Jenkins

JENKINS

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

- Jenkins is continues integration and delivery and deployment tool.
- Today is Jenkins more than CI/CD it also used as orchestration tool.
- Jenkins is open source tool.
- Initially Jenkins called as Hudson and this tool is the license of Oracle.
- Jenkins also has an enterprise tool called as Cloud Bees.
- We can extend the features by installing plugins.
- Below is the list of CI tools in market.
  - Jenkins
  - Bamboo (Atlassian tool)
  - Hudson
  - Teamcity
  - Solano

Advantages:

- It saves developers time.
- Improved software quality.
- Faster delivery decreased developer's time.
- Early tracking of defects and immediate bug defects.
- No integration step software development life cycle.
- A deployable system at any given point.
- Email notifications to developers.
- Large number of plugin support.
- It is an open source tool with great community support.
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share with the community.
- It is built with Java and hence, it is portable to all the major platforms.
- Achieves continuous integration agile development and test driven development.
- With simple steps, maven release project is automated.

- Iterative improvement to the code.
- Build failures are cached at integration stage.
- For each code commit changes an automatic build report notification generates.

1. Jenkins is open source and free

Jenkins is free to download and the source code is also available. This has spawned a growing community of developers who are actively helping each other and contributing to the Jenkins project. This ensures that you get better and more stable versions each year.

2. Jenkins comes with a wide range of plugins

Jenkins has a wide range of plugins that give a developer a lot of added features and power over the already feature-rich Jenkins installation. These plugins help developers extend Jenkins into their own custom tool for their projects.

There are few of **23+ popular Jenkins Plugins** listed below:

1. Global Build Stats
2. Job Generator
3. Disable-failed-job
4. Embeddable-build-status
5. GitHub/GitLab Pull Request Builder
6. JDK Parameter Plugin
7. Job Configuration History
8. Multiple SCMs
9. Parameterized Trigger
10. Configuration Slicing
11. Pre SCM BuildStep
12. SCM Sync Configuration
13. Restarting safely
14. Green balls
15. Build pipelines
16. Promoted Builds and Build Trigger Badge
17. Job DSL
18. Kubernetes
19. Copy Artifact
20. Jira Plugin

21. Jenkins Maven
22. Gatling
23. Test Results Analyzer

3. Jenkins integrates and works with all major tools

Jenkins being the popular tool it is, integrates with all major version control tools like **CVS, Subversion, Git, build tools like Apache Ant and Maven** and even other tools like **Kubernetes** and **Docker**.

4. Jenkins is flexible

With its wide range of plugins and open architecture, Jenkins is very flexible. Teams can use Jenkins in projects of various sizes and complexity.

Jenkins places no limits on the kinds and numbers of servers that can be integrated with it. Teams across the globe can work towards continuous delivery, seamlessly.

5. Jenkins comes with a decent API suite

Jenkins comes equipped with APIs that lets you tailor the amount of data that you fetch. This mechanism helps your server use simple web-hooks to invoke Jenkins for specific purposes.

6. Jenkins is easy to use

An active, vibrant community, regularly updated documentation and support for all major operating systems mean that a person with just decent skills can download and get Jenkins up and run in a few hours.

The active community also readily answers questions on forums and groups that encourage newcomers into taking up this technology, more readily.

7. You have a ready talent base

We all know how difficult it is to find talent for very niche technologies. Jenkins has cemented itself to be a popular tool in software development and it is absolutely possible for a recruiter to find a decent developer who is also good with Jenkins.

Disadvantages:

- UI feels outdated and unintuitive, especially to newer users.

1. Unpredictable costs

The costs of hosting the server (which isn't free) that Jenkins runs on cannot be predicted easily. It is not always possible to predict the kind of load (depending on the number of commits, volume of code, volume of assets etc.) that the server is

going to serve. Hence the cost factor, even if Jenkins itself is free, remains unpredictable.

2. Lack of governance

Jenkins' management is generally done by a single user and that leads to tracking and accountability problems with the pushed code.

There is tracking that exists on the Version Control Server but there is no tracking of that kind of Jenkins itself, which is a concern.

3. No collaboration features

Jenkins doesn't allow one developer to see the commits done by another team member, readily. This makes tracking the overall release progress a rather difficult job for larger projects. This can cause a lot of trouble to the release manager.

4. Lack of analytics

Jenkins doesn't provide any analytics (there are plugins but they are not enough) on the end-to-end deployment cycle. This again goes back to the lack of overall tracking that contributes to the lack of analytics as well.

5. Needs personnel

While Jenkins is indeed a powerful and useful tool, managing a Jenkins server needs special attention and many times, a dedicated developer. This adds man hours to the project and pushes up the overall project cost as well.

Before Jenkins	After Jenkins
Once all Developers had completed their assigned coding tasks, they used to commit their code all at same time. Later, Build is tested and deployed. Code commit built, and test cycle was very infrequent, and a single build was done after many days.	The code is built and tests as soon as Developer commits code. Jenkin will build and test code many times during the day If the build is successful, then Jenkins will deploy the source into the test server and notifies the deployment team. If the build fails, then Jenkins will notify the errors to the developer team.
Since the code was built all at once, some developers would need to wait	The code is built immediately after any of the Developer commits.

until other developers finish coding to check their build	
It is not an easy task to isolate, detect, and fix errors for multiple commits.	Since the code is built after each commit of a single developer, it's easy to detect whose code caused the built to fail
Code build and test process are entirely manual, so there are a lot of chances for failure.	Automated build and test process saving timing and reducing defects.
The code is deployed once all the errors are fixed and tested.	The code is deployed after every successful build and test.
Development Cycle is slow	The development cycle is fast. New features are more readily available to users. Increases profits.

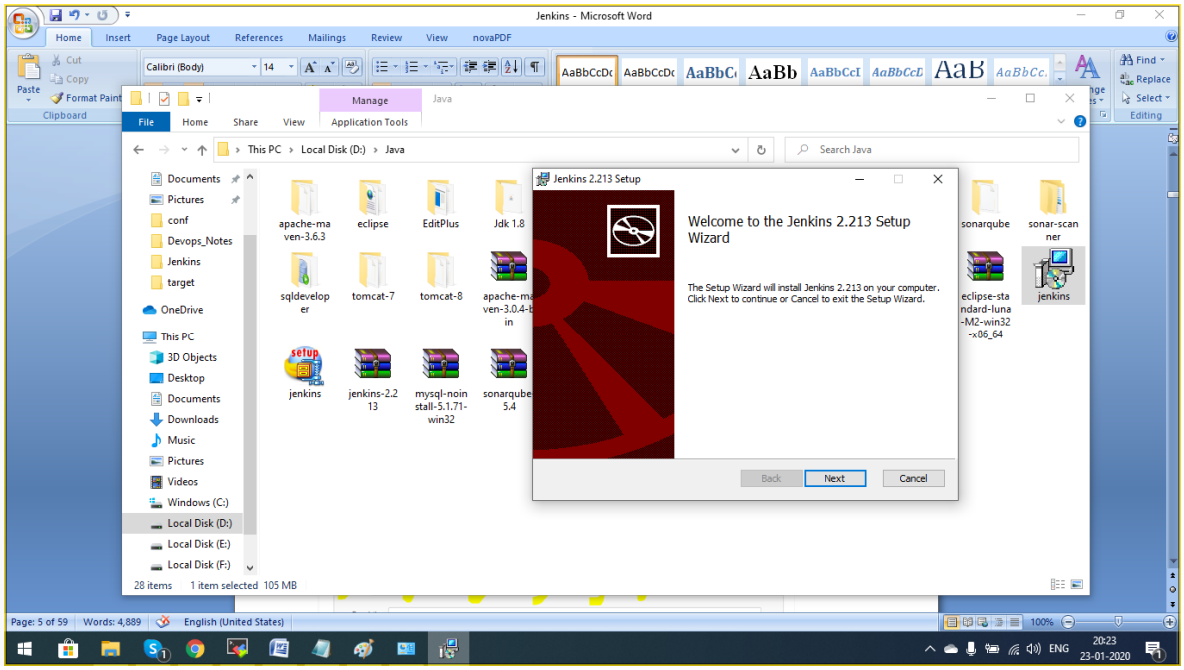
Bamboo

The Bamboo is a CI and CD server from Atlassian. Like Jenkins, Bamboo allows developers to automatically build, integrate, and test source code, then prepare the app for deployment. Bamboo also works effortlessly with Atlassian's other tools like Jira (project management) and Hipchat (team communication).The purpose of Bamboo is to provide developers with an environment which quickly compiles code for testing so that release cycles can be quickly implemented in production. It can also be customized with tons of features and add-ons that can be found at the Atlassian Marketplace. For example, it's possible to get a plugin for slack notifications.

Some reasons to evaluate and choose Bamboo include:

- The bets JIRA integration
- Flexible CI and CD pipelines
- First-class support for deployments
- Painless CI on the branch
- Automated merging
- Fast import from Jenkins
- Legendary support and resources
- Powerful build agent management
- On the fly customizations

Jenkins installation



Ubuntu

sudo apt-get update

sudo add-apt-repository ppa:openjdk-r/ppa

sudo apt-get install openjdk-8-jdk

/usr/lib/jvm/java-8-openjdk-amd64/bin

/usr/lib/jvm/java-8-openjdk-amd64

sudo wget https://www-eu.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz

tar -xvf apache-maven-3.6.3-bin.tar.gz

mv apache-maven-3.6.3 maven

sudo chmod 777 -R maven
/home/ubuntu/maven
Go to home → /home/ubuntu/ → vi .profile add below entries.

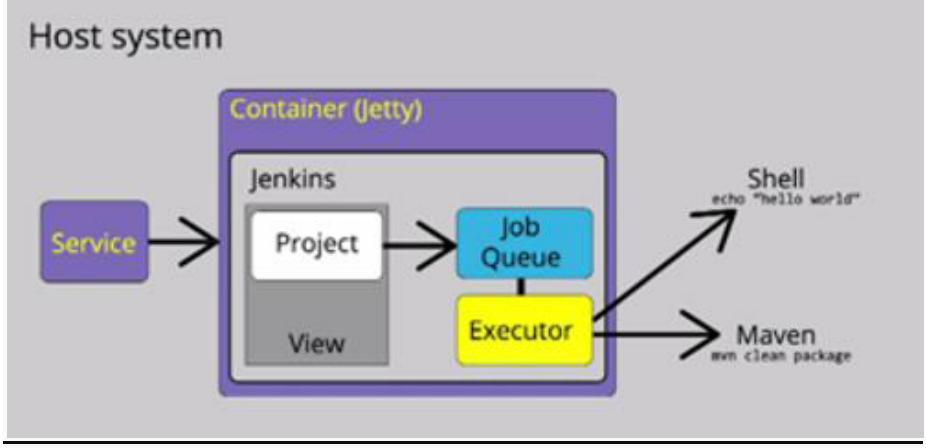
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export JAVA_HOME
MAVEN_HOME=/home/ubuntu/maven
export MAVEN_HOME
M2=/home/ubuntu/maven/bin
export M2
PATH=$PATH:$M2:$JAVA_HOME/bin
export PATH

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.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 update
sudo apt install jenkins

sudo systemctl start jenkins
sudo systemctl status Jenkins
sudo systemctl stopJenkins

sudo apt-get update
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get install openjdk-8-jdk
sudo apt-get install ca-certificates
sudo wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.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
sudo systemctl start jenkins
sudo systemctl status jenkins

Jenkins Architecture



Job configuration:

By using job configuration we can create new jobs and we need to provide the below details.

1. Where to run (Master or Slave)
2. When to run (Schedule or Manual)
3. What it has to do. (task)

Free style job:

A free style contains only template, we can configure the required details and we will create and run the job.Majority of the plugins are written to use the free-style project.

Maven project:

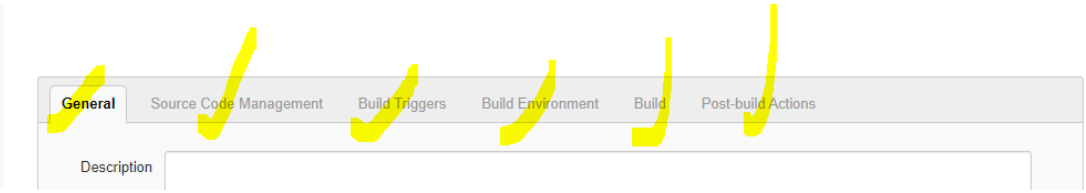
A maven project is a project that will analyze the pom.xml file in greater detail and Produce a project that's geared towards the targets that are invoked. The maven project is smart enough to incorporate build targets like the javadoc or test targets and automatically setup the reports for those targets.

Multi-configuration project:

The “multi configuration 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. We will be looking at how to configure multi configuration build jobs later on in the book.

Monitor an external job:

The “Monitor an external job” build job lets you keep an eye on non-interactive processes, such as cron jobs.



General:

The General tab has basic information about the Jenkins build you are creating, such as a description and other general build information.

Source Code Management:

The Source Code Management tab is where you specify the type of version control management system you are using, such as Git, SVN, and Mercurial

Build Trigger:

When we want to run this job i.e. specific time and date Jenkins will trigger the job.

Build environment:

If we want to provide some additional details to this job we can go for build environments.

Build:

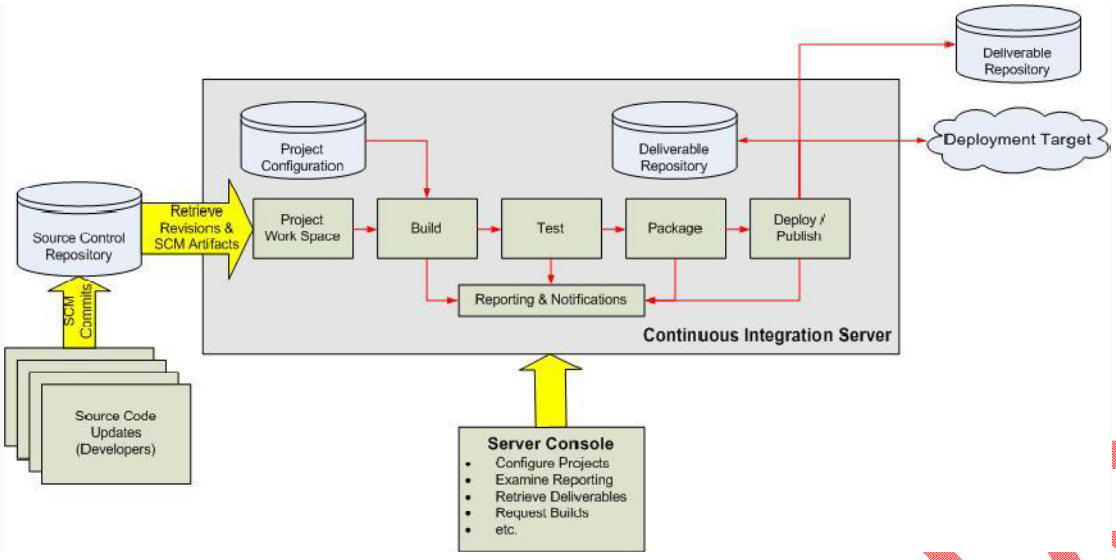
It's the exact section what it has do the Jenkins job.

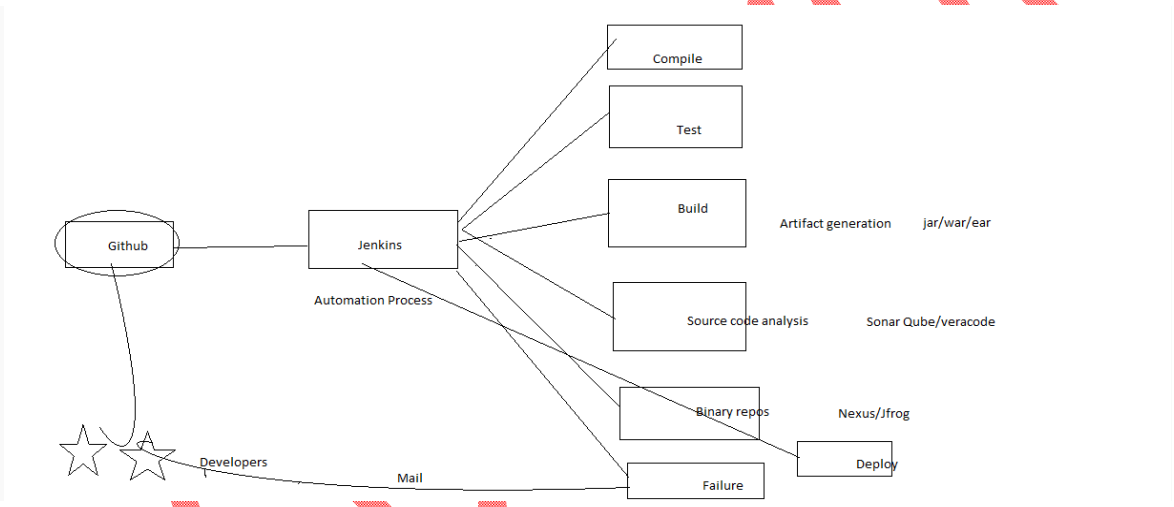
Post build action:

After complete the job if we want to provide any other details we can go for Post build actions.

Continuous Integration (CI)

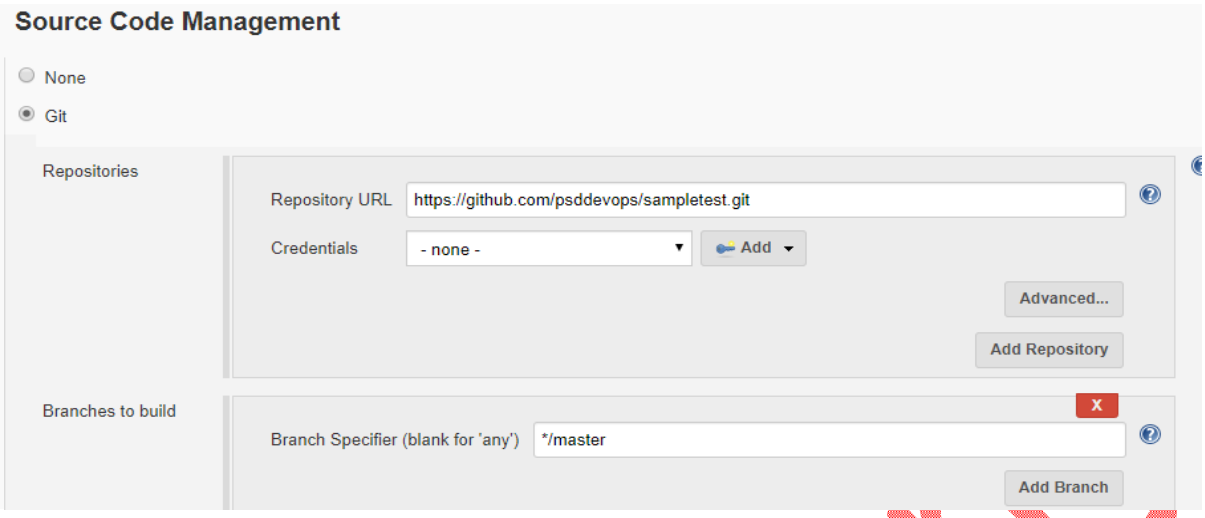
Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests.

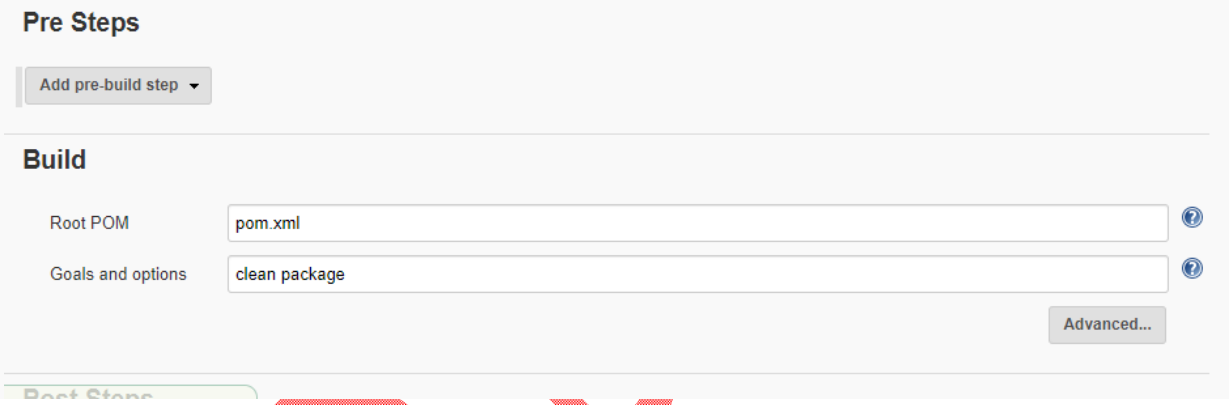




Create Jenkins job to clone the project and build a war file.

- Click new item
- Enter item name as “CID”
- Select Freestyle project and click Ok
- Under source code management select git
- Enter git URL https://github.com/psddevops/sampletest.git
- Under build section add “top level maven targets”
- Under Goals enter clean package
- Save the job.
- Run the job by clicking Build Now

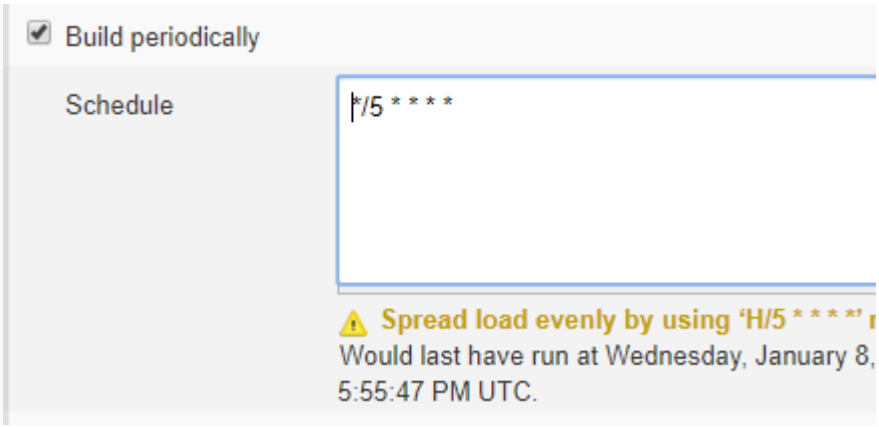




Trigger the above job automatically every five minutes

This job will run based on the time schedule; It will check the git repository on the schedule basis, Without any source code changes also this job will run.

- Select the job "CID" → click configure
- Select build triggers
  - Select Build periodically, this is the schedule give to jenkins to schedule the every five minutes
  - Under schedule put five stars → */5 * * * *
  - Save the job
  - To test this job do some changes in git, it automatically triggers this job



This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:
MINUTE HOUR DOM MONTH DOW
MINUTE Minutes within the hour (0–59)
HOUR The hour of the day (0–23)
DOM The day of the month (1–31)
MONTH The month (1–12)
DOW The day of the week (0–7) where 0 and 7 are Sunday.
To specify multiple values for one field, the following operators are available. In the order of precedence,

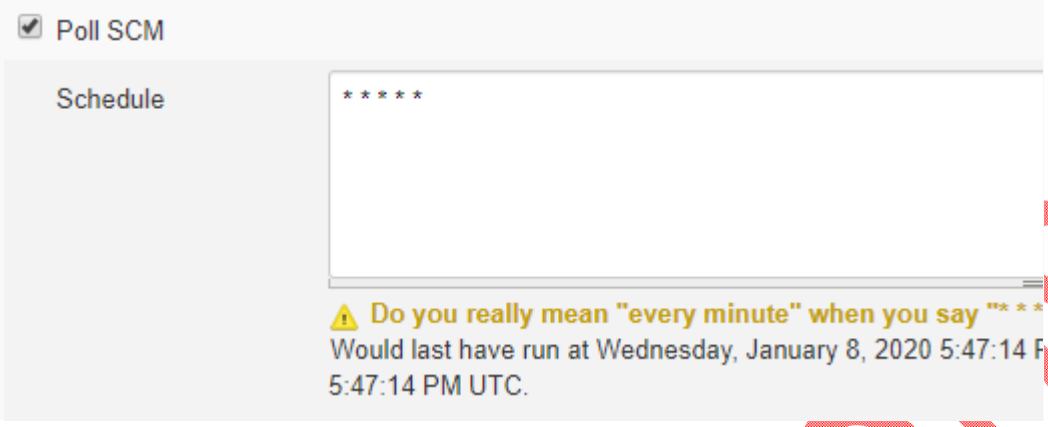
- * specifies all valid values
- M-N specifies a range of values
- M-N/X or */X steps by intervals of X through the specified range or whole valid range
- A,B,...,Z enumerates multiple values

Trigger the above job automatically if there are new changes in git

PollScm will checks the SCM,If found any changes and new commits it will run.PollScm internally uses the GIT polling log and Jenkins will run. Polling log contains the commit-id's used by last build.This concept also little burden to the Jenkins server.

- Select the job "CID" → click configure
- Select build triggers
  - Select Poll SCM, this is the schedule give to jenkins to check for updates in SCM
  - Under schedule put five stars → * * * * *
  - Save the job

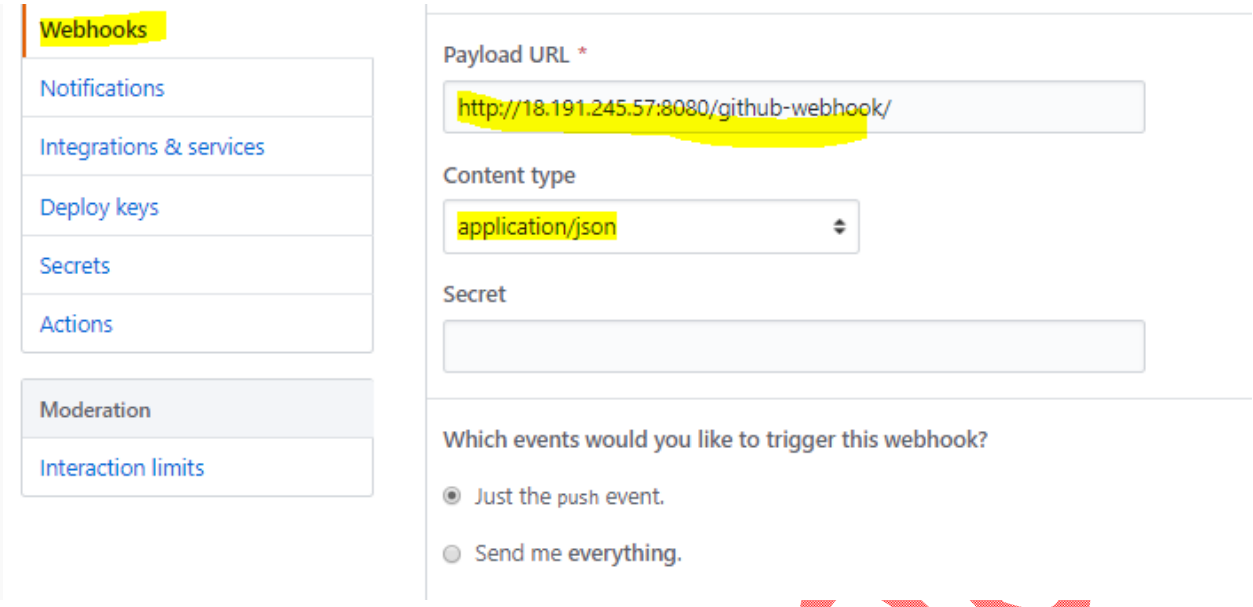
- To test this job do some changes in git, it automatically triggers this job



Trigger the above job automatically if there are new changes in git

- Select the job "CID" → click configure
- Select build triggers
  - Select GitHub hook trigger for GITScm polling, this tells jenkins if changes found in git then trigger this job
  - Select Poll SCM, this is the schedule give to jenkins to check for updates in SCM
  - Under schedule put five stars → * * * * *
  - Save the job
  - To test this job do some changes in git, it automatically triggers this job

Github side:
Open the proj → settings → Webhooks → Add webhook
Provide the Jenkins url like: http://18.191.245.57:8080/github-webhook/
Content type : application/json



Open Jenkins job → Build triggers → GitHub hook trigger for GITScm polling

Deploy the war using Jenkins

Method-1

- Generate the war file and deploy to the tomcat using ssh communication.

Goto → Jenkins server

sudo -i -u jenkins

ssh-keygen

cd .ssh

cat id_rsa.pub

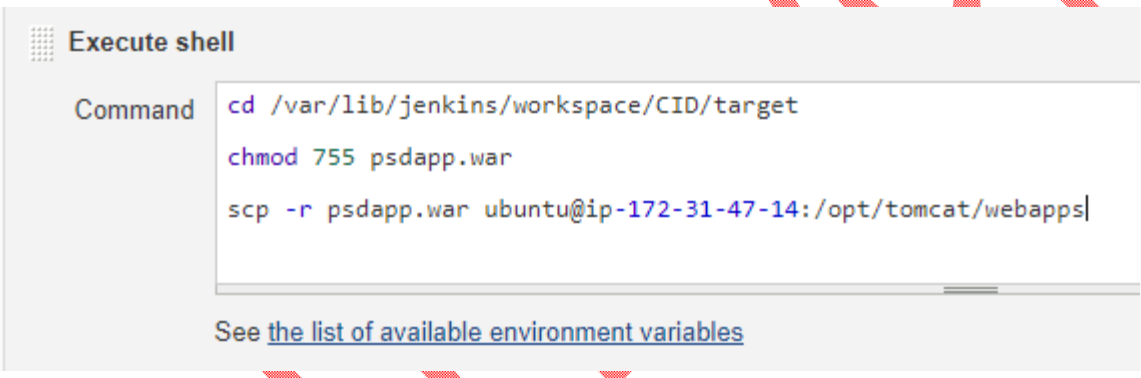
add the public key to the tomcat server.

```
ubuntu@ip-172-31-47-14:~$ cd .ssh
ubuntu@ip-172-31-47-14:~/.ssh$ ll
total 12
drwx------ 2 ubuntu ubuntu 4096 Jan  8 16:02 ./
drwxr-xr-x 5 ubuntu ubuntu 4096 Jan  8 18:06 ../
-rw------- 1 ubuntu ubuntu  797 Jan  8 16:02 authorized_keys
ubuntu@ip-172-31-47-14:~/.ssh$ vi authorized_keys
```

Click new item → Enter item name as "BuildAndDeploy" → Select Freestyle project and click Ok

- Under source code management select git
  - Enter git URL https://github.com/psddevops/sampletest.git
- Under build section add "top level maven targets" Under Goals enter clean package

Select add build step → execute shell



Run and test the application.

Method-2

For deployments we are going to use jenkins plugin, this plugin deploys the war using tomcat's manager application.

