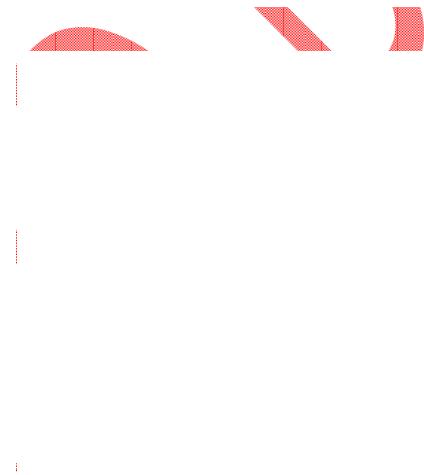




# Jenkins



# 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.	The code is built and tests as soon as Developer commits code. Jenkin will build and test code many times during the day
Code commit built, and test cycle was very infrequent, and a single build was done after many days.	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 build 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.

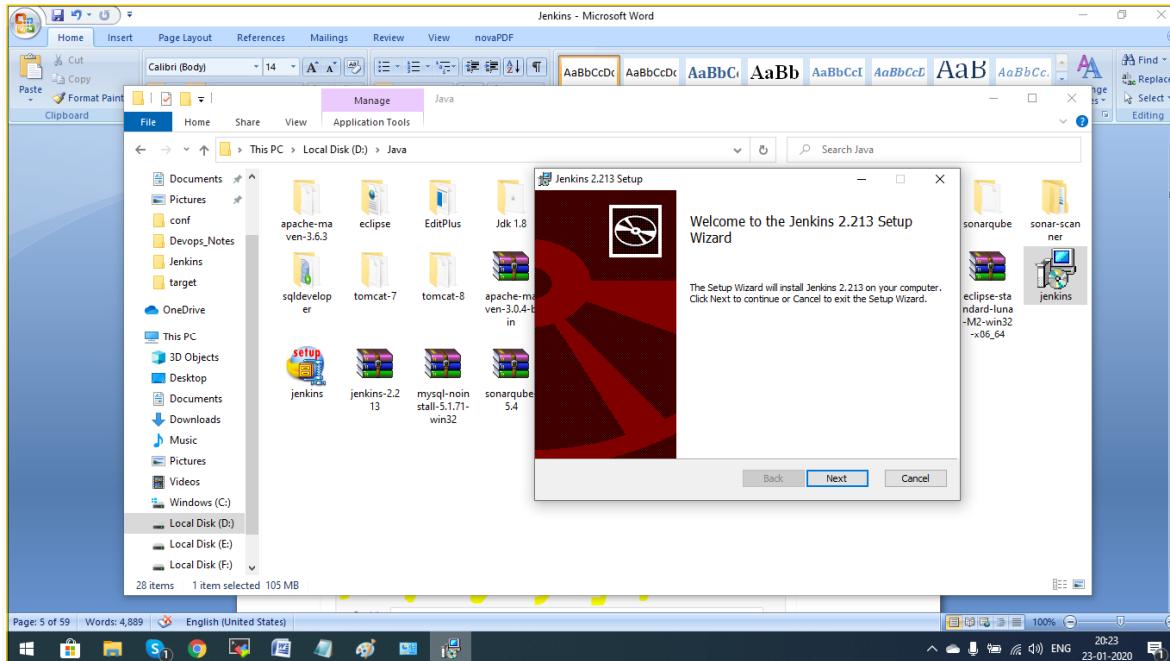
## Bamboo

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 best 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 stop Jenkins**

```
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'
```

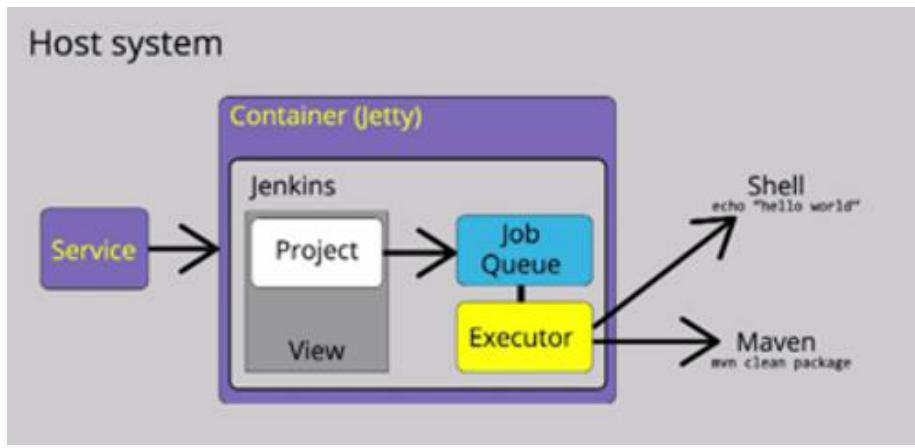
```
sudo apt-get update
```

```
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:**

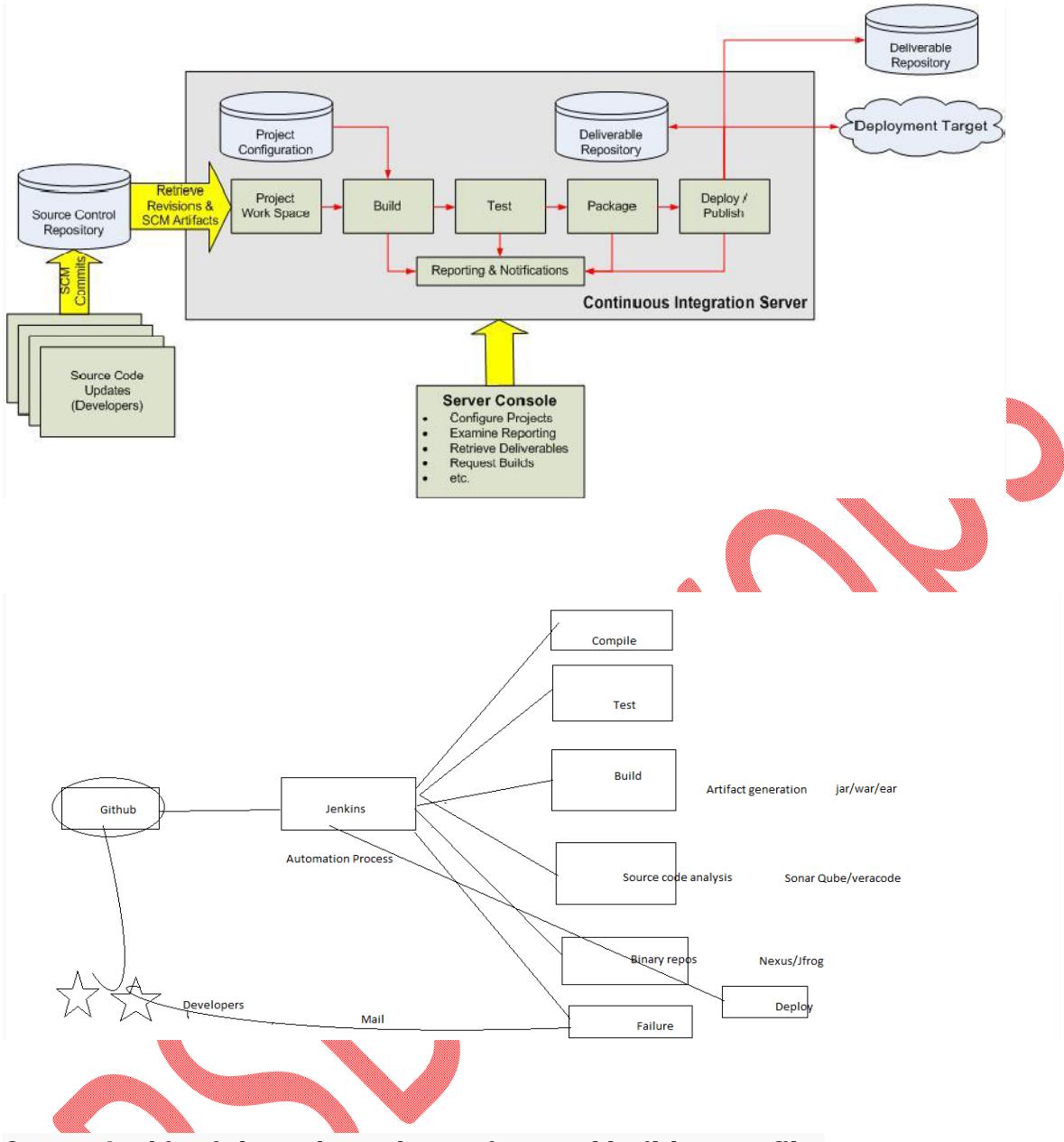
It's the exact section what it has do the Jenkins job.

### **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

**Source Code Management**

None  
 Git

**Repositories**

Repository URL: <https://github.com/psddevops/sampletest.git>

Credentials: - none -

**Branches to build**

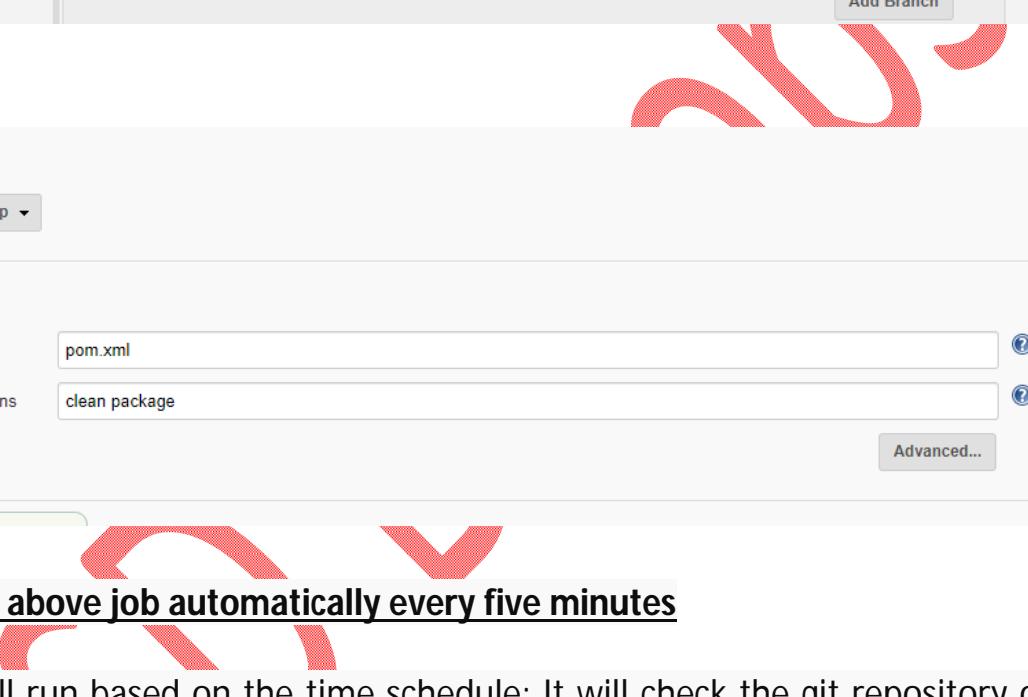
Branch Specifier (blank for 'any'): \*/master

**Pre Steps**

**Build**

Root POM: pom.xml

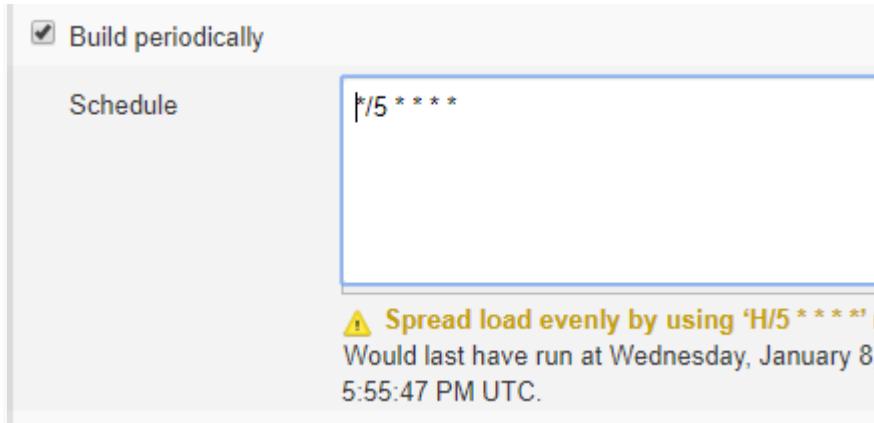
Goals and options: clean package



### 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,

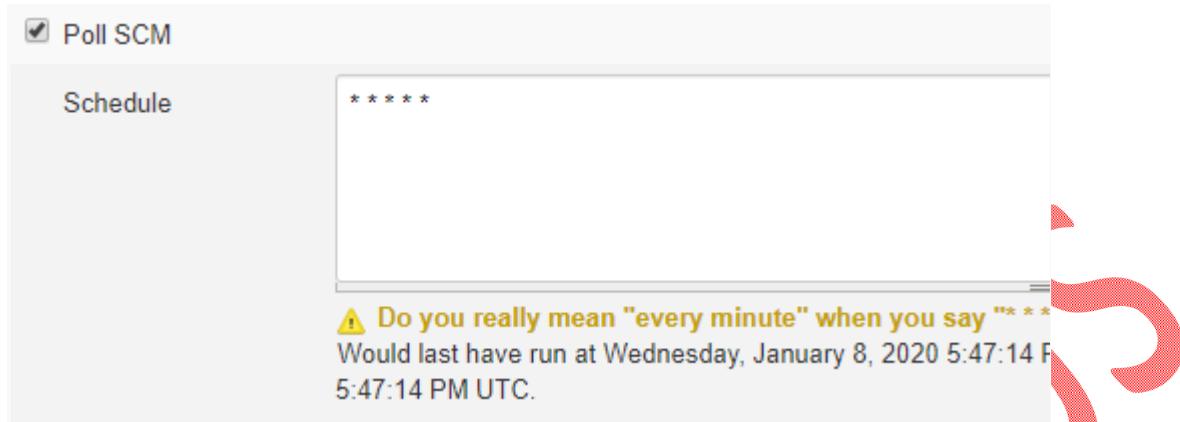
- \* specifies all valid values
- M-N specifies a range of values
- M-N/X or \*/X steps by intervals of X through the specified range or whole valid range
- A,B,...,Z enumerates multiple values

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

PollScm will check 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 given 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

**Webhooks**

- Notifications
- Integrations & services
- Deploy keys
- Secrets
- Actions

Moderation

Interaction limits

Payload URL \*

http://18.191.245.57:8080/github-webhook/

Content type

application/json

Secret

Which events would you like to trigger this webhook?

Just the push event.

Send me everything.

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
```

