a1. Write code for a simple user registration form for an event.

HTML CODE WITH INLINE CSS FOR REGISTRTION FORM ->

<!DOCTYPE html>

**<head>**

**<title>**Event Registration**</title>**

**</head>**

**<body** style="display: flex; justify-content: center; align-items: center; height: 750px; background-color: #f0f8ff;"**>**

**<form** style="border: 1px solid gray; padding: 20px; border-radius: 5px; width: 300px; background-color: white;"**>**

**<h3** style="text-align: center; font-size: 24px;"**>**Register**</h3>**

**<input** id="name" type="text" placeholder="Name" style="width: 90%; padding: 10px; margin-bottom: 15px; font-size: 18px;"**/>**

**<input** type="email" placeholder="Email" required style="width: 90%; padding: 10px; margin-bottom: 15px; font-size: 18px;"**/>**

**<input** type="tel" placeholder="Phone" required style="width: 90%; padding: 10px; margin-bottom: 15px; font-size: 18px;"**/>**

**<button** id="button" type="submit" style="width: 100%; padding: 10px; background-color: #28a745; color: white; border-radius: 3px; font- size: 18px;"**>**Submit**</button>**

**</form>**

**<script** src="script.js"**></script>**

**</body>**

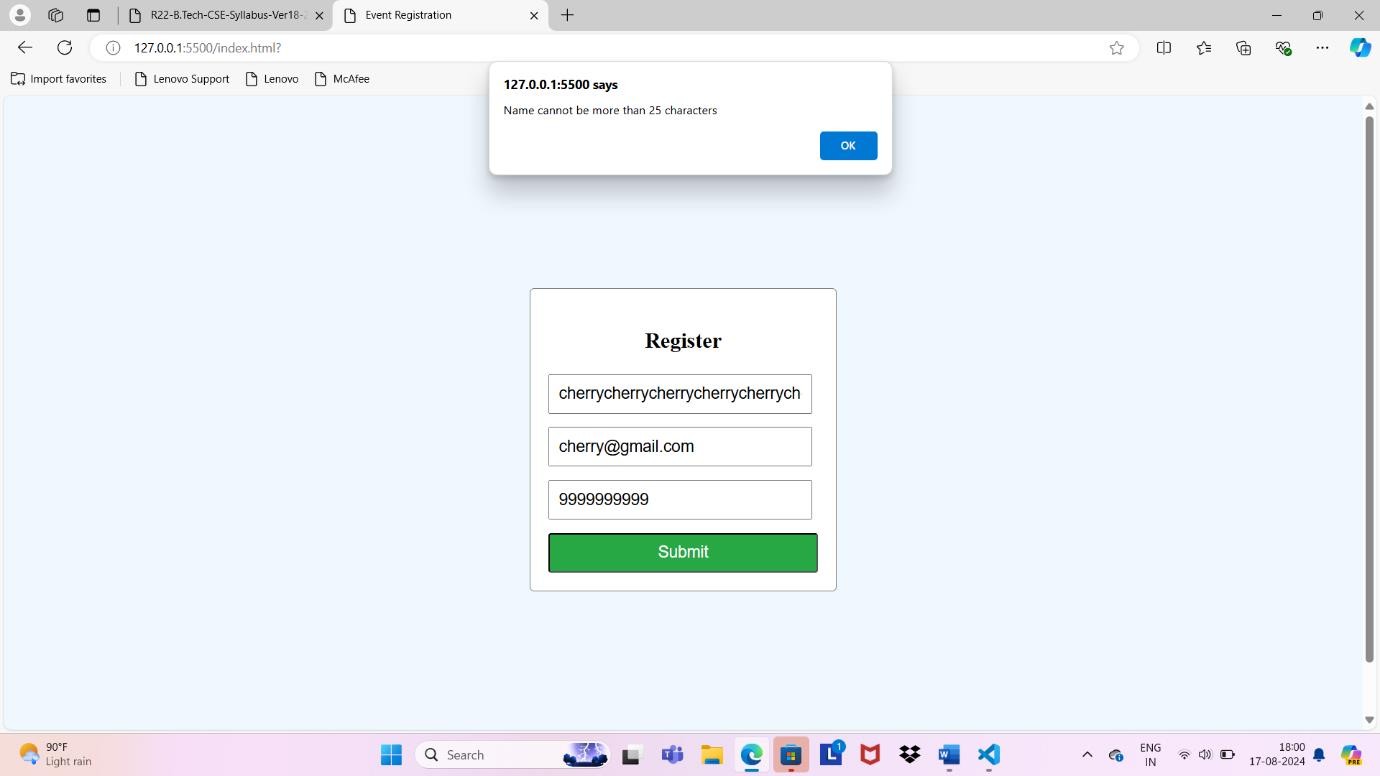
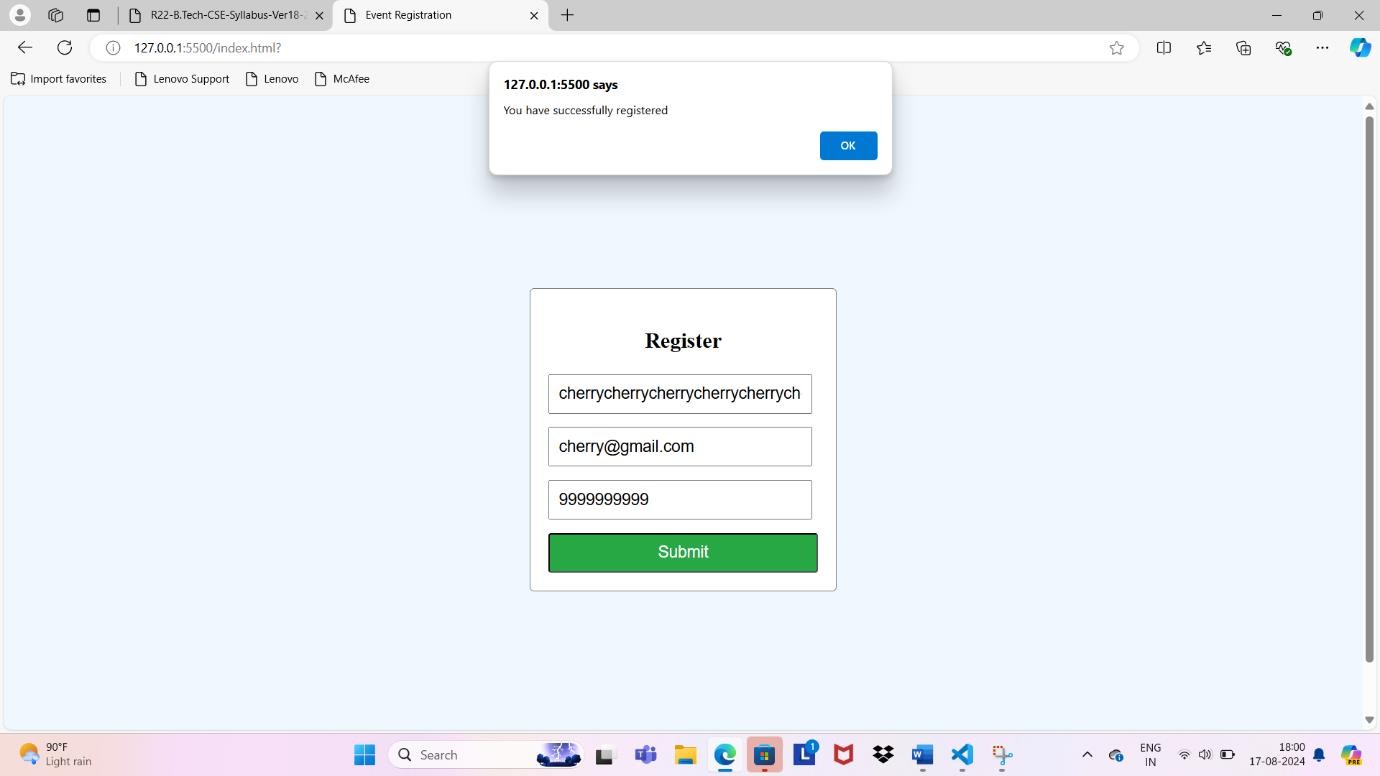
**</html>**

JAVASCRIPT CODE (TO HANDLE THE CASE WHEN MORE THAN 25 CHARACTERS ARE ENTERED IN NAME FIELD AND GET AN ALERT WHEN SUBMIT BUTTON IS CLICKED) ->

document.getElementById("button").addEventListener('click',()=>{ str = document.getElementById("name").value;

**if**(str.length>25)alert("Name cannot be more than 25 characters"); alert("You have successfully registered");

})



1. Explore Git and GitHub commands.

### VERSION CONTROL SYSTEM ->

A **Version Control System (VCS)** is a tool that helps manage changes to files, particularly source code, over time. It tracks modifications, allows multiple

people to collaborate on the same project, and ensures that a complete history of changes is maintained. VCS is essential for software development, but it can also be used for other types of documents or files.

### GIT ->

**Git** is a distributed version control system (VCS) designed to manage and track changes in source code during software development. Created by Linus

Torvalds in 2005, Git is widely used by developers to collaborate on projects, keep track of changes, and manage multiple versions of their codebase.

### GITHUB ->

**GitHub** is a web-based platform and service that provides hosting for software development and version control using Git. It offers a collaborative

environment where developers can manage and share their projects, track issues, and work together on code. GitHub is widely used in the software development community for both open-source and private projects.

Git Commands to push files to remote repo

**git init :** creates a new repository. The files are in working directory and are untracked

**git status:** Show current repository status. ( whether untracked or tracked)

**git add <file>:** adds the file to staging area .(The file is now tracked)

**git config –global user.name “cherry”:** to set a username

**git config –global user.email “**[**cherry@gmai.com”**](mailto:cherry@gmai.com)**:** to set an email

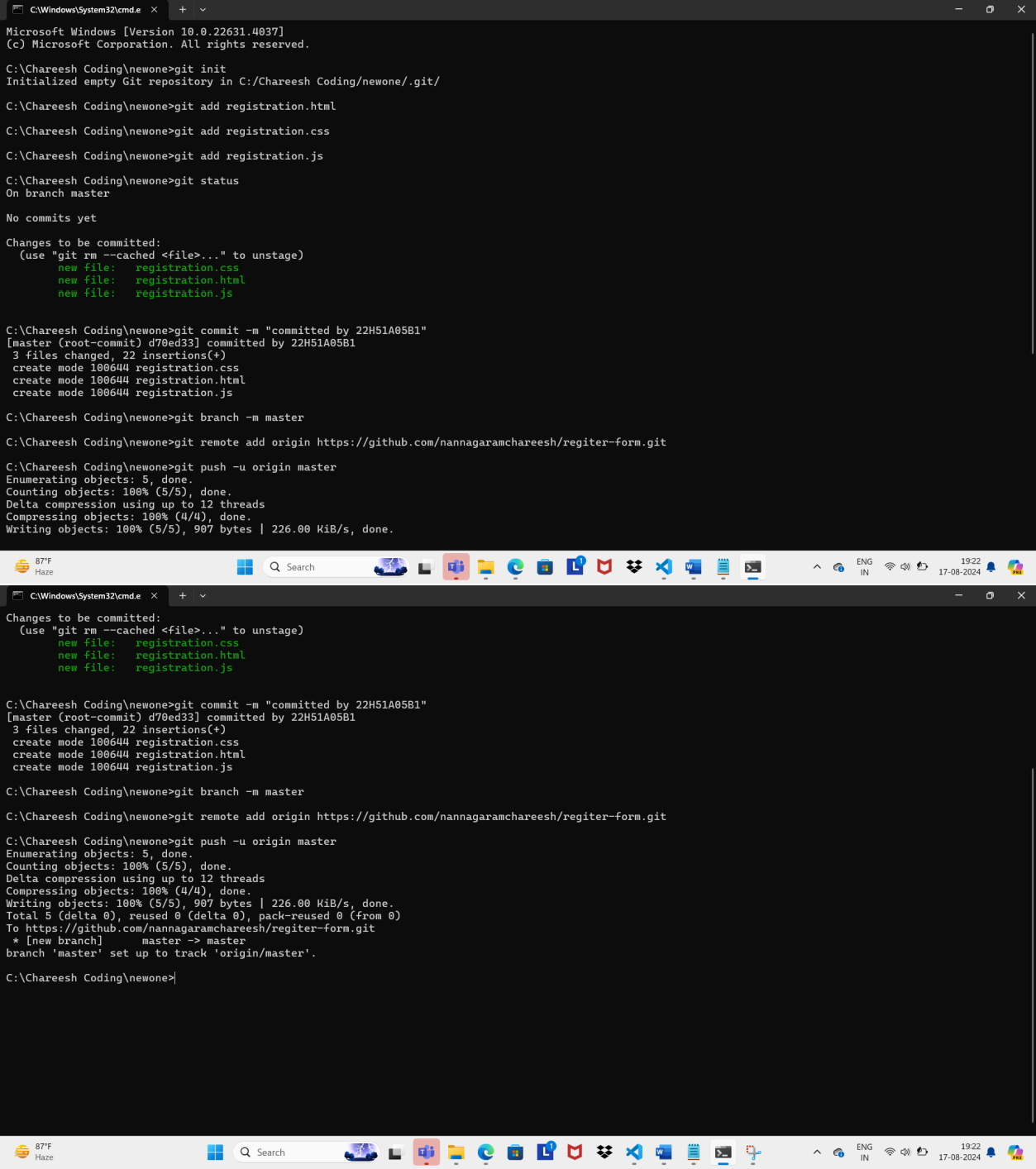
**git commit -M “commited by 22h51a05b1:** Save changes.(now the file will be in local repo)

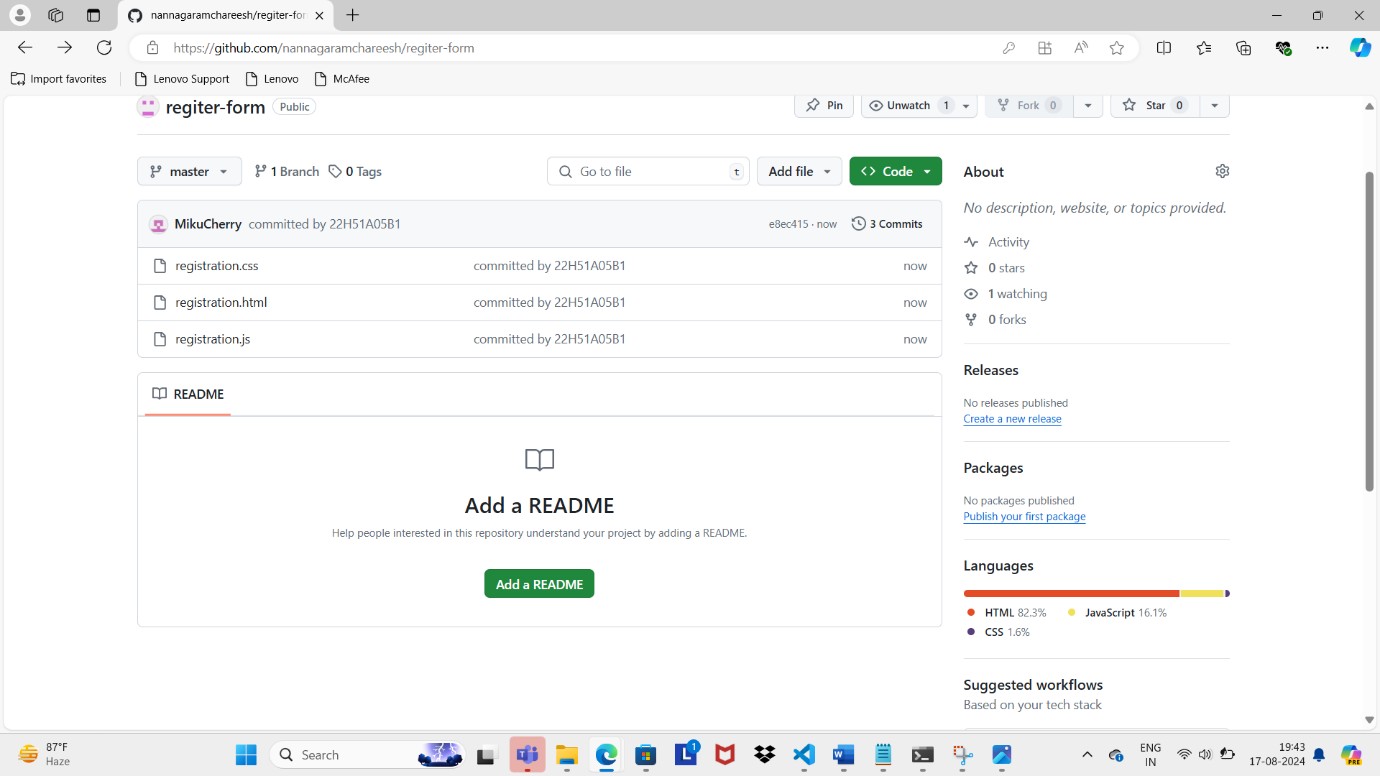
**git remote add origin ‘your repo link’:** To establish a connection between local and remote rep

**git branch:** To know you current branch

**git branch -m master:** To rename branch to master

**git push -u origin master:** To upload file to remote repo (now the file will be In remote repo)





1. Practice Source code management on GitHub. Experiment with the source code written in exercise 1.

<!DOCTYPE html>

**<html** lang="en"**>**

**<head>**

**<meta** charset="UTF-8"**>**

**<title>**Event Registration**</title>**

**</head>**

**<body** style="display: flex; justify-content: center; align-items: center; height: 750px; background-color: #f0f8ff;"**>**

**<form** style="border: 1px solid gray; padding: 20px; border-radius: 5px; width: 300px; background-color: white;"**>**

**<h3** style="text-align: center; font-size: 24px;"**>**Register**</h3>**

**<input** id="name" type="text" placeholder="Name" style="width: 90%; padding: 10px; margin-bottom: 15px; font-size: 18px;"**/>**

**<input** type="email" placeholder="Email" required style="width: 90%; padding: 10px; margin-bottom: 15px; font-size: 18px;"**/>**

**<input** type="tel" placeholder="Phone" required style="width: 90%; padding: 10px; margin-bottom: 15px; font-size: 18px;"**/>**

**<button** id="button" type="submit" style="width: 100%; padding: 10px; background-color: #28a745; color: white; border-radius: 3px; font- size: 18px;"**>**Submit**</button>**

**<button** id="button" type="submit" style="width: 100%; padding: 10px; background-color: #14142d; color: white;margin-top:15px; border- radius: 3px; font-size: 18px;"**>**Clear form**</button>**

**</form>**

**<script** src="script.js"**></script>**

**</body>**

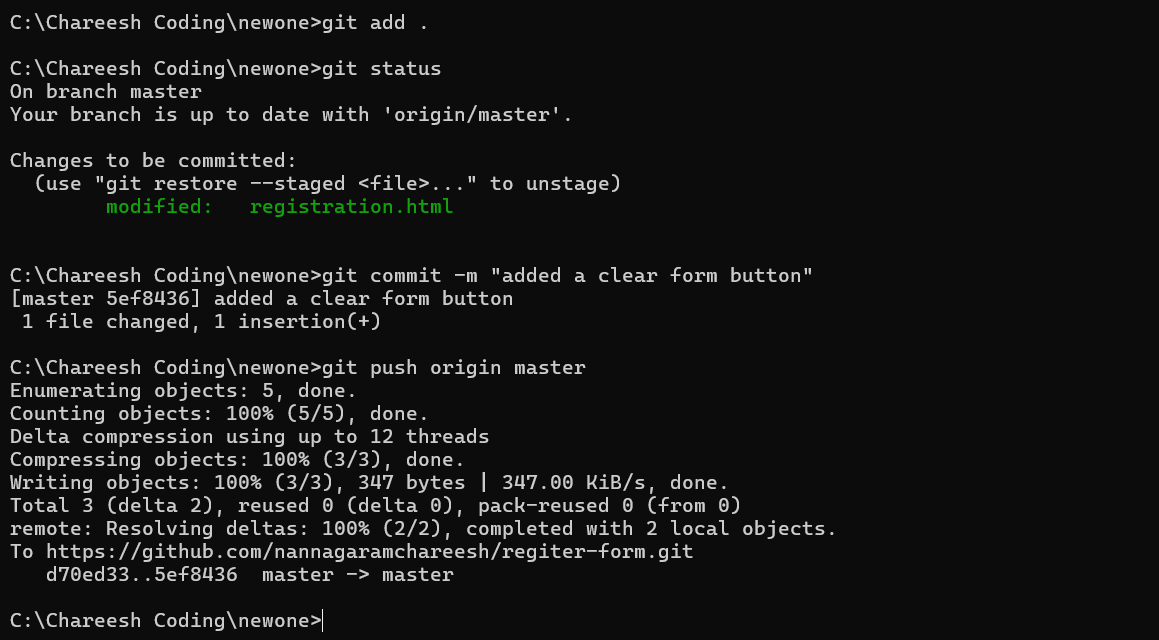
**</html>**

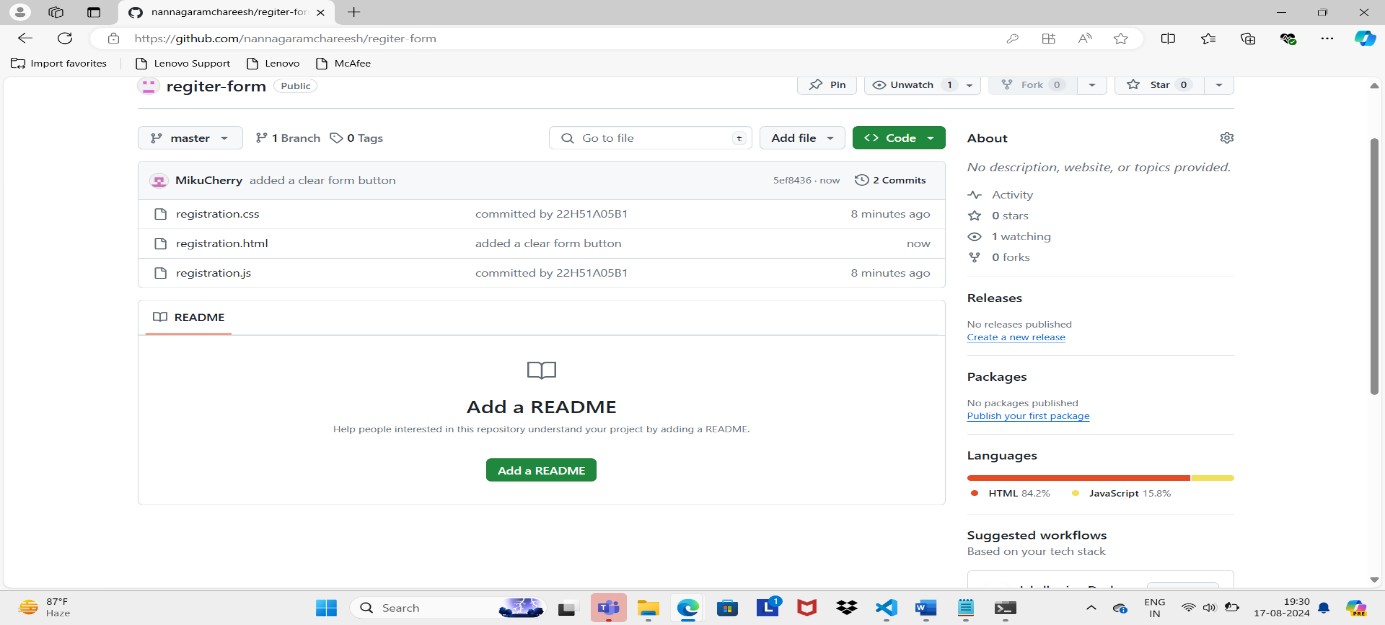
So this is the week one html code in which I added an additional button ‘clear form ‘ now lets commit these changes

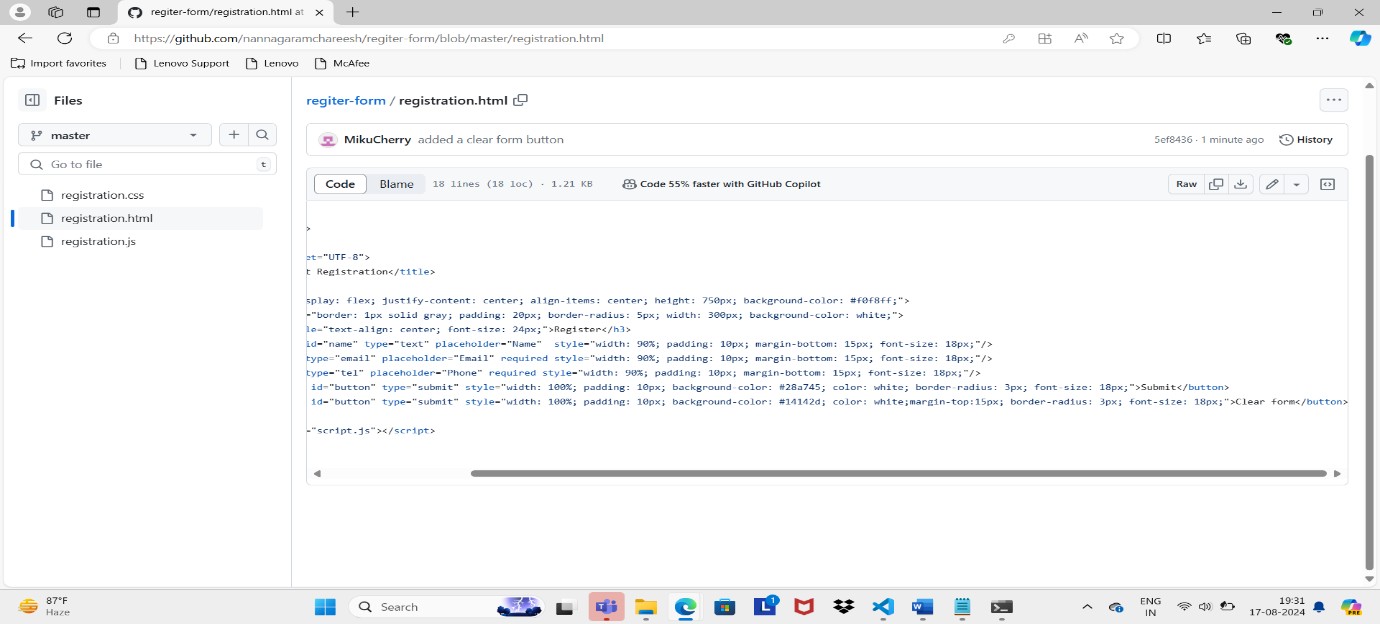
Commands to commit changes in an existing repository->

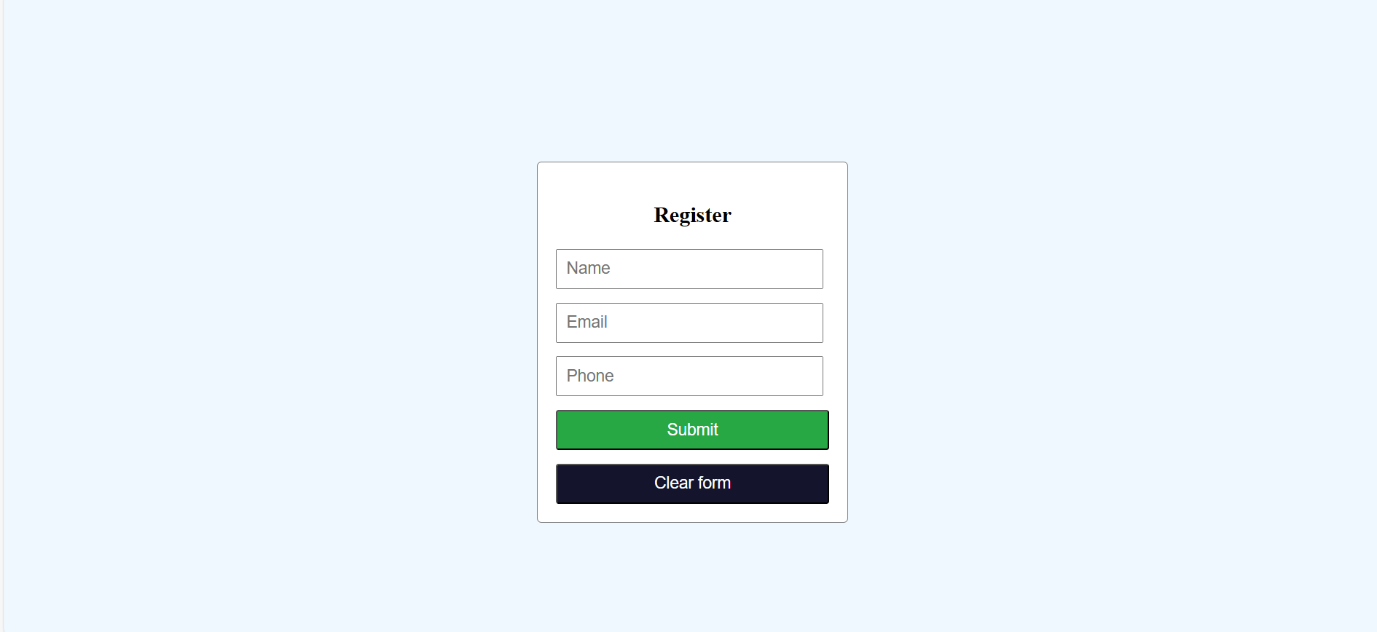
### git add .