Note: Tomcat manager application must be enabled

Go to tomcat server → cd /opt/tomcat/conf update the

```
<tomcat-users>
<role rolename="manager"/>
        <role rolename="admin"/>
        <role rolename="manager-gui"/>
<role rolename="manager-script"/>
```

```
        <role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<role rolename="admin-gui"/>
<role rolename="admin-script"/>
        <user username="tomcat" password="tomcat"
roles="admin,manager,manager-gui,manager-script,manager-jmx,manager-
status,admin-gui,admin-script"/>
        </tomcat-users>
```

Update the Context.xml in /opt/tomcat/webapps/manager/META-INF

```
<Context antiResourceLocking="false" privileged="true" >
<!--  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
       allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
<Manager
sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Numb
er|String)|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|jav
a\.util\.(?:Linked)?HashMap"/>
</Context>
```

Update the Context.xml in /opt/tomcat/webapps/host-manager/META-INF

```
<Context antiResourceLocking="false" privileged="true" >
<!--  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
       allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
<Manager
sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Numb
er|String)|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|jav
a\.util\.(?:Linked)?HashMap"/>
</Context>
```

Install Deploy to container plugin

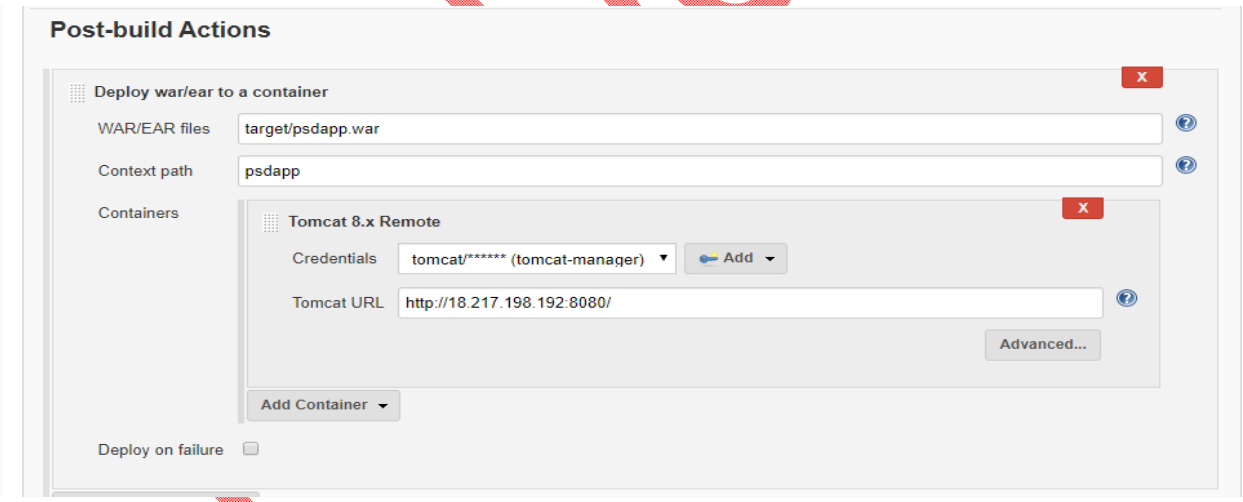
Manage Jenkins → Manage Plugins → Available →

Search for Deploy to container Plugin and select → install without restart

- Go back to jenkins home
    - Click new item
    - Enter item name as “BuildAndDeploy”

- Select Freestyle project and click Ok
- Under source code management select git
- Enter git URL https://github.com/psddevops/sampletest.git
- Under build section add “top level maven targets”
- Under Goals enter clean package
- Under post build actions
    - Select deploy war/ear to the container
    - War/Ear file → target/pdsapp.war
    - Context Path → psdapp
    - Add container → tomcat8
- Credentials
    - Add → jenkins → select username with password
    - Enter tomcat managers username and password
    - For Id field enter relevant value for example tomcat-manager and ADD
- Select the credentials
- Add tomcat url for example (http://172.31.2.228:8080/)
- Save the job

    Run the job by clicking Build Now



Email triggering jobs:

Send automated email to the dev team if build fails

1. In Order to send and receive emails we need to configure email servers in jenkins. Companies has their own email servers, we have to use those servers to trigger Emails, in our example we do not have our own server, so we are going to use Gmails SMTP server.

2. Configure gmail SMTP server in jenkins

jenkins → Manage Jenkins → Configure System

Under **E-mail Notification**

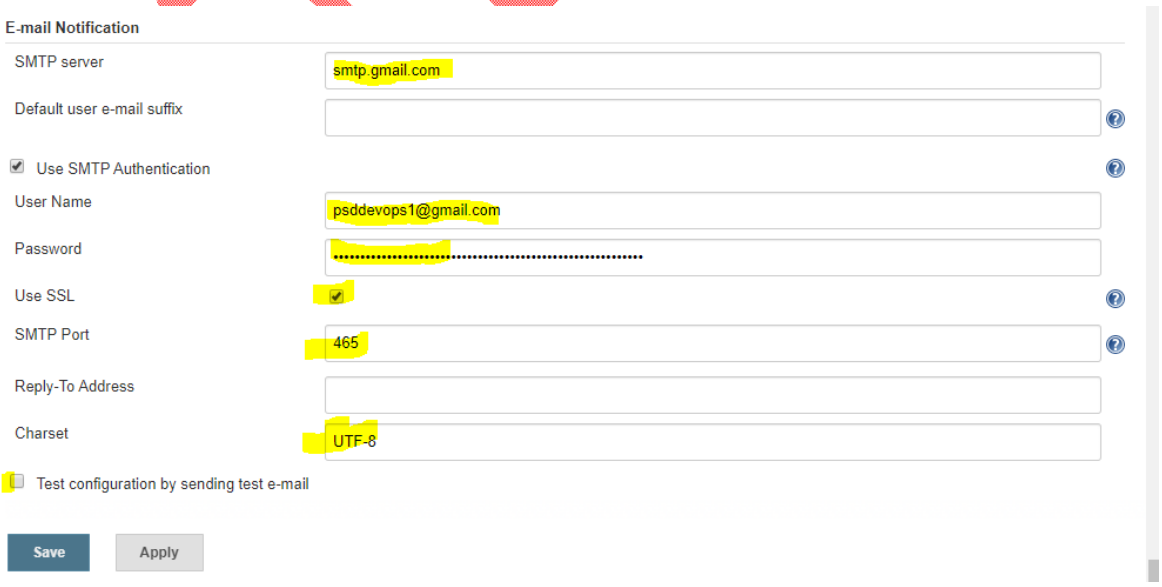
1. SMTP server → **smtp.gmail.com**
2. Select Use SMTP Authentication
3. Put your gmail id
4. Put your gmail password
5. Use SSL select
6. SMTP port 465
7. Save the configuration

Configure jenkins job to trigger email if build fails

Open your job → Configure → Post Build Actions → Add post-build action → email notification

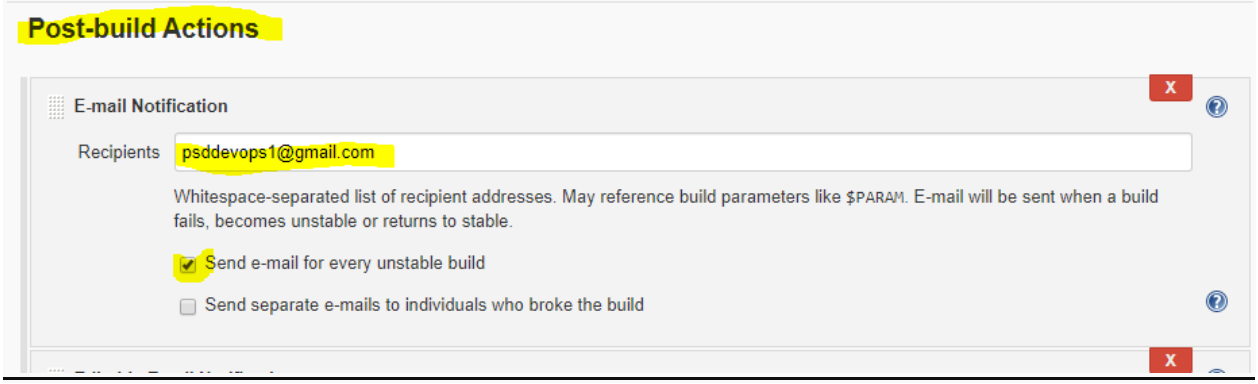
Under Recipients put teams mail id and save the configuration

Configuration:



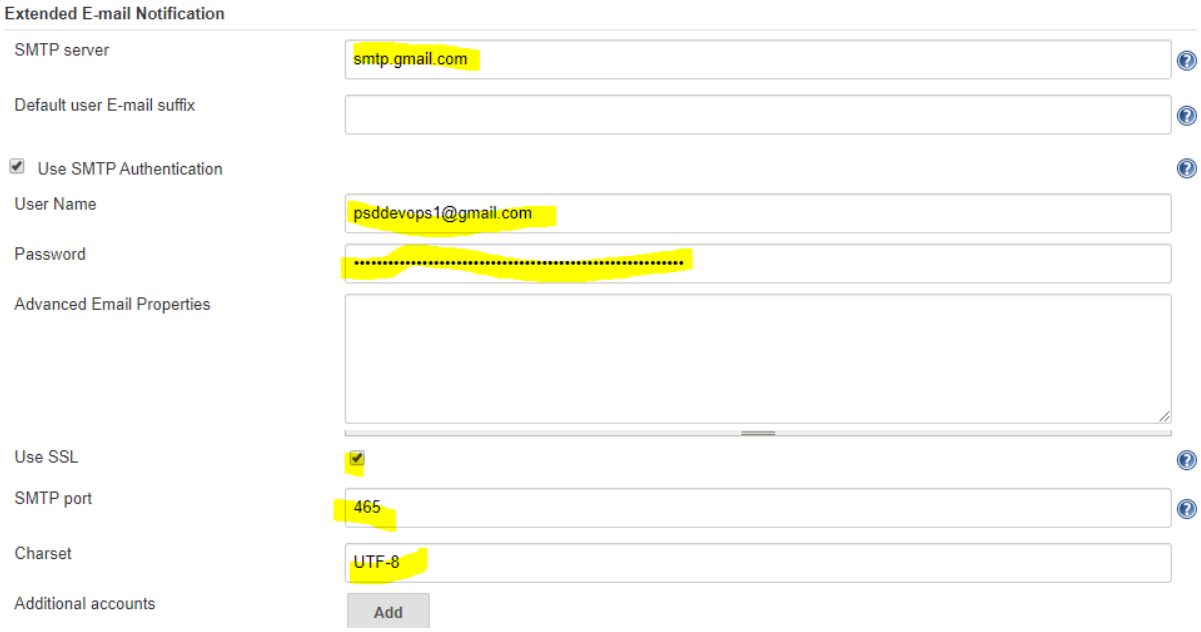
Job configuration:

Goto → Postbuild action → Email-notification



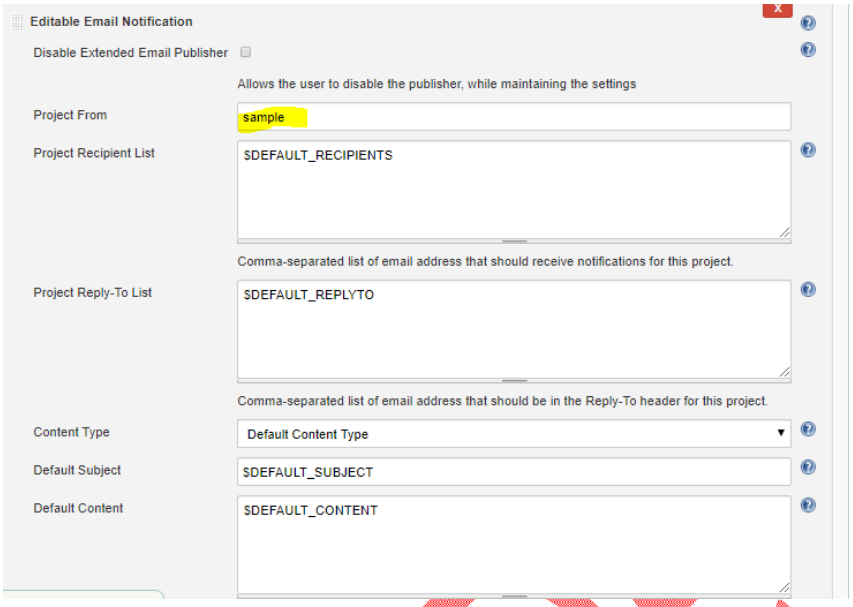
Email notifications with attachment

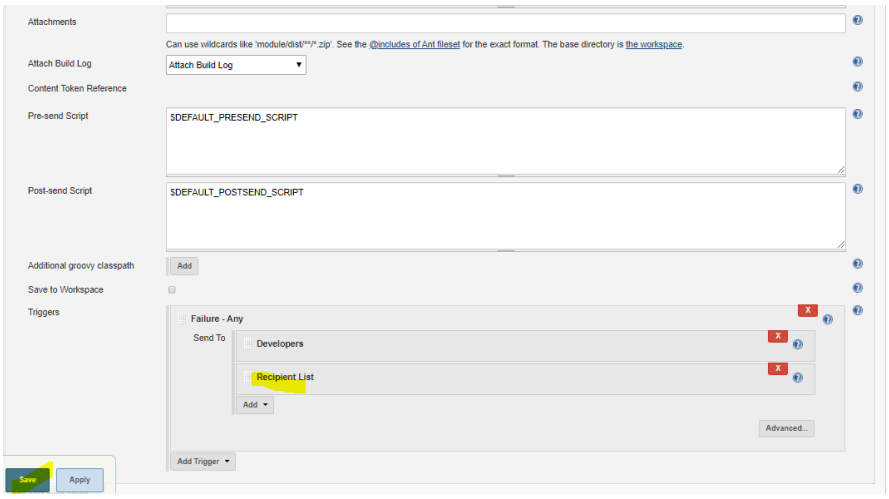
Jenkins → Manage Jenkins → Configure System → Extended E-mail Notification

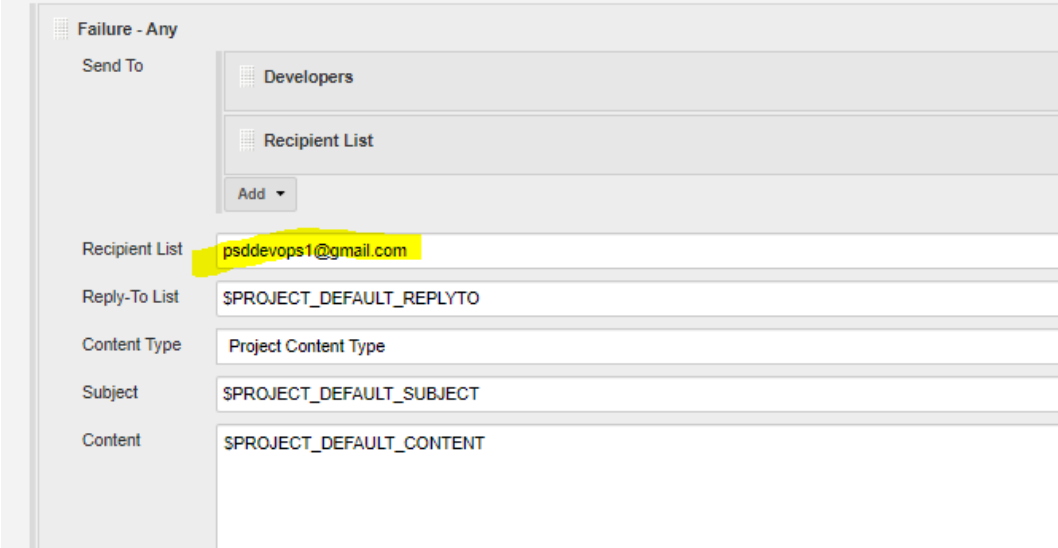


Job configuration:

Goto → Postbuild action →Editable Email-notification

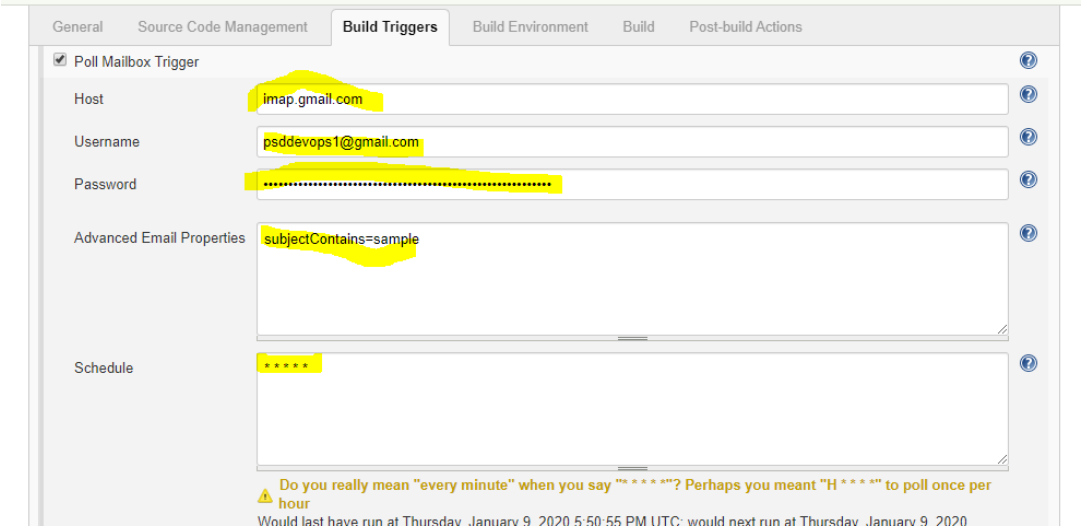






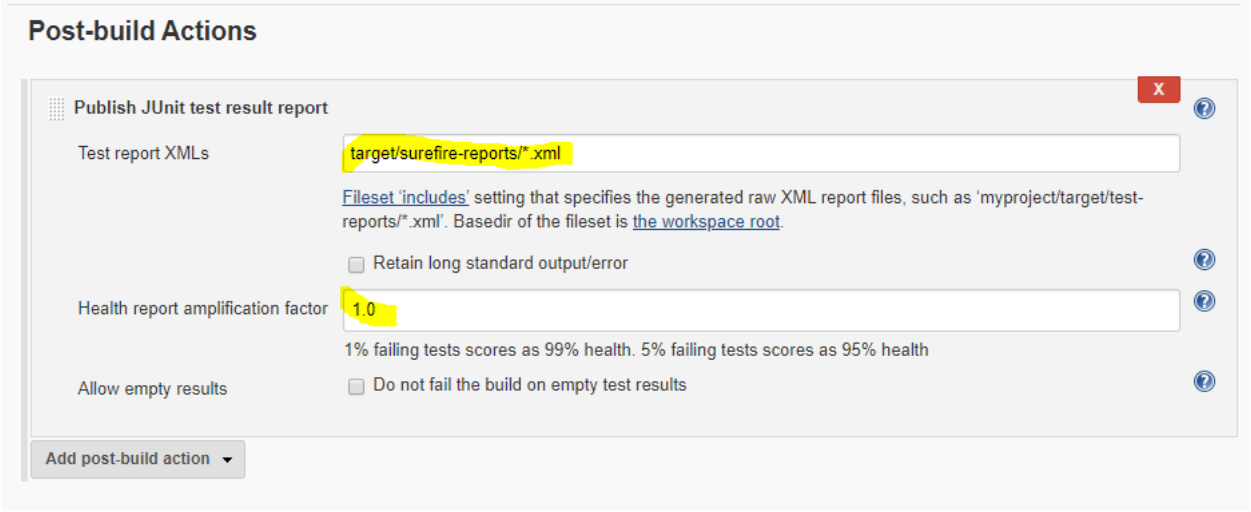
Poll mailbox:

Install the Poll Mailbox Trigger Plugin
Open the job configuration → Build triggers → Select Poll mail box trigger enter below highlighted details →Run the job.

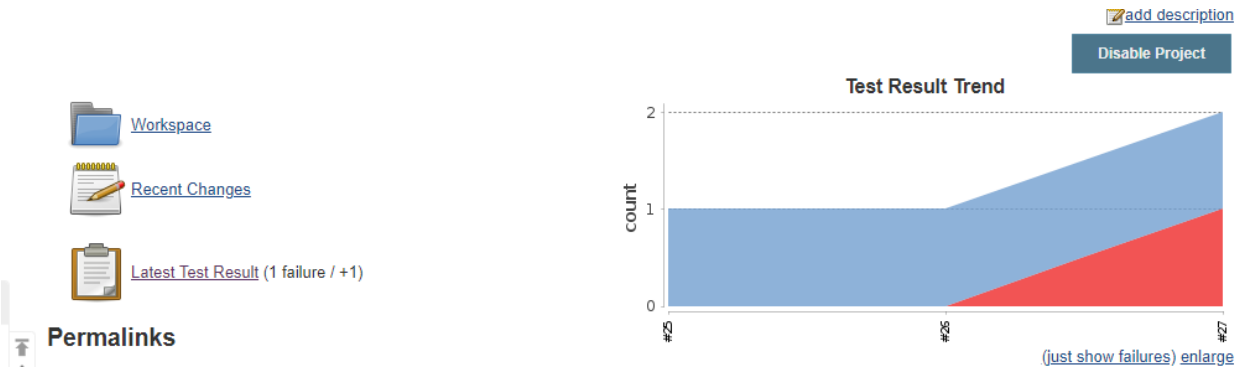


Integrating Junit test cases to the Jenkins

Go to job → Configuration → Postbuild actions →Publish Junit Test reports

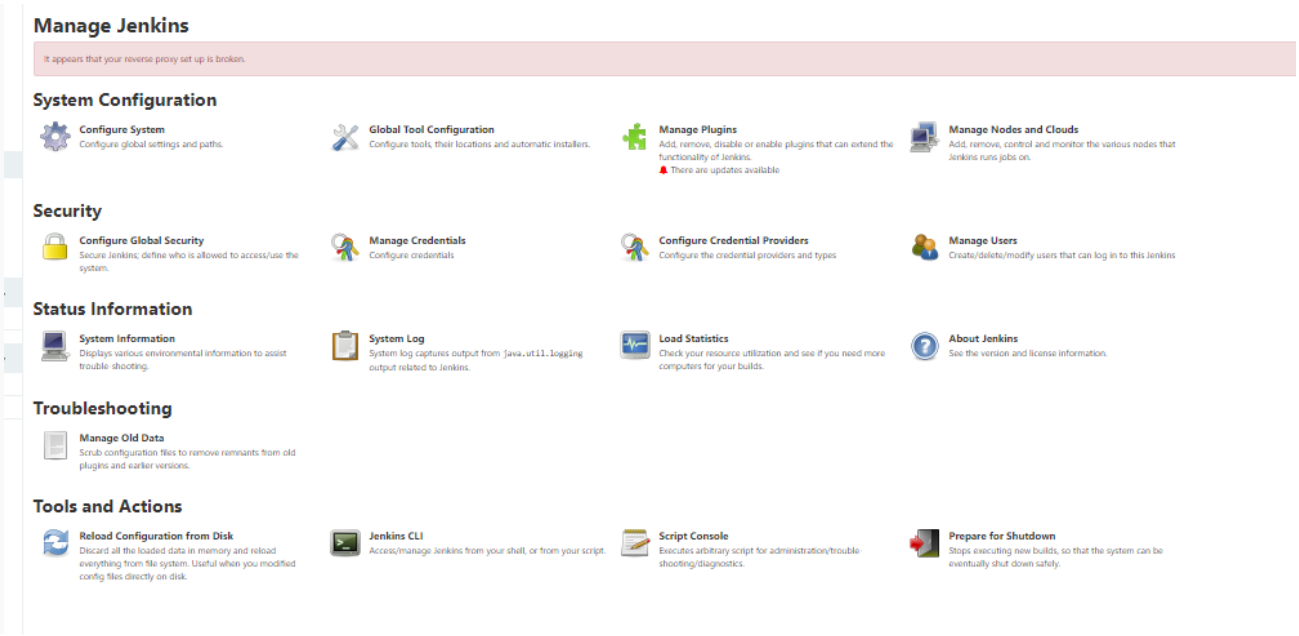


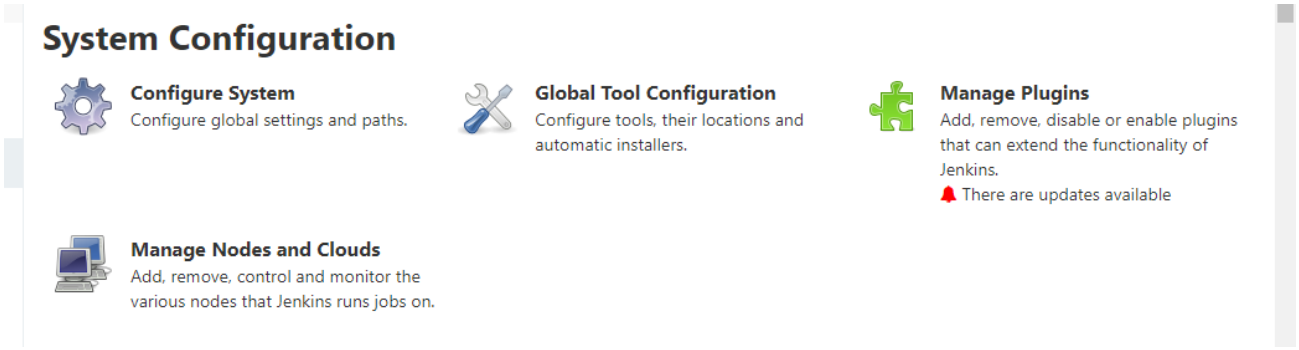
Run the job

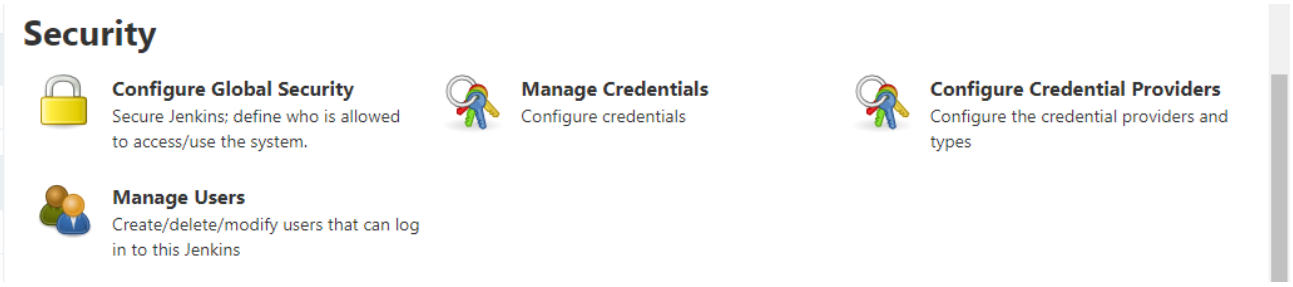


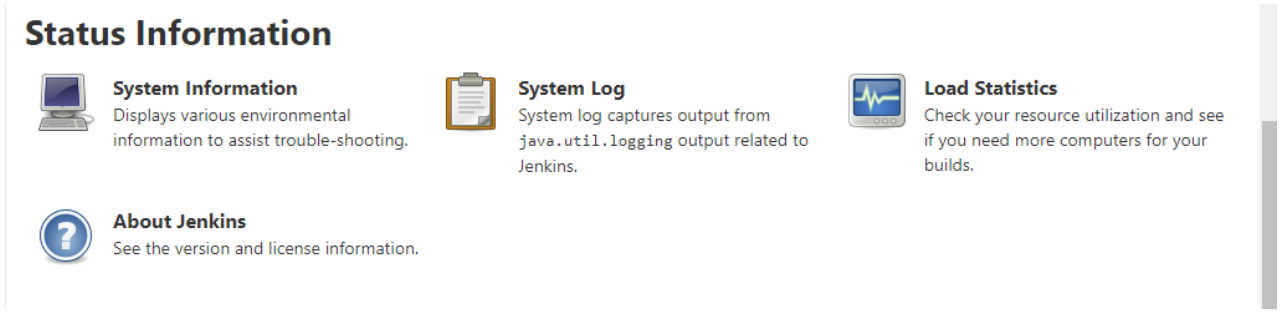
Manage Jenkins

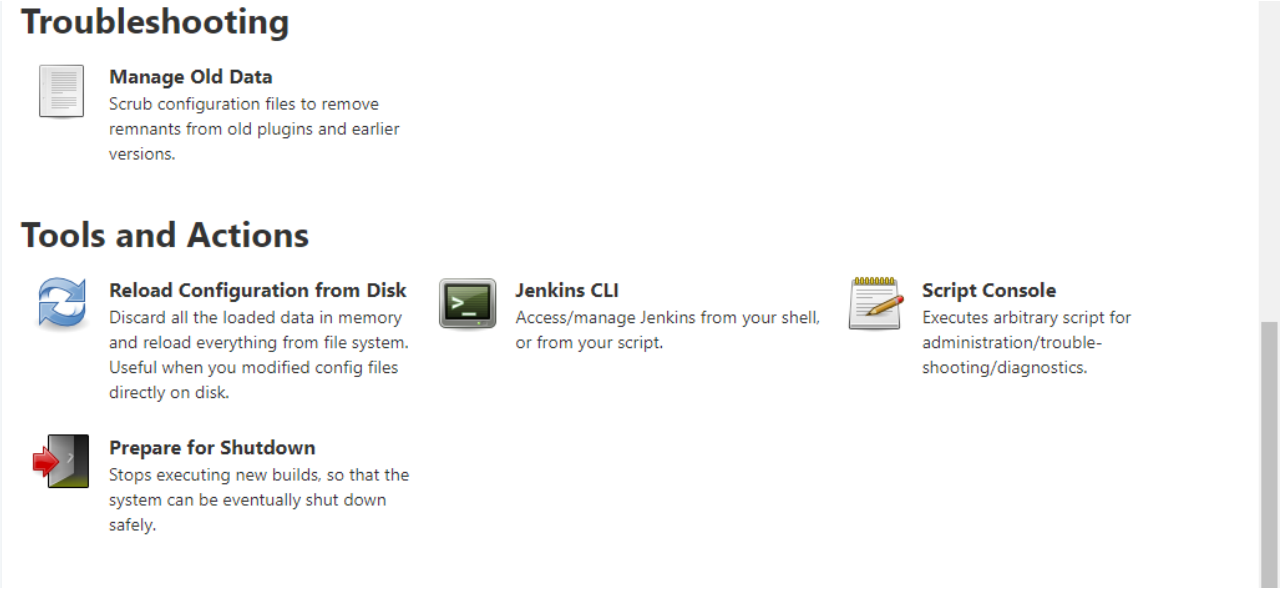
To manage Jenkins, click on the "Manage Jenkins" option from the left hand of the Jenkins Dashboard page.
When you click on the Manage Jenkins, you will get the various options to manage the Jenkins:


Manage Jenkins
It appears that your reverse proxy set up is broken.
System Configuration
Configure System
Configure global settings and paths.
Global Tool Configuration
Configure tools, their locations and automatic installers.
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
There are updates available
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Security
Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.
Manage Credentials
Configure credentials
Configure Credential Providers
Configure the credential providers and types
Manage Users
Create/delete/modify users that can log in to this Jenkins
Status Information
System Information
Displays various environmental information to assist trouble-shooting.
System Log
System log captures output from java.util.logging output related to Jenkins.
Load Statistics
Check your resource utilization and see if you need more computers for your builds.
About Jenkins
See the version and license information.
Troubleshooting
Manage Old Data
Scrub configuration files to remove remnants from old plugins and earlier versions.
Tools and Actions
Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.
Jenkins CLI
Access/manage Jenkins from your shell, or from your script.
Script Console
Executes arbitrary script for administration/trouble-shooting/diagnostics.
Prepare for Shutdown
Stops executing new builds, so that the system can be eventually shut down safely.


System Configuration
Configure System
Configure global settings and paths.
Global Tool Configuration
Configure tools, their locations and automatic installers.
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
There are updates available
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.


