OSCON 2016

Creating a Deployment Pipeline with Jenkins

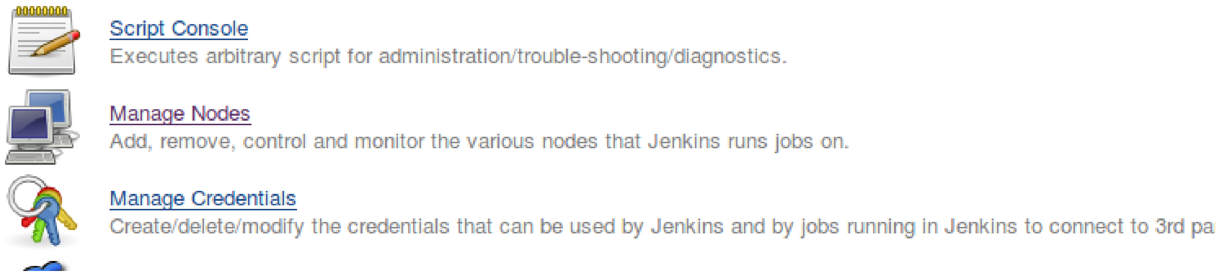
© 2016 Brent Laster

Lab 1 - Creating a Jenkins Slave Node

1. Start Jenkins by clicking on the “Jenkins on localhost” shortcut on the desktop OR opening the Firefox browser and navigating to “http://localhost:8080”.

2. Log in to Jenkins by clicking on the “log in” link in the upper right corner. User = diyuser and Password = diyuser (Note: If at some point during the workshop you try to do something in Jenkins and find that you can’t, check to see if you’ve been logged out. Log back in if needed.)

3. Click on the “Manage Jenkins” link in the menu on the left-hand side. Next, look in the list of selections in the middle of the screen, and find and click on “Manage Nodes”.



Script Console
Executes arbitrary script for administration/trouble-shooting/diagnostics.

Manage Nodes
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Manage Credentials
Create/delete/modify the credentials that can be used by Jenkins and by jobs running in Jenkins to connect to 3rd pa

4. On the next screen, click on “New Node” on the menu on the left-hand side.

5. For Node name, type in

pipeline_slave

Click on the “Dumb Slave” radio button. Then click on “OK”.

6. You should now be on the configuration screen for the new node. Find the “Remote root directory” text field. Type in the following:

/home/jenkins/pipeline_slave

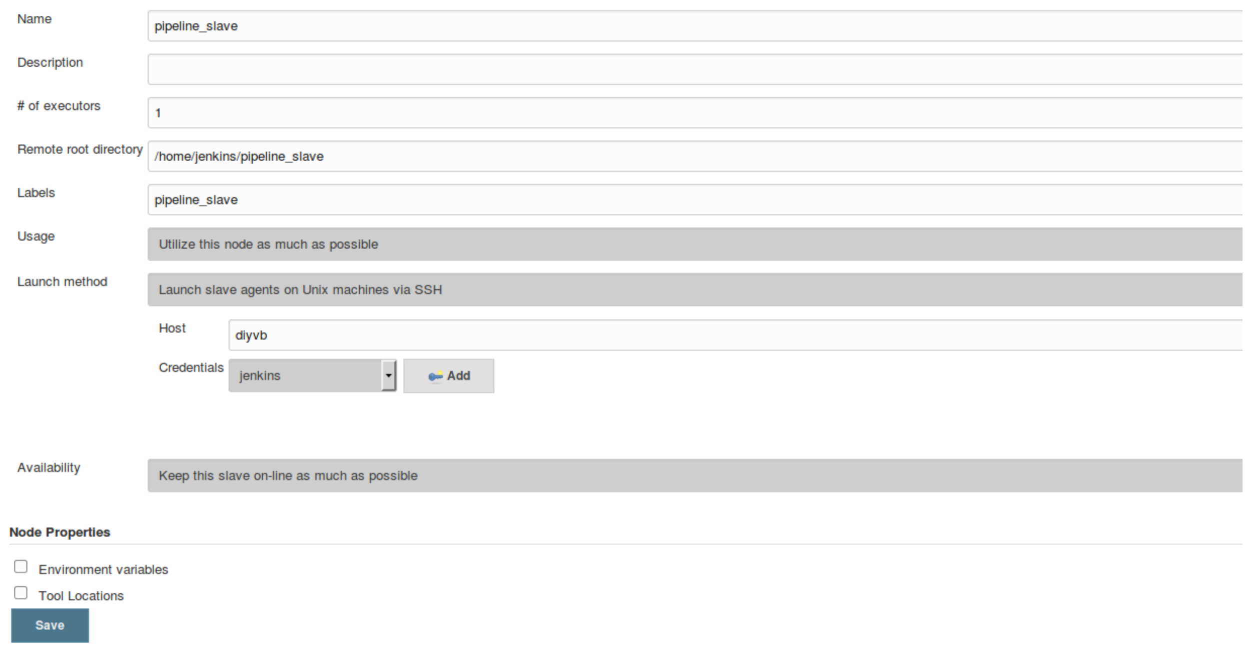
7. Under that, fill in the “Labels” area with

   pipeline_slave

8. Under the “Launch method” section, for “Host” enter the name of this machine:

   diyvb

   In the Credentials drop-down, select “jenkins”. (If you see more than one, use the first jenkins in the list.)



Name	pipeline_slave
Description	
# of executors	1
Remote root directory	/home/jenkins/pipeline_slave
Labels	pipeline_slave
Usage	Utilize this node as much as possible
Launch method	Launch slave agents on Unix machines via SSH
Host	diyvb
Credentials	jenkins
Availability	Keep this slave on-line as much as possible

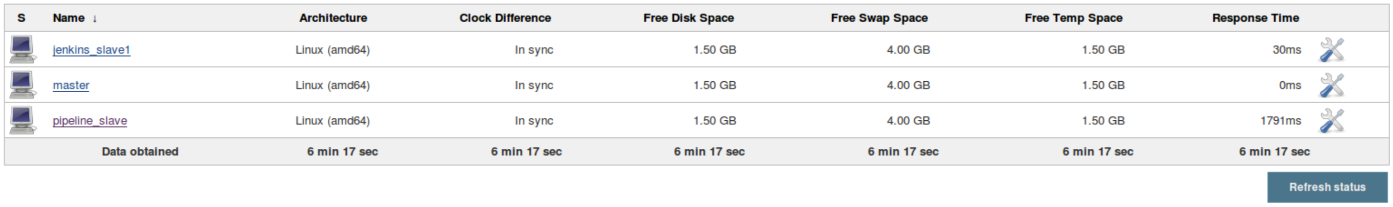
Node Properties

☐ Environment variables

☐ Tool Locations

9. Click on Save.

10. You should see a note that indicates the slave is being launched. In the upper right corner, click the “Enable Auto Refresh” link to have the screen automatically updated.



S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	jenkins_slave1	Linux (amd64)	In sync	1.50 GB	4.00 GB	1.50 GB	30ms	
	master	Linux (amd64)	In sync	1.50 GB	4.00 GB	1.50 GB	0ms	
	pipeline_slave	Linux (amd64)	In sync	1.50 GB	4.00 GB	1.50 GB	1791ms	
	Data obtained	6 min 17 sec	6 min 17 sec	6 min 17 sec	6 min 17 sec	6 min 17 sec	6 min 17 sec	

11. Click on “Back to List” to see the slave node in the list. Then click on “Back to Dashboard” to get back to the Jenkins Dashboard.

12. There are several different “views” already setup for the jobs we will use. For now, we can look at the “(1) Workshop Pipeline View”. Click that tab.

Lab 2 - Setting up the Review phase

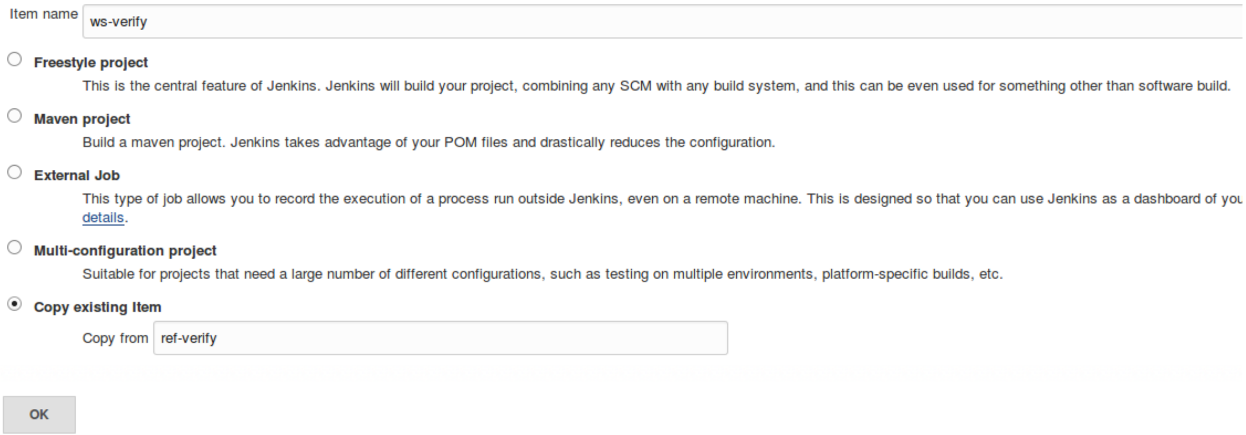
1. We want to create a new job for the verify phase of our workshop pipeline. We’ll do that by copying the reference verify job (ref-verify) and modifying it as needed.

2. On the Jenkins dashboard at http://localhost:8080, click on “New Item” in the upper left corner. For item-name, enter

ws-verify

Select “Copy existing item”. In the “Copy from” field, enter

ref-verify



Item name ws-verify

○ **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.

○ **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 you
details.

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

● **Copy existing Item**
Copy from ref-verify

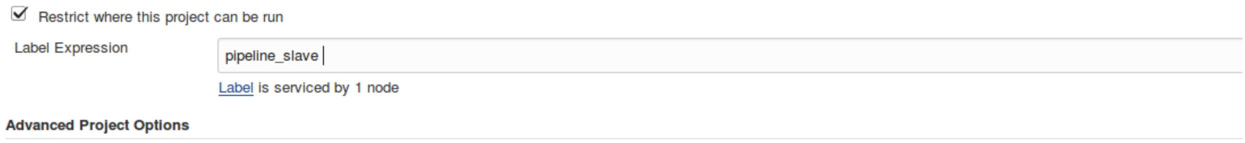
2. Click on the OK button. You will now be in the configure screen for the new ws-verify job.

3. Find the section on “Advanced Project Options”. Right above that line, you’ll see the “Restrict where this project can be run” option.

In the “Label Expression” text box, type in the name of the slave node we created in Lab 1.

pipeline_slave

(Note: If you happened to name your slave node something different, then you would use that name here. If you did that, you’ll need to modify the name in each job to be the name you used.)



☑ Restrict where this project can be run

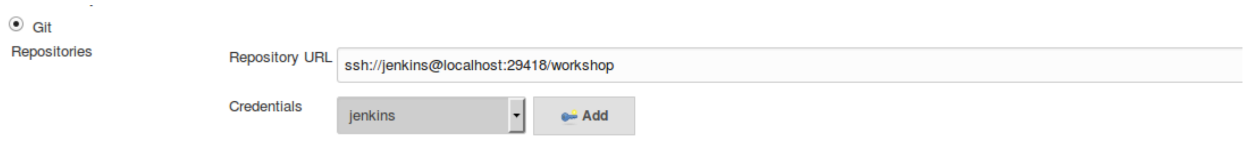
Label Expression: pipeline_slave

Label is serviced by 1 node

Advanced Project Options

4. For the workshop pipeline, we will be using a different source management repository so that needs to be updated. Find the Source Code Management section and change the path of the source code repository by replacing “reference” with “workshop”. The path should look like

ssh://diyuser@localhost:29418/workshop



● Git

Repositories

Repository URL: ssh://jenkins@localhost:29418/workshop

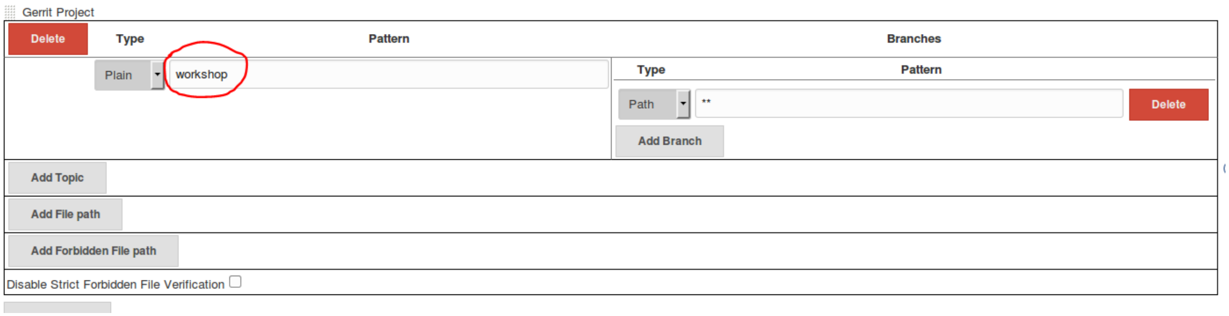
Credentials: jenkins

5. Locate the “Gerrit Trigger” section under the “Build Triggers” section on the page. This section is telling Jenkins to kick things off based on a new patchset being created in Gerrit (thus the “Patchset Created” setting in the “Trigger on” area.