**git commit -m “changes made” git push origin master**









BRANCHING -

Branching ->Branching is like making a copy of your code so you can try new things without changing the original. If your new changes work, you can combine them back with the original; if not, the original stays safe.

->So you can create a new branch and work on a different feature and then merge it back to the main branch

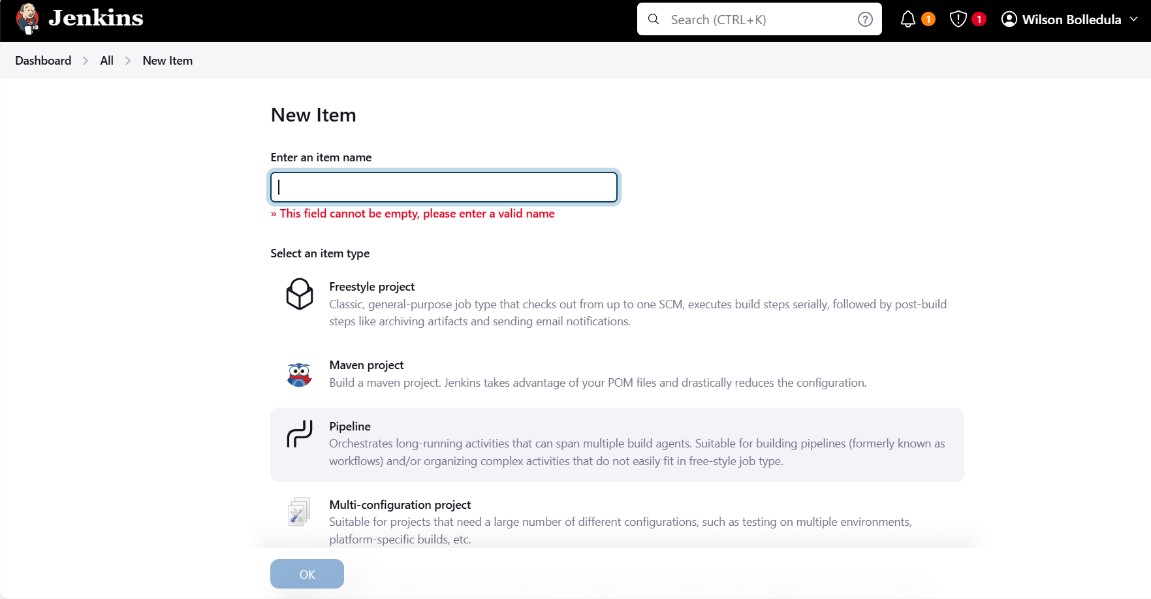
-> creating a new branch Command: git branch branchName

->switching to another branch Command: git switch branchName

->Viewing all branches Command: git branch

Now lets understand branching more clearly -

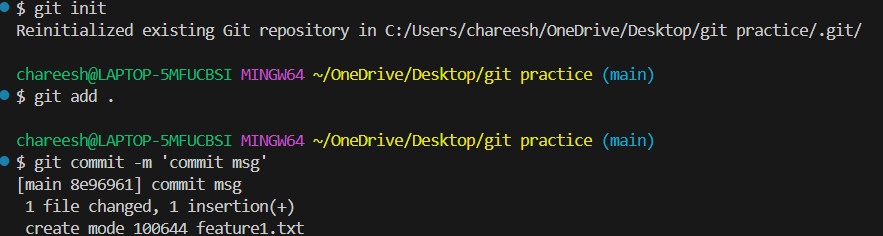
->create a folder and inside it create a text file and write some text inside it



->Now initialize git using the command : git init

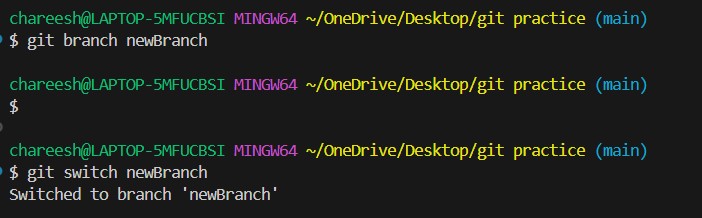
->then add it to the staging area using the command :git add .

->then commit the changes using commit -m ‘commit msg’

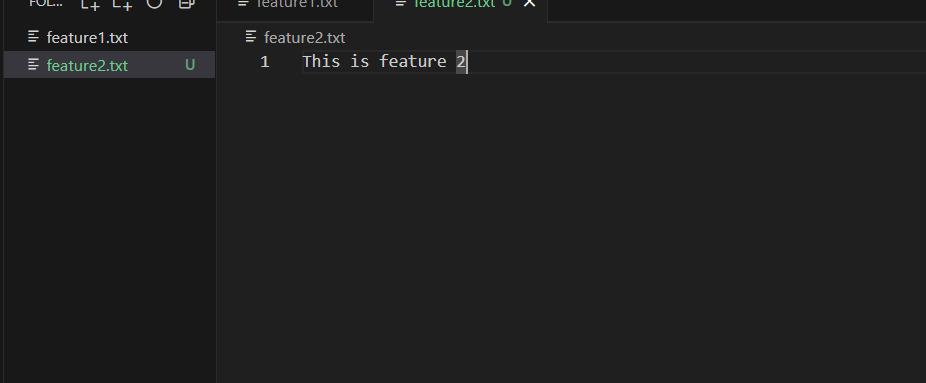


->Now create a new branch using the command : git branch newBranch

->then switch to the new branch using the command: git switch newBranch



->Now in this branch create a new file and write some text

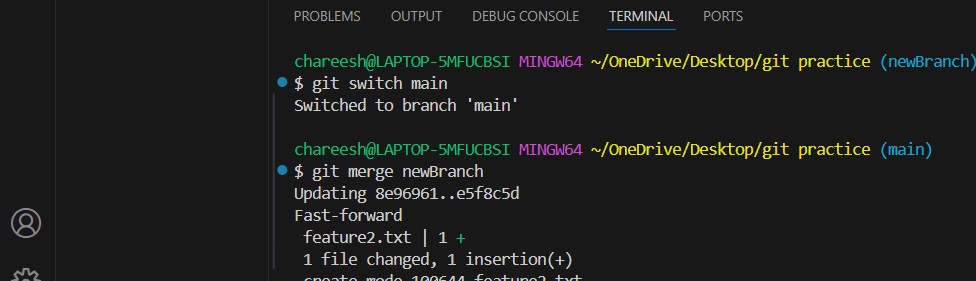


->Now execute the following command to add it to staging area and then commit stages just like we did for feature1 file in main branch

->Now switch back to main branch using the command : git switch main

->Now here you will not see the feature2 file since it was created in another branch so if you want to have that here you can merge it

-> to merge the branch execute the command : git merge newBranch

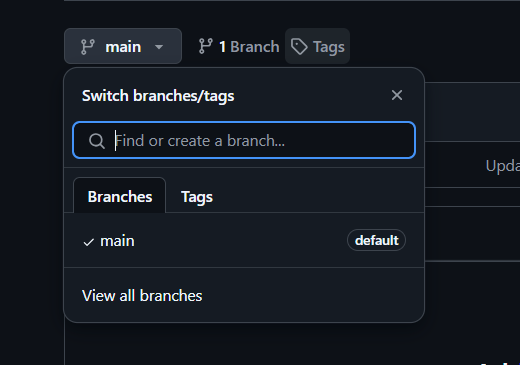


->After merging feature 2 will be available in the main branch also so this is have we do branching.

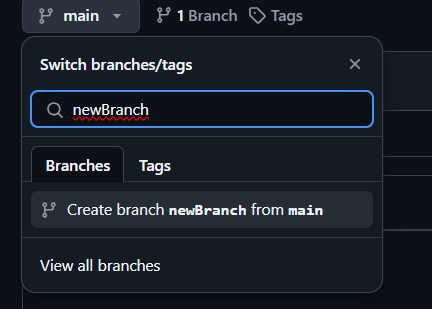
PERFORMING BRANCHING DIRECTLY THROUGH GIT HUB –

->first go to your repository where you want to create a new branch

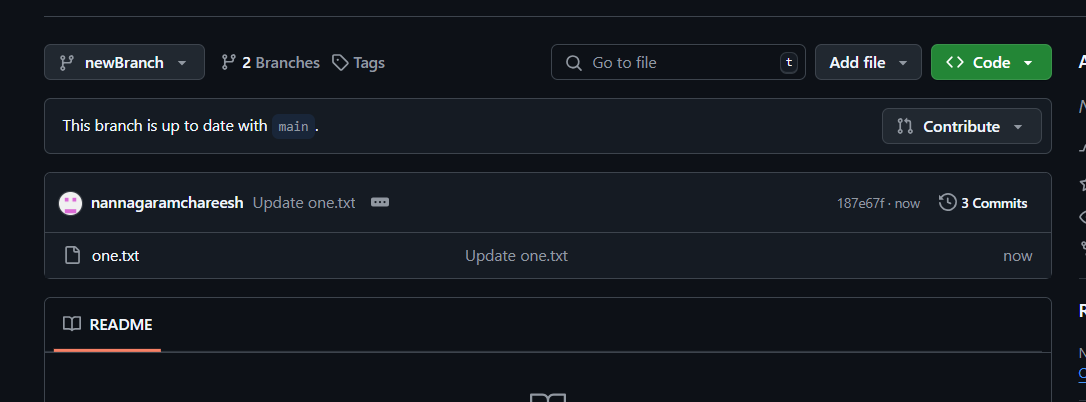
->Now on the top left you will find the current branch, select it



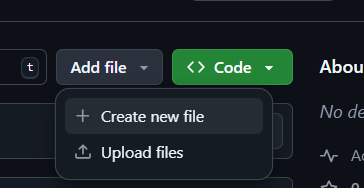
->Now write the branch name that you want to create in the search bar . If it does not exit then you will see an option saying Create branch newBranch from main.



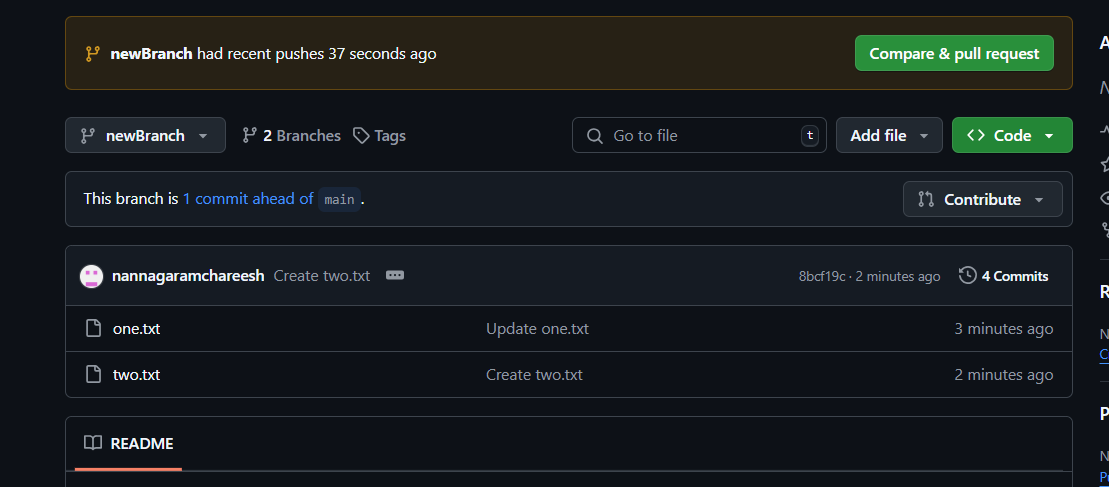
->After clicking on Create branch , the newBranch will be created and you will automatically get switched to the new branch and all the files in the main branch will be available as a copy in this newBranch

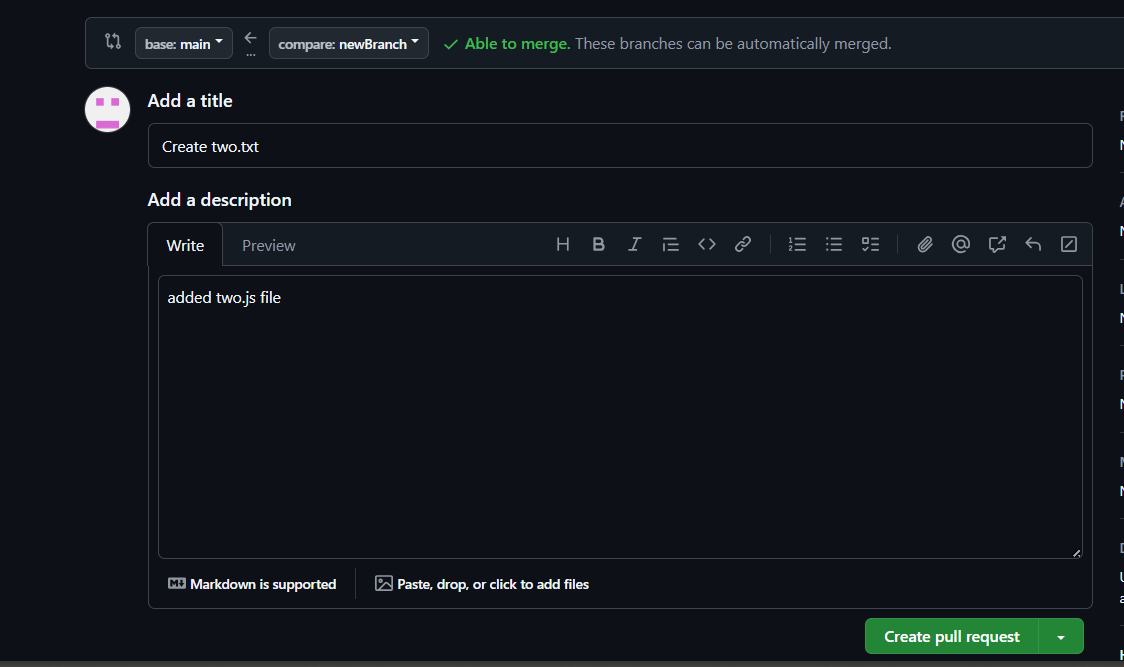


->Create a new file in this branch write something in it and commit changes

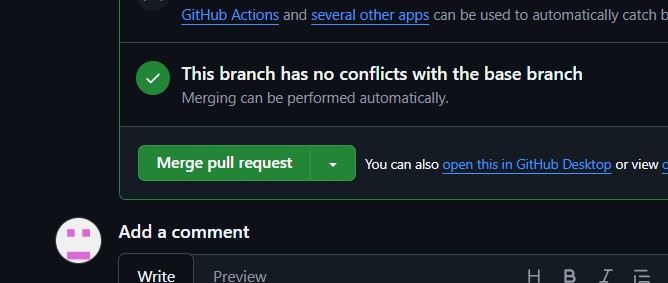


->Now to merge the changes in this newBranch to the main branch , click on compare & pull request



->Now add a commit message and click on create pull request

->Then click on Merge pull request inorder to reflect the changes made in newBranch to main branch



->Then click on confirm merge



->After this if you go to main branch you will see that the changes made in newBranch will be reflected in the main branch

So this is how branching can be done in git hub

CLONING A REPOSITORY-

Inorder to clone a repository we the git clone command.

->Go ahead and copy the repository of the project that you want to clone

->Now create a new folder and open it in vs code

->Now open terminal in vs code and execute the command : git clone repository\_url

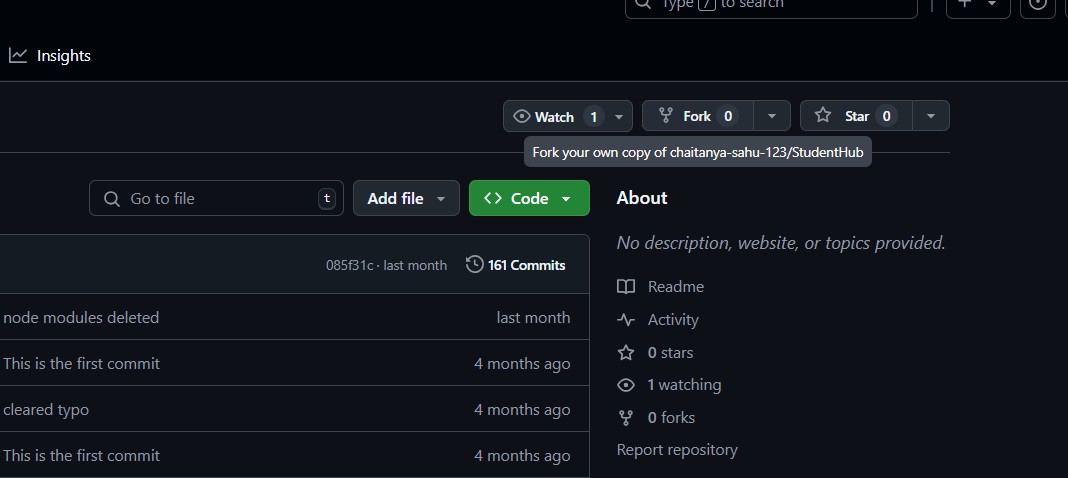
FORKING-

Forking in GitHub is the process of cloning a repository of someone else’s to your own GitHub account where you can modify it freely.

Steps to Fork and then make changes in you local repo-

->**Go to the repository**: Visit the GitHub page of the repository you want to fork.

->**Click "Fork"**: On the top-right corner of the repository page, you will see a "Fork" button. Click on it.



->Then you can clone it using git clone and make changes locally and add it back to the remote repository

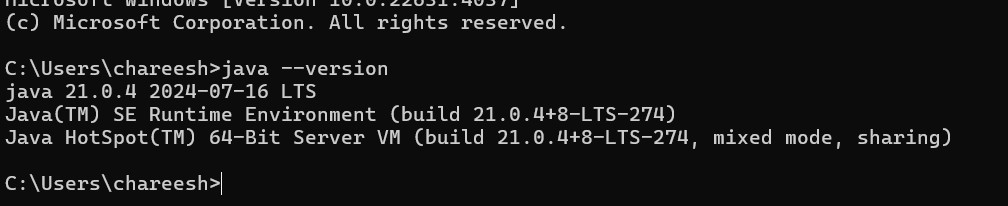
1. Jenkins installation and setup, explore the environment localhost8080 Steps to install jdk

->search for jdk download

->go to the first link of oracle and click jdk 21 and download x64 installer

->give the permissions and install it

->go to command prompt and click java –version to know the version of java



Steps to install and setup Jenkins

->search for Jenkins download

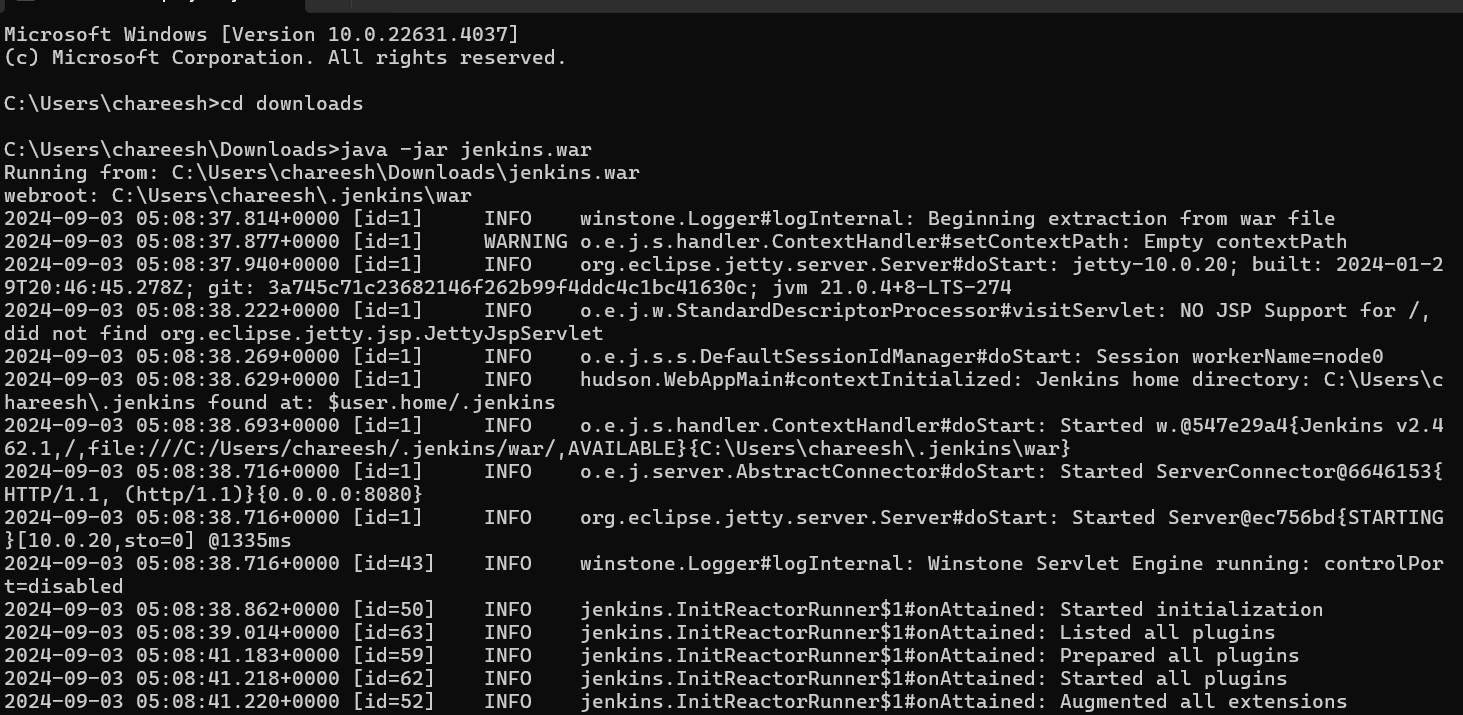
->go to the first link ([Download and deploy (jenkins.io)](https://www.jenkins.io/download/)**)**

