Jenkins by valentin udemy

pipeline {

agent any

stages {

stage('Hello') {

steps {

echo 'Hello World from Deepanshu Kumar!'

sh 'mkdir build'

sh 'touch build/computer.txt'

sh 'echo "Mainboard" >> build/computer.txt'

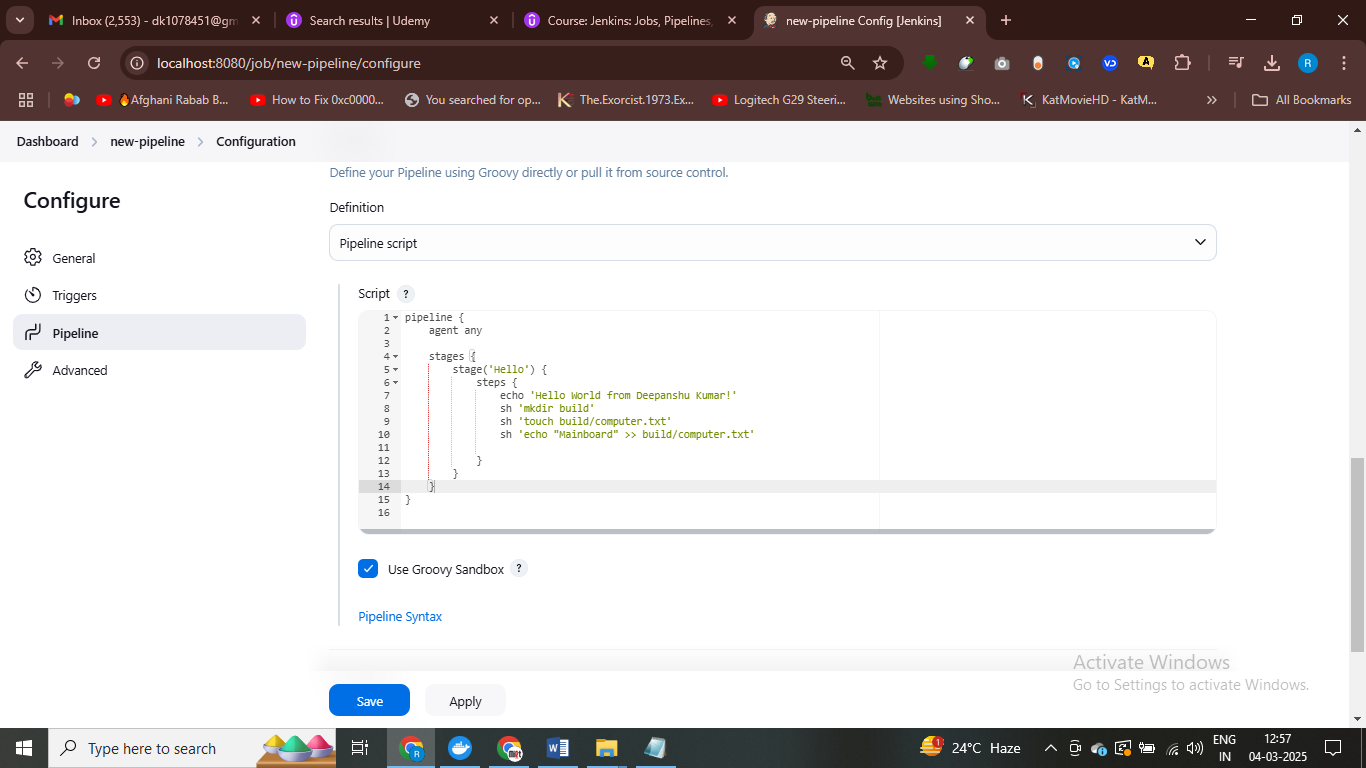
}

}

}

}

# first pipeline in Jenkins



pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building a new laptop ...'

sh 'mkdir -p build'

sh 'touch build/computer.txt'

sh 'echo "Mainboard" >> build/computer.txt'

sh 'cat build/computer.txt'

sh 'echo "Display" >> build/computer.txt'

sh 'cat build/computer.txt'

sh 'echo "Keyboard" >> build/computer.txt'

sh 'cat build/computer.txt'

}

}

}

}

# Jenkins piple ine

# Install stage view plugins

Well, essentially when this job is executed, Jenkins uses what is called workspace.

So a Jenkins workspace is a directory on the Jenkins controller where the agent where Jenkins runs,

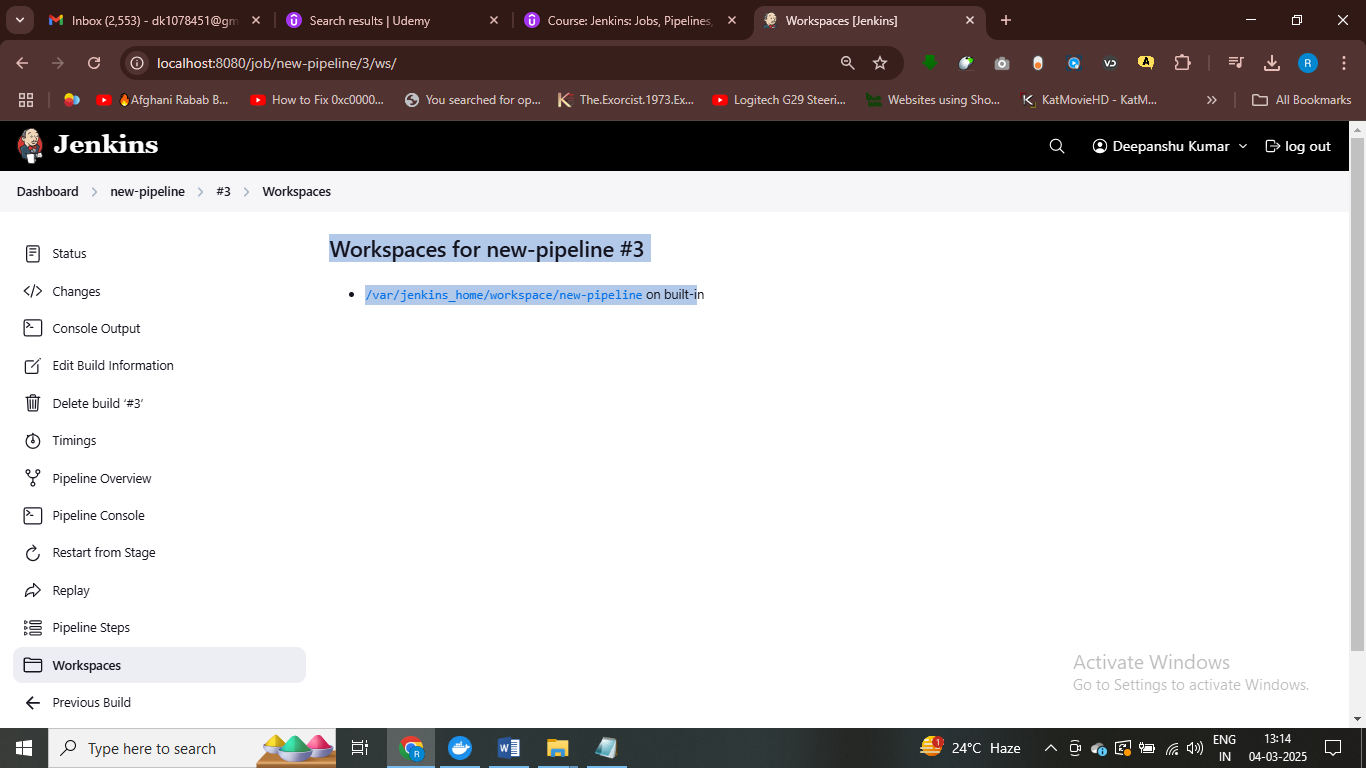
builds, and stores the build results like the build directory that we have created and the file that

