**Jenkins Installation: -**

**Jenkins:**

* It is an open-source tool
* Jenkins launched in the year 2011
* Initially it was named as Hudson
* Basically, it is a web application developed by Java

Jenkins Architecture:

Diagram

Description automatically generated

**Plugin Management :**

It gives options to connect a specific tools for our application (Like Git/maven etc)

**Node Configuration:**

To tell jenkins which nodes to connect (slaves)

**Tools Configuration:**

It gives the tools location in jenkins to execute the jobs (URL details). (Git, maven, sonarqube, docker, kubernetes, nexus etc)

**System configuration: (Master)**

To run jenkins we need some set of configuration so that can be defined in System configurations  (environmental variables)

**Job Configuration:**

The details of jobs we are going to execute that can be defined in job configurations.

Ex: patches, software release, agents schedulers

**Practical Implementation:**

To demonstrate Jenkins, let us create 2 Amazon EC2 machines. One will be a Server(Master) and another will become a Client( Slave).

Make sure the port number 8080 is enabled in the Master node. Since 8080 is the default port of jenkins.

Below the steps to install the Jenkins in Amazon Ec2. We can refer to the Jenkins website for the steps to install theJenkins in different flavours of Linux.

 # amazon-linux-extras install epel -y

 # yum update -y

 # wget -O /etc/yum.repos.d/jenkins.repo     https://pkg.jenkins.io/redhat-stable/jenkins.repo

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

 # yum upgrade

 # yum install jenkins java-1.8.0-openjdk-devel -y

 # systemctl start jenkins

 # systemctl enable jenkins

Now, we can launch the Jenkins server using the public  IP of the Master instance along with port number 8080

For Example :<http://54.190.163.97:8080/>

We can view the following Jenkins home page for the first time.

Graphical user interface, text, application, email

Description automatically generated

We can get the default admin password for first time use from the path **“/var/lib/jenkins/secrets/initialAdminPassword”**

After providing the first time password and proceed, we can view the below screen.

**Graphical user interface, text

Description automatically generated**

We can choose and “install the suggested plugins”

Below the snap of plugin installation

Graphical user interface, table

Description automatically generated with medium confidence

Once completed this, the system will ask us to create the first Admin user.

Graphical user interface, application

Description automatically generated

Once completed the above details, we can save and continue. In the next step, we can choose Save and finish.

After successful completion, we can view the default landing page as below.

Graphical user interface, application

Description automatically generated

We can view the installed plugins under Manage Jenkins→ Manage Plugins → Installed .

We can install new plugins from the available tab.

Graphical user interface, text, application, email

Description automatically generated

We can view the Global tools configuration using Manage Jenkins→ Global Tool configuration. We can provide the configuration details for Git, Maven etc there.

Graphical user interface, text, application, email

Description automatically generated

We can view the Manage Nodes using Manage Jenkins→ Manage Nodes and clouds option.

Graphical user interface, text, application

Description automatically generated

Similarly we can access the main system configuration using Manage Jenkins→ Configure Systems option.

**Graphical user interface, application, Teams

Description automatically generated**

After completing the required setup, now we are ready to create our required jobs. To create the new job, we have to choose the new item option and choose from the available items. Mostly we will use freestyle projects or pipelines.

Graphical user interface, text, application, email

Description automatically generated

To process, we can type the name and then choose the category. We can view all these things in detail later.

Before that, we can set up our slave node. Connect to our Slave node and do the below thins. Initially, we need to install the java (jdk) in slave machine in order to enable the communication between Master and Slave

# amazon-linux-extras install epel -y

# yum update -y

# yum install java-1.8.0-openjdk-devel -

To handshake the Jenkins Master and slave nodes, we need to some setup in our Master and slave nodes.

**Master- Slave setup:**

First, connect to our Master node and create a new user “jenkinsadmin” and set a password. Do the same steps in Slave as well.

# useradd jenkinsadmin

# passwd jenkinsadmin

Go to visudo file in both master and slave and update the newly created user and mention to not asking password

## Allow root to run any commands anywhere

root    ALL=(ALL)       ALL

jenkinsadmin ALL=(ALL) NOPASSWD:ALL

Go to the sshd\_config file in both master and slave and enable the password authentication with “**yes**” and restart the sshd service

# vi /etc/ssh/sshd\_config

PasswordAuthentication yes

# systemctl restart sshd

Now , ready to switch to “**jenkinsadmin**” user.

