JENKINS work book

2018

Installation and Configuration Basics

Windows

Vibranarayanan

Jenkin GitHub integration Self learning

[Download Information 6](#_Toc513668734)

[Jenkins Installation 6](#_Toc513668735)

[Post Installation steps 7](#_Toc513668736)

[Password file Location 8](#_Toc513668737)

[Initial Password submission 8](#_Toc513668738)

[PlugIn selection 8](#_Toc513668739)

[Plug-in installation screen 9](#_Toc513668740)

[Plug-in Installation Final screen 9](#_Toc513668741)

[Jenkins login screen 10](#_Toc513668742)

[User and access Management 10](#_Toc513668743)

[How to create a new user 10](#_Toc513668744)

[Checking new user login 10](#_Toc513668745)

[Setting up access permissions/Roles. 11](#_Toc513668746)

[Role base plug-in installation 11](#_Toc513668747)

[Configure Global security 11](#_Toc513668748)

[Check user permission after configuration update 12](#_Toc513668749)

[Assign Roles 12](#_Toc513668750)

[Role Types 12](#_Toc513668751)

[Role Creation 13](#_Toc513668752)

[Assign Roles 13](#_Toc513668753)

[Check Developer Role 15](#_Toc513668754)

[Check QA Role 15](#_Toc513668755)

[Simple Configurations 15](#_Toc513668756)

[System Message 15](#_Toc513668757)

[System message with HTML 16](#_Toc513668758)

[Number Of Executers: 17](#_Toc513668759)

[Manage Nodes/Labels 17](#_Toc513668760)

[Quiet period 18](#_Toc513668761)

[SCM checkout retry count 18](#_Toc513668762)

[Restrict Project Naming 18](#_Toc513668763)

[Global property 19](#_Toc513668764)

[Jenkins CLI Setup 19](#_Toc513668765)

[Why Jenkins CLI 19](#_Toc513668766)

[Start Jenkins 19](#_Toc513668767)

[Enable Global security 19](#_Toc513668768)

[Install Jenkins CLI 19](#_Toc513668769)

[Starting with Jenkins JOB without SCM Tool 20](#_Toc513668770)

[Creating a Job 20](#_Toc513668771)

[Update to General Information tab 20](#_Toc513668772)

[Update to Source code management tab 20](#_Toc513668773)

[Update to Build Tab 21](#_Toc513668774)

[Update to Post-build actions tab 21](#_Toc513668775)

[Executing builds with newly created Job 21](#_Toc513668776)

[Before executing Build 21](#_Toc513668777)

[After Executing Build 22](#_Toc513668778)

[Console LOG of successful Build 22](#_Toc513668779)

[Triggering builds remotely 22](#_Toc513668780)

[Configuring remote builds 22](#_Toc513668781)

[Build screen after Remote build execution 23](#_Toc513668782)

[Simple Java build without GIT integration 24](#_Toc513668783)

[Create simple Java Program 24](#_Toc513668784)

[Before compilation 24](#_Toc513668785)

[After compilation 24](#_Toc513668786)

[Build integration with Jenkins 24](#_Toc513668787)

[Create simple Java project Job in Jenkins 24](#_Toc513668788)

[Configure Jenkins Build 25](#_Toc513668789)

[Before executing build 25](#_Toc513668790)

[After executing build 25](#_Toc513668791)

[Console output after successful execution 26](#_Toc513668792)

[Jenkins Jobs integration with GIT 26](#_Toc513668793)

[Create Github repo 26](#_Toc513668794)

[GIT initiation in project location 26](#_Toc513668795)

[Check file status 27](#_Toc513668796)

[Adding file into git 27](#_Toc513668797)

[Setting Global property in git 27](#_Toc513668798)

[Committing Changes in git 27](#_Toc513668799)

[Adding remote repo and pushing code into repo 28](#_Toc513668800)

[Verify it in github 28](#_Toc513668801)

[Jenkins Configuration changes. 28](#_Toc513668802)

[Add github repo details 28](#_Toc513668803)

[Setting build triggers 29](#_Toc513668804)

[Build screen before committing change 29](#_Toc513668805)

[Build screen After Commit. 29](#_Toc513668806)

[Jenkins JOBs console output 30](#_Toc513668807)

[How to do Automated Deployment 30](#_Toc513668808)

[Stages in Continues Delivery and Deployment Pipeline 30](#_Toc513668809)

[Build stage 31](#_Toc513668810)

[Deployment Stage 31](#_Toc513668811)

[Test stage 31](#_Toc513668812)

[Release Stage 31](#_Toc513668813)

[Setting up sample Auto deployment 32](#_Toc513668814)

[download sample war file 32](#_Toc513668815)

[Deployment Plug-in installation 32](#_Toc513668816)

[Plug-in installation 32](#_Toc513668817)

[Setting up Jenkins Job for auto deployment. 33](#_Toc513668818)

[Tomcat server before deployment 33](#_Toc513668819)

[Tomcat screen after deployment 34](#_Toc513668820)

[Enabling Basic email Notification 34](#_Toc513668821)

[Notification configuration in Jenkins 34](#_Toc513668822)

[Test Alert verification in Gmail 34](#_Toc513668823)

[trouble shooting with Gmail email configuration 34](#_Toc513668824)

[Setting email notification for unstable builds 35](#_Toc513668825)

[Checking unstable email notification from Jenkins 35](#_Toc513668826)

[Build status 35](#_Toc513668827)

[Job's Console screen 35](#_Toc513668828)

[Email screen 35](#_Toc513668829)

[Jenkins Pipeline 36](#_Toc513668830)

[What is Pipeline 36](#_Toc513668831)

[Continues delivery Pipeline flow 36](#_Toc513668832)

[Upstream Job 36](#_Toc513668833)

[Downstream job 36](#_Toc513668834)

[How to setup Delivery Pipeline in Jenkins 36](#_Toc513668835)

[Create sample jobs 36](#_Toc513668836)

[Create Job 1 - Sample Build Job 36](#_Toc513668837)

[Create Job 2 - Sample Deploy Job 37](#_Toc513668838)

[Create Job 3 - Sample Test Job 37](#_Toc513668839)

[Verify newly created job 37](#_Toc513668840)

[Chain Newly created Jobs 37](#_Toc513668841)

[Configuration Options for Job chaining 37](#_Toc513668842)

[Configuration Update in Job2 (Deploy Job) 38](#_Toc513668843)

[Configuration Update in Job 3 (Test Job) 38](#_Toc513668844)

[Verify chaining 38](#_Toc513668845)

[Before chaining Build status 38](#_Toc513668846)

[After chaining Build status 39](#_Toc513668847)

[Jenkins Pipeline Integration 39](#_Toc513668848)

[Creating Delivery Pipeline view 39](#_Toc513668849)

[Pipeline view 40](#_Toc513668850)

[Pipeline view Configuration Changes 41](#_Toc513668851)

[Pipeline view after Configuration changes 41](#_Toc513668852)

[Pre-commit hook 42](#_Toc513668853)

[Initiate a Project in Git 42](#_Toc513668854)

[Create pre-commit hook 42](#_Toc513668855)

[Create java and text files inside test project 42](#_Toc513668856)

[Commit with Java file 43](#_Toc513668857)

[Commit without Java file 43](#_Toc513668858)

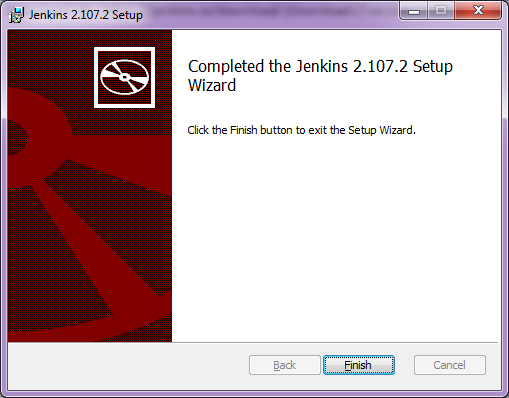
|  |  |
| --- | --- |
| **Date** | **Topics Covered** |
| 5/4/2018 | Download information to Simple Configurations (Page 1 to 17) |
| 5/07/2018 | Jenkins CLI support to Jenkins Git integrations (Page 17 to 28) |
| 05/08/2018 | * Automated deployment * Enabling automated deployment in Jenkins * Deploying EAR file to tomcat container * Configure email notification globally * setting up email notification for unstable builds   Page(28 to 33) |
| 05/09/2018 | * Delivery Pipeline * Job chaining * Jenkins Delivery Pipeline plug-in installation * sample deliver Pipeline set-up and basic configuration changes. * Pre-commit hook POC   (Page 35 to 40) |

# Download Information

Download URL : https://jenkins.io/download/ (Downloaded LT version. 2.107.2)

