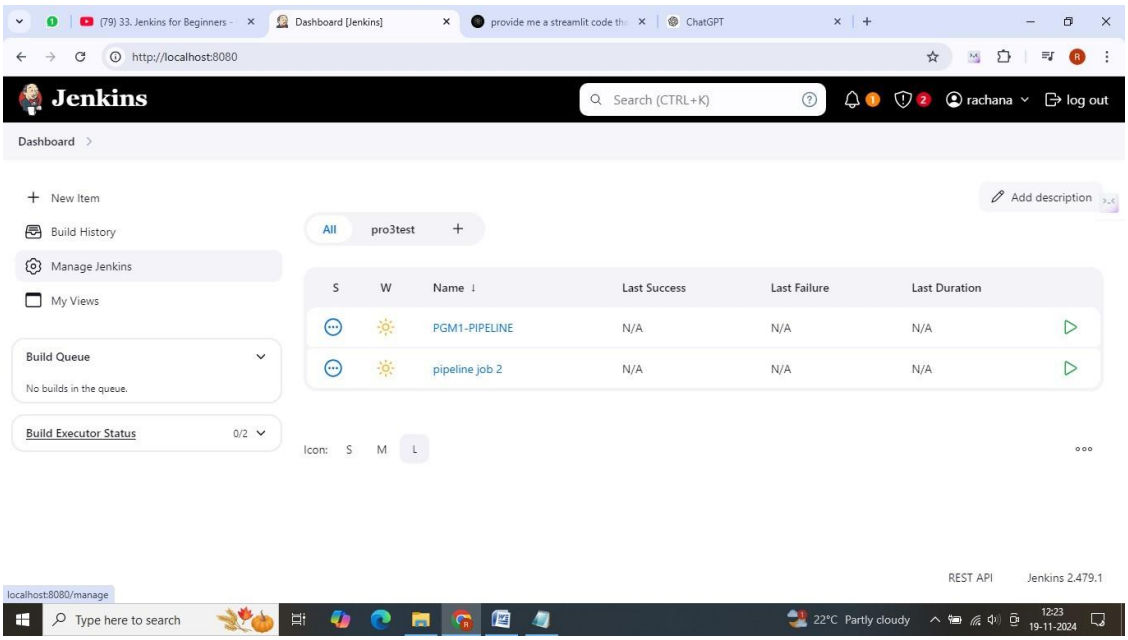

Name the new item and select free style project



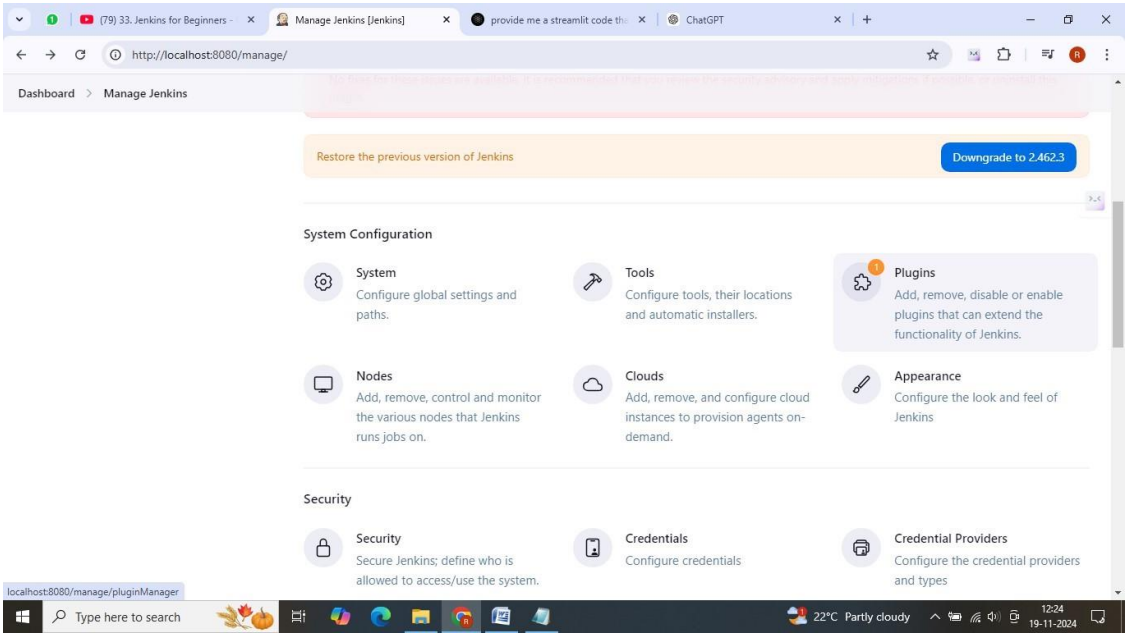
Click on build step →select execute shell → write the shell script→save and apply



