05- JENKINS- AS PIPELINE

Status	In progress
Assign	

Jenkins is a continuous Integration and Continuous Delivery System.Jenkins ia a goto tool when implementing CICD.

Some Features

1. Simple Installation and Configuration
2. Simple web based administration
3. Flexible build jobs and pipelines
4. Customisation and flexibility using the plugin eco system.
5. Scalability through distributed builds using build agents and executors
6. Excellent Online documentation and active user base.

Jenkins Terminology

1. Master

Central Service from which the administration console is served from . ALso responsible for

1. Configuration of build jobs , pipeline and plugins via the web administration console it serves
2. Build job Scheduling
3. Build job dispatching to build agents

2.Agent

Jenkins Build agent is a separate server which receives instructions in the form of build job dispatches from the jenkins master.Each agent may have its own os . One agent may have windows os for building the .net packages.

This defines where the execution should take place.Agents are used for declarative pipelines.

3.Build Project

Defines the build configuration properties required to build a unique piece of software.
Different Project types exist for different Project.

1. pipeline
2. freestyle
3. Multi-config project
4. Fodler
5. github organization
6. MultiBranch Pipeline

4.Label

user defined metadata used to group together different build agents

5.Node

Defines by label a machine which capable of executing a build project . This defines where the execution should take place. Nodes are defined in the Scripted pipeline.

6.Artifacts

Output Files generated as an outcome of a build . An artifact for example could be a jar file or zip File

7.pipelines

Modelling concept that embodies and encapsulates the core build logic and workflow.

8.stage

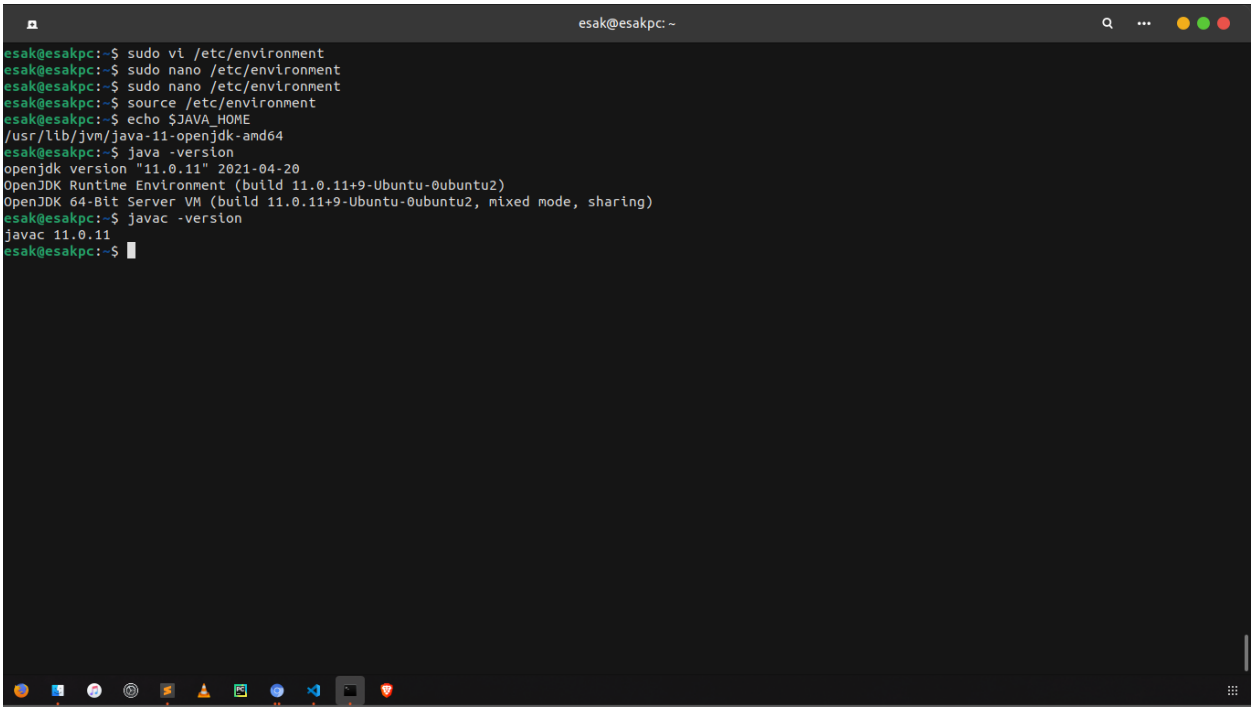
used to segment distinct sub sections of the overall pipeline

An individual stage may consist of multiple pipeline steps

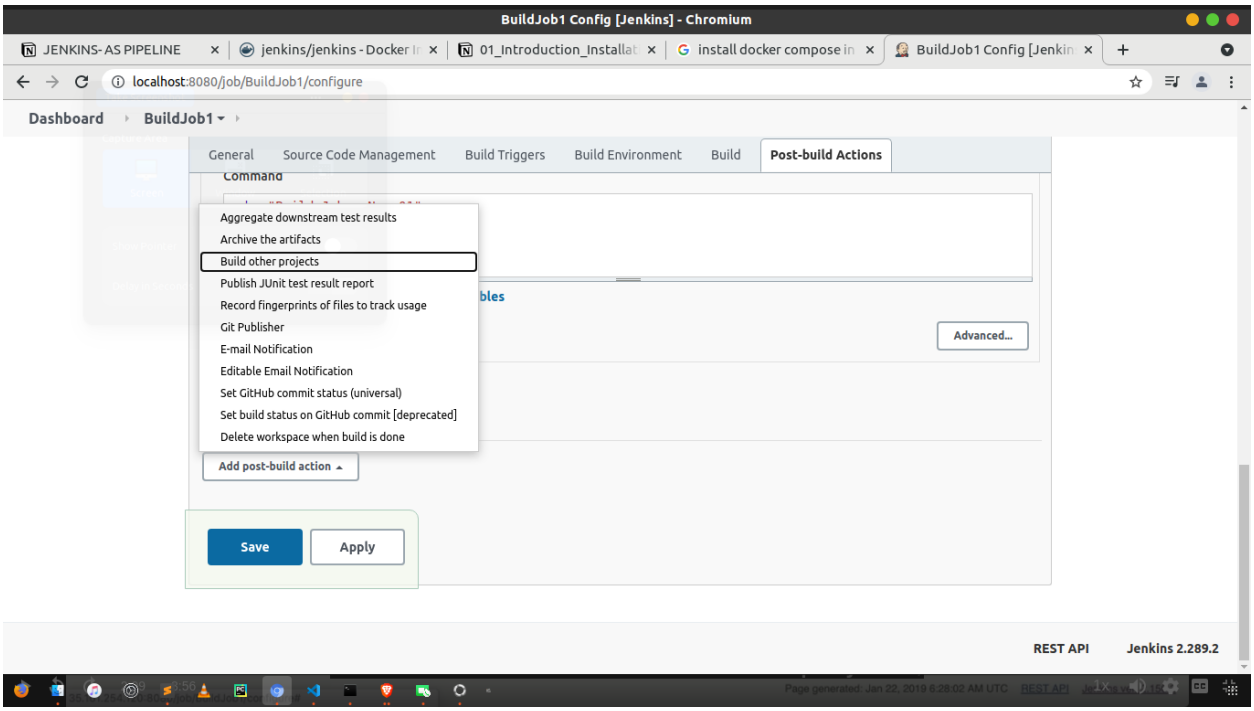
9.Workspace

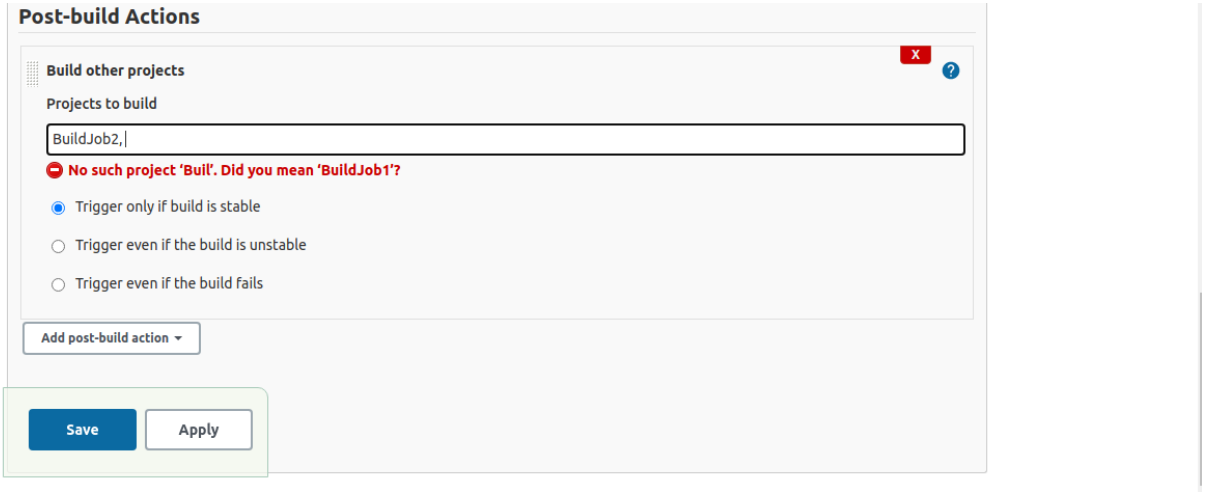
Dedicated FS directory where build work is performed for the current pipeline or build job. The workspace will be created on a particular node , for which pipeline or build jobs has been configured to run on.

Installation



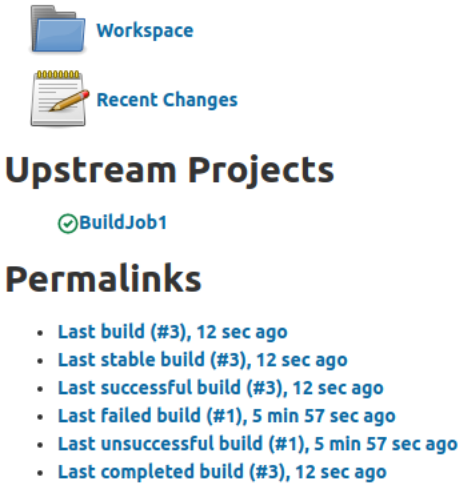
Build Chaining FreeStyle Project





We trigger the build job 2 when build job1 is successfully triggered

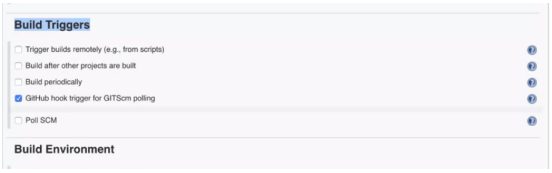
If we check the BuildJob2 it has the upstream Project

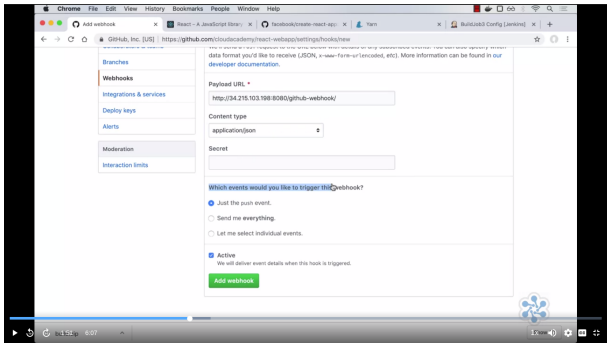


Build Triggers based on Github

we want to trigger the build when there is a change happened in the github

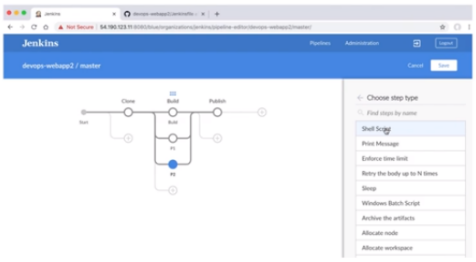
Step 2: Add github webhooks





Jenkins Pipeline

Jenkins Pipelines



Benefits

- **Speed** - Jenkins CICD pipelines by their very nature are automated
- **Productivity** - Developers maintain focus on building new features and not testing and release processes
- **Reduced Risk** - Regression testing
- **Time to Market** - Features are released quickly into production
- **Security** - Tested repeatedly to ensure that the application once released into production is battle hardened

Productivity, developers can maintain their focus

1:06 2:02 1x

Declarative Pipeline

Declarative Pipelines - Schema

```
pipeline
    agent
    --environment
    --libraries
    --options
    --parameters
    --tools
    stages
        **stage
            --agent
            --environment
            --tools
            steps
                script
            --post
    --post
```

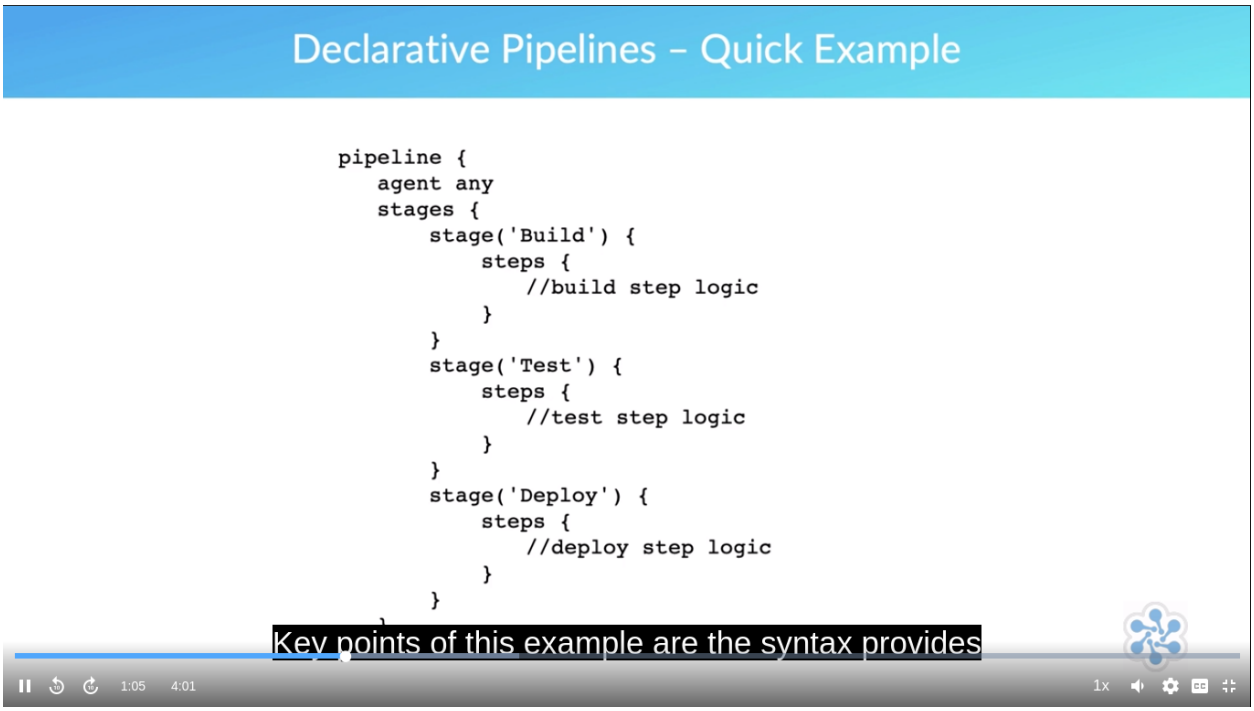
-- Optional

** Multiple

- Key points of the declarative syntax:
  - Guides the structure of the pipeline
  - Composed of several optional and mandatory sections
  - Some sections embedded in parent sections
  - Always starts with the **pipeline** keyword to indicate that it is a

a declarative pipeline is composed of several optional

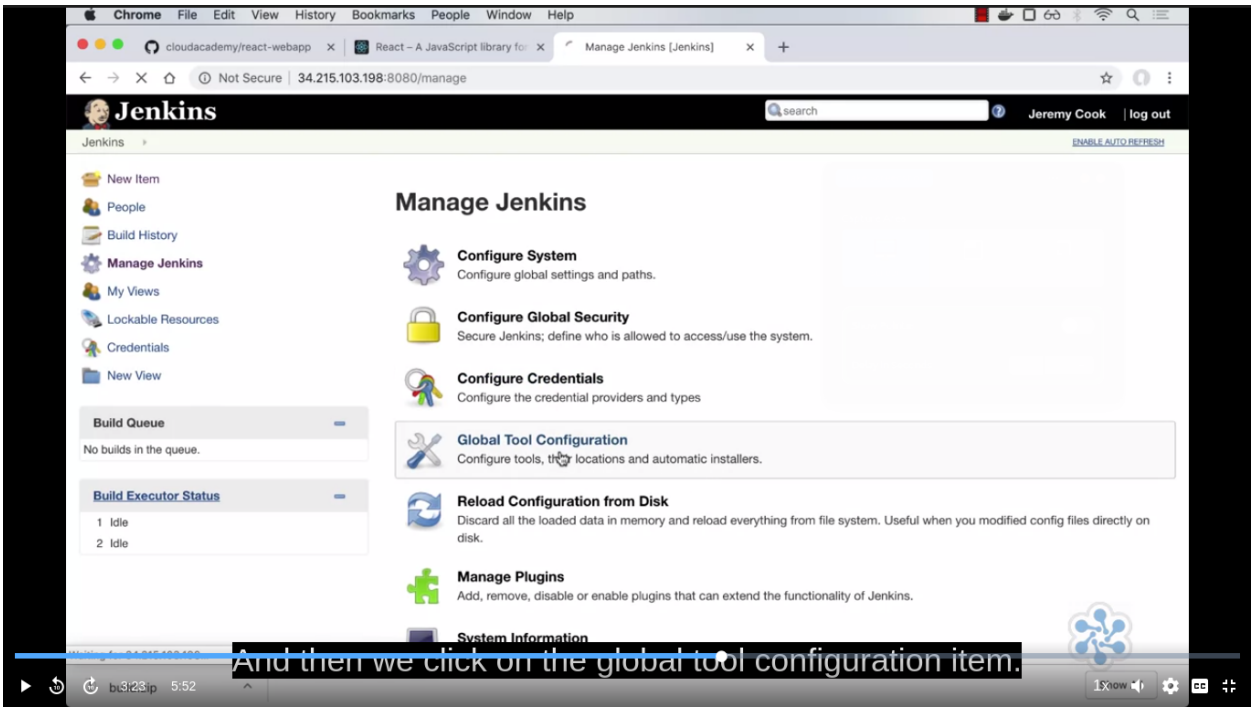
0:38 4:01 1x



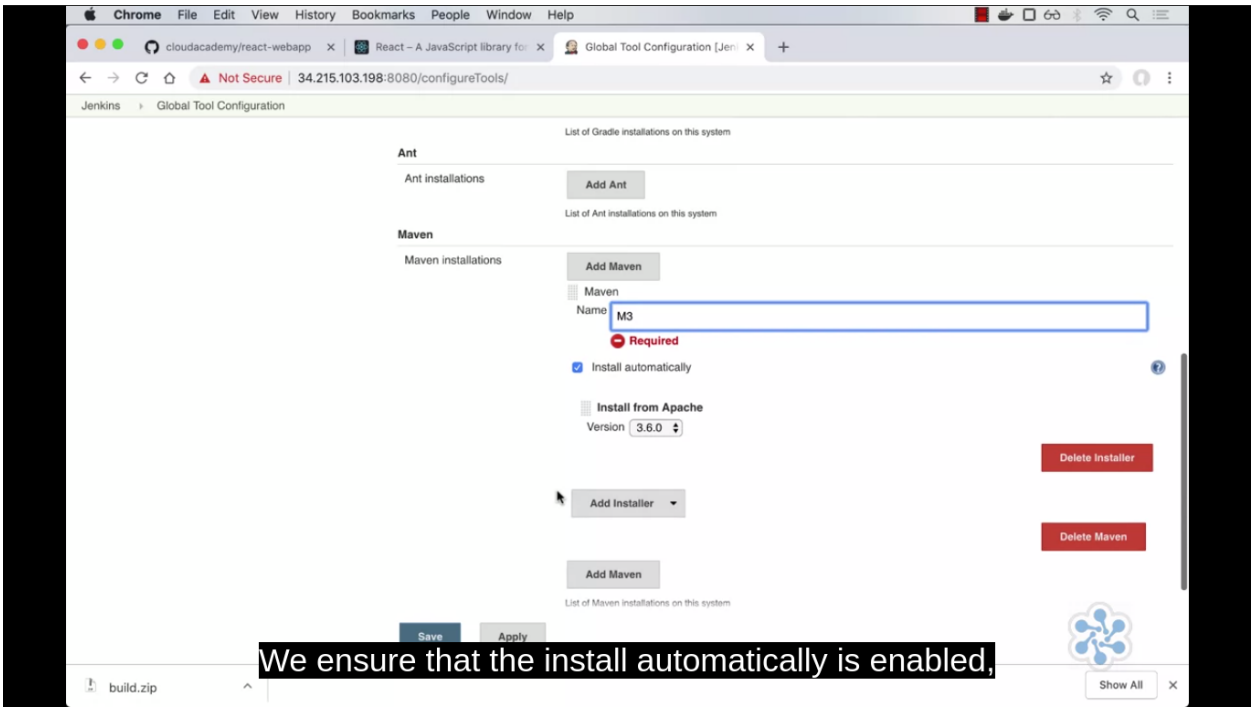

```
pipeline{
    agent any
    tools {nodejs "node}
# we are telling the jenkins nodejs has to be installed as prerequest. We want jenkins to refer the global Tool COnfiguration where we inst
    stages{
        stage('cloning'){
            steps{
                git clone "https://***.git"
            }

        }
        stage('Dependecies){
            steps{
                sh 'npm install'
            }
        }
        stage('Test){
            stpes{
                sh 'npm test'
            }
        }
        stage('Deploy'){
            steps{
                sh 'npm deploy'
            }
        }
    }
}
```

Scripted Pipeline

Lets add Maven in Global Tool Configuration



Maven will be automatically installed and it uses the name **M3**



@@

Decalrative Pipeline

Lets Build a Gradle Project

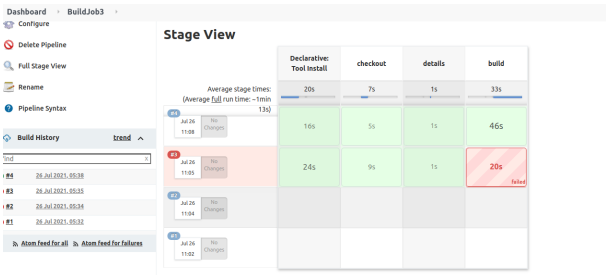
Some References

https://gist.github.com/HarshadRanganathan/97feed7f91b7ae542c994393447f3db4

Pipeline: SCM Step

Jenkins - an open source automation server which enables developers around the world to reliably build, test, and deploy their software

https://www.jenkins.io/doc/pipeline/steps/workflow-scm-step/



We are building this Project using Gradle

First Install the Gradle Globally in Jenkins using the Tool

Second, Write the Declarative Pipeline code as Below

```
pipeline{

    agent{
        label 'master'
    }
    tools {
        gradle 'gradle-4.10.2'
    }
    environment {
        VERSION = 'jellybean'
    }
    stages{
        stage('checkout'){
        steps{
            checkout ([
                $class: 'GitSCM',
                branches: [[name: 'master']],
                extensions: [[$class: 'WipeWorkspace']],
                userRemoteConfigs: [[url: 'https://github.com/clou
            ])
        }
        }
        stage('details'){
            steps{
                echo "Running ${env.BUILD_ID} on ${env.JENKINS_URL
                echo "${env.VERSION}"
            }
            when{
                environment name: 'VERSION' , value: 'jellybean'
            }
        }
        stage('build'){
            steps{
                sh "gradle build"
            }
        }
    }


}
```

Declarative Pipeline Syntax

First Declarative Pipeline

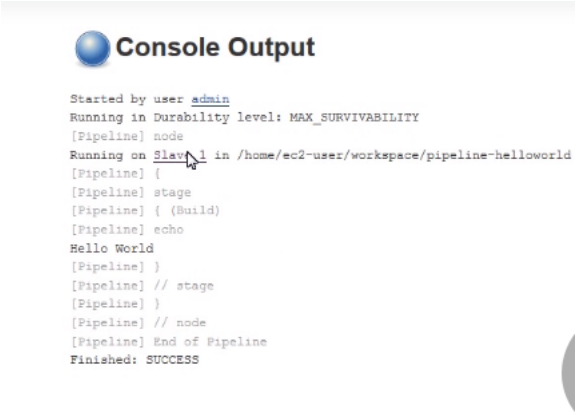
```
pipeline{   # First Line is pipeline which make it as a declarative pipeline
    agent any  # Agent defines on which machine we have to execute the Code
    stages{
        stage("Build"){
            steps{
                echo "First Build Process"
            }
        }
    }
}
```

Scripted Pipeline

```
It always Start with Node
node {
    echo "This is scripted pipeline "
}
```

If we have master slave architecture in Jenkins we can specify the agent name and it will be running in that Slave machine

```
we are going to run in slave1 MAchine

agent {
label 'slave1'
}

WE can rewrite the same as below

agent {
node {
 label 'slave1'                                  # we specify the slave Machine
  customWorkspace "/home/ec2-user/customWorkspace"   # if we want to override the default folder of jenkins
}
}
```

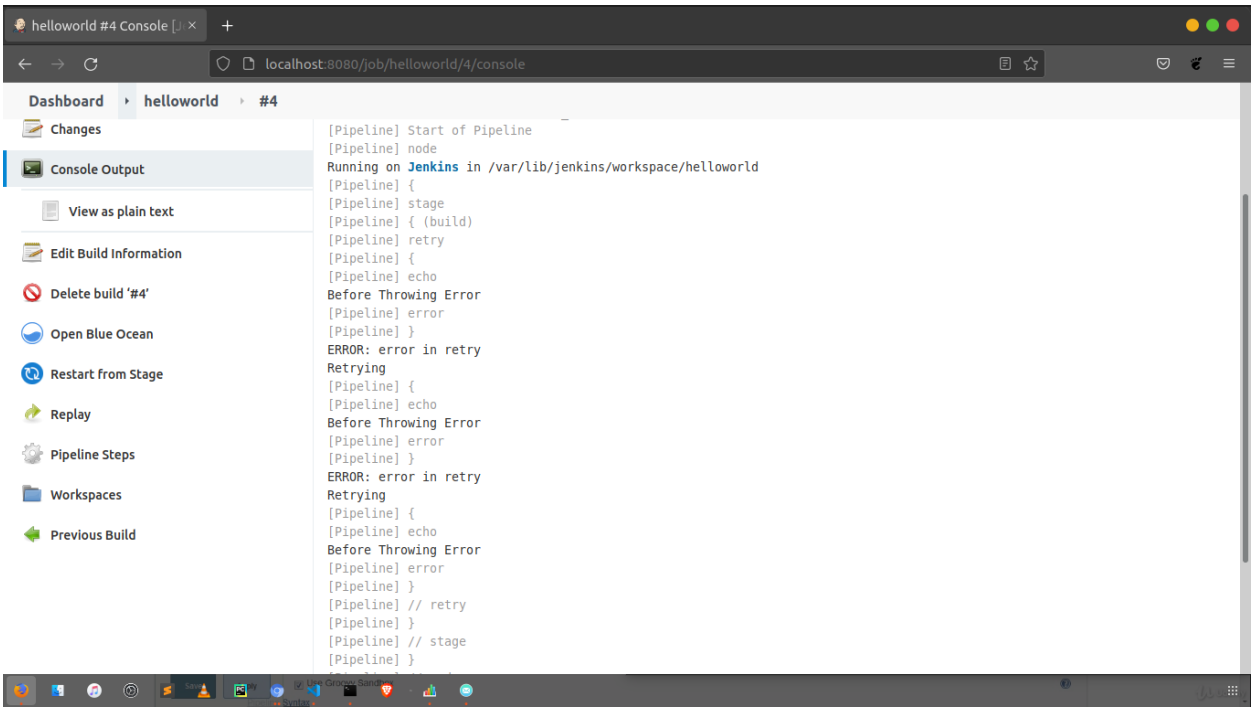


if We want to run the jenkins Script

```
pipeline{
    agent any
    stages{
        stage("Build"){
            steps{
                script{                       # Using this Script Tag we can specify/write the groovy scripts
                    def name = "esak"

                    if (name == "esak")
                        print("Hi ${name}")
                    else
                        print("Hi Guest")

                    sleep 2
                    echo "End of Program"
                }
            }
        }
    }
}
```

Retry Option

```
we are going to retry a piece of code for 3 times if it throws any error
pipeline{
    agent any
    stages{
        stage('build'){
            steps{
                retry(3){
                    echo "Before Throwing Error"
                    error "error in retry"
                }

                echo "====="
                echo "after retry(3)"   # this wont be printed if we get any error Message
            }
        }
    }
}
```



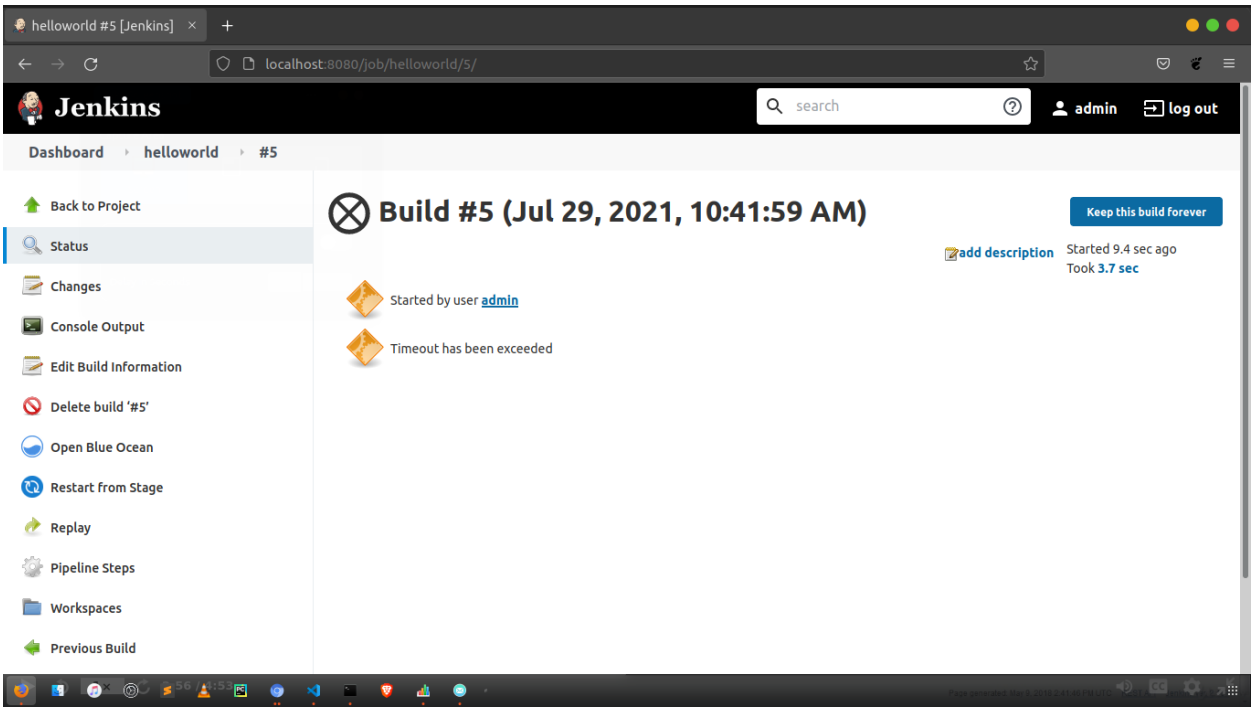
Timeout

```
pipeline{
    agent any
    stages{
        stage('build'){
            steps{
                timeout(time: 1 , unit: 'SECONDS'){
                    echo "sleeping in timeout"
                    echo "-----"
                    sleep 2
                }
            }
        }
    }
}
```

```
pipeline{
    agent any
    stages{
        stage('build'){
            steps{
                retry(3){
                    timeout(time: 1 , unit: 'SECONDS'){
                    echo "sleeping in timeout"
                    echo "-----"
                    sleep 2
                }
                }

            }
        }
    }
}
```

Check the colour of the Build#5 , it means timeout

as per the code we have provide the timeout option as 1 seconds , but we have waited for 2 seconds causing the build to Failure



Tool is used to refer the already installed global tool configuration tools and made it available in the Jenkins pipeline

```
pipeline{
    agent any
    stages{

        stage('build'){
            tools{
                maven 'M3'    ### 'M3' is the name of the maven installation in the global configuration
            }
            steps{
                sh 'mvn --version'
            }
        }

    }
}
```

```
Running on Jenkins in /var/lib/jenkins/workspace/helloworld
[Pipeline] {
[Pipeline] stage
[Pipeline] { (build)
[Pipeline] tool
Unpacking https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.6.0/apache-maven-3.6.0-bin.zip to /var/lib
/jenkins/tools/hudson.tasks.Maven_MavenInstallation/M3 on Jenkins
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ mvn --version
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-25T00:11:47+05:30)
Maven home: /var/lib/jenkins/tools/hudson.tasks.Maven_MavenInstallation/M3
Java version: 11.0.11, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: en_IN, platform encoding: UTF-8
OS name: "linux", version: "5.11.0-25-generic", arch: "amd64", family: "unix"
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Option and Retry

```
# retry the build job for 3 times if we get any error

pipeline{
    agent none
    stages{
        stage('build'){
            options{
                retry(3)
            }
            steps{
                echo "Before Error Statement"
                error "Error in the build Job"
                echo "After Error Statement"
            }
        }
    }
}
```

Predefined varibale List

Global Variable Reference

The docker variable offers convenient access to Docker-related functions from a Pipeline script. Methods needing a Jenkins agent will implicitly run a node {...} block if you have not wrapped them in one. It is a good idea to enclose a block of steps which should all run on the same node in such a block yourself.

https://opensource.triology.de/jenkins/pipeline-syntax/globals

currentBuild.result= 'FAILURE' will make the script to fail

```
pipeline{
    agent none
    stages{
        stage('build'){
            options{
                retry(3)
            }
```

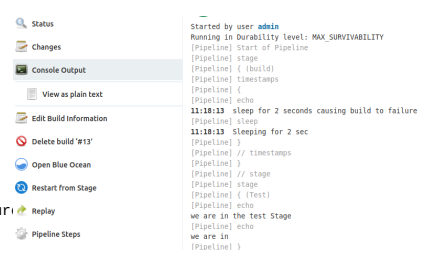
```
            steps{
                echo "Before Error Statement"
                script{
                    curre4ntBuild.result = 'FAILURE'
                }
                echo "After Error Statement"
            }
        }
    }
}
```

eventhough we specified retry and if timeout exceeds there wont be any more retries it simply exists out

TimeStamp

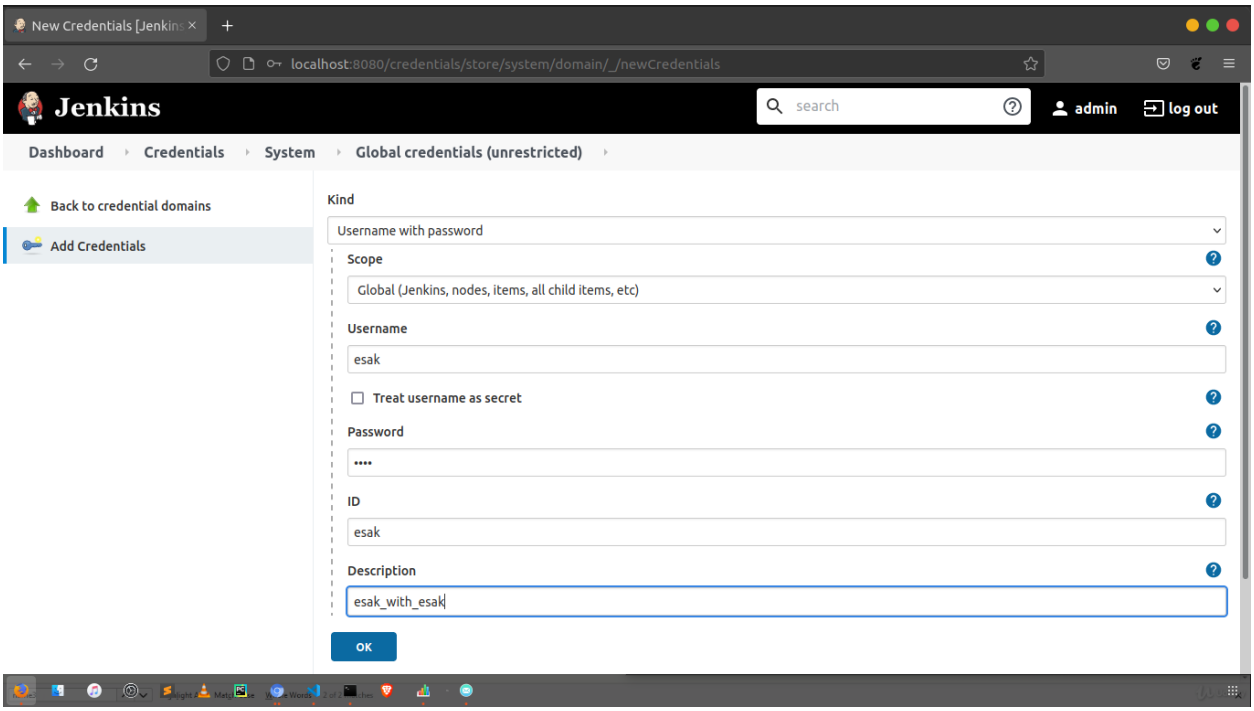
If we want to have timestamp , just like the log use timestamp plugin

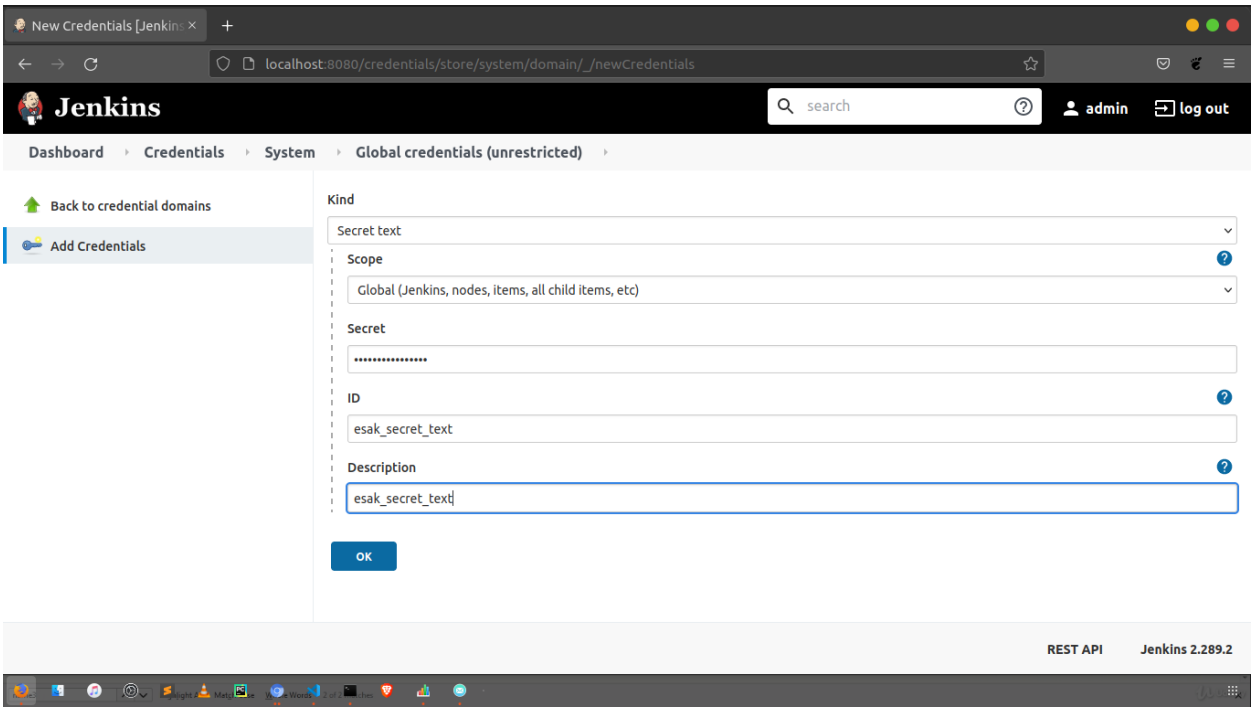
```
pipeline{
    agent none
    stages{
        stage('build'){
            options{
                timestamps()
            }
            steps{
             echo "sleep for 2 seconds causing build to failur
             sleep 2
            }
        }

        stage('Test'){
            steps{
                echo "we are in the test Stage"
                echo "we are in"
            }
        }
    }
}
```