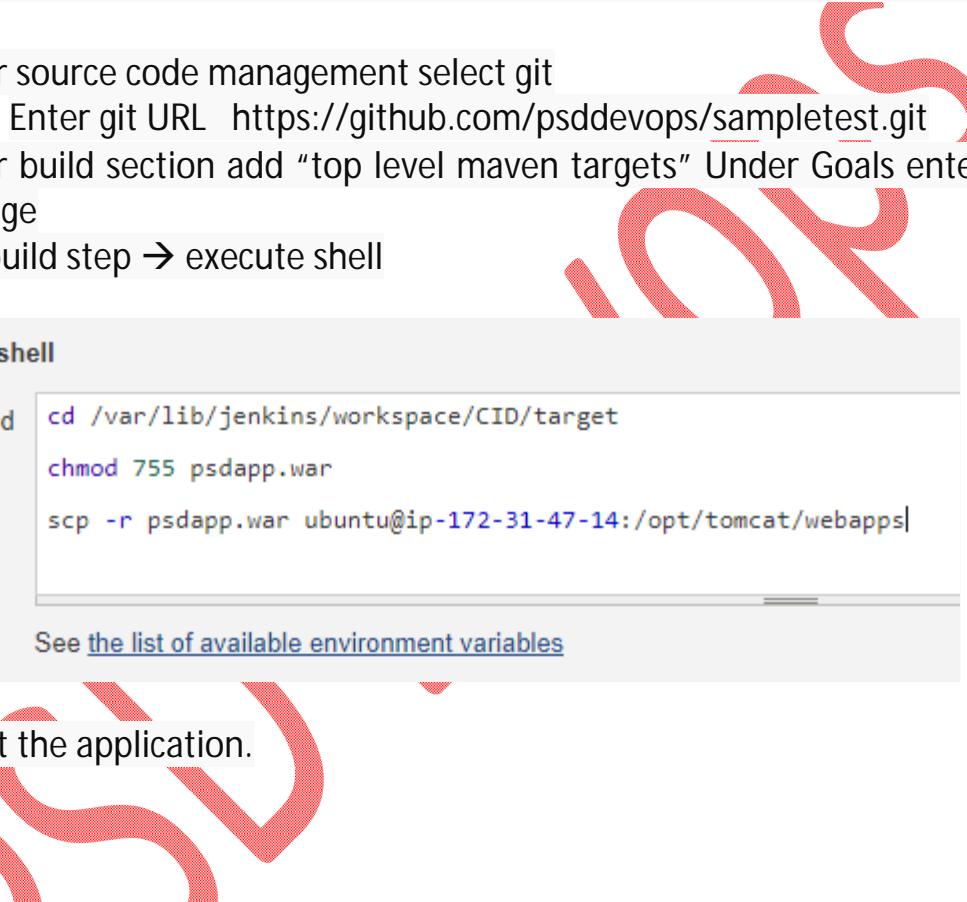
add the public key to the tomcat server.

```
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



```
Execute shell
Command
cd /var/lib/jenkins/workspace/CID/target
chmod 755 psdapp.war
scp -r psdapp.war ubuntu@ip-172-31-47-14:/opt/tomcat/webapps|
```

See [the list of available environment variables](#)

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" >

<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" >

<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

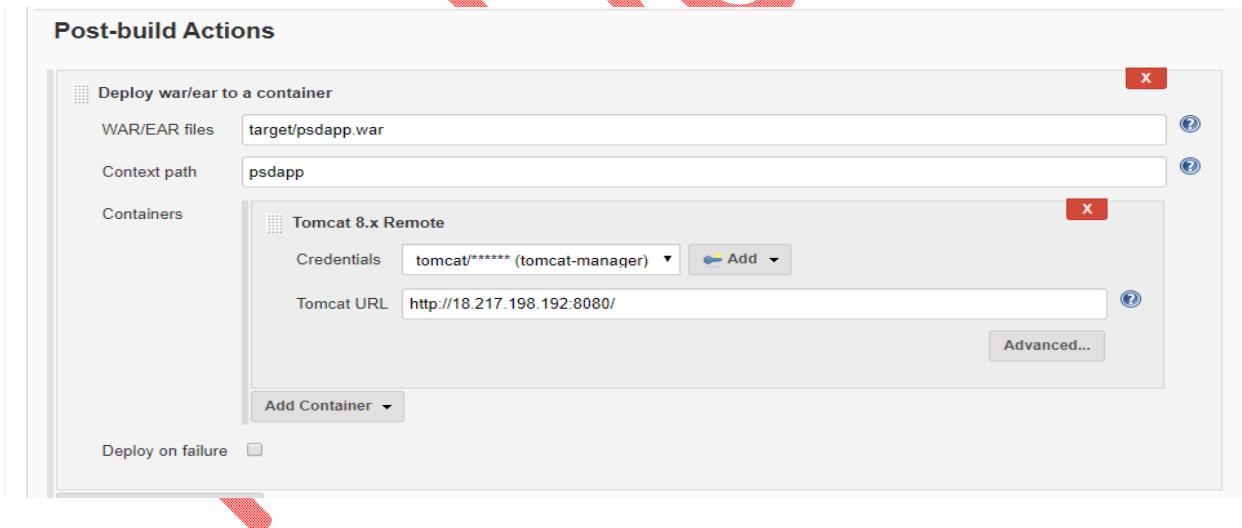
Manage Jenkins → Manage Plugins → Available →

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/psdapp.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

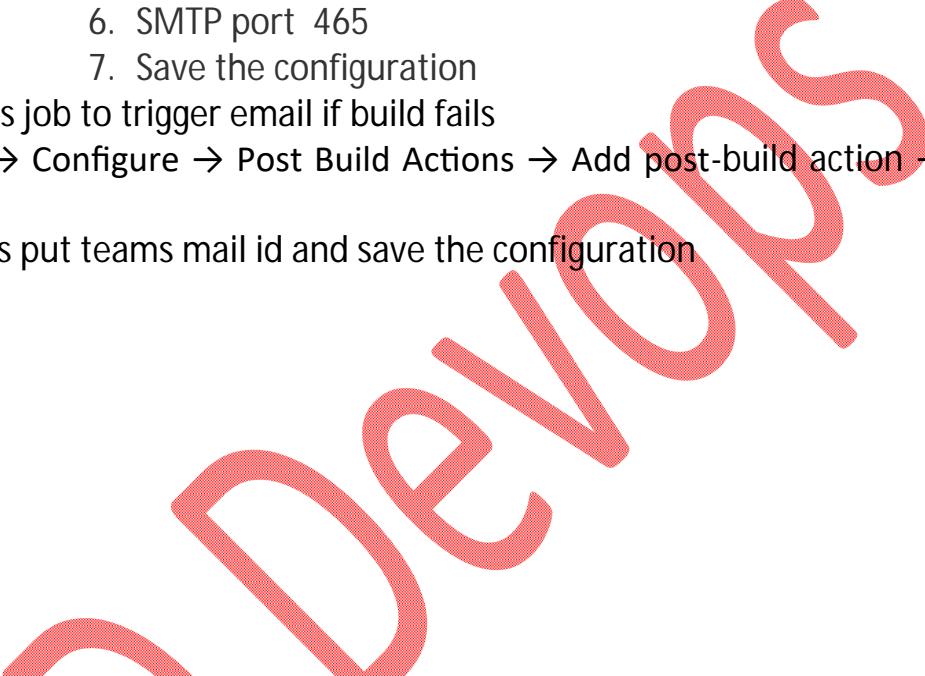
1. SMTP server → **smtp.gmail.com**
2. Select Use SMTP Authentication
3. Put your gmail id
4. Put your gmail password
5. Use SSL select
6. SMTP port 465
7. Save the configuration

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:

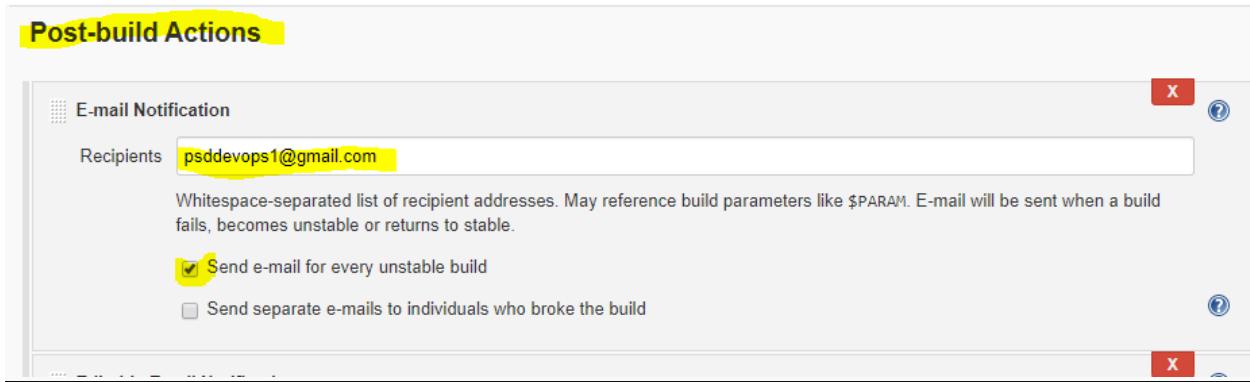


E-mail Notification

SMTP server	<input type="text" value="smtp.gmail.com"/>
Default user e-mail suffix	<input type="text"/>
<input checked="" type="checkbox"/> Use SMTP Authentication	<input type="checkbox"/>
User Name	<input type="text" value="psddevops1@gmail.com"/>
Password	<input type="password" value="....."/>
Use SSL	<input checked="" type="checkbox"/>
SMTP Port	<input type="text" value="465"/>
Reply-To Address	<input type="text"/>
Charset	<input type="text" value="UTF-8"/>

### Job configuration:

Goto → Postbuild action → Email-notification



## Email notifications with attachment

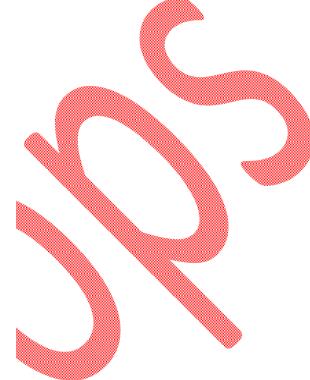
Jenkins → Manage Jenkins → Configure System → Extended E-mail Notification

The screenshot shows the 'Extended E-mail Notification' configuration in Jenkins' 'Configure System' section. It includes fields for the SMTP server ('smtp.gmail.com'), Default user E-mail suffix, 'Use SMTP Authentication' (checked), User Name ('psddevops1@gmail.com'), Password (redacted), Advanced Email Properties (a large text area), 'Use SSL' (checked), SMTP port ('465'), Charset ('UTF-8'), and Additional accounts ('Add'). The interface includes standard Jenkins UI elements like help icons and a red 'X' button.

Extended E-mail Notification	
SMTP server	smtp.gmail.com
Default user E-mail suffix	
<input checked="" type="checkbox"/> Use SMTP Authentication	
User Name	psddevops1@gmail.com
Password	[REDACTED]
Advanced Email Properties	
Use SSL	<input checked="" type="checkbox"/>
SMTP port	465
Charset	UTF-8
Additional accounts	Add

## Job configuration:

Goto → Postbuild action → Editable Email-notification



Editable Email Notification

Disable Extended Email Publisher

Allows the user to disable the publisher, while maintaining the settings

Project From

Project Recipient List

Project Reply-To List

Content Type

Default Subject

Default Content

Attachments

Can use wildcards like 'module/dist\*\*\*.zip'. See the [@includes of Ant files](#) for the exact format. The base directory is [the workspace](#).

Attach Build Log

Content Token Reference

Pre-send Script

Post-send Script

Additional groovy classpath

Save to Workspace

Triggers

Failure - Any

Send To

Recipient List

Add Trigger

Save  Apply

Failure - Any

Send To	Developers
	Recipient List
	Add ▾
Recipient List	<b>psddevops1@gmail.com</b>
Reply-To List	<b>SPROJECT_DEFAULT_REPLYTO</b>
Content Type	<b>Project Content Type</b>
Subject	<b>SPROJECT_DEFAULT_SUBJECT</b>
Content	<b>SPROJECT_DEFAULT_CONTENT</b>

### 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.

General	Source Code Management	<b>Build Triggers</b>	Build Environment	Build	Post-build Actions
<input checked="" type="checkbox"/> Poll Mailbox Trigger					
Host	<b>imap.gmail.com</b>				
Username	<b>psddevops1@gmail.com</b>				
Password	*****				
Advanced Email Properties	<b>subjectContains=sample</b>				
Schedule	*****				

**Do you really mean "every minute" when you say "\*\*\*\*\*"? Perhaps you meant "H \* \* \* \*" to poll once per hour**  
 Would last have run at Thursday, January 9, 2020 5:50:55 PM UTC; would next run at Thursday, January 9, 2020

### Integrating Junit test cases to the Jenkins

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

## Post-build Actions

Publish JUnit test result report

Test report XMLs

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/\*.xml'. Basedir of the fileset is [the workspace root](#).

Retain long standard output/error [?](#)

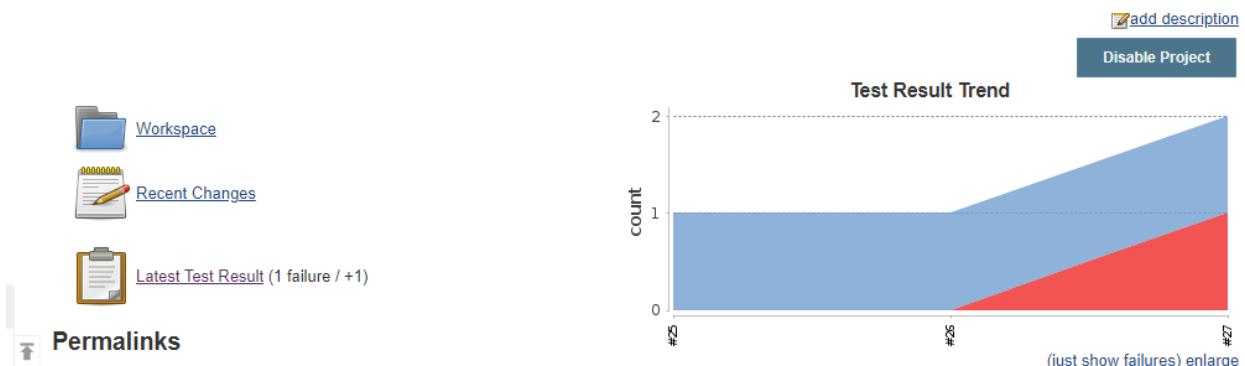
Health report amplification factor

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Allow empty results  Do not fail the build on empty test results [?](#)

Add post-build action ▾

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 system

### Status Information



#### System Information

Displays various environmental information to assist trouble shooting.



#### System Log

System log captures output from Java.vt11.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/troubleshooting/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

## 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.

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.

The screenshot shows the Jenkins configuration interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', etc. The main area is titled 'Maven Project Configuration' and contains the following fields:

- Home directory: /var/lib/jenkins
- System Message: (empty text area)
- Global MAVEN\_OPTS: (dropdown menu)
- Local Maven Repository: Default (~/.m2/repository) (dropdown menu)
- # of executors: 2
- Labels: (empty text area)
- Usage: Use this node as much as possible (dropdown menu)
- Quiet period: 5

## 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.



Jenkins > Configure Global Security

## Configure Global Security

Enable security  Disable remember me

**Access Control**

Delegate to servlet container  Jenkins' own user database  Allow users to sign up

LDAP  Unix user/group database

**Security Realm**

Anyone can do anything  Legacy mode  Logged-in users can do anything  Matrix-based security  Project-based Matrix Authorization Strategy

**Authorization**

## 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.

Jenkins > Global Tool Configuration

## Global Tool Configuration

Maven Configuration

Default settings provider: Use default maven settings

Default global settings provider: Use default maven global settings

JDK

JDK installations: Add JDK

Git

Git installations:

Git	Name: Default
	Path to Git executable: git
<input type="checkbox"/> Install automatically	<input type="checkbox"/>
Delete Git	

Add Git

## 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.

The screenshot shows the Jenkins Plugin Manager interface. The top navigation bar includes links for Back to Dashboard, Manage Jenkins, and Update Center. The main content area has tabs for Updates, Available, Installed, and Advanced, with Available selected. A search bar and a user dropdown are also present. The main table lists various .NET Development plugins:

Install	Name	Version
<input type="checkbox"/>	CCM	3.2
<input type="checkbox"/>	change-assembly-version-plugin	1.10
<input type="checkbox"/>	FxCop.Runner	1.1
<input type="checkbox"/>	MSBuild	1.29
<input type="checkbox"/>	MSTest	1.0.0
<input type="checkbox"/>	MSTestRunner	1.3.0
<input type="checkbox"/>	NAnt	1.4.3
<input type="checkbox"/>	PowerShell	1.4

## 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

The screenshot shows the Jenkins System Properties page. The left sidebar includes links for New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, My Views, Lockable Resources, Credentials, Open Blue Ocean, and New View. The main content area shows a table of system properties:

Name	Value
awt.toolkit	sun.awt.X11Toolkit
executable-war	/usr/share/jenkins/jenkins.war
file.encoding	UTF-8
file.encoding.pkg	sun.io
file.separator	/
java.awt.graphicsenv	sun.awt.X11GraphicsEnvironment
java.awt.headless	true
java.awt.printerjob	sun.print.PSPrinterJob
java.class.path	/usr/share/jenkins/jenkins.war
java.class.version	52.0
java.endorsed.dirs	/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/endorsed
java.ext.dirs	/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext:/usr/java/packages/lib/ext
java.home	/usr/lib/jvm/java-8-openjdk-amd64/jre
java.io.tmpdir	/tmp
java.library.path	/usr/java/packages/lib/amd64:/usr/lib/x86_64-linux-gnu/jni/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni/lib/
java.runtime.name	OpenJDK Runtime Environment
java.runtime.version	1.8_0_232-8u232-b09-0ubuntu1~18.04.1-b09
java.specification.name	Java Platform API Specification

## 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.

The screenshot shows the Jenkins System Log interface. At the top, there's a navigation bar with links for Back to Dashboard, Manage Jenkins, Log Recorders, All Log Messages, and Log Levels. The main area is titled "Jenkins Log" and contains a message stating "Log messages at a level lower than INFO are never recorded in the Jenkins log. Use [log recorders](#) to record these log messages." Below this, a scrollable list of log entries is displayed:

```
Jan 06, 2020 3:47:54 PM INFO hudson.model.Run execute
Test_Job #6 main build action completed: SUCCESS
Jan 06, 2020 3:48:04 PM INFO hudson.model.Run execute
Deploy_Job #6 main build action completed: SUCCESS
Jan 06, 2020 3:55:19 PM INFO Hudson.PluginManager install
Starting installation of a batch of 1 plugins plus their dependencies
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of blueocean-core-js for plugin blueocean
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of jenkins-design-language for plugin blueocean-core-js
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of blueocean-github-pipeline for plugin blueocean
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of blueocean-pipeline-api-impl for plugin blueocean-github-pipeline
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of blueocean-pipeline-scm-api for plugin blueocean-pipeline-api-impl
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of blueocean-rest for plugin blueocean-pipeline-scm-api
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of blueocean-commons for plugin blueocean-rest
Jan 06, 2020 3:55:19 PM INFO hudson.model.UpdateSitePlugin deploy
Adding dependent install of pubsub-light for plugin blueocean-pipeline-scm-api
```

## 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.

The screenshot shows the Jenkins Load statistics page. On the left, there's a sidebar with links for New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, My Views, Lockable Resources, Credentials, Open Blue Ocean, and New View. The main area is titled "Load statistics: Jenkins" and includes a message about the timespan: "Timespan: [Short](#) [Medium](#) [Long](#)". Below this is a chart showing the number of builds over time. At the bottom, there are two sections: "Build Queue" (which says "No builds in the queue.") and "Build Executor Status" (which shows "1 Idle" and "2 Idle").

## 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.

The screenshot shows the Jenkins CLI page. On the left is a sidebar with links like New Item, People, Build History, etc. The main content area has a title 'Jenkins CLI' with a sub-section 'Available Commands'. It lists several Jenkins commands with their descriptions:

Command	Description
<code>add-job-to-view</code>	Adds jobs to view.
<code>build</code>	Builds a job, and optionally waits until its completion.
<code>cancel-quiet-down</code>	Cancel the effect of the "quiet-down" command.
<code>clear-queue</code>	Clears the build queue.
<code>connect-node</code>	Reconnect to a node(s)
<code>console</code>	Retrieves console output of a build.
<code>copy-job</code>	Copies a job.
<code>create-credentials-by-xml</code>	Create Credential by XML
<code>create-credentials-domain-by-xml</code>	Create Credentials Domain by XML
<code>create-job</code>	Creates a new job by reading stdin as a configuration XML file.
<code>create-node</code>	Creates a new node by reading stdin as a XML configuration.
<code>create-view</code>	Creates a new view by reading stdin as a XML configuration.

## 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.

The screenshot shows the Jenkins Script Console page. The left sidebar is identical to the Jenkins CLI page. The main content area has a title 'Script Console' with instructions: 'Type in an arbitrary Groovy script and execute it on the server. Useful for trouble-shooting and diagnostics. Use the 'println' command to see the output (if you use System.out, it will go to the server's stdout, which is harder to see.) Example:' followed by a code example: `println(Jenkins.instance.pluginManager.plugins)`. Below the example, it says 'All the classes from all the plugins are visible. jenkins.\*, jenkins.model.\*, hudson.\*, and hudson.model.\* are pre-imported.' A large text input area is provided for running scripts.

## 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 build nodes. A build node (slave) is another machine that Jenkins can use to execute its builds.

The screenshot shows the Jenkins interface with the title 'Jenkins'. The left sidebar has links for 'Back to Dashboard', 'Manage Jenkins', 'New Node', and 'Configure'. The main content area is titled 'Nodes' and displays a table with one row for the 'master' node. The table columns are: S, Name (with a dropdown arrow), Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The 'master' node is listed with the following details: Name: master, Architecture: Linux (amd64), Clock Difference: In sync, Free Disk Space: 4.90 GB, Free Swap Space: 0 B, Free Temp Space: 4.90 GB, and Response Time: 0ms. A 'Refresh status' button is at the bottom right of the table.

## 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.

The screenshot shows the Jenkins interface with the title 'About Jenkins'. The left sidebar has links for 'Jenkins' and 'About Jenkins'. The main content area is titled 'About Jenkins 2.204.1'. It includes a note that Jenkins is a community-developed open-source automation server. Below that, it says 'Jenkins depends on the following 3rd party libraries'. A table titled 'Mavenized dependencies' lists various libraries with their Maven ID and license. The table has columns: Name, Maven ID, and License. Some examples include 'Jenkins war' (org.jenkins-ci.main:jenkins-war:2.204.1, The MIT license), 'Utility\_around\_Java\_Crypto\_API' (org.jenkins-ci.crypto-util:1.1, MIT license), and 'Apache Commons BeanUtils' (commons-beanutils:commons-beanutils:1.9.3, Apache License Version 2.0).

Name	Maven ID	License
Jenkins war	org.jenkins-ci.main:jenkins-war:2.204.1	The MIT license
Utility_around_Java_Crypto_API	org.jenkins-ci.crypto-util:1.1	MIT license
HttpCommons Client library	commons-httpclient:commons-httpclient:3.1-jenkins-1	Apache License 2.0
AOP_alliance	aopalliance:aopalliance:1.0	Public Domain
Jenkins_core	org.jenkins-ci.main:jenkins-core:2.204.1	The MIT license
FindBugs-Annotations	com.google.code.findbugs:annotations:3.0.0	GNU Lesser Public License
Apache Commons BeanUtils	commons-beanutils:commons-beanutils:1.9.3	Apache License Version 2.0
Google Guice - Core Library	com.google.inject:guice:4.0	The Apache Software License Version 2.0
Agent installer module	org.jenkins-ci.modules.slave-installer:1.6	MIT License
Spring Framework: DAO	org.springframework:spring-dao:1.2.9	The Apache Software License Version 2.0
Stapler JRebel module	org.kohsuke.stapler:stapler-jrebel:1.258	2-clause BSD license
Apache Groovy	org.codehaus.groovy:groovy-all:2.4.12	The Apache Software License Version 2.0

## 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.

This URL requires POST

The URL you're trying to access requires that requests be sent using POST (like a form submission). The button below allows you to retry accessing this URL using POST. URL being accessed:  
<http://18.216.186.28:8080/quietDown>

If you were sent here from an untrusted source, please proceed with caution.

[Retry using POST](#)

## Manage users

Add and delete the users in the Jenkins server.



Back to Dashboard

Manage Jenkins

Create User

### Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User ID	Name	Action
ppreddy	ppreddy	

## Execute the same job in parallel

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



General    Source Code Management    Build Triggers    Build Environment    Build    Post-build Actions

Description

[Plain text] [Preview](#)

Discard old builds    
  
 GitHub project    
  
 This build requires lockable resources    
  
 This project is parameterized    
  
 Throttle builds    
  
 Disable this project    
  
 Execute concurrent builds if necessary

## Parameterized jobs in Jenkins

1. Create sample free style job

**Enter an item name**

Sample\_Parameter  
» Required field

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining something other than software build.

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

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for organizing complex activities that do not easily fit in free-style job type.

**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as Java builds with different database drivers.

**Bitbucket Team/Project**  
Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories and triggers a build for each repository.

**Folder**  
Creates a container that stores nested items in it. Useful for grouping things.

General   Source Code Management   Build Triggers   Build Environment   Build   Post-build Actions

**String Parameter**

Name	name
Default Value	Raja
Description	

**Choice Parameter**

Name	choice
Choices	A B C
Description	

## Build

The screenshot shows a Jenkins build configuration. At the top, there's a section titled "Execute shell". Below it, under "Command", there are two lines of text: "echo \$name" and "echo \$choice". A yellow checkmark icon is positioned to the right of the second command.

### Parameters with radio buttons ,check boxes and Dropdown

#### 1. Install Extended Choice Parameter

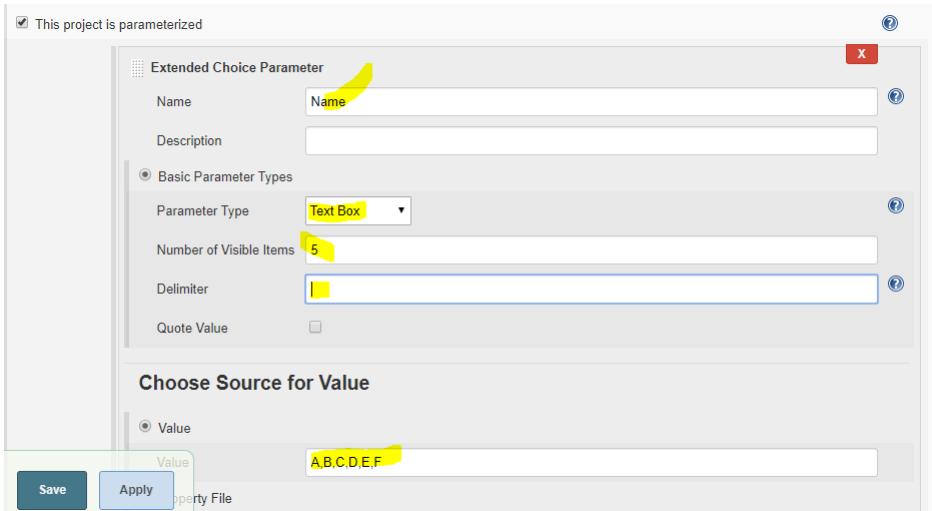
Jenkins → manage jenkins → manage plugins → Extended Choice Parameter

The screenshot shows the Jenkins "Manage Plugins" page. A specific plugin, "Extended Choice Parameter", is highlighted with a yellow circle and checked. Its description, "Adds extended functionality to Choice parameter", is visible below the checkbox. Other plugins like "Email Extension Template" and "Downstream-Ext" are also listed but not highlighted.

#### 2. Create a free style job

The screenshot shows the Jenkins "Enter an item name" screen. The field contains "sample\_multiparams". Below this, a list of project types is shown:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project something other than software build.
- Maven project**: Build a maven project. Jenkins takes advantage of your POM files ;