### ->now click on java generic package and download it

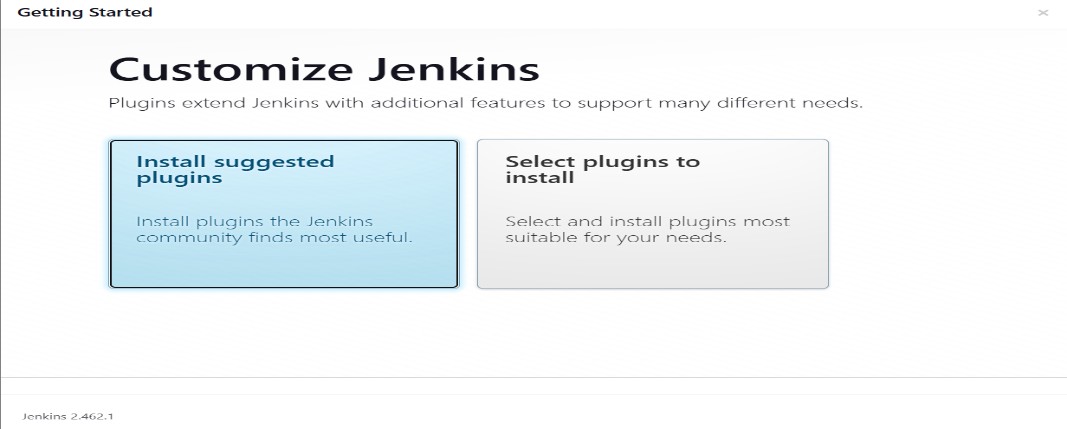
**->Now to run Jenkins go to command prompt and navigate to the folder where generic java package.war (Jenkins.war)is downloaded**

### ->Now run this command java -jar jenkins.war

**->Now a password will be generated ,Copy that password**



### ->Now open localhost:8080 and past the password that was generated

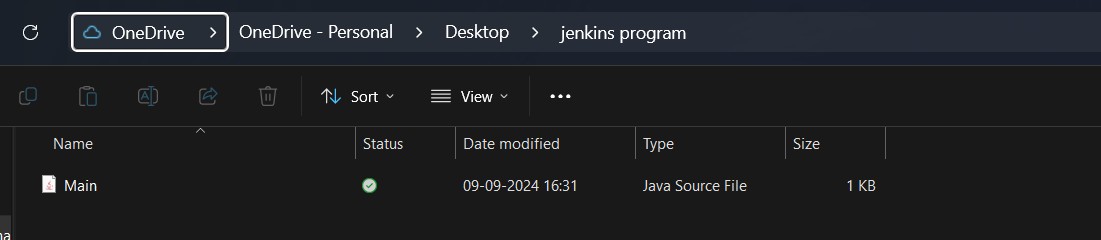
->Now click on install plug ins and install them

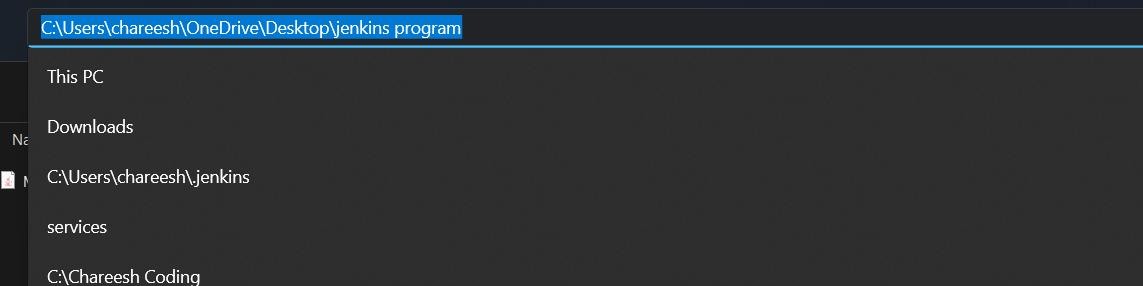


Now you can use Jenkins

1. Demonstrate continuous integration and development using Jenkins. Creating a new Job and running it-

->create a new folder and then open note pad and write a java program in it and save it in the new folder with .java extension. Now copy the new folder path where this java program is saved

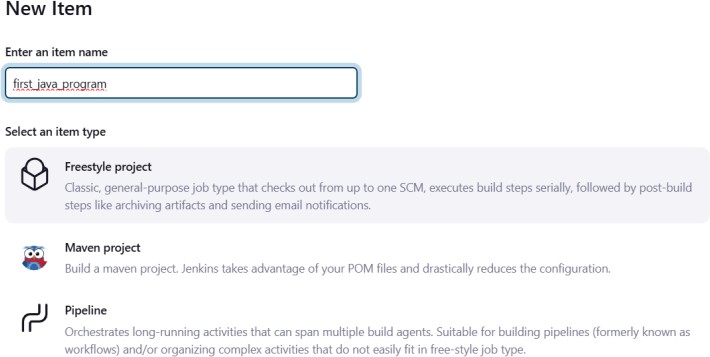




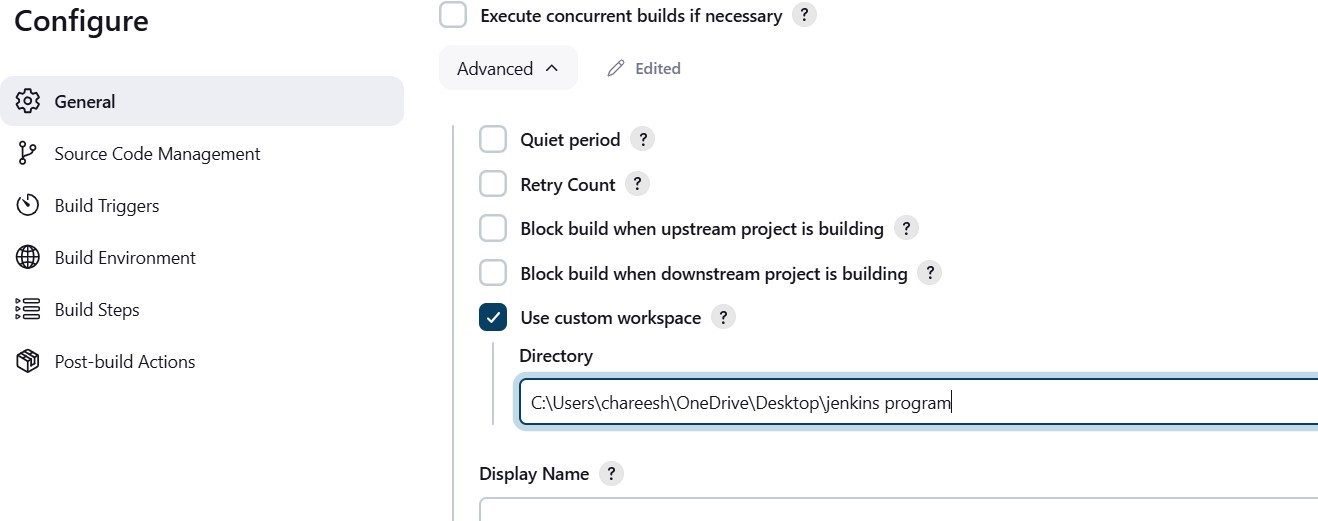
->open Jenkins and go to Dashboards and click on New item



->name the job and select freestyle project and click ok

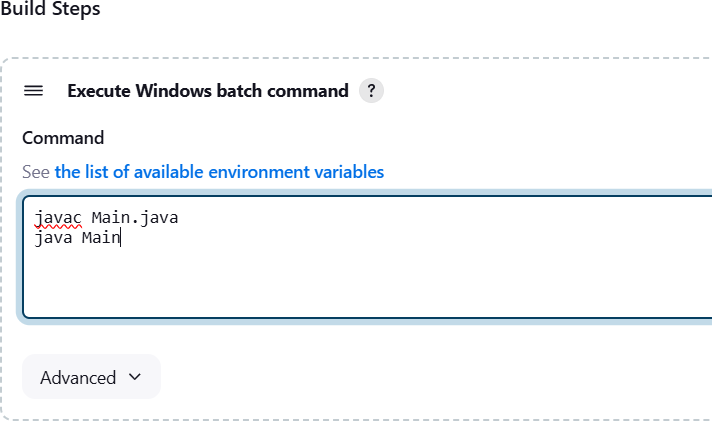


->Now in General section click on advanced and select use custom workspace and past the path that you copied



->Now in build steps section click on add build step and select **Execute Windows batch command. Now write**

### javac Main.java

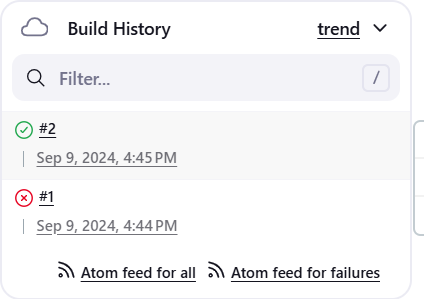
**java Main (to run the java program)**

### ->Now click on apply and then save

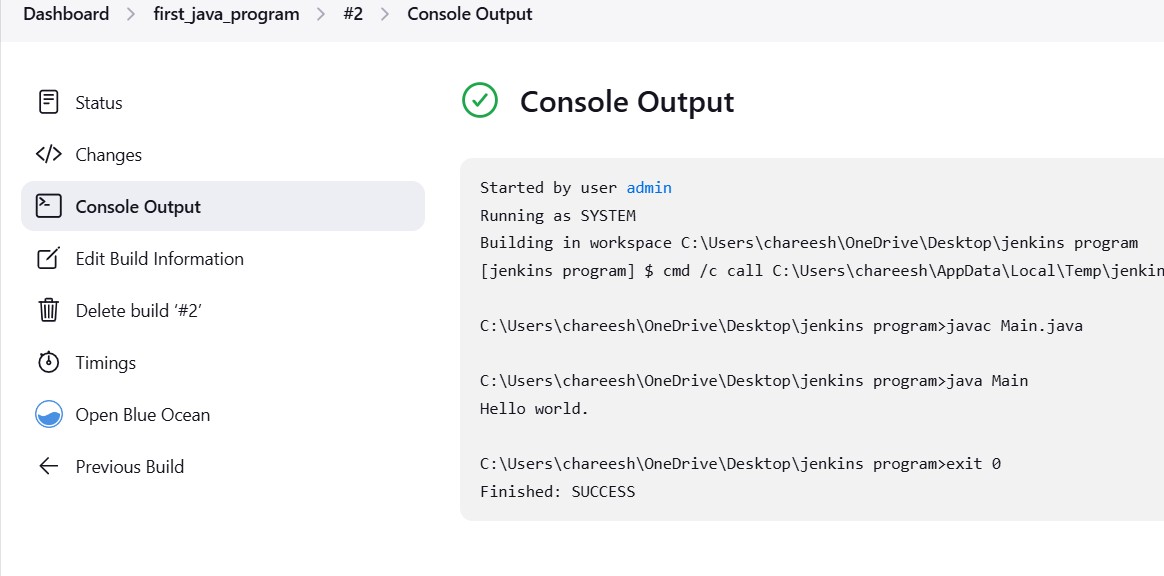
**->Now click on Build Now**



### ->If you get a tick mark then your job was finished successfully



**->click on that tick mark to see the output of the program**

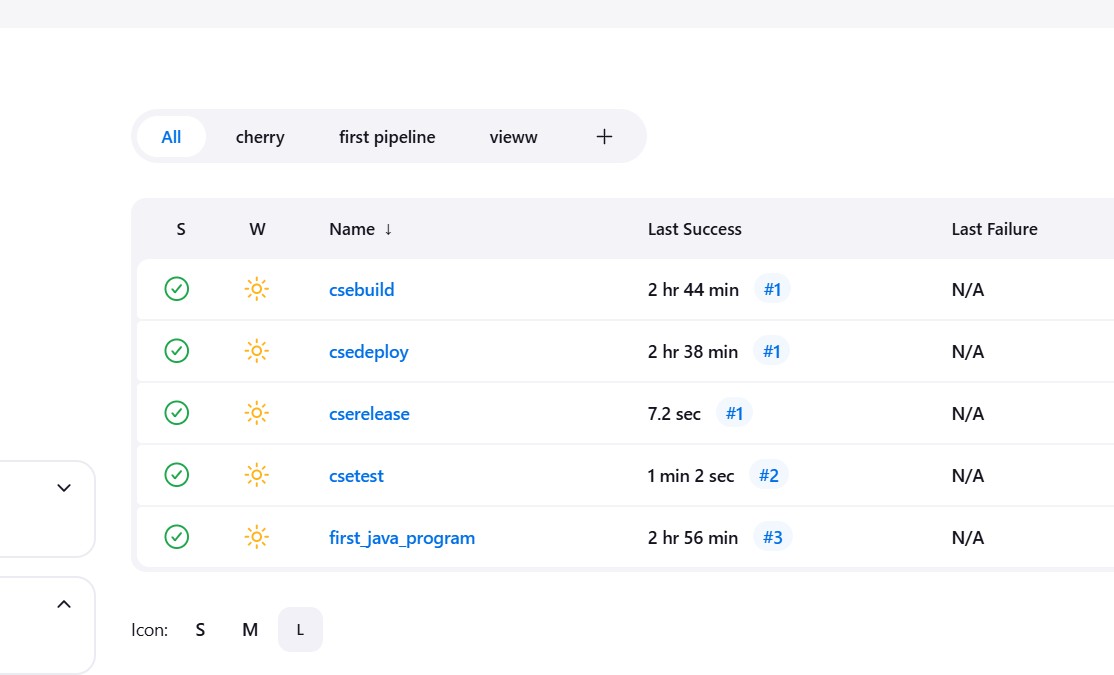


### Creating pipe line-

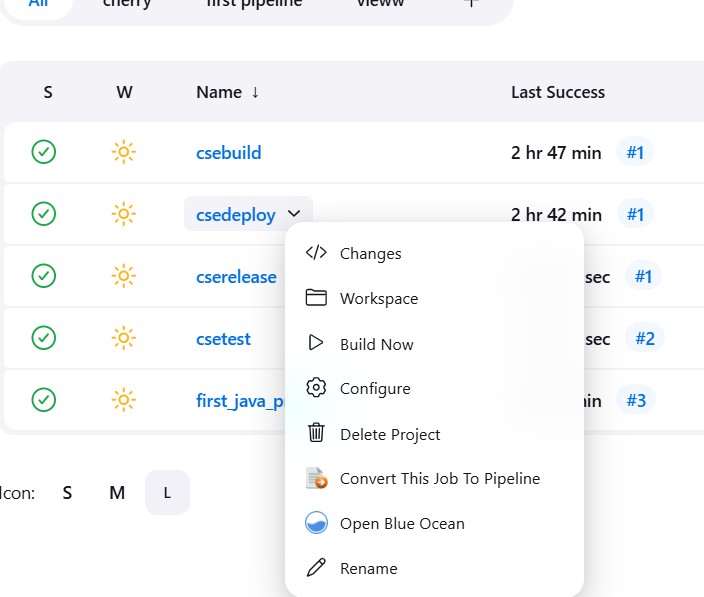
**->first we need to create 4 jobs – csebuild,csedeploy,csetest,cserelease**

### ->click on New item and name it as csebuild and select freestyle project and click ok.Dont add any path just click on add build step and write echo ‘Build successful’ and click and apply and then save.Now click on Build Now .

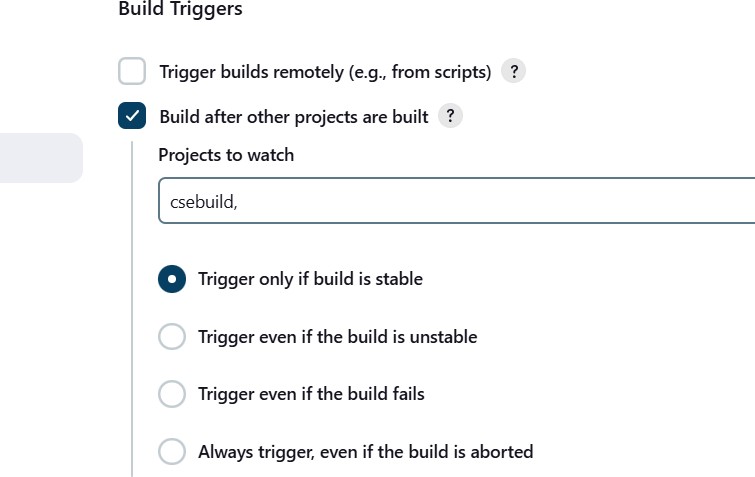
**->follow the same process and create separate jobs for csedeploy ,csetest and cserelease**



### ->Now select csedeploy and click on config



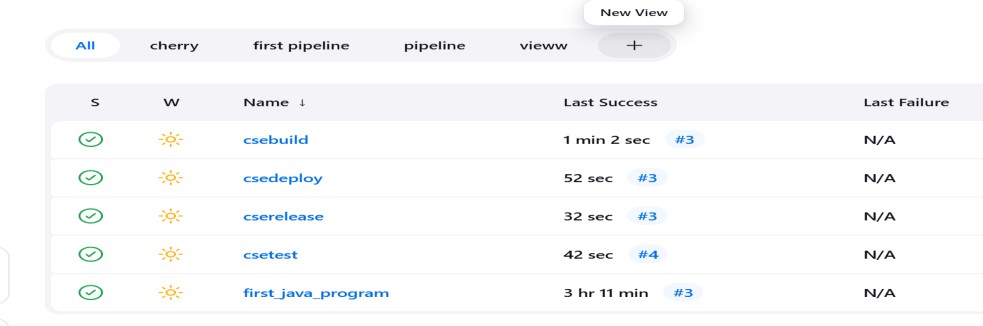
**->Now select build after other projects are build and write csebuild,then apply and save**



### ->Now go back and select csetest and click on config and then select Build after other projects are built and write csedeploy then apply and save.

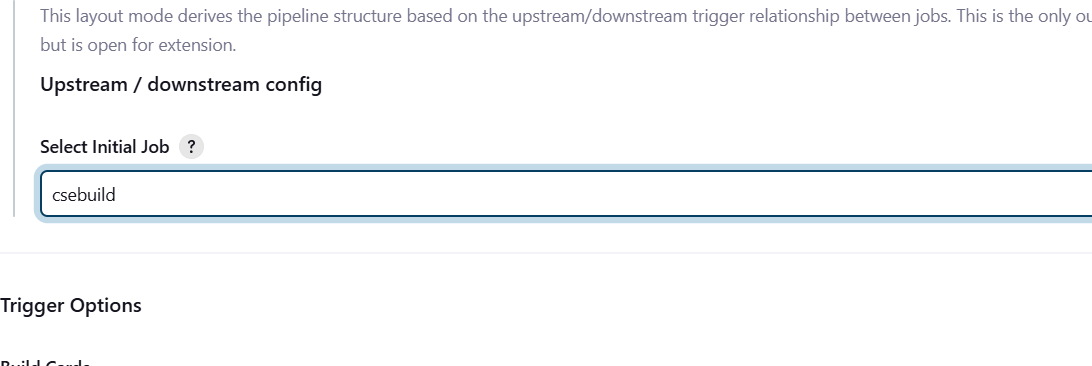
**-> Now go back and select cserelease and click on config and then select Build after other projects are built and write csetest then apply and save.**

### Now lets build a pipeline view

**->click on new view**

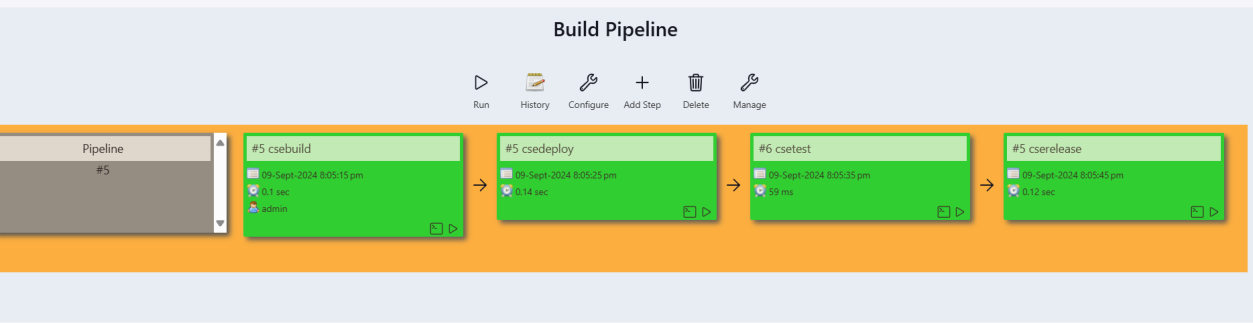
### ->Give a name and select Build pipeline view and then click on create

**->Then select csebuild as initial job**



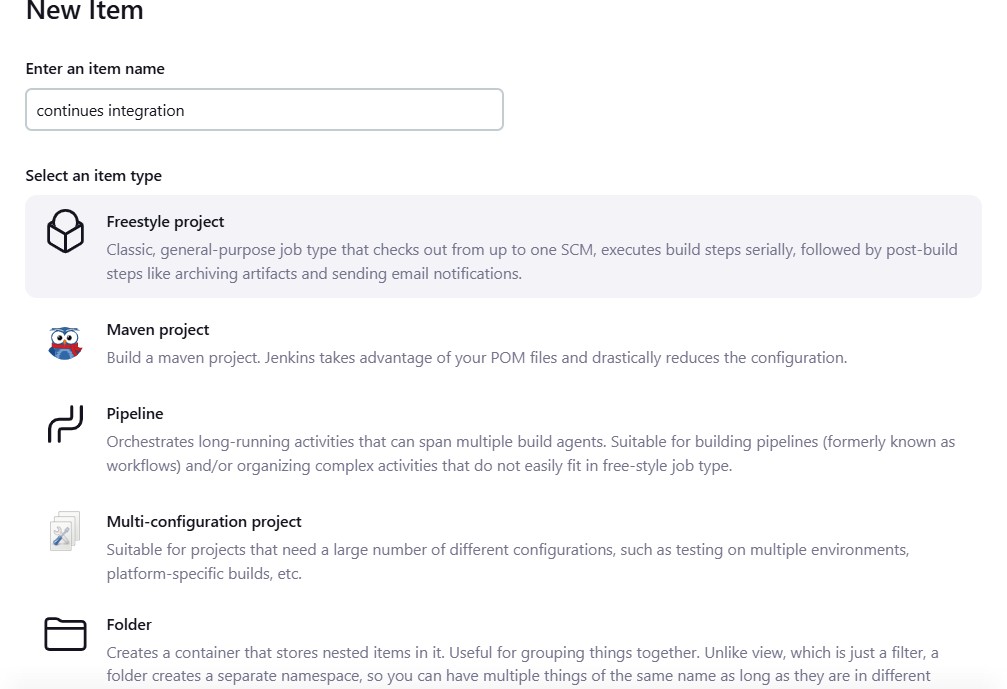
### ->Now click apply and then save

**Output-**



### Now lets implement continues integration –

**->go to jenkins and click on new item and then select free style project and click ok**



->create a java file and write a simple program in it and then create a

repository and add this java file to the repository and copy the repository

->now in the source code management section select git and past your repository url

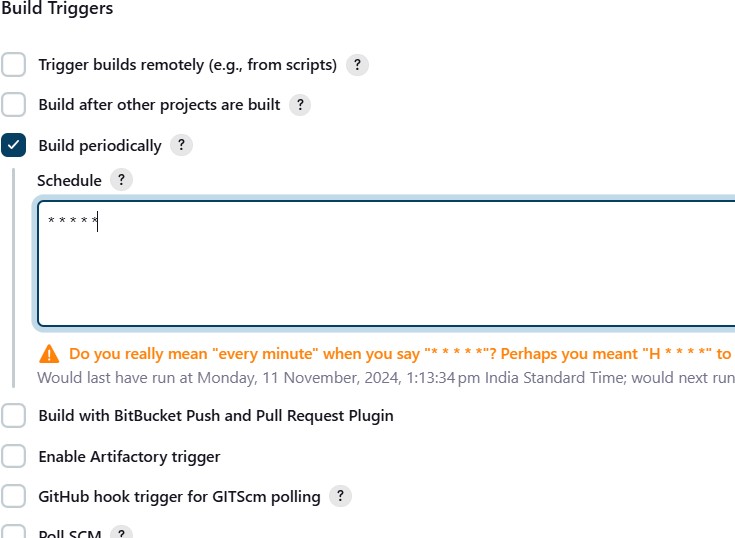


# ->Now in build steps section select execute windows batch command and to compile java write the command



**->now in the build triggers section select build periodically and in the schedule past \* \* \* \* \*.**

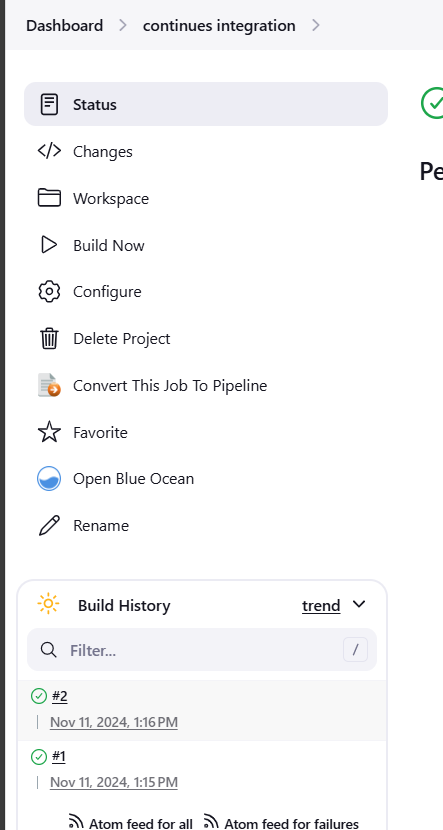
# This will execute the file every 1 minute



**->Now apply and save it**

# ->Now click on build now

**->Now every minute jenkins will build the file enabling continues integration**



# DevOps node js project

-> first install all the dependencies

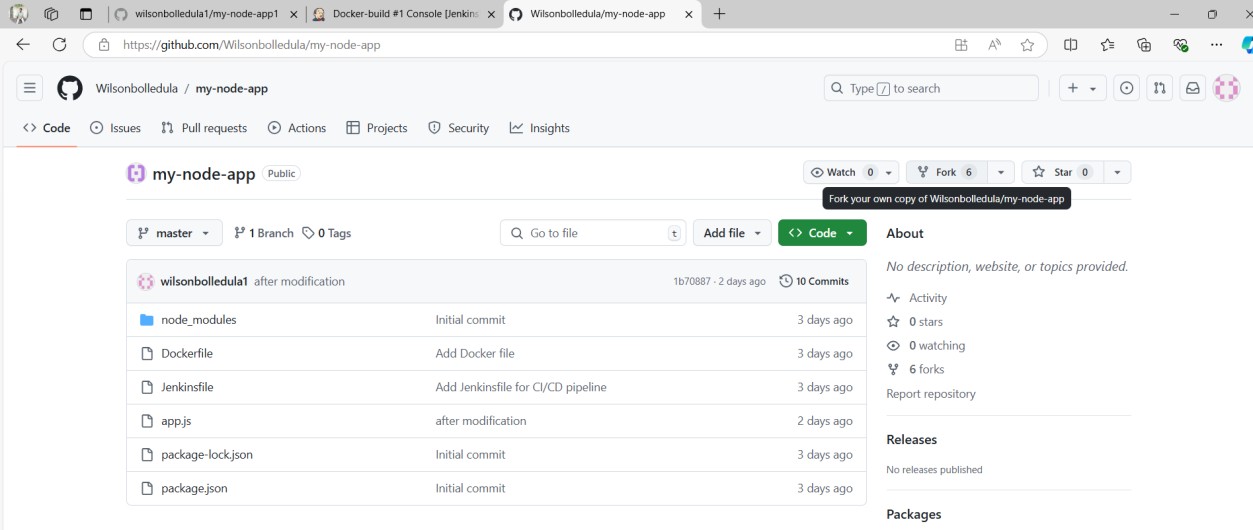
-jdk,git,docker,Jenkins

->make sure all the dependencies are installed and working properly.