Security
Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.
Manage Credentials
Configure credentials
Configure Credential Providers
Configure the credential providers and types
Manage Users
Create/delete/modify users that can log in to this Jenkins





We have a lot of options in Jenkins Management page like:

System Configuration

- Configure System
- Global Tool Configuration
- Manage Plugins
- Manage Nodes and Cloud

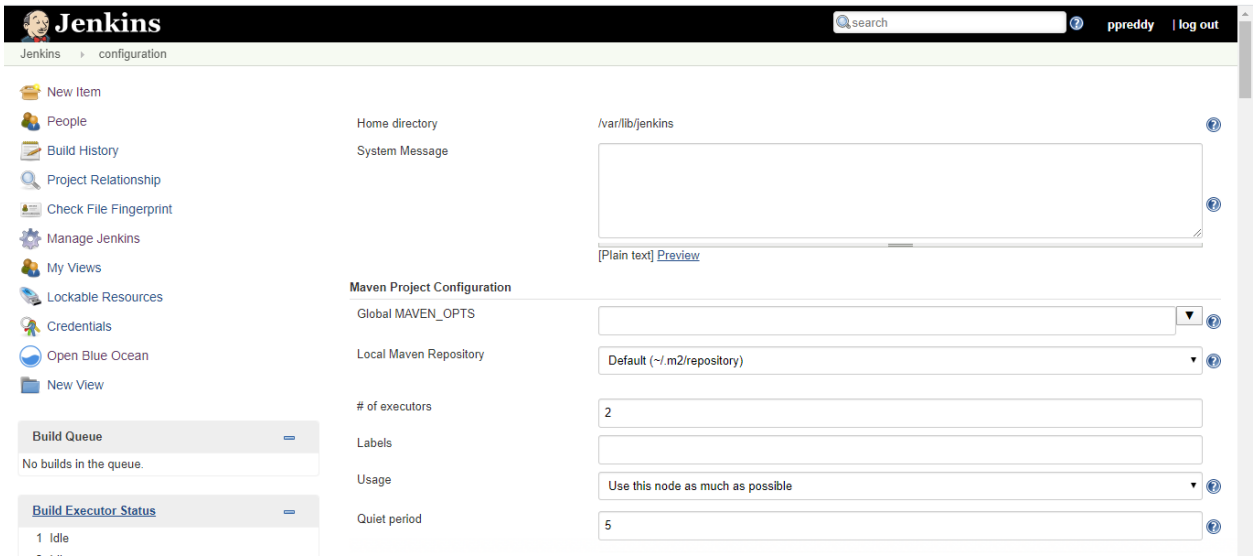
Security

- Configure Global Security
- Configure Credentials
- Reload Configuration from Disk
- System Information
- System Log
- Load Statistics
- Jenkins CLI

- Script Console
- About Jenkins
- Manage Old Data and
- Prepare for Shutdown

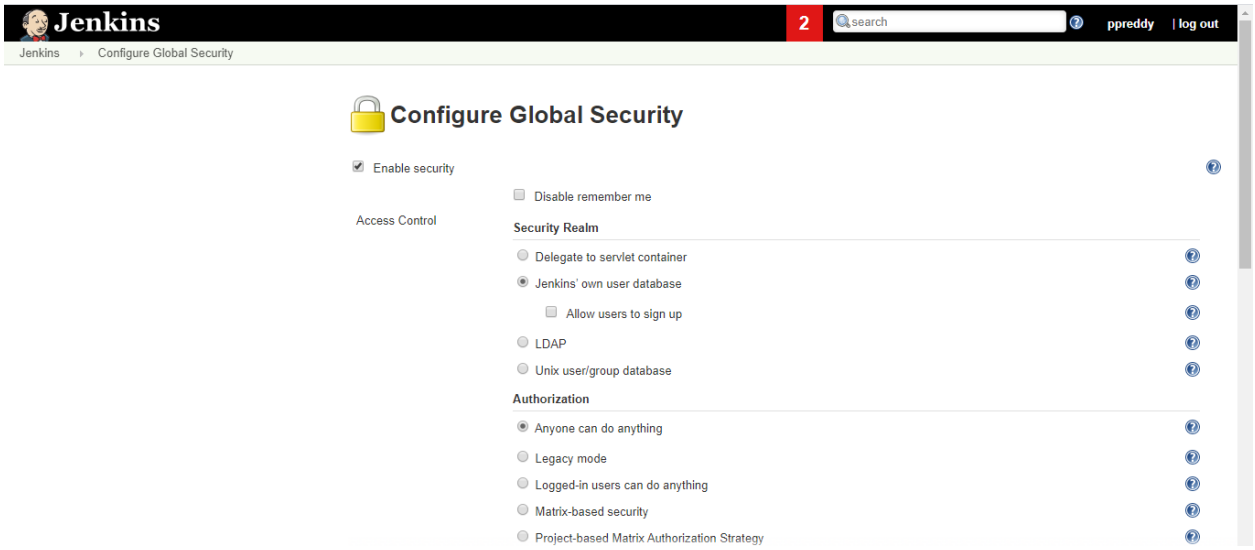
Configure System

In this, we can manage paths to the various tools to use in builds, such as the versions of Ant and Maven, as well as security options, email servers, and other system-wide configuration details. Jenkins will add the required configuration fields dynamically when new plugins are installed.



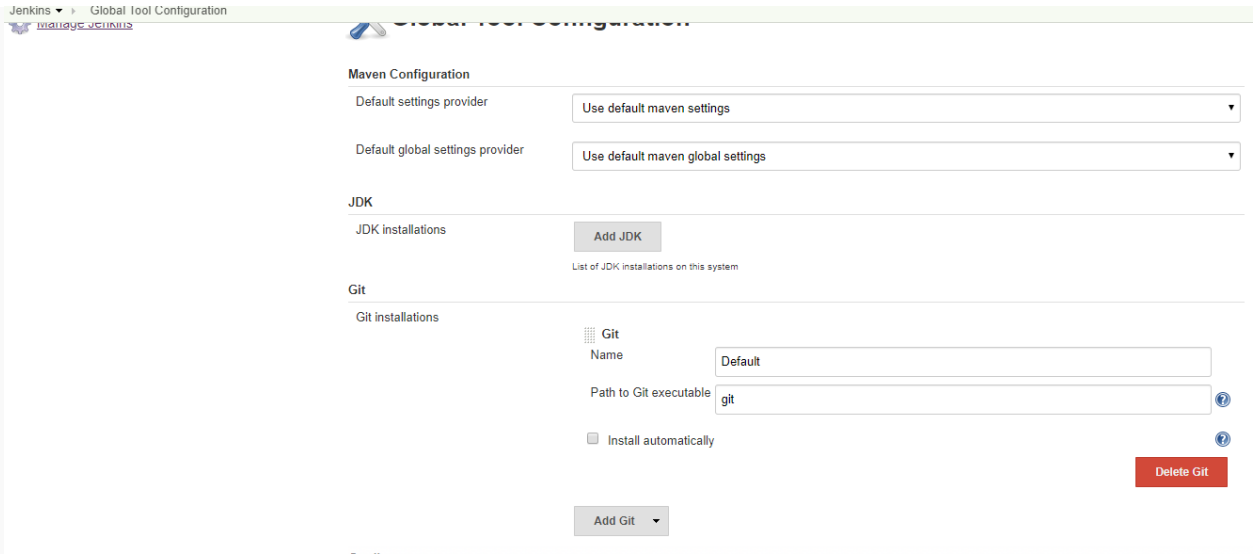
Configure Global Security

Configure Global Security option provides the ability to set up users and their relevant permissions on the Jenkins instance. By default, you will not want everyone to be able to define builds or other administrative tasks in Jenkins. So Jenkins provides the ability to have a security configuration in place.



Global Tool Configuration

In this section, we will configure various tools that we need to utilize at the time of creating a build job, for example, Java, Ant, Maven, and so on.



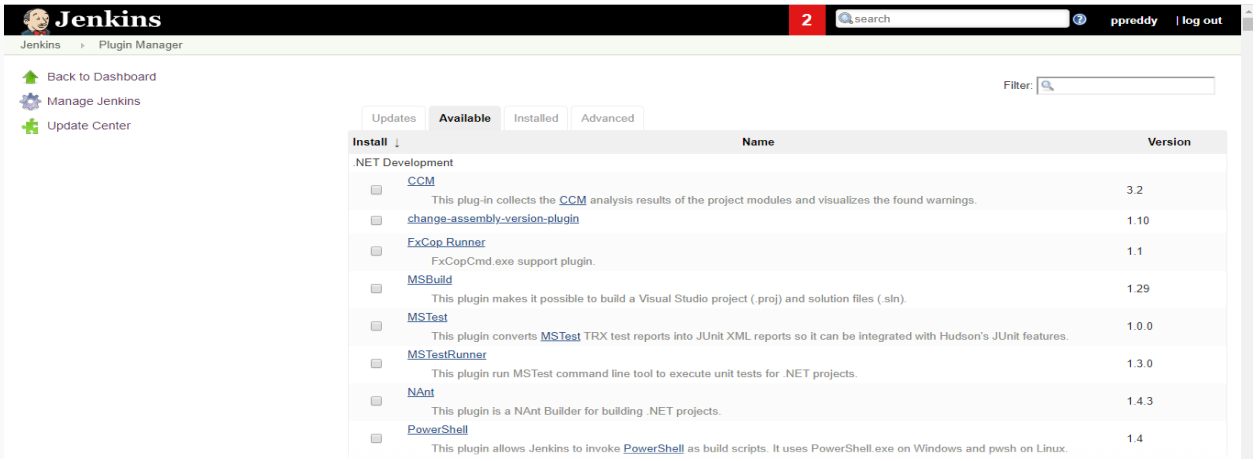
Reload Configuration from Disk

Jenkins stores all its system and job configuration details as XML files and all XML files are stored in the Jenkins home directory. Jenkins also stores all of the build histories.If you are moving build jobs from one Jenkins instance to another or archiving old build jobs, you will have to insert or remove the corresponding build

job directories to Jenkins's builds directory. You do not have to take Jenkins offline to do this?you can simply use the "Reload Configuration from Disk" option to reload the Jenkins system and build job configurations directly.

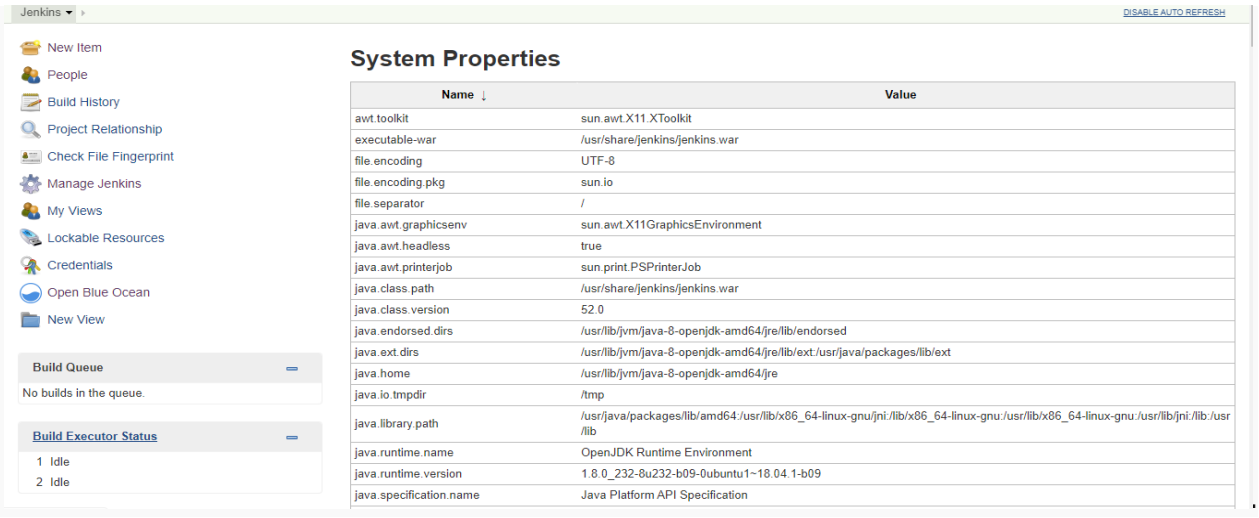
Manage Plugin

Here you can install a wide or different variety of third-party plugins from different Source code management tools such as Git, ClearCase or Mercurial, to code quality and code coverage metrics reporting. We can download, install, update, or remove the plugins from the Manage Plugins screen.



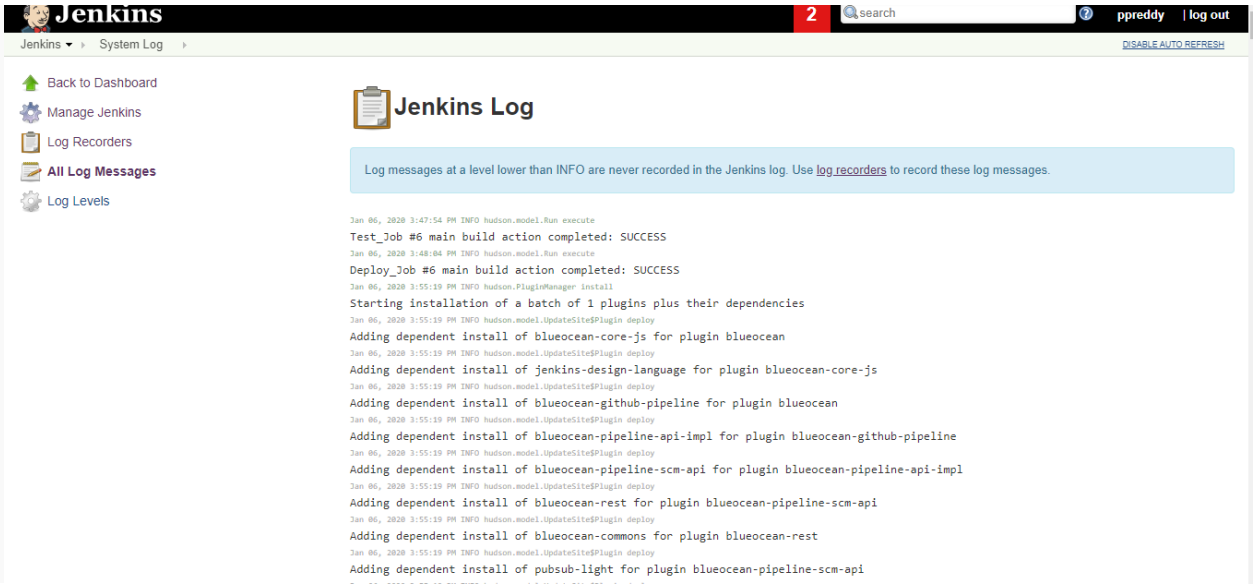
System Information

This option displays a list of all the current Java system properties and system environment variables. Here you can check what version of Java is currently running in, what user it is running under, and so forth



System Log

The System Log option is used to view the Jenkins log files in real time. As well as, the main use of this option is for troubleshooting.

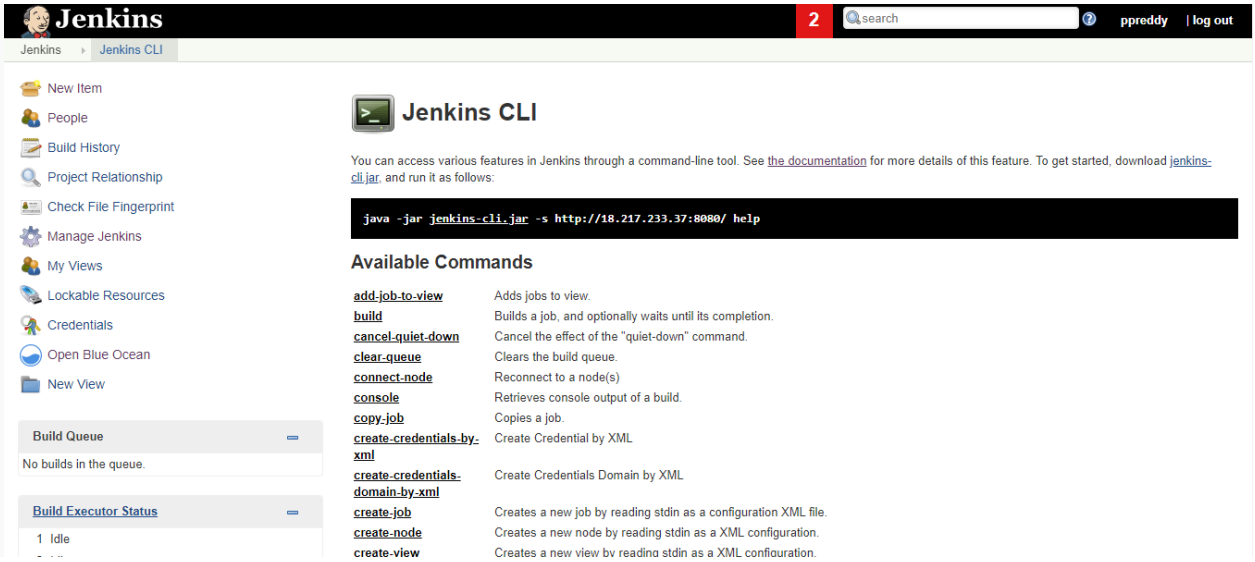


Load Statistics

This option is used to see the graphical data on how busy the Jenkins instance is in terms of the number of concurrent builds and the length of the build queue which provides an idea of how long your builds need to wait before being executed. These statistics can show you a good idea of whether extra capacity or extra build nodes are required from an infrastructure perspective.

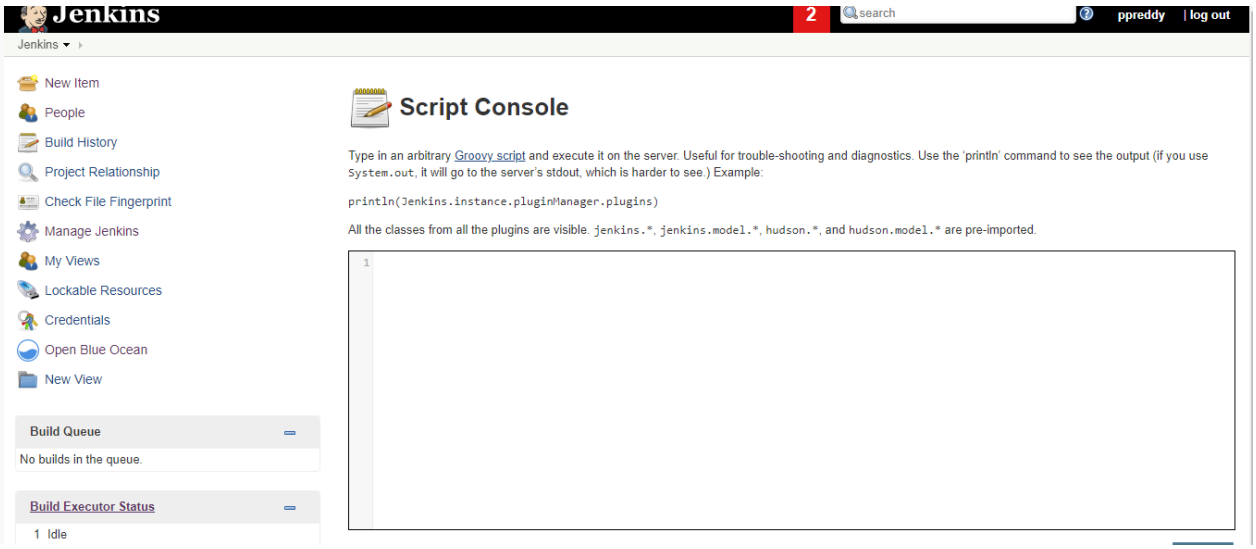
Jenkins CLI

Using this option, you can access various features in Jenkins through a command-line. To run the Jenkins through cli, first you have to download the Jenkins-cli.jar file and run it on your command prompt as follows:
java -jar jenkins-cli.jar -s *http://localhost:8080/jenkins/*
This page gives the list of available commands with their description.



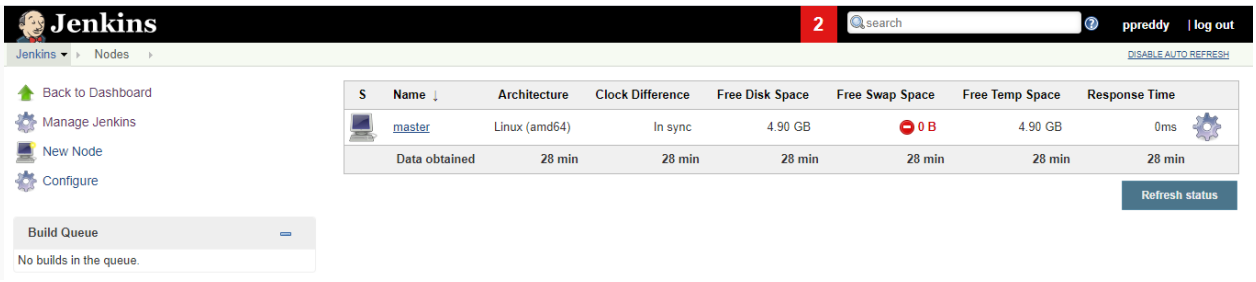
Script Console

This option allows you to run Groovy scripts on the server. This option is very useful for advanced troubleshooting since it requires a strong knowledge of the internal Jenkins architecture.



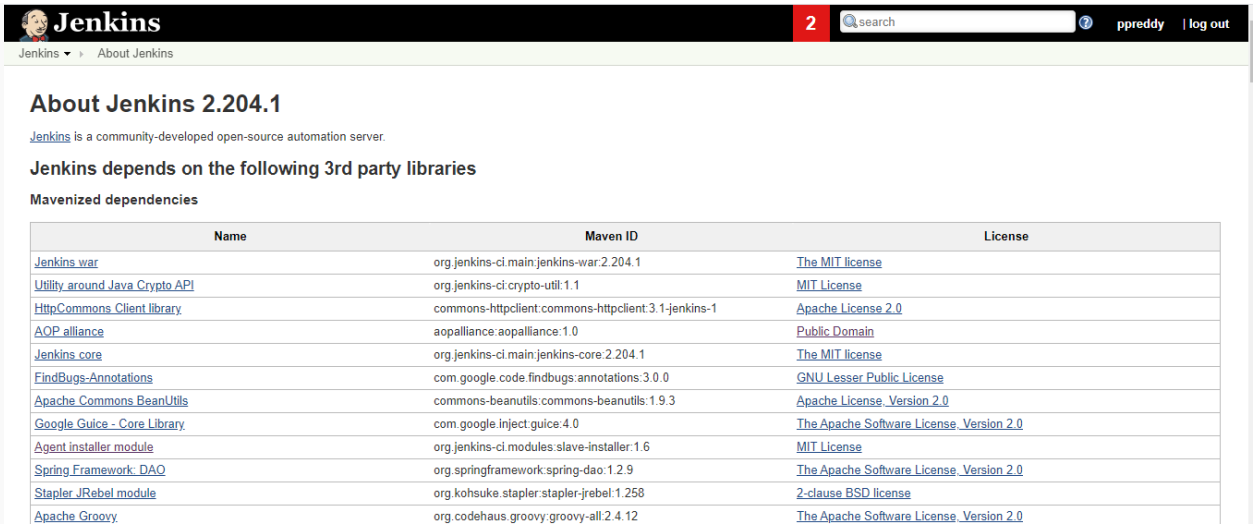
Manage nodes

Jenkins can handle parallel and distributed builds. In this page, you can configure how many builds you want. Jenkins runs concurrently, and, if you are using distributed builds, set up builds nodes. A build node (slave) is another machine that Jenkins can use to execute its builds.



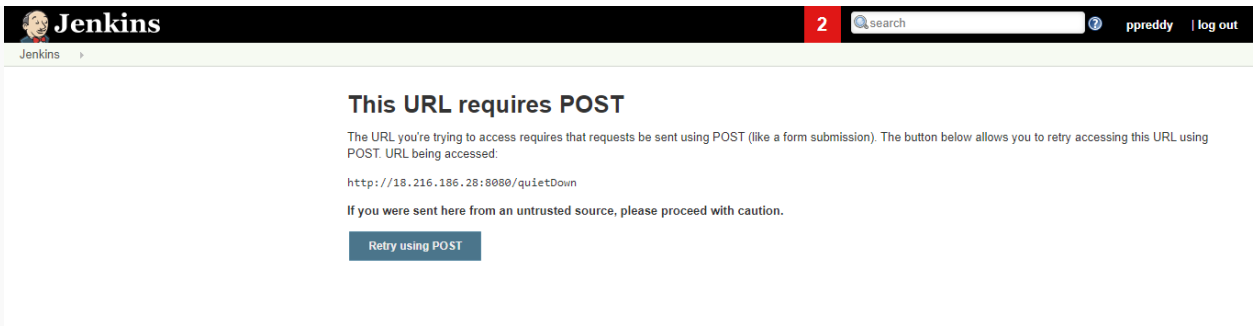
About Jenkins

This option will show the version and license information of the Jenkins you are running. As well as it displays the list of all third party libraries.



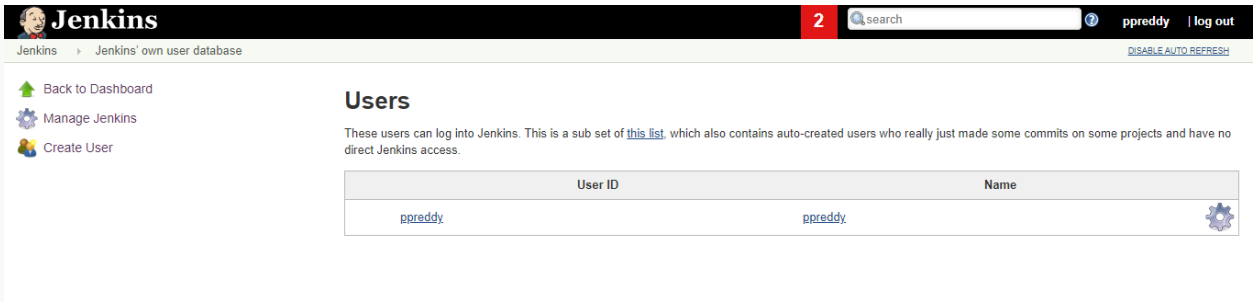
Prepare for Shutdown

If you want to shut down Jenkins, or the server Jenkins is running on, it is best not to do so when a build is being executed. To shut down Jenkins cleanly, you can use the Prepare for Shutdown link, which prevents any new builds from being started. Eventually, when all of the current builds have completed, you will be able to shut down Jenkins cleanly.



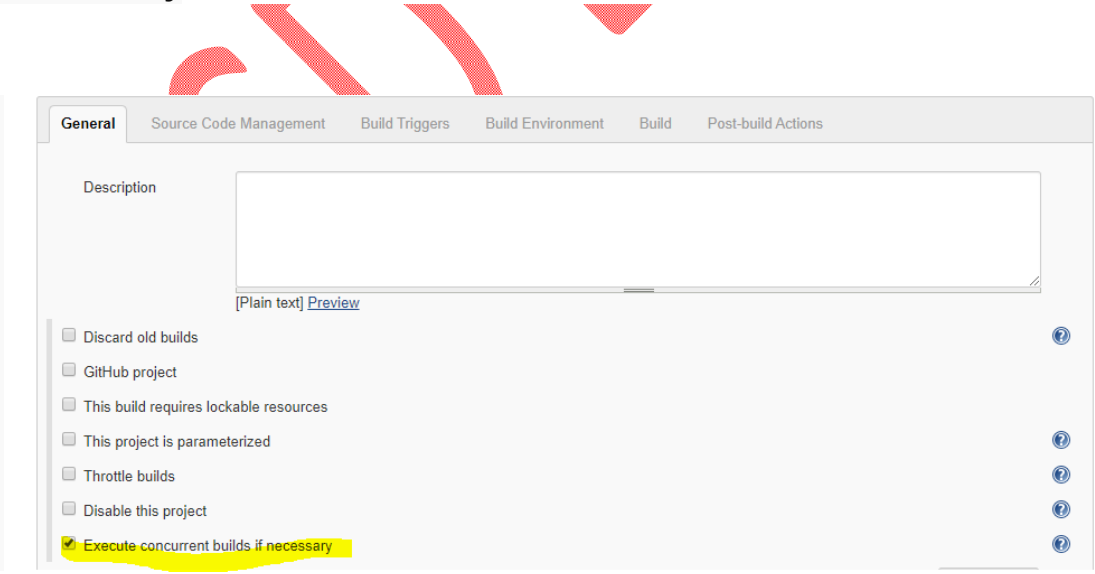
Manage users

Add and delete the users in the Jenkins server.