- Pipeline**: Orchestrates long-running activities that can span multiple build agents organizing complex activities that do not easily fit in free-style job ty
- Multi-configuration project**: Suitable for projects that need a large number of different configura
- Bitbucket Team/Project**: Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all i
- OK Folder**: Creates a container that stores nested items in it. Useful for groupin



**Build**

**Execute shell**

Command: echo \$Name

See [the list of available environment variables](#)

ops

## How to pass parameters to downstream job

### 1. Install Parameterized Trigger plugin

Jenkins → manage jenkins → manage plugins → Parameterized Trigger plugin

<input checked="" type="checkbox"/>	Multi-configuration (matrix) project type.	<a href="#">1.14</a>
<input checked="" type="checkbox"/>	<b>Parameterized Trigger plugin</b>	<a href="#">2.36</a>
<input checked="" type="checkbox"/>	<a href="#">Script Security Plugin</a>	
<input checked="" type="checkbox"/>	Allows Jenkins administrators to control what in-process scripts can be run by less-privileged users.	<a href="#">1.68</a>

### 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

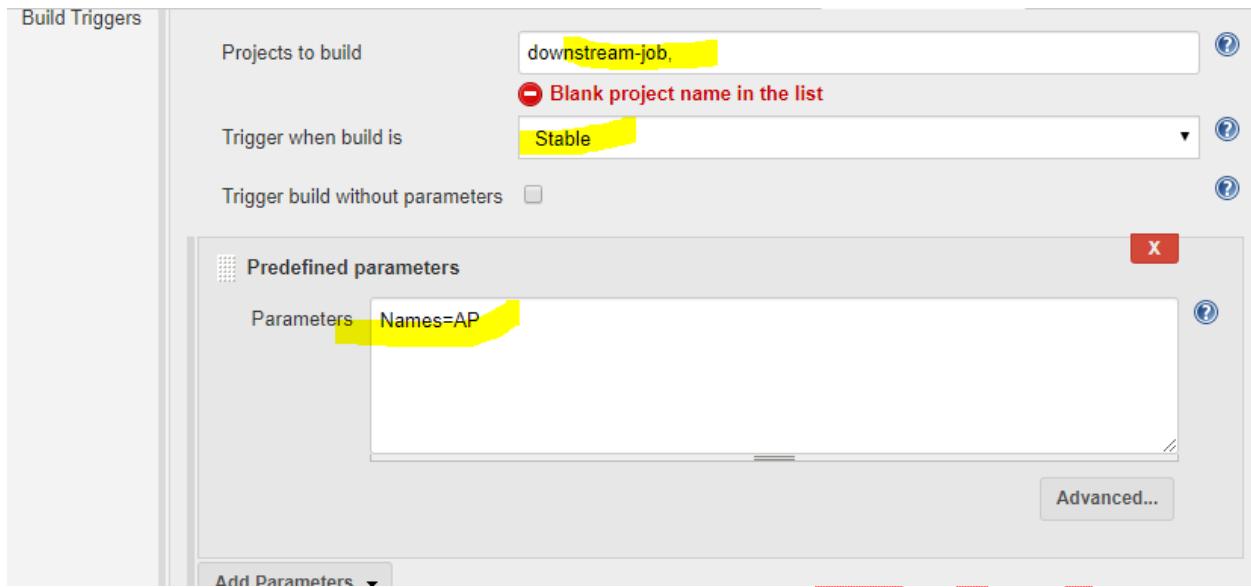
Trigger build without parameters:

Predefined parameters:

Parameters: Names=AP

Add Parameters

Advanced...



## Downstream

This project is parameterized

**String Parameter**

Name: Names

Default Value: KTK

Description:

[Plain text] Preview

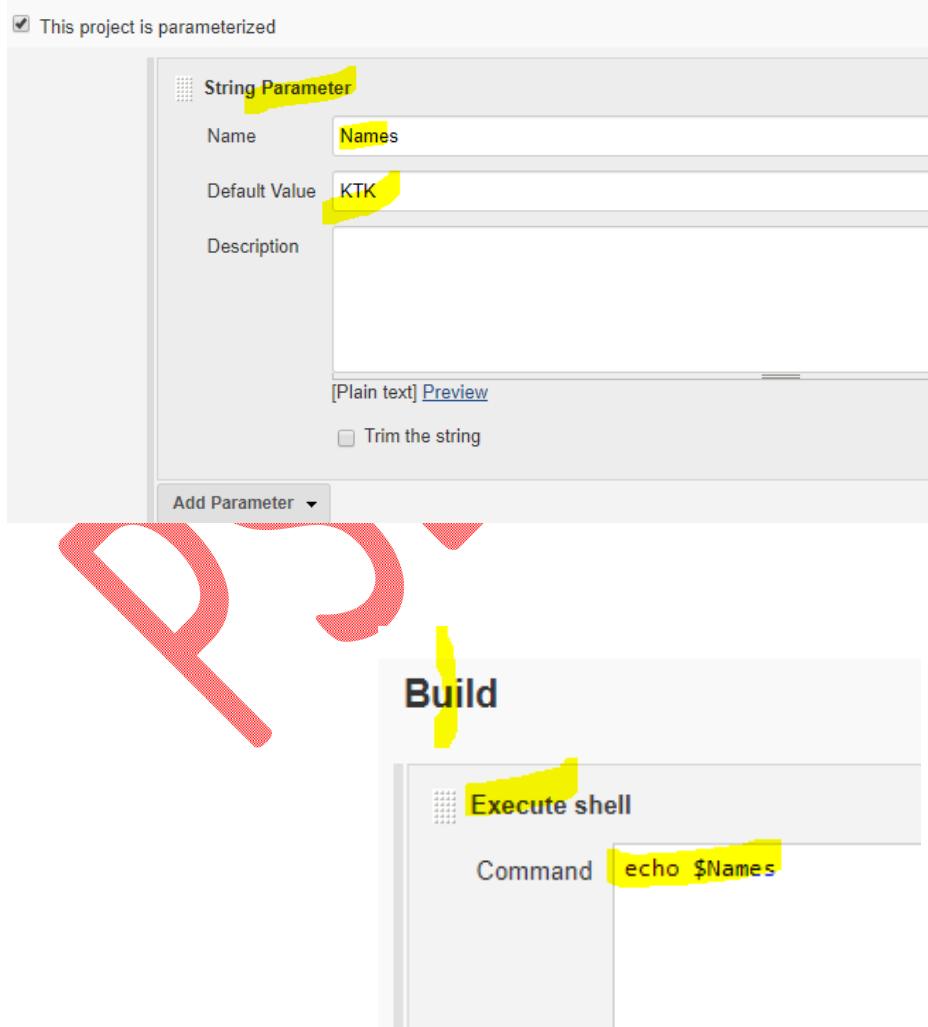
Trim the string

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
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

View name

**List View**  
Shows items in a simple list format. You can choose to show all jobs or filter by specific criteria.

**Build Monitor View**  
Shows a highly visible status of selected jobs. Ideal for monitoring critical jobs.

**Build Pipeline View**  
Shows the jobs in a build pipeline view. The common Jenkins jobs with upstream/downstream dependencies are visualized here.

**Delivery Pipeline View**  
Continuous Delivery pipelines, perfect for visualizing Jenkins jobs with upstream/downstream dependencies.

**Delivery Pipeline View for Jenkins Pipelines**  
Continuous Delivery pipelines, perfect for visualizing Jenkins Pipelines (created using the Pipeline or Workflow plugin).

**My View**  
This view automatically displays all the jobs that you have added to it.

**OK**

**Name** testview

**Description**

[Plain text] Preview

Filter build queue  
 Filter build executors

**Job Filters**

Status Filter All selected jobs

Jobs

Recurse in subfolders  
 Build\_Job  
 Deploy\_Job  
 downstream-job  
 sample\_multiparams  
 Sample\_Parameter  
 sample\_test  
 sonar\_test  
 Test\_Job

**OK** **Apply**

---

## 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

<input checked="" type="checkbox"/>	This plugin allows you to store credentials in Jenkins.	<a href="#">2.3.0</a>
<input checked="" type="checkbox"/>	<a href="#">Dashboard for Blue Ocean</a>	<a href="#">1.21.0</a>
<input checked="" type="checkbox"/>	Blue Ocean Dashboard	
<input checked="" type="checkbox"/>	<a href="#">Delivery Pipeline Plugin</a>	
<input checked="" type="checkbox"/>	This plugin visualize Delivery Pipelines (Jobs with upstream/downstream dependencies)	<a href="#">1.4.0</a>
<input checked="" type="checkbox"/>	<a href="#">Design Language</a>	<a href="#">1.21.0</a>
<input checked="" type="checkbox"/>	The Jenkins Plugins Parent POM Project	
	<a href="#">Display URL API</a>	

### 3. Add Delivery Pipeline View and configure the view

Jenkins > Delivery\_Test >

New Item      Name: **Delivery\_Test**

People      View settings

Build History      Description:

**Edit View**      Number of pipeline instances per pipeline: 3

Delete View      Display aggregated pipeline for each pipeline:

Project Relationship      Display aggregated changelog in aggregated view:

Check File Fingerprint      Group aggregated changelog by regular expression:

View Fullscreen      Number of columns: 1

Manage Jenkins      Sorting: None

My Views      Max number of pipelines: -1

Lockable Resources      Update interval: 2

Credentials      Enable start of new pipeline build:

Open Blue Ocean      Enable manual triggers:

New View

Build Queue      No builds in the queue.

Build Executor Status

OK      Apply

Pipelines

Components

Component	Name: <b>mydelivery</b>
Initial Job	<b>Build_Job</b>
Final Job (optional)	
Show upstream	<input type="checkbox"/>

Delete

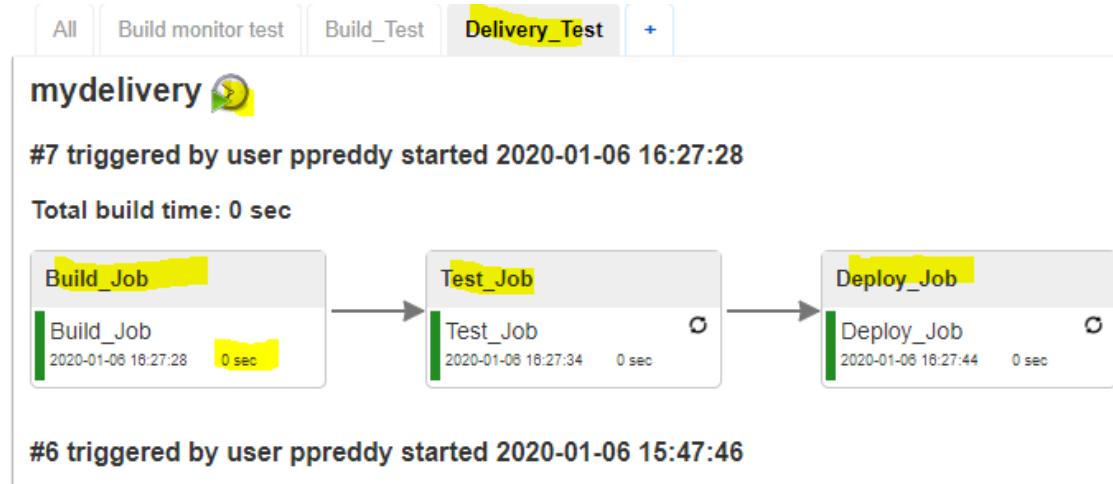
Add

Regular Expression

Add

OK      Apply

### 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.

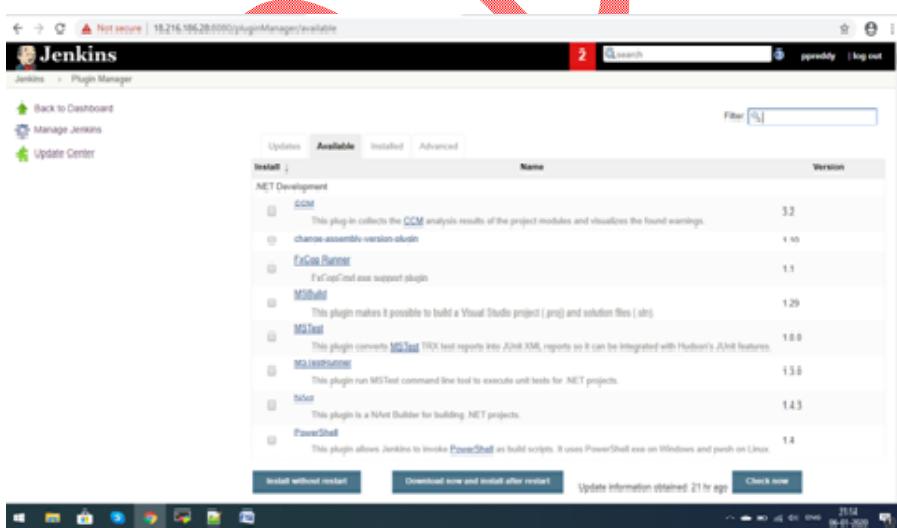
1. Chain required jobs in sequence

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

Not secure | 18.216.186.28:8080/newView

**Jenkins**

New Item

Build Pipeline View

Delivery Pipeline View

Delivery Pipeline View for Jenkins Pipelines

List View

My View

Build Queue

No builds in the queue

OK

Build Executor Status

1 idle

2 idle

Page generated: Jan 8, 2020 3:45:49 PM UTC REST API Jenkins ver. 2.204.1

Not secure | 18.216.186.28:8080/view/build\_1st/configure

**Jenkins**

Name: Build\_1st

Description:

Build Pipeline View Type: Local

Trigger Options: Poll SCM, Build Periodic

Drop Options: No Of Checkout Stages, Row Headers, Column Headers, Refresh Frequency (in seconds), URL for system CSS file, Console Output Link Style

Page generated: Jan 8, 2020 3:45:49 PM UTC REST API Jenkins ver. 2.204.1

#### 4. Run and Validate

Not secure | 18.216.186.28:8080/view/Build\_1st/

**Jenkins**

Build Pipeline

Run History Configure Add Step Delete Manage

Pipeline	Job	Start Time	Duration
#6	#6 Build_Job	Jan 6, 2020 3:47:46 PM	16 ms
#6	#6 Test_Job	Jan 6, 2020 3:47:54 PM	15 ms
#6	#6 Deploy_Job	Jan 6, 2020 3:48:04 PM	15 ms
#5	#5 Build_Job	Jan 6, 2020 3:47:29 PM	21 ms
#5	#5 Test_Job	Jan 6, 2020 3:47:39 PM	28 ms
#5	#5 Deploy_Job	Jan 6, 2020 3:47:46 PM	54 ms
#4	#4 Build_Job	Jan 6, 2020 3:30:32 PM	23 ms
#4	#4 Test_Job	Jan 6, 2020 3:30:39 PM	23 ms
#4	#4 Deploy_Job	Jan 6, 2020 3:30:49 PM	57 ms

#### Build Monitor View

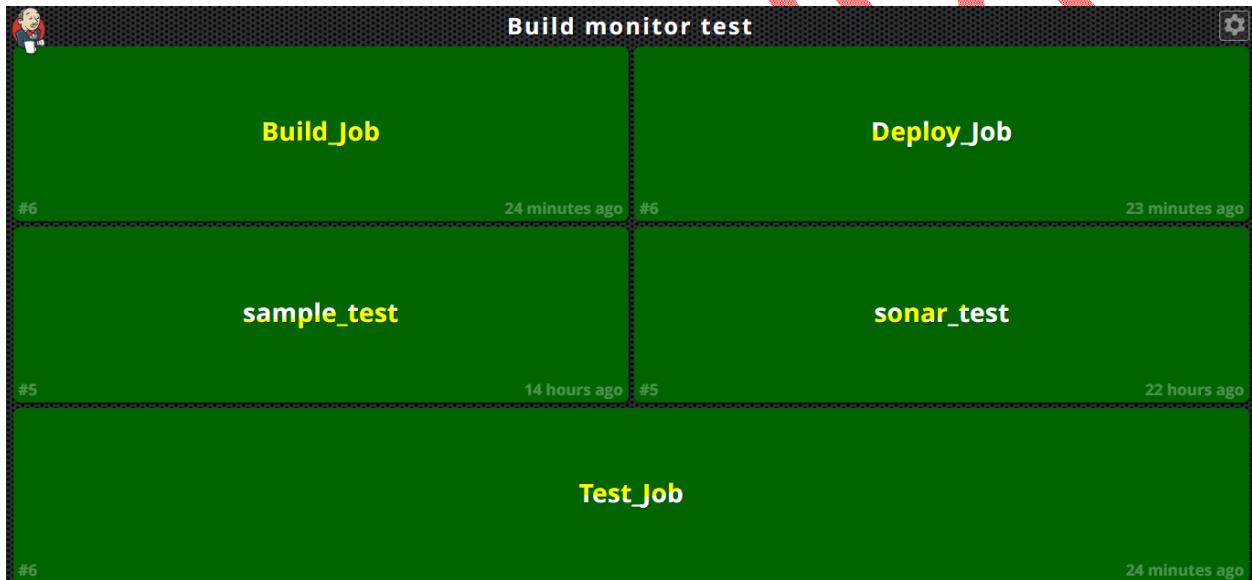
Provides a highly visible view of the status of selected Jenkins jobs. It easily accommodates different computer screen sizes and is ideal as an Extreme Feedback Device to be displayed on a screen on your office wall

## 1. Install Build monitor plugin

Jenkins → manage jenkins → manage plugins →Build monitor Plugin

The screenshot shows the Jenkins Manage Plugins interface. The search bar at the top right contains 'Build Mon'. Below it, there are tabs: 'Updates' (disabled), 'Available' (selected), 'Installed', and 'Advanced'. A sub-section titled 'Install' is shown, featuring a highlighted item 'Build Monitor View'. This item has a description: 'Provides a highly visible view 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.' Below the description are three buttons: 'Install without restart', 'Download now and install after restart', and 'Check now'. A note says 'Update information obtained: 22 hr ago'.

## 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

The screenshot shows the Jenkins Plugin Manager interface. The 'Available' tab is selected. A search bar at the top right contains the text 'Blue'. In the list, the 'Blue Ocean' plugin is highlighted with a red arrow pointing to it. The list includes other plugins like 'LFX notifier - smart lightbulbs build indicator', 'Common API for Blue Ocean', and 'Dashboard for Blue Ocean'. At the bottom of the list, there are buttons for 'Install without restart', 'Download now and install after restart', and 'Check now'.

## Switch to Blue Ocean View

Open blue ocean

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with various links. In the center, there's a table displaying build status for several projects: 'Build\_1Job', 'Destroy\_1Job', 'sample\_1Job', 'some\_1Job', and 'Test\_1Job'. Each row has columns for Name, Last Success, Last Failure, Last Duration, and Fav. Below the table, there's a legend and a 'Build Queue' section. A large red arrow points from the text 'Open blue ocean' to the 'Open Blue Ocean' button at the bottom of the sidebar.

## Switch to classic view

Pipelines | Search pipelines...

NAME	HEALTH	BRANCHES	PW
Build_Job		-	-
Deploy_Job		-	-
sample_test		-	-
sonar_test		-	-
Test_Job		-	-

1.21.0 - Core 2.204.1 - b1ebcb62 - 14th November 2019 10:33 PM

## 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.

Back to Dashboard

Manage Jenkins

Create User

**Create User**

Username:	devops
Password:	*****
Confirm password:	*****
Full name:	devops
E-mail address:	ppreddy2711@gmail.com

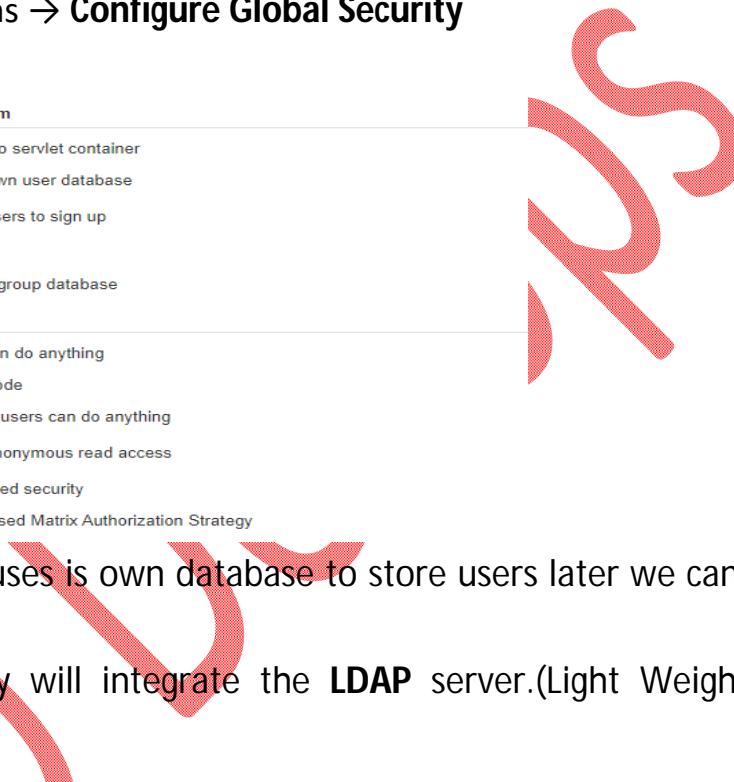
Create User

## Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User ID	Name	
 devops	devops	
 ppddy	ppddy	

- Whenever create the user is having by default admin permissions.  
Jenkins → Manage Jenkins → **Configure Global Security**



The screenshot shows the Jenkins 'Configure Global Security' page. It includes sections for 'Access Control' and 'Security Realm'. In the 'Access Control' section, 'Allow users to sign up' is checked. In the 'Security Realm' section, 'Jenkins' own user database' is selected. In the 'Authorization' section, 'Logged-in users can do anything' is selected. A large red hand-drawn scribble covers the right side of the page.

Access Control
<input type="checkbox"/> Allow users to sign up

Security Realm
<input checked="" type="radio"/> Jenkins' own user database

Authorization
<input checked="" type="radio"/> Logged-in users can do anything

- By default Jenkins is uses its 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**