->login your github account in the browser.

->click on new tab, in the new tab paste <https://github.com/Wilsonbolledula/my-node-app.git>

->after pasting the link fork the repository



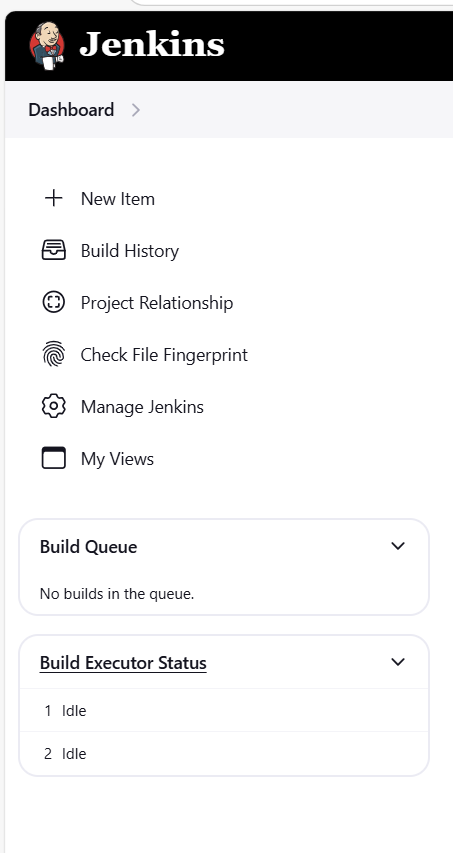
->copy the repository HTTPS url from the code



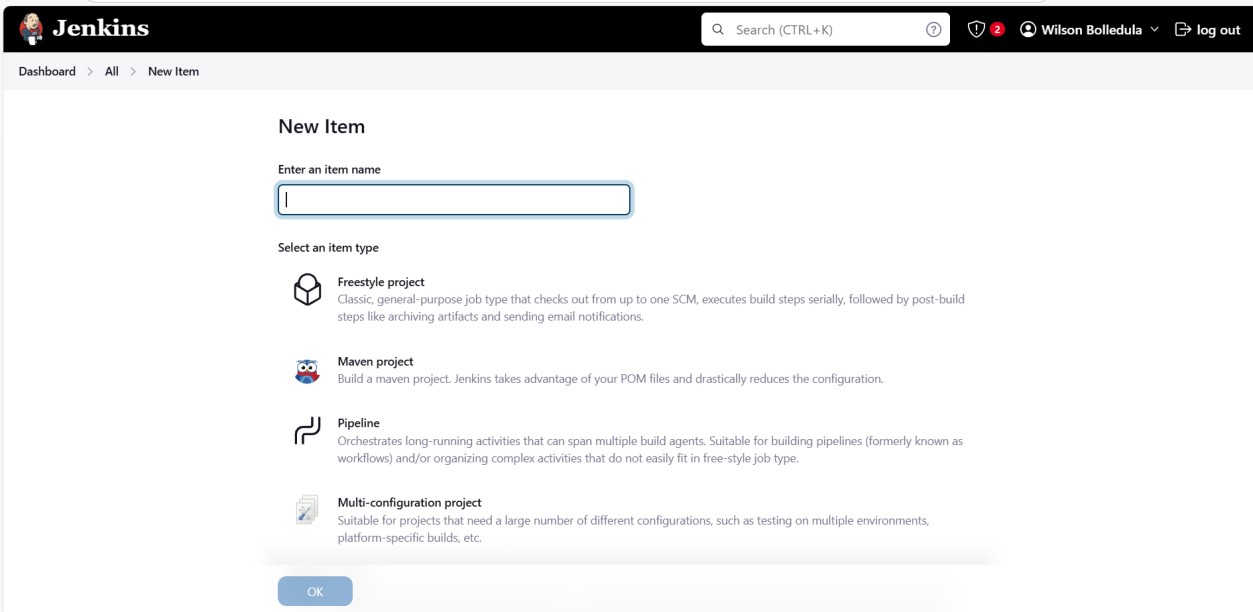
**Jenkins Project:**

->open the Jenkins in new tab and type “localhost:8080” or “localhost:8085”which you have kept the port number while installing the Jenkins.

->after opening Jenkins, click on new item which is appearing in the leftside

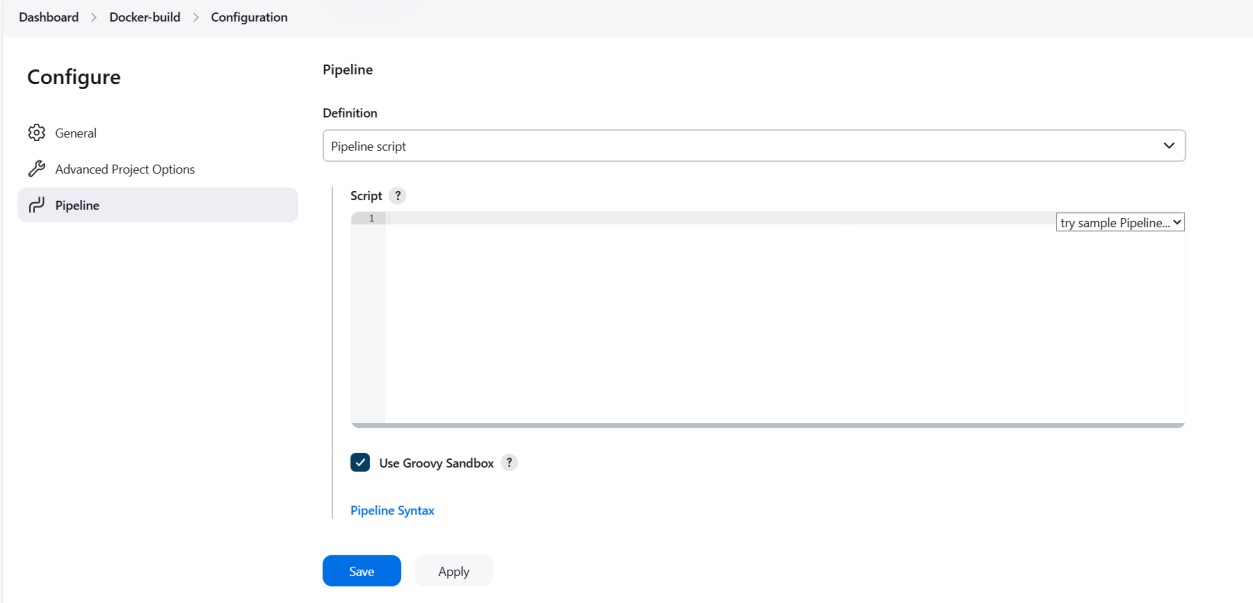


->after selecting new item you are able to see like this



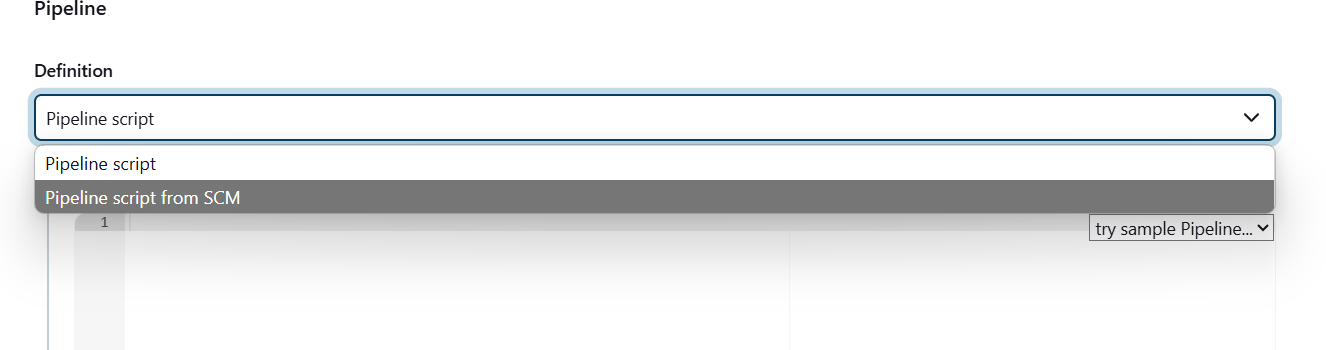
->now in the box enter any name for your Jenkins project and select pipeline project, click ok.

->after clicking ok you will get like this



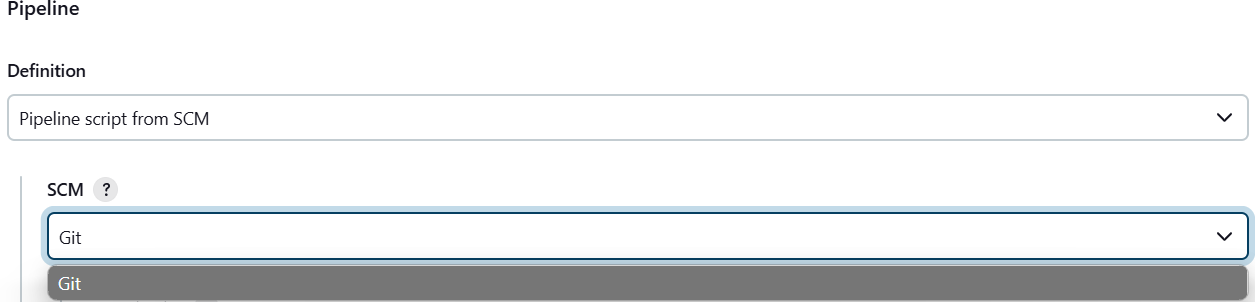
->Select Pipeline in the configure

->click on Pipeline script

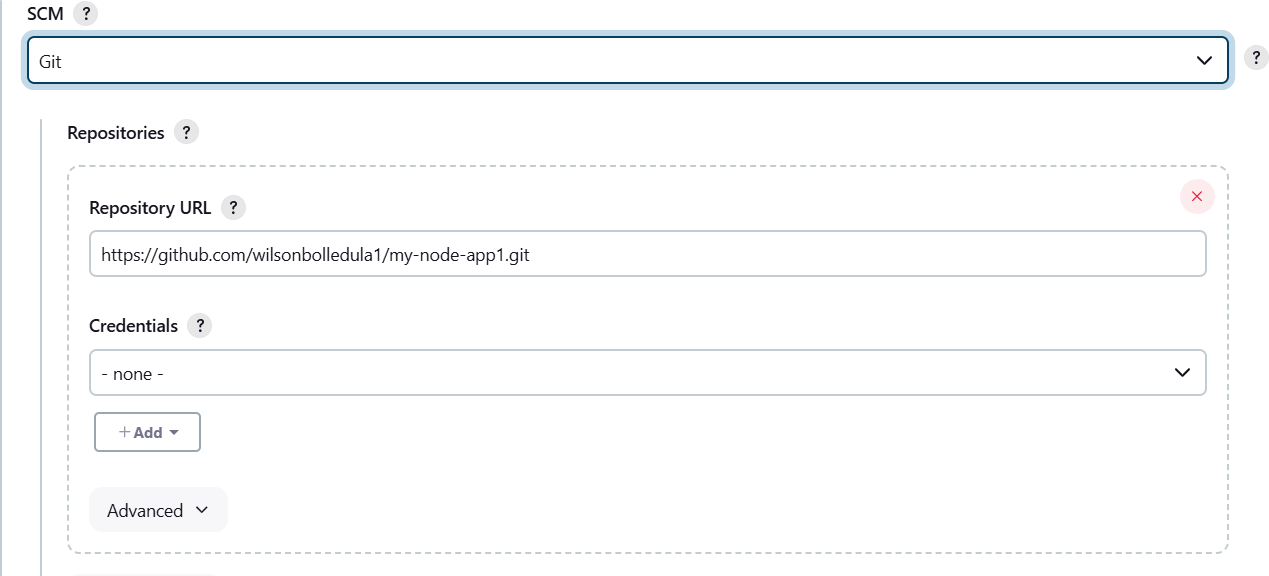


->select “pipeline script from SCM”

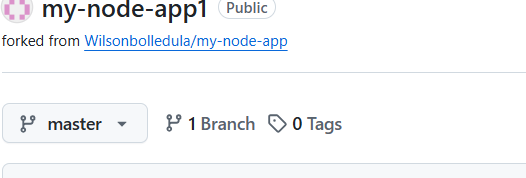
->in the SCM select git



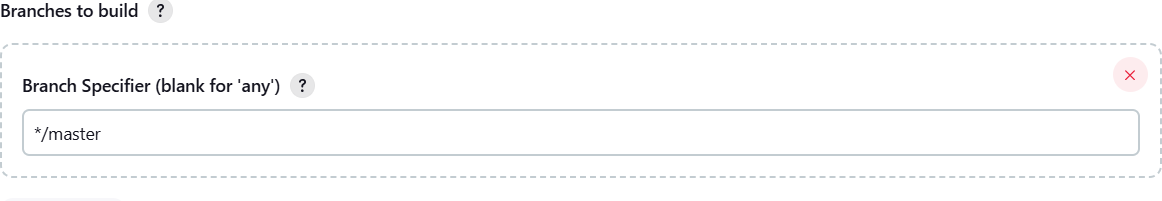
->after selecting the git paste your repository link which is previously forked and saved



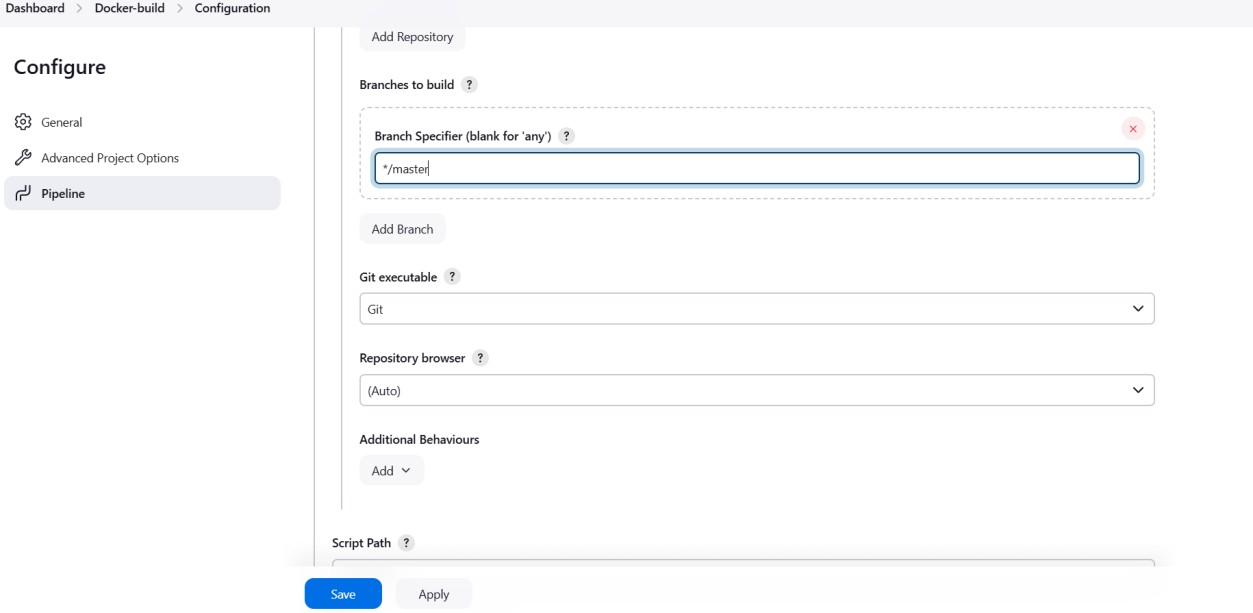
->specify your branch like “\*/master” or “\*/main” which is showing in your github repository



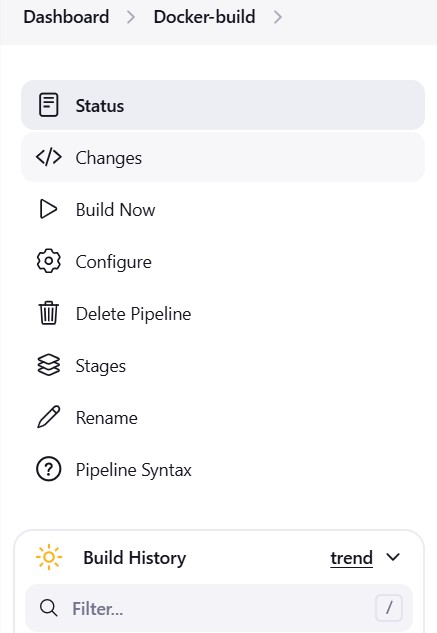
->specify here



->after that clock on Save



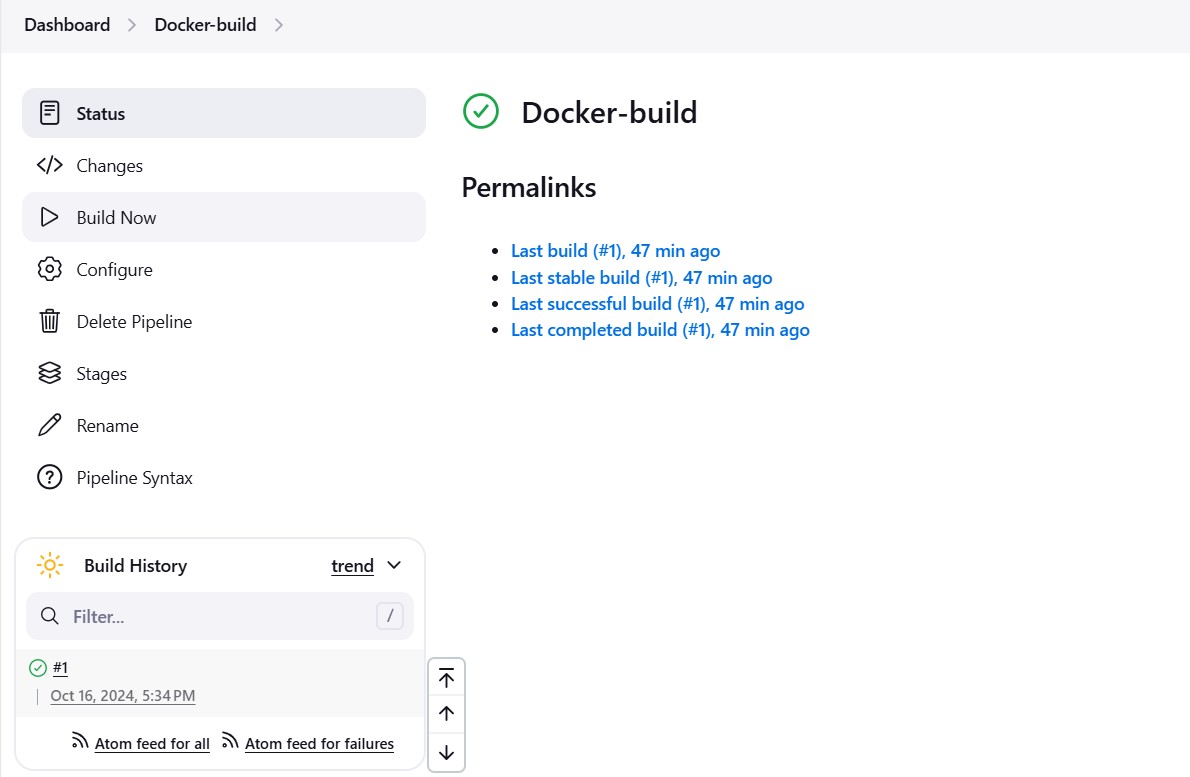
->after that you will able to see this on your screen



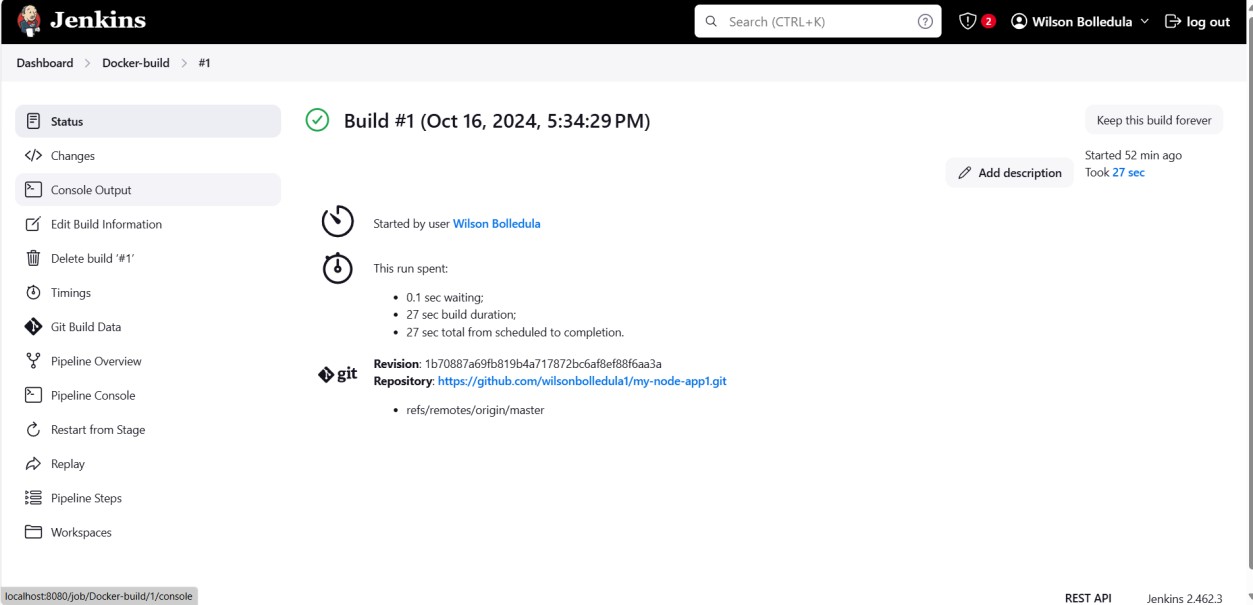
->Note:Before clicking build you should make sure that docker desktop application is opened and running in the background

