

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 our computer. To do this, we can download the latest version on the official website. We can download for our 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 the current version installed on our PC. The next thing we'll need to do is to set our username and email address. Git will use this information to identify who made specific changes to files.

To set our 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 terminal 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

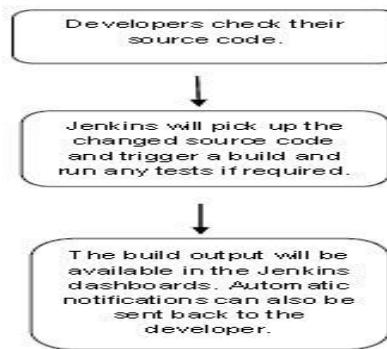
git branch -vv

- 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.



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

Downloading Jenkins

Jenkins is distributed as WAR files, native packages, installers, and Docker images. Follow these installation steps:

1. Before downloading, please take a moment to review the [Hardware and Software requirements](#) section of the User Handbook.
2. Select one of the packages below and follow the download instructions.
3. Once a Jenkins package has been downloaded, proceed to the [Installing Jenkins](#) section of the User Handbook.
4. You may also want to verify the package you downloaded. [Learn more about verifying Jenkins downloads.](#)

Download Jenkins 2.479.1 LTS for:	Download Jenkins 2.485 for:
<input checked="" type="checkbox"/> Generic Java package (.war) <small>SHA-256: cbf987b1aaab1fcex20e9411b3082fa32je3724c1bd64562dd64c1e4d4f50b470</small>	<input checked="" type="checkbox"/> Generic Java package (.war) <small>SHA-256: bcf8ecdf5e7ceffed42cc9Qah3672d631Qbf9a7a6078d476e37d9efcf4b5e90</small>
<input checked="" type="checkbox"/> Docker	<input checked="" type="checkbox"/> Docker
<input checked="" type="checkbox"/> Kubernetes	<input checked="" type="checkbox"/> Ubuntu/Debian
<input checked="" type="checkbox"/> Ubuntu/Debian	<input checked="" type="checkbox"/> Red Hat/Fedora/Alma/Rocky/CentOS
<input checked="" type="checkbox"/> Red Hat/Fedora/Alma/Rocky/CentOS	<input checked="" type="checkbox"/> Windows
<input checked="" type="checkbox"/> Windows	<input checked="" type="checkbox"/> openSUSE
<input checked="" type="checkbox"/> openSUSE	<input checked="" type="checkbox"/> Arch Linux
Third party	

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

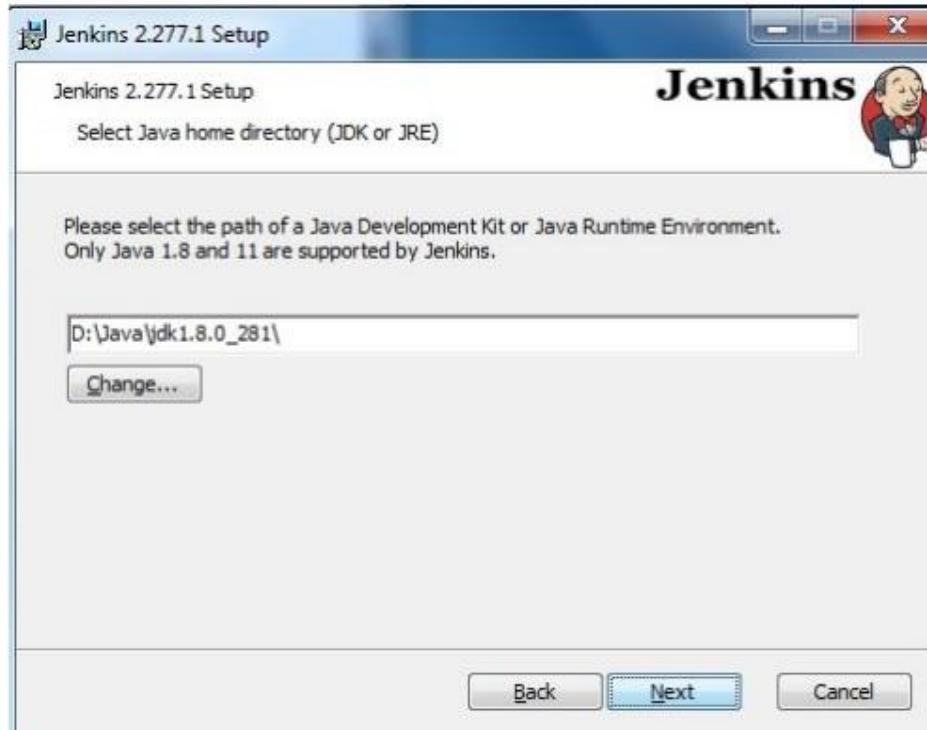
Step 4: Install Jenkins



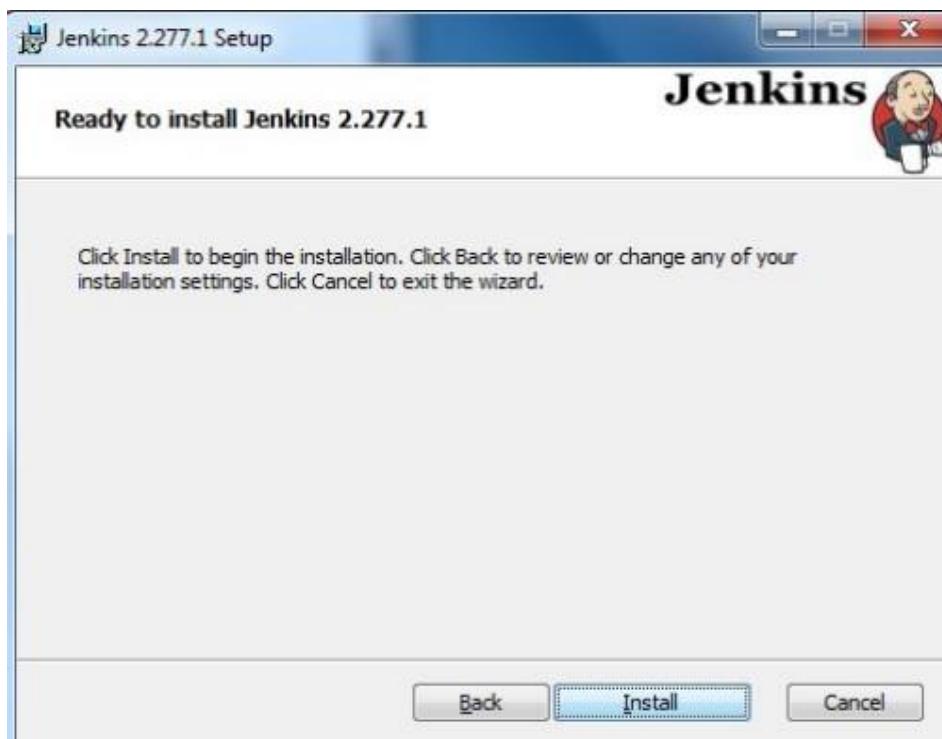
Step6: Check the port number, if it shows that port is used by some other software, then change the port number (ex:8090,8081 etc).



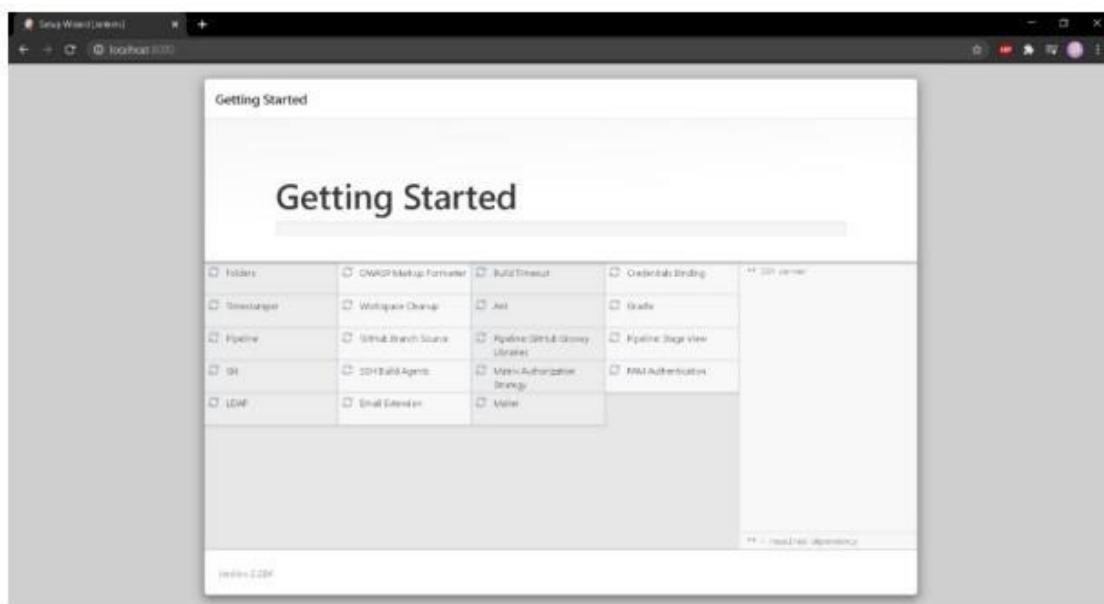
Step7: Check the path of JDK weather it is correct or not.



