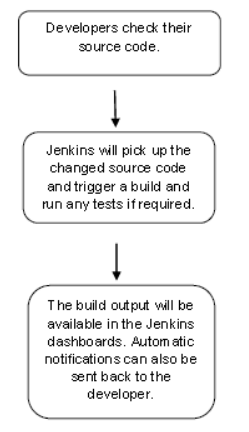
Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. You can integrate Jenkins with a number of testing and deployment technologies. In this tutorial, we would explain how you can use Jenkins to build and test your software projects continuously.

Jenkins is a software that allows continuous integration. Jenkins will be installed on a server where the central build will take place.

The following flowchart demonstrates a very simple workflow of how Jenkins works.



**What is Continuous Integration?**

Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals. This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle. Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.3

Continuous Integration, in its simplest form, involves a tool that monitors your version control system for changes. Whenever a change is detected, this tool automatically compiles and tests your application. If something goes wrong, the tool immediately notifies the developers so that they can fix the issue immediately.

Continuous Integration can also help you keep tabs on the health of your code base, automatically monitoring code quality and code coverage metrics. CI can also act as a communication tool, publishing a clear picture of the current state of development efforts. And it can simplify and accelerate delivery by helping you automate the deployment process, letting you deploy the latest version of your application either automatically or as a one-click process.

Continuous Integration helps you get your software into the hands of the testers and the end users faster, more reliably, and with less effort. **This idea of automated deployment is important**. Indeed, if you take automating the deployment process to its logical conclusion, you could push every build that passes the necessary automated tests into production. **The practice of automatically deploying every successful build directly into production is generally known as Continuous Deployment.**

**However, a pure Continuous Deployment approach is not always appropriate for everyone.** For example, many users would not appreciate new versions falling into their laps several times a week, and prefer a more predictable (and transparent) release cycle. Commercial and marketing considerations might also play a role in when a new release should actually be deployed.

The **notion of Continuous Delivery is a slight variation on the idea of Continuous Deployment** that takes into account these considerations. With Continuous Delivery, any and every successful build that has passed all the relevant automated tests and quality gates can potentially be deployed into production via a fully automated one-click process, and be in the hands of the end-user within minutes. However, the process is not automatic: it is the business, rather than IT, that decides the best time to deliver the latest changes.

Jenkins runs on Java and comes as a WAR file — a collection of related content comprising a web application and intended to be run on a server.

**Basic installation:**

For jenkins to work, we need to install some pre requisites.

1] **setup the java environment.** # for this refer to the java install doc in drive

2] setup up github account, install github on jenkin machine for **CI** part.

Then run the below cmds to install jenkins (REDHAT / CENTOS).

**# wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo**

**# ls -l /etc/yum.repos.d/**

**# cat /etc/yum.repos.d/jenkins.repo**

**# rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key**

**# cat /etc/yum.repos.d/jenkins.repo**

**# yum install jenkins**

**# systemctl start jenkins.service**

**# systemctl enable jenkins.service**

**# systemctl status jenkins.service**

**For Ubuntu.**

**# wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -**

**# sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'**

**# sudo apt-get update**

**# sudo apt-get install jenkins**

Now try to access your jenkin server at port 8080 of the node ip address.

On the very 1st  page u would be asked to provide admin password for setup, which can be found on

**# cat /var/lib/jenkins/secrets/initialAdminPassword**

After this setup the local user to run jenkin jobs.

The memory usage of a Continuous Integration server is best described as spiky—Jenkins will be creating additional JVMs as required for its build jobs, and these need memory. Another useful approach is to set up multiple build machines. Jenkins makes it quite easy to set up “slaves” on other machines that can be used to run additional build jobs. The slaves remain inactive unti a new build job is requested—then the main Jenkins installation dispatches the build job to the slave and reports on the results. This is a great way to absorb sudden spikes of build activity, for example just before a major release of your principal product. It is also a useful strategy if certain heavy-weight builds tend to “hog” the main build server—just put them on their own dedicated build agent!

If you are installing Jenkins on a Linux or Unix build server, it is a good idea to create a special user (and user group) for Jenkins. This makes it easier to monitor at a glance the system resources being used by the Jenkins builds, and to troubleshoot problematic builds in real conditions.

$ **sudo groupadd build**

$ **sudo useradd --create-home --shell /bin/bash --groups build jenkins**

In most environments, you will need to configure Java correctly for this user. For example, you can do this by defining the JAVA\_HOME and PATH variables in the .bashrc file, as shown here

**export JAVA\_HOME=/usr/local/java/jdk1.6.0**

**export PATH=$JAVA\_HOME/bin:$PATH**

Jenkins keeps all its important data in a special separate directory called the Jenkins home directory. Here, Jenkins stores information about your build server configuration, your build jobs, build artifacts, user accounts, and other useful information, as well as any plugins you may have installed. The Jenkins home directory format is backward compatible across versions, so you can freely update or reinstall your Jenkins executable without affecting your Jenkins home directory.