is inside.

So we can see that we're here inside the console log.

And we're going to see here the path to the workspace where Jenkins is saving all these files.

So we can go inside here.



So ideally we'd start with a clean workspace before the build to ensure that there are no stale or unnecessary

files left, which may impact other builds.

Now in Jenkins there's this concept of post build actions.

So sometimes we're doing all the things with stages and running commands.

But we want to do something at the end.

And we're going to do something at the end by defining a post build action.

So the first thing we're on the right here is post.

And we're going to open and close curly braces.

And you can notice here that post is at the same indentation as stages.

Right.

So that's super important to understand at which level of indentation this is happening.

Now inside post.

Here we have different options.

So for example we can do something only if the build fails or only if the build is successful.

But in this case we want to clean the workspace regardless if the build was successful or not.

So we're going to use here always.

And again between braces and all, this is going to be part of post.

So what we're going to do here inside always is clean the workspace.

When I right here clean w s and it has to be written exactly as you see it here.

# so in post we write to clean the workspace

# clean the workspace for a secure build

# we will use post and in this we define cleanws()

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building a new laptop ...'

sh 'mkdir -p build'

sh 'touch build/computer.txt'

sh 'echo "Mainboard" >> build/computer.txt'

sh 'cat build/computer.txt'

sh 'echo "Display" >> build/computer.txt'

sh 'cat build/computer.txt'

sh 'echo "Keyboard" >> build/computer.txt'

sh 'cat build/computer.txt'

}

}

}

post {

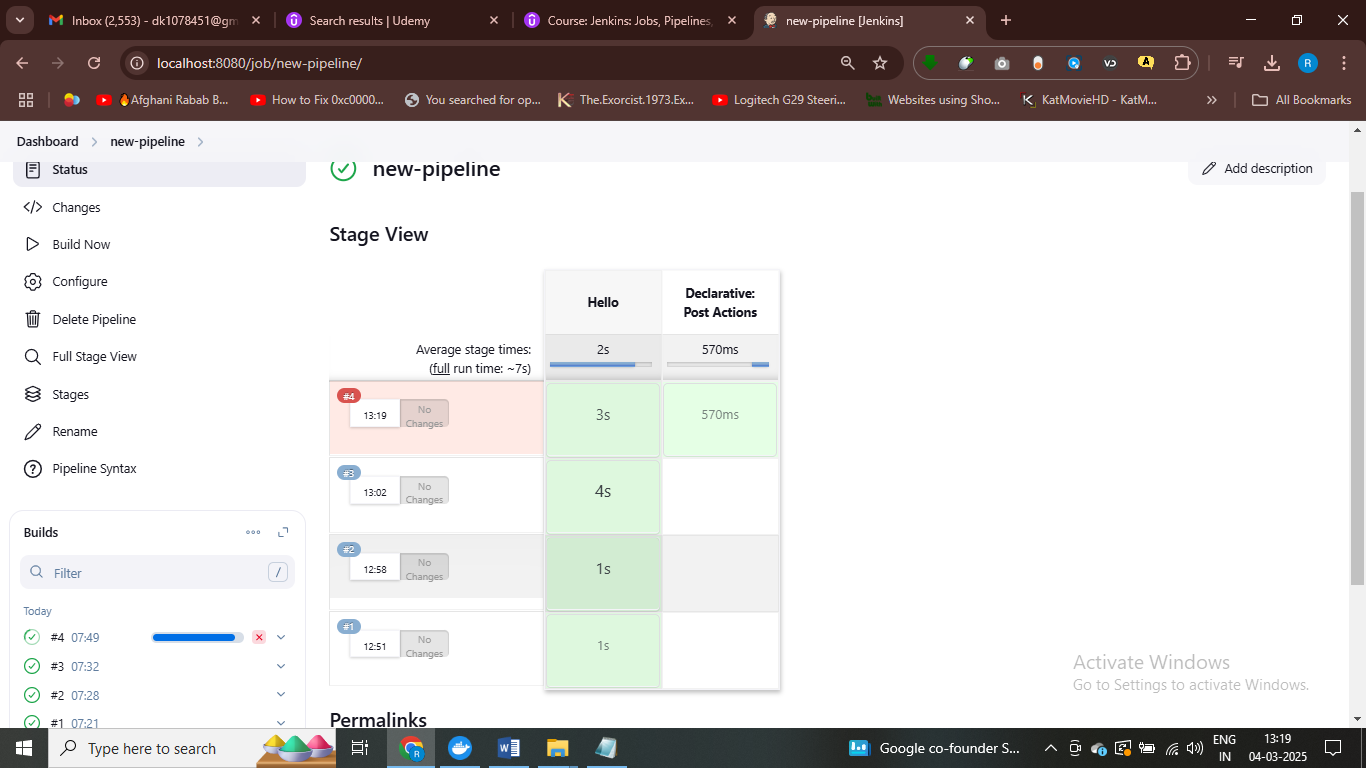
always {

cleanWs()