Step8: Click install button to install the Jenkins

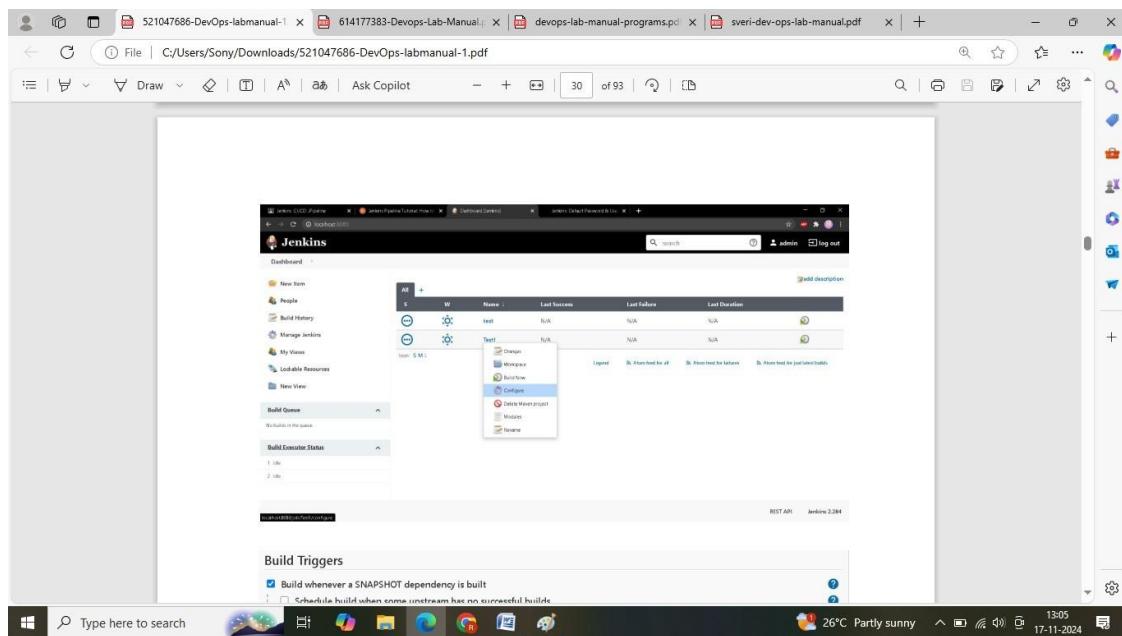


Step9:



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

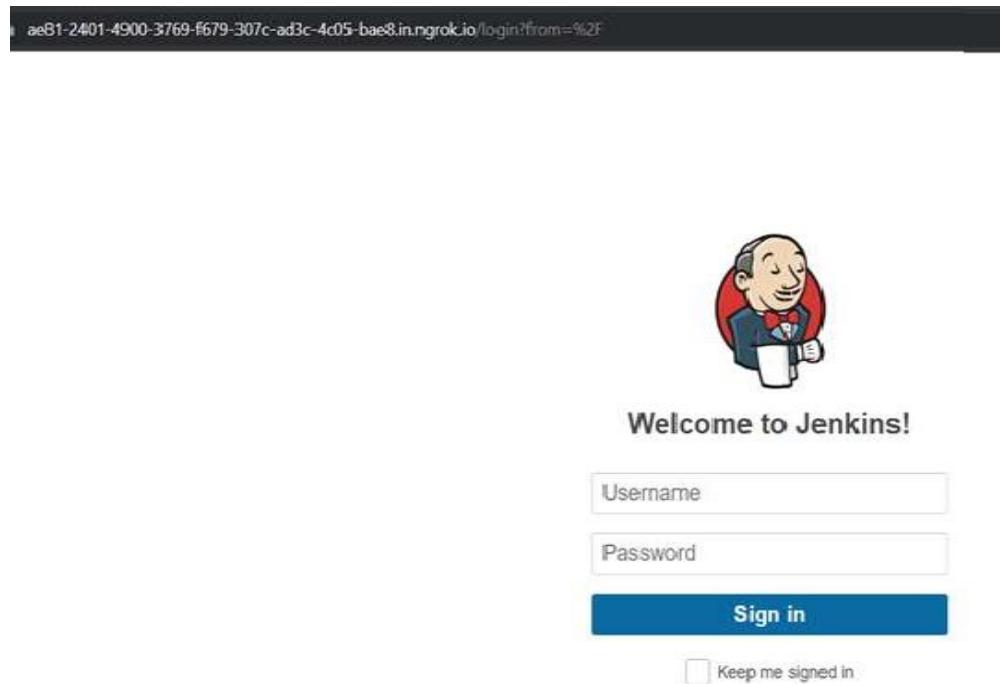
Create a java file in new folder by name test.java

Write the code and push the code to git hub repository

The screenshot shows a GitHub repository named 'Java-Code'. The repository is public and has a master branch. It contains 2 branches and 0 tags. There are 0 forks and 0 stars. The repository was last updated by 'ab7dca1' a week ago with 1 commit. The commit message is 'testjavat'. The README section is empty, and there is a button to 'Add a README'. The repository has no releases or packages published.

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

Login to jenkins

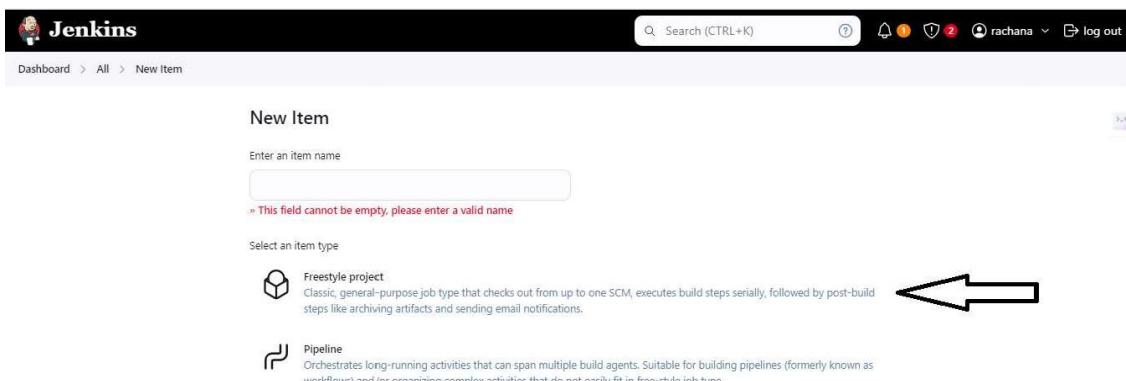


Create new Jenkins job

The screenshot shows the Jenkins dashboard at the URL `http://localhost:8080`. The dashboard has a dark header with the Jenkins logo and a search bar. On the left, there is a sidebar with links for "Build History", "Manage Jenkins", and "My Views". Below that is a "Build Queue" section stating "No builds in the queue." and a "Build Executor Status" section showing "0/2". The main area is a table displaying build information for several projects. The table columns are: Status (green checkmark), Warning (yellow sun icon), Name, Last Success, Last Failure, and Last Duration. The projects listed are: pro1, pro1-test, pro2, pros, and project1. An arrow points to the "New Item" button, which is located at the top left of the table area.

S	W	Name	Last Success	Last Failure	Last Duration
✓	⚠	pro1	2 days 15 hr #1	N/A	0.42 sec
✓	⚠	pro1-test	2 days 15 hr #2	N/A	10 sec
✓	⚠	pro2	2 days 15 hr #2	N/A	1 sec
✓	⚠	pros	2 days 15 hr #2	N/A	10 sec
✓	⚠	project1	2 days 14 hr #1	N/A	5.7 sec

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.

The screenshot shows the Jenkins job configuration page for 'test-jenkin-job'. On the left, a sidebar lists configuration sections: General, Source Code Management, Triggers, Environment, Build Steps, and Post-build. The 'Source Code Management' section is active, showing 'General' configuration. Under 'Source Code Management', 'Git' is selected. A modal dialog is open over the configuration page, prompting for a 'Repository URL' (with the message 'Please enter Git repository.') and 'Credentials' (with a dropdown menu showing 'none' and '+ Add'). The 'Repository URL' field contains 'https://github.com/varunlamp/vgp.git'. A red arrow points to this field. Another red arrow points to the 'Credentials' dropdown menu, which shows 'varunlamp/******** (vqp)' selected. The configuration page also includes fields for 'Branches to build' (set to '*/*main') and buttons for 'Save' and 'Apply'.

