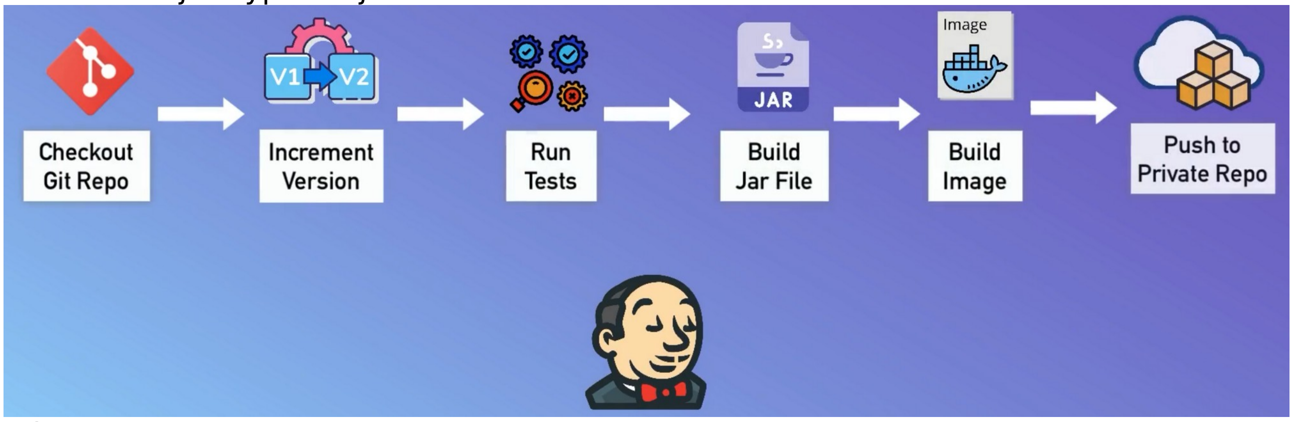
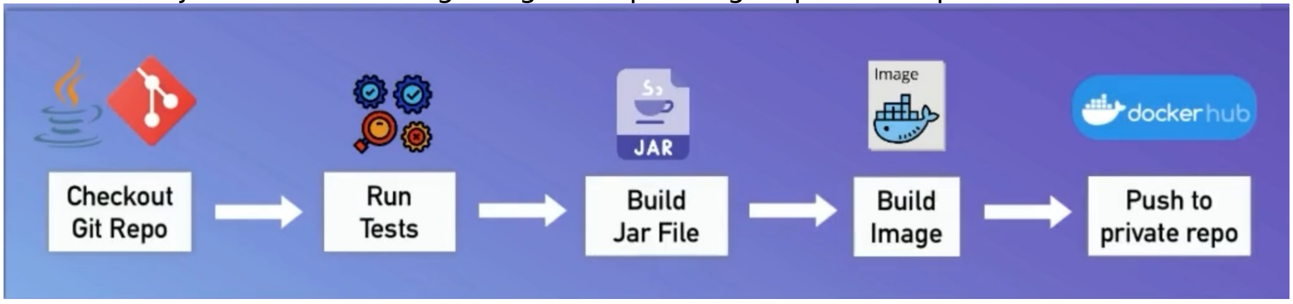
One of the most important concept in devops build automation and continuous integration in devops .
One of the most used Build automation tool jenkins.
1.setting up jenkins in server .
2. Different job types in jenkins



What we'll learn :
1.What is build automation in jenkins
2.Deploy jenkins in digital ocean server.
3.Install and using different build tools in jenkins : eg npm , maven , gradle .
4.Create free style jobs : configure connections and plugins in jenkins and connect to our git repo.
5.Docker in jenkins -> building image and pushing to private repo.



6.Building scripted pipe line
7.Jenkins file in detail.
8.create scripted multiple brach pipe line : which can handle multiple branches of our git repo.
9.configuring automated versioning .

By end of learning this we will be able to solve : How to build a scripted pipe line which is triggered automatically by code changes. And which test and build the application into a docker image and automatically incremented version and pushed the docker image into our private docker repo.

Jenkins shared library to make your pipe line code reusable for other project in company .

What is build automation ?

Build application locally eg build jar file
Build. Docker images
Pushed our image to nexus repo.
As a devops engineer u don't do all of this on your local machine .

When publishing new release manually is challenging why ?

When time come to new release version and to release of artifact .
Stash working changes and move to the master branch - update local state(by git pull_) (where u build the application from _)
U need to make sure u are logged in to the correct docker repo . (If u are going to build a docker image of your application if it is a nexus repo u have to login to docker repo in nexus .
U have to set up the test environment to run the tests .
Excuse those test and wait eg. 30-40 min .
Build a docker image and push to the repo.

So if one of the member in the team has to do this every time u release a new version of the application .

So u want a dedicated server to execute all of these things .

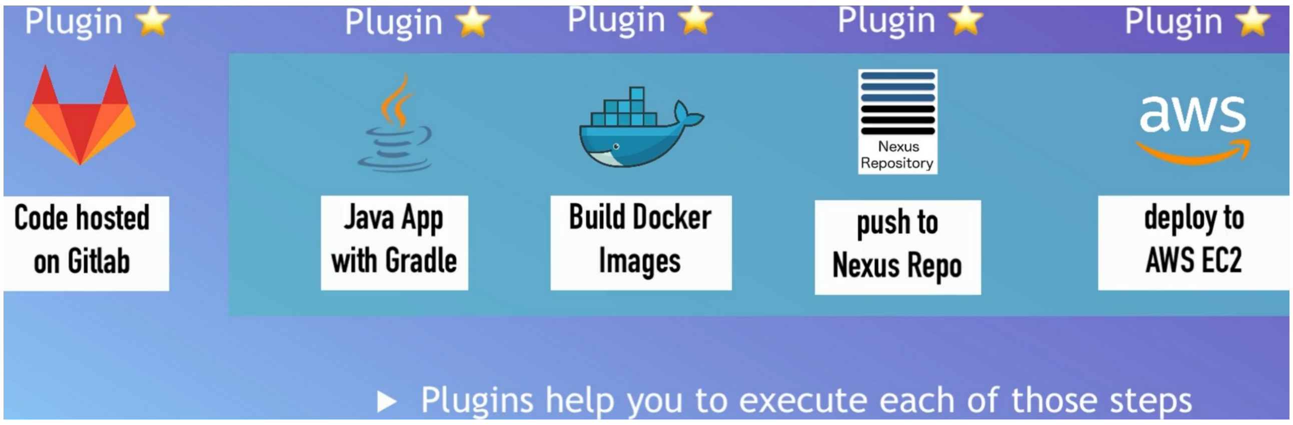
So one the developer checks code changes in the git repo test to be run application version to be build and pushed to a repo automatically . The test environment will be prepared there on the server ,docker credential configured , all the necessary tools must be installed eg: docker , npm ,gradle(to build the app before u build image).
And also u don’t want ssh in these server and execute these commands manually .
I want do all these triggered automatically when u push new changes in the master brach .

And that process of automatically triggering workflow :
Test your code -> build application (artifact ,docker image)->push to repo ->deploy new version to dev server . That process is called build automation .

One of the build automation tools in jenkins it is a software that u install in the dedicated server .
It has a ui for configuration .
So now u can install all tools to used (npm/ grade/maven), docker .
Configure that tasks(run tests , build app, deployment etc) .
Configure automatically trigger of the work flow .

What u can do with jenkins ?
U can run test
U can build artifact
Publish artifact
Deploy artifact .
Send notification to the team (weather the test are successfully or not etc) .
It is a powerful tool to help u manage these build automation process.

Integration with other technologies:
It need to integrate with many other tools .
Like docker , build tools, nexus repo , deployment server .
Jenkins act like a middle man this is handling your code changes to the tested build and deploy to the server . T**here for Jenkins has lot of plugins in order to integrate with all of these technologies.**



How does it all works ?
What do you need to configure ?

To RUN TEST : build tools need to be available :
Eg : with building tools you execute the test commands : npm test , gradlew test ,mvn test .. , configuration test environment (eg : setting up the test DB)
To BUILD APPLICATION : build tools or docker available in jenkins .
Eg: with docker u use docker commands like docker build .
With build tools you execute build command : eg: npm package , gradlew build. ..
TO PUBLISH DOCKER IMAGE : u need to store credentials in Jenkins .
To authenticate to the docker repo .

JENKINS USER SHOULD HAVE ACCESS TO ALL THESE TECHNOLOGIES AND PLATFORMS .

First we need to install jenkins and prepare all this .
Setup is only need to done once .
Plugging and credentials etc. U can use it for different projects .

Install Jenkins on a digitalOcean droplet . ?
There are two way to install Jenkins on a server .
1.install Jenkins directly on the OS . Ie: download package and install on server , create separate Jenkins user on server . Its hard to do it manually .

2. Install Jenkins as a docker container on the server : for that u don’t need to install java first and create User . And configure all of it . U just need to run docker container from jenkins official image . And thats it . Same a have done in nexus as running it as a docker container is way easy because u will have less effort of setting all the things up .

Steps to create droplet for Jenkins digital ocean :
1. Select region closes :
2. Ubuntu
3. 4 :80 : 4 cpu option.
4. Ssh key option tick
5. Create a drop let .

Name server as Jenkins-server

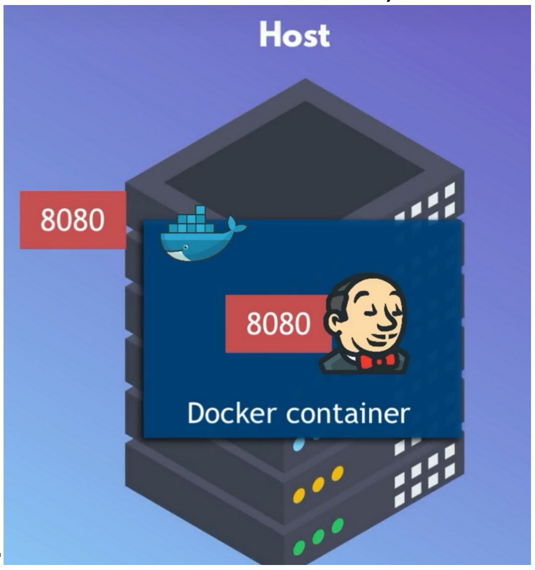
Now attach fire wall to it : networking -> firewall edit ->create fire wall
Create new file wall for Jenkins : named Jenkins-firewall ->
1.ssh.tcp.22.allipv4allipv6 (as my ip address is dynamic and its temporary set as allipv4 and all ipv6) (port 22 should be open as we need to ssh to the server)(these are the ipaddress which has access to this server)
2. Custom / tcp / 8080 (open for for incoming request)(this is where Jenkins application starts and here we are going to expose it on the host as well (and all ipv4 and all ipv6 in bound rule from every where) .

Now choose the droplet which we need to apply this firewall to .

Now we have our server ipv4 address copy that :
Terminal : ssh root@154.5.3.3

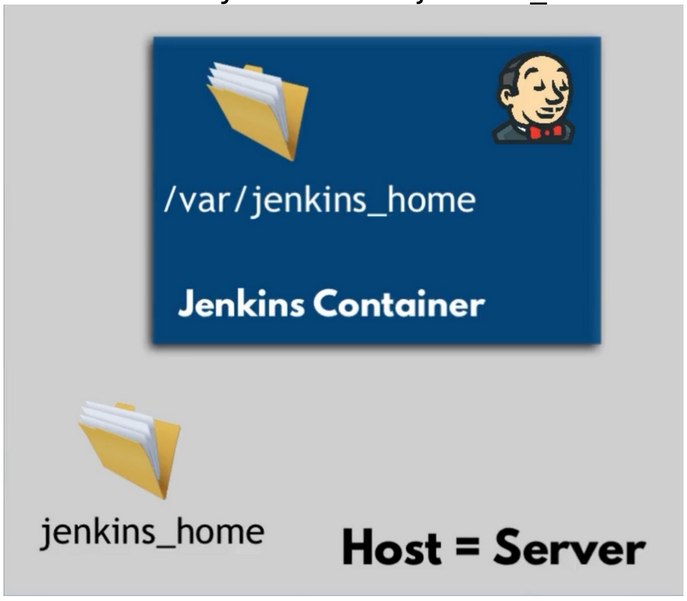
Now this server is fresh and empty there is nothing installed here .
We are installing Jenkins as a docker container there fore we don't need to install java or Jenkins directly only thing we need is docker run time .

First install docker : apt update -> apt install docker.io
Docker run 8080(we are binding it on our host):8080(jenkins will start on port 8080 inside the docker container) so we can access Jenkins application from browser .



We need to open another port in jenkins ie port 50000 this is a port where jenkins master and worker . If u have large work load that u are running with jenkins the port where the communication btw the master and a worker node is happening ; so jenkins when the application in the container starts it will start on two ports 8080 and 50000 and we are binding both of them on our server so for the first time we are not going to use the cluster of jenkins so the port is unused .

-d (which is gonna run the container in background because when we quit the terminal the container will still run in detached mode . And now important part we need to mount volumes so Jenkins just like nexus is gonna store bunch of data when we configure jenkins when we create users and permissions when we create Jenkins jobs to run different work loads , when we install plugins and so on .. all of this is stored as a data .
And if we lose this data we don't have all those.
So we need to persists those data and back them up.
So for that we attack -v (volume) jenkins_home(this is a named volume docker will create a physical path on the server where it will store that data we can reference it by name this jenkins_home will be stored in the host /server



And this will mount to actual direct on the contender
jenkins_home:/var/jenkins_home

And at last we write the name of the jenkins image : jenkins/jenkins:lts(tag) .
This is a official image of jenkins in docker hub .

:: docker run -p 8080:8080 -p 50000:50000 -d -v
jenkins_home:/var/jenkins_home jenkins/jenkins:lts

: docker ps : our jenkins contender ins running and port 8080 is exposed to the server . Ie now we can access jenkins from the browser .

: copy the ip address of jenkins server x.x.x.x:8080
: enter to the docker container of jenkins : docker exec -it containerIDOFjenkins bash

We are loged in as a jenkins user : cat
/var/jenkins_home/secrets/initialAdminPassword
Paste the password on the ui of jenkins login page .
: exit (to exit the jenkins sever)
:on host : docker volume inspect jenkins_home (we can see that password on the name volume in our server as well)
: on jenkins ui : select all the purging suggest ones .
: create our first admin user :

INTRO TO JENKINS UI:
2 roles of Jenkins app;

1. Jenkins administrator(operational or devops team manages all these) :
   administers and manages Jenkins , set up Jenkins clusters ,
Installing all the plugins , backup the data Jenkins generated .

2. Jenkins user (developers or devops team) : creating the actual jobs to run workflow .

In bigger compony these two can be different but for smaller project it can be done by one individual .

As jenkins admin : plugin , creating user , tools etc.
As jenkins user : u are interested into creating new jobs in order to automate your work flow setting up ci cd etc.

INSTALLING BUILD TOOLS :

First thing to do as a jenkins user is to create a job to automate your app workflow .

Eg: java app ;
Java app with maven build tool
So this job needs to run maven commands maven test which will run the test and maven package which will create the jar file .
There for maven needs to be available on Jenkins

Eg: javascript node application .
Node app , run test , package it and push to nexus repo
There for we need npm to be available on Jenkins . (To run npm test and npm pack etc_)

DEPENDING ON YOUR APP PROGRAMMING LANGUAGE , YOU NEED TO HAVE DIFFERENT TOOLS INSTALLED AND CONFIGURE IN JENKINS .

2 way s to install and configure those tools on Jenkins :

1. Jenkins Plugins : just install plugin (via UI) for you tool
2. Install those tools directly on the server : access the remote server where Jenkins is running . Inside the docker container when Jenkins run as a container not in the droplet , inside the container install those tools so that we can execute command on Jenkins jobs .

Configure plugin for maven
Go to manage jenkins->tools on jenkins uI : (here u can manage all your tools managing means u choose which version of tool u want to use) .

Choose maven -> name: maven-3.9 -> version:3.9.2

Install npm and node in jenkins container :

1.enter the Jenkins container from our terminal :
On host -> docker ps (to see the running container)
Enter container as a root user as we want to install npm and by default we enter container as a jenkins user : docker exec -it JenkinscontainerID bash
How ever we need permission to install npm inside that is administrator work

So therefore : docker exec -u 0 -it JenkinsContainerID bash // insider jenkins container as root user .

To see the which linux distribution your server is running :
: cat /etc/issue
: npm update
: apt install curl (using curl run a script that will install npm and nodes .
: curl -sL https://deb.nodesource.com/setup_20.x -o nodesource_setup.sh
Script is downloaded :
Now execute that script : bash nodesource_setup.sh
: apt install nodejs (now we have nodes and npm installed) .(so both of these tools are installed insider jenkins so now they are available for jobs .

Create simple freestyle jobs and plugin config
Now create the jobs in order to run those commands .
: click create job or add new item in jenkins ui -> (name)my-job -> select freestyle job .

Freestyle jobs: the most basic job type in jenkins , straight forward to setup and configure , suitable for small , small -scale projects .

Can be used to test application , build or to push

It lack some advance features provided by newer job type .

We use pipeline or multibranch pipeline in actual projects .

Now configuring my-job -> go to build step (execute shell) : this allow up to use execute normal shell commands inside the jenkins container command line : npm — version .

Involve top level maven target :
Maven 3.9 -> goals ->—version.
Select build now .

Install directly on server more flexible (npm to build tools)
Plugin = limited to provided input field .

Go to the plugin -> available plugin -> search node js install it . -> go back to tools configuration -> nodejs just got added -> for a tool to appear here it need to be installed through a plugin once install via plugin it will be available for configuration .

Add node JS -> my-nodejs ->

Go to my-job -> add build step -> execute node js script . ->installation will be available here in execute nodejs script because of the configuration this option is available here .

Configure a git repository https://gitlab.com/nanuchi/java-maven-app
: your Jenkins job is not connected to any git repository .

On my job -> config -> source code management -> git

Ls /var/jenkins_home/ (inside jenkins container) to see the all tools , plugins installed , jobs, war , users. Etc.
Ls /var/jenkins_home/jobs/my-job/builds

Ls /var/jenkins_home/workspace/my-job :: we can see the git files here eg: pom.xml src from the git repo.

Complete task from git repo in jenkins job .
The job we created is connected to git repo but not to use any thing from the repo.

Go to GitHub -> create new branch -> jenkins-job -> create folder here (freestyle-build.sh= npm —version)
Go back to job config -> copy repo url ->setup credentials -> branch name (jenkins-job)->
Build steps —> execute shell
Chmod +x freestyle-build.sh(add execute permission to the jenkins so that jenkins user can execute it)
./freestyle-build.sh
Save it -> build .

Next part :

Check out git repo Take java maven application —> run test from the project _> build jar file of that application

1. Create a new free style job on Jenkins UI .(java-maven-build)
2. In config build from git copy paste git repo url
3. Choose branch */jenkins.jobs
4. build setup : involve top level maven target (choose this plugin)
5. Maven-3.9
6. Goal test
7. Invoke top level maven target (choose another plugin)
8. Maven - 3.9
9. Goal package
10. Build now
11. So test should run and application jar will be created .
12. Save and build now .
13. Ls /var/jenkins_home/workspace/java-maven-build/target (on jenkins container terminal) here u can see the jar file .
14. All the repo checkout , test and build has been done .

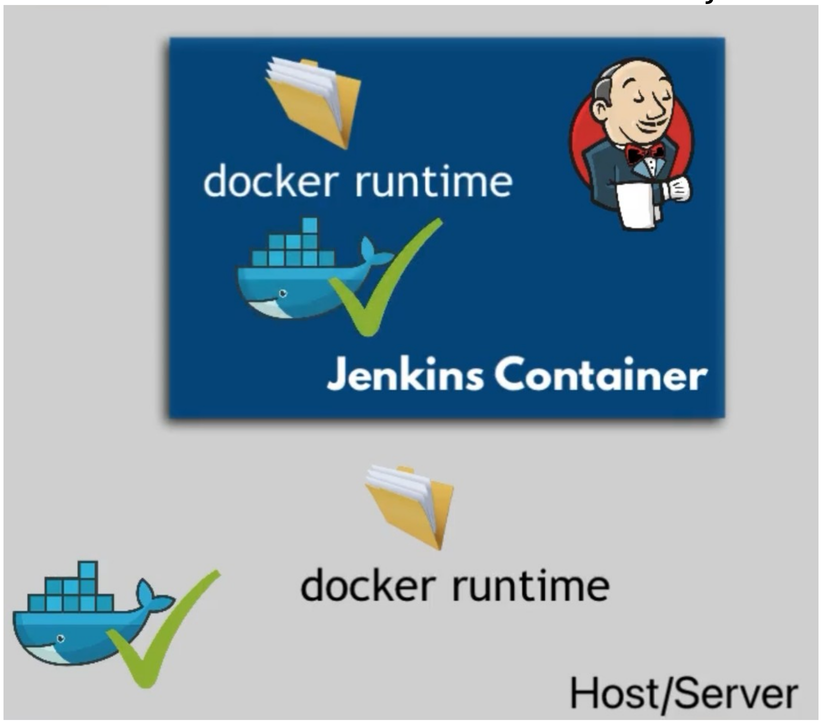
DOCKER IN JENKINS

Make docker available in jenkins :

In most of the scenario we need to build a docker image of our application . That means we need to have docker commands also available in Jenkins in order too use docker commands.

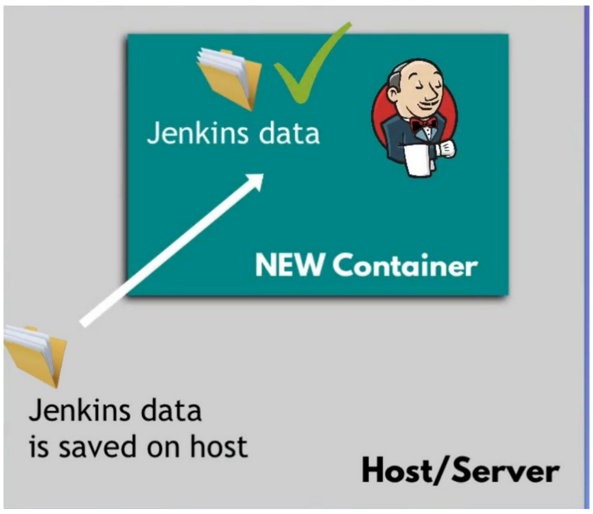
1. Way to do that is attaching the volume to jenkins from the host file .
2.

We are going to mount the runtime of docker from host/server to the Jenkins container so that docker is available in the Jenkins container it self



First correctly running jenkins container we need to start it again with new volumes

:docker stop jenkinsContainerID



We mount the data from the old jenkins container all the credentials , jobs , config , users etc. every thing we created in the old container to new container .

: docker volume ls (on host)

Command to start a new contained will all the necessary volumes :
:docker run -p 8080:8080 -p 50000:50000 -d \
-v jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock jenkins/jenkins:lts
(This will make docker commands available in the jenkins container)

Login to jenkins container as root user : docker exec -u 0 -it containerID bash

So that we can execute docker commands from the jenkins container .
: curl https://get.docker.com/ > dockerinstall && chmod 777 dockerinstall && ./dockerinstall
With this command jenkins container will fetch the latest version of docker from the official site and to the correct permission there and run to the install .

Now I need to set up the correct permissions in the docker.sock file so that we can execute docker inside the Jenkins container as a jenkins user .

Inside jenkins container : ls -l /var/run/docker.sock
:chmod 666(read and write permission for all three parties /var/run/docker.sock
:exit
:login again as jenkins user : docker exec -it containderID bash
:docker pull redis

This means we can use docker command in any of our builds . Just like we did npm .
Go to my_job build config -> I can execute docker command from execute shell ->

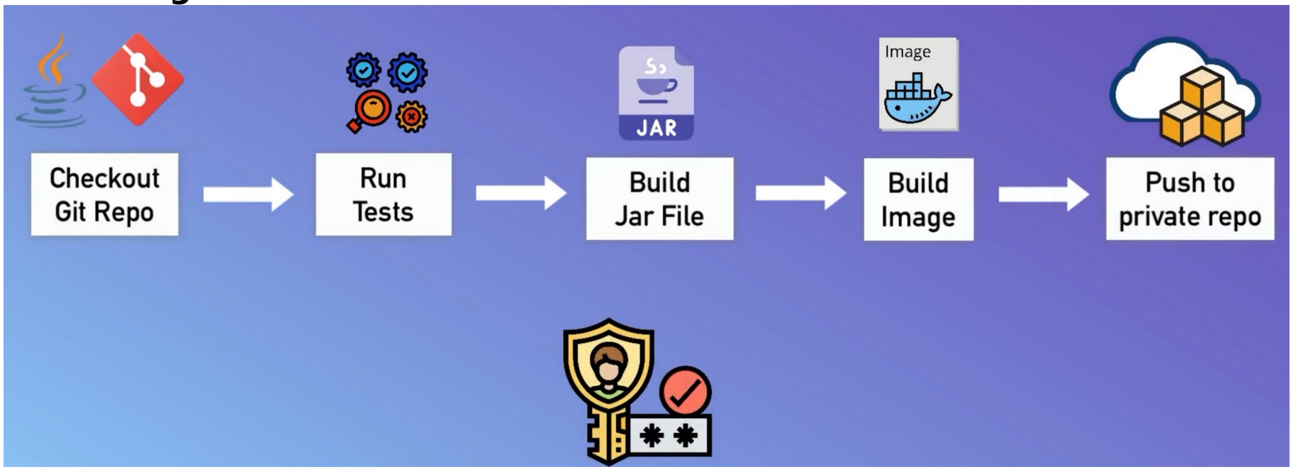
Building docker image from a jar file
1.in Jenkins-jobs branch on repo I have added docker file (docker file takes the jar file that is build with maven package command and create a docker image out of it) in order to build that dock image we need to add docker command in or job configuration

;in java-maven-build : -> execute shell -> here we execute docker command as we execute in command line

As we have made available docker inside jenkins container . -> docker build -t java-maven-app:1.0 . (Add version so that we can identify the image when we build it) in execute shell.

Now build now .

If we go to jenkins container : docker images : -> java-maven-app

Push image to docker hub


1. First step is to create credential for that docker repo .
Create docker hub account , u get one private repo for free.
Configure credentials in jenkins .

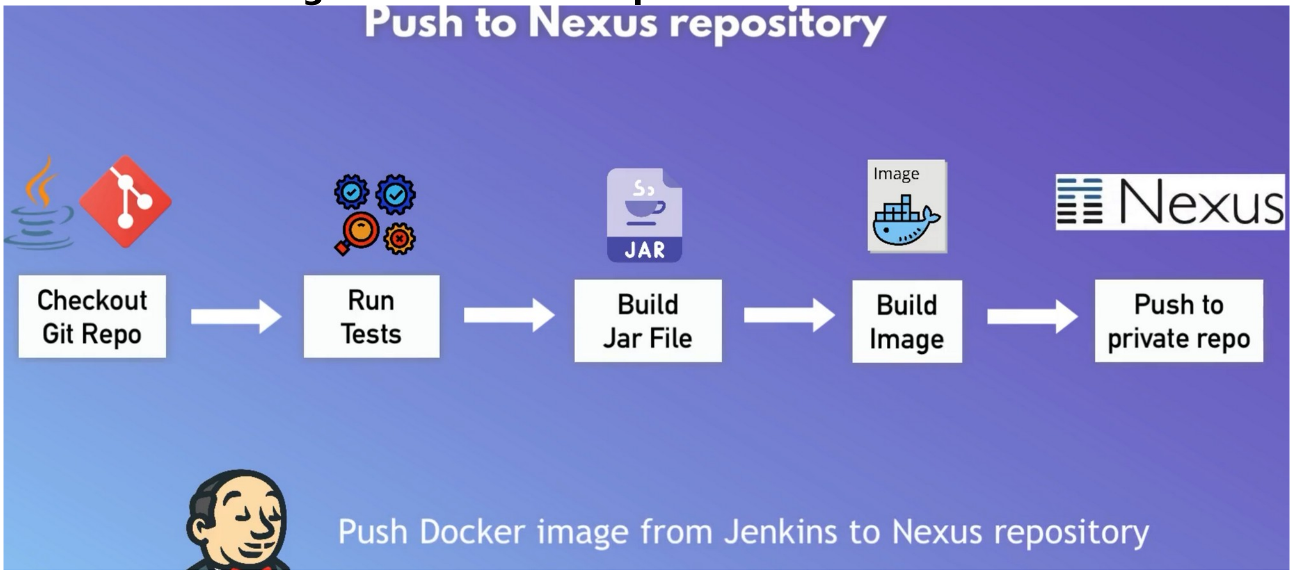
Manage jenkins -> security -> credential in jenkins -> store/system/domain add GitHub user name and password ->go to maven-java-app ->config -> build steps ->execute shell -> to push to repo (tagging the image with the name of the docker hub repo ->
Docker build -t vishalldwivedi/demo-app:1.0 .
Docker login -u $(coreference global var)USERNAME -p $PASSWORD// this will log u in docker hub . Docker hub is the default repo for docker client if u have another repo on aws or nexus u need to specify which host to push image .

Plugin for credential : build environment -> use secret text or files->binding -> username password separated -> add user name and password variables (USERNAME , PASSWORD) specific credential docker hub credential
Docker push vishalldwivedi/demo-app:1.0

Final execute shell config-> this config will build the image with the correct tag with repo name will log in to the repo and push that image to the repo
:docker build -t vishalldwivedi/demo-app:jma-1.0 .
:docker login -u $USERNAME -p $PASSWORD :: better ->echo $PASSWORD | docker login -u $USERNAME —password-stdin (provide password as a standard input best practice and not directly as a command line parameter)
:docker push vishalldwivedi/demo-app:jma-1.0

Save and build now .
This is how u build and push a docker image into a docker hub private repo from a free style build job in jenkins

Push docker image to the nexus repo :



How can we push docker image from jenkins to nexus repo .?
1. Open the nexus droplet server : copyTheIpAddress Of the server :8081
2. Browse section ->docker hosted repo
Previously we are pushing the image from our laptop to nexus repo but now From jenkins server to nexus repo .
So we need to do that configuration on jenkins it self .

Go to the droplet server not in the jenkins container ie on the host because host is where docker is installed we have mounted that info into jenkins container .
:create daemon.json file : daemon.json file is a configuration file used by a docker daemon to specify various settings to customise the behaviour and functionality of the docker daemon according to your specific needs .

: vim /etc/docker/daemon.json ->
{:
“Insecure-registries” : [we have created a docker-hosted repo go to setting and then repository with http connector : 8083 , repo is available on the host or ip address of nexus this for 8083 is open for docker repo specifically . So enter the ip address of nexus and port of the repo in our nexus] -> [“x.x.x.x:8083”]
}
So this is the docker repo which is insecure and we have to register it explicitly .
Now we have to restart docker so that these changes can be applied :
: systemctl restart docker
This will kills all the running container so jenkins container will not be running any more .

:docker start containerJenkinsID

We need to reconfigure the persimmons to the docker.socket file
Root server : ls -l /var/run/docker.sock // permissions changed again as we restart docker .
And we have done that inside the jenkins container :
: docker exec -u 0 -it containerJenkinsId bash
:chmod 666 /var/run/docker.sock
:ls -l /var/run /docker.sock
:exit
Root host : ls -l /var/run/docker.sock :: same on the host itself
Now we have reconfigure docker to allow accessing nexus explicitly allowing our docker repo on nexus as insure registry we have restarted docker readied permissions to the socket file now we can go back and configure our jenkins job .

Reconfigure our build in jenkins to push to the nexus repo instead of docker hub .

1. Manage jenkins -> credentials -> for nexus docker repo.
2. Java-maven-build -> build environment -> user secret text or file -> choose credential (nexus docker repo credential)
3.execute shell =>
:docker build -t nexusrepourl:8083/java-maven-app:1/1 .
nexusrepourl:8083/java-maven-app:1/1 . = this will be the name of the image
java-maven-app:1/1 . = this will be the tag
nexusrepourl:8083/ = this will be the repo name of the image .

:echo $PASSWORD | docker login -u $USERNAME —password-stdin x.x.x.x:8083(specify the repo host)
::docker push nexusrepourl:8083/java-maven-app:1/1

In jenkins container in terminal : if I do : docker images : we can see images already build .

FREESTYLE TO PIPLE LINE JOBS

When creating free style jobs in Jenkins .
Like building -> java app ->running test -> building a docker image -> pushing the docker image to the repo .

Coming practice in jenkins with free style jobs is actually to have them execute them one individual step.
Eg: one free style job for for running integration test
Second freestyle job for building the application
Another one for deploying the app

Freestyle jobs are like the stand alone job that execute one single step .

So jenkins user use to chain these multiple free style jobs to execute one after the another .

Go inside the config part in the java-maven-build -> config-> eg: if it only does maven-package u can choose =-> build other project to create other free style job ->
Name it -> trigger only if the build is stable .

Here , first the maven package is done then lets say second job that build docker image .

This is how the jobs are schedule before the pipe lines /

With these freestyle job -> freestyle job -> freestyle job (chained together)
Limitation : jenkins was build during the time when ui based tools are more popular than scripting your configuration
For each functionality of your build u will have multiple plugins which is limited to input field of plugins .
These chained freestyle jobs are not suitable for complex work flow .

So something much more maintainable and reliable and much more flexible was need specially with the emerge of ci cd and infrastructure as a code(from UI config to scripted config) concept .

As a devops engineers u should we working with pipe line jobs not freestyle jobs.

PIPELINE JOBS : suitable for ci/cd , scripting pipe line as code ,

This is last time we are using nexus server : from next we will use docker hub or ECR .

INTRO TO PIPLE LINE JOBS :

Create pipe line on jenkins UI .
Go to configuration view of pipeline
1.connect this pipeline build to the git repo .
2. Go to pipeline -> we do configuration in pipe line with groovy language scripting .

Groovy is a programming language fro java platform .
3. Groovy is a language we are writing pipeline configuration with .
4. Groovy sandbox-> security feature that provide a restricted execution environment
, u can execute limited no of groovy functions , without needing approval from jenkins user .
5. Best practice for infrastructure as a code is to have all the configuration in our code itself .ie: pipe line code in your git repo .
6. So switch from pipeline script to people line script from source code management(sCM)
7.config git repo -> past url from git java-maven-app _. -> set credentials -> set branch (jenkins-job)-> jenkins will search for a root file called jenkinsfile which will contains the groovy scripts that configure the pipeline
8. Save this pipeline .
9.build now .
10 create a new file in the git repo -> Jenkinsfile ->

Here we will write the groovy script for our pipeline job .

Pipeline syntax :
1. Scripted : first syntax for jenkinsfile , groovy engine ,advanced scripting capabilities ,high flexibility., if no familiar with groovy its difficult to start with .

```
Eg: node{
// groovy script
}
```

2. Declarative: more recent addition ,easy to get started with not that powerful , u have pre defined structure .

```
Eg: pipeline{
         Agent any
        Stages{
              Stage("build"){
                 Steps{
                     }
}
Stage("test"){
                 Steps{
                     }
}
Stage("deploy"){
                 Steps{
                     }
}

}
}
```

Here pipeline must be on top level ,
agent where to execute ,revelent fro jenkins cluster
Stages : where whole work is happening

In steps : are the script that actually execute some command in the jenkins server.

```
Eg: steps{
Sh npm install
Sh npm build
} for js app
```

```
Eg: stage("build"){
Steps{
Echo 'building the applicatoin'
}
}
```

11. My-pipeline -> build now .

12. In a ui view those stages are displayed separately in the pipe line build uI . With there own logs and status of execution . First stage declarative checkout is a step when we add url of the git repo .

Advantages of using a pipeline job :
When want to execute 2 task parallel
You need user input
Conditional statement like when
Set variable
No limitations
Pipeline is a super set of multiple freestyle jobs more simple way less maintenance effort.

Disadvantages of chained freestyles jobs :
Relay on plugins
Different plugins in different jobs .
Manage those plugins

Freestyle jobs are for one singular step managing all those 7 jobs and plugins and configuration etc.
Eg : to switch from maven to gradle u will have to change at least 4 to 5 jobs.

Jenkins file syntax:
Post attribute in jenkinsfile

```
pipeline{
         Agent any
        Stages{
              Stage("build"){
                  Steps{
                      }
}
Stage("test"){
                  Steps{
                      }
}
Stage("deploy"){
                  Steps{
                    }
}
Post {

//execute some logic after all the stages are done
Always{
// execute always now matter if the build fail or not.
//eg: sending mail to team about the status of build condition
}
Success{
// write script only reverent when the script only in succeed build
}
Failure{
// // write script only reverent when the script only when failed build

}

}
}
}
```

Define conditional for each stage :

```
CODE_CHANGES = getGitChanges()// some groovy script what tells if code
changed
pipeline{
        Agent any
       Stages{
             Stage(“build”){
                 Steps{
                     }
}
Stage(“test”){
When {
Expression{
   BRANCH_NAME = ‘dev’ | | BRANCH_NAME = ‘dev’ // test will only execute in
the dev
   BRANCH_NAME = ‘dev’ && CODE_CHANGES = true;
branch .
}
}
                Steps{
                    }
}
Stage(“deploy”){

                Steps{
                    }
}
}
}
```

Environment variable : jenkins provide some environment variable out of the box
Eg: like current branch name , which build no for versioning
How do u know what variable u have available from jenkins. ?
x.x.x.x:8080/env-vars.html
Will provide u with the list of all the EV u can use In your jenkins file .

U ca define your own EV in Jenkins file ->with attribute environment

```
pipeline{
        Agent any
Environment {
// EV u define here will be available for all the stages in your pipeline
NEW_VERSION = ‘1.3.0’
}
       Stages{
            Stage(“build”){
                Steps{
                     Echo ‘building app’
                   echo “ building version ${NEW_VERSION}” // this is called
interpreted of ////variable
                    }
}
Stage(“test”){
                Steps{
                    }
}
Stage(“deploy”){
                Steps{
                    }
}
}
}
```

Using credentials in jenkins file :
// if your u deploying your application in some dev server u need some credential for that :

1. Define credentials in jenkins GUI

Or u can do it in your environment block

: SERVER_CREDENTAILS = credentials(‘’)
: credentials is a method or function that binds the credentials to your env variable .

For that u need to install credentials binding plugin

:manage jenkins -> credentials -> add credential -> set user name and password -> set id
Server-credential (name) create this I will be able to use that credential in side the Jenkins file .
Pass that reference id inside :->
SERVER_CREDENTAILS = credentials (‘server-credential')
And now pass that env variable in that deploy stage :

```
pipeline{
        Agent any
Environment {
// EV u define here will be available for all the stages in your pipeline
NEW_VERSION = ‘1.3.0’
SERVER_CREDENTAILS = credentials(‘’)
}
       Stages{
            Stage(“build”){
                Steps{
                      Echo ‘building app’
                    echo “ building version ${NEW_VERSION}” // this is called
interpreted of ////variable
                    }
}
Stage(“test”){
                Steps{
                    }
}
Stage(“deploy”){
                Steps{
                 echo ‘app is deploying ‘
                 echo “deploying  with ${SERVER_CREDENTAILS}”
                 sh  “${SERVER_CREDENTAILS}”
                    }
}
}
}
```

Or wrapper syntax :
As we need that server credential only in the deploy stage :

```
stage(“deploy”){
 Steps{
   Echo ‘deploying app’
   WithCredentials([
UsernamePassword(credentials:’ server-credential’ ,usernameVariable : USER ,
passwordVariable : PWD) ] ){
Sh “some script  ${USER} ${PWD}”
}
}
```

In manage jenkins -> plugins -> I have credential plugin for create plugins inside jerkins uI . Also have credential binding plugins u use those plugins inside my jenkinsfile through env variables.

Tools attributes for build tools :
Access build tools for your projects.eg : maven , gradle , npm , jenkins so when u are building your application u want to have those tools available .
Eg: In build stage u will run sh maven install
Add those tools available in the tool attribute

U need to preinstall these tools injenkins =>manage Jenkins-> tools -maven-3.9
Name of the installation u need to provide in your jenkins file .

```
pipeline{
        Agent any
Tools{
// only 3 build tools available that jenkins file support : maven , gradle and jdk

Maven  “maven-3.9” // this will make maven command available in all the stages.
Gradle
Jdk

}
       Stages{
            Stage(“build”){
                Steps{
                    }
}
Stage(“test”){
                Steps{
                    }
}
Stage(“deploy”){
                Steps{
                    }
}
}
}
```

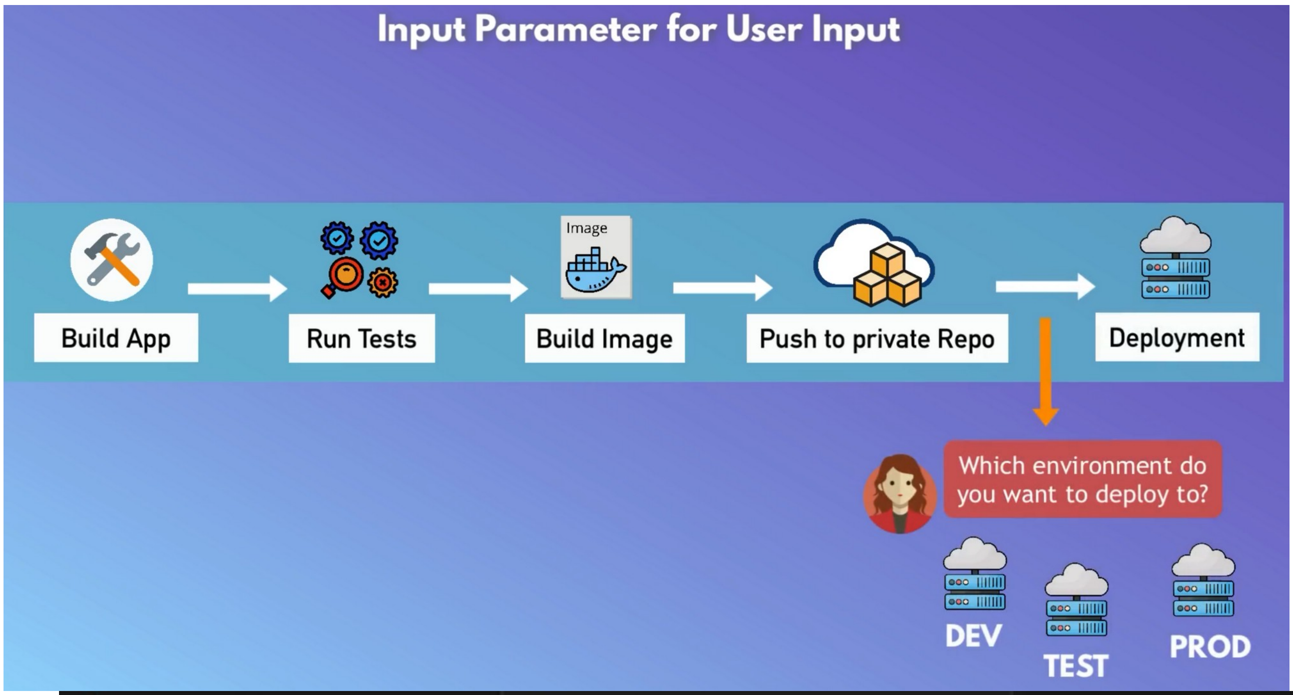
Parameters in Jenkinsfile :

If u want to provide some external config u want to provide to your build to change some behaviour eg: deploy your app to a staging server. And u want to select which version of the app u want to deploy .
Allow user to input parameters before the build starts .

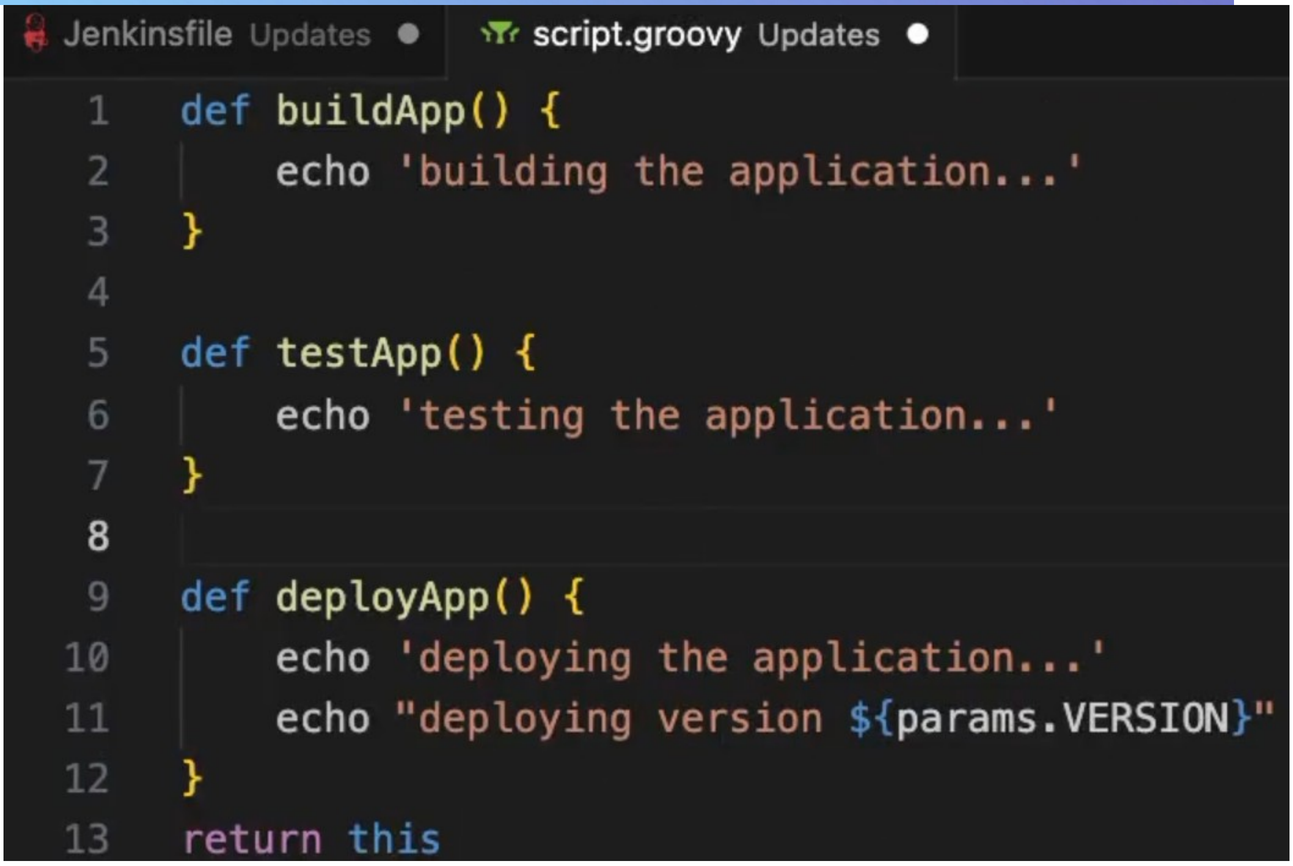
```
pipeline{
        Agent any
Parameters{
string(name:’VERSION’,defalutValue:’’,description :  ‘ version to deploy on prod
‘ )
Choice(name:’VERSION’, choices : [‘1.2.3’ , ‘1.2.4’] , description : ‘ ‘)
booleanParam(name: ‘executeTest’ , defaultValue : true , description:’ ‘ )
}
      Stages{
            Stage(“build”){
                Steps{
                    }
}
Stage(“test”){
 When{
        expression{
                 params.executeTest == true // only then execute this stage else
skip this stage.
        }
}
                Steps{
                    }
}
Stage(“deploy”){
                Steps{
                 echo ‘deploying the app’
                 echo “deploying version ${params.VERSION}”
                    }
}
}
}
```

Commet the changes in the git jenkinsfile .

Using external groovy scripts

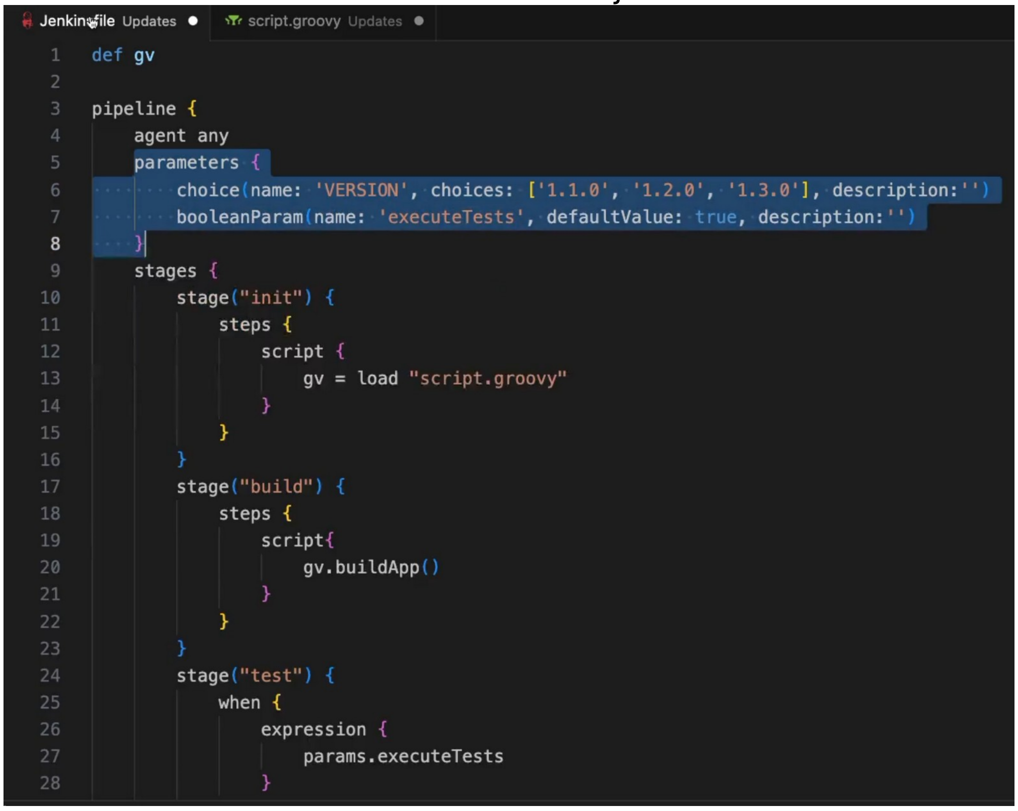


:

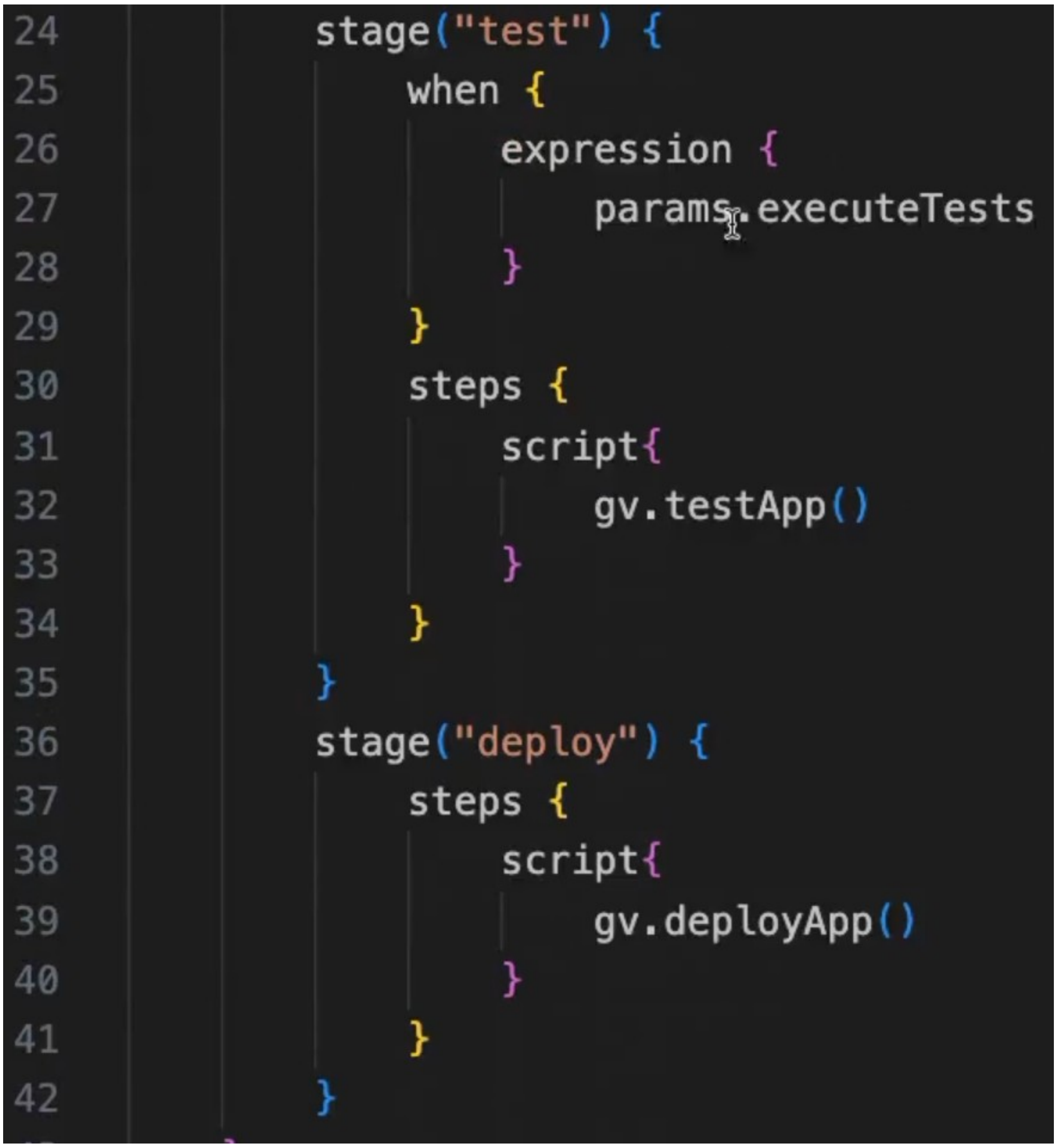


```groovy
def buildApp() {
    echo 'building the application...'
}

def testApp() {
    echo 'testing the application...'
}

def deployApp() {
    echo 'deploying the application...'
    echo "deploying version ${params.VERSION}"
}
return this
```

All env variable that are accessible in jenkinsfile are available in groovy script



```
def gv

pipeline {
    agent any
    parameters {
        choice(name: 'VERSION', choices: ['1.1.0', '1.2.0', '1.3.0'], description:'')
        booleanParam(name: 'executeTests', defaultValue: true, description:'')
    }
    stages {
        stage("init") {
            steps {
                script {
                    gv = load "script.groovy"
                }
            }
        }
        stage("build") {
            steps {
                script{
                    gv.buildApp()
                }
            }
        }
        stage("test") {
            when {
                expression {
                    params.executeTests
                }
```



```
        stage("test") {
            when {
                expression {
                    params.executeTests
                }
            }
            steps {
                script{
                    gv.testApp()
                }
            }
        }
        stage("deploy") {
            steps {
                script{
                    gv.deployApp()
                }
            }
        }
```

Exported each stage functionality in groovy script . This will make our Jenkins file more slim and structured .

Add script.groovy file in git .

If I want to test some jenkins file changes without pushing those change in the repo there is a replay section where u can edit the jerkins file and test application .

U want user to choose which server these build artifact / image need to be deployed   on
How we can choose that >?

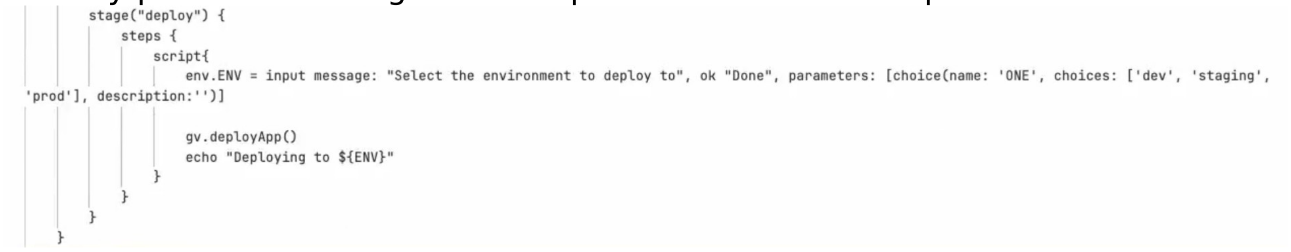
1. Go to jenkins file :

Eg: in deployment phase we want to let use to select the version :

```
stage(“deploy”){
        input{
                message “Select the environment to deploy to “
                ok “Evn selected”
                parameters{
                        Choice(name:’ONE’, choices : [‘dev’ , ‘staging’] , description :
‘ ‘)
                        Choice(name:’TWO’, choices : [‘dev’ , ‘staging’] , description :
‘ ‘)


        }
        }
        steps{
                script{
                        gv.deployApp()
                        echo “Deploying to ${ONE}”
                        echo “Deploying to ${TWO}”

        }
        }
}
```

// this parameter will only be available in this deploy stage
// inside the parameter we can duplicate these choice give the name : unique

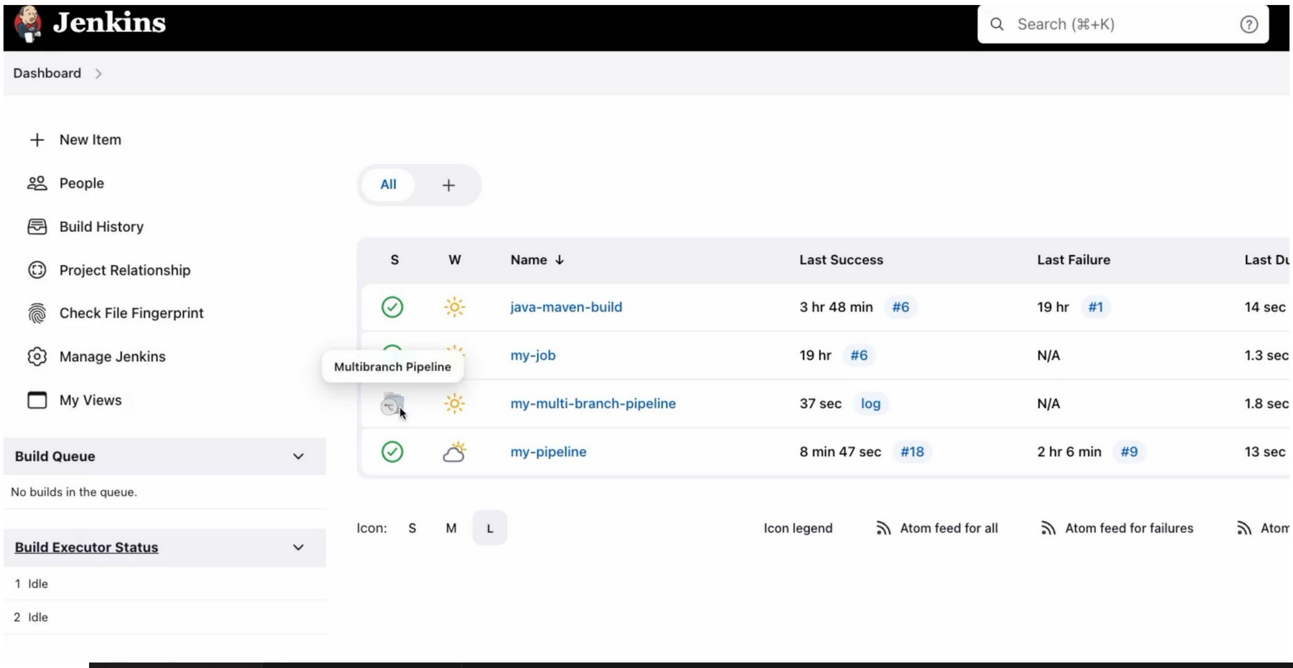
Directly pass the message and the parameters in the script ==:>



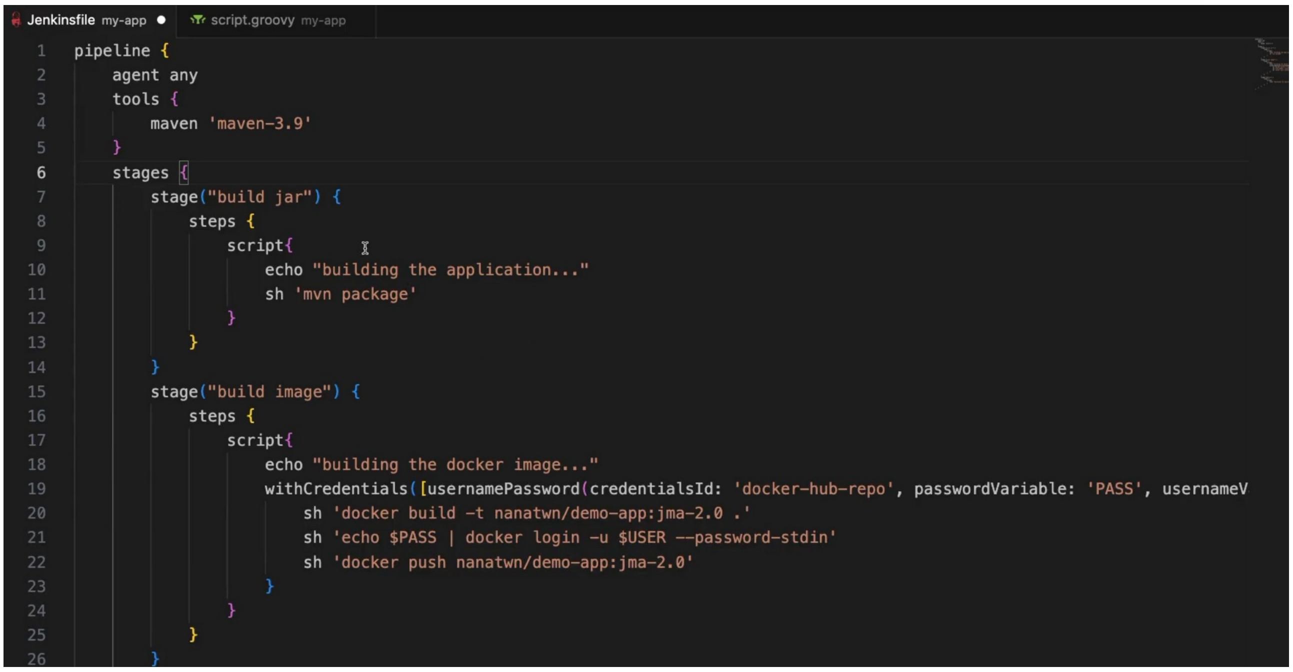
```
        stage("deploy") {
            steps {
                script{
                    env.ENV = input message: "Select the environment to deploy to", ok "Done", parameters: [choice(name: 'ONE', choices: ['dev', 'staging',
'prod'], description:'')]

                    gv.deployApp()
                    echo "Deploying to ${ENV}"
                }
            }
        }
    }
```

CREATE A FULL PIPE LINE FROM A JENKINS FILE :

1. Build jar -> stage = where the maven build command will be executed .

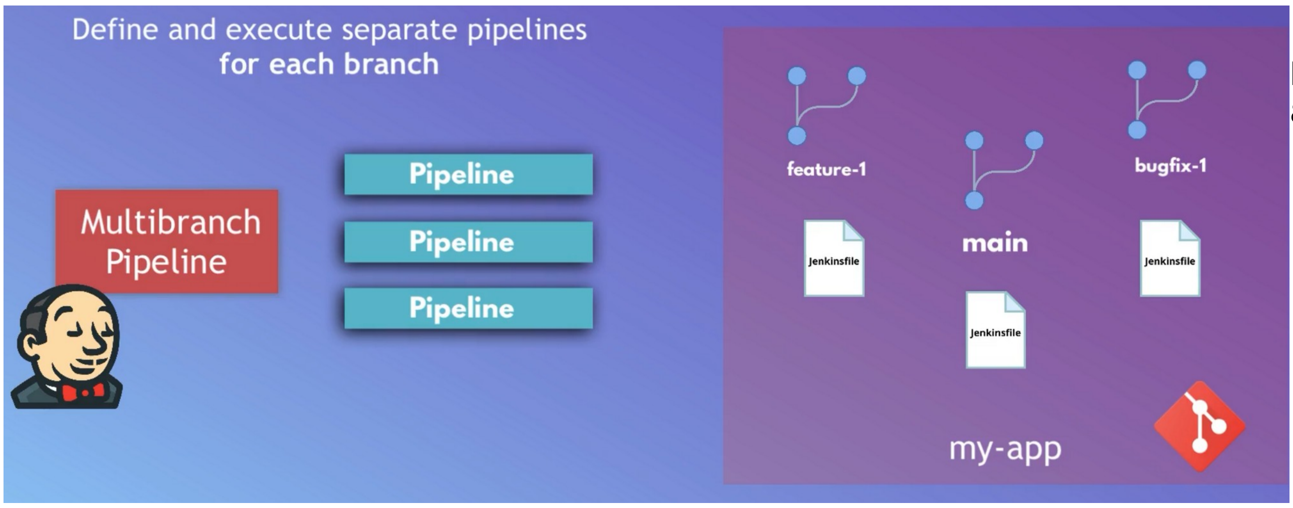


S	W	Name ↓	Last Success	Last Failure	Last Du
✓	☀	java-maven-build	3 hr 48 min #6	19 hr #1	14 sec
✓	☀	my-job	19 hr #6	N/A	1.3 sec
	☀	my-multi-branch-pipeline	37 sec log	N/A	1.8 sec
✓	⛅	my-pipeline	8 min 47 sec #18	2 hr 6 min #9	13 sec



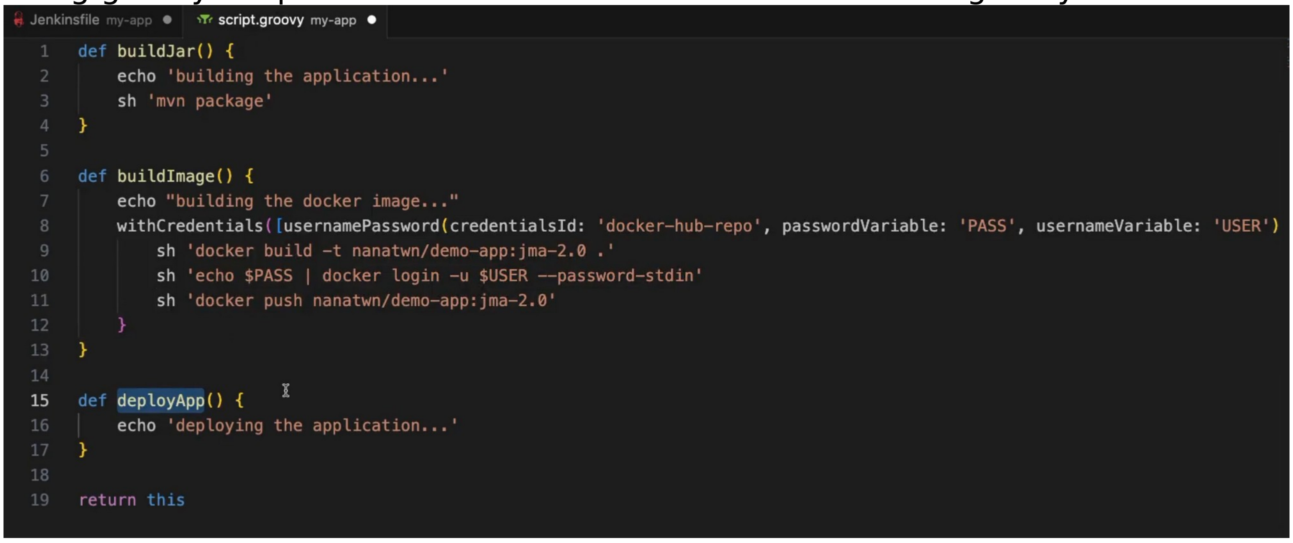
```
pipeline {
    agent any
    tools {
        maven 'maven-3.9'
    }
    stages {
        stage("build jar") {
            steps {
                script{
                    echo "building the application..."
                    sh 'mvn package'
                }
            }
        }
        stage("build image") {
            steps {
                script{
                    echo "building the docker image..."
                    withCredentials([usernamePassword(credentialsId: 'docker-hub-repo', passwordVariable: 'PASS', usernameV
                        sh 'docker build -t nanatwn/demo-app:jma-2.0 .'
                        sh 'echo $PASS | docker login -u $USER --password-stdin'
                        sh 'docker push nanatwn/demo-app:jma-2.0'
                    }
                }
            }
        }
```

, usernameVariable:’USER’)]){

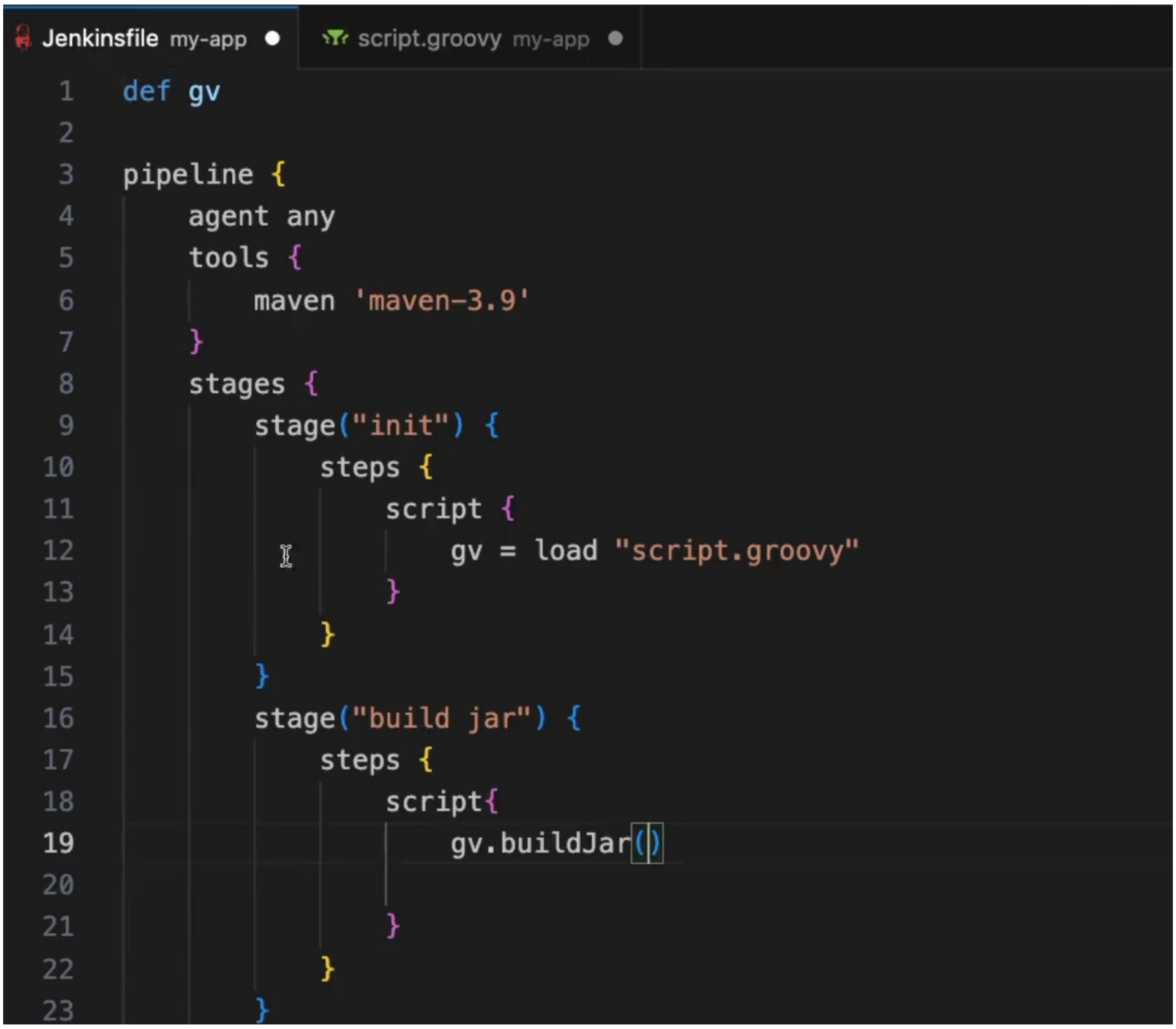


IF U are

using groovy script than u can extract all these info on that groovy file



```groovy
def buildJar() {
    echo 'building the application...'
    sh 'mvn package'
}

def buildImage() {
    echo "building the docker image..."
    withCredentials([usernamePassword(credentialsId: 'docker-hub-repo', passwordVariable: 'PASS', usernameVariable: 'USER')
        sh 'docker build -t nanatwn/demo-app:jma-2.0 .'
        sh 'echo $PASS | docker login -u $USER --password-stdin'
        sh 'docker push nanatwn/demo-app:jma-2.0'
    }
}

def deployApp() {
    echo 'deploying the application...'
}

return this
```

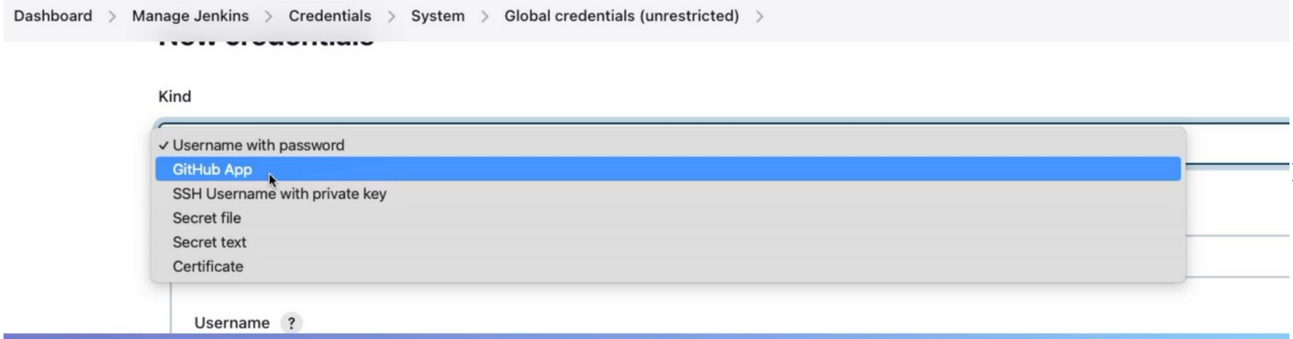


```
def gv

pipeline {
    agent any
    tools {
        maven 'maven-3.9'
    }
    stages {
        stage("init") {
            steps {
                script {
                    gv = load "script.groovy"
                }
            }
        }
        stage("build jar") {
            steps {
                script{
                    gv.buildJar()

                }
            }
        }
```

Now we have a cleaner Jenkins file where we are just referencing the functions from the external script and all of the functions logic are defined in the groovy script .

Introduction to multi branch pipeline :

Till now we have build a pipeline for one branch : ie : jenkins-job .

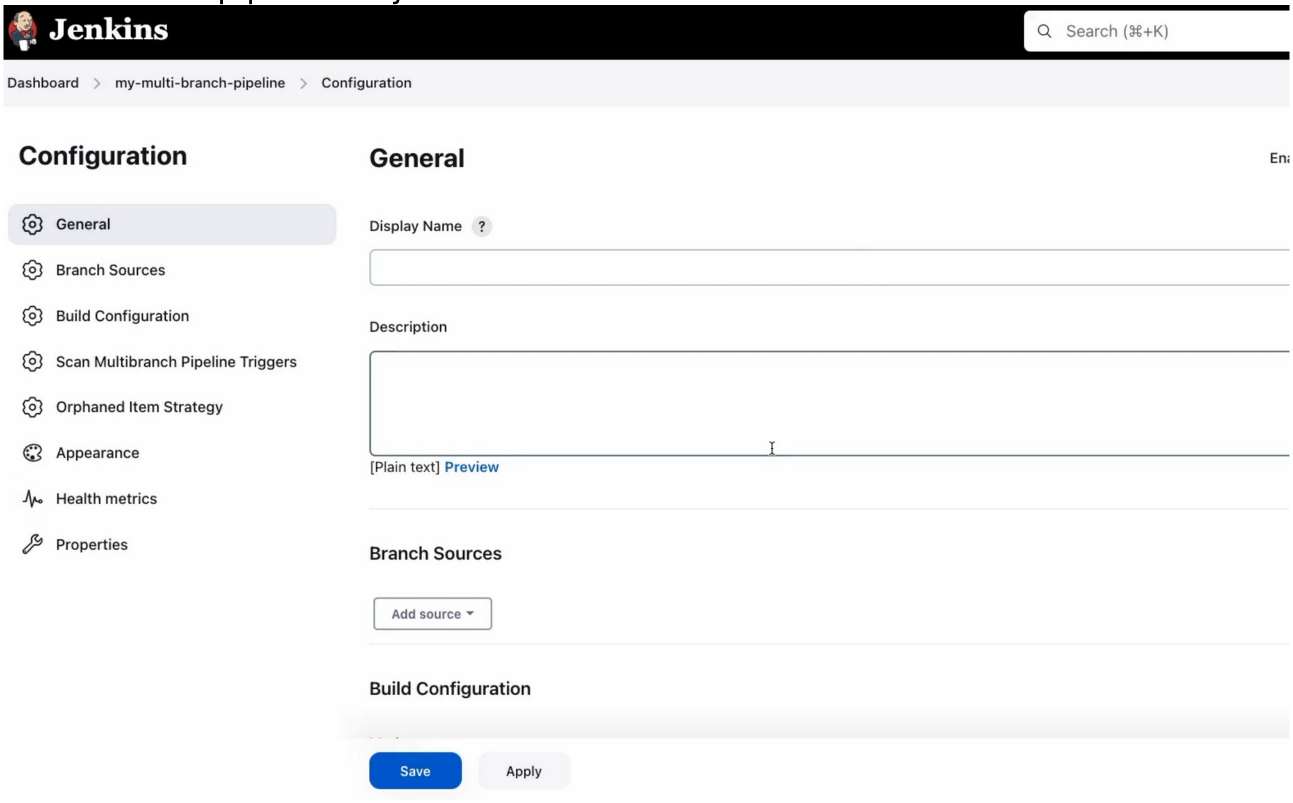

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Kind

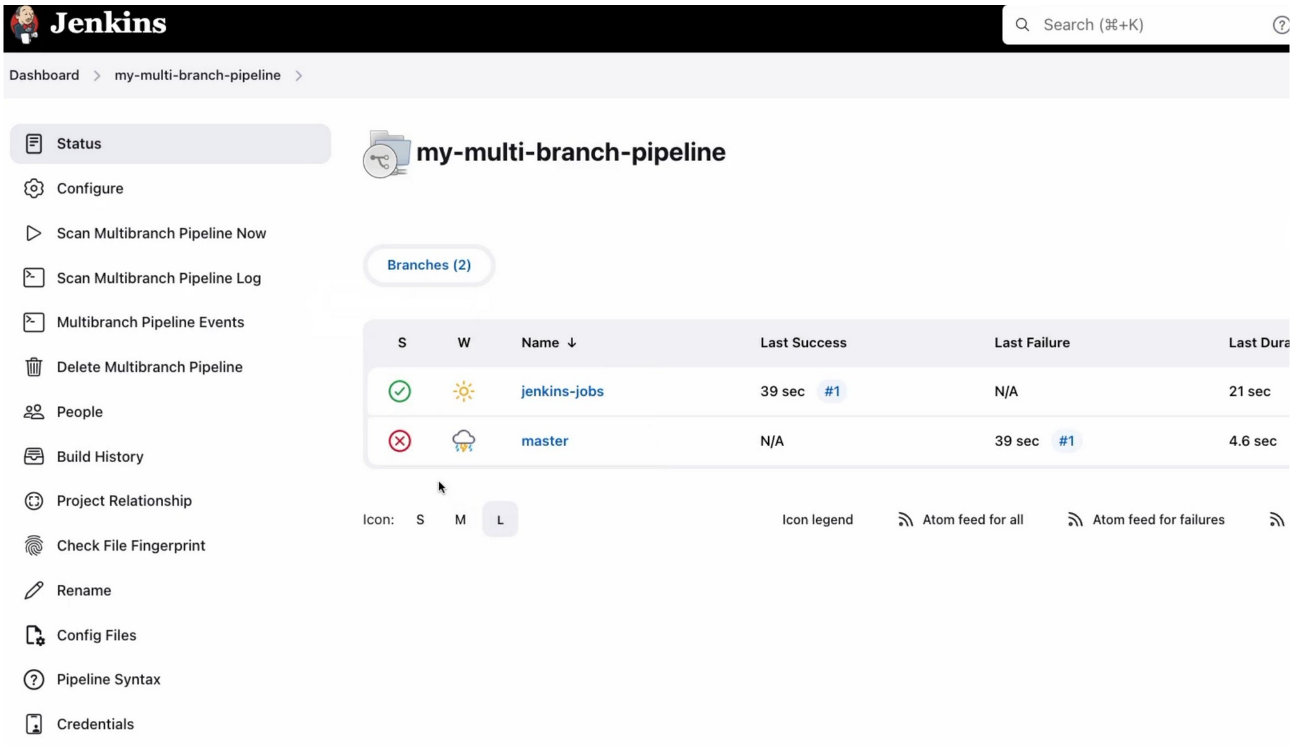
1. C
r
e
a
t
e

Credentials Types
Username & Password
Certificate
Secret File
New types based on plugins

multi branch pipeline in jenkins ->



2. This is the configuration for multi branch pipeline .
3. Branch source -> add git source for our pipeline ->
Add repo url -> add credentials. -> in behaviour (we define which branches from that repo we need to build) -> filter by name with regular expression ->
4. Rest of the configuration should be defined in the jenkins file ->
Save it -> then automatically multi branch file scan will run when ever It finds a jenkinsfile in the branch it build the branch



Jenkins

Search (⌘+K)

Dashboard > my-multi-branch-pipeline >

- Status
- Configure
- Scan Multibranch Pipeline Now
- Scan Multibranch Pipeline Log
- Multibranch Pipeline Events
- Delete Multibranch Pipeline
- People
- Build History
- Project Relationship
- Check File Fingerprint
- Rename
- Config Files
- Pipeline Syntax
- Credentials

my-multi-branch-pipeline

Branches (2)

S	W	Name ↓	Last Success	Last Failure	Last Dura
✓	☀	jenkins-jobs	39 sec #1	N/A	21 sec
✗	🌧	master	N/A	39 sec #1	4.6 sec

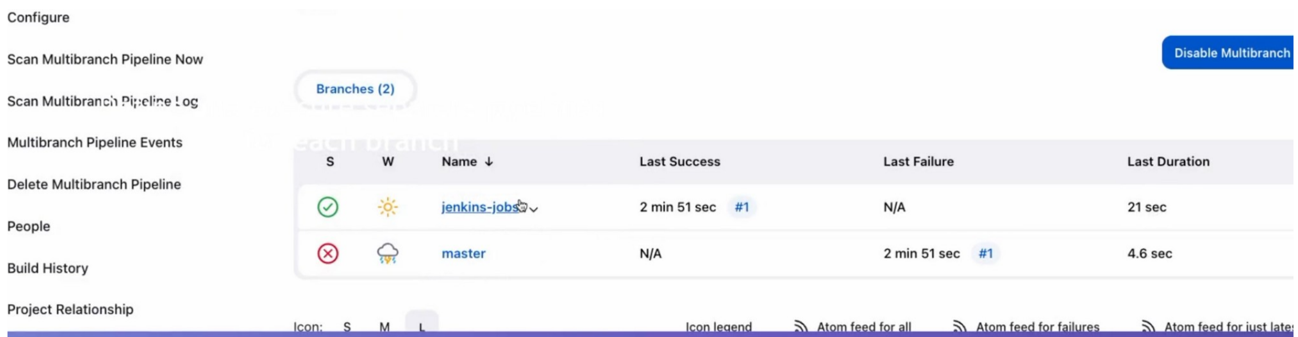
Icon: S M L

Icon legend Atom feed for all Atom feed for failures

Pipeline of each branch in my-multi-branch-pipeline

Technically we have have multipipeline define in a regular pipe line in the config option .

Usually in these type of projects u will have a one jenkinsfile that all the branches share .

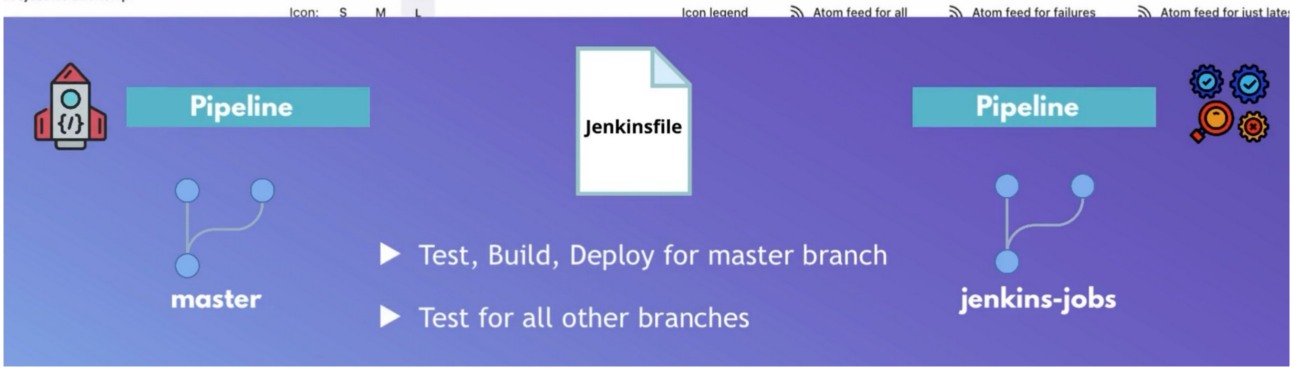

Configure
Scan Multibranch Pipeline Now
Scan Multibranch Pipeline Log
Multibranch Pipeline Events
Delete Multibranch Pipeline
People
Build History
Project Relationship

Disable Multibranch

Branches (2)

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀	jenkins-jobs	2 min 51 sec #1	N/A	21 sec
✗	⛈	master	N/A	2 min 51 sec #1	4.6 sec

Icon: S M L　Icon legend　Atom feed for all　Atom feed for failures　Atom feed for just late


Pipeline
Jenkinsfile
Pipeline
master
Test, Build, Deploy for master branch
Test for all other branches
jenkins-jobs

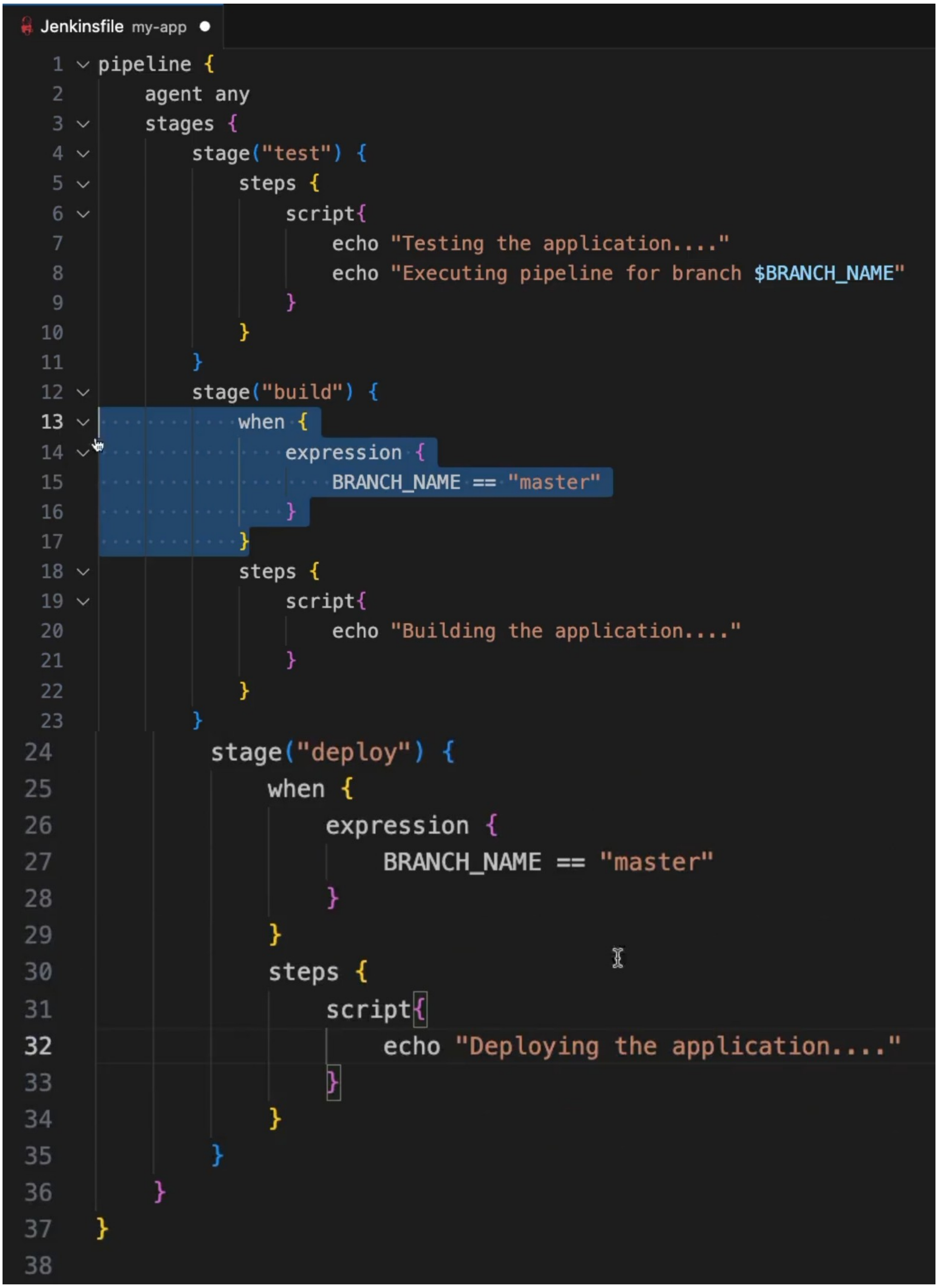
**** I want to only run test for all other branches but I want my master branch to run test , build and deploy .****

BRANCH BASED LOGIC FOR MULTIBRANCH PIPELINE:

IN Jenkins job-branch weather to execute the brach or not we need to edit that Jenkinsfile on Jenkins job - branch .

Our logic should be like test should execute for all the branches :

: BRANCH_NAME (EVN -VARIABLE) IS only available in multi branch pipeline jobs and not the regulate pipeline .
This holds the value of which one of the branches it is holding currently .



```
pipeline {
    agent any
    stages {
        stage("test") {
            steps {
                script{
                    echo "Testing the application...."
                    echo "Executing pipeline for branch $BRANCH_NAME"
                }
            }
        }
        stage("build") {
            when {
                expression {
                    BRANCH_NAME == "master"
                }
            }
            steps {
                script{
                    echo "Building the application...."
                }
            }
        }
        stage("deploy") {
            when {
                expression {
                    BRANCH_NAME == "master"
                }
            }
            steps {
                script{
                    echo "Deploying the application...."
                }
            }
        }
    }
}

```

Commit change in the jenkinsfile in jenkins-job branch :
We need these changes in the master branch as well -> go to master branch ->edit these changes in the Jenkins file of master branch .

Now go to jenkins UI -> in multi branch pipeline and do scan multi branch pipeline now .
Now u can see the to branch which contains jenkinsfile both of them are build

In logs we can see in jenkins-job pipeline : test is successful but build and deploys skipped due to when expression .

In logs we can see in master pipeline : test build and deploy successful .

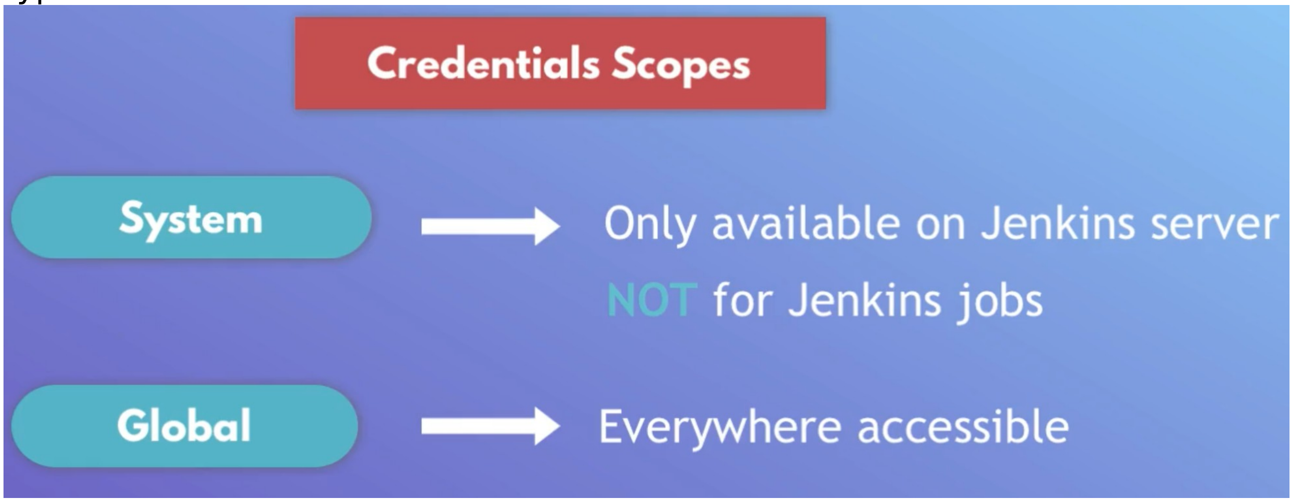
Eg: if I create new branch in github and do again scan multi branch pipeline -> we can see new branch pipeline . -> its logs says test successful build and deploy not successful due to when expression .

WRAP UP JENKINS JOB :
We have created three different type of build jobs

1. Free style jobs : executing a single task as a stand alone job eg: in continuous delivery workflow each step like test , build , deploy are individual free style jobs
2. Regular pipeline : as a simpler solution to have multiple jobs for each of the stages to have one single job for each of those steps u will get a nice view of each stages in your pipeline . Better solution of ci/cd . One regular pipeline meant for one single branch .
3. Multi branch pipeline : parent of pipeline job . One multibranch pipeline meant for multiple branch. If we want to build multiple branches in our git repo we have to have multiple pipeline jobs for each of those branches .

CREDENTIALS IN JENKINS :

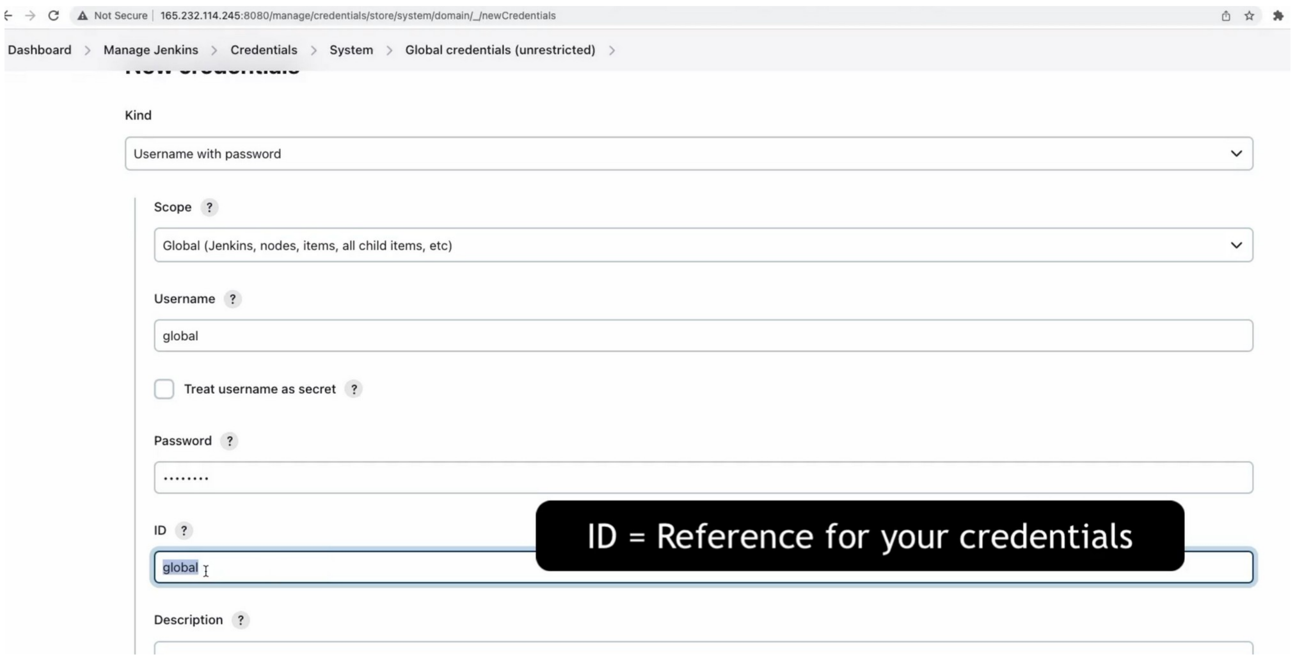
We have created credentials for our jobs in manage jenkins and we have reference them through credential plugins all of them are user name password type of credentials .


Credentials Scopes

System → Only available on Jenkins server
NOT for Jenkins jobs

Global → Everywhere accessible

We have seen two scope , system and global and there is also a third scope which is limited to your project u can see it only in multi branch pipe line.



Not Secure | 165.232.114.245:8080/manage/credentials/store/system/domain/_/newCredentials

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

global

☐ Treat username as secret ?

Password ?

••••••••

ID ?

global

ID = Reference for your credentials

Description ?