->click on BuildNow

->after build you will get success mark and build number in the bottom

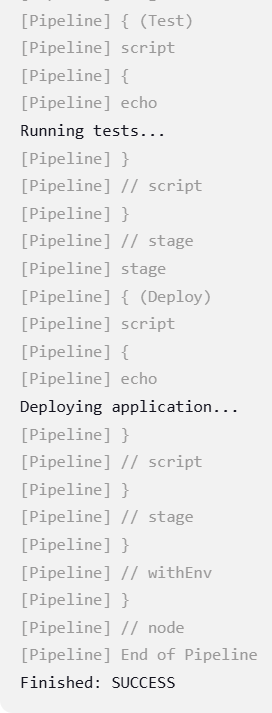


->click on that #1

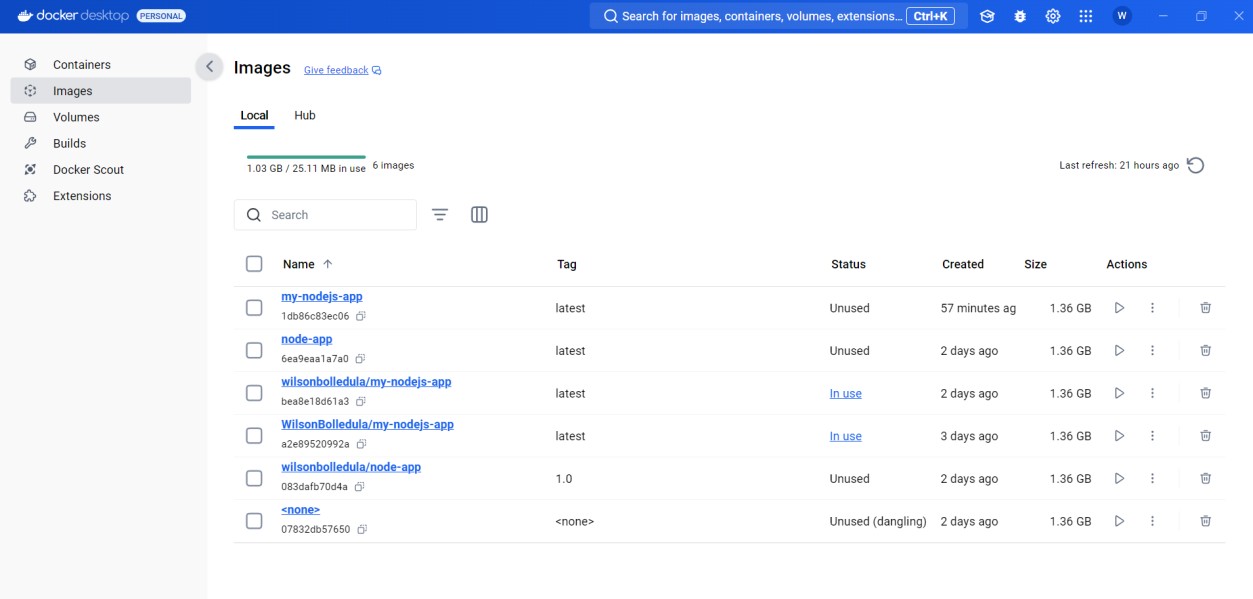


->on the left side you will able to see console outpot, click on it

-> in the bottom you should get success like



->you can see the docker iamge in the docker desktop application



**10 |** P a g e **Prepared by Faculty M Shiva Kumar and 22 batch CSE**

**Devops Maven project**

Prerequisites:

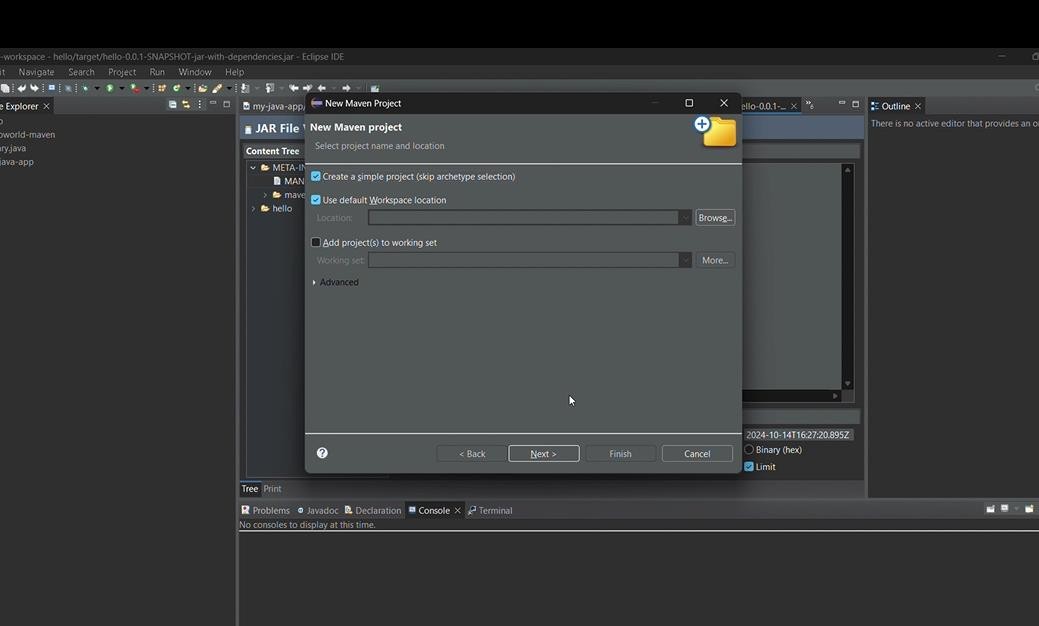
- Eclipse IDE,GitHub,Jenkins,Docker,Maven

->open eclipse and go to File > New > project

->Select Maven project from the wizard,then clicl next.

->Ensure Create a simple project(skip archetype selection) is checked,then click Next.

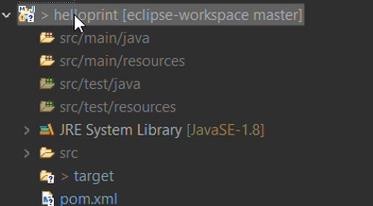
->



->Enter the following:

1. **Group Id:** com.example
2. **Artifact Id:** hello-world
3. **Version:** 1.0-SNAPSHOT
4. **Packaging:** jar Click **Finish**.

->Now,a Maven project will be created in Eclipse

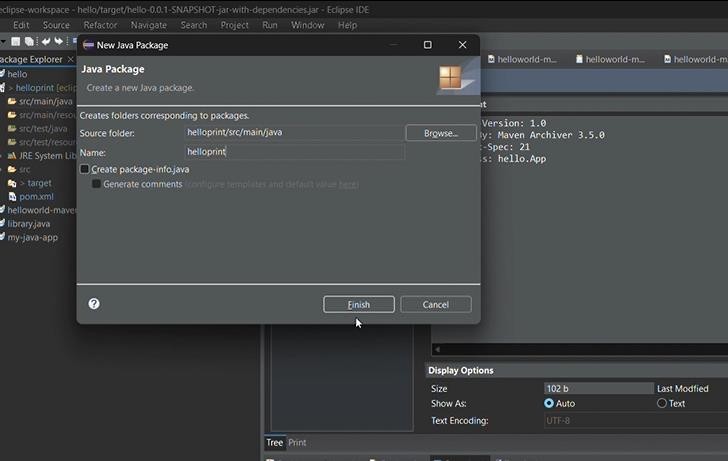


Step 2:Write a Hello world program.

->navigate to src/main/java folder inside your project in eclipse.

->create a new package com.example inside src/main/java.

->



- create a new java class App.java inside the com.example package: Code:

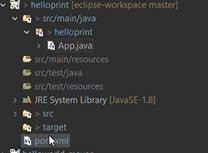
package com.example; public class App {

public static void main(String[] args) { System.out.println("Hello, World!");

}}

Step 3: Update pom.xml

->open pom.xml file in the root of your maven project.



->ensure it contains the following dependencies

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

->add the required plugins

<project xmlns="<http://maven.apache.org/POM/4.0.0>"

xmlns:x[si="h](http://www.w3.org/2001/XMLSchema-instance)ttp:[//www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0>

[http://maven.apache.org/xsd/maven-4.0.0.xsd">](http://maven.apache.org/xsd/maven-4.0.0.xsd)

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>helloworld-maven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<!-- Maven Assembly Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

<version>3.3.0</version>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass>com.example.App</mainClass>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

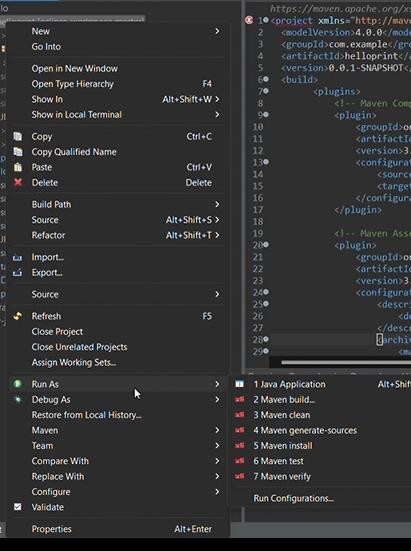
</build>

</project>

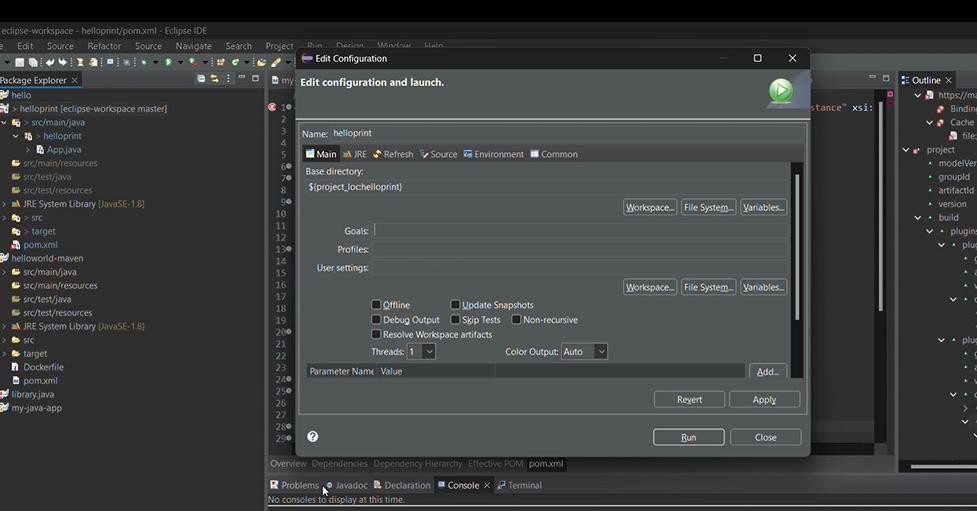
Step 4 :Build the Maven project Locally

->right-click on the project in eclipse

->select runs as > Maven Build……

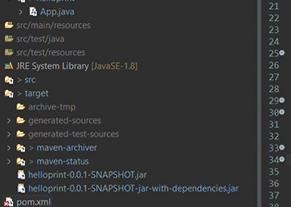


->In the Goals fiels,type clean package .



->click run.

-> this will build the project and create a JAR file in the target folder.



Step 5: Push the Maven Project to Github.

->Goto github and click new repository.

->name it hello-world-mavin

->copy the remote url for the repository

->Initialize Git in your project folder using terminal

->navigate to the directory. git init

git remote add origin https://github.com/your-username/hello-world-maven.git git push -u origin master

git add .

git commit -m “initial commit” git push -u origin master

Step 6:Set up Jenkins

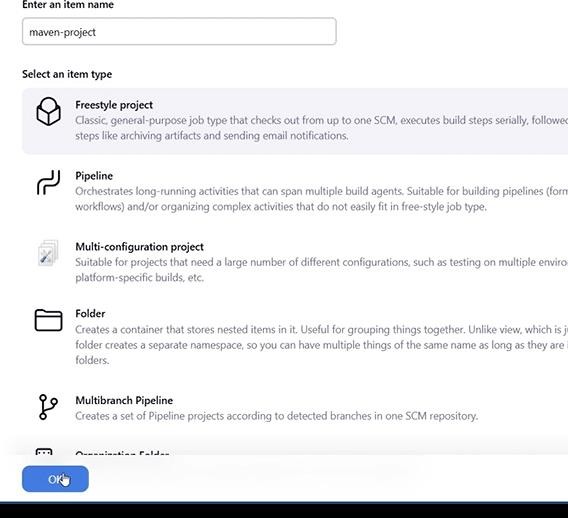
->open jenkins in your web browser and log in.

->Install the necessary Jenkins plugins:

1. Go to **Manage Jenkins** > **Manage Plugins**.
2. Install **Git**, **Maven**, and **Docker Pipeline** plugins.

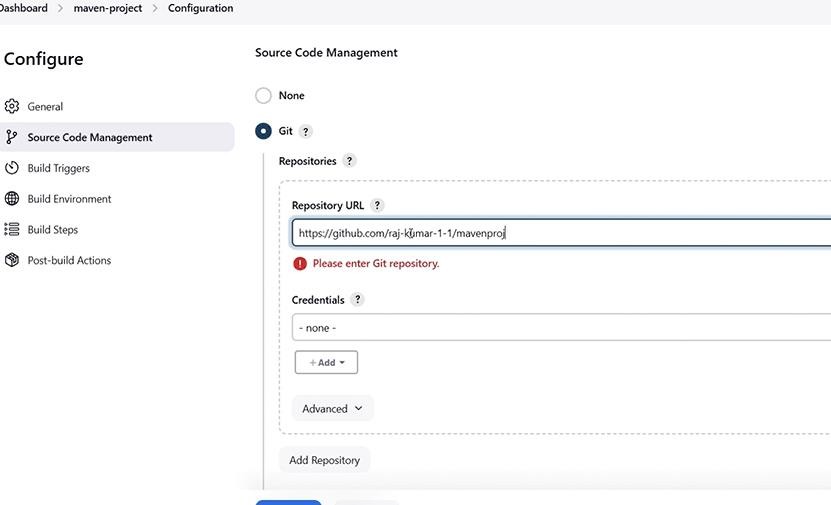
->Go back to the Jenkins dashboard and click on **New Item**.

->Select **Freestyle Project**, name it hello-world-maven, and click **OK**.



->In the **Source Code Management** section:

1. Select **Git**.
2. Add the **GitHub URL** of your repository.

3.

4. If necessary, add your GitHub credentials.

->In the **Build** section:

1. Click on **Add Build Step** and select **Invoke Top-Level Maven Targets**.
2. Set the **Goals** to clean package.

->save the configuration.

Step 7:Configure Docker in Jenkins

->Install Docker on the machine where Jenkins is running if it's not installed yet.

1. To verify Docker is running, use docker --version and docker ps in your terminal.

->In Jenkins, navigate to **Manage Jenkins** > **Configure System**.

->Scroll to the **Docker** section and ensure the Docker configuration is correct.

->In your **Jenkins project** (hello-world-maven), add a **Post-build Action** to create a Docker image

->Go to the **Post-build Actions** section.

->Add a build step **Execute shell** and add the following commands to build and run your Docker image:

docker build -t hello-world-maven-app .

docker run -d -p 8080:8080 hello-world-maven-app

->**Save** the Jenkins configuration.

Step 8 Create a Dockerfile in Your Project

->In the root of your project, create a file named Dockerfile.

->in that file add

FROM maven:3.6.3-jdk-8 AS build

WORKDIR /app COPY . .

RUN mvn clean package FROM openjdk:8-jre

WORKDIR /app

COPY --from=build /app/target/hello-world-1.0-SNAPSHOT.jar /app/hello-world.jar

# Run the application

ENTRYPOINT ["java", "-jar", "/app/hello-world.jar"]

->push the Dockerfile to github.

->in Jenkins in the item,click on build now this will create a docker image.remember while building always start the docker desktop.

->You can verify that the Docker image has been created by running the following command on the Jenkins server:

docker images

-> To check if the application is running in a container: docker ps

->The application should be running on port 8080.

->you can also see the images in the docker desktop.

**10 |** P a g e **Prepared by Faculty M Shiva Kumar and 22 batch CSE**

# DevOps java project

-> first install all the dependencies

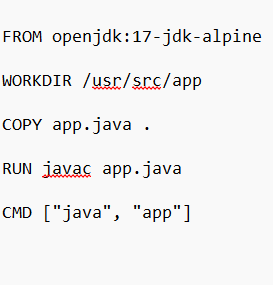
-jdk,jvm,docker,Jenkins,eclipse

->create a folder which will have the code

->open that folder in the vs code

->then create a new java file in that and write a piece of code.

->now in the same folder create a Dockerfile .

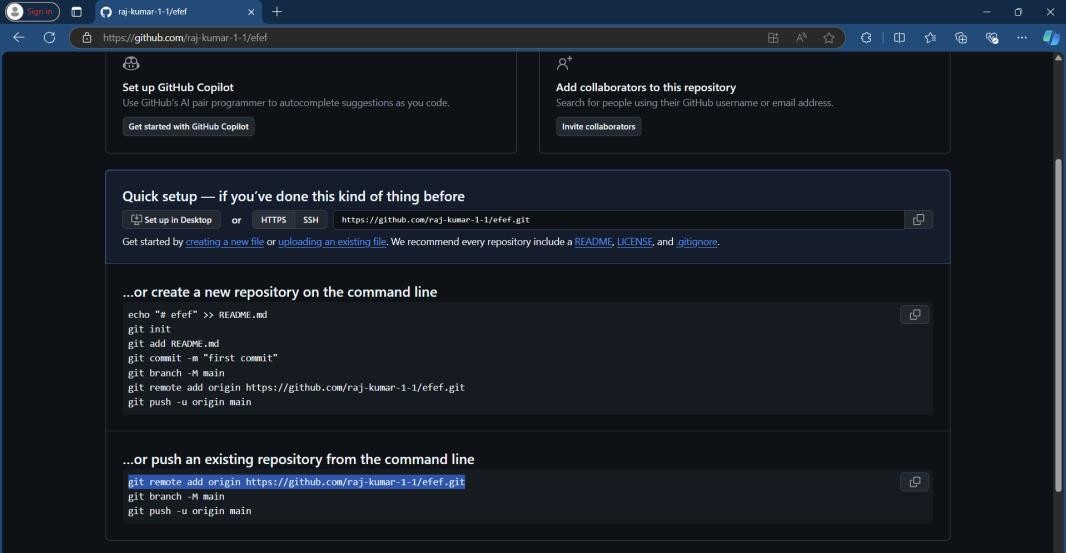
->in that docker file write the command to run the java file.

->

->save it , And push all the file in to github

# ->pushing files to git hub.

->first create a repository in the git hub and after creating the repository copy the git remote



->open the terminal ,navigate to the folder and write the commands to push files. git init

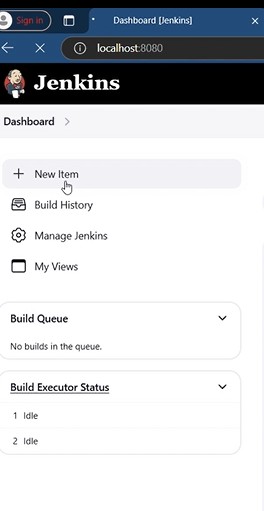
git remote add origin https://github.com/raj-kumar-1/<repository-name> git add .

git commit -m “initial commit” git push -u origin master

->after successfully pushing all the files into github ,verify in the github whether all the files have been added or not

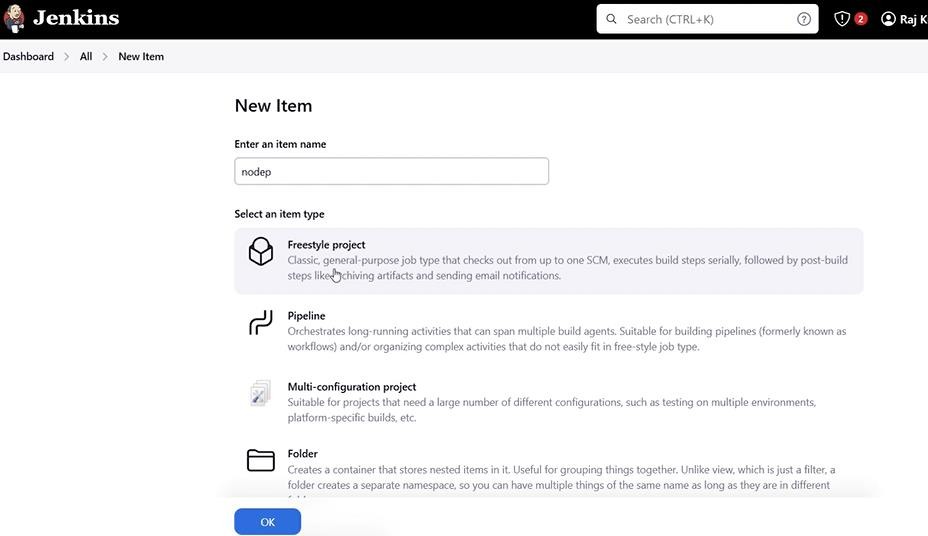
->now open Jenkins in localhost:8080

->click on the new item



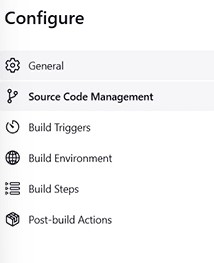
->enter the item name and select the freestyle project and click on ok.

->



->after clicking ok,a configure page will be opened

->click on the source code management



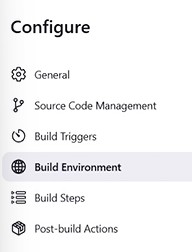
->select git and give all the required details like your repository(which we have created earlier) link and branch .

->if you have the file not in the master branch then add new branch



->\*/<your\_branch>

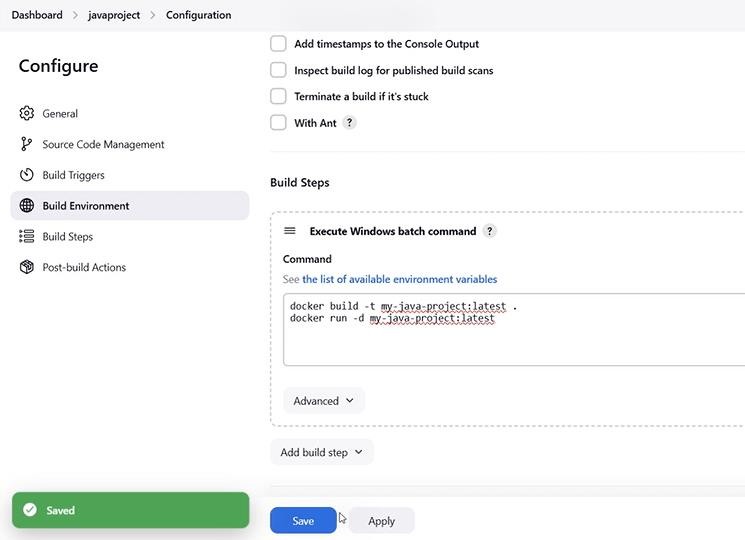
->now click on the build steps



->click on the add build step and select Execute Windows batch command

->and write the docker commands to create the docker image.

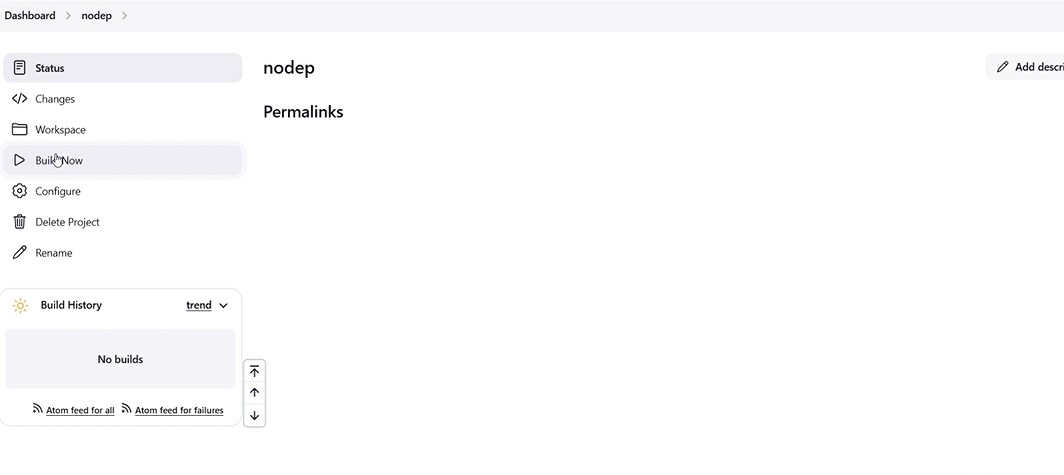
->



->later while running the image in the docker desktop if a error regarding the port,try changing the port numbers here like 8084:3000

->click apply and save.

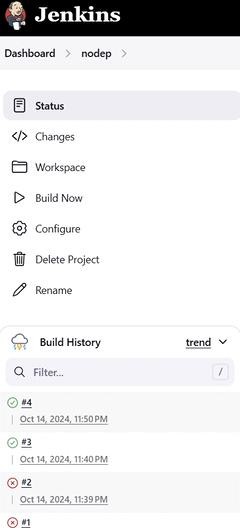
->after clicking



->make sure that you have started the docker .

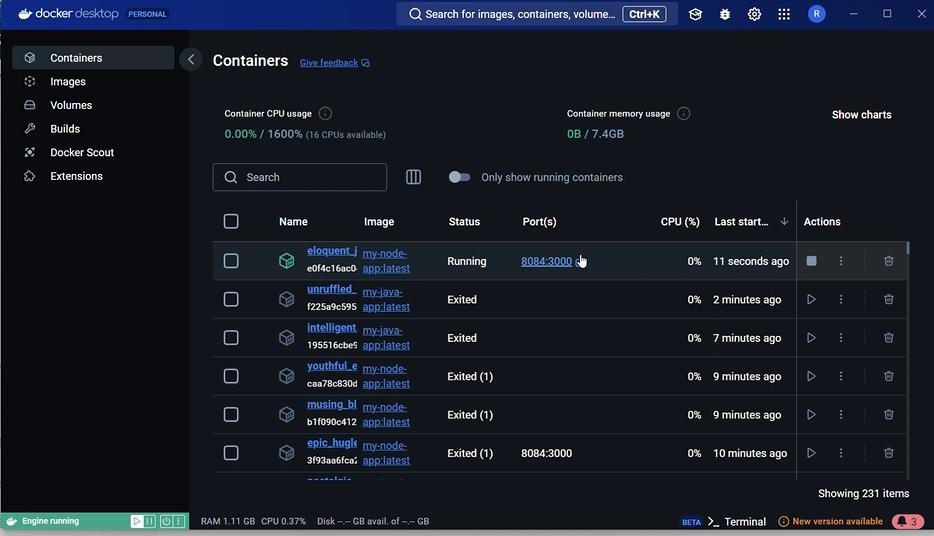
->click on build now

->after a successful build you will see the tick mark



->now open the docker

->



->search the container that is newly created.

->run the container

->after the successful run.