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

Write command: javac test.java

Java test

Build Steps

Add build step ^

Filter

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit
- Trigger/call builds on other projects

Step6:

localhost:8080/job/jenkins/configure

Jenkins / jenkins / Configuration

Configure

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

Execute Windows batch command

Command

See the list of available environment variables

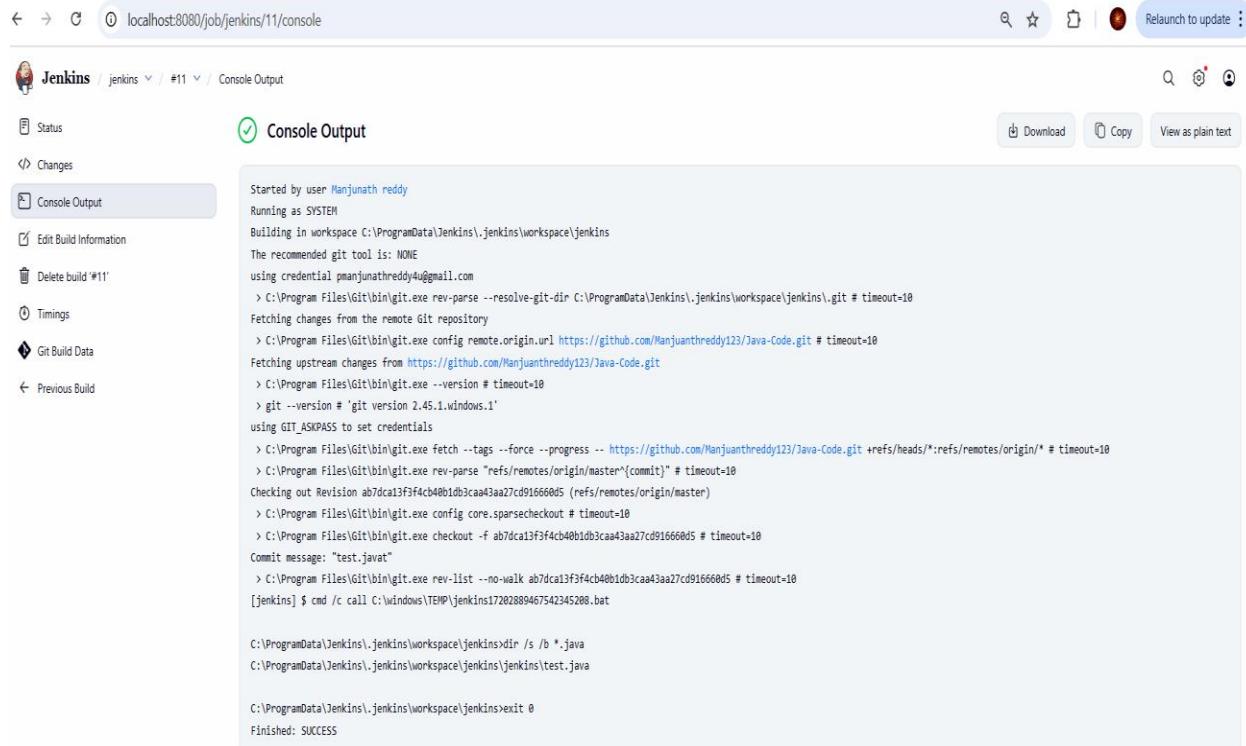
dir /s /b *.java

Advanced

+ Add build step

Step7: click on Apply → save → Build Now

Step8: see the final output in the console



The screenshot shows the Jenkins interface with the URL `localhost:8080/job/jenkins/11/console`. The left sidebar has links for Status, Changes, Console Output (which is selected), Edit Build Information, Delete build '#11', Timings, Git Build Data, and Previous Build. The main area is titled 'Console Output' with a green checkmark icon. It displays the build log for build #11, starting with 'Started by user Manjunath reddy'. The log shows the execution of various git commands to clone the repository, switch branches, and run a Java test. The build concludes with 'Finished: SUCCESS'.

```

Started by user Manjunath reddy
Running as SYSTEM
Building in workspace C:\ProgramData\Jenkins\jenkins\workspace\jenkins
The recommended git tool is: NONE
using credential manjuanthreddy4u@gmail.com
> C:\Program Files\Git\bin\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\jenkins\workspace\jenkins\.git # timeout=10
Fetching changes from the remote Git repository
> C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/Manjuanthreddy123/Java-Code.git # timeout=10
Fetching upstream changes from https://github.com/Manjuanthreddy123/Java-Code.git
> C:\Program Files\Git\bin\git.exe --version # timeout=10
> git --version # 'git' version 2.45.1.windows.1
using GIT_ASKPASS to set credentials
> C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/Manjuanthreddy123/Java-Code.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision ab7dca13f3f4cb40bd3caa43aa27cd916660d5 (refs/remotes/origin/master)
> C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
> C:\Program Files\Git\bin\git.exe checkout -f ab7dca13f3f4cb40bd3caa43aa27cd916660d5 # timeout=10
Commit message: "test.java"
> C:\Program Files\Git\bin\git.exe rev-list --no-walk ab7dca13f3f4cb40bd3caa43aa27cd916660d5 # timeout=10
[jenkins] $ cmd /c call C:\Windows\TEMP\jenkins17202889467542345288.bat

C:\ProgramData\Jenkins\jenkins\workspace\jenkins>dir /s /b *.java
C:\ProgramData\Jenkins\jenkins\workspace\jenkins\jenkins>test.java

C:\ProgramData\Jenkins\jenkins\workspace\jenkins>exit 0
Finished: SUCCESS

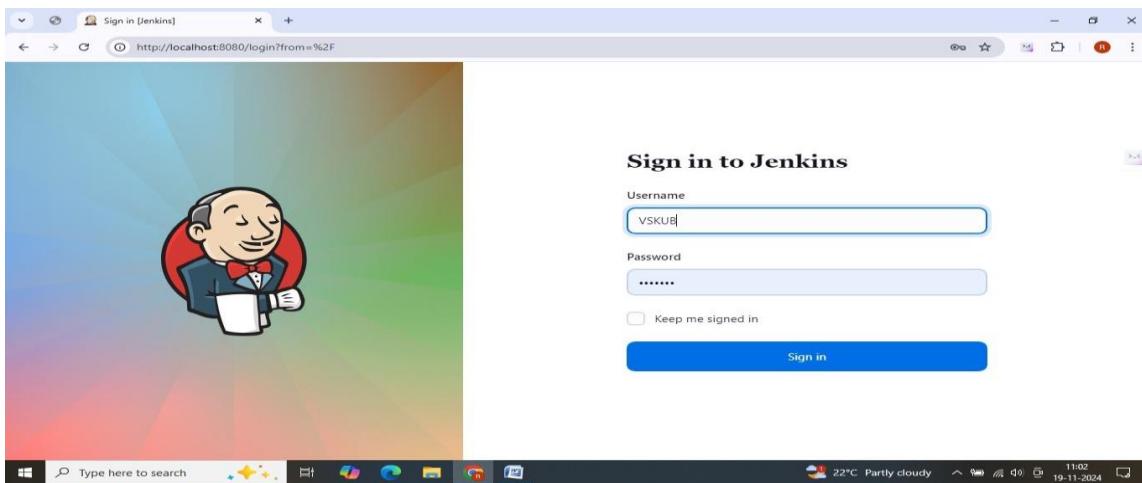
```

Experiment no: 8

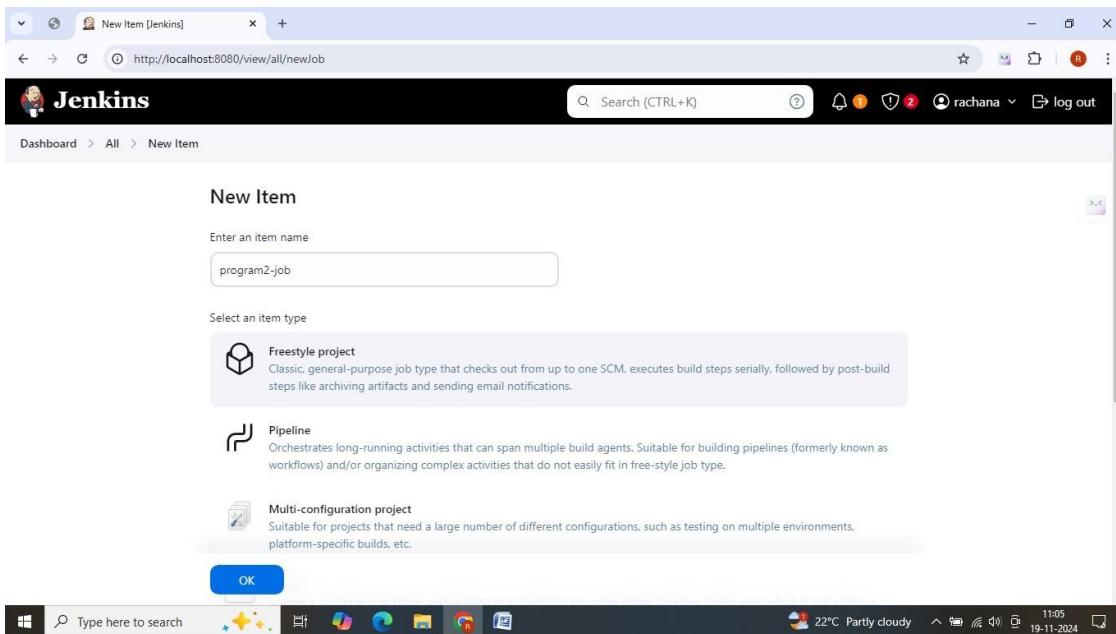
Implement jenkin program using freestyle and pipeline

