JENKINS DOCUMENTATION

		Author	
		Web site	

Jenkins

Introduction

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Continuous Deployment (CD)

Installation

In Linux Server

Create the a CICD flow for Java Web App using Freestyle Project type

- Integrate Maven software if not done.
- Integrate SonarQube with Jenkins
- Integrate Nexus with Jenkins
- Deploy the App into Tomcat
  - Through “Deploy to container” plugin.
  - Through Script
- Configure Email Functionality
- Poll SCM
- Build Periodically
- Git Web Hooks
- Discard Old Build
- Disable this project
- Delete workspace before build starts
- Add timestamps to the Console Output
- JACOCO plugin

Jenkins Directory structure

Create a CICD for Java App using Maven Project type

Plugin Management

- **Deploy to container**
- Deploy WebLogic
- WebSphere Deployer
- **Maven Integration**
- **Safe Restart**
- Next Build Number
- **JaCoCo**
- **SSH Agent**
- **Email Extension**
- SonarQube Scanner
- **Audit Trail**
- **Job Config History**
- Schedule Build
- Blue Ocean
- **Publish Over SSH**
- **ThinBackup**
- **Build Name Setter**

- **Convert To Pipeline**
- **Role based Authorization Strategy**

External Plugins Installation
Urban Code Deploy

- to specific access to specific projects

Jenkins Security

Create Views

Create a CICD for Java Project using Pipeline Project Type – Scripted way

Create a CICD for Java Project using Pipeline Project Type – Declarative way

Create the Multibranch Pipeline Project Jobs

Jenkins Backup

Jenkins Migration

Create Master/Slave

CICD implementation for Node JS Project

Jenkins Shared Libraries.

Optional Topics
- Jenkins Home Directory Change in RHEL 7.5 Version
- Jenkins CLI
- Integrate the Urban Code Deploy server with Jenkins
- Deploy the App into IBM Cloud
- Slack integration

Introduction

Jenkins, is an open source Continuous Integration, cross-platform tool written in Java. Kohsuke Kawaguchi is Creator of the Jenkins CI server in 2004. Initially, it was called Hudson, but in 2011 it was renamed to Jenkins because of disputes with Oracle.
The tool simplifies the process of integration of changes in to the project and delivery of fresh builds to users.

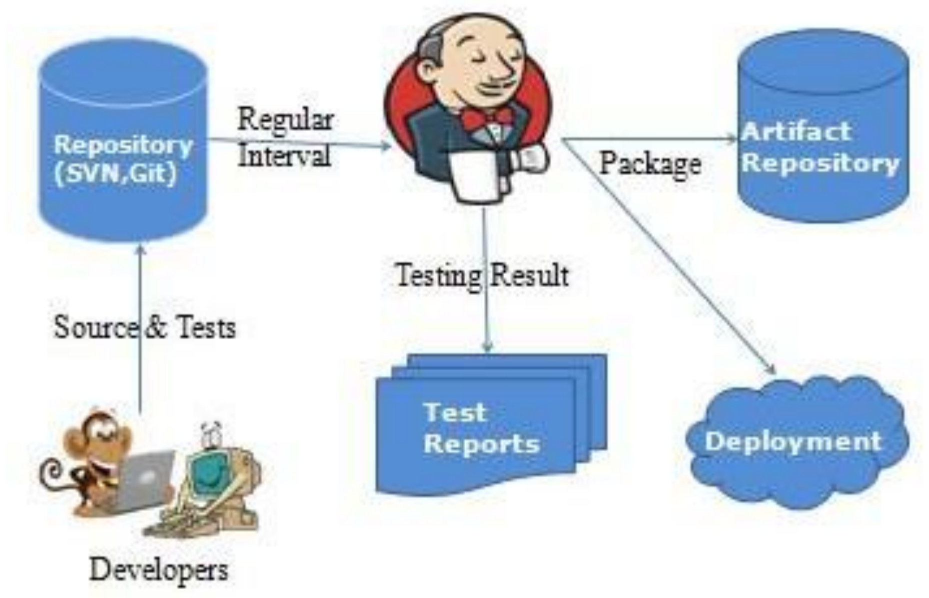
Continuous Integration: Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control.

(OR)

Continuous Integration is a development practice where developers integrate their code into a shared remote repository frequently, preferably several times a day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

CI Flow

Below diagram CI flow with Jenkins as Build tool.

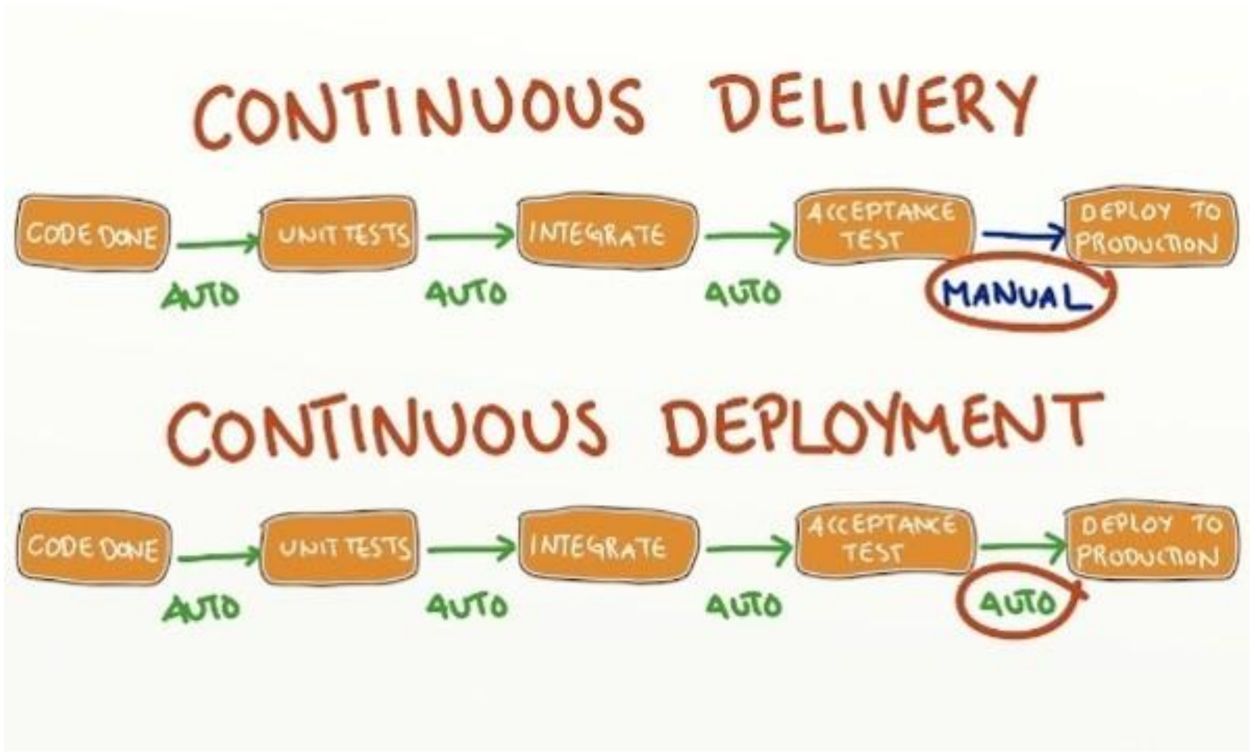


CI – Benefits

- Immediate bug detection
- No integration step in the Software Development lifecycle
- A deployable system at any given point
- Record of evolution of the project

Continuous Delivery: Any and every successful build that has passed all the relevant automated tests and quality gates can potentially be deployed in to production via fully automated one click process.

Continuous Deployment: The practicing of automatically deploying every successful build directly into production without any manual steps knows as Continuous deployment.
(OR)
It is closely related to Continuous Integration and refers to keeping your application deployable at any point or even automatically releasing to a test or production environment if the latest version passes all automated tests.



What Jenkins can do?

- Integrate with many different Version Control Systems (GitHub, CVS, SVN, TFS …)
- Generate test reports（JUnit)
- Push the builds to various artifact repositories
- Deploys directly to production or test environments
- Notify stakeholders of build status (Through Email)

Benefits of Jenkins

✓ It's an open source tool with great community support.

✓ Easy to install and It has a simple configuration through a web-based GUI, which speeds up the Job

✓ It has around 1500+ plugins to ease your work. If a plugin does not exist, just code it up and share with the community (https://plugins.jenkins.io/).

✓ Its built with Java and hence, it is portable on all major platforms.

✓ Good documentation and enriched support articles/information available on internet which will help beginners to start easy.

✓ Specifically, for a test only project, it is used to schedule jobs for regression testing without manual intervention and hence monitor infrastructural and functional health of a application. It can be used like a scheduler for integration testing and also can be used to validate new deployments/environments on a single click on a Build now button.

The diagram below depicts that Jenkins is integrating various DevOps stages:

TBD (ToBeDocument)

- -

List of popular Continuous Integration tools

SNo	Product	Is Open Source?
1	Jenkins	Yes
2	Cloudbees Jenkins	No
2	Bamboo	No
3	Cruise Control	Yes
4	Travis CI	Yes and Paid also
5	Circle CI	Yes and Paid also
6	GitLab CI	Yes and Paid
7	TeamCity	Yes and Paid

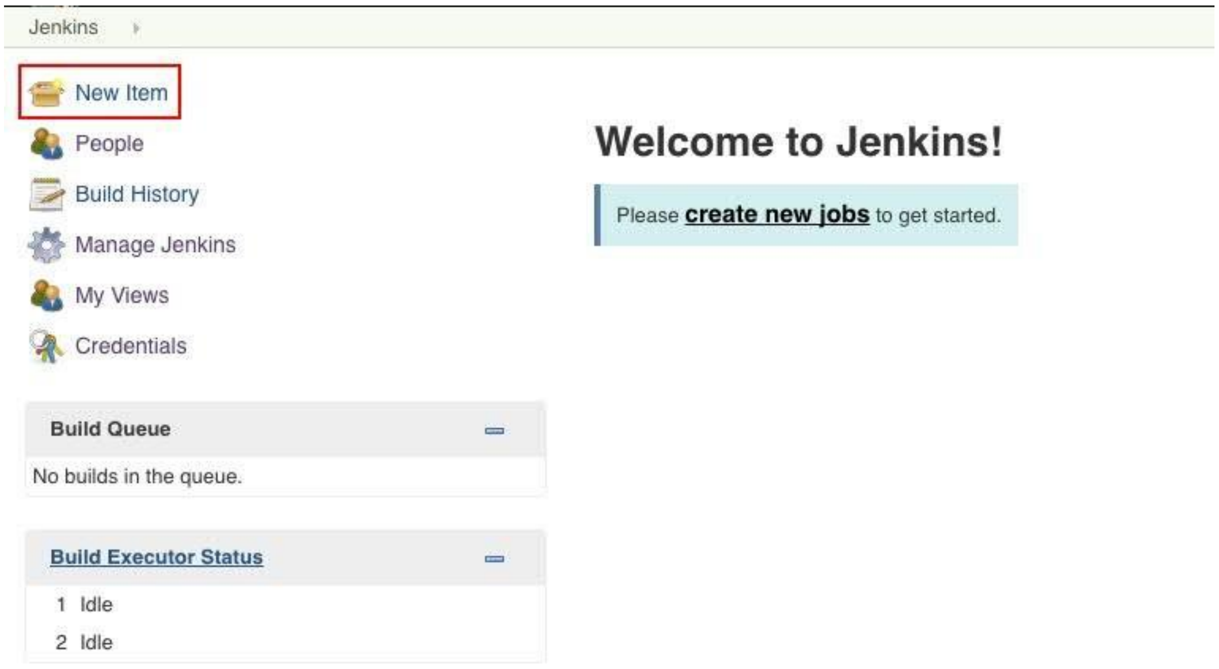
Jenkins Installation

- ➢ Jenkins is java based CI tool, so we need to install jdk/jre before installing.
- ➢ **Pre-Requisite Software:** Java (Check weather java is installed or not with java -version command)

- -

Create the project/job in Jenkins

Step 1: Login into the Jenkins, go to the Jenkins dashboard left side top corner, click on **New Item**.



Welcome to Jenkins!

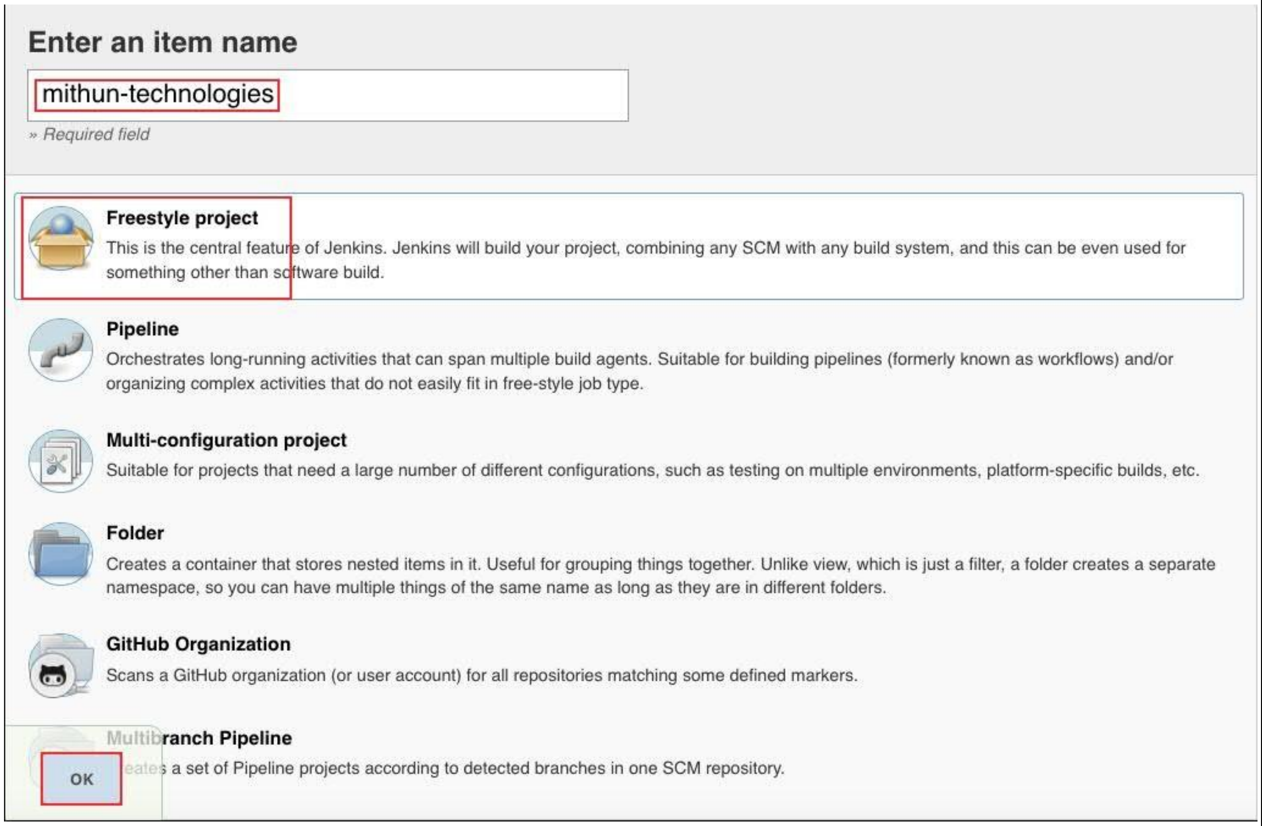
Please **create new jobs** to get started.

No builds in the queue.

1 Idle

2 Idle

Step 2: Enter the project name in **Enter an item name** input box and select the **Freestyle project** and click on **OK** Button.


Enter an item name

mithun-technologies

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or 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 testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

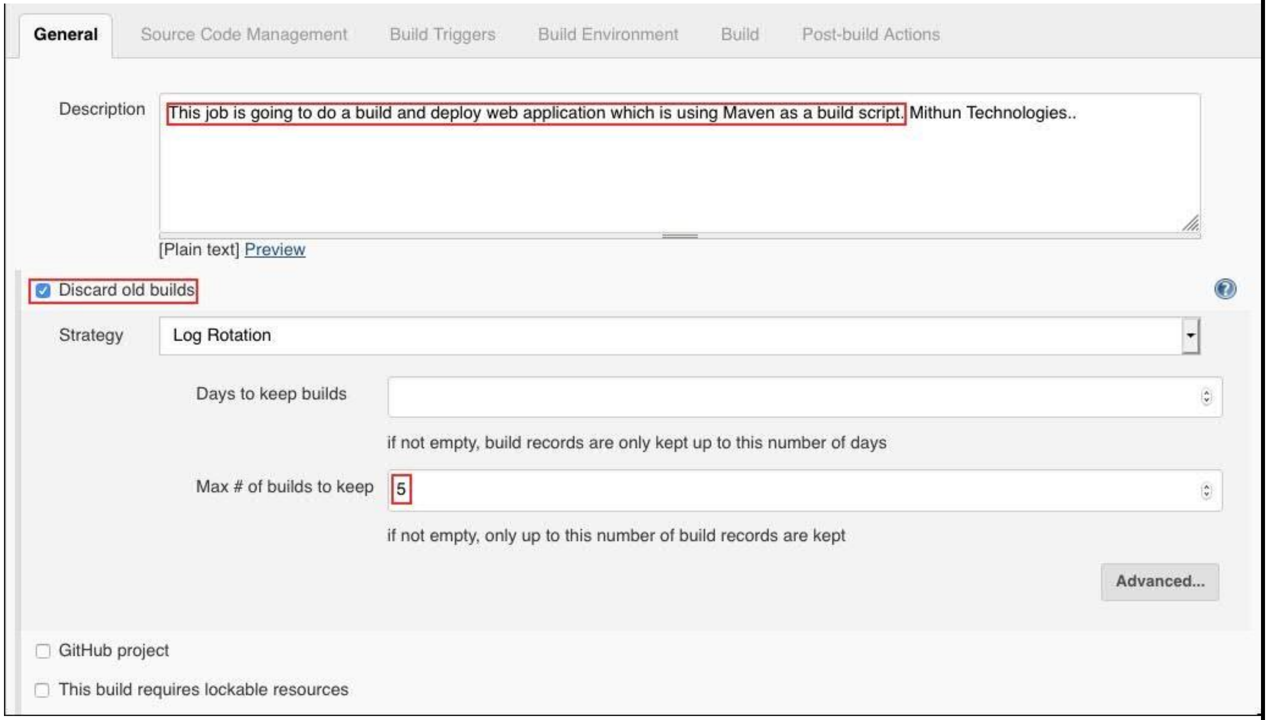
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Freestyle project: This is the central feature of Jenkins. Jenkins will build your project combining any SCM and any build system.