# su - jenkinsadmin

Create the ssh key in Master node

# ssh-keygen

Go to the path “**/home/jenkinsadmin/.ssh**” and we can view the below generated files

id\_rsa  id\_rsa.pub

Now, we can copy this **“id\_rsa.pub**” key to Slave machine.

# ssh-copy-id -i id\_rsa.pub jenkinsadmin@<private\_ip of slave machine>

Note: We can use “ip a” command to get the private ip from slave node.

Now, we can login to the slave machine from Master

# ssh jenkinsadmin@<private\_ip of slave machine>

Lets setup the slave from Jenkins Dashboard. Now, again go to the Jenkins Dashboard and go to the Create Node option.

Click on new node link and provide some node name for Slave and choose Permanent Agent radio button and click on Create.

Graphical user interface, text, application

Description automatically generated

In the next screen, we need to update below configuration details for Slave.

Number of executors- How many jobs can run at a time

Remote root directory - provide some workspace path for slave( where should run).  create some directory in slave machine. ( For ex: /tmp/jobs)

Labels- Labels used to grouping

Launch method- we can choose “Launch Agents via ssh”

Graphical user interface, application

Description automatically generated

Host - provide the private ip of Slave machine

Credentials- Add Jenkins credentials. Choose SSH username with private key. Choose ssh username with private key option in kind. Provide our jenkins username “jenkinsadmin”. Copy our master key “**id\_rsa”** from master node and put it in the private key field.

Graphical user interface, text, application, email

Description automatically generated

Host key verification strategy - choose “Manually trusted key verification strategy”

Now, we can save the configuration

It will validate and added the Slave node as below.

Graphical user interface

Description automatically generated

**Create and Run our first Job:**

* Go to new item
* Provide a name and choose freestyle project option
* We can choose our label mentioned in our Slave node in the option “restrict where this project can be run option”. This helps to run our job only in slave node.

Graphical user interface, application

Description automatically generated

* Choose the “Execute Shell” option under build
* Provide some shell bin shell command to test

**#! /bin/bash**

**mkdir /tmp/testfolder**

**echo “hi, this is slave node”**

* Now we can Save it and choose Build Now option to build the job
* Once the build is successfully completed , we can see the message in the build history.

Graphical user interface, text, application

Description automatically generated

* We can click on the build log and view the console output to view more details. We can view the success message along with the echo command output. Also, we can view the newly created “**testfolder**” in the slave machine.

Text

Description automatically generated

   Below is the summary of activities as part of creating a new job and running it.

* create a freestyle job for slave1
* Restrict the job for the slave group
* using shell to execute the commands
* use “build now” to build it

**Jenkins features:**

1. When we tell jenkins to run a task on a particular server , if that server is not available it may execute to other machines in the group but once the server is back online it will push the codes
2. If we didn’t specify any slave nodes for the job build jenkins by default it will execute in master node
3. CRON jobs can be replaced using Jenkins
4. It runs multiple jobs and captures all the console output as a job history
5. We can write a pipeline for all the stages

* commit
* build
* test
* deploy

**Jenkins Pipeline:-**

Series of task which we are going to do one after the other to achieve end result

build

test

git - repo

docker image build

**Pipeline:**

DSL - Domain specific language developed on top of Groovy

Types of Pipeline:

* **Scripted pipeline**  - mostly it is groovy based syntaxes
* **Declarative pipeline**  - Provides richer syntactical features over scripted pipeline

-It is designed to make writing the pipeline code easier

Graphical user interface, application

Description automatically generated

**Declarative pipeline syntax:**

pipeline {

}

One jenkins file is equal to multiple freestyle jobs. We can combine multiple freestyle jobs and cover them in a single pipeline.

We can easily streamline the upstream and downstream jobs.

**syntax**:

pipeline {

     agent {}    //global agent

        stages{

                   stage(‘stage1’) {

                     steps{build}

                                               }

                    stage(‘stage2’){

                       steps{test}

                                               }

                      }

                  }

**Practical pipeline scripts:**

To create a pipeline script,  choose pipeline item type and provide the name. Need to write the scripts in the pipeline section.

**Graphical user interface, text, application

Description automatically generated**

**Text

Description automatically generated**

**Basic pipeline script:**

pipeline {

    agent { label 'sonar-slave' }

    stages {

       stage ("build") {

          steps {

             echo "Build stage"

                }

              }

       stage ('Deploy') {

           steps {

             echo "Deploy stage"

                 }

               }

             }

          }

Above the very basic pipeline script which runs in the agent “sonar-slave” and shows the output as below after running the same.