Implementing Jenkins using free style project

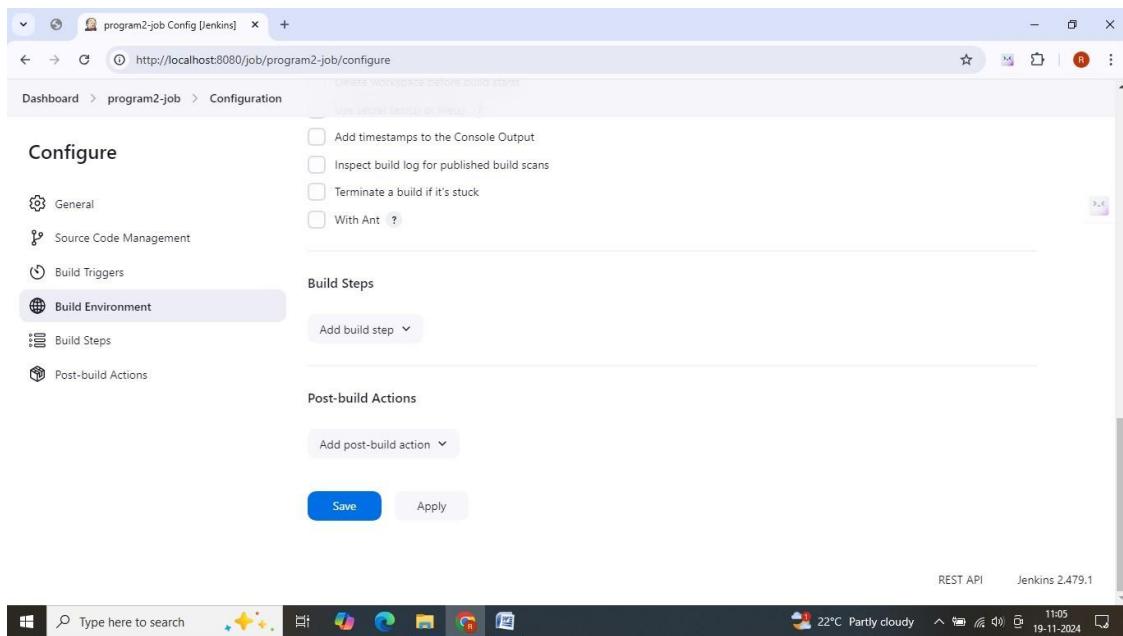
Login to jenkins



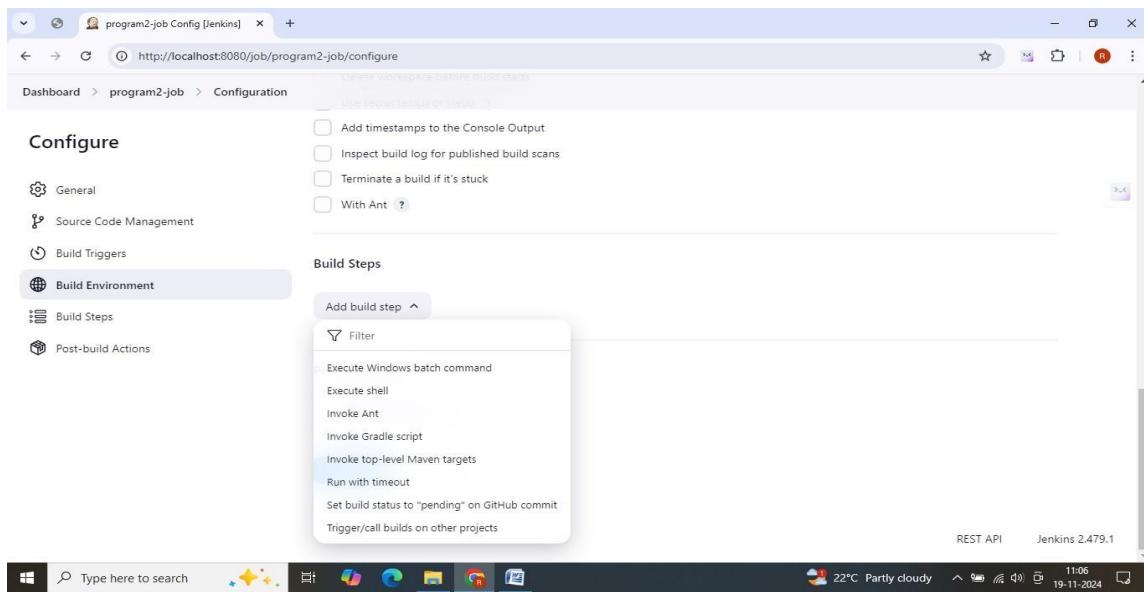
Enter new item name and select free style project



Select build steps

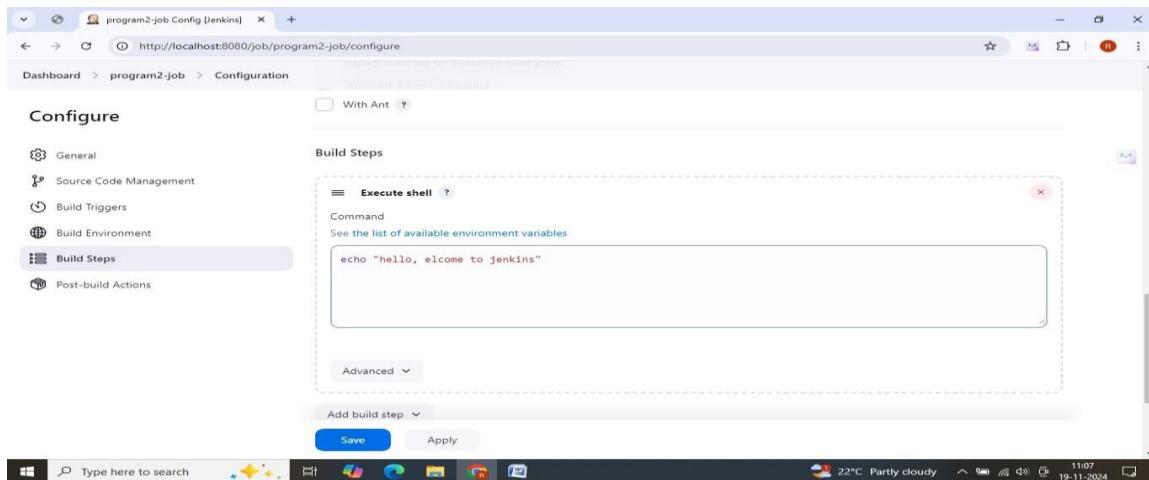


Click on execute shell



Write command

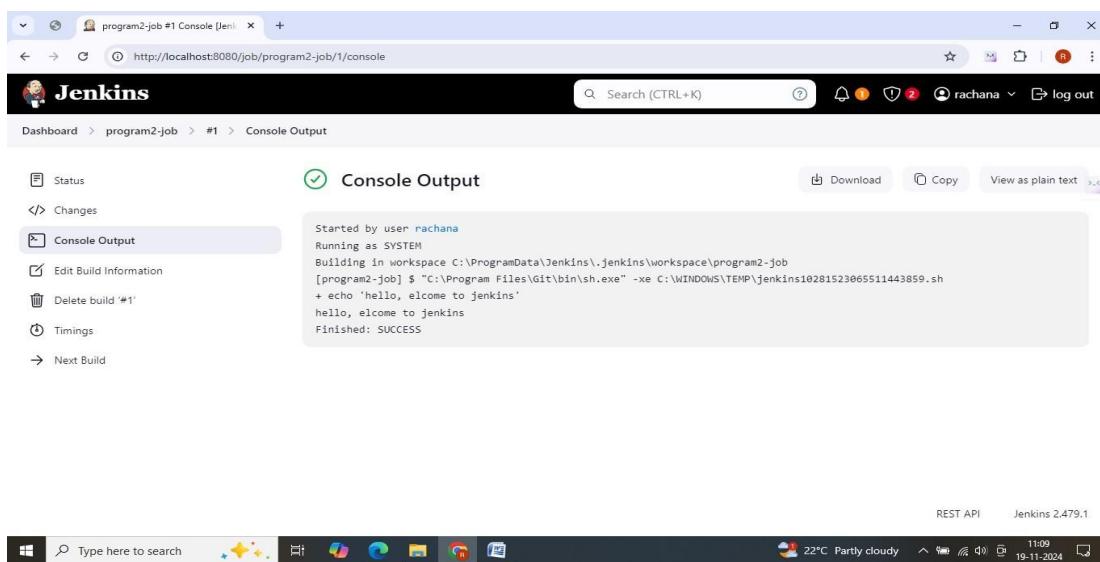
Save and apply



Click on bulid now and see the output



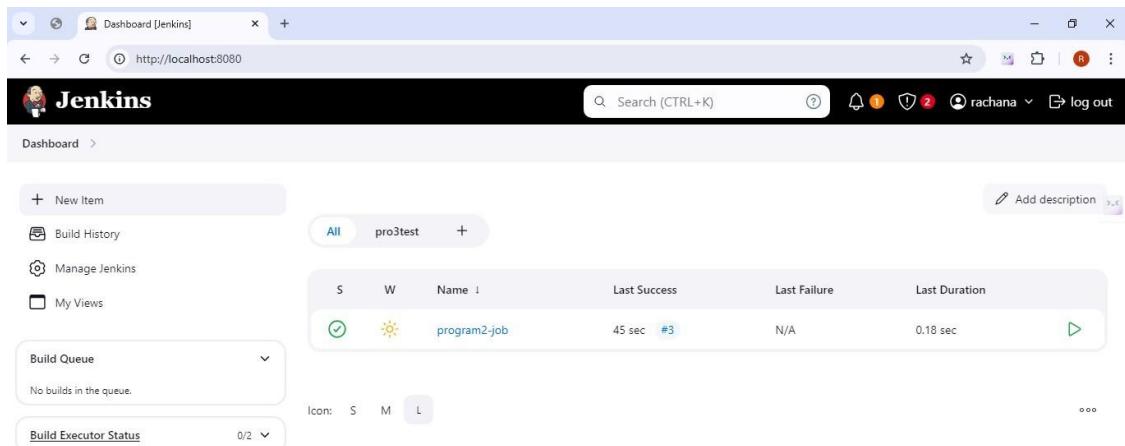
OUTPUT:



Implementing Jenkin program using pipeline

Now create new pipeline job

Click on new item

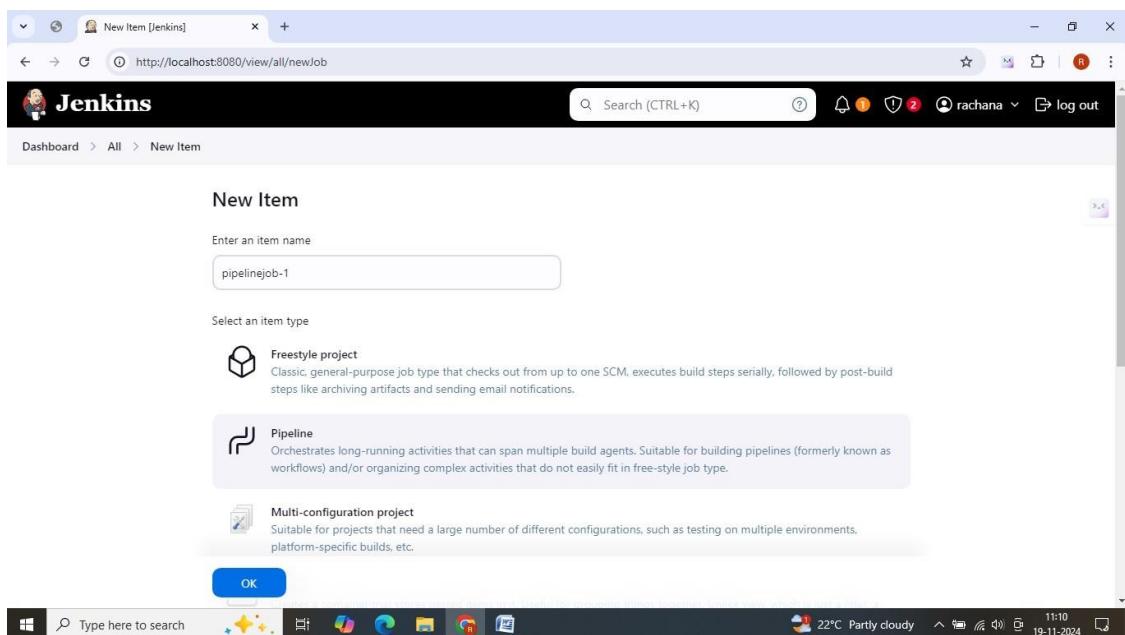


The screenshot shows the Jenkins dashboard at <http://localhost:8080>. The 'Dashboard' link is active. A prominent 'New Item' button is highlighted with a red box. Other buttons include 'Build History', 'Manage Jenkins', and 'My Views'. A search bar at the top right contains 'Search (CTRL+K)'. Below the dashboard, there's a table for 'Build Queue' showing one entry: 'program2-job' with a green success icon, last success at 45 sec, and duration of 0.18 sec. A 'Build Executor Status' section shows 0/2 available.



The taskbar shows the Jenkins icon in the system tray. The system tray also displays the date (19-11-2024), time (11:10), battery level (22°C), signal strength, and network status.

Name the job and click on pipeline



The screenshot shows the 'New Item' dialog on the Jenkins dashboard. The 'Enter an item name' field contains 'pipelinejob-1'. The 'Select an item type' section shows three options: 'Freestyle project' (selected), 'Pipeline' (highlighted with a red box), and 'Multi-configuration project'. The 'Pipeline' description states it's for long-running activities spanning multiple build agents. An 'OK' button is at the bottom.

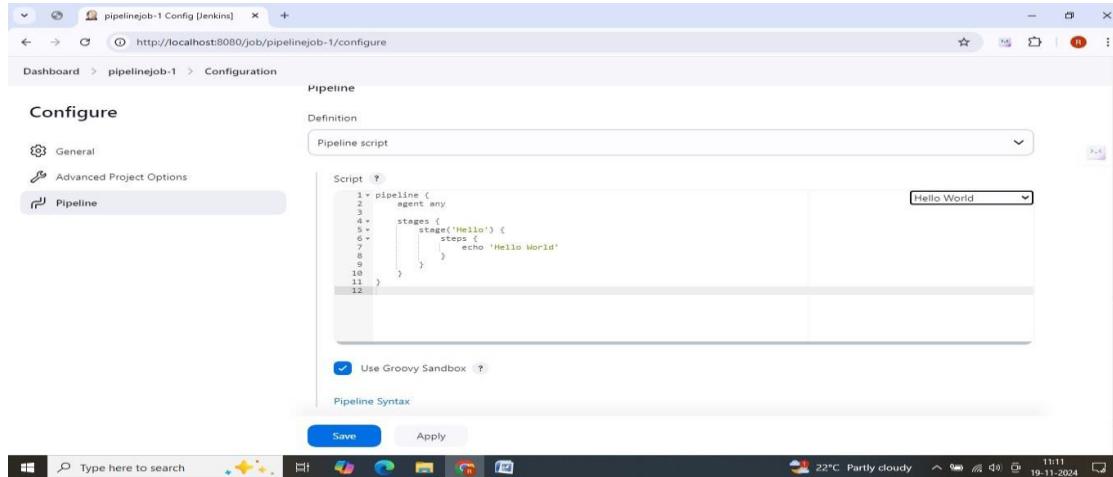


The taskbar shows the Jenkins icon in the system tray. The system tray also displays the date (19-11-2024), time (11:10), battery level (22°C), signal strength, and network status.

