FREE eBook

LEARNING jenkins

Free unaffiliated eBook created from
Stack Overflow contributors.

#jenkins

Table of Contents

About ... 1

Chapter 1: Getting started with jenkins ... 2

Remarks ... 2

Versions ... 2

Jenkins ... 2

Jenkins 1.x vs Jenkins 2.x ... 2

Examples ... 3

Installation ... 3

Upgrading jenkins(RPM installations) ... 3

Setting up Nginx Proxy ... 4

Installing Plugin from external source ... 5

Move Jenkins from one PC to another ... 5

Configure a project in Jenkins ... 5

Jenkins full Introduction in one place ... 6

Configure a simple build project with Jenkins 2 pipeline script ... 11

Chapter 2: Configure Auto Git Push on Successful Build in Jenkins ... 13

Introduction ... 13

Examples ... 13

Configuring the Auto Push Job ... 13

Chapter 3: Install Jenkins on Windows with SSH support for private GitHub repositories ... 22

Examples ... 22

GitHub pull requests fail ... 22

PSExec.exe PS Tool by Microsoft ... 22

Generate a new SSH key just for Jenkins using PSExec or PSExec64 ... 22

Create the Jenkins Credentials ... 23

Run a test pull request to verify, and your done ... 25

Chapter 4: Jenkins Groovy Scripting ... 27

Examples ... 27

Create default user ... 27

Disable Setup Wizard ... 27

How to get infromation about Jenkins instance ... 28
How to get information about a Jenkins job ... 28
Chapter 5: Role Strategy Plugin ... 29
Examples ... 29
Configuration ... 29
Manage Roles ... 29
Assign Roles ... 30
Chapter 6: Setting up Build Automation for iOS using Shenzhen ... 33
Examples ... 33
iOS Build Automation Setup using Shenzhen ... 33
Chapter 7: Setting up Jenkins for iOS build automation. ... 34
Introduction ... 34
Parameters ... 34
Remarks ... 34
Examples ... 34
Time Table Example ... 34
Credits ... 36

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: jenkins

It is an unofficial and free jenkins ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official jenkins.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with jenkins

Remarks

Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle.

Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands. The primary developer of Jenkins is Kohsuke Kawaguchi. Released under the MIT License, Jenkins is free software.

Builds can be started by various means, including being triggered by commit in a version control system, by scheduling via a cron-like mechanism, by building when other builds have completed, and by requesting a specific build URL.

Versions

Jenkins

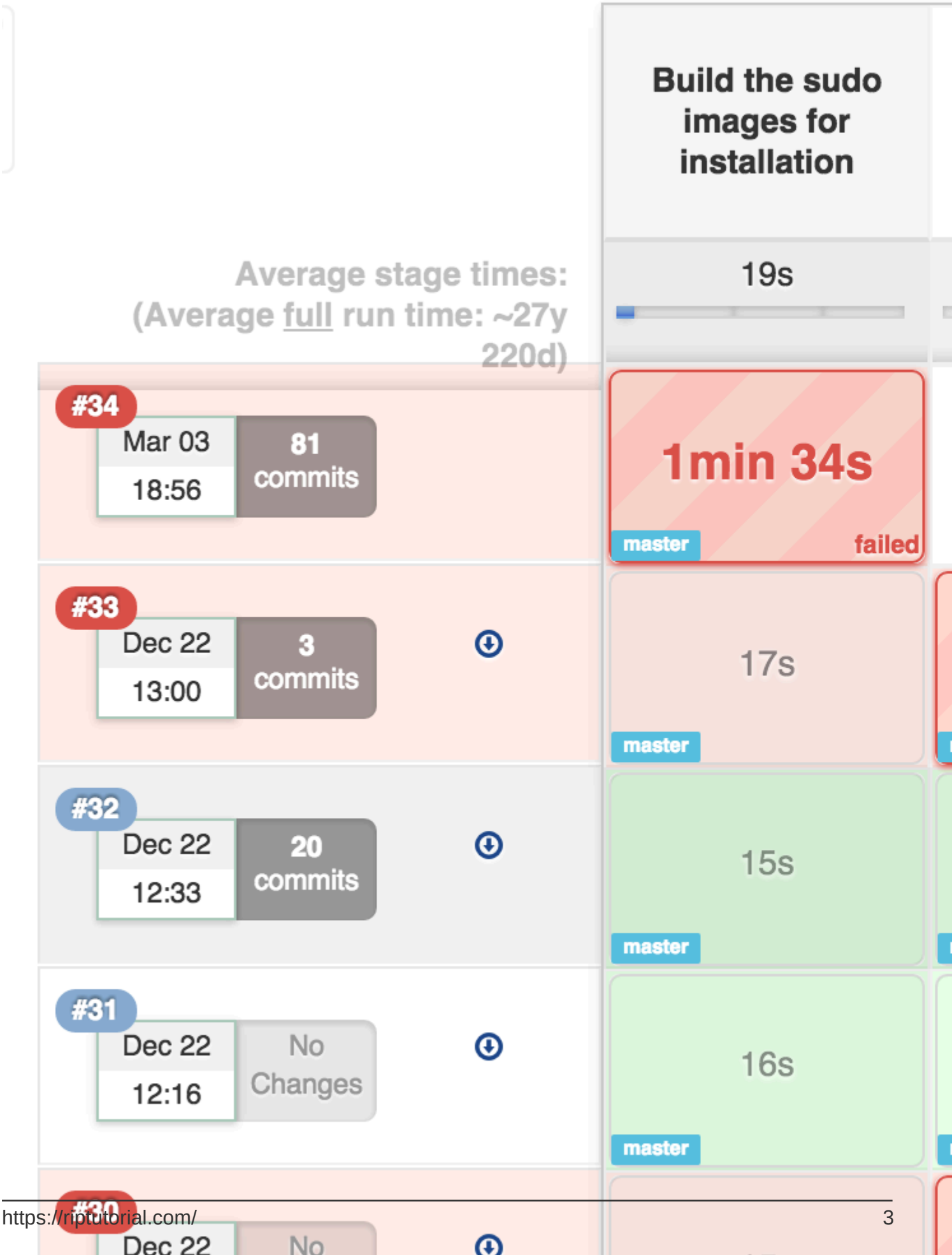
Version	Release Date
1.656	2016-04-03
2.0	2016-04-20

Jenkins 1.x vs Jenkins 2.x

Jenkins is (and still is) a continuous integration (CI) system that allows automation of software development process, such as building code on SCM commit triggers. However the growing need for continuous delivery (CD) has requested that Jenkins evolves for a pure CI system to a mix of CI and CD. Also, the need to undustrialize Jenkins jobs has been growing and classic Jenkins 1.x `Freestyle/Maven jobs` started to be too limited for certain needs.

Under Jenkins 1.x a plugin called `workflow-plugin` appeared to allow developers to write code to describe jobs. Jenkins 2 goes further by adding built-in support for `Pipeline as Code`. The main benefit is that pipelines, being Groovy scripts files, can be more complex than UI-configured freestyle jobs and can be version-controlled. Jenkins 2 also adds a new interface that makes it easy to visualize different "stages" defined in a pipeline and follow the progress of the entire pipeline, such as below :

Stage View


Build the sudo images for installation
Average stage times:
(Average full run time: ~27y 220d)
19s
#34
Mar 03
18:56
81 commits
1min 34s
master
failed
#33
Dec 22
13:00
3 commits
17s
master
#32
Dec 22
12:33
20 commits
15s
master
#31
Dec 22
12:16
No Changes
16s
master

2. Replace jenkins.war in following location with new WAR file. /usr/lib/jenkins/jenkins.war`
3. Restart Jenkins
4. Check pinned plugins and unpin if required
5. Reload Configuration from Disk

note: For Jenkins 2 upgrades for bundled jetty app server,disable AJP port(set `JENKINS_AJP_PORT="-1"`) in `/etc/sysconfig/jenkins`.

Setting up Nginx Proxy

Natively, Jenkins runs on port 8080. We can establish a proxy from port 80 -> 8080 so Jenkins can be accessed via:

```
http://<url>.com
```

instead of the default

```
http://<url>.com:8080
```

Begin by installing Nginx.

```
sudo aptitude -y install nginx
```

Remove the default settings for Nginx

```
cd /etc/nginx/sites-available

sudo rm default ../sites-enabled/default
```

Create the new configuration file

```
sudo touch jenkins
```

Copy the following code into the newly created `jenkins` file.

```
upstream app_server {
  server 127.0.0.1:8080 fail_timeout=0;
}

server {
  listen 80;
  listen [::]:80 default ipv6only=on;
  server_name ;

  location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;

    if (!-f $request_filename) {
        proxy_pass http://app_server;
        break;
    }
  }
```

```
}
```

Create a symbolic link between sites-available and sites-enabled:

```
sudo ln -s /etc/nginx/sites-available/jenkins /etc/nginx/sites-enabled/
```

Restart the Nginx proxy service

```
sudo service nginx restart
```

Jenkins will now be accessible from port 80.

Installing Plugin from external source

```
java -jar [Path to client JAR] -s [Server address] install-plugin [Plugin ID]
```

The client JAR must be the CLI JAR file, not the same JAR/WAR that runs Jenkins itself. Unique IDs can be found on a plugins respective page on the Jenkins CLI wiki (https://wiki.jenkins-ci.org/display/JENKINS/Plugins)

Move Jenkins from one PC to another

This worked for me to move from Ubuntu 12.04 (Jenkins ver. 1.628) to Ubuntu 16.04 (Jenkins ver. 1.651.2). I first installed Jenkins from the repositories.

1. Stop both Jenkins servers
2. Copy `JENKINS_HOME` (e.g. /var/lib/jenkins) from the old server to the new one. From a console in the new server:

   ```
   rsync -av username@old-server-IP:/var/lib/jenkins/ /var/lib/jenkins/
   ```

3. Start your new Jenkins server

You might not need this, but I had to

- `Manage Jenkins` and `Reload Configuration from Disk`.
- Disconnect and connect all the slaves again.
- Check that in the `Configure System > Jenkins Location`, the `Jenkins URL` is correctly assigned to the new Jenkins server.

Configure a project in Jenkins

Here we will be checking out the latest copy of our project's code, run the tests and will make the application live.To achieve that, follow below steps:

1. Open Jenkins in browser.
2. Click the ***New Job*** link.
3. Enter project name and select the ***Build a free-style software project*** link.
4. Click on ***Ok*** button.

5. Under the ***Source code management section***, select the radio box next to your source code management tool. In my case I have selected ***Git***.

Provide url of git repo like `git://github.com/example/example.git`

6. Under the ***Build triggers***, select the radio box next to ***Poll SCM***.
7. Provide `*****` in ***Schedule*** box. This box is responsible to trigger the build at regular intervals. `*****` specifies that, the job will get trigger every minute for changes in git repo.
8. Under the ***Build*** section, click the ***Add Build Step*** button and then select the option by which you want to build the project. I have selected ***Execute Shell***. In the command box write the command to build,run the tests, and deploy it to prod.
9. Scroll down and ***Save***.

So above we have configured a basic project in Jenkins which will trigger the build at every minute for change in your git repository. Note: To setup the complex project, you may have to install some plugins in Jenkins.

Jenkins full Introduction in one place

1. Jenkins :

Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle.

In a nutshell, Jenkins is the leading open source automation server. Built with Java, it provides hundreds of plugins to support building, testing, deploying and automation for virtually any project.

Features : Jenkins offers the following major features out of the box, and many more can be added through plugins:

Easy installation: Just run java -jar jenkins.war, deploy it in a servlet container. No additional install, no database. Prefer an installer or native package? We have those as well. Easy configuration: Jenkins can be configured entirely from its friendly web GUI with extensive on-the-fly error checks and inline help. Rich plugin ecosystem: Jenkins integrates with virtually every SCM or build tool that exists. View plugins. Extensibility: Most parts of Jenkins can be extended and modified, and it's easy to create new Jenkins plugins. This allows you to customize Jenkins to your needs. Distributed builds: Jenkins can distribute build/test loads to multiple computers with different operating systems. Building software for OS X, Linux, and Windows? No problem.