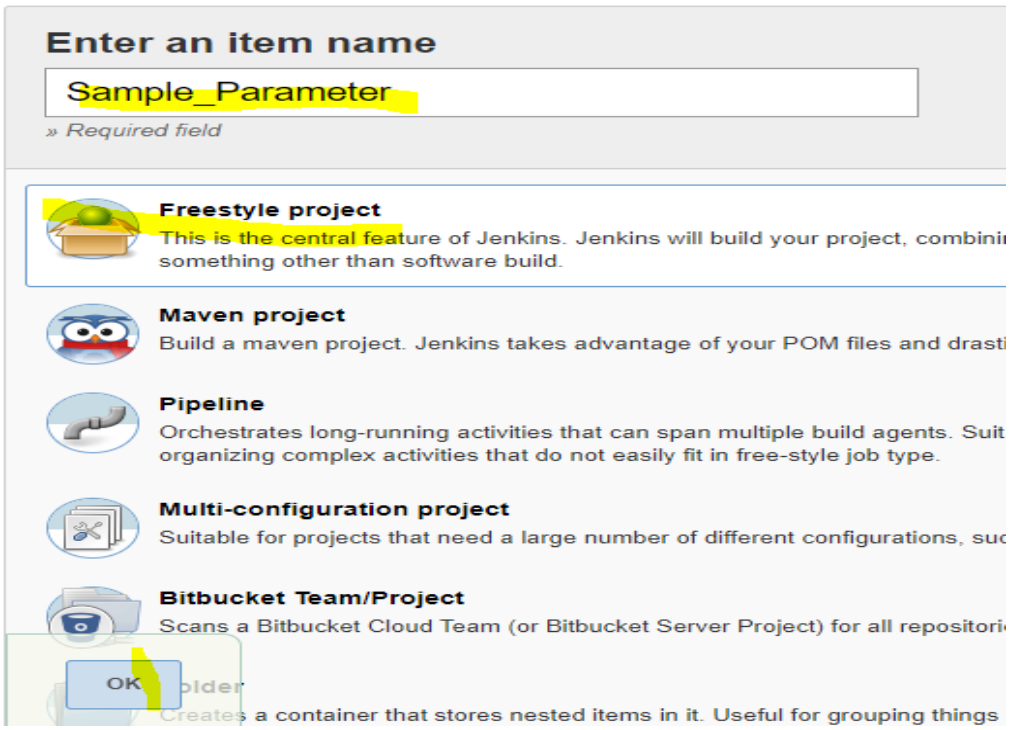
Execute the same job in parallel

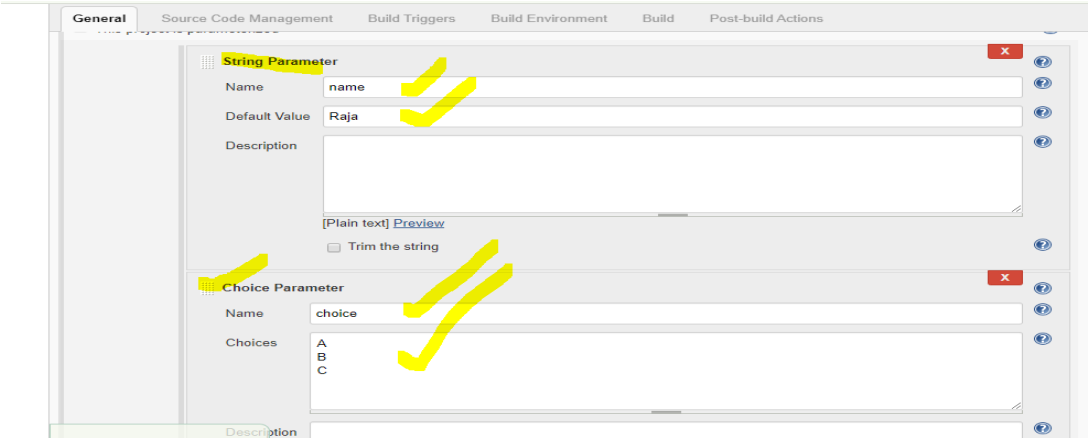
Open the job → Go to configuration → General → Select **Execute concurrent builds if necessary**

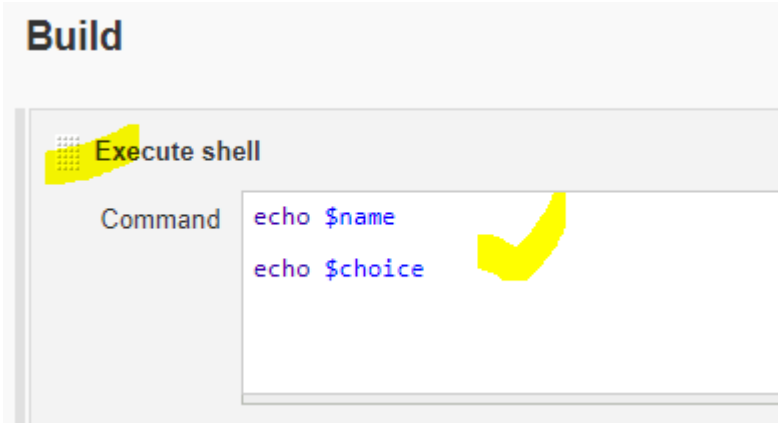


Parameterized jobs in Jenkins

1. Create sample free style job

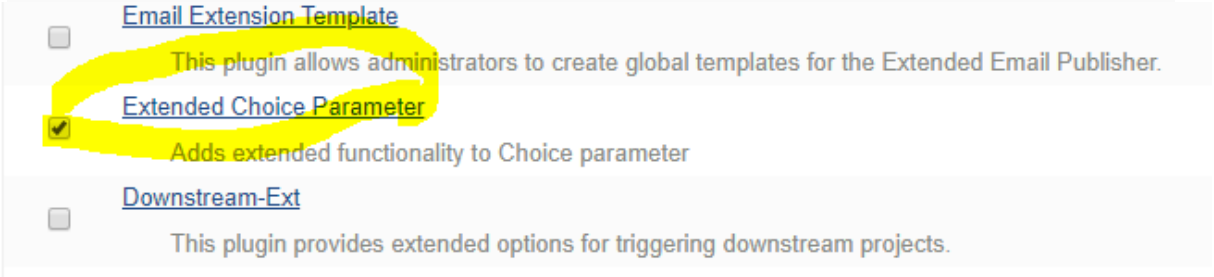




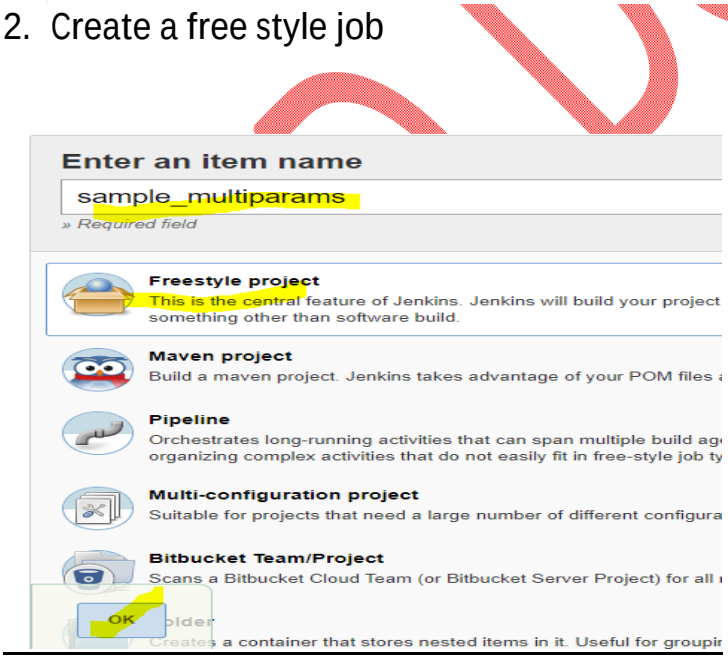


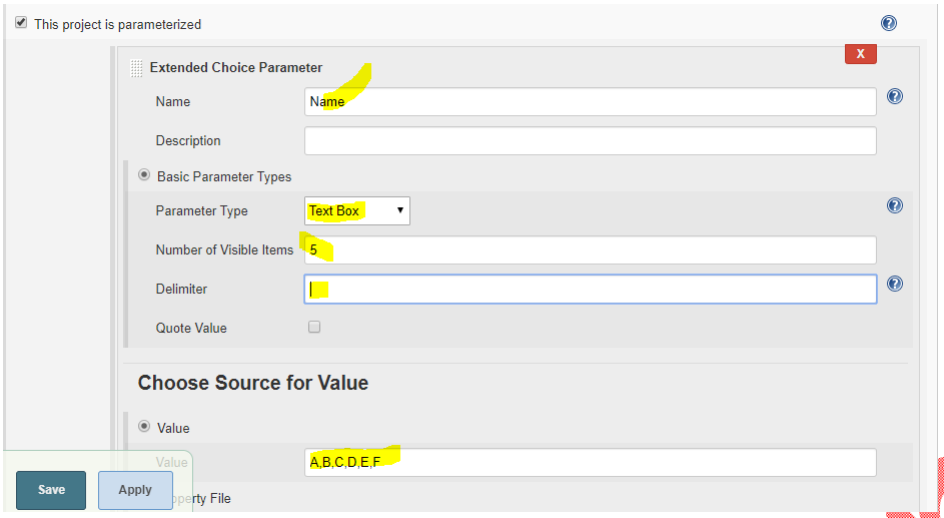
Parameters with radio buttons ,check boxes and Dropdown

1. Install Extended Choice Parameter
   Jenkins → manage jenkins → manage plugins →Extended Choice Parameter



2. Create a free style job

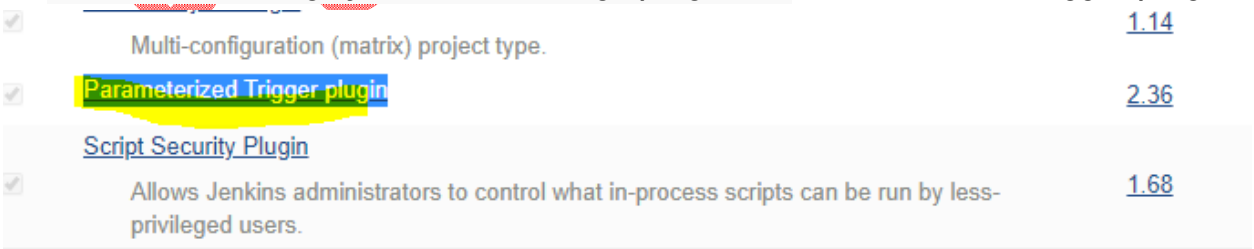




How to pass parameters to downstream job

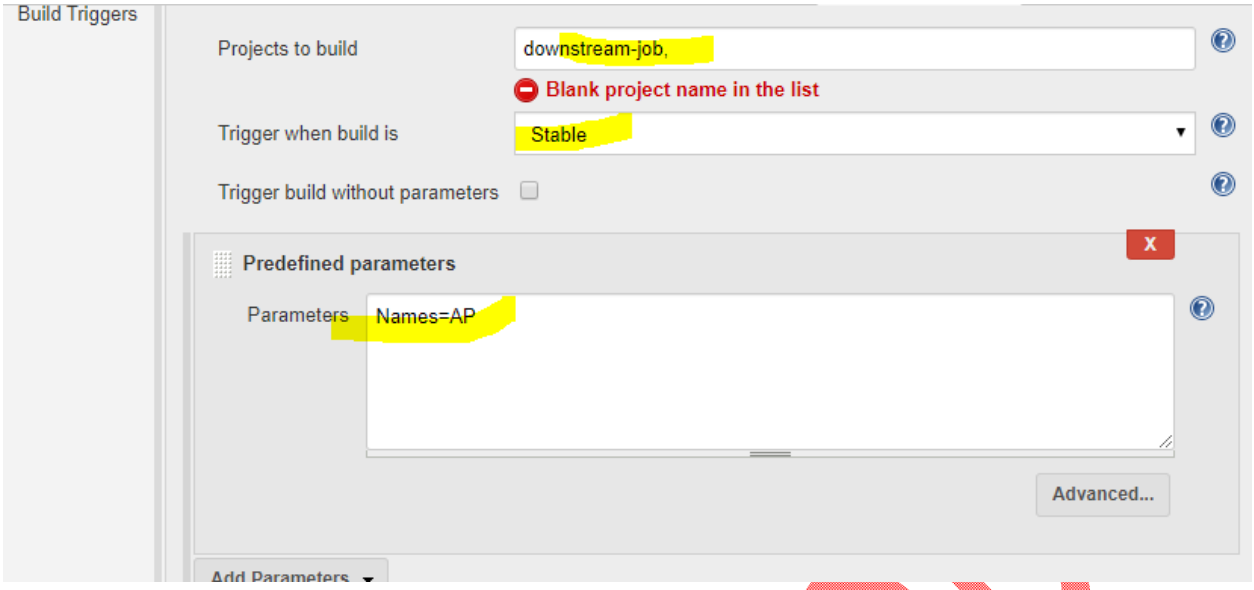
1. Install **Parameterized Trigger** plugin

   Jenkins → manage jenkins → manage plugins →Parameterized Trigger plugin

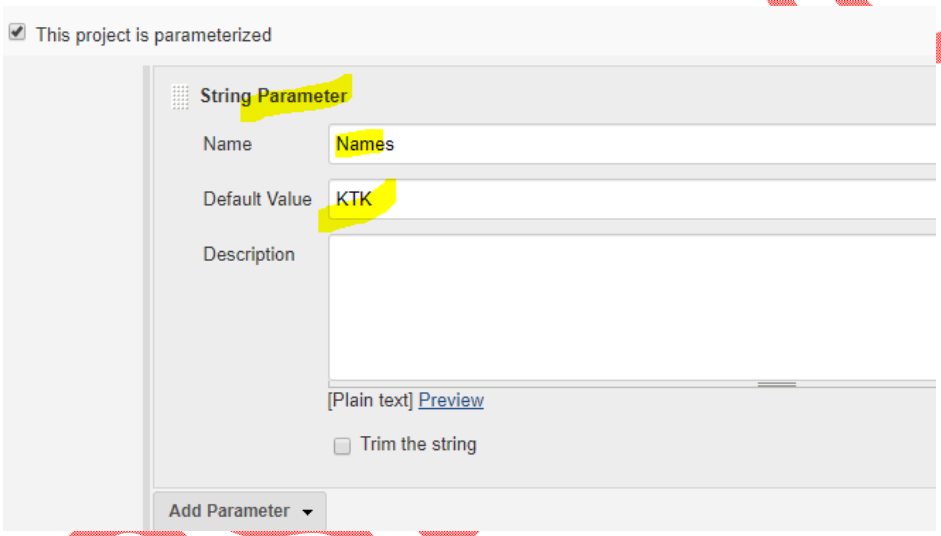


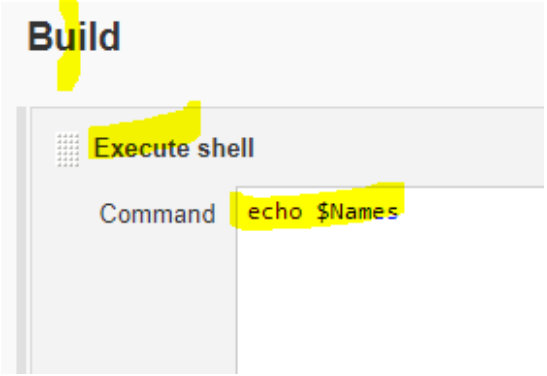
2. Create two jobs upstream-job and downstream-job

Upstream


Build Triggers
Projects to build
downstream-job,
Blank project name in the list
Trigger when build is
Stable
Trigger build without parameters
Predefined parameters
Parameters
Names=AP
Advanced...
Add Parameters

Downstream


This project is parameterized
String Parameter
Name
Names
Default Value
KTK
Description
[Plain text] Preview
Trim the string
Add Parameter


Build
Execute shell
Command
echo $Names

Jenkins CLI

Jenkins has a built-in command line interface that allows users and administrators to access Jenkins from a script or shell environment. This can be convenient for scripting of routine tasks, bulk updates, troubleshooting, and more.

Adv:

- Easier
- Faster
- Memory mgmt
- CI

1. Download the jenkins-cli.jar from jenkins

Jenkins →manage jenkins →Jenkins CLI

Running the job from CMD

java -jar jenkins-cli.jar -s http://18.216.186.28:8080/ build sample_test

java -jar jenkins-cli.jar -s http://18.216.186.28:8080/ build sample_test -s –v (with output)

java -jar jenkins-cli.jar -s http://18.216.186.28:8080/ build sample_test -s -v –username ppreddy – password ppreddy

java -jar jenkins-cli.jar -s http://18.216.186.28:8080/ build downstream-job -p Names=APL(with params)

```
java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy -webSocket help

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy version

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy list-jobs

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build sample_test

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build sample_test -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build sample_parameters -p Name="raja" -p State="AP" -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build ci-job -p TARGET_BRANCH="master" -p Environment="Dev" -p Service="true"

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy safe-restart
```

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy delete-job test

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy disable-job myjob

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth vamsi:11dca2731ff681228a221752922a7d79ea disable-job myjob

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f list-jobs

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f build sample_test

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f build sample_test -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f build sample_parameters -p Name="raja" -p State="AP" -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f delete-job sample

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f safe-restart

Login Jenkins using username and Password

java -jar jenkins-cli.jar -s http://localhost:8080/ help --username devops --password passw0rd

Get the Version of Jenkins

#java -jar jenkins-cli.jar -s http://localhost:8080/ version --username devops --password passw0rd

Get all the jobs of Jenkins

#java -jar jenkins-cli.jar -s http://localhost:8080/ list-jobs --username devops --password passw0rd

Delete the Job

#java -jar jenkins-cli.jar -s http://localhost:8080/ delete-job ant-java-job-dev --username devops --password passw0rd

#java -jar jenkins-cli.jar -s http://localhost:8080/ disable-job ant-web-job-dev --username devops --password passw0rd

While executing above command if you see Enter passphrase, follow the below configuration.
(Manage Jenkins ---> Configure Global Security ---> enable the Enable Security ---> Apply and Save.)

Manage Jenkins ---> Configure System --->SSH Public Keys (Enter here any value, same value u can use in CLI)

Jenkins CLI Commands

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy -webSocket help

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy version

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy list-jobs

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build sample_test

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build sample_test -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build sample_parameters -p Name="raja" -p State="AP" -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy build ci-job -p TARGET_BRANCH="master" -p Environment="Dev" -p Service="true"

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy safe-restart

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy delete-job test

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:ppreddy disable-job myjob

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth vamsi:11dca2731ff681228a221752922a7d79ea disable-job myjob

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f list-jobs

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f build sample_test

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f build sample_test -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f build sample_parameters -p Name="raja" -p State="AP" -s -v

java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f delete-job sample java -jar jenkins-cli.jar -s http://3.23.64.120:8080/ -auth ppreddy:116f789cba0e7b67e12b52c2803798401f safe-restart

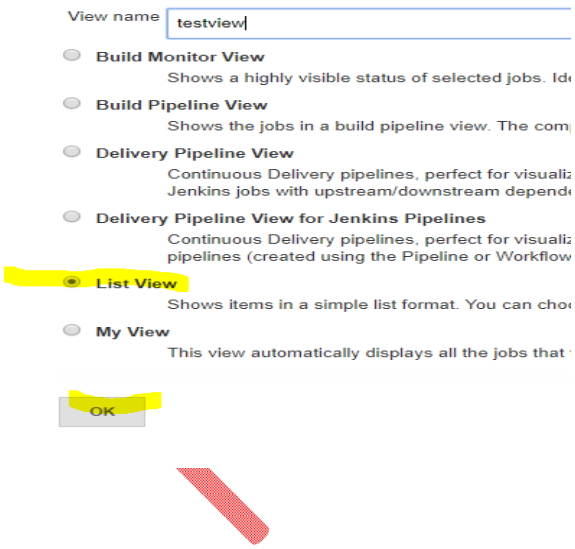
Views

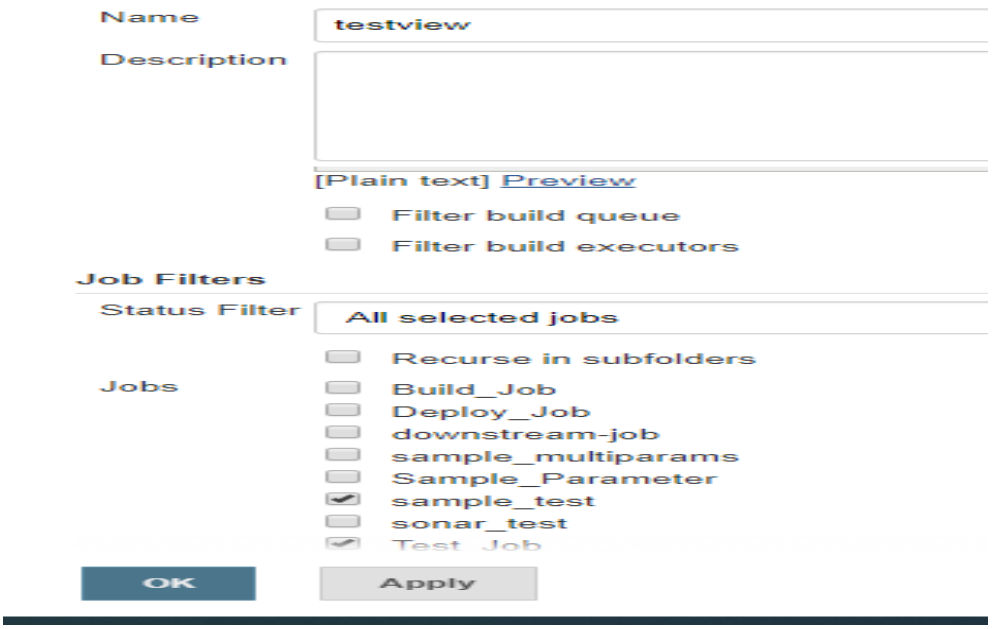
Views in Jenkins allow us to organize jobs and content into tabbed categories, which are displayed on the main dashboard. As a Jenkins instance expands, it is logical to create associated views for appropriate groups and categories.

Ex:

List view
Delivery Pipeline view
Build Pipeline view
Build Monitor view
Delivery Pipeline View for Jenkins Pipelines

List view





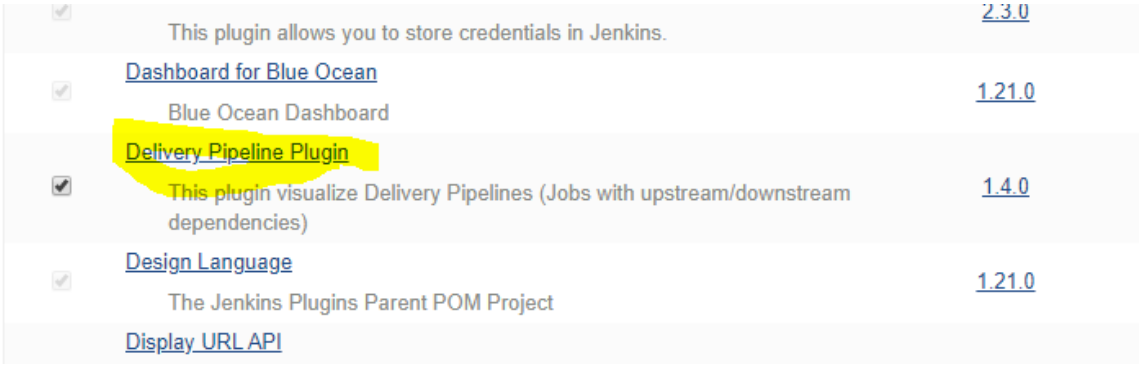
Delivery Pipeline

This plugin visualize Delivery Pipelines (Jobs with upstream/downstream dependencies).

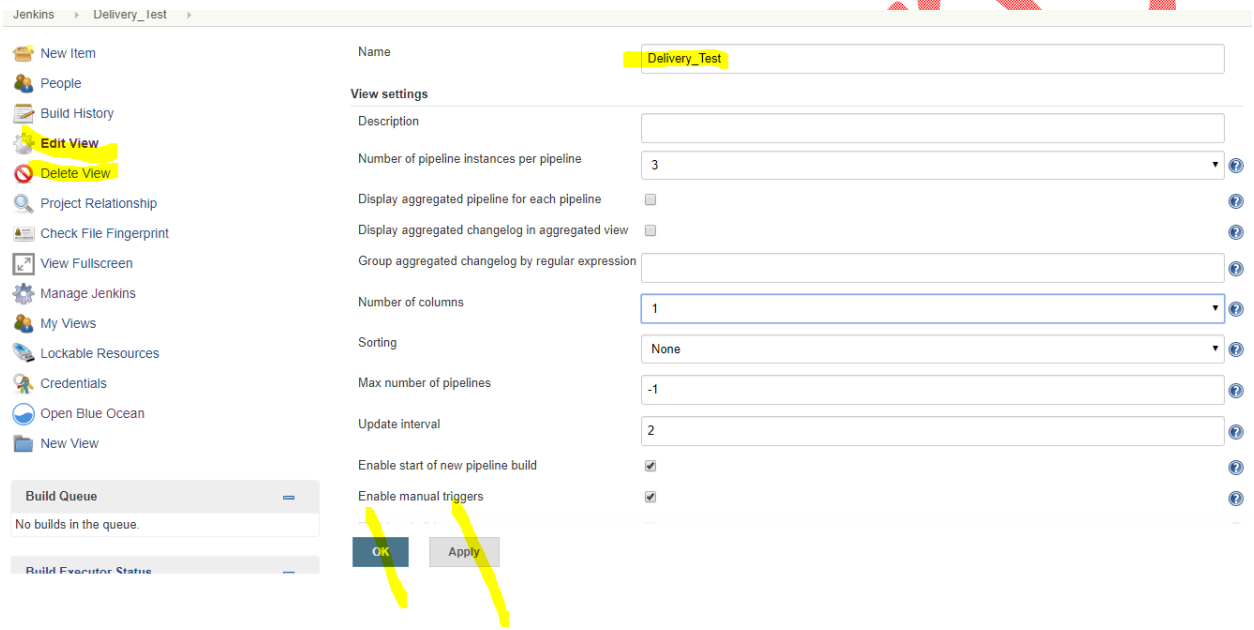
Visualization of Continuous Delivery pipelines. Renders pipelines based on upstream/downstream jobs or Jenkins pipelines. Provides a full screen view for information radiators. In Continuous Delivery, feedback and visualization of the delivery process is one of the most important aspects. When using Jenkins as a build/CI/CD server, it is with the Delivery Pipeline plugin possible to visualize one or more delivery pipelines in the same view, even in full screen.

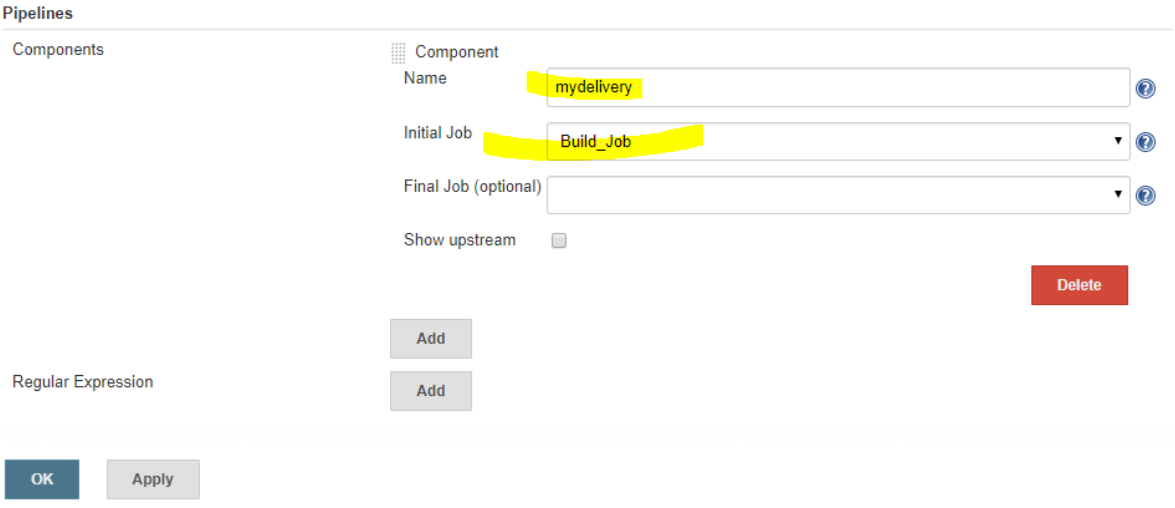
1. Chain required jobs in sequence
   Add upstream/downstream jobs
   Job A --> Job B --> Job C

2. Install Delivery Pipeline Plugin
   Jenkins → manage jenkins → manage plugins → Delivery Pipeline

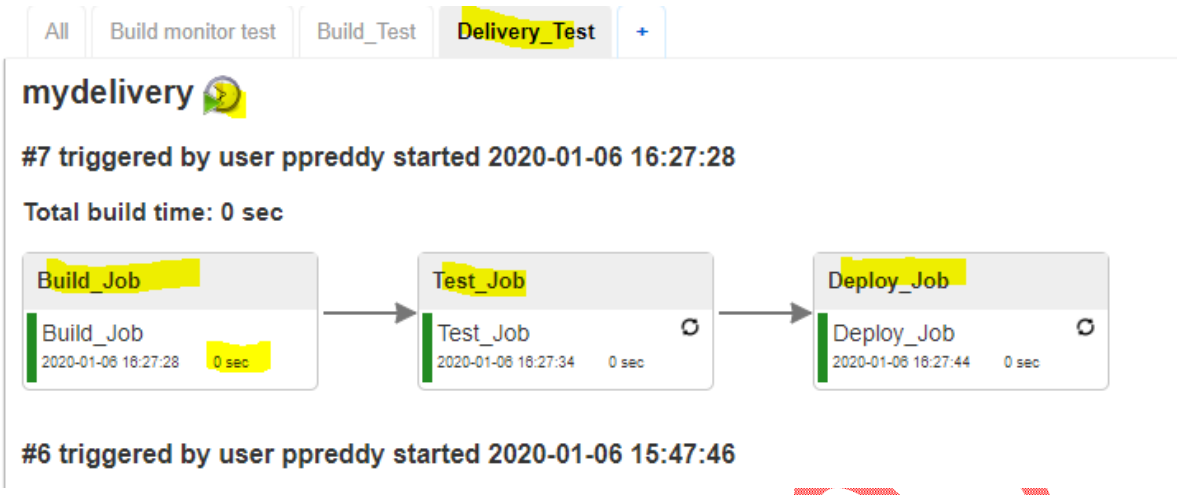


3. Add Delivery Pipeline View and configure the view





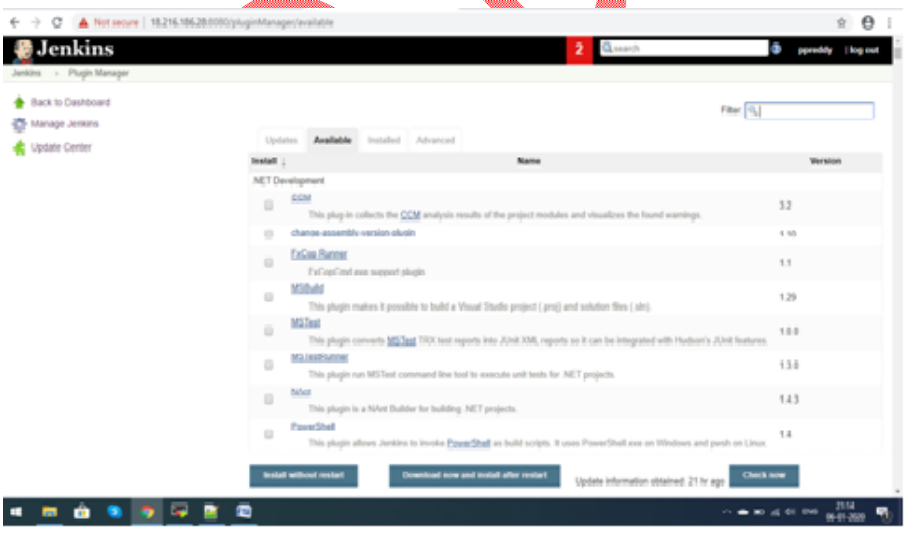
4. Run and Validate



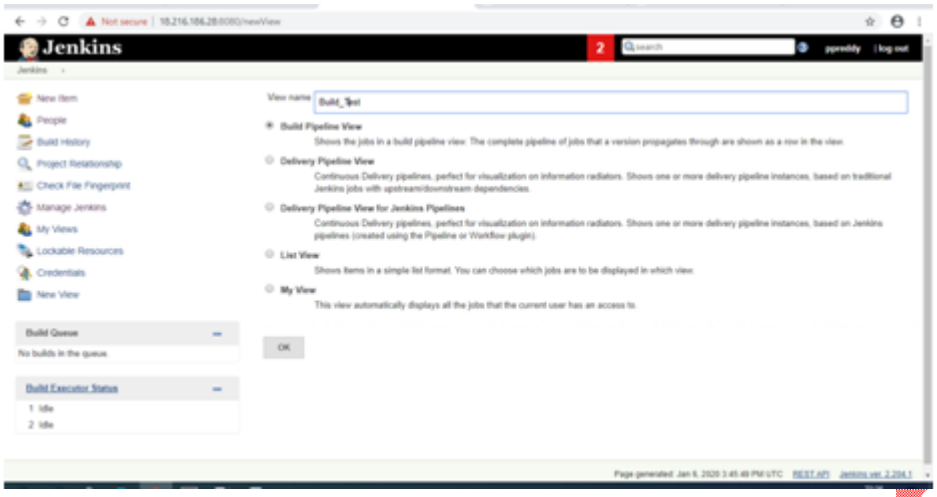
BUILD PIPELINE

This plugin renders upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.