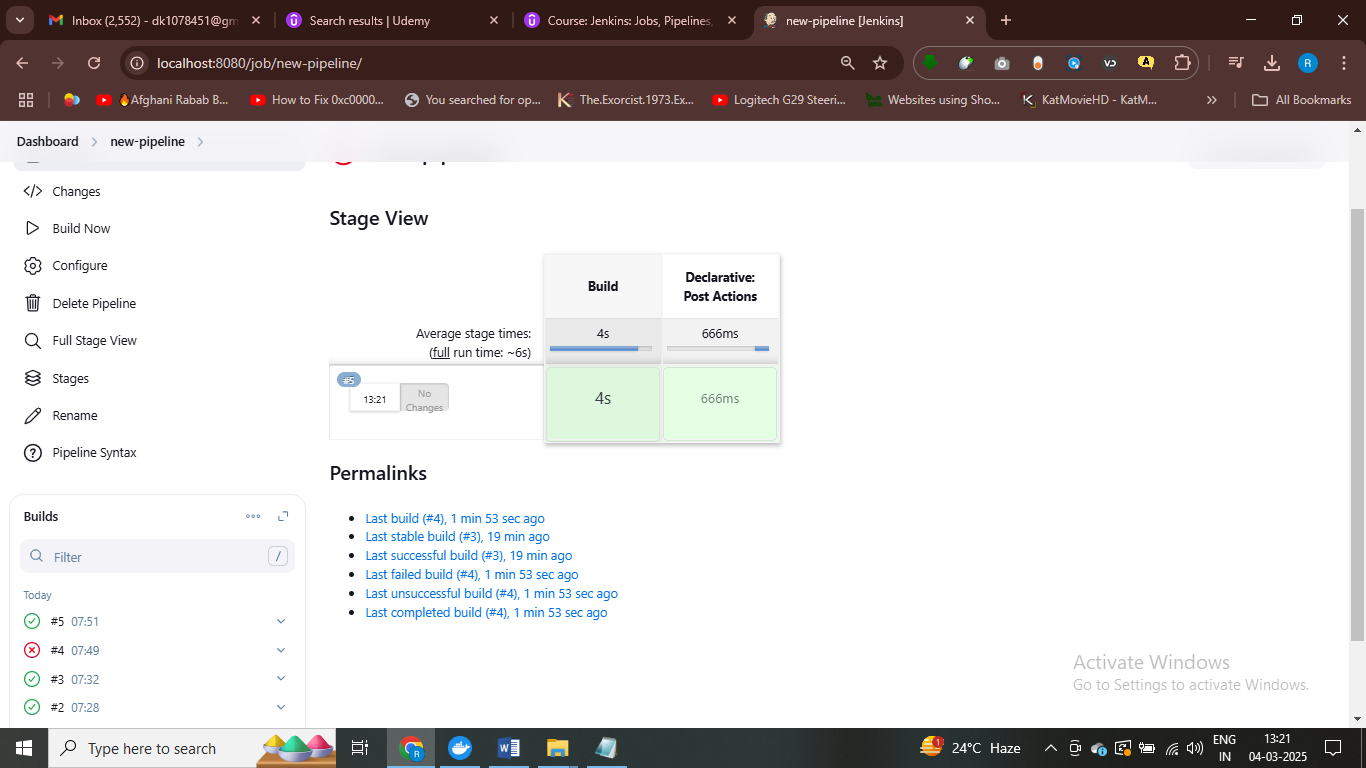
}

}

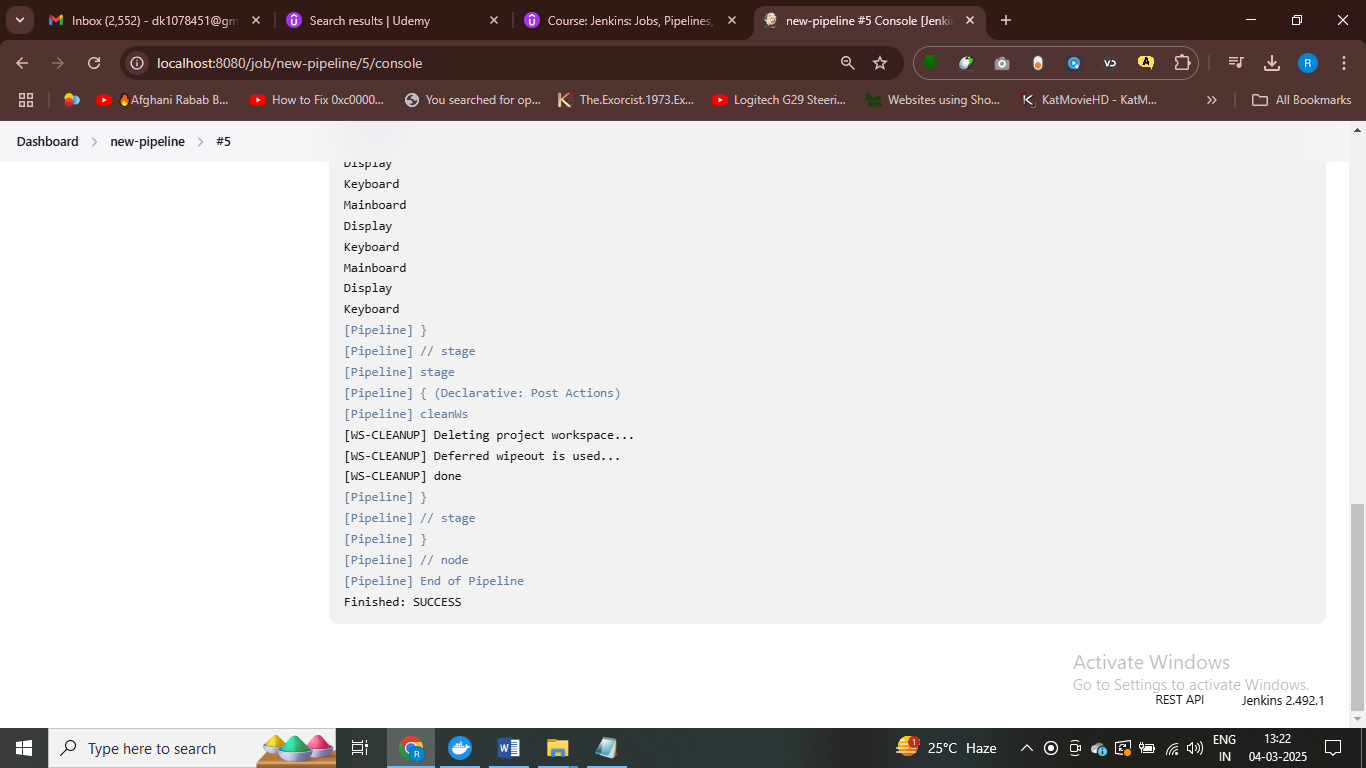
}



# new stage has been added



# succesfuly deleted workspaces



# it will clean the workspaces before it build

Storing build artifcafts

Now, essentially, Jenkins has a specific order how these actions are executed and essentially always

is always executed before success or failure, which is another option.

So there are multiple ways how we can fix things.

And one of this would be for example, to use success and failure or to divide this clean workspace,

but for example, an added benefit of not clearing the workspace at the end would be the fact that we

can still take a look at all the files that have been generated.

This is useful for debugging things.

So what we can do, for example, is clean the workspace at the beginning of the job so we can remove

this instruction from here and essentially remove all this altogether.

We're only going to keep here success.

And what we can do here is, as a first thing, we're going to clear the workspace.

When we're starting the build, we can add a completely new stage where we're clearing the workspace,

but we can just use this instruction here.

So let's take a look at how this is working as it is right now.

We're going to see here that the execution is successful.

And what we're going to see here is that taking a look at the stage.

So it starts by first deleting everything that is in the workspace.

And then it starts the build process.

And then it is archiving the artifacts.

So we have essentially fixed two problems.

Now we can see here right on top this new thing is because we have used this artifacts that we have

archived.

So we're going to get here always like the last successful artifacts.

pipeline {

agent any

stages {

stage('Build') {

steps {

cleanWs()

echo 'Building a new laptop ...'

sh 'mkdir -p build'

sh 'touch build/computer.txt'

