

## OSCON 2017

### Creating a Deployment Pipeline with Jenkins 2

#### Bonus Labs (Optional)

Version 1.1 – 4/19/17

Brent Laster

These are optional bonus labs for Creating a Deployment Pipeline with Jenkins 2. These labs are intended to be done only after all of the class labs have been completed so that the full pipeline will be in place.

#### Bonus Lab 1 – Creating and Using a Jenkinsfile

**Purpose:** In this optional lab, we'll see how to convert our pipeline into a Jenkinsfile.

1. Now that we have our pipeline working, let's convert it from having to run in the Jenkins application to an external **Jenkinsfile** that can be stored with our source code.

First, **select** and **copy** all the code for your pipeline from the script window to the clipboard.

Now, switch to a terminal session. Change to the ~/workshop directory.

```
cd ~/workshop
```

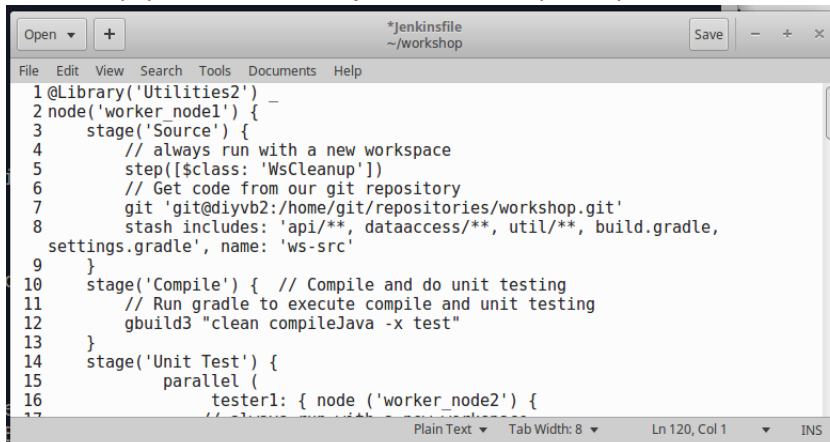
Create a new branch to hold the Jenkinsfile we are going to create.

```
git checkout -b jfile
```

This will create the branch and switch to it with one command. Open up an editor to create the Jenkinsfile.

```
gedit Jenkinsfile
```

In the empty editor window, **paste** the code you copied from Jenkins.

A screenshot of a text editor window titled '\*Jenkinsfile' with the path '~/.workshop'. The editor has a menu bar with 'File', 'Edit', 'View', 'Search', 'Tools', 'Documents', and 'Help'. The script content is as follows:

```
1@Library('Utilities2')
2node('worker_node1') {
3    stage('Source') {
4        // always run with a new workspace
5        step([$class: 'WsCleanup'])
6        // Get code from our git repository
7        git 'git@diyvb2:/home/git/repositories/workshop.git'
8        stash includes: 'api/**, dataaccess/**, util/**, build.gradle,
settings.gradle', name: 'ws-src'
9    }
10    stage('Compile') { // Compile and do unit testing
11        // Run gradle to execute compile and unit testing
12        gbuild3 "clean compileJava -x test"
13    }
14    stage('Unit Test') {
15        parallel (
16            tester1: { node ('worker_node2') {
17                // Run gradle to execute compile and unit testing
```

2. We need to make a couple of changes to our script to work outside of Jenkins.

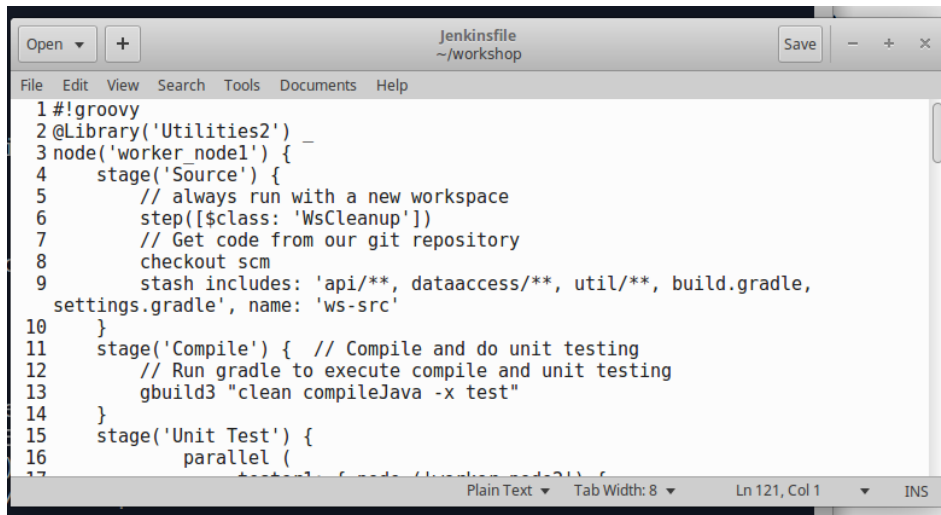
Add a line at the top of the script to signify this is a groovy file.