What we need to change here is the project that Gerrit will be using and that Jenkins will trigger on.

6. Locate the “Gerrit Project” section (just below “Dynamic Trigger Configuration”). Under “Type” on the left-hand side (next to the red “Delete” button), select “Plain” and in the text field, enter

workshop



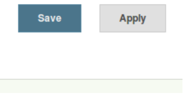
Gerrit Project

Type	Pattern	Branches
Plain	workshop	Type: Path Pattern: **

Disable Strict Forbidden File Verification □

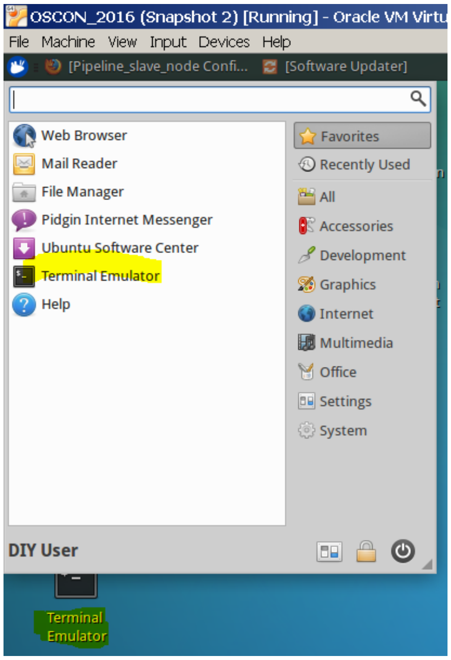
7. Find the “Build” section and notice that we’re invoking a local version of Gradle as the build step and passing it a task of “build”. We’ll be talking more about Gradle and how this all fits in as we go along.

8. Save your changes by clicking on the Save button at the bottom of the screen.



Lab 3 - Pushing to Gerrit for Review

1. Open up a terminal session by either clicking on the “Terminal Emulator” shortcut on the desktop or by selecting the “Terminal Emulator” selection from the system menu (drop down from the mouse in the upper-lefthand side).



2. Once the terminal session opens, cd to the **workshop** directory.

```
cd workshop
```

3. In this directory, we already have a cloned version of the project that we will update and push out to Gerrit for the Jenkins verify build and code review.

4. Under the “workshop” directory, in the “web/src/main/webapp” subdirectory, edit the “agents.html” file.

(You can use mousepad or gedit editors.)

```
gedit web/src/main/webapp/agents.html
```

Find the line for the main header (about 20 lines down).

```
<h1>R.O.A.R (Registry of Animal Responders) Agents</h1>
```

Add a second header line after it substituting in your name. Be sure to include the html tabs (<h2> and </h2>).

```
<h2> Managed by ( your name here ) </h2>
```

5. Save your file (File->Save) and quit/exit the editor.

6. From the workshop directory, do a local build to make sure things compile.

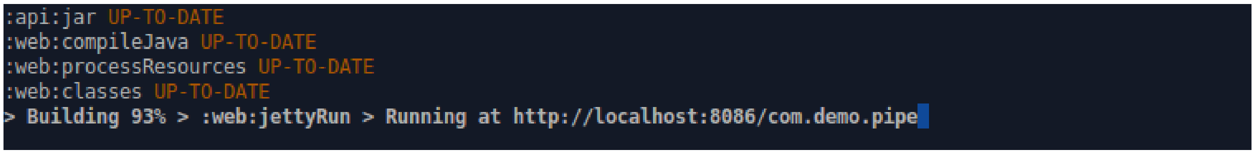
```
gradle build
```

This should finish with a "BUILD SUCCESSFUL" message.

7. Run the local instance to make sure the changes look as expected. Do this by running the jettyRun task (note case) and then opening a browser to **http://localhost:8086/com.demo.pipeline**.

gradle jettyRun

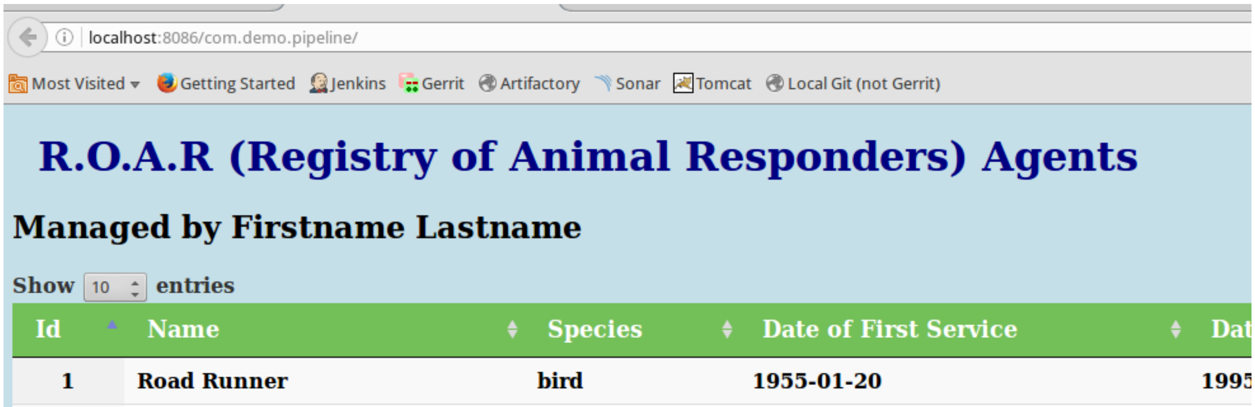
(Note: This will get to some percentage and then start the system running. Don't kill this - it is working.)



```
:api:jar UP-TO-DATE
:web:compileJava UP-TO-DATE
:web:processResources UP-TO-DATE
:web:classes UP-TO-DATE
> Building 93% > :web:jettyRun > Running at http://localhost:8086/com.demo.pipe
```

<open in browser tab> **http://localhost:8086/com.demo.pipeline**

NOTE: note that the URL is pipeline on the end not just pipe as shown.



R.O.A.R (Registry of Animal Responders) Agents

Managed by Firstname Lastname

Show 10 entries

Id	Name	Species	Date of First Service	Dat
1	Road Runner	bird	1955-01-20	1995

8. Close the browser tab (not the entire session) and kill the running job in the terminal **(Ctrl-C) .**

9. Since we've verified that things work as expected, push the change over to the Gerrit remote. First configure your user.name. (user.email is already configured) Be careful of the typing. Note that there are two dashes/hyphens before global and user.name is two words with only a period inbetween. Also the name string should be enclosed in quotes since it has a space.

```
git config --global user.name “Firstname Lastname”
```

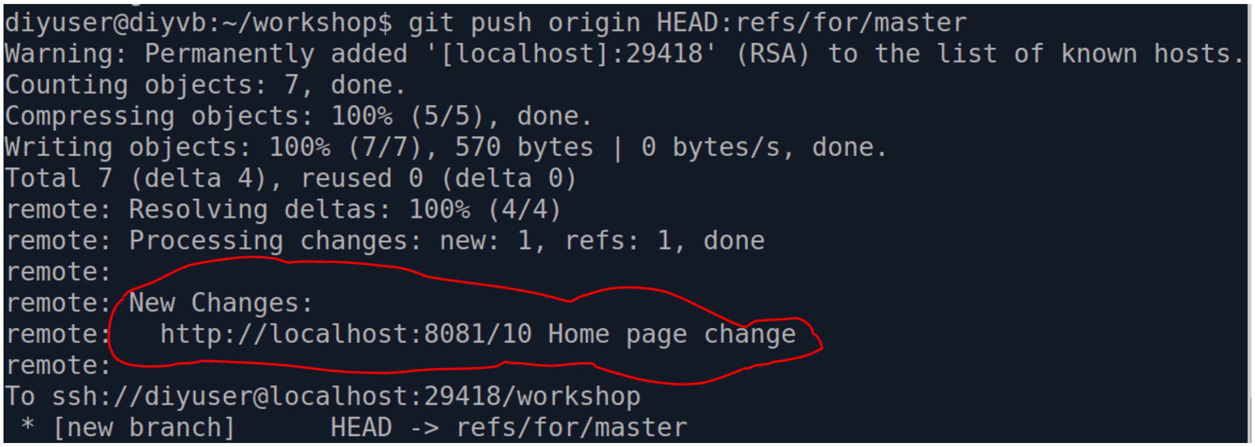
10. Now add and commit the change. This assumes again that you are still in the workshop directory. (Ignore any messages about line ending changes.)

```
git add web/src/main/webapp/agents.html

git commit -m “home page change”
```

11. Now push the change to Gerrit. Note the syntax - that's “HEAD”, then a colon, then “refs/for/master” - all together.

```
git push origin HEAD:refs/for/master
```

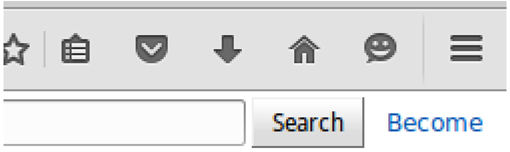
This will create a new change and a new patchset. This is not in the remote repository yet. It's being held by Gerrit for review. You can find the full link to the new change and it's first patchset (revision) in the push commands' output. Look for something like shown below.

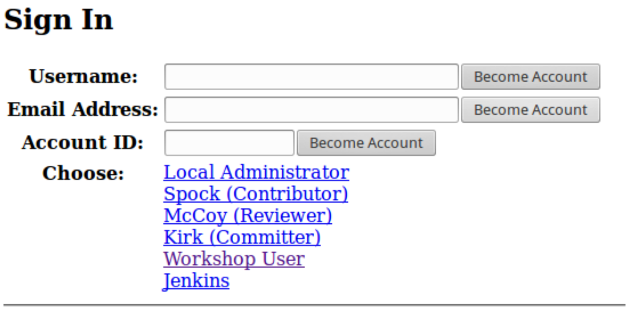


```
diyuser@diyvb:~/workshop$ git push origin HEAD:refs/for/master
Warning: Permanently added '[localhost]:29418' (RSA) to the list of known hosts.
Counting objects: 7, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 570 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4)
remote: Processing changes: new: 1, refs: 1, done
remote:
remote: New Changes:
remote:   http://localhost:8081/10 Home page change
remote:
To ssh://diyuser@localhost:29418/workshop
 * [new branch]      HEAD -> refs/for/master
```

12. Open up the Gerrit website in a separate tab in the browser at **http://localhost:8081**. (There's also a shortcut in the bookmarks toolbar named "Gerrit"). This session is configured in a debug mode that lets us easily switch between preconfigured users.

