**JENKINS**

Jenkins is an *open-source* automation tool written in Java programming language that allows continuous integration. Jenkins offers a straightforward way to set up a continuous integration or/and continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

The following are the main or most popular Jenkins use cases:

* **Continuous Integration**: With Jenkins pipelines, we can achieve CI for both applications and infrastructure as code.
* **Continuous Delivery**: You can set up well-defined and automated application delivery workflows with Jenkins pipelines.

Jenkins achieves CI (Continuous Integration) and CD (Continuous Deployment) with the help of plugins. Plugins are used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you must install the plugins for that tool.

**ADVANTAGES OF JENKINS:**

* It is an open-source tool.
* It is free of cost.
* It does not require additional installations or components. Means it is easy to install.
* Easily configurable.
* It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
* It is built in java and hence it is portable.
* It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows, or Linux.
* Easy support since its open source and widely used.
* Jenkins also supports cloud-based architecture so that we can deploy Jenkins in cloud-based platforms.

**Installation of Jenkins on RHEL/Centos/Amazon Linux:**

1. Download the latest LTS version of Jenkins repository from the official website.

$ sudo wget -O /etc/yum.repos.d/jenkins.repo \

<https://pkg.jenkins.io/redhat-stable/jenkins.repo>

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

1. Upgrade the existing packages in the instance.

$ sudo yum upgrade

1. Install openjdk-11 as a dependency for Jenkins package.

$ sudo yum whatprovides java-11

$ sudo yum install java-11-openjdk -y

1. Install Jenkins from the repository.

$ sudo yum install jenkins

1. Reload the daemon for effecting the changes.

$ sudo systemctl daemon-reload

**Starting and enabling Jenkins:**

1. Enable Jenkins to run on startup.

sudo systemctl enable jenkins

1. Start the Jenkins service.

sudo systemctl start jenkins

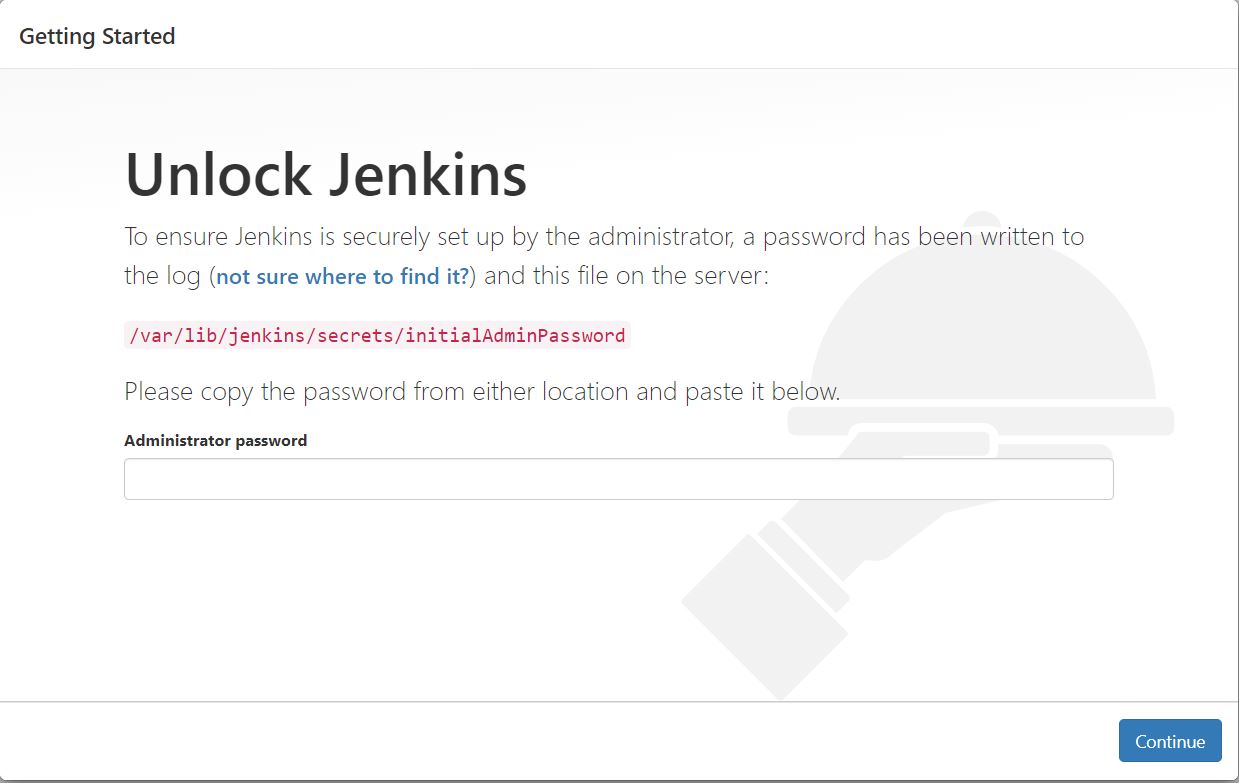
1. We can verify if the Jenkins service has started correctly using.

sudo systemctl status Jenkins

1. If the service is running/active, we can connect to Jenkins on a browser client by connecting to our VM’s IP address and port 8080.

If there is a firewall enabled on your instance and preventing the connection to port 8080, disable the firewalld or add exception to Jenkins for port 8080.

