JENKINS

INTRODUCTION
==========

What is jenkins ??

Jenkins is an application that monitors executions of repeated jobs, such as building a software project.

Now jenkins can do a lot of things in an automated fashion and if a task is repeatable and it can be done in a same way over time, jenkins can do it not only doing it but it can automate the process, means jenkins can notify a team when a build fails, jenkins can do automatic testing(functional & performance) for builds.

Traditionally, development makes software available in a repository, then they give a call to operations/submit a ticket to helpdesk and then operations builds and deploys that software to one or more environments, one this is done, there is usually a QA team which loads and executes performance test on that build and makes it ready for production.

So what jenkins does is a lot of these are repeatable tasks, which can be automated by using the jenkins.

Jenkins has a large number of plugins which helps in this automation process.

Continuous Integration
==========

is a development practise that requires developers to integrate code into a shared repository several times per day (repos in subversion, CVS, mercurial or git). Each check-in is then verified by an automated build, allowing everyone to detect and be notified of problems with the package immediately.

Build Pipeline
==========

is a process by which the software build is broken down in sections:

- **Unit test**
- **Acceptance test**
- **Packaging**
- **Reporting**
- **Deployment**
- **Notification**

The concepts of **Continuous Integration**, **Build Pipeline** and the new "**DevOps**" movement are revolutionizing **how we build, deploy and use software**.

Tools that are effective in automating multiple phases of these processes (like Jenkins) **become more valuable in organizations where resources, time or both are at a premium.**

Installation of Jenkins

==========

- We need **java** to work with jenkins.
  - **# sudo yum -y install java-1.8.0-openjdk**
  - **# sudo yum -y install java-1.8.0-openjdk-devel**
  - **# java -version** ***{ confirm java version}***
- Generally jenkins runs on port 8080, so we need to make sure that there is no other service that is running and listening on port 8080

- Now we need to add jenkins repo, to our repository list, so that we can pull down and install jenkins package.
  - # sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
  - We will run a key so that we can trust this repo and pull down jenkins package
  - # sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
- Now our key has been imported we can do
  - # sudo yum -y install jenkins
- Now let's enable the service
  - # sudo systemctl enable jenkins
- Now let's enable the service
  - # sudo systemctl start jenkins
- Now if you do **# ip-address:8080** you can see jenkins home

Now if you do # cat /etc/passwd | grep jenkins

jenkins:x:996:994:Jenkins Automation Server:/var/lib/jenkins:/bin/false

Creates a user called jenkins, and keeps this user away from login, this is the default behaviour of jenkins coz we are right now on master and building everything here.

But in real time we have two things here **master** and **build slaves**
Master: where jenkins is installed and administration console is
Build slaves: these are servers that configured to off load jobs so that master is free

Creating Users

==========

Manage Jenkins → Manage Users → Create User Left Side → Fill details

Create Users : tester 1, tester 2, developer 1 and developer 2

Create new jobs
==========
testingJob ⇒ Execute Shell ⇒ echo "Testing Team Jobs Info"

developmentJob ⇒ Execute Shell ⇒ echo "Development Team Jobs Info"

Now login with the tester1 and see the list of jobs.
Now login with the developer1 and see the list of jobs.

As you can see, both the testing and development team jobs are visible across different teams, which would be a security concern, but this is the default behavior of jenkins.

Manage Jenkins → Configure Global Security → Authorization → Logged in users

Now let's see how we can secure the jenkins to make jobs only visible to testing and development teams.

For this we need to install new plugin called **Role-based Authorization Strategy.**

Let's see what are **PLUGINS** first

Plugins: plugins enhances jenkins power and usability.

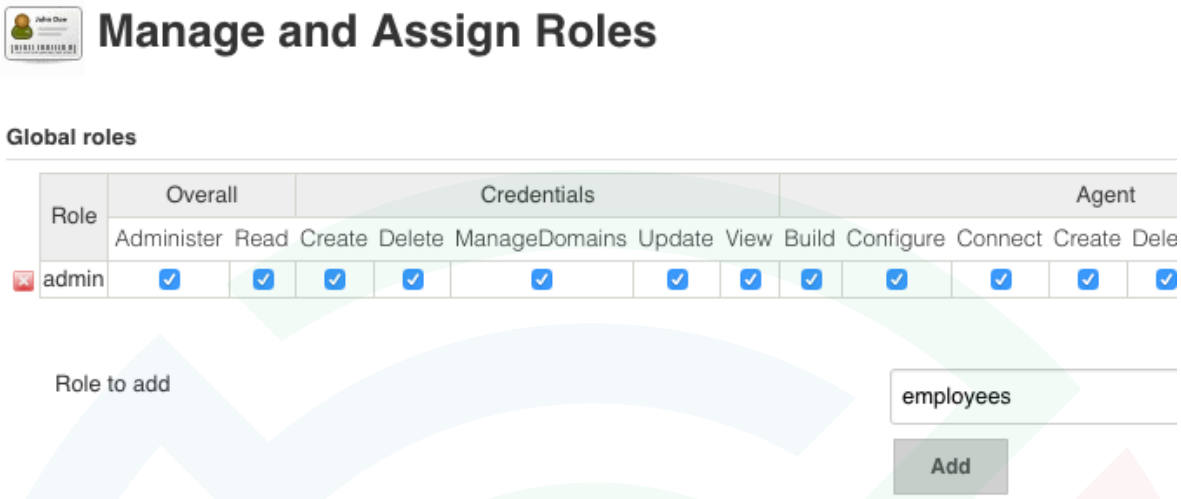
Installing Plugin
==========
Manage Jenkins → Manage Plugins → Available → Search Role-based → Install without restart.