A Free-Style project is a project that can incorporate almost any type of build. The Free-Style project is the more "generic" form of a project. You can execute shell/dos scripts, invoke ant, and a lot more. 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 “multiconfiguration 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 multiconfiguration 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.



Description: This job is going to do a build and deploy web application which is using Maven as a build script. Mithun Technologies..

[Plain text] Preview

☑ Discard old builds

Strategy: Log Rotation

Days to keep builds:

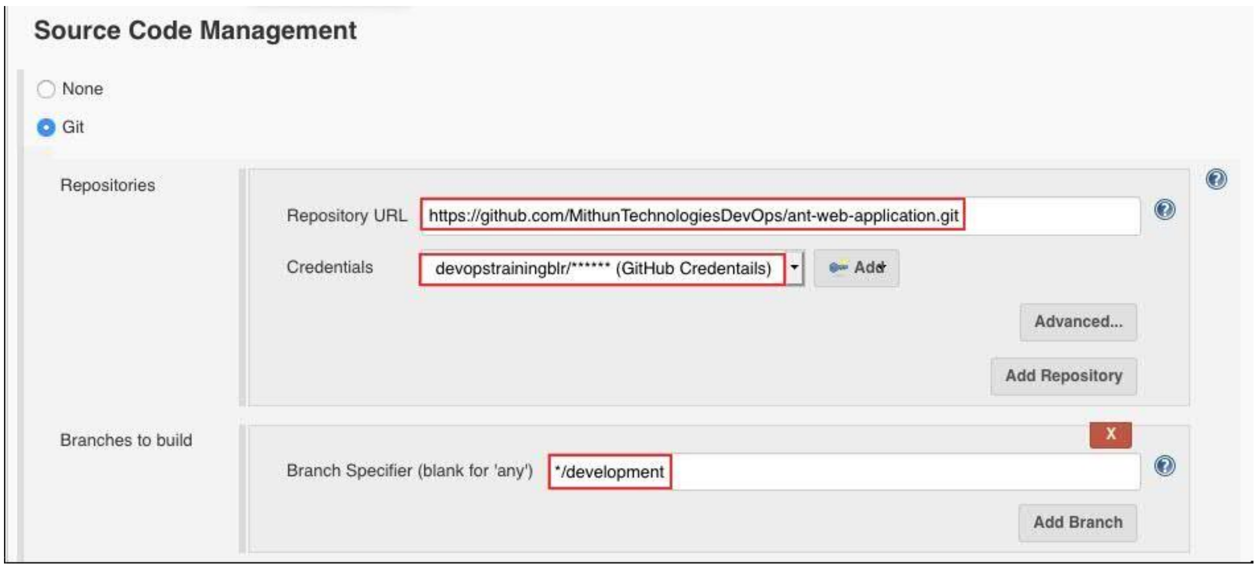
if not empty, build records are only kept up to this number of days

Max # of builds to keep: 5

if not empty, only up to this number of build records are kept

☐ GitHub project

☐ This build requires lockable resources


Source Code Management

○ None

● Git

Repositories

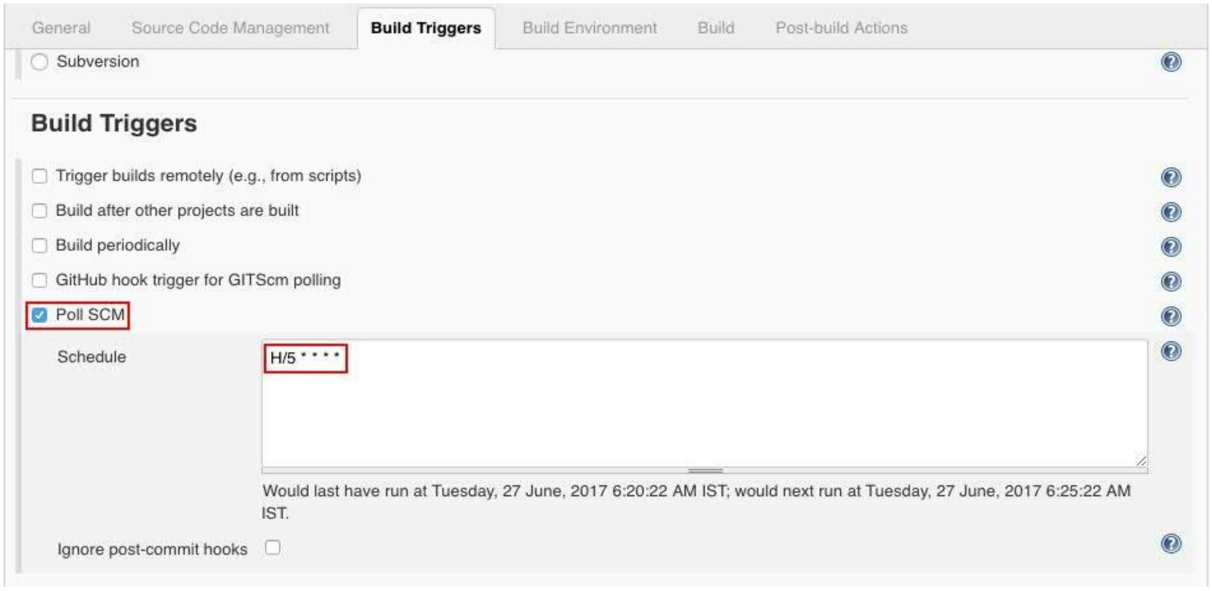
Repository URL: https://github.com/MithunTechnologiesDevOps/ant-web-application.git

Credentials: devopstrainingblr/****** (GitHub Credentails)

Branches to build

Branch Specifier (blank for 'any'): */development

Specify when and how your build should be triggered. The following example polls the Git repository every 5 min. It triggers a build, if something has changed in the repo.



Subversion

Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [x] Poll SCM

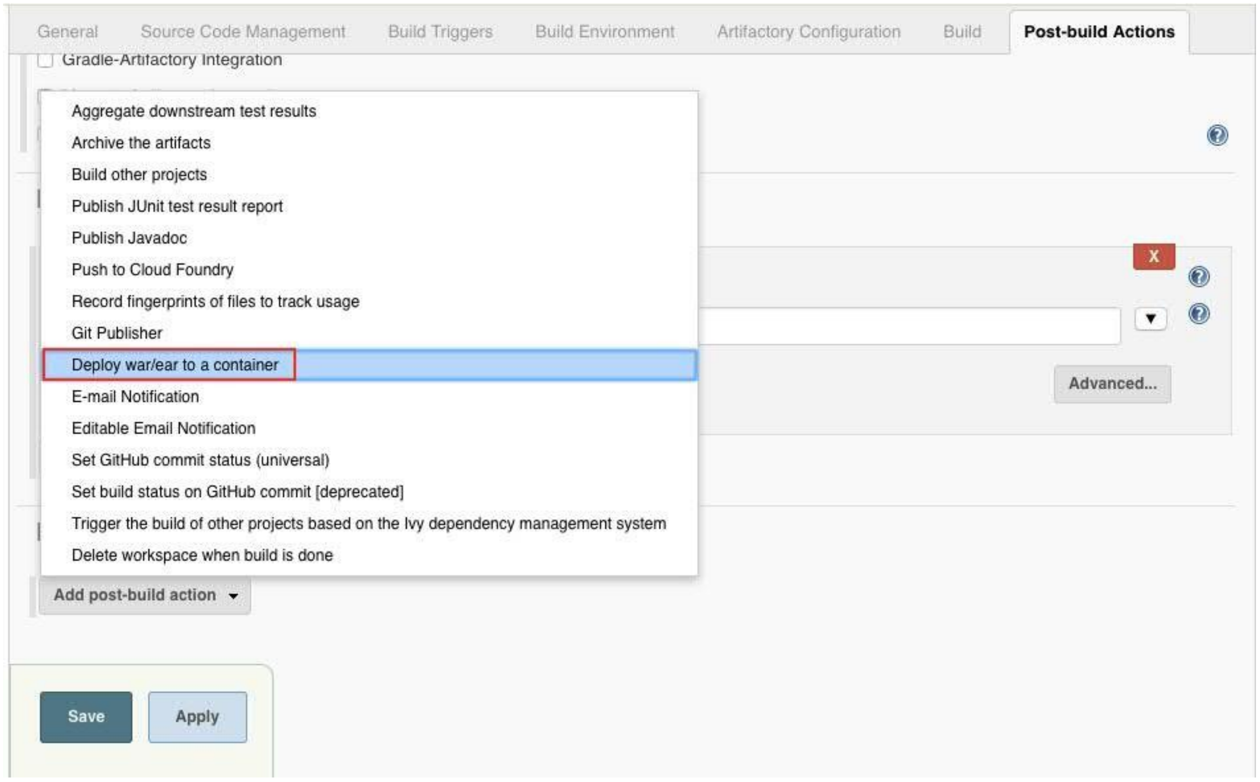
Schedule: H/5 * * * *

Would last have run at Tuesday, 27 June, 2017 6:20:22 AM IST; would next run at Tuesday, 27 June, 2017 6:25:22 AM IST.

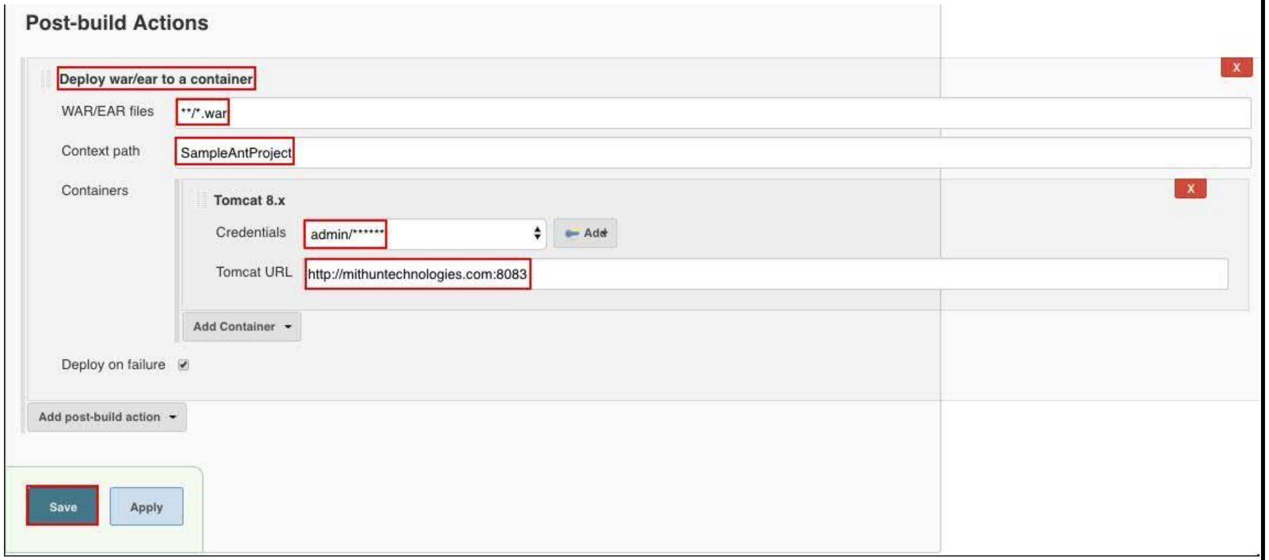
Ignore post-commit hooks [ ]

Deploy the application into Tomcat

Install the **"Deploy to container"** plugin.
Open the job which you want to configure deploy, and click on Configure and in **Post-build actions** tab, click on **ADD POST-BUILD ACTION** and select the **Deploy war/ear to container** as follows.



General
Source Code Management
Build Triggers
Build Environment
Artifactory Configuration
Build
Post-build Actions
Gradle-Artifactory Integration
Aggregate downstream test results
Archive the artifacts
Build other projects
Publish JUnit test result report
Publish Javadoc
Push to Cloud Foundry
Record fingerprints of files to track usage
Git Publisher
Deploy war/ear to a container
E-mail Notification
Editable Email Notification
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Trigger the build of other projects based on the Ivy dependency management system
Delete workspace when build is done
Add post-build action
X
Advanced...
Save
Apply



Deploy war/ear to a container

WAR/EAR files: **/*.war

Context path: SampleAntProject

Containers: Tomcat 8.x

Credentials: admin/******

Tomcat URL: http://mithuntechnologies.com:8083

Deploy on failure ✓

Error:

```
Caused by: org.codehaus.cargo.container.tomcat.internal.TomcatManagerException: The username you provided is
not allowed to use the text-based Tomcat Manager (error 403)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.invoke(TomcatManager.java:704)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.list(TomcatManager.java:876)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.getStatus(TomcatManager.java:889)
        at
org.codehaus.cargo.container.tomcat.internal.AbstractTomcatManagerDeployer.redeploy(AbstractTomcatManagerDeplo
yer.java:173)
        ... 17 more
Caused by: java.io.IOException: Server returned HTTP response code: 403 for URL:
http://localhost:8085/manager/text/list
        at sun.net.www.protocol.http.HttpURLConnection.getInputStream0(HttpURLConnection.java:1894)
        at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1492)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.invoke(TomcatManager.java:571)
        ... 20 more
```

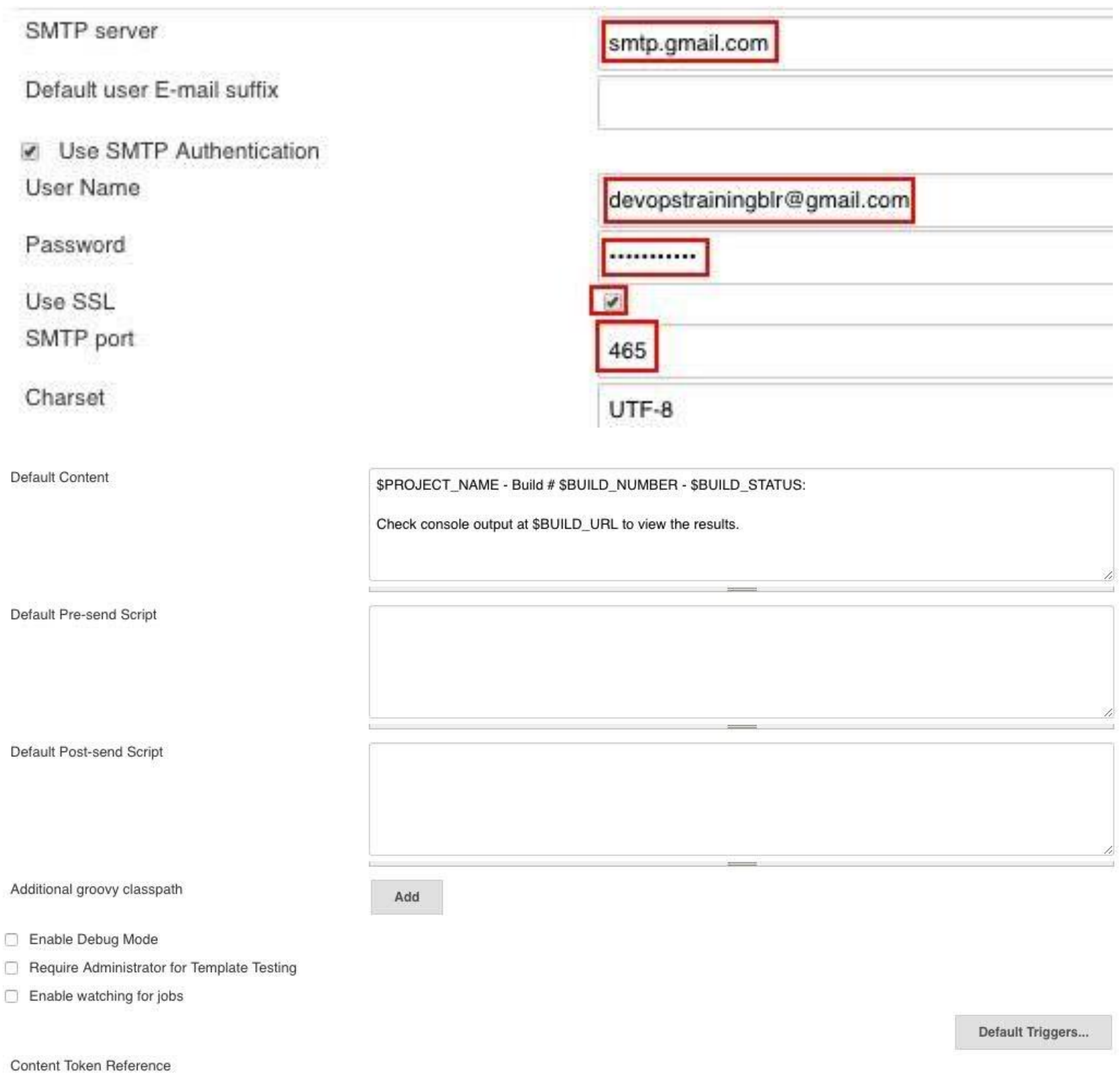
Solution: Need to add rule in tomcat-users.xml file as follows.