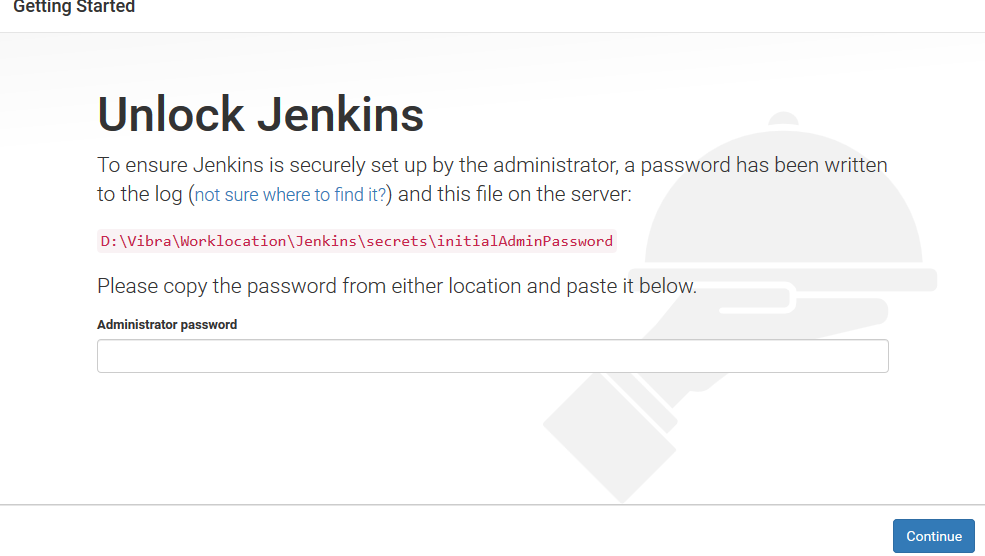
# Jenkins Installation

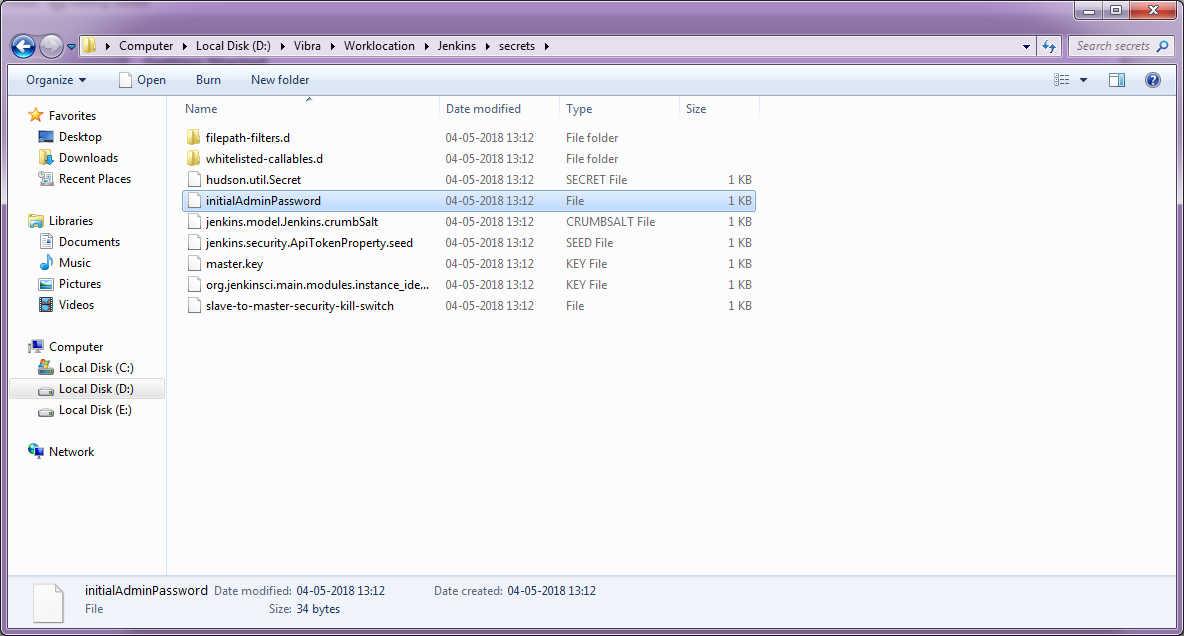
Run exe and finish installation

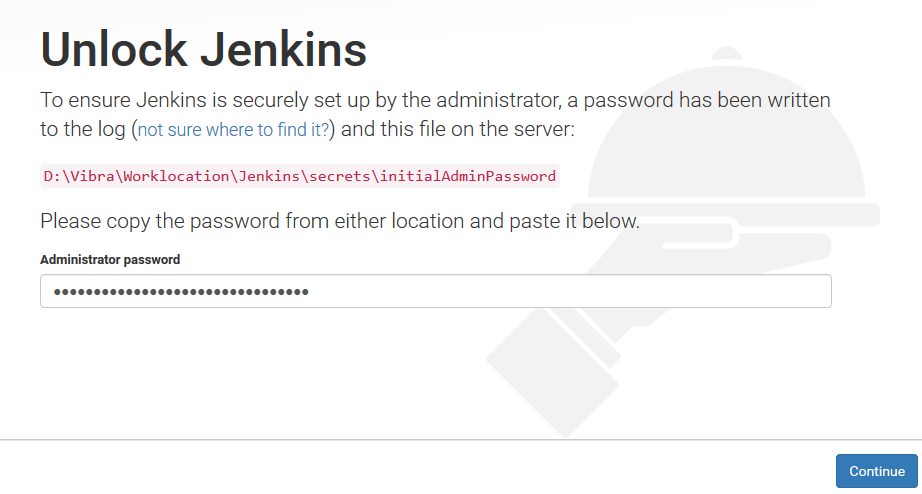


# Post Installation steps

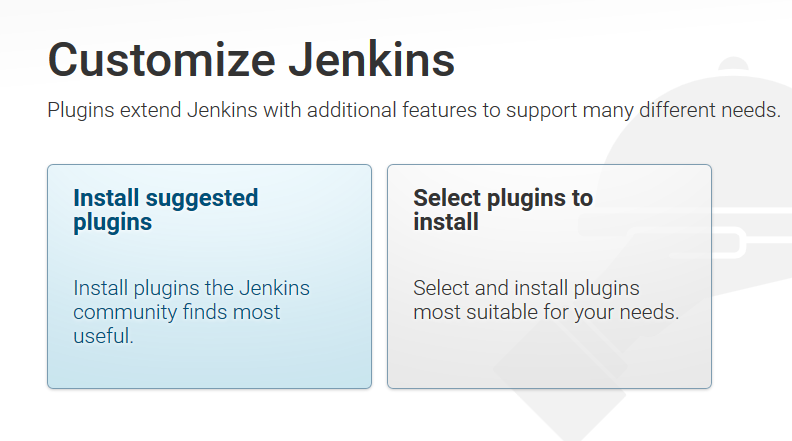
After installation, access Jenkins using Browser. (http://localhost:8080) Enter Admin password to unlock Jenkins. Password location mention in below location under secrets folder. copy past below URL or copy PWD from below location and past it in text box and press continue button.

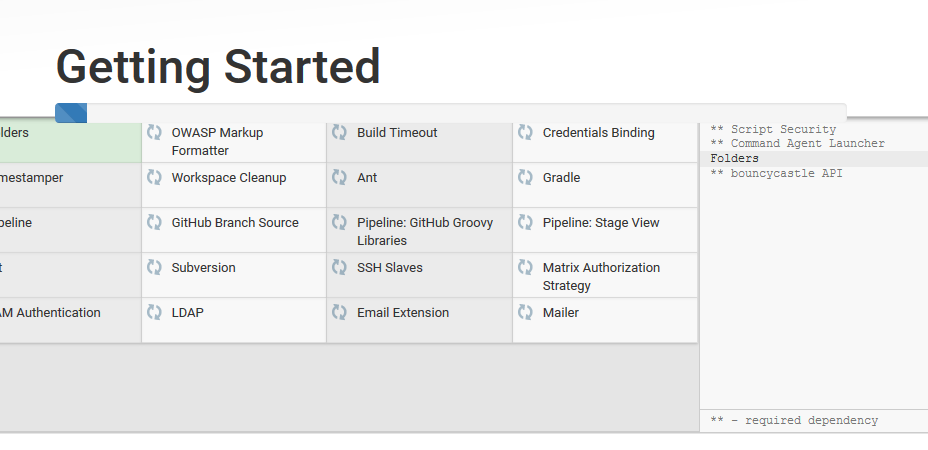


Password file Location

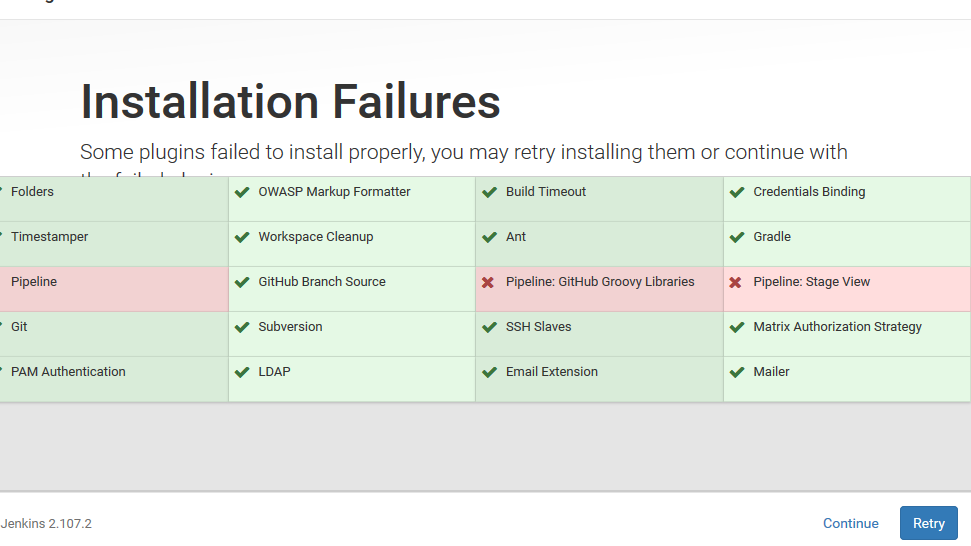
Initial Password submission 

# PlugIn selection

select plugin options based on you need. i have selected suggested plugin.

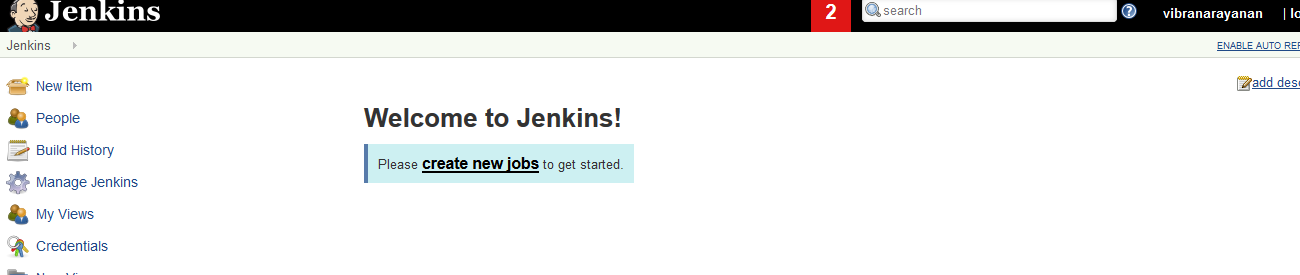
Plug-in installation screen

#### Plug-in Installation Final screen



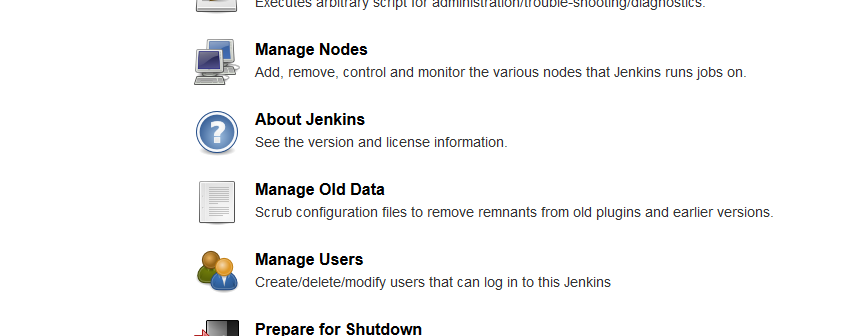
you can check installed Plug-ins under .Jenkins/Plug-in directory or in the Jenkins UI. Click on continue and you can set-up initial admin user account if needed or you can click continue as admin button at the bottom of the screen. (i have created a admin user vibra)

# Jenkins login screen

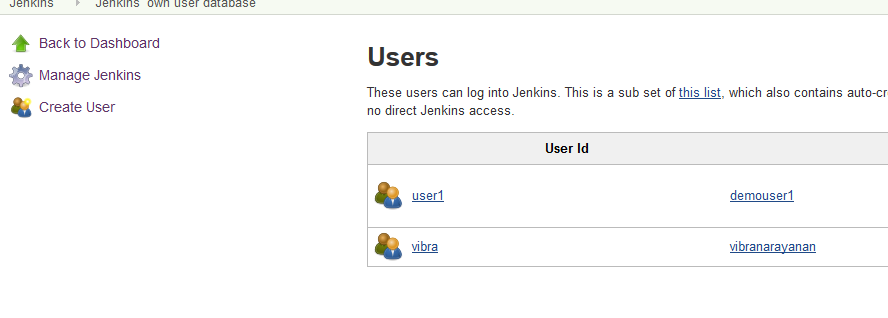


# User and access Management

#### How to create a new user

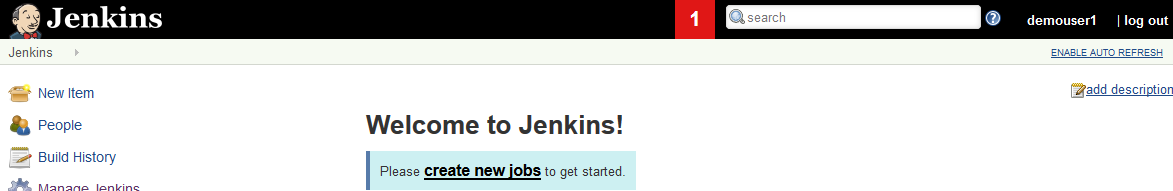


Click on manage users in above screen. Click on Create User link and enter required details, and submit. new user will be created. user1 created in below screen.