The screenshot shows the Jenkins Global Security configuration page. At the top, there is a note: "Unix user/group database". Below it, the "Authorization" section has three options: "Anyone can do anything" (radio button), "Legacy mode" (radio button), and "Logged-in users can do anything" (radio button, which is selected). Below these are two checkboxes: "Allow anonymous read access" (unchecked) and "Matrix-based security" (checked). A yellow box highlights the "Matrix-based security" checkbox. Another yellow box highlights the "Project-based Matrix Authorization Strategy" link below it.

**Matrix-based security**

**Project-based Matrix Authorization Strategy**

Matrix-based security

The main table displays permission levels for various users and groups across different Jenkins operations. The columns include Overall, Credentials, Agent, Job, Run, View, SCM, Lockable Resources, and various Jenkins-specific actions like Build, Disconnect, Delete, Create, Connect, Configure, Cancel, and Workspace. The rows list Anonymous Users, Authenticated Users, devops, and ppddy. A yellow box highlights the "Configure" column for devops. Another yellow box highlights the "View" column for ppddy.

User/group	Overall	Credentials	Agent	Job	Run	View	SCM	Lockable Resources	Build	Disconnect	Delete	Create	Connect	Configure	Cancel	Workspace
Anonymous Users	<input type="checkbox"/>															
Authenticated Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>							
devops	<input type="checkbox"/>															
ppddy	<input checked="" type="checkbox"/>															

User/group	Overall	Credentials	Agent	Job	Run	View	SCM	Lockable Resources	Build	Disconnect	Delete	Create	Connect	Configure	Cancel	Workspace
Anonymous Users	<input type="checkbox"/>															
Authenticated Users	<input type="checkbox"/>															
devops	<input checked="" type="checkbox"/>															
ppddy	<input checked="" type="checkbox"/>															

## Project-based Matrix Authorization Strategy

We can provide the job level permission to the users.

Project-based Matrix Authorization Strategy

	Overall	Credentials	Agent	Job	Run	View	SCM	Lockable Resources	
User/group	View	Unlock	Reserve						
	Delete	Read	Tag						
	More	Discover	Create	Configure	Create	Configure	Update	Replay	
	Delete	Discover	Create	Configure	Cancel	Configure	Update	Replay	
Anonymous Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
devops	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>					

Add user or group...

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



General    Source Code Management    Build Triggers    Build Environment    Build    Post-build Actions

[Plain text] [Preview]

Enable project-based security

Inheritance Strategy    Do not inherit permission grants from other ACLs

This object will not inherit the [global security settings](#), or any permissions from its ancestors. Only permissions explicitly enabled here will be granted. To ensure that users are not inadvertently locked out from Jenkins, an exception is made for the Overall/Administer permission: Administrators of Jenkins will still have access to this object even if not explicitly granted here.

	Credentials	Job	Run	SCM	Tag	Update	Replay	Delete	Workspace	Move	Discover	Configure	Cancel	Build	View	Update	ManageDomains	Create
User/group																		
Anonymous Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
devops	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>														

Add user or group...



## Jenkins authentication and authorization

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

Updates    Available    Installed    Advanced

Install ↓

**Role-based Authorization Strategy**

Enables user authorization using a Role-Based Authorization Strategy.

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

**Configure Global Security**

**Enable security**

Disable remember me

**Access Control**

Delegate to servlet container  
 Jenkins' own user database

- Allow users to sign up
- LDAP
- Unix user/group database

**Authorization**

Anyone can do anything  
 Legacy mode  
 Logged-in users can do anything  
 Matrix-based security  
 Project-based Matrix Authorization Strategy

**Role-Based Strategy**

Markini Formatter

### 4. Manage Jenkins → Manage and Assign Roles



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

Global roles		Overall	Credentials	Agent	Job	Run	View	SCM	Lockable Resources																			
Role	Employee	Administrator	Read	Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag	Reserve	Unlock	View
Employee	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
admin	<input checked="" type="checkbox"/>																											

#### b. Create and add the Item roles Jobs\_Development and Jobs\_Testing and pattern as Dev.\* and Test.\*

Item roles		Credentials	Job	Run	SCM	Lockable Resources																				
Role	Pattern	Create	Delete	ManageDomains	Update	View	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag	Reserve	Unlock	View
Jobs_Devel	"Dev.*"	<input checked="" type="checkbox"/>																								
Jobs_Test	"Test.*"	<input checked="" type="checkbox"/>																								

#### c. Assign the roles to our users.

Manage Jenkins → Manage and Assign Roles → Assign Roles

**Assign Roles**

**Global roles**

User/group	Employee	admin	
Pushpanath	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add  Add

**Item roles**

User/group	Jobs_Development	Jobs_Testing	
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add  Add

**Assign Roles**

**Global roles**

User/group	Employee	admin	
Developer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Tester	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Pushpanath	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add  Add

**Item roles**

User/group	Jobs_Development	Jobs_Testing	
Developer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Tester	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add  Add

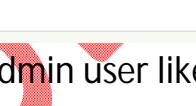
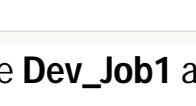
**Node roles**

User/group	
Anonymous	<input type="checkbox"/>

User/group to add  Add

**Save** **Apply**

## 5. Create the jobs as admin user like **Dev\_Job1** and **Test\_Job**

[add description](#)

All	W	Name ↓	Last Success	Last Failure	Last Duration	
		Dev_Job1	N/A	N/A	N/A	
		Test Job	N/A	N/A	N/A	

Icon: [S](#) [M](#) [L](#)

[Legend](#)  [Atom feed for all](#)  [Atom feed for failures](#)  [Atom feed for just latest builds](#)

## 6. Login to Dev and Test users

search [Developer](#) | [log out](#)

[ENABLE AUTO REFRESH](#)

[add description](#)

All	W	Name ↓	Last Success	Last Failure	Last Duration
		Dev_Job1	N/A	N/A	N/A

Icon: S M L

[Legend](#) [Atom feed for all](#) [Atom feed for failures](#) [Atom feed for just latest builds](#)

All	W	Name ↓	Last Success	Last Failure	Last Duration
		Test_Job	N/A	N/A	N/A

Icon: S M L

[Legend](#) [Atom feed for all](#) [Atom feed for failures](#) [Atom feed for just latest builds](#)

## 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.

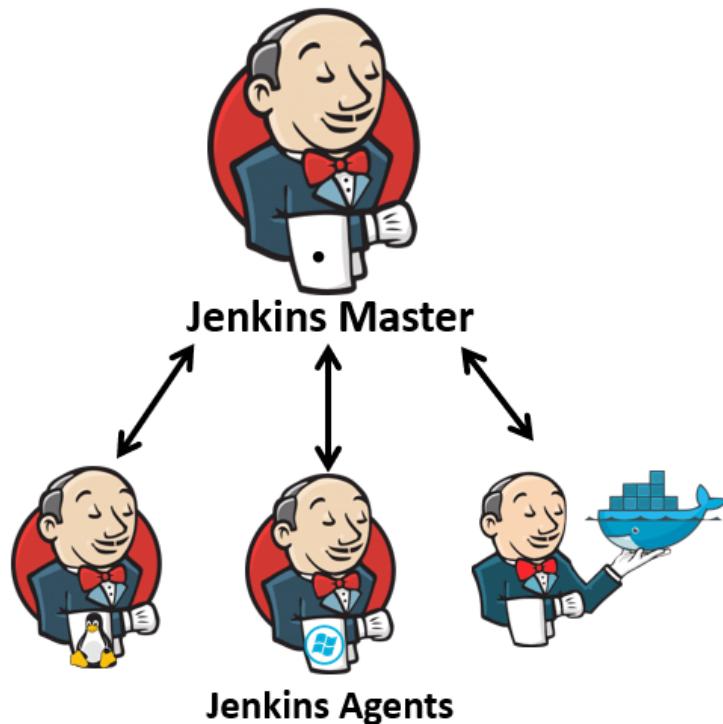
We use this for the following use cases.

- 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:q2nvqHz0pJns0ogJzehSxFQmhLyPuskYQ5/1FV5G9E jenkins@ip-172-31-22-23
The key's randomart image is:
+-- [RSA 3072] --+
|+.. E.oo |
|.* . . |
|+ o . + |
|... . . |
|o.+ . S |
|=o =. . |
|.=+=.* *|
|=*= 0+. |
| ooB=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 AAAAABAAQABAAAQc056fWNoDSoIqbTpuloKfwu0cHEcg5DIMBj7rClvwjd0muK0KbaDK5Uf12fpagvdIbBio+dcp0MMKU3/NtQ
6ULv6652zKzBf2xZynJZH48R+42LhUW10u9WM9M08o8+SzaIxzn161y8ZZqwpr0m0XYS72b4JIoVCu+PMhjMDa0epQDLtGdfAz6Mt17r9B3bBCJz1wPQwo2
PmwzID88h80M0fYgZ8UfYjg+K3Td6ADX6eTa0INH9CV/T7y9JUtgk3K2JPuZXomNRMnUBPCEQ31ZFqVgPgpCRgasVECsKs11t7509T510f/zd0Gg1jlHRvhJ
210opwrYmpdei/AThK7kH7Xw0UF4L4B9xpn429y2qT5vRdpLrzb1kXdbax7/07V7fNMu+FVDNh1nbbF/LwW8NeIZBk/kW0kZnhseQTS+DPwFoc1GZ3bcQBIX4i
V1Kz0BzT0iDejnLE2Edgvpm+6NyvKEoQaB83qtX0Bi3txNAI8f+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-----
b3B1vnzaC1rZKxtdJEAAAAABG5vbmluAAAAAEbm9uZQAAAAAAAAAAABAlwAAAAdzc2gtcn
NhAAAAAAEAAAQAAAYEAvuehcDaA0qCkm06bpAcN8LKHxH100yDgye6wpb8HStPrpCgkJG2g
yuVh0dn6YGr3SGw7qPnXUDCNvzb0L0C7+uuds7ssw9swcpjwR4uPeFuN14VftdlvVj
PTEPKPPks21M55etcvGwasMka9Nj12Eu9m+CskF0rvjzIyzA2jnguUAjRnxM+1Lze6/Q
d2w01c9cDOMKNNj5MyA/PIfkDDnshfmFH216vit03egA1+nk2idR/0lf06+8vSVLUpNy
tiT7mV6Jj1TWhEN4mRaIajxqK0yGrFRARjEtdbe+tvU+ZHZf3thoNvSRb45tdtkK
ck2JqXXowv4Su5B+11qFbeCaAfcaz-Nvt tqk+b0xaS0W9ZF3Wm1+901e3zTlvhVQzYd
Z22xfyFvDx1GOZP5fJGZ4bHkE0vgz8BaHNrm2d3EASF+IldsmawUzog3o5yxhHyL6z
vuJLylhAEGGgnf6-V9AYt7cTQCPH/p+gPB5o54TSAAFKL8Qf4/EBeHAAAB3NzaC1yc2
EAAAGBALn0X2dNkgIptOm6Wgp/C5BwrcRydkMgxsnusKw/B0k6kaQoQjCRTMrLR/X+mbq
96hsGKj51ylAw2qTfB21D0Qu/rnb0Dr/bFnKclkeJx71juFBxS71Yz0xOdyz5LNo
jHoExrLxLmDmvTSDzdhLzvgkjhUK748yGmWn0561AMu0z18DPoy2Xuv0HdsElnPxA9D
CjY-bDMgPzytgw5/ITnxR910r4dn3oANfpSno4gf03X90vvL0151aTcrYk+5leY1E
ydQ8IRDejkwpw0BaKjGBqxUQkyjXw3Vkt1PmWb9n904adW0Uldg+EnuXsInCtial16LB8
0EruqftdahQxvggh3GmfjB3LapPm9F2ktFlwRwd1ptfv/TxTxb0y74VM2#WdtsX8vbw1
4hkNtGT+rBsNmegX5BNL4M/AVhzlZndtAEhfj1UpmgfPlm6IN60csTVr2c+mb7l18o0Q0hB
ohZeq1fQGLEsE0Ax/6fdeweau+QAAABAEAAAGAYWZ71ce7btu4y+sHqvgk9ufyW
x4dCJfEmkvh9vXnyLw4CYZ4oPQzCtzQxxW8V+zCrzs9UATAP/t188C27Ugap9L6bnB
6p9qn5LrgqQAnptBerury1Ybj11bM6lEWZceYztT6Rw0Bw9Cuf8gV2ijHxGr6Dxfiz
x1p07/w73cqzNsR01+F0ihzb2rLc9w5X-PdnhxW+hiKGVLRLshEP9ear10bMphbfmsSfk
z8n0baRC32E34FF1afzw/+ymH+HFnq7VAPSFRgsXyLtvSiegaADBA0paV1Ecnn99PpkB
2gdWz/mJPoVRll/gf2/L87Yhi3Bn5Yhg3x85eZc87Gh1yZ27j1r/ml15lyhukwm5ipf
aoqfEy+c0/ei0kyM7Xwp5qa0o0rlrkFzy1lw6gAHbc4Lc+b3B7CpkNxZCJ3+vh11ve/gPk
1N4CCR21l3jC0t0REsmzt6RSLL2ikGcc6mK0QxLtbJbs9dCu2l0wK9wPQXy7chZM1lign
mrIkVj3yAzJm0ndoJpXTdT1ewyBm0wAAAMEAyxRoZBTjKMTEff63Y1pNb7FkP7pxe4
Y3Nr6y8CkuQFFJB/RbdwMUGQbqdGe497601tffTrxevffjg9jsE5fjvzbMsXkqg/dkjo
1dqh6cXZn72U3PEubkYRkyt9Vt6/H2F651Vj-xgN3hKtBy+Rb6u2E1duzuYT8
2eQhMqtn1zVOVDiu1oTeQgbMmeeCpZLqID51c6aDm3m9FtqgOD81S3ohiTH/zY7Vxh
a0J9hJYz1Qc2dAAA2plbmtpbnNaAXAtMtcyLTMxLTIzA9D
-----END OPENSSH PRIVATE KEY-----
jenkins@ip-172-31-22-23:~/ssh$
```