sh 'echo "Mainboard" >> build/computer.txt'

sh 'cat build/computer.txt'

sh 'echo "Display" >> build/computer.txt'

sh 'cat build/computer.txt'

sh 'echo "Keyboard" >> build/computer.txt'

sh 'cat build/computer.txt'

}

}

}

post {

success {

archiveArtifacts artifacts: 'build/\*\*'

}

}

}

# stroing build artifacts in Jenkins file

# before storing artifacts – we simply clearing the workspaces intilally clearing the workspaces

# this sh refer to shell in this we have, we simply write cmd here

# Manually Interupting the pipeline execution

And sleep takes an argument in seconds.

So we're going to set it for 600 seconds.

So this will put essentially this job to sleep.

So let's go ahead and click on save.

<https://www.udemy.com/course/jenkins-ci-cd-pipelines-devops-for-beginners/learn/lecture/42401134#overview>

<https://github.com/vdespa/learn-jenkins-app> - fork this repository

make it local and then install dependencies – npm i

* Use docker as build environment

When we want to use Docker in a stage, we need to specify here agent configuration.

And here within the agent we need to specify the image that we want to use.

Now we need to specify here a public image.

And what we're going to do here is for example if want to run something with node we will specify here

the image node and also specific versions.

# Install docker pipeline plugin install

# to use docker – we need to specify docker image and

pipeline {

agent any

stages {

stage('W/O Docker') {

steps {

sh 'echo "without docker"'

}

}

stage('W/ Docker'){

agent {

docker {

image 'node:18-alpine'

}

}

steps {

sh 'echo "with Docker"'

}

}

}

}

# to using docker so we need to specify agen and image

pipeline {

agent any

stages {

stage('W/O Docker') {

steps {

sh 'echo "without docker"'

}

}

stage('W/ Docker'){

agent {

docker {

image 'node:18-alpine'

}

}

steps {

sh 'echo "with Docker"'

sh 'npm --version'