<user username="admin" password="passw0rd" roles="admin-gui,manager-gui,**manager-script**"/>

Enable email notification

Step 1) Install Email Extension Plugin as follows.
Manage Jenkins ---> Manage Plugins ---> Install **"Email Extension Plugin "**

Step 2) Add the smtp server host as follows.
Click on Manage Jenkins ---> Configure System --->



SMTP server: smtp.gmail.com

Default user E-mail suffix:

[✓] Use SMTP Authentication

User Name: devopstrainingblr@gmail.com

Password: ••••••••••••

Use SSL: [✓]

SMTP port: 465

Charset: UTF-8

Default Content:

```
$PROJECT_NAME - Build # $BUILD_NUMBER - $BUILD_STATUS:

Check console output at $BUILD_URL to view the results.
```

Default Pre-send Script:

Default Post-send Script:

Additional groovy classpath

[ ] Enable Debug Mode

[ ] Require Administrator for Template Testing

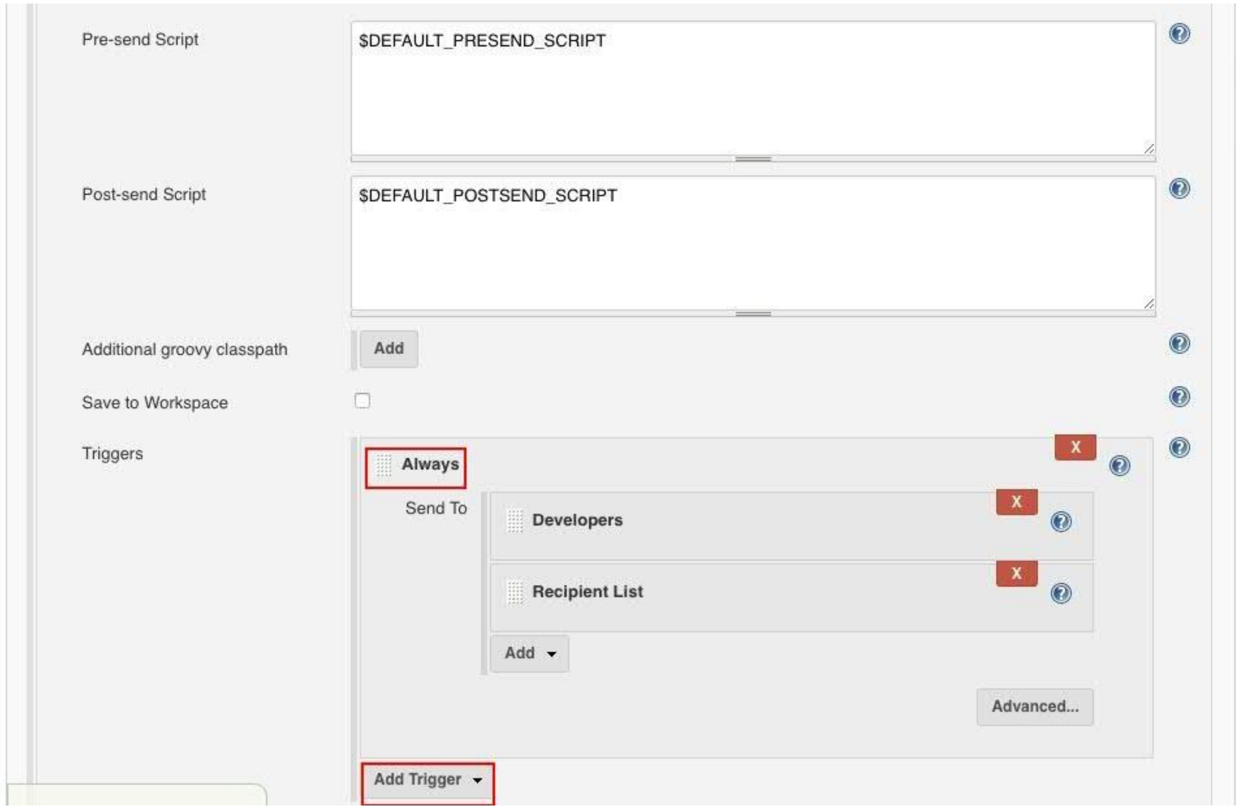
[ ] Enable watching for jobs

Content Token Reference

Step 3: In Job configure Editable Email as follows.

Select any Job, which we need to configure Email notification ---> Click on Configure ---> Select the **Post-build Actions** section.

Click on Advanced Settings …
It will expand and will show more settings and click on **Add Trigger** and select the **Always**.


Pre-send Script: $DEFAULT_PRESEND_SCRIPT

Post-send Script: $DEFAULT_POSTSEND_SCRIPT

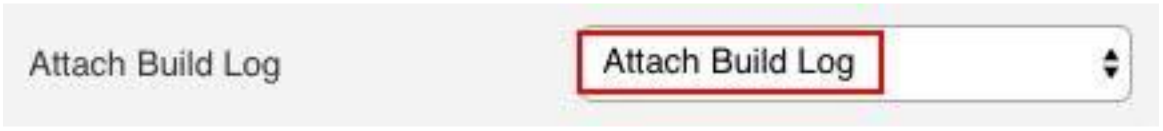
Additional groovy classpath

Save to Workspace

Triggers: Always

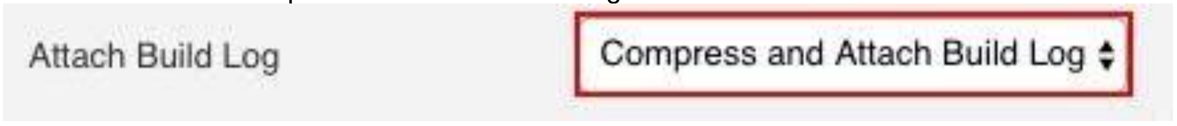
Send To: Developers, Recipient List

We can enable to attach the build logs while sending mail, as follows.


Attach Build Log: Attach Build Log

Output mail is like below.

We can enable to Compress and Attach Build Log to email as follows.

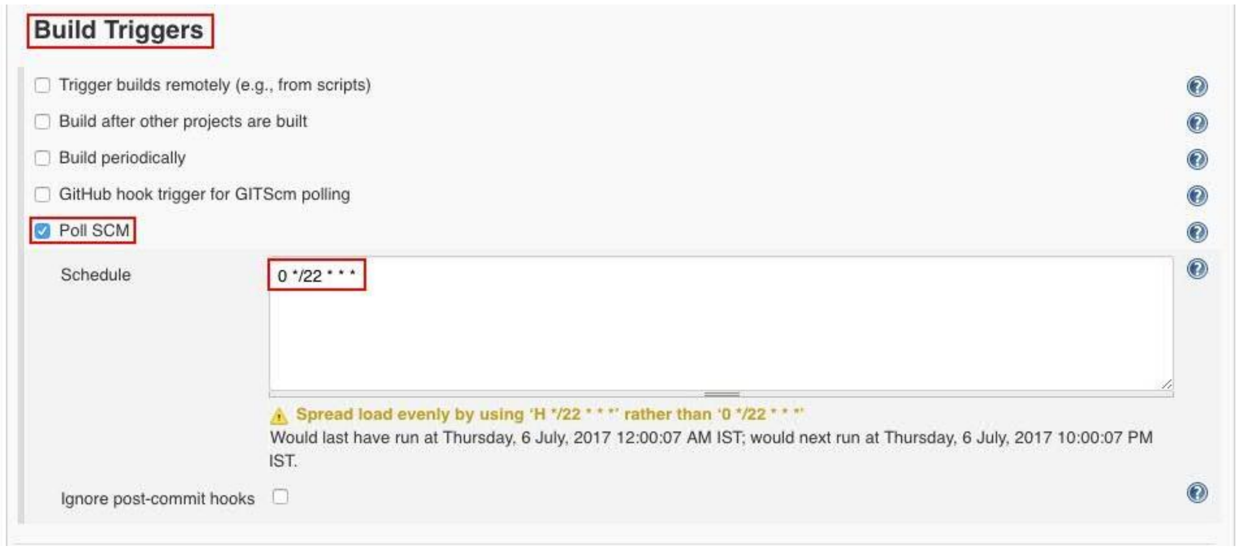

Attach Build Log: Compress and Attach Build Log

Output mail is like below.

How to enable the Poll SCM in Jenkins?

Step 1: Install the “**Git plugin**” in Jenkins.

Step 2: Select the job which you need to enable hook and click on Configure ---> In **Build Triggers** Section enable the **Poll SCM**
And provide the values as follows.


Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [✓] Poll SCM

Schedule: `0 */22 * * *`

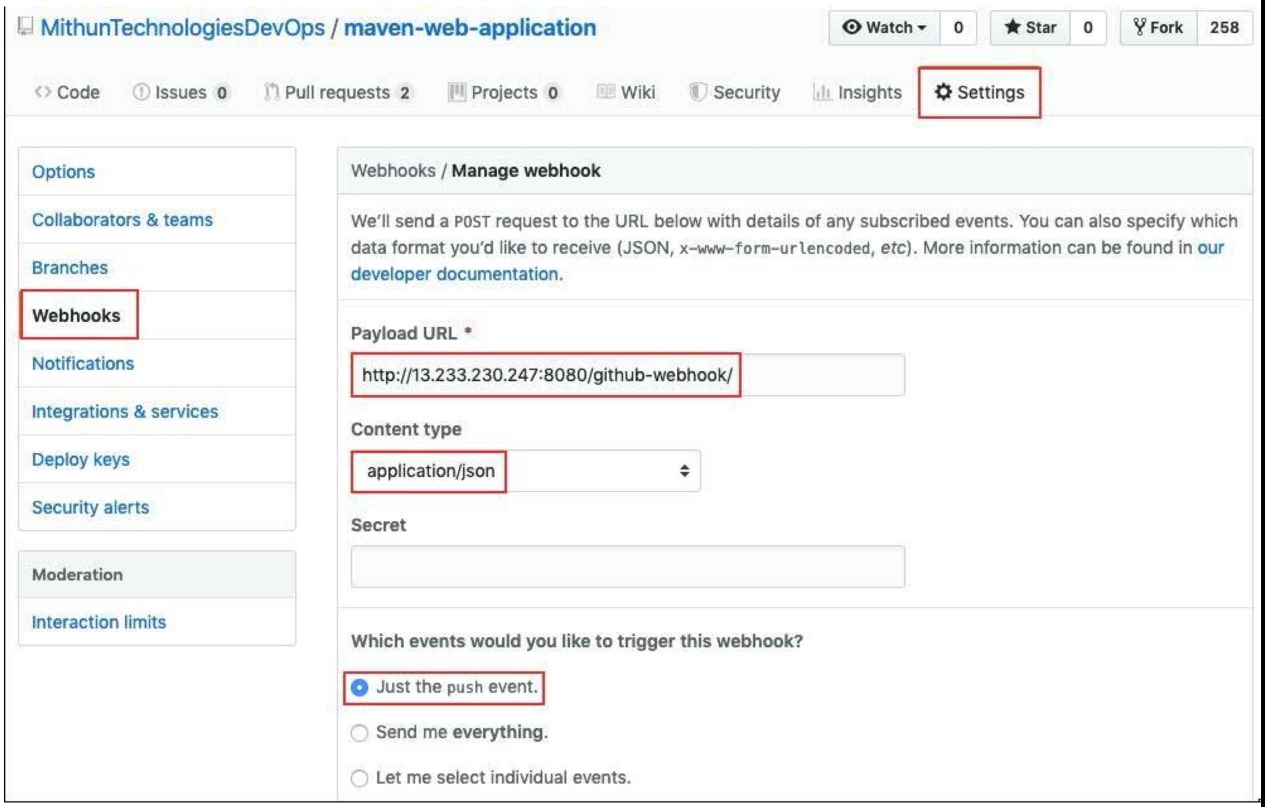
⚠ Spread load evenly by using 'H */22 * * *' rather than '0 */22 * * *'
Would last have run at Thursday, 6 July, 2017 12:00:07 AM IST; would next run at Thursday, 6 July, 2017 10:00:07 PM IST.

Ignore post-commit hooks [ ]

GitHub webhook
Login into GitHub and select the repository for which repo we need to enable.

- Open your repository on GitHub.
- Click '**Settings**' on the navigation bar on the right-hand side of the screen.
- Click '**Webhooks** ' on the navigation bar on the left-hand side of the screen.
- Click '**Add webhook**' to add the webhook.

Once you click on Add webhook url, it will ask the Payload URL, give the Jenkins url and Content type as follows.



Webhooks / **Manage webhook**

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More information can be found in our developer documentation.

Payload URL *: http://13.233.230.247:8080/github-webhook/

Content type: application/json

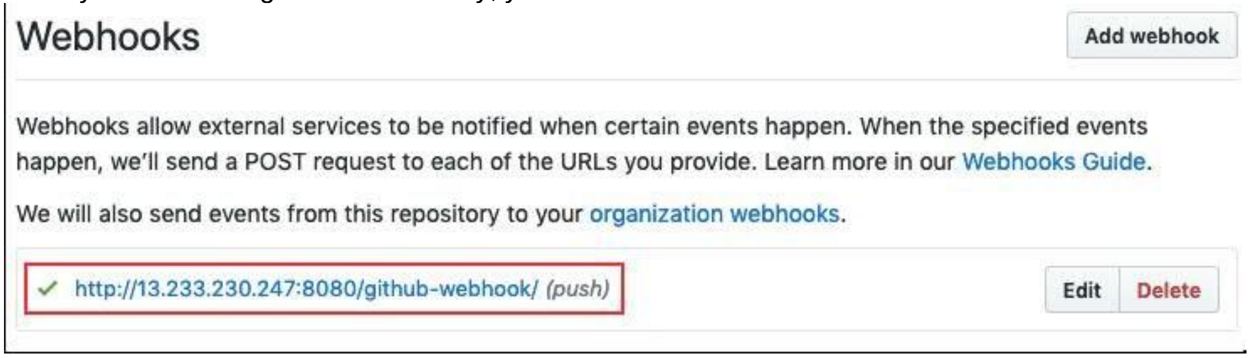
Secret:

Which events would you like to trigger this webhook?

- (●) Just the push event.
- ( ) Send me **everything**.
- ( ) Let me select individual events.

Click on ‘**Add webhook**’.

Once you have configured successfully, you will see as follows.



Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

We will also send events from this repository to your organization webhooks.

✓ http://13.233.230.247:8080/github-webhook/ *(push)*

Edit Delete

Configuring Jenkins Project : We now have Jenkins configured to run builds automatically when code is pushed to central repositories. However, Jenkins doesn’t run all builds for all projects. To specify which project builds need to run, we have to modify the project configuration.

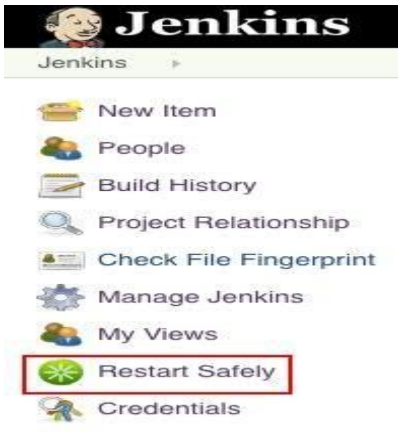
- In Jenkins, go to the **project configuration** of the project for which you want to run an automated build.
- In the ‘Build Triggers’ section, select **'Github hook trigger for GITScm Polling'.**
- **Save** your project.

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://13.233.230.247:8080/safeRestart
(jenkins_url)/restart - Forces a restart without waiting for builds to complete.
Ex: http://13.233.230.247:8080/restart

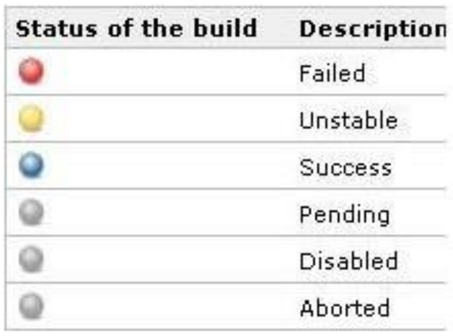
(OR)

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



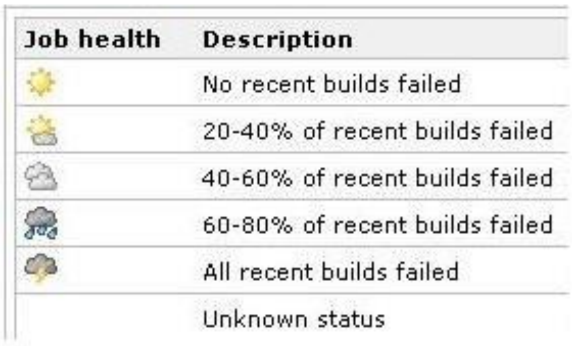
Disable Build:
A disabled Build will not be executed until you enable it again. This option often comes in handy to suspend a build during maintenance work or major refactoring.

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 Directory Structure

jenkins : This is the default Jenkins home directory (may be .hudson in older installations) and it will be placed in user's home directory (C:\Users\MITHUN_ADMIN\ ---> Windows & /Users/mithunreddy/ --> MAC and /var/lib/jenkins → Linux).

Jenkins home directory contains the below sub directories and configuration files (.xml).

```
+- jobs
  +- [JOBNAME]      :Sub directory for each job
     +- config.xml      : Job configuration file
     +- latest              : Symbolic link to the last successful build)
     +- builds
        +- [BUILD_ID]     : for each build one build id
           +- build.xml      : build result summary
           +- log               : log file
           +- changelog.xml  (change log)
+- logs          ()
+- nodes          ()
+- plugins       : This directory contains all the plugins that you have installed.
+- secrets       ()
+- updates          : This is an internal directory used by Jenkins to store information
                        about available plugin updates.

+- userContent    : You can use this directory to place your own custom content onto your Jenkins
                           server. You can access files in this directory at
http://localhost/jenkins/userContent (if
                           you are running Jenkins on an application server) or
                           http://localhost:8080/userContent (if you are running in stand-alone mode).
+- users              : If you are using the native Jenkins user database, user accounts will
                          be stored in this directory.

+- war                 : This directory contains the expanded web application. When you start
                          Jenkins as a stand-alone application, it will extract the web application into
                          this directory.

+- config.xml     (jenkins root configuration)
+- *.xml         (other site-wide configuration files)
+- fingerprints  (stores fingerprint records)
+-workspace: This directory contains all jobs source code.
```

---

http://localhost:8080/configure

Home directory: By default, Jenkins stores all of its data in this directory on the file system. Under the Advanced section, you can choose to store build workspaces and build records elsewhere.

There are a few ways to change the Jenkins home directory:

- Edit the JENKINS_HOME variable in your Jenkins configuration file (e.g. /etc/sysconfig/jenkins on Red Hat Linux).
- Use your web container's admin tool to set the JENKINS_HOME environment variable.

- Set the environment variable JENKINS_HOME before launching your web container, or before launching Jenkins directly from the WAR file.
- Set the JENKINS_HOME Java system property when launching your web container, or when launching Jenkins directly from the WAR file.
- Modify web.xml in jenkins.war (or its expanded image in your web container). This is not recommended.

This value cannot be changed while Jenkins is running.
It is shown here to help you ensure that your configuration is taking effect.

Ex: /Users/BhaskarReddy/.jenkins is for my Jenkins which is installed in my local MAC.

Workspace Root Directory: Specifies where Jenkins will store workspaces for builds that are executed on the master.

Build Record Root Directory: Specifies where Jenkins will store build records on the file system. This includes the console output and other metadata generated by a build.

System Message: This message will be displayed at the top of the Jenkins main page.

# of executors: It shows the ow many builds run at a time. E.g.: If give 2, here two builds are running.

Labels:

Usage: Controls how Jenkins schedules builds on this node.

Quiet period:

SCM checkout retry count:

Restrict project naming:

Naming Strategy

Strategy

Default ---> This is the default configuration and allows the user to choose any name they like.

Pattern ----> Define a pattern (regular expression) to check whether the job name is valid or not. Forcing the check on existing jobs, will allow you to enforce a naming convention on existing jobs - e.g. even if the user does not change the name, it will be validated with the given pattern at every submit and no updates can be made until the name confirms.

This option does not affect the execution of jobs with non-compliant names. It just controls the validation process when saving job configurations.