#### Checking new user login

Tried to login with newly created user account, I can able to login successfully.

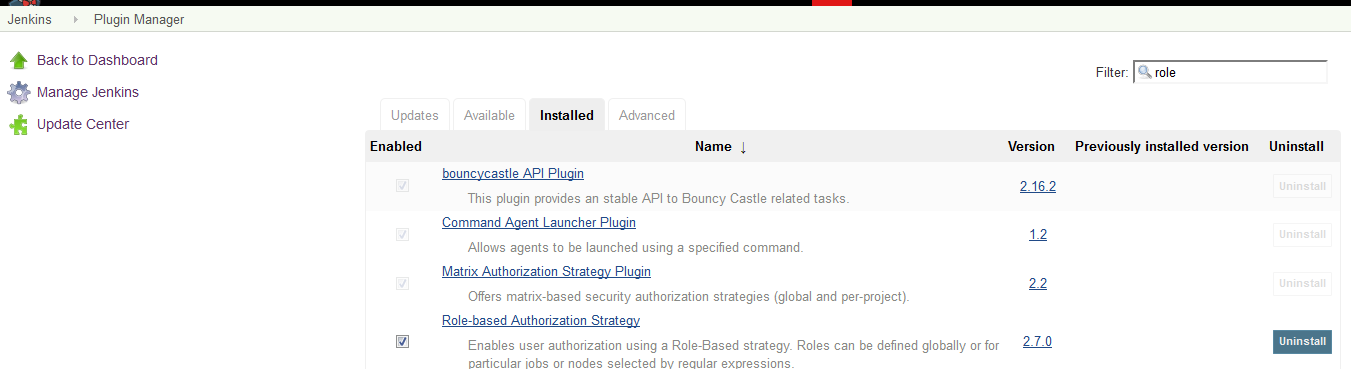


# Setting up access permissions/Roles.

since newly created user has all the permissions of admin. this is not needed in real time scenario. we can restrict user permissions as described below. we can separate developer and QE etc.

#### Role base plug-in installation

login with admin credentials. install Role-based Authorization plug-in you can search it in available plug-ins and install it .



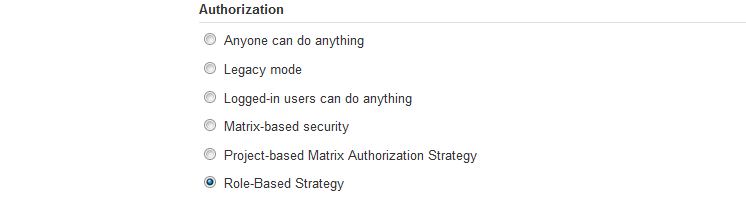
#### Configure Global security

Manage Jenkins >> Configure Global security



Make sure "Enable security " Checkbox checked.

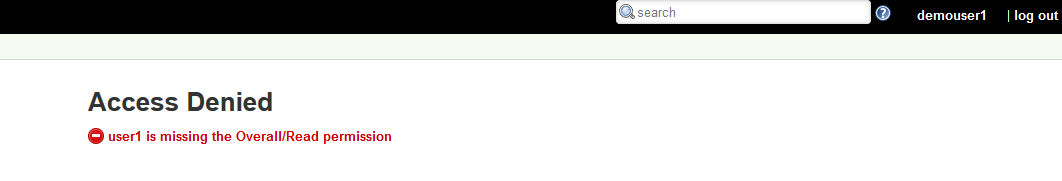
Scroll down and check radio button under Authorization section.



Checking user permissions after plug-in installation and configuration set-up.

#### Check user permission after configuration update

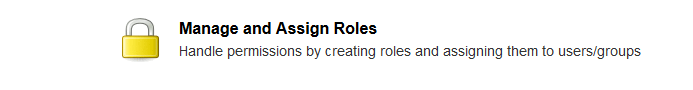
Login with newly created user now ("user1")



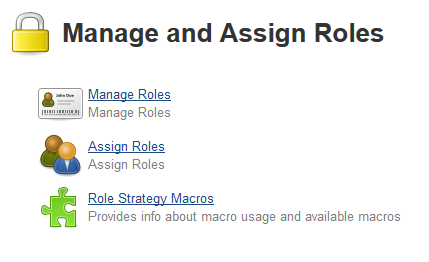
#### Assign Roles

Logout and login as admin user.

Manage Jenkins >> Manage and Assign Roles



Click Mange Roles



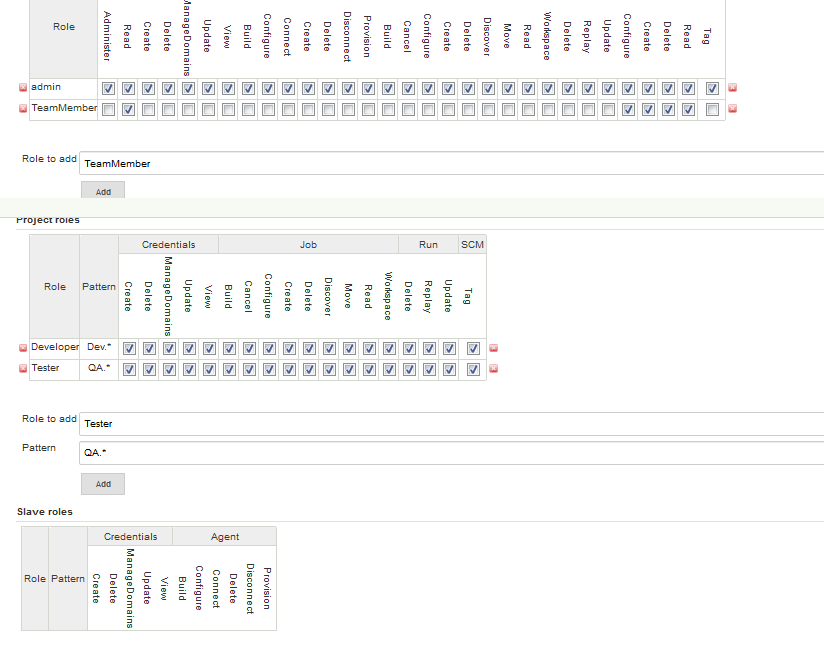
#### Role Types

you can see Three level of Roles

1) Global roles: Roles which can be applied globally (in our POC, created Team Member role with Read permission )   
2) Project roles: Project specific roles (Dev, QE etc) in our example created two roles developer and tester . with Patten Dev.\* and QA.\*

3) Slave Roles: (Need to explore)  
Select required permissions and "Apply" and "Save"

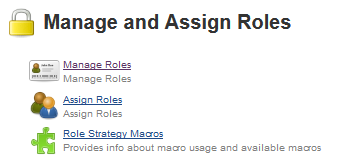
#### Role Creation



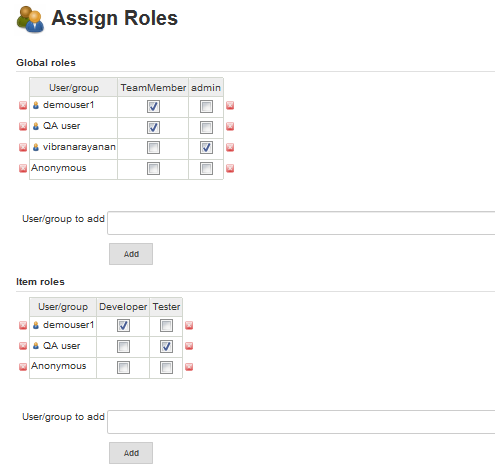
Now we have Created required Roles. we need to assign these to users

#### Assign Roles

Go to Mange Jenkins >> Manage and Assign Roles >> Assign Roles



Add "user1" and "user2" under Global Roles section and select "Team member " check box.

Add "user1" and "user2" under Project Roles section select "Developer"/"Tester" Roles.

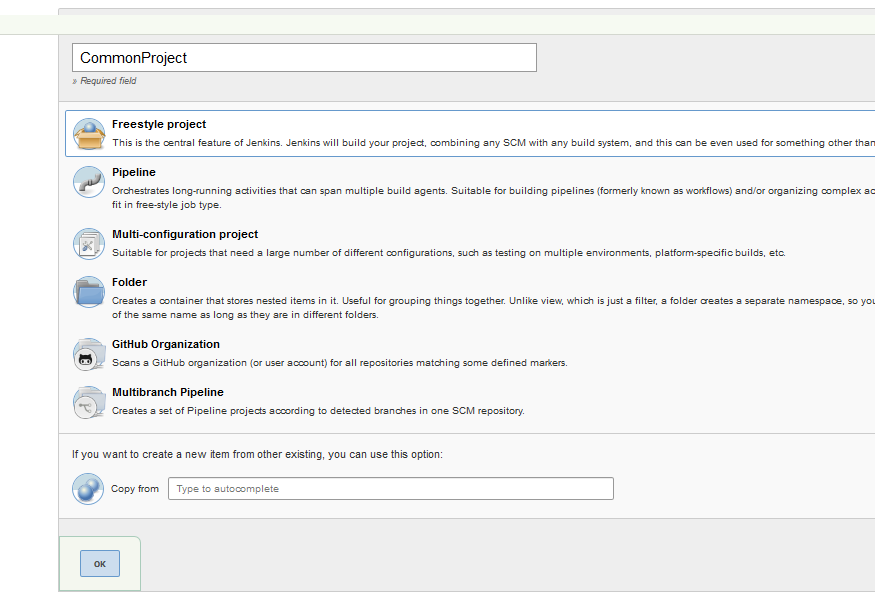
Now Role assignment over. We Also created Project level Roles, Developer and tester. To check this create below projects with same name, since we applied pattern DEV.\* and QA.\* .

1) DevProject

2) QAProject

3) CommonProject

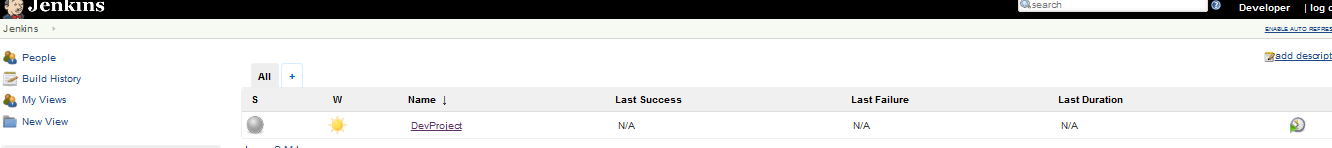
(goto Dashboard >> New Item) select "Free style project" >> Apply and Save. follow these and create above three projects.



Admin can able to list all three project created. Check below screen

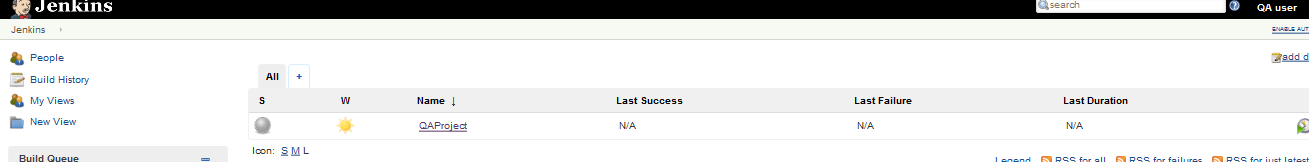


#### Check Developer Role

Login with Developer Role (user1) only DevProject listed. because of Role we assigned.