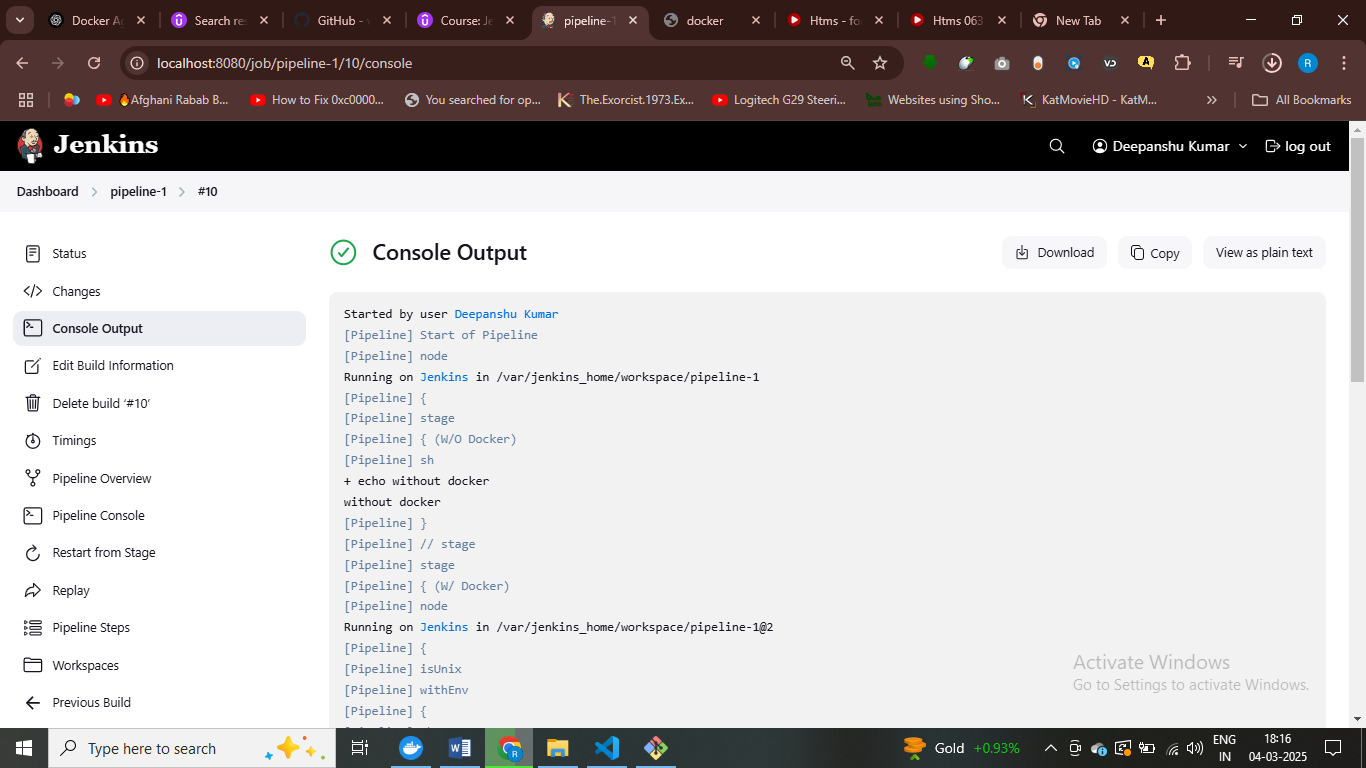
}

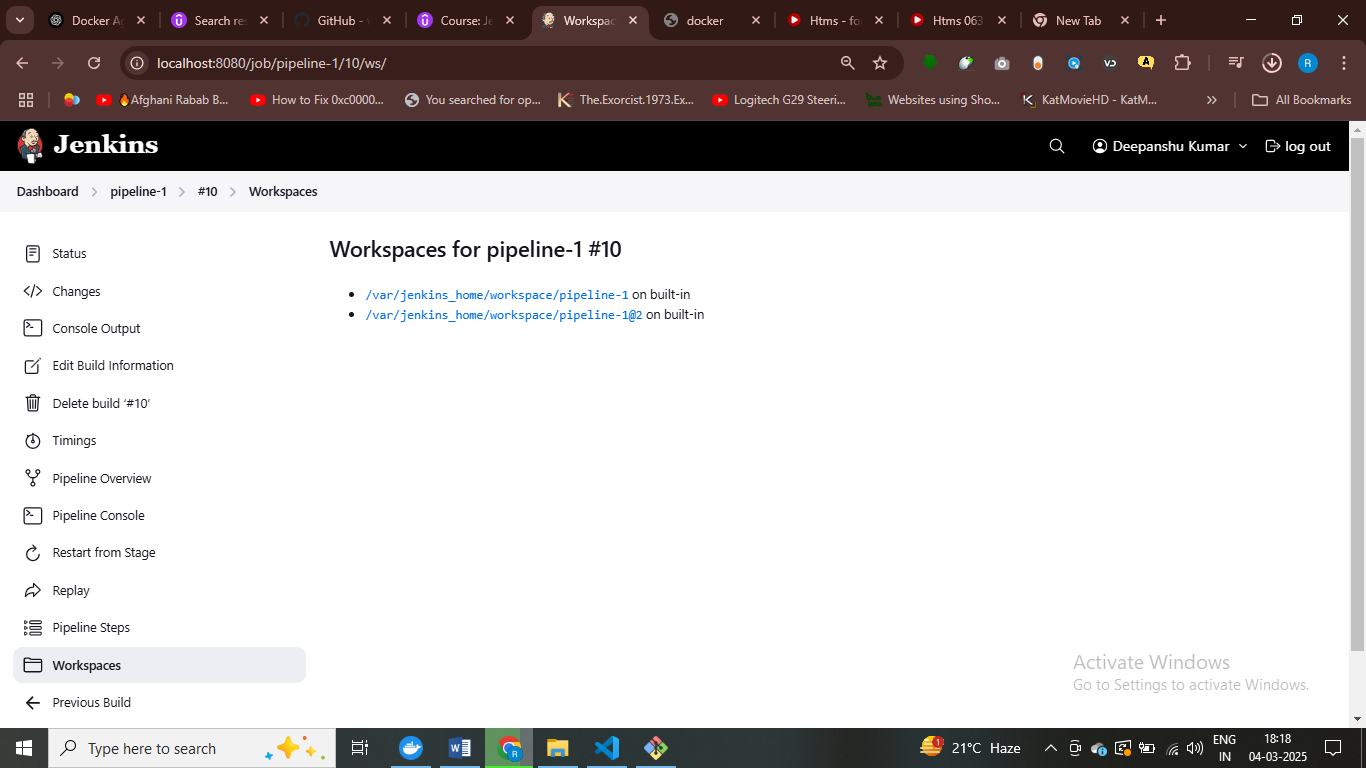
}

}

}

# we want to check npm version





# pipeline is running in different workspaces  
# all the stages in pipeline in the same workspaces

Now, when we're going here to pipeline, we're not going to write the pipeline script here.

But what we're gonna do here is from the select list, we're going to select Pipeline Script from SQM.