**#!/groovy**

Next, because we will be storing this file with the project code, we can use a shortcut for the source control command. Change the **git** line in the **Source** stage to be just:

**checkout scm**

Afterwards, your editor should look like this (note changed lines 1 and 8).



```
1#!/groovy
2@Library('Utilities2') _
3node('worker_node1') {
4    stage('Source') {
5        // always run with a new workspace
6        step([class: 'WsCleanup'])
7        // Get code from our git repository
8        checkout scm
9        stash includes: 'api/**, dataaccess/**, util/**, build.gradle,
10        settings.gradle', name: 'ws-src'
11    }
12    stage('Compile') { // Compile and do unit testing
13        // Run gradle to execute compile and unit testing
14        gbuild3 "clean compileJava -x test"
15    }
16    stage('Unit Test') {
17        parallel (
```

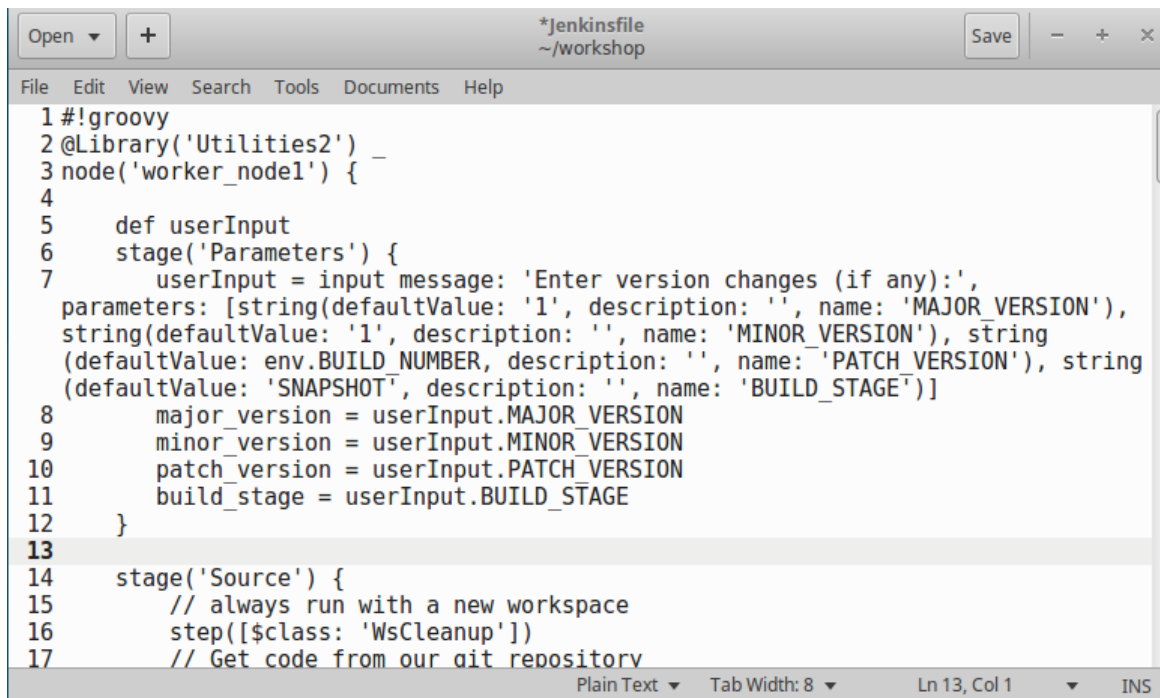
3. We will also need to add in code to input the parameters since we will be running this outside of Jenkins.