#### Check QA Role

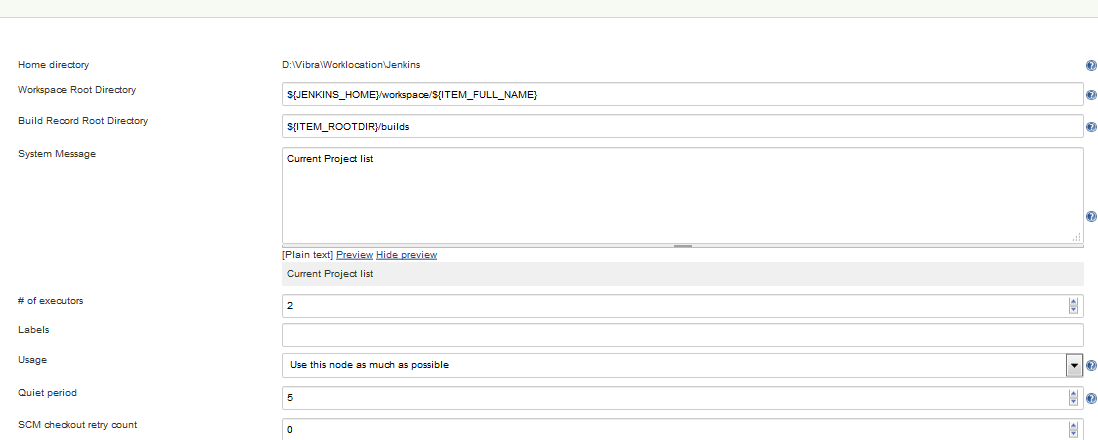
Logged in with QA user (user2) because of user's Role, QA project listed after login.



# Simple Configurations

#### System Message

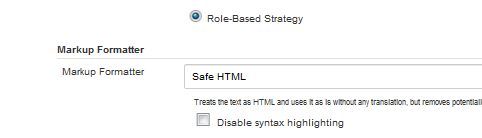
you can find Jenkins home directory and root directory where files systems located. you can enter system message. this will be displayed in dash board.



Entered Text displayed in dashboard section.

#### System message with HTML

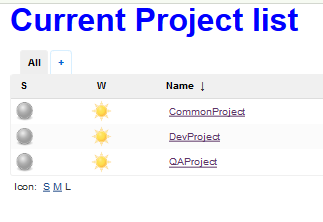
since currently this configured as Plain Text, we can configure this as HTML as well. this support can be enabled in Configure Global security section . "Apply" and "Save" ,Now Go back and update text with HTML tags.



Text updated with HTML tags



Text in Dash board.



#### Number Of Executers:



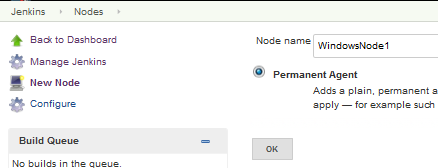
Maximum number of parallel jobs can be executed in this Jenkins instance .

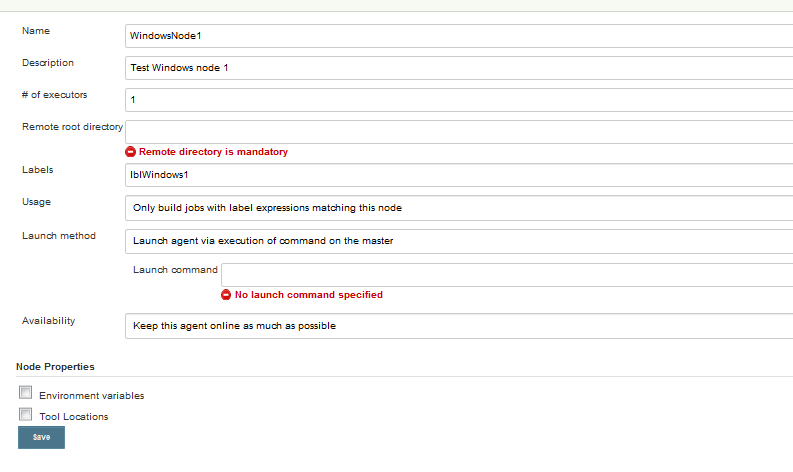
#### Manage Nodes/Labels

Goto Manage Jenkins >> Manage Nodes >> New Node

we can make our current Jenkins instance as Master and we can add other Machines as Node.

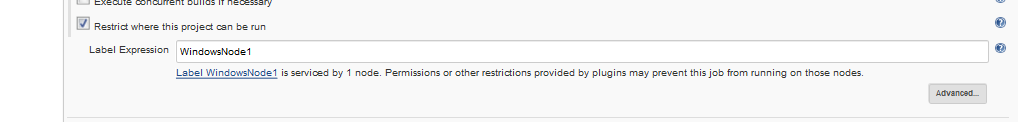
Enter Node name and click OK.



Number of Executers is max of concurrent builds can be performed in this node. and label you can give a label and usage section select Only jobs with this label expression matching this node.

Go to any of Jenkins Dash board and select any one of the project and select configure.

in general tab you can check in below check box and you can restrict the execution based on labels we selected.(it can be a regular expression)



#### Quiet period

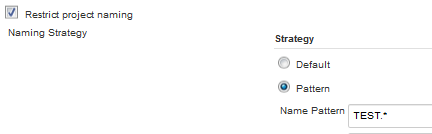
this will be the sleep period for this jenkin instance to trigger a particular job. some time Code commit may take time to complete. this period will help to trigger a job in specific interval. This will be very useful when we have automated jobs configure like Poll SCM etc. this will give some time to start the execution. we can update based on project nature.

#### SCM checkout retry count

Maximum number of retry if any failures during Jenkins to SCM connectivity to get build.

#### Restrict Project Naming

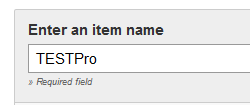
You can define a project naming Patten , by selecting "Patten" and provide name Patten and save.



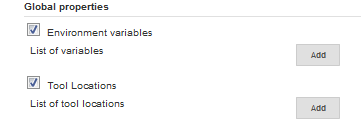
Now go and create Project. you can see the error message in red saying it is not matching Patten.



With Valid project name matching with Patten.



#### Global property



you can add Key value pair and use it globally.

# Jenkins CLI Setup

#### Why Jenkins CLI

* Easy to use
* Fast
* Memory management better than UI
* Continues Integration through CLI with scripts

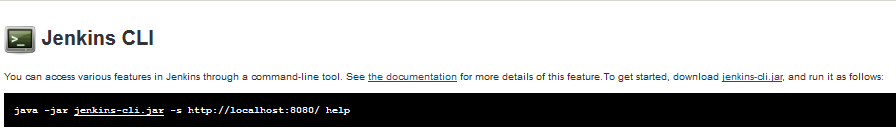
#### Start Jenkins

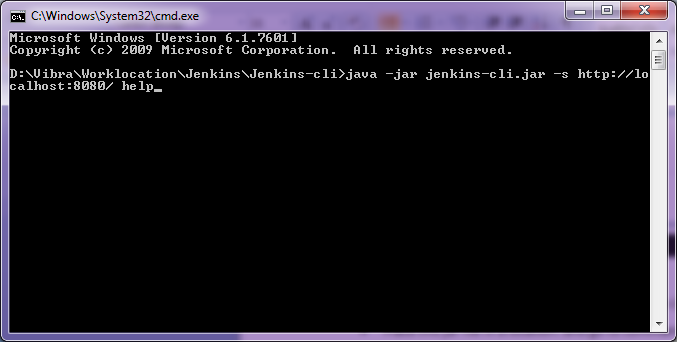
* Go to Installation directory
* go to command prompt
* type Jenkins.exe status
* If it is stopped , if it is try to start
* #Jenkins.exe start <enter>
* Now check in browser ("http://localhost:8080")

#### Enable Global security

* Go to manage Jenkins >> Configure Global security >> Enable Security.

#### Install Jenkins CLI

* access CLI url ("http ://localhost:8080/cli/ ")
* Click on Jenkins-cli.jar link and download Jar file.(refer above screen)
* Place this jar file in a location. and go to command prompt of that location
* execute the command showed in Jenkins CLI URL (refer above screen)



* refer above screen,

Note: Make sure Java installed and ENV variables set. if not please download and install java and set the ENV path.

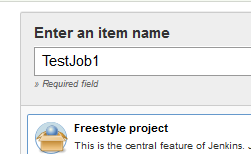
* if you have any issue with executing command listed in Jenkins UI, check if you need to add authentication parameters to the commands or you can set anyone can do anything in authentication section of Global config section to test this.

## Starting with Jenkins JOB without SCM Tool

#### Creating a Job

##### Update to General Information tab

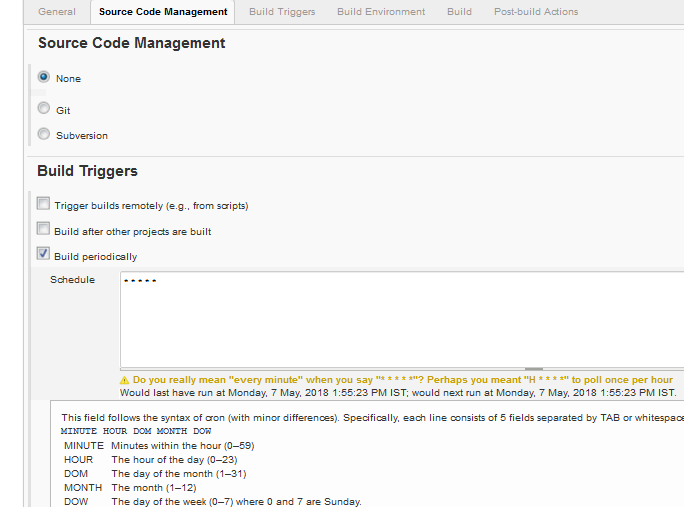
Goto Jenkins Dasboard >> Add new item >> enter Job title and select Project type as Freestyle. and click OK



##### Update to Source code management tab

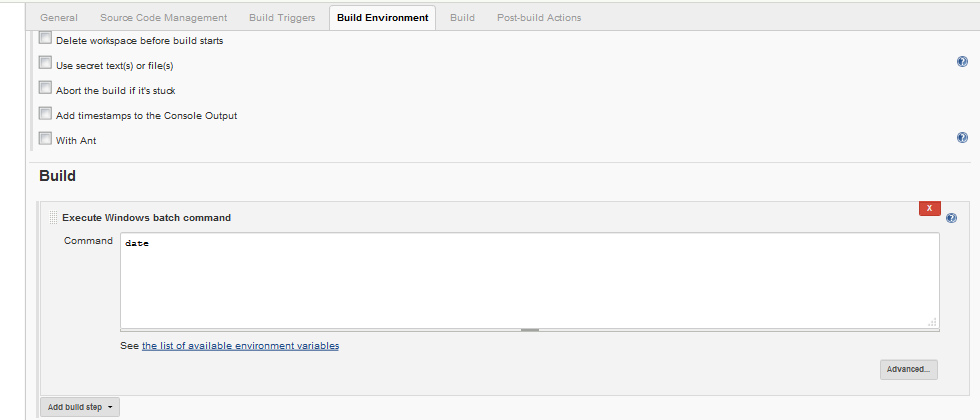
Refer below screen.

Update build periodically and update schedule details. this is similar like setting a Corn Job  
In below example it is set for every minute . you can get more details, by click on help icon given nearby schedule column.



##### Update to Build Tab

you can select Build option based on project need. currently updated to execute batch script. (execute date command for this POC)



##### Update to Post-build actions tab

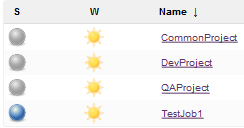
You can set actions like notification or any other options based on need. here i left as it is.

## Executing builds with newly created Job

#### Before executing Build

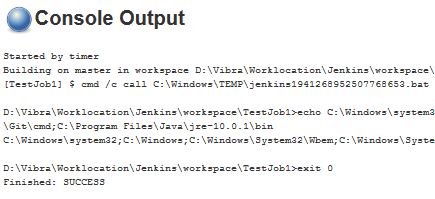


#### After Executing Build



you can see TestJob1 turned to blue . since we executed successful builds. Weather column image will be updated based on build failures. if you see any job with red , that is due to build failures. Please check on console log and take required action.