Manage Jenkins → Configure Global Security → Authorization, now we can see the new option **Role-Based Strategy.**

Select the **Role-Based Strategy → Apply → Save**
Now if i login with the **tester** or **developer** user i won't have access, i can only access with the admin user.

Now let's see how we can go and **create some roles** and based on roles we should grant access to users:
Manage Jenkins → Manage and Assign Roles → Manage roles →

Global Roles: check **Role to add,** give something like **employee** and click add



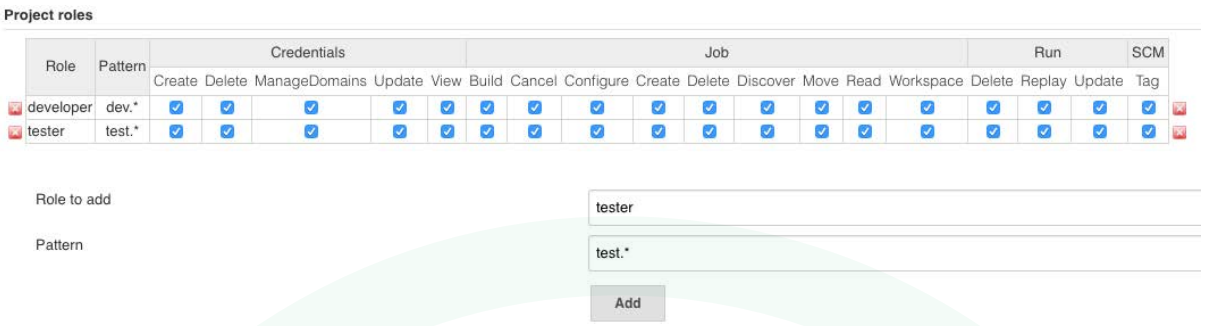
and give overall **read access** and over all **view access**

Project Roles: here we can **create roles specific to a project**,

Role to add: developer && Pattern: dev.* and check everything, now developers will only have access to projects that start with **dev** but nothing else. Click **Apply** and **Save**

Role to add: tester && Pattern: test.* and check everything, now testers will only have access to projects that start with **test** but nothing else. Click **Apply and save**

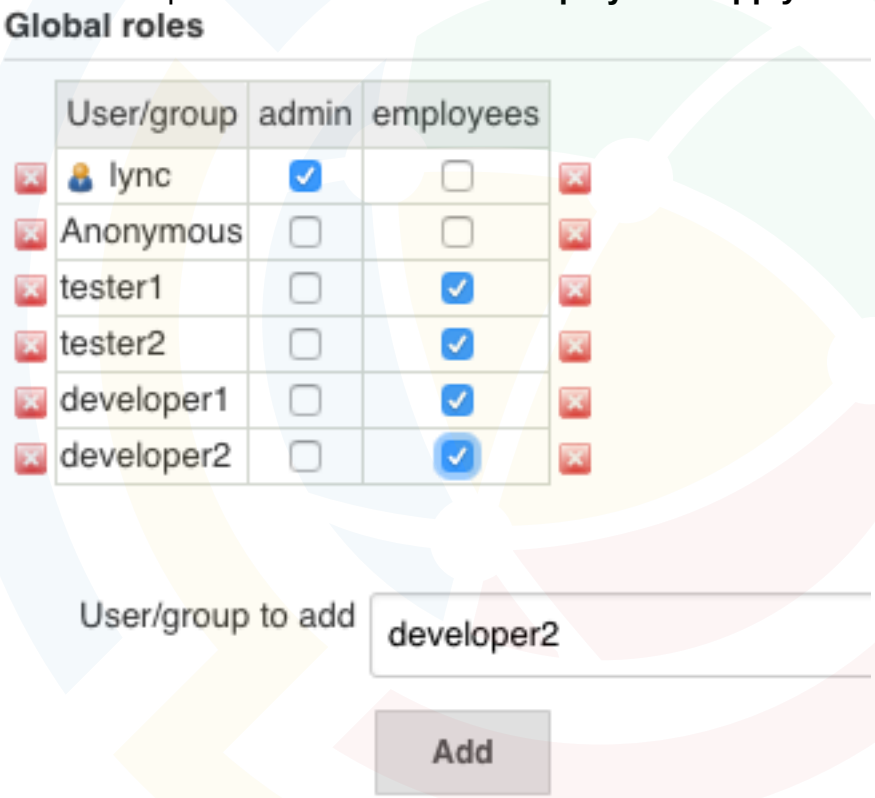
- So we have created an **employee role at global level** and we created two roles **developer and tester at project level**.