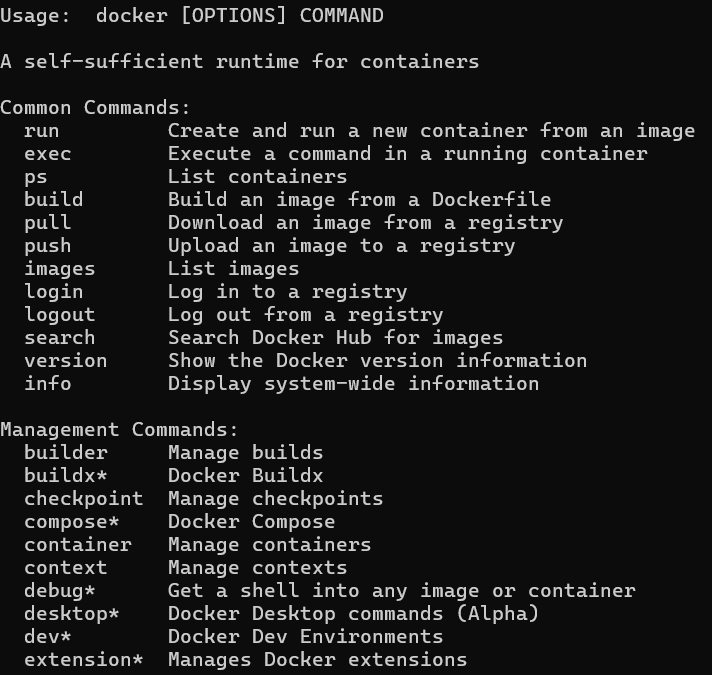
->you will be seeing the output printing hello world.

**WEEK 6**

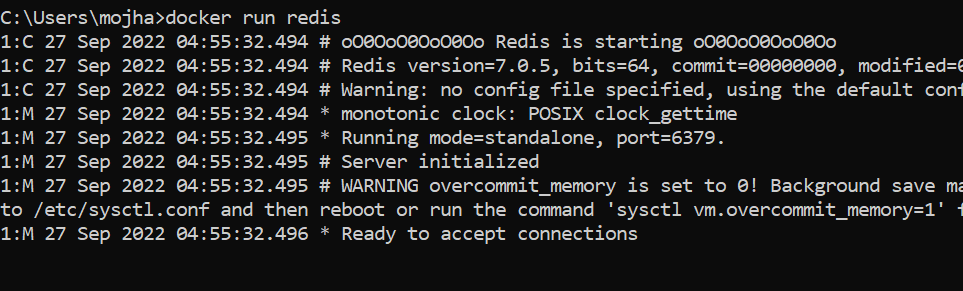
**Aim:** Explore Docker commands for content management.

# DOCKER COMMANDS:

1. **Get help regarding Docker commands**: docker help

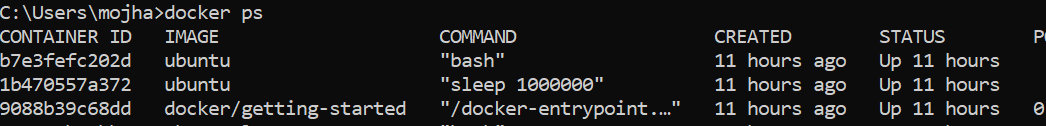


## **Docker run command**: docker run <image\_name> This command is used to run a container from an image.



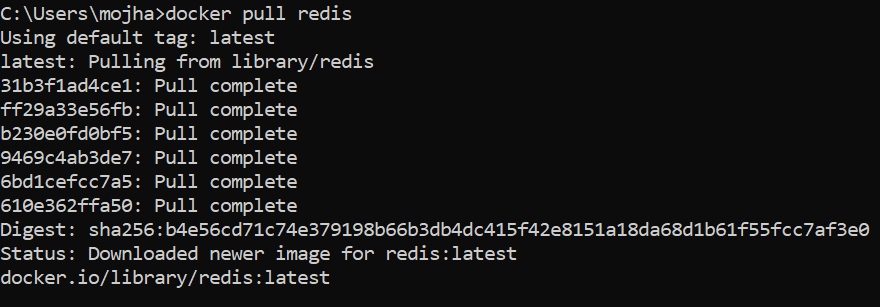
1. **Docker ps**: docker ps

## Lists all the running containers



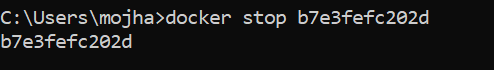
1. **Docker Pull:** docker pull <image\_name>

## This command allows you to pull any image which is present in the docker hub.



1. **Docker Stop:** docker stop <container\_ID>

## This command allows you to stop a container if it has crashed or you want to switch to another one.



1. **Docker Start:** docker start <container\_ID>

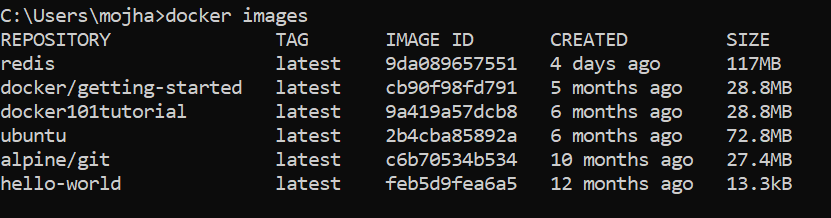
## Suppose you want to start the stopped container again, you can do it with the help of this command.

1. **Docker rm:** docker rm <container\_name or ID> To delete a container.

docker remove an image

1. **Docker Images:** docker images

## Lists all the pulled images which are present in your system.



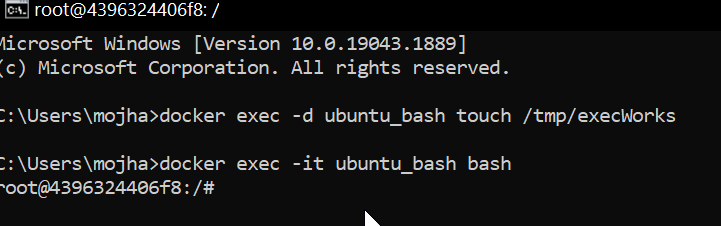
1. **Docker exec:** docker exec <flag> Some important flags:

## -d flag: for running the commands in the background.

-i flag: it will keep STDIN open even when not attached.

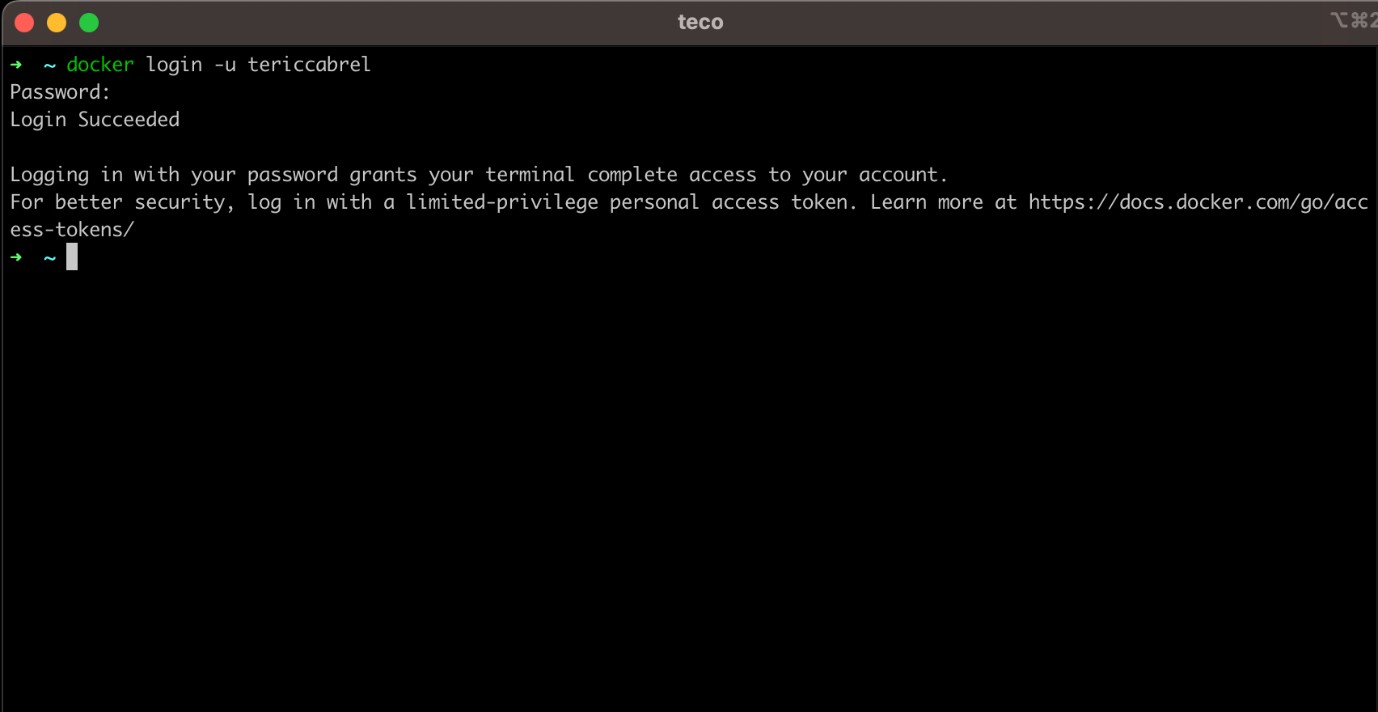
## -e flag: sets the environment variables

This command allows us to run new commands in a running container.



1. **Docker Login:** docker login

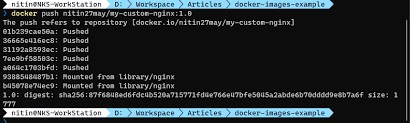
## The Docker login command will help you to authenticate with the Docker hub by which you can push and pull your images.



1. **Docker Push:** docker push <Image name/Image ID>

## Once you build your own customized image by using

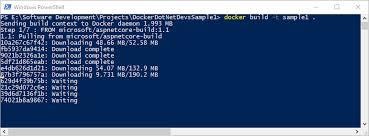
Dockerfile you need to store the image in the remote registry which is DockerHub for that you need to push your image by using the following command.



1. **Docker Build:** docker build -t image\_name:tag .

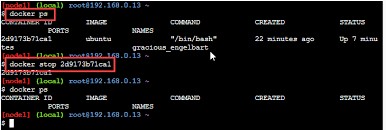
## In the place of image\_name use the name of the image you build with and give the tag number and . “dot” represents the current directory.

The docker build command is used to build the docker images with the help of [Dockerfile.](https://www.geeksforgeeks.org/docker-concept-of-dockerfile/)



1. **Docker Stop:** docker stop <container\_name or id>

## You can stop and start the docker containers where you can do the maintenance for containers. To stop and start specific containers you can use the following commands.



1. **Stop Multiple Containers:** docker stop <container1>

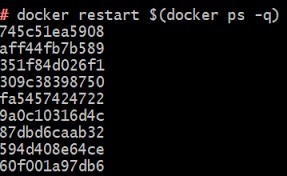
## <container2> <container3>

Instead of stopping a single container. You can stop multiple containers at a time by using the following commands.

# Docker Restart:

## docker restart <container\_name\_or\_id>

While running the containers in Docker you may face some errors and containers fails to start. You can restart the containers.



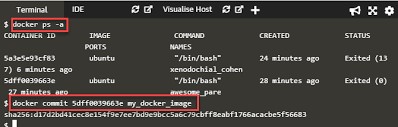
# Docker Commit command:

## docker commit <container\_name\_or\_id>

<new\_image\_name>:tag

## After running the containers by using the current image you can make the updates to the containers by interacting with

the containers from that containers you can create an image by using the following commands.



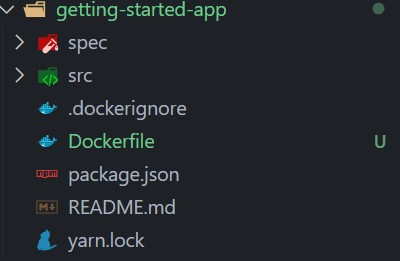
## Some other useful commands:

1. **docker inspect command:** It will helps to debug the docker image if any errors occurred while building an image or pulling the image.
2. **docker save command:** It will save the docker image in the form of dockerfile.
3. **docker rmi command:** It will remove the docker image.
4. **docker cp command:** To copy the file from docker host to the docker containers.

## **docker rm:** Docker rm command will remove the containers which are in the stop condition.

### WEEK-7

**Develop a simple containerized application using Docker.**

1. Clone the getting-started-app repository using the following command:
   1. git clone https://github.com/docker/getting-started- app.git
2. View the contents of the cloned repository. You should see the following files and sub-directories.

├── getting-started-app/

│ ├── .dockerignore

│ ├── package.json

│ ├── README.md

│ ├── spec/

│ ├── src/

│ └── yarn.lock

* Now create a Docker file in the same directory and add the following code

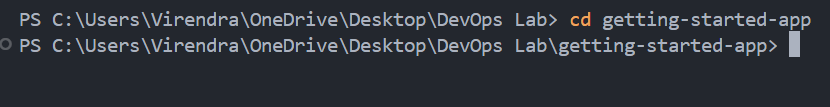
[FROM](https://docs.docker.com/reference/dockerfile/#from) node:18-alpine [WORKDIR](https://docs.docker.com/reference/dockerfile/#workdir) /app

[COPY](https://docs.docker.com/reference/dockerfile/#copy) . .

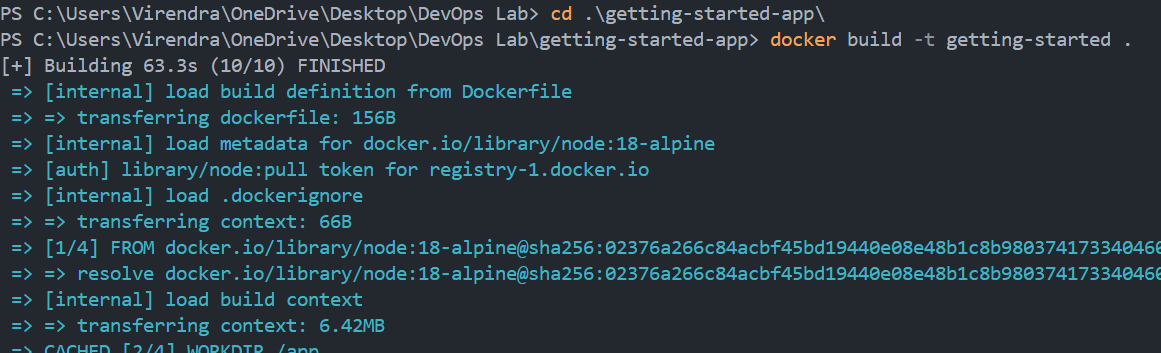
[RUN](https://docs.docker.com/reference/dockerfile/#run) yarn install --production [CMD](https://docs.docker.com/reference/dockerfile/#cmd) ["node", "src/index.js"] [EXPOSE](https://docs.docker.com/reference/dockerfile/#expose) 3000

* In the terminal, make sure you're in the getting-started-app directory. Replace /path/to/getting-started-app with the path to your getting-started-app directory.

o cd /path/to/getting-started-app



### Build the Image

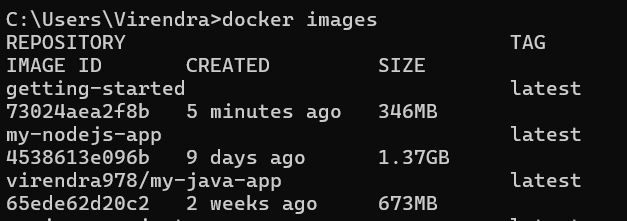
Run the following commands in the terminal docker build -t getting-started .

**15 |** P a g e

**Prepared by Faculty M Shiva Kumar and 22 batch CSE**

**Students**

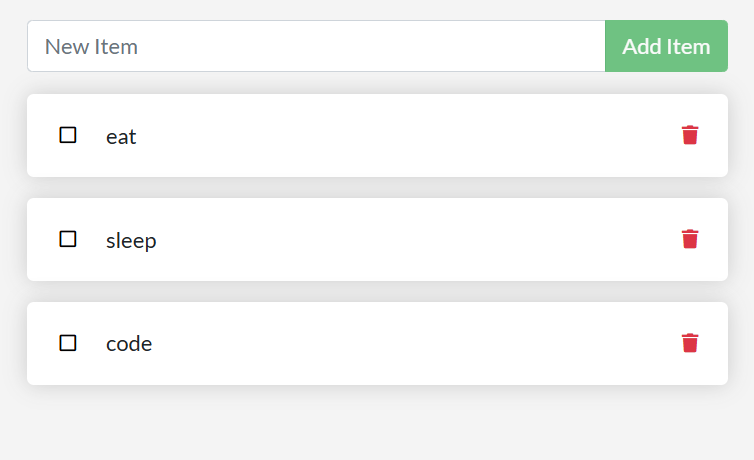
* You can verify whether the image is created or not by running the command docker images which will list all the images and you can find the **getting-started** image. Also you can check the image in the docker desktop images section.



* Now run the image using the command

docker run -dp 127.0.0.1:3000:3000 getting-started

* Open browser and type https://localhost:3000 to check the output the image is running on port 3000.



### WEEK-8

**Integrate Kubernetes and Docker**

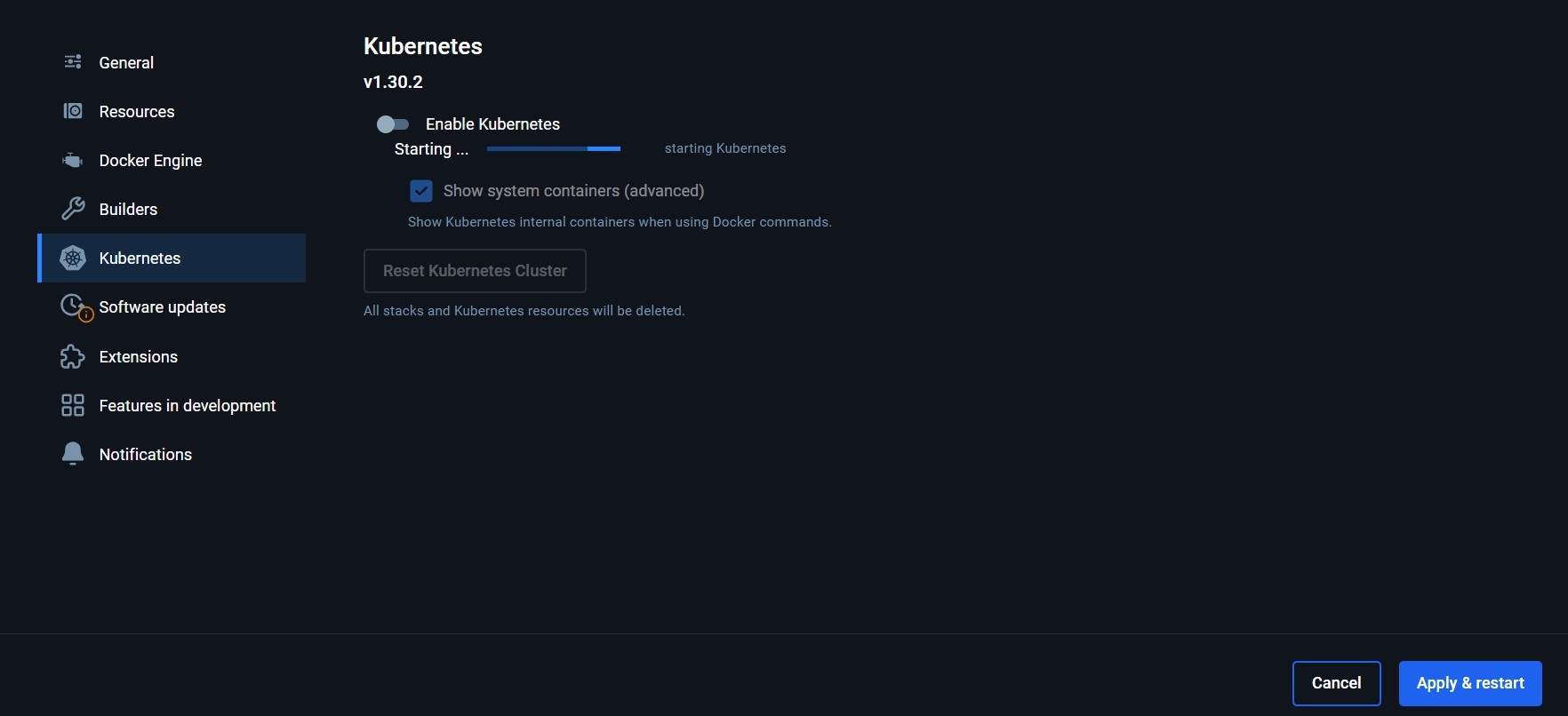
Docker Desktop includes a standalone Kubernetes server and client, as well as Docker CLI integration that runs on your machine.

The Kubernetes server runs locally within your Docker instance, is not configurable, and is a single-node cluster. It runs within a Docker container on your local system, and is only for local testing.

Turning on Kubernetes allows you to deploy your workloads in parallel, on Kubernetes, Swarm, and as standalone containers. Turning on or off the Kubernetes server does not affect your other workloads.

**Install and turn on Kubernetes**

* From the Docker Dashboard, select the **Settings**.
* Select **Kubernetes** from the left sidebar.
* Next to **Enable Kubernetes**, select the checkbox.
* Select **Apply & Restart** to save the settings and then select **Install** to confirm. This instantiates images required to run the Kubernetes server as



containers, and installs the /usr/local/bin/kubectl command on your machine.

The kubectl binary is not automatically packaged with Docker Desktop for Linux. To install the kubectl command for Linux, see [Kubernetes documentation](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)

1. . It should be installed at /usr/local/bin/kubectl.

By default, Kubernetes containers are hidden from commands like docker ps, because managing them manually is not supported. Most users do not need this option. To see these internal containers, select **Show system containers (advanced)**.

When Kubernetes is turned on and running, an additional status bar in the Docker Dashboard footer and Docker menu displays.

**Note**

Docker Desktop does not upgrade your Kubernetes cluster automatically after a new update. To upgrade your Kubernetes cluster to the latest version, select **Reset Kubernetes Cluster**.

[**Use the kubectl command**](https://docs.docker.com/desktop/kubernetes/#use-the-kubectl-command)

Kubernetes integration provides the Kubernetes CLI command at

/usr/local/bin/kubectl on Mac and at C:\Program Files\Docker\Docker\Resources\bin\kubectl.exe on Windows.

If you have already installed kubectl and it is pointing to some other environment, such as minikube or a GKE cluster, ensure you change the context so that kubectl is pointing to docker-desktop:

kubectl config get-contexts

kubectl config use-context docker-desktop

**Tip**

Run the kubectl command in a CMD or PowerShell terminal, otherwise kubectl config get-contexts may return an empty result.If you installed kubectl using Homebrew, or by some other method, and experience conflicts, remove

/usr/local/bin/kubectl.

You can test the command by listing the available nodes:

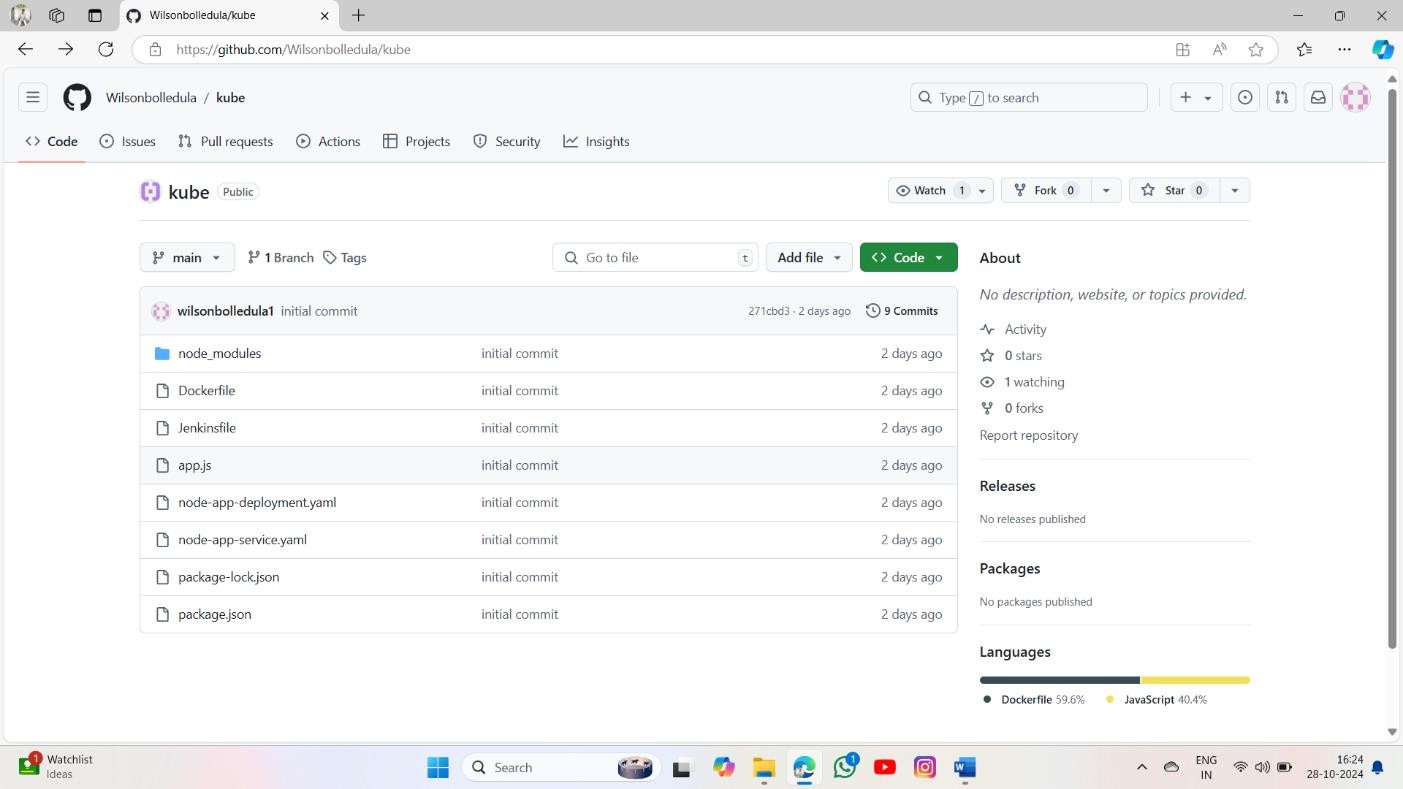
kubectl get nodes

|  |  |  |  |
| --- | --- | --- | --- |
| NAME  VERSION | STATUS | ROLES | AGE |
| docker-desktop | Ready | control-plane | 3h |
| v1.29.1  **WEEK- 9** |  |  |  |

### Automate the process of running containerized application developed in exercise 7 using Kubernetes.

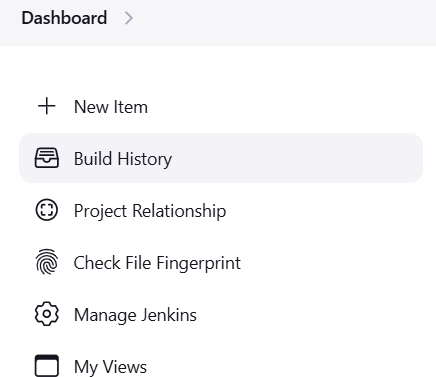
**Step1:**

1. Clone this repository to your local repository https://github.com/shiv4j/kube
2. Make sure that cloned repository consist of “node-app-deployment.yaml”, “node-app-service.yaml” files in folder.
3. Push this local repository to github (or) you can fork that repository from [https://github.com/shiv4j /kube.git](https://github.com/shiv4j/kube.git)
4. After completion of pushing or forking of kube foleder into your github repository you should able to see like this that contains all the files.