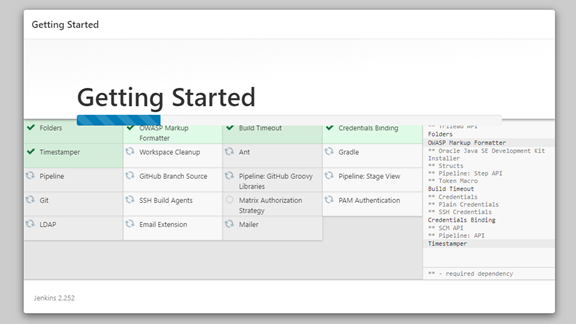
1. Enter url in browser http://<IP>:8080/
2. A login page will be opened and to unlock Jenkins give administrator password.



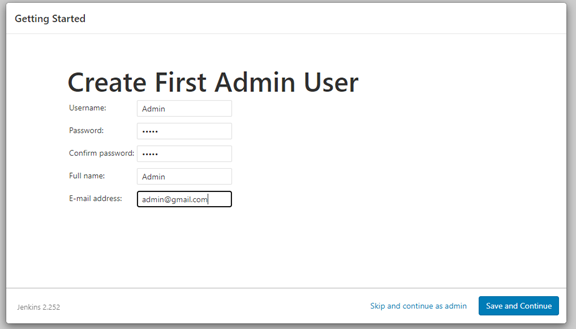
1. Add the password by copying from the above path.

$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

1. Click on Install the suggested plugins.



1. Add the Admin details and credentials to save it.



NOTE: If you forgot the password change no security tag “false” in /var/lib/jenkins/config.xml

**JENKIN JOBS:**

**PROJECT-1 : CREATING FIRST JOB IN JENKINS:**

1) Go to Jenkins Dashboard

2) Click on New Item

* Enter Item Name (Job Name)
* Select ***Free Style Project*** & Click OK
* Enter some description.
* Click on **'Build'** tab.
* Click on 'Add Build Step' and select 'Execute Shell'.

3) Enter below shell script

echo "Welcome to All"

touch raghu.txt

echo "Hello All, Welcome to Jenkins Sessions" >> raghu.txt

echo "Completed..!!"

4) Apply and Save

Note: With above steps we have created JENKINS Job

5) Click on 'Build Now' to start Job execution

6) Click on 'Build Number' and then click on 'Console Output' to see job execution details.

**Jenkins Home Directory in EC2:** /var/lib/jenkins/workspace/

**$ cd /var/lib/jenkins/workspace/**

7) Go to Jenkins home directory and check for the job name --> check the file created inside the job

**PROJECT-2: JENKINS JOB WITH GIT HUB REPO + MAVEN - INTEGERATION**

**PRE-REQUISITES: JAVA, MAVEN AND GIT CLIENT**

# Git installation In EC2 VM

$ sudo yum install git -y

**JDK INSTALLATION IN JENKINS USING PLUGINS:**

Jenkins Dashboard -> Manage Jenkins -> **Global Tools Configuration (Tools)** -> Add JDK -> Choose JDK 8 -> Configure Oracle account credentials to download JDK.

**MAVEN INSTALLATION IN JENKINS:**

Jenkins Dashboard -> Manage Jenkins --> **Global Tools Configuration** -> Add maven

Sample GitHub Repo For Practise

Git Hub Repo URL-1 : <https://github.com/javabyraghu/maven-web-app.git>

**Steps To Create Jenkins Job with GubHub + Maven**

1) Connect to EC2 instance in which Jenkins server got installed

2) Start Jenkins Server

3) Access Jenkins Server Dashboard and Login with your Jenkins credentials

4) Create Jenkins Job with Git Hub Repo

* New Item
* Enter Item Name (Job Name)
* Select 'Free Style Project' & Click OK
* Enter some description.
* **Go to "Source Code Management" Tab and Select "Git"**
* **Enter Project "GitHub Repository URL"**
* Add Your GitHub account credentials.
* Go to "Build tab"
* Click on Add Build Step and Select **'Invoke Top Level Maven Targets'**
* Select Maven and enter goals *'clean package'*.
* Click on Apply and Save

Note: With above steps we have created JENKINS Job

5) Click on 'Build Now' to start Job execution

6) Click on 'Build Number' and then click on 'Console Output' to see job execution details.

Jenkins Home Directory in EC2: **/var/lib/jenkins/workspace/**

Go to Jenkins workspace and then go to job folder then go to target folder there we see war file created.

**PROJECT-3: JENKINS JOB WITH GIT REPO + MAVEN + TOMCAT SERVER**

**Pre-requisite: Install Java and Tomcat – 9 in different EC2 Instance**

1) Go to Tomcat server folder and configure below users in "tomcat-users.xml" file (it will be available in tomcat / conf folder)

<role rolename="manager-gui" />

**<role rolename="manager-script" />**

<role rolename="admin-gui" />

<user username="tomcat" password="tomact" roles="manager-gui" />

<user username="admin" password="admin" roles="manager-gui,**manager-script**,admin-gui"/>

2) Go to Jenkins Dashboard -> Manage Jenkins --> Manage Plugins -> Go to Available Tab -> Search For

"***Deploy To Container***" Plugin -> Install without restart.

3) Create Jenkins Job

* New Item
* Enter Item Name (Job Name)
* Select Free Style Project & Click OK
* Enter some description.
* Go to "Source Code Management" Tab and Select "Git"
* Enter Project "GitHub Repository URL"
* Add Your GitHub account credentials.
* Go to "Build tab"
* Click on Add Build Step and Select 'Invoke Top Level Maven Targets'
* Select Maven and enter goals 'clean package'.
* Click on 'Post Build Action' and select 'Deploy war/ear to container' option.
* Give path of war file (You can give like this also: \*\*/\*.war )
* Enter Context Path (give project name)
* Click on 'Add Container' and select Tomcat version 9.x
* Add Tomcat server credentials (give the username & password which is having manager-script role)
* Enter Tomcat Server URL (http://ec2-vm-ip:tomcat-server-port)
* Click on Apply and Save

4) Run the job now using 'Build Now' option and see 'Console Output' of job.