Continuous Integration servers use a lot of memory. This is the nature of the beast—builds will consume memory, and multiple builds being run in parallel will consume still more memory. So you should ensure that your build server has enough RAM to cope with however many builds you intend to run simultaneously. Jenkins naturally needs RAM to run, but if you need to support a large number of build processes, it is not enough just to give Jenkins a lot of memory**. In fact Jenkins spans a new Java process each time it kicks off a build**, so during a large build, the build process needs the memory, not Jenkins.

**Installing jenking with a jar file**

# java -jar jenkin.war

the above cmd would spit out a password, make a note of it. jenkin would run port 8080 by default.

**Configuring the Tools:**

Before we get started, we do need to do a little configuration. More precisely, we need to tell Jenkins about the build tools and JDK versions we will be using for our builds.

Click on the **Manage Jenkins link** on the home page. This will take you to the Manage Jenkins page, the central one-stop-shop for all your Jenkins configuration. From this screen, you can configure your Jenkins server, install and upgrade plugins, keep track of system load, manage distributed build servers, and more.

There are many features available for now just click on “**Configure System**”.

The screen contains a lot of information, but most of the fields contain sensible default values, so you can safely ignore them for now.

By default, Jenkins stores all of its data in this directory on the file system.

Under the Advanced section, you can choose to store build workspaces and build records elsewhere.  
**/var/lib/jenkins**

By default the jenkin user has no shell access, we have to manually edit the **/etc/passwd** file.

jenkins:x:997:994:Jenkins Automation Server:/var/lib/jenkins:/bin/**false --- to**

jenkins:x:997:994:Jenkins Automation Server:/var/lib/jenkins:/bin/**bash**

plus we need to setup passwordless login for jenkin user and also setup sudo access for it.

**#visudo**

**jenkins ALL=(ALL) NOPASSWD: ALL**

**Defaults !requiretty**

For now, you will just need to configure the tools required to build our sample project. The application we will be building is a Java application, built using Maven. So in this case, all we need to do is to set up a recent JDK and Maven installation.

There are diff style projects available in jenkins.

1] Free Style Project.

**How to change the default home directory of jenkins.**

By default it is at where the jenkin war file was extracted.

Create a New Directory where you want to put ur Jenkins new home.

Edit the ".bashrc" file in ur home directory. Add below line in the file, this would make it permanent.

**export JENKINS\_HOME:/path/of/new/location.**

Now copy all data from the old location to the new location.

Then restart the Jenkins application

To view if this has changed, log into jenkins GUI,

Click on Manage Jenkins

Select Configure System.

There the new value should come up.

There is also one more useful technique of getting jenkins info.

browse this url; here we can also see Environment varibales.

**http://localhost:8080/systeminfo**

**Basic Of Jenkins.**

To setup basic functioning, how the jenkin server behaves. We can do it by going this way.

Click on **Manage Jenkins**.

Click on **Configure Jenkins**.

This will give us access to various parameters of jenkins as a whole.

This is the path where the builds would be stored.

**#Build Record Root Directory**.

This helps to control how many jobs can run in parallel on the jenkin server.

**# of executors: 5**

This help to label the nodes , i.e task would have a label selector, and they would execute task depending upon the label mentioned in the build.

**#Label**

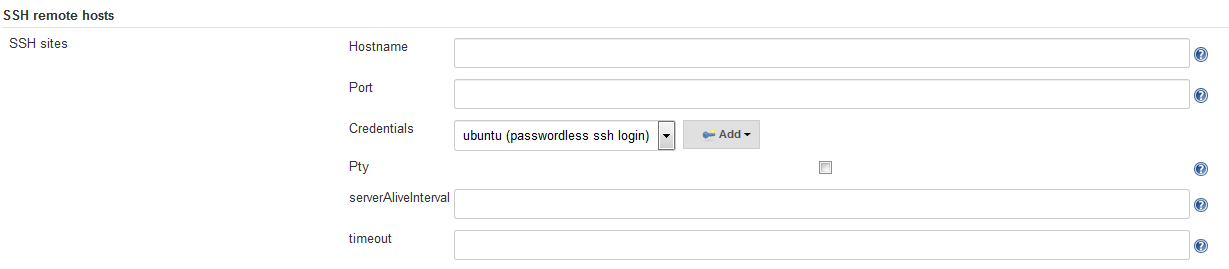
This sets the tone amounting to many build should be executed using this node, we have the option to

“**Use this node as much as possible**”

“**Only build jobs label expression matching this node**”

**#Usage**

One of the important feature over here is to set the **ssh remote host**. We can feed everything required for remote execution of a cmd and script. And simply go on adding this host in our build job by selecting it in the “**Build Environment**” section by selecting “**Execute shell script on remote host using ssh**”.



This help in setting environment variable for the jenkin as in whole. Like say we have installed java ona diff location that the default, we can pass on that location to jenkin using this. It accepts input in key value pair. Like JAVA\_HOME, ANT\_HOME.

**#Environment variable**

Plus there are so many option to setup over here, im listing a few of them.