Global properties

Environment variables

Tool Locations
SonarQube servers

etc.…

To Install any Jenkins Plugin, follow below steps

Manage Jenkins ---> Manage Plugins ---> Select the Plugin name ---> Install Without Restart

Plugin Management

- Safe Restart
- Next Build Number
- Email Extension
- SonarQube Scanner
- Maven Integration
- Schedule Build
- Artifactory Plugin
- Cloud Foundry
- Blue Ocean
- Deploy to container
- Maven Integration
- JACOC
- SSH Agent
- Publish Over SSH
- ThinBackup
- Build Name Setter
- Convert To Pipeline
- **JobConfigHistory:** This plugin saves a copy of the configuration file of a job (config.xml) for every change made and of the system configuration. You can also see what changes have been made by which user if you configured a security policy.
- Repository browser
- Role-based Authorization Strategy:
- Slack Notification Plugin:
- Cobertura Plugin: In UI we will see as Coverage Trend.
- Hudson global-build-stats plugin:
- Delivery Pipeline View:
- Enable project-based security

Install Plugin using Jenkins CLI.

java -jar jenkins-cli.jar -s http://52.66.245.44:8080/ -auth mithuntechnologies:passw0rd install-plugin http://updates.jenkins-ci.org/download/plugins/audit-log/1.0/audit-log.hpi

---

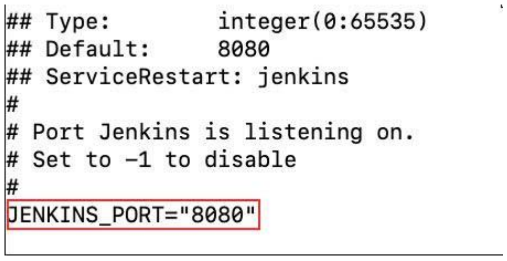
Port number change for Jenkins

By default, 8080 is the default port, change from 8080 something like 8082 as follow.

In Ubuntu update the below file.
#vi /etc/default/jenkin

then restart the service with below command.
service jenkins restart

In RHEL/CentOS update the below file.
#vi /etc/sysconfig/jenkins



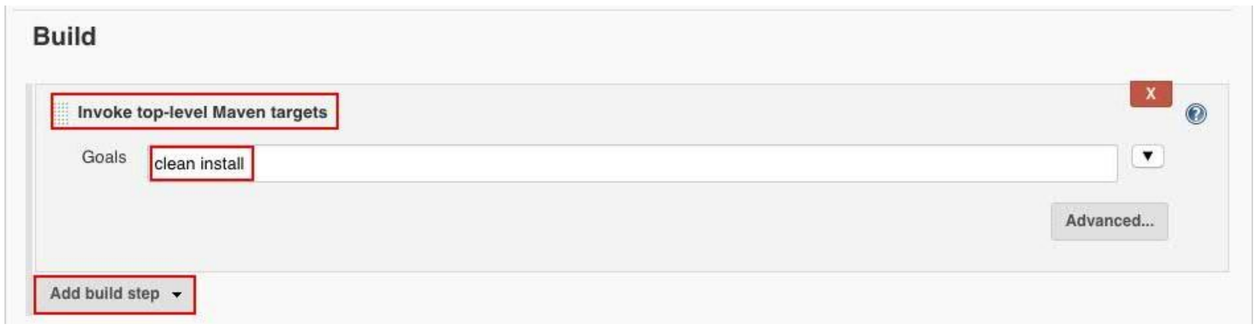
```
## Type:          integer(0:65535)
## Default:       8080
## ServiceRestart: jenkins
#
# Port Jenkins is listening on.
# Set to -1 to disable
#
JENKINS_PORT="8080"
```

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

Create the Maven project/job in Jenkins

Method 1:
Install the **Maven Integration Plugin** and follow the below steps.

Create the Job using Freestyle project and in the Build section click on Add build step and select the Invoke Top level Maven targets.



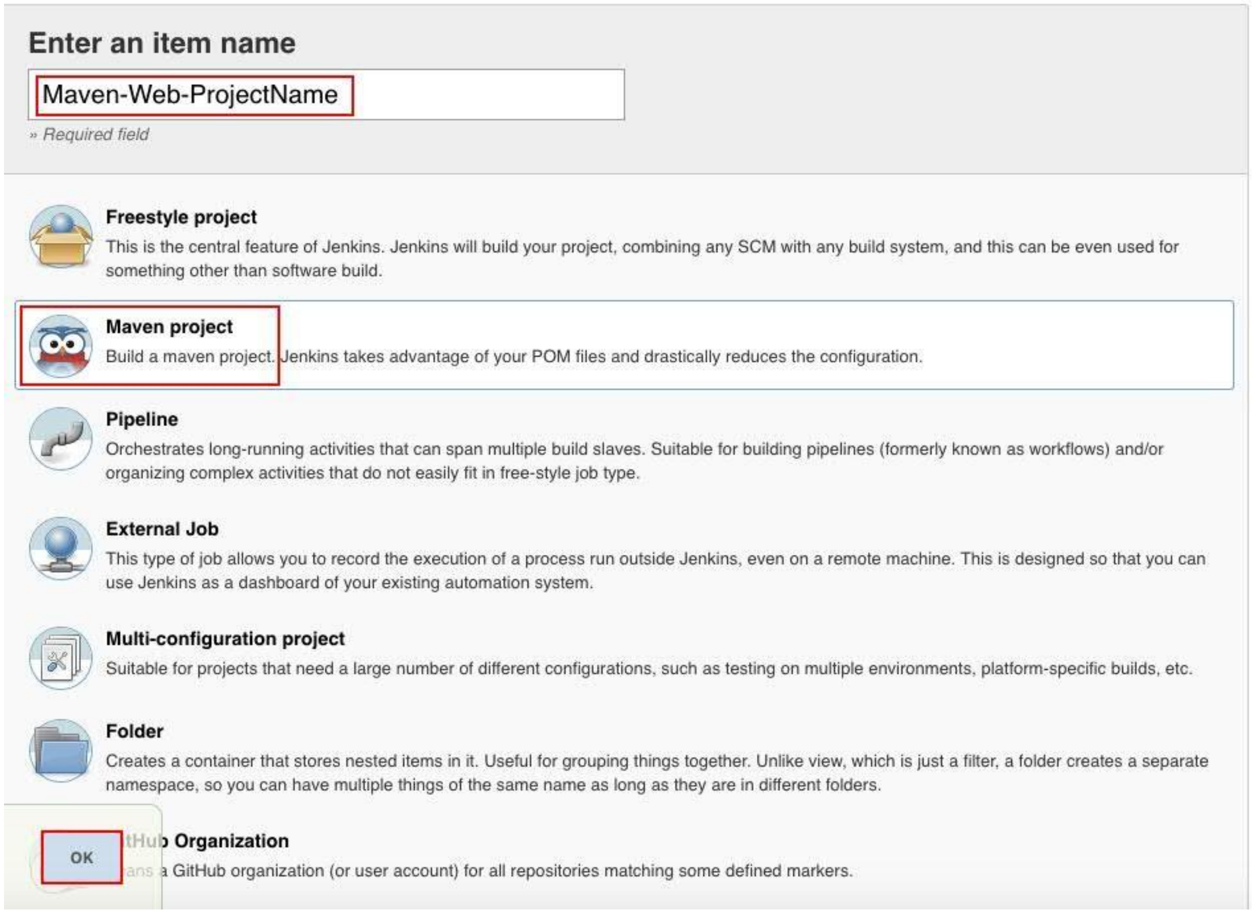
Build

Invoke top-level Maven targets

Goals: clean install

Method 2:
Install the **Maven Integration plugin** and follow the below steps.

Create the New Item as follows.
Provide the item name and select the Maven project and click on OK.



Enter an item name

Maven-Web-ProjectName

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

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

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

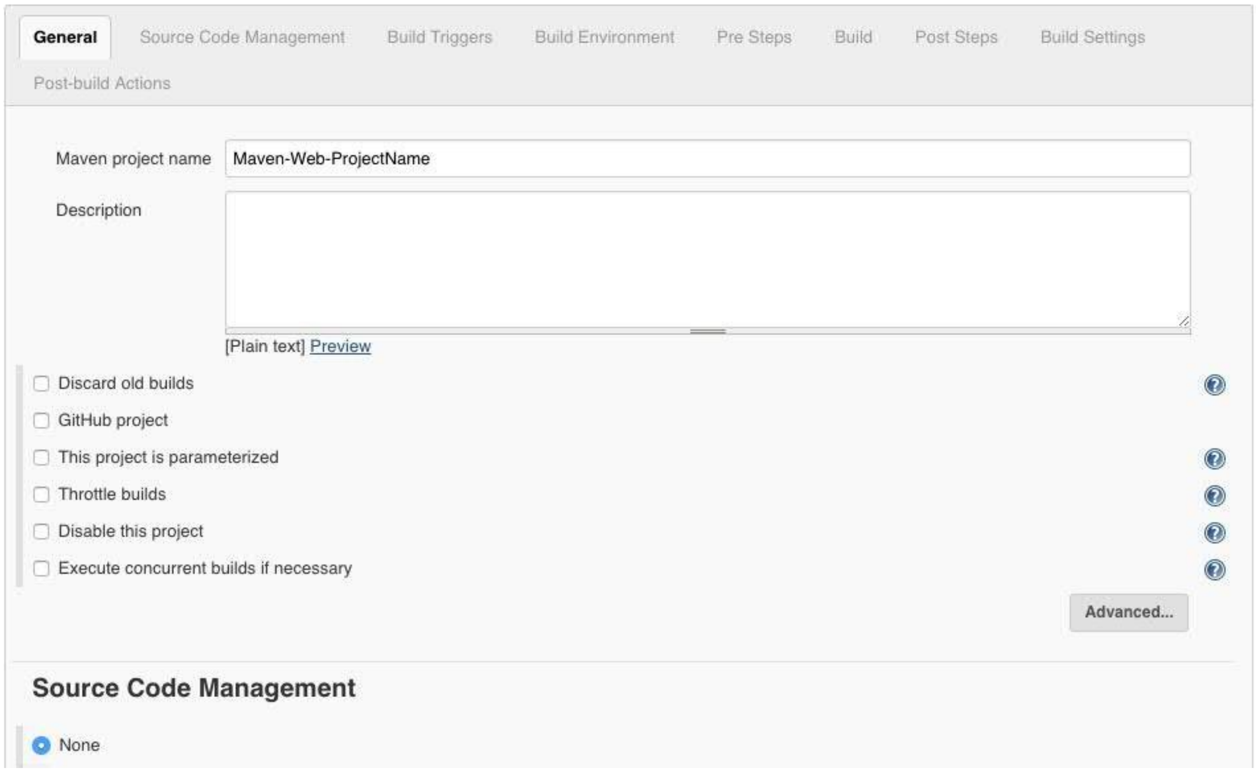
External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

tHub Organization
ans a GitHub organization (or user account) for all repositories matching some defined markers.

Once you click on OK, you will come to jobs configuration page as follows.



Maven project name: Maven-Web-ProjectName

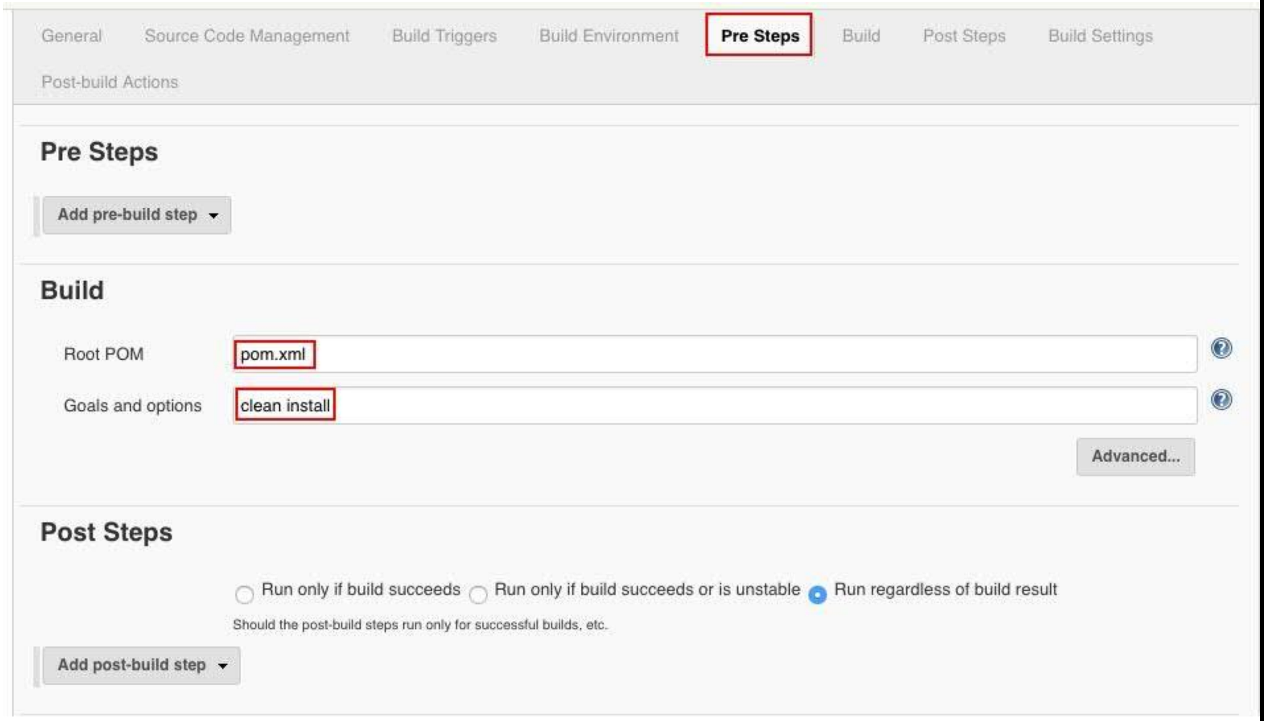
Description

[Plain text] Preview

- [ ] Discard old builds
- [ ] GitHub project
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Disable this project
- [ ] Execute concurrent builds if necessary

Source Code Management

● None



Pre Steps

Build

Root POM: pom.xml

Goals and options: clean install

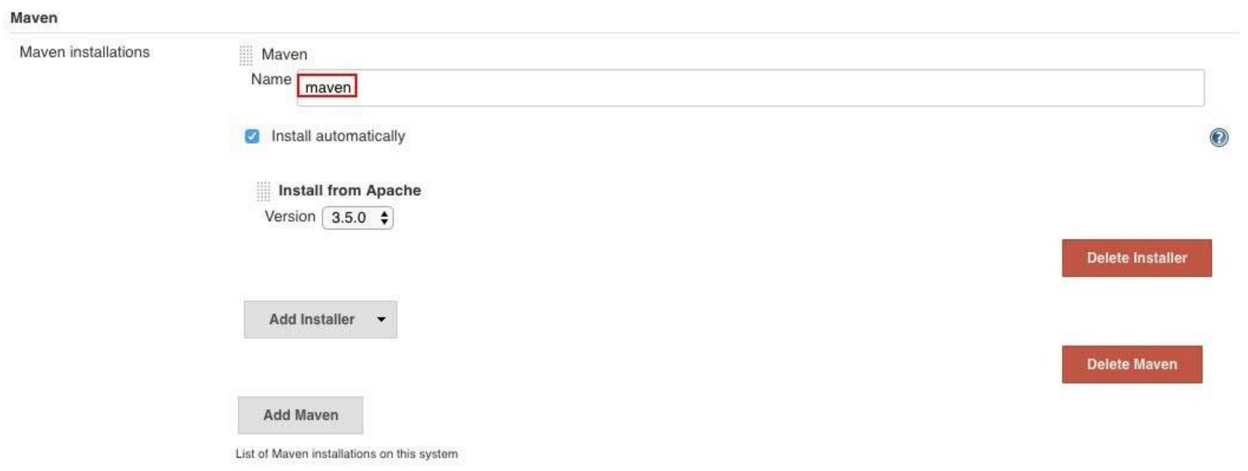
Post Steps

○ Run only if build succeeds ○ Run only if build succeeds or is unstable [✓ Run regardless of build result]

Should the post-build steps run only for successful builds, etc.

Once you provide all the details click on Save.

http://localhost:8080/configureTools/


Maven

Maven installations

Maven

Name: maven

☑ Install automatically

Install from Apache

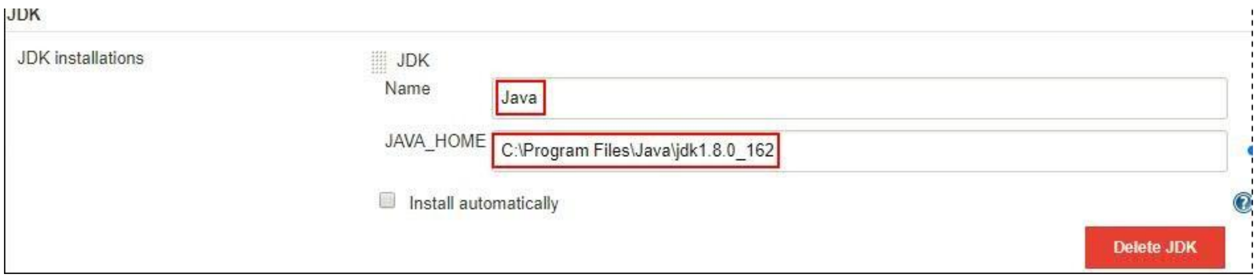
Version: 3.5.0

Possible Errors

```
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
```

Solution1
Set the class path for Java.

Solution2
Go to the Jenkins Dashboard ---> Click on Manage Jenkins ---> Global Tool Configuration ---> in **JDK** section give the full path where u have installed the Java.