5) Once Job executed successfully, go to tomcat server dashboard, and see application should be displayed.

6) Click on the application name (it should display our application)

**JENKINS JOBS WITH BUILD PARAMETERS:**

Parameters are dynamic inputs given to Jenkin Jobs while building Tasks. These are good approach to choose a String, Boolean or Choice Parameter types.

**Follow below steps:**

* Create New Item
* Enter Item Name & Select Free Style Project
* Select **"This Project is parameterized"** in *General Section*
* Select Choice Parameter
* Name: Branch Name
* Choices: Enter every branch name in nextline
* Update Branches to Build: **\*/${BranchName}**

**CREATING USERS IN JENKINS:**

We cannot share administrative credentials to everyone in organization. So, Jenkins allows us to create multiple users as per our requirements.

**Follow Below steps:**

* Go to *Manage Jenkins*
* Choose *Users*
* Click on *Create Users*

Note: By default, admin role will be available, and we can create custom role based on requirement

In Role we can configure what that Role assigned user can do in Jenkins

In Assign Roles we can add users to role.

**To Modify the Roles for Users**

* Go to *Manage Jenkins*
* Go to *Security*
* Go to *Authorization Option*
* Select *Matrix Based Security* and Add User then Select Roles
* Apply and Save

**INSTALLING JENKINS PLUGINS:**

One of the core features of Jenkins is the integration of Jenkins plugins. They help add functionalities over the core to give us more powerful tools regarding the project.

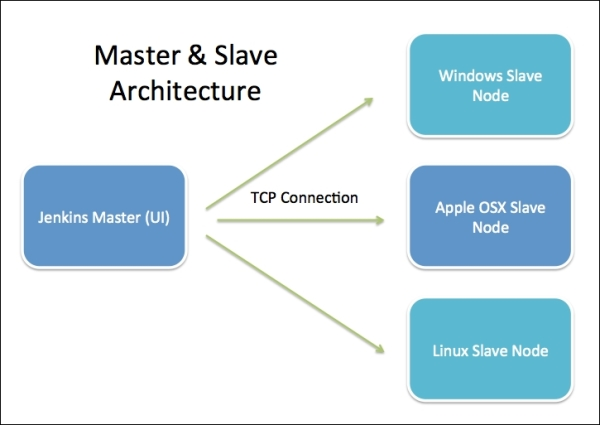
1. To list all the plugins supported by Jenkins, go to Jenkins plugins index.
2. Once logged in, head over to the ‘*Manage Jenkins’* tab on the left-hand side. This is where we would handle all the installed plugins, as well as add or remove new ones.
3. Under the *Plugin* tab, we can search for a plugin or see all the available plugins.
4. By selecting a plugin and clicking on install without restart, we can install the plugin and check its functionality sooner, rather than having to wait to restart Jenkins.
5. When we need to uninstall a plugin, head over to the installed tab, select the plugin that we would like to remove and click on uninstall. However, we must make sure to restart Jenkins for the changes to reflect.

**JENKINS ARCHITECTURE:**

Jenkins follows Master Node-Slave/Child Node architecture to manage distributed builds. In this architecture, slave/child node and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

1. Jenkins Master
2. Jenkins Node



**Jenkins Master:** The main server of Jenkins is the Jenkins Master. It is a web dashboard and by default it runs on port 8080. With the help of Dashboard, we can configure the jobs/projects, but the build takes place in Child Nodes/Slave. One Child Nodes slave is configured by default. We can add more nodes using IP address, username, and password.

**The server's job or master's job is to handle:**

* Scheduling build jobs.
* Dispatching builds to the nodes/slaves for the actual execution.
* Monitor the nodes/slaves (possibly taking them online and offline as required).
* Recording and presenting the build results.
* A Master/Server instance of Jenkins can also execute build jobs directly.

**Jenkins Node:** Jenkins Node is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular child node machine, or particular type of slave machine, or simply let the Jenkins to pick the next available slave/node.

The Jenkins Master and Slave architecture offers several benefits:

* **Scalability:** With Jenkins Agents, you can *distribute the workload* across multiple machines, enabling parallel execution of build jobs and accommodating a higher number of concurrent builds.
* **Specialized Environments:** Jenkins Agents can be configured with specific software, libraries, or hardware configurations to handle specialized build or deployment tasks. For example, you might have Agents with different operating systems or specific tools installed.
* **Security and Isolation:** By using Jenkins Agents, you can control access and permissions, allowing different teams or users to have their own dedicated Agents. This provides security and isolation of build environments.
* **Optimized Resource Usage:** With Jenkins Agents, you can utilize resources efficiently by running builds on idle machines or on machines with spare capacity. This helps optimize resource usage and reduces build queue times.