13. In the upper right corner, click on the Become link. Then click on "Workshop User".





Sign In

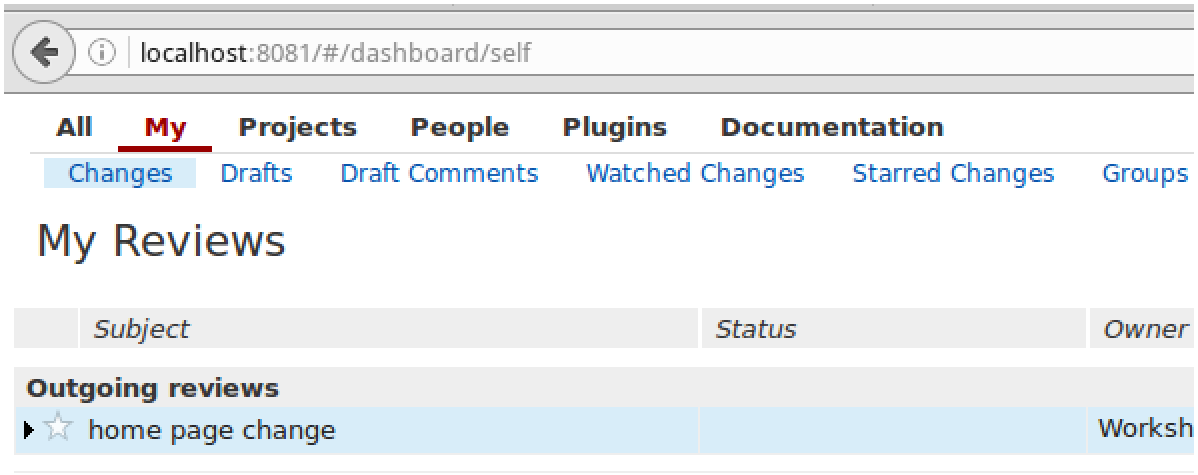
Username: [ ] Become Account

Email Address: [ ] Become Account

Account ID: [ ] Become Account

Choose: Local Administrator
Spock (Contributor)
McCoy (Reviewer)
Kirk (Committer)
Workshop User
Jenkins

14. You should see the change you just pushed under the "Outgoing reviews" section. (If you don't see it, click on My->Changes in the menu.)

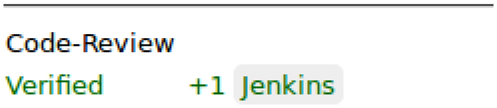


My Reviews

Subject	Status	Owner
Outgoing reviews		
☆ home page change		Worksh

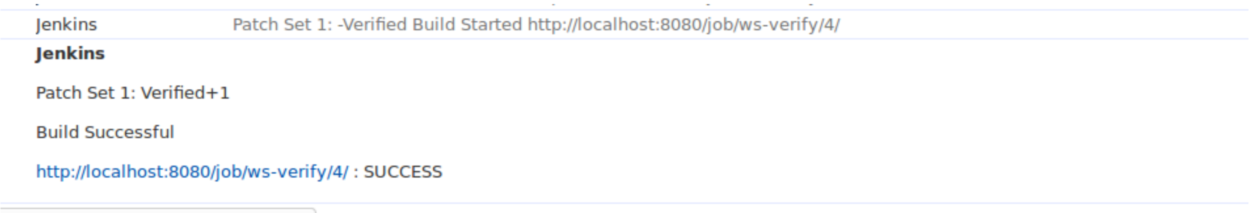
Click on the commit message there (“home page change”) to open up the review.

15. On this page, near the middle right, find the section with the verification checks for Code-Review and Verified. The Verified +1 means that Jenkins did a verification build on this change. The absence of any vote next to Code-Review means it is waiting on a code-review. (We will take care of that in the next lab.)



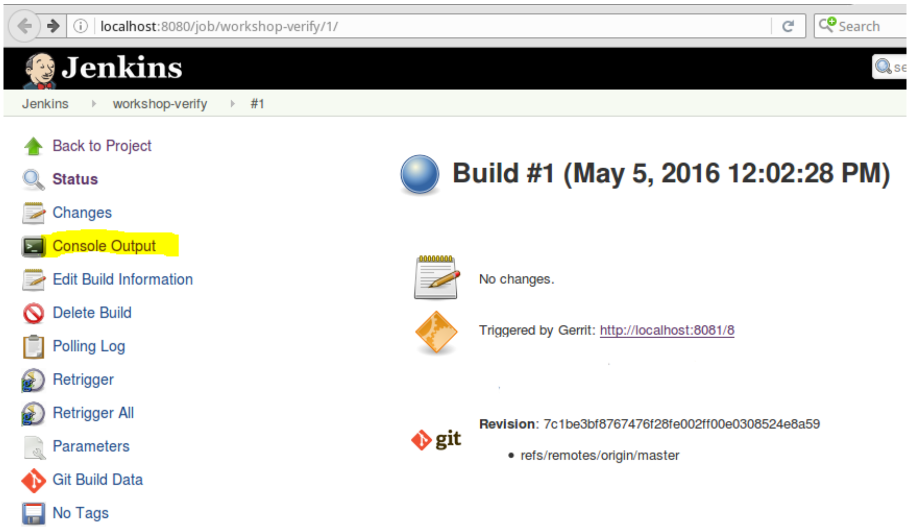
Code-Review		
Verified	+1	Jenkins

16. Look at the history at the bottom of this change window. There are lines for the Jenkins verification build. Click on the line with the Jenkins “Build Started” link to expand it and then **click on that link.**



```
Jenkins          Patch Set 1: -Verified Build Started http://localhost:8080/job/ws-verify/4/
Jenkins

Patch Set 1: Verified+1

Build Successful

http://localhost:8080/job/ws-verify/4/ : SUCCESS
```

This will take you to a build output page in Jenkins for that run of your workshop-verify job.



Build #1 (May 5, 2016 12:02:28 PM)

No changes.

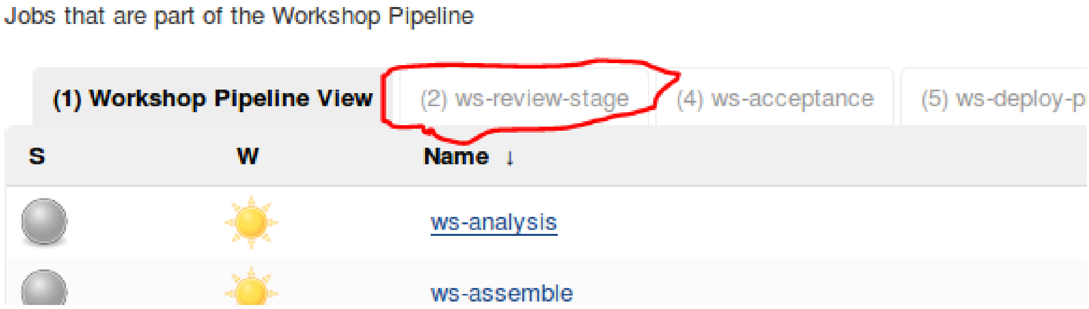
Triggered by Gerrit: http://localhost:8081/8

Revision: 7c1be3bf8767476f28fe002ff00e0308524e8a59

- refs/remotes/origin/master

17. You can click on the “Console Output” link to see the actual output of the build step.

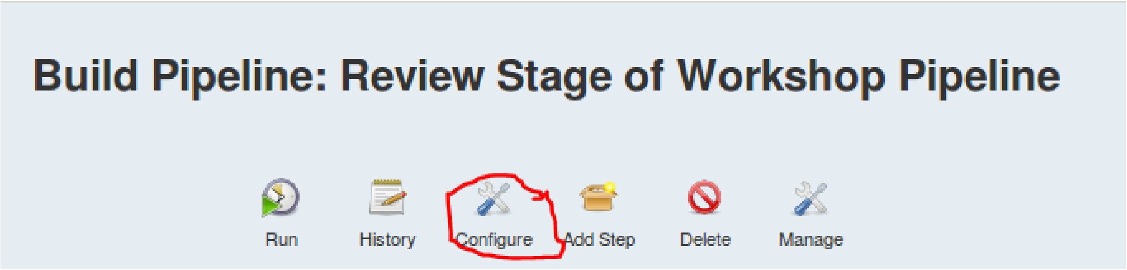
18. One last step here is that we need to update the ws-review-stage starting job (since we now have a job we can use). Back on the Jenkins homepage, in the row of tabs above the jobs, select the tab for “(2) ws-review-stage”.



Jobs that are part of the Workshop Pipeline

S	W	Name ↓
●	☀	ws-analysis
●	☀	ws-assemble

19. On the next screen, select the “Configure” icon.

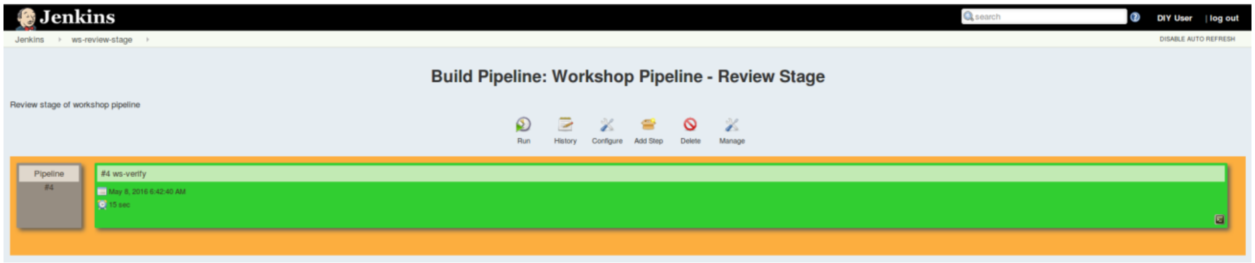


Build Pipeline: Review Stage of Workshop Pipeline

20. On the configuration page, find the section for “Select Initial Job”. Change that from “ref-verify” to “ws-verify”.

21. Save your changes.

22. You should now be able to go back and see the pipeline view of the review stage. Go to the main Jenkins page (http://localhost:8080) and click on the tab labelled “ws-review-stage” in the list of tabs across the top.



Build Pipeline: Workshop Pipeline - Review Stage

Review stage of workshop pipeline

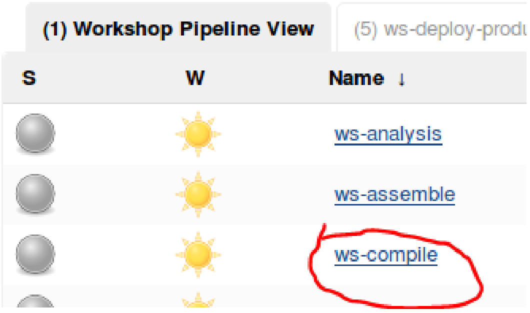
#4 ws-verify

May 8, 2016 6:42:40 AM

15 sec

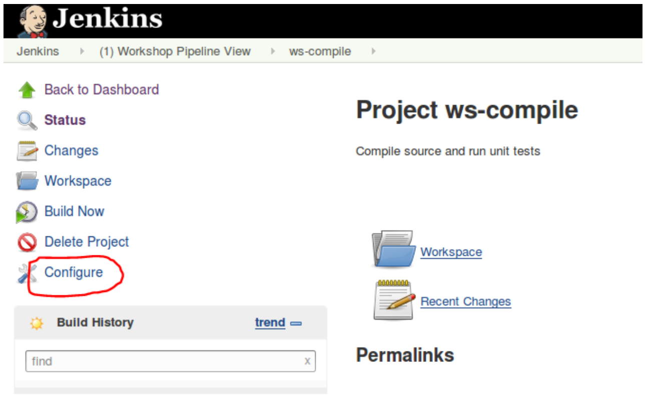
Lab 4. Setting up the Compile Job

1. Start out on the Jenkins dashboard and click the link in the job list to open up ws-compile. This is the job in our pipeline that will compile our code and run unit tests against it.



S	W	Name ↓
●	☀	ws-analysis
●	☀	ws-assemble
●	☀	ws-compile

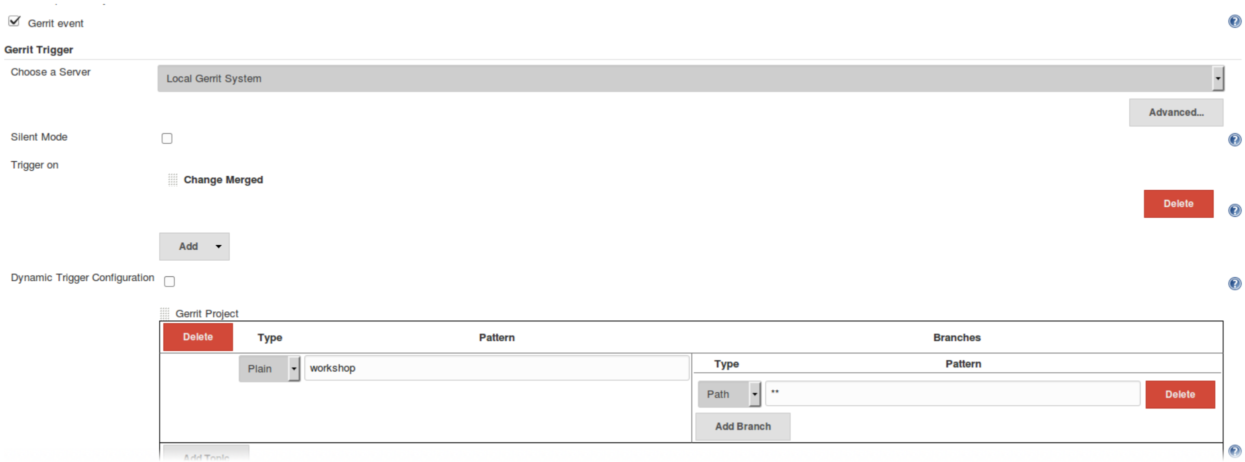
2. Click on the Configure button in the left-hand menu.