Write descriptive code

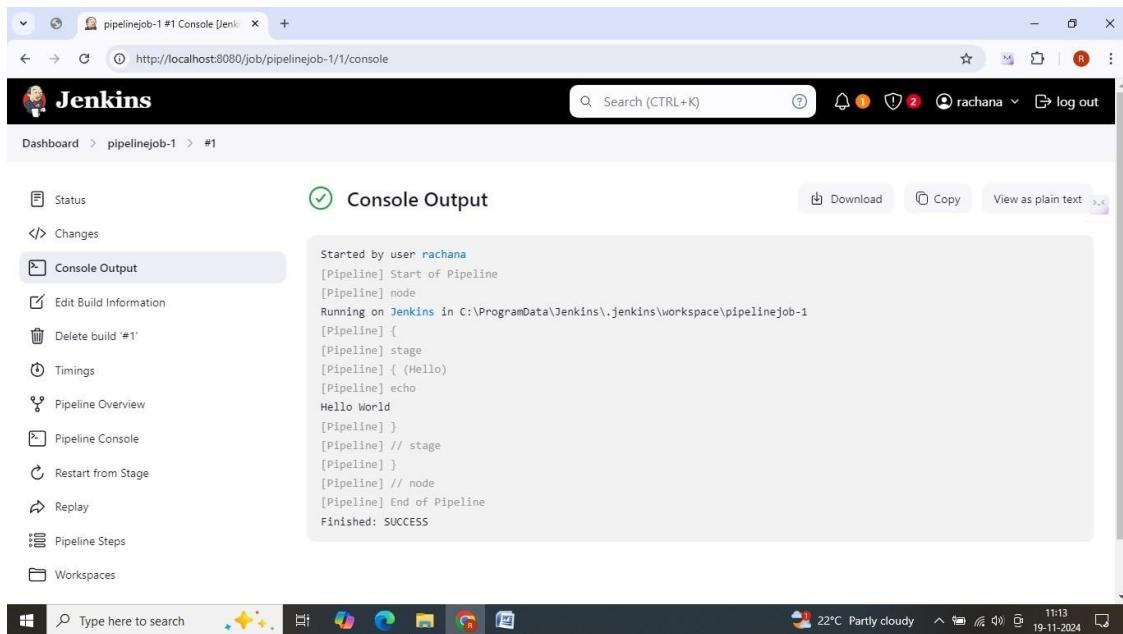
```
pipeline {
    agent any
    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

Click on apply and save



Click on build now

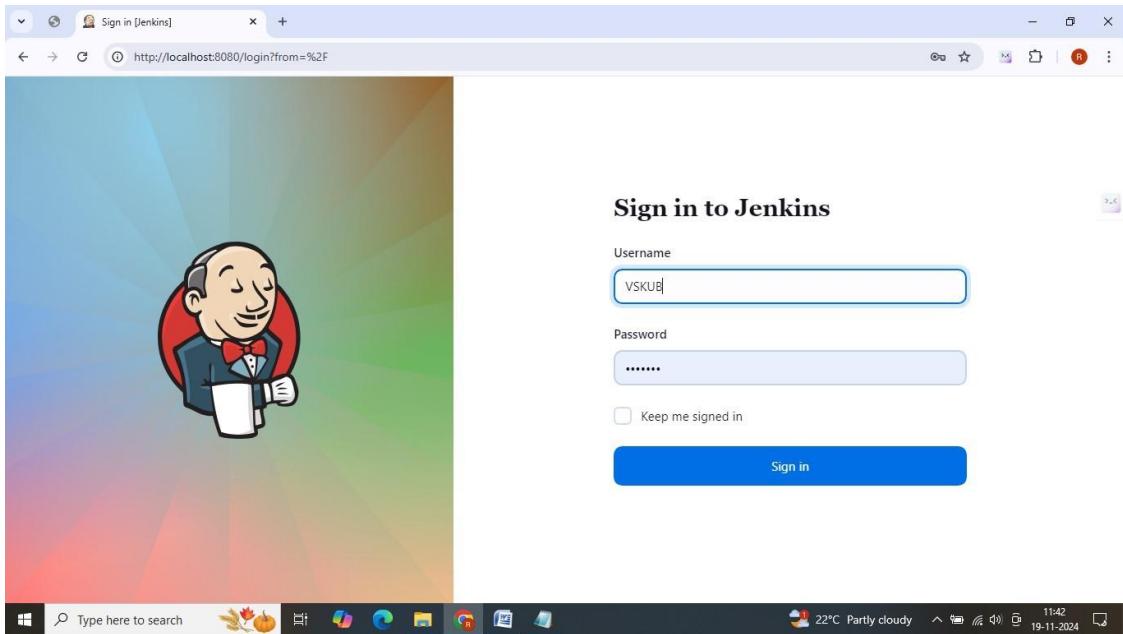
And see the output



Experiment 9

Create first Jenkins Pipeline using Build Pipeline

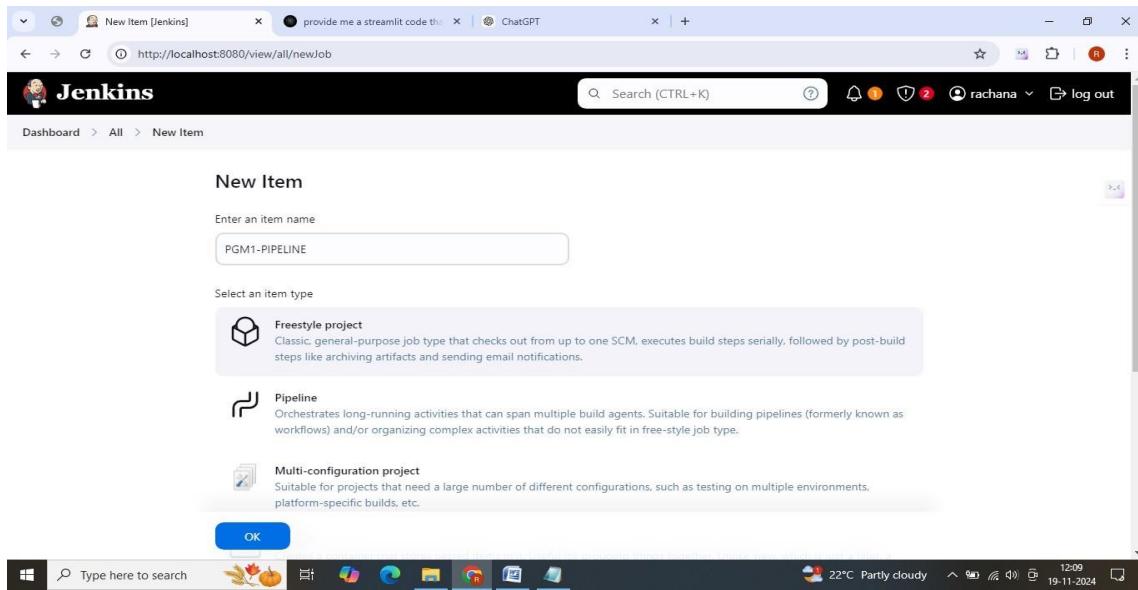
Login to jenkins



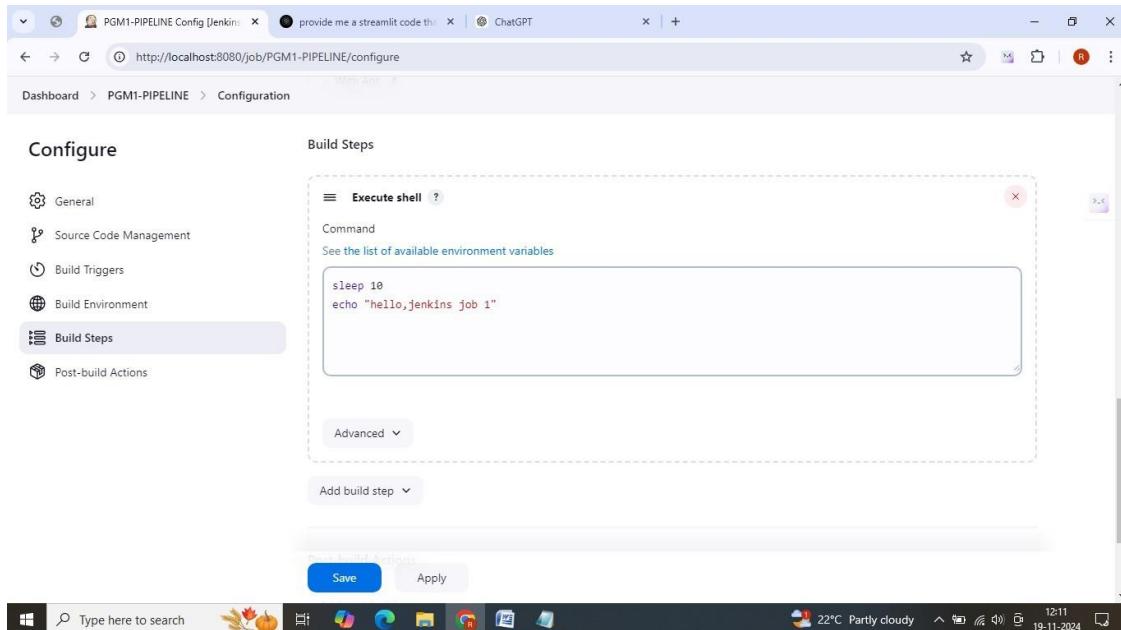
Create new job

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	pipelinejob-1	29 min #1	N/A	6.4 sec
✓	☀️	program2-job	33 min #3	N/A	0.18 sec

Write the job name and click on free style project



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



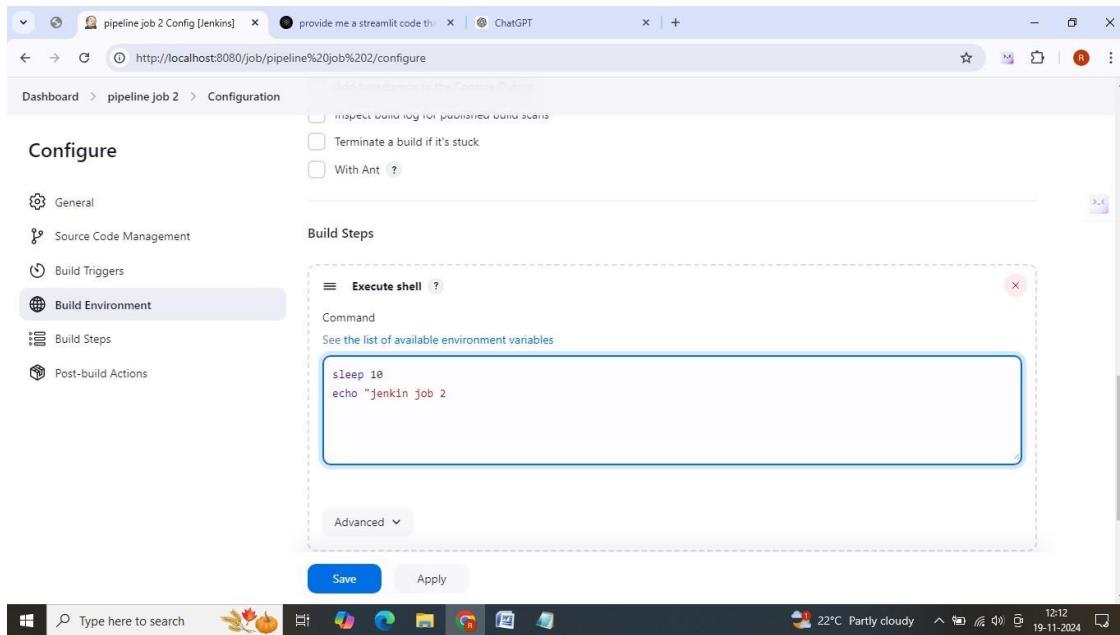
Create another new item

The screenshot shows the Jenkins dashboard at <http://localhost:8080>. A new item named "pro3test" has been created. The dashboard includes sections for Build History, Manage Jenkins, and My Views. A table displays the status of the "PGM1-PIPELINE" job, which is currently N/A for Last Success, Last Failure, and Last Duration. A sidebar shows the Build Queue and Executor Status.

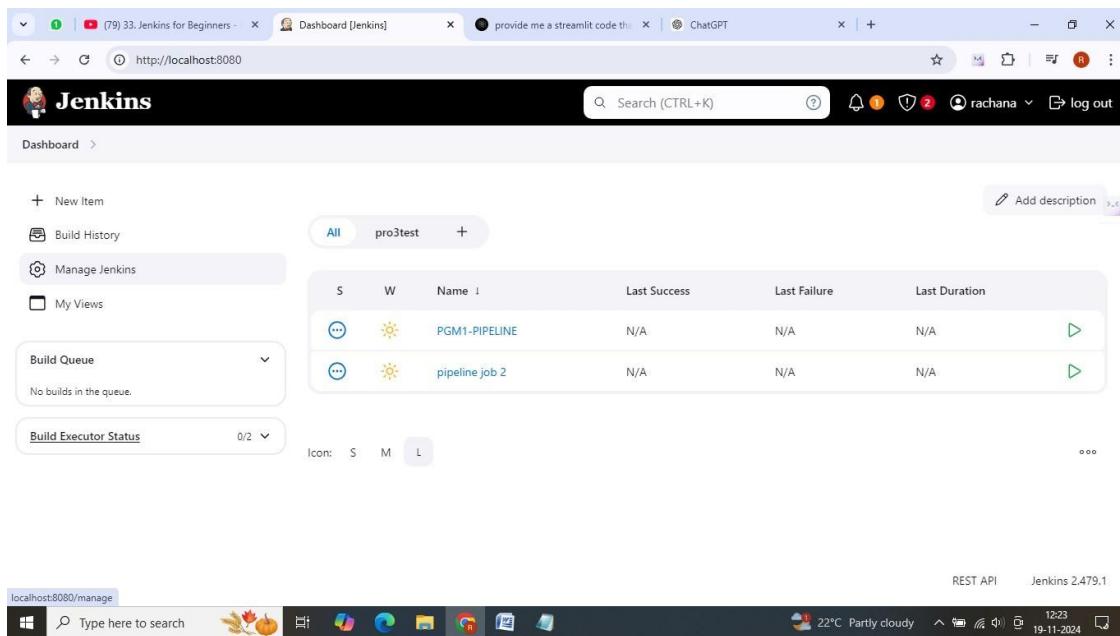
Name the new item and select free style project

The screenshot shows the "New Item" creation dialog in Jenkins. The user has entered "pipeline job 2" as the item name. Under "Select an item type", the "Freeestyle project" option is selected, described as a general-purpose job type. Other options shown are "Pipeline" and "Multi-configuration project". An "OK" button is visible at the bottom left.

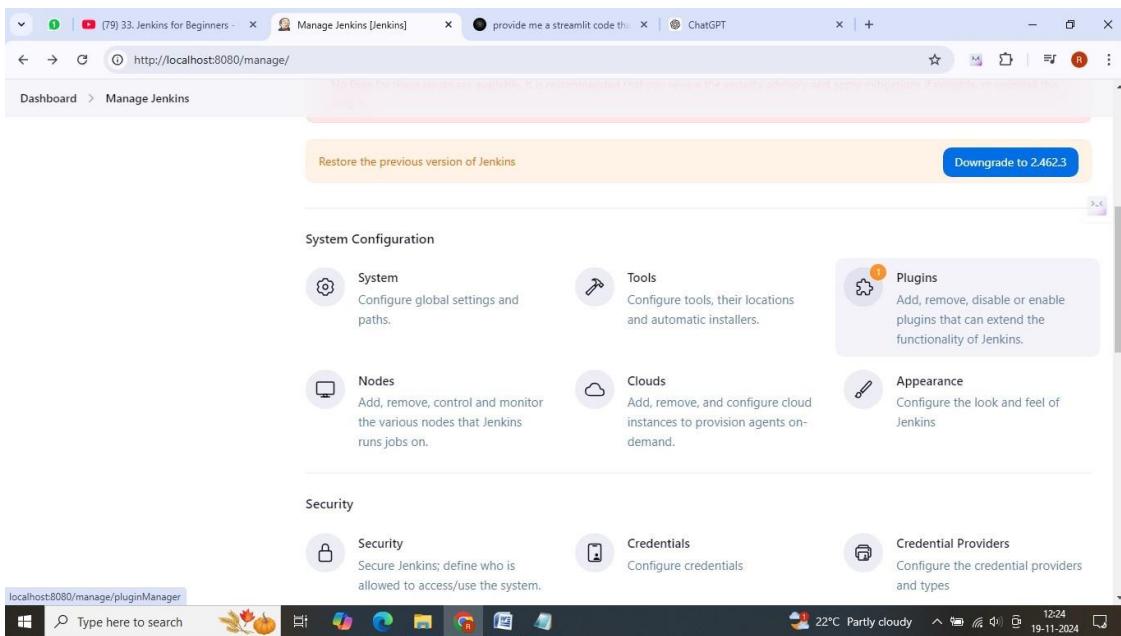