**GitHub** # we can pass on github creds which would be used globally in jenkins.

**Git plugin**

**Subversion**

**Global Passwords**

**Shell**

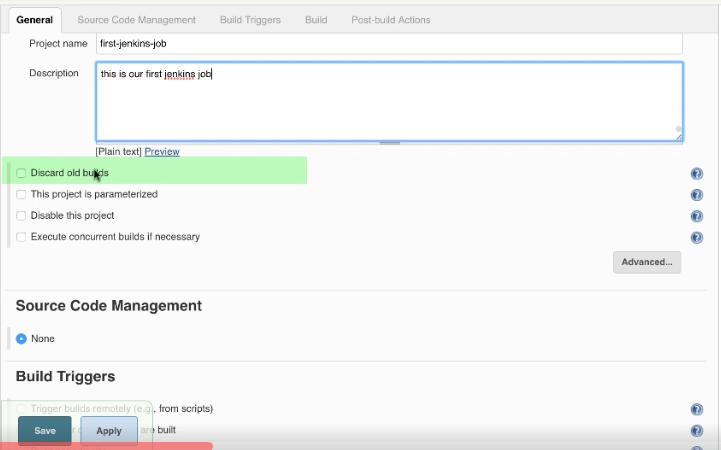
**Extended E-mail Notification**

**Creating our 1st jenkins Job.**

After loggin in, on the screen itself option would be to create a New jobs. So what are **jobs.**

Any task that is executed by jenkins is called a Job. Clicking on create Job would lead us to a page, that would ask us to define a name for our job and select a type of project. Different type of Jobs are explained above.

Suppose we selected the Free style project.



We will be greeted with a page asking for more information about our jobs.

There are some tabs available like General, Source code management, Build Triggers, Build, Post Build Actions.

**1]** In the general tab we will see the **Discard old builds** option. So in jenkins when a job is build a artifact is created, this artifact consume space on hdd, even without a artiface every build consume disk and memeory, so we need to manage them over the period of time. Clicking on **Discard old builds**  gives us the option to keep builds upto a certain days or upto a no of builds to keep.

Moving on we have the option of Disable this project, which is useful and a handy feature, what that basically does is suspends the execution of the job. Its like we can create a job and keep it ready for the right time, or we can even use it to pause a daily job for temporary basis like (Maintenance).

**2] Source code Management**

Jenkin supports CDS and SVN out of the box. Will add later on about this.

**3]**  **Build Triggers**.

This basically defines how and when a job would be triggered.

We have different option within this.

**A] Trigger builds remotely.**

This basically provies us with a Token url, which upon pasting in browser or doing a CURL to it would result in

executing that jobs, without even logging in the jenkins GUI. For eg it will be like something like this.

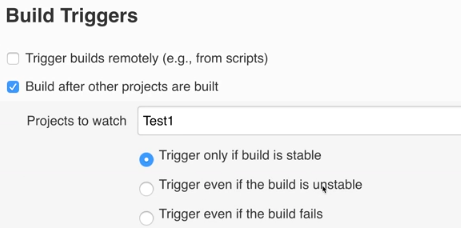
**http://lcalhost:8080/jobs/test1**

**B] Build After projects are build.**

In this we are basically joining multiple builds together. i.e output of one build allows or denies execution of the

next build in line. So we can have 2 jobs test1, and test2. Test 2 would check status of test1 and decide on

execution.



**C] Build Periodically.**

This is exactly conjobs, settign up time when the job execution takes place.

**D] Poll SCM**

This basically allows a build to get triggered as soon as some one Commits to github/svn. **I**t will also ask you to

setup time duration just like in Build periodically.

**4] Build.**

This is important part, here is where we would be defining our Jobs i.e what to actually do. Click on Add build Step

It will ask to execute a **windows batch cmd**, **Execute Shell** & **invoke top level targets** , this lis twill keep on growing

depending upon the number of plugins installed**.**

we will select the Execute shell option , where we would put in out cmd to be executed.

For eg lets say I want to run a java prog. 1st its going to where the code is kept, then compile it and at last run it.

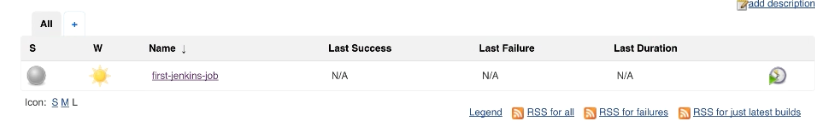


**5] Post Build Actions**.

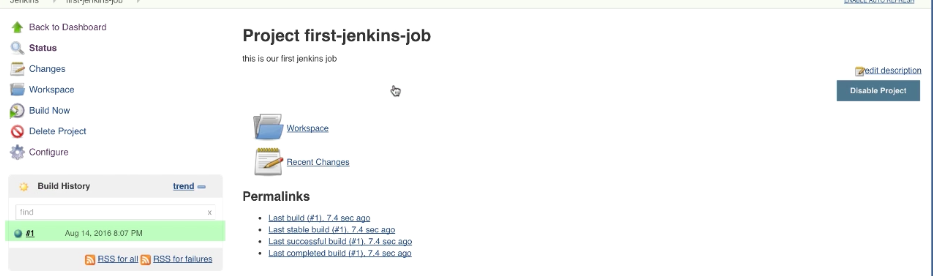
These are mostly used to send email notification upon successful execution of the builds. We can also use to store

results of our job.

Then click on save and apply. Now go back to the main page where u would see the job we just created.