#### Console LOG of successful Build



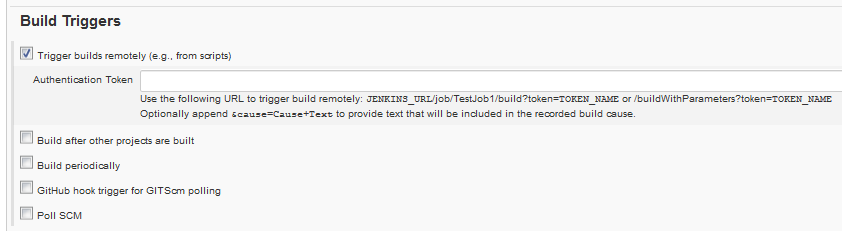
In above log you can see "echo %PATH%" command execute successfully.

## Triggering builds remotely

Current Build status  


#### Configuring remote builds

Go to Jobs configuration section. under build trigger section deselect existing selection. our case it is build periodically. select "build trigger remotely"



you can see URL format, just below Authentication token field (EX: **JENKINS\_URL**/job/TestJob1/build?token=TOKEN\_NAME or /buildWithParameters?token=**TOKEN\_NAME**)

Replace JENKINS\_URL with your host name /IP. our case it is local host.

1) **http://localhost:8080**/job/TestJob1/build?token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME

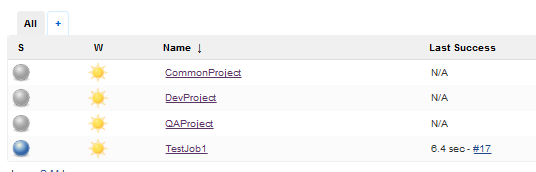
2) update Authentication Token field (EX: REMOTETOKEN123)

3) Update token name in above URL (**http://localhost:8080**/job/TestJob1/build?token=TOKEN\_NAME or /buildWithParameters?token=**REMOTETOKEN123**)

4) Access above update URL in other browser session http://localhost:8080/job/TestJob1/build?token=REMOTETOKEN123

You can see #17 created earlier it was #16.

#### Build screen after Remote build execution



## Simple Java build without GIT integration

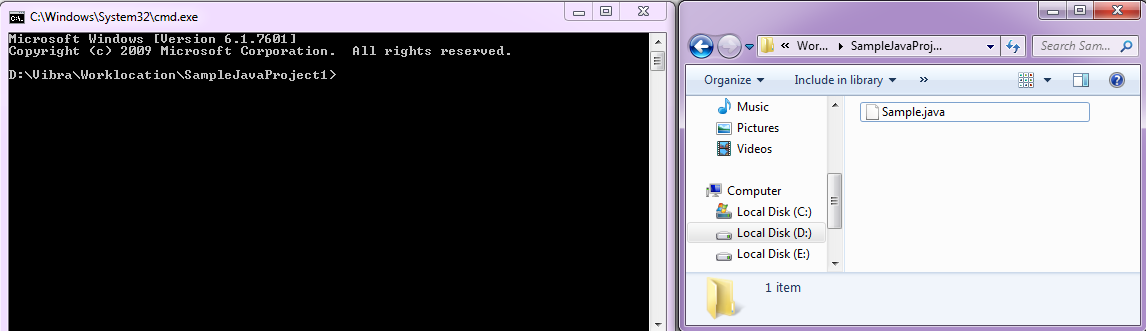
#### Create simple Java Program

Create simple java programs and compile and validate it is working correctly. if not you can use this embedded java program and compile it.

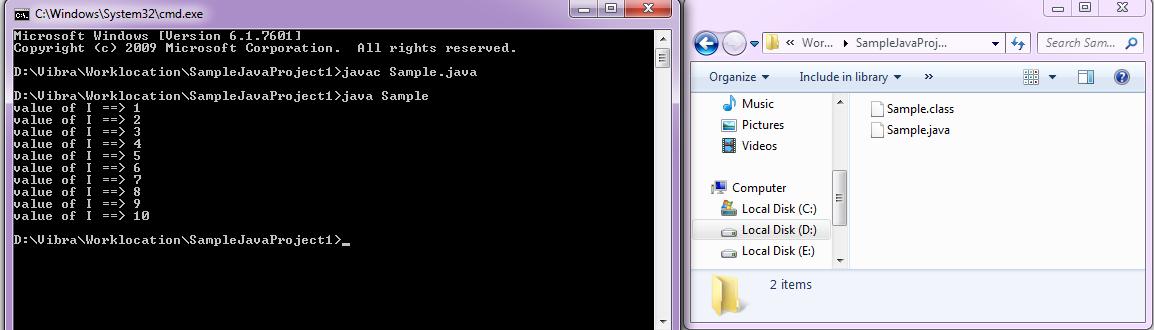


below screen is a snapshot before compiling java program.

#### Before compilation



#### After compilation



you can see in right side folder class file created by executing Javac command and you can see output by executing Java Sample command.

#### Build integration with Jenkins

Now you can integrate and build using Jenkins, by following below steps.

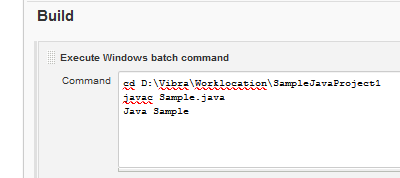
##### Create simple Java project Job in Jenkins

Jenkins Dash board >> New item (Freestyle project)

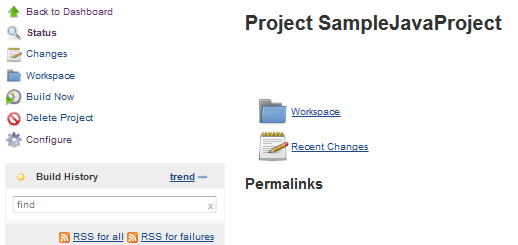


##### Configure Jenkins Build

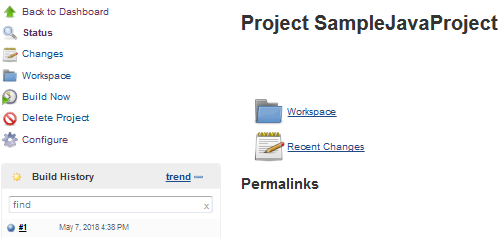
Under Build section select execute via windows batch commands and add commands to execute/compile java program. below screen will explain basic steps to execute.



##### Before executing build



##### After executing build

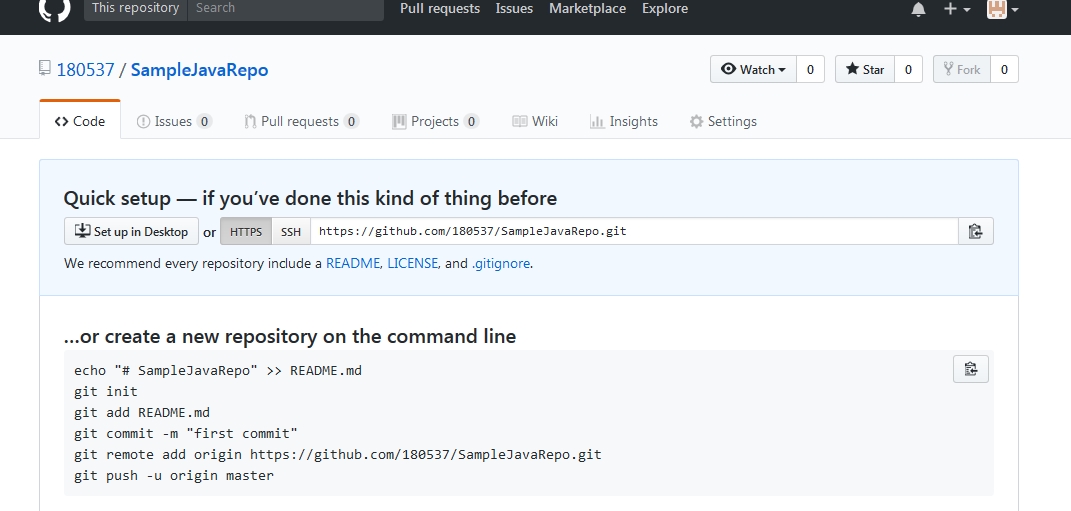


##### Console output after successful execution

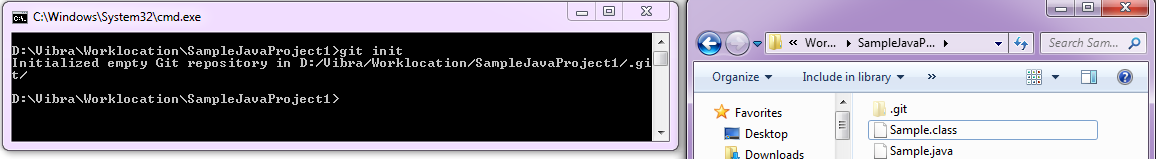


## Jenkins Jobs integration with GIT

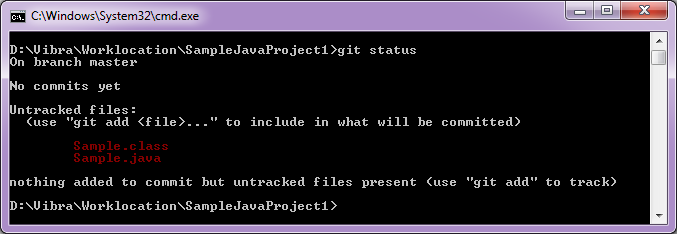
#### Create Github repo

Login to your github account and create a sample repo. if you don't have github acount create one in github.com and proceed further. Make sure github plug-in installed and available for use. 

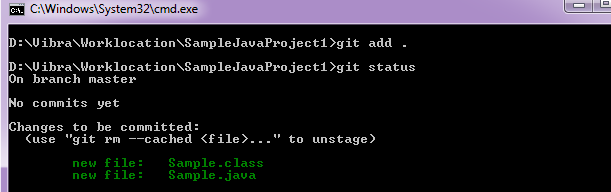
#### GIT initiation in project location

  
execute git init inside your project location you can see .git created under your project folder this is the key for all your git transactions.

#### Check file status

  
execute git status command inside your project directory. you can see files and its status. in above screen you can see 2 file available for commit and it is not added into git tracking.

#### Adding file into git



Now you can add files into git by executing "git add ." this will add all new files.

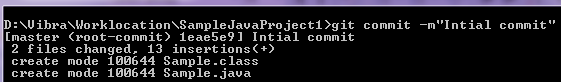
#### Setting Global property in git

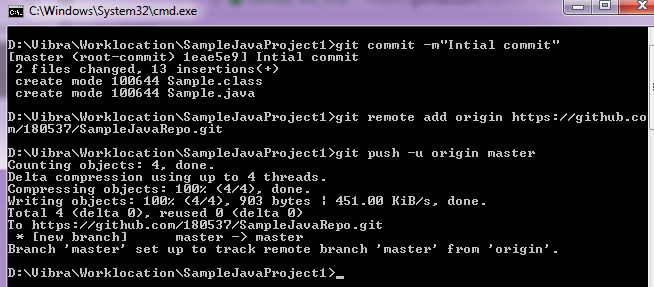
Before committing your change in git make sure your email and user name added/updated into git. you can achive this by executing below commands.

git config --global user.email "you@example.com"  
git config --global user.name "Your Name"



#### Committing Changes in git

  
By executing git Commit you can you can commit files into git.