Project ws-compile

Compile source and run unit tests

3. On the configuration page find the “Build Triggers”. In this area, under the “Gerrit Triggers” section, notice that we are triggering based on a Gerrit event again. But this time we are using the “Change Merged” event for when content gets reviewed, approved, submitted and merged into the repository. Also notice that under “Dynamic Trigger Configuration” and under “Gerrit Project”, we are looking at the “workshop” project and all branches. Nothing needs changing here.



☑ Gerrit event

Gerrit Trigger

Choose a Server: Local Gerrit System

Silent Mode □

Trigger on: Change Merged

Dynamic Trigger Configuration □

Gerrit Project

Type	Pattern	Branches
Plain	workshop	Type: Path Pattern: **

4. Find the "Build Environment" section. Previously, when we built locally, we relied on cached versions of the dependencies we needed in the local gradle cache on the disk. Now we want to use "official" versions of artifacts out of our Artifactory instance. We will let Jenkins handle the Gradle-Artifactory integration rather than us doing it in our Gradle script. To do this, enable (check) the "Gradle-Artifactory Integration" checkbox.

Check the box for "Gradle-Artifactory Integration".

5. Under "Deployment Details", (if not already set) go ahead and set the "**Artifactory deployment server**" to our local one at

http://localhost:8082/artifactory

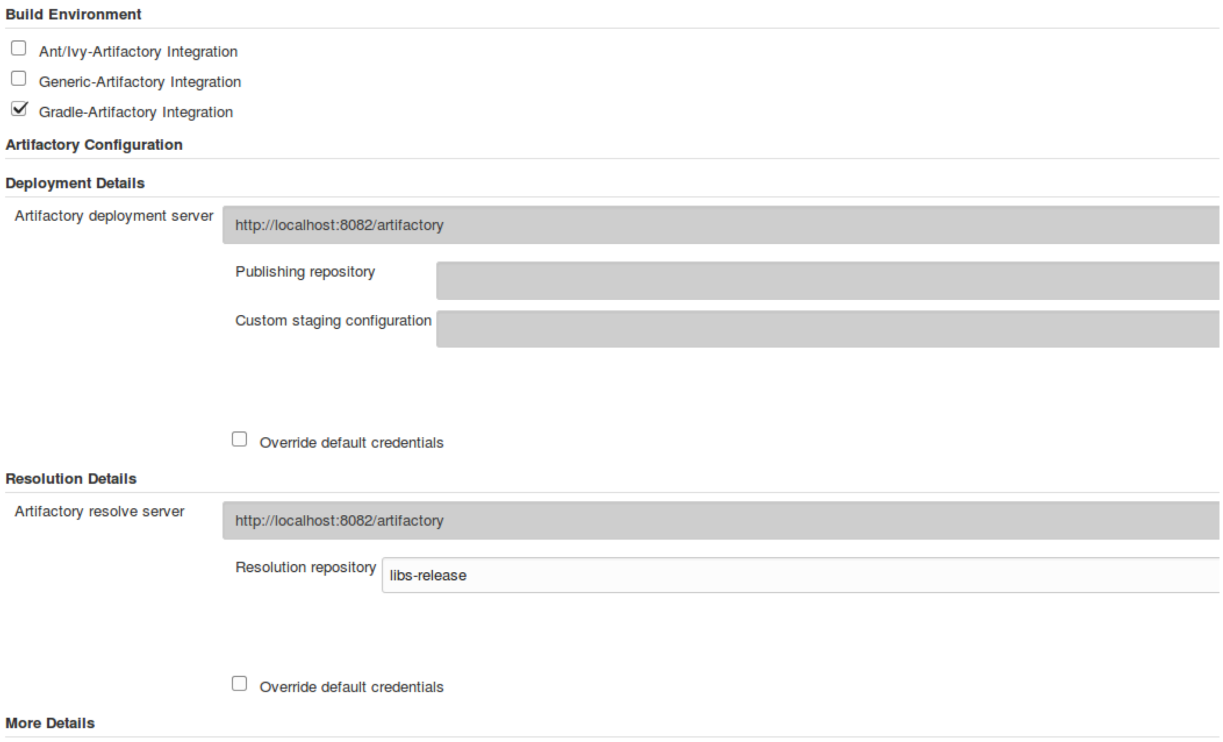
(Even though we aren't deploying anything in this job, it will avoid an error message.)

6. Under "Resolution Details", (if not already set) set the "**Artifactory resolve server**" to the same value:

http://localhost:8082/artifactory

- Since we will be resolving artifacts from our Artifactory system, we need to specify a repository to get the artifacts from. In the "**Resolution repository**" field, select the "**Different Value**" button at the end and then type in

libs-release



Build Environment

□ Ant/Ivy-Artifactory Integration

□ Generic-Artifactory Integration

☑ Gradle-Artifactory Integration

Artifactory Configuration

Deployment Details

Artifactory deployment server: http://localhost:8082/artifactory

Publishing repository

Custom staging configuration

□ Override default credentials

Resolution Details

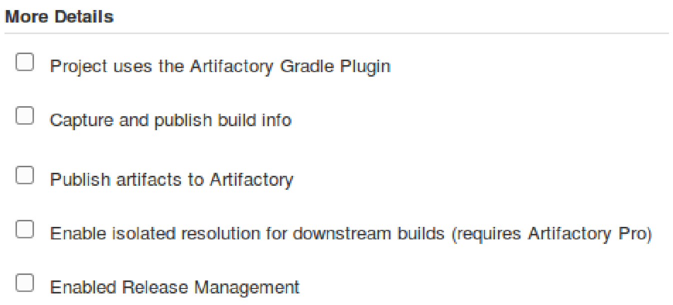
Artifactory resolve server: http://localhost:8082/artifactory

Resolution repository: libs-release

□ Override default credentials

More Details

7. Under “**More Details**” uncheck all the check boxes since we’re not publishing anything.



More Details

□ Project uses the Artifactory Gradle Plugin

□ Capture and publish build info

□ Publish artifacts to Artifactory

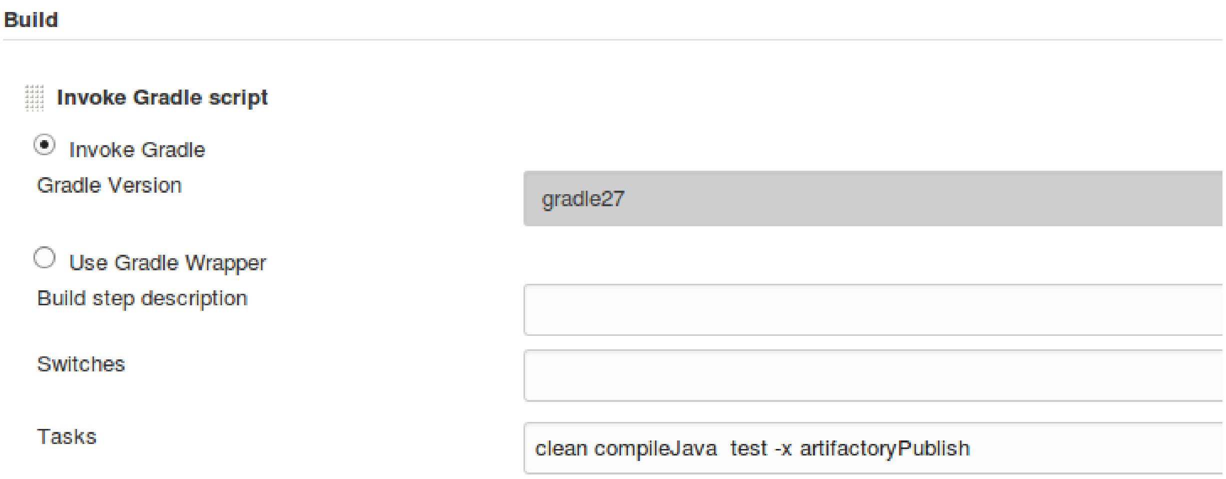
□ Enable isolated resolution for downstream builds (requires Artifactory Pro)

□ Enabled Release Management

8. Under the “Invoke Gradle build script” section in the “**Build**” section, enter the following for “**Tasks**”

clean compileJava test -x artifactoryPublish

(What we are doing here is telling it to clean any previous builds (clean), compile our Java code (compileJava), run our unit tests (test), but don’t run the default artifactory publish (-x artifactoryPublish).   We’re not trying to publish anything to Artifactory yet, just resolve dependencies from it.)



Build

Invoke Gradle script

● Invoke Gradle

Gradle Version: gradle27

○ Use Gradle Wrapper

Build step description:

Switches:

Tasks: clean compileJava test -x artifactoryPublish

10. Scroll down to “**Post-build Actions**”. The last thing to notice here is that we have an action setup after the build runs to “**Trigger parameterized build on other projects**”.

11. What this is doing is triggering another job to start after this one is done AND passing it one or more parameters. In this case, when we use the Persistent_Workspace=${WORKSPACE} string, we are telling it to set the variable Persistent_Workspace to the value of the current step’s workspace.

We’re going to pass this along to the next step so it can just reference the workspace directory with all the compiled code in it and not have to recompile it again in its own workspace.

12. Save your changes by clicking on the **Save** button at the bottom.

Lab 5. Setting up the Analysis job.

1. Note that we skipped setup on the **Integration Tests** job which was next in line. Since there isn't very much interesting here, this one is already done for you. However, you can open it up and look around if you want.

Click on the **ws-integration-tests** job name and then on the **Configure** button again on the left-hand side.

Under the "**Restrict where this project can be run**" setting, we have this set to "**pipeline_slave**" again to run on our new node.

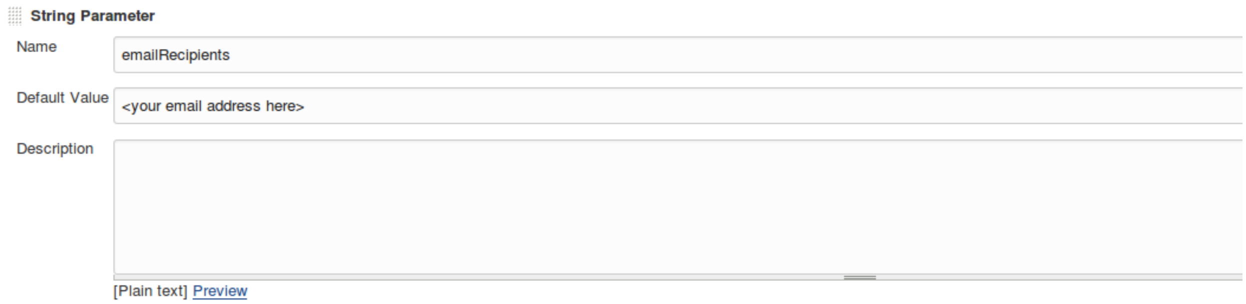
Under the "**Build**" section, we have an "**Execute shell**" command that fires up mysql and creates a test database for us to use in our integration tests by inputting an sql file with all the commands. (You can look at this file out in the diyuser home directory if you want.)

Then, still in the "**Build**" section, we invoke Gradle with our "**integrationTest**" task.

Finally, as a "**Post-build Action**", we are invoking the next job in line "**ws-analysis**" and passing our workspace location on to it as we did in the **ws-compile** job.

2. Now, let's work on the ws-analysis job. On the Jenkins dashboad (homepage), click on the **ws-analysis** job name and then on the **Configure** button again on the left-hand side.