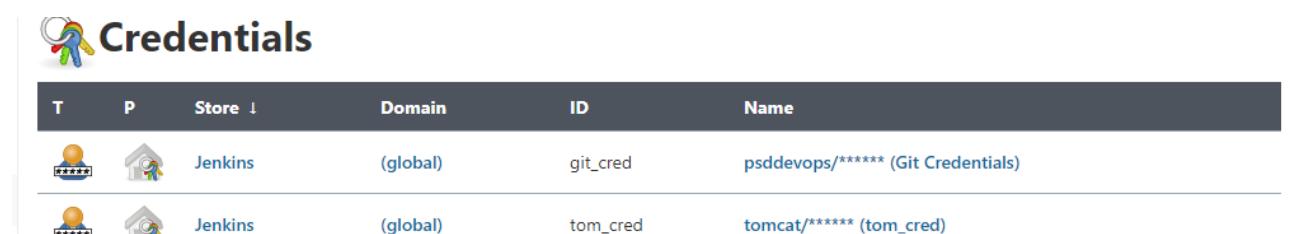
## *2. Setup Credentials on Jenkins*

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'.

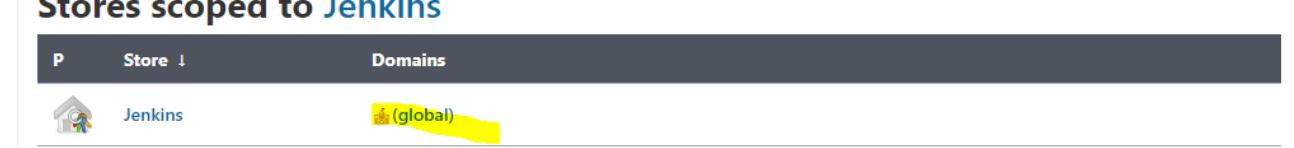


The screenshot shows the Jenkins 'Global credentials (unrestricted)' page. It lists two entries:

ID	Name	Kind	Description
git_cred	psddevops/******** (Git Credentials)	Username with password	Git Credentials
tom_cred	tomcat/******** (tom_cred)	Username with password	tom_cred

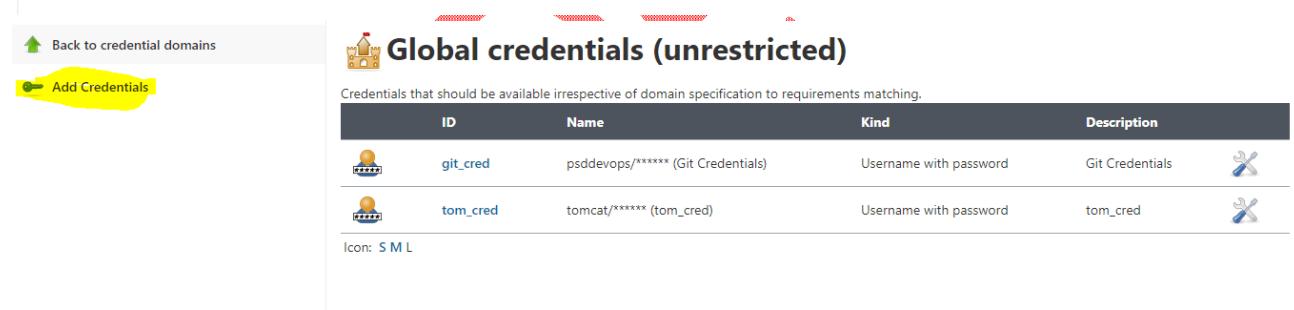
Below the table, there are icons for S, M, and L.

**Stores scoped to Jenkins**



The screenshot shows the 'Stores scoped to Jenkins' page. It lists one entry under 'Domains': Jenkins (global).

**Add Credentials**



The screenshot shows the 'Add Credentials' page. The 'Add Credentials' button is highlighted with a yellow box.

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.

Description	Slave-1 node agent server	<a href="#">?</a>
# of executors	1	<a href="#">?</a>
Remote root directory	/home/jenkins	<a href="#">?</a>
Labels	myslavegroup	<a href="#">?</a>
Usage	Use this node as much as possible	<a href="#">?</a>
Launch method	Launch agents via SSH	<a href="#">?</a>
Host	18.221.239.196	<a href="#">?</a>
Credentials	jenkins (jenkins_cred) <a href="#">Add</a>	<a href="#">?</a>
Host Key Verification Strategy	Known hosts file Verification Strategy	<a href="#">?</a>
<a href="#">Advanced...</a>		
Availability	Keep this agent online as much as possible	<a href="#">?</a>

**Node Properties**

- Disable deferred wipeout on this node [?](#)
- Environment variables [?](#)
- Tool Locations [?](#)

[Save](#)

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.



Jenkins > Nodes

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	master	Linux (amd64)	In sync	4.94 GB	0 B	4.94 GB	0ms
2	Slave-1	Linux (amd64)	In sync	5.33 GB	0 B	5.33 GB	29ms
Data obtained 0.63 sec 0.63 sec 0.63 sec 0.63 sec 0.63 sec 0.63 sec							
<a href="#">Refresh status</a>							

Creating sample job in slave

Jenkins → mytest\_slave

**General** Source Code Management Build Triggers Build Environment Build Post-build Actions

- Discard old builds
- GitHub project
- Delivery Pipeline configuration
- This build requires lockable resources
- This project is parameterized
- Throttle builds
- Disable this project
- Execute concurrent builds if necessary
- Restrict where this project can be run

Label Expression **mystagegroup**

Label mystagegroup matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

Advanced...