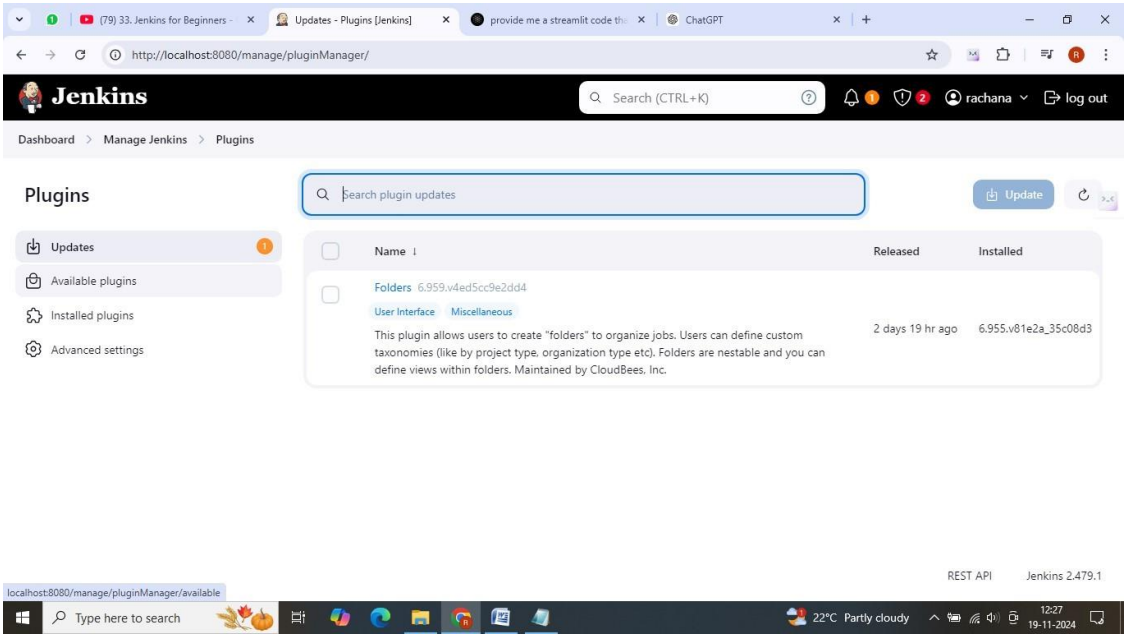

Click on manage jenkins



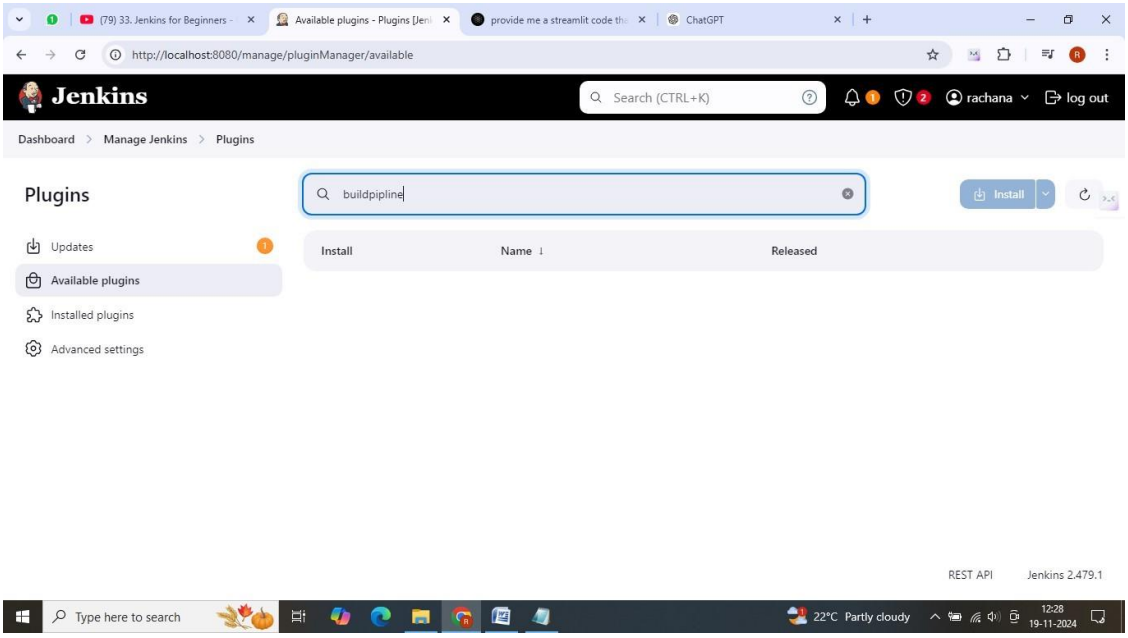
Click on plugins



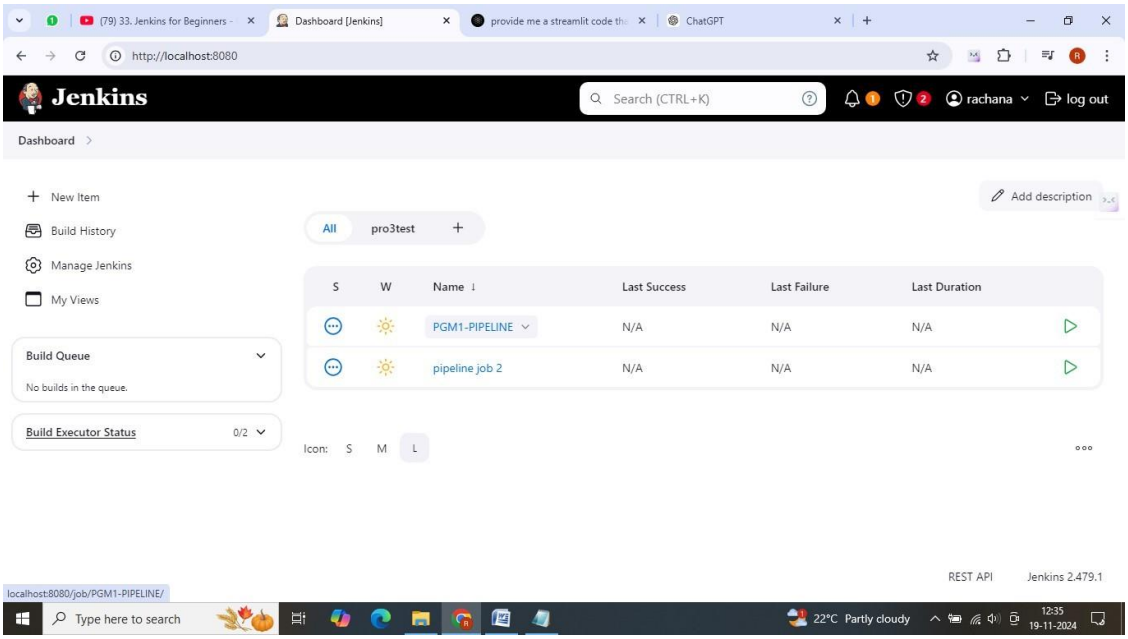

Click on available plugins



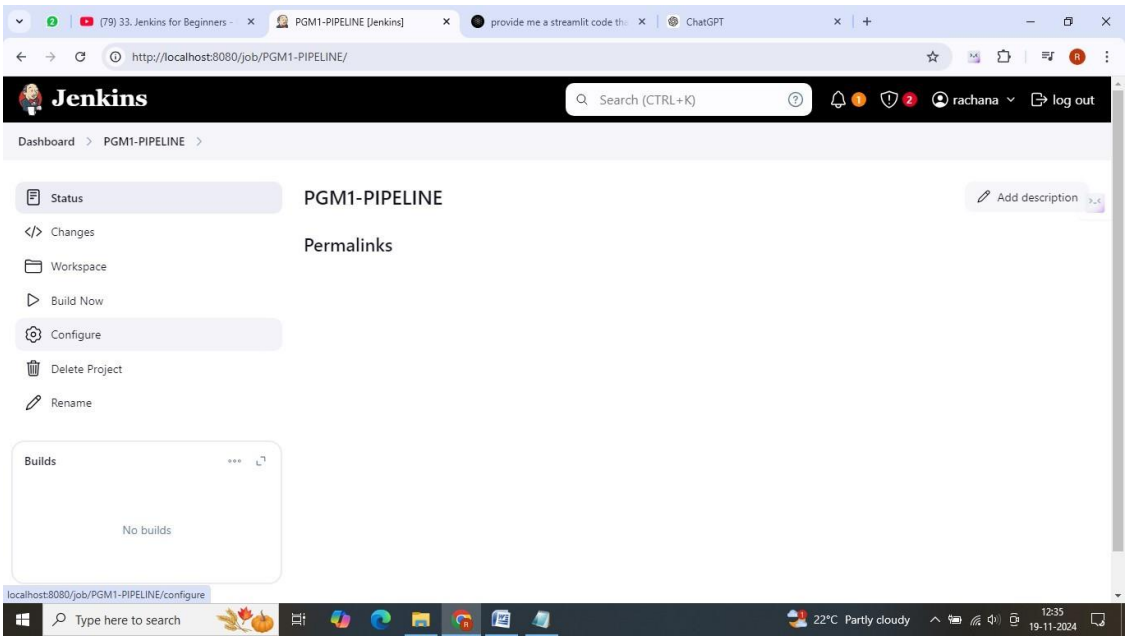