**Working with Jenkins Master Child Node Architecture:**

**Step-1: Create Jenkins Master**

Follow Above Jenkins Installation Process and check Jenkins up and running with URL : http://<Public-IP>:8080/

**Step-2: Create Jenkins Slave/Agent/Node**

* Create A EC2 instance
* Connect to EC2 using Mobaxterm
* Install Git client
* Install Java Software (JDK/JRE)
* Create one directory in /home/ec2-user (ex: projectnode) [MUST]

**Step-3: Configure Child Node in Jenkins Master Node**

* Go to Jenkins Dashboard
* Go to Manage Jenkins
* Go to Nodes
* Click on 'New Node' -> Enter Node Name -> Select Permanent Agent
* Enter Remote Root Directory (/home/ec2-user/slavenode )
* Enter Label name as “**node-1**”
* Select Launch Method as **'Launch Agent Via SSH'**
* Give Host as 'Slave **VM DNS URL**'
* Add Credentials (Select Kind as : ***SSH Username with private key*** )
* Enter Username as: ec2-user
* Select Private Key as Enter Directly and add private key.
* Open Git-Bash from your .pem file location and execute below command to get private key from pem file

$ cat <key-pair-file-name>.pem

It will display private key on Git Bash terminal (Just copy and paste in jenkins)

* Select Host Key Strategy as **'Manually Trusted Key Verification Strategy'**
* Click on Apply and Save (We can see configure slave)

**Step-4: Create Job using Jenkins Slave Machine**

* Go to Jenkins Server and Create Jenkins Job
* Under *General Section* of Job creation process, select ***"Restrict Where This Project Can Run"*** and enter Slave Node Label name and finish job creation ***(node-1)***
* Execute the Job using 'Build Now' option.
* Job will be executed on Child Node (Go to Job Console Output and verify execution details)

**JENKINS PIPELINE**

It is a powerful tool for defining and managing continuous integration and delivery (CI/CD) pipelines. It allows you to **script** your *build, test, and deployment* processes in a declarative manner, providing flexibility, extensibility, and reusability. Jenkins Pipeline can be written in either Groovy (a scripting language for the Java platform) or in a more recent addition, called the Jenkins Declarative Pipeline syntax.

**Jenkins pipelines:**

1) **Scripted**: Groovy Based, Older pipeline, needs Groovy Coding Basics.

2) **Declarative**: Recent Pipeline, Easy to configure, Works based on Jenkins **DSL**.

* Traditionally, Jenkins jobs were created using Jenkins UI called *Free Style jobs*.
* In Jenkins 2.0, Jenkins introduced a new way to create jobs using the technique called **Pipeline as Code (PaC).**
* In pipeline as code technique, jobs are created using a script file that contains the steps to be executed by the job.
* In Jenkins, that scripted file is called Jenkinsfile.

**1. SCRIPTED PIPELINE:**

Scripted Pipeline is the original and more flexible form of Jenkins Pipeline. It uses Groovy scripting language to define the pipeline steps. In a Scripted Pipeline, you have complete control over the flow of execution and can use conditional statements, loops, and custom logic. Here's an example of a Scripted Pipeline:

**node {**

stage('Clone Repository') {

echo 'Cloning Repository ..'

}

stage('Build Application') {

echo 'Build Application ..'

}

stage('Test Application') {

echo 'Testing application ..'

}

stage('Quality Check') {

echo 'Quality Checking ..'

}

stage('Upload to Artifactory') {

echo 'Uploading to Artifactory Repository ..'

}

**}**

**2. DECLARATIVE PIPELINE:**

Declarative Pipeline is a newer, more structured approach to defining pipelines in Jenkins. It provides a more opinionated and concise syntax for pipeline definition, aiming to simplify pipeline creation and readability. Declarative Pipelines are written in a more declarative manner and follow a predefined structure. Here's an example of a Declarative Pipeline:

pipeline {

agent any

stages {

stage('Clone') {

steps {

echo 'Cloning Application'

}

}

stage('Build') {

steps {

echo 'Compile and Build application'

}

}

stage('Test') {

steps {

echo 'Executing Test cases'

}

}

stage('Quality') {

steps {

echo 'Checking Code Quality'

}

}

stage('Artifactory') {

steps {

echo 'Uploading to Artifactory Repository'

}

}

stage('Deployment') {

steps {

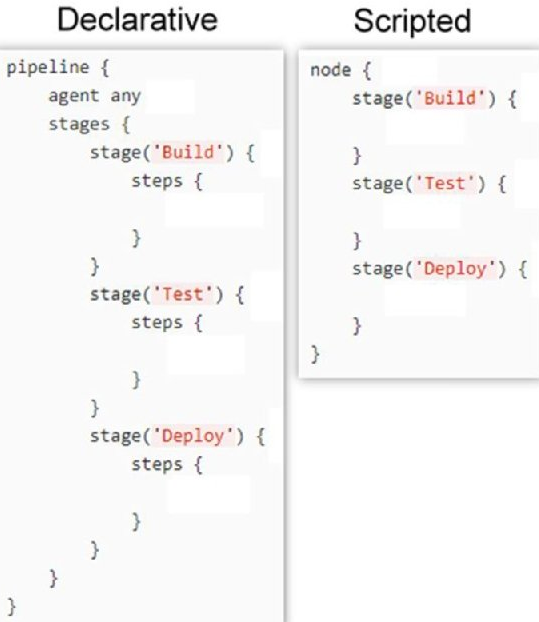
echo 'Deploying to Container..'