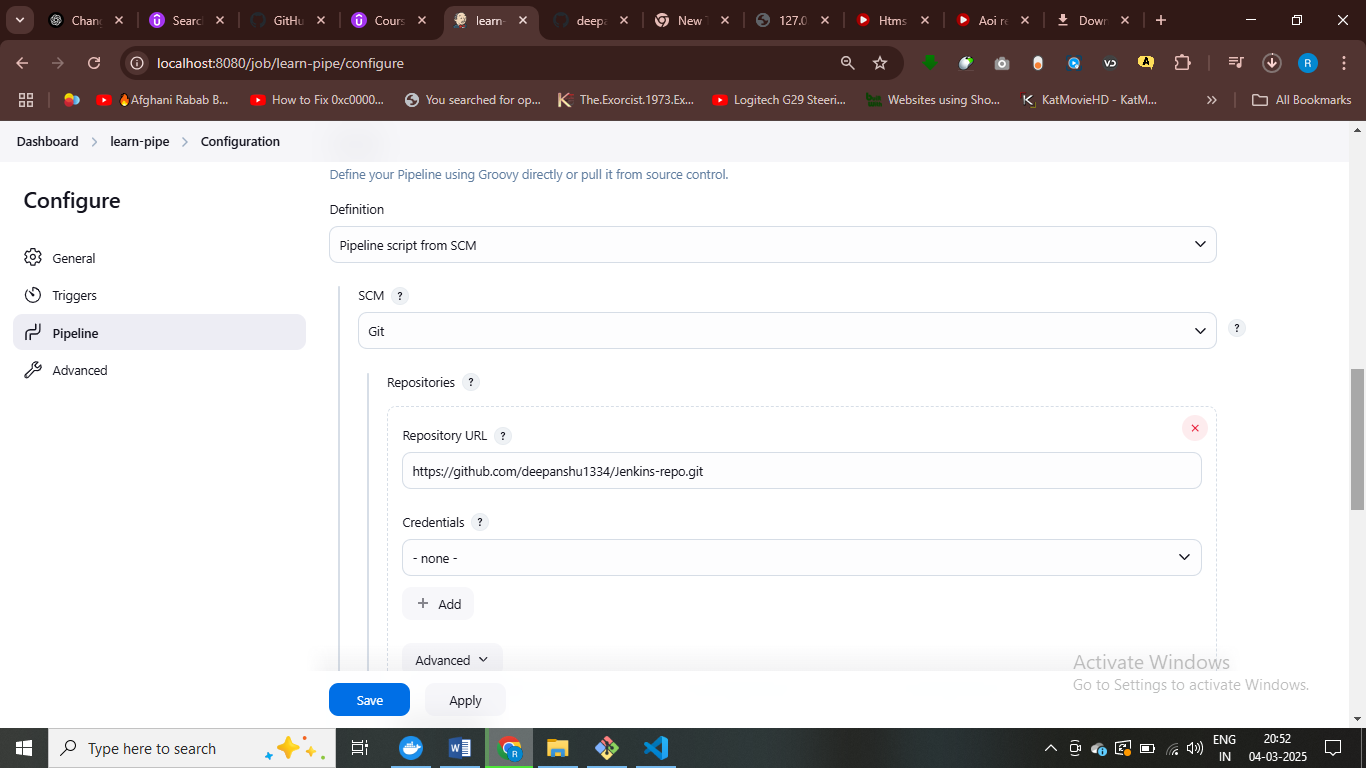
So SQM simply means source code management.

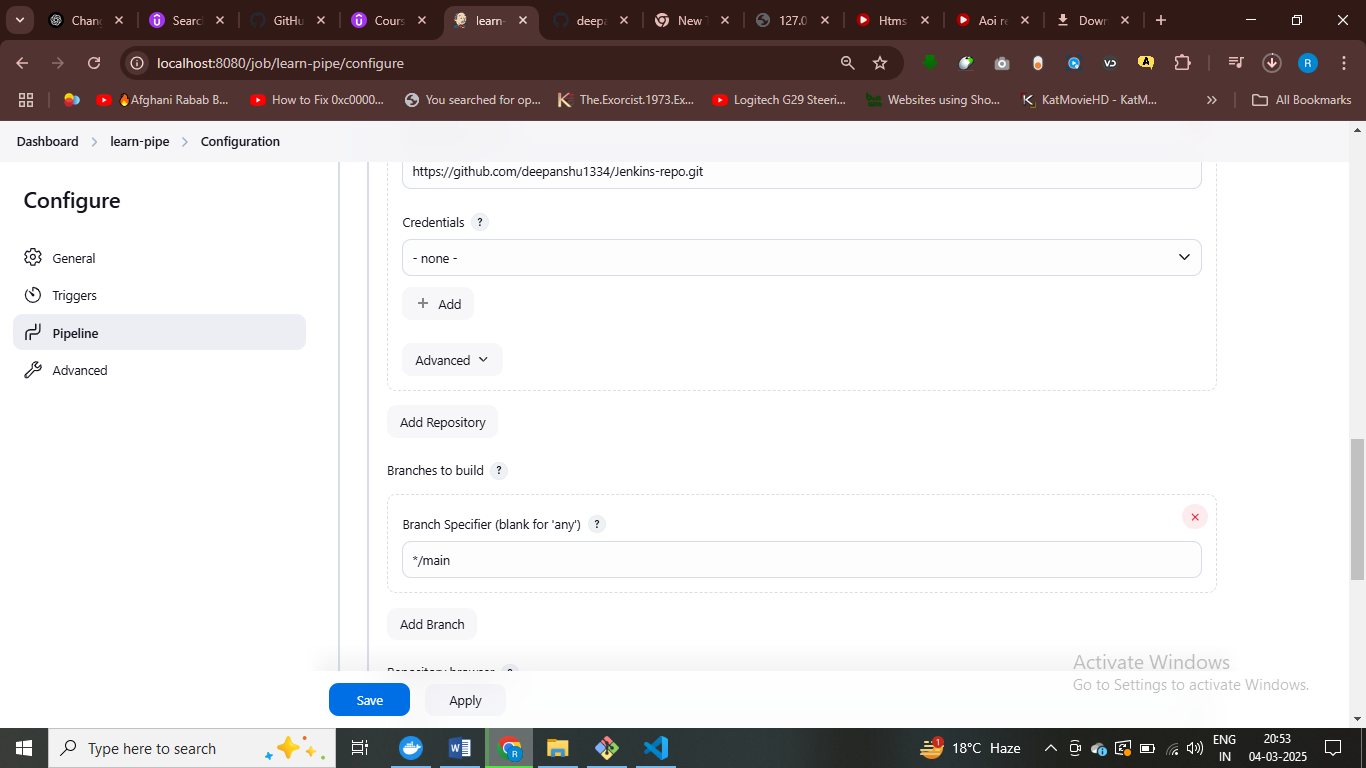
Essentially we're saying we want to get this from somewhere else.