Search for bulid pipeline and install



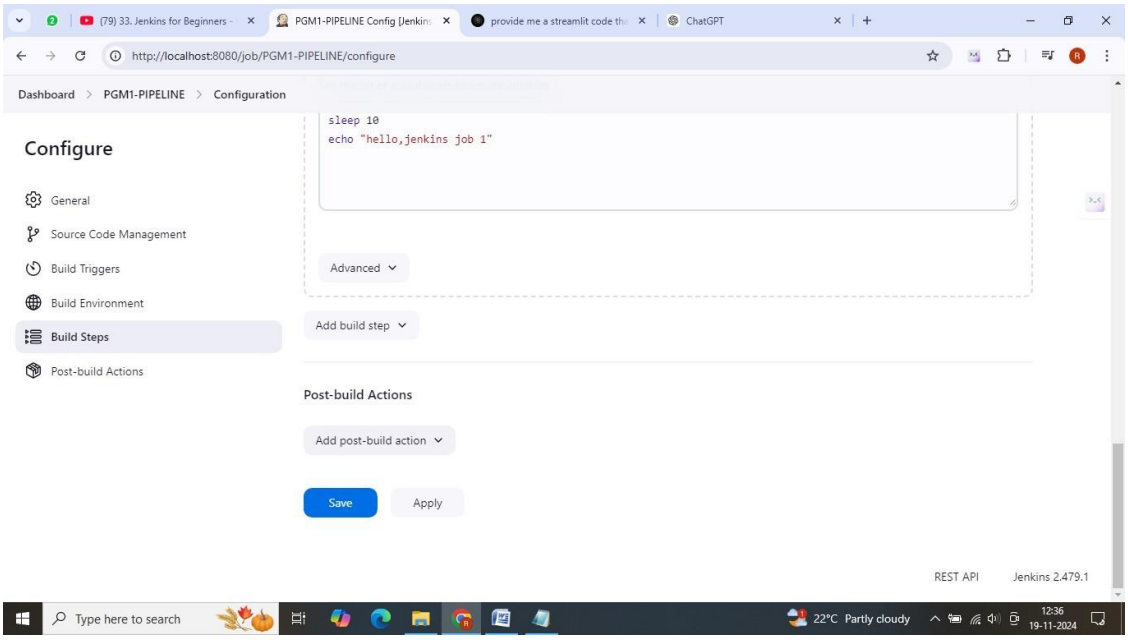

Click on first job created



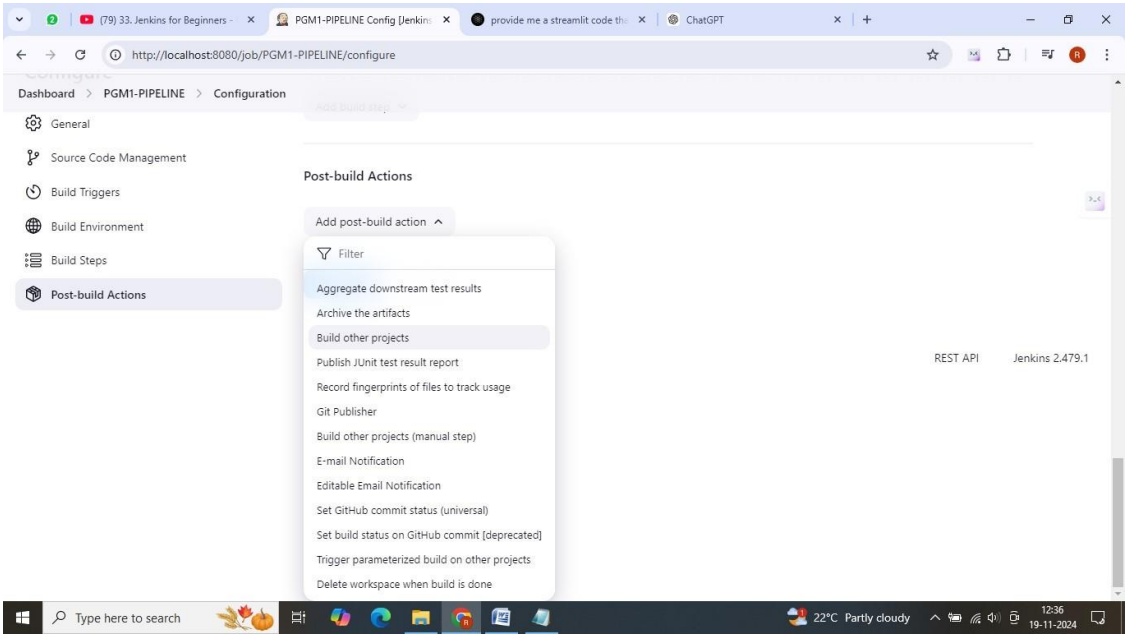
Click on configure

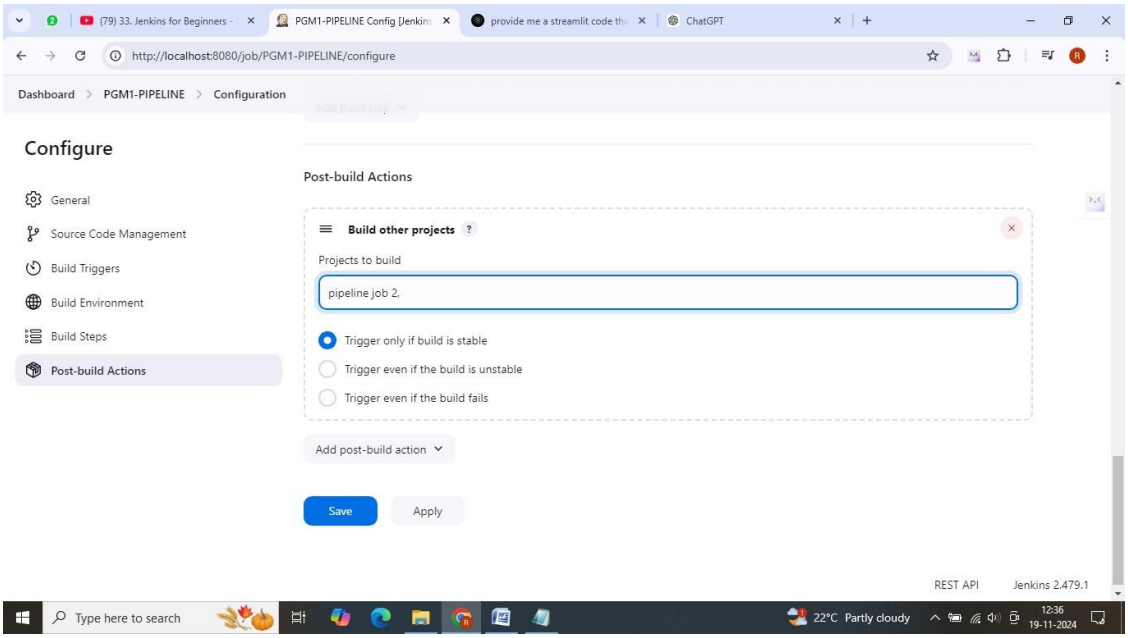



Click on post build actions