}

}

}

}



**PIPELINE PROJECTS**

**PROJECT-1: GITHUB + MAVEN**

pipeline {

agent any

environment {

PATH="${PATH}:/opt/maven/bin"

}

stages {

stage('Git Clone') {

steps {

git branch: 'main', credentialsId: 'GitHub-Credentials', url: 'https://github.com/javabyraghu/maven-web-app.git'

}

}

stage('Maven Build') {

steps {

sh "mvn clean package"

}

}

}

}

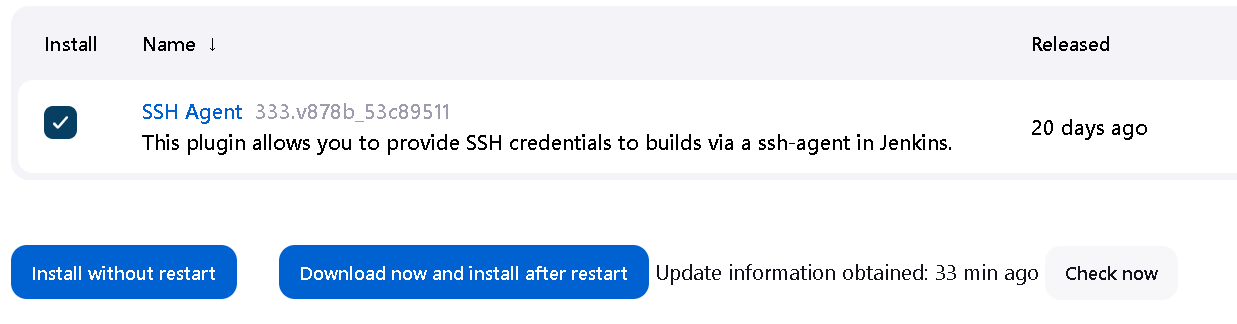
**PROJECT-2: GITHUB + MAVEN+TOMCAT**

**A.** Create EC2 Instance using below RAW Script (***Paste under Advanced Details > User data while creating EC2 Instance***)

[**https://raw.githubusercontent.com/javabyraghu/EC2UserScripts/main/AmazonLinux2023Tomcat-9.sh**](https://raw.githubusercontent.com/javabyraghu/EC2UserScripts/main/AmazonLinux2023Tomcat-9.sh)

**B.** Add SSH-Agent Plugin Jenkins Master

* Manage Jenkins
* Manage Plugins
* Choose Available Plugins
* Install SSH Agent Plugin without restart.



C. Go to Pipeline Syntax and choose **sshagent: SSH Agent.**

* Select kind as **SSH Username with Private key**.
* Enter username as ec2-user and paste private key file. [\_\_.pem file data]
* Click on generate code.

pipeline {

agent any

environment {

PATH="${PATH}:/opt/maven/bin"

}

stages {

stage('Git Clone') {

steps {

git branch: 'main', credentialsId: 'GitHub-Credentials', url: 'https://github.com/javabyraghu/maven-web-app.git'

}

}

stage('Maven Build') {

steps {

sh "mvn clean package"

}

}

stage('Deploy to tomcat') {

steps {

sshagent(['Tomcat-SSH-ID']) {

sh 'scp -o StrictHostKeyChecking=no target/01-maven-web-app.war ec2-user@3.108.63.142:/opt/tomcat/webapps'

}

}

}

}

}

**\*) Note:**

Execute below command inside tomcat machine if you face permission denied issue in build console:

$ sudo su –

$ cd /opt/tomcat

**$ chmod o+rwx webapps**

**PROJECT-3: GITHUB + MAVEN + SONARQUBE + NEXUS + TOMCAT**

**A.** Create **SonarQube** EC2 Instance using below RAW Script (***Paste under Advanced Details > User data while creating EC2 Instance***)

[**https://raw.githubusercontent.com/javabyraghu/EC2UserScripts/main/AWS2023Sonarqube7.sh**](https://raw.githubusercontent.com/javabyraghu/EC2UserScripts/main/AWS2023Sonarqube7.sh)

**B.** Create **Nexus** EC2 Instance using below RAW Script (***Paste Under Advanced Details > User data while creating EC2 Instance***)

[**https://raw.githubusercontent.com/javabyraghu/EC2UserScripts/main/AWS2023SonatypeNexus.sh**](https://raw.githubusercontent.com/javabyraghu/EC2UserScripts/main/AWS2023SonatypeNexus.sh)