```
pipeline {
    agent none
    environment {
        name1 = "esakki"
        name2 = "sankar"
    }
    stages {
        stage('build') {

            steps{
                echo "name1 ${name1}"
                echo "name2 ${name2}"
                sh "printenv"
            }

        }
    }
}
```

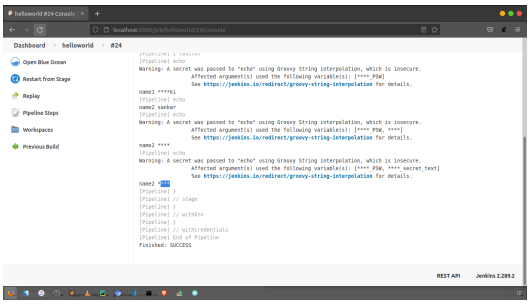
Lets Add Secret






```
pipeline {
    agent none
    environment {
        name1 = "esakki"
        name2 = "sankar"
        esak = credentials('esak')
        esak_secret_text = credentials('esak_secret_text')
    }
    stages {
        stage('build') {
```

```
            steps{
                echo "name1 ${name1}"
                echo "name2 ${name2}"
                echo "username ${esak_USR}"
                echo "password ${esak_PSW}"
                echo "name2 ${esak_secret_text}"
            }

        }
    }
}

# esak is a field having username and password.
# To access the username part we have to use esak_USR
# to access the password part user esak_PSW
# USR and PSW will be cpas always
```

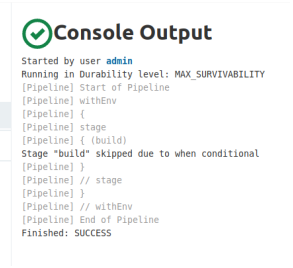


Printing the Secrets in Jenkins, it will be in the protected format

When in Jenkins

when
Required: No
At least one of the following conditions must be specified. If there is more than one condition, all conditions must return true for the stage to execute

```
pipeline{
    agent none
    environment {
        DEPLOY_TO = 'PROD'
        DEPLOY_VERSION = '2.1'
    }
    stages {
        stage('build'){
            when {
                // environment name: 'DEPLOY_TO' , value: 'PRODUCTION'
                // equals expected: 'production', actual: DEPLOY_TO
                // not {
                //     equals expected: 'production' , actual: DEPLOY_TO
                // }
                // we can use groovy expression
                // expression {
                //     DEPLOY_TO == 'QA'
                // }
                // allOf - when all the Conditions are true stage will be executed
                // allOf {
                //     environment name:DEPLOY_TO, value: "PRODUCTION"
                //     environment name:DEPLOY_VERSION, value:"2.1"
                // }
                // anyOf - when any the Conditions are true stage will be executed
                anyOf{
                    environment name:DEPLOY_VERSION, value:'2.1'
                    environment name:DEPLOY_TO, value:'PROD'
                }
            }

            steps{
                echo "Building"
            }
        }
    }

}
```

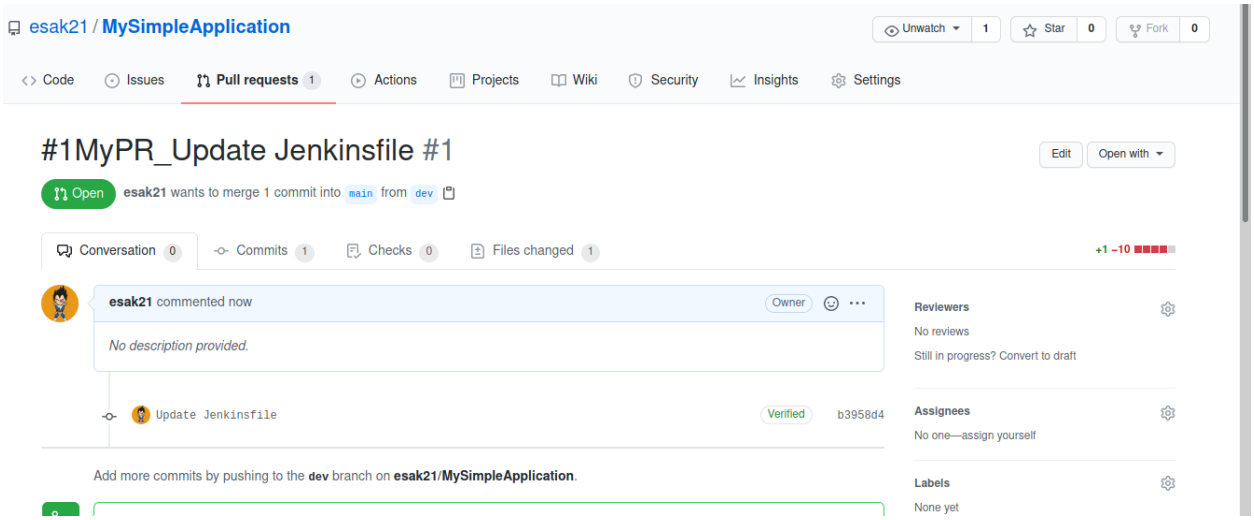


Checking the Branch Condition

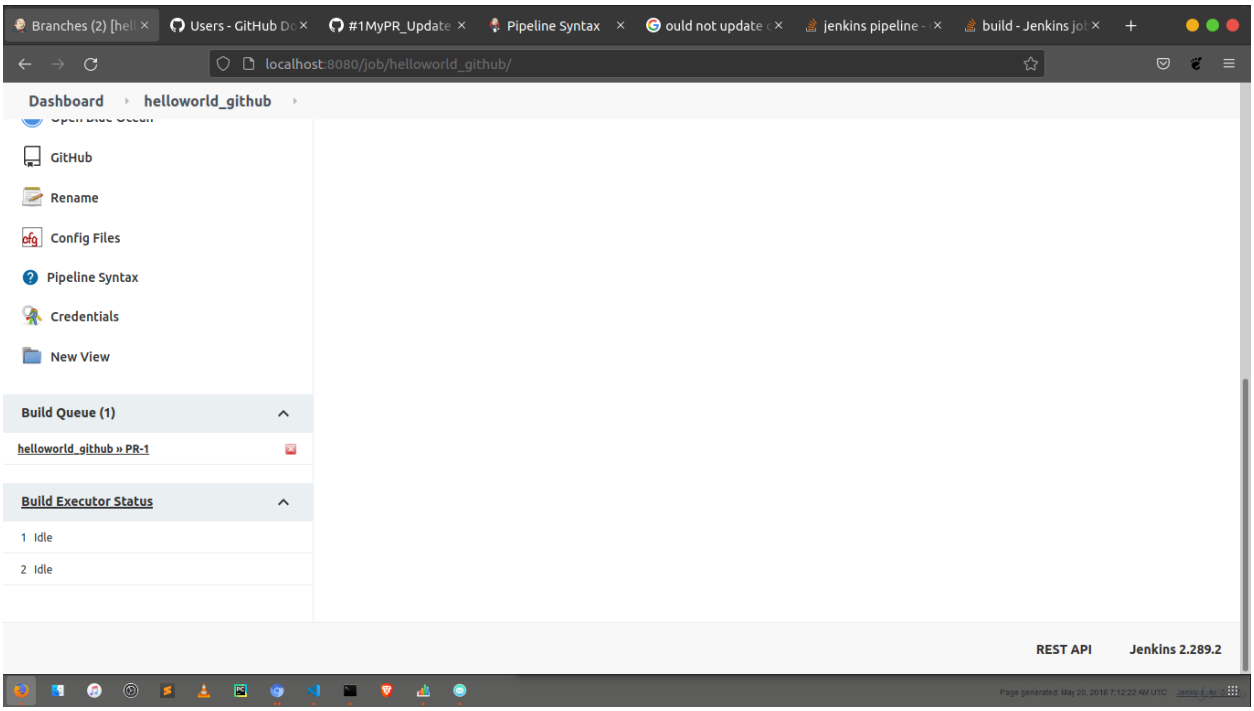
```
pipeline{
    agent none
    environment {
```

```
        DEPLOY_TO = 'PROD'
        DEPLOY_VERSION = '2.1'
    }
    stages {
        stage('build master'){
            when {
                branch 'master'
            }

            steps{
                echo "Building the master"
            }
        }

        stage('build dev'){
            when {
                branch 'dev'
            }
            steps {
                echo "Building the dev"
            }
        }
    }

}

If we want to do a Build process based on the commit message if the commit message has "some_text" , build will be executed

when {
  changelog '.*some_text.*'
}

when {
  changeRequest()
}
```

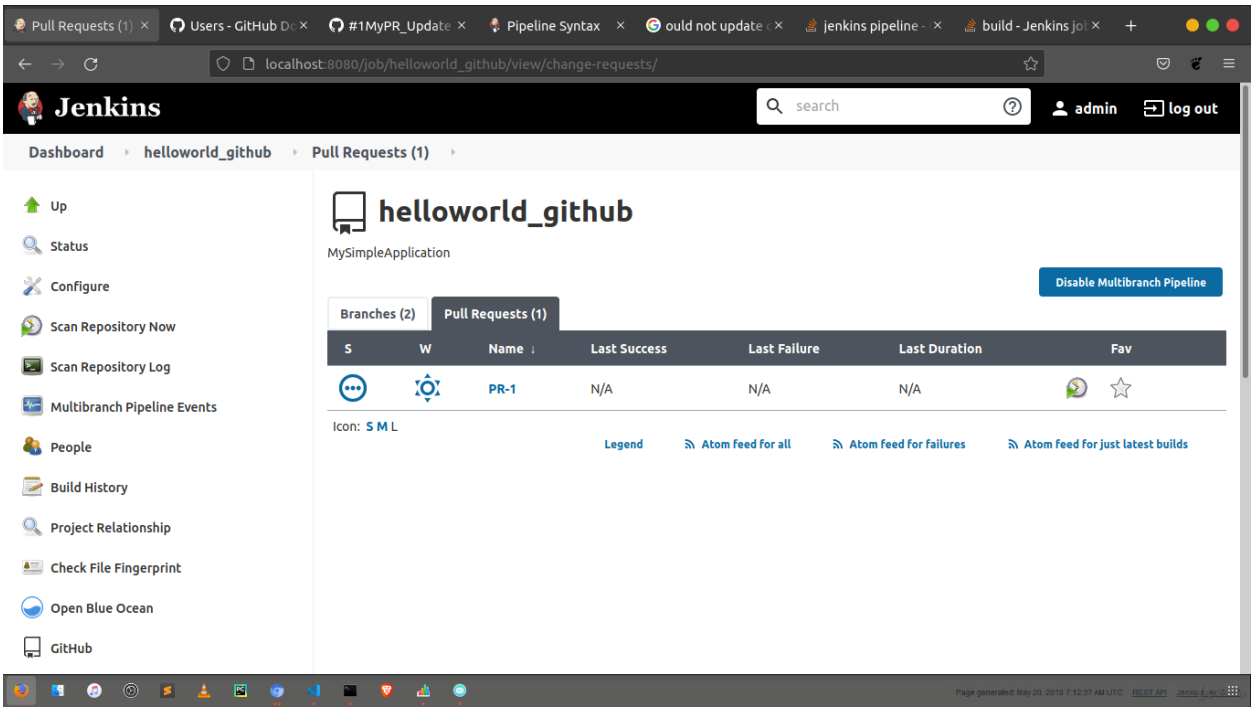
i have created a PR

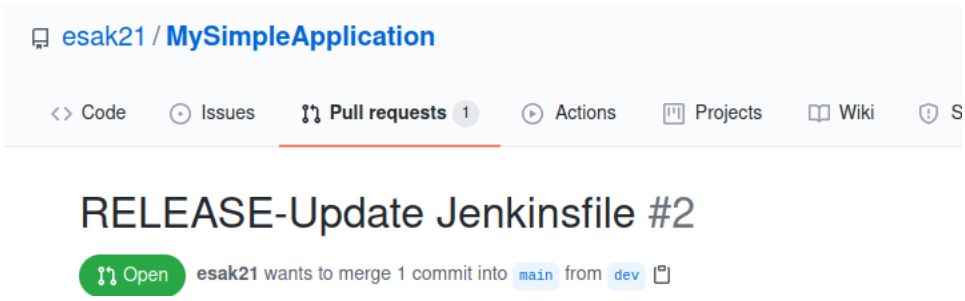


Click the Scan Repository Now, Now PR has been picked up like below



Build has been executed






```
pipeline{
    agent none
    environment {
        DEPLOY_TO = 'PROD'
        DEPLOY_VERSION = '2.1'
    }
    stages {
        stage('build master'){
            when {
                // When PR IS Created
                // changeRequest()
                // When Pr is created with the Title RELEASE-
                changeRequest title: "RELEASE-"
            }

            steps{
                echo "Building the master"
            }
        }
    }

}
```

if the commited files are .js file then kick start the build process

```
chnageset glob: "*.js"
chnageset glob: "*.js", casesensitive: true
```

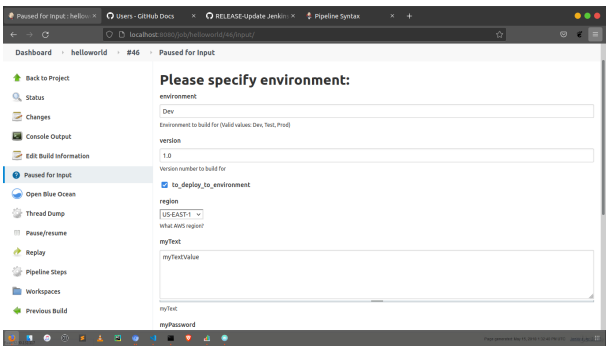
Run Jobs in parallel

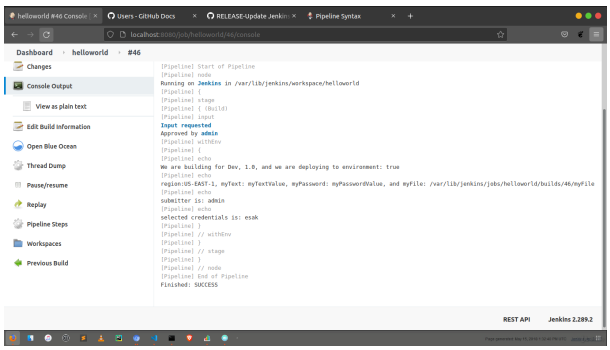
```
pipeline {
    agent none
    stages {
        stage('Staging Running in Parallel ') {
           failFast true  // if one of the stage failed , rest of the stage is also failed
            parallel {
                stage('stage1') {
                    steps{
                        echo "stage1 executing"
                        sleep 10
                    }
                }
                stage('stage2'){
                    steps{
                       echo "stage2 executing"
                        sleep 2
                    }
                }
                stage ('stage3') {
                    steps{
                    echo "stage3 executing "
                    sleep 10
                    }
                }
            }
        }
```

```
    }
}
```

Input

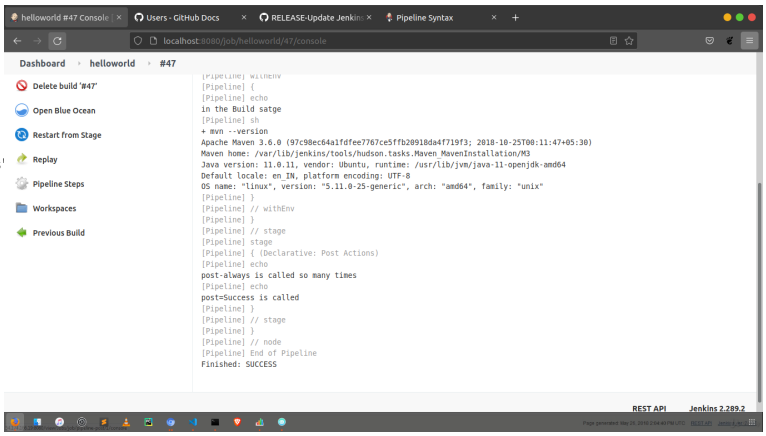
```
pipeline{
    agent any

    stages{
        stage('Build'){
      input{
                    message "Please specify environment:"
                    ok "OK"
           submitter "dummyuser,admin@localhost.com"
           submitterParameter "whoIsSubmitter"
                    parameters {
             string(name: 'environment', defaultValue: 'Dev', description: 'Environment to build for (Valid values: Dev, Test, Prod)')
             string(name: 'version', defaultValue: '1.0', description: 'Version number to build for')
             booleanParam(name: 'to_deploy_to_environment', defaultValue: true, description: '')
             choice(choices: 'US-EAST-1\nUS-WEST-2', description: 'What AWS region?', name: 'region')
             text(name:'myText', defaultValue:'myTextValue', description:'myText')
             password(name:'myPassword', defaultValue:'myPasswordValue', description:'myDescription')
             file(name:'myFile', description:'fileDescription')
             credentials(name:'myCredentials', description:'myCredentailsDesc', required:true)
                    }
                }

           steps{
          echo "We are building for ${environment}, ${version}, and we are deploying to environment: ${to_deploy_to_environment}"
          echo "region:${region}, myText: ${myText}, myPassword: ${myPassword}, and myFile: ${myFile}"
          echo "submitter is: ${whoIsSubmitter}"
          echo "selected credentials is: ${myCredentials}"
      }
        }
    }
}
```





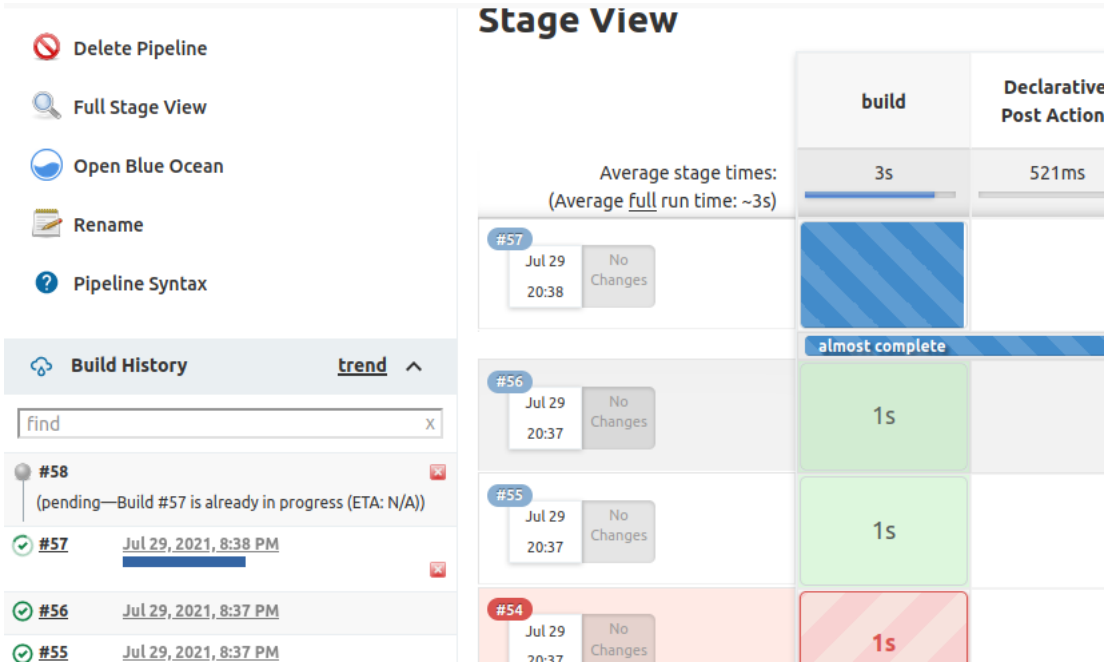
```
pipeline{
    agent any

    stages{
        stage('build'){
            tools{
                maven 'M3'
            }
            steps{
                echo "in the Build satge"
                sh 'mvn --version'
            }
        }
    }
    post{
```

```
        success{
            echo "post=Success is called"
        }
        failure{
            echo "post-> Failure is called"
        }
        always{
            echo "post-always is called so many times"
        }
    }
}
```



OPTIONS at Pipeline Level

If we disable the concurrent Build , then it will wait for the build to Finish

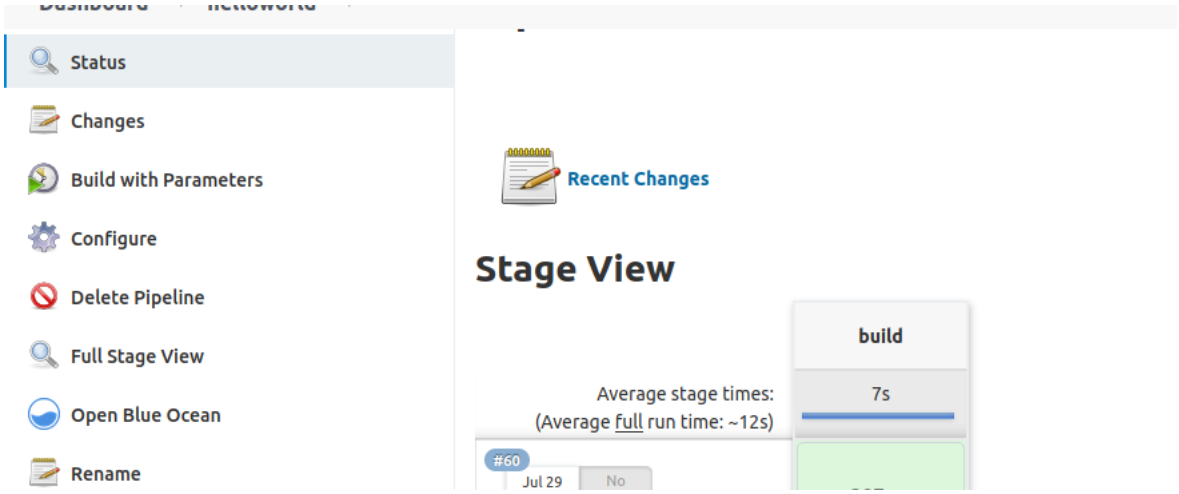



```
pipeline{
    agent any
    options{
        //buildDiscarder(logRotator(numToKeepStr: '1'))  // it will maintain only one build run
        // all the Build run will be deleted from the jenkins server
        // We can disable saqme job is active in 2 instances
        //disableConcurrentBuilds()

        //  It will avoid  every commit we push , it avoid the build behaviour
        //overrideIndexTriggers(true)
        // If the Build is unstable we will skip the result of the stages which were dependant
        //skipStagesAfterUnstable()
        //
        //checkoutToSubdirectory("mypackages")
        // when using docker , each stage will use a new docker contaienr
        newContainerPerStage()

    }
```

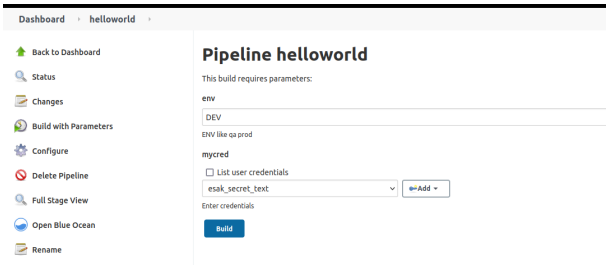
```
    stages{
        stage('build'){
            tools {
                maven 'M3'
            }
            steps{
                sh "mvn --version"
                sleep 20
            }
        }
    }
}
```

Parameters at pipeline level same as the stage

First time it run with the default parameters ,



second time it will ask for the values




```
pipeline {
    agent any
    parameters {
        string(name: 'env', defaultValue: 'DEV', description: 'ENV
        credentials(name: 'mycred', description: 'Enter credential

    }
    stages{
        stage('build'){
            steps{
                echo "building"
            }
        }
    }
}
```

Triggers

```
Cron

pipeline{
    agent none
    triggers {
        cron('0 1 2 3 1-7')
    }
    stages{
```

```
        stage('build'){
            echo "building the Java Project"
        }
    }
}

pipeline{
    agent any
    triggers {
        //cron('0 1 2 3 1-7')
        pollSCM('* * * * *')
    }
    stages{
        stage('build'){
          steps{
            checkout([
                $class: 'GitSCM',
                branches: [[name: "origin/main"]],
                userRemoteConfigs: [[
                    url: 'https://github.com/esak21/MySimpleApplication.git'
                ]]
            ])
            echo "building the Java Project"
          }
        }
    }
}


  If we want to trigger the Job based on the Previous Job ( comma will act as OR Condition Here )
pipeline{
    agent any
    triggers {
        //cron('0 1 2 3 1-7')
        //pollSCM('* * * * *')
        upstream(upstreamProjects: 'pipeline-triggers-upstream-job1,pipeline-triggers-upstream-job2',
        threshold: hudson.model.Result.SUCCESS)//UNSTABLE, FAILURE, NOT_BUILT, ABORTED
    }
    stages{
        stage('build'){
          steps{
            checkout([
                $class: 'GitSCM',
                branches: [[name: "origin/main"]],
                userRemoteConfigs: [[
                    url: 'https://github.com/esak21/MySimpleApplication.git'
                ]]
            ])
            echo "building the Java Project"
          }
        }
    }
}
```

Jenkins With Docker Declarative Pipeline

```
pipeline {
    agent {
        docker{
            image 'maven:3.8.1-jdk-11-slim'
            args '-e someEnv=dev'
            alwaysPull false
            // If jenkins has write Permission it will create the Folder in the server and copy the artifacts to it
            customWorkspace '/var/lib/jenkins/workspace/MyCustomSpace'
            // if we have any slaves to run we can use it like
            //label 'slave1'
        }
    }
    stages{
        stage('Build'){
            steps{
                sh 'mvn -version'
                sh 'echo ${someEnv}'
            }
        }
    }
}
```

DockerFile

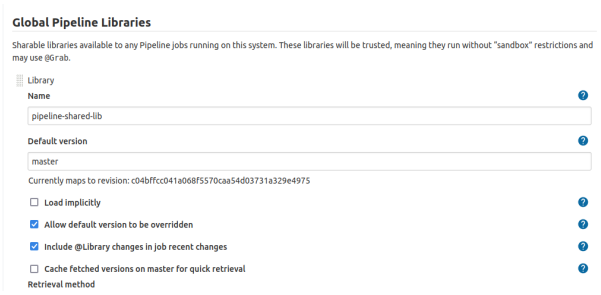
```
DockerFile has to be  in the Root Directory

pipeline{
    agent any
    stages {
        stage('checkout'){
            steps{
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: 'origin/main']],
                    userRemoteConfigs:[[
                        url: 'https://github.com/esak21/MySimpleApplication.git'
                    ]]
                ])
            }
        }
        stage("build"){
            steps{
                sh 'cat /etc/lsb-release'
                echo 'Build Completed'
            }
        }
    }
}

If the dockerFile is nort avvailble in the root of the projetc directory we have to specify it manually

inside the stages we have to specify it
stage('dockerFile'){
            steps{
                dockerfile{
                    dir 'prod/docker'
                      If we specify the file with different name then
                      filename 'mycustomDockerFile'
                }
            }
        }
```

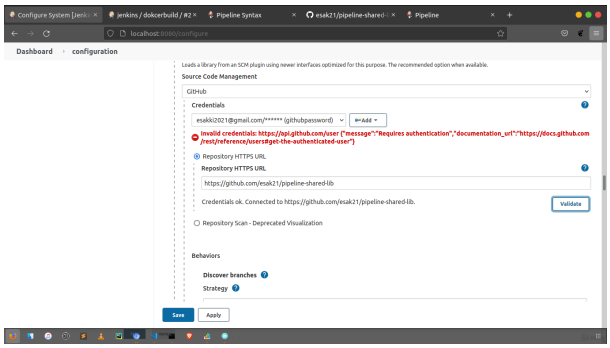
We can run groovy code in the declarative pipeline using script tag

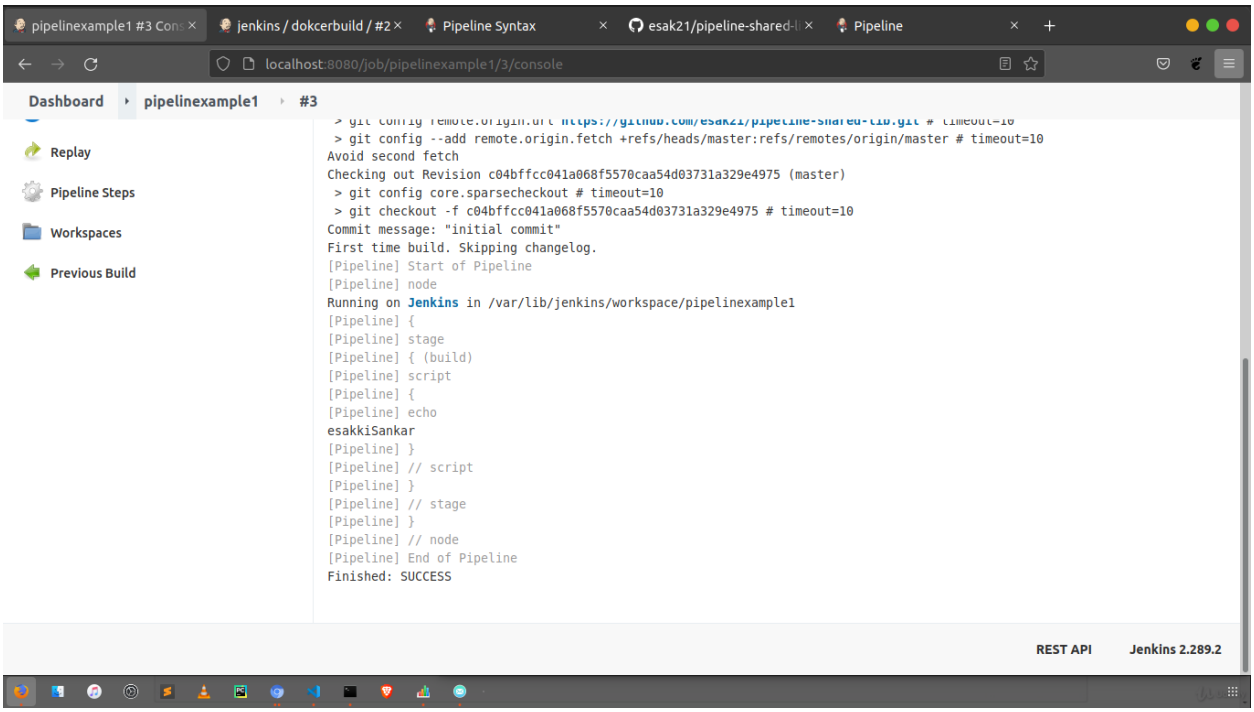
```
def nameOfPerson = "john"

def printName(name){
    echo name
}

pipeline{
    agent any
    stages{
        stage('build'){
            steps{
                    script{
                        printName(nameOfPerson)
                    }
            }
        }
    }
}
```

Adding common Pipeline groovy code as lib

pipeline-shared-lib








```
@Library('pipeline-shared-lib') _     // we are importing all the functions

pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                script {
                    swissknife.printName "esakkiSankar"
                }
            }
        }
    }
}
Above code uses the master Branch , if incase you want to use other branches like dev the
@Library('pipeline-shared-lib@dev') _

@Library(['pipeline-shared-lib@dev', 'pipeline-shared-lib-code']) _
```

Scripted Pipeline

```
### Scripted pipeline always start with "node"

node {
    echo "This is scripted pipeline "
}


node {
    stage('build'){
        // we can run this echo Step in any node
        node {
        echo "This is scripted pipeline "

        }

    }
}


####
Without node block also we can write the code

stage{
    echo  "Stage1"
}

stage {
    echo "stage 2"
}


###################################
# Using Docker Image with the scripted pipeline
# uncheck the Groovy sandbox
# https://stackoverflow.com/questions/38276341/jenkins-ci-pipeline-scripts-not-permitted-to-use-method-groovy-lang-groovyobject

node {
    stage('build') {
        def mavenImage = docker.image('maven:3.8.1-jdk-11-slim')
        mavenImage.inside{
            sh 'mvn --version'
        }
    }
}


node {
    stage('build') {
        def mavenImage = docker.image('maven:3.8-jdk-8')
        // To pull the Image
        mavenImage.pull()

        mavenImage.inside(
            '-e someENV=dev',
            { sh 'echo $someENV'}
        )
    }
}



################################# To USe Docker File
node {
  stage('Build'){
    checkout([$class: 'GitSCM',
      branches: [[name: "origin/master"]],
      userRemoteConfigs: [[
      url: 'https://github.com/pipelineascodecourse/pipeline-agent-dockerfile.git']]
    ])

    def myCustomUbuntuImage = docker.build("my-ubuntu:my-latest")

      def myCustomUbuntuImage = docker.build("my-ubuntu:my-latest", "--file mycistomdockerFile . ")

    myCustomUbuntuImage.inside {
      sh 'cat /etc/lsb-release'
    }

  }
```

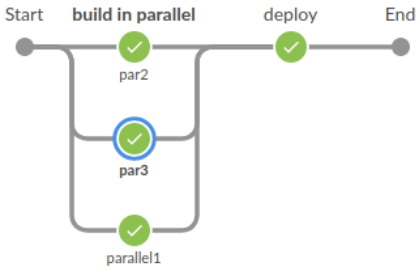
```
}



node {
  stage('Build'){
    checkout([$class: 'GitSCM',
      branches: [[name: "origin/master"]],
      userRemoteConfigs: [[
      url: 'https://github.com/pipelineascodecourse/pipeline-agent-dockerfile.git']]
    ])

    def myCustomUbuntuImage = docker.build("my-ubuntu:my-latest","--tag mydockerfile:example .")

    myCustomUbuntuImage.inside {
      sh 'cat /etc/lsb-release'
    }
  }
}



## Getting SHell values and exit status
node{
    stage('build'){
        def shelloutput = sh(
            script: "ls -la",
            returnStdout: true
        ).trim()

        echo "Shell Output is ${shelloutput}"

        def shellstatus = sh(
            script: "exit 1",
            returnStatus: true
        )

        echo "Shell outout is ${shellstatus}"
    }
}

# Lets REtry the Build
node {
    stage("build"){
        retry(3) {
            error "Manual Error thrown by me"
        }
    }
}

### TImeOut
# ##### Pipeline will be aborted after the timeout was exceeded
node {
    stage("build"){
        timeout(time: 1, unit:'SECONDS') {
            sleep 5
        }
    }
}


##########################
node {
    timestamps{

    stage("build"){
        echo "we have timestap"
    }
    stage("deploy"){
        echo "we have timestamp"
    }
    }
}

#########################################################
# Credentials
node {
    withCredentials( [ usernamePassword(
        credentialsId: "artifactory",
        usernameVariable: 'USER_NAME',
        passwordVariable: 'PASSWD'
```

```
    )] ){
        echo "USER is ${USER_NAME}"
        echo "PASSWORD is ${PASSWD}"
    }
}


###################################
If we want to implement when we have to use the if condition

node {
    def name ="mycoolday"
    def isGroovyCool = false

    withEnv(['DEPLOY_TO=production']){
        stage('build'){

            //when env condition
            if(env.DEPLOY_TO == 'production')
                println "name is ${name}"

            if (isGroovyCool == false)
                println "groovy is always Cool"

            if ( name == "mucoolday" && !isGroovyCool)
               println "name is ${name} and grovvy is Cool"
        }
    }
}

node {
    if (env.BRANCH_NAME == "dev"){
        stage('build'){
            echo "building the Dev stage"
        }
    }
}