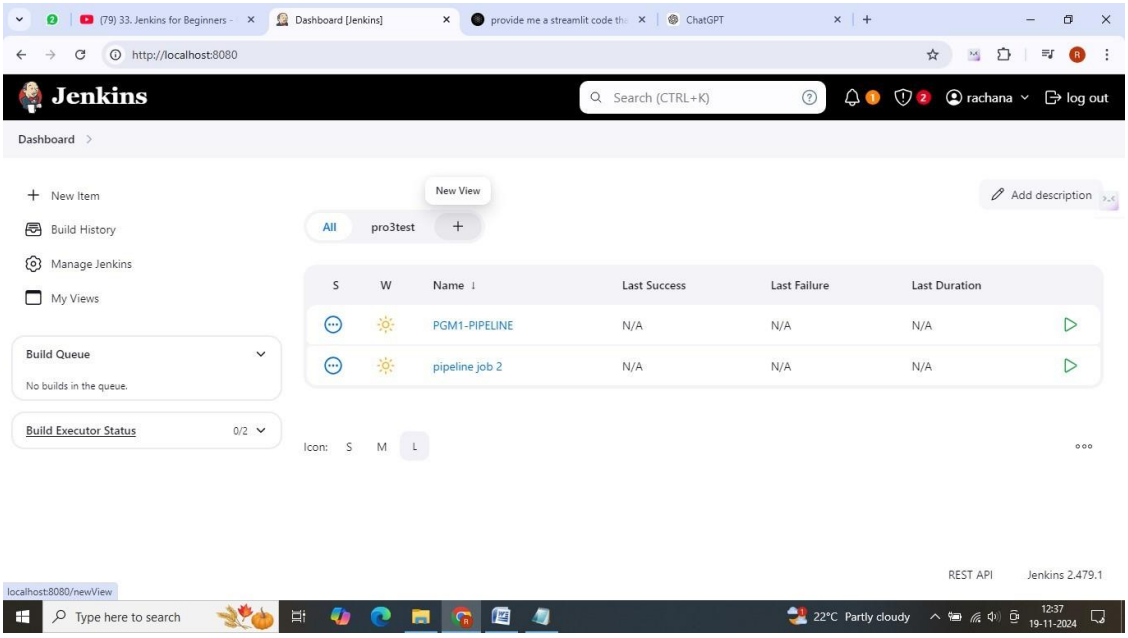


Select build other projects



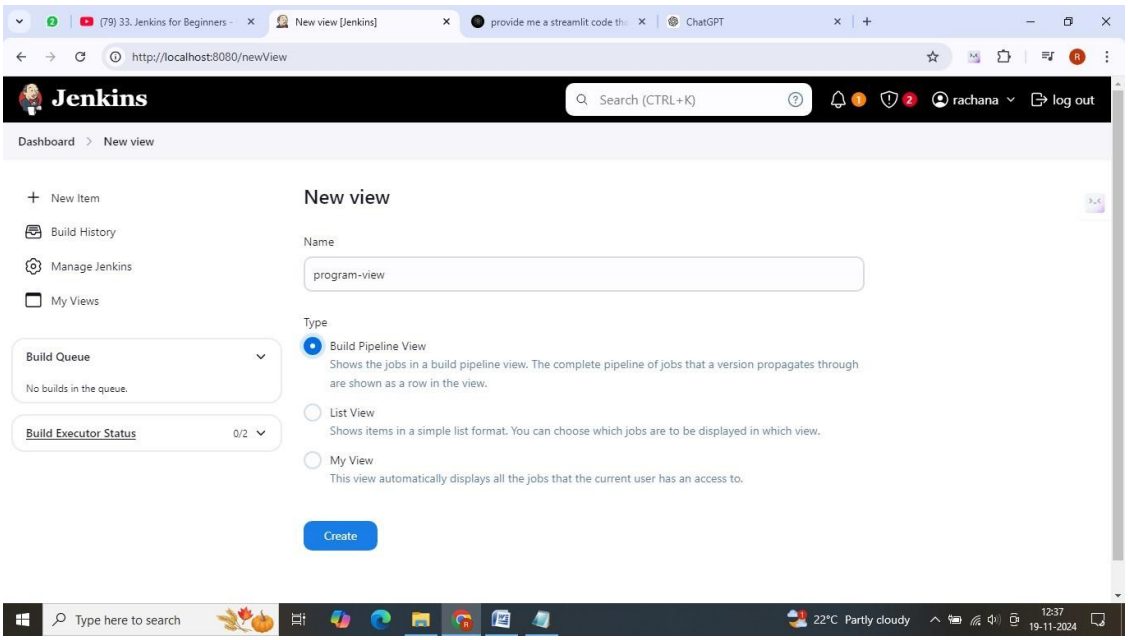



Click on plus button



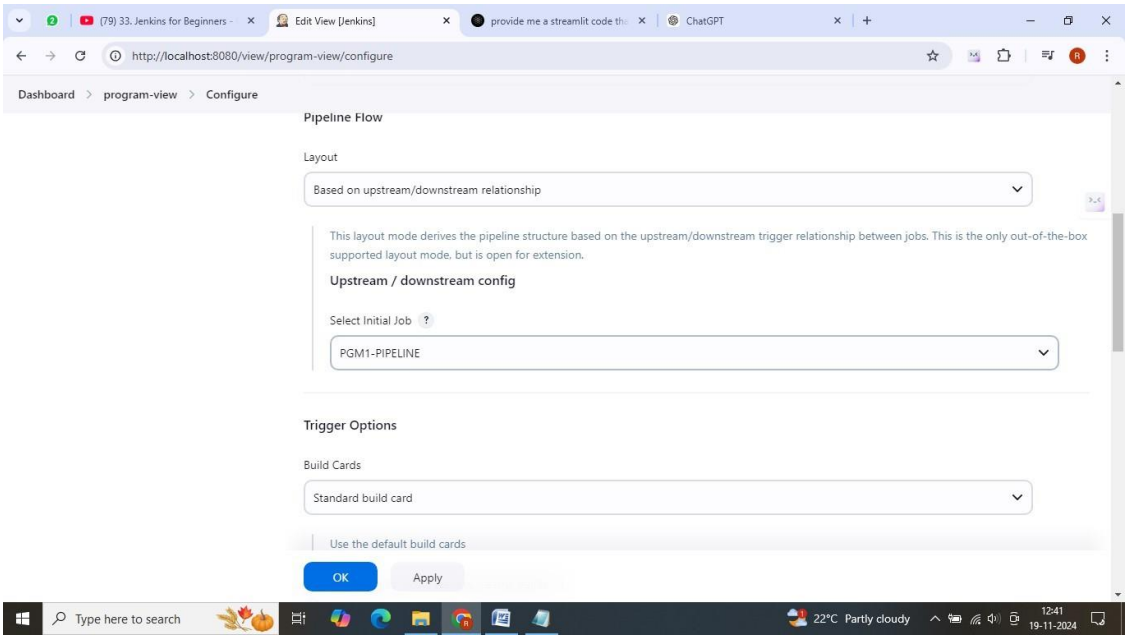

Provide name

Bulid pipeline view

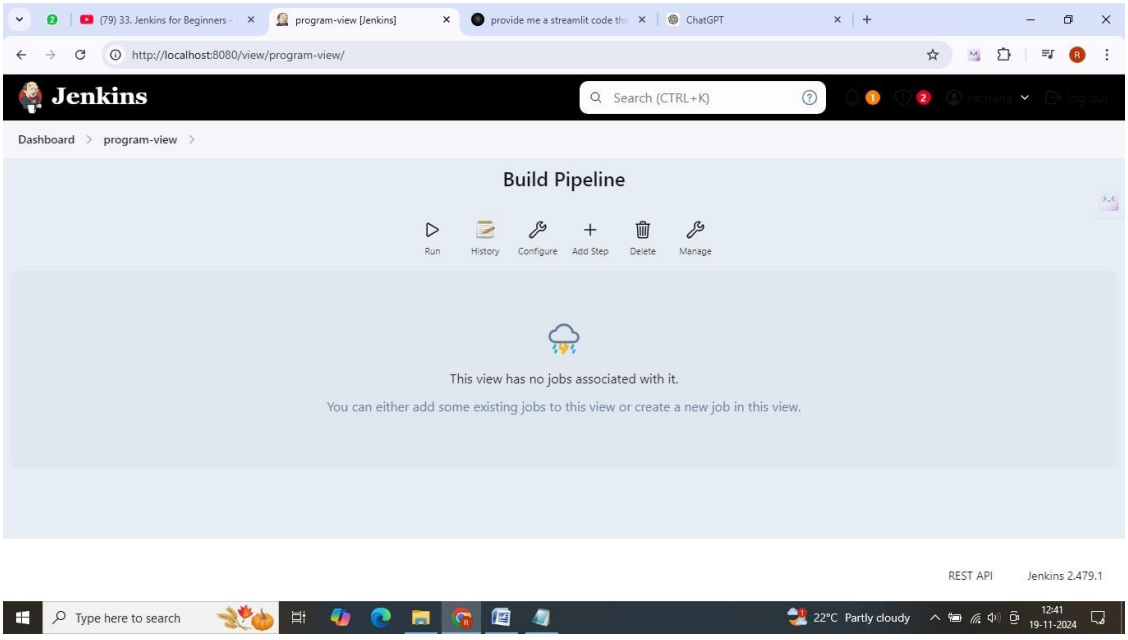