**C.** Add SonarQube Scanner and Nexus Artifact Uploader Plugins Jenkins Master

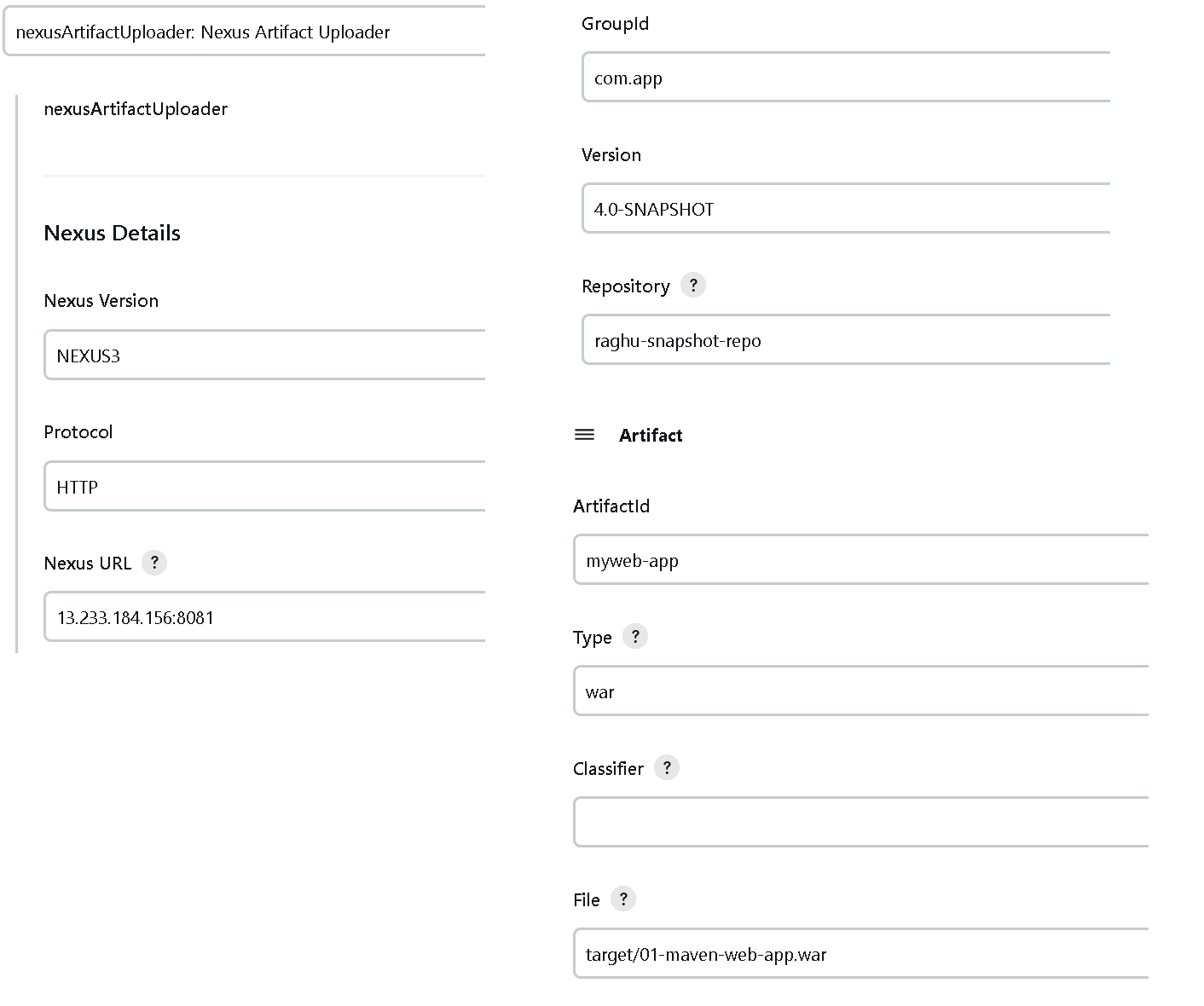
* Manage Jenkins
* Manage Plugins
* Choose Available Plugins
* Search and select “**SonarQube Scanner** and **Nexus Artifact Uploader**”.
* **Install SSH Agent Plugin** for tomcat.
* Install without restart.

D. Add SonarQube Server Details to Jenkins System.

* Manage Jenkins
* Go to System
* SonarQube Servers
* Add SonarQube Server
* Name: Sonar-Server-7.8
* Server URL: http://3.110.153.105:9000/ (Give your sonar server URL here)
* Add Sonar Server Token (**Token we should add as secret text**)
* Save it.

E. Add Nexus Details to Job

* Create one Repository (ex: 4.0-SNAPSHOT) Under Nexus Server.
* Choose nexusArtifactUploader under Jenkins Pipeline Syntax Generator (in Jenkins)
* Add Nexus Credentials as username and password.
* Enter Details as shown below.



pipeline {

agent any

environment {

PATH="${PATH}:/opt/maven/bin"

}

stages {

stage('Git Clone') {

steps {

git branch: 'main', credentialsId: 'GitHub-Credentials', url: 'https://github.com/javabyraghu/maven-web-app.git'

}

}

stage('Maven Build') {

steps {

sh "mvn clean package"

}

}

stage('SonarQube') {

steps {

withSonarQubeEnv('SonarQube-7.8') {

sh "mvn sonar:sonar"

}

}

}

stage('Upload to Nexus') {

steps {

nexusArtifactUploader artifacts: [[artifactId: 'myweb-app', classifier: '', file: 'target/01-maven-web-app.war', type: 'war']], credentialsId: 'Nexus-Login', groupId: 'com.app', nexusUrl: '13.233.184.156:8081', nexusVersion: 'nexus3', protocol: 'http', repository: 'raghu-snapshot-repo', version: '4.0-SNAPSHOT'

}

}

stage('Deploy to tomcat') {

steps {

sshagent(['Tomcat-SSH-ID']) {

sh 'scp -o **StrictHostKeyChecking=no** target/01-maven-web-app.war ec2-user@13.233.104.21:/opt/tomcat/webapps/'