3. The first part of this job is configuring parameters for the sonarqube metrics we'll be measuring and using. Find the one labeled "**emailRecipients**" and enter an email address where you can get mail in the "**Default Value**" field.



String Parameter

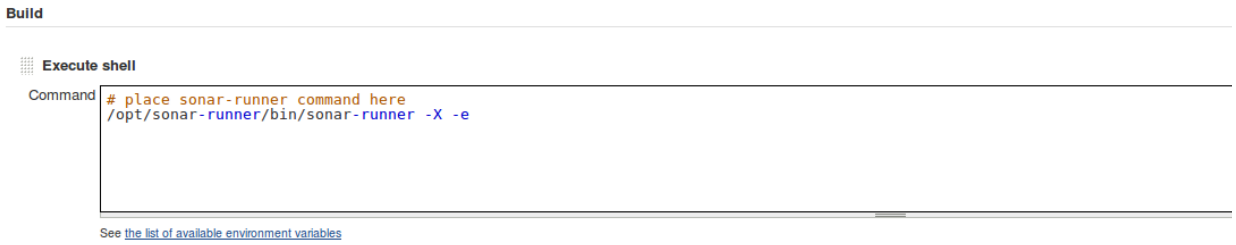
Name: emailRecipients

Default Value: <your email address here>

Description:

4. Find the "**Build**" section on the page and the (mostly) empty "**Execute shell**" box. This is where we'll enter the command to run the sonar-runner. Type in:

/opt/sonar-runner/bin/sonar-runner -X -e



Build

Execute shell

Command

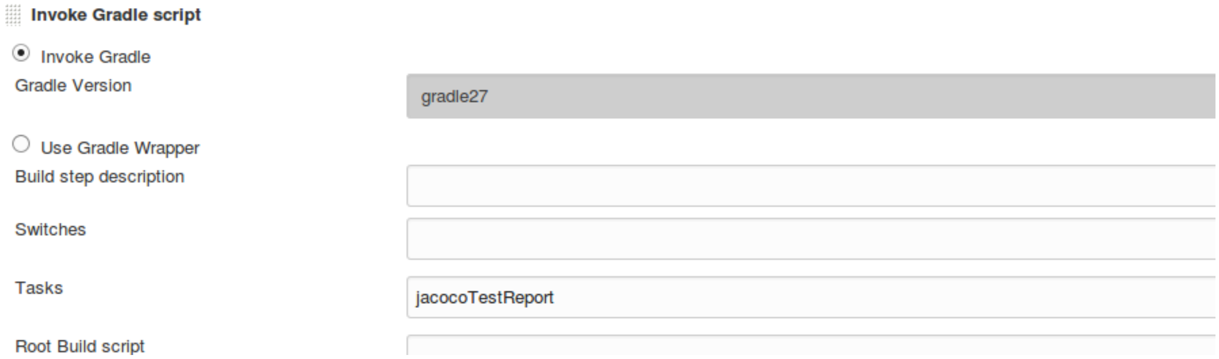
```
# place sonar-runner command here
/opt/sonar-runner/bin/sonar-runner -X -e
```

See the list of available environment variables

5. Next, in the step to “**Invoke Gradle script**”, we’ll turn on the jacoco coverage reports.

In the “Tasks” entry, type

jacocoTestReport



Invoke Gradle script

● Invoke Gradle

Gradle Version: gradle27

○ Use Gradle Wrapper

Build step description:

Switches:

Tasks: jacocoTestReport

Root Build script:

6. Now, we need to add in a post-build step to process the metrics and email the report. Scroll down to the bottom of the screen and find the button for “**Add post-build action**”.

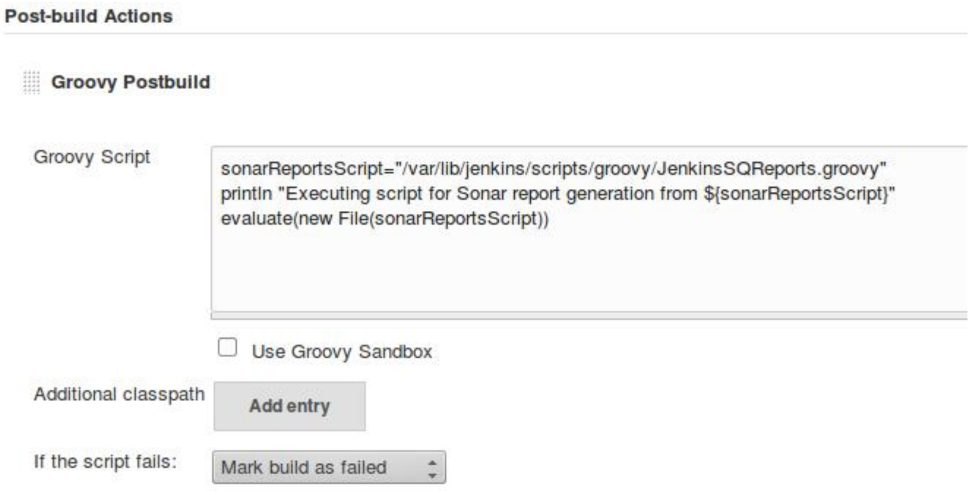
- Click on that button and select “**Groovy Postbuild**”.

- Scroll back up and just under the “**Postbuild Actions**” section, you’ll see a new entry for “**Groovy Postbuild**”.   In the “**Grooy Script**” textbox, you can just copy in the text from the file on your desktop named **ws-analysis-commands.txt**. (Recommended approach.)

OR, if you prefer to type them in, here they are.

```
sonarReportsScript="/var/lib/jenkins/scripts/groovy/JenkinsSQReports.groovy"

println "Executing script for Sonar report generation from ${sonarReportsScript}"

evaluate(new File(sonarReportsScript))
```

- Change the “**If script fails**” setting to “**Mark build as failed**”.



```
sonarReportsScript="/var/lib/jenkins/scripts/groovy/JenkinsSQReports.groovy"
println "Executing script for Sonar report generation from ${sonarReportsScript}"
evaluate(new File(sonarReportsScript))
```

☐ Use Groovy Sandbox

Additional classpath: Add entry

If the script fails: Mark build as failed

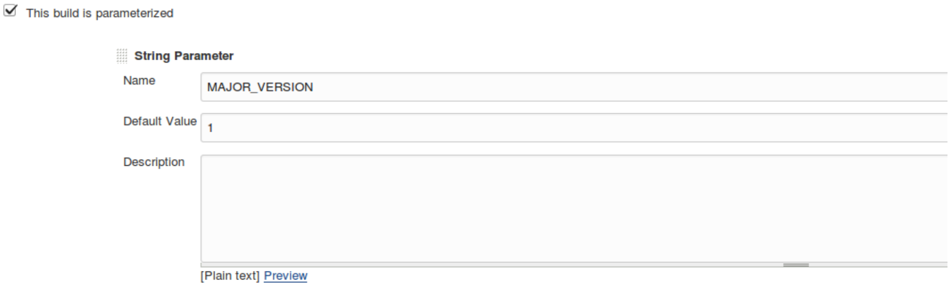
7. **Save** your changes.

Lab 6. Setting up the Assemble Job

In this job, we'll update the version number and attach a file in the war.

1. On the Jenkins dashboad (homepage), click on the **ws-assemble** job name and then on the **Configure** button again on the left-hand side.

2. We start out by defining values for the components of the artifact using parameters. If you want to change the version number for the war, you can change the values for MAJOR_VERSION, MINOR_VERSION, etc. here under the "**This build is parameterized**" section. To change these values, enter a different value in the "**Default Value**" section. Or you can leave the default values.



☑ This build is parameterized

String Parameter

Name: MAJOR_VERSION

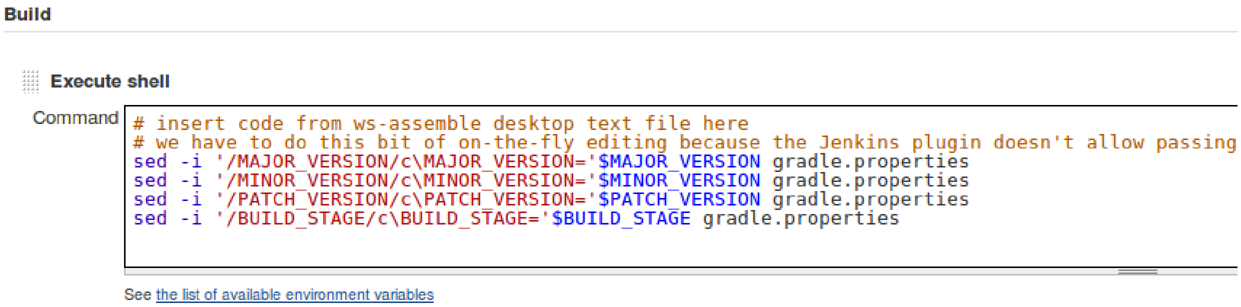
Default Value: 1

Description:

[Plain text] Preview

3. Now, we'll add the commands to put the version number in the gradle properties file. Under the "**Build**" section, in the "**Execute Shell"** section, enter the text from the **ws-assemble-commands.txt** file on the desktop. Or if you prefer to type them in, ,they are:

sed -i '/MAJOR_VERSION/c\MAJOR_VERSION='$MAJOR_VERSION gradle.properties
sed -i '/MINOR_VERSION/c\MINOR_VERSION='$MINOR_VERSION gradle.properties
sed -i '/PATCH_VERSION/c\PATCH_VERSION='$PATCH_VERSION gradle.properties
sed -i '/BUILD_STAGE/c\BUILD_STAGE='$BUILD_STAGE gradle.properties



Build

Execute shell

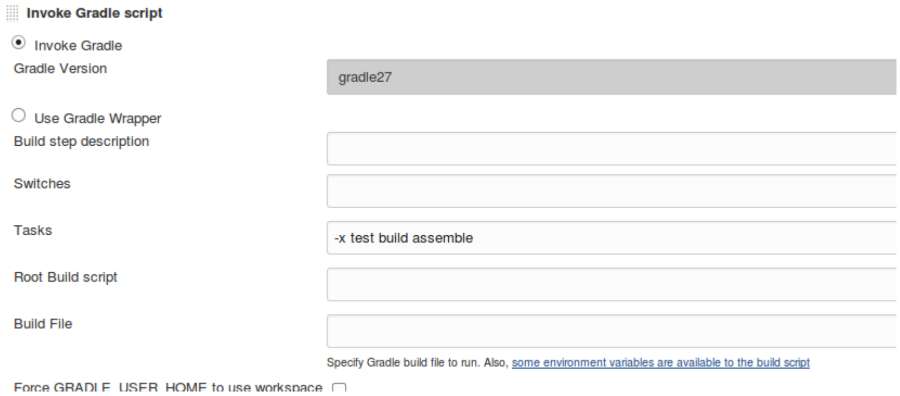
Command

```
# insert code from ws-assemble desktop text file here
# we have to do this bit of on-the-fly editing because the Jenkins plugin doesn't allow passing
sed -i '/MAJOR_VERSION/c\MAJOR_VERSION='$MAJOR_VERSION gradle.properties
sed -i '/MINOR_VERSION/c\MINOR_VERSION='$MINOR_VERSION gradle.properties
sed -i '/PATCH_VERSION/c\PATCH_VERSION='$PATCH_VERSION gradle.properties
sed -i '/BUILD_STAGE/c\BUILD_STAGE='$BUILD_STAGE gradle.properties
```

See the list of available environment variables

4. Underneath that, in the “**Invoke Gradle script**” section, add the following in the “**Tasks**” section.

-x test build assemble



Invoke Gradle script

● Invoke Gradle

Gradle Version: gradle27

○ Use Gradle Wrapper

Build step description:

Switches:

Tasks: -x test build assemble

Root Build script:

Build File:

Specify Gradle build file to run. Also, some environment variables are available to the build script

Force GRADLE_USER_HOME to use workspace □

5. **Save** your changes.

Lab 7. Setting up the Publish Artifact Job

The point of this job in our pipeline is to publish the artifacts we've built, tested, and assembled to an artifact repository for later processing. To do this, the main thing we need to do is enable Artifactory integration with publishing.

1. On the Jenkins dashboad (homepage), click on the **ws-publish-artifact** job name and then on the **Configure** button again on the left-hand side.

2. Find the "**Build Environment**" section. We're going to add the Gradle-Artifactory integration to be able to get artifacts from it.