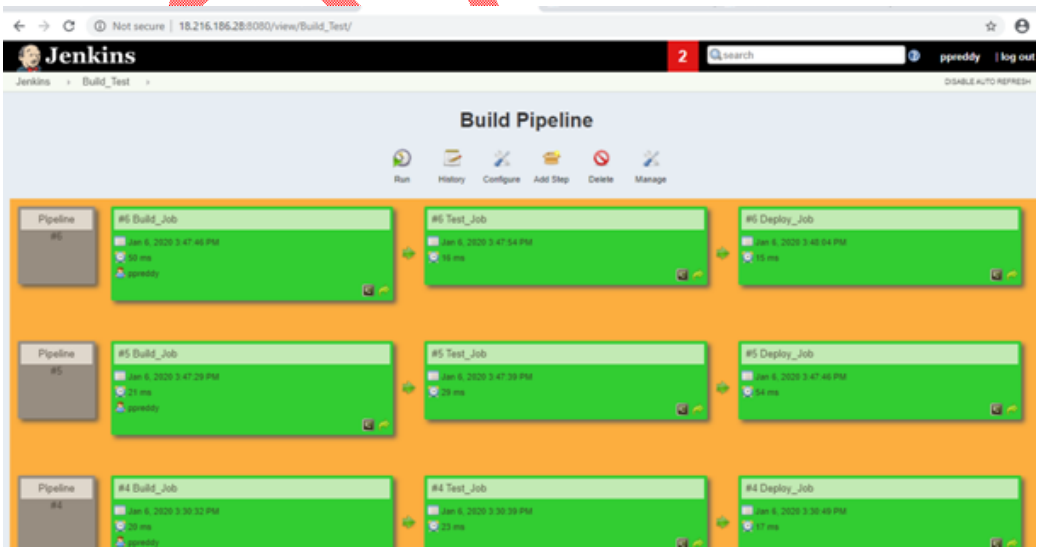
1. Chain required jobs in sequence
   Add upstream/downstream jobs
   Job A --> Job B --> Job C
2. Install Build Pipeline Plugin
   Jenkins → manage jenkins → manage plugins →Build Pipeline Plugin



3. Add Build Pipeline View and configure the view



4.Run and Validate

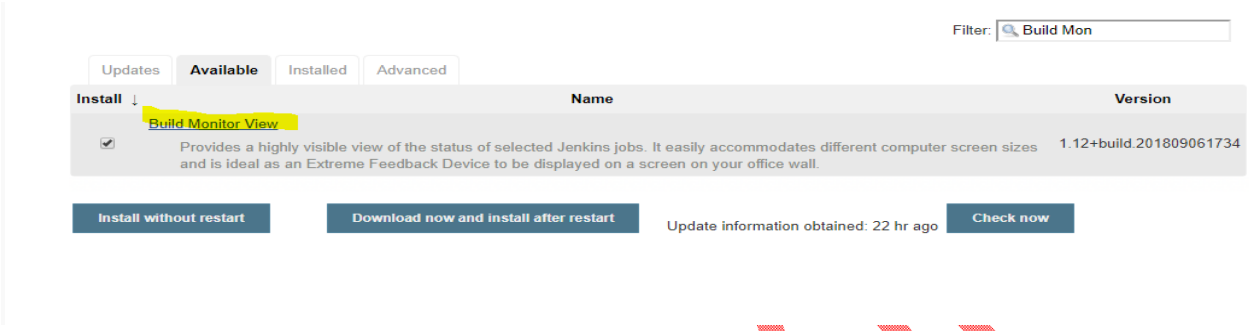


Build Monitor View

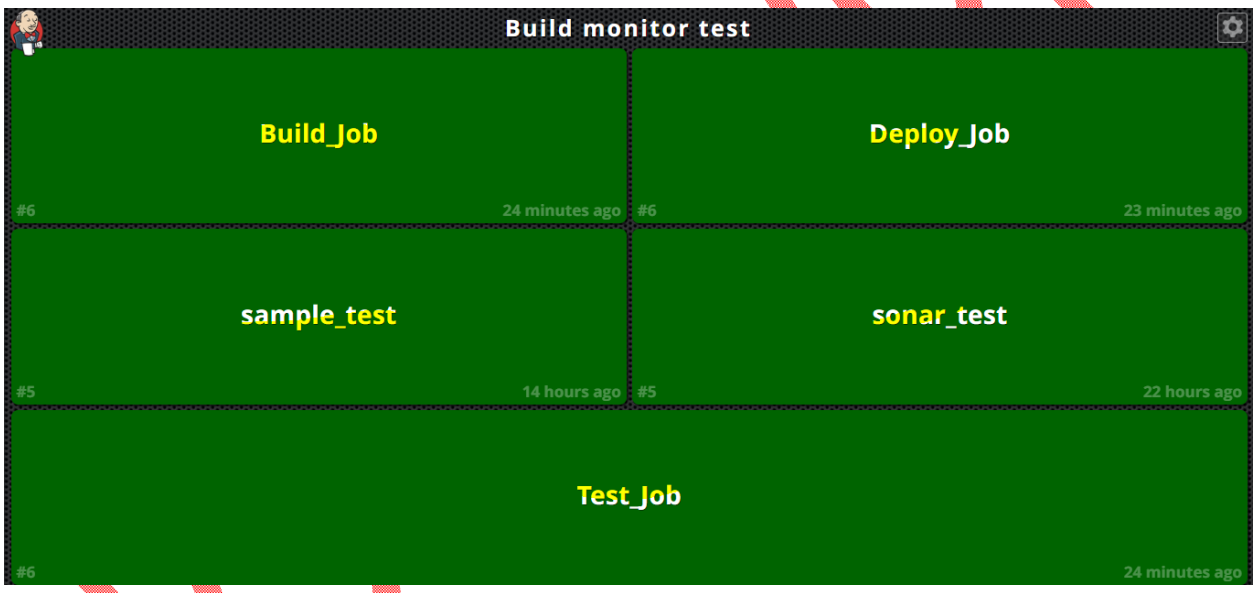
Provides a highly visible view of the status of selected Jenkins jobs. It easily accommodates different computer screen sizes and is ideal as an Extreme Feedback Device to be displayed on a screen on your office wall

1. Install Build monitor plugin

   Jenkins → manage jenkins → manage plugins →Build monitor Plugin



2. Configure the Build Monitor Plugin



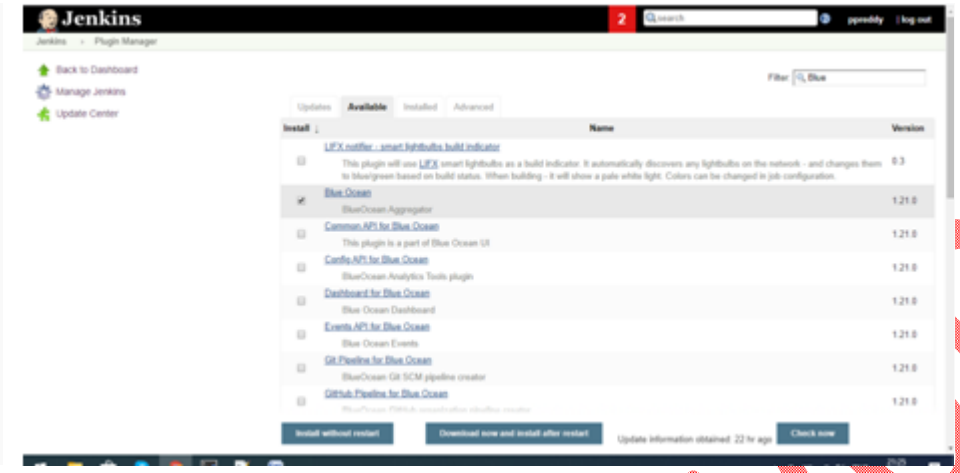
Jenkins Blue Ocean

Blue Ocean is a new User Interface (UI) and User Experience (UX) for Jenkins It is designed to make Jenkins UI more efficient (reduces clutter and increases the clarity).

In simple words: Blue Ocean is a new User Interface for Jenkins and provides an interactive view for Jenkins Pipeline (and jobs).

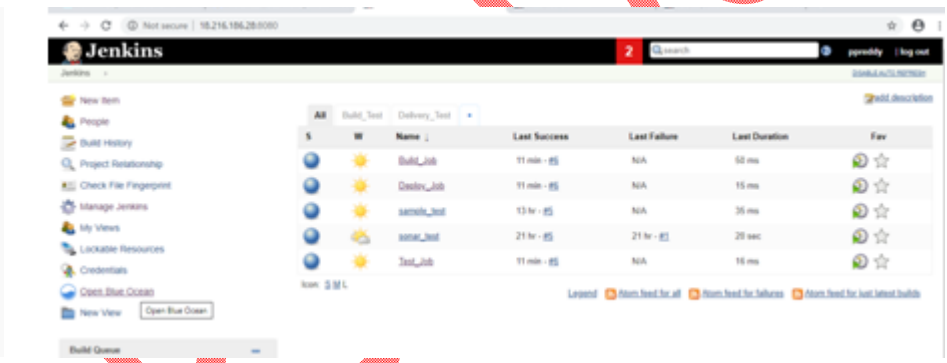
Install Blue Ocean Plugin

Jenkins → manage jenkins → manage plugins→ Blue Ocean

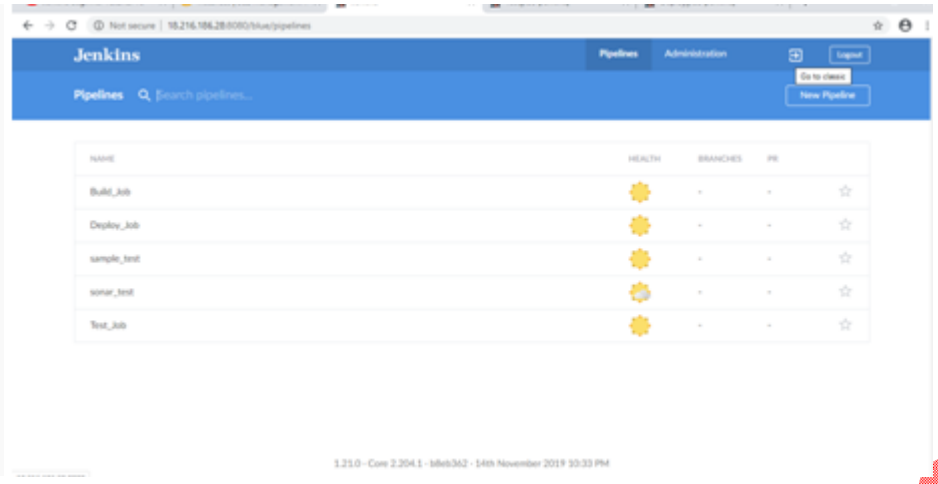


Switch to Blue Ocean View

Open blue ocean



Switch to classic view



Jenkins directory structure/Jenkins_Home

cd /var/lib/jenkins

workspace: This is the folder where jenkins keeps the project specific files

jobs: This contains configuration details of the job

Plugins: This folder contains all installed plugins

Nodes: the information about jenkins slaves is kept under this folder

Users: The information about the jenkins users is kept under this folder

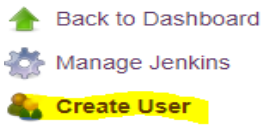
userContent: The files kept under this folder is served via http

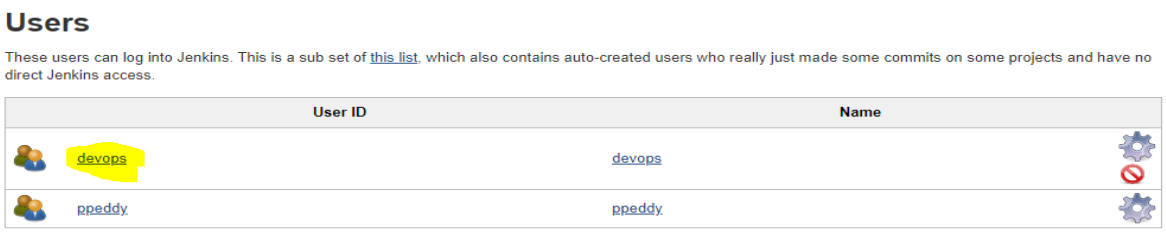
Managing users in Jenkins

This allows us to add/modify/delete users in jenkins

Adding new user: Jenkins → Manage Jenkins → Manage Users → Create User

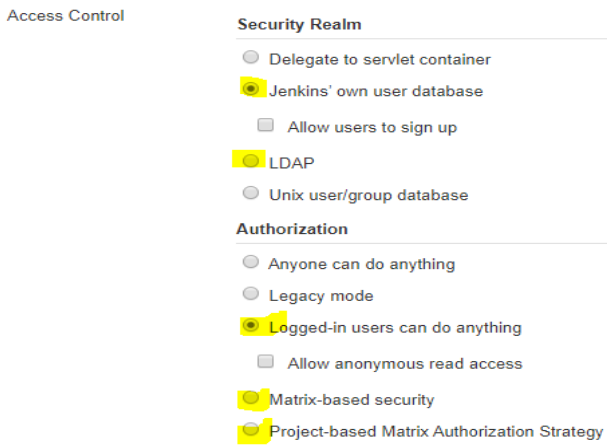
Fill the user details and click create user.





- Whenever create the user is having by default admin permissions.

Jenkins → Manage Jenkins → **Configure Global Security**



- By default Jenkins is uses is own database to store users later we can change database.
- In real-time company will integrate the **LDAP** server.(Light Weight Direct Access Protocol).

Security Realm

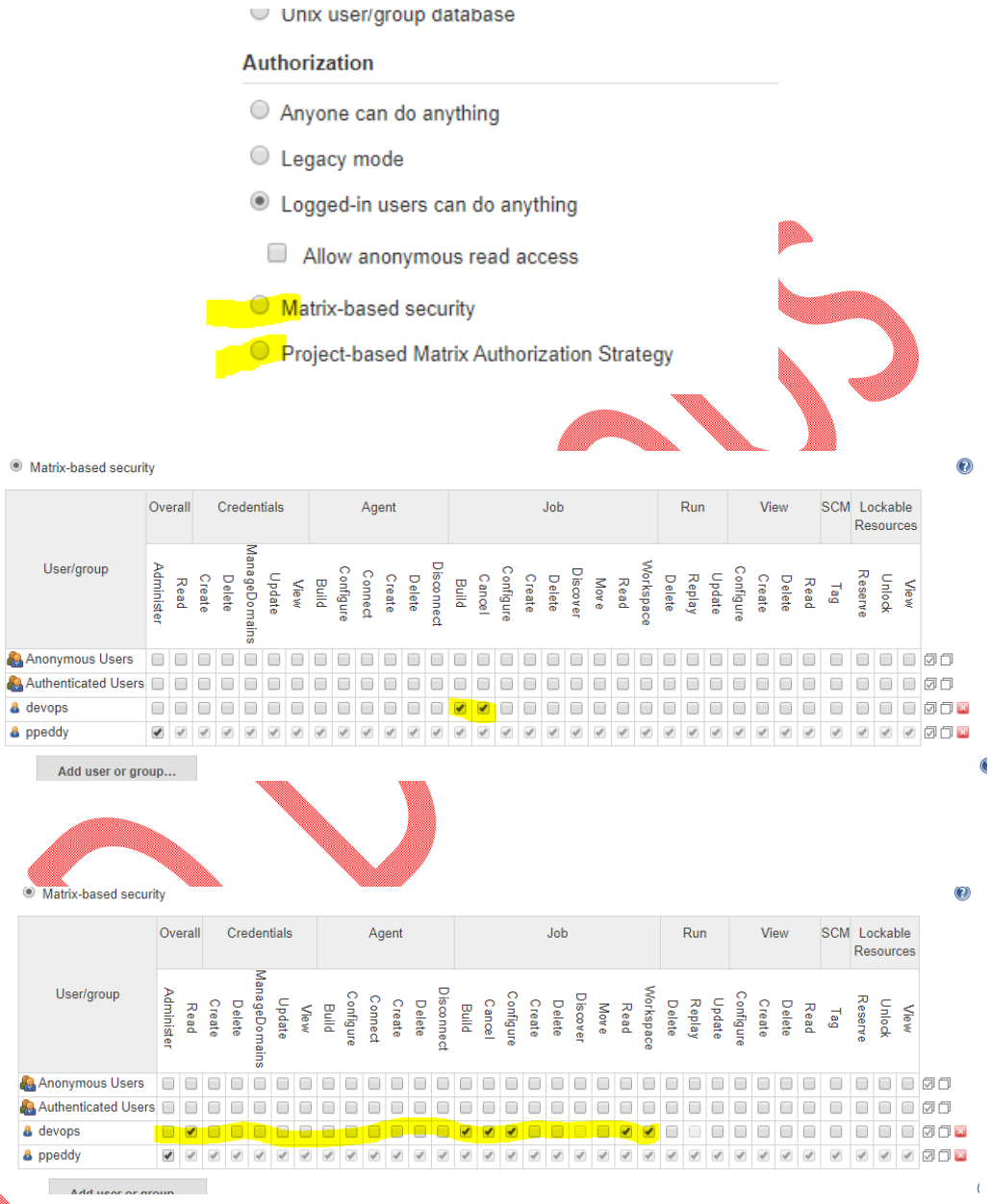
This all about where jenkins maintains user credentials and their roles

1. Jenkins own database (default option)
2. LDAP (Lightweight Directory Access Protocol)
   Maintaining user credentials and their roles in LDAP give more performance than maintaining in DB.
   By integrating jenkins with LDAP we can grant access to the users already present in LDAP.
1. Unix user/group database

Matrix-based security

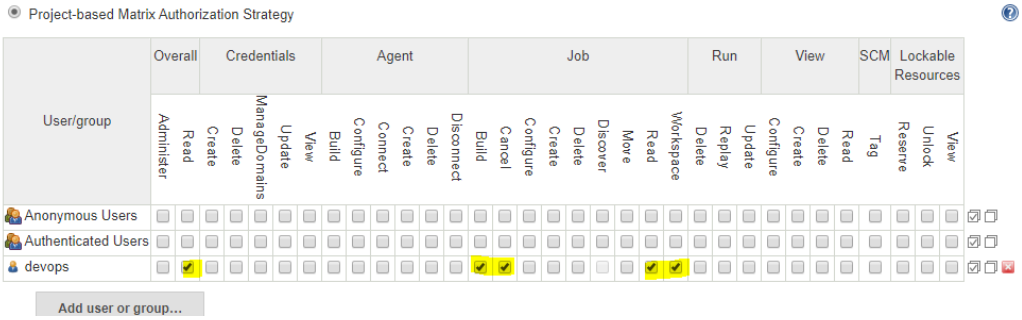
Jenkins has matrix based security option to configure granular permissions to the users in the Jenkins

Jenkins → Manage Jenkins → Configure Global Security → under Authorization select **matrix-based security**

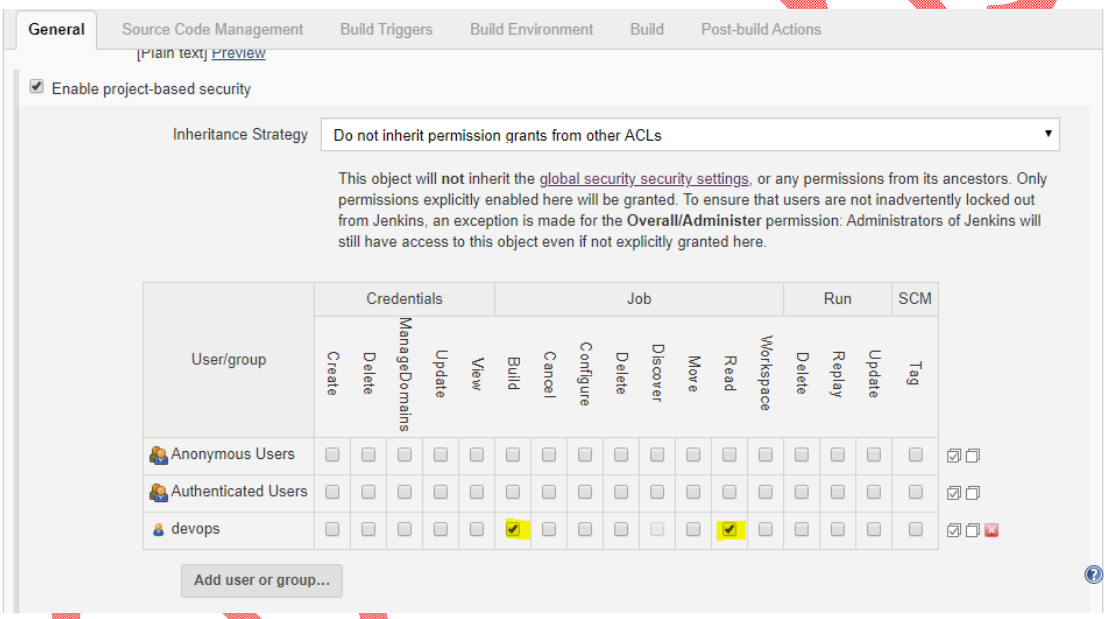


Project-based Matrix Authorization Strategy

We can provide the job level permission to the users.

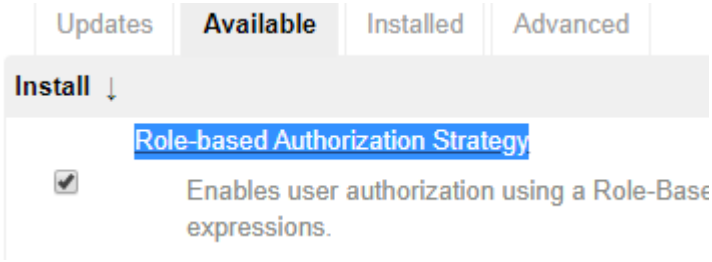


Open the job → General → Select Enable project-based security Inheritance Strategy:

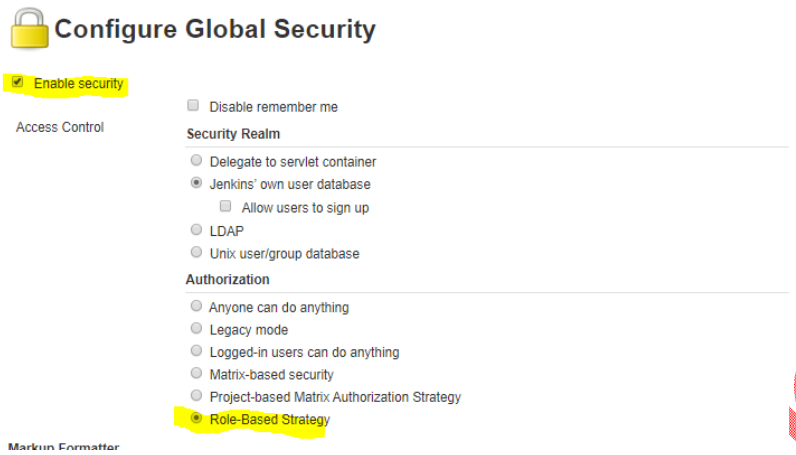


Jenkins authentication and authorization

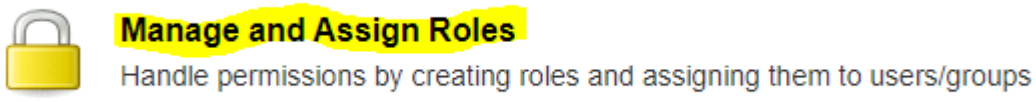
1. Create and configure new users
2. Create and manage new roles
   Install Role-based Authorization Strategy plugin



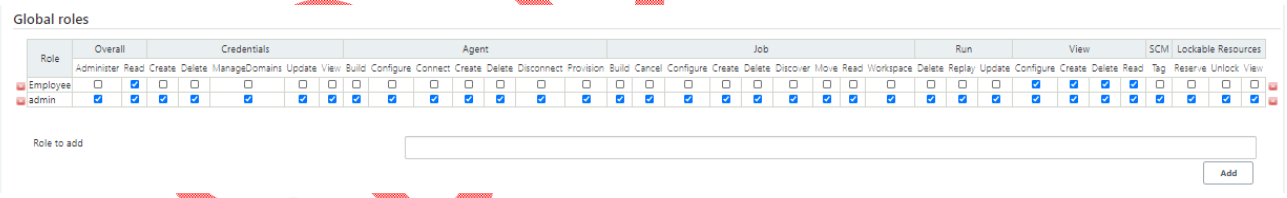
3. Manage Jenkins - Configure Global Security - Authorization - Role Based Strategy



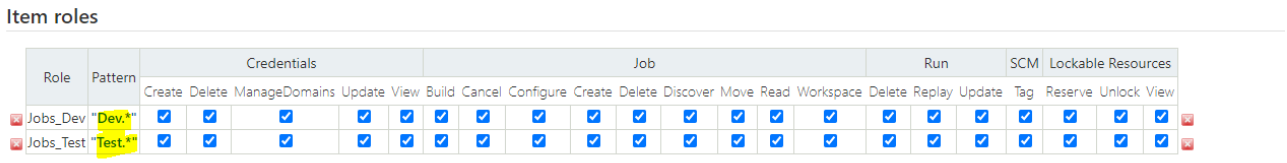
4. Manage Jenkins → Manage and Assign Roles



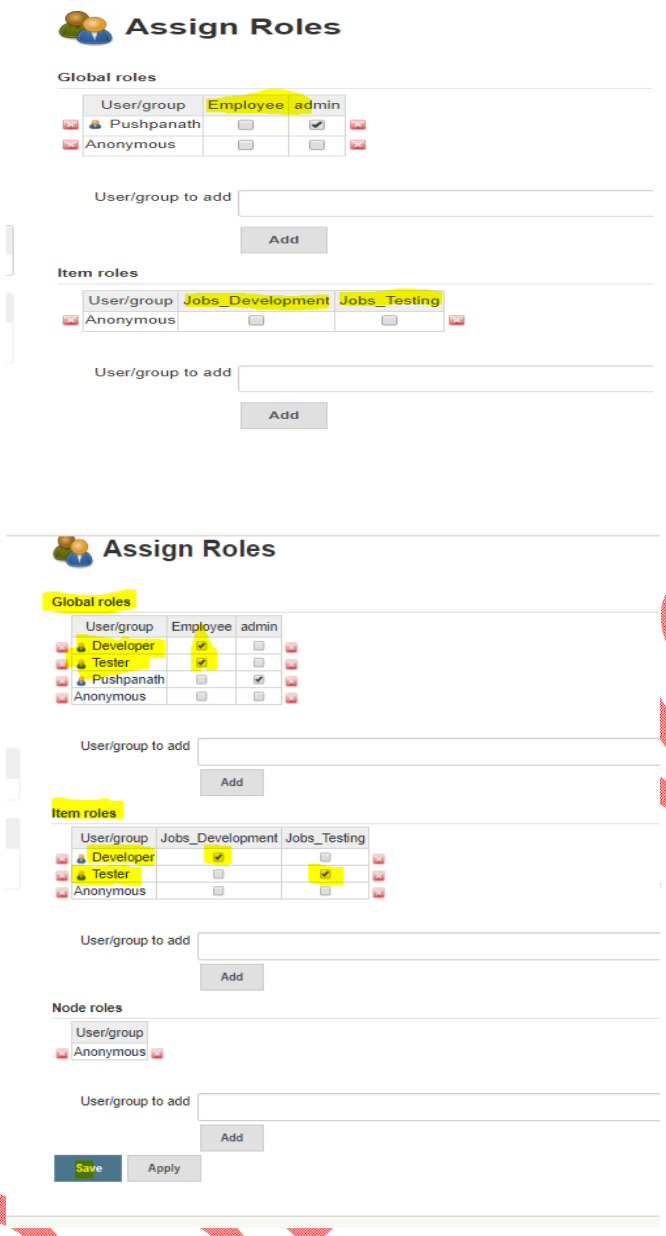
a. Create and add the Global role **Employee**



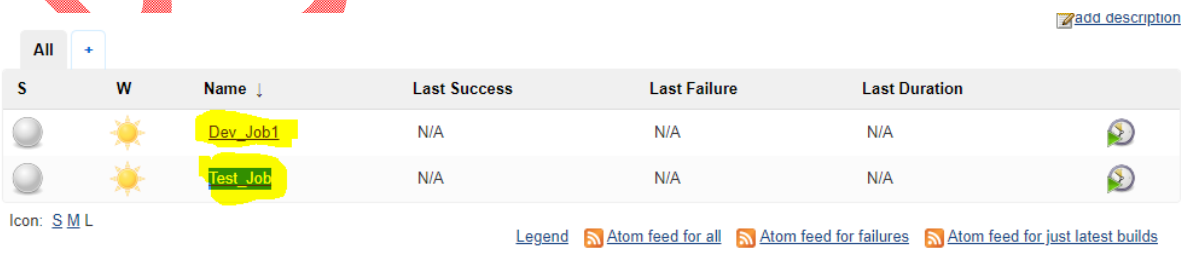
b. Create and add the Item roles Jobs_Development and Jobs_Testing and pattern as Dev.* and Test.*



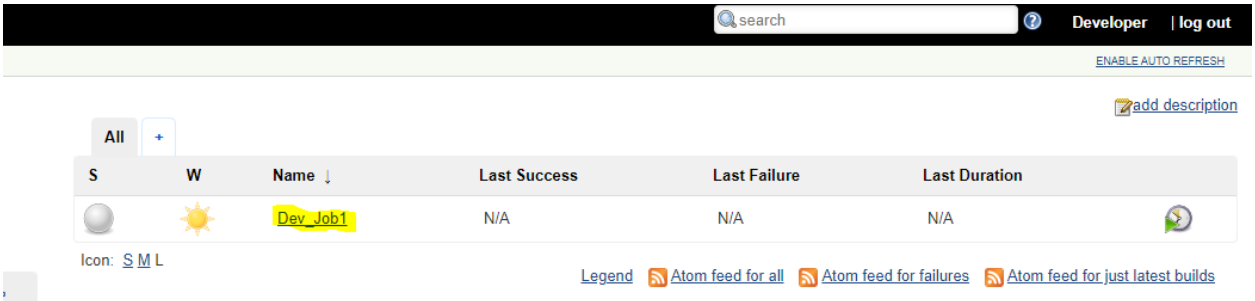
c. Assign the roles to our users.
Manage Jenkins → Manage and Assign Roles→ Assign Roles

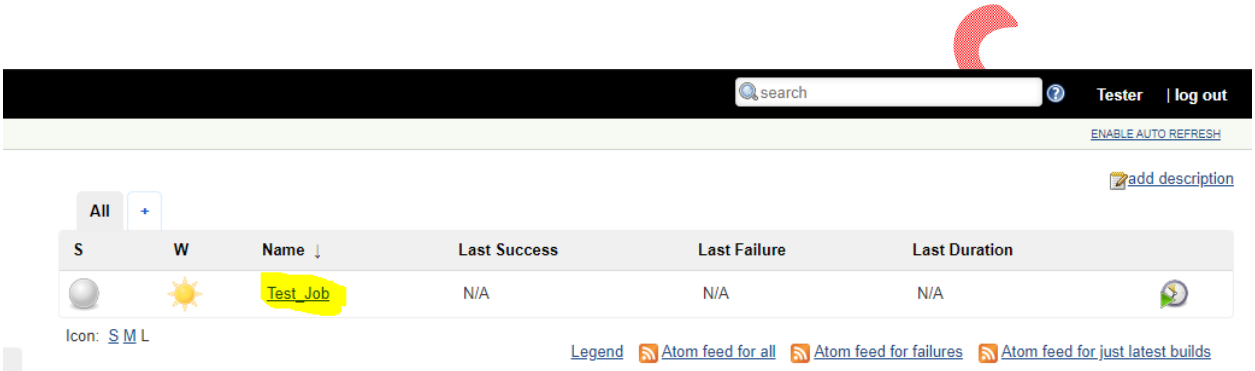


5. Create the jobs as admin user like **Dev_Job1** and **Test_Job**



6. Login to Dev and Test users





Master Slave Configuration

Jenkins master slave configuration to create cluster of machines (Nodes),Where we can distribute the load. It's similar to load balancing.

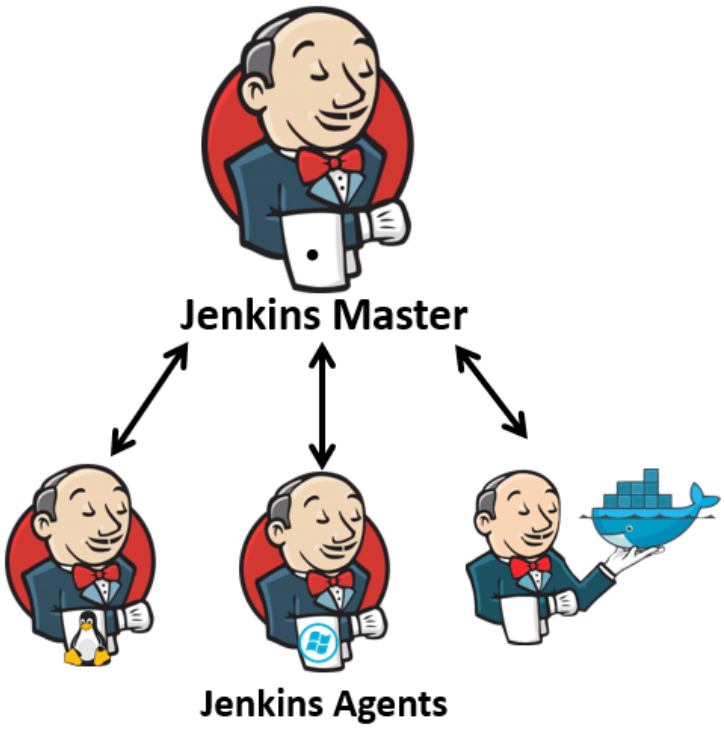
If you have larger and heavier projects which get built on a regular basis and running all of these builds on a central machine may not be the best option. In such case, you can configure other Jenkins machines to be slave machines to take the load off the master Jenkins server.Sometimes you might also need several different environments to test your builds, in this scenario using a slave to represent each of your required environments is good idea.The master-slave architecture of Jenkins is used for distributed build environments, where the workload of building projects is distributed to multiple agent nodes or slaves. We can also different environments for each build. Since each slave runs a separate program called a **slave agent**, there is no require to install the full Jenkins (package or compiled binaries) on a slave. There are a variety of ways to start slave agents, but at the end of the slave agent a Jenkins master needs to establish a bi-directional communication link (for example a TCP/IP socket) in order to operate.