}

}

}

}

}

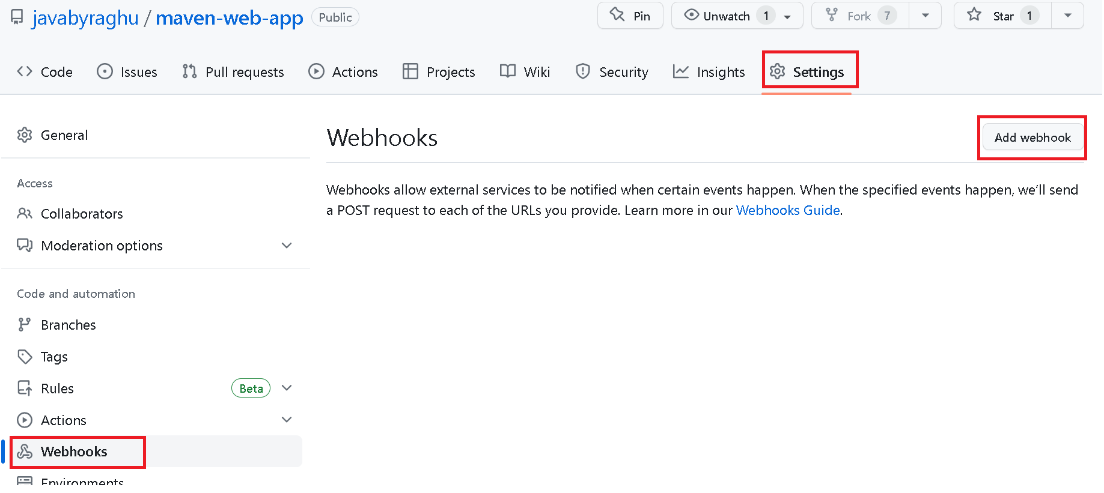
**INTEGRATING JENKINS WITH GIT USING WEBHOOKS**

**WEBHOOKS:**

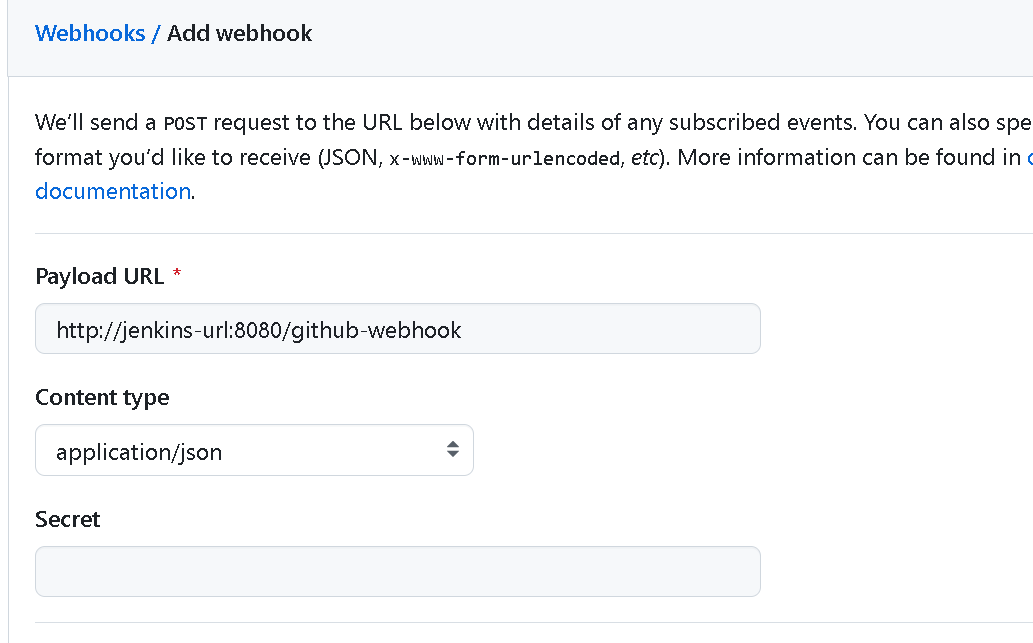
Webhooks allows external services to be notified when certain events happen. When the specified events happen, we’ll send POST request to each of the URLs you provide.

**Steps to be followed:**

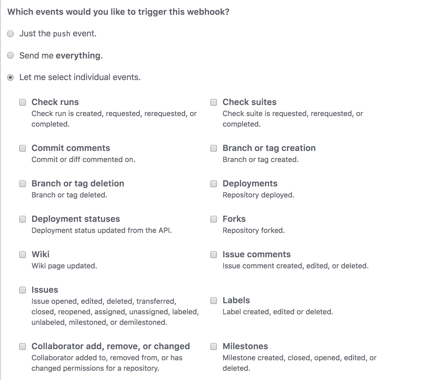
1. Go to GitHub repository and click on ‘Settings’.
2. Click on Webhooks and then click on ‘Add webhook’.

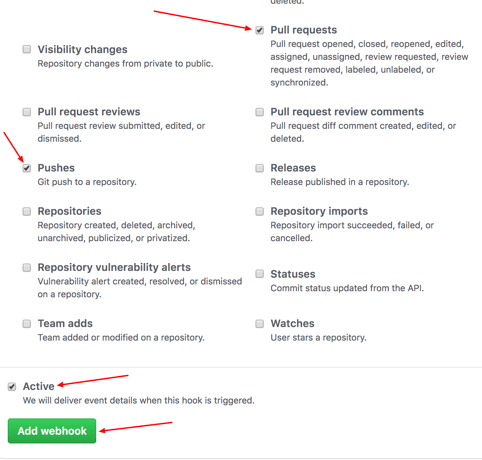


1. In the ‘Payload URL’, paste your Jenkins environment URL. At the end of this URL add /github-webhook/. In the ‘Content type’ select: ‘application/json’ and leave the ‘Secret field empty.



1. In this page ‘Which events would you like to trigger this webhook?’ choose ‘Let me select individual events. ‘Then, check ‘pull requests’ and ‘Pushes’. At the end of this option, make sure that the ‘active’ option is checked and click on ‘Add webhook’.





**CONFIGURING JENKINS:**

1. In Jenkins, Click on ‘New Item’ to create a new project
2. Give your project a name, then choose ‘Freestyle project’ and finally, click on ‘OK’.