We can click on the Job name, where it will show you the detail information about the job.



Clicking on the Build job option , would initiate a run of the job.

**Integrating Github with jenkins.**

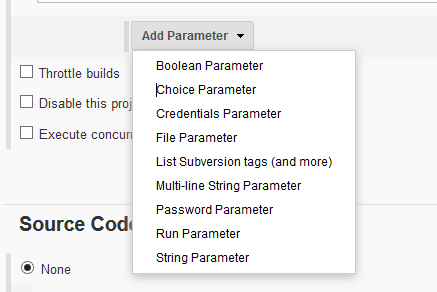
If we want to use github with jenkins we will have to install github plugin, that can be done from the main jenkin page click on manage jenkin.

**How to run a script on remote host using the Jenkins.**

To accomplish this we will be using the git, but it can also be done without it. We will have to create a Free style project.

Within that select “**This project is Parameterized**”, this is a cool option that give us the opportunity to define variables, select “**String Parameter**” . over here we can define variable ina key-value pair .

There are different options available like,



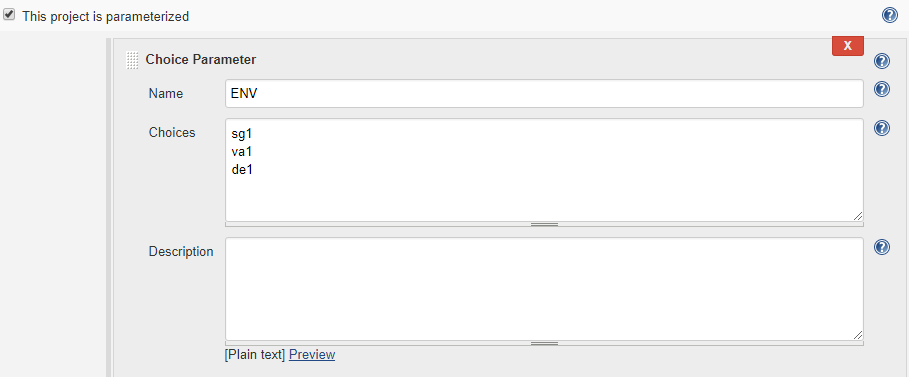
For example, you might have a project that runs tests on demand by allowing users to upload a zip file with binaries to be tested. This could be done by adding a ***File Parameter*** here.

Or you might have a project that releases some software, and you want users to enter release notes that will be uploaded along with the software. This could be done by adding a ***Multi-line String*** *Parameter* here.

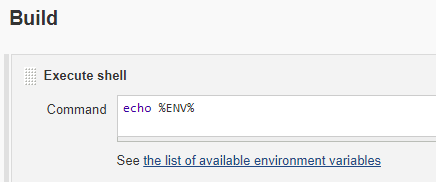
For eg we use **Choice Parameter.**

This is very useful if u have multiple regions, or stages in ur environment.

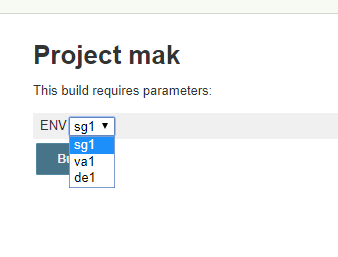
A below eg shows of multiple region, we would be asked to select atleast one of them, at the time of build. Below window is of the time of Build creation.



In the Build Section, we could use our varibale like this. Look how the name “**ENV**” is refrenced in the build section.



At the Time execution, we would be presented with this option. So our script would get executed on particular regions or hosts.



Now we can also use username and password, which we already specified in env, and later we can refrence them, without having to worry about being it exposed to someone else.

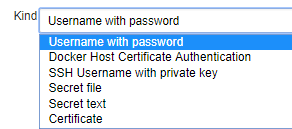
Yeah but now we have 2 options of doing this thing, either we can select a option from Parametiezed “**Credential Parameter**” or use “**Build Environment**”.

So either way, 1st create the username/password combo.

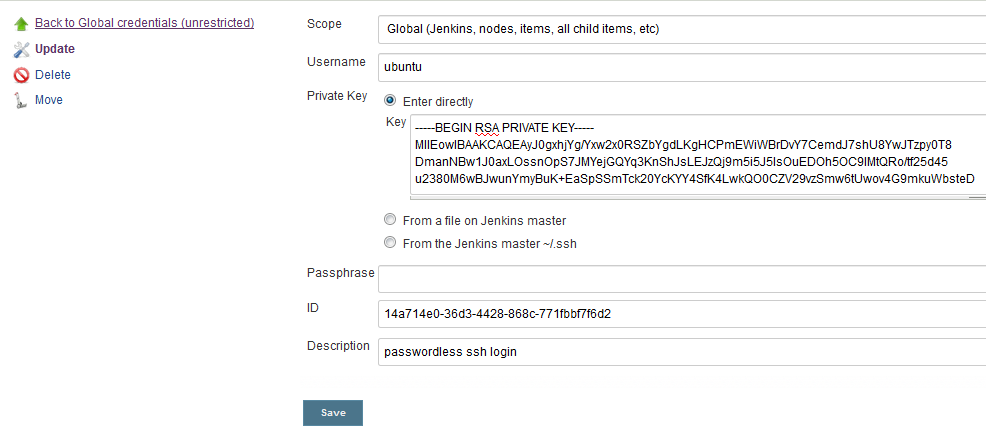
On the main jenkins page, on the left hand side click on Credentials, select System, then click on “[**Global credentials (unrestricted)**](http://13.126.50.223:8080/credentials/store/system/domain/_)”.