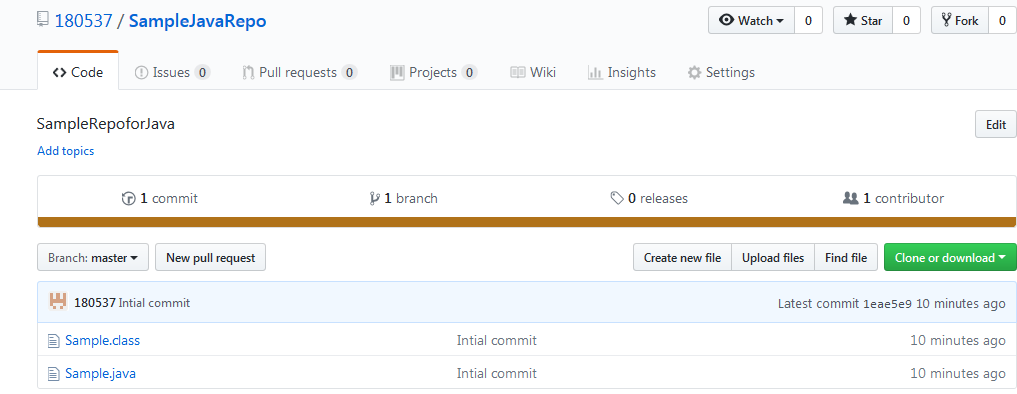
Adding remote repo and pushing code into repo

you can achieve this by executing "git add origin <repo location>" . you can note command/repo location during repo creation and you can use it here.

"git push -u origin master" will push your changes to master branch.

#### Verify it in github

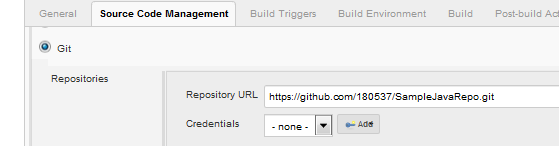
now you can check it in your github account, under newly created projects these files were added.



#### Jenkins Configuration changes.

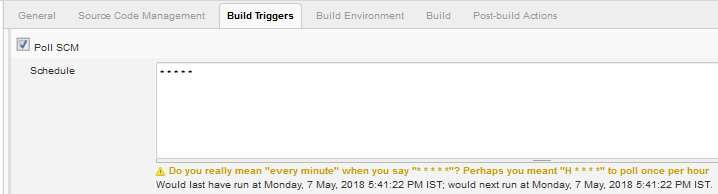
##### Add github repo details

go to your project/Job configuration and select git under SCM tab. it will ask for repo URL. you can copy the URL from your github clone option, and you can past it here.

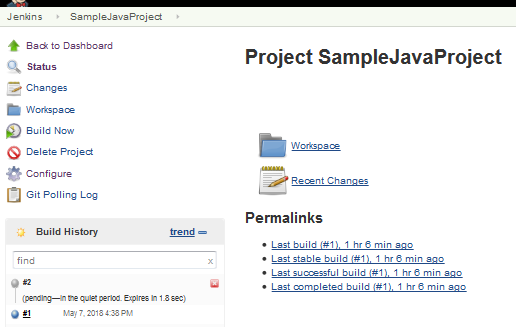
  
EX: https://github.com/180537/SampleJavaRepo.git

##### Setting build triggers

you can set Poll SCM option and for testing purpose it is scheduled for every minutes. this is not recommended. you can set interval based on project need. below setup will look for any change in github and create build with new changes committed if any.

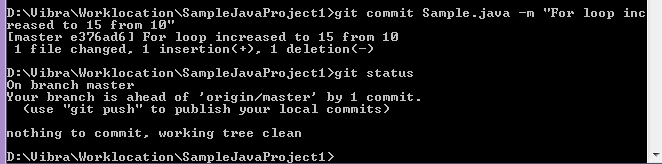


#### Build screen before committing change



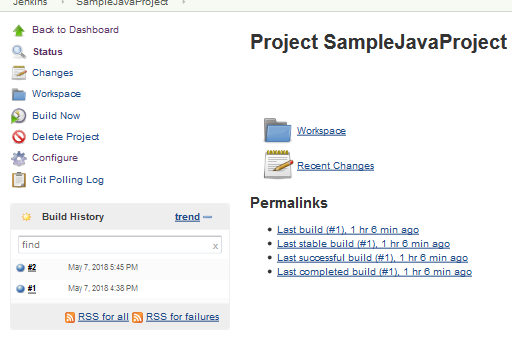
Do changes in you Java program and commit. for commit you can refer earlier steps "[Committing changes into git](#_Committing_Changes_in)"

Changes committed into main and pushed to githup repo by executing "git push -u origin master"



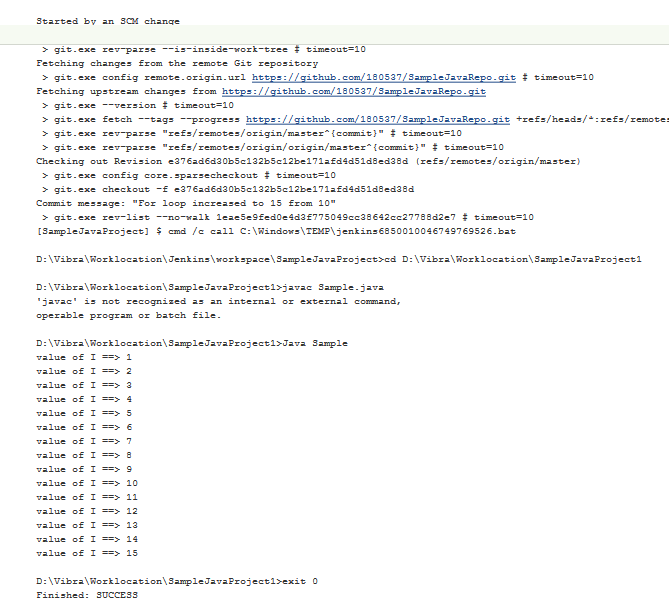
#### Build screen After Commit.

Now you can see #2 build automatically created after git commit.



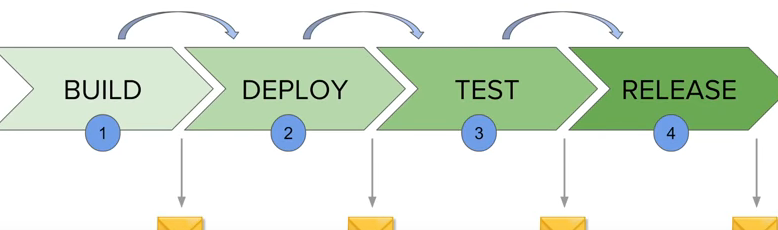
Jenkins JOBs console output

you can see build is successful and output values shows from 1 to 15. same changes available in github. Make sure after commit you are pushing changes to githup repo. by executing "git push -u origin master"



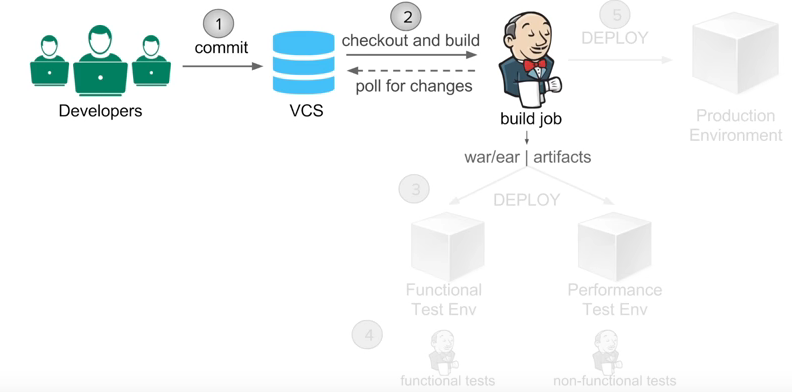
## How to do Automated Deployment

#### Stages in Continues Delivery and Deployment Pipeline

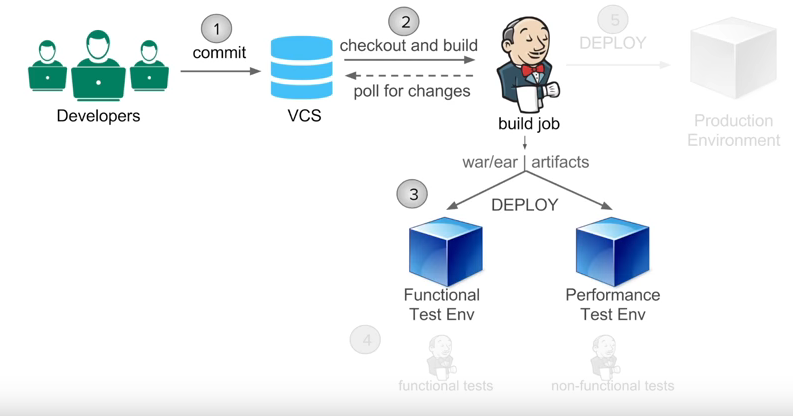


each 4 stages are managed by jobs and these jobs are chained. based on successful state of each job/stage process will moved to next stage/job and each stage can be activated by Notification based on need.