JDK

JDK installations

JDK

Name: Java

JAVA_HOME: C:\Program Files\Java\jdk1.8.0_162

☐ Install automatically

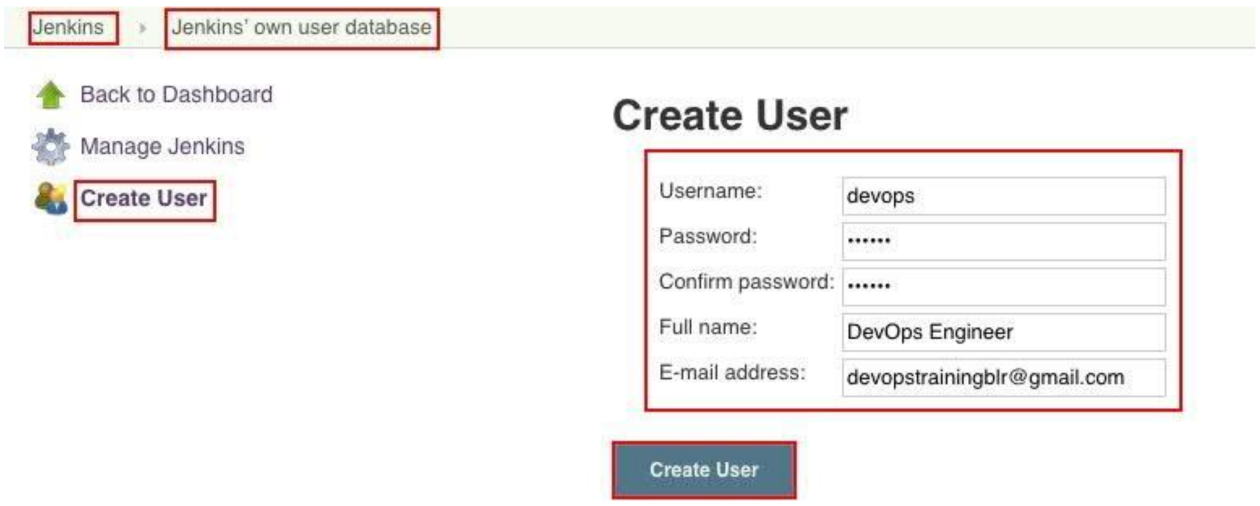
- -

Jenkins - Security

How to create the users in Jenkins?

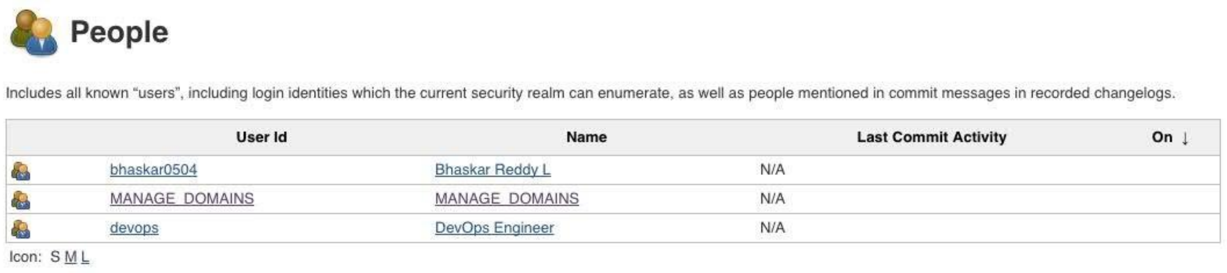
Click on Manage Jenkins ---> Manage Users ---> Create User ---> Provide the below details

Username:
Password:
Confirm password:
Full name:
E-mail address:
Click on Create User



Username:	devops
Password:	••••••
Confirm password:	••••••
Full name:	DevOps Engineer
E-mail address:	devopstrainingblr@gmail.com

How to see the list of Users in Jenkins?
Once you logged into Jenkins Dashboard
Go to Left Side Navigation Bar ---> Click on People
You will see list of users available in Jenkins.



Includes all known "users", including login identities which the current security realm can enumerate, as well as people mentioned in commit messages in recorded changelogs.

	User Id	Name	Last Commit Activity	On ↓
	bhaskar0504	Bhaskar Reddy L	N/A	
	MANAGE_DOMAINS	MANAGE_DOMAINS	N/A	
	devops	DevOps Engineer	N/A	

How to remove/delete the User in Jenkins?

Click on Manage Jenkins ---> Manage Users ---> click on below Gear icon one circle with cross symbol

It will ask Are you sure about deleting the user from Jenkins? confirmation message Click on ---> Yes

Now User is deleted successfully.

How to change the password for existing users?

Note: TBD

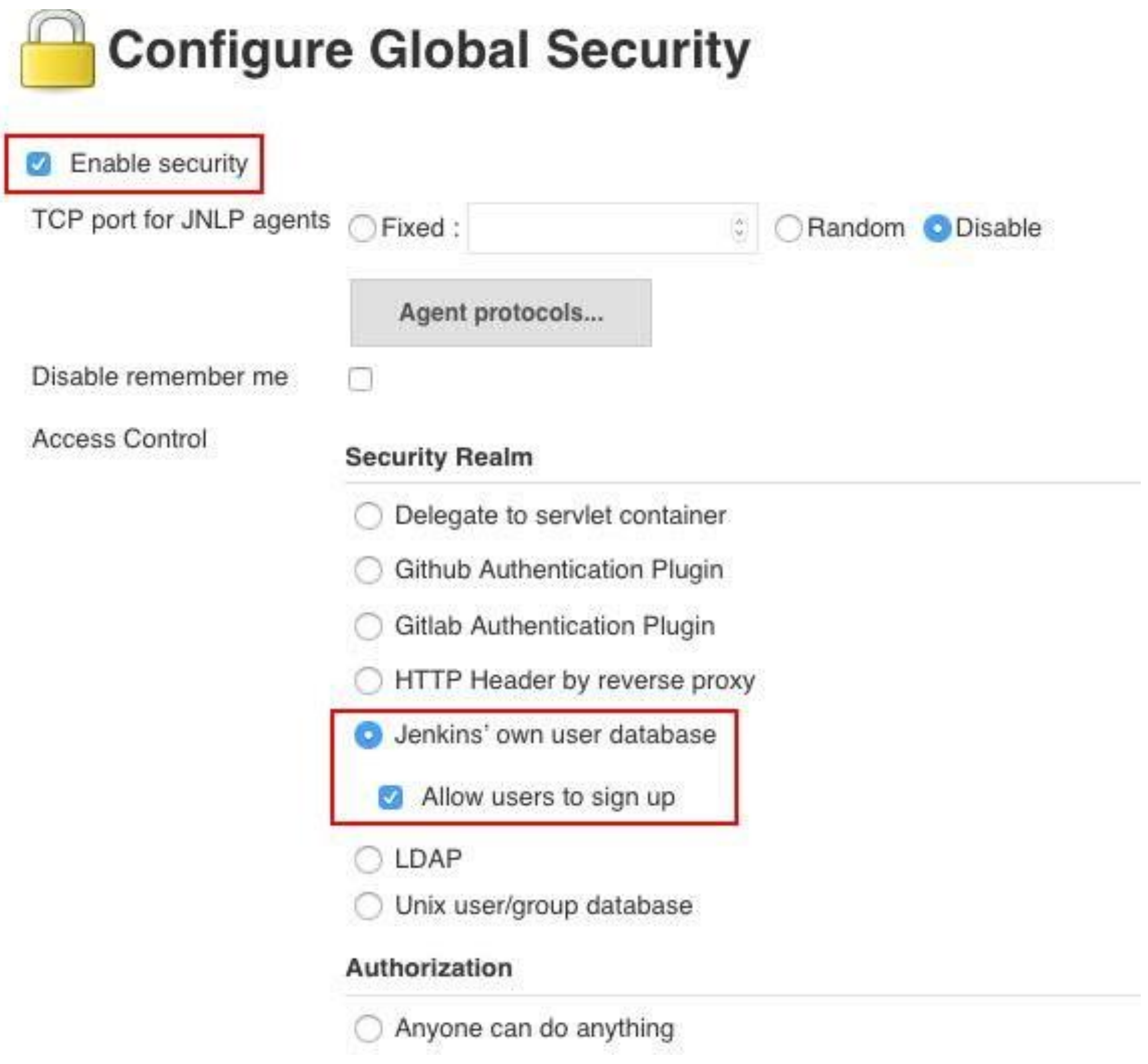
Project-based Matrix Authorization Strategy is an authorization method using which we can define which user or group can do what actions on which job. This gives us a fine-grained control over user/group permissions per project.

To Enable the Project-based Matrix Authorization Strategy need to configure in Jenkins as follows.

Step 1: Click on Manage Jenkins and choose the 'Configure Global Security' option.

Step 2: Click on Enable Security option.
As an example, let's assume that we want Jenkins to maintain it's own database of users, so in the Security Realm, Select the radio button of 'Jenkins' own user database'.

Step 3: Under Authorization, select "Project-based Matrix Authorization Strategy" and add 2 or 3 users, one administrator (say devops) and a regular user (say user1 and user2).



Configure Global Security

[✓] Enable security

TCP port for JNLP agents: ○ Fixed : ○ Random [● Disable]

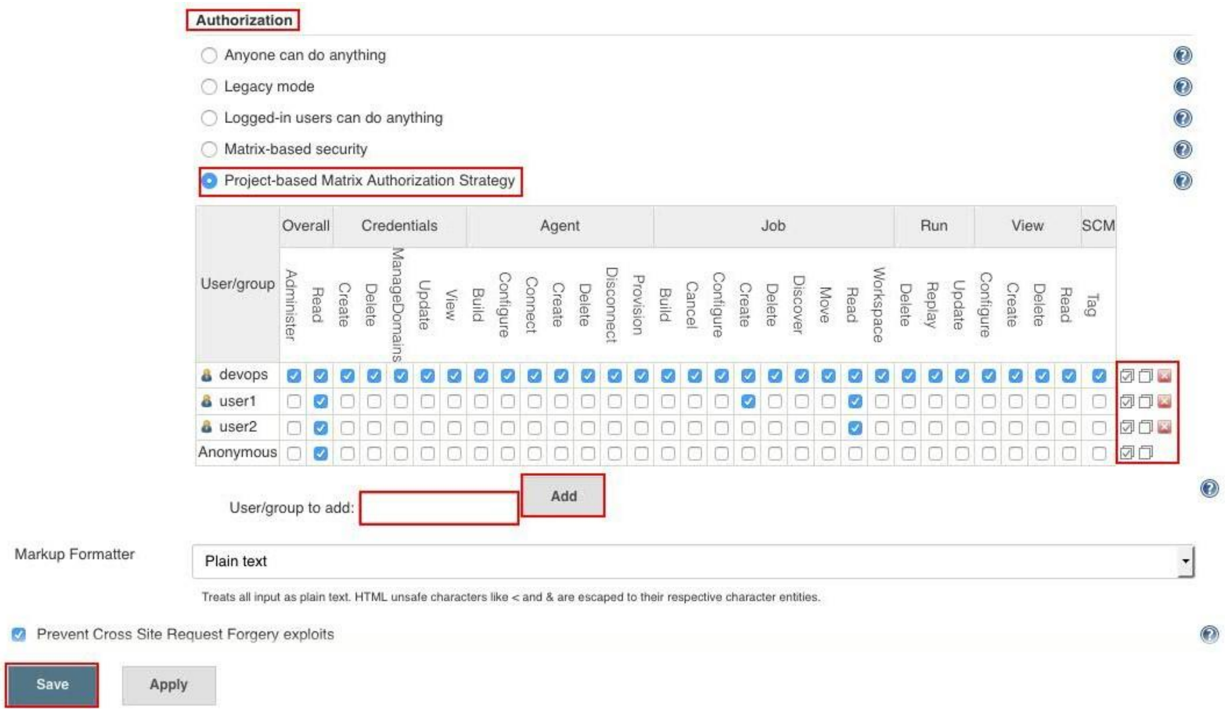
Agent protocols...

Disable remember me: [ ]

Access Control

Security Realm
- ○ Delegate to servlet container
- ○ Github Authentication Plugin
- ○ Gitlab Authentication Plugin
- ○ HTTP Header by reverse proxy
- [● Jenkins' own user database]
  - [✓] Allow users to sign up
- ○ LDAP
- ○ Unix user/group database

Authorization
- ○ Anyone can do anything



Authorization

Anyone can do anything

Legacy mode

Logged-in users can do anything

Matrix-based security

[✓ Project-based Matrix Authorization Strategy]

User/group	Overall		Credentials					Agent							Job									Run			View				SCM
	Administer	Read	Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag
devops	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
user1		✓																✓				✓									
user2		✓																				✓									
Anonymous		✓																													

User/group to add:

Add

Markup Formatter

Plain text

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

[✓] Prevent Cross Site Request Forgery exploits

Save

Apply

All the checkboxes present besides users are for setting global permissions. Select all checkboxes against admin user to give admin full permissions.

For user1, we are selecting read permissions under jobs. With this, user1 would now have read permission to view all jobs which we would be creating later on.

We have to provide read permission under “Overall” category to any regular user otherwise the user won’t be able to see anything after login.

All the checkboxes present besides users are for setting global permissions. Select all checkboxes against admin user to give admin full permissions. For user1, we are selecting read permissions under jobs. With this, user1 would now have read permission to view all jobs which we would be creating later on. We have to provide read permission under “Overall” category to any regular user otherwise the user won’t be able to see anything after login.

Finally, you can click on Save button.

Below scenario will applicable in Matrix based security

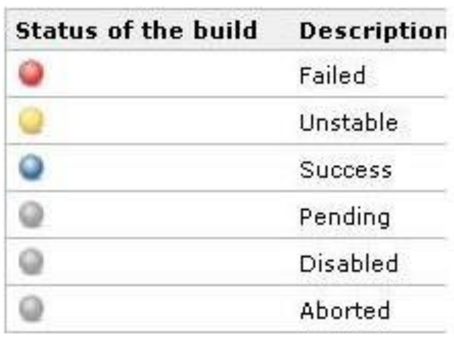
Error : Access Denied

<<User>> is missing the Overall/Read permission

If you get this error, Pease follow below steps.

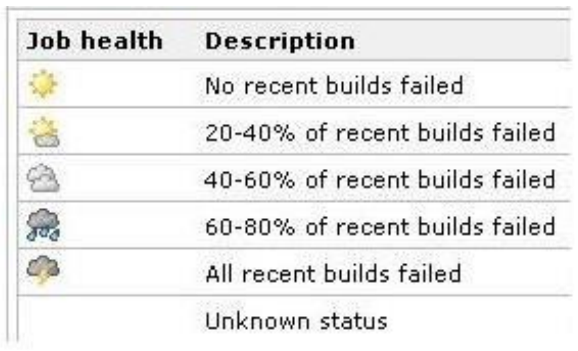
Solution:
Click on Manage Jenkins ---> Configure Global Security ---> User/group to add: Enter the user Name and click on Add button and --->
Enable the appropriate feature ---> Click on Save Button.

Jenkins Build Status Icon Colours



Status of the build	Description
🔴	Failed
🟡	Unstable
🔵	Success
⚪	Pending
⚪	Disabled
⚪	Aborted

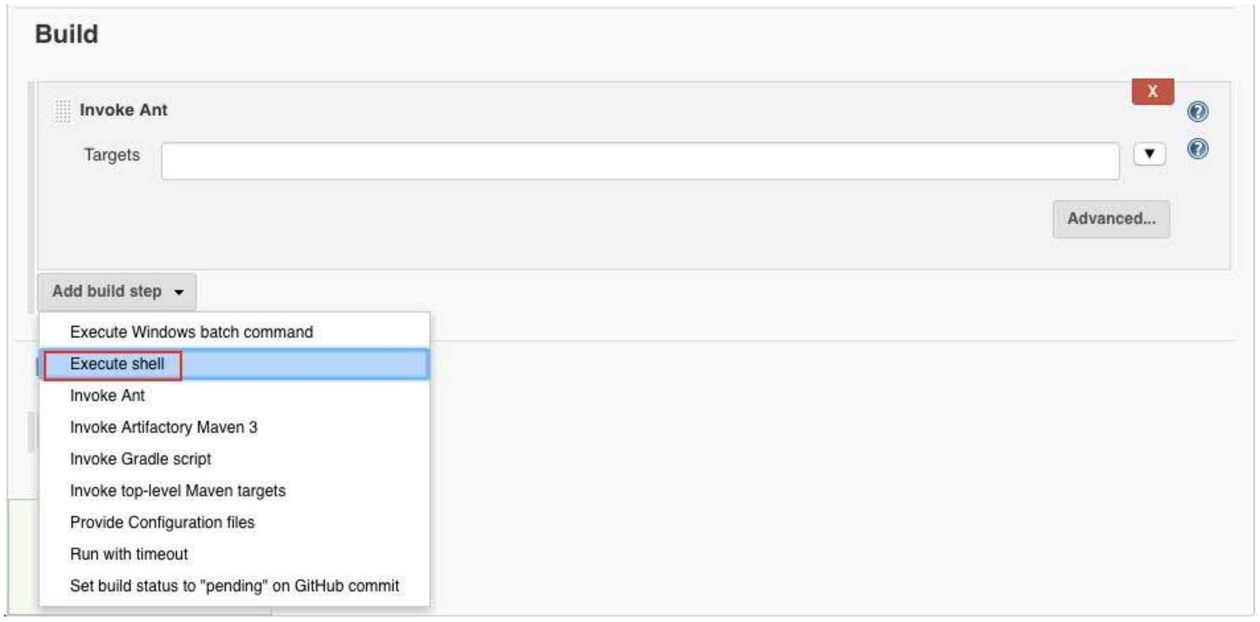
Figure a: Build status



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

Deploy the Application Through Script


Build
Invoke Ant
X
Targets
Advanced...
Add build step
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Artifactory Maven 3
Invoke Gradle script
Invoke top-level Maven targets
Provide Configuration files
Run with timeout
Set build status to "pending" on GitHub commit

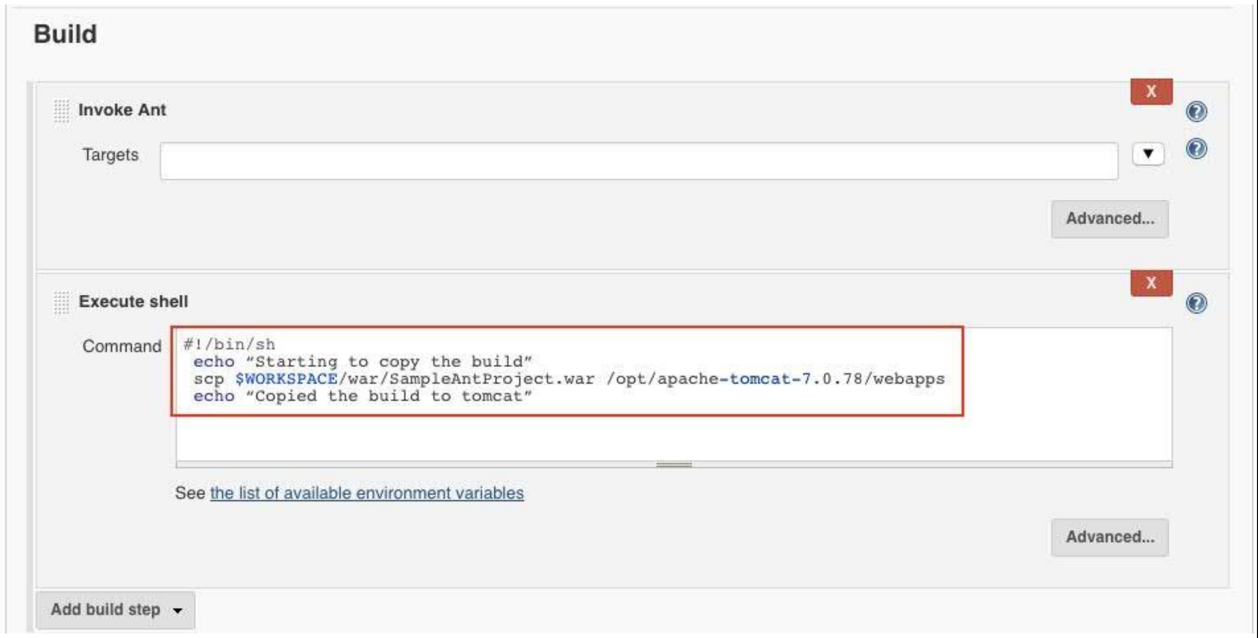
Add the below script in **Execute shell**