Click on “**Add Credential**”.

Now we have multiple option within that.



Like say we are setting up a FTP server, we can pass on Username/password with the **Username with Password** option, if it’s a ssh thing we can use the **SSH Username with Private key**. Is eg is shown below. Here we fill the actual username passphrase if available and the private key file.



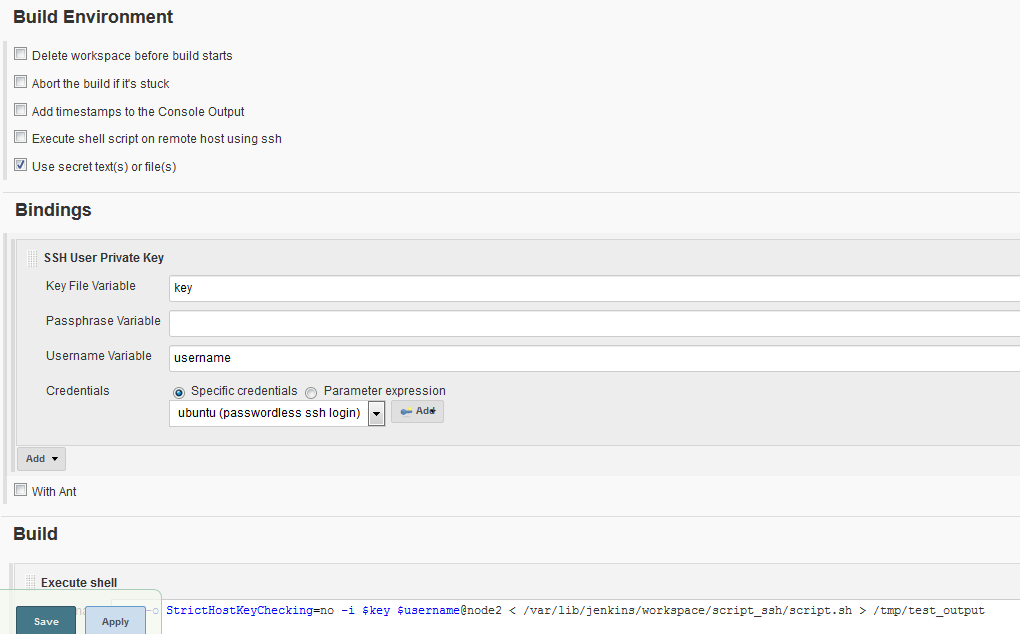
**Build Environment**:

Once this is setup, while ceating the job, in the Build Environment section select the **Use secret text or file option**.

Than a Bindings section would pop up. In that we can refrence our already build existing ssh username private key by selecting it from a drop down list, or can add a new one by clicking on ADD option.

As you can see we seperated the **key file variable**, and **username variable** with a unique name. the reason for doing is so that while we would be writing the actual build step to execute the script , we would require to put in the **ssh key path/value** and the **ssh username**. As u have seen we have refrenced them by using the **$ sign, $key** means take the value of private key specified in the credential section, **$username** means, pick username value from Username section of the Credential section.

And later on we wrote a simple cmd that executed a script on the remote server that refrenced our Build Environment bindings and performed a handsfree execution and that to it was secure.



Script Execution before using bindings

**# ssh -o StrictHostKeyChecking=no -i /var/lib/jenkins/workspace/script/id\_rsa ubuntu@node2 < /var/lib/jenkins/workspace/script/script.sh > /tmp/test\_output**

Script Execution After using bindings

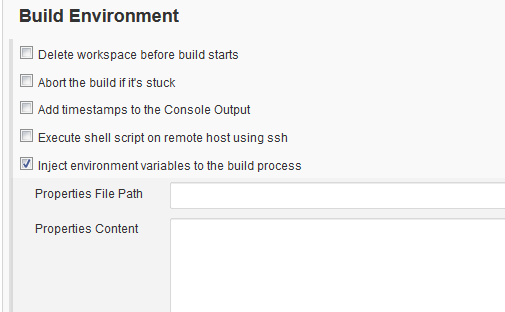
**# ssh -o StrictHostKeyChecking=no -i $Key $username@node2 < /var/lib/jenkins/workspace/script/script.sh > /tmp/test\_output**

**How to pass on Environment variable to the build.**

For this we will need to install the “**Environment injector**” plugin. Once that is done, during the build creation we will get the “**inject environment variables to the build process**” option.

Within this there are multiple option to decribe a variable in a **key:value** pair fashion.

We can even define passwords and later on access them like we did before with **$variable\_name.**



**How to create a user in jenkins, plus control , manager and assign Roles to these uers.**

**Admin user** is the default user created at installation.

To create a new user we will have to click on **Manage Jenkins & Click on Manage User**.