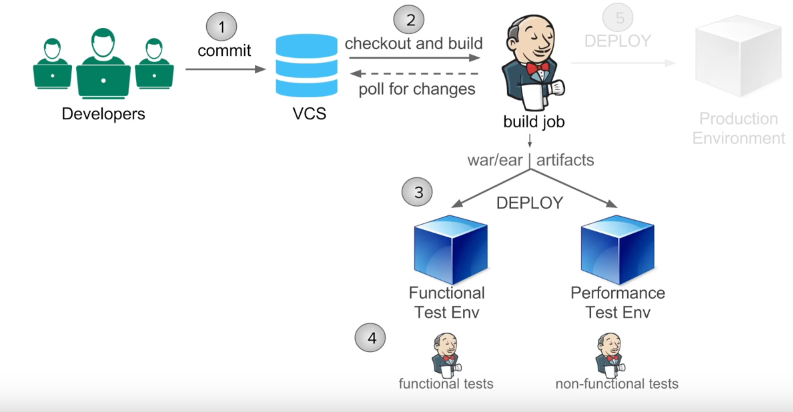
##### Build stage



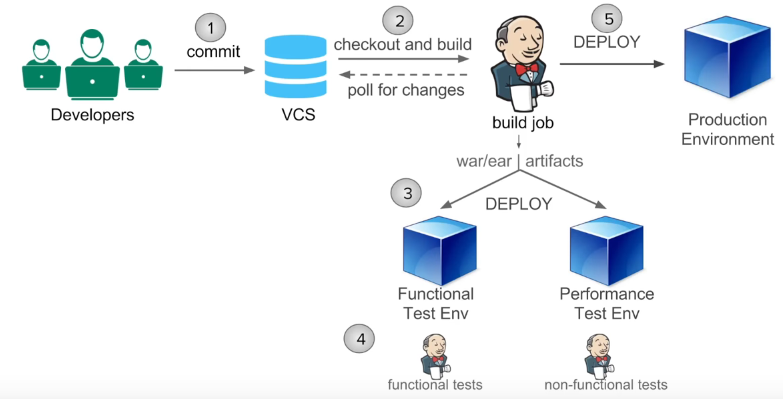
##### Deployment Stage



##### Test stage



##### Release Stage



## Setting up sample Auto deployment

#### download sample war file

you can download sample war file from internet . or use below sample file

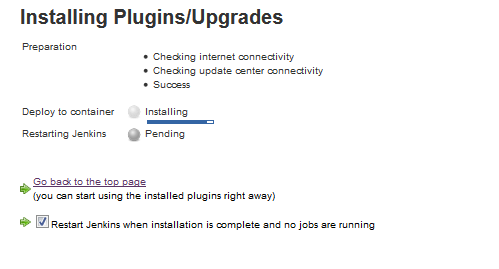


#### Deployment Plug-in installation

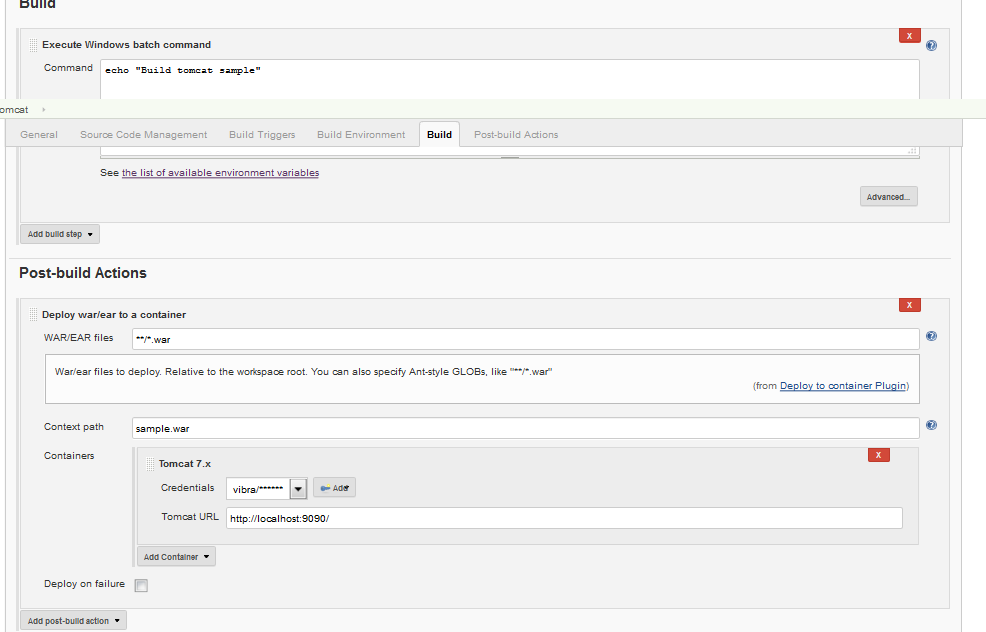
##### Plug-in installation

#### 

Go to manage plug-in section and search for Deploy to container plug-in and install.



#### Setting up Jenkins Job for auto deployment.

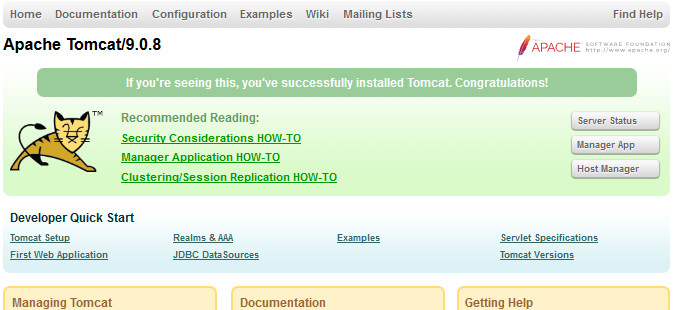
Create a job for this automated deployment - refer below screen for more details.

place the war file inside Jenkins work space specific to this job in this POC this is "D:\Vibra\Worklocation\Jenkins\workspace\AutomatedDeploymentTomcat".

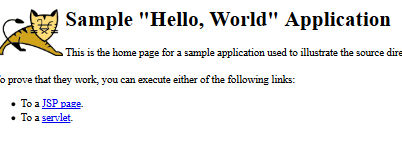
under build section type some windows command, since we just executing build action no specific action performed in this POC.

in Post build action section \*\*/\*.war in war/ear files section and select container as 7.x this will work for latest version too. and make sure your tomcat instance running. and make sure required roles defied under tomcat user-role.xml and restarted.

#### Tomcat server before deployment



#### Tomcat screen after deployment

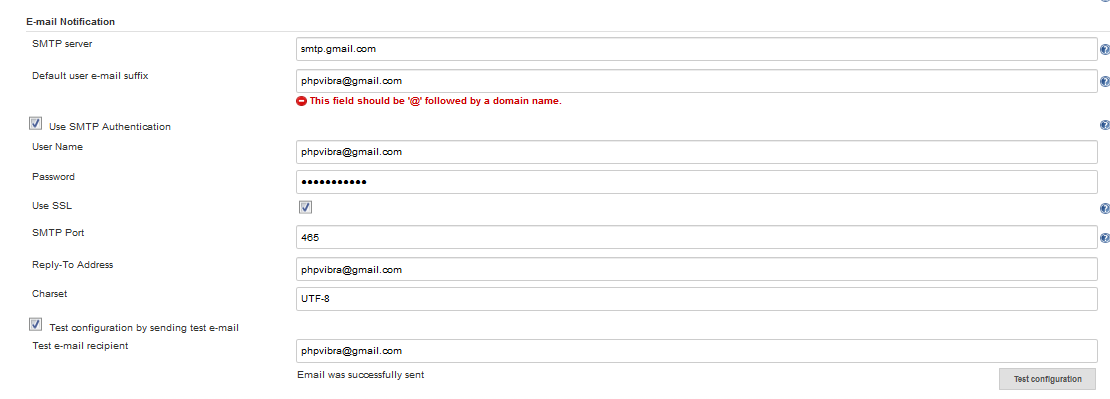


## Enabling Basic email Notification

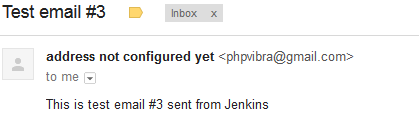
#### Notification configuration in Jenkins

Go to manage Jenkins >> system configurations >> email notification

in this POC configured with Gmail. enter required SMTP and SSL details and test connection you will get a test email from jenkins.

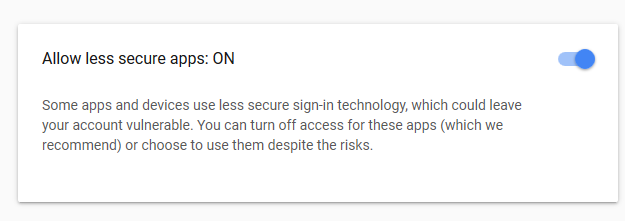


#### Test Alert verification in Gmail

Login to Gmail and check if you got test alert.

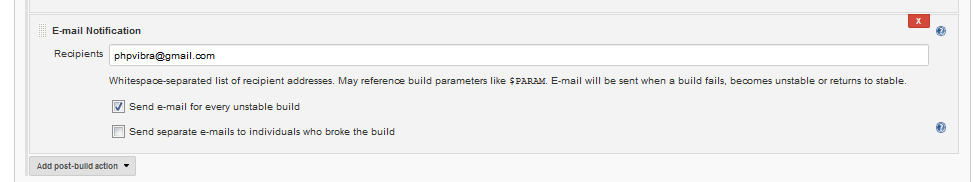
#### trouble shooting with Gmail email configuration

if you get error during test email from Jenkins make sure you enabled below options. you select my account in Gmail and click sign in to google and this option will be available at the bottom. after enabling you should receive emails from Jenkins



#### Setting email notification for unstable builds

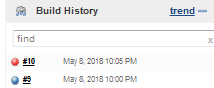
Go to job's configurations and at the bottom, we can update email notification section. in this POC updated with my email id to notify unstable buils.



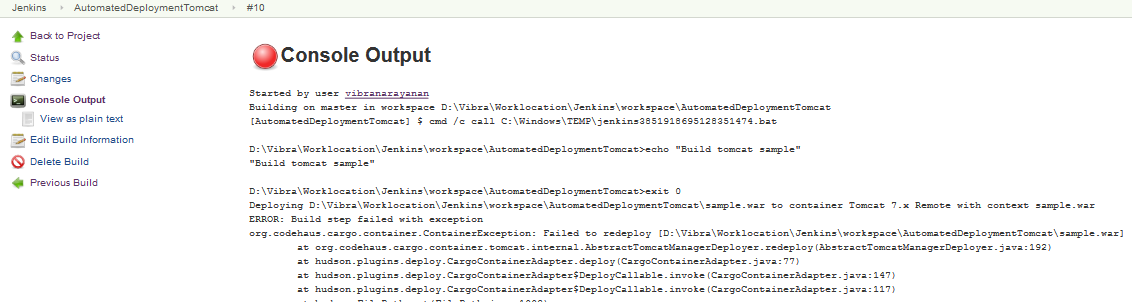
#### Checking unstable email notification from Jenkins

To check this, stopped tomcat server and tried to create a build. since tomcat is not running so deployment will fail. in this case job status will be updated as unstable build. this will trigger a email to configured email in that particular Job.

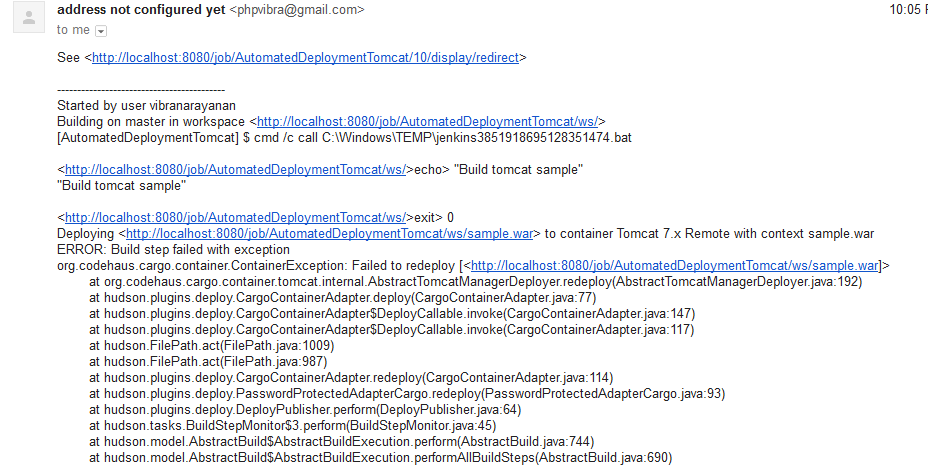
##### Build status



##### Job's Console screen



##### Email screen

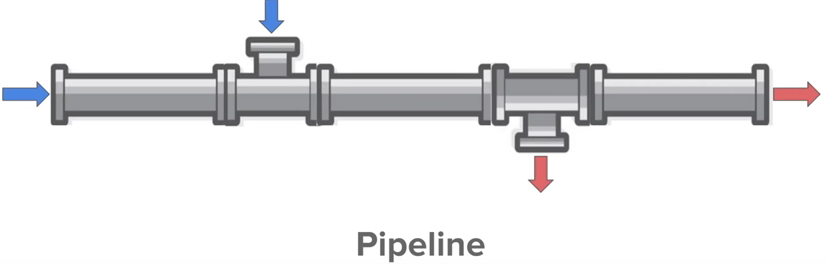


## Jenkins Pipeline

#### What is Pipeline

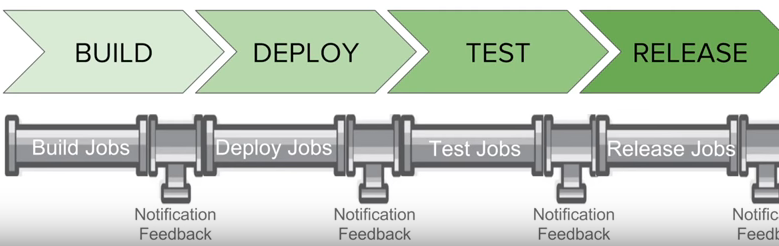
Pipeline is a work flow with group of events or jobs that are chained and integrated with each other in sequence.