Jenkins → mytest\_slave → #1

**Console Output**

Started by user psddevops  
Running as SYSTEM  
Building remotely on **Slave-1 (mystagegroup)** in workspace /home/jenkins/workspace/mytest\_slave  
[mytest\_slave] \$ /bin/sh -xe /tmp/jenkins3171782125918018605.sh  
+ echo This is slave test  
This is slave test  
Finished: SUCCESS

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

The screenshot shows the Jenkins Global credentials (unrestricted) configuration page. A new credential is being created with the following details:

- Kind:** SSH Username with private key
- Scope:** Global (Jenkins, nodes, items, all child items, etc.)
- ID:** jenkins\_cred
- Description:** jenkins\_cred
- Username:** jenkins
- Private Key:** Enter directly (radio button selected). The value is a long string of characters representing an SSH private key, starting with "-----BEGIN OPENSSH PRIVATE KEY-----".
- Passphrase:** (empty field)

A large red 'X' is drawn over the entire form area.

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

The screenshot shows the Jenkins Global credentials (unrestricted) list page. The table displays three credentials:

ID	Name	Kind	Description
git_cred	psddevops/***** (Git Credentials)	Username with password	Git Credentials
tom_cred	tomcat/***** (tom_cred)	Username with password	tom_cred
jenkins_cred	jenkins (jenkins_cred)	SSH Username with private key	jenkins_cred

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.

The screenshot shows the Jenkins Nodes page. At the top, there is a table with one row for the 'master' node. The columns include S, Name (master), Architecture (Linux (amd64)), Clock Difference (In sync), Free Disk Space (5.76 GB), Free Swap Space (0 B), Free Temp Space (5.76 GB), Response Time (0ms), and a gear icon for configuration. Below the table is a 'Refresh status' button. To the left of the table, there are links for Back to Dashboard, Manage Jenkins, New Node (highlighted in yellow), and Configure.

The screenshot shows the 'New Node' configuration dialog. The 'Node name' field is set to 'Slave-1'. The 'Permanent Agent' radio button is selected. A tooltip explains: 'Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.' Below the configuration fields is an 'OK' button.

The screenshot shows the 'New Node' configuration dialog for 'Build Slave'. The 'Name' field is 'Build-Slave', 'Description' is 'This slave build the jobs', 'Labels' are 'My\_Slave\_Group', and 'Usage' is 'Only build jobs with label expressions matching this node'. Under 'Launch method', 'Launch agent agents via SSH' is selected, with 'Host' set to '172.31.20.63', 'Credentials' set to 'ubuntu (slave\_cred)', and 'Host Key Verification Strategy' set to 'Non verifying Verification Strategy'. The 'Availability' section has 'Keep this agent online as much as possible' checked. At the bottom are 'Advanced...' and 'Save' buttons.

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Build-Slave	Linux (amd64)	In sync	5.29 GB	0 B	5.29 GB	16ms
	master	Linux (amd64)	In sync	5.76 GB	0 B	5.76 GB	0ms
Data obtained	21 sec	21 sec	21 sec	21 sec	21 sec	21 sec	21 sec

[Refresh status](#)

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

Restrict where this project can be run

Label Expression

[Label My\\_Slave\\_Group](#) is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

[Advanced...](#)

## Jenkins Build Status



### Disable Build:

A disabled Build will not be executed until you enable it again. This option often comes in handy 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

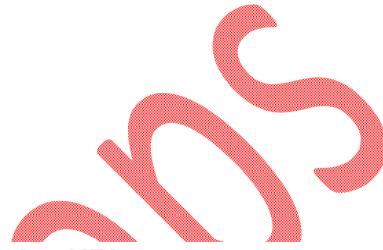
Figure b: Weather reports

#### Jenkins Build Status Icon Colours

	BLUE: Success
	GREY: Disabled, Aborted and Pending
	YELLOW: Unstable
	RED: Failed

#### Jenkins Job Weather Status Icon Colours


**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

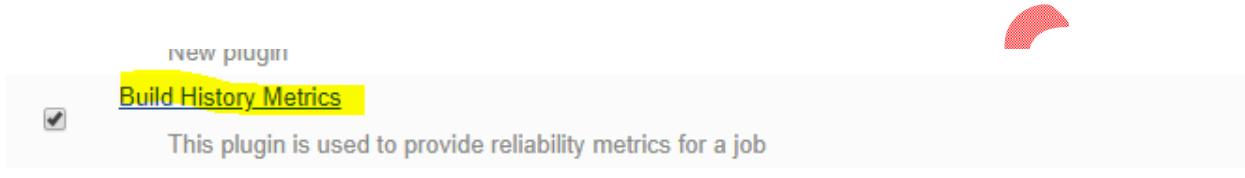
## **Jenkins - Metrics and Trends**

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:

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

Install **Build History Metrics** plugin and select **without restart** button.

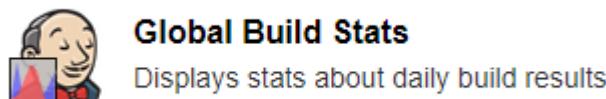


Select any job and go to that job page. When you go to that job page, you will see a table with the calculated metrics. Metrics are displayed for the last 7 days, last 30 days and all time.

## Trends in Jenkins

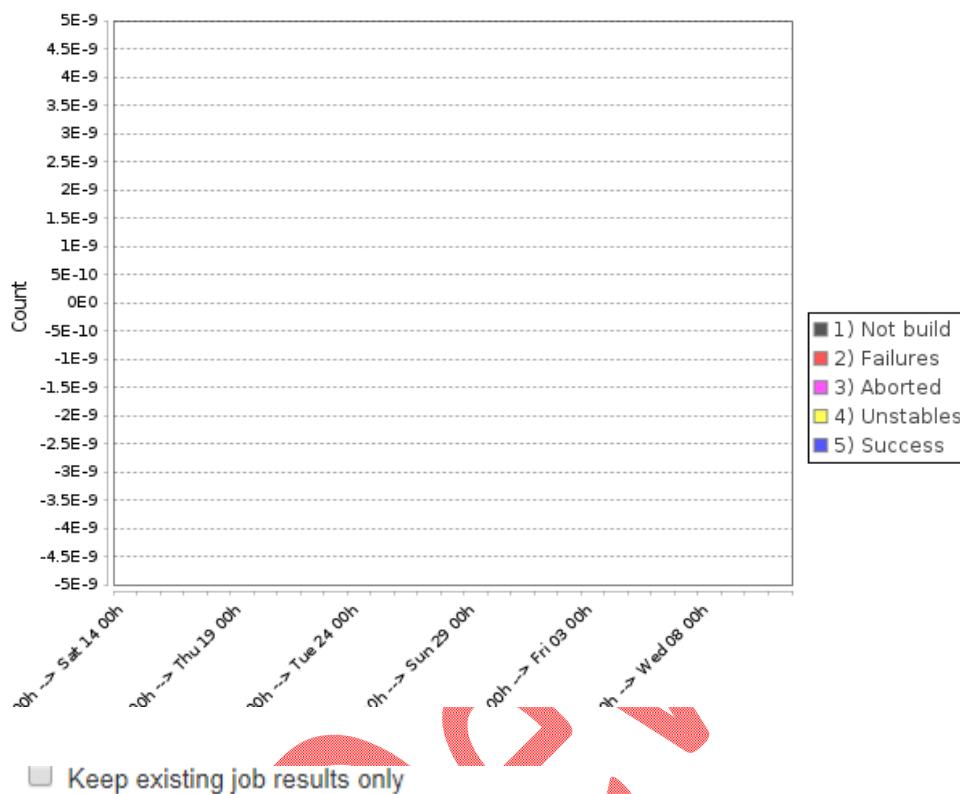
To see the overall Trends, Jenkins provides many plugins like **Build-metrics plugin**. This plugin is used to gather the information from within the builds and Jenkins and display them in a graphical format

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**".

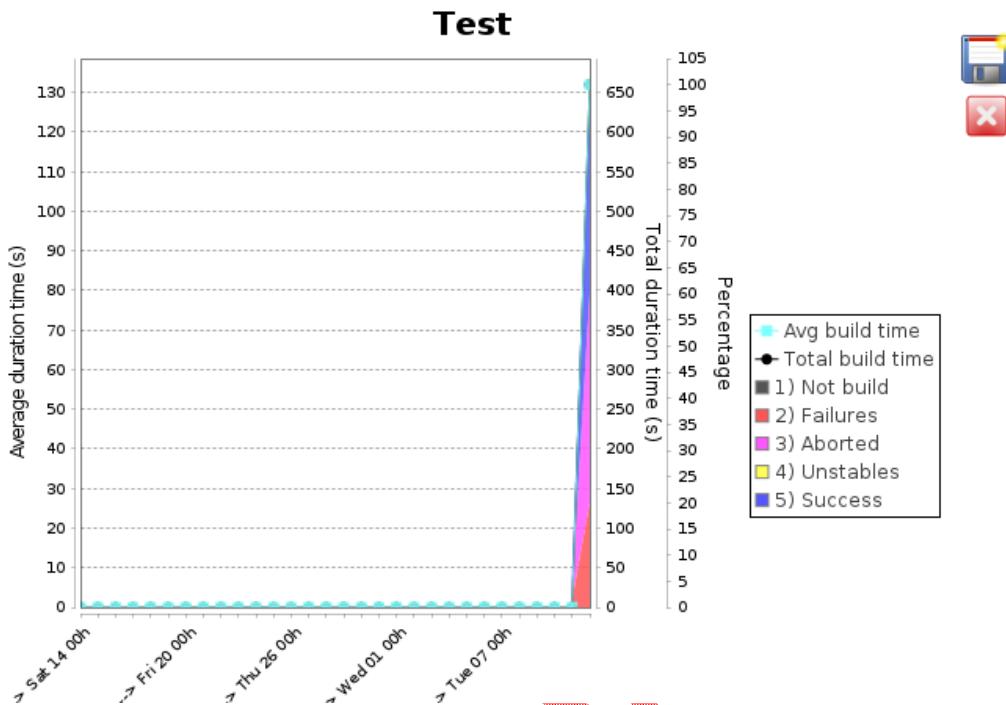
## Test



### Data Initialization

Click button below to initialize build statistics  
Job results read will be merged with already reco.  
**Data successfully initialized !** [Refresh page](#)

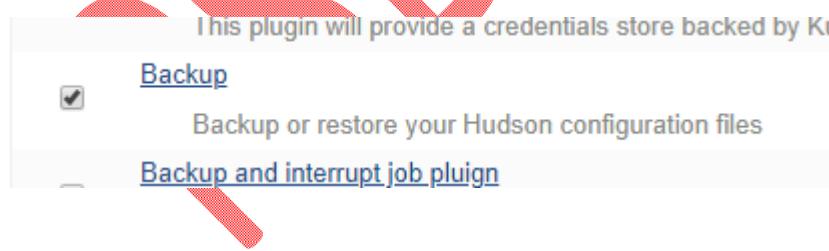
**Initialize stats**



## 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.

## Backup manager

Backup or Restore Jenkins configuration files

# Backup manager

[Setup](#)[Backup Hudson configuration](#)[Restore Hudson configuration](#)

### Backup configuration

Hudson root directory `/var/lib/jenkins`

Backup directory

Format

File name template

Custom exclusions

Verbose mode

Configuration files (.xml) only

No shutdown

### Backup content

Backup job workspace

Backup builds history

Backup maven artifacts archives

Backup fingerprints

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



## Backup manager



[Setup](#)



[Backup Hudson configuration](#)



[Restore Hudson configuration](#)

### Best practices in Jenkins

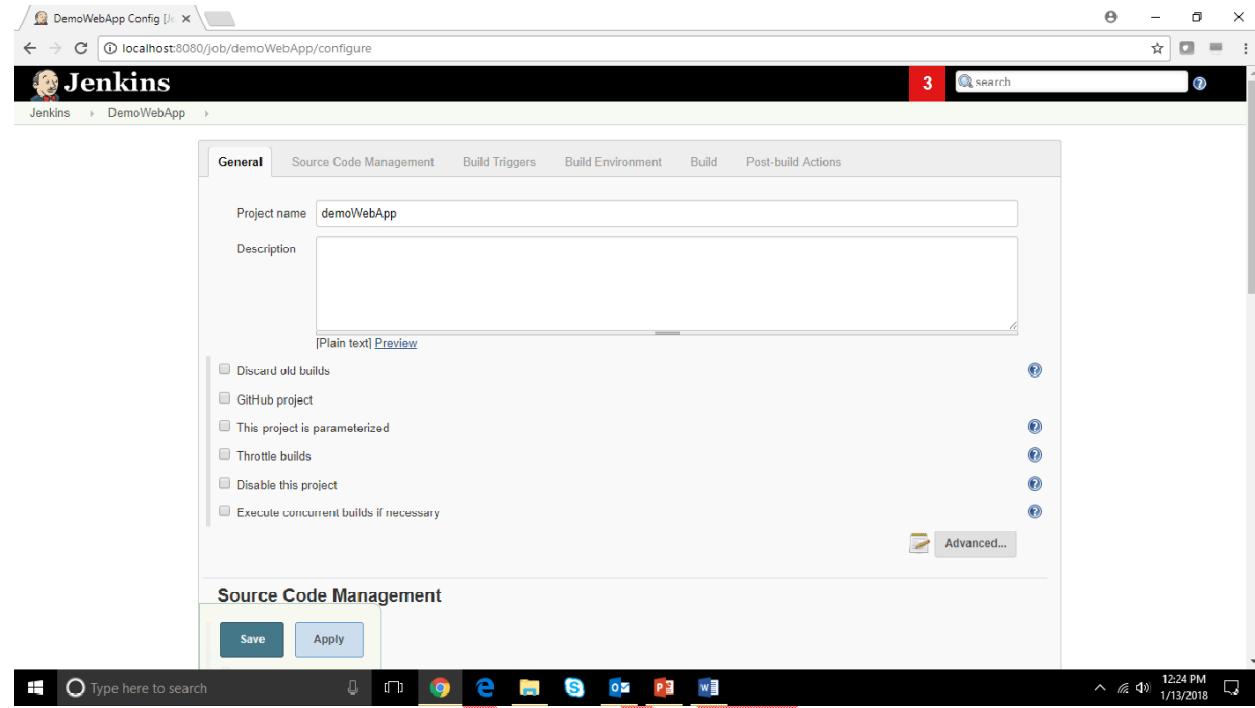
- Always secure Jenkins
- In larger systems don't build the master.(All jobs are running in master it will become slow)
- Backup Jenkins home regularly (If we delete some jobs again it will if we restart)
- Integrate the Jira (or) Bugzilla tools.
- Setup Jenkins home having more disk space.
- Archive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