Click **Create user**, put name password over there.

Once u login with the new user, on the top right section, just beside the username, there is drop down option.

which will give us **confiure option**, which we can use to modify user setting.

Over there we would also get the API token of the new user, we can reset password.

we can even set SSH public keys.

Now our new local user has all the admin privilege. So how do we restrict it.

We will have to install the **"role-strategy"** plugin in Jenkins.

we can either do this via GUI or we can download the plugin file(**.hpi**) from repository and put it in the "**/jenkin\_home/plugins/**" directory.

Then restart the Jenkin Application.

Then Login the GUI, click on **Manage Jenkins**, select **Configure Global security** within that tick “**Enable Security”**.

Then in **Authorization** section select **"role base statergy"**.

Click Apply and save.

Now we did not put any privilege in new user, i.e it cant just login and will throw an error.

For enabling role on a user go to the **Manage jenkins** again, Click on "**Manage and assign roles**".

Within this there are 3 options

1] Manage roles

2] Assign Roles

3] Role Stratergy Macros.

Select Manage Roles.

There are 3 types of Roles

**Global Roles**

**Project Roles**

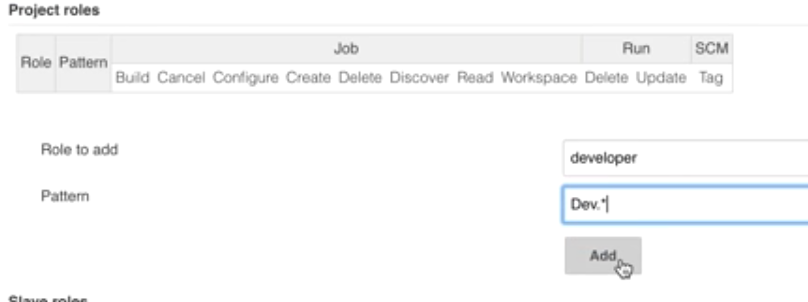
**Slave Roles**

Global roles have the power of fine tuning and assiging a normal user role of admin. Plus we can also limit as to what functions it can do if not a admin user. Suppose we can create a role over here that limits a user to view all jobs throughout. We will name it as **Employee Role**.

Project Roles are specific to a project or Group of Project.

In the **Role to add** section we will have to add Role name that we want to define. Like say **Developer**.

In **Pattern** section add a regex or exact name of the project. Like say **Dev.\*** . So what this will does is it will create a Role by the name Developer and it will allow access to the Projects starting by the nam Dev.   
Once this is done we will have to assign roles to the user.

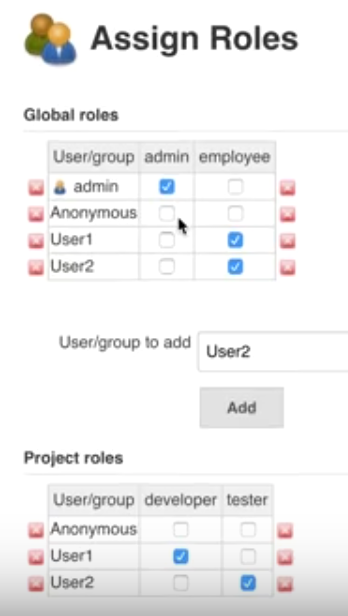


For assigning roles, go back one step in the “**Manage and Assign Role**” section. Click on assign Role.

We would be adding the 2 user which we created in Project roles, plus we would also need to give them the lobal reader access.

We will give user1 developer role.

User 2 will get tester role.



**How to setup Email Notification in Jenkins.**

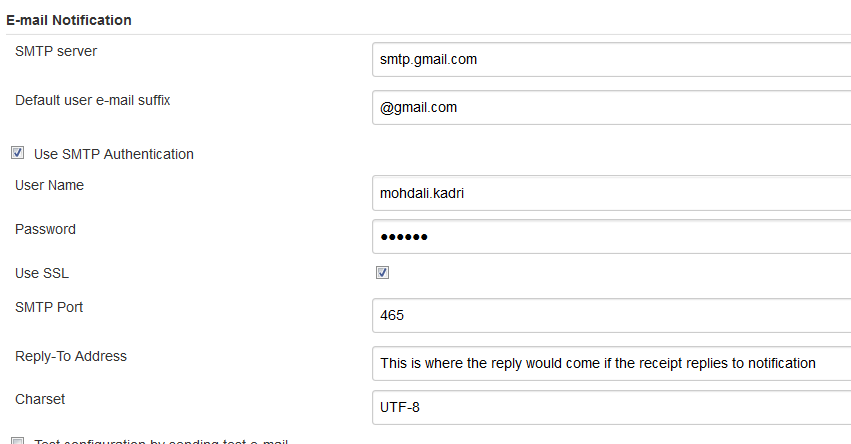
Go to **Manage Jenkins**.

Select **Configure Jenkins**.

In **Email Notification** section.

After feeling all this when we create a jobs, in the **Post build action** we can specify to **send email**, which would be send from the email id specified in this section.

This below eg is shown of Gmail.com, we can specify our own.