Linux/MAC for Tomcat
#!/bin/sh
echo "Starting to copy the build artifact"

cp $WORKSPACE/war/SampleAntProject.war /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/apache-tomcat-9.0.6/webapps/
echo "Deployed the build artifact into tomcat server successfully"

Windows
echo "Starting to copy the build"
copy %WORKSPACE%\war\SampleAntProject.war C:\\apache-tomcat-8.5.23\webapps\
echo "Copied the build to tomcat"

Linux/MAC for WIldFly
#Deploy in WildFly server
#!/bin/sh
 echo "Starting to copy the build"
 cp $WORKSPACE/war/SampleAntProject.war /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/wildfly11.0.0.Final/standalone/deployments/
 echo "Copied the build to WildFly successfully"



```
#!/bin/sh
 echo "Starting to copy the build"
 scp $WORKSPACE/war/SampleAntProject.war /opt/apache-tomcat-7.0.78/webapps
 echo "Copied the build to tomcat"
```

Note: If we want to deploy in Tomcat, which is installed in any remote machine, use below lines of code.

scp $WORKSPACE/war/SampleAntProject.war <<User Name>>@<<ServerIP>>:/opt/apache-tomcat-7.0.78/webapps

- -

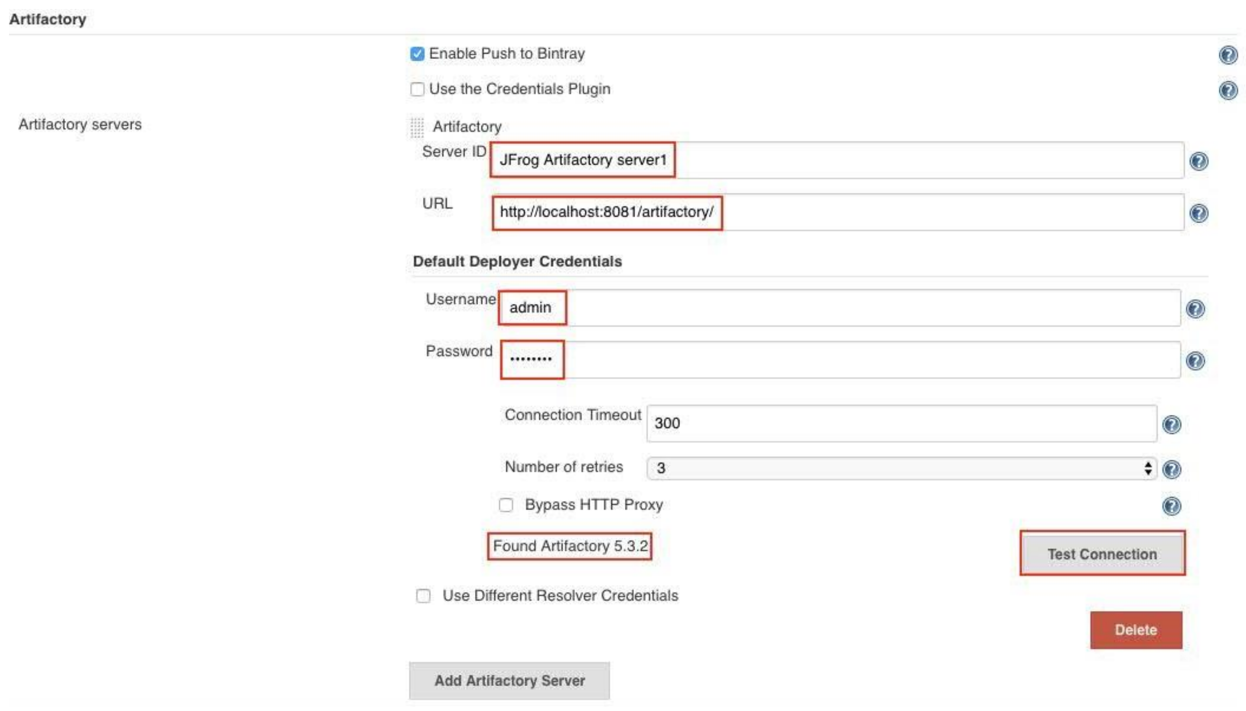
cp %JENKINS_HOME%\jobs\%JOB_NAME%\builds\%BUILD_NUMBER%\log C:\Users\windows7\Downloads\newfolder\

- -

Integrate JFrog Artifactory with Jenkins

Install **"Artifactory Plugin"** plugin.

Got to the Manage Jenkins ---> Configure System --->
In the **Artifactory** section fill the below details and click on Save.



Server ID: JFrog Artifactory server1
URL: http://localhost:8081/artifactory/
Username: admin
Password: ••••••••
Connection Timeout: 300
Number of retries: 3
Found Artifactory 5.3.2

Note: Once you entered all the details click on **TEST CONNECTION**. IF connection is succeeded you will see the message like **Found Artifactory <<Version>>**.

Jenkins – Metrics and Trends

There are various plugins which are available in Jenkins to showcase metrics for builds which are carried out over a period of time. These metrics are useful to understand your builds and how frequently they fail/pass over time. As an example, let's look at the '**Build History Metrics plugin**'. This plugin calculates the following metrics for all of the builds once installed

Mean Time To Failure (MTTF)
Mean Time To Recovery (MTTR)
Standard Deviation of Build Times

Enable LDAP security to Jenkins

http://www.scmgalaxy.com/tutorials/complete-guide-to-use-jenkins-cli-command-line

Jenkins CLI

Jenkins has a built-in command line interface (CLI) 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.

Advantages of Jenkins CLI:

- Easier
- Faster
- Memory management
- Automation tasks.

Pre-Requisites

a) Jenkins server should run.

b) Enable security as follows.

Go to Jenkins dashboard in Home page (e.g http://localhost:8080/) -> Manage Jenkins

-> Configure Global Security -> Click on "Enable security" checkbox

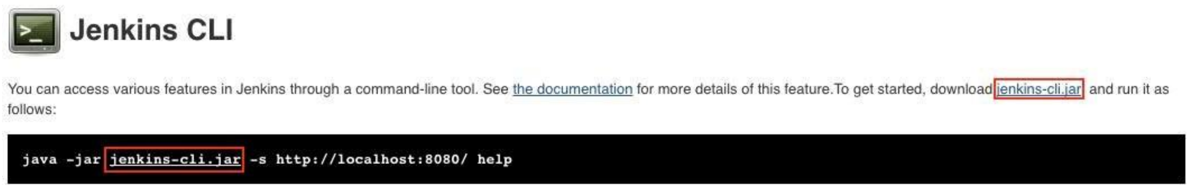
You can also configure "Access Control" and "Authorization" option in Global Security page.

Download the Jenkins CLI jar file as follows.

Method 1

Open the below url

http://localhost:8080/cli/



Jenkins CLI

You can access various features in Jenkins through a command-line tool. See the documentation for more details of this feature.To get started, download jenkins-cli.jar and run it as follows:

```
java -jar jenkins-cli.jar -s http://localhost:8080/ help
```

Click on Jenkins-cli.jar.

Method 2

Click on below url, it will automatically download the jar file.

http://<<Jenkins Server URL>>/jnlpJars/jenkins-cli.jar

Example: http://localhost:8080/jnlpJars/jenkins-cli.jar

Here

Copy into any folder as follows

```
#cp jenkins-cli.jar /opt/jenkins/
```

Go to the directory where Jenkins-cli.jar is there and run the below command to get the help.

Login Jenkins using username and Password

```
# java -jar jenkins-cli.jar -s http://localhost:8080/  -auth mithuntechnologies:passw0rd help
```

Get the Version of Jenkins

```
#java -jar jenkins-cli.jar -s http://localhost:8080/  -auth mithuntechnologies:passw0rd version
```

Get all the jobs of Jenkins

```
#java -jar jenkins-cli.jar -s http://localhost:8080/  -auth mithuntechnologies:passw0rd version  list-jobs
```

Delete the Job

```
#java -jar jenkins-cli.jar -s http://localhost:8080/ -auth mithuntechnologies:passw0rd version delete-job ant-java-job-dev
```

```
#java -jar jenkins-cli.jar -s http://localhost:8080/  -auth mithuntechnologies:passw0rd version  disable-job ant-web-job-dev
```

Jenkins Pipeline Project

In Jenkins Pipeline project, we will use one file called Jenkinsfile, in this file we will write groovy code to build process.
We will write Jenkinsfile in 2 ways.
1) Declarative way
2) Scripted way.

1) **Scripted Pipeline Syntax**
2) **Declarative Pipeline Syntax**

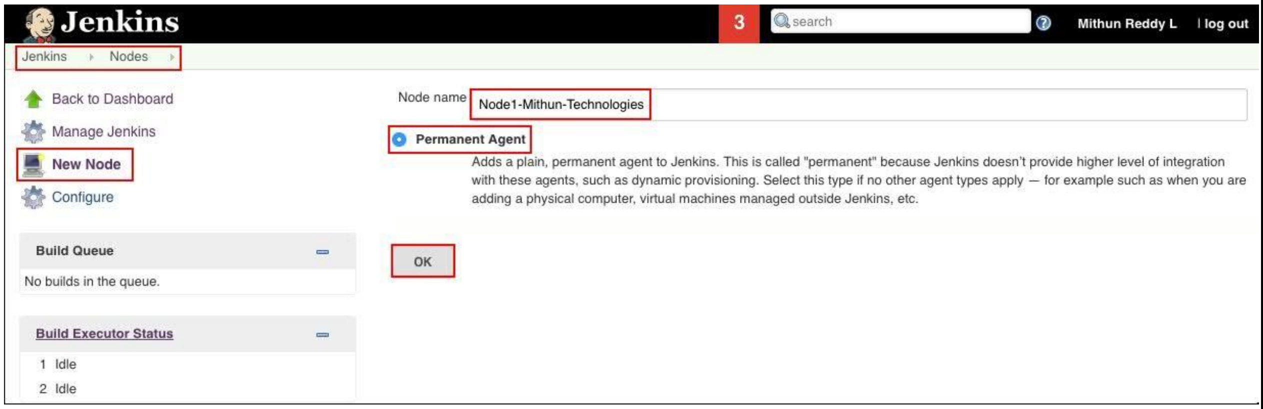
Jenkins Multi Branch Pipeline Project

Required Plugins
1) Pipeline: Multibranch

Blue Ocean Plugin

Jenkins Master-Slave setup

Manage Jenkins ---> Manage Nodes ---> New Node
Provide the Node name and click on **OK** button.



Node name: Node1-Mithun-Technologies

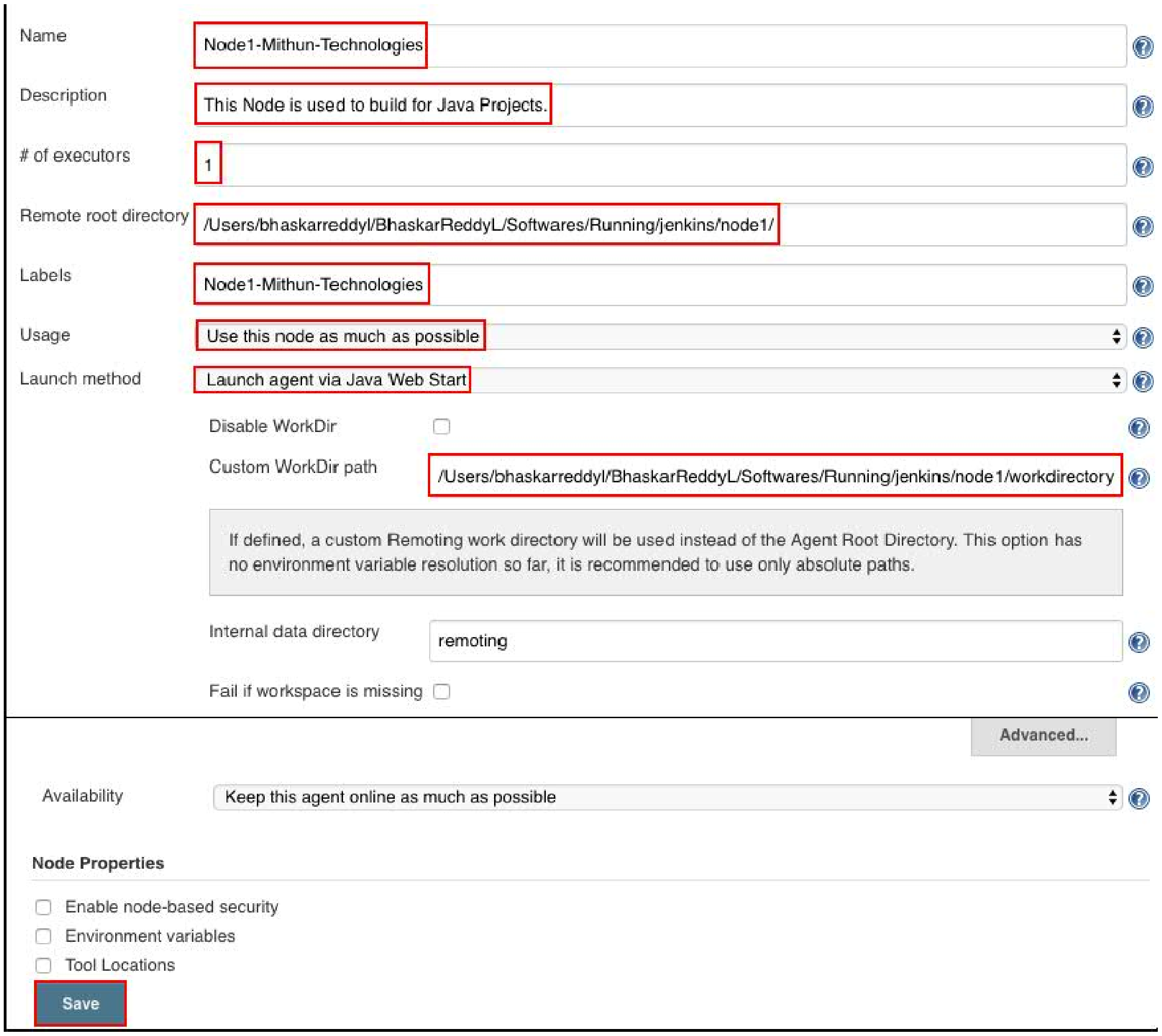
Permanent Agent

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.

No builds in the queue.

1 Idle
2 Idle

Provide all the details as follows and click on **Save** button.



Name	Node1-Mithun-Technologies
Description	This Node is used to build for Java Projects.
# of executors	1
Remote root directory	/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/
Labels	Node1-Mithun-Technologies
Usage	Use this node as much as possible
Launch method	Launch agent via Java Web Start
Disable WorkDir	☐
Custom WorkDir path	/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory
Internal data directory	remoting
Fail if workspace is missing	☐
Availability	Keep this agent online as much as possible

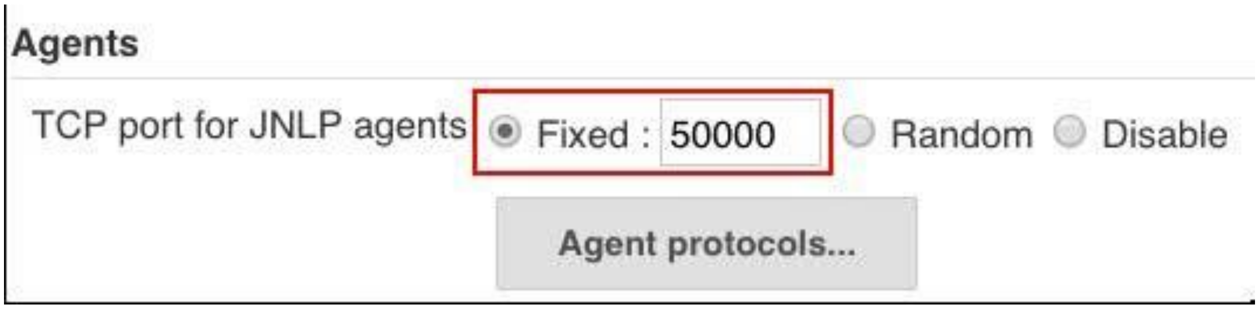
If defined, a custom Remoting work directory will be used instead of the Agent Root Directory. This option has no environment variable resolution so far, it is recommended to use only absolute paths.

Node Properties

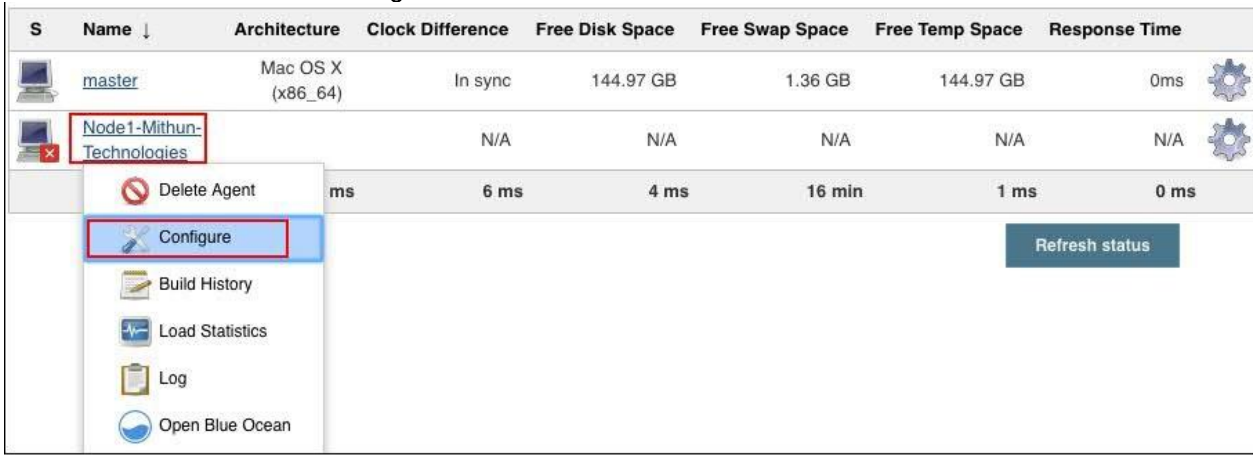
- ☐ Enable node-based security
- ☐ Environment variables
- ☐ Tool Locations

Note: Suppose if you don't see "**Launch agent via Java Web Start**" option, do the below configurations.