Select initial job

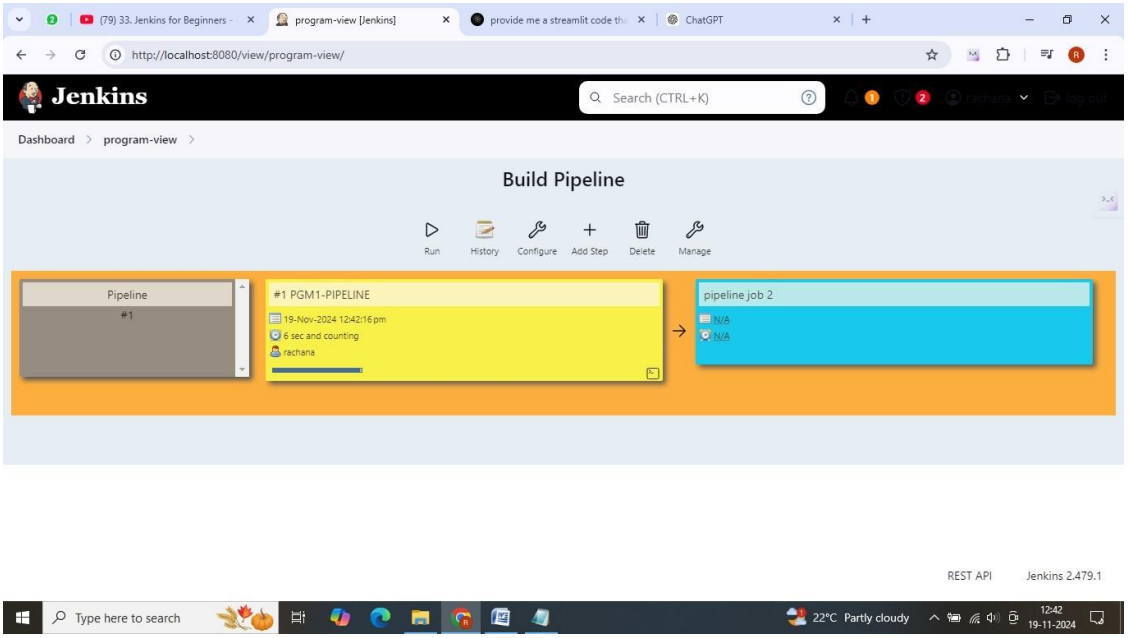
Click apply and save

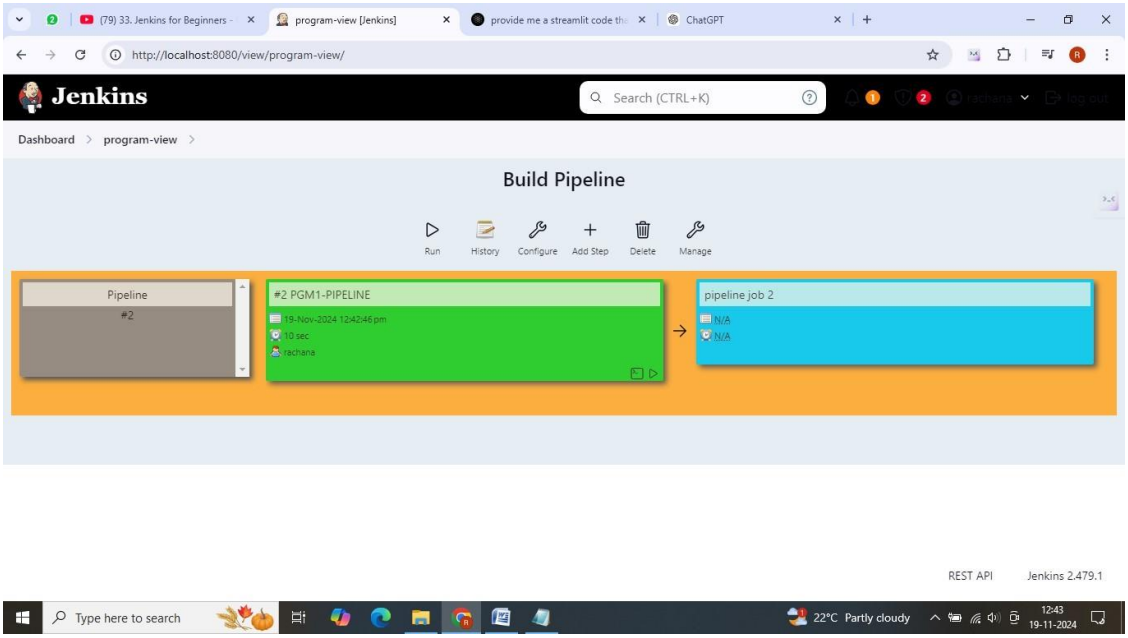



Click on RUN



OUTPUT:


(79) 33. Jenkins for Beginners -
program-view [Jenkins]
provide me a streamlit code th
ChatGPT
http://localhost:8080/view/program-view/
Jenkins
Search (CTRL+K)
Dashboard
program-view
Build Pipeline
Run
History
Configure
Add Step
Delete
Manage
Pipeline
#1
#1 PGM1-PIPELINE
19-Nov-2024 12:42:16 pm
6 sec and counting
rachana
pipeline job 2
N/A
N/A
REST API
Jenkins 2.479.1
Type here to search
22°C Partly cloudy
12:42
19-11-2024


(79) 33. Jenkins for Beginners -
program-view [Jenkins]
provide me a streamlit code th
ChatGPT
http://localhost:8080/view/program-view/
Jenkins
Search (CTRL+K)
Dashboard
program-view
Build Pipeline
Run
History
Configure
Add Step
Delete
Manage
Pipeline
#2
#2 PGM1-PIPELINE
19-Nov-2024 12:42:46 pm
10 sec
rachana
pipeline job 2
N/A
N/A
REST API
Jenkins 2.479.1
Type here to search
22°C Partly cloudy
12:43
19-11-2024