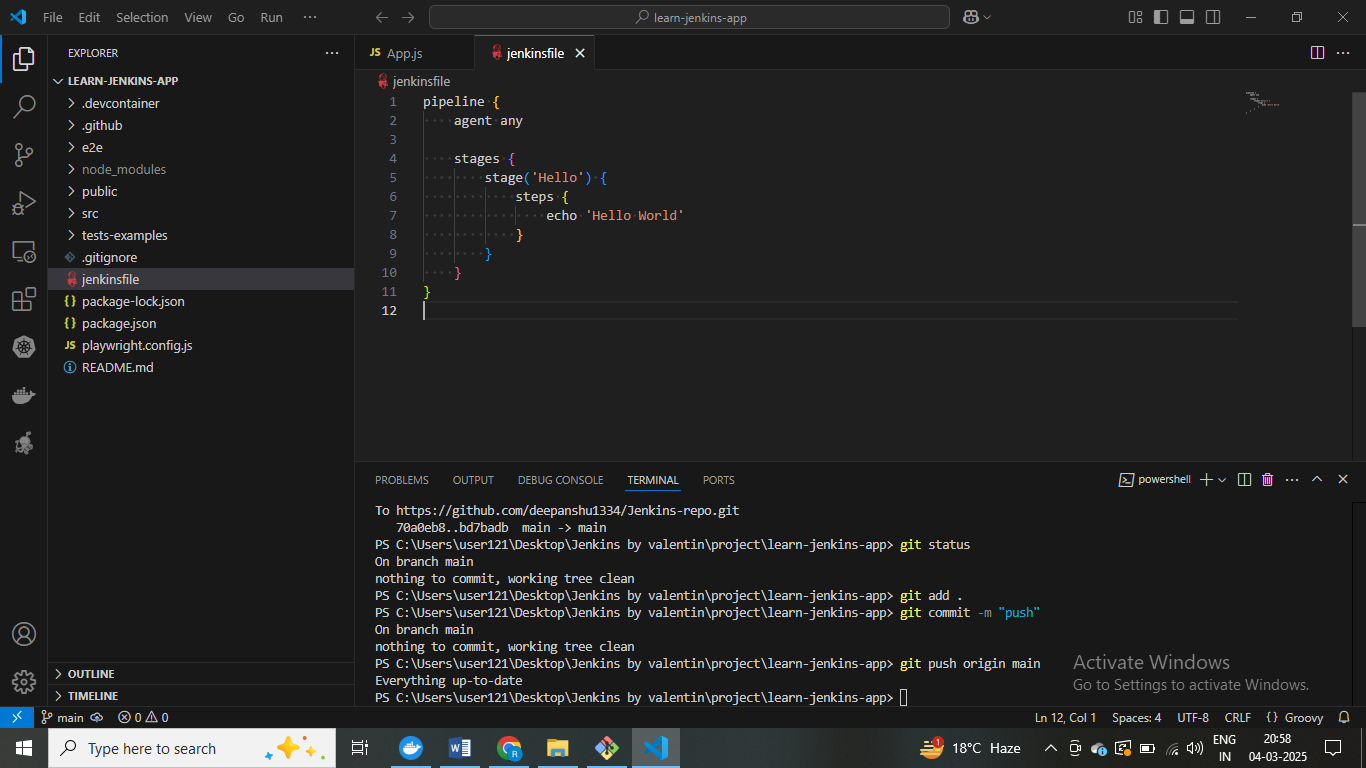
We don't want to define it here.

We want to get it from somewhere.

So the SCM in this case is going to be git.







First build your application , tehn we

We are using agent docker and in docker spcify image

pipeline {

    agent any

    stages {

        stage('Build') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    ls -la

                    node --version

                    npm --version

                    npm ci

                    npm run build

                    ls -la

                '''

            }

        }

    }

}

# we are using docker agent and first we listing all the files ,after this we checking node version , and build the appn

We have stage build and we are using docker as agent to pull node image and listing all files and directories

pipeline {

    agent any

    stages {

        stage('Build') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    ls -la

                    node --version

                    npm --version

                    npm ci

                    npm run build

                    ls -la

                '''

            }

        }

        stage('Test') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    test -f build/index.html

                    npm test

                '''

            }

        }

    }

}

Diffrent new stage added test – checking if index.html exists inside build folder

What is the JUnit test report?

Generally speaking, a JUnit report is a file generated by the JUnit testing framework in Java projects.

And this file typically uses the XML format.

This report provides detailed information about the results of the different test cases, including

a pass or fail status.

Execution times.

Error messages, even like other statistics that are relevant.

This report are crucial for CI CD because it allows us to automatically take a look at the execution

and to identify how many tests have failed, and generally to keep track on what exactly has failed.

Now, we could take a look at the logs and see what has been reported there, but it's much more convenient

to take a look at the consolidated report.

Our project does not use Java, and it doesn't even use the JUnit testing framework.