Click on build step → select execute shell → write the shell script → save and apply



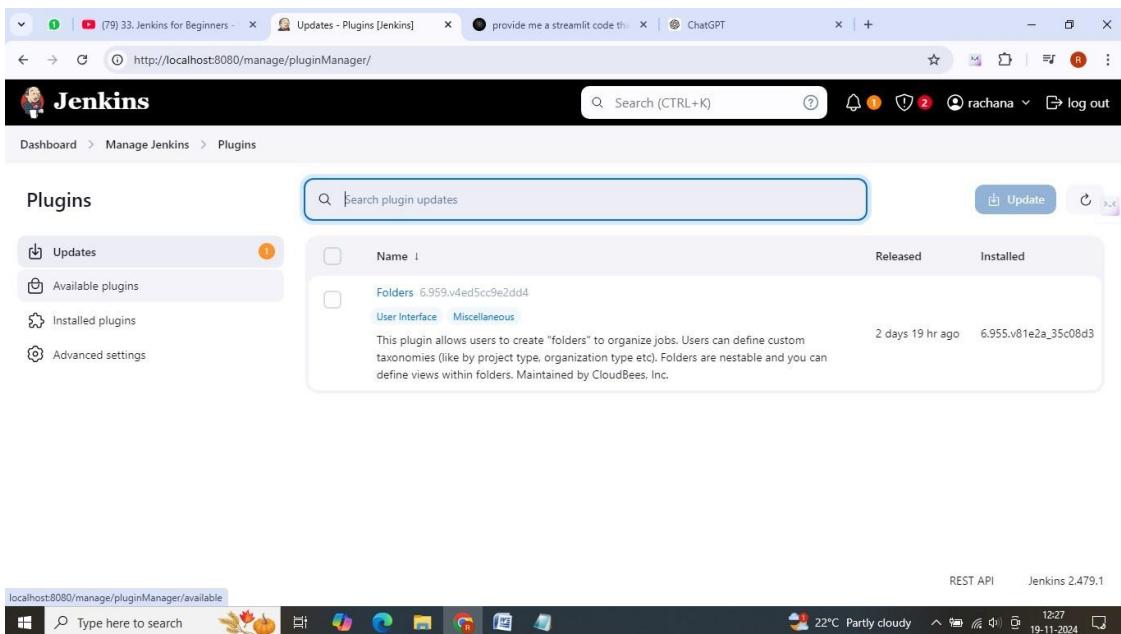
Click on manage jenkins



Click on plugins



Click on available plugins



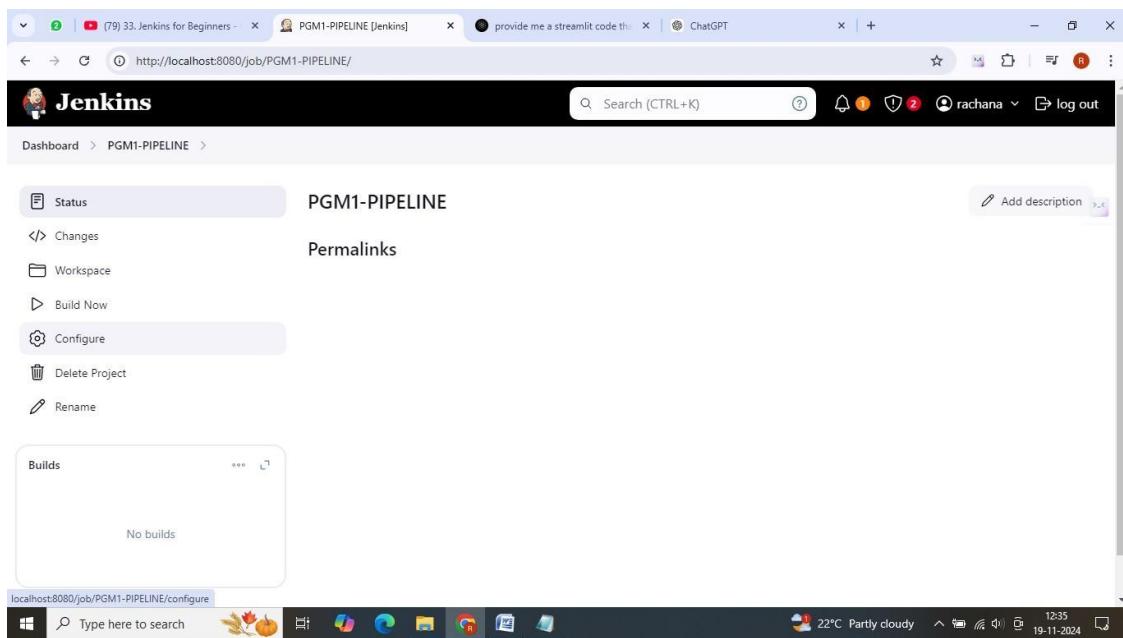
Search for build pipeline and install

The screenshot shows the Jenkins plugin manager interface. A search bar at the top contains the text "buildpipeline". Below the search bar, there are tabs for "Updates" (with 1 update), "Available plugins" (selected), "Installed plugins", and "Advanced settings". On the right, there are "Install" and "Uninstall" buttons. The results table has columns for Name, Last Success, Last Failure, and Last Duration. One result, "PGM1-PIPELINE", is highlighted.

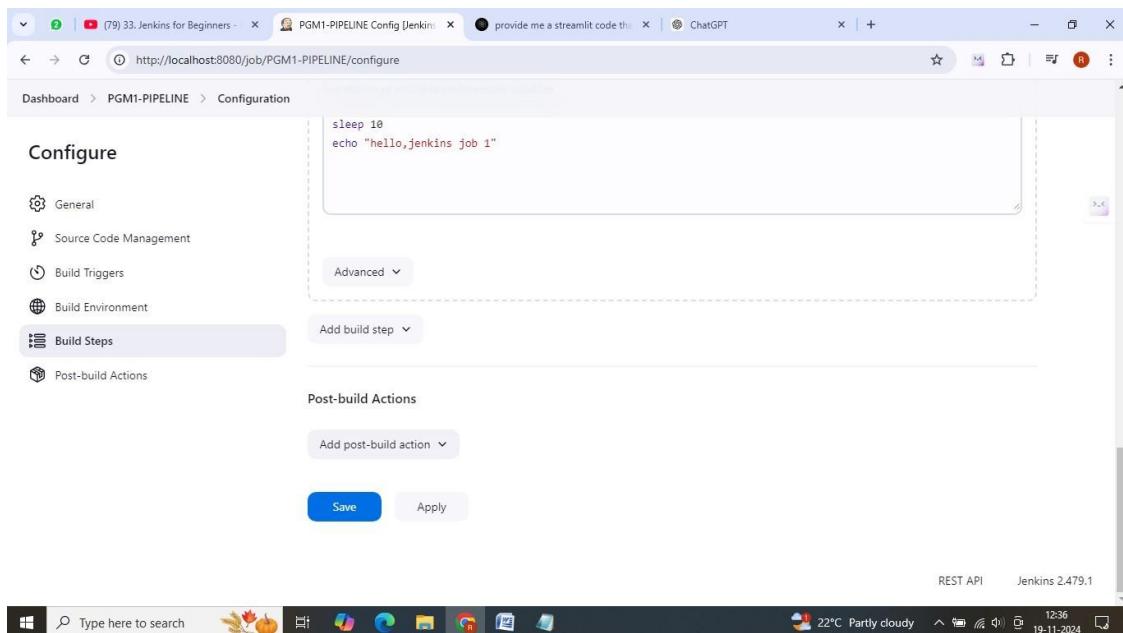
Click on first job created

The screenshot shows the Jenkins dashboard. The left sidebar includes links for "New Item", "Build History", "Manage Jenkins", and "My Views". The main area displays a table of jobs. The first job listed is "PGM1-PIPELINE", which was just created. The table columns are S (Status), W (Last Build), Name, Last Success, Last Failure, and Last Duration. The status for PGM1-PIPELINE is "S" (Success) and "W" is "N/A". The job "pipeline job 2" is also listed with similar status. At the bottom, there is a "Build Executor Status" section showing 0/2 executors available, and a legend for icons S, M, L.

Click on configure



Click on post build actions



Select build other projects