We use this for the following use cases.

- Improve Jenkins performance
- We can also use separate slave for different workload type, for example

Slave-1 → Java
Slave-2 → .Net
Slave-3 → Terraform
Slave-4 → Chef

Master is the machine on which we installed and configure jenkins
Slave is a machine added under the master, Slaves are controlled by master based on the job configuration.



Configuring Master/Slave configuration

1. Configure Jenkins Master Credentials

When you got the master server Jenkins installed, we need to configure the master server itself. By default, there are different ways to start Jenkins agent nodes, we can launch the agent nodes through SSH, windows administrative account, and via Java Web Start (JNLP), pick the best way depending on your environment setup and operating system. We will launch the agent nodes through ssh, and we need to setup Jenkins credentials on our master server.

sudo -i -u jenkins
ssh-keygen

```
ubuntu@ip-172-31-22-23:~$ sudo -i -u jenkins
jenkins@ip-172-31-22-23:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/jenkins/.ssh/id_rsa):
/var/lib/jenkins/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/jenkins/.ssh/id_rsa
Your public key has been saved in /var/lib/jenkins/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:q2nvqHz0pJns0ogJzzehSxFQmhLyPuskYQ5/1FV5GgE jenkins@ip-172-31-22-23
The key's randomart image is:
+---[RSA 3072]----+
|+..     E.oo     |
|.*       .o .    |
|+ o     .  +     |
|.. . . .  .      |
|o.+ . . S        |
|=o =. . ..       |
|.==+.* *.        |
| =*o= O+.        |
|  oooB=oo        |
+----[SHA256]-----+
```

cd .ssh

cat id_rsa.pub

```
jenkins@ip-172-31-22-23:~$ cd .ssh
jenkins@ip-172-31-22-23:~/.ssh$ ls -lrt
total 12
-rw-r--r-- 1 jenkins jenkins  444 Nov 24 14:04 known_hosts
-rw-r--r-- 1 jenkins jenkins  577 Dec  3 14:32 id_rsa.pub
-rw------- 1 jenkins jenkins 2610 Dec  3 14:32 id_rsa
jenkins@ip-172-31-22-23:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC56FwNoDSoIqbTpuloKfwuQcHEcg5DIMbJ7rClvwdJOmukKCQkbaDK5Uf12fpgavdIbBio+dcpQMMKU3/NtQ
6ULv6652ztKzBf2xZynJZHi48R+42LhUW10u9WM9MQ8o8+SzaIxznl61y8ZZqwwpr00mXYS72b4JIoVCu+PMhjMDaOepQDLtGdfAz6Mtl7r9B3bBCJz1wPQwo2
PmwzID88h8oMOfyGZ8UfYjq+K3Td6ADX6eTaOINH9CV/Tr7y9JUtSGk3K2JPuZXomNRMnUBPCEQ3iZFqVqPGpCRgasVECskS11t75O9T5lQfZ/dOGg1jlHRvhJ
210opwrYmpdei/AThK7kH7XWoUF4L4B9xpn429y2qT5vRdpLRZb1kXdabX7/07V7fNMu+FVDNh1nbbF/LwW8NeIZBk/kW0kZnhseQTS+DPwFoc1GZ3bcQBIX4i
V1KZoBZTOiDejnLE2Edgvpm+6NYvKEDoQaB83qtX0Bi3txNAI8f+n6A8HmhLhPk= jenkins@ip-172-31-22-23
jenkins@ip-172-31-22-23:~/.ssh$
```

cat id_rsa

```
jenkins@ip-172-31-22-23:~/.ssh$ cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAuehcDaA0qCKm06bpaCn8LkHBxHIOQyDGye6wpb8HSTprpCgkJG2g
yuVH9dn6YGr3SGwYqPnXKUDDClN/zbUOlC7+uuds7SswX9sWcpyWR4uPEfuNi4VFtdLvVj
PTEPKPPks2iMc5SetcvGWasMKa9NJl2Eu9m+CSKFQrvjzIYzA2jnqUAy7RnXwM+jLZe6/Q
d2wQic9cD0MKNj5sMyA/PIfKDDn8hmfFH2I6vit03egA1+nk2jiDR/Qlf06+8vSVLUhpNy
tiT7mV6JjUTJlATwhEN4mRalajxqQkYGrFRArJEtdbe+TvU+ZUH2f3ThoNY5R0b4SdtdKK
cK2JqXXovwE4SuSB+llqFBeC+AfcaZ+Nvctqk+b0XaS0WW9ZF3Wnl+/9Ole3zTLvhVQzYd
Z22xfy8FvDXiGQZP5FtJGZ4bHkE0vgz8BaHNRmd23EASF+IldSmaAWUzog3o5yxNhHYL6Z
vujWLyhA6EGgfN6rV9AYt7cTQCPH/p+gPB5oS4T5AAAFkL8QF4e/EBeHAAAAB3NzaC1yc2
EAAAGBALnoXA2gNKgiptOm6Wgp/C5BwcRyDkMgxsnuskW/B0k6a6QoJCRtoMrlR/XZ+mBq
9OhsGKj51ylAwwpTf821DpQu/rrnbO0rMF/bFnKclkeLjxH7jYuFRbXS71Yz0xDyjz5LNo
jHOeXrXLxlmrDCmvTSZdhLvZvgkihUK748yGMwNo56lAMu0Z18DPoy2Xuv0HdsEInPXA9D
CjY+bDMgPzyHygw5/IZnxR9iOr4rdN3oANfp5No4g0f0JX9OvvL0lS1IaTcrYk+5leiY1E
ydQE8IRDeJkWpWo8akJGBqxUQKyRLXW3vk71PmVB9n904aDWOUdG+EnbXSinCtial6L8B
OEruQftdahQXgvgH3Gmfjb3LapPm9F2ktFlvWRd1ptfv/TtXt80y74VUM2HWdtsX8vBbw1
4hkGT+RbSRmeGx5BNL4M/AWhzUZndtxAEhfiJXUpmgFlM6IN6OcsTYR2C+mb7o1i8oQOhB
oHzeq1fQGLe3E0Ajx/6foDweaEuE+QAAAAMBAAEAAAGAYWZ71Ce7btu4y+sHHqvgk9ufyW
x4THM/s0x7AFulVYhmHPtq9Cr7s1GtAgHSS/G+3RLFWMKNF+/2P9K/gub5XGnCy+m0CA44
qf//Is0nbxSPQLyqzA5nF3wMKP2994MdyWVejQrUKsHFNG+HPFS5Gsug0FHeZWyjJr3yrR
jrZAw+iSdvpLtCl1/G7M73YvYaIndbstgRQ228S+8ogSDEANKj6QP4afK0NF0iqb/a2uhS
iMziOfKh1XD0tF5tbJz3JjO9+BmN+W+Bg8NqpsEqGZvG9ATOWRxB9CTTgFFug+TGnDRTCo
2bGgoNOD/hOjO9aJj9YVIyemnqKn7mOpFAjLOw430p7W2AQhkSMK5qKlGOHyJC4vUXei6p
whX7bBlfhO1NQIv099TWggQ1LaddB2RI6iFqFNWj2ArZHWTk88SceCN5/kkcTsZAj5G+60
7CXlMdSn2nPX+ihT5s5YX6ACUrP3CuXceD4DKvPOf75lKa6wC8Iydc97vc2Jl31OyBAAAA
wQCJfEfmKvH9vXKnyLvm4CYZ4oWPCztQUxxC8OV8+zCR2s9UATAP/tI88C27Ugap9L6bnB
6p9gn5cLRgpQLAnpt8erUrYlYbjIIlbM6lEWZCeYYtT6YRw0Bw9Cuf8gV2ijHxGr6DXfiz
xUPd7/w7JcqzNsROi+FOHzhB2rLC9W5X+PdnhX+MiKGVLRlshBEP9eaK10bMphbftmSsfk
z8nObaCRC32E34FFlaFzw/+yhM+hFng7VAPSFRgsXyLtvSiegAAADBAOpaVlEcnn99PrkB
2GdWz/mJPoVRll/qf2/L87YHio3Bn5Yhq3X85eZc8rGHlyWZ2tj1R/mLl5LhYukWwN5ipf
aoqEY+qcQ/eiOkyM7Xxpw5qa0olorkFzyyLw6qAHBc4LG+b3B7CpkWxCZJJ+Vh1lvE/gPk
1N4CCR2Il3JC0tOlREsmzt6rS5L2ikGcC6NmKQxLtJBs9dCUzU0wK9wQRXy7cHzMZiI1gn
mRiKvJ3yAzRJm0nd0JpXOTdT1eWyBm0wAAAMEAyxR0ZBfJKMTZEff63Y1pNyb7FKp7pxe4
YJNry8CkuQFFJ8/RbuDWUGQbgdGe4976Q1tfFTr2xevfFjG9JsE5fjvzVbMSxKkq/dvKjO
lDqh6cXZNLOnUvzaLN72U3PEuBcykRYyD9Vt6/H2F6S1VJ+XgN3hKTby+Rb6u2EiDzuYT8
2oQhMqtnlzVOVDiuIQeTvGqbMNeeCpZLqID5lC6aD63m5c9FtqgOD8lS3oHiTH/zY7VXxh
aOJqhJYZiQCx2DAAAAF2plbmtpbnNAaXAtMTcyLTIxLTIzAQID
-----END OPENSSH PRIVATE KEY-----
jenkins@ip-172-31-22-23:~/.ssh$
```

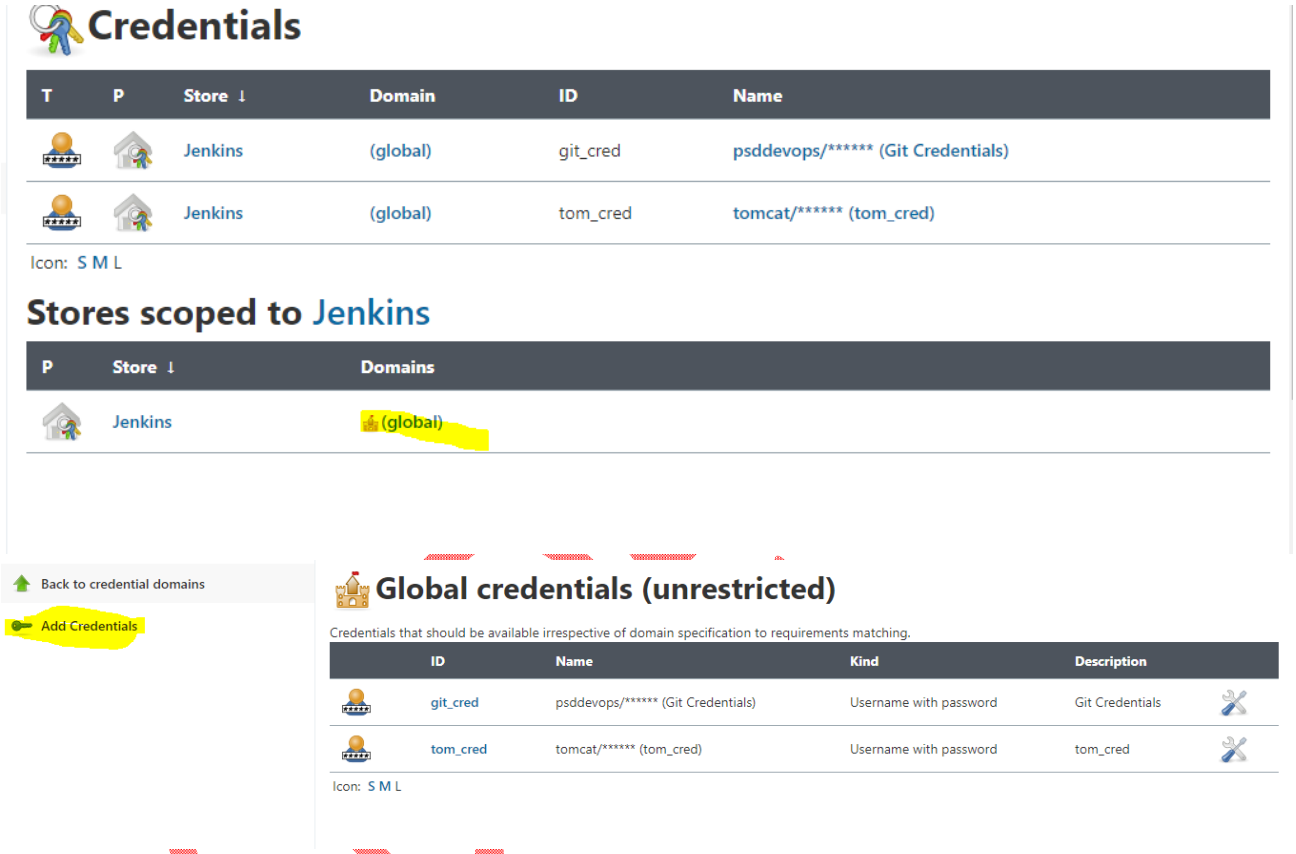
2. Setup Credentials on Jenkins

Open your Jenkins dashboard and add the configure the credentials.

Jenkins → Manage Jenkins → Managed Credentials

And click the 'global' domain link.

Now click 'Add Credentials'.



Now choose the authentication method.

Kind: SSH Username with private key

Scope: Global

Username: jenkins

Private key: Enter directly and paste the 'id_rsa' private key of Jenkins user from the master server.

Click 'OK'.

3. Set up slave nodes

Now we will setup slave nodes server by installing java on those server, and create a new Jenkins user.

Install the 'software-properties-common' packages and add the java PPA repository and install java OpenJDK using apt command below.

sudo apt install software-properties-common apt-transport-https –y
sudo add-apt-repository ppa:openjdk-r/ppa –y
sudo apt install openjdk-8-jdk –y

```
ubuntu@ip-172-31-32-190:~$ java -version
openjdk version "1.8.0_275"
OpenJDK Runtime Environment (build 1.8.0_275-8u275-b01-0ubuntu1~20.04-b01)
OpenJDK 64-Bit Server VM (build 25.275-b01, mixed mode)
ubuntu@ip-172-31-32-190:~$
```

Create the jenkins user in slave node
useradd -m -s /bin/bash jenkins
passwd jenkins

```
root@ip-172-31-32-190:/home/ubuntu# useradd -m -s /bin/bash jenkins
root@ip-172-31-32-190:/home/ubuntu# passwd jenkins
New password:
Retype new password:
passwd: password updated successfully
root@ip-172-31-32-190:/home/ubuntu#
```

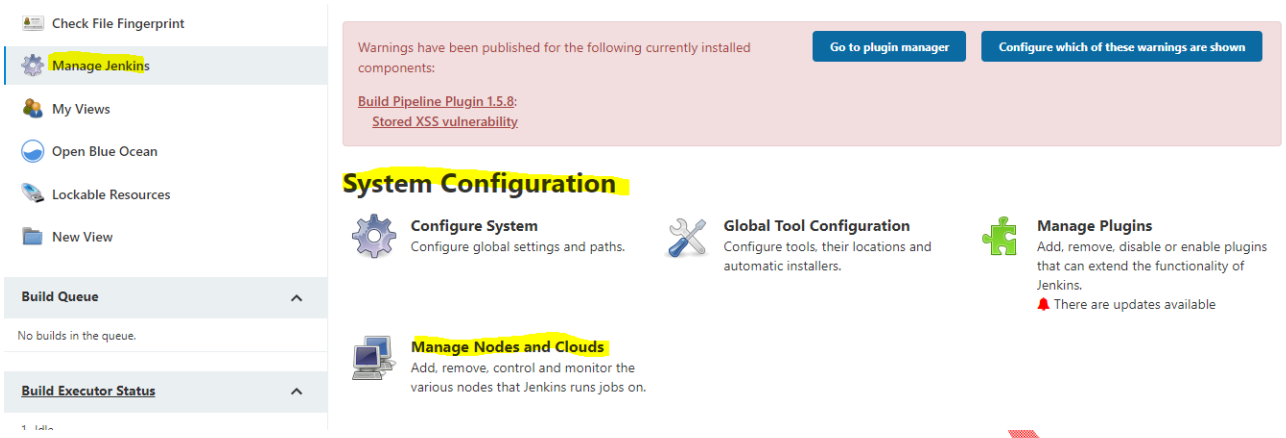
Copy the SSH Key from Master to Slave
ssh-copy-id jenkins@ip-172-31-32-190

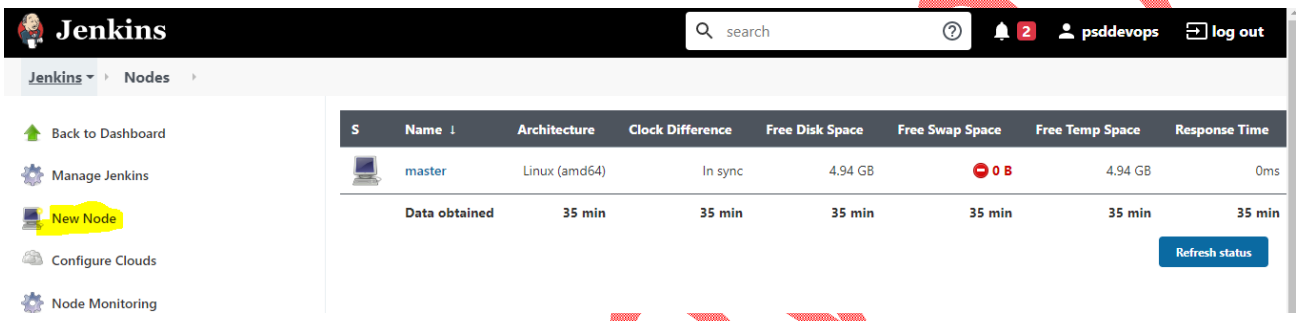
```
jenkins@ip-172-31-22-23:~/.ssh$ ssh-copy-id jenkins@ip-172-31-32-190
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/var/lib/jenkins/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
jenkins@ip-172-31-32-190's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'jenkins@ip-172-31-32-190'"
and check to make sure that only the key(s) you wanted were added.

jenkins@ip-172-31-22-23:~/.ssh$
```

4. Add New Slave Nodes

On the Jenkins dashboard, click the 'Manage Jenkins' menu, and click 'Manage Nodes'.

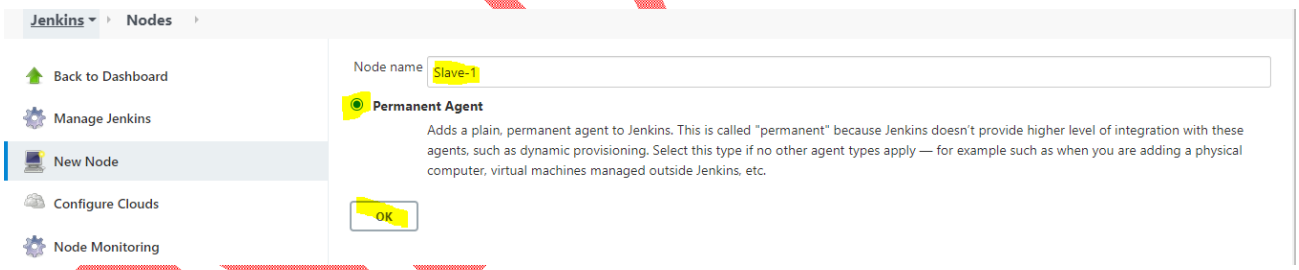
Manage Jenkins → System Configuration → Manage Nodes and Clouds





Click the 'New Node'

Type the node name 'Slave-1', choose the 'permanent agent', and click 'OK'.



Enter the below details

Description: Slave-1 node agent server

Remote root directory: /home/jenkins

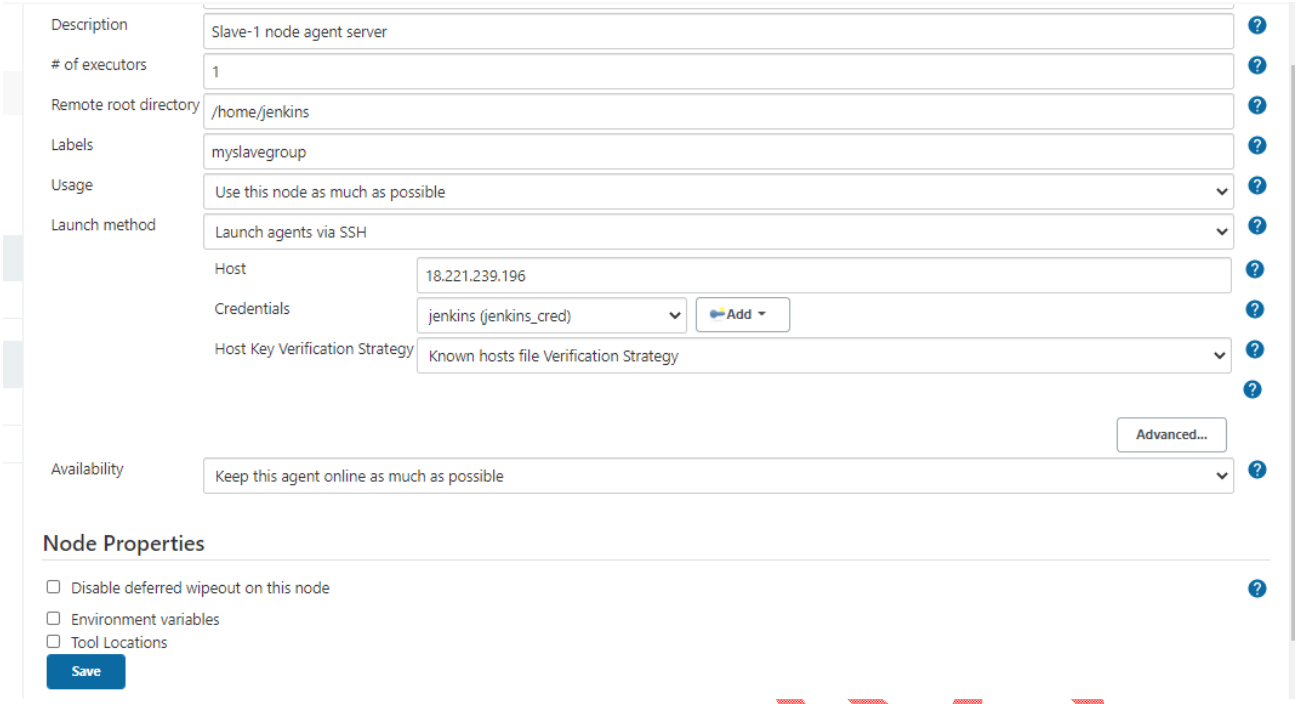
Labels: myslavegroup

Launch method: Launch slave agent via SSH,

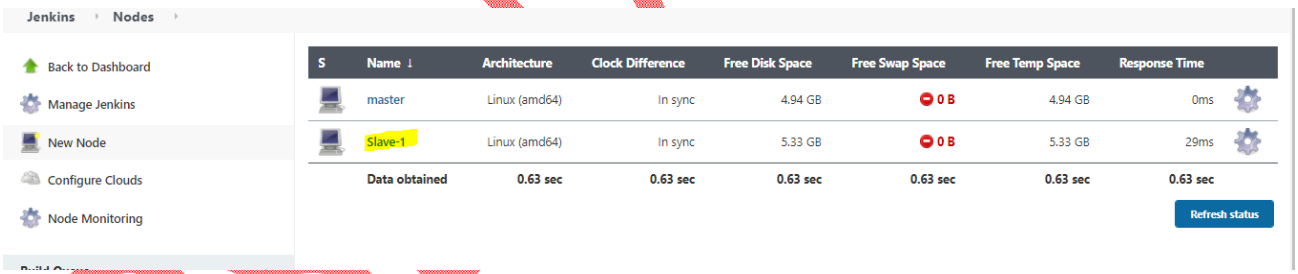
type the host ip address '18.221.239.196',

Choose the authentication using 'Jenkins' credential.

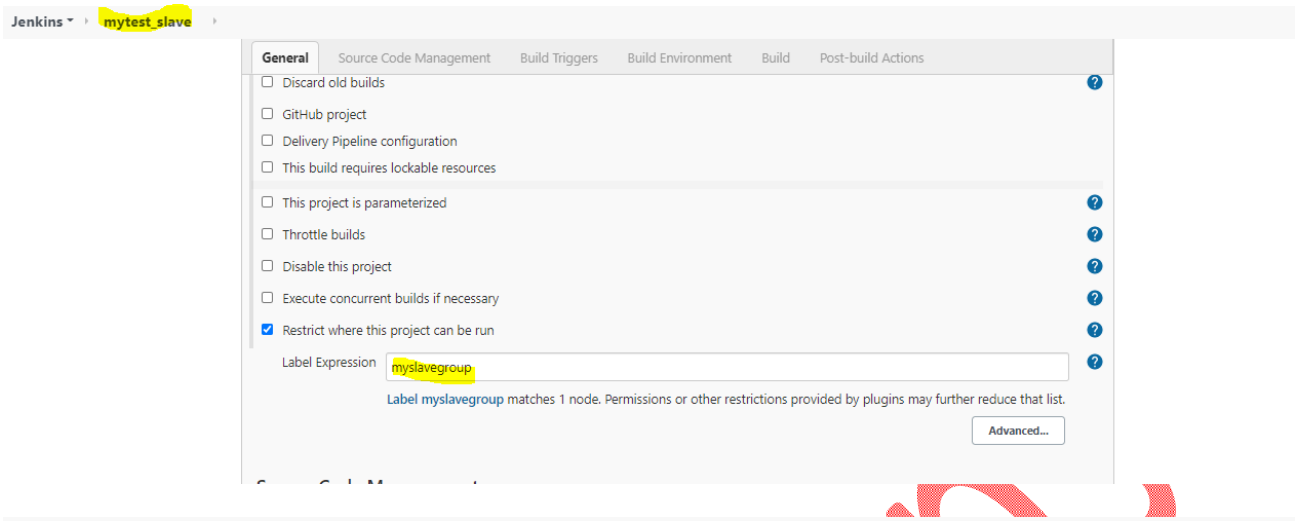
Availability: Keep this agent online as much as possible.



Now click **'Save'** button and wait for the master server to connect to all agent nodes and launch the agent services.
Below are the results when the master server is connected to all agent nodes.



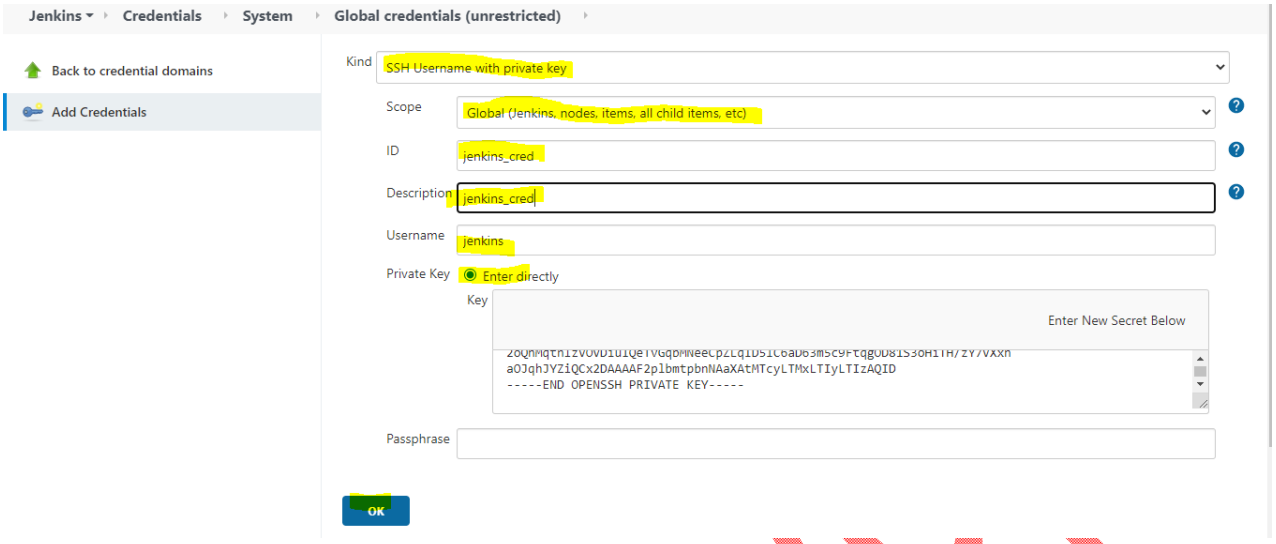
Creating sample job in slave



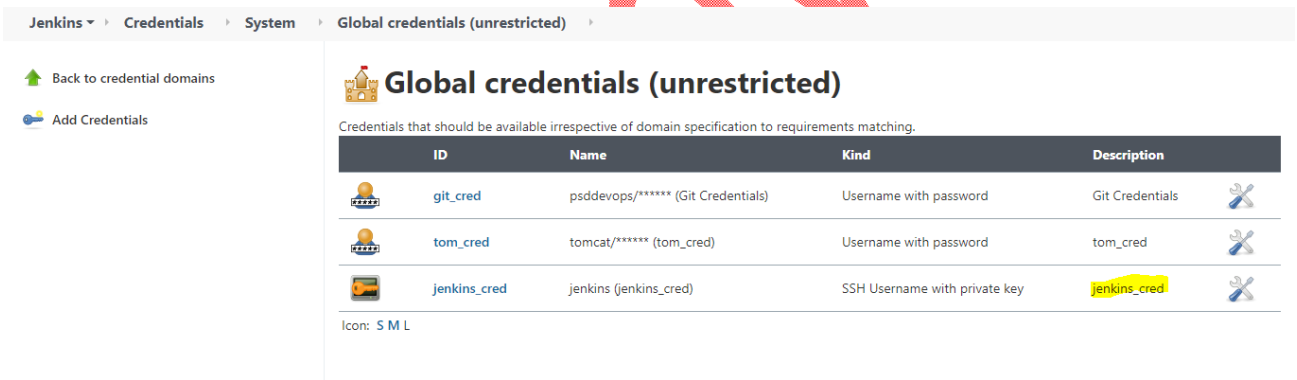
In slave node

```
jenkins@ip-172-31-32-190:~$ ll
total 1528
drwxr-xr-x 6 jenkins jenkins    4096 Dec  5 06:43 ./
drwxr-xr-x 4 root    root       4096 Dec  3 14:58 ../
-rw------- 1 jenkins jenkins     247 Dec  3 15:27 .bash_history
-rw-r--r-- 1 jenkins jenkins     220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 jenkins jenkins    3771 Feb 25  2020 .bashrc
drwx------ 3 jenkins jenkins    4096 Dec  5 06:30 .cache/
-rw-r--r-- 1 jenkins jenkins     807 Feb 25  2020 .profile
drwx------ 2 jenkins jenkins    4096 Dec  3 15:21 .ssh/
drwxrwxr-x 4 jenkins jenkins    4096 Dec  5 06:30 remoting/
-rw-rw-r-- 1 jenkins jenkins 1521553 Dec  5 06:30 remoting.jar
drwxrwxr-x 3 jenkins jenkins    4096 Dec  5 06:43 workspace/
jenkins@ip-172-31-32-190:~$ cd workspace/
jenkins@ip-172-31-32-190:~/workspace$ ll
total 12
drwxrwxr-x 3 jenkins jenkins 4096 Dec  5 06:43 ./
drwxr-xr-x 6 jenkins jenkins 4096 Dec  5 06:43 ../
drwxrwxr-x 2 jenkins jenkins 4096 Dec  5 06:43 mytest_slave/
jenkins@ip-172-31-32-190:~/workspace$
```

Way-2



And the Jenkins credential with ssh auth key method have been created.