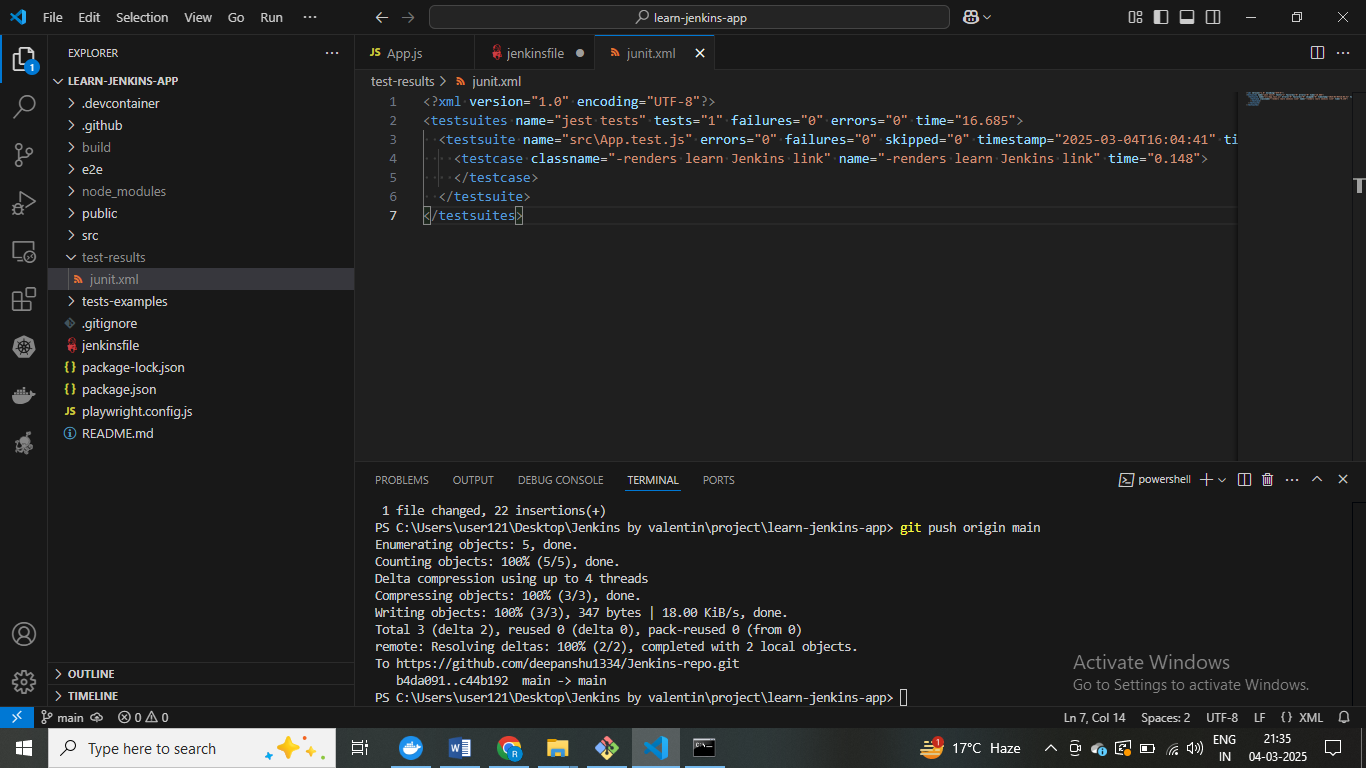
Actually, this JUnit format is so popular that essentially every testing framework has the possibility,

at least to generate it and to store its test results using this format.

When we have executed the npm test command, we have executed the test.

But this test execution has also created a new directory.

And this directory is the test results directory because this is how it is configured.



# it will create junit.xml file after applying – npm test

You will see they will finishing here.

You can add here a post action.

And inside the post action.

We can use always because we want to use this.

Always.

Regardless if the build has been successful or not.

And inside here we can use JUnit and we will specify JUnit where the path is.

All we have to do is write here for example test results forward slash.

J unit XML.

Super important that you double check this path and make sure that you don't have any spelling mistakes,

pipeline {

    agent any

    stages {

        stage('Build') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    ls -la

                    node --version

                    npm --version

                    npm ci

                    npm run build

                    ls -la

                '''

            }

        }

        stage('Test') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    test -f build/index.html

                    npm test

                '''

            }

        }

    }

    post{

        always{

            junit 'test-results/junit.xml'

        }

    }

}

Post always means every time we build . junit test will be executed

Comments in Jenkins file

pipeline {

agent any

stages {

/\*

stage('Build') {

agent {

docker {

image 'node:18-alpine'

reuseNode true

}

}

steps {

sh '''

ls -la

node --version

npm --version

npm ci

npm run build

ls -la

'''

}

}

\*/

stage('Test') {

agent {

docker {

image 'node:18-alpine'

reuseNode true

}

}

steps {

sh '''

#test -f build/index.html

npm test

'''

}

}

}

post {

always {

junit 'test-results/junit.xml'

}

}

}

# Running e2e test

C:\Users\user121\Desktop\Jenkins by valentin\project\learn-jenkins-app>npm run build

> learn-jenkins-app@0.1.0 build

> react-scripts build

Creating an optimized production build...

Browserslist: caniuse-lite is outdated. Please run:

npx update-browserslist-db@latest

Why you should do it regularly: https://github.com/browserslist/update-db#readme

Browserslist: caniuse-lite is outdated. Please run:

npx update-browserslist-db@latest

Why you should do it regularly: https://github.com/browserslist/update-db#readme

Compiled successfully.

File sizes after gzip:

46.64 kB build\static\js\main.31a5a74b.js

1.78 kB build\static\js\787.a1dc0f1b.chunk.js

538 B build\static\css\main.fd921077.css

The project was built assuming it is hosted at /.

You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

You may serve it with a static server:

npm install -g serve

serve -s build

Find out more about deployment here:

https://cra.link/deployment

C:\Users\user121\Desktop\Jenkins by valentin\project\learn-jenkins-app>npm install -g serve

added 88 packages in 18s

24 packages are looking for funding

run `npm fund` for details

C:\Users\user121\Desktop\Jenkins by valentin\project\learn-jenkins-app>serve -s build

┌────────────────────────────────────────┐

For end to end test we need running version of our appn

<https://playwright.dev/docs/docker>

pipeline {

    agent any

    stages {

        /\*

        stage('Build') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    ls -la

                    node --version

                    npm --version

                    npm ci

                    npm run build

                    ls -la

                '''

            }

        }

        \*/

        stage('Test') {

            agent {

                docker {

                    image 'node:18-alpine'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    #test -f build/index.html

                    npm test

                '''

            }

        }

        stage('E2E') {

            agent {

                docker {

                    image 'mcr.microsoft.com/playwright:v1.39.0-jammy'

                    reuseNode true

                }

            }

            steps {

                sh '''

                    npm install serve

                    node\_modules/.bin/serve -s build &

                    sleep 10

                    npx playwright test

                '''

            }

        }

    }

    post {

        always {

            junit 'jest-results/junit.xml'

        }

    }

}

# we are going to run test ,

# as a post actions , we can publish it reports

# Publishing an HTML REPORT

# Blue ocean plugin in Jenkins

<https://www.udemy.com/course/jenkins-ci-cd-pipelines-devops-for-beginners/learn/lecture/42640544#overview>