Manage Jenkins ---> Configure Global Security ---> enable the **TCP port for JNLP agents**
(by default, it is Disabled.)

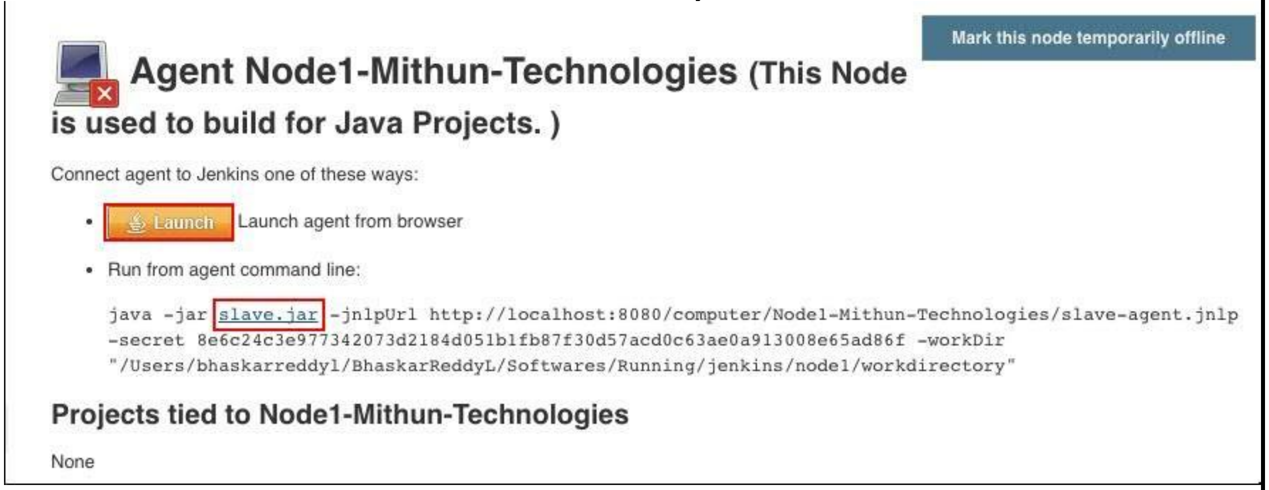


Once you click on Save you will see the Nodes and Master detail, and select the Node which we have created and click on configure.



S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Mac OS X (x86_64)	In sync	144.97 GB	1.36 GB	144.97 GB	0ms
	Node1-Mithun-Technologies		N/A	N/A	N/A	N/A	N/A
		ms	6 ms	4 ms	16 min	1 ms	0 ms

You will see below screen and click download the slave.jar file.



Agent Node1-Mithun-Technologies (This Node is used to build for Java Projects.)

Connect agent to Jenkins one of these ways:

- Launch agent from browser
- Run from agent command line:

```
java -jar slave.jar -jnlpUrl http://localhost:8080/computer/Node1-Mithun-Technologies/slave-agent.jnlp
-secret 8e6c24c3e977342073d2184d051b1fb87f30d57acd0c63ae0a913008e65ad86f -workDir
"/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory"
```

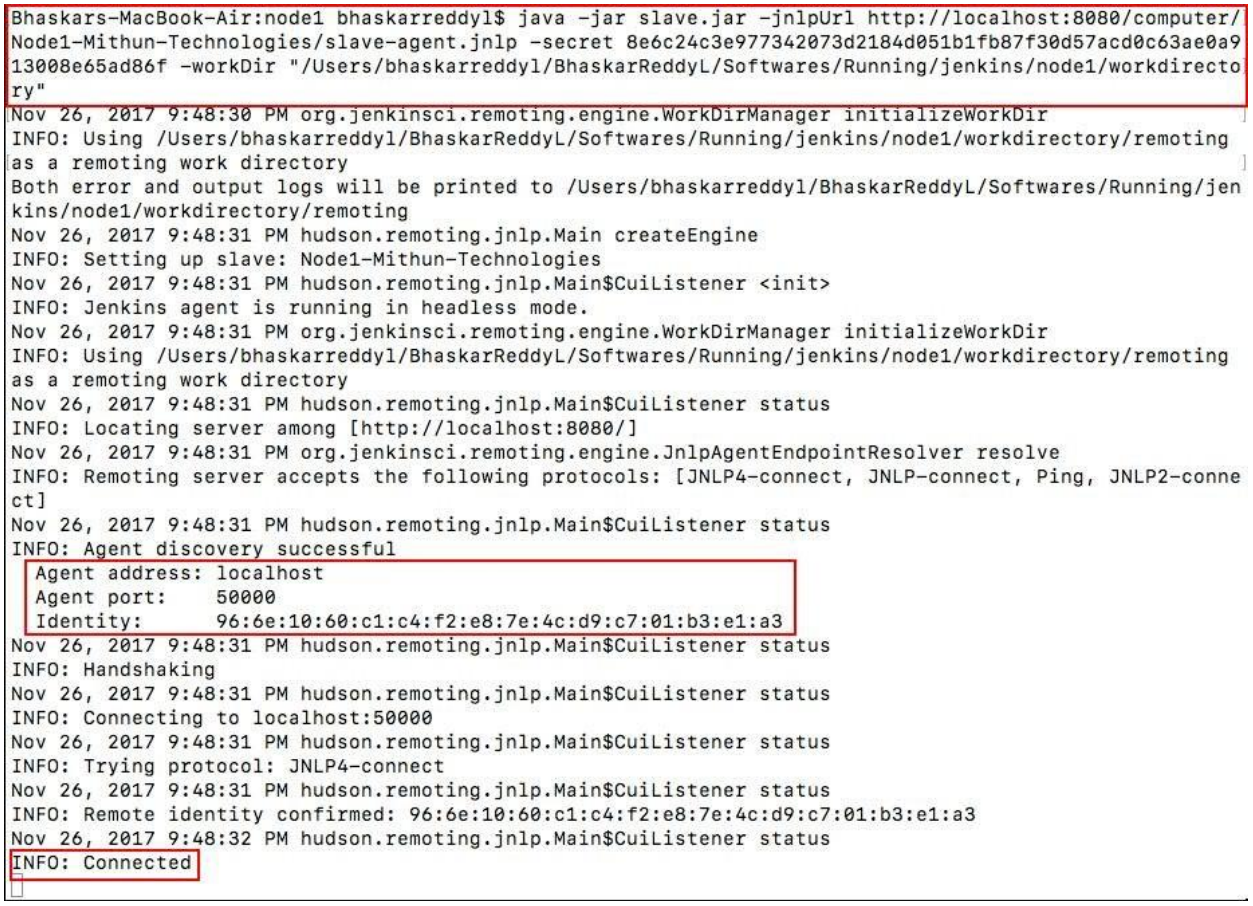
Projects tied to Node1-Mithun-Technologies

None

Copy slave.jar file into any directory
(/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1)

Go to the path where slave.jar copied and run the below command.

java -jar agent.jar -jnlpUrl http://localhost:8080/computer/Node1-Mithun-Technologies/slave-agent.jnlp -secret 8e6c24c3e977342073d2184d051b1fb87f30d57acd0c63ae0a913008e65ad86f -workDir "/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory"

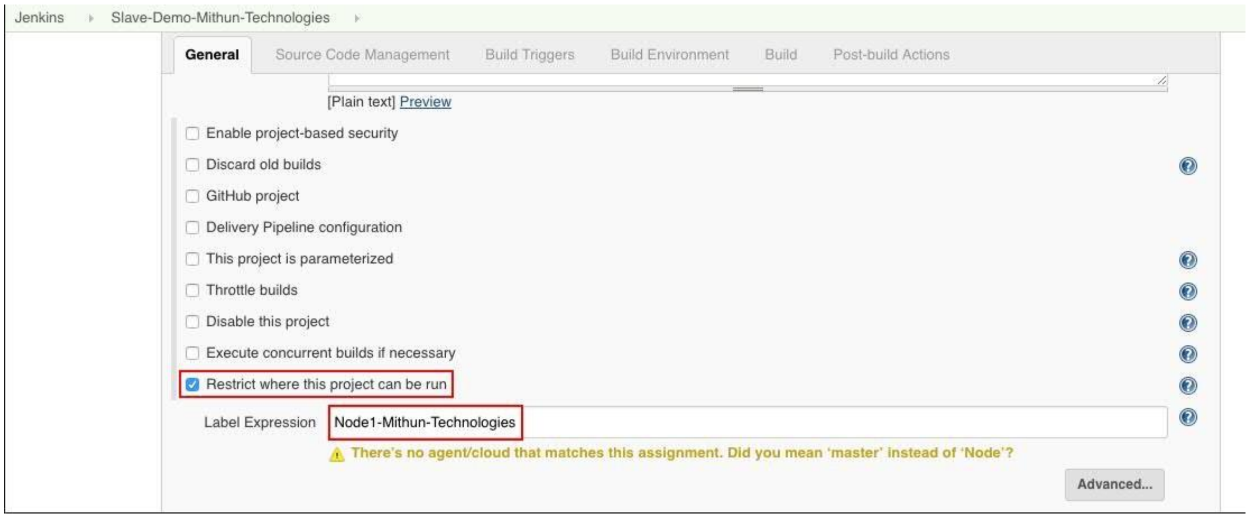


```
Bhaskars-MacBook-Air:node1 bhaskarreddyl$ java -jar slave.jar -jnlpUrl http://localhost:8080/computer/
Node1-Mithun-Technologies/slave-agent.jnlp -secret 8e6c24c3e977342073d2184d051b1fb87f30d57acd0c63ae0a9
13008e65ad86f -workDir "/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirecto
ry"
Nov 26, 2017 9:48:30 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory/remoting
as a remoting work directory
Both error and output logs will be printed to /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jen
kins/node1/workdirectory/remoting
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main createEngine
INFO: Setting up slave: Node1-Mithun-Technologies
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener <init>
INFO: Jenkins agent is running in headless mode.
Nov 26, 2017 9:48:31 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory/remoting
as a remoting work directory
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://localhost:8080/]
Nov 26, 2017 9:48:31 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, JNLP-connect, Ping, JNLP2-conne
ct]
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: localhost
  Agent port:    50000
  Identity:      96:6e:10:60:c1:c4:f2:e8:7e:4c:d9:c7:01:b3:e1:a3
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to localhost:50000
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Remote identity confirmed: 96:6e:10:60:c1:c4:f2:e8:7e:4c:d9:c7:01:b3:e1:a3
Nov 26, 2017 9:48:32 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connected
```

Now slave become communicating to node and it is live.

Now you can use this slave for job creation.

Create one Freestyle project/any kind of project and select the Restrict where this project can be run and select the Node which you have crated.



- [ ] Enable project-based security
- [ ] Discard old builds
- [ ] GitHub project
- [ ] Delivery Pipeline configuration
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Disable this project
- [ ] Execute concurrent builds if necessary
- [x] Restrict where this project can be run

Label Expression: Node1-Mithun-Technologies

There's no agent/cloud that matches this assignment. Did you mean 'master' instead of 'Node'?

Provide the Git url and click on **Save** button.

- -

Jenkins Home Directory Change in RHEL 7.5 Version

By Default, Jenkins home directory will be in /var/lib/jenkins in RHEL.

We can change the Jenkins default home directory to your custom directory(/opt/mithuntechnologies/jenkins).

Stop the Jenkins service if it is running.

sudo su -
service jenkins status
service jenkins stop

Create a directory mithuntechnologies in opt dir as follows.

#mkdir -p /opt/mithuntechnologies

## Copy the jenkins dir to

cp -r /var/lib/jenkins/ /opt/mithuntechnologies/

##Change the ownership as follows.

chown -R jenkins:jenkins /opt/mithuntechnologies/jenkins/

##Change the permissions as follows.

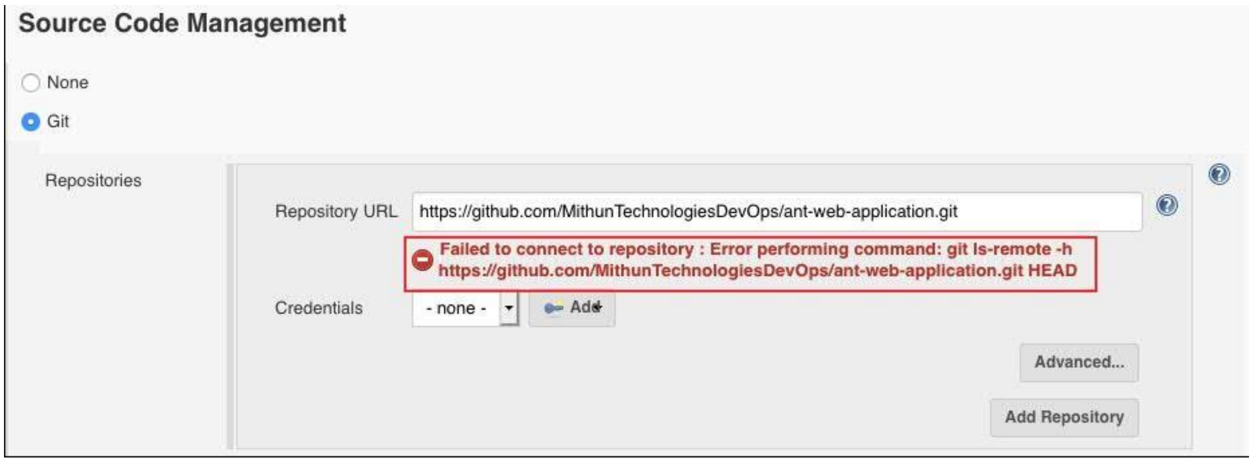
chmod -R 775 /opt/mithuntechnologies/jenkins/

##Start the jenkins service as follows.

service jenkins start

Possible Errors and Solutions:

Issue:


Source Code Management

None

[✓ Git]

Repositories

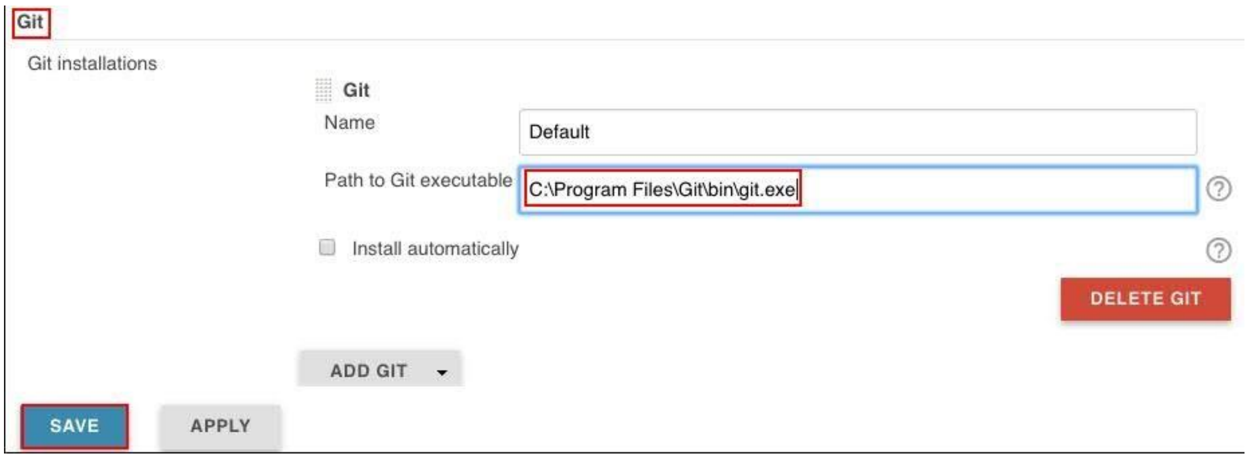
Repository URL: https://github.com/MithunTechnologiesDevOps/ant-web-application.git

Failed to connect to repository : Error performing command: git ls-remote -h https://github.com/MithunTechnologiesDevOps/ant-web-application.git HEAD

Credentials: - none -

Solution – Windows OS

Go to the Jenkins dashboard, Click on Manage Jenkins -→ Global Tool Configuration
In Git option,
Give the Gitbash installed path in **Path to Git executable** text filed as follows.


Git

Git installations

Git

Name: Default

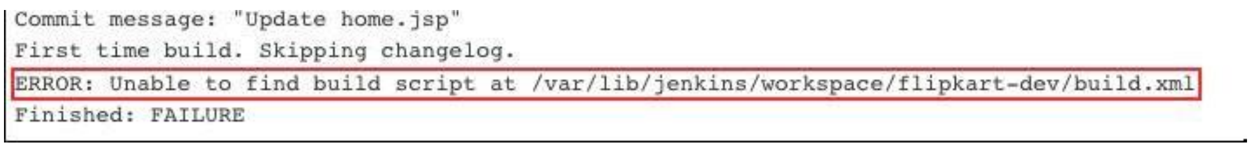
Path to Git executable: C:\Program Files\Git\bin\git.exe

Install automatically

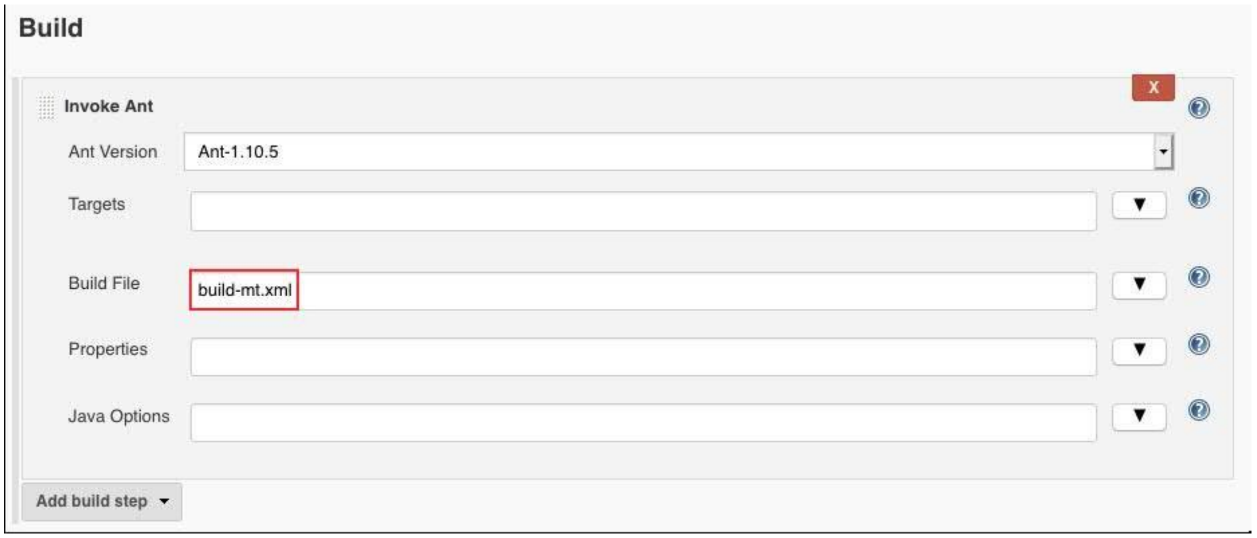
Solution – Linux

Install the git.

Issue:


```
Commit message: "Update home.jsp"
First time build. Skipping changelog.
ERROR: Unable to find build script at /var/lib/jenkins/workspace/flipkart-dev/build.xml
Finished: FAILURE
```

In Build step, give the build file name as in below screen shot.


Build

Invoke Ant

Ant Version: Ant-1.10.5

Targets:

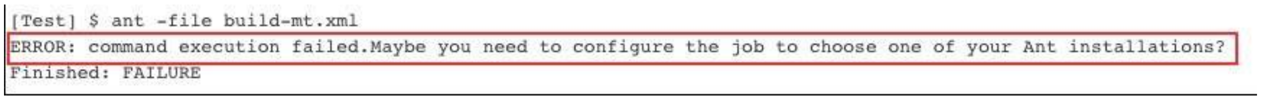
Build File: build-mt.xml

Properties:

Java Options:

Issue:

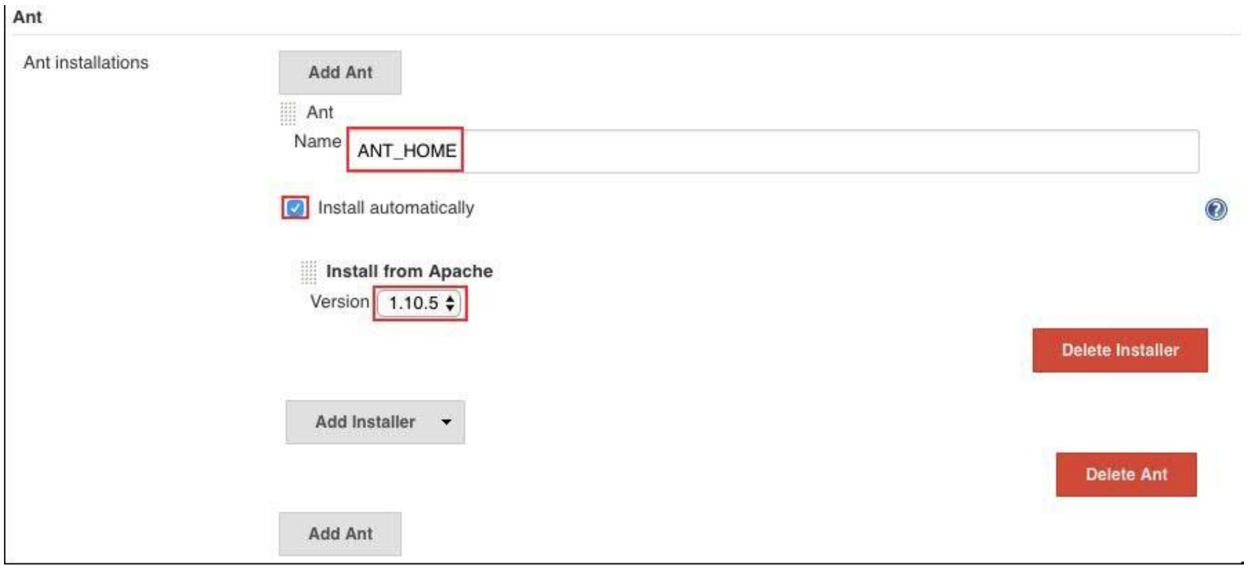
While building if you see below error



```
[Test] $ ant -file build-mt.xml
ERROR: command execution failed.Maybe you need to configure the job to choose one of your Ant installations?
Finished: FAILURE
```

Solution:

Go to the Manage Jenkins ---> Global Tool Configuration ---> Ant ---> Ant Installations...


Ant

Ant installations

Ant

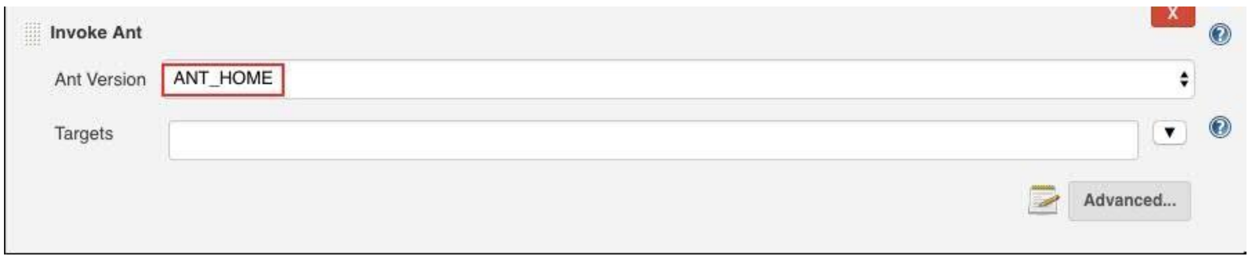
Name: ANT_HOME

[✓] Install automatically

Install from Apache

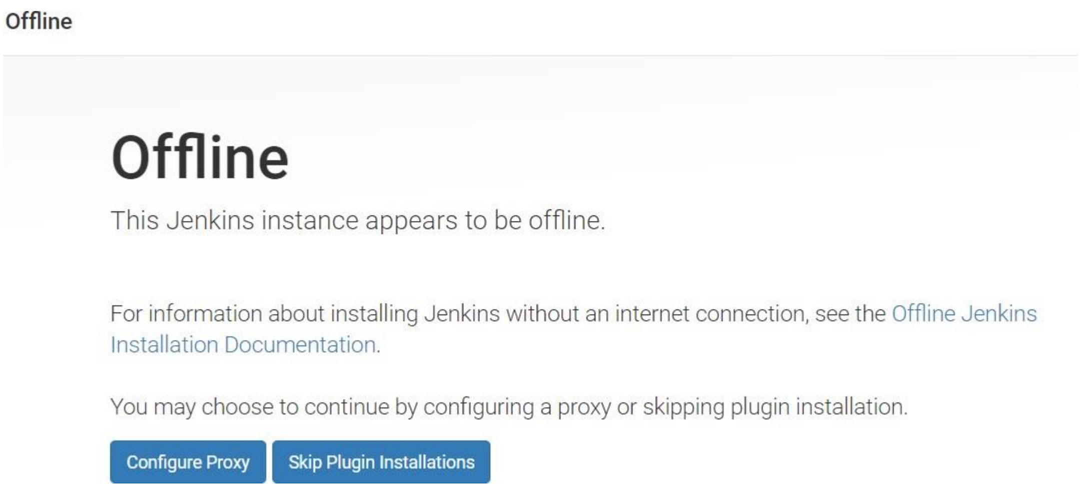
Version: 1.10.5

and in Job, select the Ant Versions as follows.



Installation Issues:

Issue 1: Offline


Offline

Offline

This Jenkins instance appears to be offline.

For information about installing Jenkins without an internet connection, see the Offline Jenkins Installation Documentation.

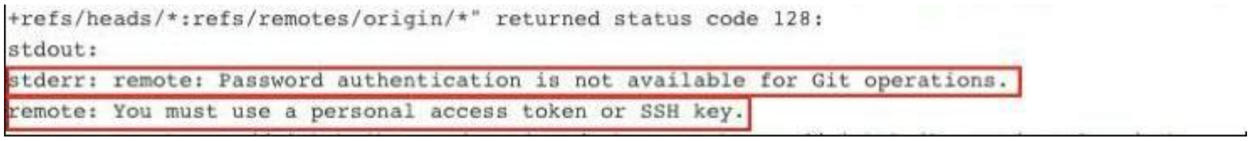
You may choose to continue by configuring a proxy or skipping plugin installation.

Solution

jenkinshomedir/hudson.model.UpdateCenter.xml and change url to use **http** instead of **https**.

Once you changed from https to http, you need to restart the Jenkins.

Issue

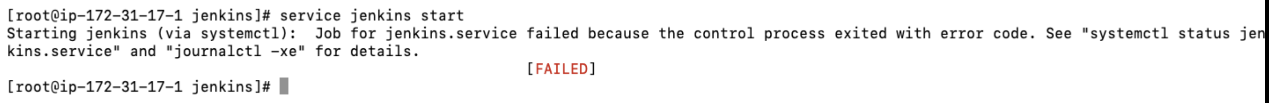

```
+refs/heads/*:refs/remotes/origin/*" returned status code 128:
stdout:
stderr: remote: Password authentication is not available for Git operations.
remote: You must use a personal access token or SSH key.
```

Solution

If you see this error, generate SSH or PAT and use these keys instead of password.

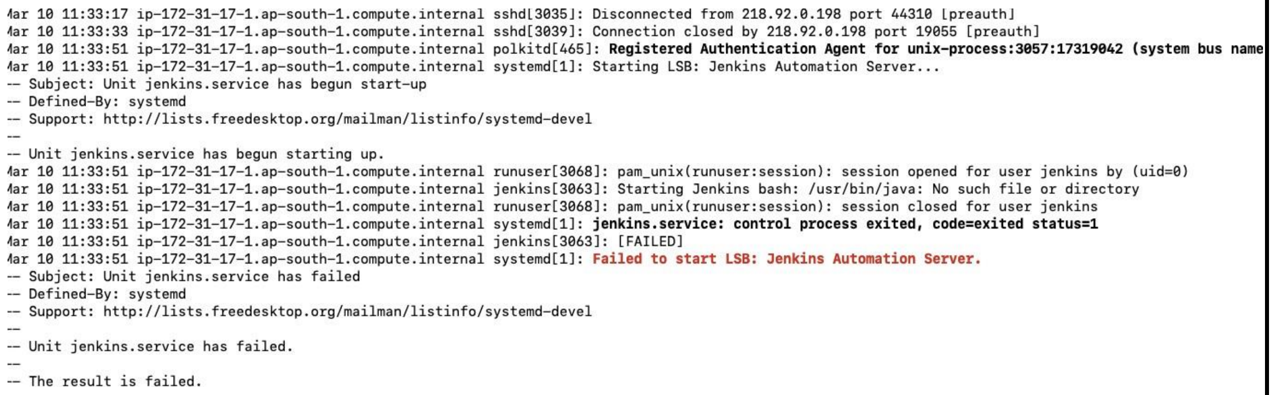
Issue Jenkins Start

#service Jenkins start



```
[root@ip-172-31-17-1 jenkins]# service jenkins start
Starting jenkins (via systemctl):  Job for jenkins.service failed because the control process exited with error code. See "systemctl status jen
kins.service" and "journalctl -xe" for details.
                                                           [FAILED]
[root@ip-172-31-17-1 jenkins]# █
```

#journalctl -xe

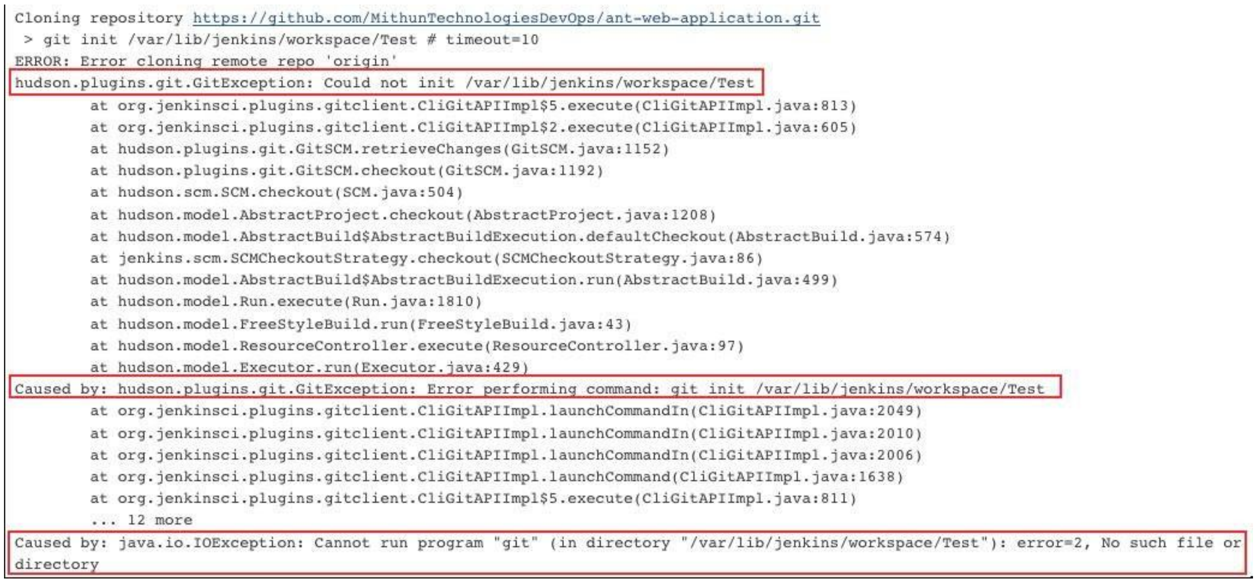


```
Mar 10 11:33:17 ip-172-31-17-1.ap-south-1.compute.internal sshd[3035]: Disconnected from 218.92.0.198 port 44310 [preauth]
Mar 10 11:33:33 ip-172-31-17-1.ap-south-1.compute.internal sshd[3039]: Connection closed by 218.92.0.198 port 19055 [preauth]
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal polkitd[465]: Registered Authentication Agent for unix-process:3057:17319042 (system bus name
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal systemd[1]: Starting LSB: Jenkins Automation Server...
-- Subject: Unit jenkins.service has begun start-up
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit jenkins.service has begun starting up.
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal runuser[3068]: pam_unix(runuser:session): session opened for user jenkins by (uid=0)
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal jenkins[3063]: Starting Jenkins bash: /usr/bin/java: No such file or directory
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal runuser[3068]: pam_unix(runuser:session): session closed for user jenkins
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal systemd[1]: jenkins.service: control process exited, code=exited status=1
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal jenkins[3063]: [FAILED]
Mar 10 11:33:51 ip-172-31-17-1.ap-south-1.compute.internal systemd[1]: Failed to start LSB: Jenkins Automation Server.
-- Subject: Unit jenkins.service has failed
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit jenkins.service has failed.
--
-- The result is failed.
```

Solution

Install the java.

Issue:



```
Cloning repository https://github.com/MithunTechnologiesDevOps/ant-web-application.git
 > git init /var/lib/jenkins/workspace/Test # timeout=10
ERROR: Error cloning remote repo 'origin'
hudson.plugins.git.GitException: Could not init /var/lib/jenkins/workspace/Test
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl$5.execute(CliGitAPIImpl.java:813)
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl$2.execute(CliGitAPIImpl.java:605)
        at hudson.plugins.git.GitSCM.retrieveChanges(GitSCM.java:1152)
        at hudson.plugins.git.GitSCM.checkout(GitSCM.java:1192)
        at hudson.scm.SCM.checkout(SCM.java:504)
        at hudson.model.AbstractProject.checkout(AbstractProject.java:1208)
        at hudson.model.AbstractBuild$AbstractBuildExecution.defaultCheckout(AbstractBuild.java:574)
        at jenkins.scm.SCMCheckoutStrategy.checkout(SCMCheckoutStrategy.java:86)
        at hudson.model.AbstractBuild$AbstractBuildExecution.run(AbstractBuild.java:499)
        at hudson.model.Run.execute(Run.java:1810)
        at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:43)
        at hudson.model.ResourceController.execute(ResourceController.java:97)
        at hudson.model.Executor.run(Executor.java:429)
Caused by: hudson.plugins.git.GitException: Error performing command: git init /var/lib/jenkins/workspace/Test
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn(CliGitAPIImpl.java:2049)
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn(CliGitAPIImpl.java:2010)
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn(CliGitAPIImpl.java:2006)
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommand(CliGitAPIImpl.java:1638)
        at org.jenkinsci.plugins.gitclient.CliGitAPIImpl$5.execute(CliGitAPIImpl.java:811)
        ... 12 more
Caused by: java.io.IOException: Cannot run program "git" (in directory "/var/lib/jenkins/workspace/Test"): error=2, No such file or
directory
```

Solution:

Install the Git.

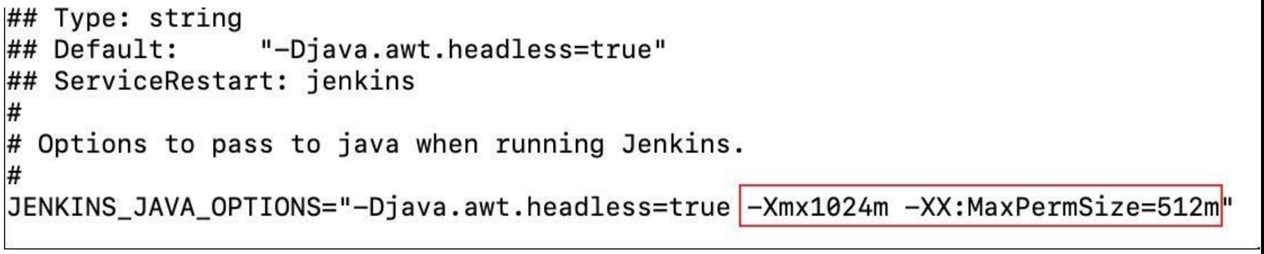
Issue:

There is insufficient memory for the Java Runtime Environment to continue.

Solution:

Increase the JVM size as follows.

vi /etc/sysconfig/jenkins



```
## Type: string
## Default:     "-Djava.awt.headless=true"
## ServiceRestart: jenkins
#
# Options to pass to java when running Jenkins.
#
JENKINS_JAVA_OPTIONS="-Djava.awt.headless=true -Xmx1024m -XX:MaxPermSize=512m"
```

Resources:

https://jenkins.io/ ---> Download software
https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+as+a+Windows+service

http://www.tothenew.com/blog/jenkins-implementing-project-based-matrix-authorization-strategy/ ---> User Access

https://support.cloudbees.com/hc/en-us/articles/216118748-How-to-Start-Stop-or-Restart-your-Instance

https://www.jdev.it/deploying-your-war-file-from-jenkins-to-tomcat/ ---> Deploy into Tomcat