Adding a slave to the master

- Jenkins → Manage Jenkins → Manage Nodes → New Node
- Node Name → Build-Slave
- Select Permanent Agent and click OK
- # of executors → 1 (one executor represents one thread, to run multiple jobs concurrently specify bigger value)
- Remote root directory → /home/ubuntu/jenkin_jobs (An agent needs to have a directory dedicated to Jenkins. Specify the path to this directory on the agent)
- Labels → "JavaJobs", labels are used to link a job to run on a specific slave
- Usage
    - Use as much as possible

- Only build jobs with label expressions matching this job (We are using this option for this demo)

Launch Method, Launch slave via ssh

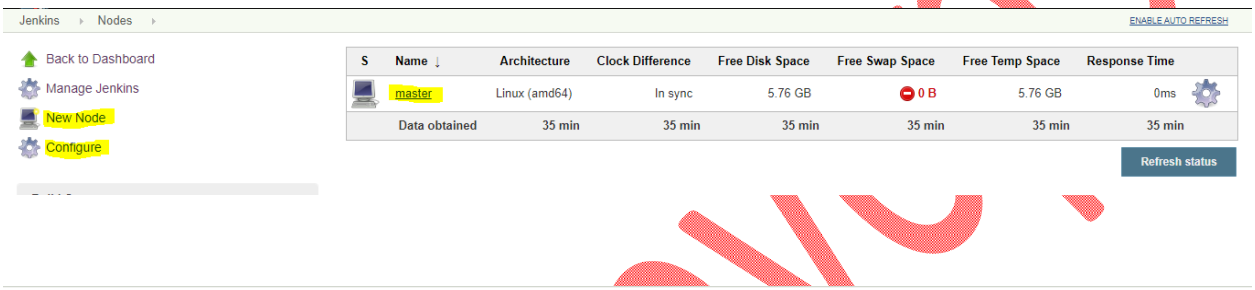
Host → IP address of the slave

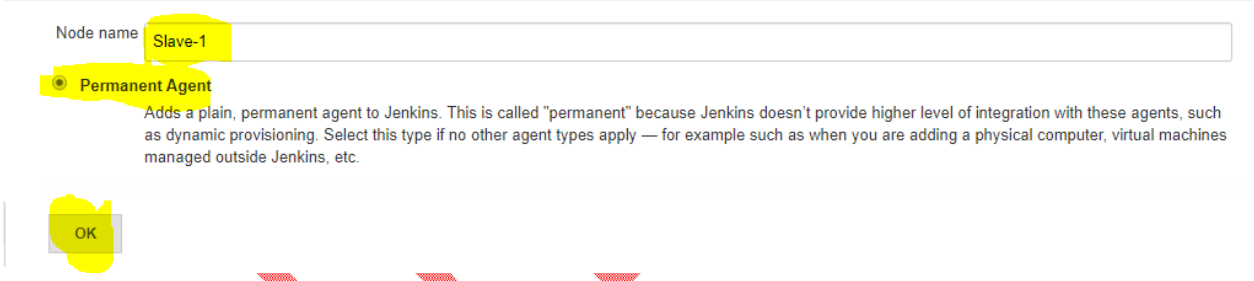
Create new credentials for ssh into slave, and select this credentials

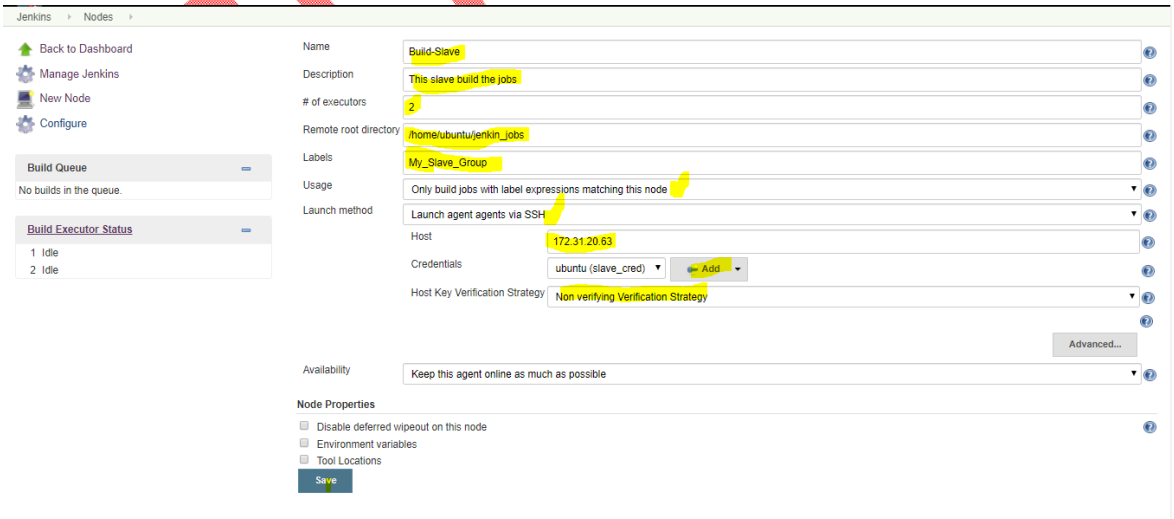
Host Key Verification Strategy → Non verifying verification strategy

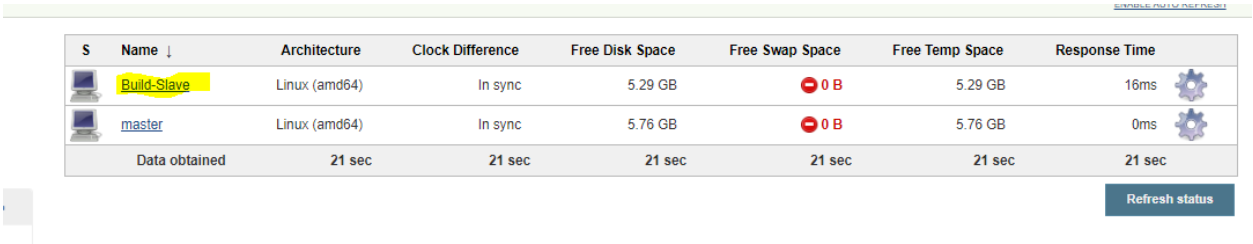
Availability → keep this agent online as much as possible and click save

Note: On slave java must exists, apart from this, for example if our job requires git, maven etc we also need to install those tools as well.

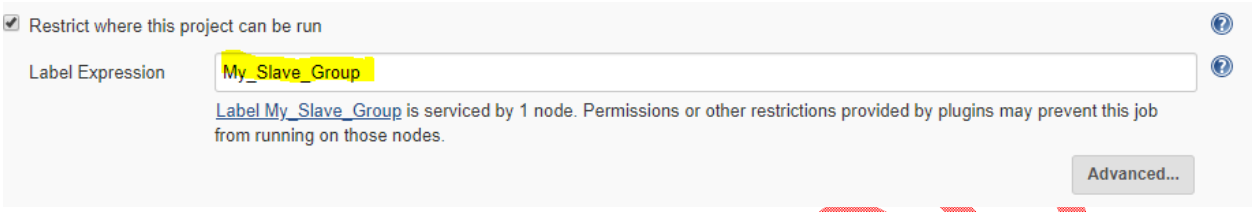








Go to Job configuration → Select ☑ Restrict where this project can be run→ Save && RUN



Jenkins Build Status

Disable Build:
A disabled Build will not be executed until you enable it again. This option often comes in handly to suspend a build during maintenance work or major refactoring.

Once the project is configured in Jenkins then all future builds are automated. It has basic reporting features like status and weather reports (job health).

Status of the build	Description
	Failed
	Unstable
	Success
	Pending
	Disabled
	Aborted

Figure a: Build status

Job health	Description
	No recent builds failed
	20-40% of recent builds failed
	40-60% of recent builds failed
	60-80% of recent builds failed
	All recent builds failed
	Unknown status

Figure b: Weather reports

Jenkins Build Status Icon Colours

 BLUE: Success

 GREY: Disabled, Aborted and Pending

 YELLOW: Unstable

 RED: Failed

Jenkins Job Weather Status Icon Colours

 60-80 of recent builds failed.

 No recent builds failed.

Note: **Need to do more document on these icons.**

To restart Jenkins manually, you can use either of the following URLs:

(jenkins_url)/safeRestart - Allows all running jobs to complete. New jobs will remain in the queue to run after the restart is complete.
Ex: http://localhost:8080/safeRestart
(jenkins_url)/restart - Forces a restart without waiting for builds to complete.
Ex: http://localhost:8080/restart

(OR)

You can install one plug called **SafeRestartPlugin** , once installed it will give one option Jenkins dashboard as follows.

Port Number Change

Open the /etc/default/jenkins file and change the below parameter with custom port number.
JENKINS_PORT

By default 8080 is the port, change from 8080 something like 8082 as follow.
#vi /etc/sysconfig/jenkins

JENKINS_PORT="8082"

Once you change the port restart the jenkins service by using below command.
#service jenkins restart

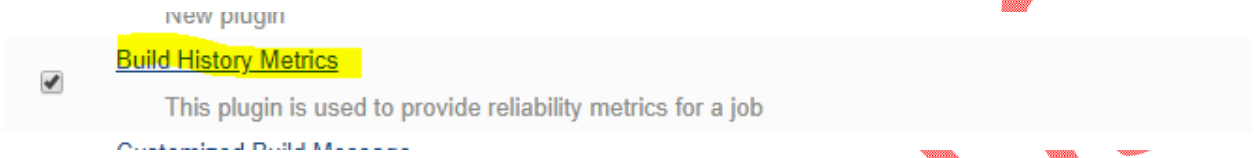
Jenkins - Metrics and Trends

Jenkins provides many plugins to present the metrics for builds which are carried out over a period of time. These metrics are useful to see the build and to understand how frequently they fail/pass over time.

Let's see the "Build History Metrics Plugin", this plugin is used to calculate the following metrics for all of the builds once installed:

- MTTF (Metrics Time to Failure)
- MTTR (Mean Time to Recovery)
- Standard Deviation of Build Times

Install **Build History Metric**s plugin and select **without restart** button.

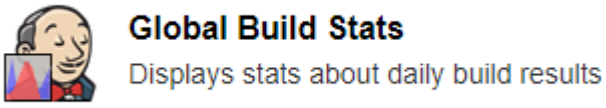


Select any job and go to that job page. When you go to that job page, you will see a table with the calculated metrics. Metrics are displayed for the last 7 days, last 30 days and all time.

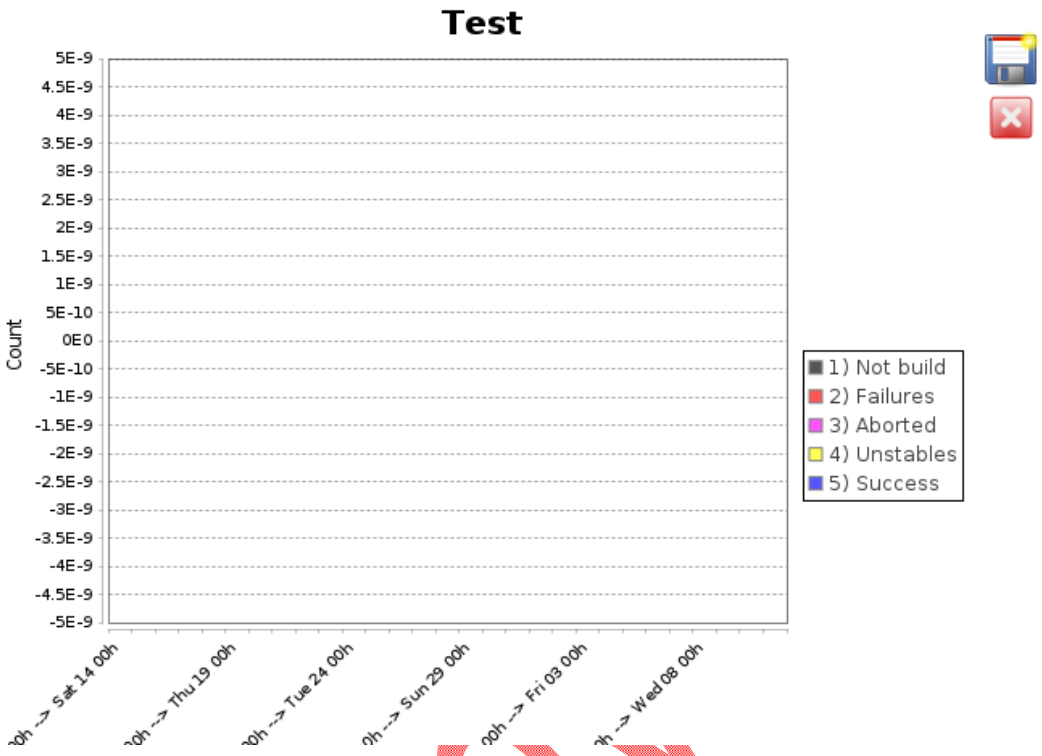
Trends in Jenkins

To see the overall Trends, Jenkins provides many plugins like **Build-metrics plugin**. This plugin is used to gather the information from within the builds and Jenkins and display them in a graphical format

Go to manage Jenkins →Scroll down, and now you will see an option called "Global Build Stats". Click on this option.



Click on the **Initialize stats** button. This button is used to gather all the existing records for builds which have already been carried out, and charts can be created based on these results. Once you click this button, you will see a text "**data successful initialized**".



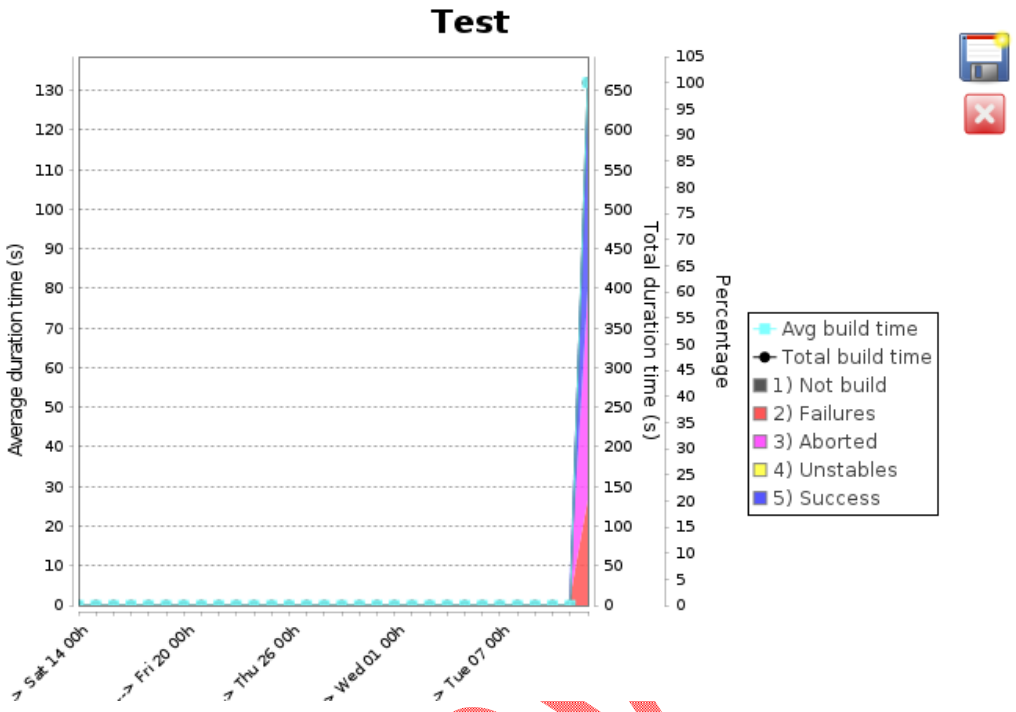
Keep existing job results only

Update retention strategies

Data Initialization

Click button below to initialize build statistics
Job results read will be merged with already reco
Data successfully initialized ! Refresh page

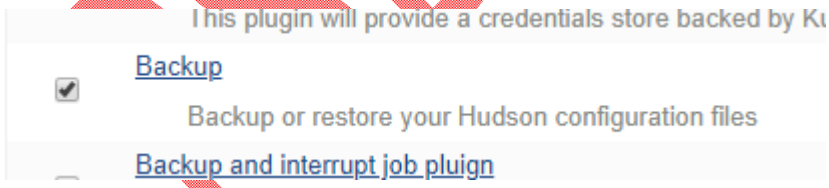
Initialize stats



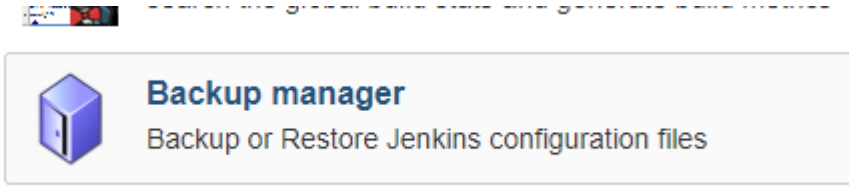
Jenkins Backup Plugins

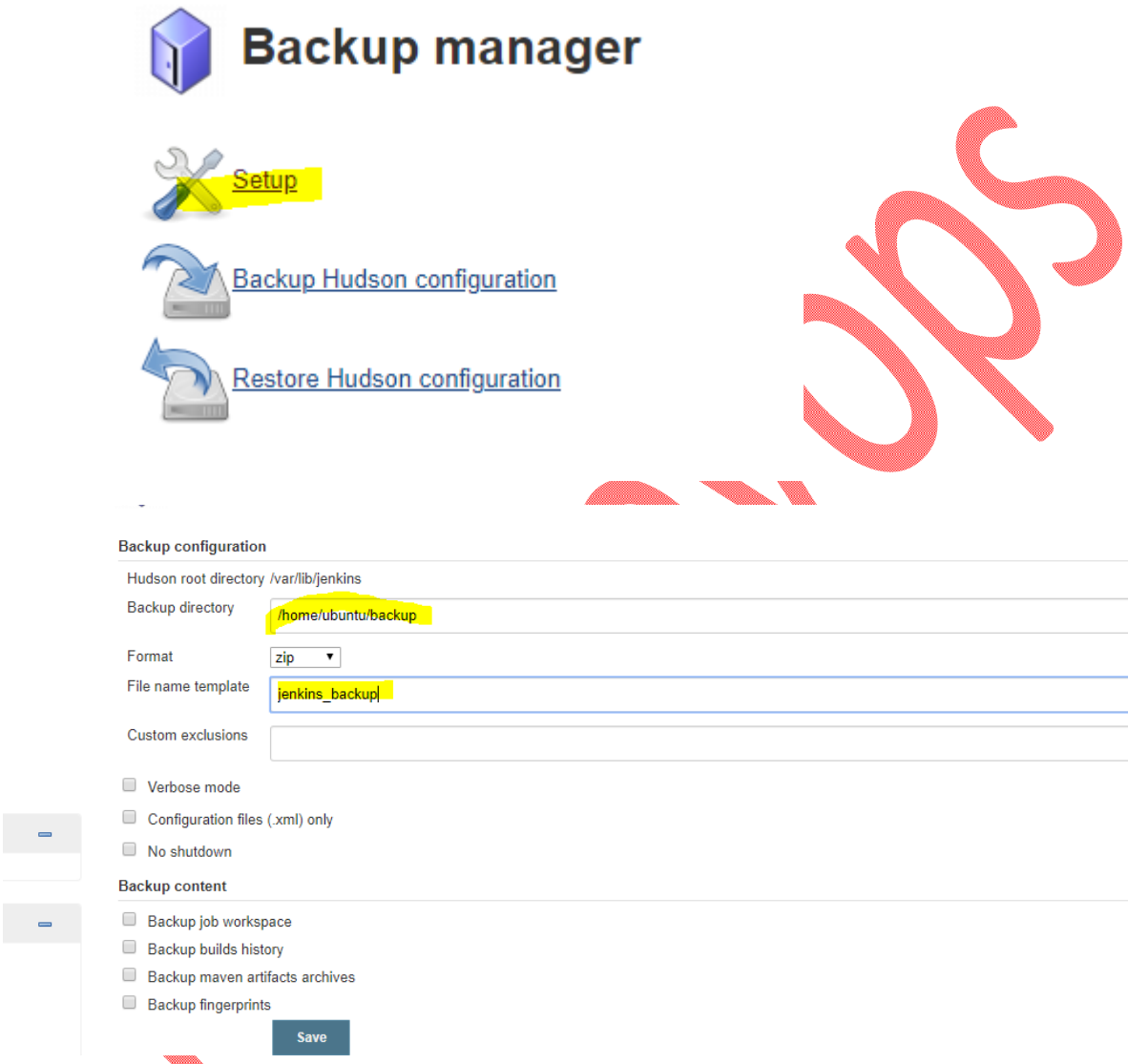
It is important to have a Jenkins backup with its data and configurations. It includes job configs, plugins, build logs, plugin configuration, etc.Jenkins provides a backup plugin which can be used to get backup critical configuration settings related to Jenkins. Let's see how to configure backup plugin in Jenkins:

Install "**Backup**" plugin

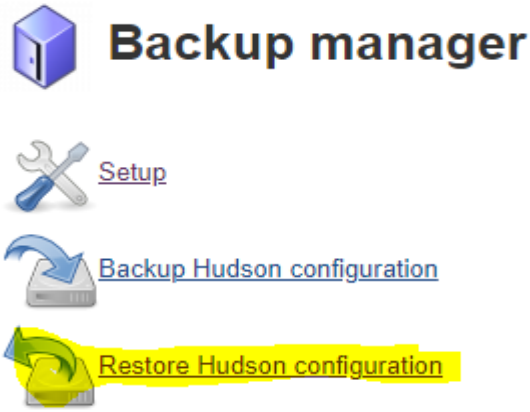


Go to the **Manage Jenkins** option, and scroll down, you will see **Backup manager** as an option. Select this option.





To recover from a backup, go to the **Backup manager** screen, and click on **Restore Hudson configuration**.



Best practices in Jenkins

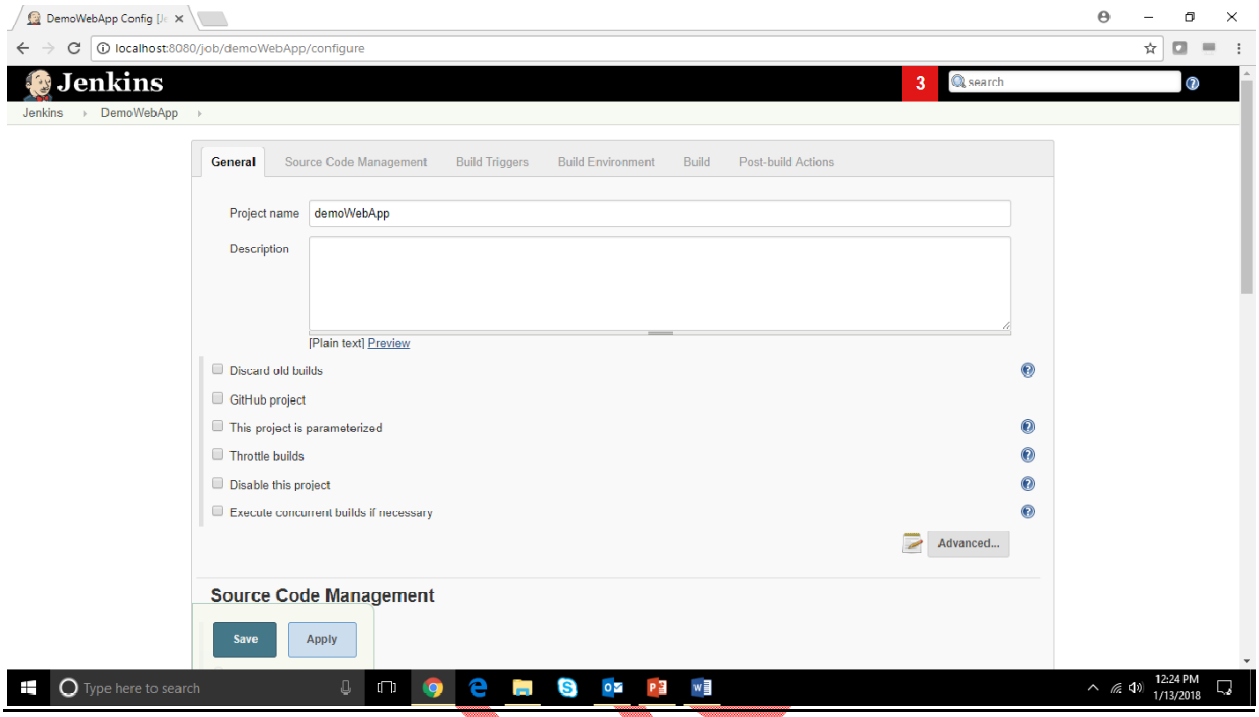
- Always secure Jenkins
- In larger systems don't build the master.(All jobs are running in master it will become slow)
- Backup Jenkins home regularly (If we delete some jobs again it will if we restart)
- Integrate the Jira (or) Bugzilla tools.
- Setup Jenkins home having more disk space.
- Archive unused jobs before removing.
- Avoid scheduling all jobs to start at the same time.(we will face performance issue)

Jenkins Terminologies

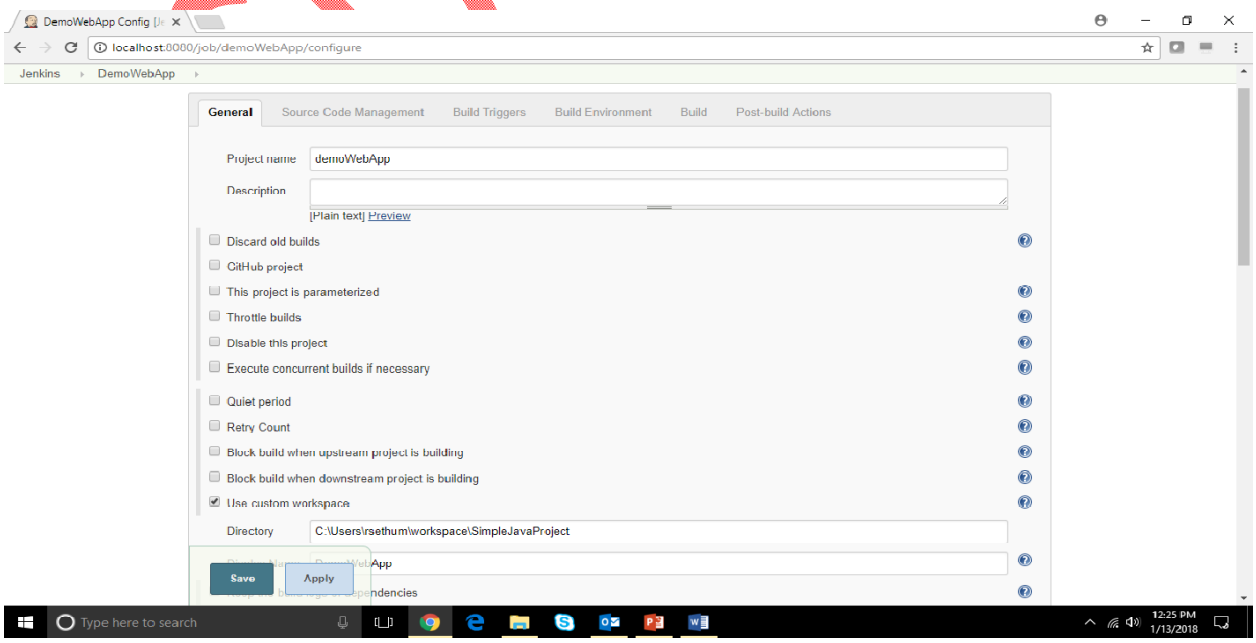
Term used in Jenkins	Description
Upstream project	A project can have one or several upstream projects, which means that current project is scheduled depending if the upstream project is built successfully or not. If the upstream project is successful the current project is added to the build queue. If the upstream project is broken the current project will not be added to the build queue.
Downstream project	A project can have one or several downstream projects. The downstream projects are added to the build queue if the current project is built successfully. It is an option to add the downstream project to the build queue even if the current project is unstable (default is off).
Stable build	A build is stable if it was built successfully and no publisher reports it as unstable.
Unstable build	A build is unstable if it was built successfully and one or more publishers report it unstable. For example if the JUnit publisher is configured and a test fails then the build will be marked unstable.
(Un)Stable project	A project is (un)stable if its most recent (completed) build is (un)stable.
Successful build	A build is successful when the compilation reported no errors.
Broken build Failed Build	A build is broken if it failed during building. That is, it is not successful.
Broken project	A project is broken if its most recent (completed) build is broken.
Slave	Slaves are computers that are set up to build projects for a master. Jenkins runs a separate program called "slave agent" on slaves.When slaves are registered to a master, a master starts distributing loads to slaves.
Publisher	A publisher is part of the build process other than compilation, for example JUnit test runs. A publisher may report *stable* or *unstable* result depending on the result of its processing. For example, if a JUnit test fails, then the whole JUnit publisher may report *unstable*.
Completed Build	A build is completed, if it was started and finished with any result, including failed builds.

Selenium Integration

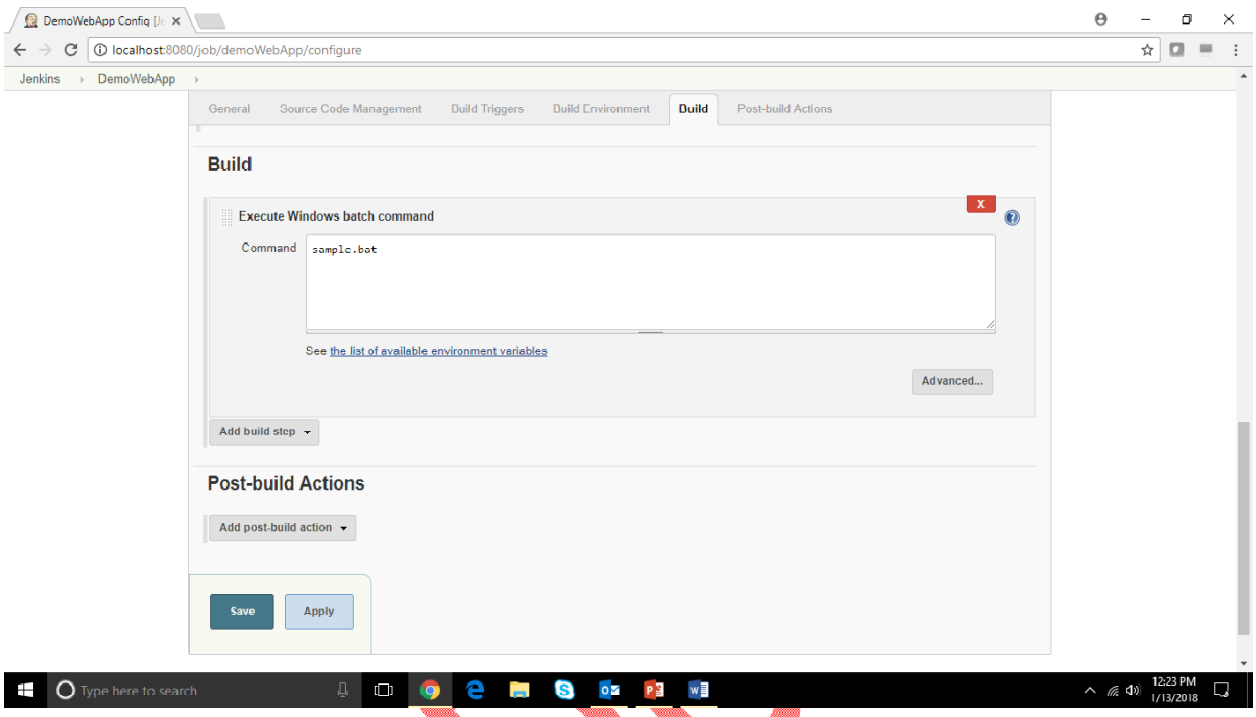
Click on advanced



Declare a customer workspace

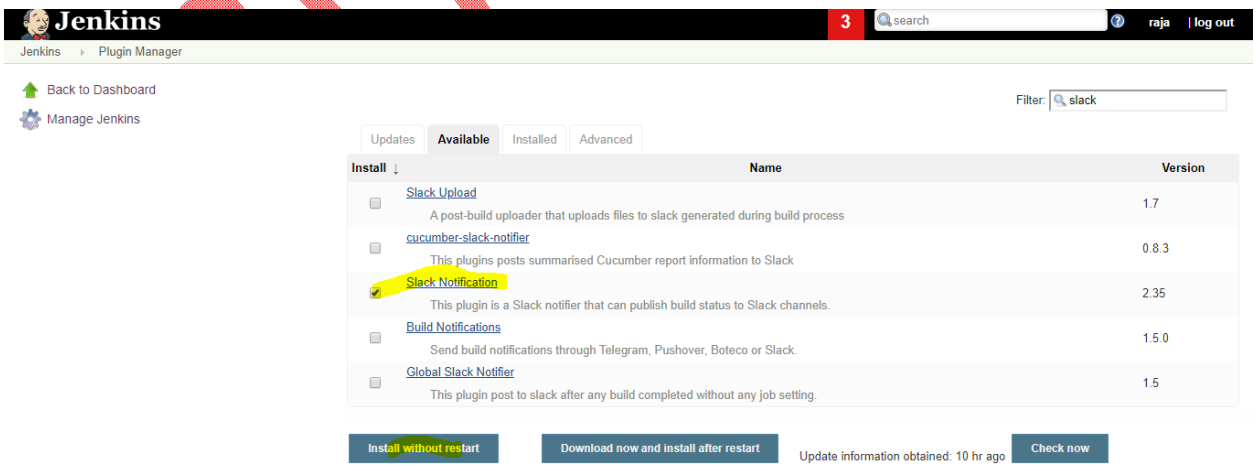


In the build section of the job, mention the name of the batch file created and then perform Build now operation in Jenkins.