Check the box for "**Gradle-Artifactory Integration**".

3. Under "**Deployment Details**", go ahead and set (if not already set) the "**Artifactory deployment server**" to our local one at

http://localhost:8082/artifactory

4. Since we will be resolving artifacts from our Artifactory system, we need to specify a repository to get the artifacts from. In the "**Publishing repository**" field, select the "**Different Value**" button at the end and then type in

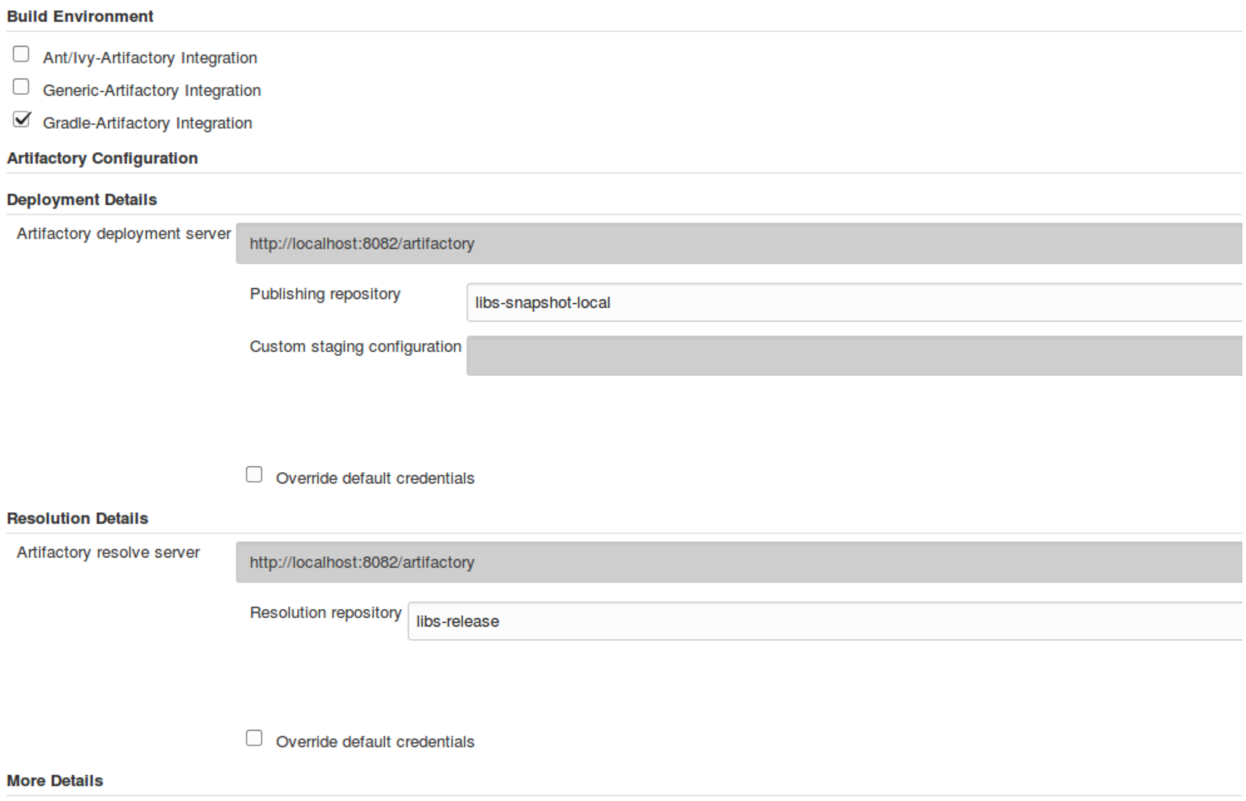
libs-snapshot-local

5. Under "**Resolution Details**", set the "**Artifactory resolve server**" to the same value as the deployment server (if not already set):

http://localhost:8082/artifactory

6. Since we will be publishing artifacts from our Artifactory system, we need to specify a repository to put the artifacts into. In the "Publishing repository" field, select the "Different Value" button at the end and then type in

libs-release



Build Environment

☐ Ant/Ivy-Artifactory Integration

☐ Generic-Artifactory Integration

☑ Gradle-Artifactory Integration

Artifactory Configuration

Deployment Details

Artifactory deployment server: http://localhost:8082/artifactory

Publishing repository: libs-snapshot-local

Custom staging configuration:

☐ Override default credentials

Resolution Details

Artifactory resolve server: http://localhost:8082/artifactory

Resolution repository: libs-release

☐ Override default credentials

More Details

7. Under the “**More Details**” section, make sure the following options are checked.

   Capture and publish build info

   Include environment variables

   Allow promotion of unstaged builds

   Publish artifacts to Artifactory

      Publish Maven descriptors

   Use maven-compatible patterns

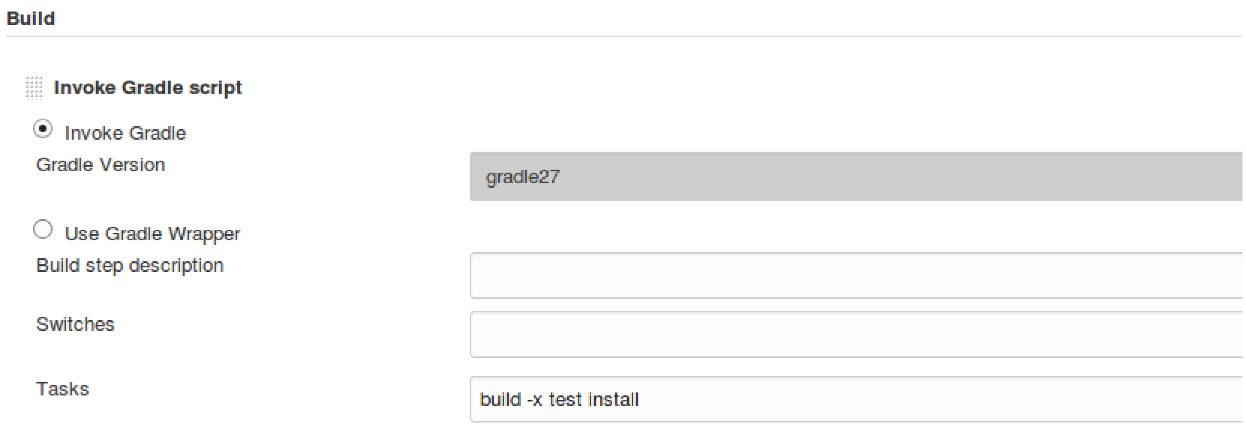
In the “**Exclude patterns**” field, type

   ***.jar**

8. In the “**Build**” section, for the step to “**Invoke Gradle script**”, add the following in the **Tasks** field:

   build -x test install

(Note that we have the install target to cause the Maven integration in Gradle to produce the pom.)



Build

Invoke Gradle script

● Invoke Gradle

Gradle Version: gradle27

○ Use Gradle Wrapper

Build step description:

Switches:

Tasks: build -x test install

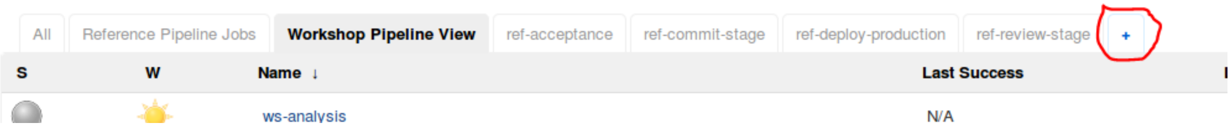
9. **Save** your changes.

Lab 8. Setting up the Commit Stage of the Pipeline

In this lab, we'll set up the pipeline view for the set of jobs we just finished configuring.

1. Change back to the jenkins dashboard if not already there (http://localhost:8080)

2. Above the list of jobs, on the line with the various tabs, click the "+" sign at the end.



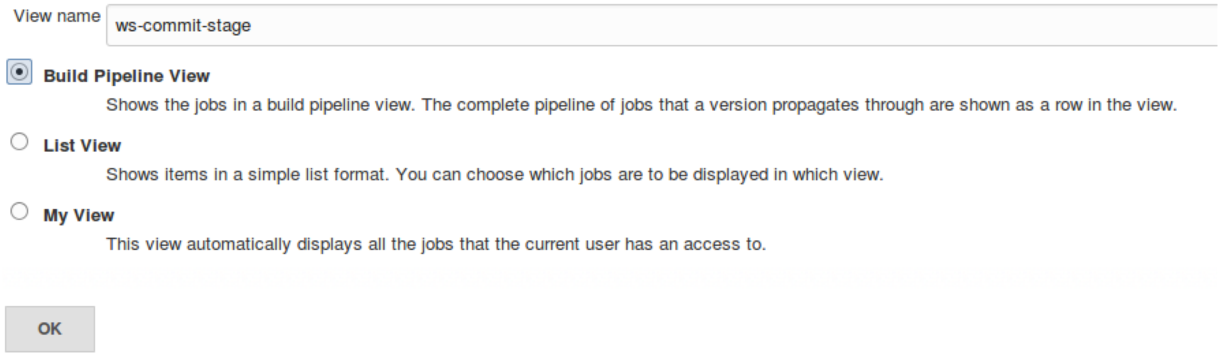
S	W	Name ↓	Last Success
●	☀	ws-analysis	N/A

3. You'll now be on the configuration page for setting up the view. For the "**View Name**", you can type in

(3) ws-commit-stage

(Note: The "(3)" here is optional but if present will sort the tab into the desired place in the list of tabs.)

4. Select the button for **Build Pipeline View** and select **OK**.



View name: ws-commit-stage

● **Build Pipeline View**
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

○ **List View**
Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

○ **My View**
This view automatically displays all the jobs that the current user has an access to.

5. On the next page, add a description in the "**Description**" field if you want.

6. Find the "**Build Pipeline View Title**" field and enter something like:

Workshop Pipeline - Commit Stage

7. For “**Layout**”, select “**Based on upstream/downstream relationships**”.

8. Set the “**Select Initial Job**” to be

    ws-compile

9. The suggested value for “**No. of Displayed Builds**” is **1**.

10. Select “**Yes**” for the first two radio buttons: “**Restrict triggers to most recent successful builds**” and “**Always allow manual trigger on pipeline steps**”

11. Save the changes.

12. You can now see your view by clicking on the tab at the top of the list of jobs.

13. If a job has never been run, you will see a message like:

“This view has no jobs associated with it. You can either add some existing jobs to this view or create a new job in this view.”

This is ok and will be resolved once the job is run.

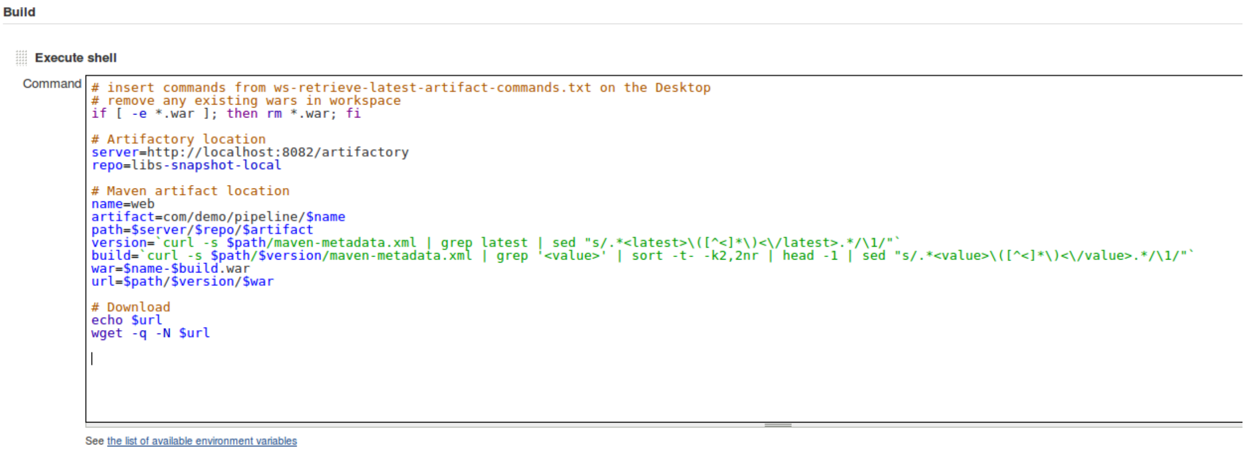
Lab 10. Creating the job to retrieve the latest artifact

The purpose of this job is to retrieve the latest artifact from Artifactory.   There are easier ways to do this via Artifactory Pro if you buy it.   Since we are going with the free version, we will use some scripting to examine the POM files and extract the latest.