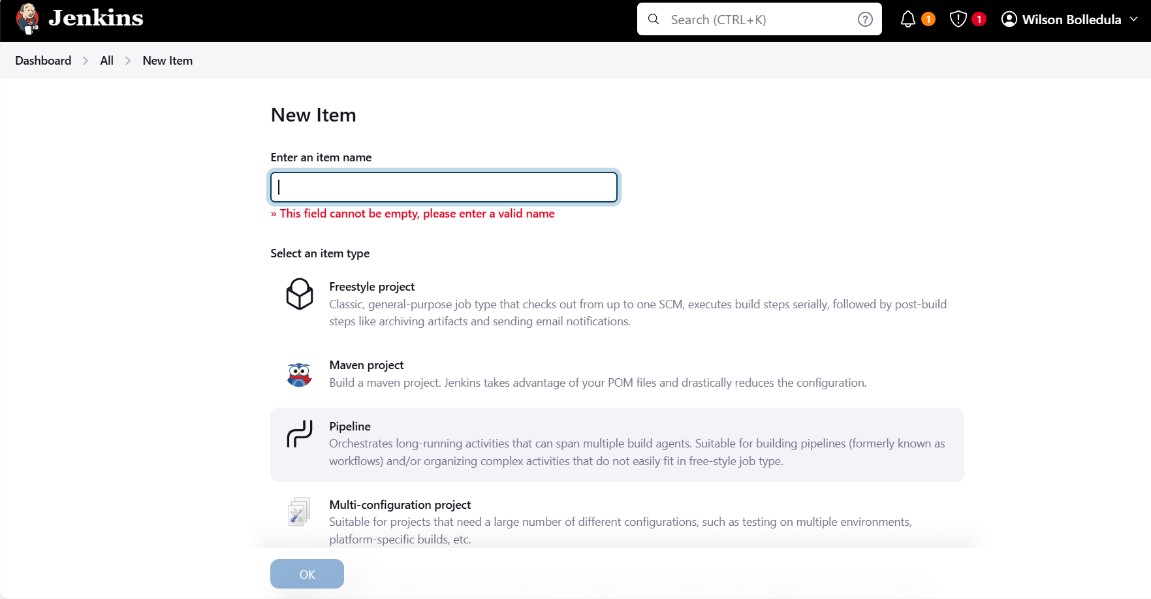


### Step-2:

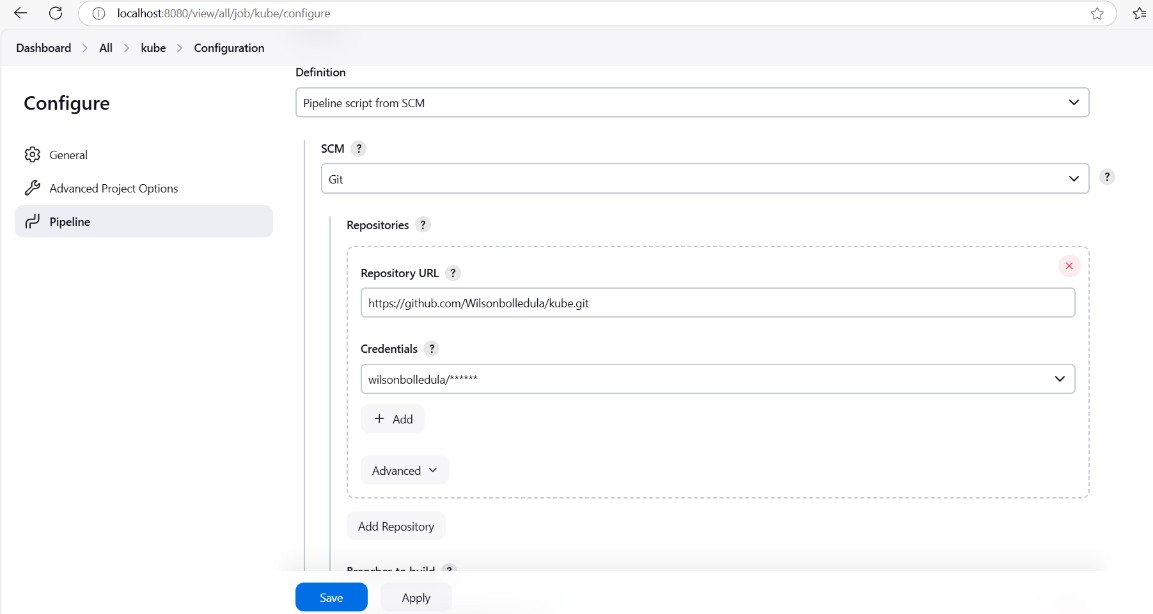
* Push your github repository to the Jenkins.
* click on new item



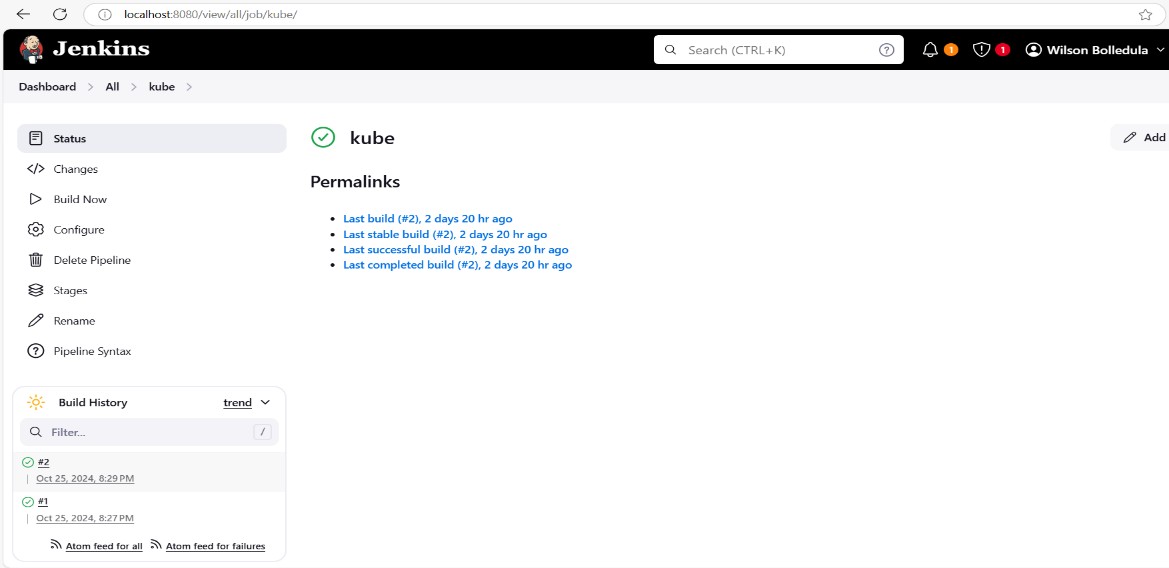
* Enter a name , select pipeline project and click on ok



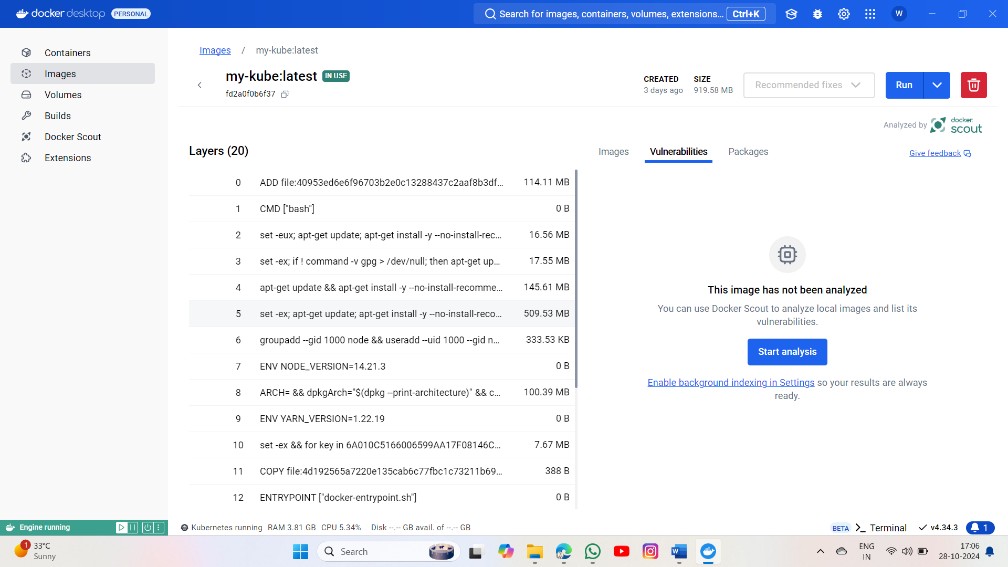
* In the configure paste your repository url and specify your branch whether it is main or master based on your github repository after that apply and save it.



* after creation of your Jenkins project build it, the build should be shown in green colour



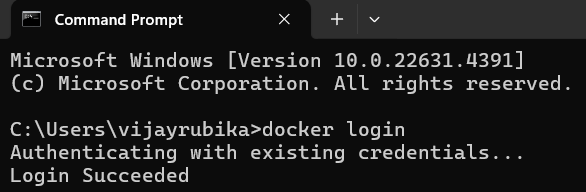
* you will get docker image for this project , like showed in the below



### Step-3:

* Push the docker image into dockerhub

>open command prompt and run the command “**docker login**”



* tag your iamge using this syntax

**docker tag <local-image-name>:<tag> yourusername/yourrepo:<tag>** for example: docker tag my-kube:latest wilsonbolledula/my-kube:latest here username is your dockerhub account username

>push the image to dockerhub

**docker push yourusername/yourrepo:<tag>** example: docker push wilsonbolledula/my-kube:latest Step-3:

* Start the Kubernetes Syntax **: minikube start**



**>**Apply node-app-deployment.yaml file

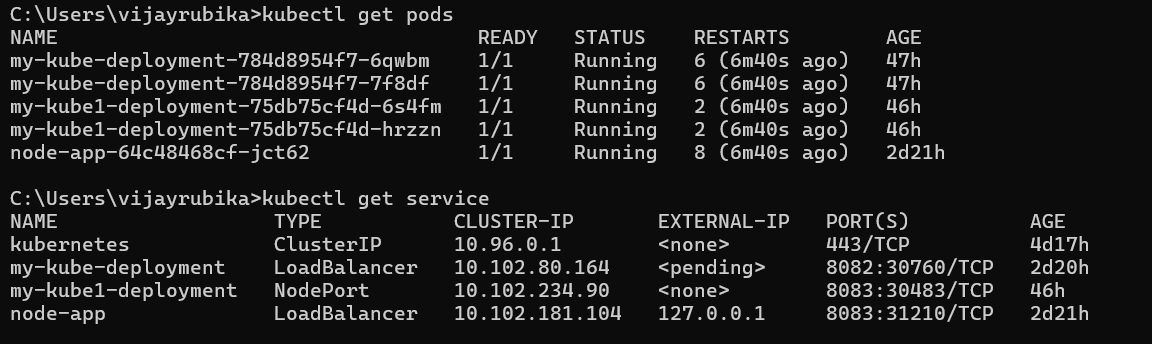
### Syntax: kubectl apply -f node-app-deployment.yaml

* Apply node-app-service.yaml file

### Syntax: kubectl apply -f node-app-service.yaml

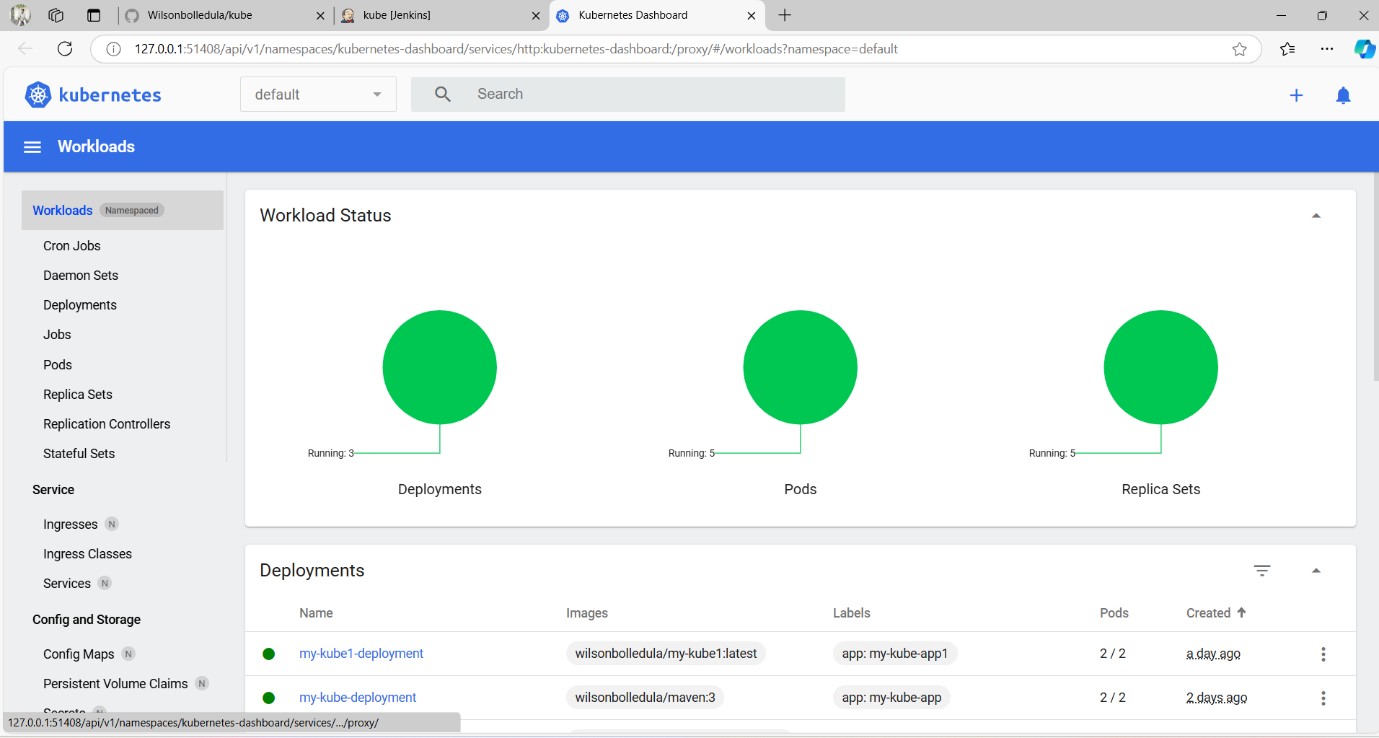
>that will apply to the deployments and services

>To check that type command “kubectl get pods” and “kubectl get service**”**



* To open the Kubernetes dashboard type = “**minikube dashboard**”

That will open Kubernetes dashboard automatically on your default primary browser



* Finally you can checkout your deployments and pods here

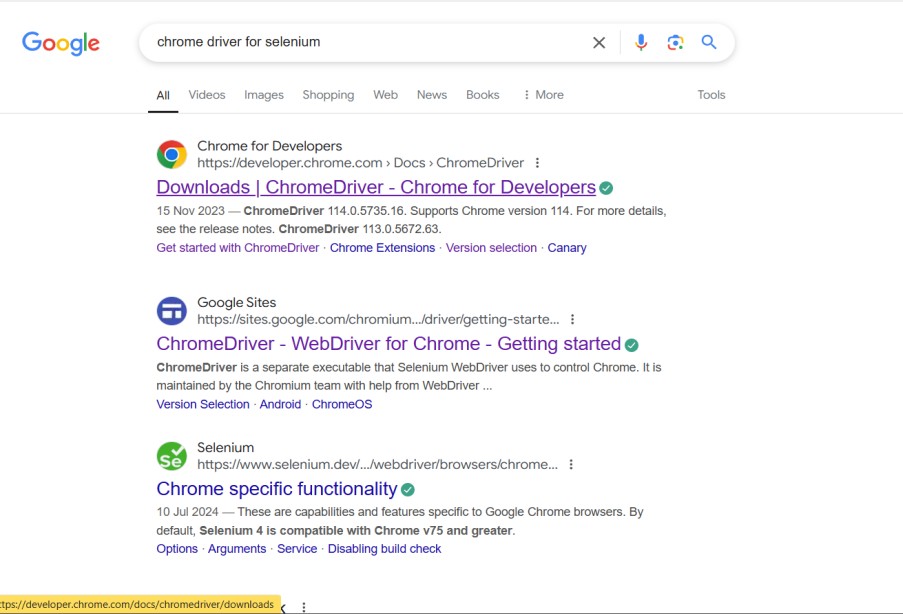
### WEEK– 10

**Install and Explore Selenium for automated testing.**

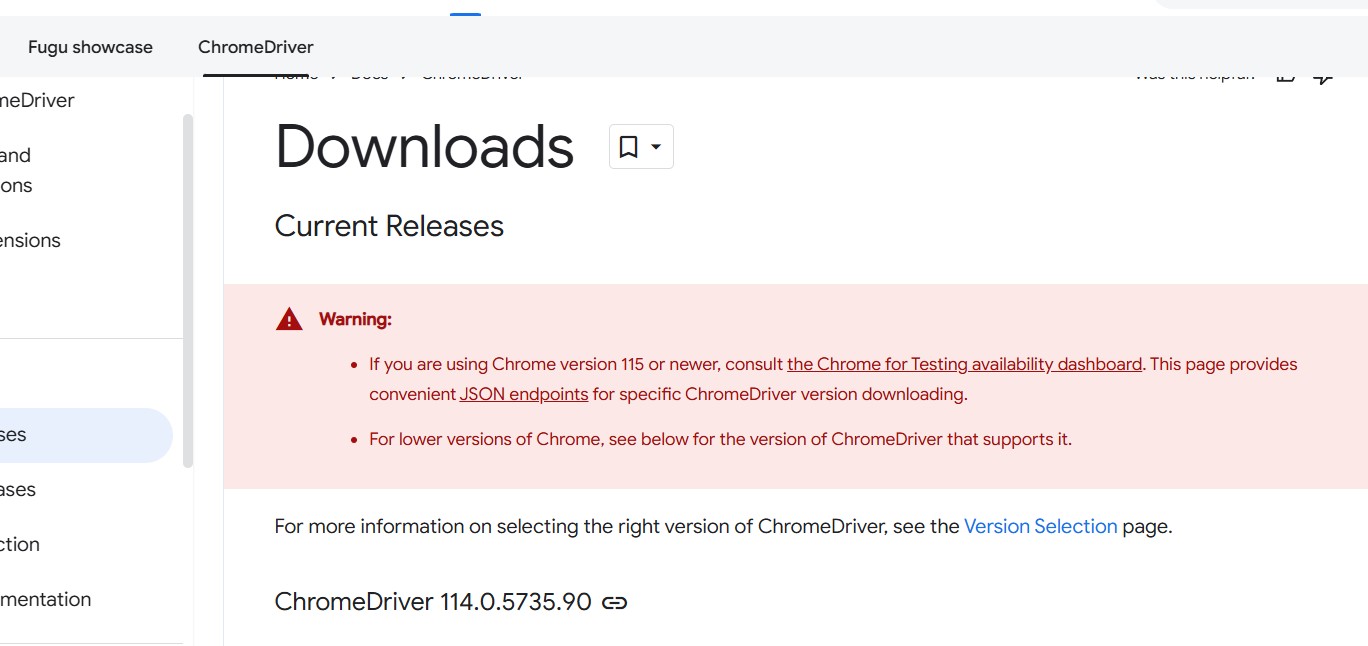
1. Install Python if not already installed, from the official Python website.
2. Install Selenium using pip:

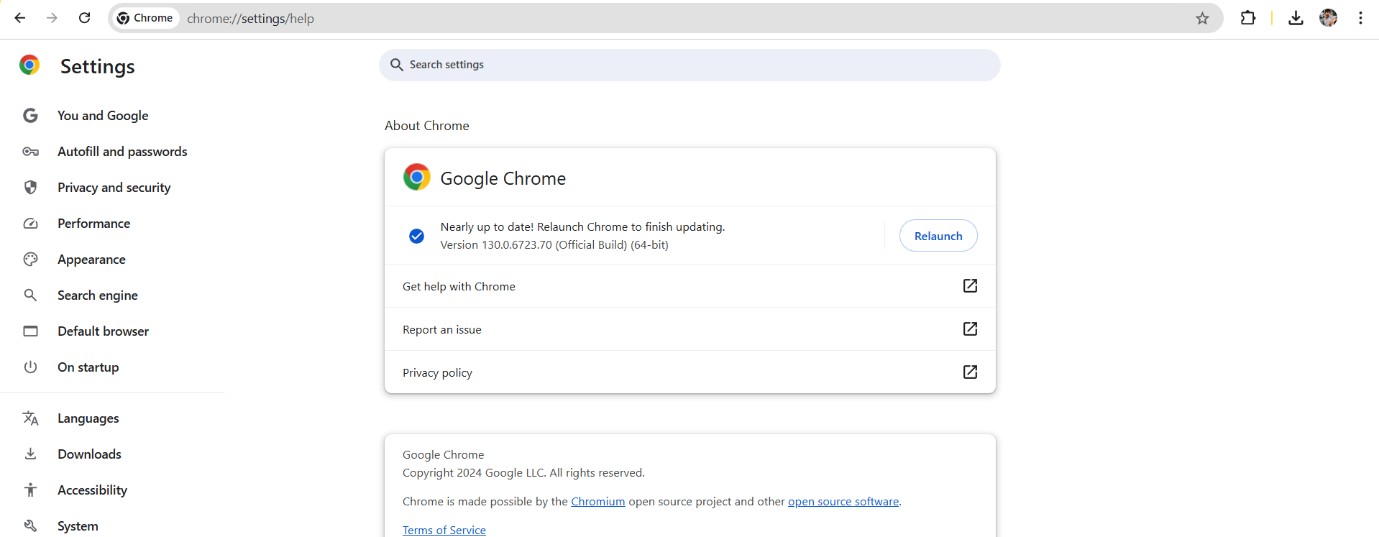
### pip install selenium

1. Download the WebDriver for the specific browser you want to automate (Chrome, Firefox, etc.).

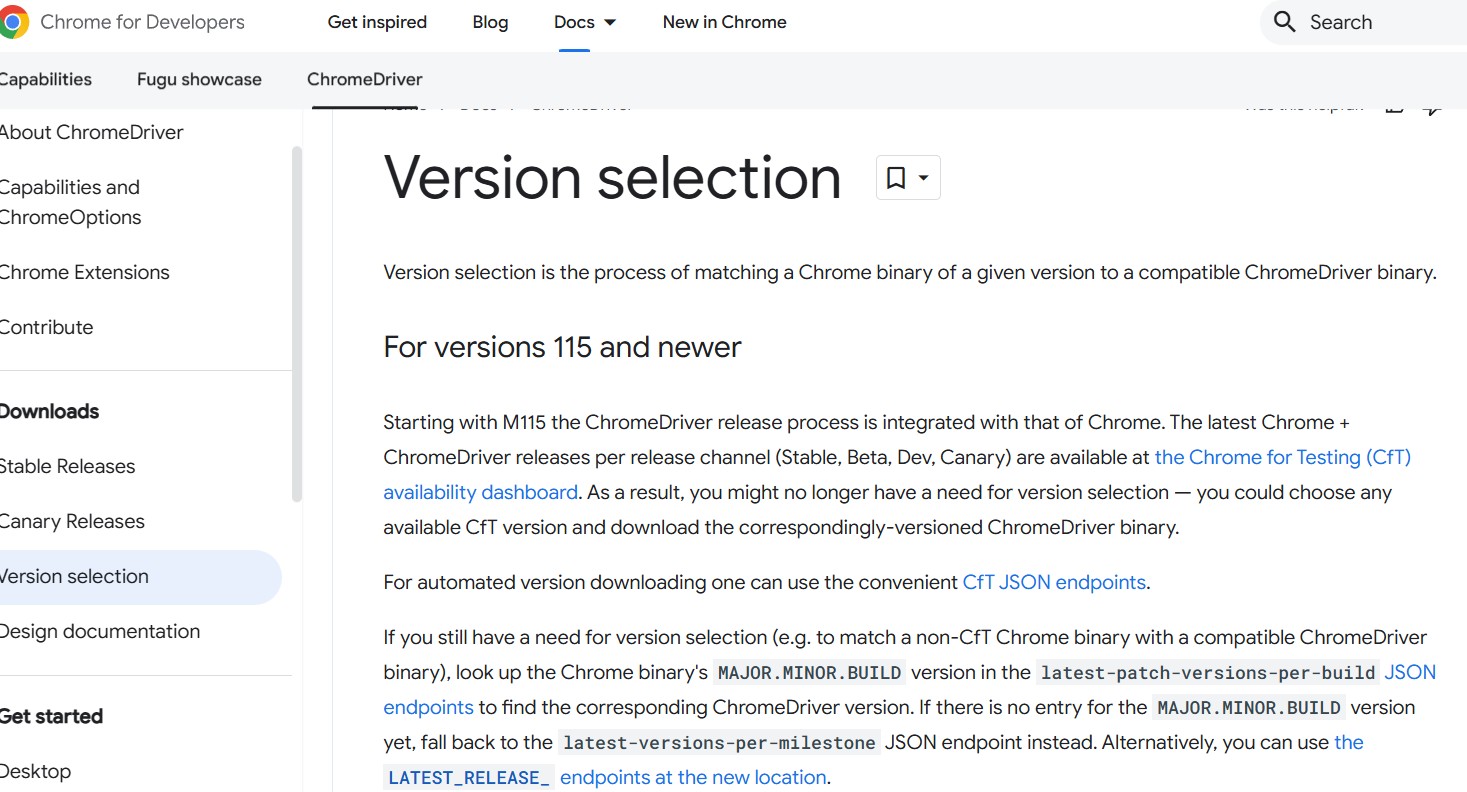


1. Open the 1st link
2. Click on version selection and check your version in chrome by chrome://settings/help

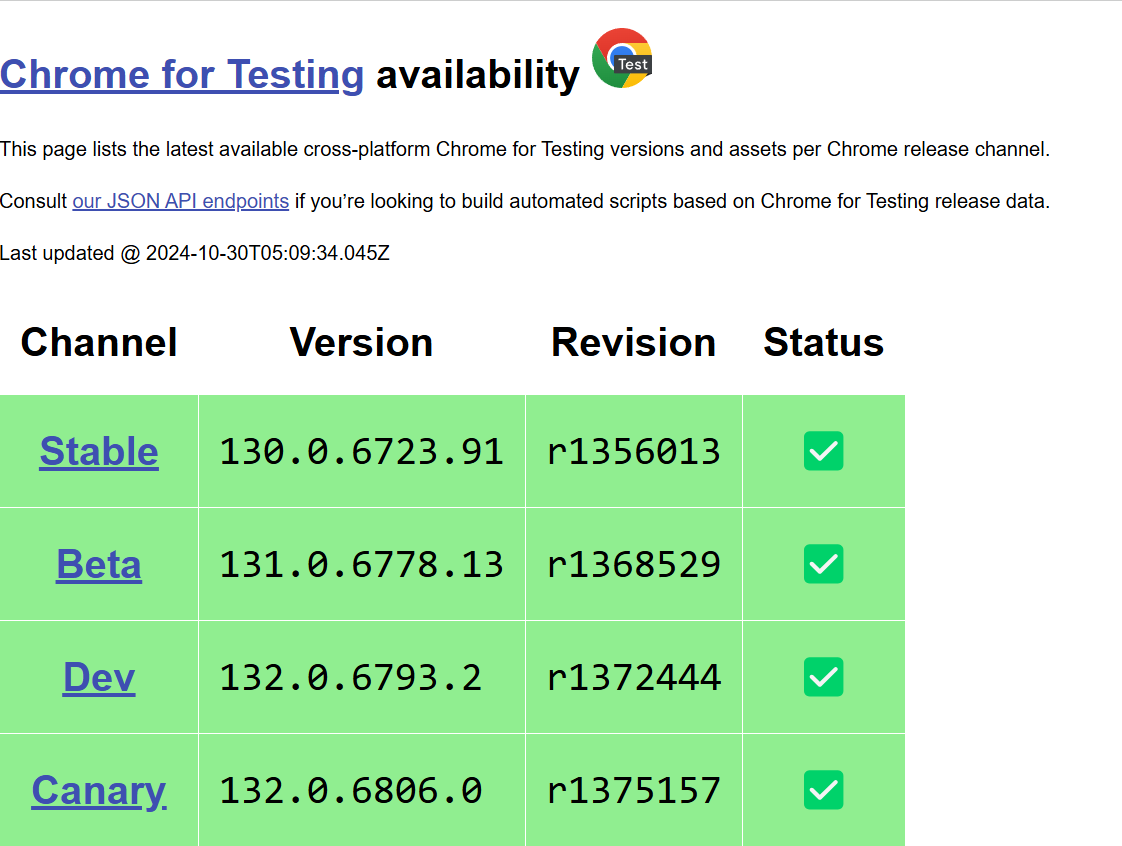




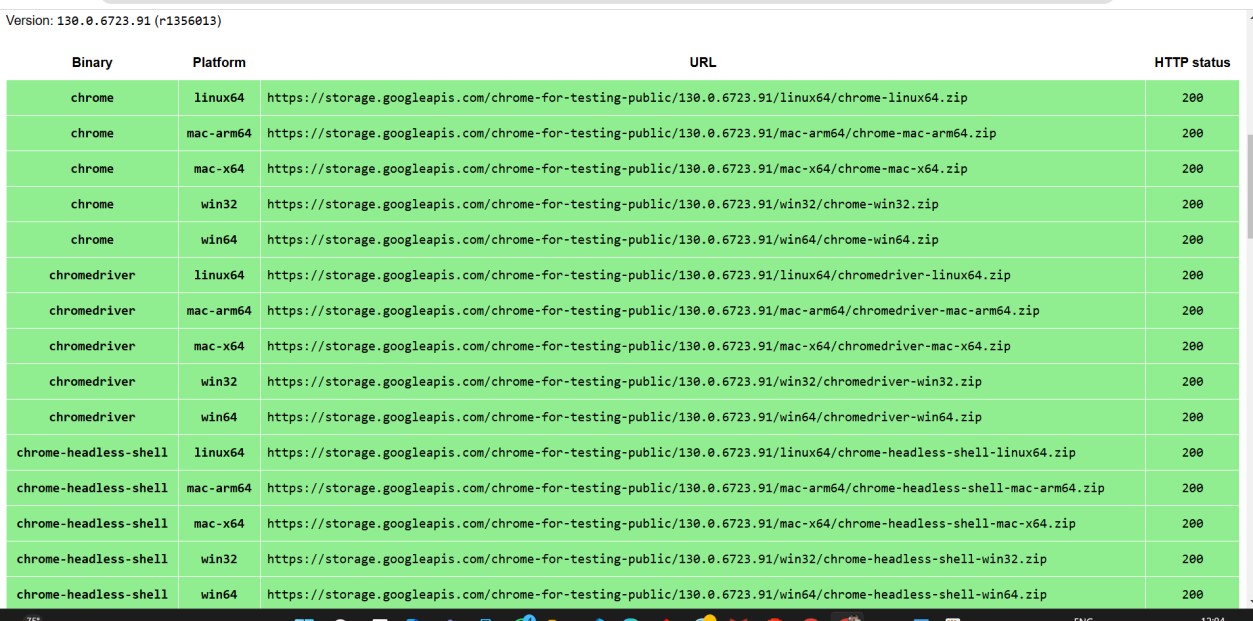
### Click on the Chrome for Testing (CfT) availability dashboard.



1. Click on stable



1. Select according to you os and Copy and paste chromedriver link in chrome it will download automatically



1. the file will be downloaded , extract the files and set the path
2. open VS code and create a .python file and add this code from selenium import webdriver

from selenium.webdriver.common.by import By from selenium.webdriver.common.keys import Keys

from selenium.webdriver.chrome.service import Service import time

# Specify the path to your ChromeDriver

chrome\_service = Service("**C:\\Program Files\\chromedriver- win64\\chromedriver.exe**")

# Make sure to include the actual 'chromedriver.exe' file # Set up the WebDriver for Chrome

driver = webdriver.Chrome(service=chrome\_service) try:

# Step 1: Open Google in the browser driver.get("https://[www.google.com"](http://www.google.com/))

# Step 2: Locate the search box using the name attribute search\_box = driver.find\_element(By.NAME, "q")

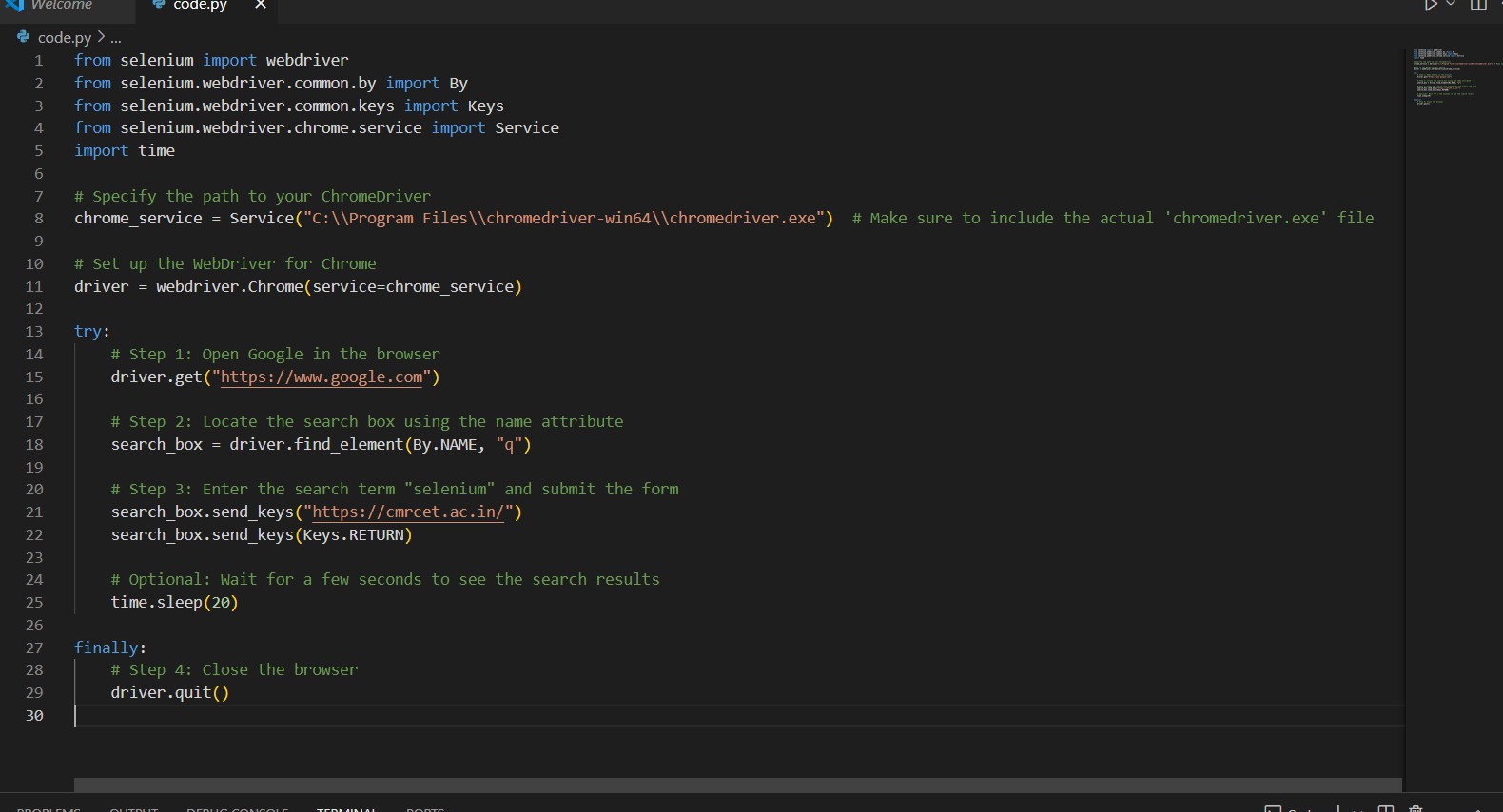
# Step 3: Enter the search term "selenium" and submit the form search\_box.send\_keys("https://cmrcet.ac.in/") search\_box.send\_keys(Keys.RETURN)

# Optional: Wait for a few seconds to see the search results time.sleep(20)

finally:

# Step 4: Close the browser driver.quit()

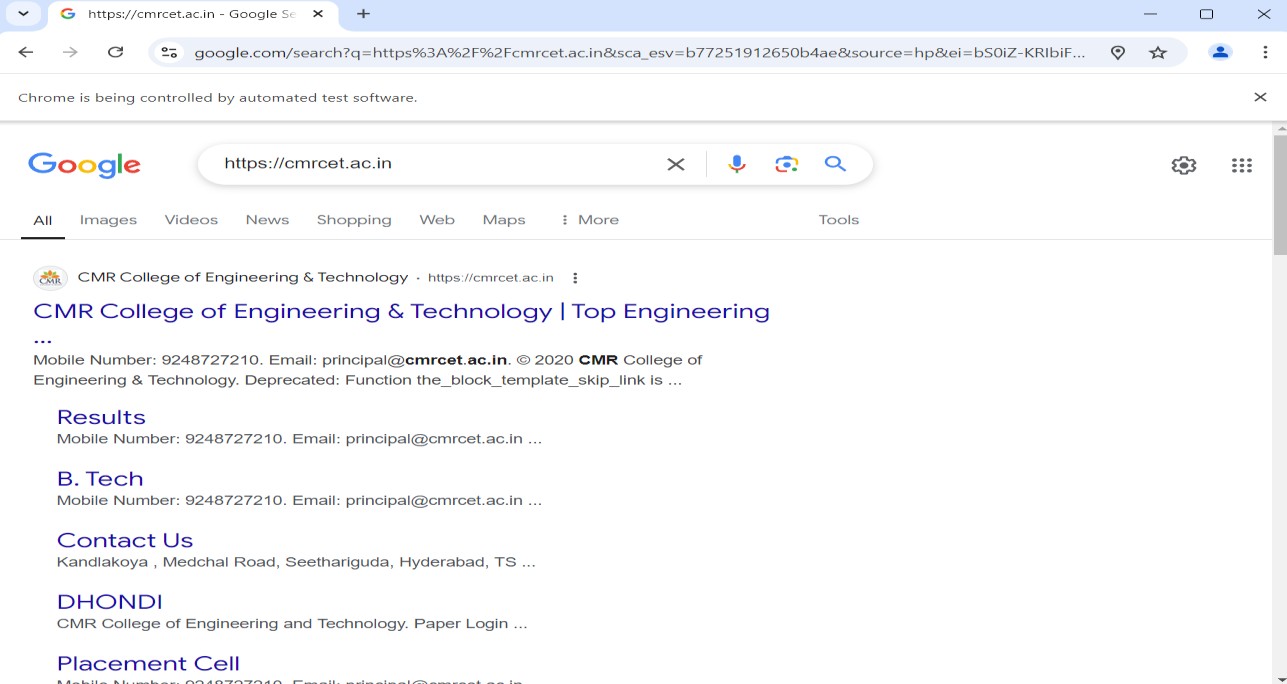
1. Make sure to change the path of the driver in the code to avoid errors



1. Run the code

### Output :

Cmrcet official website will open in chrome browser



### WEEK – 11

**Write a simple program in JavaScript and perform testing using Selenium.**

1. Open VS code and create html and javascript files
2. Create html file and paste this code :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sum Calculator</title>

<style>

body{

text-align: center;

}

</style>

</head>

<body>

<h1>Sum Calculator</h1>

<input type="number" id="num1" placeholder="Enter first number">

<input type="number" id="num2" placeholder="Enter second number">

<button id="add">Add</button>

<p>Result: <span id="result">0</span></p>

<script>

function calculateSum(a, b) { return a + b;

}

document.getElementById('add').addEventListener('click', function() { const num1 = parseInt(document.getElementById('num1').value, 10); const num2 = parseInt(document.getElementById('num2').value, 10); const result = calculateSum(num1, num2); document.getElementById('result').textContent = result;

});

</script>

</body>

</html>

1. Create on javascript file and paste this code :

const webdriver = require('selenium-webdriver'); const assert = require('assert');

const driver = new webdriver.Builder().forBrowser('chrome').build();

async function runTest() { try {

// Open the HTML file in the browser

await driver.get('file://' + dirname + '/index.html');

// Find the input elements and enter values

const num1 = await driver.findElement(webdriver.By.id('num1')); await num1.sendKeys('50');

const num2 = await driver.findElement(webdriver.By.id('num2')); await num2.sendKeys('10');

// Click the "Add" button

const addButton = await driver.findElement(webdriver.By.id('add')); await addButton.click();

// Get the result text and verify it

const result = await driver.findElement(webdriver.By.id('result')); const text = await result.getText();

assert.strictEqual(text, '60', 'Sum calculation is incorrect');

console.log('Test passed: Sum is correct');

} catch (error) {

console.error('Test failed:', error);

} finally {

// Wait for user input to close the browser console.log('Press any key to exit...'); process.stdin.setRawMode(true); process.stdin.resume(); process.stdin.on('data', async () => {

await driver.quit(); process.exit(0);

});

}

}

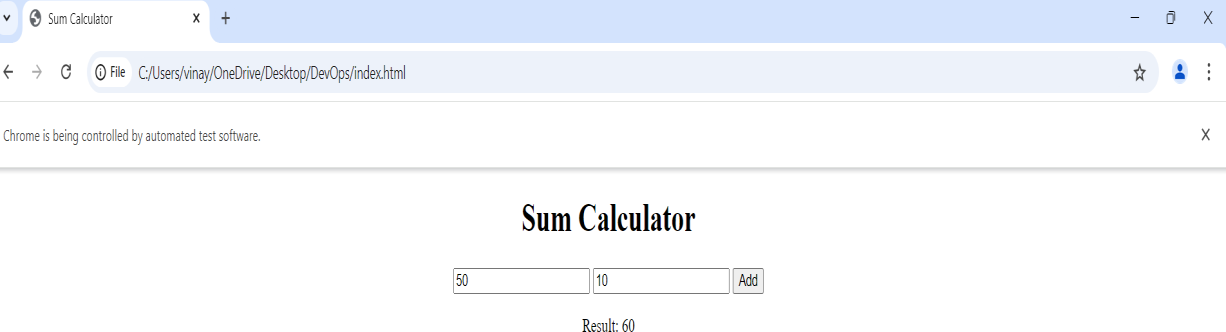
runTest();

### Open terminal

1. **npm install selenium-webdriver**

### node test.js

**Output:**



# WEEK - 12

### 12. Develop test cases for the above containerized application using selenium. Prerequisites :

Docker Desktop

Python (for running scripts locally if needed) Selenium (pip install selenium)

### Steps for execution:

Clone this repository to your local repository https://github.com/Srivaishnavi08/tests

### Or follow the below steps Step-1:

Create a directory named selenium-test and navigate to the current directory path. tests/

**├──** Dockerfile

├── index.html

├── SeleniumTest.py

└── docker-compose.yml

### Step 2: Create the index.html File

This is your sample web page that Selenium will interact with.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Selenium Login Example</title>

</head>

<body>

<!-- Homepage -->

<div id="homepage">

<h1>Welcome!!!!!!</h1>

<!-- Get Started Free Button -->

<a href="#loginPage" id="get-started" onclick="navigateToLogin()">Get started</a>

</div>

<!-- Login Page -->

<div id="loginPage" style="display: none;">

<h2>Login to the Page</h2>

<form onsubmit="return validateLogin()">

<label for="user\_email\_login">Email:</label>

<input type="email" id="user\_email\_login" name="user\_email\_login" required>

<br><br>

<label for="user\_password">Password:</label>

<input type="password" id="user\_password" name="user\_password" required>

<br><br>

<button type="submit" name="commit">Login</button>

</form>

<p id="error-message" style="color: red; display: none;">Invalid credentials, please try again.</p>

</div>

<!-- Dashboard Section (only shown after successful login) -->

<div id="dashboard" style="display: none;">

<h2>Welcome to Your Dashboard!</h2>

<p>This is the dashboard area you see after a successful login.</p>

</div>

<script>

// Function to navigate to the login page function navigateToLogin() {

document.getElementById('homepage').style.display = 'none'; document.getElementById('loginPage').style.display = 'block';

}

// Function to validate login credentials function validateLogin() {

const email = document.getElementById('user\_email\_login').value; const password = document.getElementById('user\_password').value;

if (email === ["ab](mailto:abc@gmail.com)c[@gmail.com"](mailto:abc@gmail.com) && password === "password") {

// Hide login page and display dashboard document.getElementById('loginPage').style.display = 'none'; document.getElementById('dashboard').style.display = 'block'; return false; // Prevent actual form submission

} else {

// Show error message if credentials are incorrect document.getElementById('error-message').style.display = 'block'; return false; // Prevent actual form submission

}

}

</script>

</body>

</html>

### Step 3: Create the Dockerfile

This Dockerfile sets up a simple HTTP server to serve your index.html file.

# Use the official Python image as the base image FROM python:3.9

# Set the working directory in the container WORKDIR /app

# Copy the index.html file to the container COPY index.html .

# Expose port 8000 for the HTTP server EXPOSE 8000

# Start a simple HTTP server to serve the index.html file CMD ["python", "-m", "http.server", "8000"]

### Step-4: Create the Selenium Test Script

This script will automate testing of your HTML page using Selenium.

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC import time

print("Test Execution Started") options = webdriver.ChromeOptions()

options.add\_argument('--ignore-ssl-errors=yes') options.add\_argument('--ignore-certificate-errors')

# Start the Selenium WebDriver driver = webdriver.Remote(

command\_executor='http://localhost:4444/wd/hub', options=options

)

# Maximize the window size driver.maximize\_window() time.sleep(10)

driver.get("http://host.docker.internal:8000") # Access the local server time.sleep(10)

try:

# Wait for the "Get started free" link to be clickable link = WebDriverWait(driver, 30).until(

EC.element\_to\_be\_clickable((By.LINK\_TEXT, "Get started"))

)

link.click() # Click the link

time.sleep(10) # Wait for any resulting page to load

WebDriverWait(driver, 10).until( EC.presence\_of\_element\_located((By.ID, "user\_email\_login))

)

WebDriverWait(driver, 10).until( EC.presence\_of\_element\_located((By.ID, "user\_password"))

)

# Enter login credentials

username = driver.find\_element(By.ID, "user\_email\_login") password = driver.find\_element(By.ID, "user\_password") login\_button = driver.find\_element(By.NAME, "commit")

username.send\_keys("[abc@gmail.com")](mailto:abc@gmail.com) # Replace with actual username password.send\_keys("password") # Replace with actual password login\_button.click()

# Check for a post-login element (adjust to your page's unique element for logged-in users) try:

error\_message = WebDriverWait(driver, 10).until( EC.visibility\_of\_element\_located((By.ID, "error-message"))

)

time.sleep(10)

print("Login failed: Incorrect credentials") except:

# No error message found, proceed with checking for dashboard WebDriverWait(driver, 10).until(

EC.visibility\_of\_element\_located((By.ID, "dashboard")) # Replace with actual post- login element ID

)

print("Login Successful!")

except Exception as e:

print(f"An error occurred while trying to click the link: {e}")

finally:

# Ensure the browser quits after execution driver.quit()

print("Test Execution Completed!")

### Step 5: Create the docker-compose.yml File

This file defines two services: your HTML server and the Selenium Chrome container.

app: build:

context: .

dockerfile: Dockerfile container\_name: html-server ports:

- "8000:8000"

selenium:

image: selenium/standalone-chrome container\_name: selenium-chrome ports:

- "4444:4444"

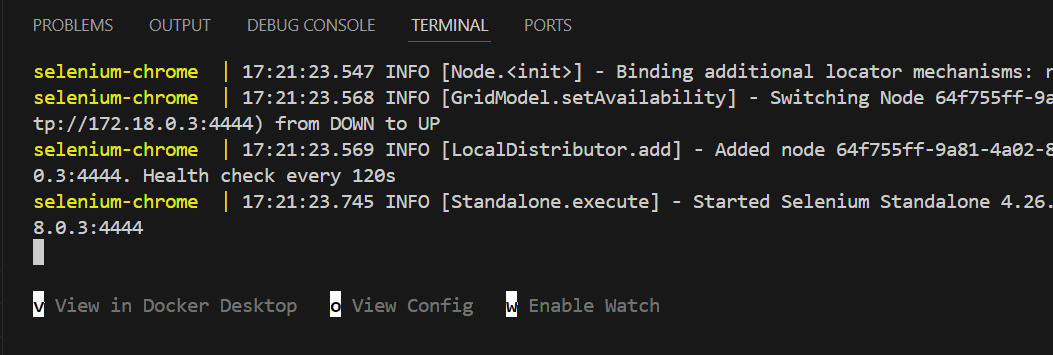
depends\_on:

### - appp-6: Build and Run Your Docker Containers

In terminal use **docker-compose up –build**

This command will:

1. Build the Docker image for your HTML server.
2. Pull the Selenium standalone Chrome image.
3. Start both services.

Press **v** to navigate to docker.

And Click on the links.

### 8000:8000

**4444**

### Step 7: Relenium Test Script

While your containers are running, open a new terminal and run:

In terminal