**Jenkins Interview Questions:**

**which tool have you used for implement CI?CD?**

Jenkins

**Any alternative tool do you know for CI/CD?**

bamboo/udeploy/teamcity/tfs..

**What is continuous integration?**

(Get the code --> compile --> test --> package --> push package to artifactory) --> Build failure notifications

Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied.

**What type of jobs have you configured in jenkins?**

Freestyle software project

pipeline

**what are the types of jobs available in jenkins?**

**Freestyle software project**

Freestyle build jobs are general-purpose build jobs, which provides a maximum of flexibility. This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**External Job**

This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

**Multiconfiguration job**

The “multiconfiguration project” (also referred to as a “matrix project”) lets you run the same build job in many different configurations. This powerful feature can be useful for testing an application in many different environments, with different databases, or even on different build machines

**Pipeline projects for larger projects-**

This is a new type of Jenkins project that is applicable when you need to setup continuous delivery pipeline or say you want to define the deployment pipeline as code. Let us take an example wherein you need to orchestrate all the stages in delivering an application as one single pipeline project-

***App build -> Unit testing -> Deliver -> Testing -> Deploy***

Each stage is a job that executes the command and calls in the other stage and it can even pass arguments. These pipelines can be built either through a web interface or through a scripted file (JenkinsFile) which can be checked in a source control file. The advantage of scripting a pipeline is that you can create pipelines automatically for all branches and execute requests all through a single file – JenkinsFile. And such a file can be reviewed, audited and edited by multiple users.

**An excellent example of pipeline projects are Infrastructure as a Code (IaaC).**

**what is the difference between freestyle and pipeline?**

**Freestyle software project**

Freestyle build jobs are general-purpose build jobs, which provides a maximum of flexibility. This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**pipeline**

Jenkins Pipeline is a stack of Jenkins plugins and other tools which helps implementing and continuous integration and delivery pipelines. In Jenkins, Pipelines are written in DSL code which implements this continuous integration and delivery pipeline jobs.

**what is declarative pipeline?**

The Declarative pipeline is a new feature that is added to create the pipeline. This is basically written in a Jenkinsfile which can be stored into a [source code management system](https://digitalvarys.com/introduction-to-version-control-systems/) such as Git. Declarative pipelines is an ideal solution for the simple continuous delivery pipeline as it has very limited and pre-defined structure.

Lets see the structure and syntax of the Declarative pipeline.

The Agent is where the whole pipeline runs. Example, Docker. The Agent has following parameters:

* **any** – Which mean the whole pipeline will run on any available agent.
* **none** – Which mean all the stages under the block will have to declared with agent separately.
* **label** –  this is just a label for the Jenkins environment
* **docker** –  this is to run the pipeline in Docker environment.

The Declarative pipeline code will looks like this:

**pipeline {**

**agent { label 'node-1' }**

**stages {**

**stage('Source') {**

**steps {**

**git 'https://github.com/digitalvarys/jenkins-tutorials.git''**

**}**

**}**

**stage('Compile') {**

**tools {**

**gradle 'gradle4'**

**}**

**steps {**

**sh 'gradle clean compileJava test'**

**}**

**}**

**}**

**}**

**what is script based pipeline?**

The scripted pipeline is a traditional way of writing the Jenkins pipeline as code. Ideally, Scripted pipeline is written in Jenkins file on web UI of Jenkins. Unlike Declarative pipeline, the scripted pipeline strictly uses groovy based syntax. Since this, The scripted pipeline provides huge control over the script and can manipulate the flow of script extensively. This helps developers to develop advance and complex pipeline as code.

Let us see the structure and syntax of the scripted pipeline

**Node Block:**

Node is the part of the Jenkins architecture where Node or agent node will run the part of the workload of the jobs and master node will handle the configuration of the job. So this will be defined in the first place as

node {

}

**Stage Block:**

Stage block can be a single stage or multiple as the task goes. And it may have common stages like

* Cloning the code from SCM
* Building the project
* Running the Unit Test cases
* Deploying the code
* Other functional and performance tests.

So the stages can be written as mentioned below:

stage {

}

So, Together, the scripted pipeline can be written as mentioned below.

**stage {**

**}**

**So, Together, the scripted pipeline can be written as mentioned below.**

**node ('node-1') {**

**stage('Source') {**

**git 'https://github.com/digitalvarys/jenkins-tutorials.git''**

**}**

**stage('Compile') {**

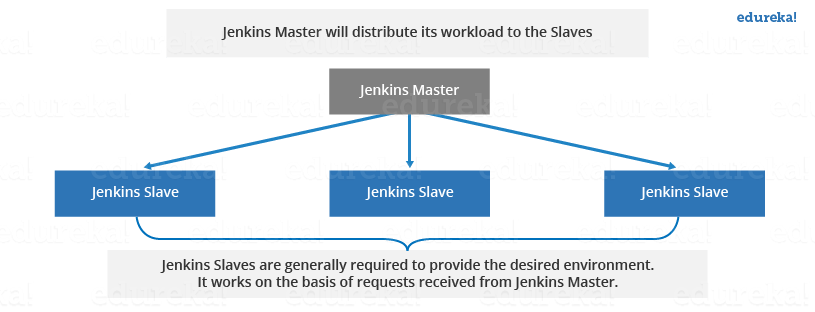
**def gradle\_home = tool 'gradle4'**

**sh "'${gradle\_home}/bin/gradle' clean compileJava test"**

**}**

**}**

**what is master/slave architecture?**



**what is the use of master/slave?**

**Jenkins Master**

Your main Jenkins server is the Master. The Master’s job is to handle:

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

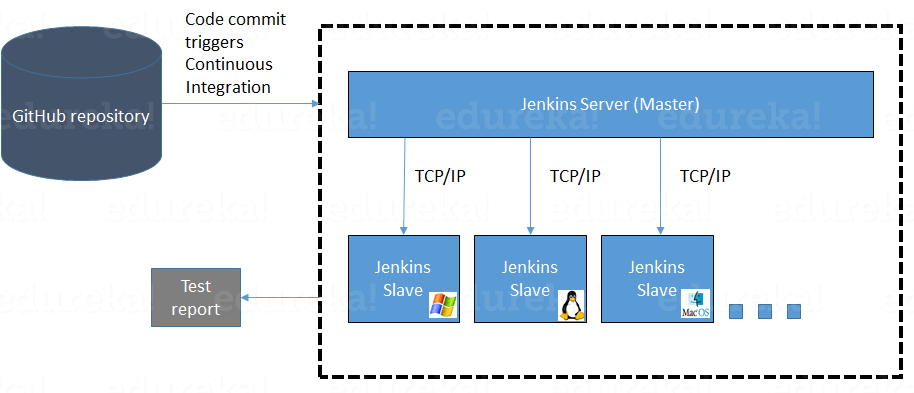
## **Jenkins Slave**

A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

* It hears requests from the Jenkins Master instance.
* Slaves can run on a variety of operating systems.
* The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
* You can configure a project to always run on a particular Slave machine, or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

Now let us look at an example in which Jenkins is used for testing in different environments **like: Ubuntu, MAC, Windows etc.**

The diagram below represents the same:



**How many slaves we can connect?**

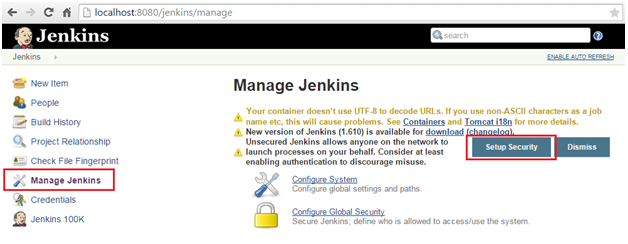
53 slaves in total. 4 physical slaves: 1 mac mini, 3 other 64 cores 128GB ram 2TB ssd disk.

**How many ways we can provide security for your jenkins server?**

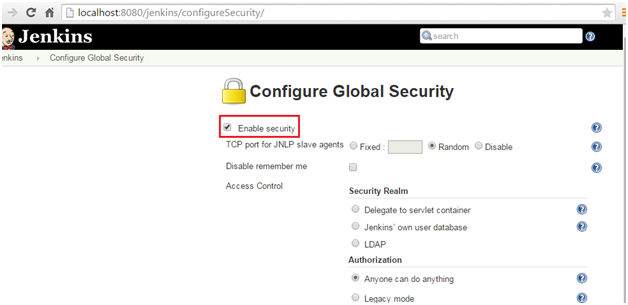
The very basic thing needs to be understood here is that to secure the Jenkins because it is open. Anyone can make use of the URL for accessing the Jenkins and perform the tasks of all kinds available in Jenkins. Hence, this needs to be secured. As a best practice, it is recommended to always secure Jenkins and then configure the global security. The best way is to use Jenkins by configuring to our own local database.

Below are the steps to be followed in order to secure Jenkins:

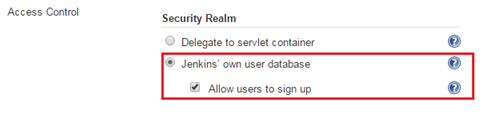
1. Deploy the Jenkins.war and start the server.
2. Open the Jenkins home page and click on Manage Jenkins.
3. In the Manage Jenkins page, click on Setup Security button.



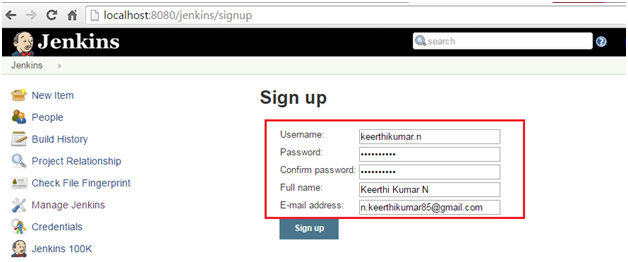
1. In the next page, select the enable security check box.



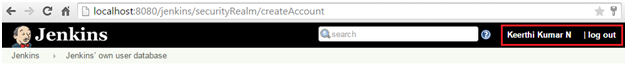
1. Here, the very first thing to be done is to set the security realm. The easiest way to do this is to have Jenkins with our own database. To achieve this, select the option Jenkins own user database. Also, ensure that Allow users to sign up checkbox is also checked. Save the configuration.



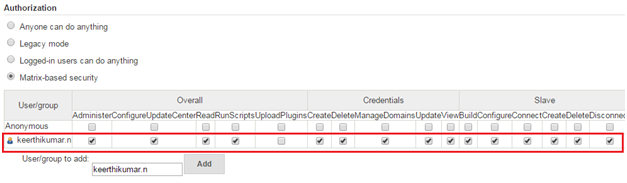
1. Now a link Sign up will be available. Click on the same and fill the form to sign up. Once successful, log in with the account created.



You can see the details in the Navbar, once you are logged in.



1. Now click on the Manage Jenkins & select Configure Global Security. Under the security realm section, uncheck the option Allow users to sign up. This will ensure that no new users can be created with your permission.
2. Now, we need to configure the authentication for the accounts. The 2 best options preferred are Matrix-based security & Project-base project authorization strategy. This enables you to set per user for the actions which they can perform. Here, I have considered Matrix-based security



1. Save the form. Logout and login again.
2. A login page will be displayed and login with the created account.