## When PR is Ready

node{
  stage('Build'){
    println "env.CHANGE_ID: ${env.CHANGE_ID}"
    println "env.CHANGE_URL: ${env.CHANGE_URL}"
    println "env.CHANGE_TITLE: ${env.CHANGE_TITLE}"
    println "env.CHANGE_AUTHOR: ${env.CHANGE_AUTHOR}"
    println "env.CHANGE_AUTHOR_DISPLAY_NAME: ${env.CHANGE_AUTHOR_DISPLAY_NAME}"
    println "env.CHANGE_AUTHOR_EMAIL: ${env.CHANGE_AUTHOR_EMAIL}"
    println "env.CHANGE_TARGET: ${env.CHANGE_TARGET}"

    if(env.CHANGE_TITLE == "when-pr"){
      echo "pull request is found"
    }
    else {
      echo "pull request is not found"
    }
  }
}


## Running Test IN Parallel
# 3 jobs will run in parallel and when its completed next stage will be executed
# if any of thenm is failed pipeline will be failed

node {
    stage('build in parallel'){
        parallel 'parallel1':{
            echo "Pip 1 is running "
            sleep 10
        },
        "par2":{
            echo "pip2 is running"
            sleep 5
        },
        "par3": {
            echo "par3 is running"
            sleep 10
        },
        failFast:true
    }
    stage("deploy"){
```

```
        echo "deploying"
    }
}

# Getting Input

node {
    def userInput = ""
    stage("build"){
        userInput = input ( id: 'userInput',

            message: 'please specify the environment',
            submitterParameter: "wholesubmitter",
            parameters:[
                string(name:"environment", defaultValue:"dev", description:"Server to be deployed"),
                string(name:"version", defaultValue: "1.0"),
                choice(choices:"US_EAST_1\nUS_WQEST_2", description :"aws region", name:"region"),
                credentials(name:"mycdredentials", required:true, description:"AWS CREDENTIALS")
            ])

        echo "we are building for ${userInput.environment}"
        echo "we are building in ${userInput.region}"
        echo "whole code ${userInput.wholesubmitter}"
    }
}



If the user doesnt provide the values within the specific periiod, if user dont specify any input within the timeout we get some exception

node{
    try{

    stage("build"){
        echo "building"
        currentBuild.result = "SUCESS"
    }
    }catch(err){

    }finally{
        def currentResult = currentBuild.result
        echo "always will be executed"
        if ( currentResult == "SUCESS"){
            println "we are in finally block"
        }
    }
}


# BuildDiscarder
Only one Build History will be maintained by jenkins

node {
    properties(
        [
            buildDiscarder(logRotator(numToKeepStr: '1'))
        ]
    )

    stage("build"){
        echo "Hello World"
    }
}

# DIsable Concurrent Builds

node {
    properties(
        [
            buildDiscarder(logRotator(numToKeepStr: '10')),
            disableConcurrentBuilds()
        ]
    )

    stage("build"){
        sleep(time:10, unit:'SECONDS')
        echo "Hello World"
    }
}
## Download workspace insa custom Folder
node {
```

```
    properties(
        [
            buildDiscarder(logRotator(numToKeepStr: '10')),
            disableConcurrentBuilds()
        ]
    )
    stage("build"){
        ws("/var/lib/jenkins/workspace/scp1/myfolder") {

            checkout([
                $class: 'GitSCM',
                branches: [[name: "origin/main"]],
                userRemoteConfigs:[[

                url: 'https://github.com/esak21/IACS_BRAINIACS.git'
                ]]
            ])

                sleep(time:10, unit:'SECONDS')
                echo "Hello World"
        }
    }
}

## pULLING DOCKER IMAGE FOR EACH sTAGE
node{
    stage('build'){

        def mavenImage = docker.image("maven:3.5.3-jdk-10-slim")
        mavenImage.inside{
              sh "mvn -v"
         }

    }
    stage('deploy'){
            def mavenImage = docker.image("maven:3.5.3-jdk-10-slim")
            mavenImage.inside{
                sh "mvn -v"
            }
    }
}


###################################
# imp Parametres ###
node() {
   // adds job parameters within jenkinsfile
   properties([
     parameters([
       string(
         defaultValue: 'Dev',
         description: 'Environment to build for (Valid values: Dev, Test, Prod)',
         name: 'environment'
       ),
       booleanParam(
         defaultValue: true,
         description: '',
         name: 'to_deploy_to_environment'
       ),
     choice(
    choices: 'US-EAST-1\nUS-WEST-2',
    description: 'What AWS region?',
    name: 'region'
    ),
     text(
     name:'myText',
     defaultValue:'myTextValue',
     description:'myText'
     ),
     password(
     name:'myPassword',
     defaultValue:'myPasswordValue',
     description:'myDescription'
     ),
     file(
     name:'myFile',
     description:'fileDescription'
     ),
     credentials(
     name:'myCredentials',
     description:'myCredentailsDesc',
```

```
     required:true)
     ])
   ])

    echo "We are building for ${params.environment}, and we are deploying to environment: ${params.to_deploy_to_environment}"
  echo "region:${params.region}, myText: ${params.myText}, myPassword: ${params.myPassword}, and myFile: ${params.myFile}"
  echo "selected credentials is: ${params.myCredentials}"
}


node {
    def mavenHome = tool name:'M3' , type: 'maven'
    sh "${mavenHome}/bin/mvn -v "
}

#################################3
# Triggers
node {
    properties([

        pipelineTriggers([

            cron('0 1 2 3 1-7')
        ])
    ])

    stage('build'){
        echo "helloworld"
    }
}

####
# Git piolling log will poll the git for every minute
# pollSCM every 1 minute s
node{
    properties([
        pipelineTriggers([
            pollSCM('* * * * * ')
        ]),
    ])
    stage('checkout'){
        checkout([
            $class: 'GitSCM',
            branches:[[name: "origin/main"]],
            userRemoteConfigs:[[
                url: "https://github.com/esak21/IACS_BRAINIACS.git"
            ]]
        ])
    }
}
```



Integrate Jenkins With SonarQube, Artifactory ,

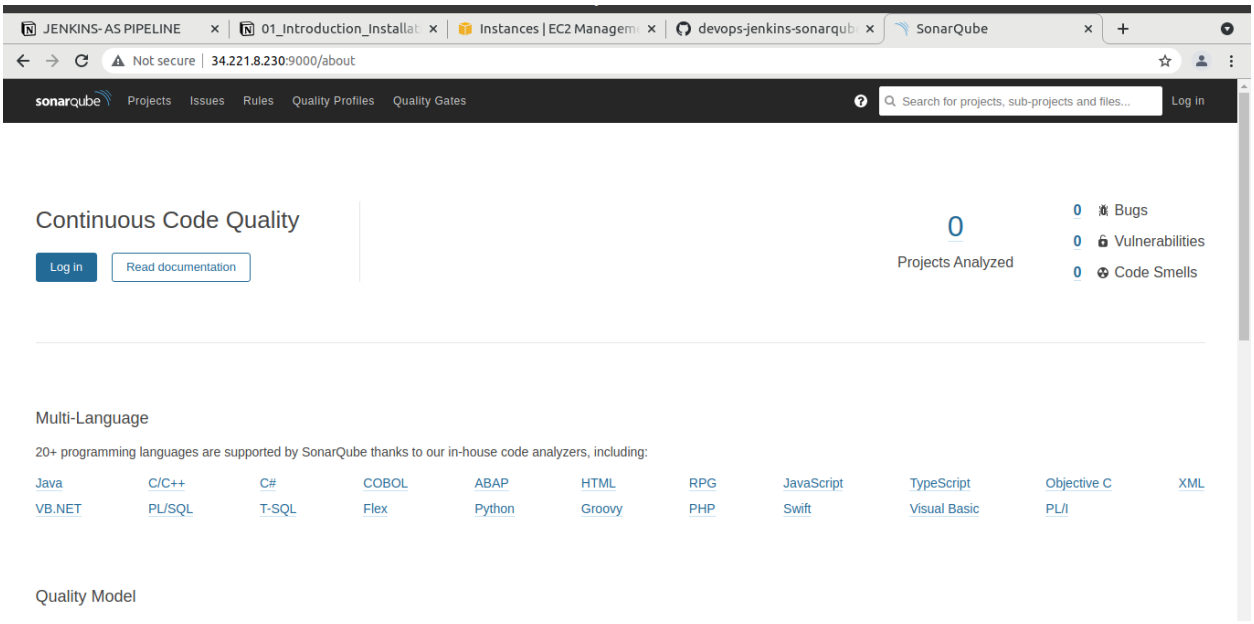
IssuesFaced in Sonar Qube

change the Vm memeory to 2 GB

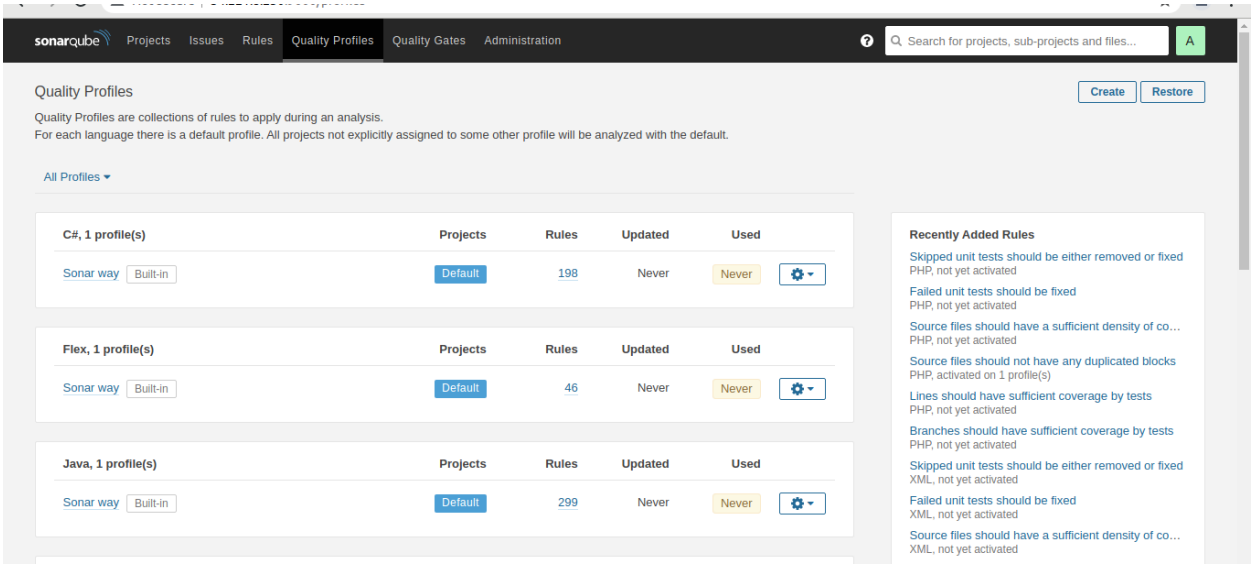
```
sysctl -w vm.max_map_count=262144
systemctl restart docker
```

Run the Below Docker Compose File

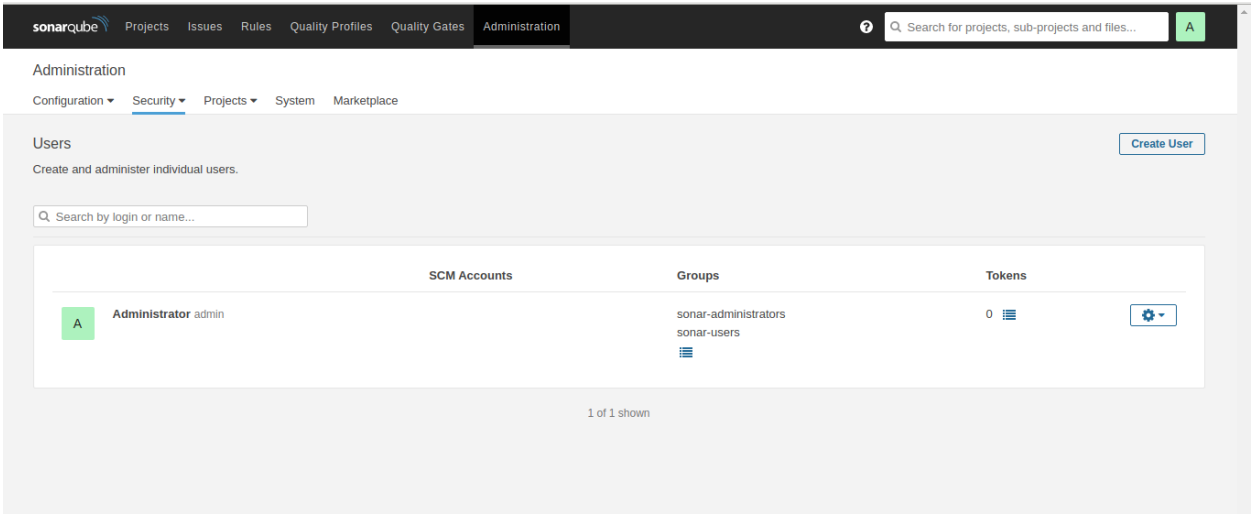
```
### JENKINS FILE

node {

    def GRADLE_HOME = tool name:gradle-4.10.2
    def SONARQUBE_HOSTNAME= 'sonarqube'
    stage('checkout'){
        git url: 'https://github.com/cloudacademy/devops-webapp.git'
    }

    stage('build'){
        sh '${GRADLE_HOME}/bin/gradle build'
    }

    stage('sonar-scanner'){
        def sonarqubescannerHome = tool name: 'sonar' , type: 'hudson.plugins.sonar.SonarRunnerInstallation'

        withCredentials([string(credentialId: 'sonar', variable:' sonarLogin')]){
            sh "${sonarqubescannerHome}/bin/sonar-scanner -e -Dsonar.host.url=http://${SONARQUBE_HOSTNAME}:9000 -Dsonar.login=${sonarLogin}
        }
    }

}
```



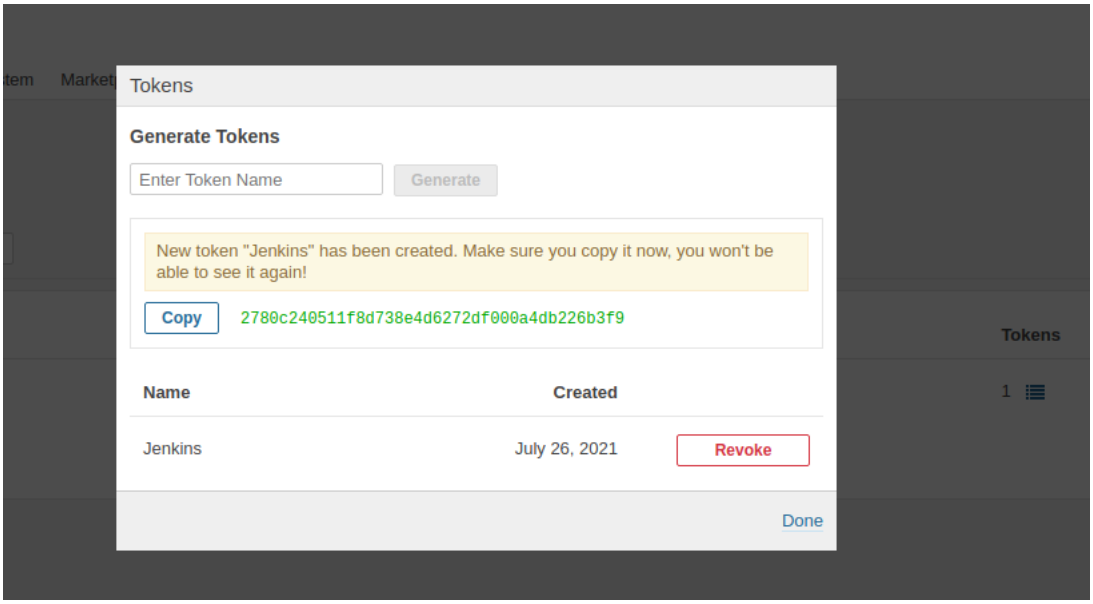
Click om the Quality Profile



Admin - > Security → users



Lets Generate a Token 2780c240511f8d738e4d6272df000a4db226b3f9

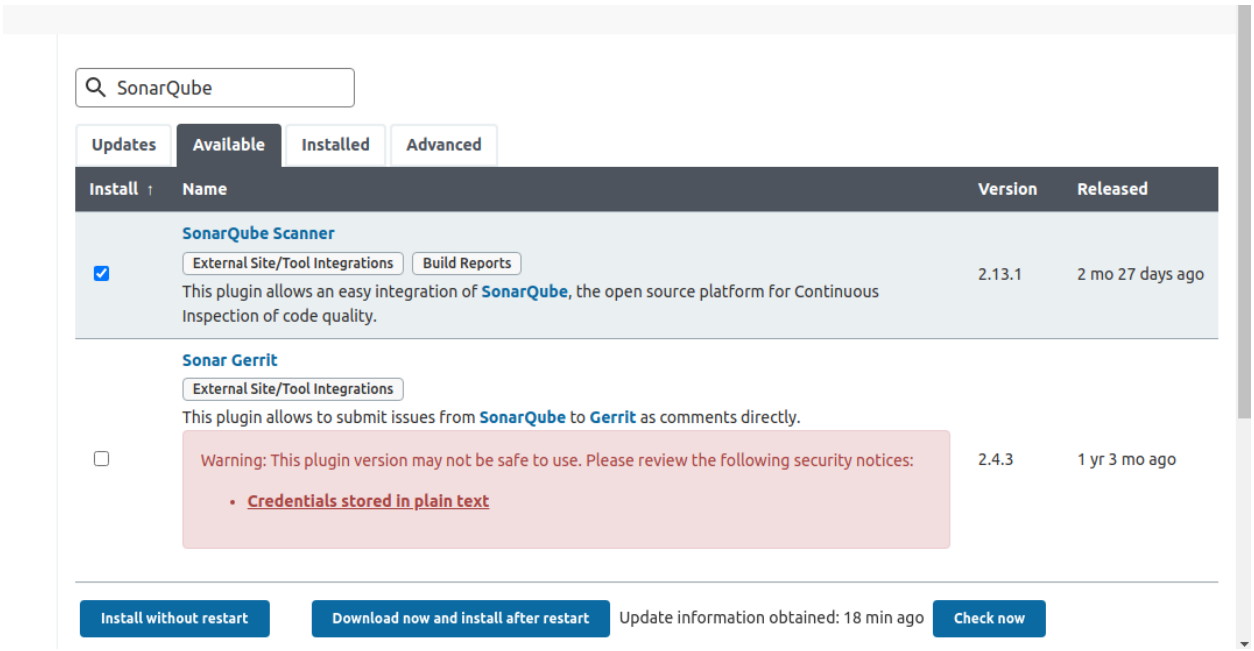


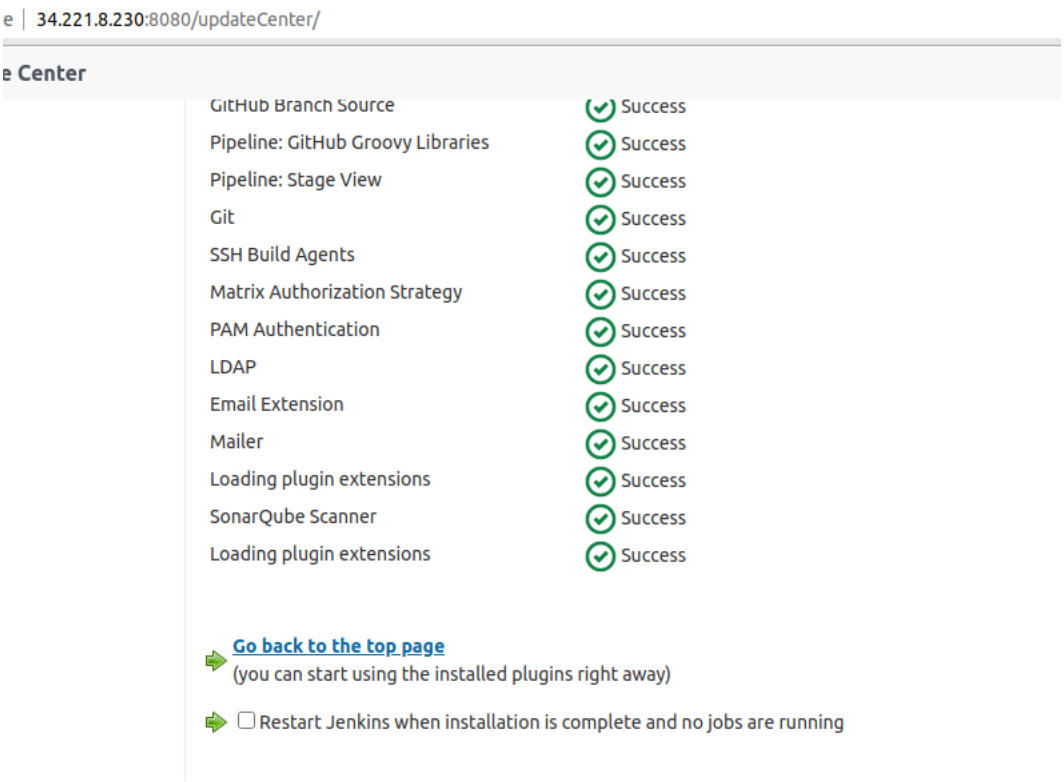
Install Jenkins in the EC2 machine

Install Sonar Qube Scanner Plugin

Manage Jenkins → Manage Plugins → Plugin Manager

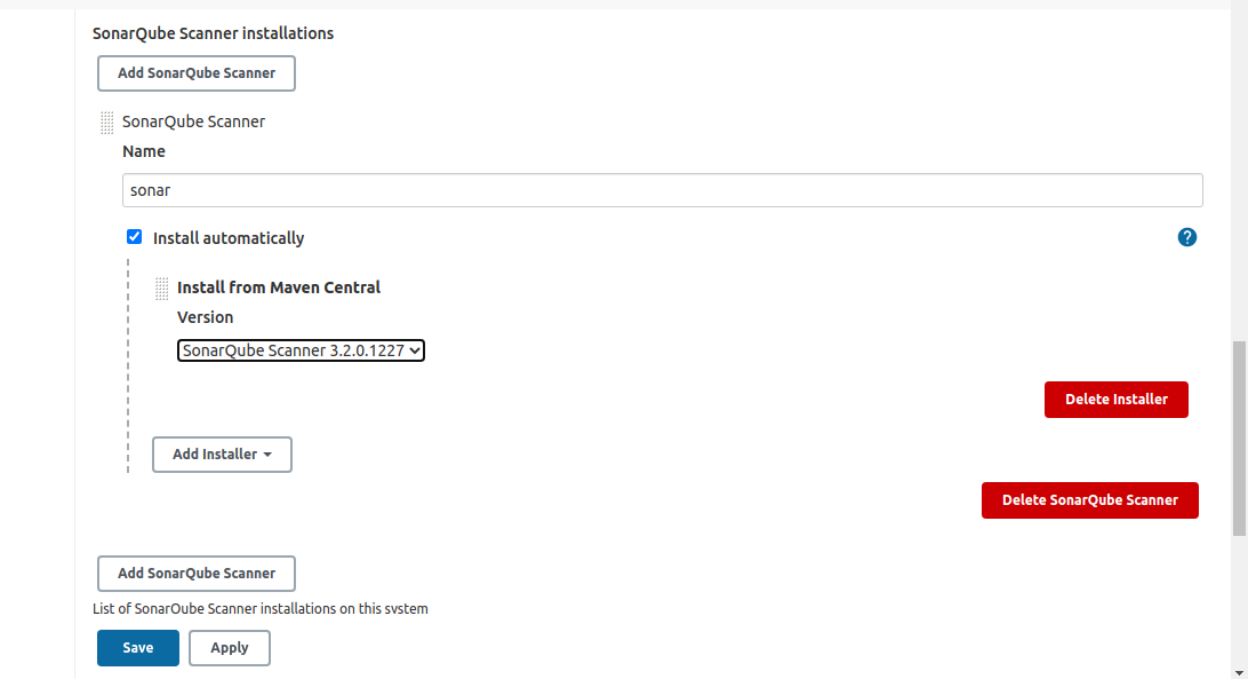
Click Available → SonarQube and Select **Install without restart**





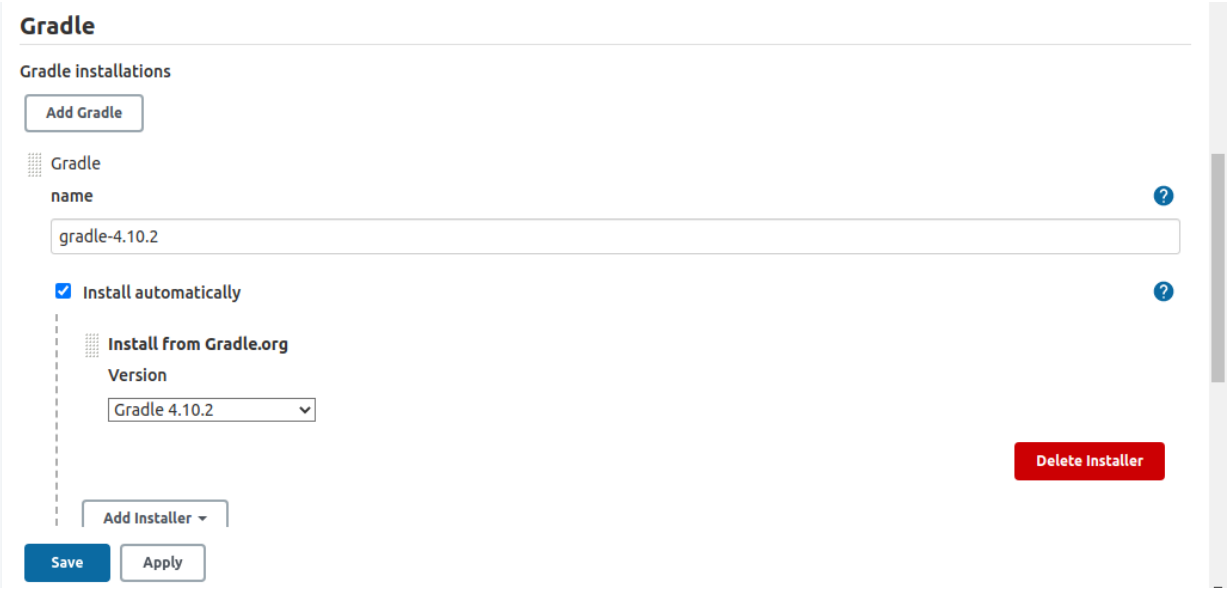
Step 2

Global Tool Configuration → Name as "sonar" and values as SonarQube Scanner 3.2.0.1227

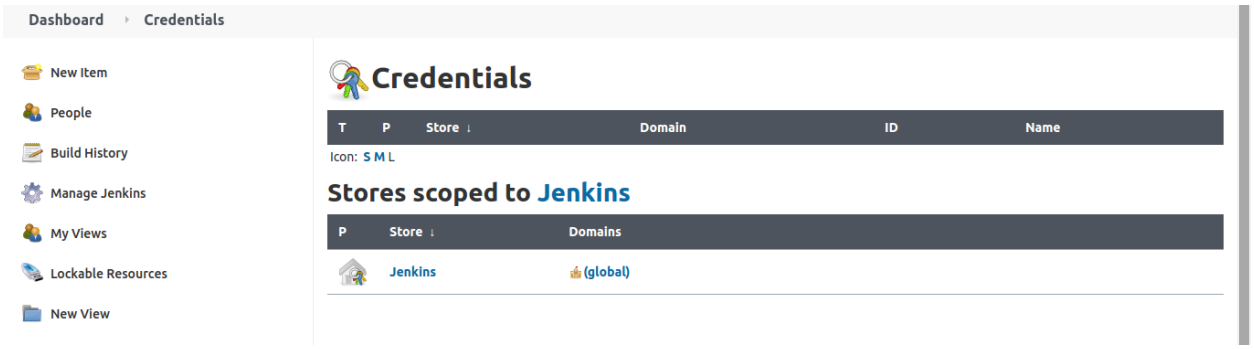


Install Gradle Plugin as well

name as gradle-4.10.2 values as gradle-4.10.2



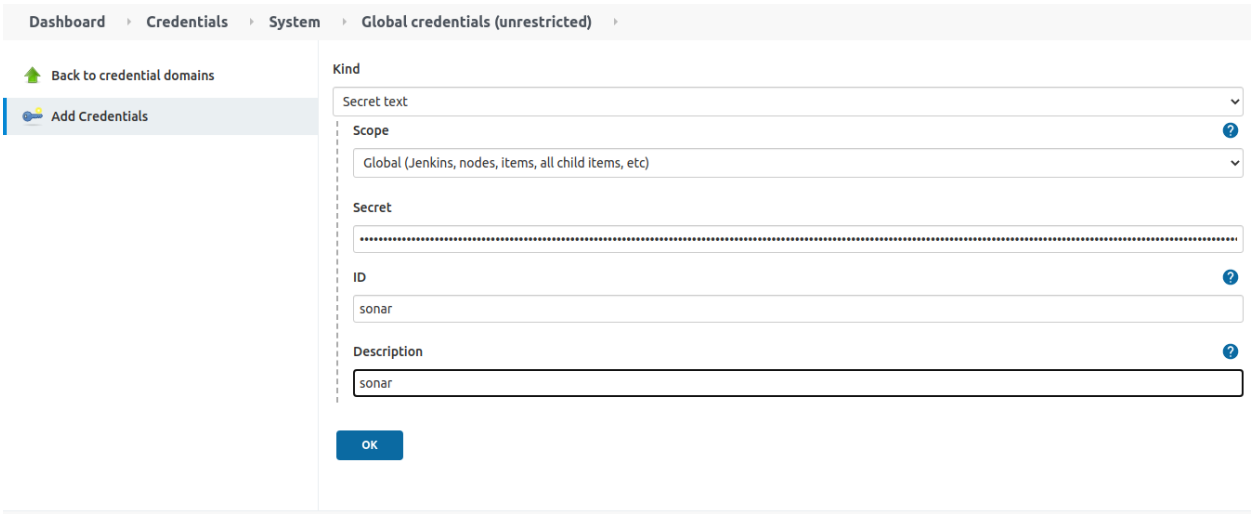
Step 3

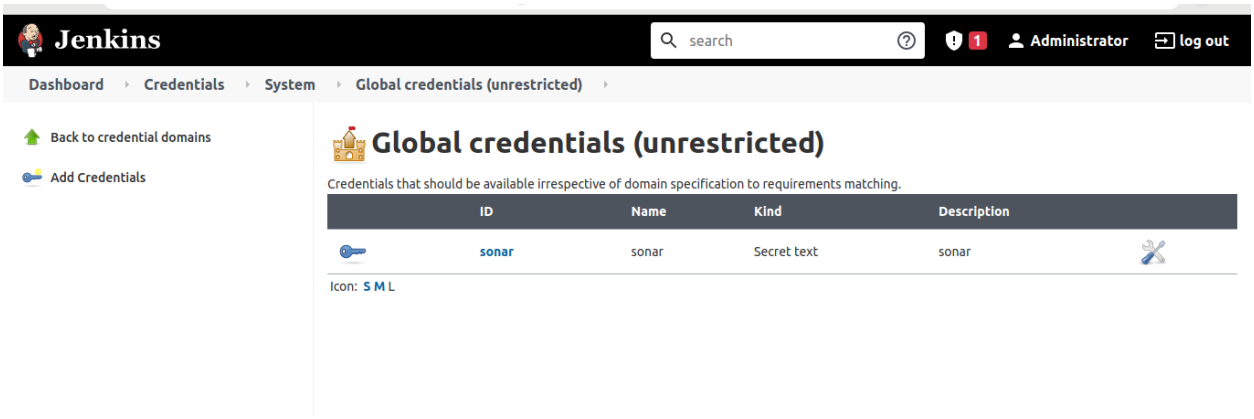


Configure SonarQube Credentials

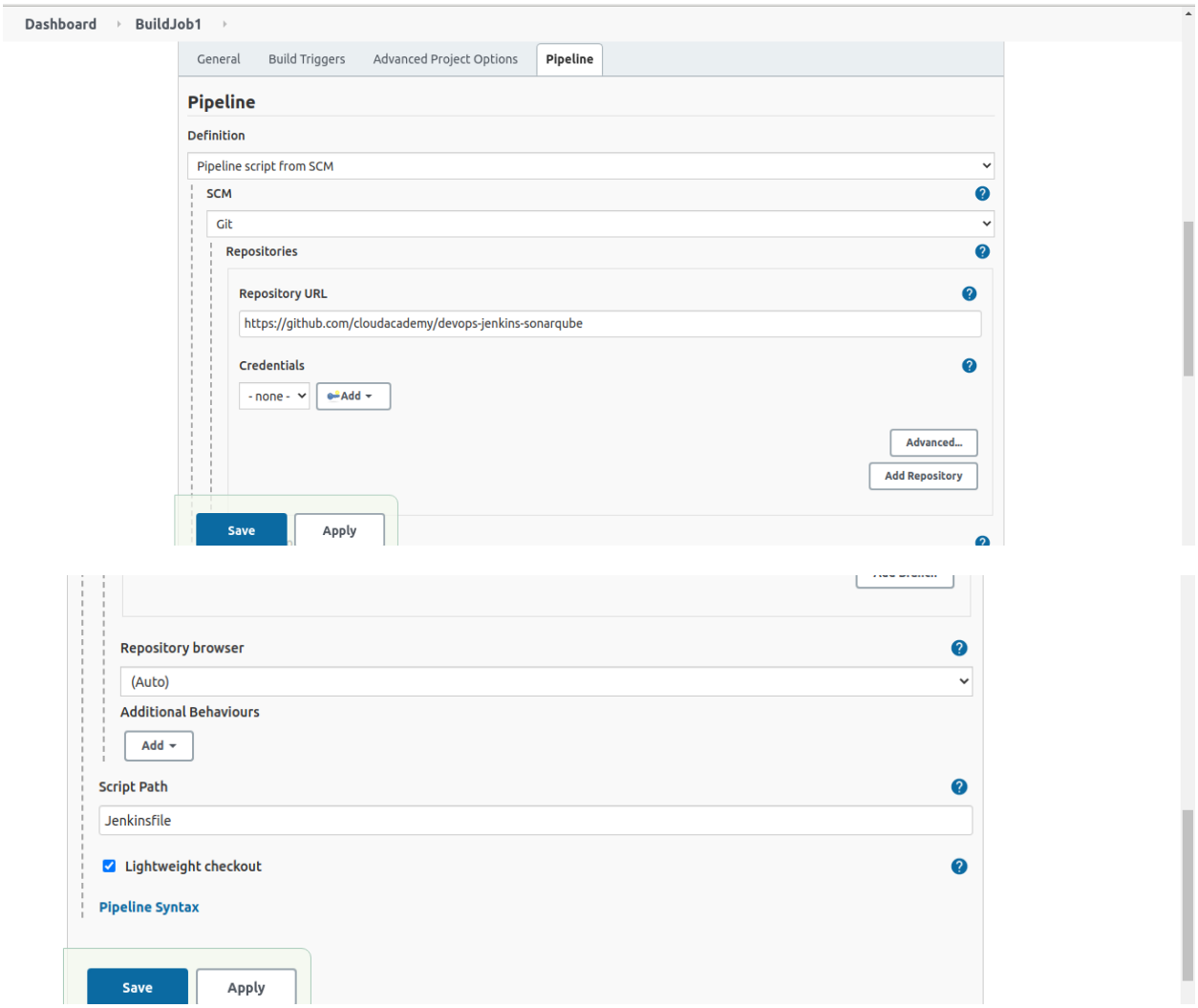
Click Credentials ⇒ Click global → Click Credentials ⇒

Set Kind to Sonar , Scope as Global

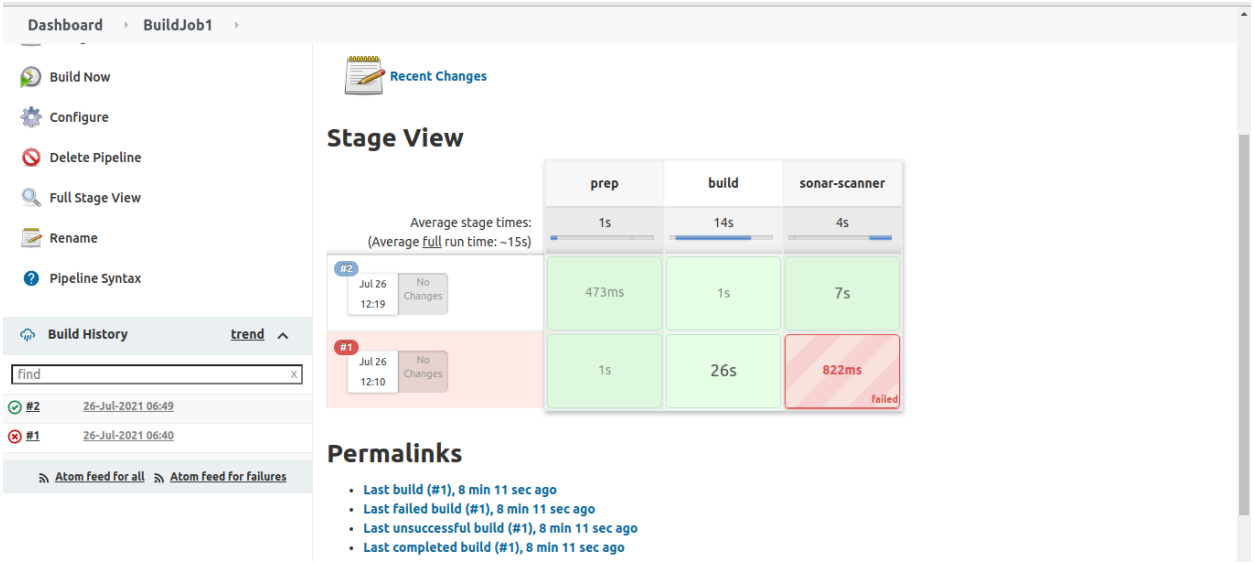




Create and Execute Jenkins Gradle pipeline Job

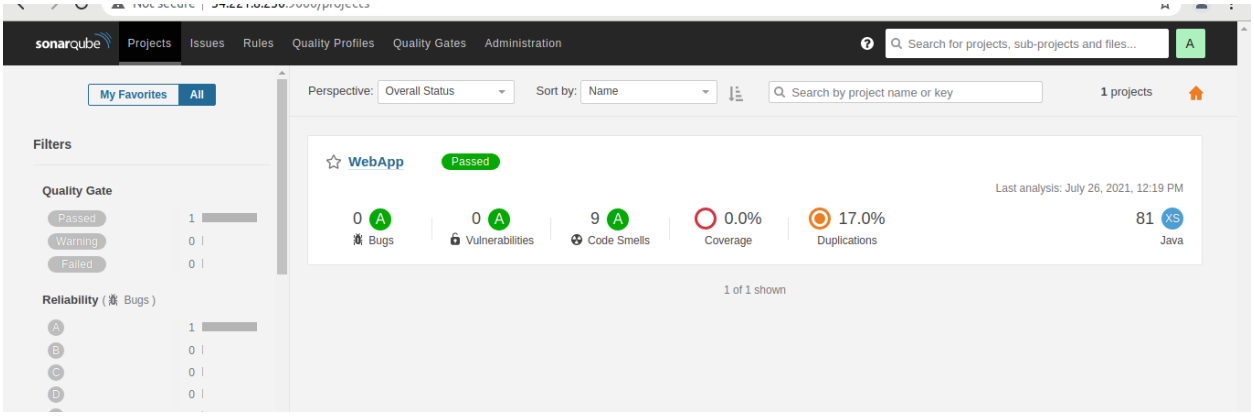


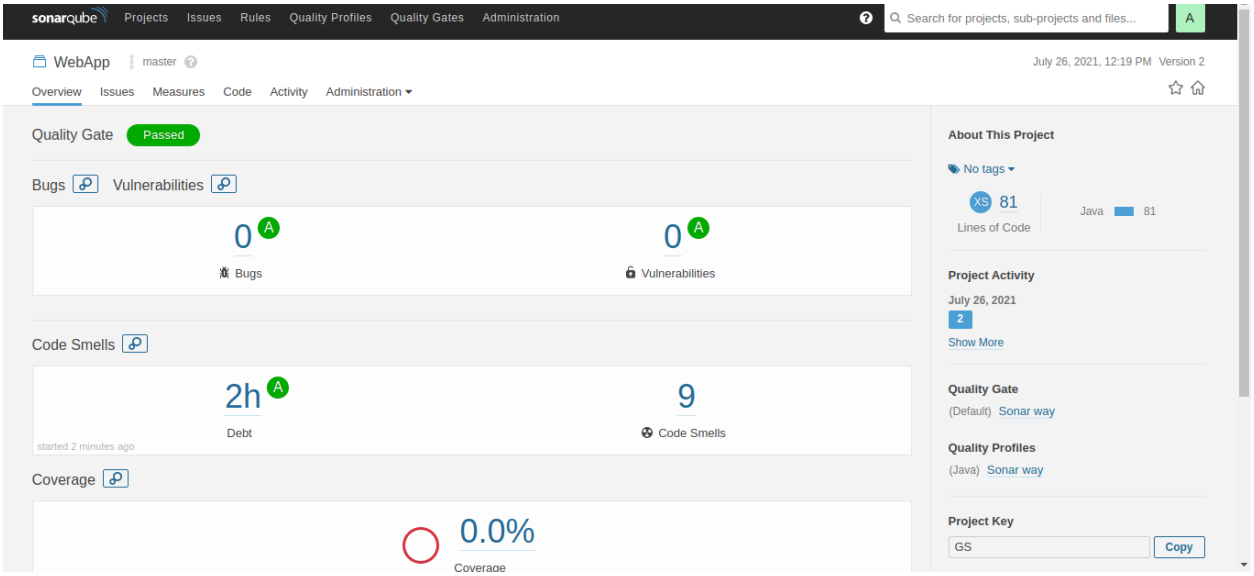
Run the Pipeline Jpb



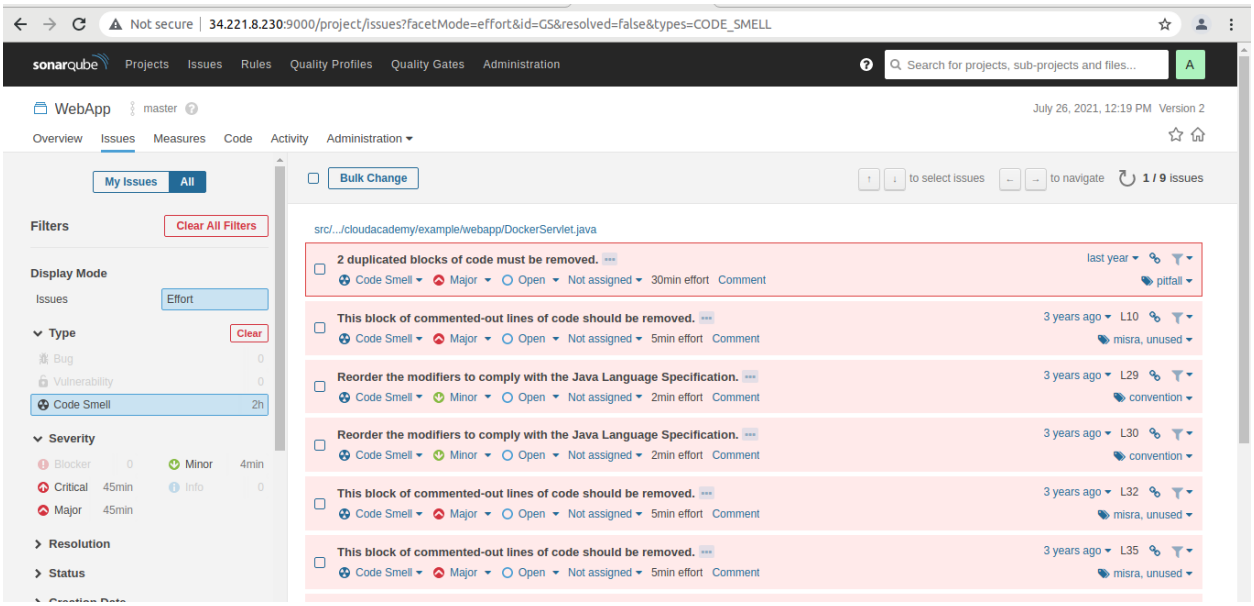
Go and Check in Sonar Cube

we can see the WebApp Project in Sonar Cude and its metrics are displayed as below

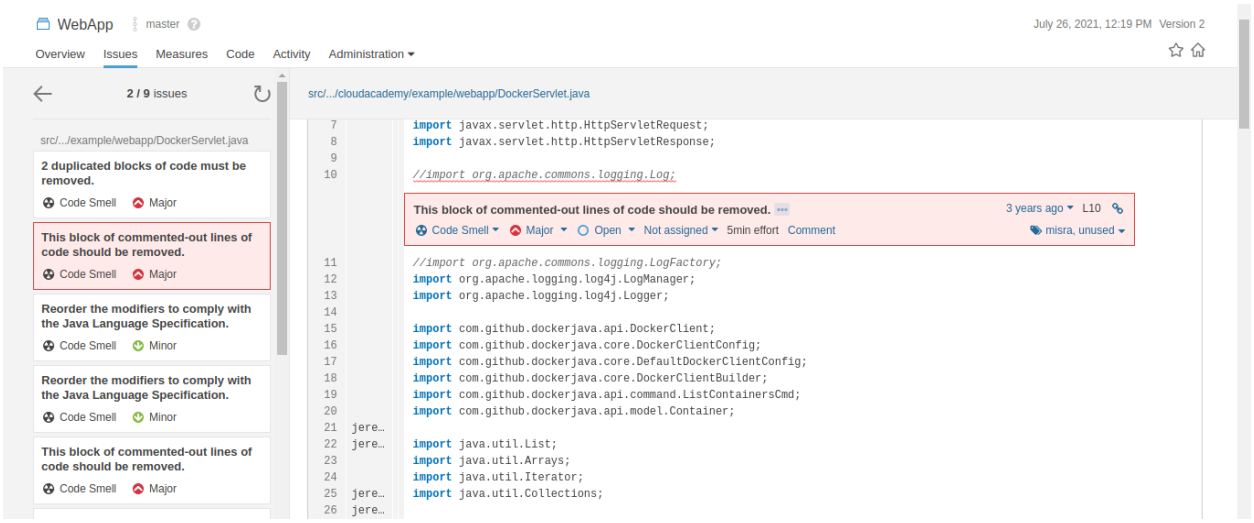




Click on the code-smells it will take you to the issues area



Click the Red boxes to see the issues



Work on the Code and update the Git Branch

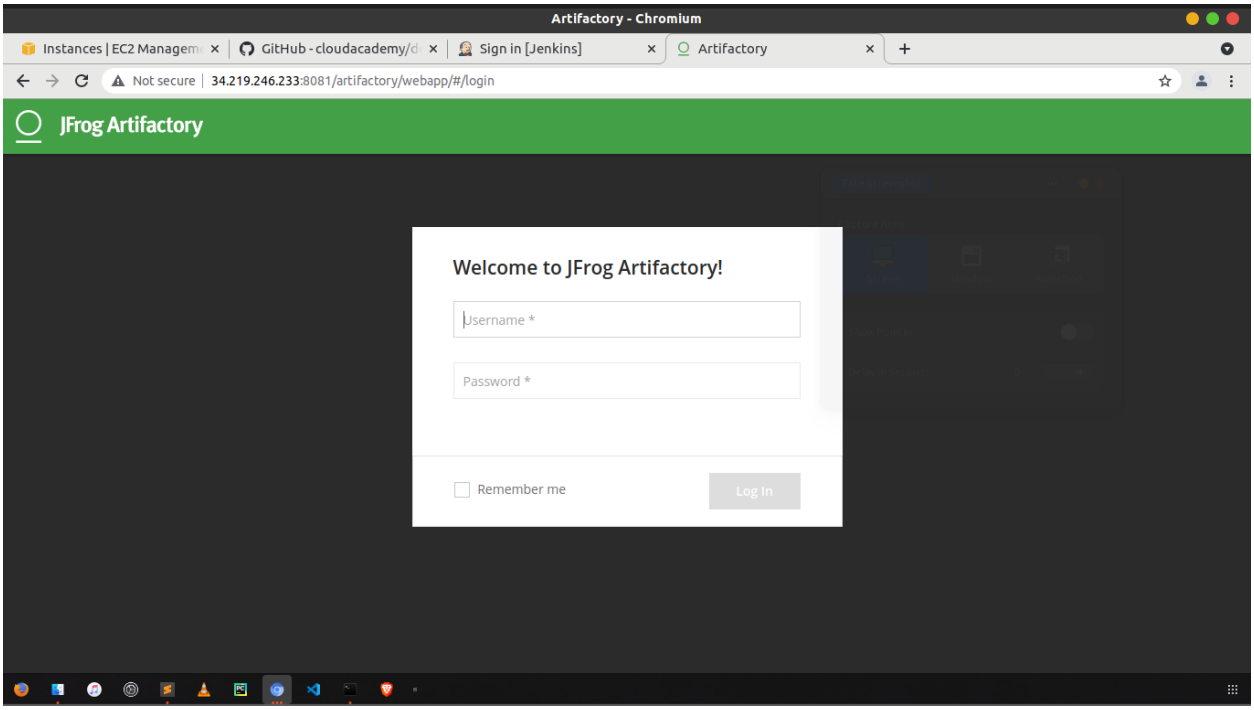
Jenkins Pipeline with Artifactory Integration

Run the below Docker-compose file