This involves three parts:

- a. Define a global variable that can store the input values and be accessed across stages
- b. Define and execute the **input** step (we'll encapsulate this in a new stage)
- c. Update our **Assemble** stage to use the values from the **input** step.

For simplicity, the code you need to add is in the file named **Parameters** on the desktop. Open that file and **copy** the code from it into the clipboard.

**Paste** the code in your **Jenkinsfile** after the **node** definition (and before the **Source** stage). After this, your Jenkinsfile should look like this at the top (spacing and lines 5-12 added).

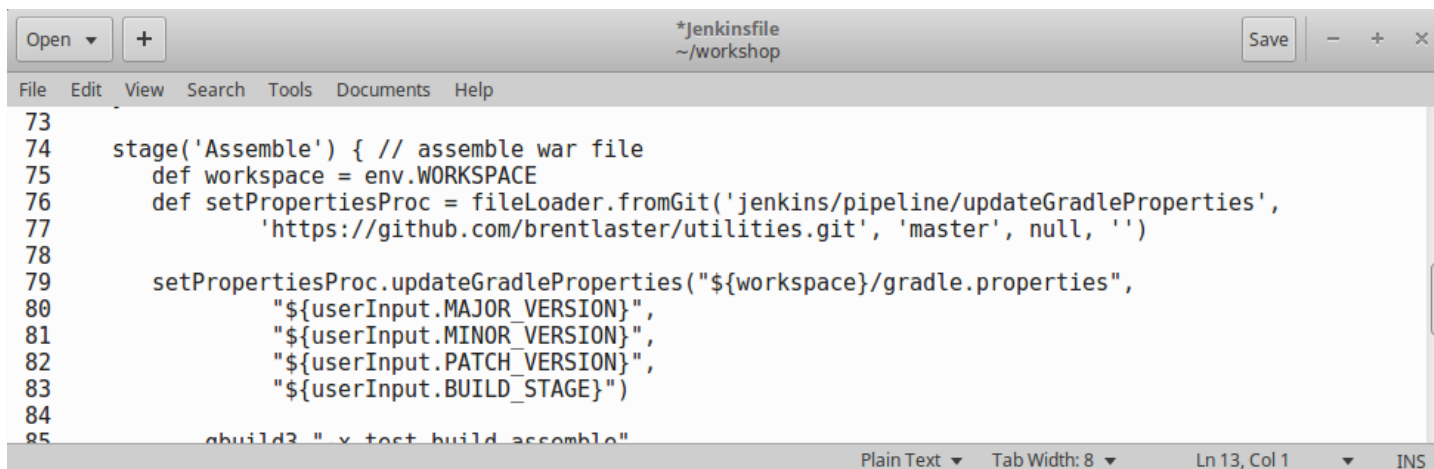


```
1#!/groovy
2@Library('Utilities2') _
3node('worker_node1') {
4
5    def userInput
6    stage('Parameters') {
7        userInput = input message: 'Enter version changes (if any):',
8        parameters: [string(defaultValue: '1', description: '', name: 'MAJOR_VERSION'),
9        string(defaultValue: '1', description: '', name: 'MINOR_VERSION'), string
10        (defaultValue: env.BUILD_NUMBER, description: '', name: 'PATCH_VERSION'), string
11        (defaultValue: 'SNAPSHOT', description: '', name: 'BUILD_STAGE')]
12        major_version = userInput.MAJOR_VERSION
13        minor_version = userInput.MINOR_VERSION
14        patch_version = userInput.PATCH_VERSION
15        build_stage = userInput.BUILD_STAGE
16    }
17    stage('Source') {
18        // always run with a new workspace
19        step([$class: 'WsCleanup'])
20        // Get code from our git repository
```

- Now we need to update the **Assemble** stage. Scroll down to the **Assemble** stage and, in the call to **updateGradleProperties**, update each of the parameters to have **userInput**. in front of them instead of **params**.