**How to install a plugin for jenkins from outside the jenkin store.**

We will need the “.**hpi”** file of the plugin, **.hpi** is the extension of jenkins plugin. Afte that

Go the **Manage plugin.**

Click on **Advance** tab**.**

Scroll down to **Upload a Plugin,** click browse and upload.

We will require a reload of jenkins to take this into effect.

This can be helpful if we are behind a proxy, and cant connect to internet.

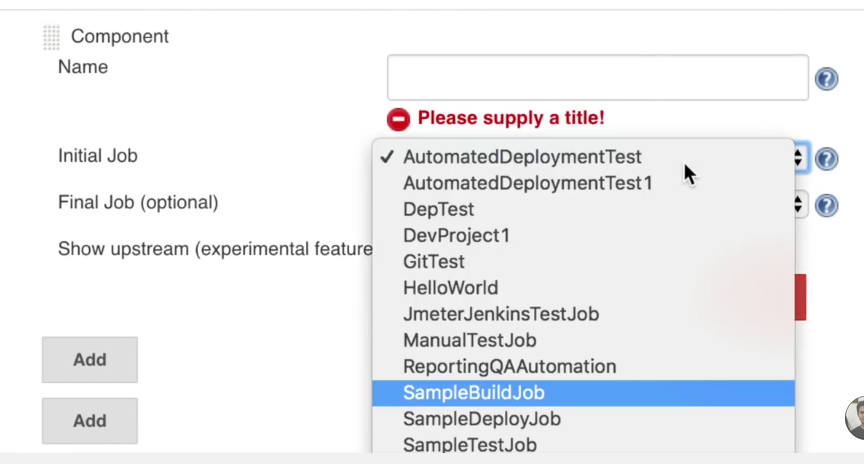
**Delivery Pipeline Plugin.**

This provide us with a Graphical pipeline view. Once we install this plugin, on the main dashboard, on the right handside

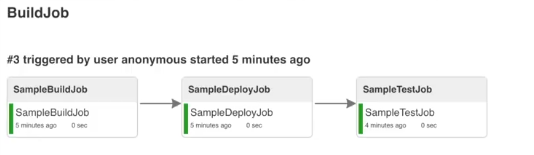
Click the plus button.



After this it will ask for to setup a viewname for DeliveryPipeline. And select the “**Delivery Pipeline View**” as shown below. After this we will get a list of option to fine tune, within that main option is to assign a existing job to the Pipeline view, so we will select our 1st “**sampleBuldjob**” and click on apply and save . This would generate us the pipeline view.



It would give us a view like this

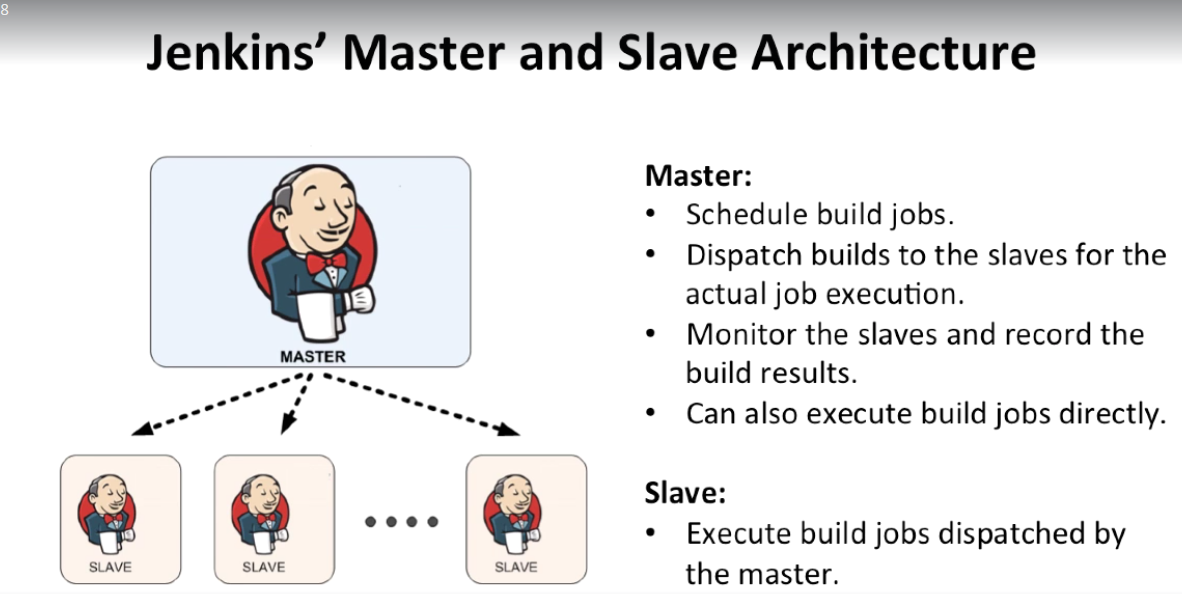
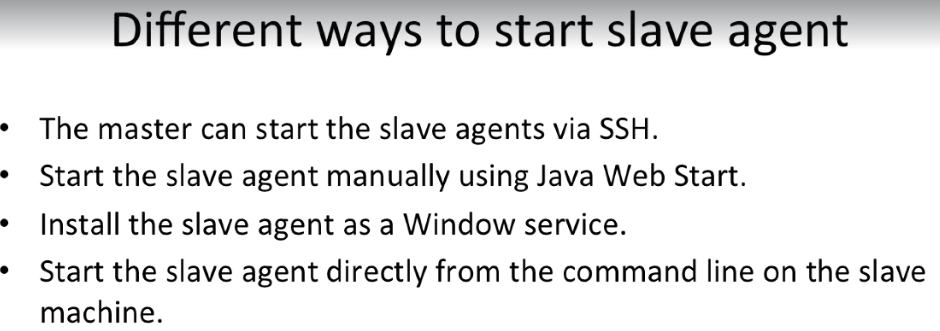