Every job in a pipeline has some dependency on one or more other jobs.



Every section will do some process and in this case this will be a job.

#### Continues delivery Pipeline flow



##### Upstream Job

A job to be executed before current job.

##### Downstream job

A Job to be executed after current job.

## How to setup Delivery Pipeline in Jenkins

#### Create sample jobs

These jobs can be used to set our delivery pipeline flow.

##### Create Job 1 - Sample Build Job

|  |  |
| --- | --- |
|  |  |

##### Create Job 2 - Sample Deploy Job

|  |  |
| --- | --- |
|  |  |

##### Create Job 3 - Sample Test Job

|  |  |
| --- | --- |
|  |  |

##### Verify newly created job

Very all three jobs are executing with our any issues. you can manually build and verify these three jobs . Verified and all jobs completed successfully.



#### Chain Newly created Jobs

Job 1 (Build job )is the parent Job in our POC. so no changes required in this Job Configuration. we can start update Job configuration from deploy to test. After chaining Jobs flow will be

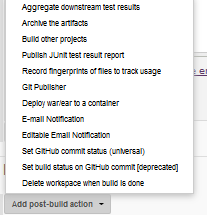
Build >> Deploy >> Test

After successful execution of Build Job(Parent job), other two jobs (child/downstream jobs) will be executed one by one based on upstream job's status.

##### Configuration Options for Job chaining

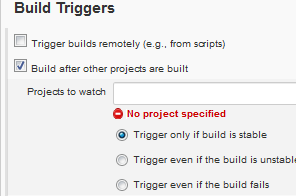
this configuration done in two ways

1) Post build action  
under this section you can select build other project option and mention upstream Job name



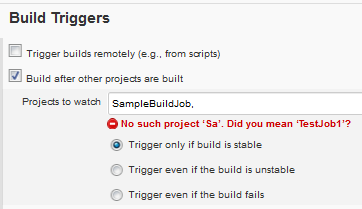
2) Build triggers

Same option is available under each job's build trigger option.

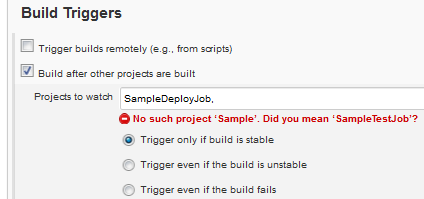


##### Configuration Update in Job2 (Deploy Job)

this POC followed #2 chaining option



##### Configuration Update in Job 3 (Test Job)



Now all three jobs are chained each other starting from parent Job build.

#### Verify chaining

##### Before chaining Build status

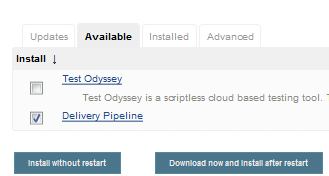


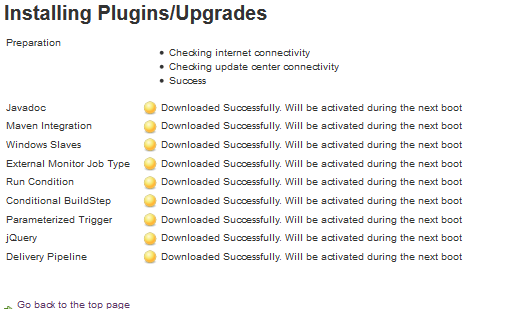
##### After chaining Build status

Executed Build Job manually and all other jobs executed automatically one after another



## Jenkins Pipeline Integration

This Plug-in available under Manage Jenkins >> Manage plug-in >> Available section .

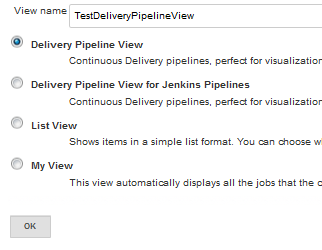


#### Creating Delivery Pipeline view

Go to Jenkins dash board. click on "+" in Tab section.



Enter Pipeline details and submit "Ok".



You can keep default values. Go to pipeline section and update initial/parent job details.

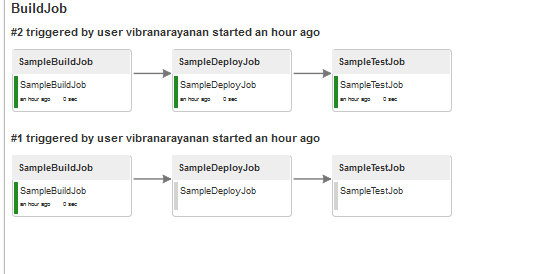
Click Add button Pipeline >> Components.



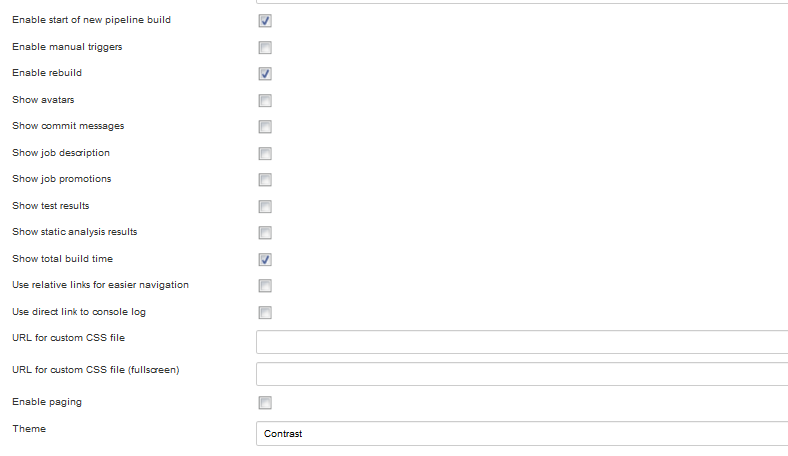
Enter Title and select parent job in initial Job field. in this POC this will be Sample Build Job.

#### Pipeline view

Now 3 jobs listed in pipeline view. You can see #2 build with green, Since this build trigged after jobs were linked. #1 build shows as gray in Deploy and Test job since these Jobs were not linked initially.

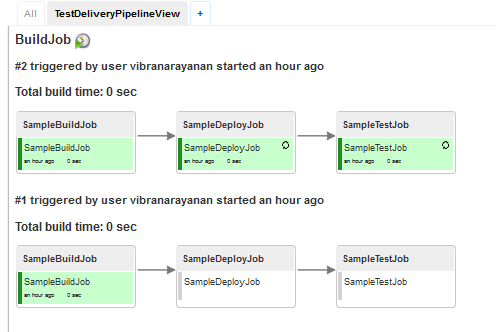


#### Pipeline view Configuration Changes

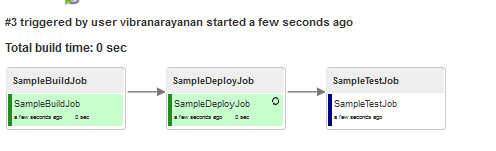


#### Pipeline view after Configuration changes

You Can see build button added in this view , rebuild option added in each Job, Total build time added and view theme update.



Created a build in pipeline view, green colour Jobs denotes completed and Blue colour jobs denotes in-progress.

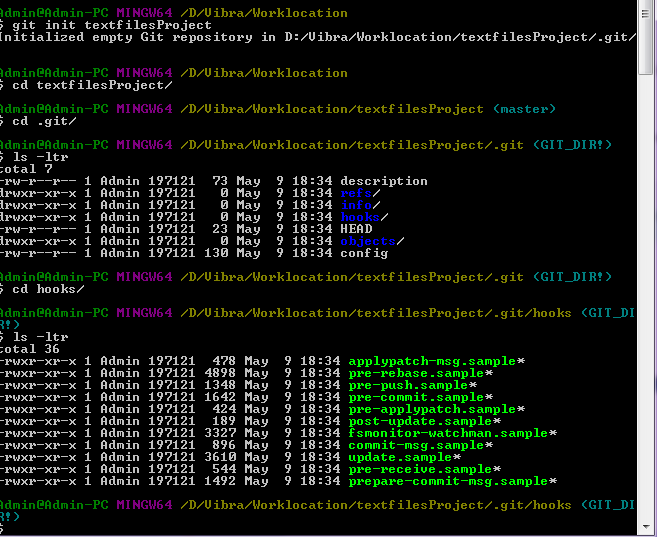


## Pre-commit hook

pre-commit hook will help to validate the commit and allow user to commit if test/validation is pass. below POC will try to check, user committing text files alone into git.

#### Initiate a Project in Git

Initiate a project which will have only text files.



after project initiation, go to .git/hooks folder you can see list of sample scripts which can be performed. we will take a scenario of pre commit.

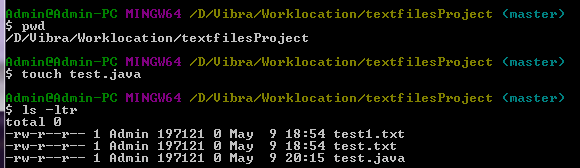
#### Create pre-commit hook

create pre-commit file inside .git/hooks folder. created one pre-commit hook to check user committing only text file. if any other files committed commit will fail with error message. below script is a sample you can change the script based on you experience in shell/bash script.

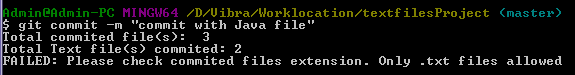


This script will automatically triggered when user try to commit.

#### Create java and text files inside test project

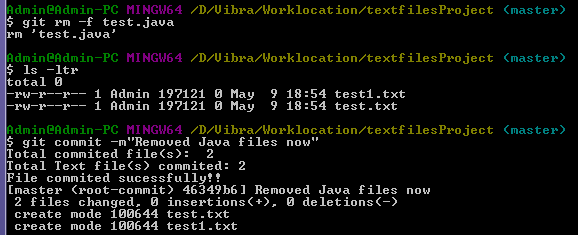
 you can see 2 text files and one java file created in textfilesProject.

#### Commit with Java file

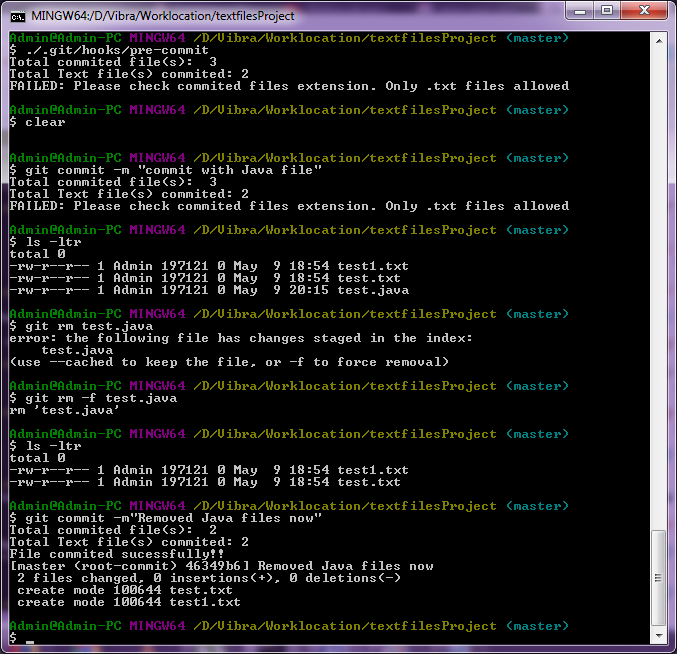


as expected this commit failed and got error custom error message and count of total files committed and txt files count.

#### Commit without Java file



Removed Java file and committed, as expected pre-hook script allowed this commit and got total count and text file count and custom message.



After git push these files added in to github.