A screenshot of a computer

Description automatically generated

1. Click on Source Code Management’ tab
2. Click on Git and paste your GitHub repository URL

A screenshot of a computer

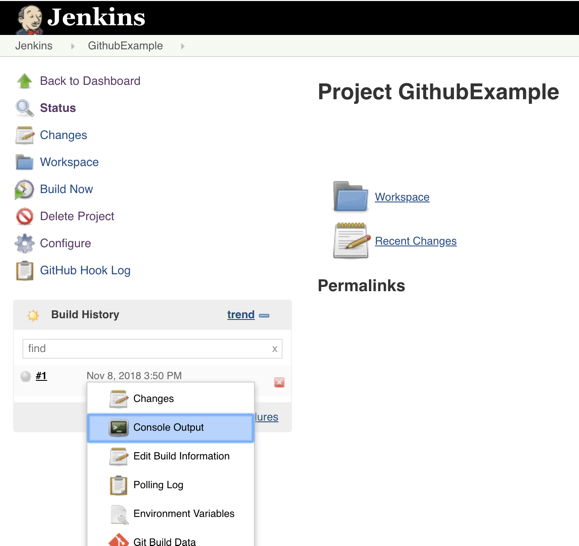
Description automatically generated with medium confidence

1. Click on the Build Triggers’ tab and then on the ‘GitHub hook trigger for GITScm polling’ or choose the trigger of your choice.

A screenshot of a computer

Description automatically generated with low confidence

1. Now Apply & save.
2. Your GitHub repository is integrated with your Jenkins project. With this Jenkins GitHub integration, you can now use any file found in the GitHub repository and trigger the Jenkins job to run with every code commit.
3. Go back to your GitHub repository, edit the file and commit changes. We will now see how Jenkins ran the script after the commit.
4. Go back to your Jenkins project and you’ll see that a new job was triggered automatically form the commit we made at the previous step. Click on the little arrow next to the job and choose ‘Console Output’.



1. You can see that Jenkins was able to pull the updated file and run it.

**FAQs:**

**Q) What is Jenkins pipeline?**

A) Jenkins pipeline represents the automation of the stages (build, test, release, deploy etc.) in DevOps life cycle. The Jenkins achieves this by using a set of plugins that support implementation of continuous integration and continuous delivery.

**Q) Explain declarative and scripted pipeline?**

A) **Scripted Pipeline**: This is the original pipeline syntax in Jenkins and is written in Groovy code. It allows for more complex logic and control flow than the simpler Declarative Pipeline syntax.

**Declarative Pipeline**: This is a newer pipeline syntax in Jenkins that is designed to be more user-friendly and easier to read than the Scripted Pipeline. It uses a YAML-based syntax to define stages, steps, and other aspects of the pipeline.

**Q) What are different stages in Jenkins CI/CD pipeline?**

A) Jenkinsfile contains the script for building the Jenkins pipeline. It is also known as pipeline as a code. There are two types – scripted pipeline and declarative pipeline. The stages in Jenkins declarative pipeline are build, test, deploy, checkout, release, verify, clean-up etc.

**Q) What are the Jenkins features?**

A) The main features of Jenkins are:

**Easy installation:** The java-based program is autonomous and platform agnostic.

**Straightforward to configure:** The user-friendly web interface makes configuration an easy process.

**Customizable with massive plugin availability:** Currently, there are over 1700 available plugins to extend Jenkins' functionality. For any missing features, import additional external or custom-made plugins.

**Open-source:** Jenkins is completely open-source and free to use, making it a cheap technology to implement.

**Q) How to take backup of my jobs in Jenkins?**

A) Jenkins provides various ways to back up your jobs. Here are some methods you can use:

**1. Backup plugin:** Jenkins has a Backup plugin that can be used to back up your entire Jenkins installation, including jobs, configurations, and other data. You can install this plugin from the Jenkins plugin manager, and then use it to schedule automatic backups or create manual backups on demand.

**2. Copy job directory:** Another way to take backup of jobs is to simply copy the job directory from the Jenkins installation directory. You can find the job directory under $JENKINS\_HOME/jobs. Simply copy the directory to a backup location on a regular basis.

**Q) What is workspace in Jenkins?**

A) In Jenkins, a workspace is a directory on a Jenkins agent where a job's workspace is created to perform build and test operations. When a job is executed, Jenkins creates a new workspace for that job on the node where the job is run. This workspace is a temporary directory where the job's source code, build artifacts, and other files are stored during the build process.

**Q) Where the Jenkins configuration and Build information are saved?**

A) The default home directory for Jenkins is /var/lib/jenkins

Where all the Jenkins all configurations such as jobs, users, Plugins information are saved.

Build information will be save in a Build directory -/var/lib/Jenkins/jobs/<jobname>

Build steps will be triggered and SCM steps also are executed in Workspace/root directory –var/lib/Jenkins/workspace/<jobname>

**Q) What is POM in Jenkins?**

A) A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration information for Maven such as construction directory, source directory, test source directory, dependency, Goals, plugins etc.. It contains default values for most projects.

**Q) What is mean by Jenkins job?**

A) A Jenkins job is a set of instructions that defines how Jenkins should perform a specific task. Jobs can be created and configured in Jenkins to perform a variety of tasks, such as compiling code, running tests and deploying software to production.

**Q) What are the different stages in Jenkins pipeline?**

A) The below are common stages in Jenkins pipeline.

1. Checkout: This stage checks out the source code from a version control system (such as Git) and prepares it for build.

2. Build: This stage compiles the source code and creates an executable artifact, such as a JAR or WAR file.

3. Test: This stage runs automated tests to verify that the software is functioning correctly and meets the specified requirements.

4. Deploy: This stage deploys the artifact to a test or production environment.

5. Verify: This stage runs additional tests to verify that the deployed software is working correctly in the target environment.

6. Release: This stage marks the artifact as ready for release and makes it available for deployment to production.

7. Clean-up: This stage cleans up any temporary files or resources created during the pipeline execution.

**Q) If we forgot Jenkins Password, then how to recover it?**

1. Go to /var/lib/jenkins/
2. Open : config.xml file
3. Set Value for useSecurity as false
4. ex: <useSecurity>false</useSecurity>
5. Re-start jenkins and try to access