1. On the Jenkins dashboad (homepage), click on the **ws-retrieve-latest-artifact** job name and then on the **Configure** button again on the left-hand side.

2. Scroll down to the "**Build**" section.

3. Find the "**Execute shell**" command field (has this line in it - # insert commands from **ws-retrieve-latest-artifact-commands.txt** on the Desktop). Open up that file on the Desktop of your machine and copy and paste the commands into it. (It is not recommended that you type this one in since there is specific formatting involved.



Build

Execute shell

Command

```
# insert commands from ws-retrieve-latest-artifact-commands.txt on the Desktop
# remove any existing wars in workspace
if [ -e *.war ]; then rm *.war; fi

# Artifactory location
server=http://localhost:8082/artifactory
repo=libs-snapshot-local

# Maven artifact location
name=web
artifact=com/demo/pipeline/$name
path=$server/$repo/$artifact
version=`curl -s $path/maven-metadata.xml | grep latest | sed "s/.*<latest>\([^<]*\)<\/latest>.*/\1/"`
build=`curl -s $path/$version/maven-metadata.xml | grep '<value>' | sort -t- -k2,2nr | head -1 | sed "s/.*<value>\([^<]*\)<\/value>.*/\1/"`
war=$name-$build.war
url=$path/$version/$war

# Download
echo $url
wget -q -N $url
```

See the list of available environment variables

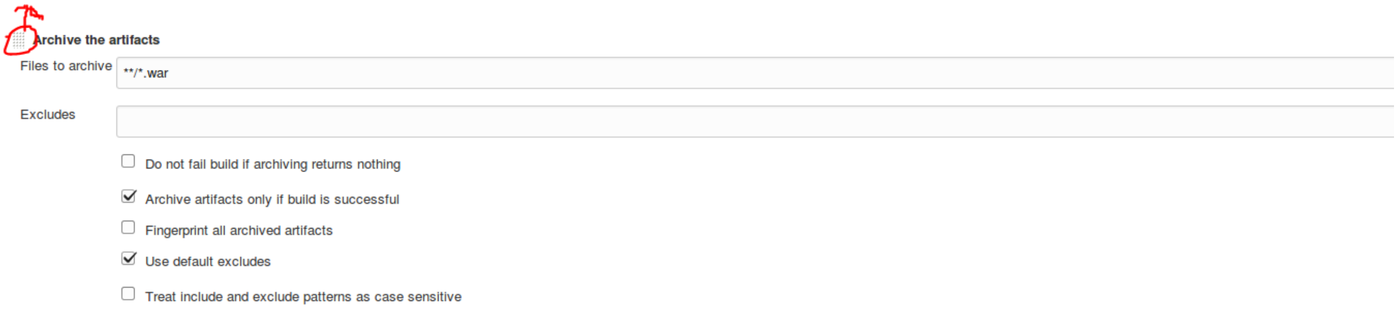
4. Add a **Post-build Action** to archive the artifacts.

Click on the "**Add post-build action**" button (near the bottom) and select "**Archive the artifacts**". A new step will appear in the **Post-build Actions** section. This should be the first step under the "**Post-build Actions**" section (before the "**Trigger parameterized builds on other projects**" step). If it is not, then drag the step by the handle up before the "Trigger…" step. The handle is shown in red in the figure bel

5. Fill in the "**Files to archive**" field with

****/*.war**

The boxes for “**Archive artifacts only if build is successful**” and “**Use default excludes**” should be checked.



Archive the artifacts

Files to archive: **/*.war

Excludes:

☐ Do not fail build if archiving returns nothing

☑ Archive artifacts only if build is successful

☐ Fingerprint all archived artifacts

☑ Use default excludes

☐ Treat include and exclude patterns as case sensitive

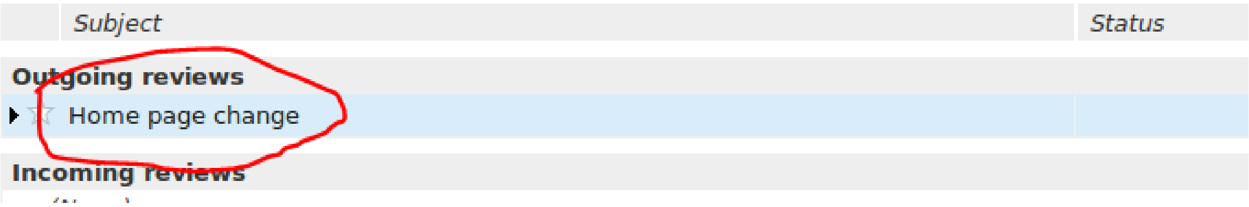
6. After dragging the command up to the correct position, **save** your changes.

7. You should now be able to run the **ws-retrieve-latest-artifact job** (click on the **Build Now** link).

Lab 9. Running a change through the Commit Stage

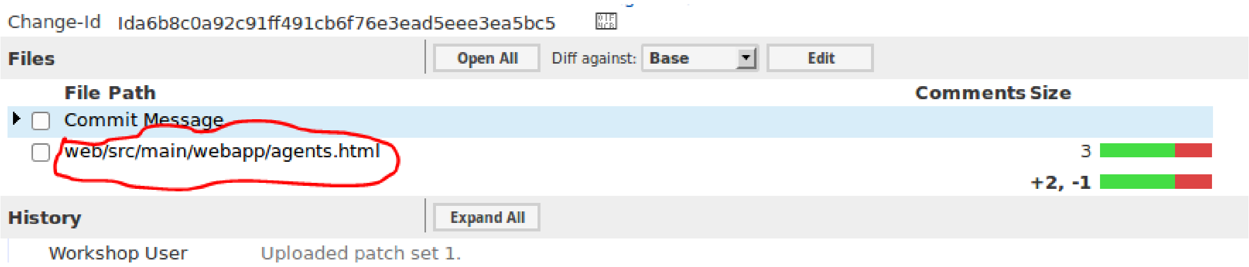
In this lab, we'll review and submit our change in Gerrit and see it move through the Commit Stage of the pipeline.

1. Change back to the Gerrit instance at **http://localhost:8081**.
2. You should be running as "**Workshop User**" - that should be shown in the upper right corner.
3. Click on the single outgoing review you have. (Click on the Subject.)



	Subject	Status
Outgoing reviews		
▸ ☆	Home page change	
Incoming reviews		

4. You'll now be on the page for the change. For this case, we'll be the author of the change AND the reviewer (this is not typical). Down around the middle of the page, click on the name of the file (agents.html). This will bring up a screen to let you see the differences being put forward.



Change-Id Ida6b8c0a92c91ff491cb6f76e3ead5eee3ea5bc5

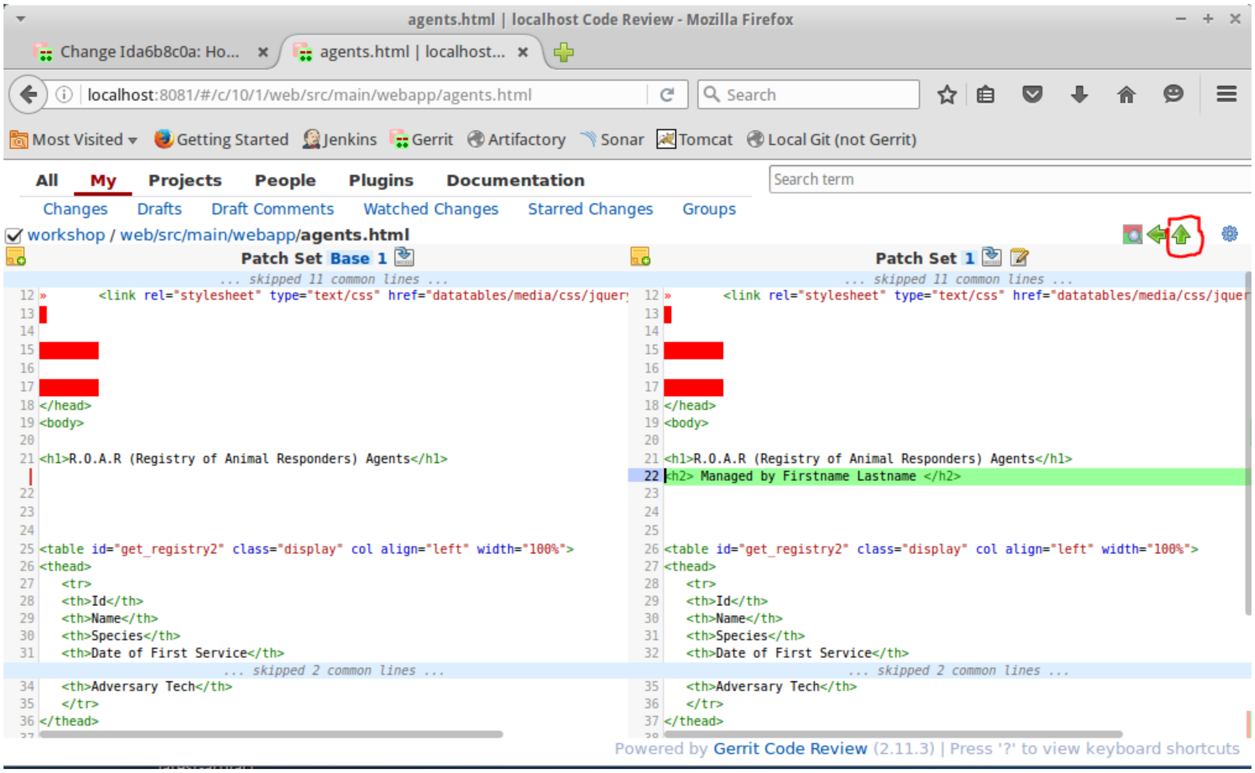
Files Open All Diff against: Base Edit

	File Path	Comments	Size
▸ □	Commit Message		
□	web/src/main/webapp/agents.html		3 🟩🟥
			+2, -1 🟩🟥

History Expand All

Workshop User Uploaded patch set 1.

5. You'll see a screen that will look similar to the one below. For the sake of time, we won't do any commenting or further review here. After you look at the differences, click on the green "up arrow" in the upper right corner to get back to the previous page.



workshop / web/src/main/webapp/agents.html

Patch Set Base 1

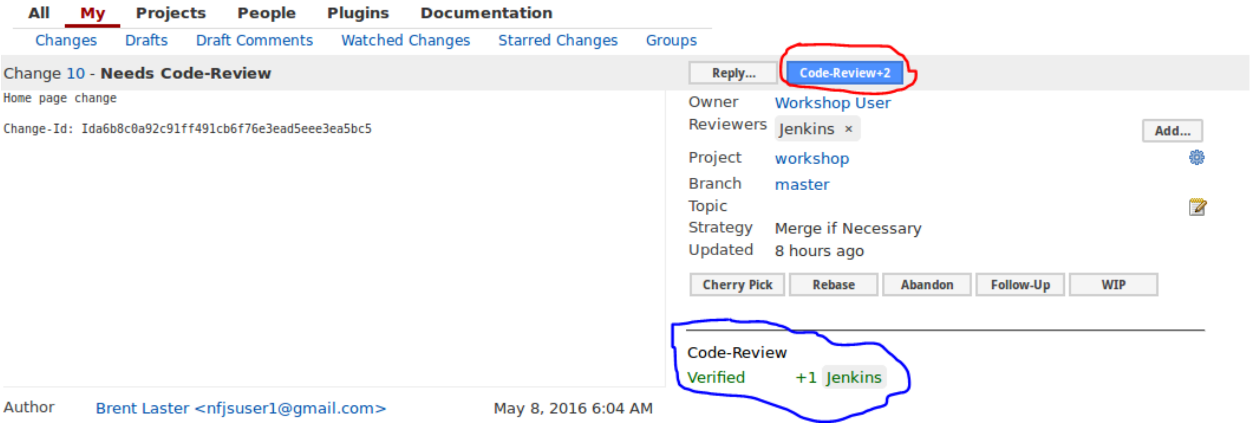
```
... skipped 11 common lines ...
»    <link rel="stylesheet" type="text/css" href="datatables/media/css/jquer


</head>
<body>

<h1>R.O.A.R (Registry of Animal Responders) Agents</h1>



<table id="get_registry2" class="display" col align="left" width="100%">
<thead>
   <tr>
   <th>Id</th>
   <th>Name</th>
   <th>Species</th>
   <th>Date of First Service</th>
... skipped 2 common lines ...
   <th>Adversary Tech</th>
   </tr>
</thead>
```

Patch Set 1

```
... skipped 11 common lines ...
»    <link rel="stylesheet" type="text/css" href="datatables/media/css/jquer


</head>
<body>

<h1>R.O.A.R (Registry of Animal Responders) Agents</h1>
<h2> Managed by Firstname Lastname </h2>



<table id="get_registry2" class="display" col align="left" width="100%">
<thead>
   <tr>
   <th>Id</th>
   <th>Name</th>
   <th>Species</th>
   <th>Date of First Service</th>
... skipped 2 common lines ...
   <th>Adversary Tech</th>
   </tr>
</thead>
```

6. Back on the change screen, on the right center section, find the "**Verified +1 Jenkins**" line. This means that Jenkins did a test build (our **ws-verify** project) and it was successful (+1). We now need to complete the other category of check "**Code-Review**".