```
node{

    def server = Artifactory.server 'artifactory'

    def rtGradle = Artifactory.newGradleBuild()

    withCredentials([usernamePassword(
        credentialsId: 'artifactory',
        usernameVariable: 'USERNAME',
        passwordVariable: 'PASSWORD'
    )]) {
        server.username = "${USERNAME}"
        server.password = "${PASSWORD}"
    }

    def builfInfo

    stage('clone'){
        git url: 'https://github.com/cloudacademy/devops-webapp.git'
    }

    stage('Artifactory Config'){
        rtGradle.tool = "gradle-4.10.2"
        rtGradle.deployer repo:'gradle-release-local', server: server
        rtGradle.resolver repo:'jcenter', server: server

    }

    stage('build'){

        rtGradle.run rootDir: "./", buildFile: 'build.gradle', tasks: 'clean build'

    }

    stage("publish"){
        buildInfo = rtGradle.run rootDir: "./", buildFile: 'build.gradle', tasks: 'artifactoryPublish'
    }

    stage("deploy"){
```

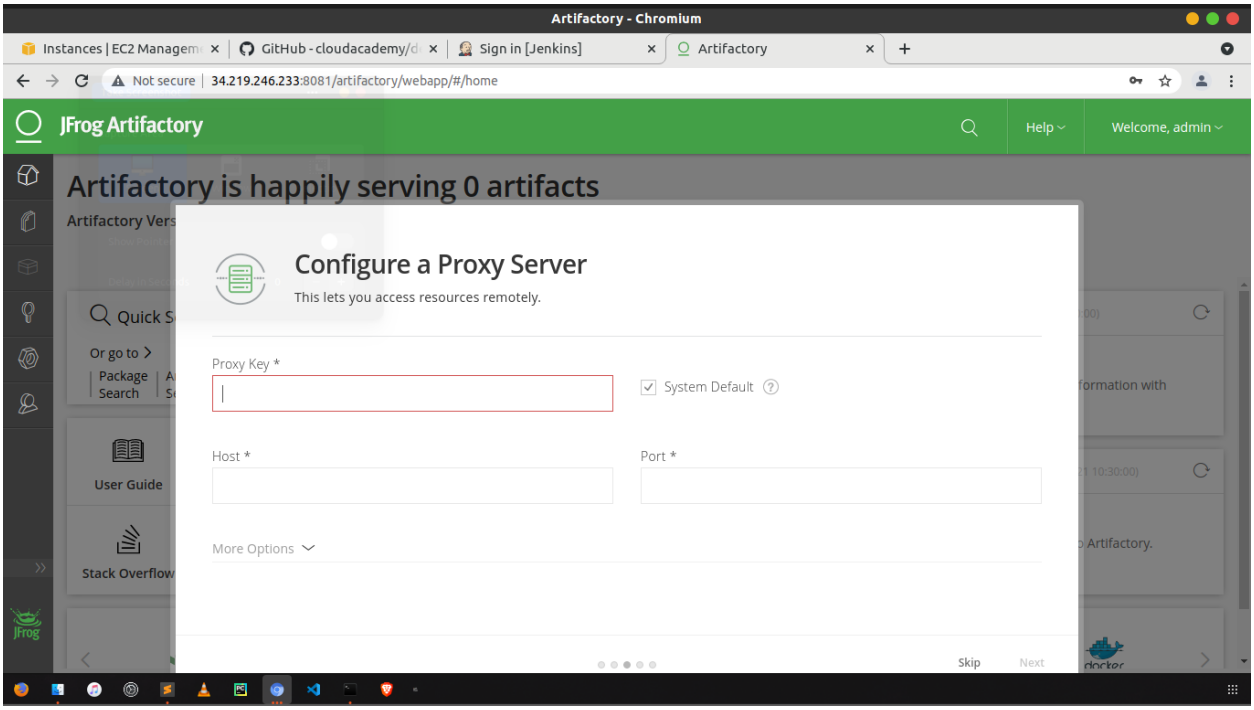
```
        server.publishBuildInfo buildInfo
    }
}
```

Check artifactory is up and running

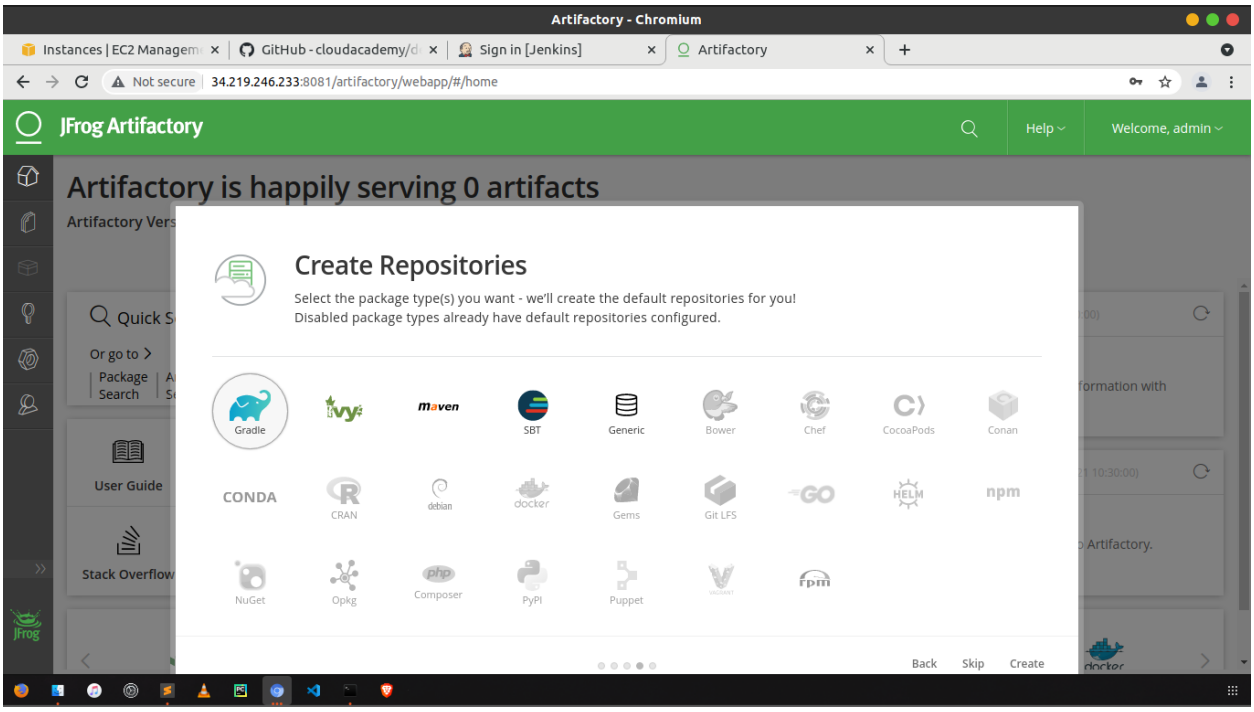


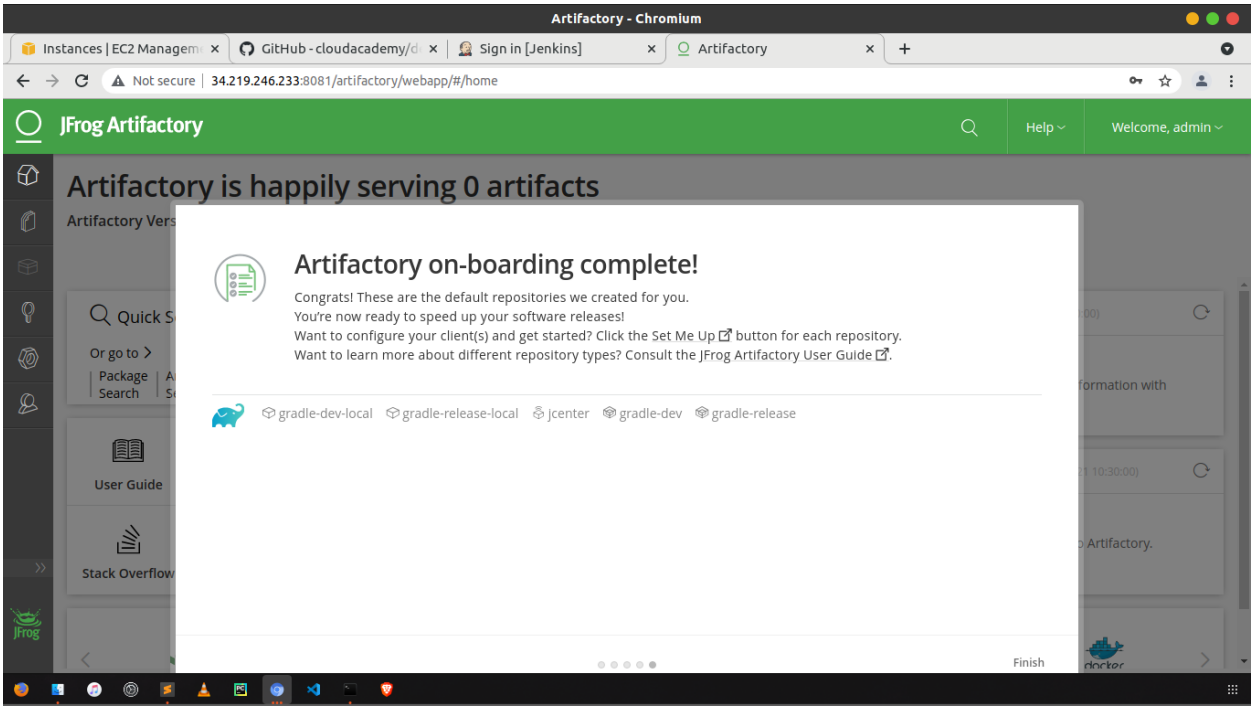
Login to Artifactory with admin and password. Once logged in changed the password to qwerty123

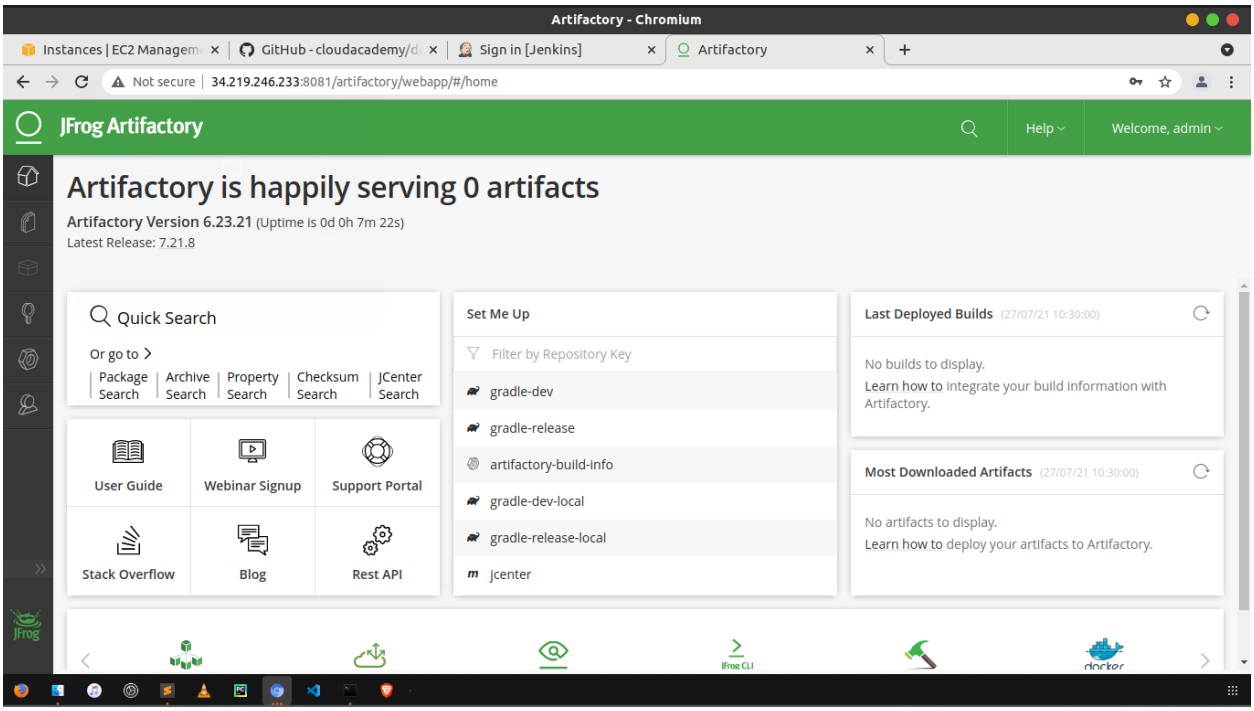
DOnt Change anything in the below , use default settings



Now we are going to generate Gradle repositories

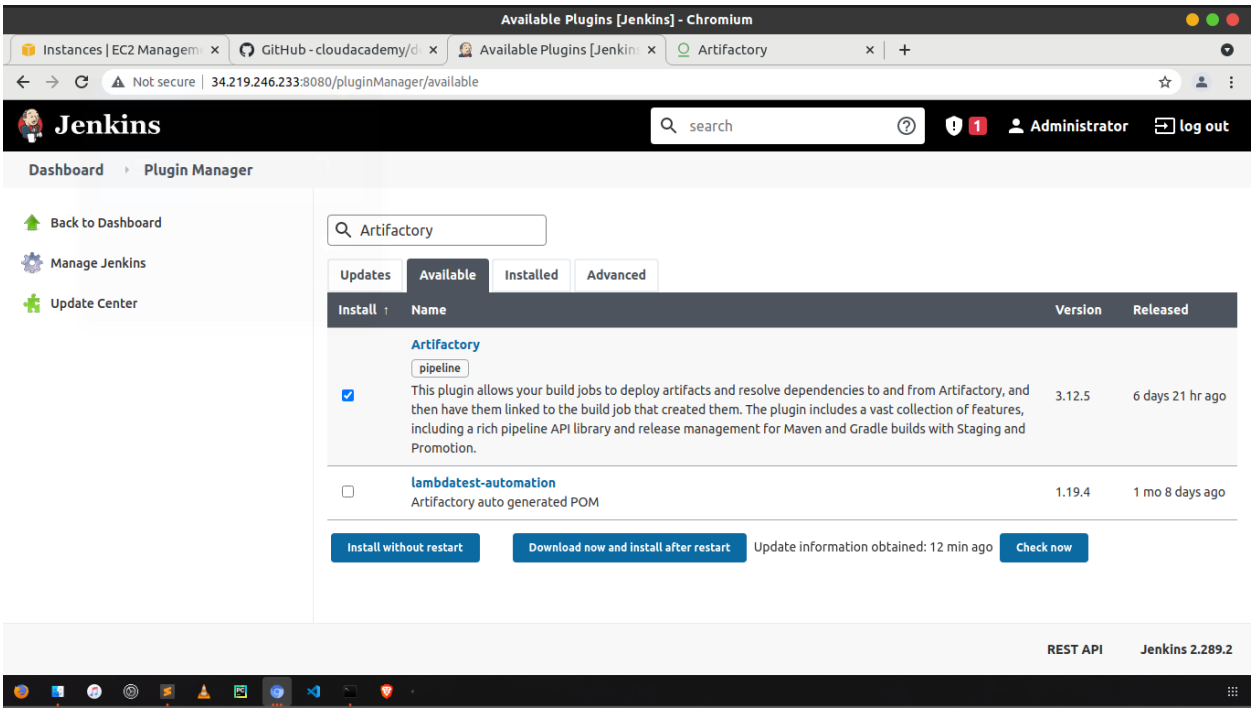


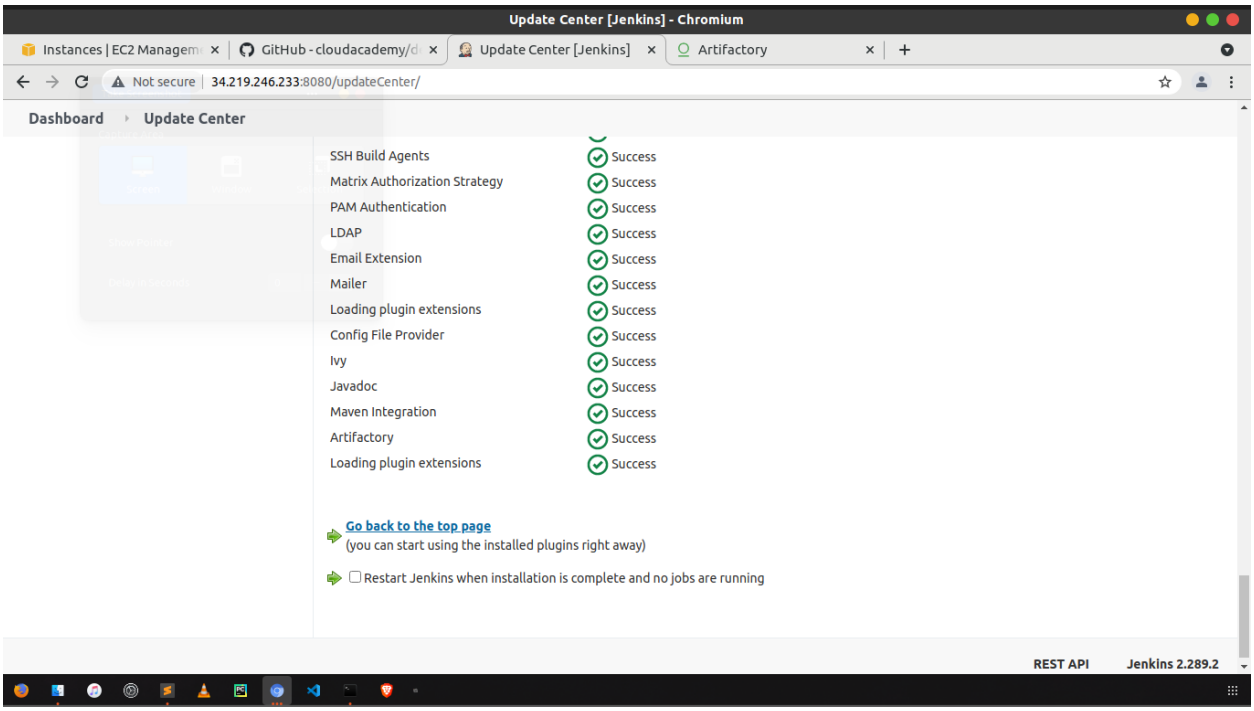




Setting up Gradle and Artifactory plugin

`Manage Jenkins ⇒ Manage Plugins ⇒ Go to Available ⇒ Search for` **`artifactory`**

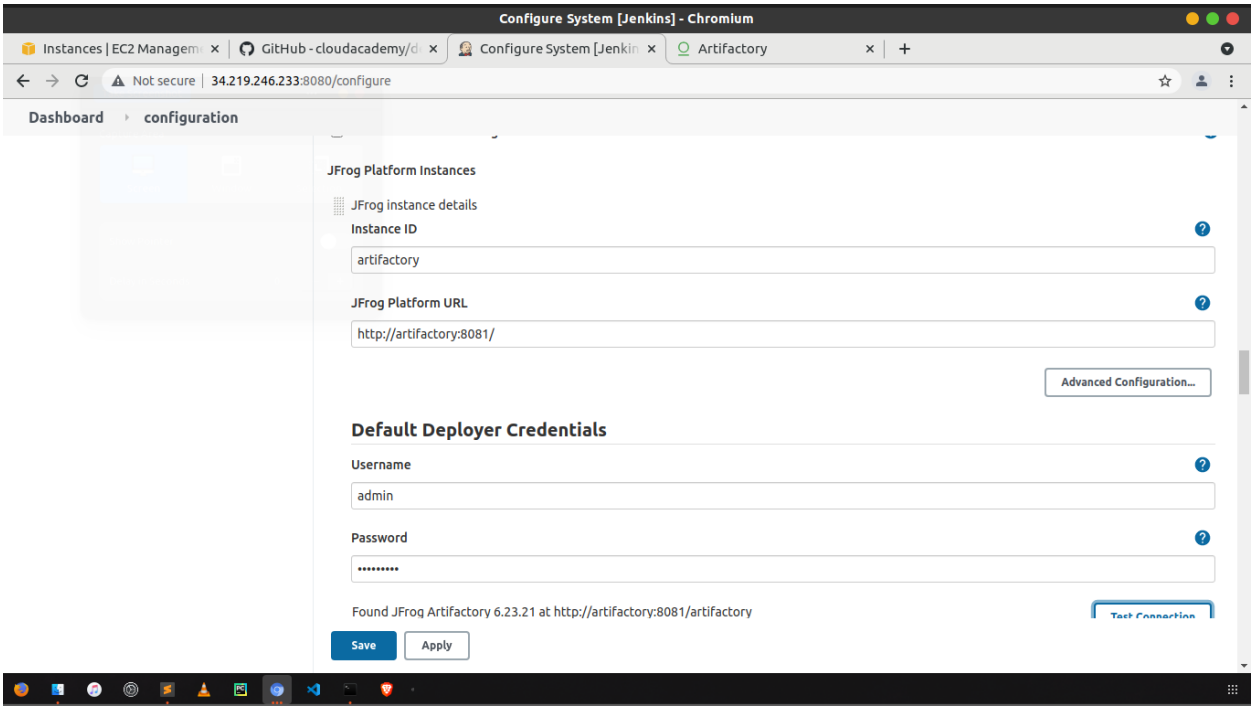




Lets Configure the Artifactory

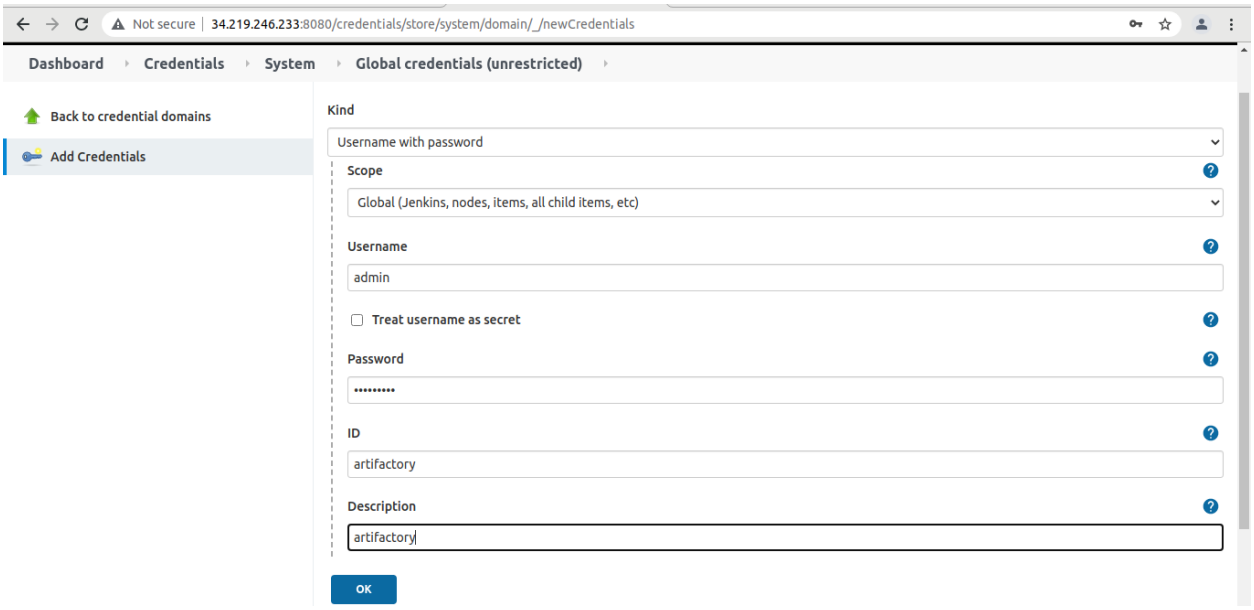
Click the Manage jenkins menu item ⇒ configure system

Provide the required details and test the connection



Lets Store the artifactory credentials

go to Manage Credentials

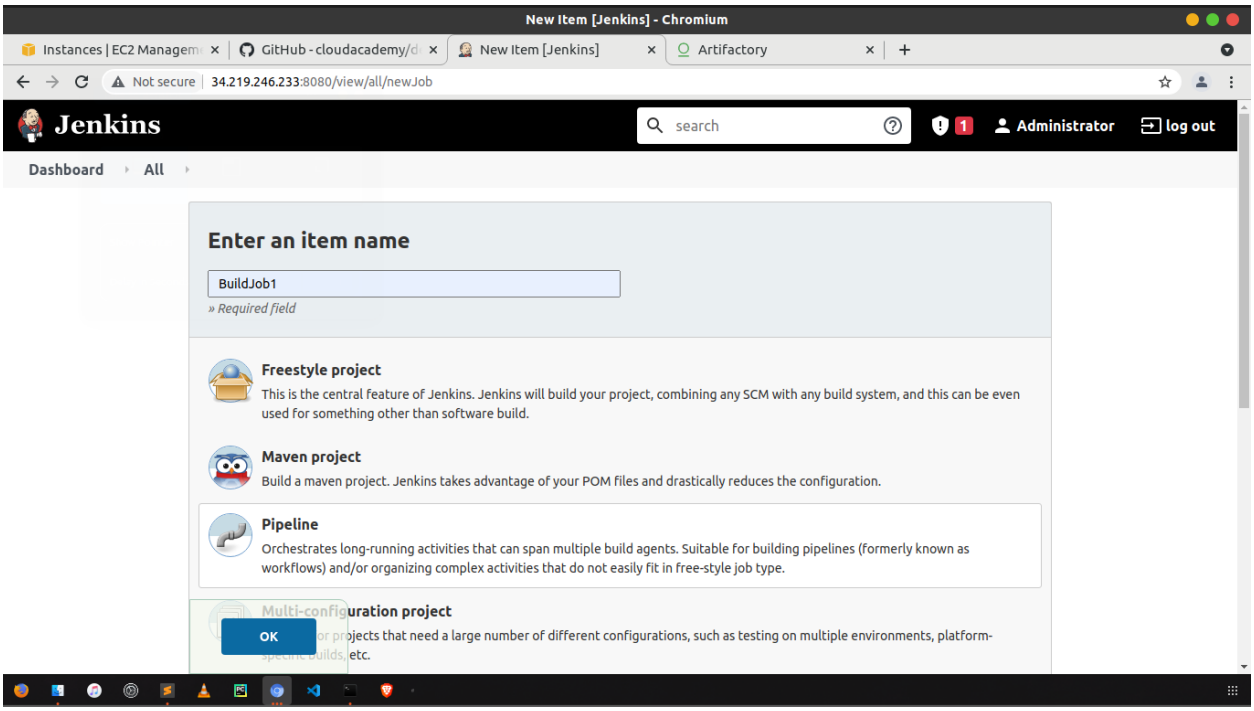


Add Gradle Plugin

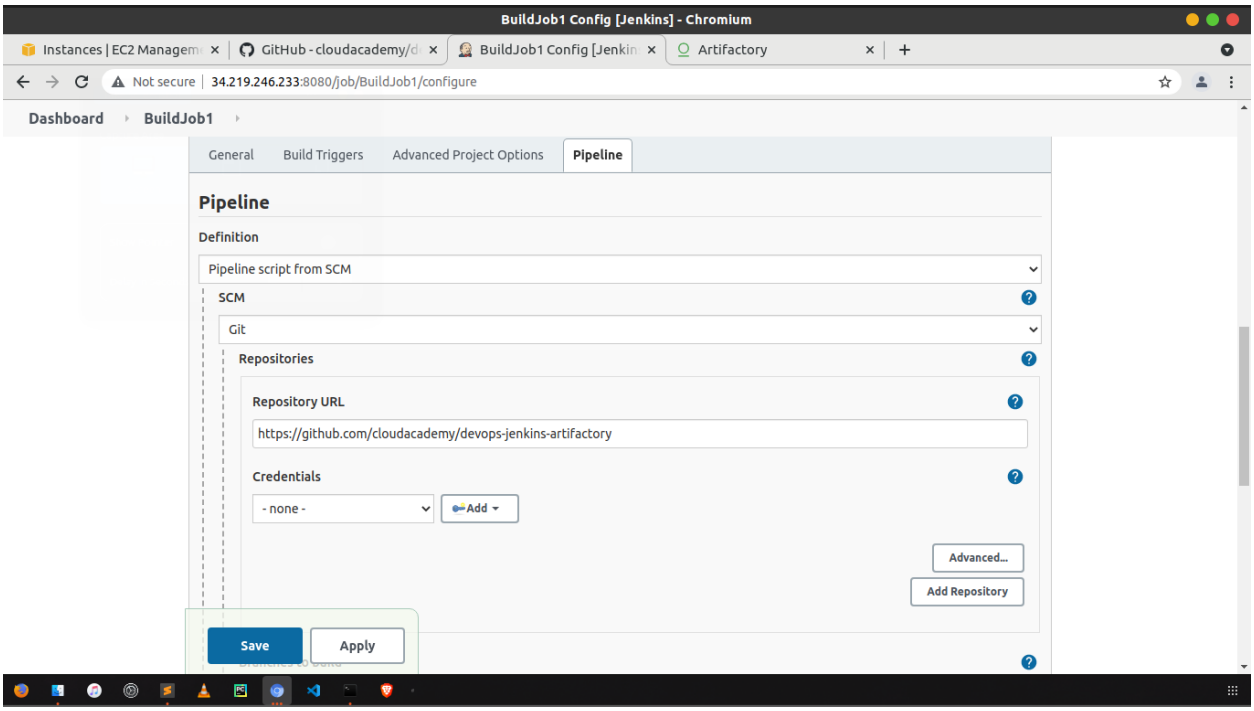
Go to Global Tool Configuration and add gradle 5.6.4

Lets Build a Jenkins Pipeline For Gradle Project

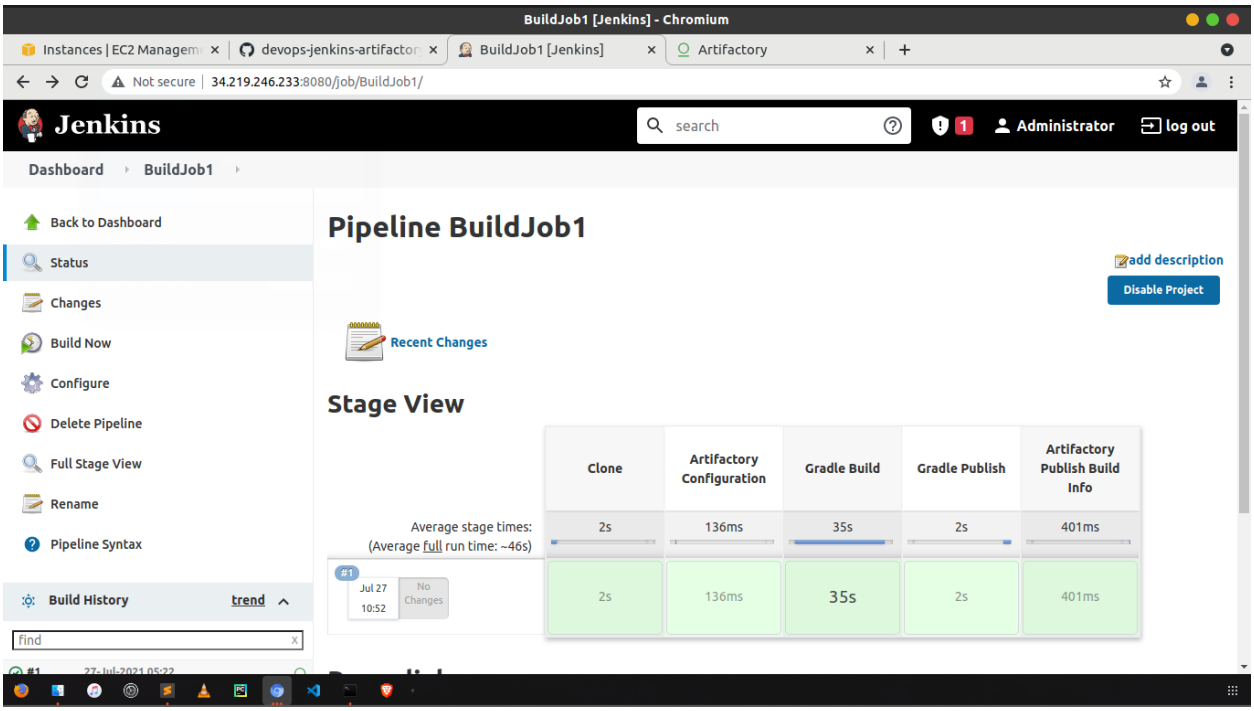
Create a new project as pipeline Project



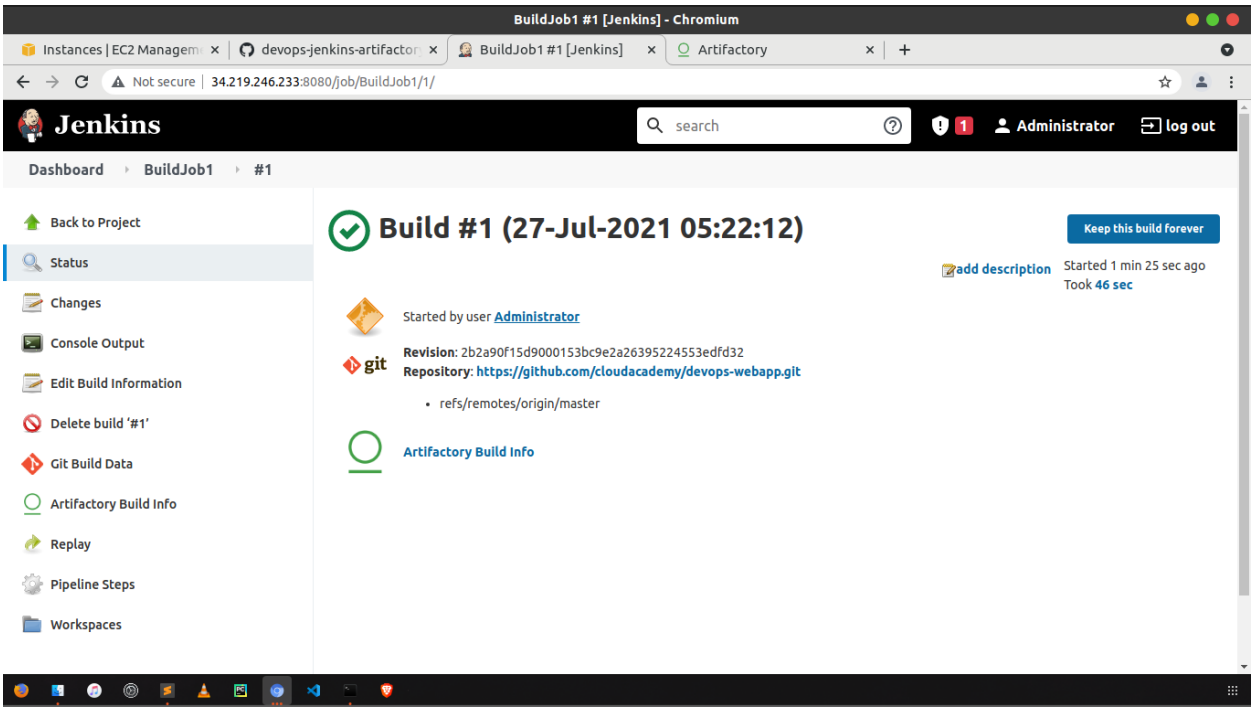
Build Project from GIT SCM



Trigger the Build



Build completed without any issues



Check the Build status

Click on the artifactory link

Notice how there are *Artifactory Build Info* menu links that will take you directly into the build information registered within Artifactory. These links currently use the internal Artifactory

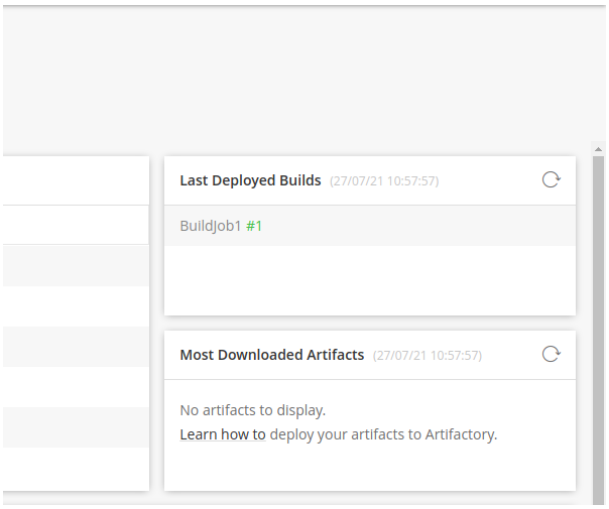
docker host name within the URL, so it will not work from your browser. To get these links to work, replace the internal Artifactory docker host name with the Public IP address that is assgined to your *cicd.platform.instance* EC2 instance. For example:

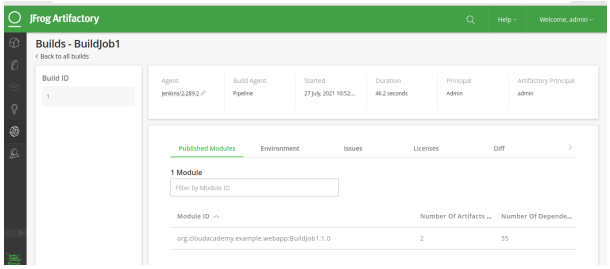
`http://artifactory:8081/artifactory/webapp/builds/BuildJob1/1`

becomes

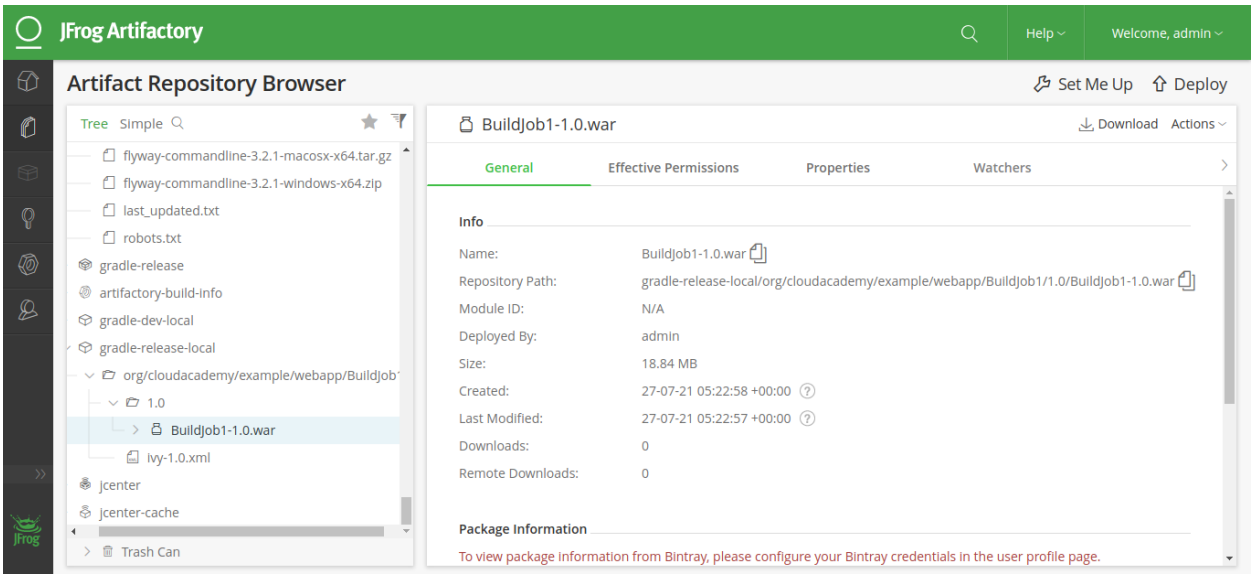
`http://54.149.60.36:8081/artifactory/webapp/builds/BuildJob1/1`

if its not opening go to the artifactory itself and check for it

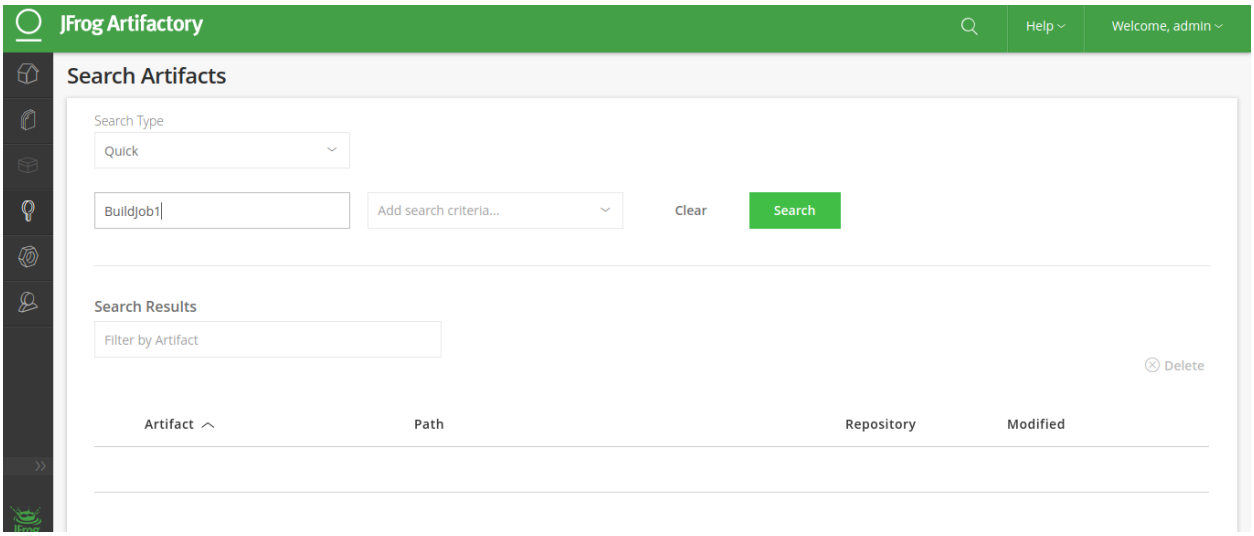


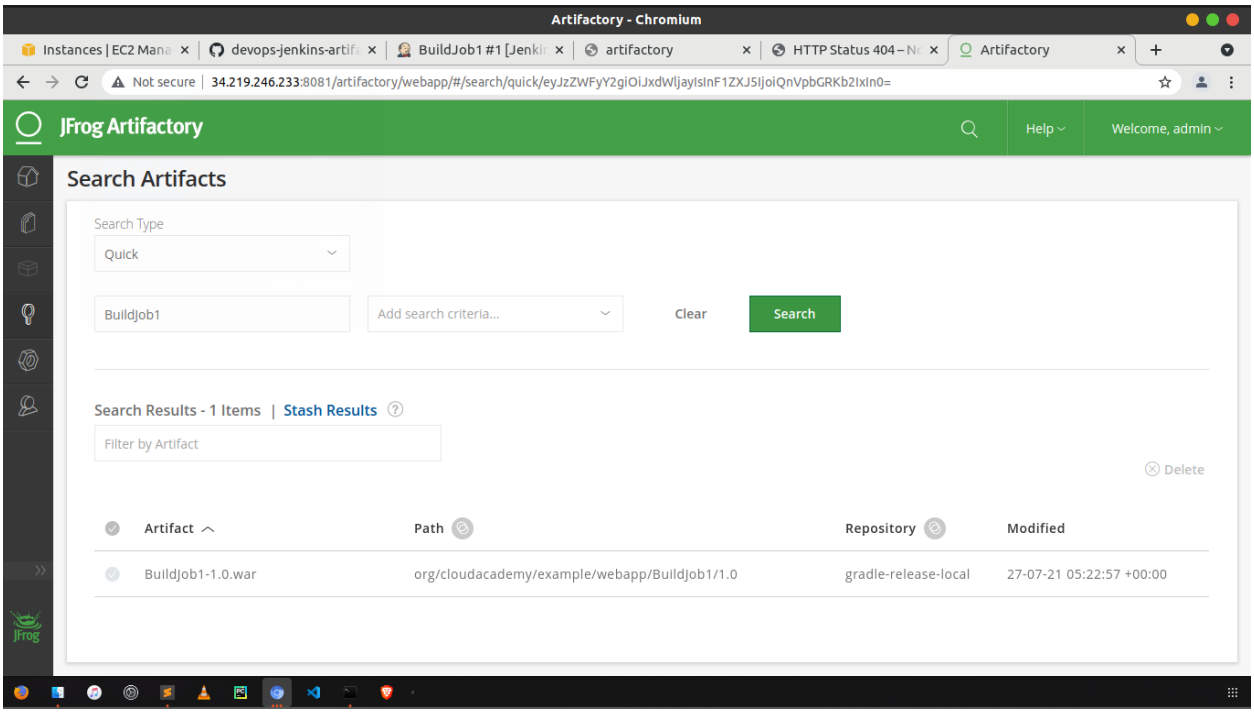


We can view the war File in the gradle-release-local repo as below



Lets search the Artifacts





https://github.com/cloudacademy/devops-jenkins-artifactory/blob/master/Jenkinsfile

Scripted Pipeline Syntax

To upload or download files you first need to create a spec which is a JSON file that specifies which files should be uploaded or downloaded and the target path.

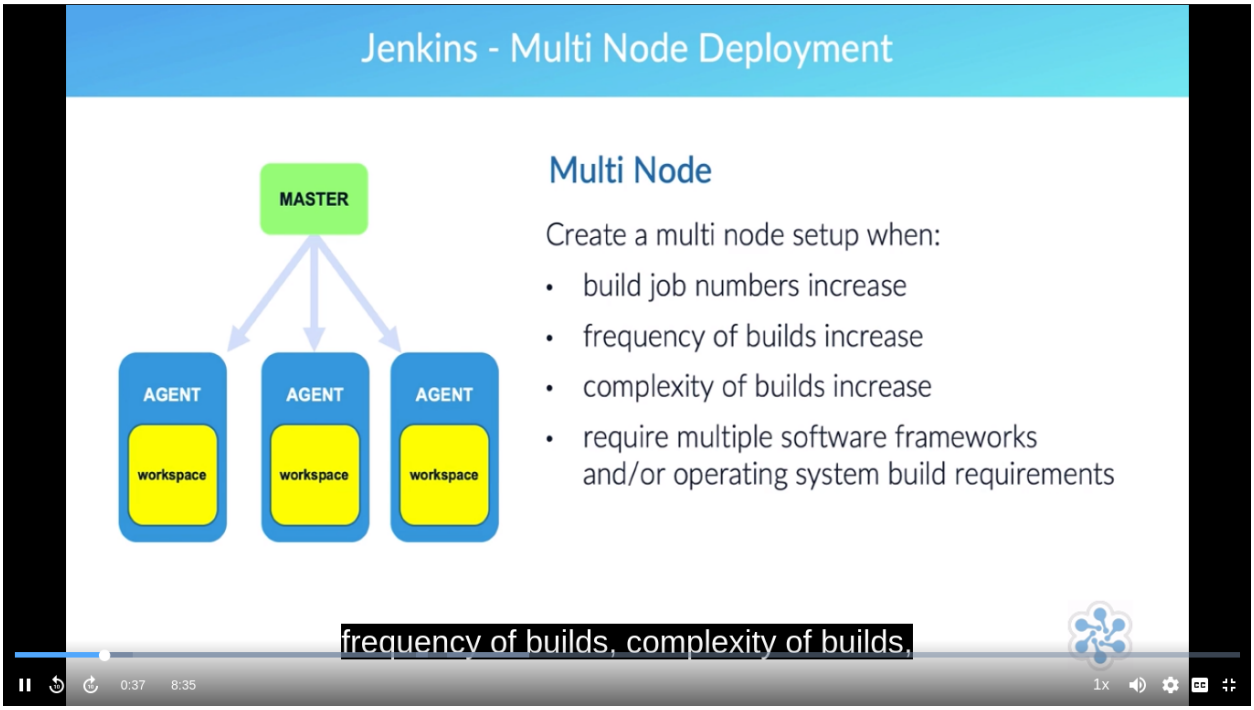
https://www.jfrog.com/confluence/display/JFROG/Scripted+Pipeline+Syntax

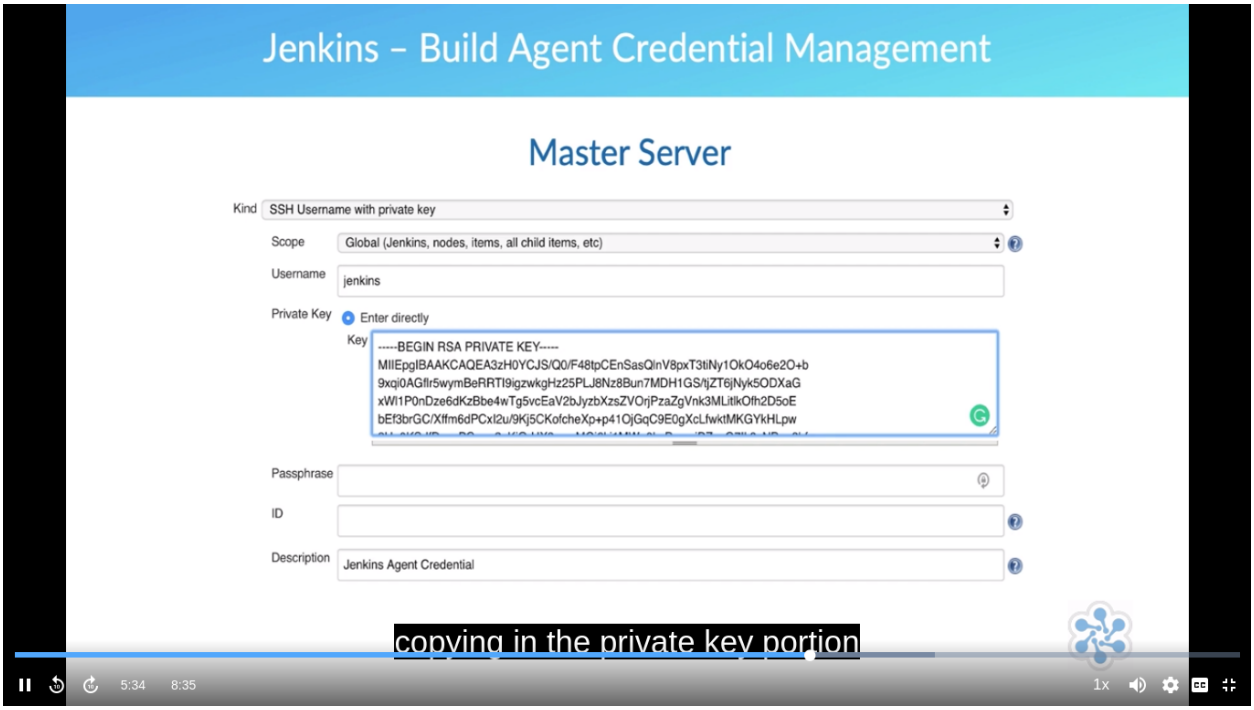
Jenkins Multi Node

Jenkins master can be configured to distribute build jobs amongst multiple jenkins build agents

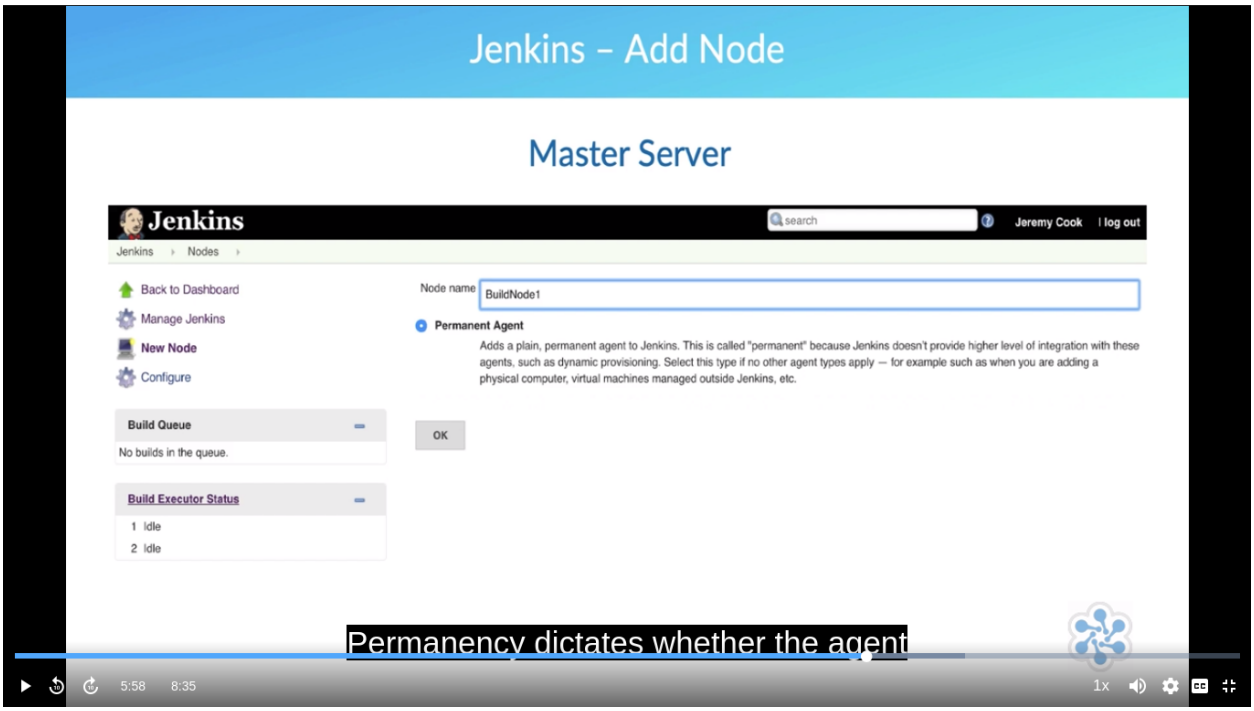
Requirement needed when you dont want to compromise the security of the master server.

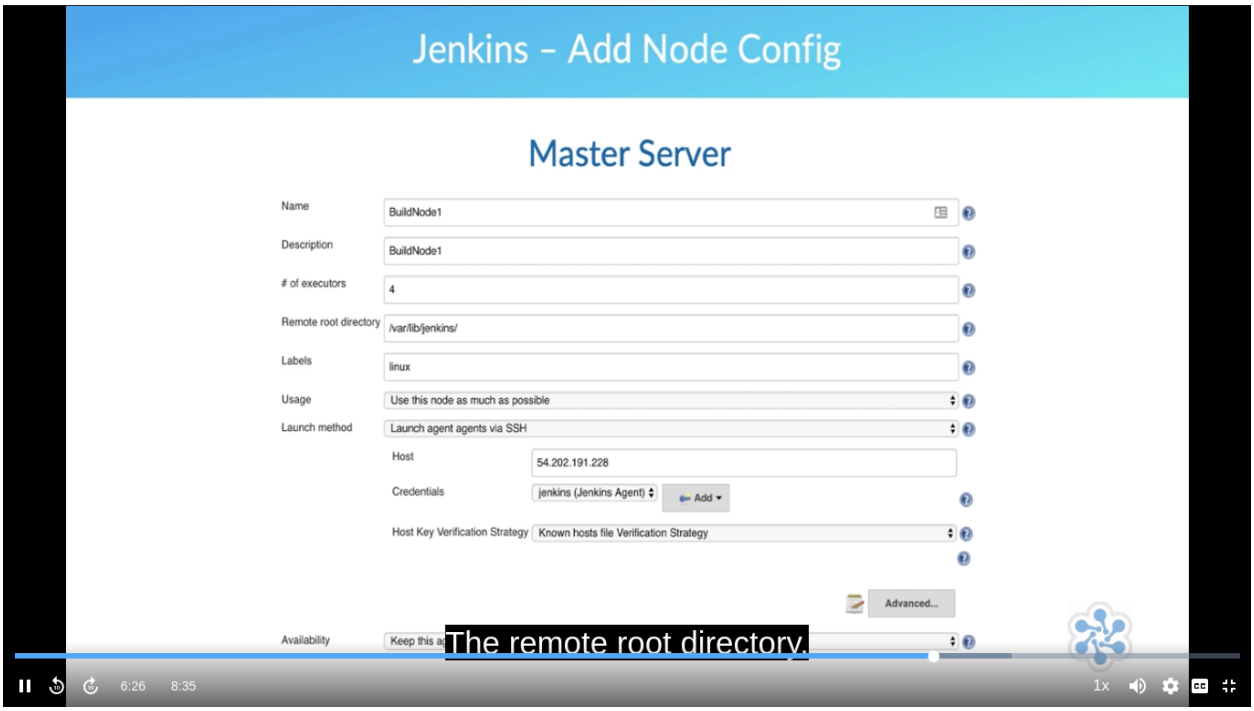
Connection between master and agents are based on the tCP/IP network connections





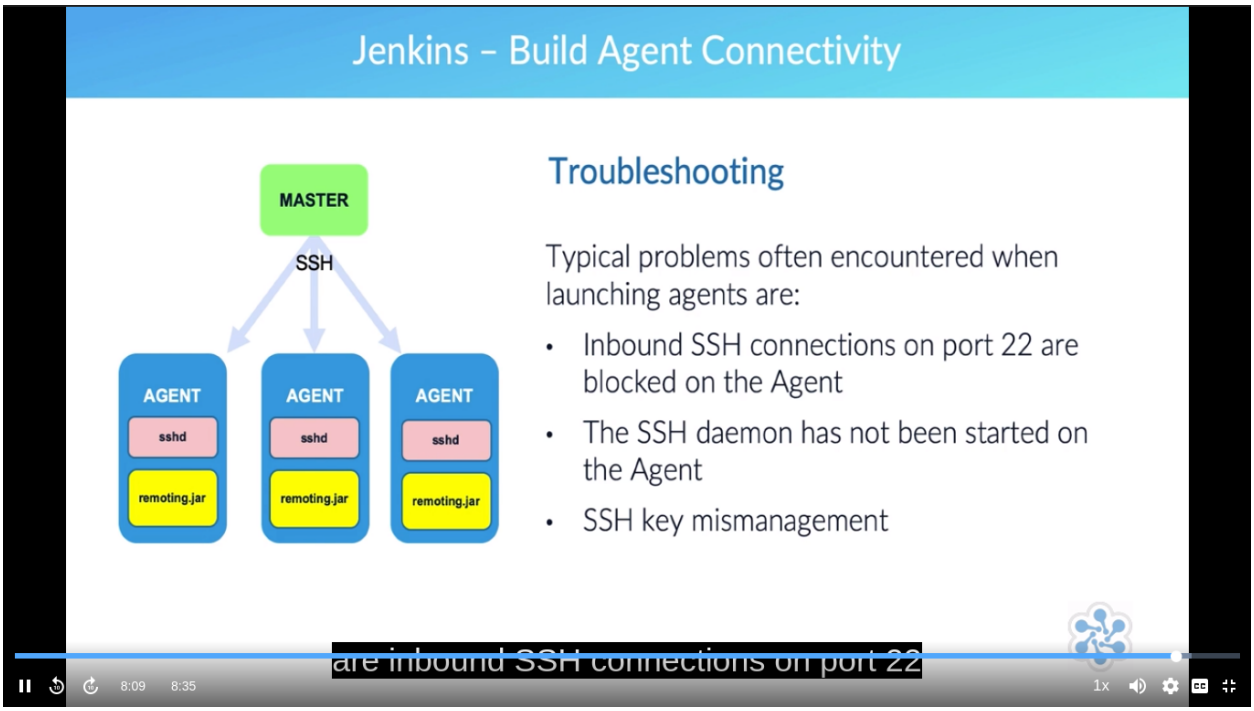
Lets Create a New Node





View the Dashboard

Typical Master slave Jenkins



Jenkins Pipeline

it contains all of our pipeline information

can be treated as other coding Files , it can be versioned as well.

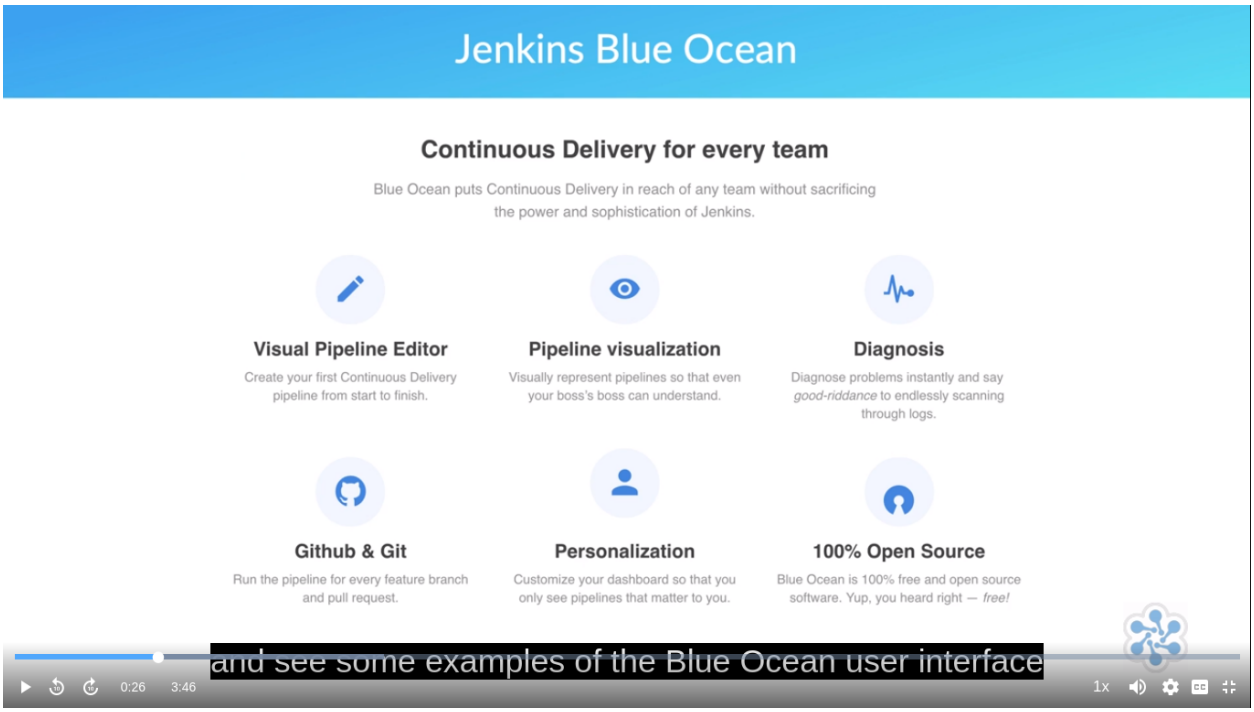
this File should not have credentials stored in it. we should use internal jenkins credential store and then jenkins have them to query at the runtime.

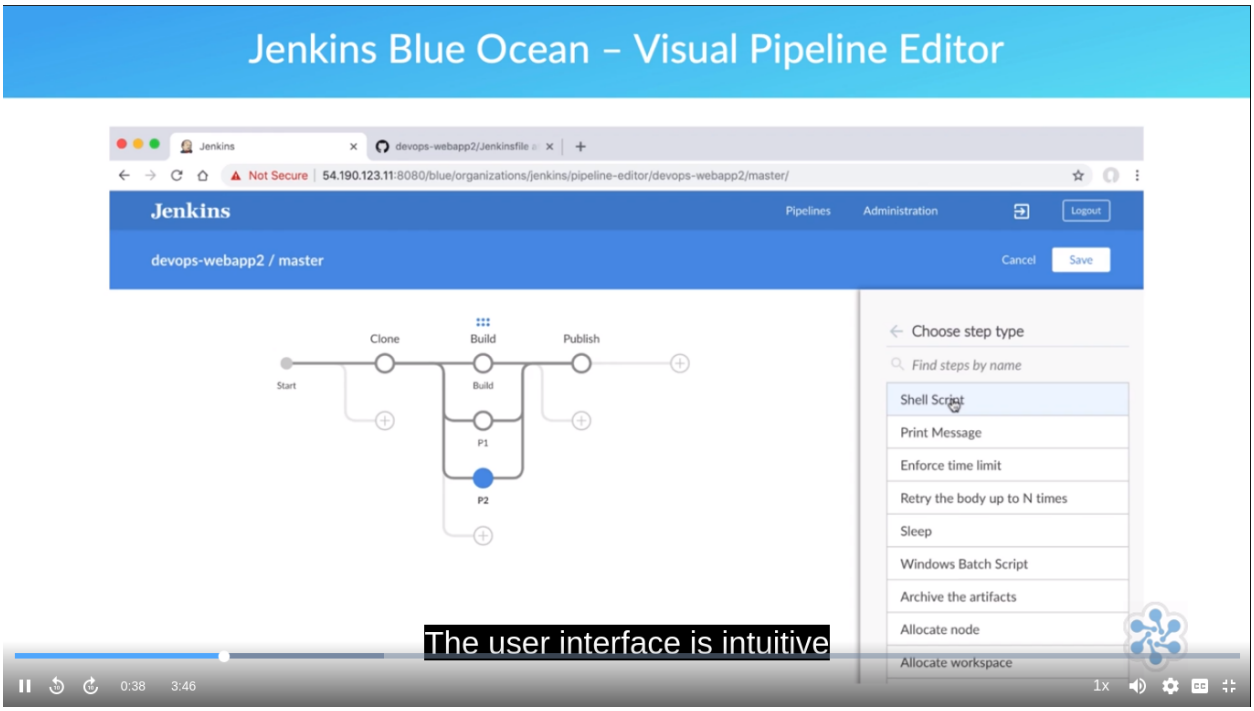
```
environment{
AWS_ACCESS_KEY_ID = credentials('key-id')
AWS_SECRET_ACCESS_KEY= credentials('secret-id')
}
```

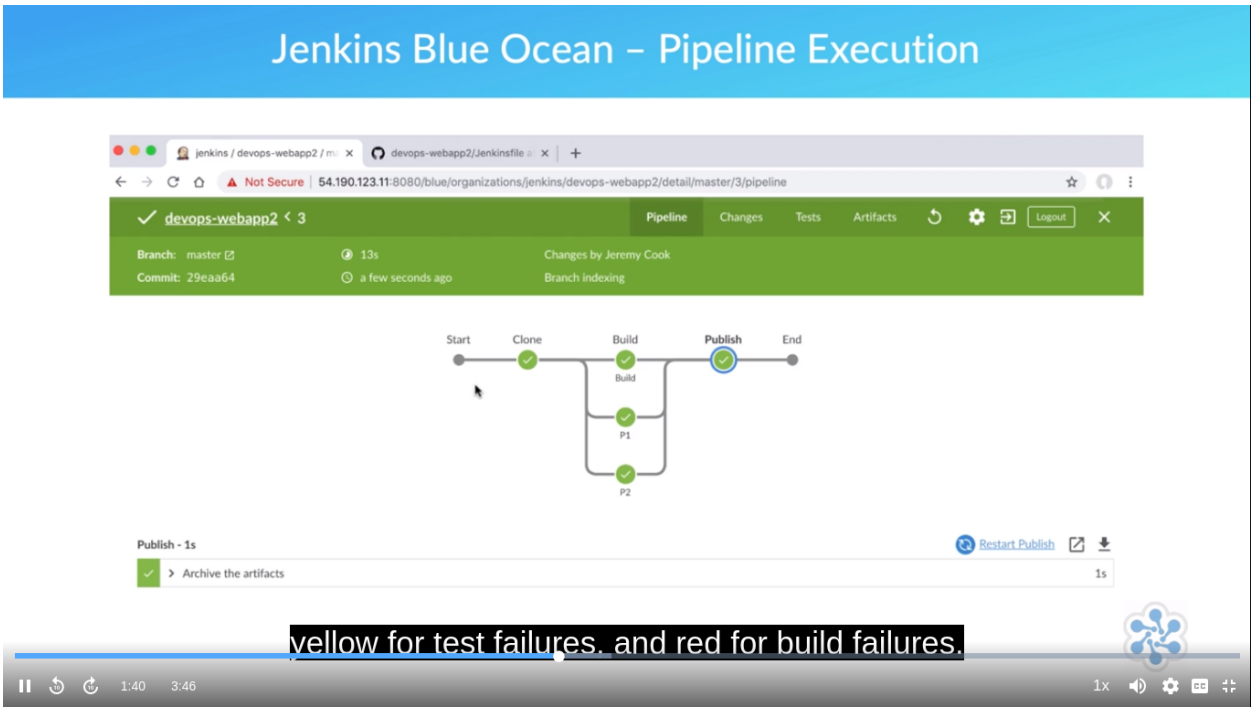
BlueOcean

blueocen makes pipeline as first class citizen when it comes to administration

blueocean interface

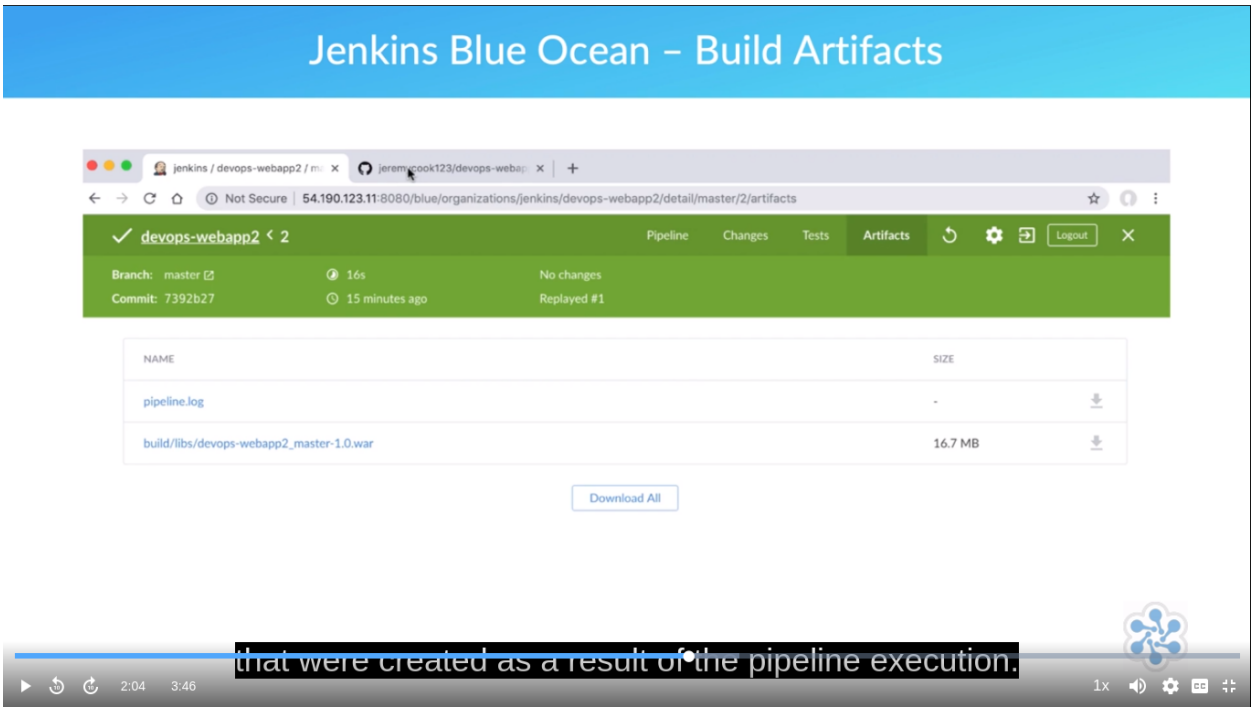






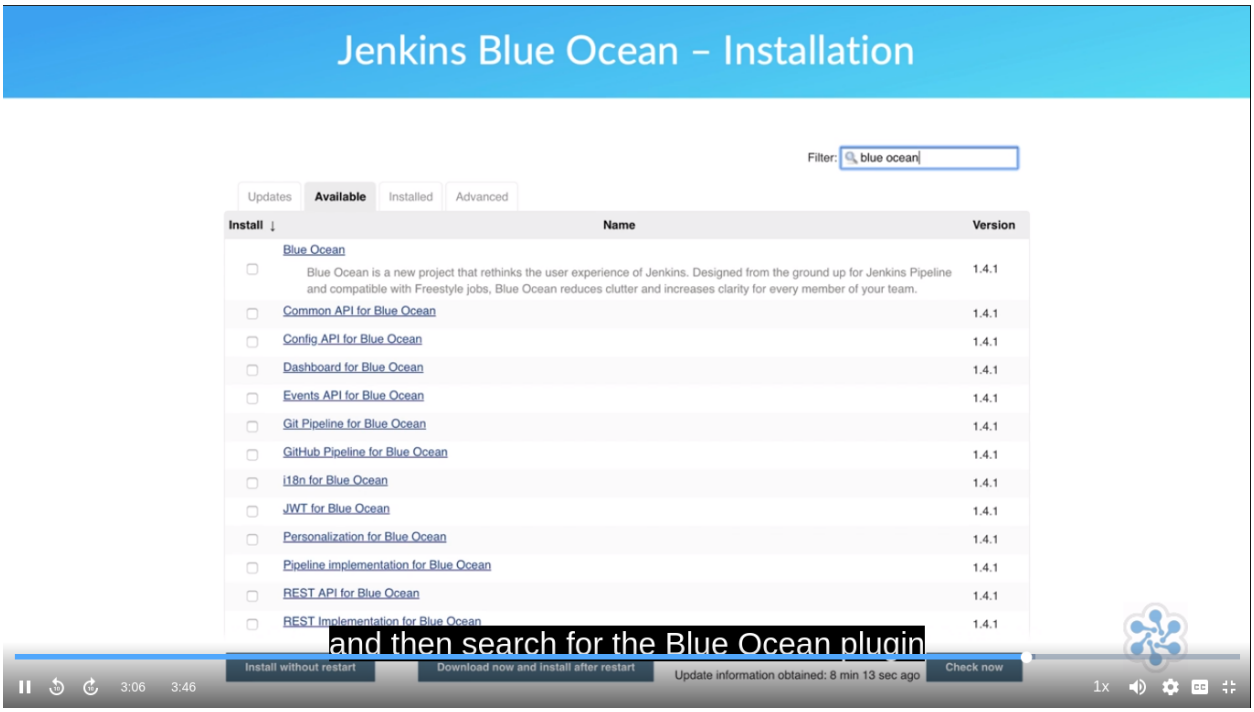
Click on the View will give the build logs

Artifact views can used to easily get the artifacts



INstall Blueocean

Blueocean is the new user experience for Jenkins

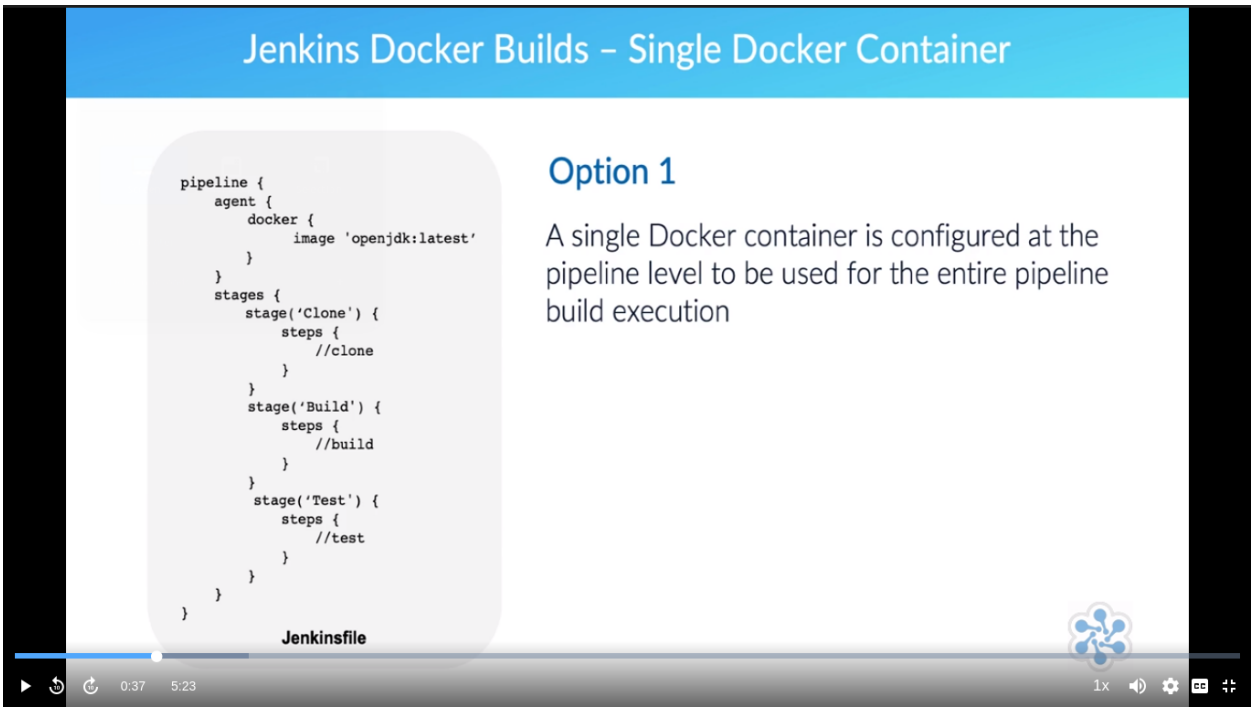


We can use the BlueOcean to Create the Jenkins File as well.

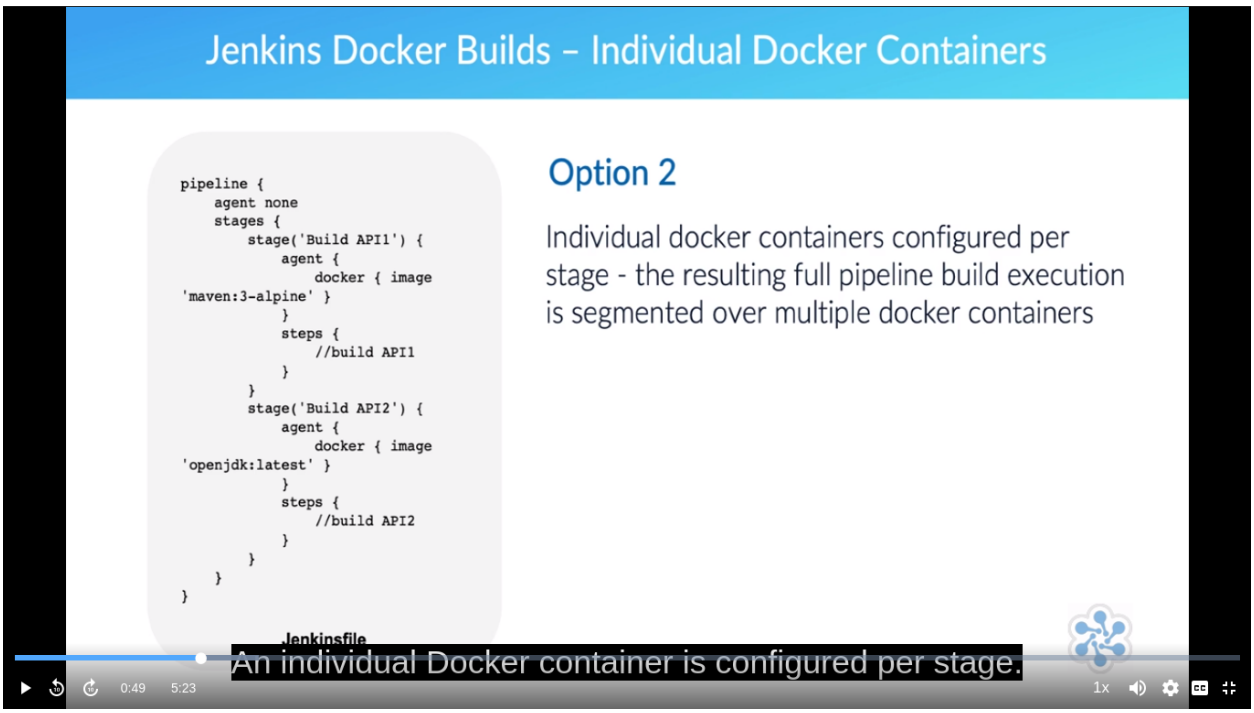
Jenkins and Docker

Integrating Docker into the jenkins build System

A single Docker container is configured at the pipeline level to be used for the entire pipeline build execution



Individual docker container configured per stage - resulting pipeline build execution is segmented over multiple docker containers

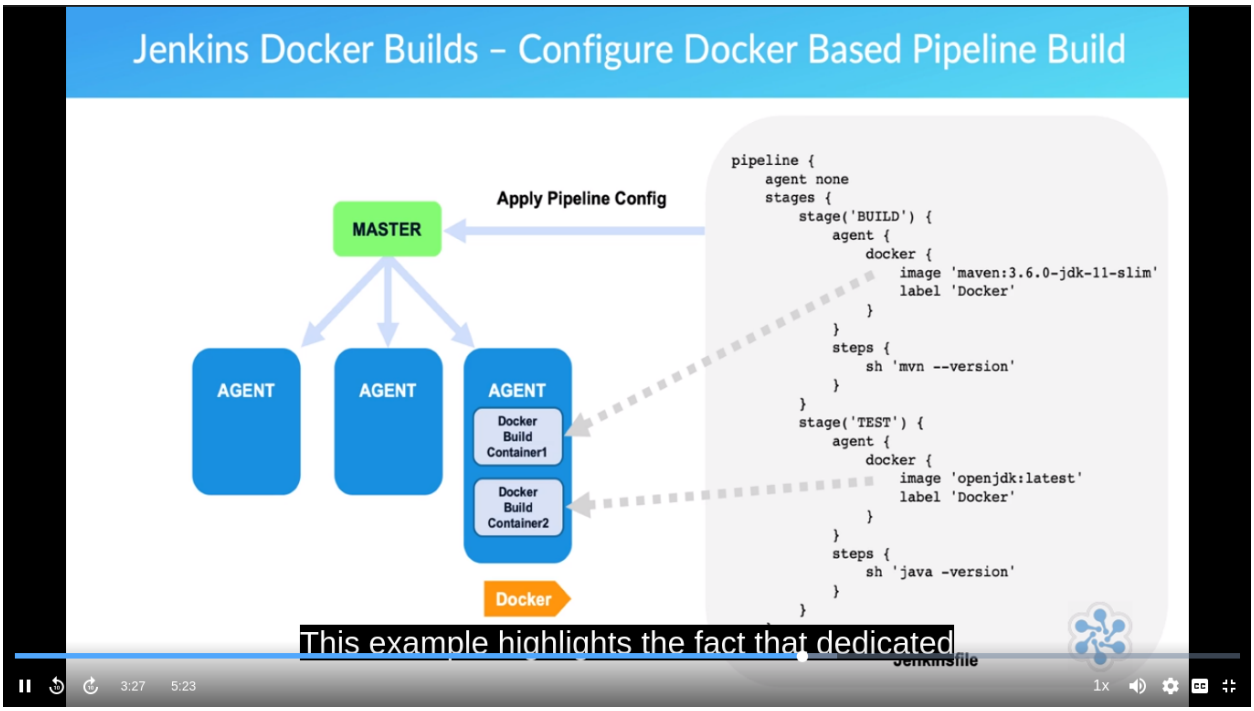


Docker File hosted in the Project Root Directory

Pipeline configuration is instructed to perform a docker build to create a customised docker image complete with updated build tool chain, and then later used for the actual pipeline build execution.

Step 0 :- We need to install docker in all the agent as well the master

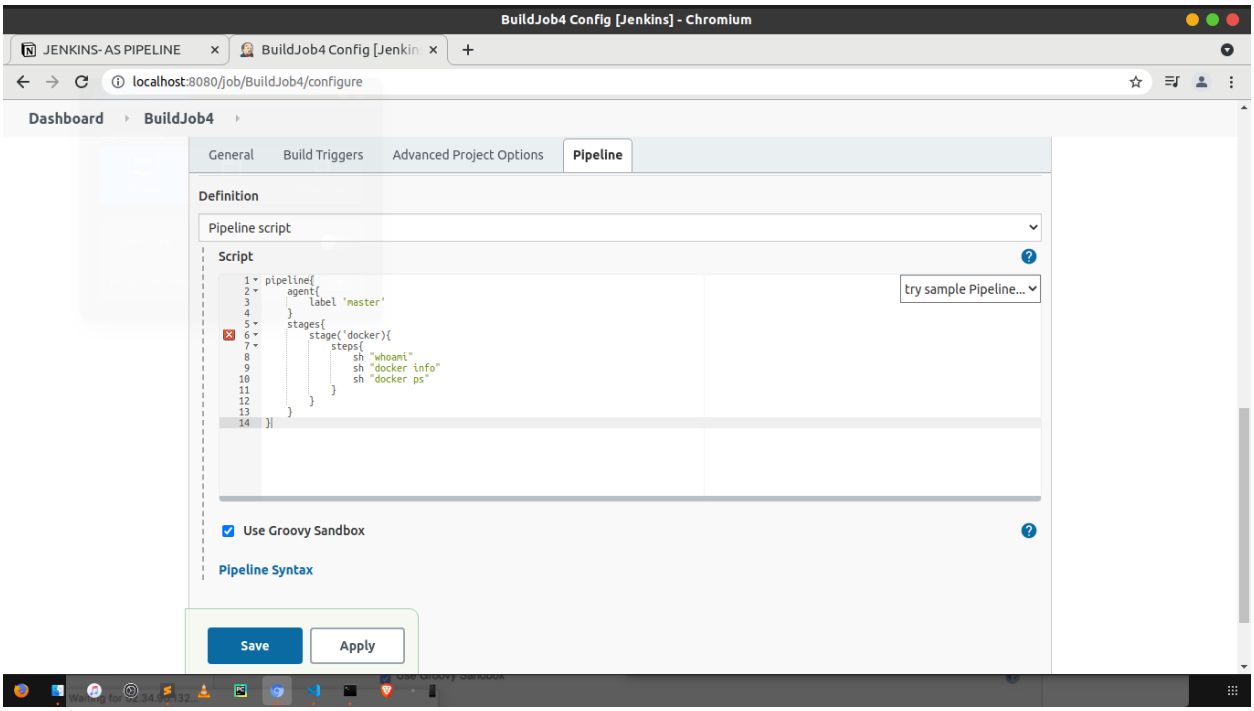
We can use Different Docker images for different stages as well

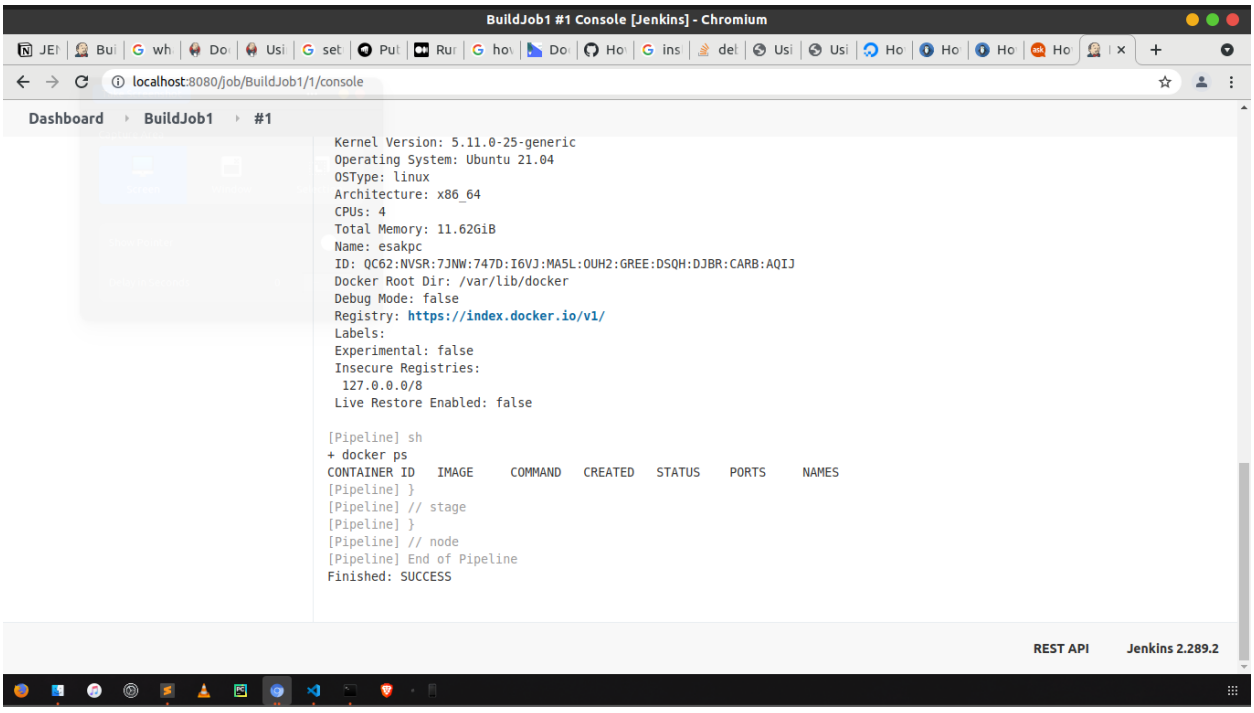


Lets Create A Simple Docker Pipeline Job

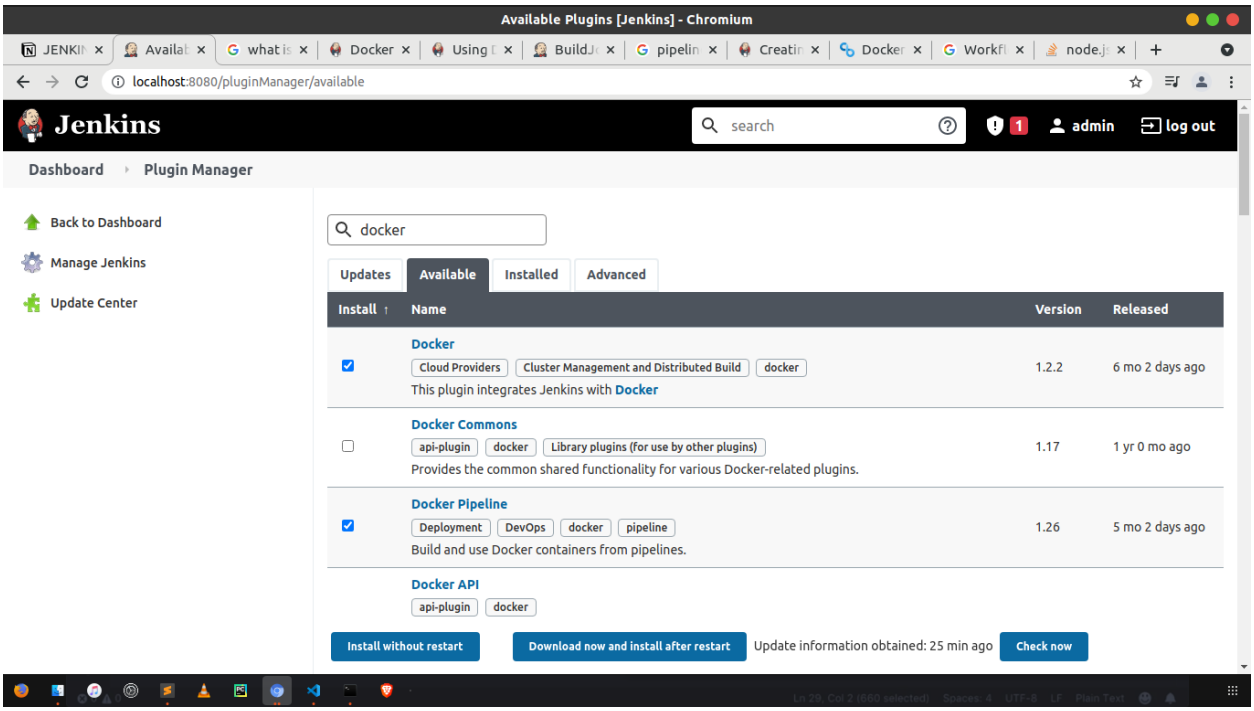
Add the below Pipeline code

```
pipeline{
    agent{
        label 'master'
    }
    stages{
        stage('docker){
            steps{
                sh "whoami"
                sh "docker info"
                sh "docker ps"
            }
        }
    }
}
```





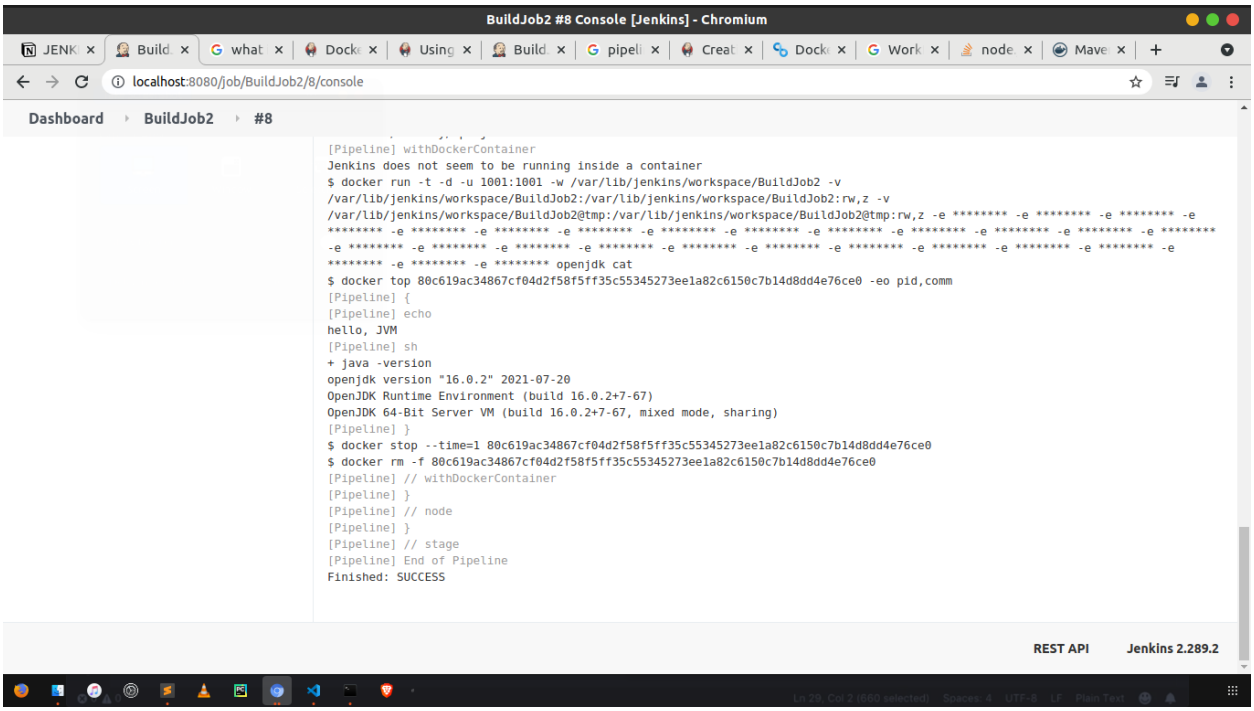
Install Docker and dockerpipeline plugin




```
pipeline{
    agent none
    stages{
        stage('maven'){
        agent {
            docker {
                image 'maven'
                label 'master'
            }
        }
            steps{
                echo "Hello, Maven"
                sh "mvn --version"
            }
        }
        stage('JDK'){
            agent {
                docker{
                    image 'openjdk'
                    label 'master'
                }
            }
            steps{
                echo "hello, JVM"
                sh 'java -version'
            }
        }
    }

}
```

We have Download 2 Images and we build it locally

it will be created automatically and once the process is completed it will delete the images



Intergate Docker with Splunk using Jenkins

GitHub - cloudacademy/devops-webapp: Demonstration Java Servlet based WebApp using Gradle for Build Management

Permalink Failed to load latest commit information. Demonstration Java Servlet based WebApp using Gradle for Build Management You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/cloudacademy/devops-webapp

GitHub - cloudacademy/devops-jenkins-docker-splunk: Jenkins + Docker + Splunk Integration

Jenkins + Docker + Splunk Integration. Contribute to cloudacademy/devops-jenkins-docker-splunk development by creating an account on GitHub.

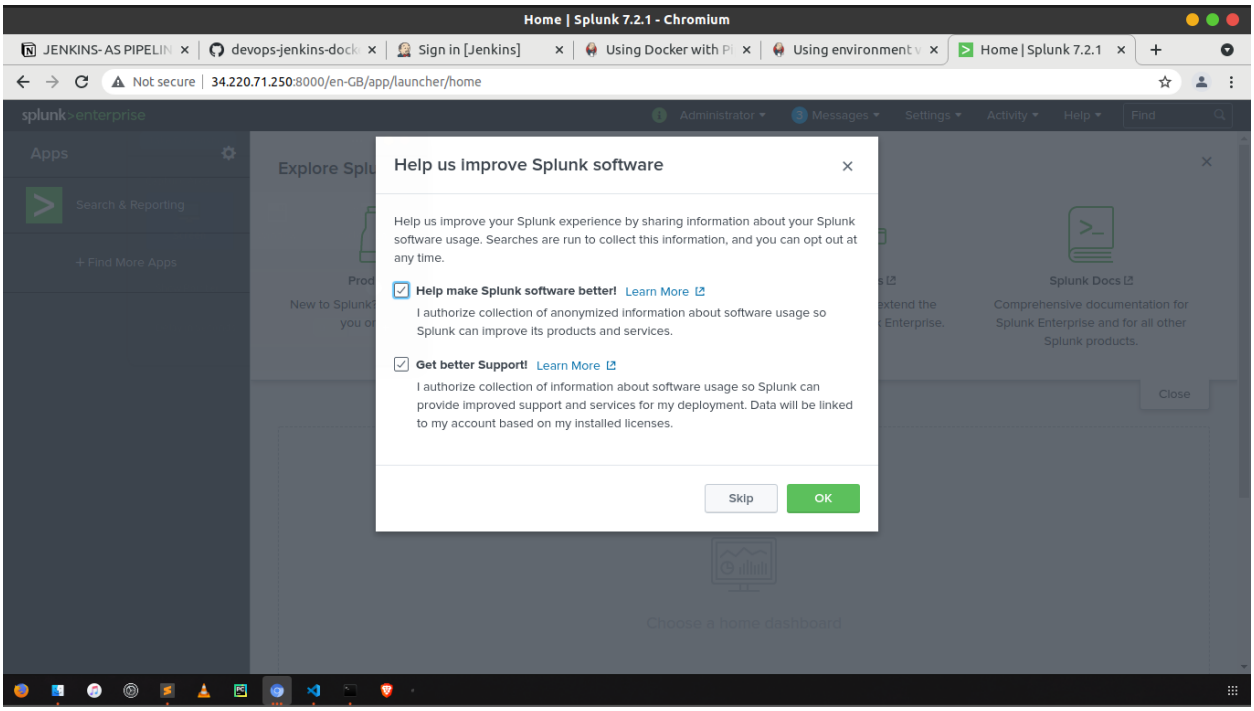
https://github.com/cloudacademy/devops-jenkins-docker-splunk

Checking the Docker container in Table Format

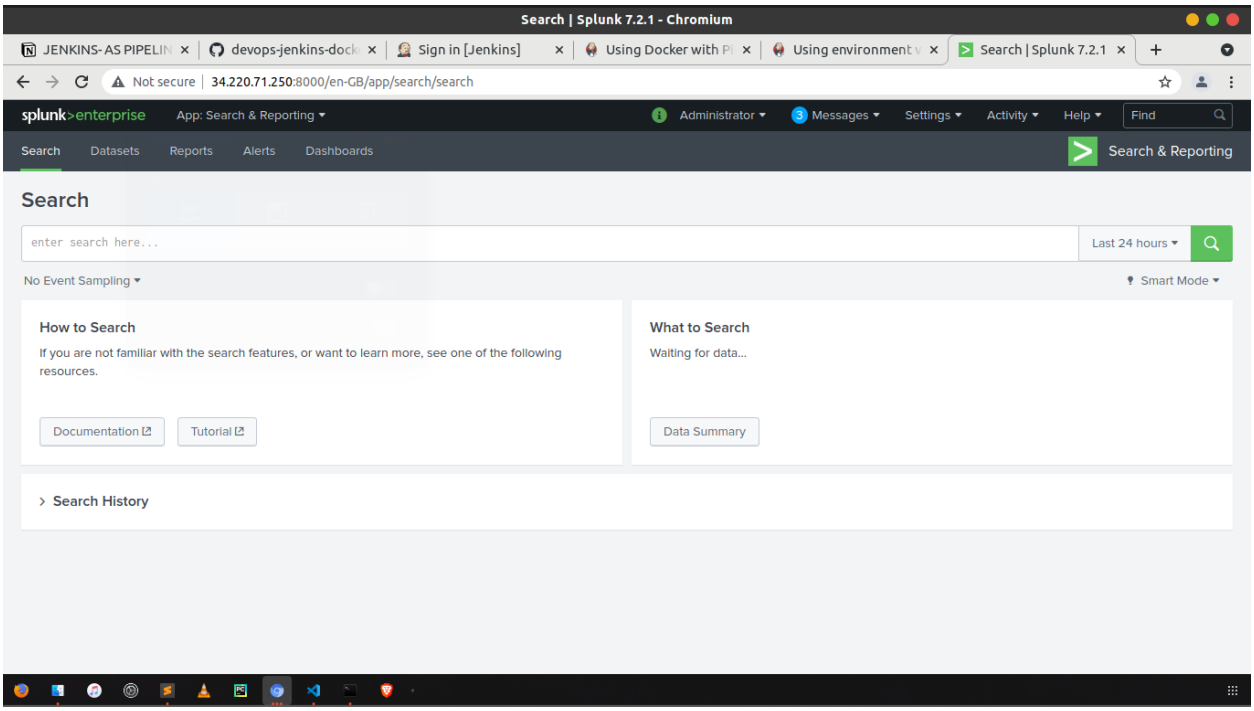
```
docker ps -a --format "table {{.Names}}\t{{.Status}}"

docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

Lets Configure the Splunk with Free Trail Licence

1. Login With the provided username admin and password qwerty123

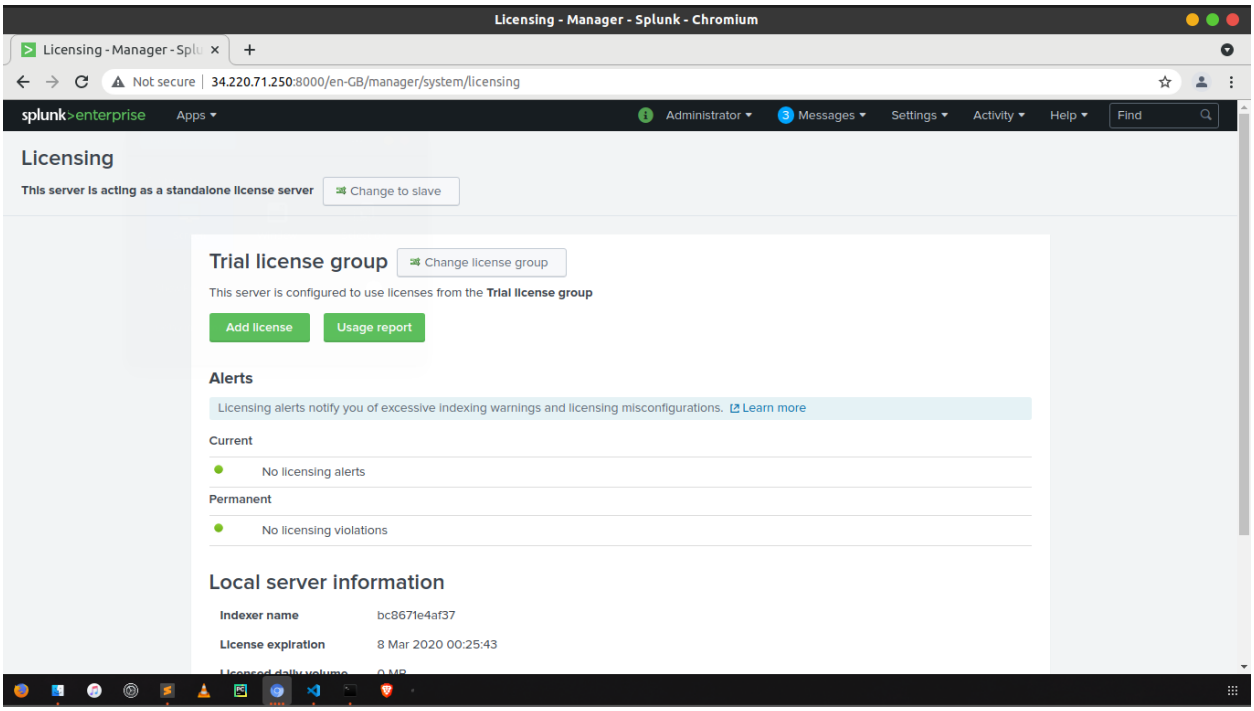


2. Click on the "search and Reporting"

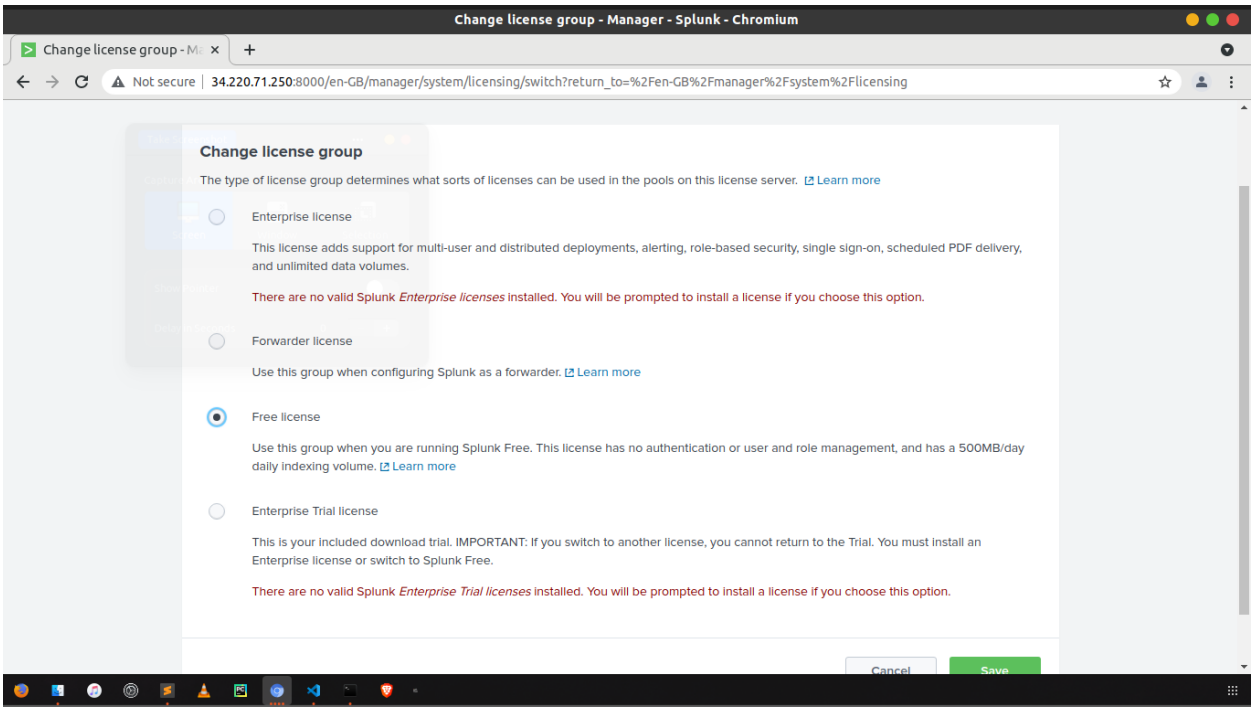


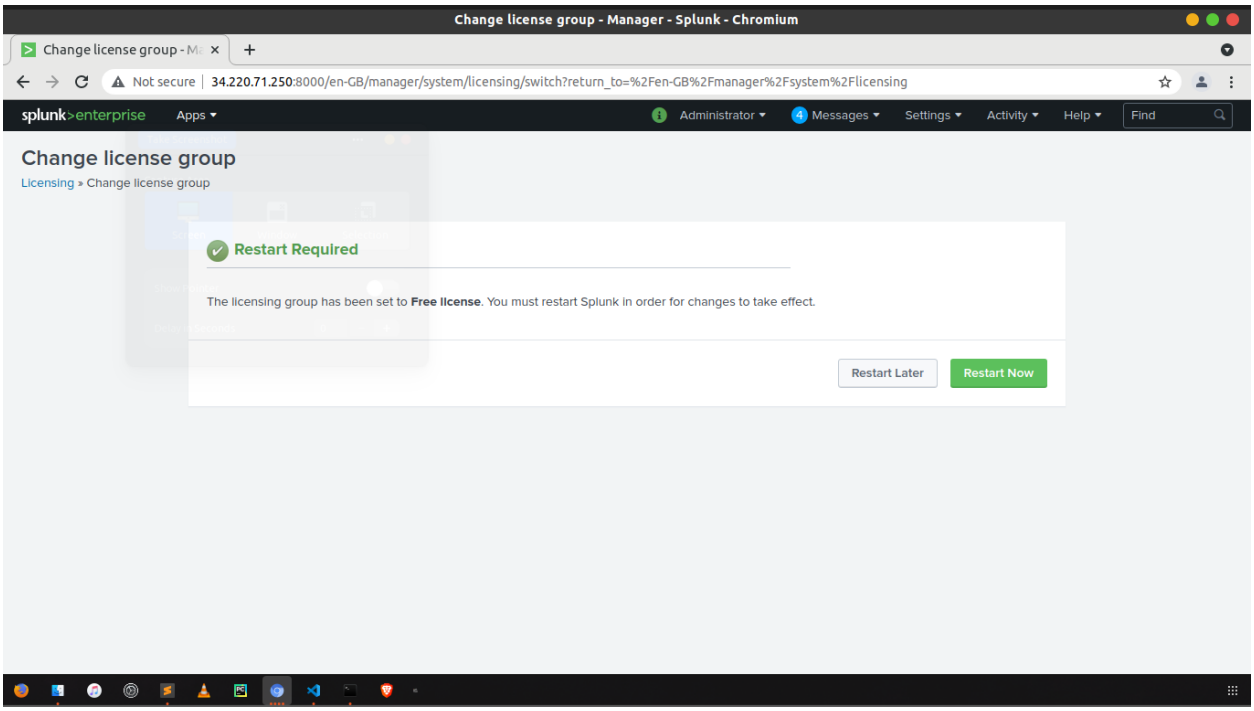
3. Reconfigure Splunk with Trail licence

Goto Setting → Licencing

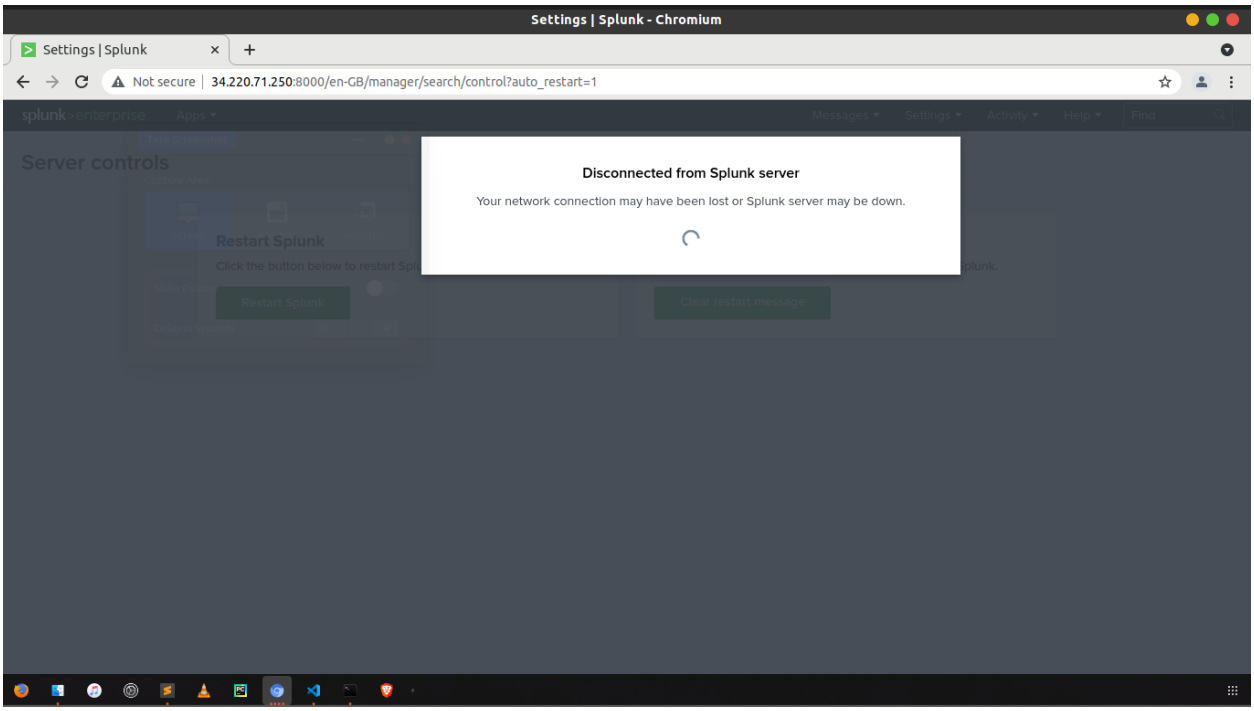


Click on Change Licence Group and Choose Trail Licence

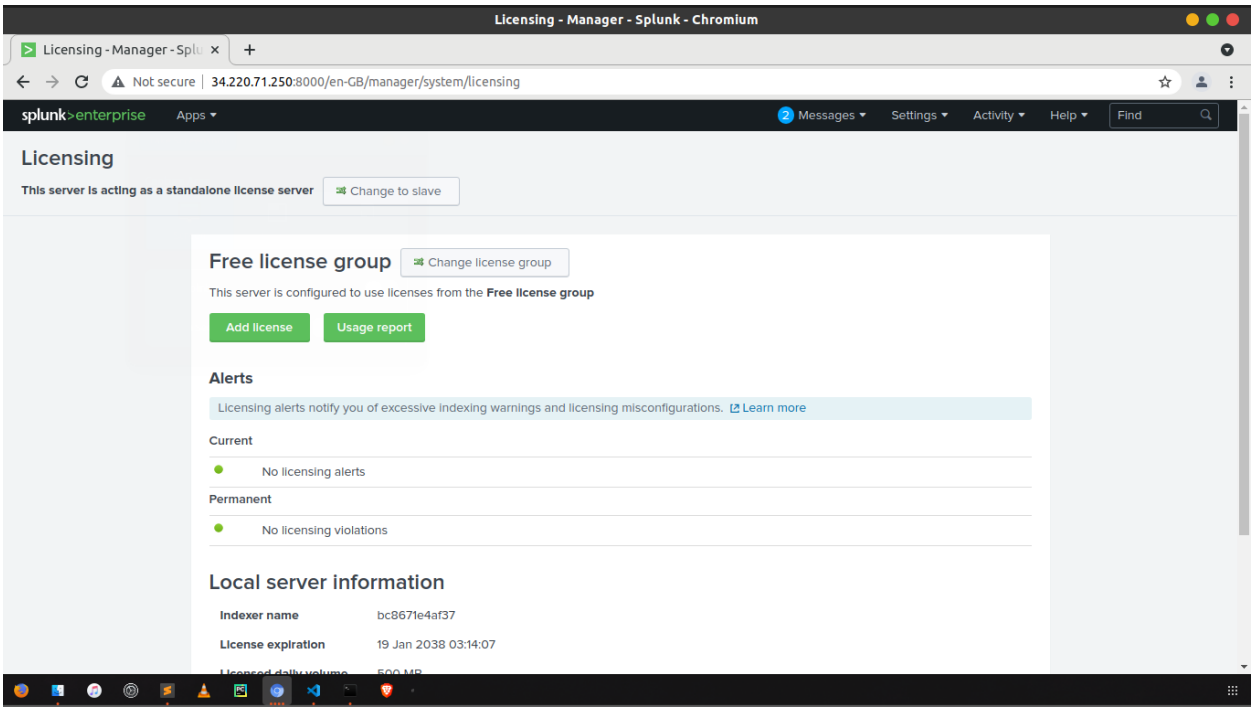




Restart the Splunk



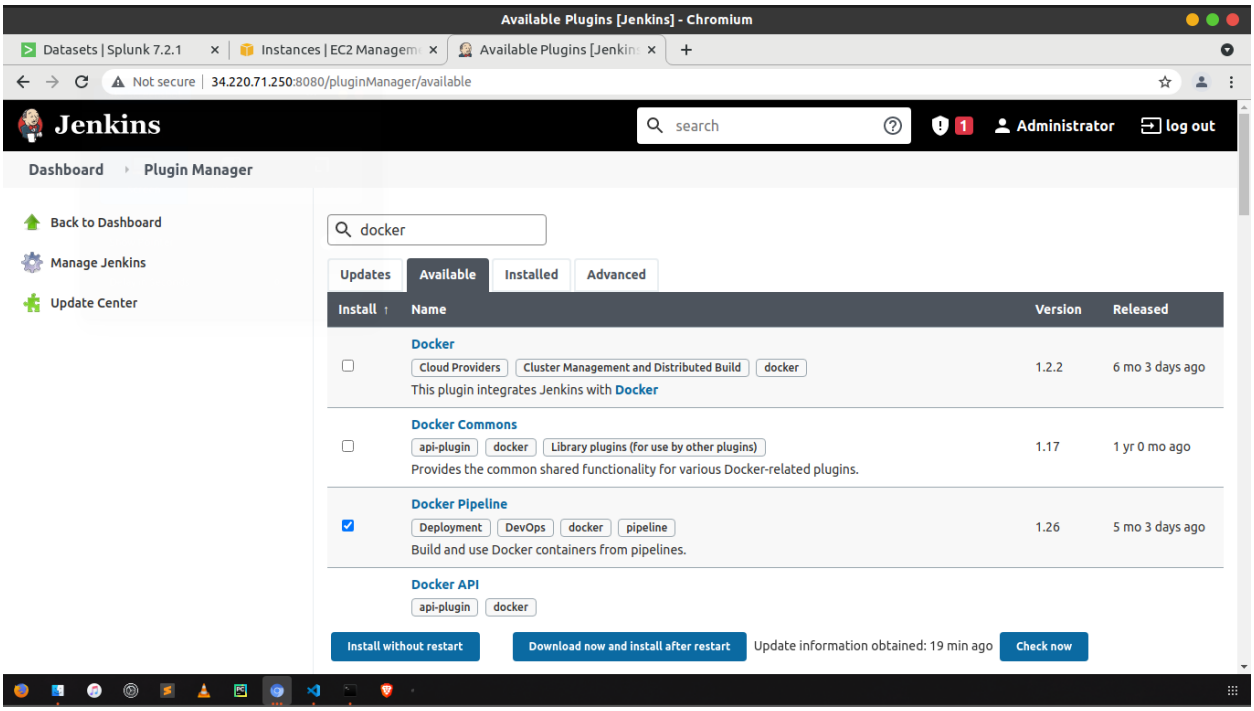
Once its Restarted , verify the Splunk is running in free licence



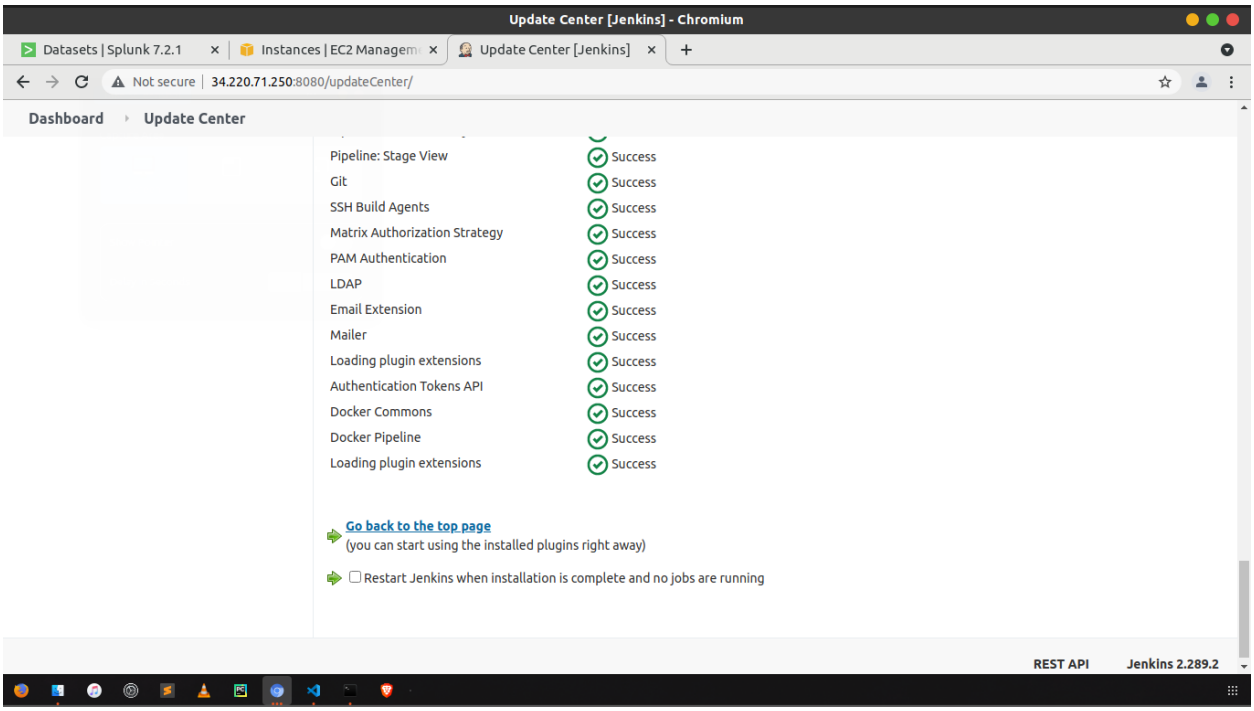
> In the following Lab Steps, you will configure a Jenkins build pipeline to build a custom Docker container, which will host a sample Java servlet web application, also built within Jenkins. The sample Java servlet web application is designed to use Log4J2 to perform logging. The corresponding log files generated at runtime will be published to Splunk and, therefore, will be available for analysis within the Search & Reporting area.

In this Lab Step, you will continue using the Jenkins administration web console. You will configure the Gradle and Docker tools required to compile the sample Java servlet web application into a WAR (Web application ARchive) artifact file, and then a docker image that will use Tomcat to serve the WAR artifact file created in the first build step. You will configure the Gradle and Docker tools to be automatically installed at build time by Jenkins.

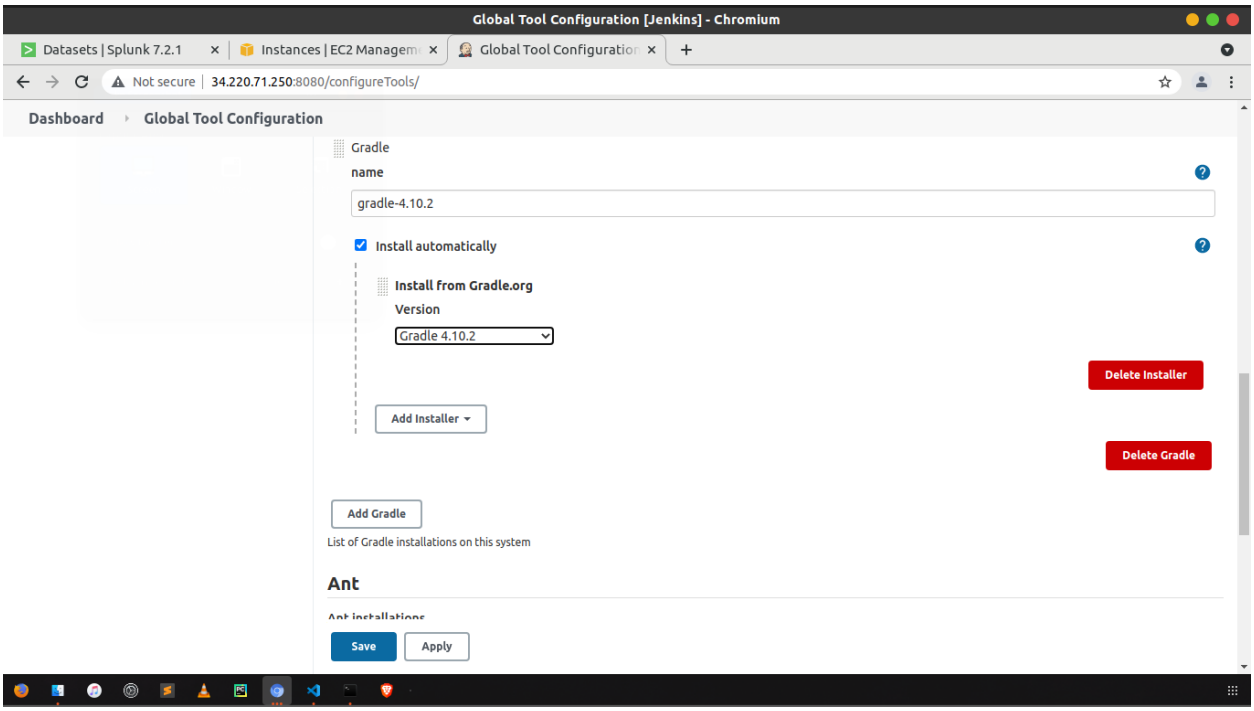
Step 0 Install Docker pipeline Plugin



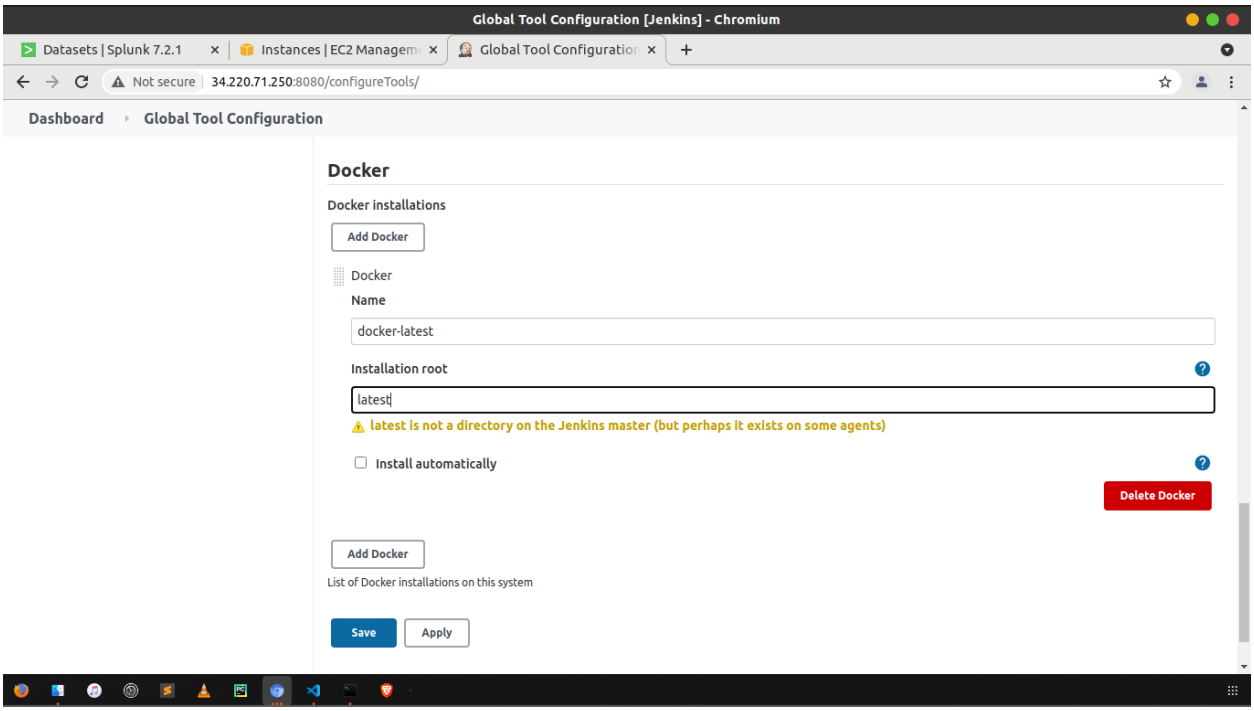
Verify it installed sucessfully



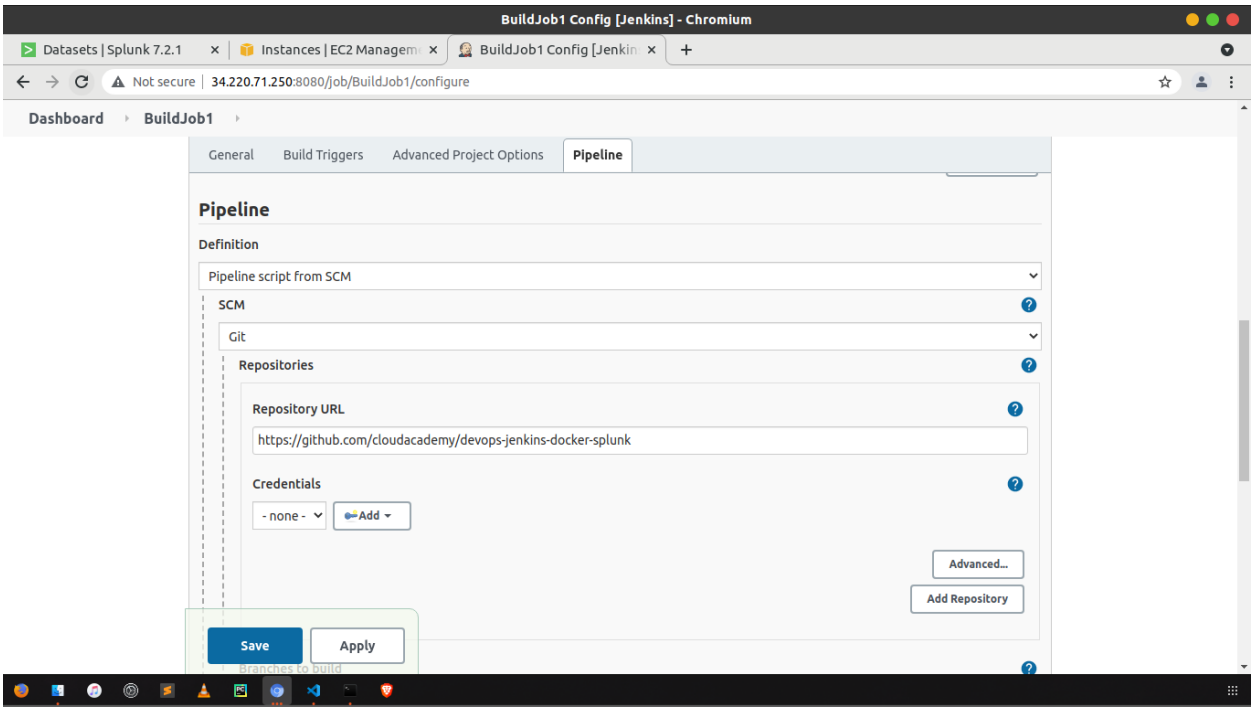
Install Gradle in Jenkins



Install Docker



Lets Create Jenkins Pipeline



Lets Deep Dive into the Jenkins File

devops-jenkins-docker-splunk/Jenkinsfile at master · cloudacademy/devops-jenkins-docker-splunk

Jenkins + Docker + Splunk Integration. Contribute to cloudacademy/devops-jenkins-docker-splunk development by creating an account on GitHub.

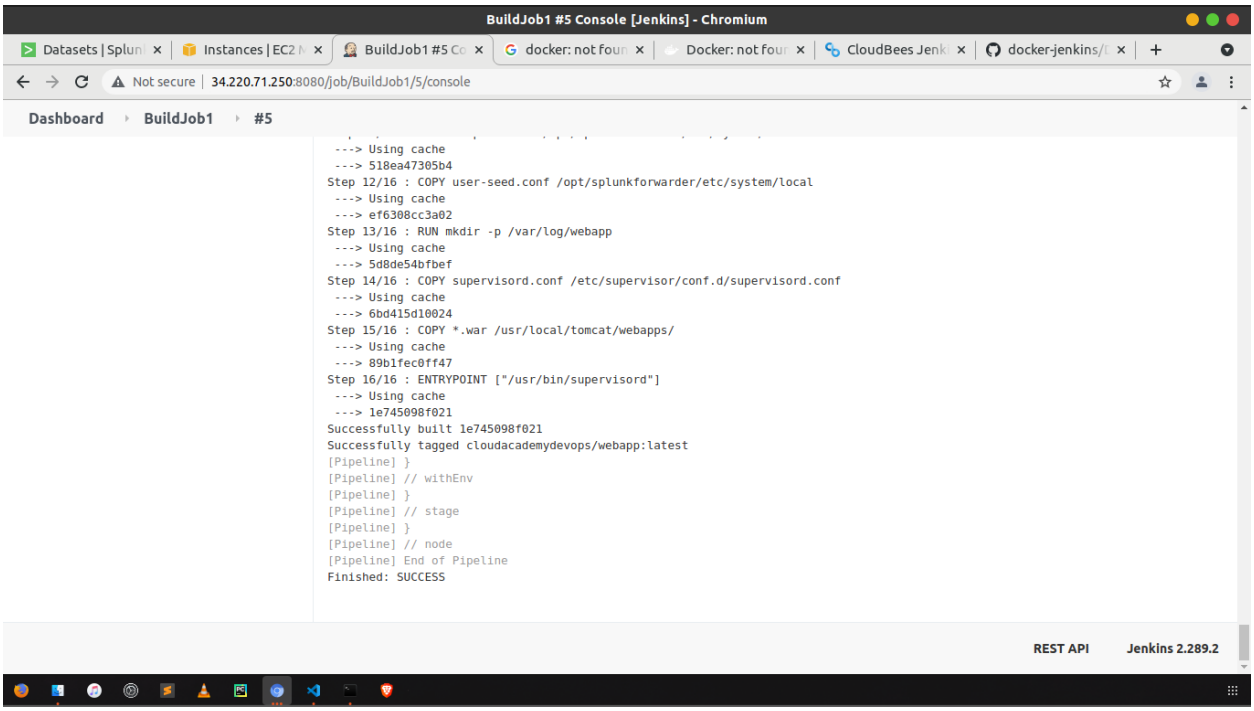
https://github.com/cloudacademy/devops-jenkins-docker-splunk/blob/master/Jenkinsfile

cloudacademy/c
jenkins-docker-
Jenkins + Docker + Splunk Inte
1 Contributor
0 Issues
4 St

```
//START-OF-SCRIPT
node {
    def SPLUNK_HOSTNAME='splunk'
    def DOCKER_HOME = tool name: 'docker-latest'
    def GRADLE_HOME = tool name: 'gradle-4.10.2', type: 'hudson.plugins.gradle.GradleInstallation'
    def REPO_URL = 'https://github.com/cloudacademy/devops-webapp.git'
    def DOCKERHUB_REPO = 'cloudacademydevops/webapp'

    stage('Clone') {
        git url: REPO_URL
    }

    stage('Build') {
        sh "${GRADLE_HOME}/bin/gradle build"
        sh "ls -la build/libs/*.war"
        sh "echo ====================="
        sh "cp build/libs/*.war docker/webapp.war"
        sh "pwd"
        sh "ls -la"
        sh "ls -la ./docker"
    }

    stage ('Docker Build') {
        docker.withTool('docker-latest') {
            sh "printenv"
            sh "pwd"
            sh "ls -la"
            def image1 = docker.build("${DOCKERHUB_REPO}:${BUILD_NUMBER}", "./docker")
            def image2 = docker.build("${DOCKERHUB_REPO}:latest", "./docker")
        }
    }
```

```
    /*
    stage ('Docker Push') {
        docker.withTool('docker-latest') {
            withCredentials([usernamePassword(credentialsId: 'dockerhub',
                            usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD')]) {
                sh "docker login -u ${USERNAME} -p ${PASSWORD} https://index.docker.io/v1/"
                sh "docker push ${DOCKERHUB_REPO}:${BUILD_NUMBER}"
                sh "docker push ${DOCKERHUB_REPO}:latest"
            }
        }
    }
    */
}
//END-OF-SCRIPT
```

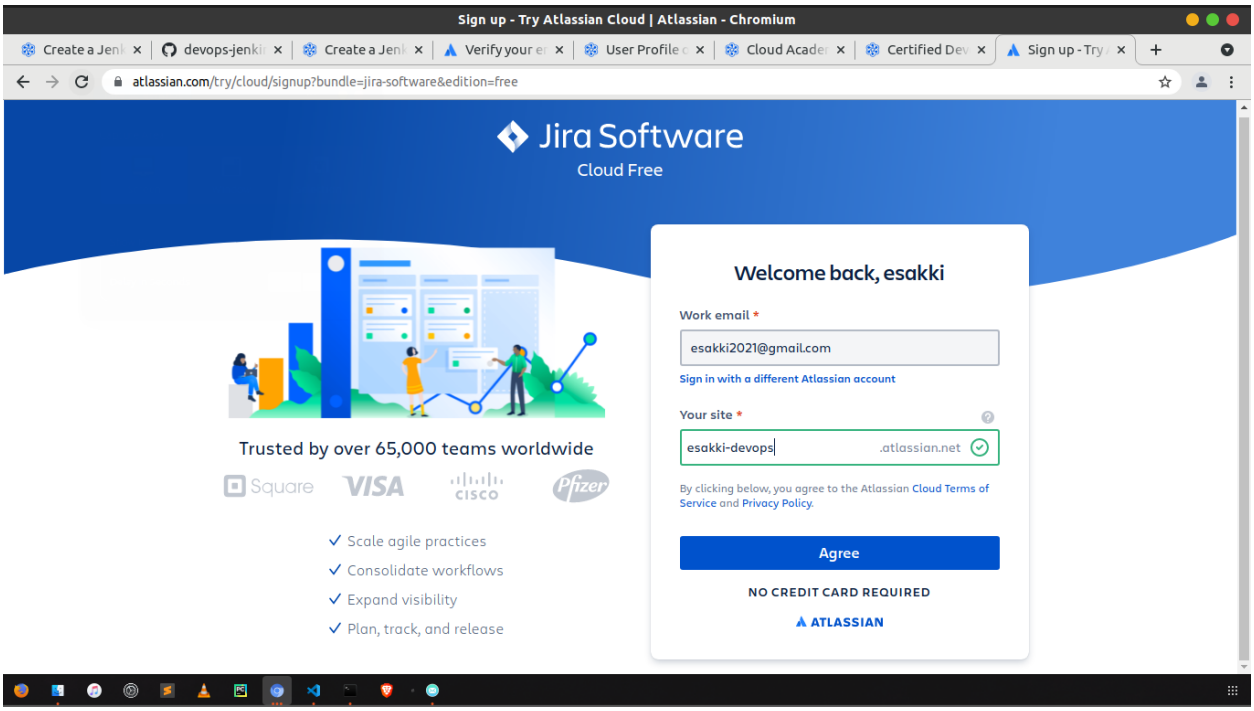
Lets trigger the Build



Integrating Jenkins and Jira

> Integrating Jenkins and Jira together provides you with a solution that can be used to report issues as they happen for any CICD pipeline build job. Jenkins can be configured to publish any and all build results, artifacts, and/or bugs directly into Jira. DevOps teams can then use Jira to project manage the required fixes necessary to get their applications into production.

Create a Atlassian Account



GitHub - cloudacademy/devops-jenkins-jira: Jenkins + Jira Integration

Jenkins + Jira Integration. Contribute to cloudacademy/devops-jenkins-jira development by creating an account on GitHub.

https://github.com/cloudacademy/devops-jenkins-jira

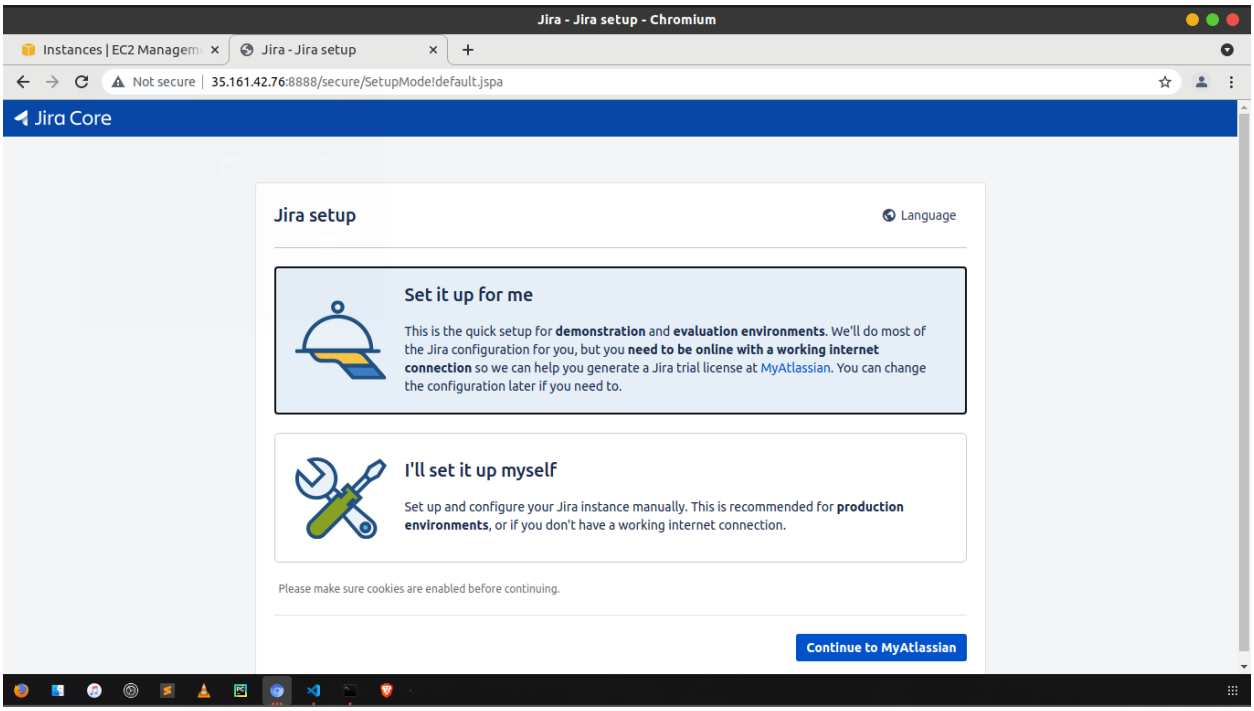
cloudacademy/**devops-jenkins-jira**

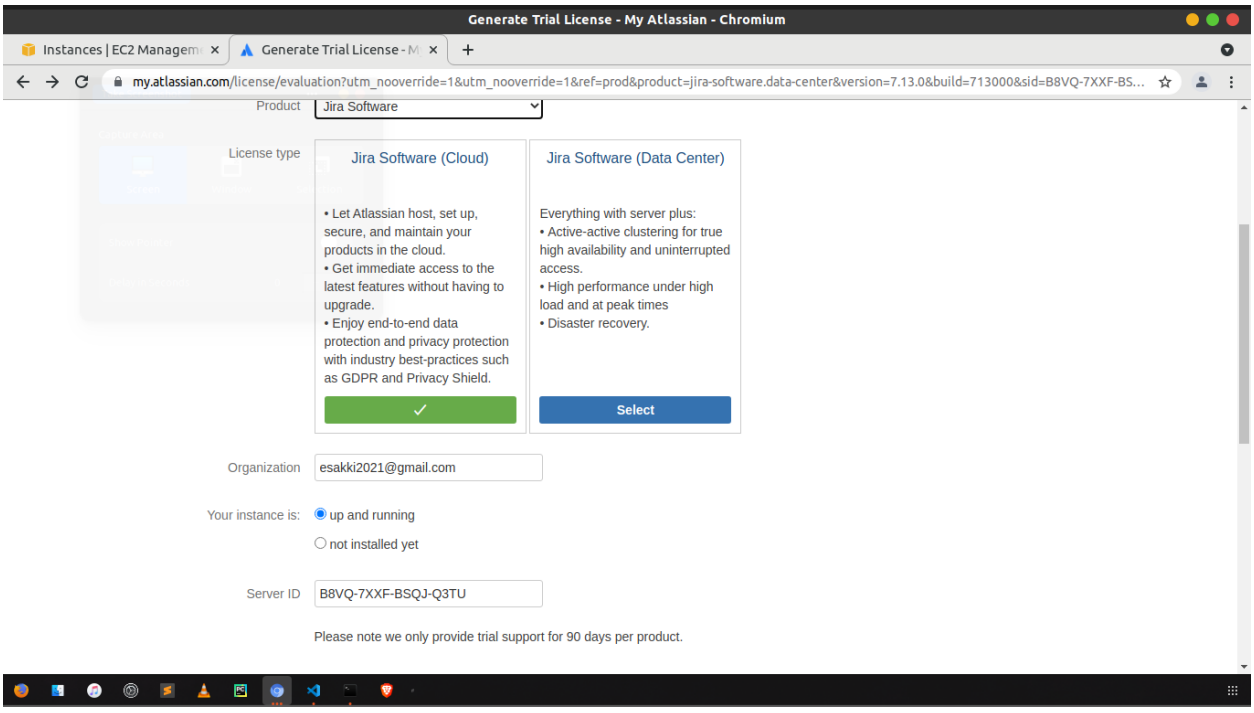
Jenkins + Jira Integration

2 Contributors · 0 Issues · 3 Stars · 13 Forks

Bring the Dokcer-compose File Up

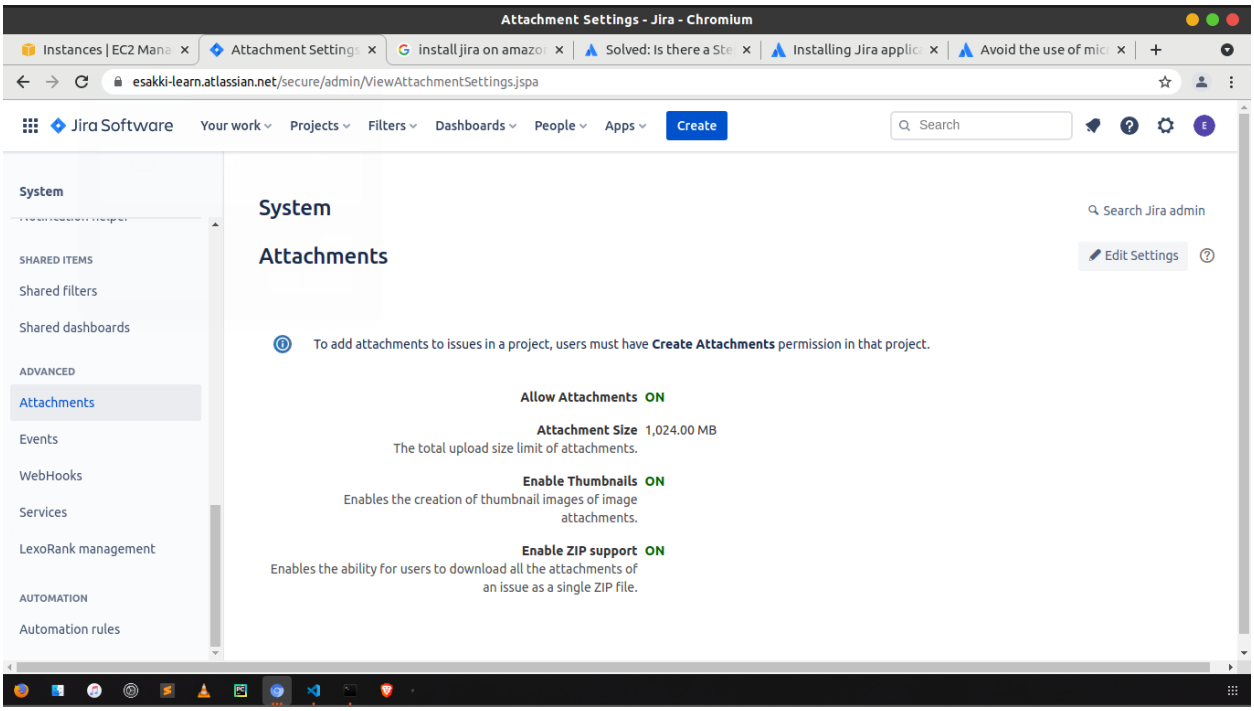
checking the Jira Installation





Once the Setup Is Completed

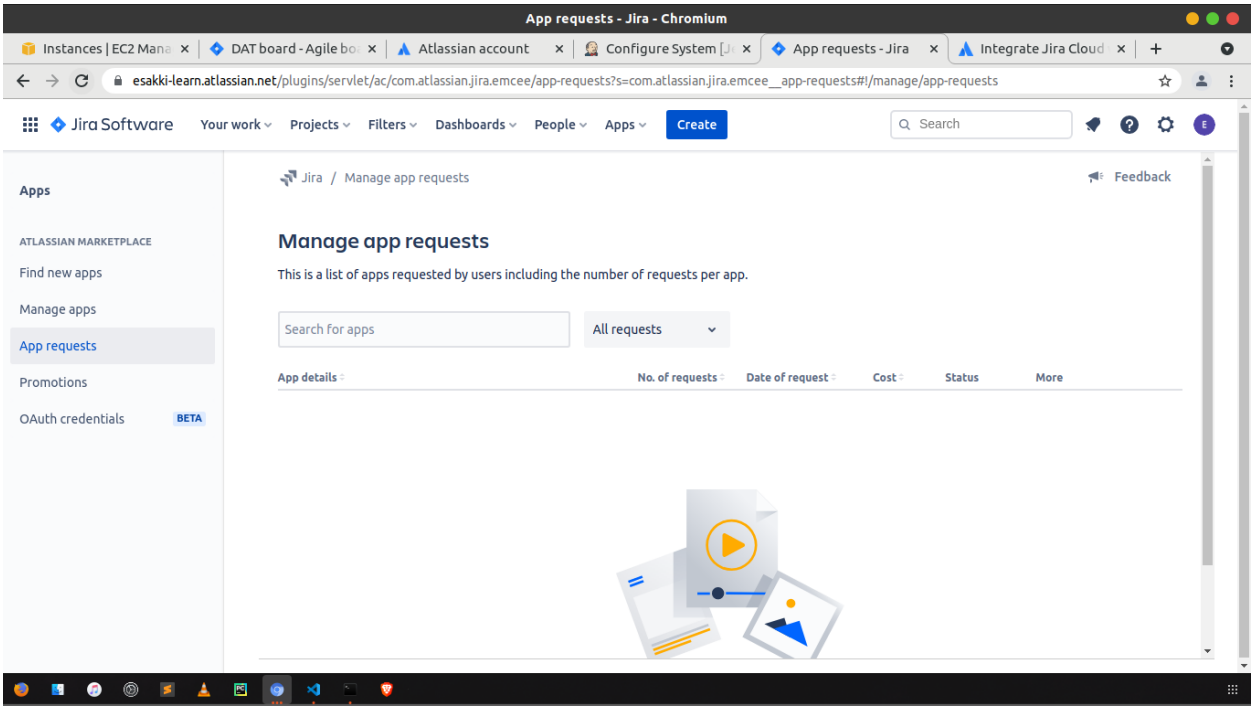
goto Seetings → system → Check for advanced in the left Side panel →

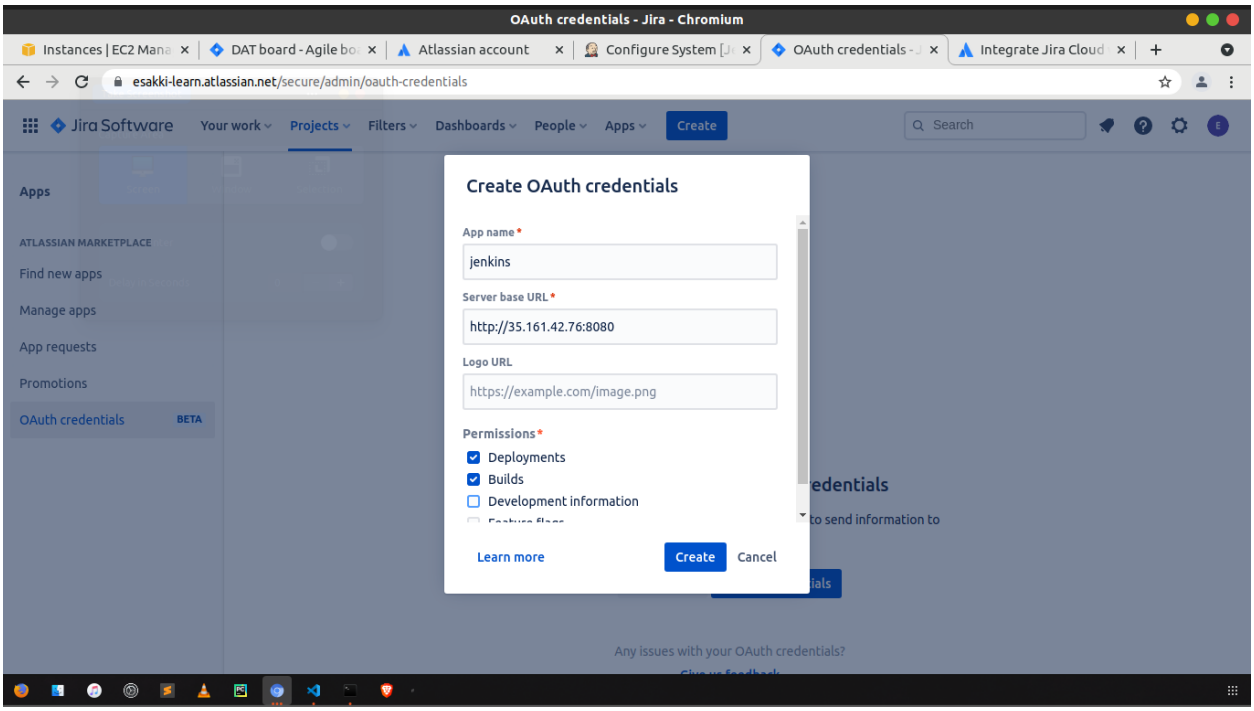


Lets Configure Jira

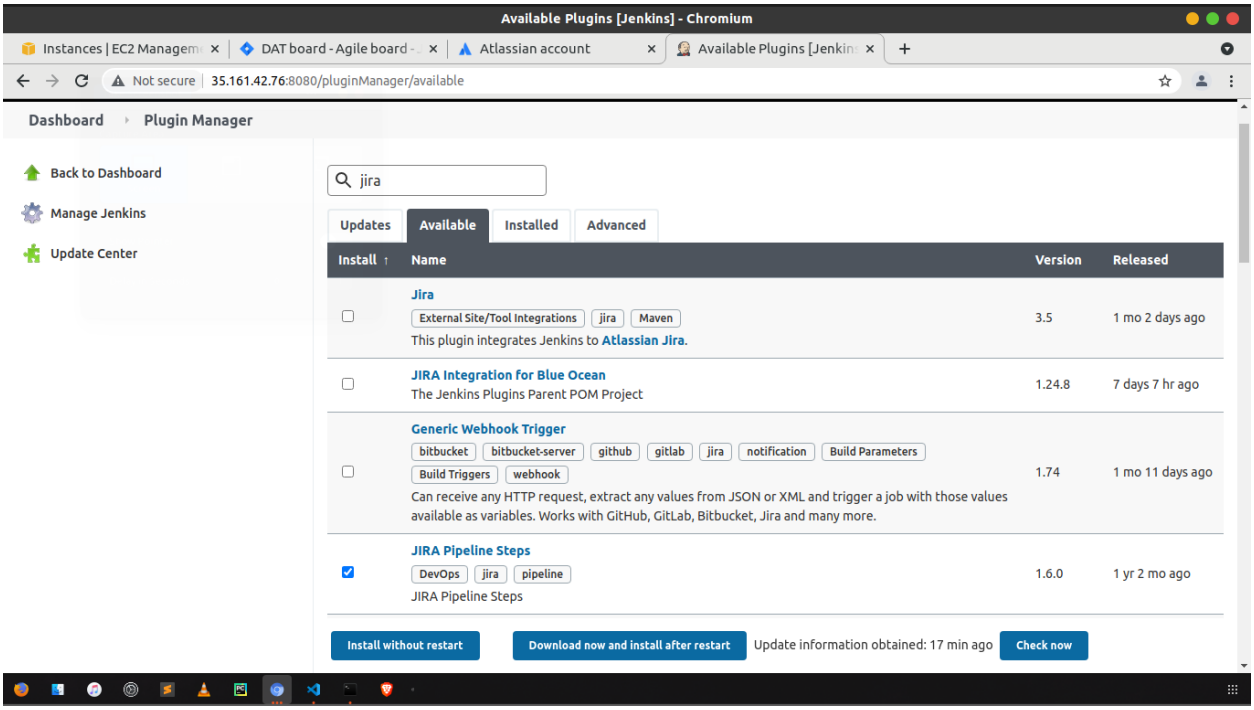
Create OAUTH Token

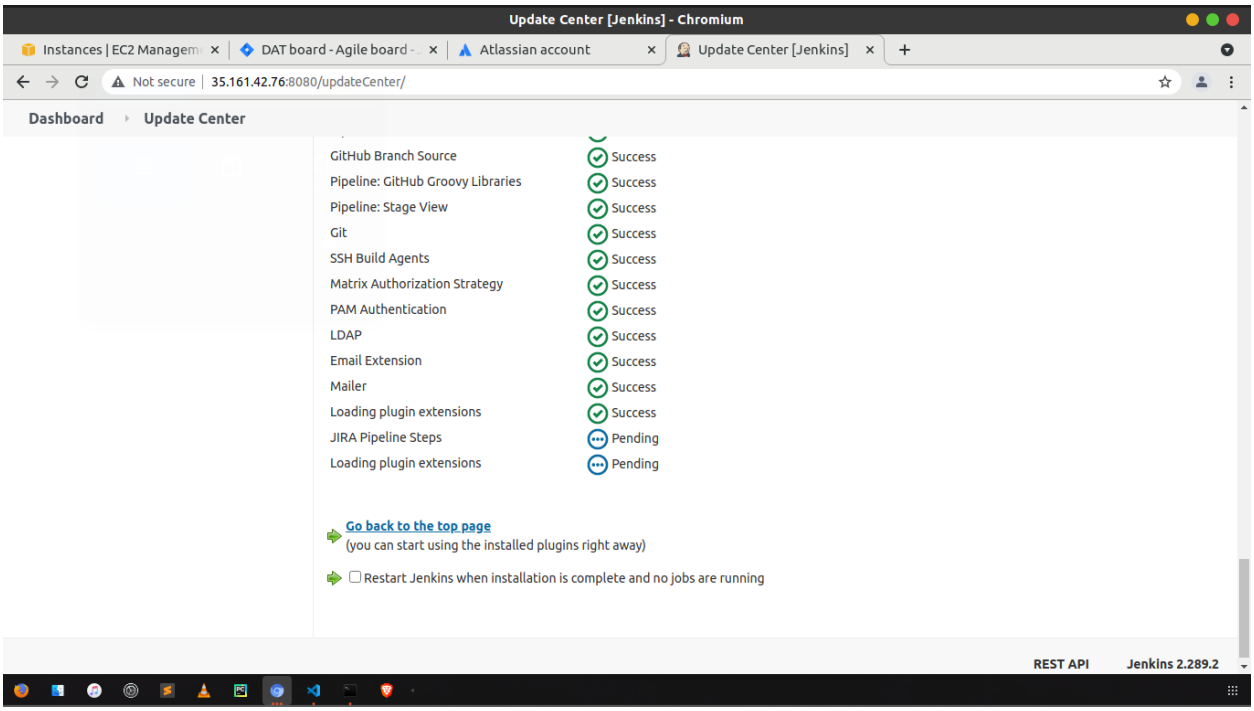
Settings → App ⇒ OAUTHCREDENTIALS →



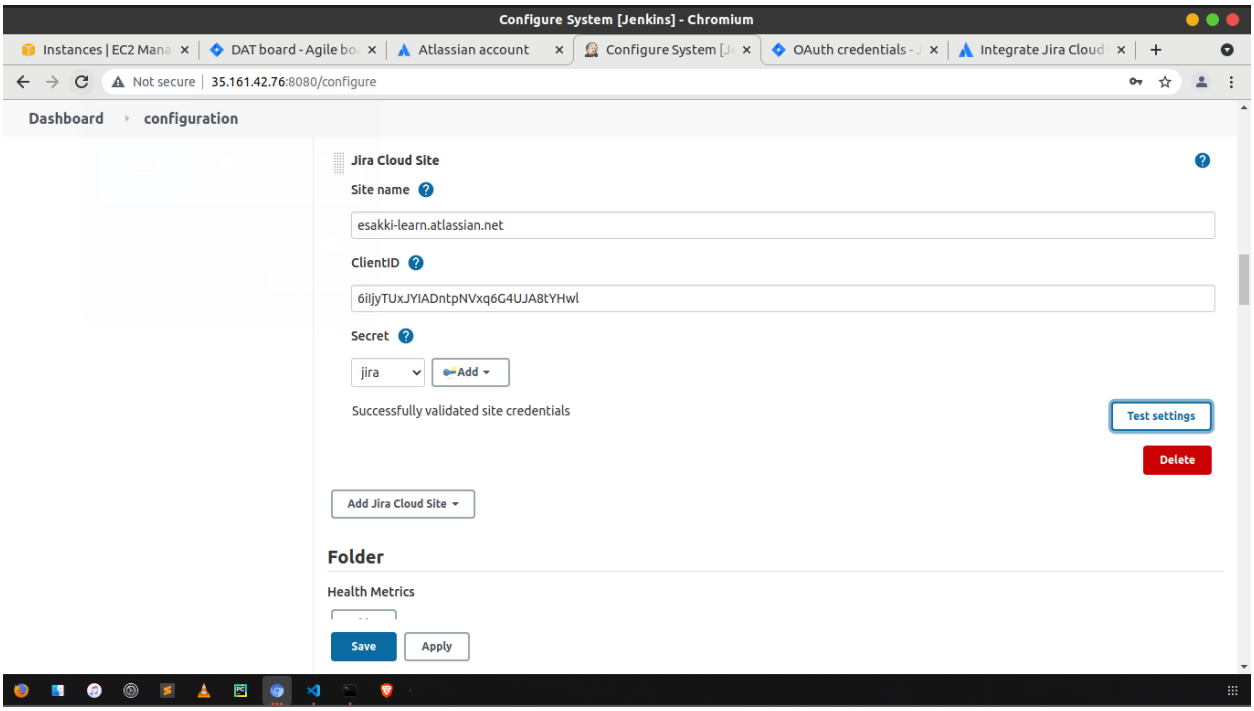


Install jira pipeline plugin

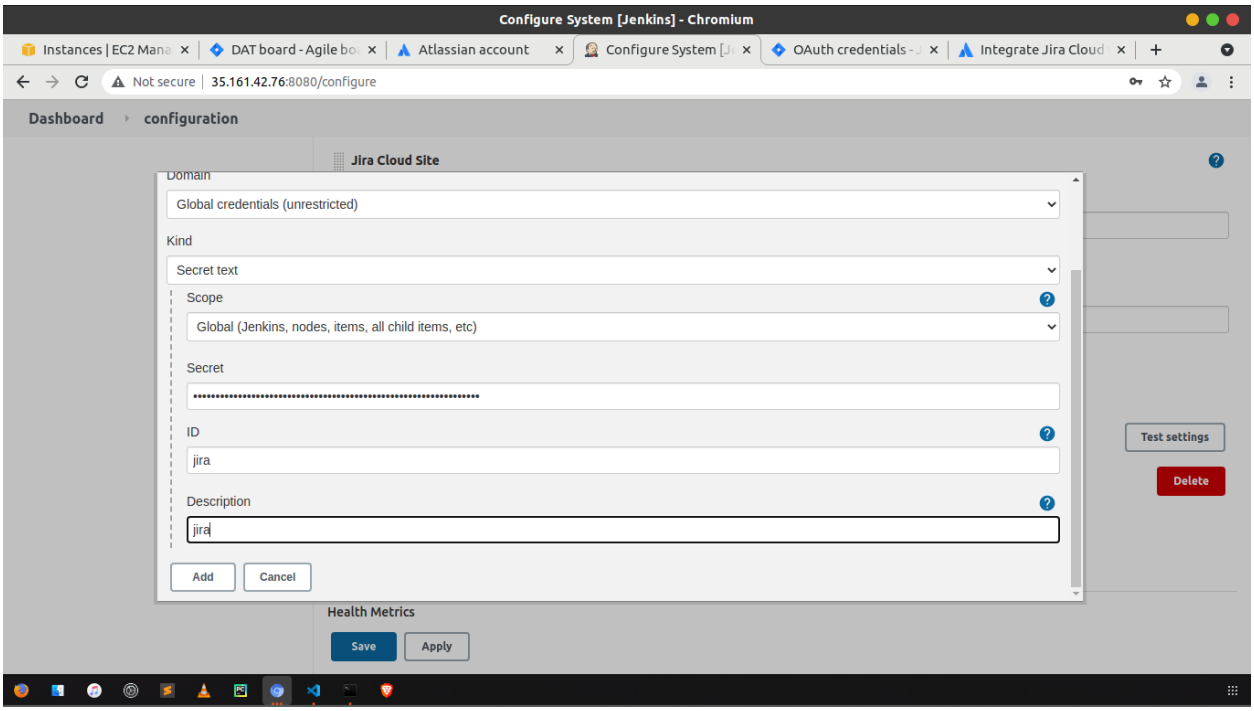




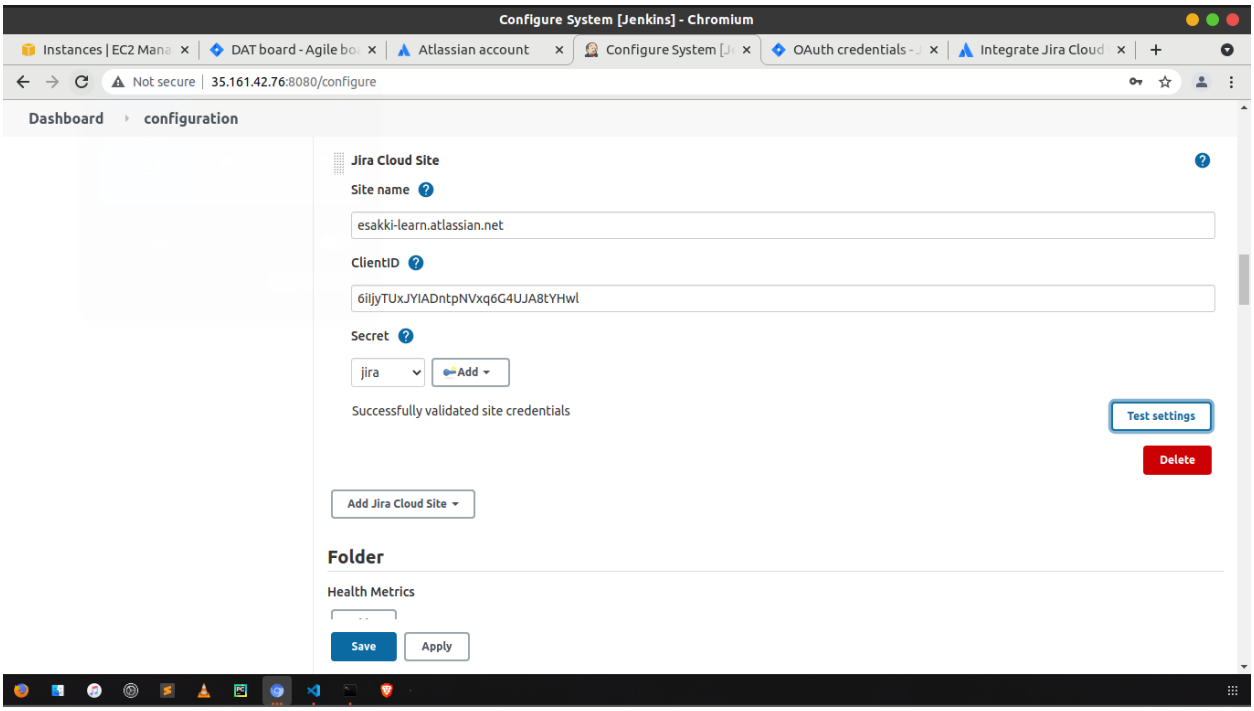
Lets Configure the Plugin



Add Secret



test The Connection




```
node{
def server = Artifactory.server 'artifactory'

def rtGradle = Artifactory.newGradleBuild()
```

```
withCredentials([usernamePassword(
    credentialsId: 'artifactory',
    usernameVariable: 'USERNAME',
    passwordVariable: 'PASSWORD'
)]) {
    server.username = "${USERNAME}"
    server.password = "${PASSWORD}"
}

def builfInfo

stage('clone'){
    git url: '<https://github.com/cloudacademy/devops-webapp.git>'
}

stage('Artifactory Config'){
    rtGradle.tool = "gradle-4.10.2"
    rtGradle.deployer repo:'gradle-release-local', server: server
    rtGradle.resolver repo:'jcenter', server: server

}

stage('build'){

    rtGradle.run rootDir: "./", buildFile: 'build.gradle', tasks: 'clean build'

}

stage("publish"){
    buildInfo = rtGradle.run rootDir: "./", buildFile: 'build.gradle', tasks: 'artifactoryPublish'
}

stage("deploy"){
    server.publishBuildInfo buildInfo
}
```

GitHub - cloudacademy/jenkins-cicd-adv: Jenkins CICD Advanced Demo Code and Configs

Contains Jenkins code and configuration examples as used within the Jenkins CI/CD - Advanced course.

https://github.com/cloudacademy/jenkins-cicd-adv

cloudacademy/**jenkins-cicd-adv**

Jenkins CICD Advanced Demo Code and Configs

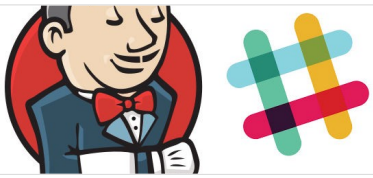
1 Contributor 0 Issues 0 Stars 5 Forks

Integration With Slack

Jenkins and Slack Integration

Slack is a chat software. Slack has hook's to provide communication from your tools to your team. Jenkins can send notification from all your jobs to your team. It can use channels to send especific notification for specific teams. First, create your

https://medium.com/@gustavo.guss/jenkins-and-slack-integration-fbb031cd7895



Slack

https://techwithesak.slack.com/services/B029CAEMVTR?added=1

```
def COLOR_MAP = [
    'SUCCESS': 'good',
    'FAILURE': 'danger',
]

def getBuildUser() {
    return currentBuild.rawBuild.getCause(Cause.UserIdCause).getUserId()
}
```

```
node {
    def GRADLE_HOME = tool name: 'gradle-4.10.2', type: 'hudson.plugins.gradle.GradleInstallation'
    def REPO_URL = 'https://github.com/cloudacademy/devops-webapp.git'

    stage('clone'){
        git url: REPO_URL
    }
    try{
        stage('build'){

            throw new Exception("Throw to stop pipeline")
            sh "${GRADLE_HOME}/bin/gradle bud --info 2>1 | tee gradle.build.${BUILD_NUMBER}.log"
            //currentBuild.result = "FAILED"
        }
    }catch(e){
        echo 'Build is failed sending Email'
        emailext (
        subject: "STARTED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
        body: """<p>STARTED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]':</p>
            <p>Check console output at &QUOT;<a href='${env.BUILD_URL}'>${env.JOB_NAME} [${env.BUILD_NUMBER}]</a>&QUOT;</p>""",
        recipientProviders: [[$class: 'DevelopersRecipientProvider'], [$class: 'RequesterRecipientProvider']]
    )

    slackSend(
        channel: '#dalilyalerts',
        color: COLOR_MAP[currentBuild.currentResult],
        message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME} build ${env.BUILD_NUMBER} by \n More info at: ${env.BUILD_URL}"
    )
    }




}
```

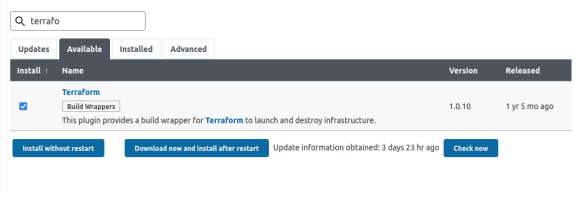
Ansible COde to start and delete DOcker in a MAchine

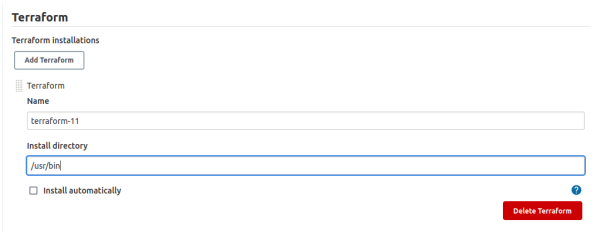
```
---
- name: playbook to install the docker
  hosts: all
  become: true
  tasks:
  - name: Remove older version of Docker
    yum:
      name: [docker, docker-client, docker-client-latest , docker-common, docker-latest , docker-latest-logrotate, docker-logrotate, docker
      state: absent

  - name: Installing Docker Prerequisite packages
    yum:
      name: [yum-utils, device-mapper-persistent-data, lvm2, python3, python3-pip]
      state: installed

  - name: Download the Docker
    get_url:
      url: https://download.docker.com/linux/centos/docker-ce.repo
      dest: /etc/yum.repos.d/docker-ce.repo
      mode: 0644

  - name: Install docker
    yum:
      name: docker-ce
      state: present

  - name: Install docker pip packages
    pip:
      name: docker

  - name: Start the Docker
    ansible.builtin.service:
      name: docker
      enabled: yes
      state: started
```

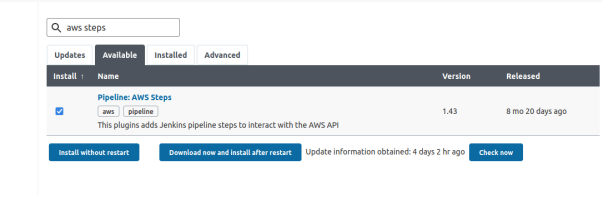
```
- name: Add ansadmin to docker group
  ansible.builtin.shell:
    cmd: usermod -aG docker ansadmin


- name: Start the Docker
  ansible.builtin.service:
    name: docker
    enabled: yes
    state: restarted


- name: Check the Docker
  ansible.builtin.shell:
    cmd: sudo docker run -d --name myapphello esak2021/myhello:1.1.1

- name: Get Docker Info
  docker_host_info:
    containers: yes
  register: container_result

- name: Stop the Containers
  docker_container:
    name: myapp
    state: stopped
  #loop: "{{ container_result.containers | map(attribute='Id') | list }}"

- name: Delete the Containers
  docker_container:
    name: myapp
    state: absent

- name: Remove Images
  docker_image:
    state: absent
    name: hello-world

- name: Start nginx Container
  docker_container:
    name: myapp
    image: nginx
```

Kick Start Terraform from jenkins

Install terraform Plugin







Lets Create a new Jenkins Job

```
pipeline{
    agent any

    parameters{
        choice(name: 'environment', choices: 'dev\nqa', description: 'Workspace/environment file to use for deployment')
        string(name: 'version', defaultValue: '', description: 'Version variable to pass to Terraform')
        booleanParam(name: 'autoApprove', defaultValue: false, description: 'Automatically run apply after generating plan?')
    }
    environment {
        myinfraaws   = credentials('awscredentials')

    }

    stages{

    stage("checkout"){
        steps{
            checkout([
                $class: 'GitSCM',
                branches: [[name: "origin/main"]],
                userRemoteConfigs:[[
                    url: "https://github.com/esak21/MySimpleApplication.git"
                ]]

            ])
        }
    }
    stage("plan"){
        steps{
            script{

                sh (
                    script: 'terraform init -input=false',
                    returnStatus: true
                ) == 0
                echo "Terrform inity was sucessful"
                sh """ terraform plan -input=false -out myappinfraplan"""
                sh """ terraform show -no-color myappinfraplan > myappinfraplan.txt """
            }
        }

    }
    stage("approval"){
        when {
            not{
                equals expected:true, actual: params.autoApprove
            }
        }
        steps{
            script {
                def plan = readFile 'myappinfraplan.txt'
                input message: "Do you want to apply the plan?",
                   parameters: [text(name: 'Plan', description: 'Please review the plan', defaultValue: plan)]
                }
            }
    }

    stage('Apply') {
        steps {
            script {
                sh (script:"terraform apply -input=false myappinfraplan ", returnStatus: true) == 0
                echo "Terraform Apply was sucessful"
            }
        }
    }

    stage("destroyit"){
        steps{

            script {
                def userInput = input(id: 'confirm', message: 'destroy AWS Resources?', parameters: [ [$class: 'BooleanParameterDefinition'

                sh(script: "terraform destroy --auto-approve", returnStatus: true) == 0
                echo  "AWS resources were destroyed sucessfully"
            }

        }
    }
```

```
}
post {
        always {
            archiveArtifacts artifacts: 'myappinfraplan.txt'
        }
    }


}
```

Jenkinsfile for running Terraform

Instantly share code, notes, and snippets. Jenkinsfile for running Terraform You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to

https://gist.github.com/gazoakley/87dcc16d28fd05acda4ba0a4be5ac387