The screenshot shows the Jenkins configuration interface for a job named 'PGM1-PIPELINE'. In the left sidebar, under 'Post-build Actions', the 'Build other projects' option is selected. A dropdown menu is open, listing several actions. The 'Build other projects' option is highlighted. Other listed actions include: Aggregate downstream test results, Archive the artifacts, Publish JUnit test report, Record fingerprints of files to track usage, Git Publisher, Build other projects (manual step), E-mail Notification, Editable Email Notification, Set GitHub commit status (universal), Set build status on GitHub commit [deprecated], Trigger parameterized build on other projects, and Delete workspace when build is done.

The screenshot shows the Jenkins configuration interface for a job named 'PGM1-PIPELINE'. In the left sidebar, under 'Post-build Actions', the 'Build other projects' option is selected. A modal dialog titled 'Build other projects' is open, showing a 'Projects to build' input field containing 'pipeline job 2.' Below it, there are three radio button options: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. At the bottom of the dialog are 'Save' and 'Apply' buttons.

Click on plus button

The screenshot shows the Jenkins dashboard with the 'Build Queue' section. It lists two items:

S	W	Name	Last Success	Last Failure	Last Duration
...	...	PGM1-PIPELINE	N/A	N/A	N/A
...	...	pipeline job 2	N/A	N/A	N/A

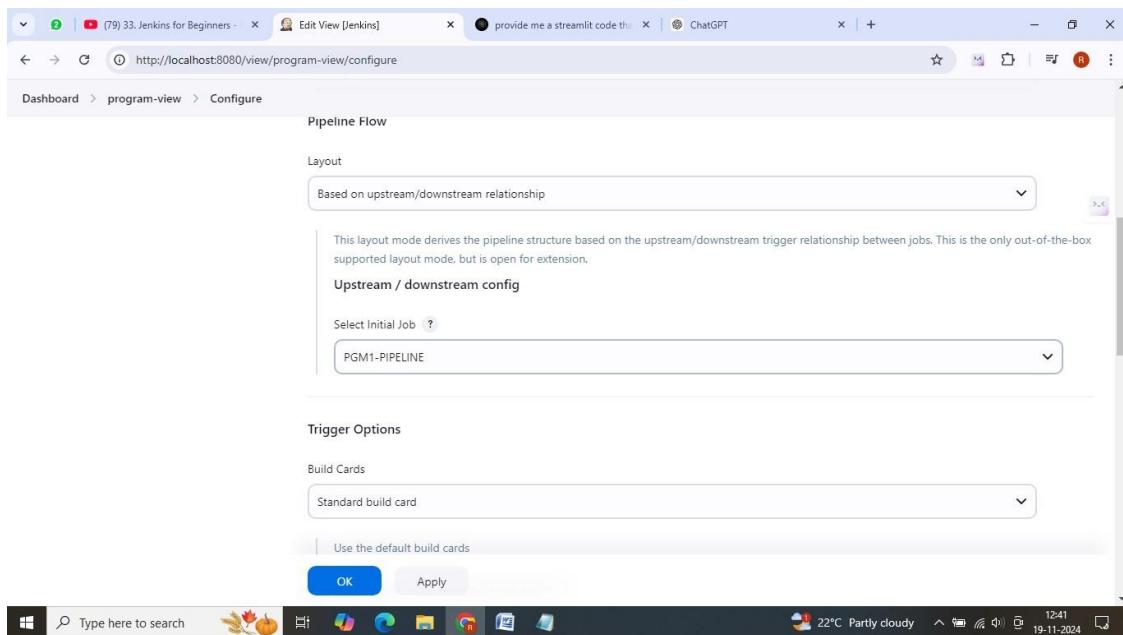
Provide name

Bulid pipeline view

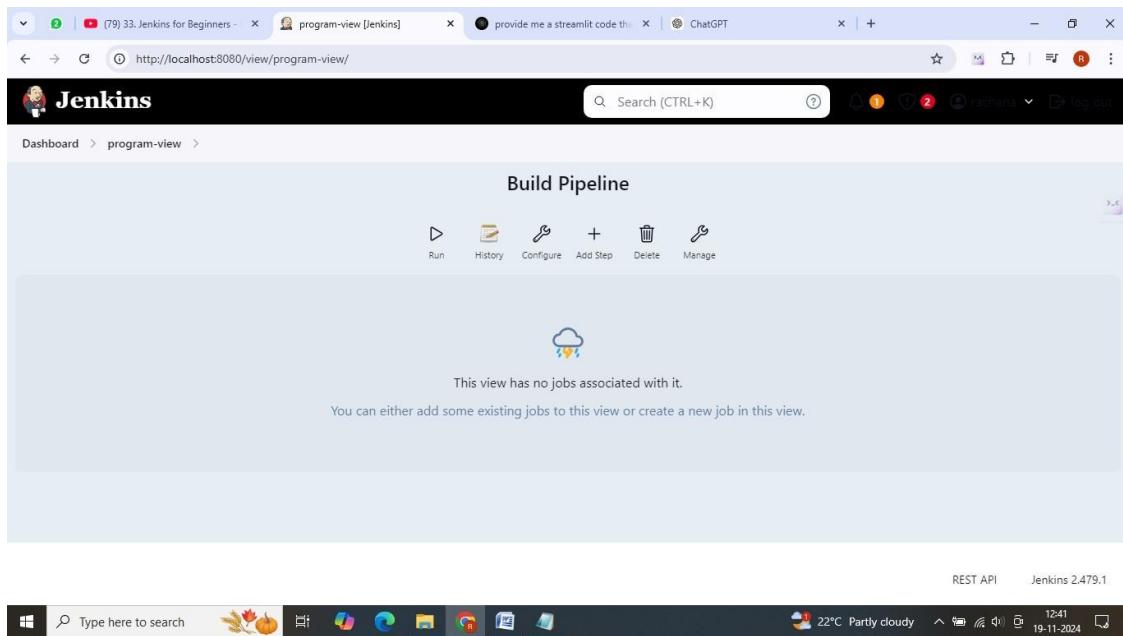
The screenshot shows the 'New view' creation page. The 'Name' field is filled with 'program-view'. The 'Type' section has 'Build Pipeline View' selected. Other options shown are 'List View' and 'My View'. A 'Create' button is at the bottom.

Select initial job

Click apply and save



Click on RUN



OUTPUT:

