JENKINS SHARED LIBRARY :

What is jenkins shared library ? Why we need it?

Eg: we have a 10 microservices in java-maven all these micro services contribute to one application . In jenkins we would need multi branch pipelines for each of those micro services. We would build each micro service as a separate application .

That means u have multiple projects that have there own jenkins files . That are building a pipeline in jenkins also some of those microservices are also java maven application and we are building docker images from all of those , we have logic inside those jenkinsfiles that are 90 % same

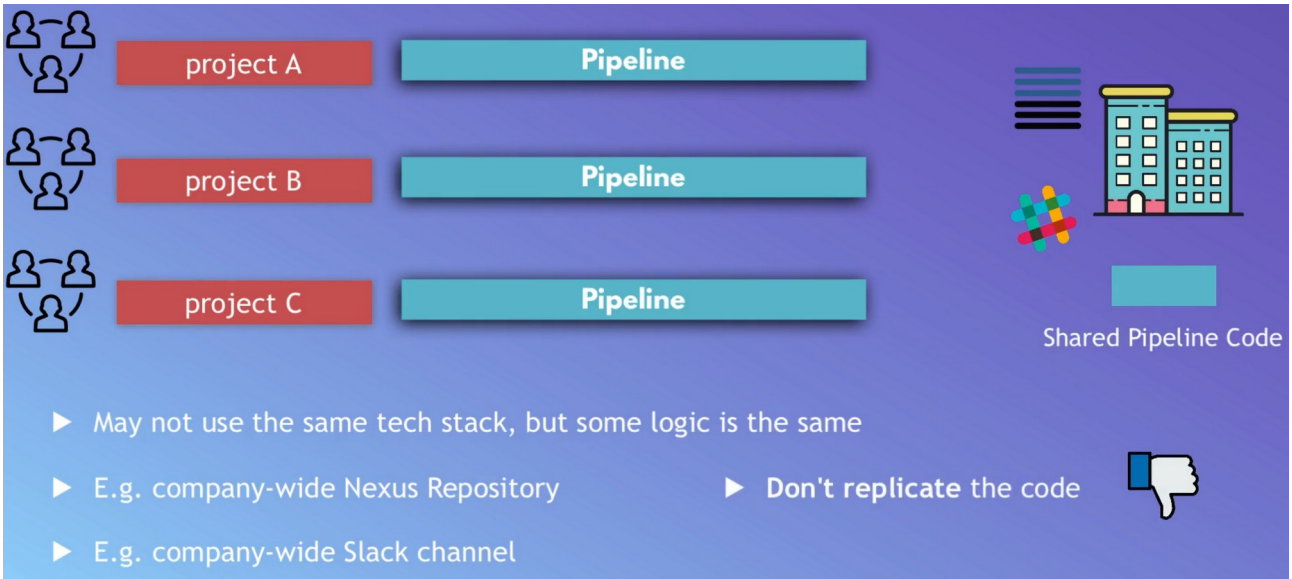
script.groovy 520 bytes Edit Replace Delet

```
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

```

All of this logic would gonna be same for all of those microservices . That means we need to copy that same jenkins file and groovy script and replicate that all 10 different projects. Solution for that is jenkins shared library .

: JSL is a extension to a pipeline ,
It has its own repo which is written in groovy
We can write all the logic which is shared across many different application and reference that logic in jenkins file in each projects .

ANOTHER USE CASE OF JSL:

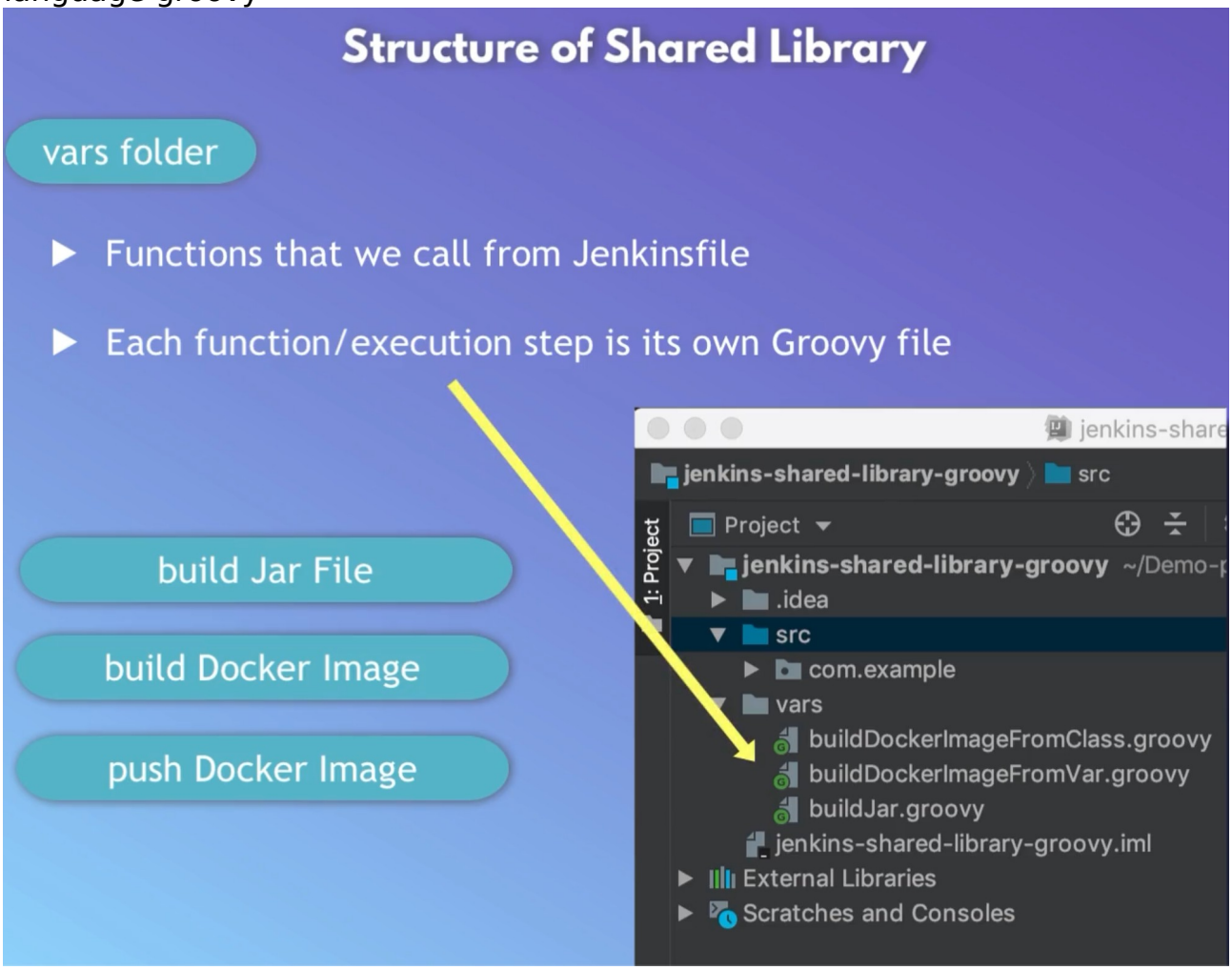


:code reusability , easy maintenance , consistency and standardisation , faster pipeline creation . Improved collaboration in the company even they are not working in the same application .

1. Create a Jenkins shared library
2. Make shared library available in jenkins
3.use shared library In Jenkins file in diff projects .

Create shared library project/repo :
1.create repo -> write groovy code there .
2. Make this shared library available for our project in jenkins .
3.use shared library in Jenkins to extend the pipeline .

Create repo :
1. Create a new project(named jenkins-shared-library) in ide -> choose language groovy



Src folder: helper code
Resource folder - use external library , non groovy files .

2. Create a var folder as new directory ->

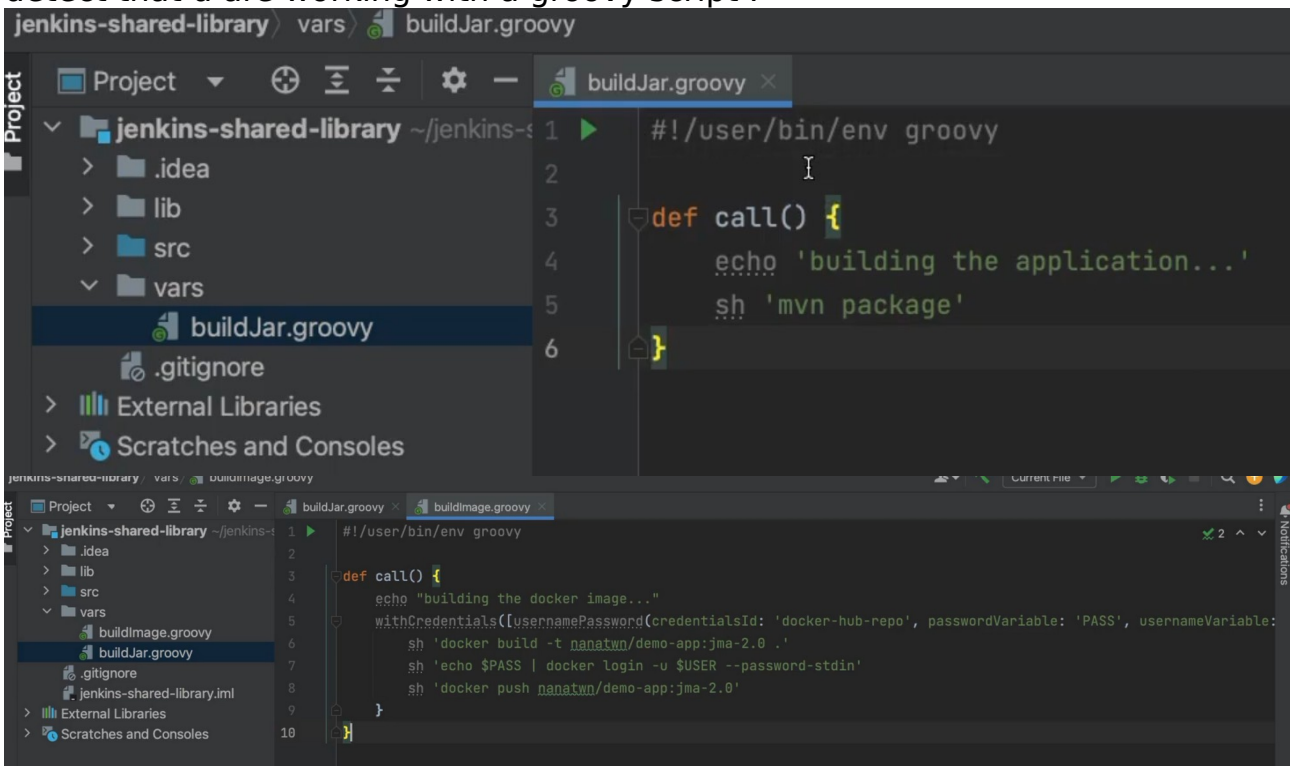
Function name in the groovy file should be same in Jenkins shared library functions

script.groovy 520 bytes — Edit | Replace | Delete

```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
```

so that it can refer easily .

Add : #!/user/bin/env groovy :on top of .groovy file : this will let your editor detect that u are working with a groovy script .



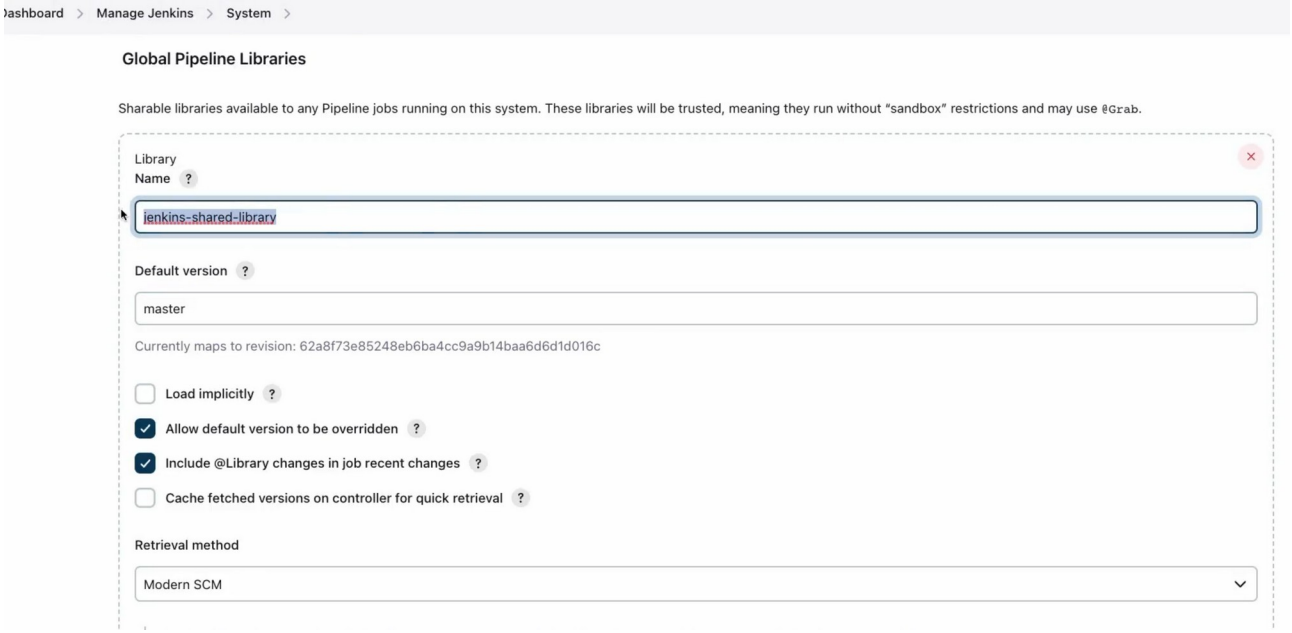
2. Now create a repo in git hub so that we can use it in jenkins -> push our local repo to remote git repo. ->
Cd jenkins-shared-libraray -> git init -> git remote add origin repoURL ->git add .
->git commit -m”abc” -> git push -u origin master
3. Make shared library globally available (to make it available to use in jenkins file):

Go to manage jenkins -> system -> global pipeline libraries -> name it jenkins-shared-library->set default version (master branch = any new change to the repo will be immediately available for whole jenkins pipeline) -> retrieval method (modern SCM)->

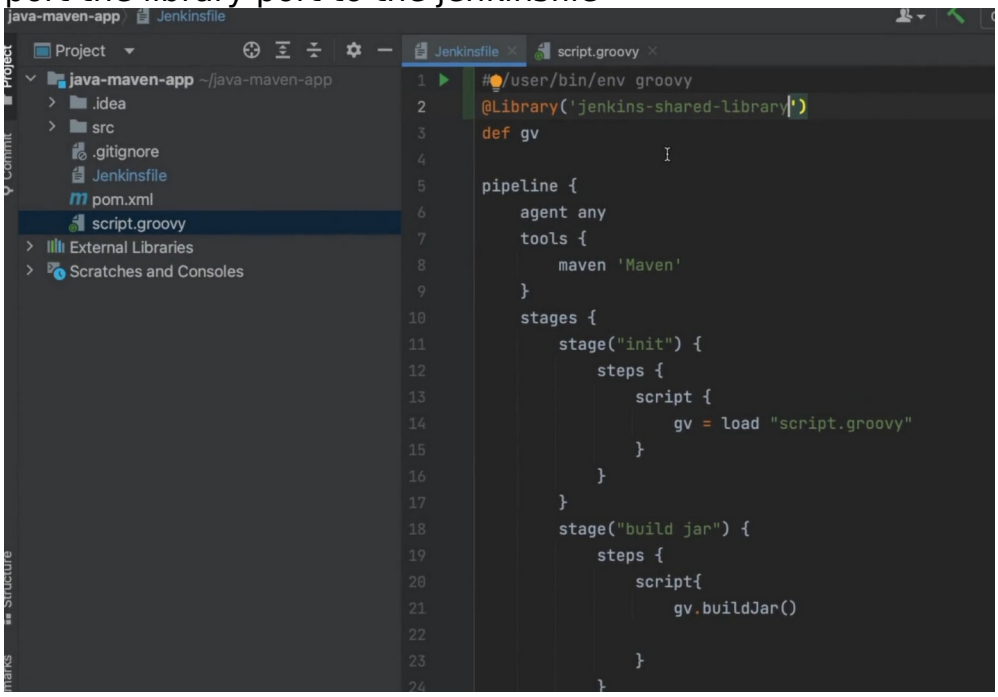
Choose git -> jenkins shared library repo URL -> add credentials for GitHub -> save .

This library will be available for jenkins files in all pipelines .

4. Use shared library in jenkins file : go back to java-maven app -> create new branch from master naming it as(Jenkins shared library) ->

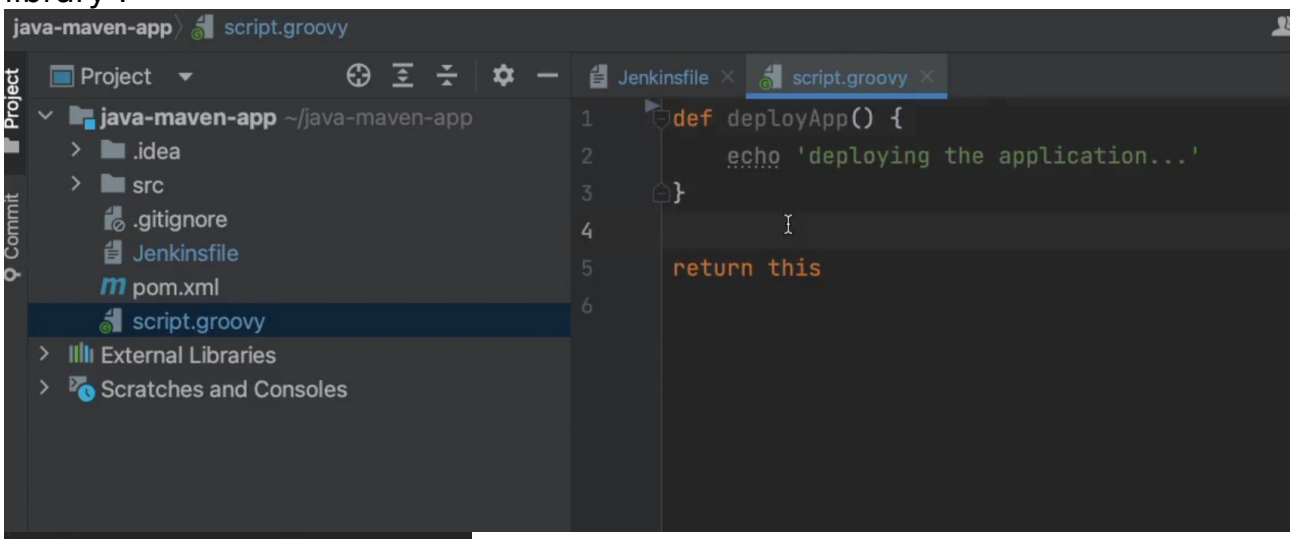


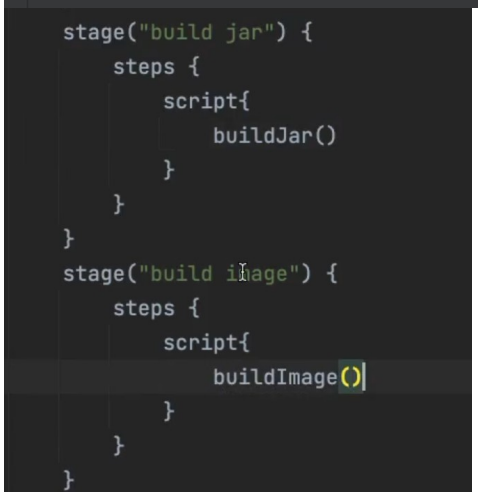
Set the same name in the start of the jenkinsfile of global pipeline lib ->this will port the library port to the jenkinsfile

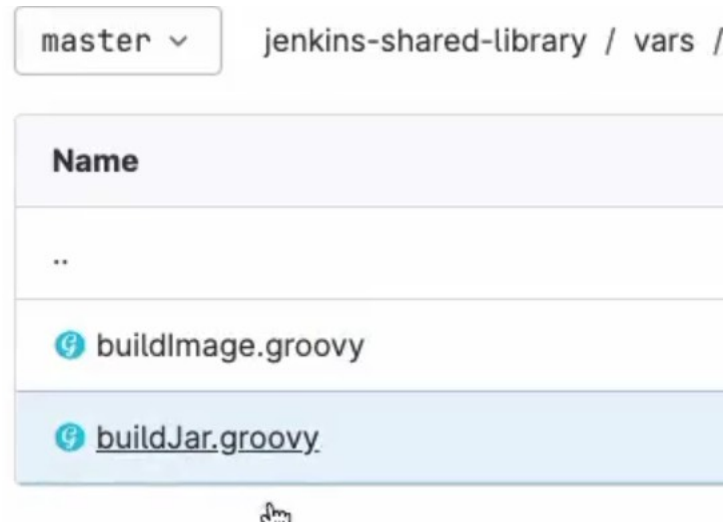


If u don’t have a **def gv** then u need to add _ at last eg: @Library()_

Remove the script from script.groovy as we are defining them in shared library :

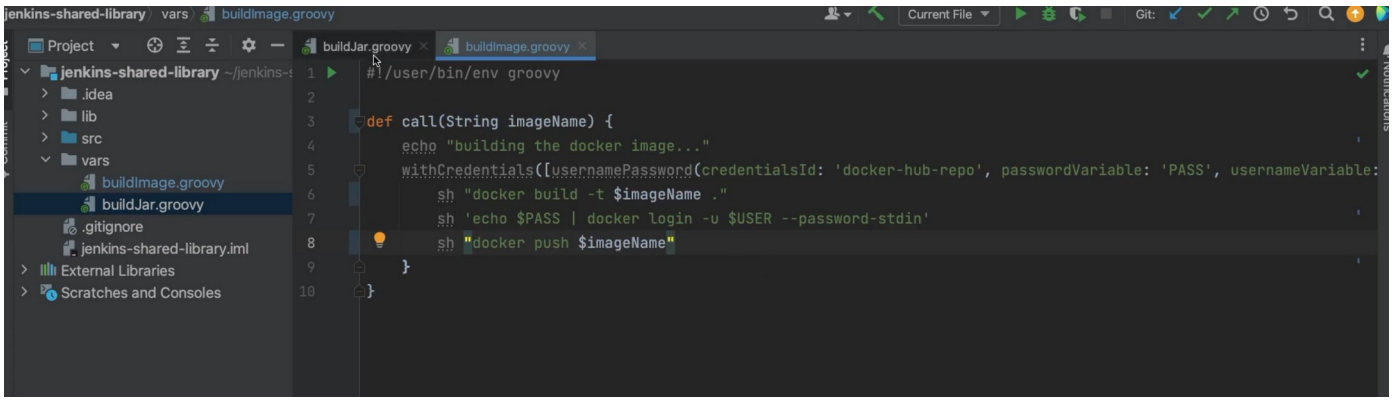




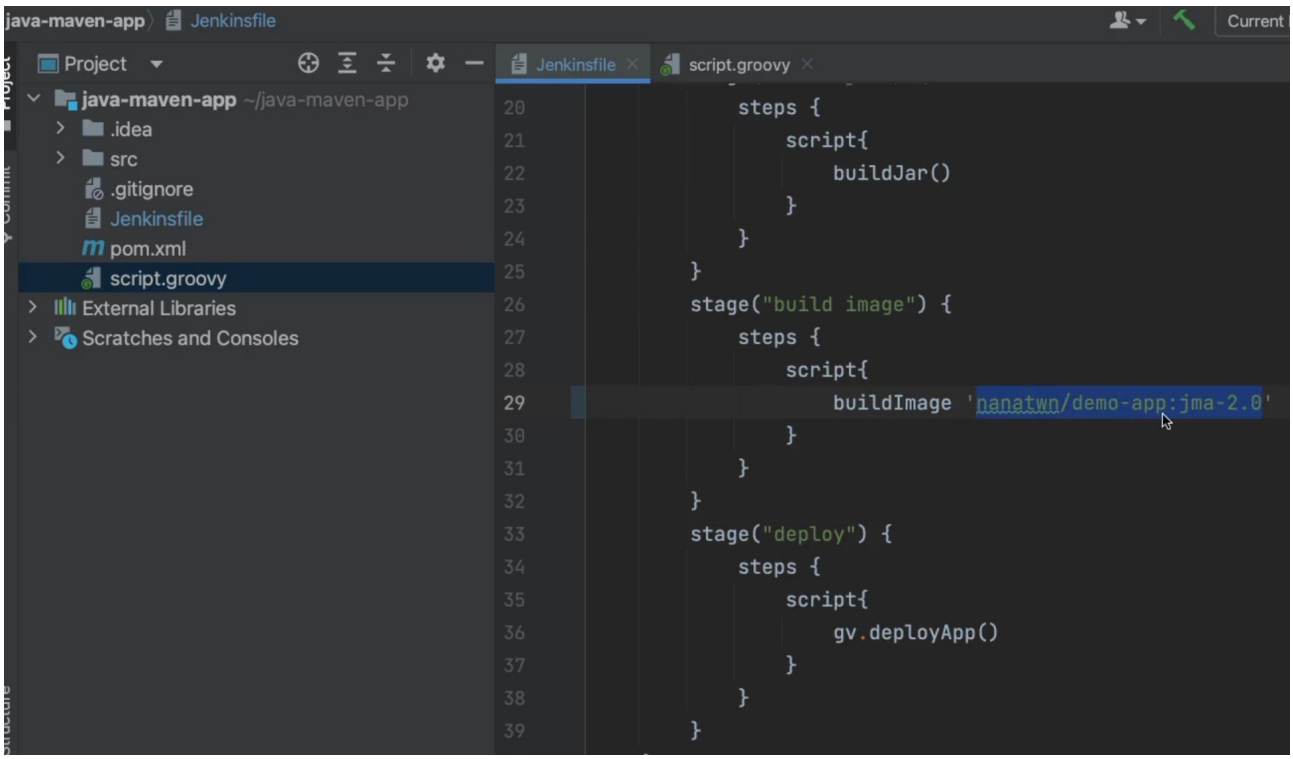


These new function in jenkinsfile are refered from the shared library we defined .

5. Commit these change in remote repo in java-maven-app (jenksin-shared-lib branch)



We have hard coded the image name here reponame= nanatwn/demo-app : (imageName and tag)jma-2.0 this is not good it can happen each build can produce a new tag so this whole thing should actually be a parameter
Here we have parameterise the build image function so that we can pass imageName ,rep and the tag from Jenkins file .



Here we have passed the parameters .

Dont forget to commit these change in jenkinsfile of jenkinssharedlib branch . Also shared_library changes need to be committed

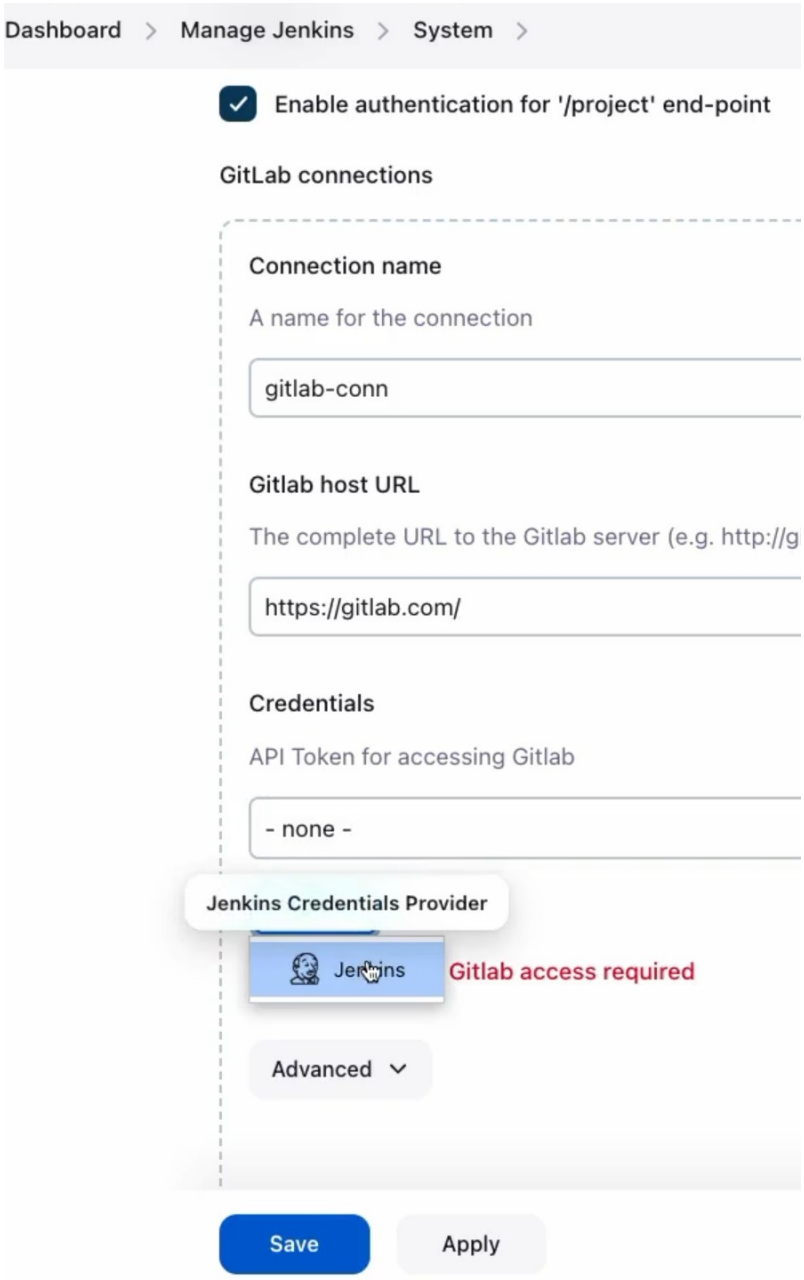
HOW TO TRIGGER JENKINS BUILD JOBS :

1.Manually :use cases if u have a separate pipeline that deploy to production
2. U want pipeline job to be triggered automatically when changes happens in git repo.
We want GitHub to notify jenkins that repo code is change u can build now. .
3. By scheduling them : trigger job/pipeline at scheduled time . Use case: for long running tests and test them self take more times . So u schedule them eg: in night when no one is working .

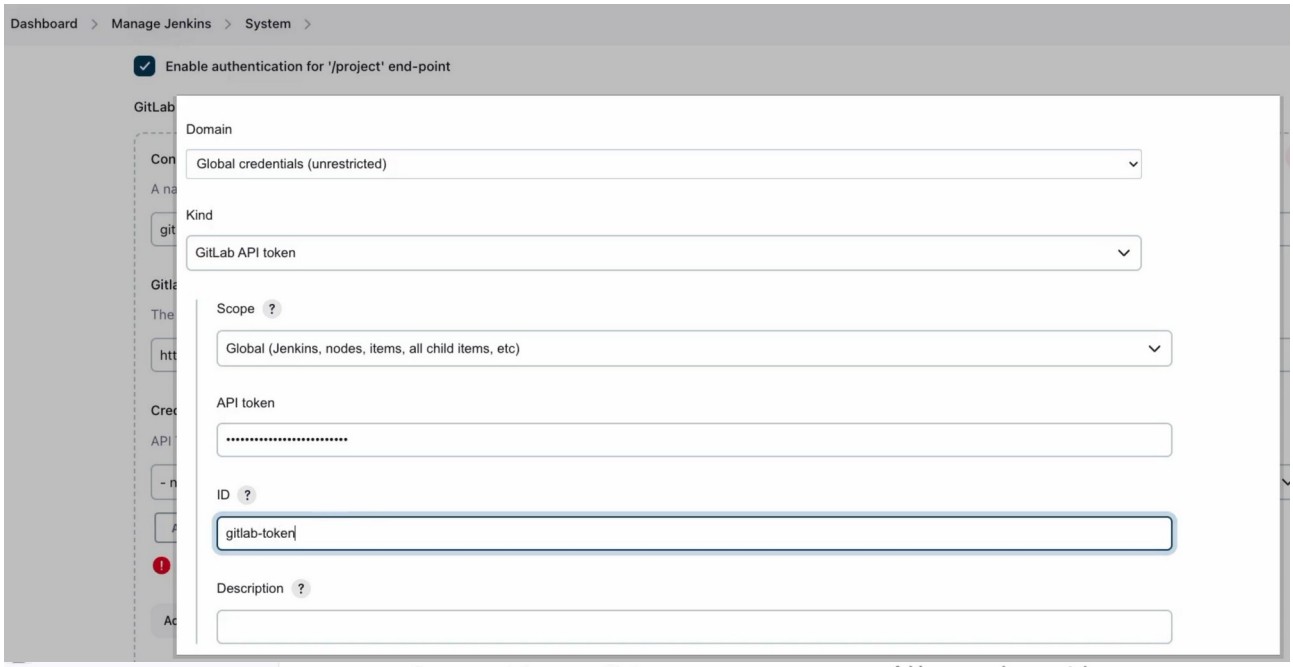
Automatic triggering :

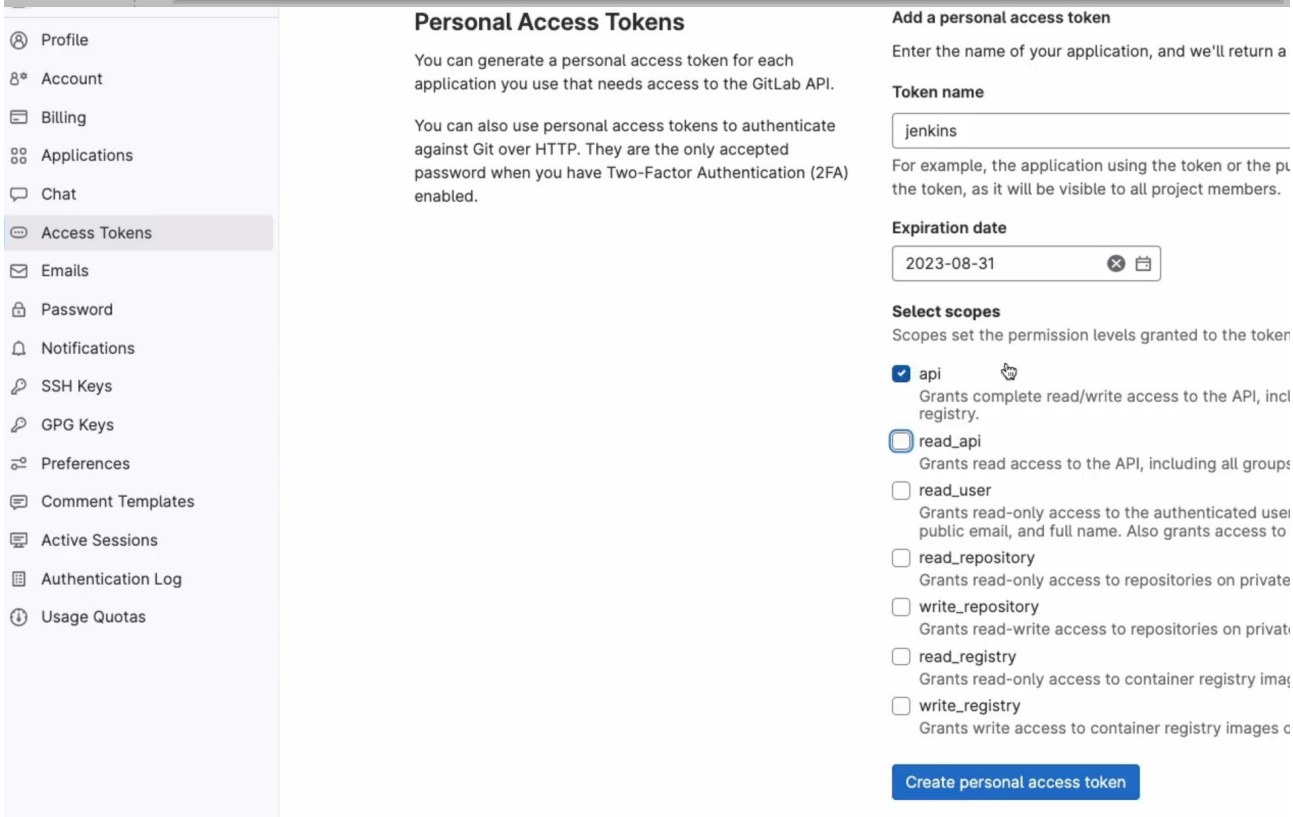
For that we need jenkins plugins :

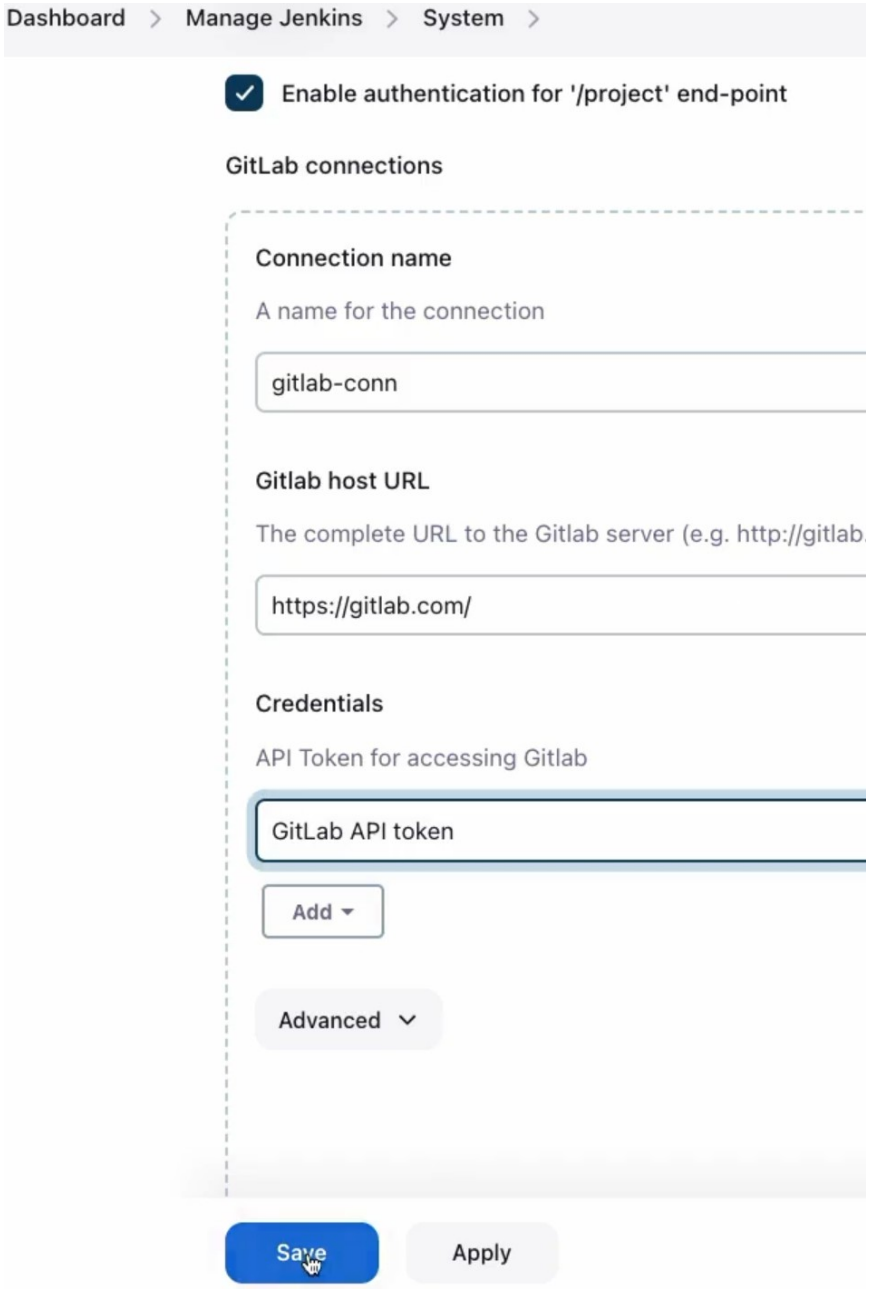
1. Do configuration in jenkins and next in our GitHub java-maven-app
2. Manage jenkins-> plugins ->install gitlab plugin(for build triggers)->mange Jenkins -> system-> u can see git Lab configuration ->

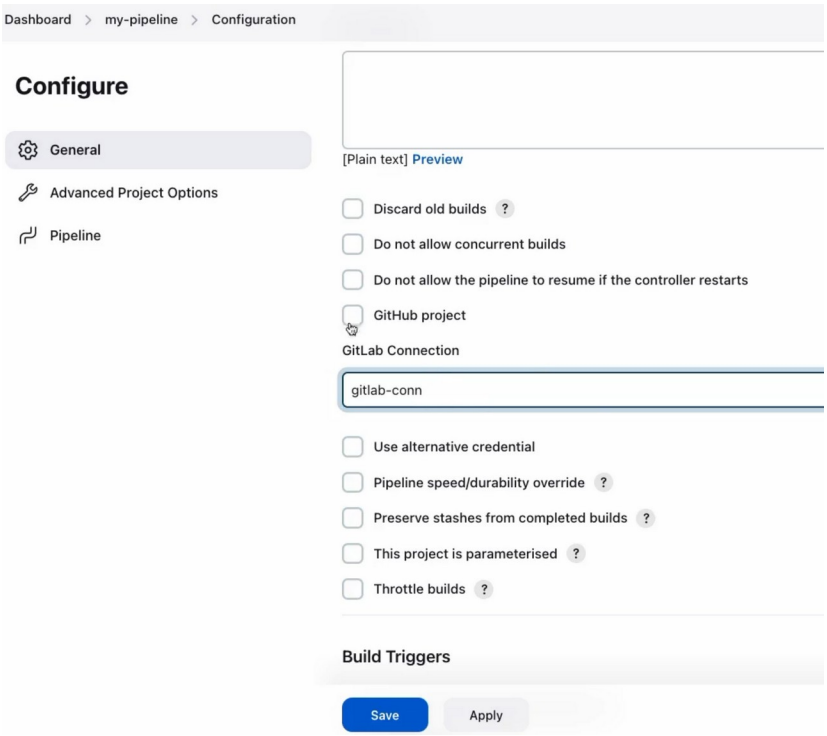

Dashboard > Manage Jenkins > System >
Enable authentication for '/project' end-point
GitLab connections
Connection name
A name for the connection
gitlab-conn
Gitlab host URL
The complete URL to the Gitlab server (e.g. http://g
https://gitlab.com/
Credentials
API Token for accessing Gitlab
- none -
Jenkins Credentials Provider
Jenkins
Gitlab access required
Advanced
Save
Apply

When ever a application is talking to our gitlab account we can let that application to authenticate via a access token

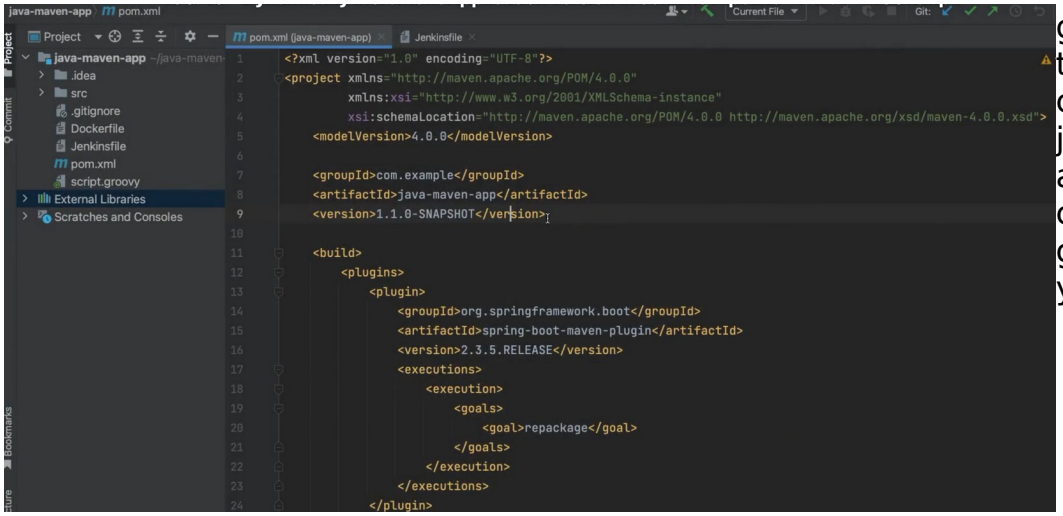

Dashboard
Manage Jenkins
System
Enable authentication for '/project' end-point
GitLab
Domain
Global credentials (unrestricted)
Kind
GitLab API token
Scope
Global (Jenkins, nodes, items, all child items, etc)
API token
ID
gitlab-token
Description

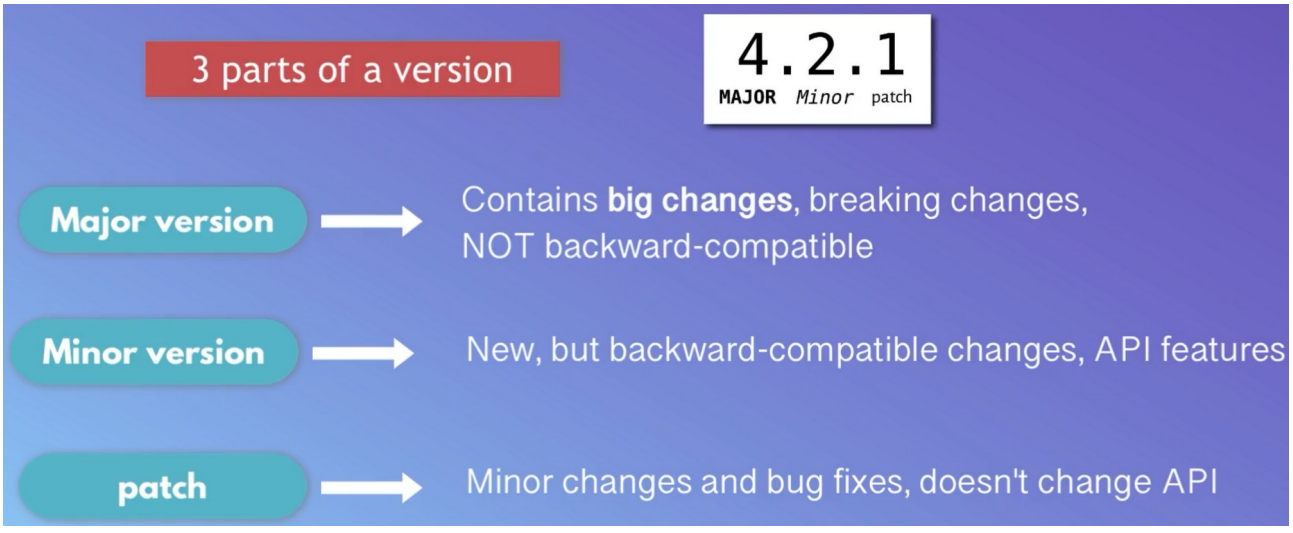

Profile
Account
Billing
Applications
Chat
Access Tokens
Emails
Password
Notifications
SSH Keys
GPG Keys
Preferences
Comment Templates
Active Sessions
Authentication Log
Usage Quotas
Personal Access Tokens
You can generate a personal access token for each application you use that needs access to the GitLab API.
You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.
Add a personal access token
Enter the name of your application, and we'll return a
Token name
jenkins
For example, the application using the token or the pu
the token, as it will be visible to all project members.
Expiration date
2023-08-31
Select scopes
Scopes set the permission levels granted to the token
api
Grants complete read/write access to the API, incl
registry.
read_api
Grants read access to the API, including all groups
read_user
Grants read-only access to the authenticated user
public email, and full name. Also grants access to
read_repository
Grants read-only access to repositories on private
write_repository
Grants read-write access to repositories on privat
read_registry
Grants read-only access to container registry imag
write_registry
Grants write access to container registry images c
Create personal access token


Dashboard
Manage Jenkins
System
Enable authentication for '/project' end-point
GitLab connections
Connection name
A name for the connection
gitlab-conn
Gitlab host URL
The complete URL to the Gitlab server (e.g. http://gitlab.
https://gitlab.com/
Credentials
API Token for accessing Gitlab
GitLab API token
Add
Advanced
Save
Apply

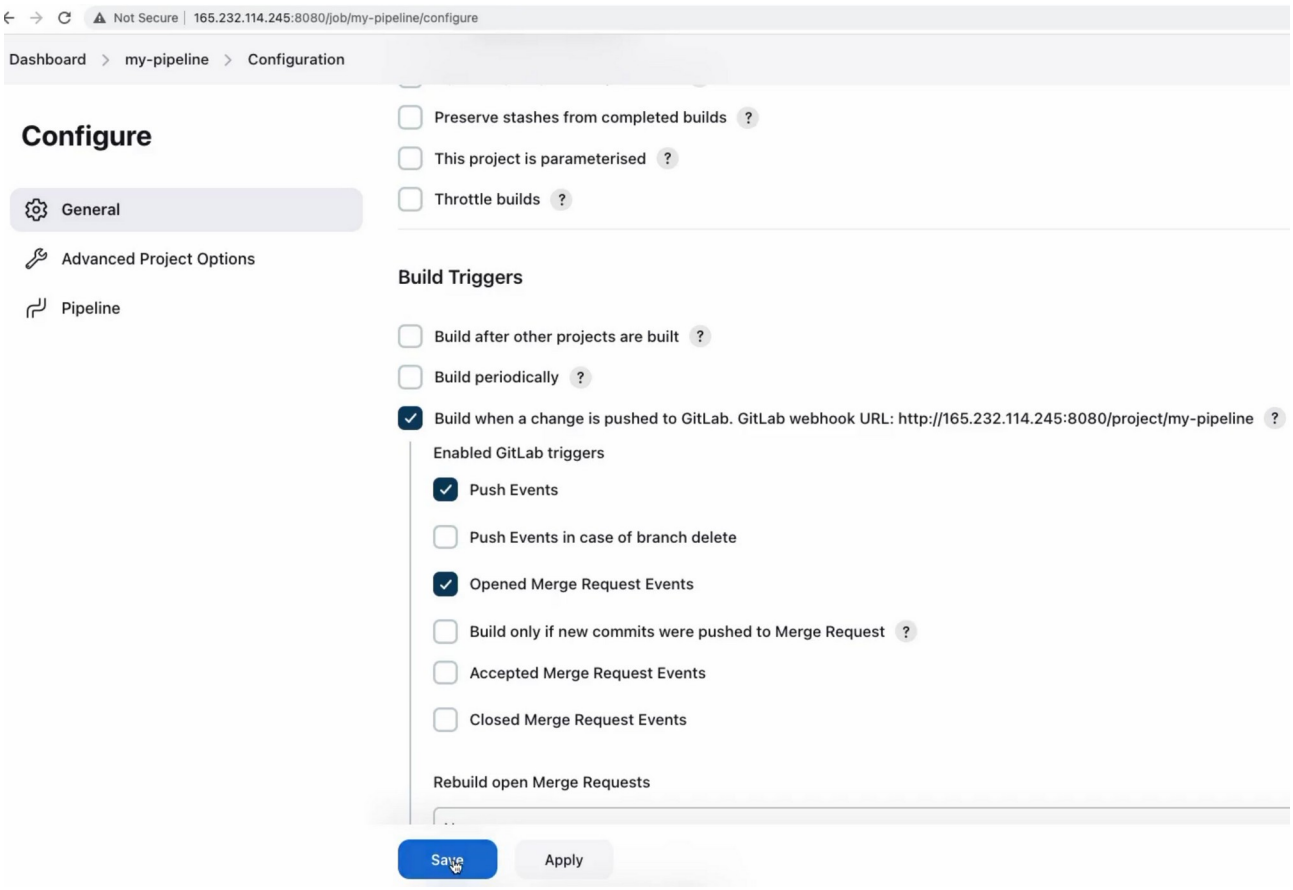

Dashboard > my-pipeline > Configuration
Configure
General
Advanced Project Options
Pipeline
[Plain text] Preview
Discard old builds
Do not allow concurrent builds
Do not allow the pipeline to resume if the controller restarts
GitHub project
GitLab Connection
gitlab-conn
Use alternative credential
Pipeline speed/durability override
Preserve stashes from completed builds
This project is parameterised
Throttle builds
Build Triggers
Save
Apply

If u have multiple gitlab lab accounts that u all want to connect with jenkins u will have a list of them and u can choose which git lab connection your job should use



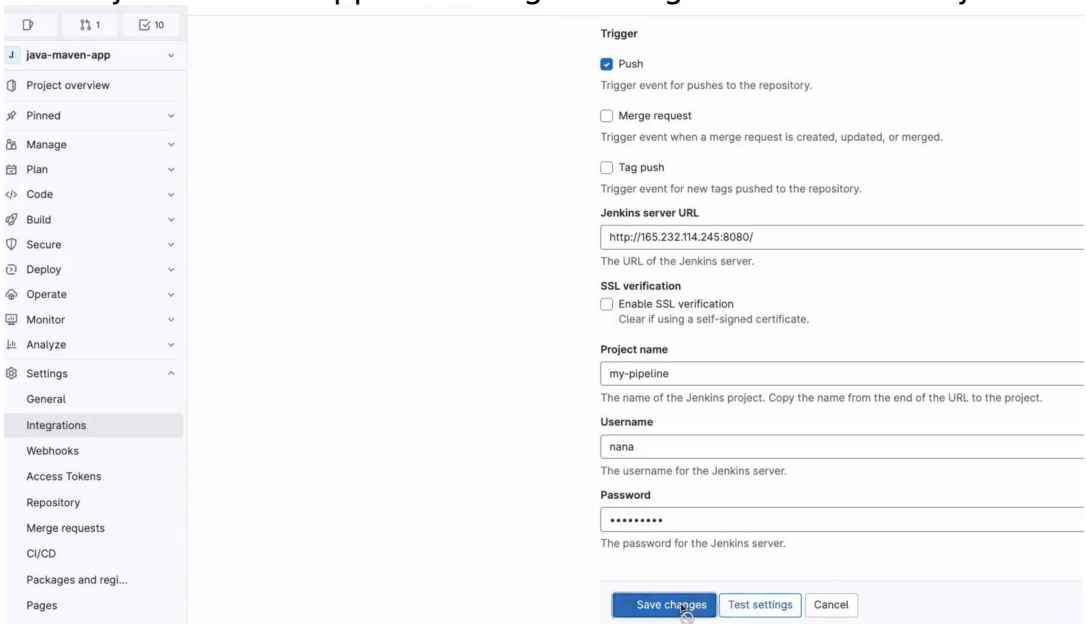


.



When ever push events for merge request event happens in gitlab it will send a web hook url to my Jenkins : and trigger the pipeline to build .

Second part is to configure git lab to automatically send notification to jenkins when ever a commit or a push happens
Got to java-maven-app -> setting -> integration -> choose jenkins ->



There use the jenkins remote server url and the project name should be same as the job name and also username and password should be same as Jenkins user name and password

If u want to test without pushing any thing u can use test settings option : u can see on jenkins a build was triggered .

Configuring automatic trigger of jenkins jobs for multi branch pipeline :

We need a another plugin to trigger multi branch pipeline : for that we need another way :
Go to manage jenkins-> plugins -> available plugins -> multi branch scan webhook trigger -> go to dashboard -> inside multi branch job -> configuration -> build config-> scan multi branch pipeline trigger -> scan by web hook (this will let us to trigger build every time we push changes to repo)

This token will be used for the communication in git lab and jenkins or jenkins with any other git repo = name trigger token :any name eg : gitlabtoken .

Go to gitlab ->setting -> web hooks-> url paste :

Search page

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

http://165.232.114.245:8080/multibranch-webhook-trigger/invoke?token=gitlabtoken

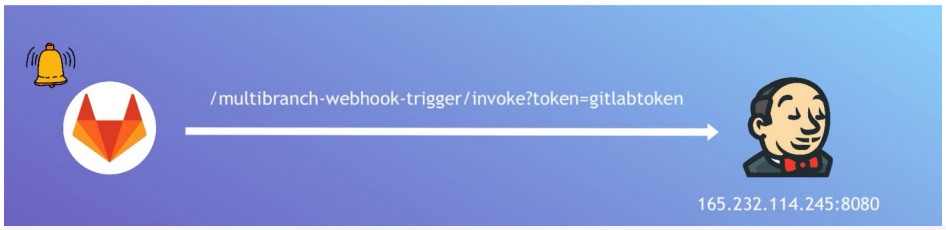
URL must be percent-encoded if it contains one or more special characters.

- (•) Show full URL
- ( ) Mask portions of URL
  Do not show sensitive data such as tokens in the UI.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger



Dashboard > java-maven-app > Configuration

Configuration

- General
- Branch Sources
- Build Configuration
- Scan Multibranch Pipeline Triggers
- Orphaned Item Strategy
- Appearance
- Health metrics
- Properties

Script Path ?

Jenkinsfile

Scan Multibranch Pipeline Triggers

- [ ] Periodically if not otherwise run ?
- [x] Scan by webhook ?

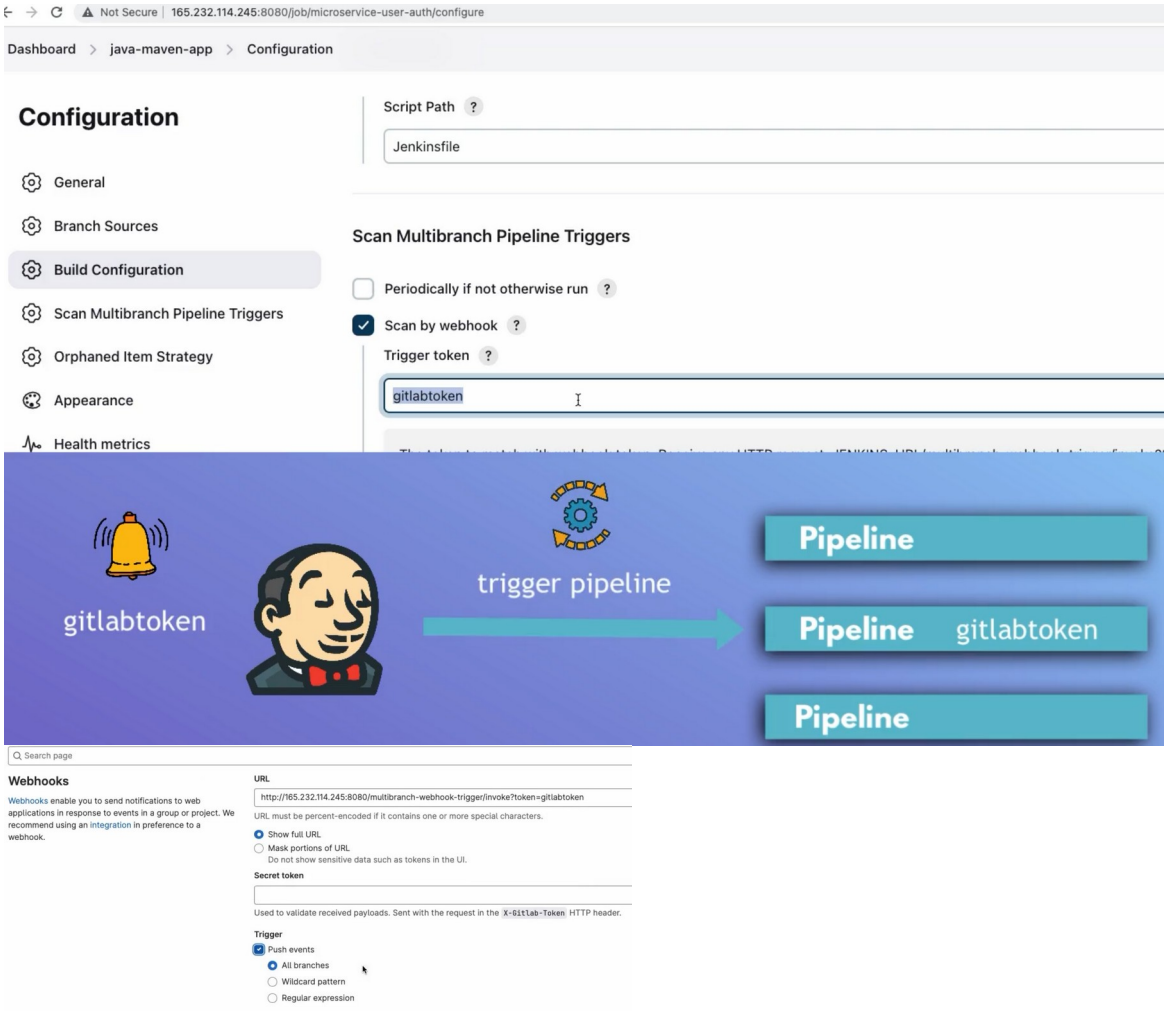
Trigger token ?

gitlabtoken

The token to match with webhook token. Receive any HTTP request, JENKINS_URL/multibranch-webhook-trigger/invoke?token=[Trigger token]
If a token match, than a multibranch scan will bi triggered.

(from Multibranch Scan Webhook Trigger)

Orphaned Item Strategy



VERSIONING YOUR APPLICATION :

Your app has a version and each build tool and package manager tool will maintain that version in its own main build file .

For gradle = build.gradle file , npm = package.json file , maven = pom.xml file .

So every package manager tool will have its own file where dependencies are listed plugins etc . And where the version of the application is also defined

```
#!/usr/bin/env groovy

pipeline {
    agent any
    tools {
        maven 'Maven'
    }
    stages {
        stage('increment version') {
            steps {
                script {
                    echo 'incrementing app version...'
                    sh 'mvn build-helper:parse-version versions:set \
                        -DnewVersion=\\\${parsedVersion.majorVersion}.\\\${parsedVersion.minorVersion}.\\\${parsedVersion.nextIncrementalVersion} \
                        versions:commit'
                    def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'
                    def version = matcher[0][1]
                    env.IMAGE_NAME = "$version-$BUILD_NUMBER"
                }
            }
        }
        stage('build app') {
            steps {
                script {
                    echo "building the application..."
                    sh 'mvn clean package'
                }
            }
        }
        stage('build image') {
            steps {
                script {
                    echo "building the docker image..."
                    withCredentials([usernamePassword(credentialsId: 'docker-hub-repo', passwordVariable: 'PASS', usernameVariable: 'USER')]) {
                        sh "docker build -t nanajanashia/demo-app:${IMAGE_NAME} ."
                        sh "echo $PASS | docker login -u $USER --password-stdin"
                        sh "docker push nanajanashia/demo-app:${IMAGE_NAME}"
                    }
```

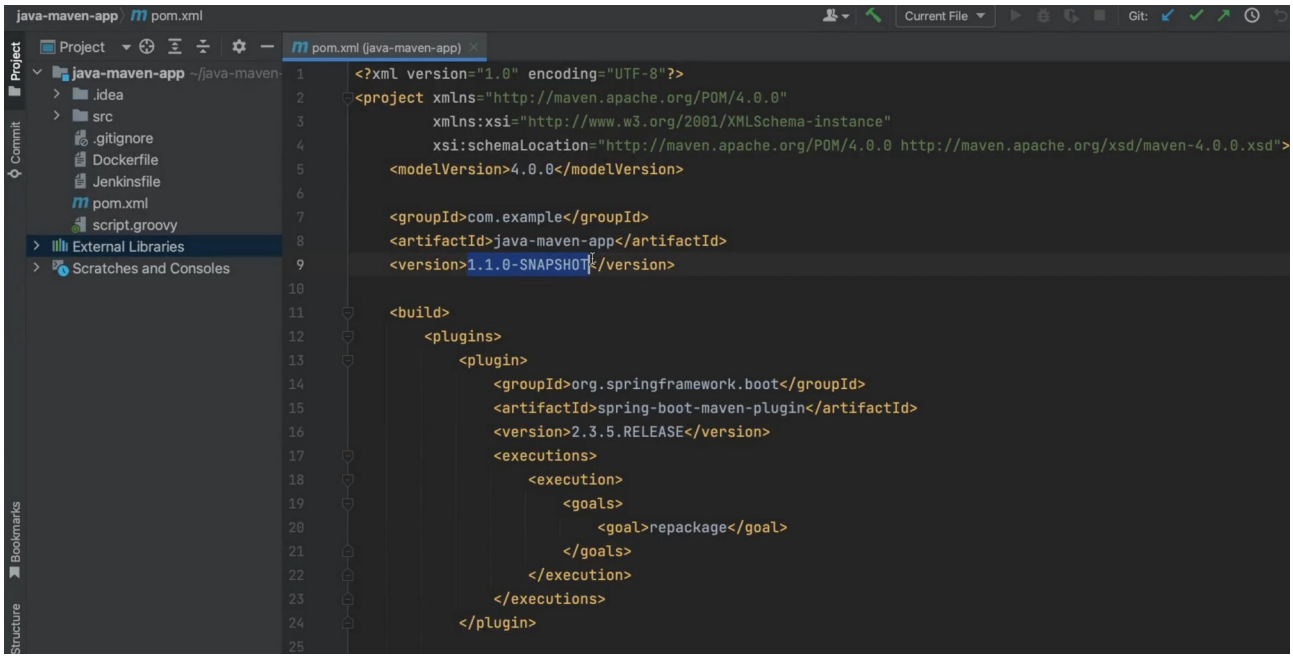
<version>1.1.0-SNAPSHOT</version>

This is the version that you or your dev team has decide on to how to version your application .
This versioning is relevant when we are releasing the new version of our application .
The user that is using that application or webpage need to get latest version free from those bugs and to see latest feature added in new version .

The most common way to increase version is ->

-SNAPSHOT : is called a suffix that u can add to your version for more information snapshot is usually used for development version eg ; if u want to test but not release that versions

How to u change the version :
Manually ?
Do it automatically !!
Specially when u are building cicd jenkins pipeline automation for your application .
So when u commit change to jenkins jenkins build pipeline should increment the version and release new application .
Each build tool eg : npm , maven , gradle , etc they all have plugins or commands used for increment the version .



In cd java-maven-app terminal : we will write a command which will automatically upgrade the version defined here

: command is : ==> mvn build-helper:parse-version versions:set \ next line commands
: -DnewVersion=\${parsedVersion.majorVersion}.\${parsedVersion.minorVersion}.\${parsedVersion.nextIncrementalVersion} versions:commit

//versions:commit (will remove the old copy of the pom.xml file and replace it with new copy with new version .

// it have majorVersion and nextMajorVersion , nextMinorVersion and minorVersion , incrementalVersion and nextIncrementalVersion.

So build-helper is a maven plugin that has a command called parse-version

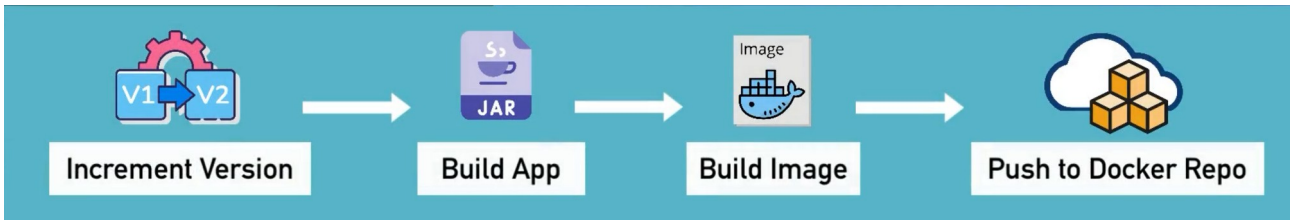
This parse-version goes and finds the pom file finds the version tag and parsers the version inside into 3 parts ->major , minor , increment .
Versions is a plugin : set is a command this will set the version btw these <version > tags.
Below we have incremented the incremental patch version .

```
Processing change of com.example:java-maven-app:1.1.0-SNAPSHOT -> 1.1.1
Processing com.example:java-maven-app
    Updating project com.example:java-maven-app
        from version 1.1.0-SNAPSHOT to 1.1.1
```

We have Same concept for different package manager /build tools :
Eg: read version from package.json ,increment and save back and Same for gradle also .

Now if it is a part of CICD we are introduction a new version in every new commit . ?

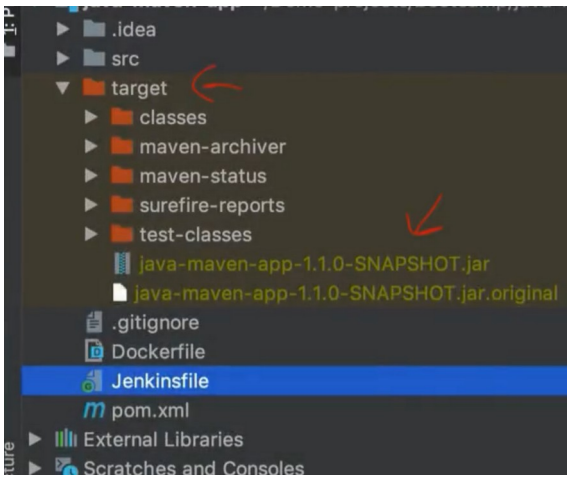
We have executed these commands locally how ever in real life u are not going to do it locally in your computer this should be part of CICD pipeline so these commands should actually be executed by jenkins when build process is running so as a next step we are going to integrate this step of setting a new version pom.xml file into jenkins file as part of build pipeline .



Increment application version on jenkins : java-maven-app:

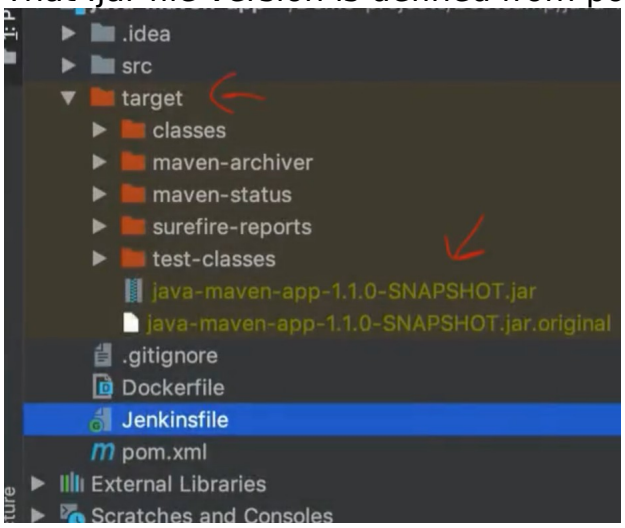
In jenkinsfile : build stage -> sh ‘mvn package’ = build a jar file with version from pom.xml

And this is where automatically generated build jar file is located in target folder .



And this is a jar file that is build using maven package
Ie: after running command : mvn package = u will see a target folder appeared .

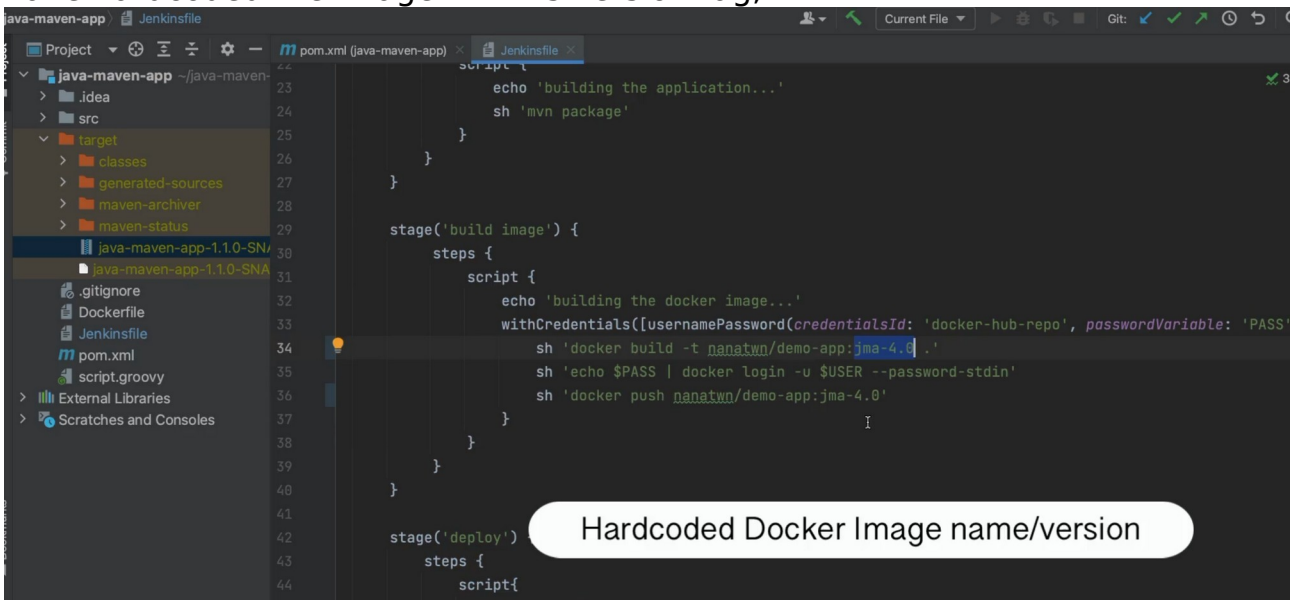
That .jar file version is defined from pom.xml version



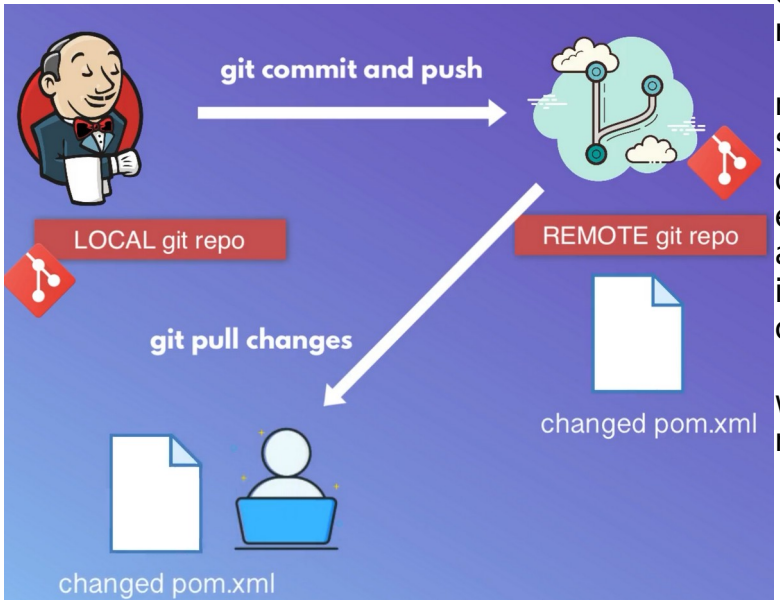
That means we need to do version increment before application build so that we get a jar file with correct incremented version .

So before the build app stage we are going to create a new stage ‘increment version’

Note: every time we are releasing a new version we are not not releasing a jar file but rather docker image every time a application new version is released that means we need a version for our docker image but here we can see we have hardcoded the image with its version tag;



So we need to increment this docker image version as well . Good practice is to use a application version for docker image version and we need to set it dynamically every time we run a build pipeline



We need to pass this $IMAGE_NAME variable in docker build command every time we build a new application version add “ “ as its a variable not single cots .

So we need to get the value of major , minor or

increment version of app to map it with imageName version so how do we access these value and save it in a variable solution is by reading it from pom.xml and u need to tell it inside pom.xml I am looking for something like this -> **<version>1.1.0-SNAPSHOT</version>** and since we don't know the version yet add a regular expression in between version tag and save that value in a variable named it matcher .

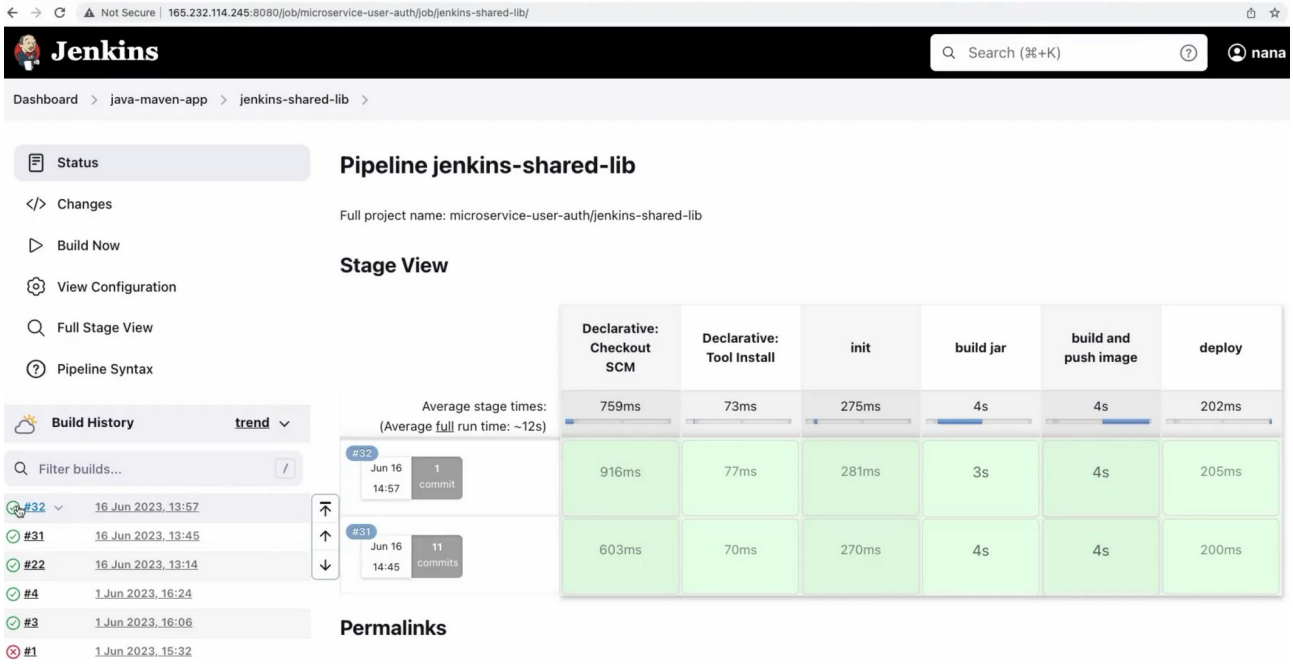
Matcher is trying to match every singe element it can find in pom.xml and save it into an array and since we have only one version tag in pom.xml it will have only one element .
Ie: index 0 will be the version itself and index 1 will be the text inside version.

:def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'

We can assign to version variable : def version = matcher[0][1]

Now we can use this variable version as image name

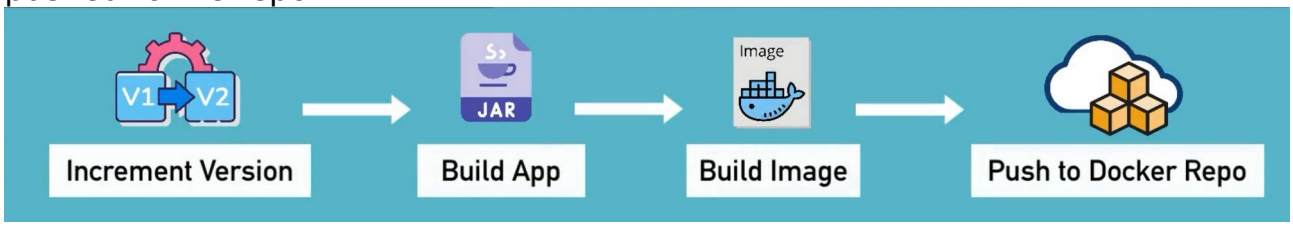
We want to append a build not to this version -> "$version-$BUILD_NUMBER" , here build no is a env variable that jenkins makes available for us in pipeline job.



These nos on bottom left 31,32.. are the build no. so very build has a no. and is unique for that specific build .

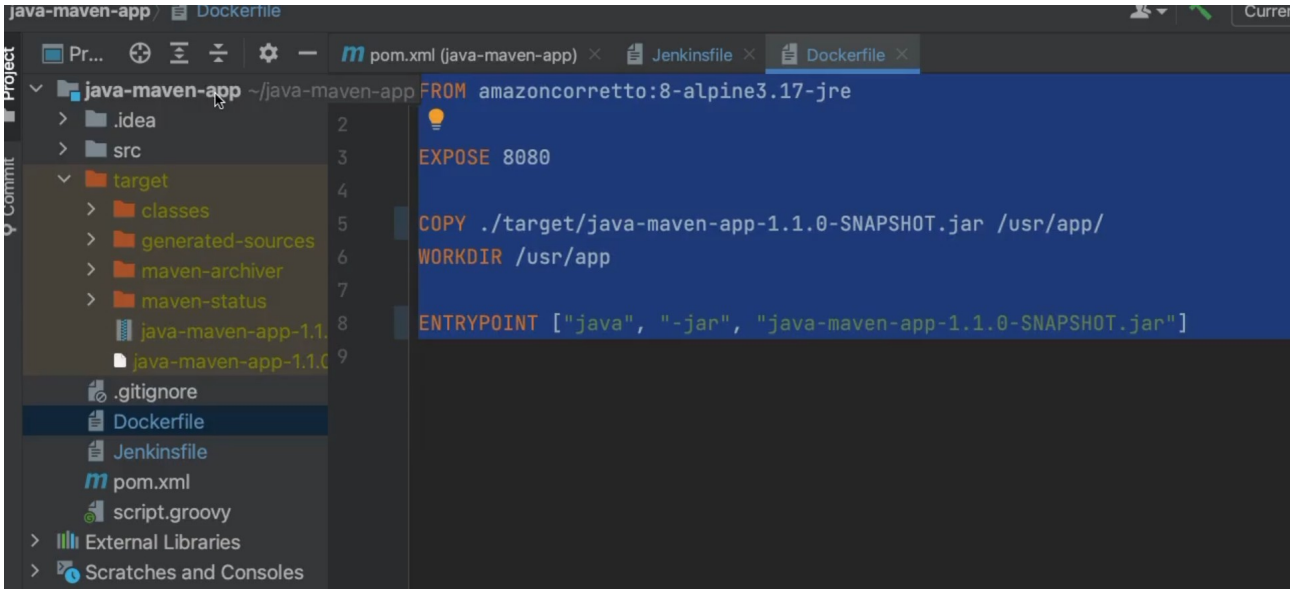
env.IMAGE_NAME = "$version-$BUILD_NUMBER" and we are appending that as a suffix to our version . And making this as a IMAGE_NAME ;

So now every time a new pipeline is build after a commit a version will be incremented and new docker image with new image tag created and then pushed to the repo .



REPLACE NEW VERSION IN DOCKER FILE :

Docker built command take a docker file and builds a new image so if we look at our docker file .

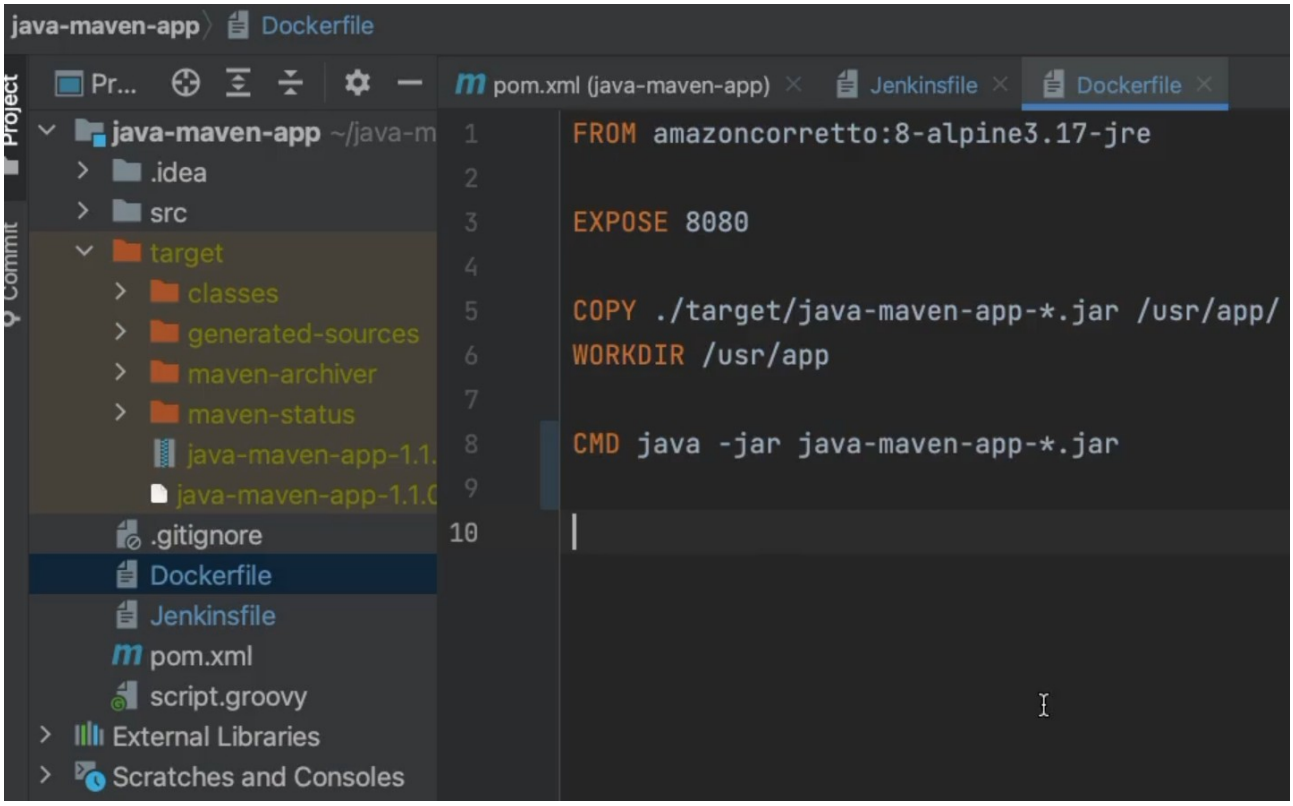


These are the command that will be executed to build docker image .
And right now u can see we have hard coded the jar file name . So right now it is matching to this version 1.1.0-SNAPSHOT how ever if we run the pipeline and it increment the version to 1.1.1 these command will not work any more because there is no jar file with version 1.1.1 here in docker file so we need to make this dynamic not hard coded so make it just take a jar file form target folder java-maven-app* so this is a regular expression replace meant of any version after java-maven-app .

And the write jar file will be copies in the image file system
Now inside the image that jar file need to get executed right
So when ever we run this image as a container

Java , jar and jar file will get executed .
Instead of entry point -> CMD java -jar java-maven-app-*.jar

Final docker file look =>

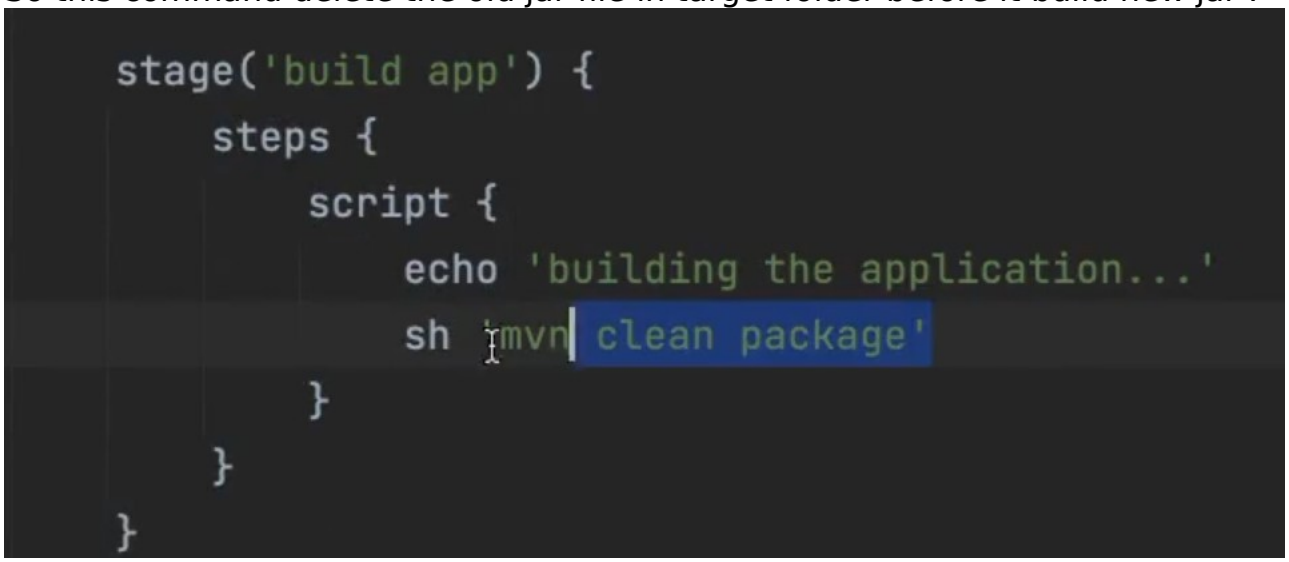


We need to make one more change in jenkins file :
When we change to version on pom.xml from 1.1.0 to 1.1.1 and run : mvn package
It will generate another jar file .

So this expression in docker file : COPY ./target/java-maven-app-*.jar /usr/app/
Will match both the jar file in target folder and we only need one in one image so java will not know which one to execute .

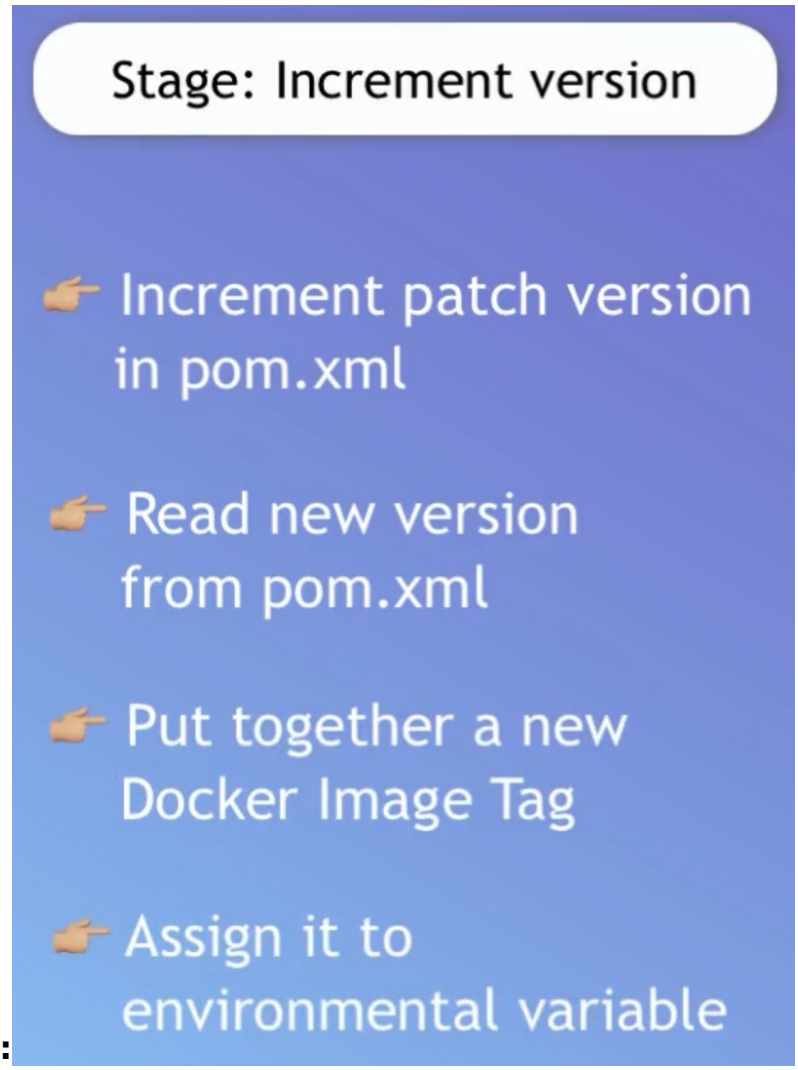
Means before we build the jar file we need to clean the target folder with old jar file so new one get generated with single jar file . : use command : mvn clean package

So this command delete the old jar file in target folder before it build new jar .



So make change in jenkinsfile .

CONCLUSION :



```
stages {
    stage('increment version') {
        steps {
            script {
                echo 'incrementing app version...'
                sh 'mvn build-helper:parse-version versions:set \
                    -DnewVersion=\\\${parsedVersion.majorVersion}.\\\${parsedVersion.minorVersion}.\\\${parsedVersion.nextIncrementalVers
                    versions:commit'
                def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'
                def version = matcher[0][1]
                env.IMAGE_NAME = "$version-$BUILD_NUMBER"
            }
        }
    }
```

THESE ABOVE COMMANDS WILL BE EXECUTED ON JENKINS SERVER .

STAGE BUILD : 1. CLEAN TARGET FOLDER , 2. PACKAGE JAR FILE .
With mvn clen package command jenkins server will clean the target folder and build the new application jar file .

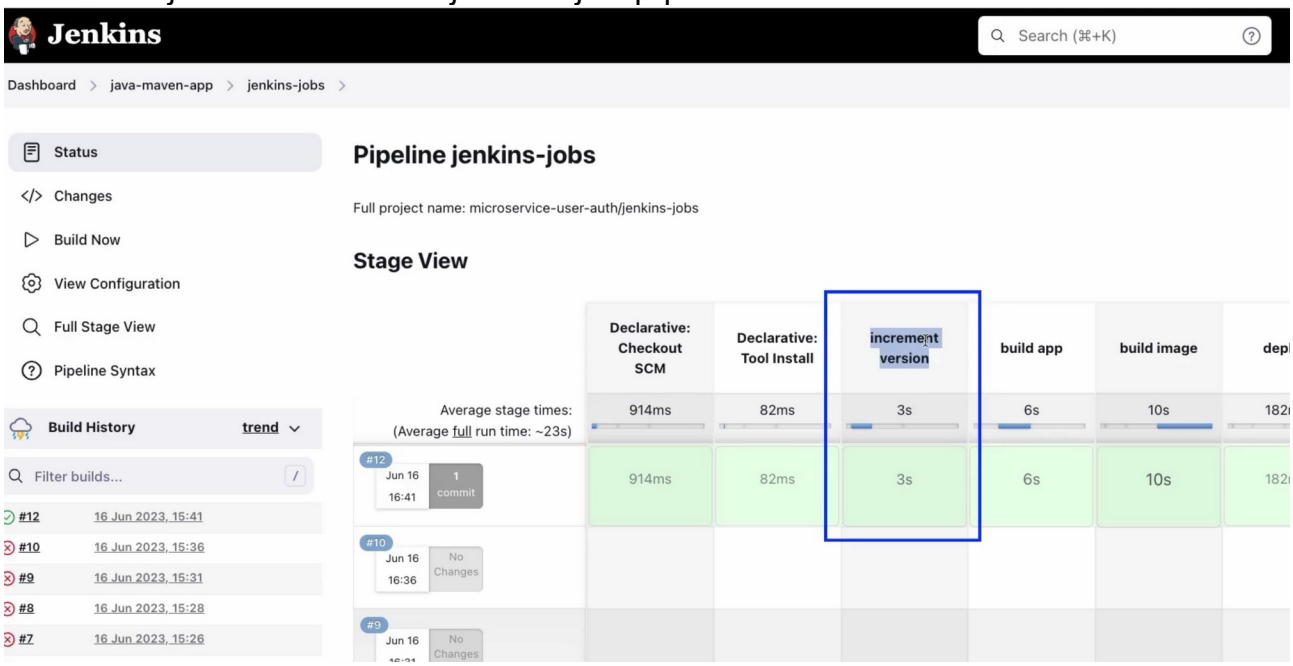
```
    stage('build app') {
        steps {
            script {
                echo "building the application..."
                sh 'mvn clean package'
            }
        }
    }
```

Then jenkins will build a docker image -> STAGE : BUILD DOCKER IMAGE : build docker image from docker file and tag image with repo URL and name .
Then jenkins will log in to docker repo and will push the new image to that repo .

```
stage('build image') {
    steps {
        script {
            echo "building the docker image..."
            withCredentials([usernamePassword(credentialsId: 'docker-hub-repo', passwordVariable: 'PASS', usernameVariable: 'USER')]) {
                sh "docker build -t nanajanashia/demo-app:${IMAGE_NAME} ."
                sh "echo $PASS | docker login -u $USER --password-stdin"
                sh "docker push nanajanashia/demo-app:${IMAGE_NAME}"
            }
        }
    }
}
```

Now commit these change to your git repo(in jenkins-job branch) : git add . ->git commit -m”” -> git push ->

Now of to jenkins and build jenkins-job pipeline ->



COMMIT VERSION BUMPS FROM THE JENKINS

But what happens now if we make some change commit them to git repo then re run this Jenkins-jobs build again . ?
We can still see the same versioning from 1.1.0 to 1.1.1 but why ?

So this change is present locally in jenkins this change never get committed to the git repo . So every time jenkins build runs . It always starts with this 1.1.0 version .

So it never ends up committing this 1.1.1 new version to the git repo there for in every build it has to start with that old version from the git repo .

How we can commit from jenkins to the git repo >?

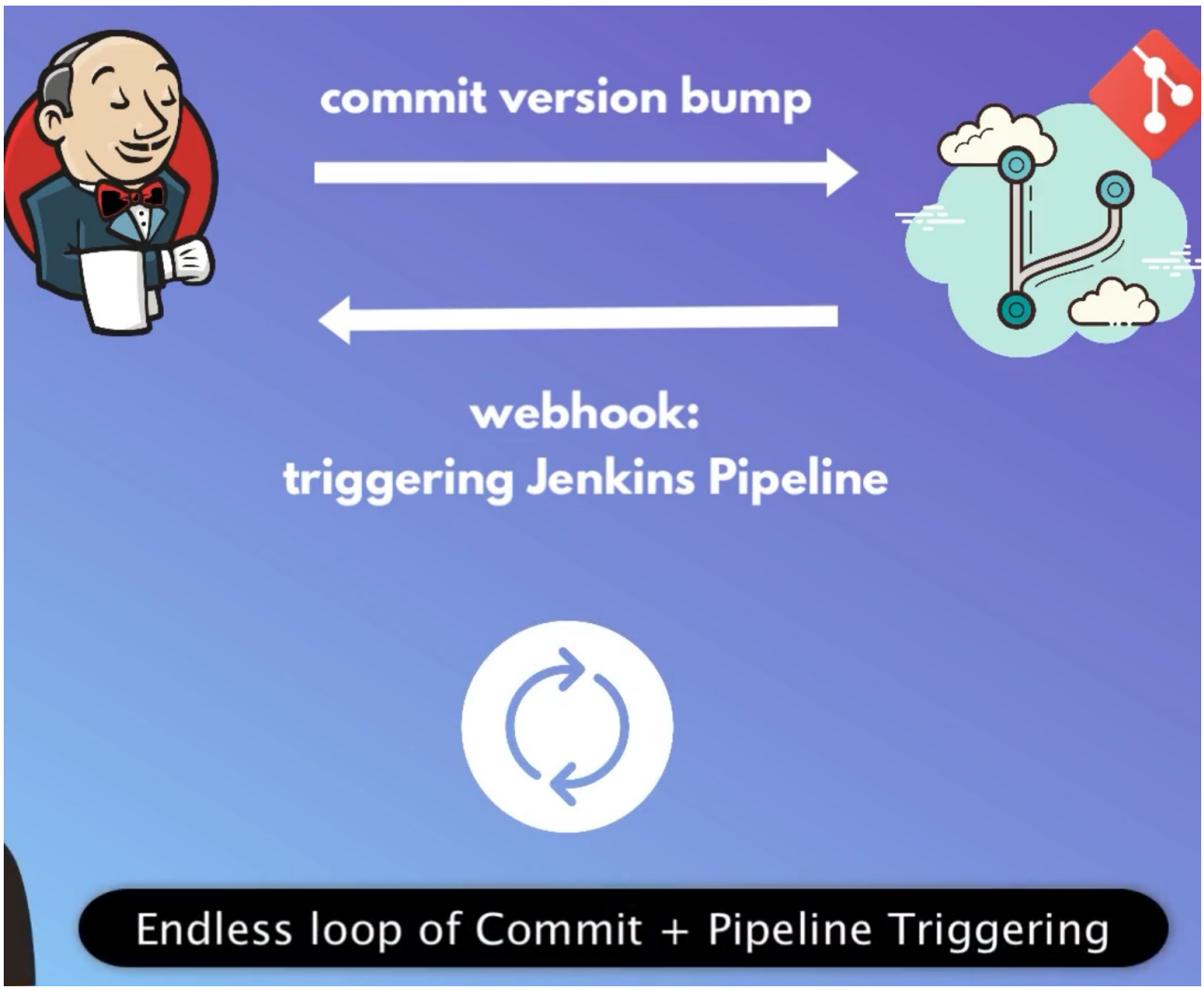
After deploy stage in jenkins file ->
Add new stage -> first we need to access git from here there fore we need access credential from git repo just like to connect to docker hub in build image stage we needed credential to connect to docker hub -> use withCredentails for that

So we need credential id from jenkins credentials -> named gitlab-credentails(in my case) -> we need to tell jenkins what this origin is going to be .

```
49          stage('commit version update') {
50              steps {
51                  script {
52                      withCredentials([usernamePassword(credentialsId: 'gitlab-credentials', passwordVariable: 'PASS', usernameVariable: 'USER')]) {
53                          // git config here for the first time run
54                          sh 'git config --global user.email "jenkins@example.com"'
55                          sh 'git config --global user.name "jenkins"'
56
57                          sh "git remote set-url origin https://${USER}:${PASS}@github.com/Vishalldwivedi/devOps-java_maven_app.git"
58                          sh 'git add .'
59                          sh 'git commit -m "ci: version bump"'
60                          sh 'git push origin HEAD:jenkins-jobs'
61                      }
62                  }
63              }
64          }
65      }
66  }
```

We need to specify ‘git push origin HEAD:jenkins-jobs’ to specify to push the commit not to a commit hash but to the Jenkins-jobs.

One more thing to take care when we are comment from jenkins to git repo =>



How to solve this problem ??
We need to find a way to detect that commit is made up by jenkins and not up to stop this we hook automatic tirgger of building pipeline . And ignore the jenkins commit to git repo . We can solve this problem by using a plugin

Go to mange jenkins -> plugins -> ignore committer strategy -> go to multi branch pipeline config -> build strategies is a option just got added through that new plugin ->
Ignore commiter strategy

```
('commit version update'){
teps {
  script {
      withCredentials([usernamePassword(credentialsId: 'gitlab-credentials', passwordVariable: 'PASS', usernameVari
          sh 'git config --global user.email "jenkins@example.com"'
          sh 'git config --global user.name "jenkins"'

          sh 'git status'
          sh 'git branch'
          sh 'git config --list'

          sh "git remote set-url origin https://${USER}:${PASS}@gitlab.com/twn-devops-bootcamp/latest/08-jenkins/ja
          sh 'git add .'
          sh 'git commit -m "ci: version bump"'
          sh 'git push origin HEAD:jenkins-jobs'}
      }
  }
```