The stage view and console output clearly shows the list of stages and time taken to run the stage.

Graphical user interface, chart, treemap chart

Description automatically generated

Graphical user interface, table

Description automatically generated with medium confidence

**Running jobs on specific agent or node:(controlling the agent in the stage level)**

pipeline {

    agent none

    stages {

       stage ("build") {

          agent { label 'group\_java’ }

          steps {

             sh "echo 'Build stage'"

                }

              }

       stage ('Deploy') {

           agent { label 'group\_slave\_sonar' }

           steps {

             sh "echo 'Deploy stage'"

                 }

               }

             }

          }

As per the console , we can view the different stages of the job running on different nodes.

Graphical user interface, text, application, email

Description automatically generated

**Customising the folder: ( Workspace)**

pipeline {

    agent none

    stages {

       stage ("build") {

          agent {

             node {

                 label 'group\_java'

                 customWorkspace '/tmp/jenkins'

                  }

                }

          steps {

             sh "echo 'Build stage'"

sh "mkdir testdir"

                }

              }

       stage ('Deploy') {

           agent { label 'group\_java' }

           steps {

             sh "echo 'Deploy stage'"

                 }

               }

             }

          }

As per the console, we can view that the job is running in the custom workspace and create the directory in that space as per our pipeline steps.

Graphical user interface, text, application, email

Description automatically generated

Environment block:( Declaring environment variable in Global level)

With the help of this feature, we can set some environment variables at the global level and can be used in any stage.

pipeline {

    agent { label 'slave' }

    environment {

       JAVA = 'narayananjob'

                }

    stages {

       stage ("Stage1") {

          steps {

             sh "echo 'java home $JAVA'"

                }

              }

       stage ('Stage2') {

           agent { label 'group\_java' }

           steps {

             echo env.JAVA

                 }

               }

             }

          }



Environment block:( Declaring environment in a stage level)

In this example, we declare an environment variable both globally and stage level.

The scope of the environment variable declared inside the stage is restricted within the stage whereas the global environment variable applies across all the stages.

pipeline {

    agent { label 'group\_java' }

    environment {

       JAVA = 'GLOBAL'

                }

    stages {

       stage ("Stage1") {

           environment {

                JAVA = 'Local'

                      }

          steps {

             sh "echo 'java home $JAVA'"

                }

              }

       stage ('Stage2') {

           agent { label 'group\_java' }

           steps {

             echo env.JAVA

                 }

               }

             }

          }

Graphical user interface, text, application, email

Description automatically generated

Passing the Parameters in Pipeline:

Earlier, we implemented the parameterization in the project. Now, we are going to achieve the same from the pipeline This will be useful for passing the different parameters during run time. For example, we can pass the different environment details, branch details etc..

In the below example, we are using and demonstrating different types of parameters like String, text, boolean, choice, password, file..

pipeline {

      agent any

      parameters {

          string(name: 'BRANCH', defaultValue: 'Task1', description: 'Branch to push?')

  text(name: 'RELEASE', defaultValue: '', description: 'Enter about version')

  booleanParam(name: 'TOGGLE', defaultValue: true, description: 'TOGGLE this value')

  choice(name: 'CHOICE', choices: ['One', 'Two', 'Three'], description: 'Select the version')

  password(name: 'PASSWORD', defaultValue: 'SECRET', description: 'Enter the password')

  file(name: "file.properties", description: 'Choose a file to upload')

      }

      stages {

             stage ('Feature\_release')   {

        steps {

    echo "Branchname ${params.BRANCH}"

echo "Version: ${params.RELEASE}"

echo "Toggle: ${params.TOGGLE}"

echo "Choice: ${params.CHOICE}"

echo "Password: ${params.PASSWORD}"

}

}

             }

}

Graphical user interface, text, application, email

Description automatically generated

**Graphical user interface, text, application, email

Description automatically generated**

**Sample Pipeline Program: -**

<https://github.com/hakdogan/jenkins-pipeline>

<https://medium.com/hackernoon/continuous-delivery-of-react-app-with-jenkins-and-docker-8a1ae1511b86>

<https://www.yatis.io/how-to-setup-ci-cd-for-react-using-jenkins-and-docker-on-aws-s3/>

<https://www.clickittech.com/devops/ci-cd-docker/>