### Jenkins Terminologies

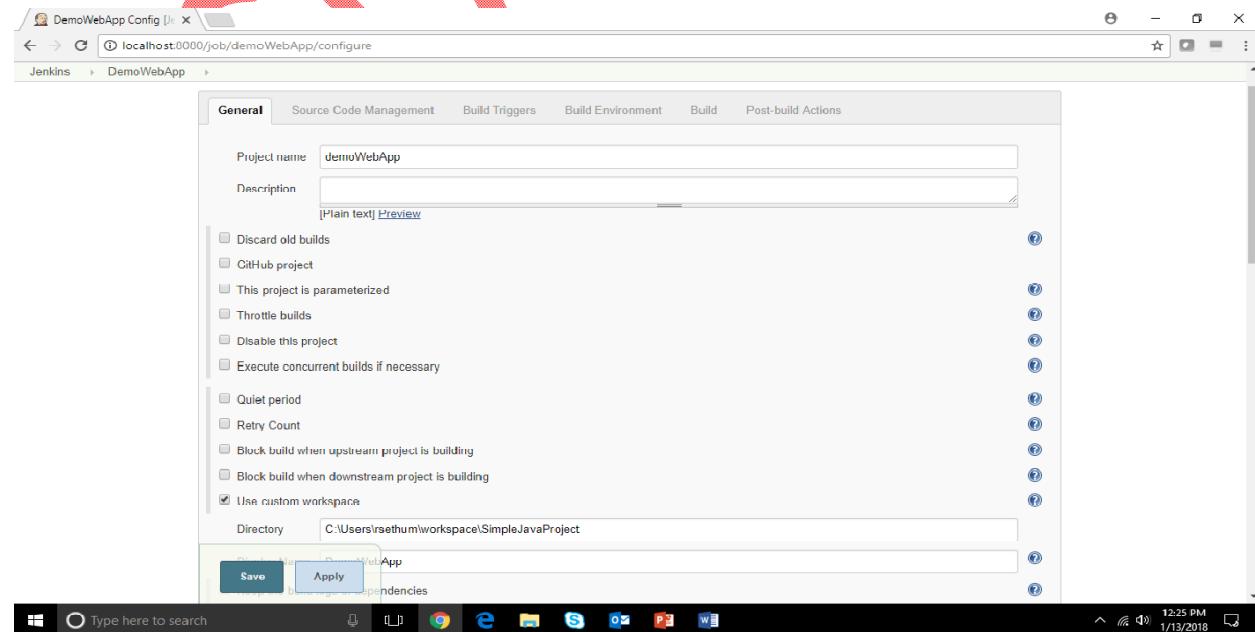
Term used in Jenkins	Description
Upstream project	A project can have one or several upstream projects, which means that current project is scheduled depending if the upstream project is built successfully or not. If the upstream project is successful the current project is added to the build queue. If 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 <i>stable</i> or <i>unstable</i> result depending on the result of its processing. For example, if a JUnit test fails, then the whole JUnit publisher may report <i>unstable</i> .
Completed Build	A build is completed, if it was started and finished with any result, including failed builds.

## Selenium Integration

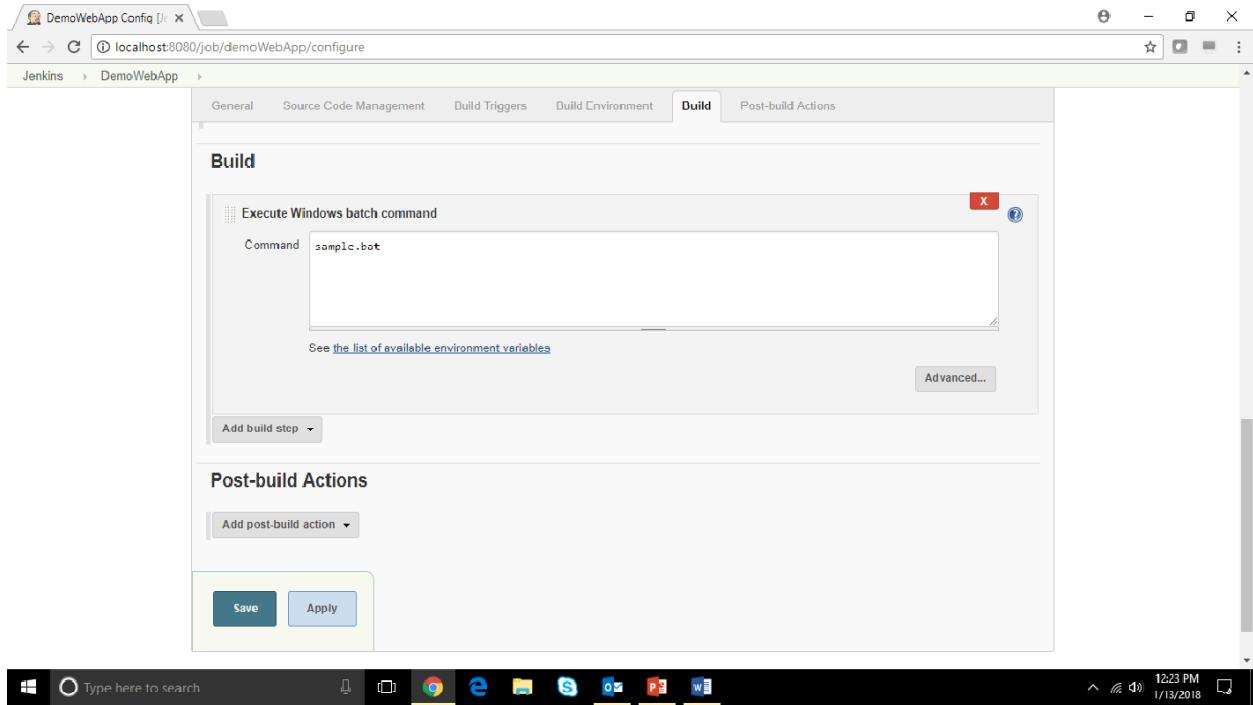
Click on advanced



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

The screenshot shows the Jenkins Plugin Manager interface. The 'Available' tab is selected, and a search filter for 'slack' is applied. A yellow box highlights the 'Slack Notification' plugin, which is described as a Slack notifier for publishing build status. Other available plugins listed include 'Slack Upload', 'cucumber-slack-notifier', 'Build Notifications', and 'Global Slack Notifier'. At the bottom of the page are buttons for 'Install without restart', 'Download now and install after restart', and 'Check now'. The Jenkins header shows the user 'raja'.

## Add our channel to jenkins CI

Manage apps...

X esc

### Add apps to # training

View App Directory

Q jel

From the App Directory

Trello Collaborate on Trello projects without leaving Slack. Install

Jira Cloud Easily connect Jira Cloud projects to your Slack channels Install

Jenkins CI An open source continuous integration server. Install

Browse apps > Jenkins CI > New configuration

**Jenkins CI**  
An open source continuous integration server.

Jenkins CI is a customizable continuous integration server with over 600 plugins, allowing you to configure it to meet your needs.

This integration will post build notifications to a channel in Slack.

**Post to Channel**  
Start by choosing a channel where Jenkins notifications will be posted.  
# training or create a new channel

Add Jenkins CI integration

Copy the token and workspace and configure

Test configuration by sending test e-mail

Slack

Workspace: psddevops

Credential: Slack Add

Default channel / member id: training

Custom slack app bot user:

Success

Advanced...

Test Connection

SCM Polling

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
```

PSD Devops

## 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

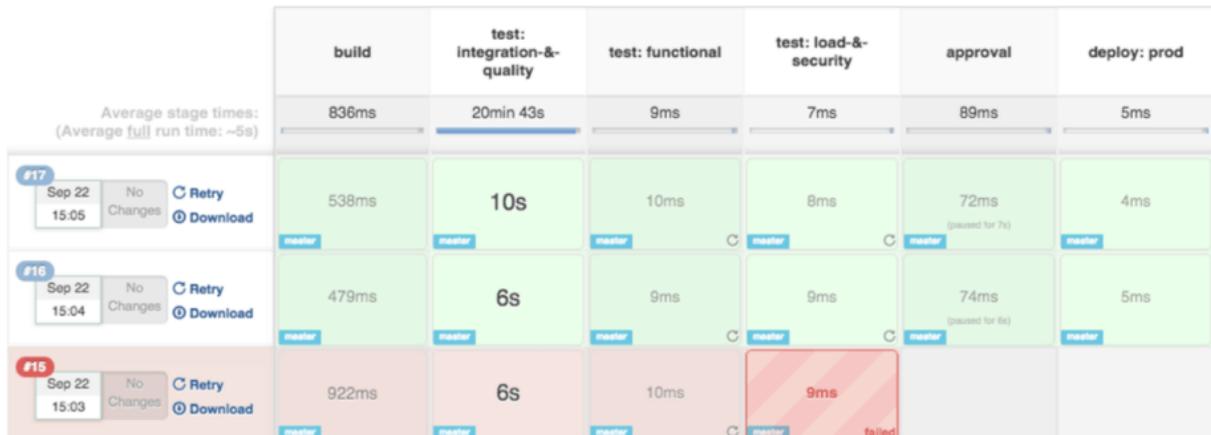
This section allows generation of different stages on your pipeline that will be visualized as different segments when the job is run.

## 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"
                    }
                }
            }
        }
    }
}
```

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 {
    agent any
    stages {
        stage('Sample') {
            steps {
                echo "Scripted block"
            }
        }
    }
}
```

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:

```

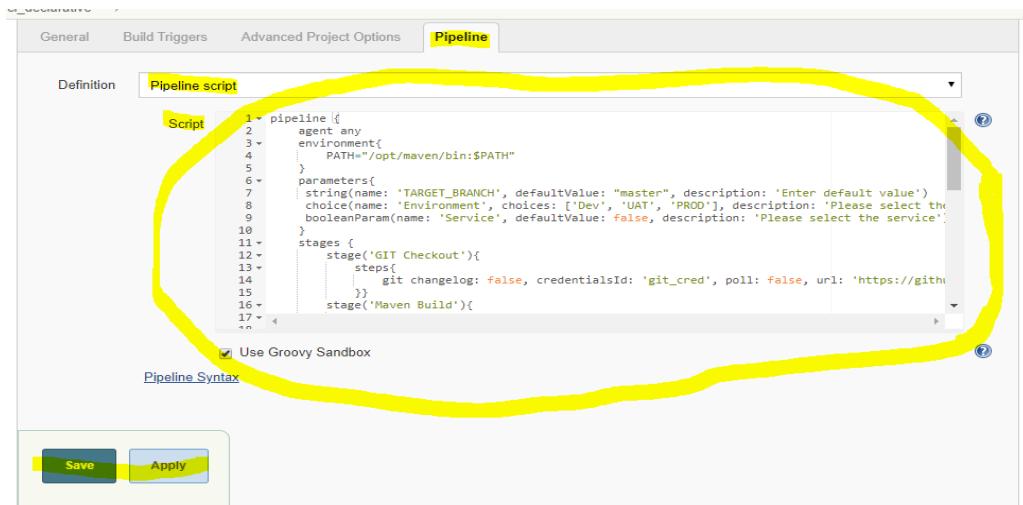
pipeline {
agentany
stages {
stage('Deploy stage') {
when {
branch'master'
}
steps {
echo'Deploy master to stage'
...
}
}
}
}
```

How many ways we can write Jenkins pipelines ?

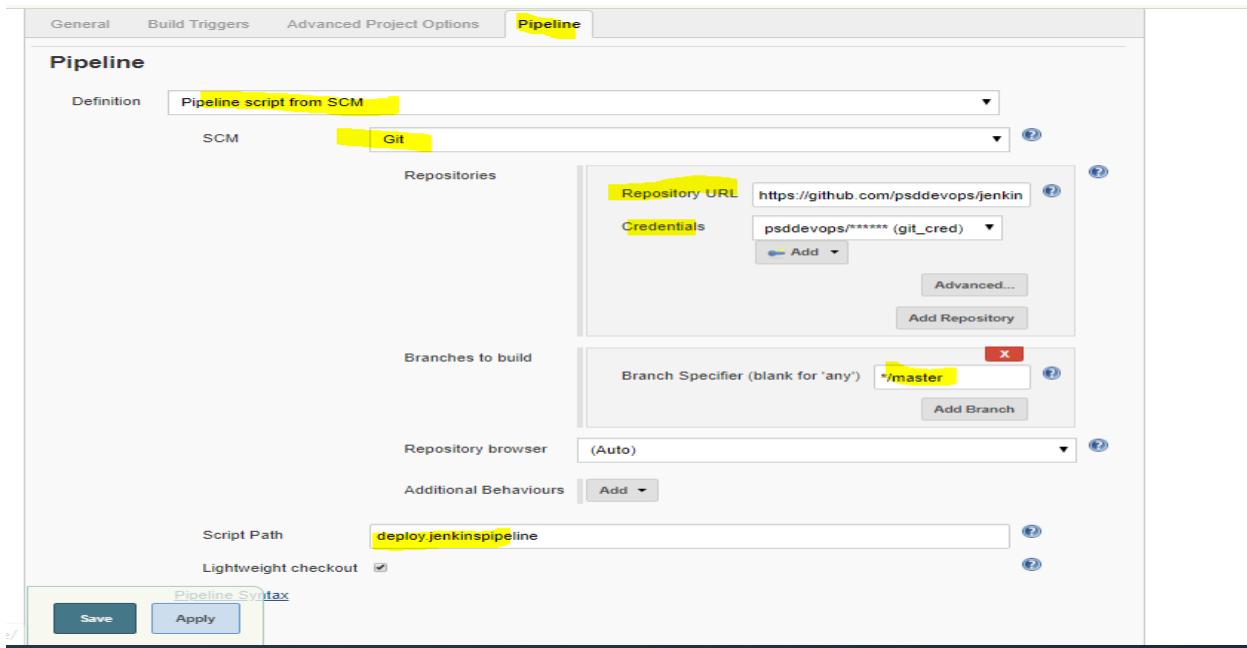
2 – Ways

1. In-line
2. Pull from SCM

## In-line



## Pull from SCM



Thanks .....

All The Best .....

X'