**Jenkins Master and Slave Setup.**

Jenkins works on a **'master->many slaves'** principle. The master node is responsible for creating jobs and managing the slave nodes, where the jobs are actually performed. The master node is essentially doing management work and farming out preset jobs to the most appropriate site.

A slave is a computer that is set up to offload build projects from the master and once connection is established between master and slave , tasks distribution is done automatic. Each slave runs a separate program called a “slave agent” . There is no need to install the full Jenkins on a slave. There are various ways to start slave agents, but in the end the “slave agent” and “Jenkins master” needs to establish a bi-directional communication link (for example a TCP/IP socket.) in order to operate.

Each slave then informs the master of the outcome and all of the job results are collated on the master node for easy viewing.

For setting up client nodes, we wiil have to add following plug-in to Jenkins as shown below

a. **Node and Label Parameter Plug-in**

**Setup Guide for master and slave configuration:**

**Implementing the slave node via ssh agent.**

1] For this we will require a no password ssh login from master to slave node.

2] Master node will be using a specific user “**jenkins**” to start the slave agent.

3] in our udemy video he used jenkin user on master and copied key to root user inslave machine.

4] Now download slave agent program to save machine.

Create bin dir in the root user home dir.

**# cd /root/bin**

**# wget** [**http://master\_node\_ip:8080/jnlpJars/slave.jar**](http://master_node_ip:8080/jnlpJars/slave.jar)

**# ls**

slave.jar

**# apt-get install default-jre**

5] now our slave setup is done, go to the master node url,

a) On the **master machine go to Manage Jenkins > Manage Nodes**.

b) Click on **New Node**.

c) Enter the **node name** and select as **Permanent Agent** and press OK.

Now it will ask us to how to set defaul behavour of the jenkin slave, like how many jobs to be executed through this slave. What to label this slave node, and etc.

Over here as shown in below fig, **executor** is a important factor. It basically decides how many jobs would run parallelly on the slave machine.

A good value for the number of executor start would be the number of CPU cores on the machine.

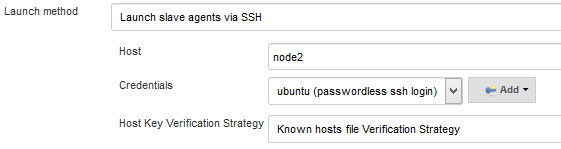
Setting a higher value would cause each build to take longer, but could increase the overall throughput.

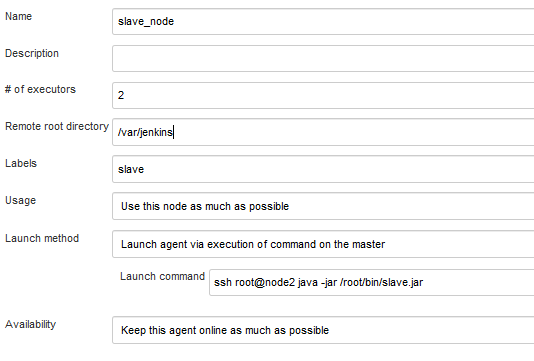
**Remote root directory:** place where our jobs would be built temporarily, this should be specified.

**Labels:** This helps in labelling the node, plus also helps in grouping a group of machines together.

**Launch method:**  How do we want our slave agent to be called on slave node. As shown in the below eg, we are basically ssh ing into slave node and running the cmd.

Other method of launch agent via ssh.





Slave node GUI setup page.

**Availability:** Controls when Jenkins starts and stops this agent. Keep this agent online as much as possible.

Take this agent online and offline at specific times :

In this we set “Startup Schedule” and “Scheduled Up time”

Startup Schedule : schedule cron job to start and stop node.

Examples:

every fifteen minutes : H/15 \* \* \* \*

Scheduled Up time : The number of minutes to keep the node up for. If this is longer than the startup schedule, then the node will remain constantly on-line.Take this agent online when in demand, and offline when idle

**How to create a Job with a Label to execute on a particular node.**

In real life scenarios, we might want a node to be reserved for certain kinds of jobs.

Like if u have a job that run performance tests, you may want them to only run on a specially configured machine, while preventing all other jobs from using it.

To do so, you may restrict where these tests jobs may run, by giving them a label expression matching that machine.

For this to work we should have already labelled our nodes (master & slave ) both. Then only our jobs labelling would work. To set this up, we would go in the particular job, click on restrict where this job run, and put the label of the node where we want this to run and that’s it.