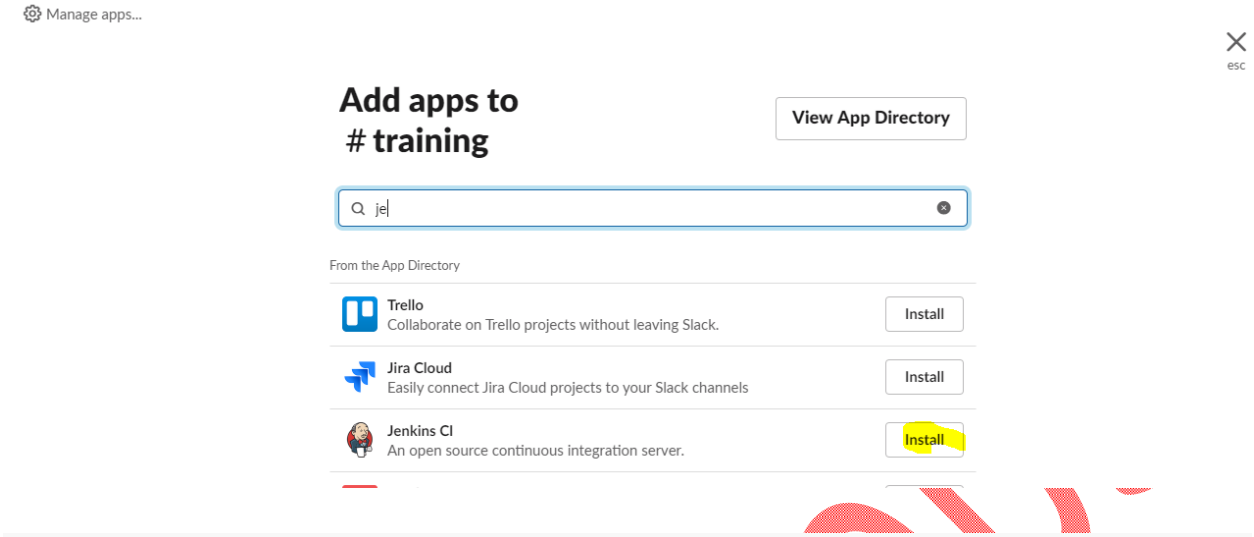


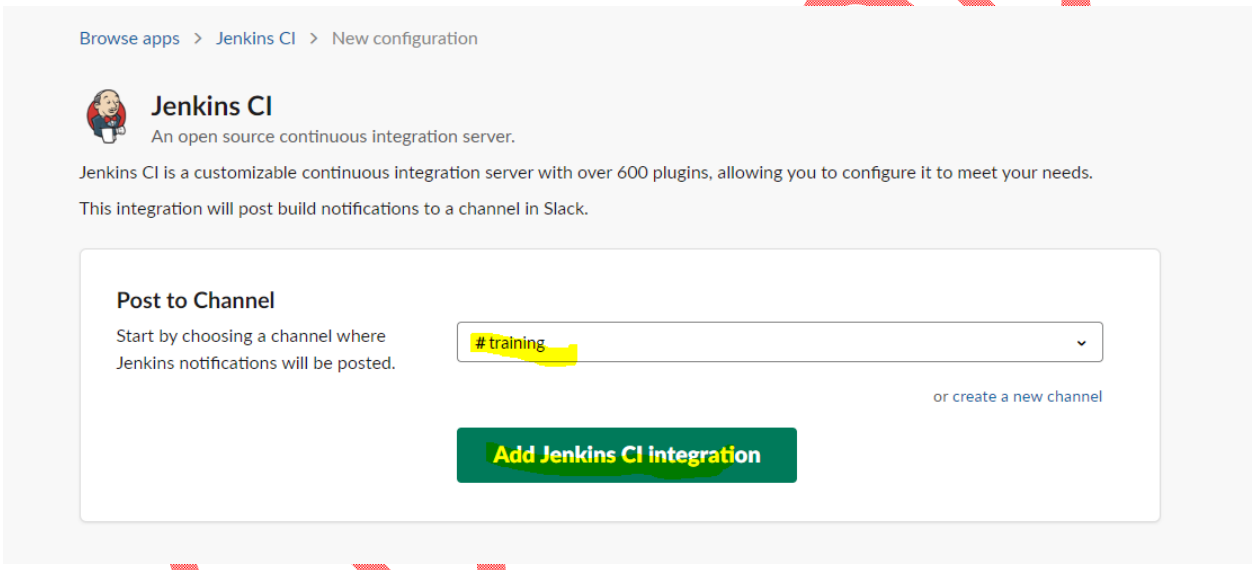
Enable Slack Notifications

Install Slack Notification plugin

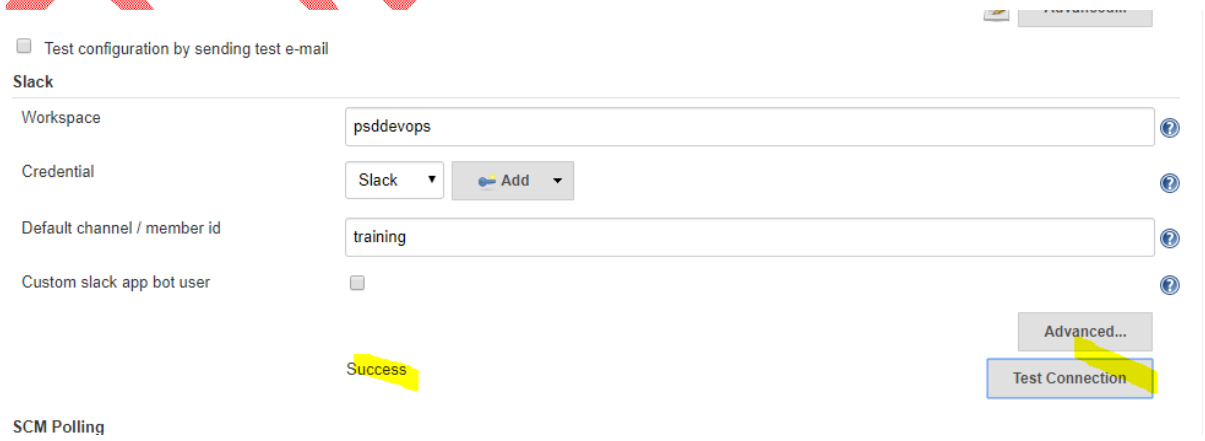


Add our channel to jenkins CI

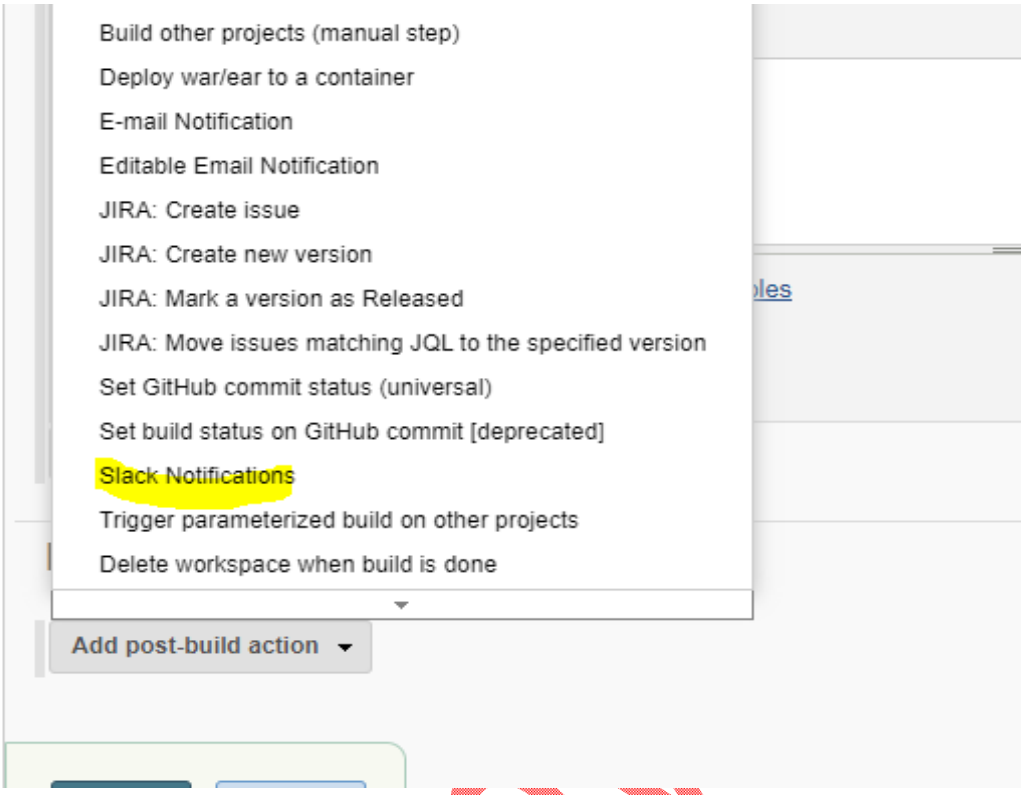


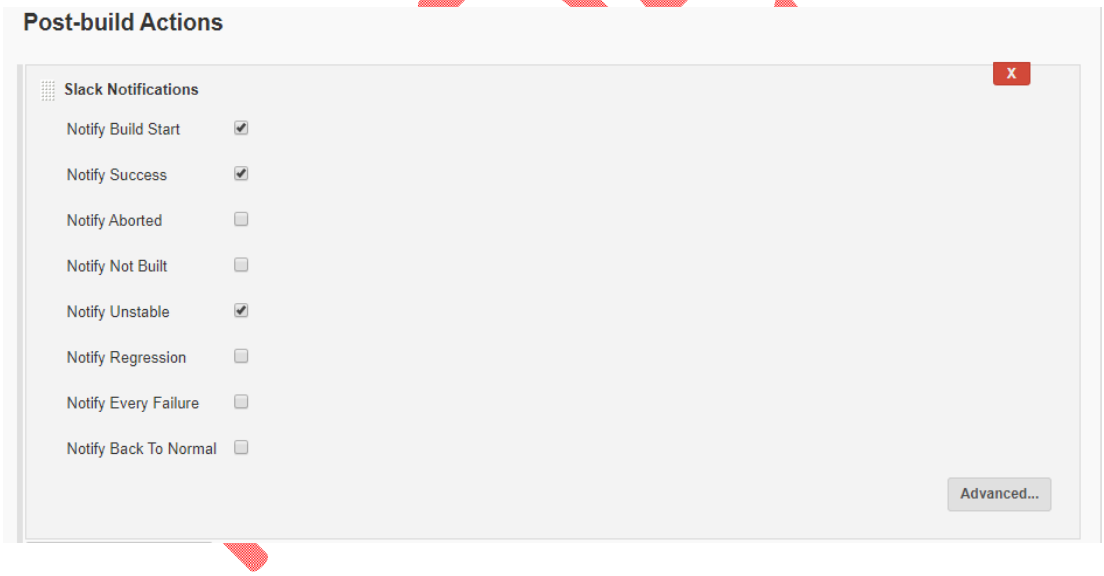


Copy the token and workspace and configure



Go to post build actions and configure the slack details.


Build other projects (manual step)
Deploy war/ear to a container
E-mail Notification
Editable Email Notification
JIRA: Create issue
JIRA: Create new version
JIRA: Mark a version as Released
JIRA: Move issues matching JQL to the specified version
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Slack Notifications
Trigger parameterized build on other projects
Delete workspace when build is done
Add post-build action


Post-build Actions
Slack Notifications
Notify Build Start
Notify Success
Notify Aborted
Notify Not Built
Notify Unstable
Notify Regression
Notify Every Failure
Notify Back To Normal
Advanced...

Console Output

```
Started by user raja
Running as SYSTEM
Building on master in workspace /var/lib/jenkins/workspace/test
[test] $ /bin/sh -xe /tmp/jenkins6186169339891940941.sh
+ echo /var/lib/jenkins/workspace/test
/var/lib/jenkins/workspace/test
[Slack Notifications] found #1 as previous completed, non-aborted build
[Slack Notifications] will send OnSuccessNotification because build matches and user preferences allow it
Finished: SUCCESS
```

Jenkins Pipeline

- Jenkins Pipeline (or simply "Pipeline") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.
- A *continuous delivery pipeline* is an automated expression of your process for getting software from version control right through to your users and customers.
- By using DSL(Domain Specific Language) i.e. groovy syntax we can implement Jenkins pipeline.

Advantages

- Migration will become easy from one server to other server.
- If anything goes wrong we can revert easily.
- Pipelines have control (In the form of the code).
- You can create pipelines automatically for all branches and execute pull requests with just one Jenkins File.
- You can review your code on the pipeline
- You can audit your Jenkins pipeline
- This is the singular source for your pipeline and can be modified by multiple users.

Why Use Jenkins's Pipeline?

Jenkins is an open continuous integration server which has the ability to support the automation of software development processes. You can create multiple automation jobs with the help of use cases, and run them as a Jenkins pipeline.

Here are the reasons why you use should use Jenkins pipeline:

- Jenkins pipeline is implemented as a code which allows multiple users to edit and execute the pipeline process.
- Pipelines are robust. So if your server undergoes an unforeseen restart, the pipeline will be automatically resumed.
- You can pause the pipeline process and make it wait to resume until there is an input from the user.
- Jenkins Pipelines support big projects. You can run multiple jobs, and even use pipelines in a loop.

Jenkins provides you with two ways of developing your pipeline code: Scripted and Declarative.

Scripted pipelines, also known as "traditional" pipelines, are based on ***Groovy*** as their ***Domain-specific language***.

Declarative pipelines provide a simplified and morefriendly syntax with specific statements for defining them, without needing to learn ***Groovy.***

Scripted Pipelines

- 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.
- 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.
- Scripted Jenkins pipeline runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to translate the pipeline into atomic commands.
- Scripted Pipelines follow a more **imperative programming model**.
- Scripted type has very few limitations that to with respect to structure and syntax that tend to be defined by Groovy, thus making it ideal for users with more complex requirements.
- The scripted pipeline is defined within a 'node'.

Syntax

```
node ('node-1') {
 stage('Source') {
  git 'https://github.com/digitalvarys/jenkins-tutorials.git''
 }
 stage('Compile') {
  def mvn_home = tool 'maven3'
  sh "'${mvn_home}/bin/mvn' clean compile"
 }
}
```

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

Syntax:

```
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:

Syntax:

```
stage {
    }
```

Declarative pipelines

- Declarative pipeline syntax offers an easy way to create pipelines. It contains a predefined hierarchy to create Jenkins pipelines. It gives you the ability to control all aspects of a pipeline execution in a simple, straight-forward manner.
- Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write. This code is written in a Jenkins file which can be checked into a source control management system such as Git.
- Declarative type imposes limitations to the user with a more strict and pre-defined structure, which would be ideal for simpler continuous delivery pipelines.
- The declarative pipeline is defined within a block labelled 'pipeline'.
- Declarative Pipeline encourages a declarative programming model.

Syntax:

```
pipeline {
agent any
stages {
stage ('build') {
        ...
        }
stage ('test: integration-&-quality') {
        ...
        }
stage ('test: functional') {
        ...
        }
stage ('test: load-&-security') {
        ...
        }
stage ('approval') {
...
}
stage ('deploy:prod') {
...
                }
        }
}
```

Note:

The **key difference** between Declarative pipeline and Scripted pipeline would be with respect to their **syntaxes** and their **flexibility**.

A valid Declarative pipeline must be defined with the "pipeline" sentence and include the next required sections:

- **Agent**
- **Stages**
- **Stage**
- **Steps**

Also, these are the available directives:

- **Environment (Defined at stage or pipeline level)**
- **Input (Defined at stage level)**
- **Options (Defined at stage or pipeline level)**

- **Parallel**
- **Parameters**
- **Post**
- **Script**
- **Tools**
- **Triggers**
- **When**

We will now describe each of the listed directives/sections, starting with the required ones.

Agent

Jenkins provides the ability to perform distributed builds by delegating them to "agent" nodes. Doing this allows you to execute several projects with only one instance of the Jenkins server, while the workload is distributed to its agents.

Agents should be labeled so they can be easily identified from each other. For example, nodes can be labeled by their platform (Linux, Windows, etc), by their versions or by their location, among others. The "agent" section configures on which nodes the pipeline can be run. Specifying "agent any" means that Jenkins will run the job on any of the available nodes.

Syntax

```
pipeline {
    agent any
    ...
    }
```

Stages

This section allows generation of different stages on your pipeline that will be visualized as different segments when the job is run.

Syntax

```
pipeline {
    agent any
    stages {
    ...
        }
    }
```

Stage

At least one "stage" section must be defined on the "stages" section. It will contain the work that the pipeline will execute. Stages must be named accordingly since Jenkins will display each of them on its interface.Jenkins graphically splits pipeline execution based on the defined stages and displays their duration and whether it was successful or not.

Syntax

```
pipeline {
agent any
stages {
stage ('build') {
...
}
stage ('test: integration-&-quality') {
...
}
stage ('test: functional') {
...
}
stage ('test: load-&-security') {
...
}
stage ('approval') {
...
}
stage ('deploy:prod') {
...
}       }       }
```

Steps

The last required section is "steps," which is defined into a "stage." At least one step must be defined in the "steps" section.

Syntax

Linux and MacOS, shell is supported.

```
steps {
sh 'echo "A one line step"'
sh '''
echo "A multiline step"'
cd /tests/results
ls -lrt
'''
}
```

Windows, bat or PowerShell can be used.

```
steps {
bat "mvn clean test -Dsuite=SMOKE_TEST -Denvironment=QA"
powershell ".\funcional_tests.ps1"
}
```

Environment

This directive can be both defined at stage or pipeline level, which will determine the scope of its definitions. When "environment" is used at the "pipeline" level, its definitions will be valid for all of the pipeline steps. If instead it is defined within a "stage," it will only be valid for the particular stage.

Syntax

At the "pipeline" level:

```
pipeline {
agent any
environment {
OUTPUT_PATH = './outputs/'
}
stages {
stage ('build') {
...
}
...
```

```
}
}
```
Here, "environment" is used at a "stage" level:

```
pipeline {
agent any
stages {
stage ('build') {
environment {
OUTPUT_PATH = './outputs/'
}
...
}
...
}}
```

Input

The "input" directive is defined at a stage level and provides the functionality to prompt for an input. The stage will be paused until a user manually confirms it.
The following configuration options can be used for this directive:

- message: This is a required option where the message to be displayed to the user is specified.
- id: Optional identifier for the input. By default, the "stage" name is used.
- ok: Optional text for the Ok button.
- submitter: Optional list of users or external group names who are allowed to submit the input. By default, any user is allowed.
- submitterParameter: Optional name of an environment variable to set with the submitter name, if present.
- parameters: Optional list of parameters to be provided by the submitter.

Syntax

```
pipeline {
agent any
stages {
stage ('build') {
input{
message "Press Ok to continue"
```

```
submitter "user1,user2"
parameters {
string(name:'username', defaultValue: 'user', description: 'Username of the user pressing Ok')
}
}
steps {
echo "User: ${username} said Ok."
}
}       }       }
```

Options

Defined at pipeline level, this directive will group the specific options for the whole pipeline. The available options are:

- buildDiscarder
- disableConcurrentBuilds
- overrideIndexTriggers
- skipDefaultCheckout
- skipStagesAfterUnstable
- checkoutToSubdirectory
- newContainerPerStage
- timeout
- retry
- Timestamps

Syntax

```
pipeline {
        agent any
        options {
        retry(3)
                }
        stages {
                ...
                }
        }
pipeline {
  options {
```

```
  buildDiscarder(logRotator(numToKeepStr: '30', artifactNumToKeepStr: '30'))
 }
 ...
}
```

- daysToKeepStr: history is only kept up to this days.
- numToKeepStr: only this number of build logs are kept.
- artifactDaysToKeepStr: artifacts are only kept up to this days.
- artifactNumToKeepStr: only this number of builds have their artifacts kept.

Parallel

Jenkins pipeline Stages can have other stages nested inside that will be executed in parallel. This is done by adding the "parallel" directive to your script. An example of how to use it is provided:

```
stage('run-parallel-branches') {
 steps {
  parallel(
   a: {
    echo "Tests on Linux"
   },
   b: {
    echo "Tests on Windows"
   }
  )
 }
}
```

Starting with Declarative Pipeline version 1.2, a new syntax was introduced, making the use of the parallel syntax much more declarative-like.

The previous script rewritten with this new syntax will look like:

```
pipeline {
 agent none
 stages {
  stage('Run Tests') {
   parallel {
    stage('Test On Windows') {
     agent {
      label "windows"
     }
     steps {
```

```
        bat "run-tests.bat"
       }
      }
      stage('Test On Linux') {
       agent {
        label "linux"
       }
       steps {
        sh "run-tests.sh"
       }
      }
     }
    }
   }
}
```

Any of the previous pipelines will look like this:

Both scripts will run the tests on different nodes since they run specific platform tests. Parallelism can also be used to simultaneously run stages on the same node by the use of multithreading, if your Jenkins server has enough CPU.

Some restrictions apply when using parallel stages:

A stage directive can have either a parallel or steps directive but not both.

A stage directive inside a parallel one cannot nest another parallel directive, only steps are allowed. Stage directives that have a parallel directive inside cannot have "agent" or "tools" directives defined.

Parameters

This directive allows you to define a list of parameters to be used in the script. Parameters should be provided once the pipeline is triggered. It should be defined at a "pipeline" level and only one directive is allowed for the whole pipeline.

String and Boolean are the valid parameter types that can be used.

```
pipeline {
agentany
parameters {
string(name: 'user', defaultValue: 'John', description: 'A user that triggers the pipeline')
 }
stages {
stage('Trigger pipeline') {
steps {
echo"Pipeline triggered by ${params.USER}"
  }
 }
}
}
```

Post

Post sections can be added at a pipeline level or on each stage block and sentences included in it are executed once the stage or pipeline completes. Several post-conditions can be used to control whether the post executes or not:

always: Steps are executed regardless of the completion status.

changed: Executes only if the completion results in a different status than the previous run.

fixed: Executes only if the completion is successful and the previous run failed

regression: Executes only if current execution fails, aborts or is unstable and the previous run was successful.

aborted: Steps are executed only if the pipeline or stage is aborted.

failure: Steps are executed only if the pipeline or stage fails.

success: Steps are executed only if the pipeline or stage succeeds.

unstable: Steps are executed only if the pipeline or stage is unstable.

Since sentences included in a pipeline post block will be run at the end of the script, cleanup tasks or notifications, among others, can be performed here.

```
pipeline {
agentany
stages {
stage('Some steps') {
steps {
   ...
  }
 }
```

```
 }
post {
always {
echo“Pipelinefinished”
bat. /performCleanUp.bat
 }
 }
}
```

Script

This step is used to add Scripted Pipeline sentences into a Declarative one, thus providing even more functionality. This step must be included at "stage" level.

Several times blocks of scripts can be utilized on different projects. These blocks allow you to extend Jenkins functionalities and can be implemented as shared libraries. More information on this can be found at Jenkins shared libraries. Also, shared libraries can be imported and used into the "script" block, thus extending pipeline functionalities.Next we will provide sample pipelines. The first one will only have a block with a piece of Scripted pipeline text, while the second one will show how to import and use shared libraries:

```
pipeline {
agentany
stages {
stage('Sample') {
steps {
echo"Scripted block"
script {
   }
  }
 }
}
}
```

Please refer to our post about Scripted pipelines at How to Use the Jenkins Scripted Pipeline for more information on this topic.

Tools

The "tools" directive can be added either at a pipeline level or at the stage level. It allows you to specify which maven, jdk, or gradle version to use on your script. Any of these tools, the three supported at the time of writing, must be configured on the

"Global tool configuration" Jenkins menu.Also, Jenkins will attempt to install the listed tool (if it is not installed yet). By using this directive you can make sure a specific version required for your project is installed.

```
pipeline {
agentany
tools {
maven'apache-maven-3.0.1'
 }
stages {
  ...
 }
}
```

Triggers

Triggers allow Jenkins to automatically trigger pipelines by using any of the available ones:

cron: By using cron syntax, it allows to define when the pipeline will be re-triggered.

pollSCM: By using cron syntax, it allows you to define when Jenkins will check for new source repository updates. The Pipeline will be re-triggered if changes are detected. (Available starting with Jenkins 2.22).

upstream: Takes as input a list of Jenkins jobs and a threshold. The pipeline will be triggered when any of the jobs on the list finish with the threshold condition.

Sample pipelines with the available triggers are shown next:

```
pipeline {
agentany
triggers {
//Execute weekdays every four hours starting at minute 0
cron('0 */4 * * 1-5')
 }
stages {
  ...
 }
}
pipeline {
agentany
triggers {
//Query repository weekdays every four hours starting at minute 0
pollSCM('0 */4 * * 1-5')
 }
```

```
stages {
  ...
 }
}
pipeline {
agentany
triggers {
//Execute when either job1 or job2 are successful
upstream(upstreamProjects:'job1,job2',threshold:hudson.model.Result.SUCCESS)
 }
stages {
  ...
 }
}
```

When

Pipeline steps could be executed depending on the conditions defined in a "when" directive. If conditions match, the steps defined in the corresponding stage will be run. It should be defined at a stage level.

For a full list of the conditions and its explanations refer to Jenkins declarative pipeline "when" directive.

For example, pipelines allow you to perform tasks on projects with more than one branch. This is known as multibranched pipelines, where specific actions can be taken depending on the branch name like "master", "feature*", "development", among others. Here is a sample pipeline that will run the steps for the master branch:

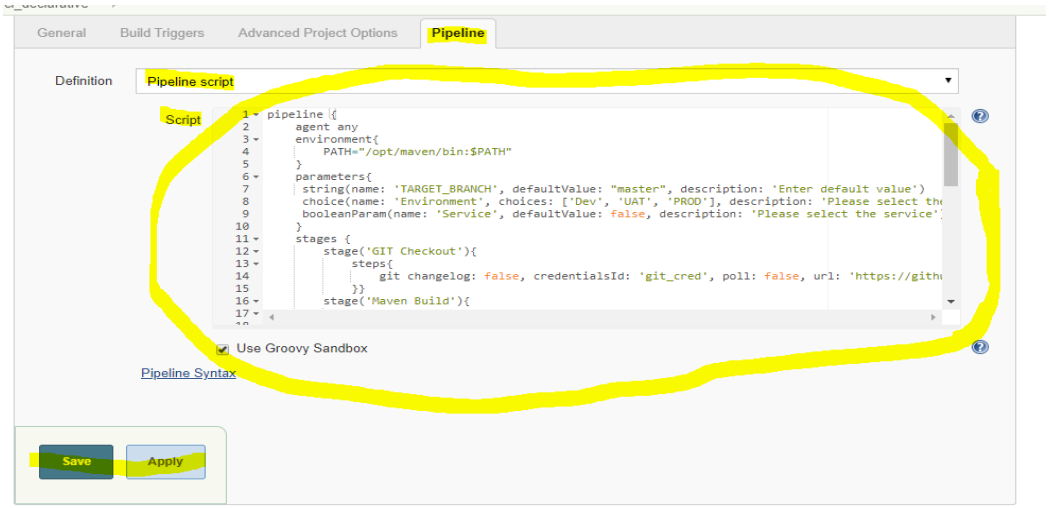
```
pipeline {
agentany
stages {
stage('Deploy stage') {
when {
branch'master'
   }
steps {
echo'Deploy master to stage'
    ...
  }
 }
}
}
```

How many ways we can write Jenkins pipelines ?

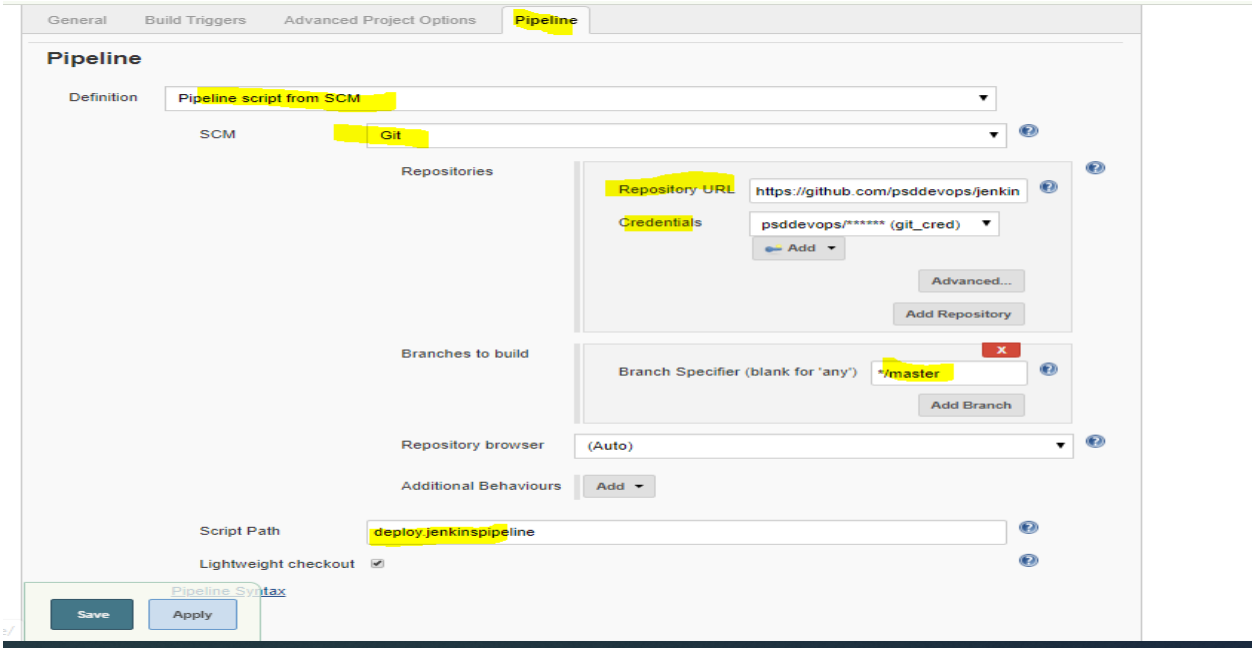
2 – Ways

1. In-line
2. Pull from SCM

In-line



Pull from SCM



Thanks.....

All The Best.....