7. Gerrit provides a shortcut to approve a change - the blue button at the top. Click on the "**Code Review+2**" button.



Change 10 - **Needs Code-Review**

```
Home page change

Change-Id: Ida6b8c0a92c91ff491cb6f76e3ead5eee3ea5bc5
```

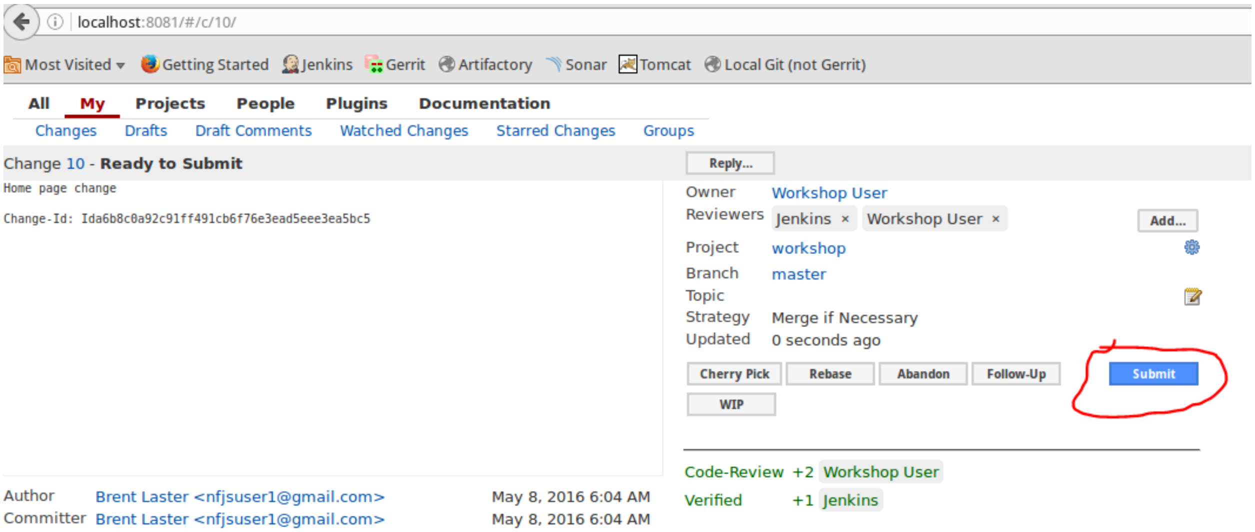
Owner	Workshop User
Reviewers	Jenkins ×
Project	workshop
Branch	master
Topic	
Strategy	Merge if Necessary
Updated	8 hours ago

Code-Review
Verified +1 Jenkins

Author Brent Laster <nfjsuser1@gmail.com> May 8, 2016 6:04 AM

8. Now that we've "reviewed" the code change, and approved it (via pushing the button), you'll see that the "**Code-Review**" row now has a "**+2 Workshop User**" next to it. Since both **Code-Review** and **Verified** are green and have positive values, we are ready to tell Gerrit to try and merge it in to the repository.

To do this, press the **Submit** button.



Change 10 - **Ready to Submit**

```
Home page change

Change-Id: Ida6b8c0a92c91ff491cb6f76e3ead5eee3ea5bc5
```

Owner Workshop User
Reviewers Jenkins × Workshop User ×
Project workshop
Branch master
Topic
Strategy Merge if Necessary
Updated 0 seconds ago

Code-Review +2 Workshop User
Verified +1 Jenkins

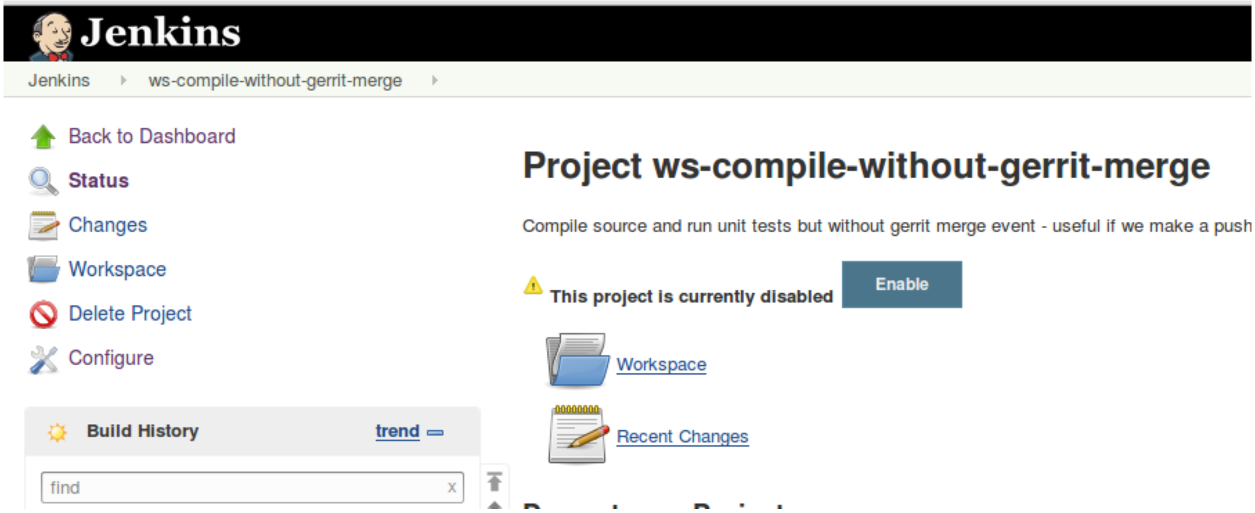
Author Brent Laster <nfjsuser1@gmail.com> May 8, 2016 6:04 AM
Committer Brent Laster <nfjsuser1@gmail.com> May 8, 2016 6:04 AM

Alternate approach for the ws-verify/ws-compile steps.

If, for some reason, your submit and merge for the **ws-compile** step does not succeed, you can use an alternate job that does not go through **Gerrit**. The **ws-compile-without-gerrit-merge** step is already setup to use. You would just push directly from the working directory and integrate it with the **ws-commit-stage** as outlined below.

1. On the Jenkins dashboard, click on the **ws-compile-without-gerrit-merge** job name. This job is currently Disabled since it is setup to poll for SCM changes. To use it, we will need to enable it.

Click on the **Enable** button to do this.



Project ws-compile-without-gerrit-merge

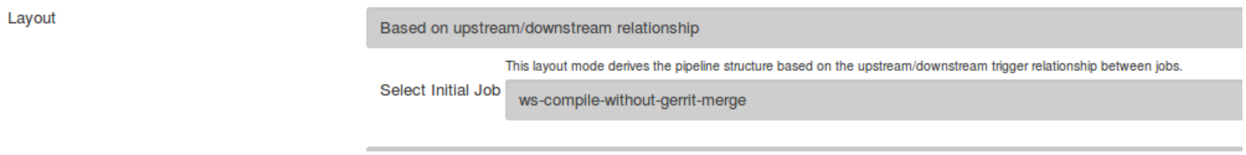
Compile source and run unit tests but without gerrit merge event - useful if we make a push

This project is currently disabled

2. Now we need to modify our pipeline view to start with this job. Switch back to the dashboard. On the top, in the list of views, click on the "**ws-commit-stage**" tab at the top.

3. Then, click on the "**Configure**" icon.

4. In the Layout section, find the "Select Initial Job" field and change it to use "ws-compile-without-gerrit-merge".

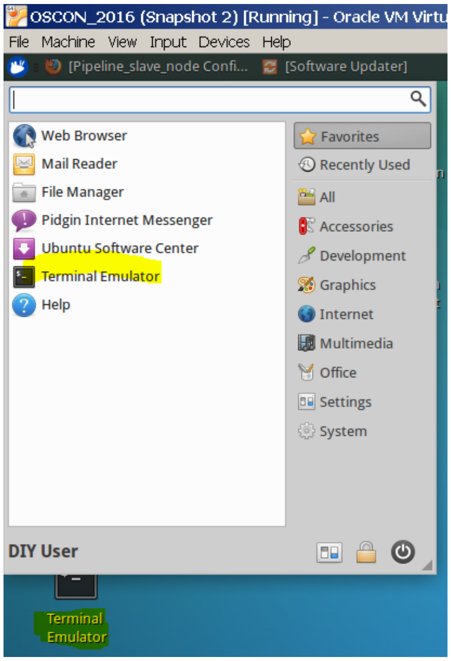


Layout: Based on upstream/downstream relationship

This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs.

Select Initial Job: ws-compile-without-gerrit-merge

5. If you already have a terminal session open, switch to it. If not, open up a terminal session by either clicking on the “Terminal Emulator” shortcut on the desktop or by selecting the “Terminal Emulator” selection from the system menu (drop down from the mouse in the upper-lefthand side).



6. In the terminal session, if not already there, cd to the **workshop** directory.

```
cd workshop
```

7. Under the “workshop” directory, in the “web/src/main/webapp” subdirectory, edit the “agents.html” file.

(You can use mousepad or gedit editors.)

```
gedit web/src/main/webapp/agents.html
```

Make some change to one of the header lines. Either

```
<h1>R.O.A.R (Registry of Animal Responders) Agents</h1>
```

or

```
<h2> Managed by ( your name here ) </h2>
```

8. Save your file (File->Save) and quit/exit the editor.

9. From the workshop directory, do a local build to make sure things compile.

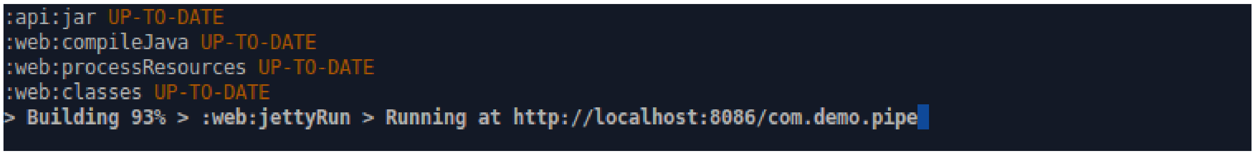
```
gradle build
```

This should finish with a “BUILD SUCCESSFUL” message.

10. Run the local instance to make sure the changes look as expected. Do this by running the jettyRun task (note case) and then opening a browser to **http://localhost:8086/com.demo.pipeline**.

```
gradle jettyRun
```

(Note: This will get to some percentage and then start the system running. Don’t kill this - it is working.)



```
:api:jar UP-TO-DATE
:web:compileJava UP-TO-DATE
:web:processResources UP-TO-DATE
:web:classes UP-TO-DATE
> Building 93% > :web:jettyRun > Running at http://localhost:8086/com.demo.pipe
```

<open in browser tab> **http://localhost:8086/com.demo.pipeline**

NOTE: note that the URL is pipeline on the end not just pipe as shown.

11. Close the browser tab (not the entire session) and kill the running job in the terminal **(Ctrl-C) .**

12. Now add and commit the change. This assumes again that you are still in the workshop directory. (Ignore any messages about line ending changes.)

```
git add web/src/main/webapp/agents.html
git commit -m “home page change”
```

13. Now push the change to Gerrit. This will use the standard Gerrit syntax - not the syntax for Gerrit.

```
git push origin master
```