Assigning Roles To Users

==========

- **Manage Jenkins → Manage and assign roles → Assign roles → Global Roles → User/group** to add → Add Users tester1, tester2, develoepr1 and developer2 to **Global roles** as **employee → Apply**



  - Under **Item Roles** **User/group** to add → Add Users tester1, tester2, developer1 and developer2 to **Item roles** → Now add Users tester1 & tester2 to **Tester Roles** and Users developer1 & developer2 to **Developer Roles → Apply**
  - So we created users, we created roles and we assigned roles

Now if you login with tester users you can only see jobs related to testing, and similarly if you login with developer you can see only development related jobs.

Jenkins [Master - Slave Config]

==========

We know we have user jenkins, who has no shell and this user is the owner of jenkins application but beyond that it's a normal user. So what we are going to do is manage the global credentials.

Now we should make a decision that when we run jobs either locally or remotely we are going to run them with the jenkins user using ssh so that we can control slaves.

So on our system(master) we are going to change jenkins user, so that we can login with jenkins user, let's go and do it

vi /etc/passwd {jenkins change **/bin/false** to **/bin/bash**}
sudo su jenkins

Make sure our jenkins user is a sudoer
usermod -aG google-sudoers jenkins
vi /etc/sudoers or # sudo visudo
Below root add the following
    # root ALL=(ALL)   ALL
    # jenkins ALL=(ALL)   NOPASSWD: ALL

Switch to jenkins user
    # sudo su jenkins
    # cd   { make sure you are in jenkins home /var/lib/jenkins}
    # ssh-keygen

Click on Credentials on homepage of jenkins → Click on Global credentials → Adding some credentials → Kind (select SSH username with private key) → Scope (global) → Username (jenkins) → Private Key (From jenkins master ~/.ssh) → Ok

Now this sets jenkins user account to be available for SSH key exchange with other servers. This is imp coz we want the ability so that single jenkins user can be in control to run our jobs remotely so that master can off load its jobs to slaves.

Slave
=====
Now create a new centos server(machine) where you will be getting new ip {1.2.3.4}
Create a user jenkins **# useradd jenkins**
    # usermod -aG google-sudoers jenkins
    # sudo su jenkins -
cd   { make sure you are in jenkins home /home/jenkins}

Now do password less authentication steps on both machines :

vi authorized_keys (add public key of other machine)

chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
Now if everything is good then you should be able to login into the slave machine without password.

Doing builds on slaves
==========
So we talked about this a lot, that we don't want to do builds much on master.
We are going to create dumb slave nodes, slaves doesn't need to know anything about implementation they should be able to just run jobs and be controlled by master.

We are going to use this slave nodes in order to off-load the build processing to other machines so that the master server doesn't get CPU, IO or N/W load etc in managing large number of jobs across multiple servers multiple times a day.

So we are going to **create a slave node**:

Now make sure there is key exchange b/w both the machines.

- # **Manage Jenkins** (scroll down) → **Manage Nodes** → Master (Now master is always going to be included by default here if we click on master) → Configure → Usage : Only build jobs with label matching nodes(for the most part we want master only to use for controlling jobs we have setup) → Save

Manage Jenkins (scroll down) → Manage Nodes → New Node → Node name : Remote slave 1 → **#** of executors : 3 (up to 3 concurrent jobs) → Remote root Directory: (/home/jenkins) Labels: remote1 → Usage: (as much as possible) → Launch method: Launch slave via SSH → Host: ip of slave → Credentials: Service acc(Give settings of **Kind**: SSH username with private key **&&** Private key: From jenkins master ~/.ssh) → Availability: Keep online as much as possible → Save
Install java and javac on all slaves.
Then click on node and see the **log**.

Now if i goto jenkins home i can see both master and slave and their executors

Building a job on slave
==========
New job → Under General (□ Restrict where this project can run)(Label expression: remote1 or expression we gave while creating slave) → Build → Execute Shell → in command give # pwd # uname -a **#** hostname → Save → Run build.