Installation :

```
$ wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -

$ sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ >
/etc/apt/sources.list.d/jenkins.list'
$ sudo apt-get update
$ sudo apt-get install jenkins
to do more refer link :
```

Ref : https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Ubuntu

Ref : http://www.vogella.com/tutorials/Jenkins/article.html

Ref : https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

JENKINS_HOME directory Jenkins needs some disk space to perform builds and keep archives. You can check this location from the configuration screen of Jenkins. By default, this is set to ~/.jenkins, but you can change this in one of the following ways: Set "JENKINS_HOME" environment variable to the new home directory before launching the servlet container. Set "JENKINS_HOME" system property to the servlet container. Set JNDI environment entry "JENKINS_HOME" to the new directory. See the container specific documentation collection for more about how to do this for your container. You can change this location after you've used Jenkins for a while, too. To do this, stop Jenkins completely, move the contents from old JENKINS_HOME to the new home, set the new JENKINS_HOME, and restart Jenkins. JENKINS_HOME has a fairly obvious directory structure that looks like the following:

JENKINS_HOME

```
+- config.xml     (jenkins root configuration)
 +- *.xml          (other site-wide configuration files)
 +- userContent    (files in this directory will be served under your
http://server/userContent/)
 +- fingerprints   (stores fingerprint records)
 +- plugins        (stores plugins)
 +- workspace (working directory for the version control system)
     +- [JOBNAME] (sub directory for each job)
 +- 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)
                 +- build.xml      (build result summary)
                 +- log            (log file)
                 +- changelog.xml  (change log)
```

Jenkins Build Jobs :

Creating a new build job in Jenkins is simple: just click on the “New Job” menu item on the Jenkins dashboard. Jenkins supports several different types of build jobs, which are presented to you when you choose to create a new job

Freestyle software project

Freestyle build jobs are general-purpose build jobs, which provides a maximum of flexibility.

Maven project The “maven2/3 project” is a build job specially adapted to Maven projects. Jenkins understands Maven pom files and project structures, and can use the information gleaned from the pom file to reduce the work you need to do to set up your project.

Workflow

Orchestrates long-running activities that can span multiple build slaves. Suitable for building

pipelines and/or organizing complex activities that do not easily fit in free-style job type.

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

Multiconfiguration job 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.

1. Building a software project (free style)

Jenkins can be used to perform the typical build server work, such as doing continuous/official/nightly builds, run tests, or perform some repetitive batch tasks. This is called "free-style software project" in Jenkins. Setting up the project Go to Jenkins top page, select "New Job", then choose "Build a free-style software project". This job type consists of the following elements: optional SCM, such as CVS or Subversion where your source code resides. optional triggers to control when Jenkins will perform builds. some sort of build script that performs the build (ant, maven, shell script, batch file, etc.) where the real work happens optional steps to collect information out of the build, such as archiving the artifacts and/or recording javadoc and test results. optional steps to notify other people/systems with the build result, such as sending e-mails, IMs, updating issue tracker, etc.

Builds for Non-Source Control Projects There is sometimes a need to build a project simply for demonstration purposes or access to a SVN/CVS repository is unavailable. By choosing to configure the project as "None" under "Source Code Management" you will have to:

1. Build the Project at least once, (it will fail), but Jenkins will create the structure jenkins/workspace/PROJECTNAME/
2. Copy the project files to jenkins/workspace/PROJECTNAME/
3. Build again and configure appropriately

Jenkins Set Environment Variables

When a Jenkins job executes, it sets some environment variables that you may use in your shell script, batch command, Ant script or Maven POM . See the list of variable by clicking on ENVIRONMENT_VARIABLE

Configuring automatic builds

Builds in Jenkins can be triggered periodically (on a schedule, specified in configuration), or when source changes in the project have been detected, or they can be automatically triggered by requesting the URL:

http://YOURHOST/jenkins/job/PROJECTNAME/build

This allows you to hook Jenkins builds into a variety of setups. For more information (in particular doing this with security enabled), see Remote access API.

Builds by source changes

You can have Jenkins poll your Revision Control System for changes. You can specify how often Jenkins polls your revision control system using the same syntax as crontab on Unix/Linux. However, if your polling period is shorter than it takes to poll your revision control system, you may end up with multiple builds for each change. You should either adjust your polling period to be longer than the amount of time it takes to poll your revision control system, or use a post-commit trigger. You can examine the Polling Log for each build to see how long it took to poll your system.

Alternatively, instead of polling on a fixed interval, you can use a URL trigger (described above), but with /polling instead of /build at the end of the URL. This makes Jenkins poll the SCM for changes rather than building immediately. This prevents Jenkins from running a build with no relevant changes for commits affecting modules or branches that are unrelated to the job. When using /polling the job must be configured for polling, but the schedule can be empty.

Builds by e-mail (sendmail)

If you have the root account of your system and you are using sendmail, I found it the easiest to tweak /etc/aliases and add the following entry: jenkins-foo: "|/bin/wget -o /dev/null

http://YOURHOST/jenkins/job/PROJECTNAME/build"

and then run "newaliases" command to let sendmail know of the change. Whenever someone sends an e-mail to "jenkins-foo@yoursystem", this will trigger a new build. See this for more details about configuring sendmail. Builds by e-mail (qmail) With qmail, you can write /var/qmail/alias/.qmail-jenkins as follows: |/bin/wget -o /dev/null http://YOURHOST/jenkins/job/PROJECTNAME/build"

2. Building a Maven project

Jenkins provides a job type dedicated to Maven 2/3. This job type integrates Jenkins deeply with Maven 2/3 and provides the following benefits compared to the more generic free-style software project.

Jenkins parses Maven POMs to obtain much of the information needed to do its work. As a result, the amount of configuration is drastically reduced.

Jenkins listens to Maven execution and figures out what should be done when on its own. For example, it will automatically record the JUnit report when Maven runs the test phase. Or if you run the javadoc goal, Jenkins will automatically record javadoc.

Jenkins automatically creates project dependencies between projects which declare SNAPSHOT dependencies between each other. See below. Thus mostly you just need to configure SCM information and what goals you'd like to run, and Jenkins will figure out everything else.

This project type can automatically provide the following features:

Archive artifacts produced by a build

Publish test results

Trigger jobs for projects which are downstream dependencies

Deploy your artifacts to a Maven repository

Breakout test results by module

Optionally rebuild only changed modules, speeding your builds

Automatic build chaining from module dependencies

Jenkins reads dependencies of your project from your POM, and if they are also built on Jenkins, triggers are set up in such a way that a new build in one of those dependencies will automatically start a new build of your project. Jenkins understands all kinds of dependencies in POM. Namely,parent POM

```
<dependencies> section of your project
<plugins> section of your project
<extensions> section of your project
<reporting> section of your project
```

This process takes versions into account, so you can have multiple versions/branches of your project on the same Jenkins and it will correctly determine dependencies. **Note** that dependency version ranges are not supported, see [https://issues.jenkins-ci.org/browse/JENKINS-2787][1] for the reason.

This feature can be disabled on demand - see configuration option Build whenever a SNAPSHOT dependency is built

Installation :

1 . go into Manage Jenkins>>configure System

2. in maven tab “Click on maven installation.....

You can either get Jenkins to install a specific version of Maven automatically , or provide a path to a local Maven installation (You can configure as many versions of Maven for your build projects as you want, and use different versions of Maven for different projects. If you tick the Install automatically checkbox, Jenkins will download and install the requested version of Maven for you and install it to the tools directory in the Jenkins home directory.

How to Use It

First, you must configure a Maven installation (this step can be skipped if you are using DEV@cloud). This can be done by going to the system configuration screen (Manage Jenkins-> Configure System). In the “Maven Installations” section, 1) click the Add button, 2) give it a name such as “Maven 3.0.3” and then 3) choose the version from the drop down.

Now, Jenkins will automatically install this version any time it’s needed (on any new build machines, for example) by downloading it from Apache and unzipping it.

Create a new Maven Job :

1. Clicking “New Job / New Item” on left hand
2. Give it a name
3. Choose the “Build a Maven 2/3 project”
4. Save your job

Now you need to configure of your job

1. Choose the SCM you want to use (ex. Using git)

2. choose maven target to call

3. add Repository URL and Credential.

4. check user private maven repo:

You can also define the custome path for the same.

5 . Build Project

Build your project by clicking on build now and click on the progress bar in the left hand “Build Executor Status” to watch jenkins install Maven, checkout your project, and build it using maven.

Logging:

https://wiki.jenkins-ci.org/display/JENKINS/Logging

Script Console :

Useful for trouble-shooting, diagnostics or batch updates of jobs Jenkins provides a script console which gives you access to all Jenkins internals. These scripts are written in Groovy and you'll find some samples of them in this page.

Configure a simple build project with Jenkins 2 pipeline script

Here we will be creating a Groovy pipeline in Jenkins 2 to do the following steps :

- Verify every 5 minutes if new code has been commited to our project
- Checkout code from SCM repo
- Maven compile of our Java code
- Run our integration tests and publish the results

Here are the steps we will :

1. Make sure we have at least a 2.0 Jenkins version (you can check that in the bottom-right corner of your page) such as :

2. On Jenkins home page, click on **New Item**

3. Enter project name and select **Pipeline**

4. In **Build Triggers** section, select **Poll SCM** option and add the following 5 minutes CRON schedule : `*/5 * * * *`

5. In **Pipeline** section, choose either **Pipeline Script** or **Pipeline Script from SCM**

6. If you selected **Pipeline Script from SCM** on previous step, you now need to specify your SCM repository (Git, Mercurial, Subversion) URL in **Repository URL** such as `http://github.com/example/example.git`. You also need to specify the **Script Path** of your Groovy script file in your example.git repository, e.g. `pipelines/example.groovy`

7. Copy the following Groovy code, either directly in the Groovy script window if you previously clicked **Pipeline Script** or in your `example.groovy` if you choosed **Pipeline Script from SCM**

```
node('remote') {
    // Note : this step is only needed if you're using direct Groovy scripting
    stage 'Checkout Git project'
    git url: 'https://github.com/jglick/simple-maven-project-with-tests.git'
    def appVersion = version()
    if (appVersion) {
        echo "Building version ${appVersion}"
    }

    stage 'Build Maven project'
    def mvnHome = tool 'M3'
    sh "${mvnHome}/bin/mvn -B -Dmaven.test.failure.ignore verify"
    step([$class: 'JUnitResultArchiver', testResults: '**/target/surefire-reports/TEST-
*.xml'])
}
def version() {
    def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'
    matcher ? matcher[0][1] : null
}
```

Here you go, you should now be able to compile and test your first Jenkins project using Jenkins 2 Groovy pipeline.

Read Getting started with jenkins online: https://riptutorial.com/jenkins/topic/919/getting-started-with-jenkins

Chapter 2: Configure Auto Git Push on Successful Build in Jenkins

Introduction

This document will take you through the steps to configure a Jenkins job that allows user to setup auto push on successful build.The push operation can be controlled by the user. User can choose if they want to perform the auto push operation on successful build or not.

Examples

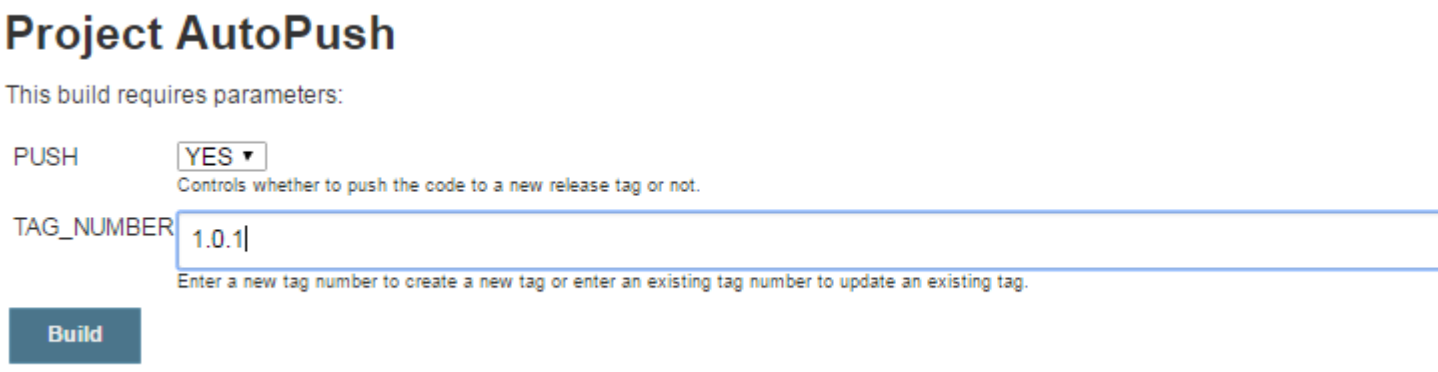
Configuring the Auto Push Job

Create a build job (according to your requirement). For this example I have created a freestyle job (AutoPush) to perform ANT build.

We are going to create two variables, PUSH (Choice Parameter) and TAG_NUMBER (String Parameter).

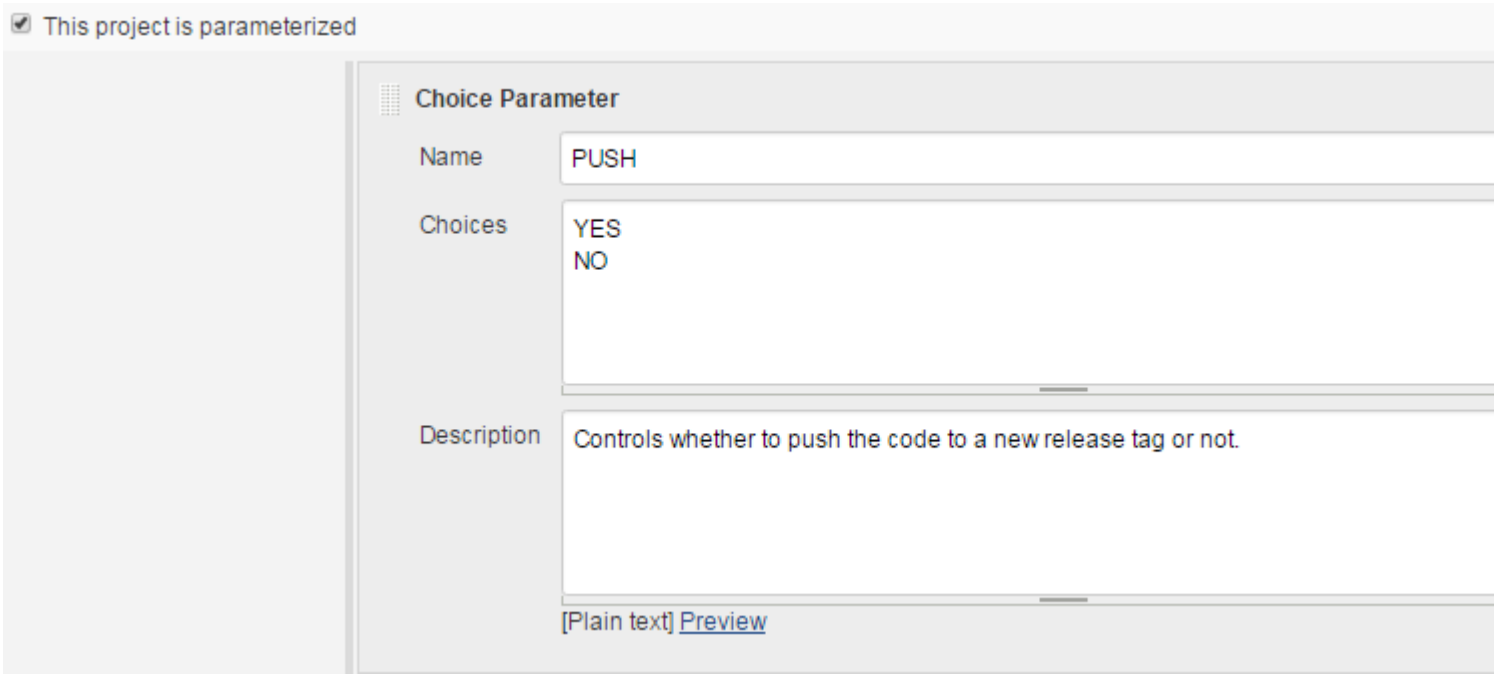
We can choose the value YES or NO for PUSH, this will decide whether to push the code to a tag or not on successful build.

We can specify a tag name (ex. 1.0.1) for TAG_NUMBER to create a new tag (ex. 1.0.1) in the remote repository with the same name or specify an existing tag name to update an existing tag.

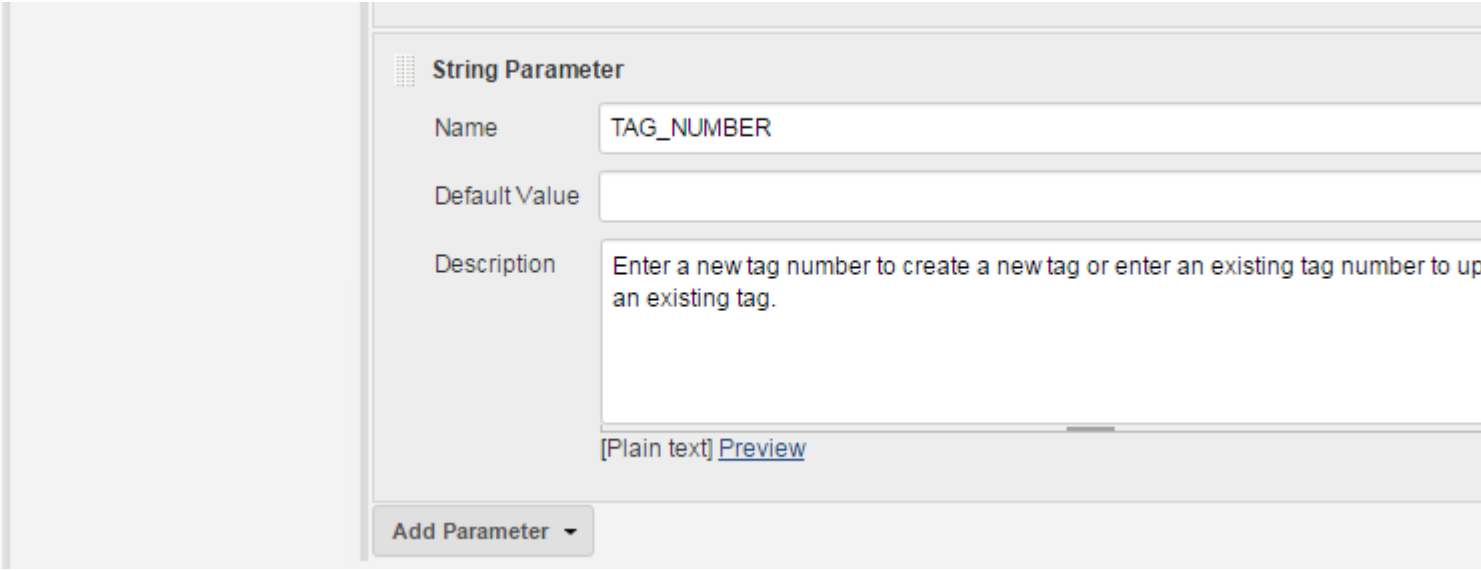


Now let's move on to the job configuration.

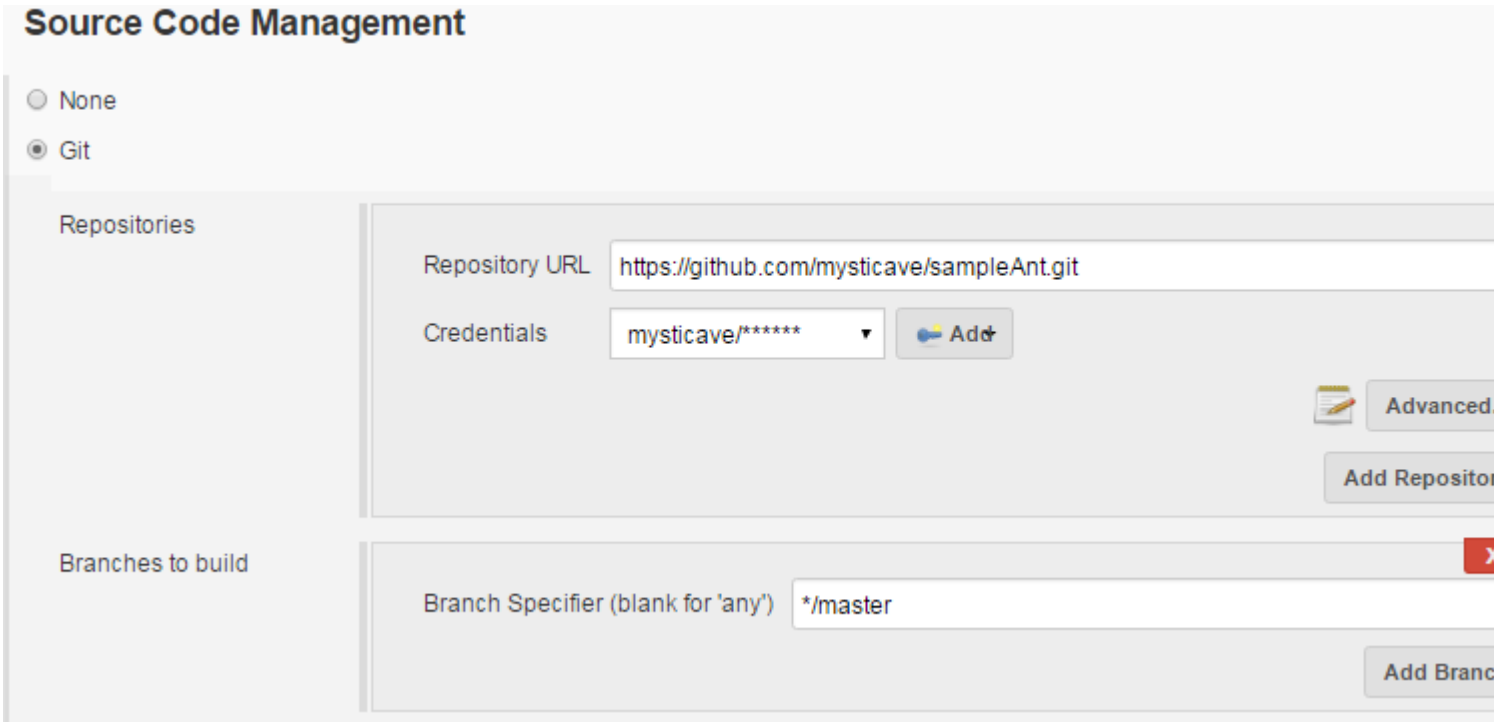
1. Check the "This project is parameterized" checkbox and create a Choice Parameter called "PUSH" and provide YES and NO as the choices. This parameter will decide whether you want to push the code to a specific Tag/Release or not.



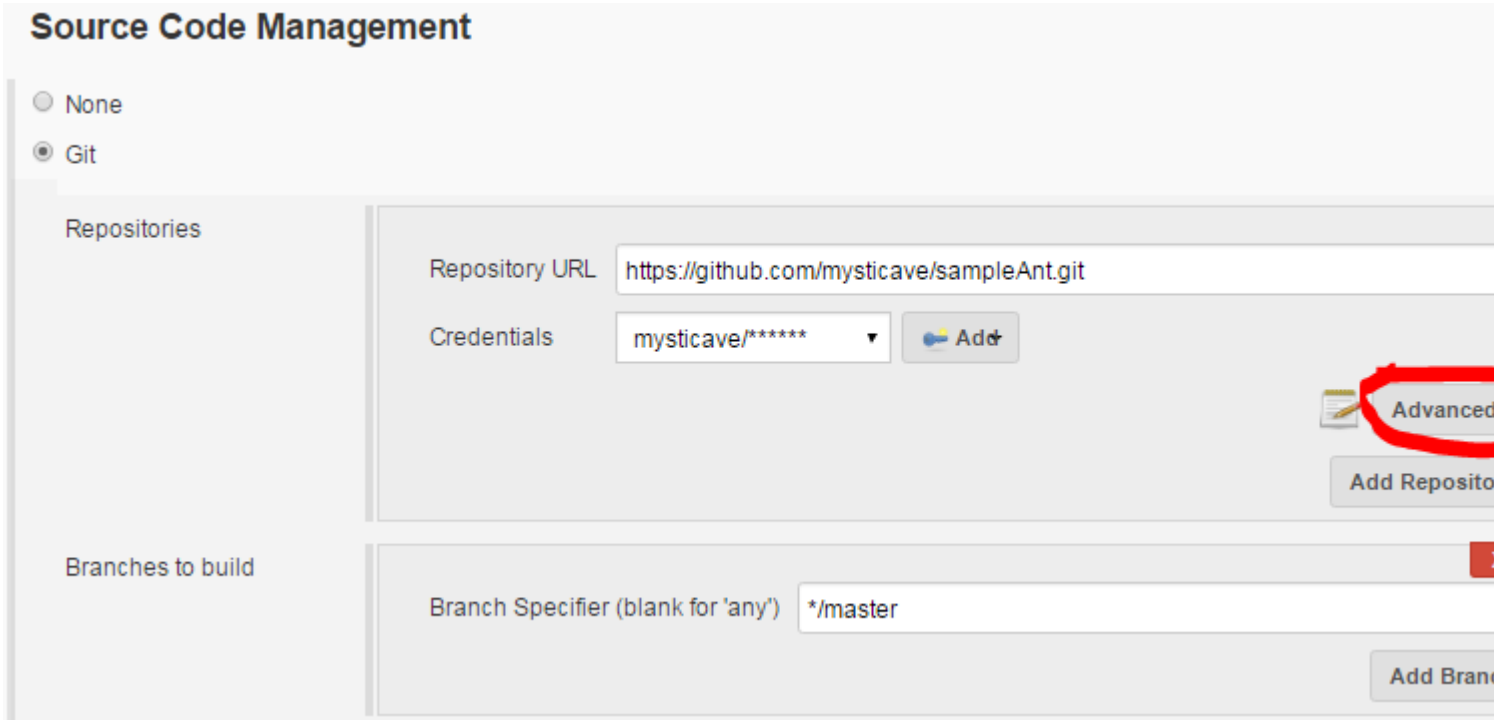
2. Then create a String Parameter called “TAG_NUMBER”, using this parameter we can specify a new tag number to create a new tag or specify an existing tag number to update an existing tag.

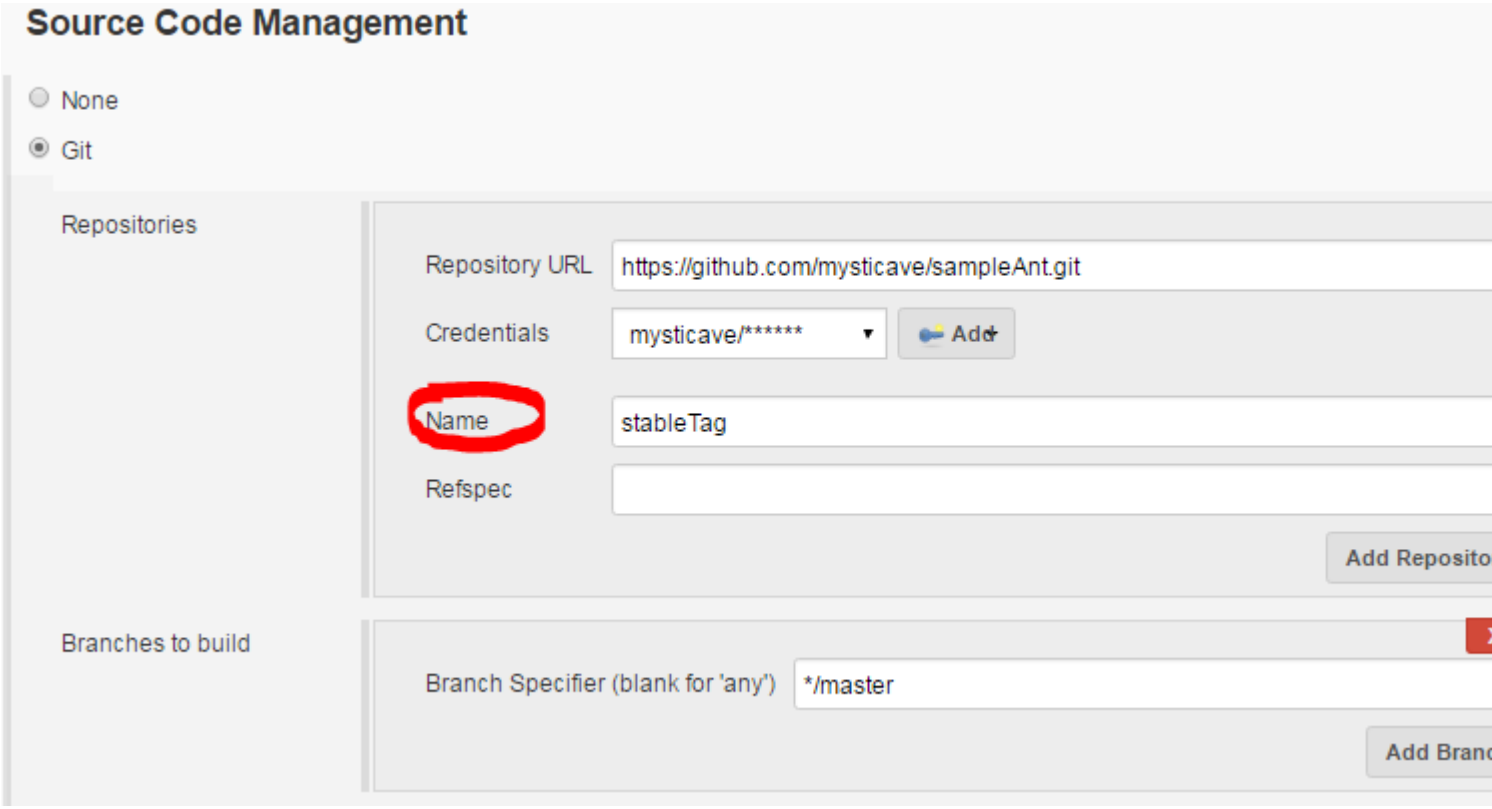


3. In Source Code Management section choose Git and provide the repository URL. This repository contains the source code that you are going to build and after a successful build a release tag will be created on the same repository.



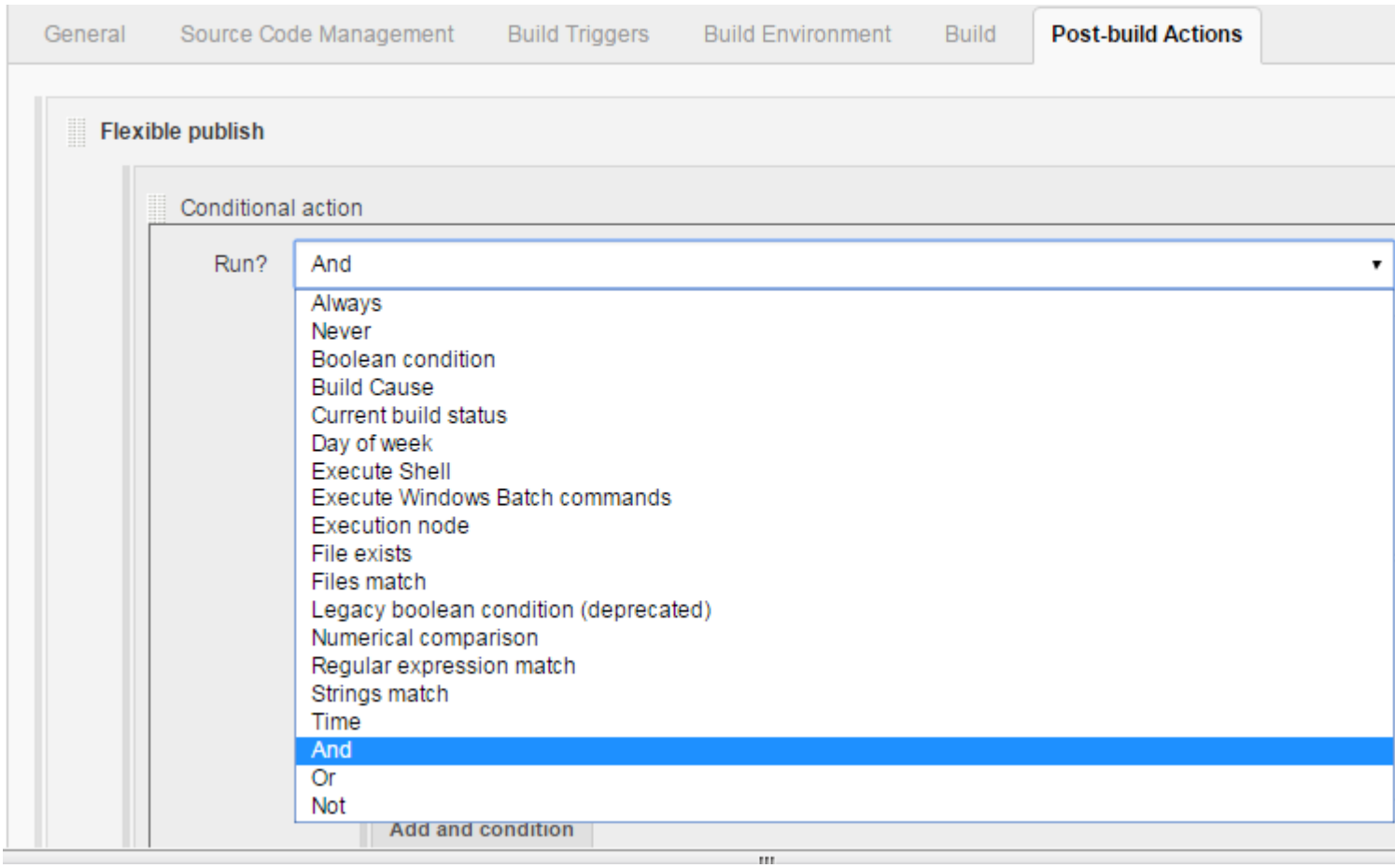
4. After adding the repository details click on advanced and provide a name to your repository which will later get referred in the Git Publisher plugin to identify the repository.

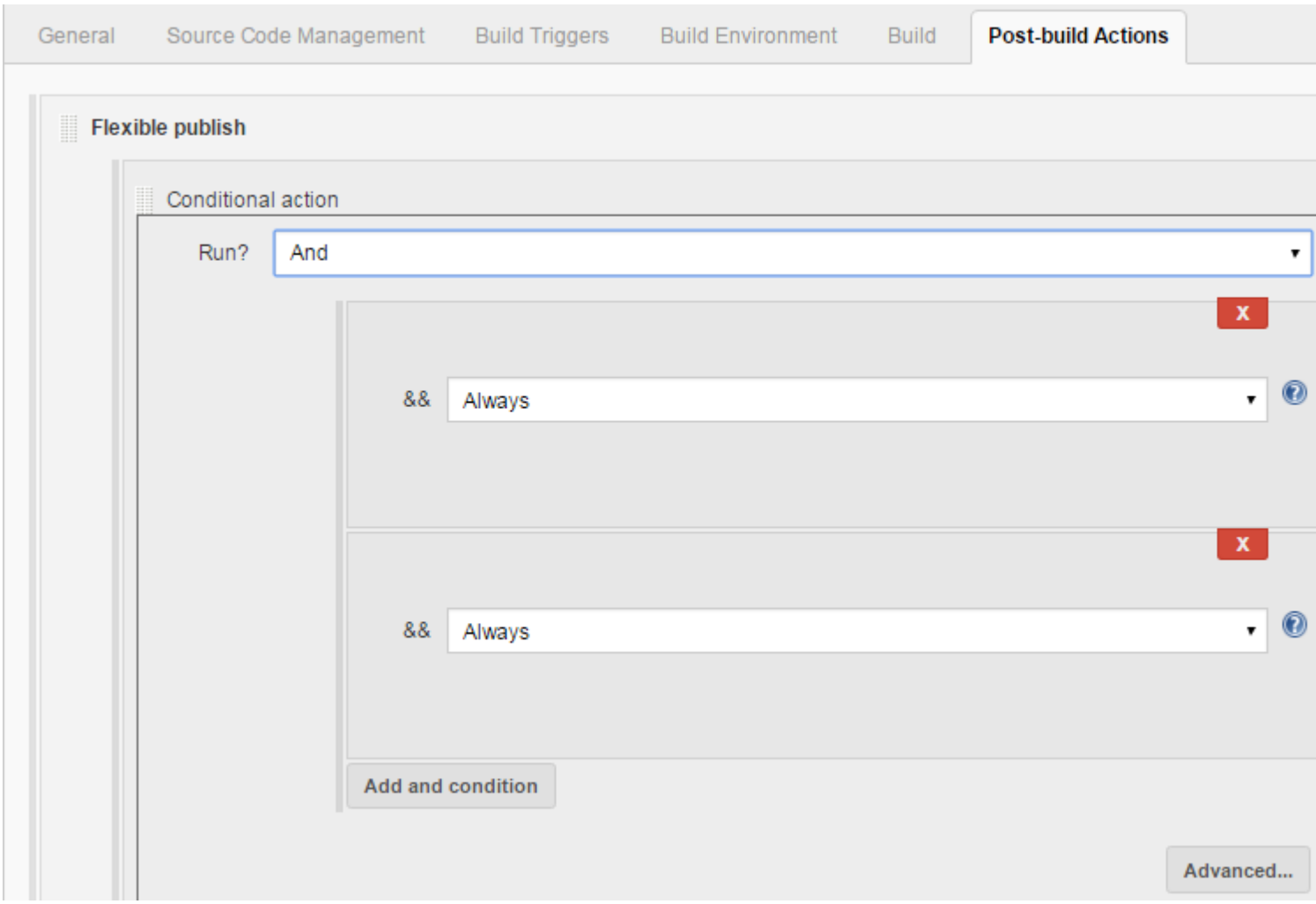


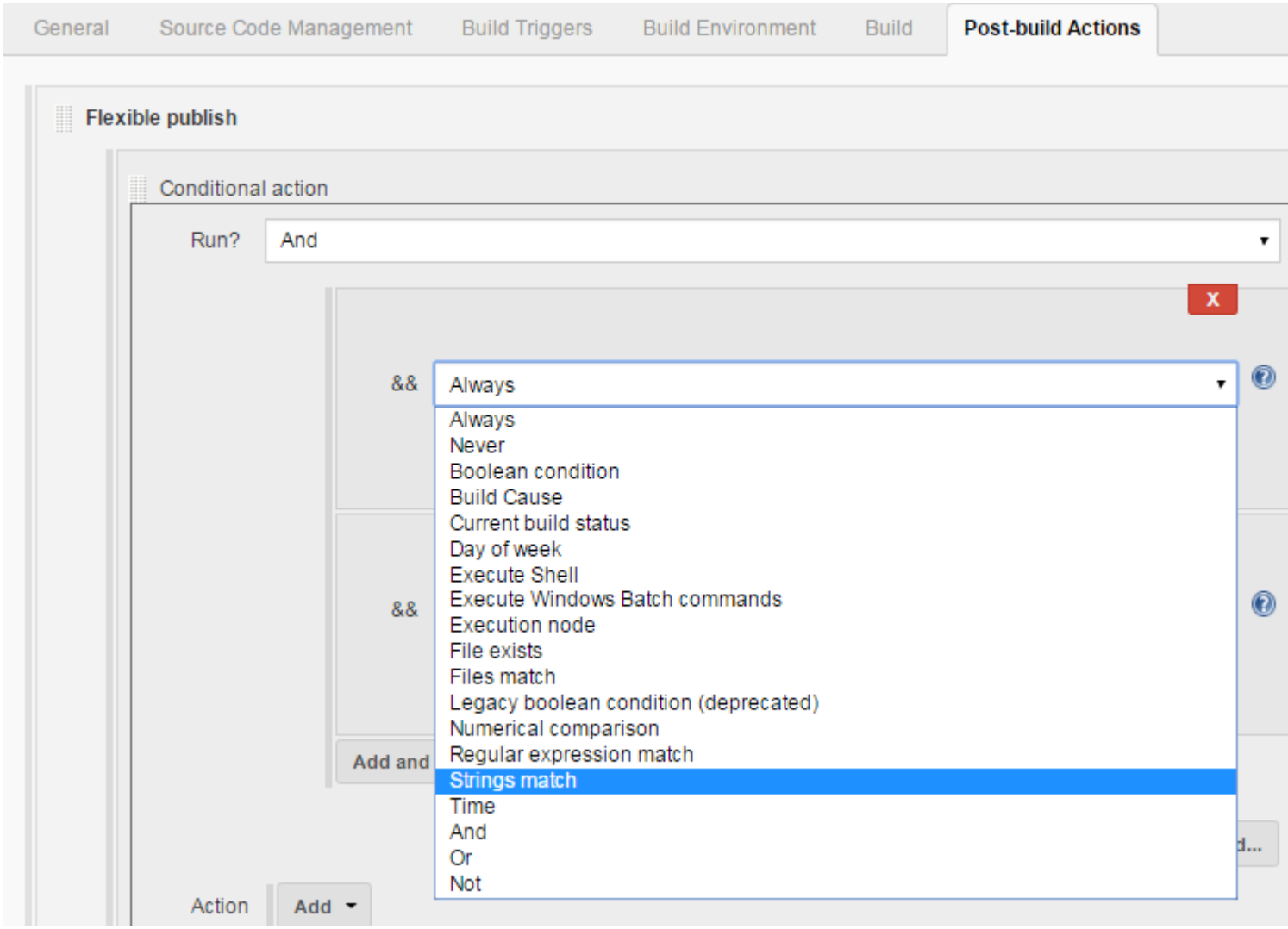


5. Then add the build step. In this example I am building an ANT project.

6. Now in “Post-build Actions” section select “Flexi Publish” plugin. Select the value “And” from the dropdown for Conditional action (Run?). Then select “String Match” from the dropdown for the Run condition (&&).

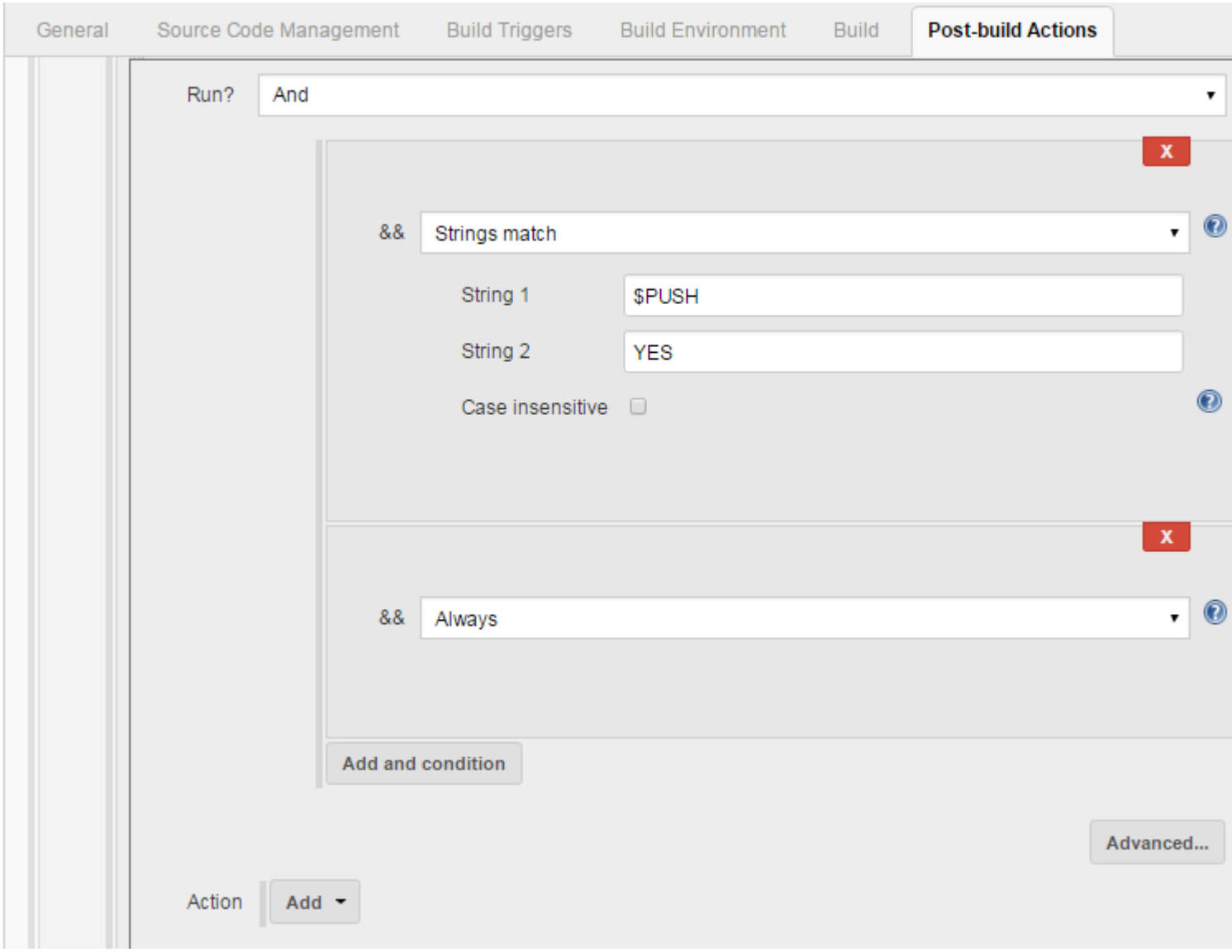

General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Flexible publish
Conditional action
Run?
And
Always
Never
Boolean condition
Build Cause
Current build status
Day of week
Execute Shell
Execute Windows Batch commands
Execution node
File exists
Files match
Legacy boolean condition (deprecated)
Numerical comparison
Regular expression match
Strings match
Time
And
Or
Not
Add and condition


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Flexible publish
Conditional action
Run?
And
x
&&
Always
x
&&
Always
Add and condition
Advanced...

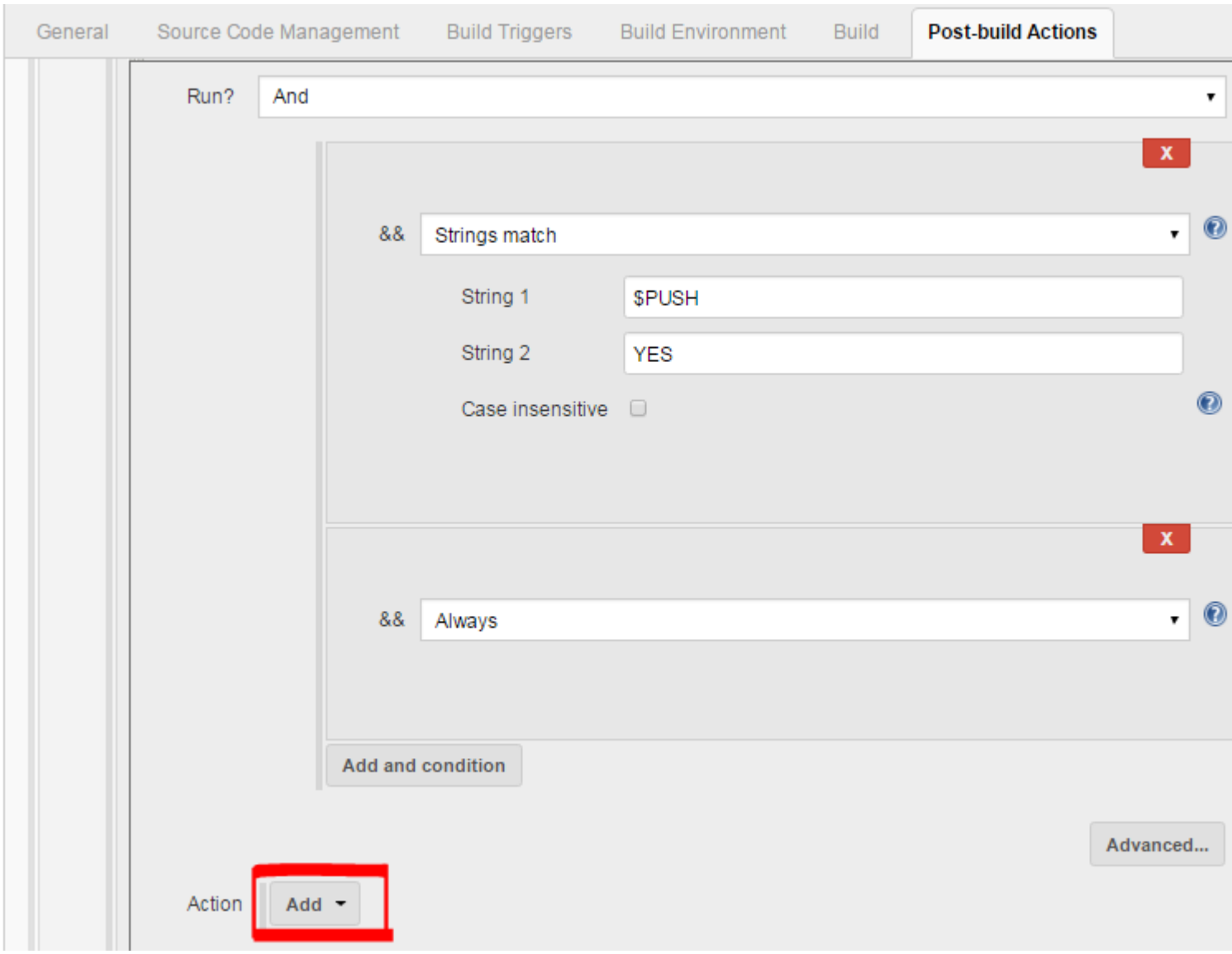


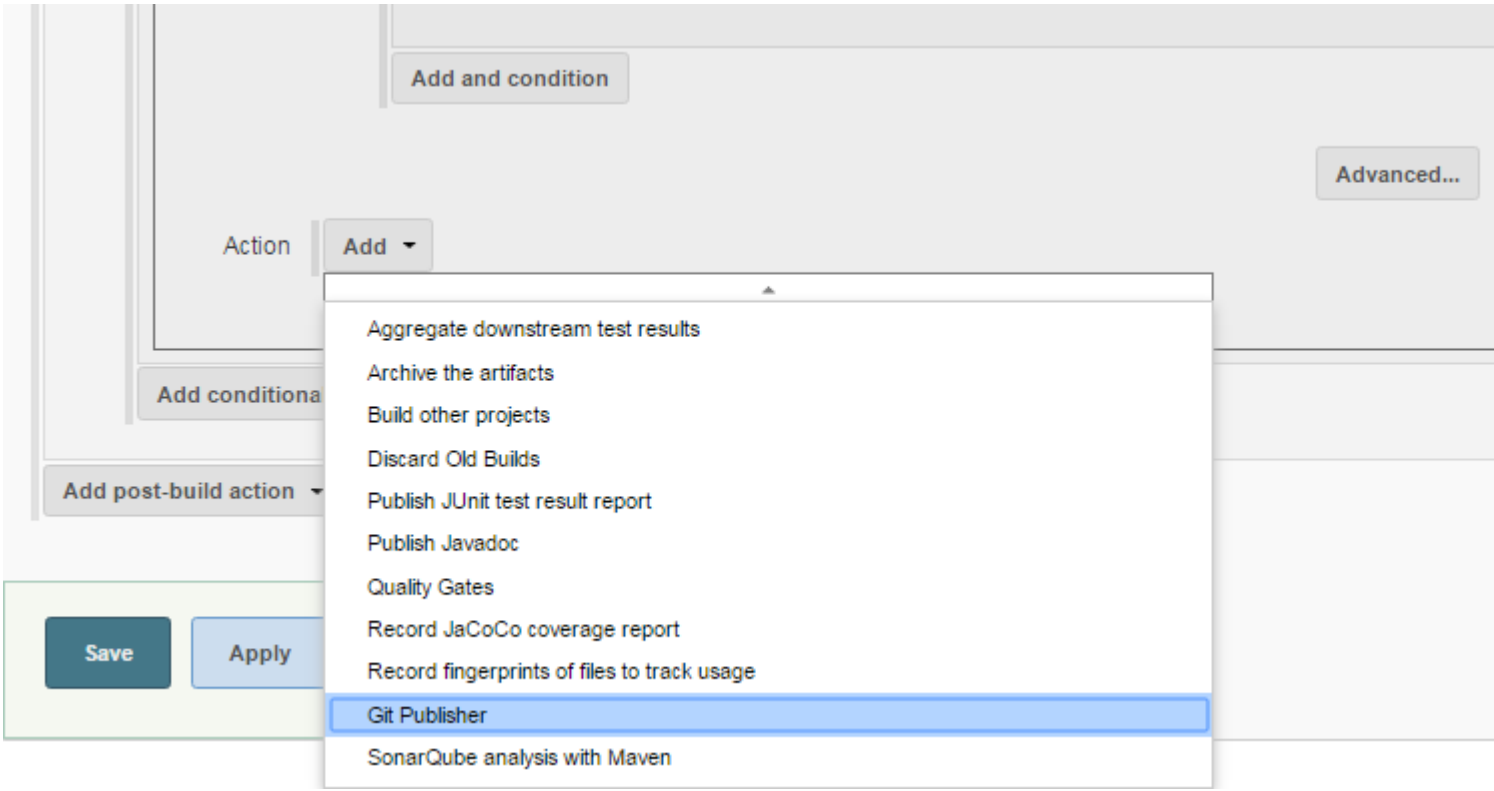
7. After selecting the string match specify $PUSH as String 1 value and YES as String 2 value. So when you will run the build if you choose the value of PUSH as YES, it will compare the String 1 (=$PUSH) and String 2 (=YES) and trigger the Git push operation and if you choose NO it won't trigger the Git push operation.

```
Choose the value of PUSH -> YES OR NO -> Chosen value "YES"
then, $PUSH = YES
AS String 1 = $PUSH => String 1 = YES
Again, String 2 = YES, hence String 2 == String 1 (String match)
Then, trigger the Git push action.
```

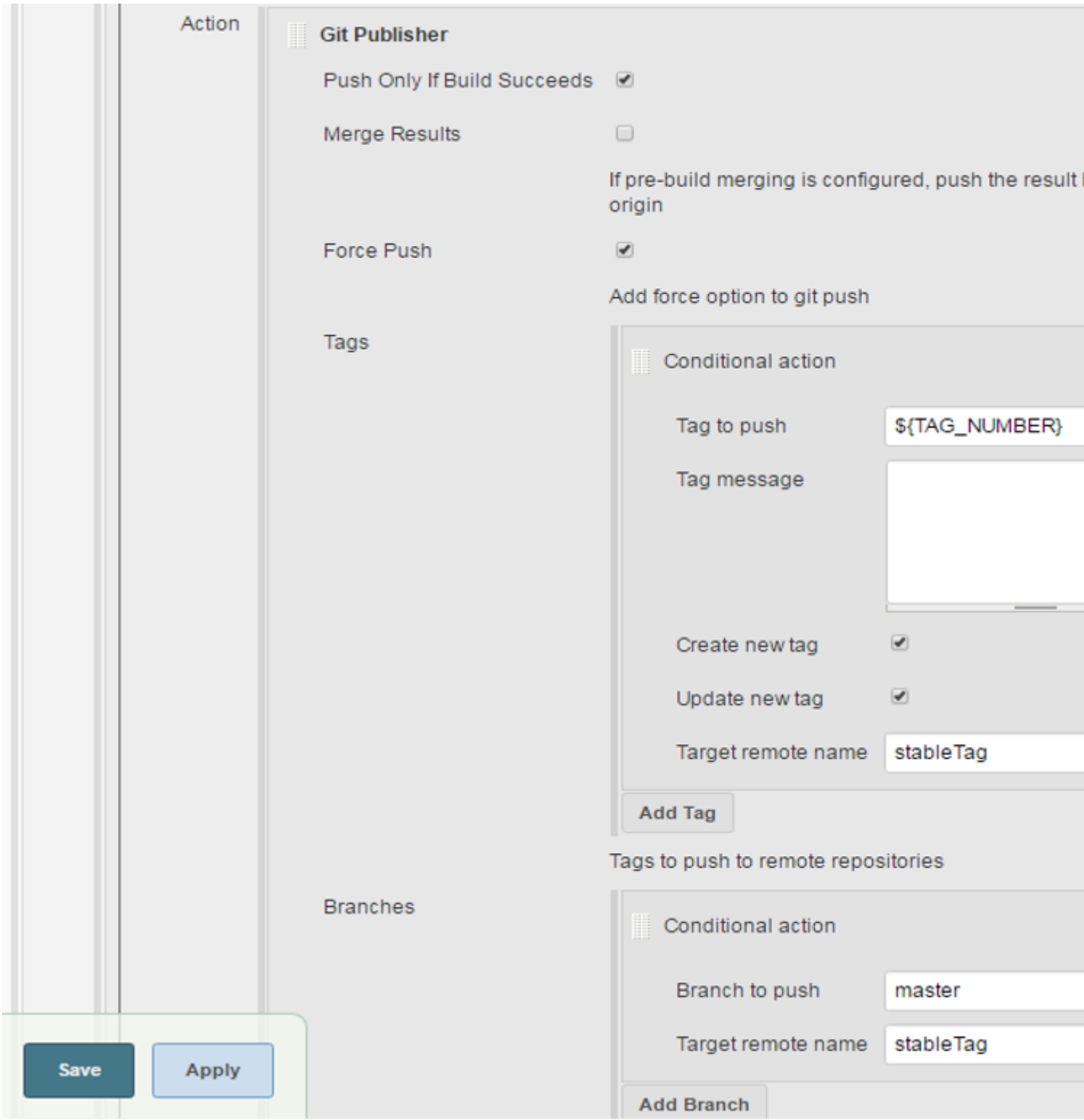


8. Now click on Add dropdown option to add the Git publisher action that will be triggered on the basis of the string match condition.


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Run?
And
X
&&
Strings match
String 1
$PUSH
String 2
YES
Case insensitive
X
&&
Always
Add and condition
Advanced...
Action
Add


Add and condition
Advanced...
Action
Add
Aggregate downstream test results
Archive the artifacts
Build other projects
Discard Old Builds
Publish JUnit test result report
Publish Javadoc
Quality Gates
Record JaCoCo coverage report
Record fingerprints of files to track usage
Git Publisher
SonarQube analysis with Maven
Add conditiona
Add post-build action
Save
Apply

9. After selecting Git Publisher, do the configuration as follows:



After the configuration save the job and you are done.

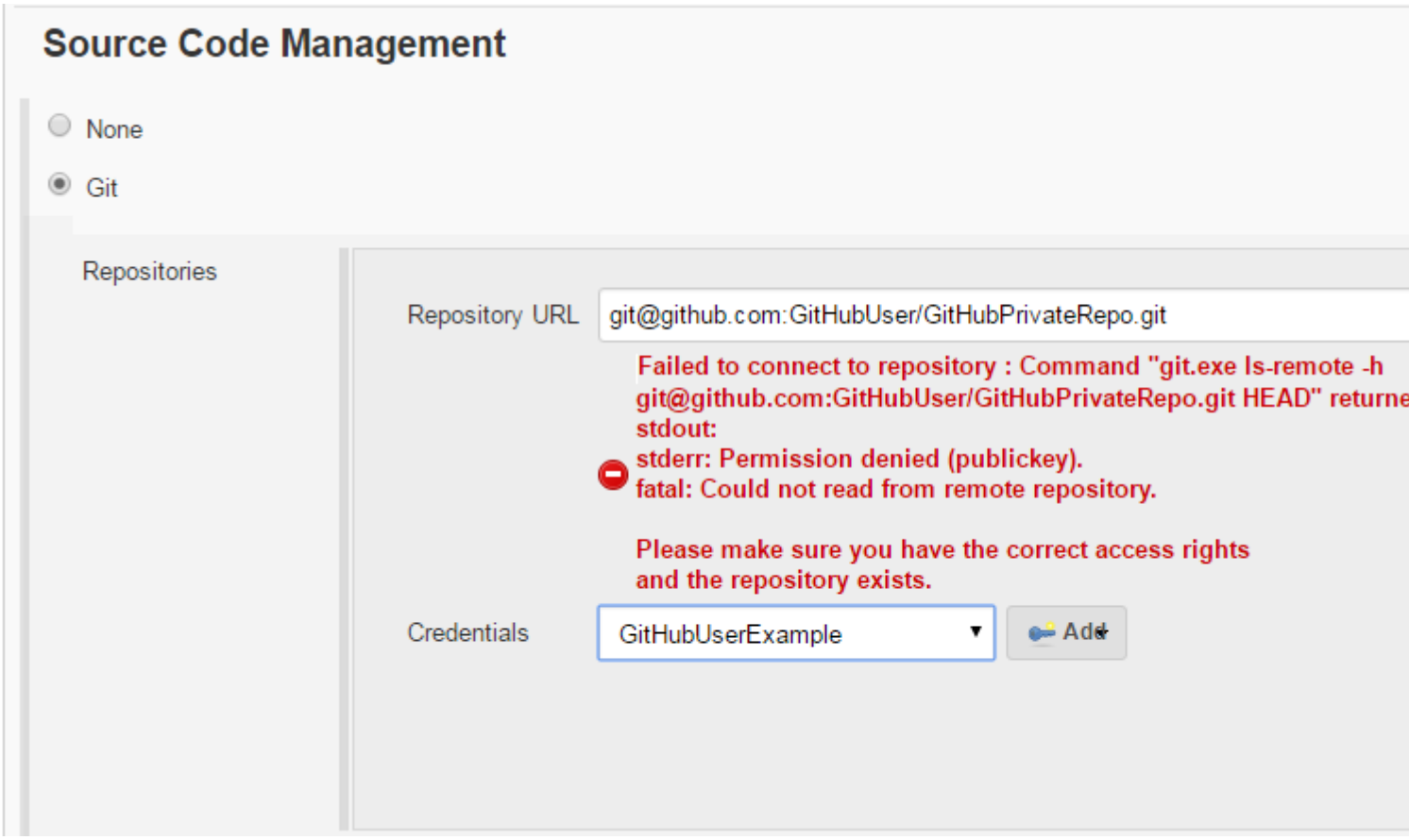
Read Configure Auto Git Push on Successful Build in Jenkins online:
https://riptutorial.com/jenkins/topic/8972/configure-auto-git-push-on-successful-build-in-jenkins

Chapter 3: Install Jenkins on Windows with SSH support for private GitHub repositories

Examples

GitHub pull requests fail

Out of the box installations of Jenkins with the Git and SSH plugins will not work when attempting to pull a private repository from GitHub.



PSExec.exe PS Tool by Microsoft

The first step to fix this issue I found was to download PSTools and extract the tools to a convenient location on the build server (e.g. c:\Programs\PSTools is there I extracted mine).

Generate a new SSH key just for Jenkins using PSExec or PSExec64

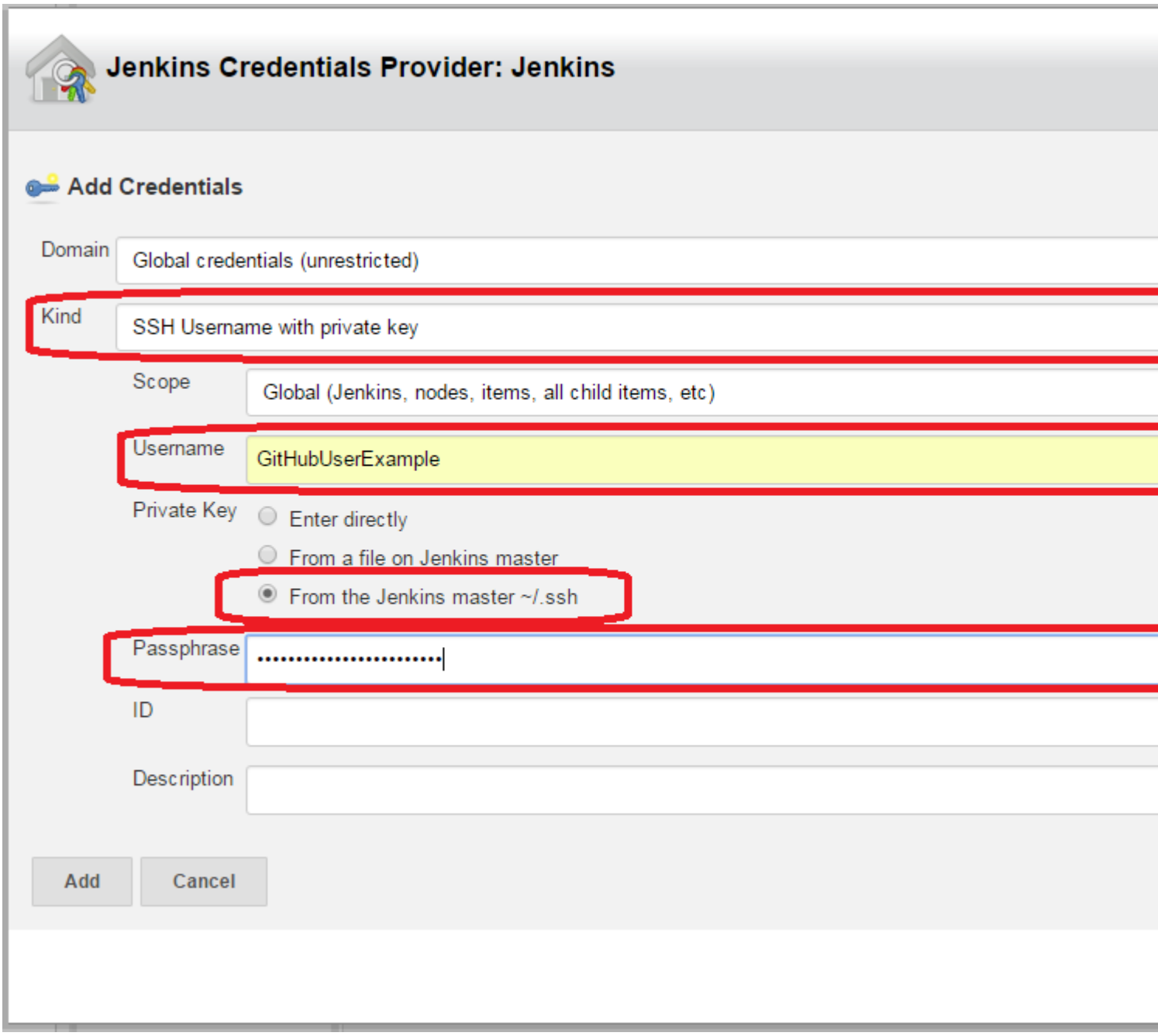
1. First open the Command prompt and "Run as Administrator".
2. Once the command prompt is open navigate to the PSTools directory.
3. From the command prompt we need to run git-bash using PSExec or PSExec64 as the Local Service, which Jenkins is running on the build server as by default.
4. We will use the -i switch to run PSExec as interactive and the -s switch to run git-bash as the local service

5. Follow the instructions for creating a ssh key on GitHub - Generating a new SSH key and adding it to the ssh-agent
6. If you are on a 64bit Windows system, then copy the .ssh folder to C:\Windows\SysWOW64\config\systemprofile.ssh (this was not needed on my 64bit Windows system, but there where some instructions that indicated the .ssh files should be stored there, something to keep in mind if you are having problems still).
7. Add the public SSH key to your github keys.

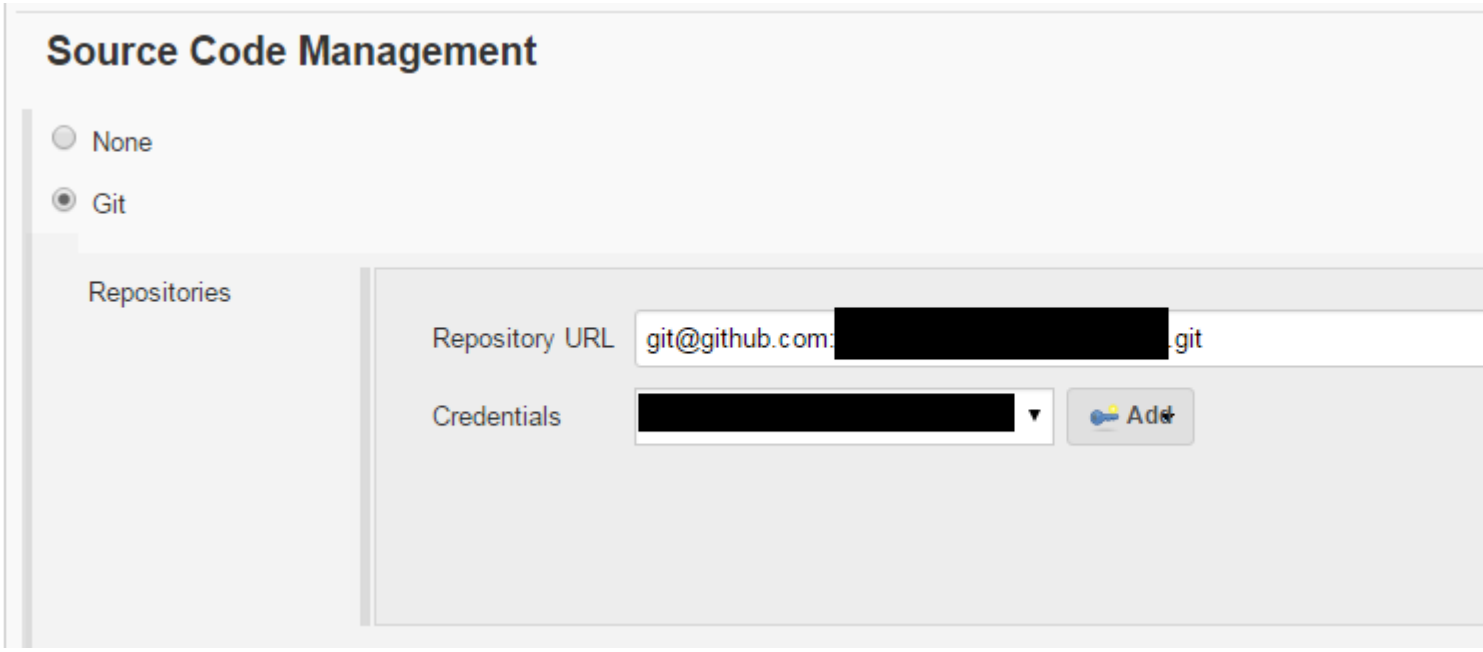
```
Your Commandline should look similar to this:

C:\Programs\PSTools> PSExec.exe -i -s C:\Programs\Git\git-bash
```

Create the Jenkins Credentials

The hard part is over! Now just create the credentials to be used in Jenkins. Use your own Username and the passphrase used to create the SSH Key.

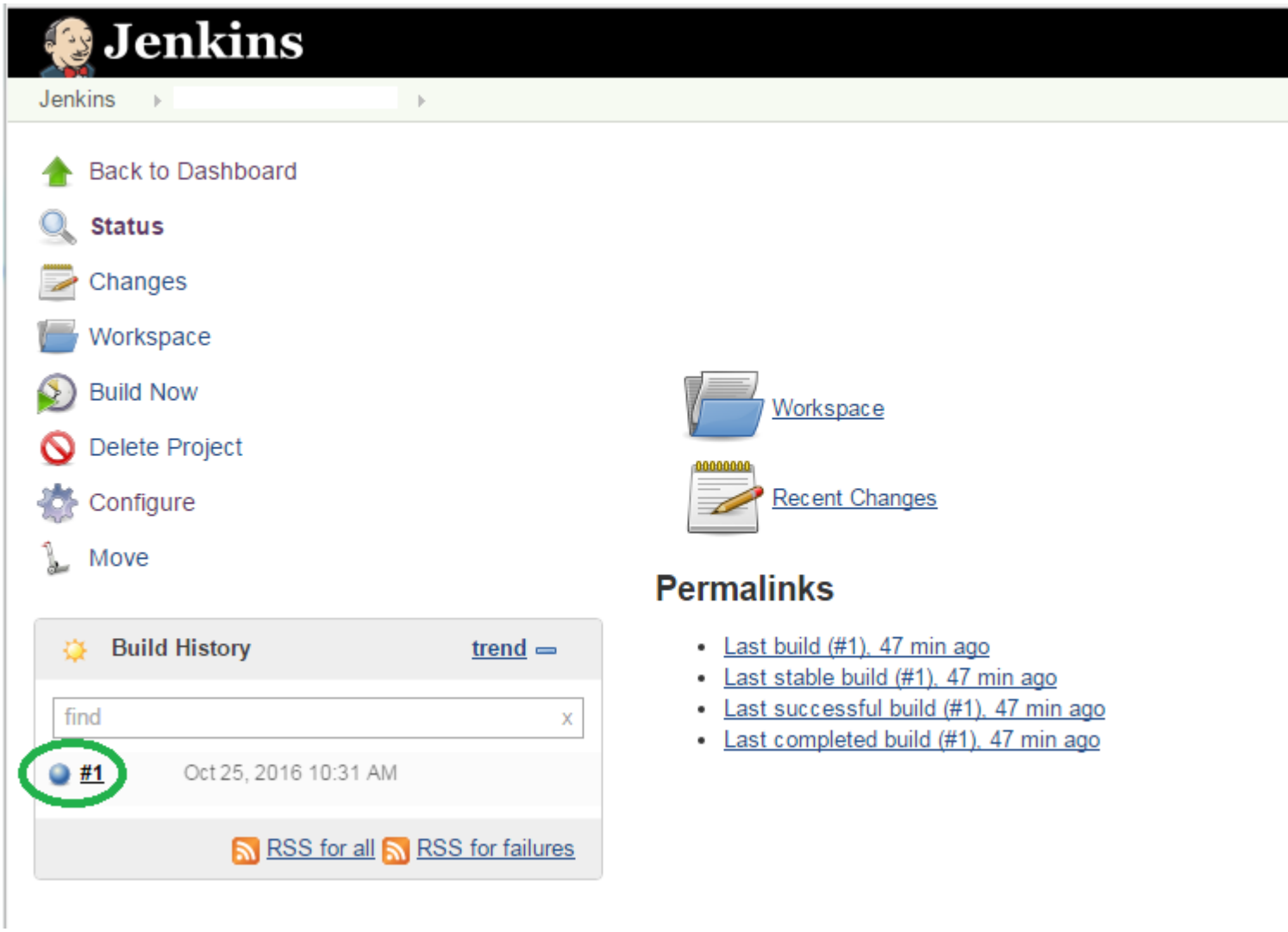


This is what it should look like now (with your own private github repo and user name:



Run a test pull request to verify, and your done.

Save and run a test pull request and your should no longer have any further problems with having Jenkins use SSH on your Windows build machine.



Read Install Jenkins on Windows with SSH support for private GitHub repositories online: https://riptutorial.com/jenkins/topic/7626/install-jenkins-on-windows-with-ssh-support-for-private-github-repositories

Chapter 4: Jenkins Groovy Scripting

Examples

Create default user

1. Create groovy file by path `$JENKINS_HOME/init.groovy.d/basic-security.groovy`

In Ubuntu 16 Jenkins home directory places in `/var/lib/jenkins`

2. Place in file next code

```
#!groovy

import jenkins.model.*
import hudson.security.*

def instance = Jenkins.getInstance()

def hudsonRealm = new HudsonPrivateSecurityRealm(false)

hudsonRealm.createAccount("admin_name","admin_password")
instance.setSecurityRealm(hudsonRealm)
instance.save()
```

3. Restart Jenkins service
4. After Jenkins starts you need to remove `$JENKINS_HOME/init.groovy.d/basic-security.groovy` file

Disable Setup Wizard

1. Open Jenkins default config file and add in `JAVA_ARGS` next key `-Djenkins.install.runSetupWizard=false`

In Ubuntu 16 default file places in `/etc/default/jenkins`

2. Create groovy file by path `$JENKINS_HOME/init.groovy.d/basic-security.groovy`

In Ubuntu 16 Jenkins home directory places in `/var/lib/jenkins`

3. Place in file next code

```
#!groovy

import jenkins.model.*
import hudson.util.*;
import jenkins.install.*;

def instance = Jenkins.getInstance()

instance.setInstallState(InstallState.INITIAL_SETUP_COMPLETED)
```

4. Restart Jenkins service
5. After Jenkins starts you need remove `$JENKINS_HOME/init.groovy.d/basic-security.groovy` file

After this Jenkins doesn`t ask you to confirm that you are admin and you will not see plugins install page.

How to get infromation about Jenkins instance

Open your jenkins instance script console http://yourJenkins:port/script following is an example for how to get information about this instance. copy the code to the console and click "Run".

```
/* This scripts shows how to get basic information about Jenkins instance */
def jenkins = Jenkins.getInstance()
println "Jenkins version: ${jenkins.getVersion()}"
println "Available JDKs: ${jenkins.getInstance().getJDKs()}"
println "Connected Nodes:"
jenkins.getNodes().each{
  println it.displayName
}
println "Configured labels: ${jenkins.getLabels()}"
```

In this example you will see information about the Jenkins version, JDKs, agents(slaves) and labels.

How to get information about a Jenkins job

Open your jenkins instance script console http://yourJenkins:port/script following is an example for how to get information about a sepcific job. copy the code to the console, change the jobName to the required job and click "Run".

```
/*This script shows how to get basic information about a job and its builds*/
def jenkins = Jenkins.getInstance()
def jobName = "myJob"
def job = jenkins.getItem(jobName)

println "Job type: ${job.getClass()}"
println "Is building: ${job.isBuilding()}"
println "Is in queue: ${job.isInQueue()}"
println "Last successfull build: ${job.getLastSuccessfulBuild()}"
println "Last failed build: ${job.getLastFailedBuild()}"
println "Last build: ${job.getLastBuild()}"
println "All builds: ${job.getBuilds().collect{ it.getNumber()}}"
```

first we get the Jenkins instance object, then using this instance we get the job object (item). from the job object we can get different information such as: is it currently building, is it in the queue, the last build, last build by status, and a lot more.

Read Jenkins Groovy Scripting online: https://riptutorial.com/jenkins/topic/7562/jenkins-groovy-scripting

Chapter 5: Role Strategy Plugin

Examples

Configuration

Manage Roles

Global Roles- Create roles with selected set of Jenkins features e.g. Usually for a development project, 2 roles can be created.

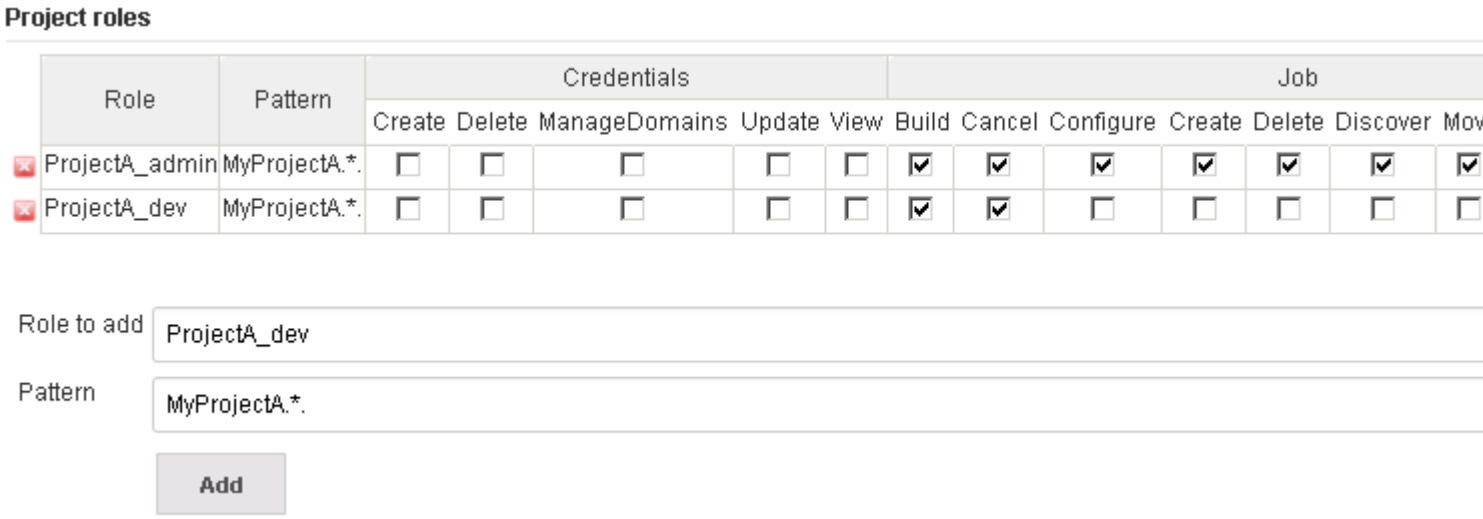
1. Developer- Global role can be set to only **Overall** : Read
2. ProjectOwner- Global role can be set to **Overall** : Read

This restricts developer and project owner to read access to all Jenkins features.

Project Roles- Create roles by restricting user access respective jenkins job and credential features using regular expressions.

E.g. for a development project 'MyProjectA'; project owners needs to have full permissions to Jobs and developers need Build access to Jenkins jobs. So we create below roles:

- **ProjectA_admin**- check all options under Job viz. *Build, Cancel, Configure, Create, Delete, Discover, Move, Read, Workspace*
- **ProjectA_dev** - check options Build, Cancel, Read, Workspace under Job



To restrict above projects to respective project owners and developers, all jobs must follow a pre-defined pattern.

Assume 'MyProjectA' needs 3 jenkins build jobs: *MyProjectA_Dev_Build, MyProjectA_QA_Build, MyProjectA_Nightly_Sonar_Analysis*

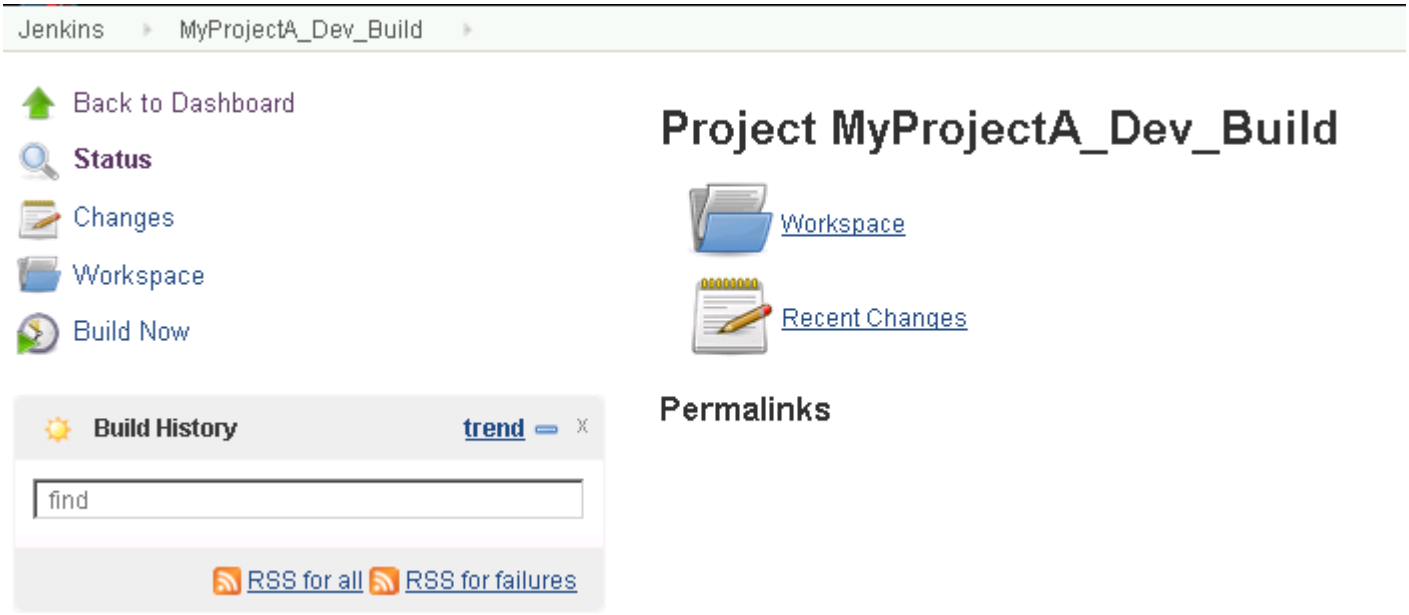
To restrict project owner and developers of project 'MyProjectA' to above build jobs, provide '*Pattern*' as **MyProjectA.***.

Assign Roles

Helps to assign users or project groups to respective Global or Project roles. E.g. to assign a developer 'Gautam' to Developer global role, provide the user name 'Gautam', click *Add* and select the check box next to 'Gautam' and below Developer global role.

Similarly add the user under project roles and select respective project roles to assign required project roles.

If you notice below screenshots you can see user 'gautam' has access only to projects starting with MyProjectA. Also, user's access is restricted to build and configure is missing.



Read Role Strategy Plugin online: https://riptutorial.com/jenkins/topic/5741/role-strategy-plugin

Chapter 6: Setting up Build Automation for iOS using Shenzhen

Examples

iOS Build Automation Setup using Shenzhen

Part I : Setup the Mac machine to use shenzhen

Go to terminal

Install Shenzhen

> sudo gem install shenzhen
>
> sudo gem install nomad-cli

Download XCode command line utility

> xcode-select --install

Popup shows up with the below text

> The xcode-select command requires the command line developer tools. Would you like to install the tools now?”

Click - Install

Create project directory

gitclone your project

> git clone https://akshat@bitbucket.org/company/projectrepo.git

Build project using below command

> ipa build --verbose

PS: If you see any errors please select the Active Provisioning Profile and commit to the project files. and perform `ipa build --verbose` again.

Read Setting up Build Automation for iOS using Shenzhen online: https://riptutorial.com/jenkins/topic/8002/setting-up-build-automation-for-ios-using-shenzhen

Chapter 7: Setting up Jenkins for iOS build automation.

Introduction

Now you can define Continuous Integration and Continuous Delivery (**CI/CD**) process as code with Jenkins 2.0 for your projects in iOS 10. Activities like to build, test, code coverage, check style, reports, and notifications can be described in only one file.

To read the complete article go to Pipeline in Jenkins 2.0 as Code for iOS 10 and XCode 8

Parameters

Parameter	Details
node('iOS Node')	Jenkins Node with Mac OS. If Jenkins is installed in Mac OS use `node {....}`

Remarks

The article is written in both languages: English and Spanish.

Examples

Time Table Example

The source code can be cloned or downloaded from GitHub to test it.

```
node('iOS Node') {

    stage('Checkout/Build/Test') {

        // Checkout files.
        checkout([
            $class: 'GitSCM',
            branches: [[name: 'master']],
            doGenerateSubmoduleConfigurations: false,
            extensions: [], submoduleCfg: [],
            userRemoteConfigs: [[
                name: 'github',
                url: 'https://github.com/mmorejon/time-table.git'
            ]]
        ])

        // Build and Test
        sh 'xcodebuild -scheme "TimeTable" -configuration "Debug" build test -destination
```

```
"platform=iOS Simulator,name=iPhone 6,OS=10.1" -enableCodeCoverage YES |
/usr/local/bin/xcpretty -r junit'

        // Publish test restults.
        step([$class: 'JUnitResultArchiver', allowEmptyResults: true, testResults:
'build/reports/junit.xml'])
    }

    stage('Analytics') {

        parallel Coverage: {
            // Generate Code Coverage report
            sh '/usr/local/bin/slather coverage --jenkins --html --scheme TimeTable
TimeTable.xcodeproj/'

            // Publish coverage results
            publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false,
reportDir: 'html', reportFiles: 'index.html', reportName: 'Coverage Report'])


        }, Checkstyle: {

            // Generate Checkstyle report
            sh '/usr/local/bin/swiftlint lint --reporter checkstyle > checkstyle.xml || true'

            // Publish checkstyle result
            step([$class: 'CheckStylePublisher', canComputeNew: false, defaultEncoding: '',
healthy: '', pattern: 'checkstyle.xml', unHealthy: ''])
        }, failFast: true|false
    }

    stage ('Notify') {
        // Send slack notification
        slackSend channel: '#my-team', message: 'Time Table - Successfully', teamDomain: 'my-
team', token: 'my-token'
    }
}
```

Read Setting up Jenkins for iOS build automation. online:
https://riptutorial.com/jenkins/topic/8868/setting-up-jenkins-for-ios-build-automation-

Credits

S. No	Chapters	Contributors
1	Getting started with jenkins	acalb, Community, Gautam Jose, Girish Kumar, Katu, Pablo, Pom12, Priyanshu Shekhar, Rogério Peixoto, S.K. Venkat, Seb, Tyler
2	Configure Auto Git Push on Successful Build in Jenkins	ANIL
3	Install Jenkins on Windows with SSH support for private GitHub repositories	Riana
4	Jenkins Groovy Scripting	RejeeshChandran, serieznyi, Tidhar Klein Orbach
5	Role Strategy Plugin	Gautam Jose
6	Setting up Build Automation for iOS using Shenzhen	Ichthyocentaurs
7	Setting up Jenkins for iOS build automation.	Manuel Morejón