For example, **\${params.MAJOR\_VERSION}** should be changed to **\${userInput.MAJOR\_VERSION}**

Once you are done, that section should look like the figure below.



```
73
74 stage('Assemble') { // assemble war file
75     def workspace = env.WORKSPACE
76     def setPropertiesProc = fileLoader.fromGit('jenkins/pipeline/updateGradleProperties',
77     'https://github.com/brentlaster/utilities.git', 'master', null, '')
78
79     setPropertiesProc.updateGradleProperties("${workspace}/gradle.properties",
80     "${userInput.MAJOR_VERSION}",
81     "${userInput.MINOR_VERSION}",
82     "${userInput.PATCH_VERSION}",
83     "${userInput.BUILD_STAGE}")
84
85     sh "mvn test -Dtest=build-assembly"
```

- Save** your changes to the **Jenkinsfile**. Now, we will need to update it in Git so we can point Jenkins to the SCM project with it in there. Execute the following commands to do this (still in the same workshop directory).

**git add Jenkinsfile**

© 2017 Brent Laster

```
git commit -m "created Jenkinsfile"
git push origin jfile:jfile
```

- Now, we'll see how Jenkins can automatically create a project for us when we have a **Jenkinsfile** with the source code.

Go back to the Jenkins dashboard and click on **New Item**.

Give it a name of **mbdemo** and select **Multibranch Pipeline** for the type. Click the **Ok** button.

Scroll down to the **Branch Sources** area and click the **Add Source** button. Select **Git** for the type.

In the **Project Repository** field, enter the path to our project on the Git remote on this system:

**/home/git/repositories/workshop**

In the Credentials drop-down, select **jenkins2 (Jenkins SSH)**

In the **Scan Multibranch Pipeline Triggers**, check (select) the box for **Periodically if not otherwise run**. You can leave the default interval as-is.

**Save** your changes.

Jenkins will now go off and scan the branches in the git project for any that have a Jenkinsfile. When it finds one (such as our **jfile** branch), it will automatically create a new job to build the code in that branch using the Jenkinsfile and will try to run it.

**Build Queue**

No builds in the queue.

**Build Executor Status**

**master**

1 Idle

2 Idle

**worker\_node1**

1 Idle

2 part of mbdemo » jfile #1

**worker\_node2**

1 Idle

2 part of mbdemo » jfile #1

3 part of mbdemo » jfile #1

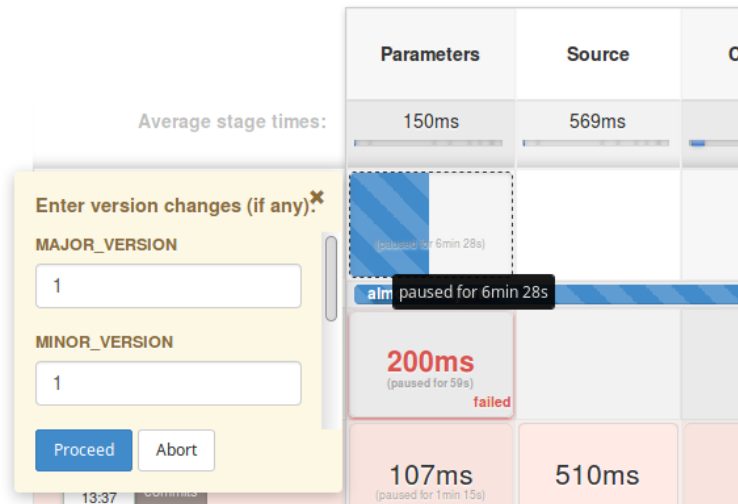
```
> git --version # timeout=10
using GIT_SSH to set credentials Jenkins2 SSH
> git fetch --tags --progress origin +refs/heads/*:refs/remotes/origin/* --prune
Getting remote branches...
Seen branch in repository origin/jfile
Seen branch in repository origin/master
Seen 2 remote branches
Checking branch jfile
    'Jenkinsfile' found
Met criteria
Scheduled build for branch: jfile
Checking branch master
    'Jenkinsfile' not found
Does not meet criteria
Done.
[Thu Apr 13 01:45:03 EDT 2017] Finished branch indexing. Indexing took 0.77 sec
Finished: SUCCESS
```

7. Go to the **jfile** stage view by clicking the **Up** link in the upper left corner (or hitting the back button) then selecting the **mbdemo** project and then selecting the **jfile** project (the project is to build that branch and so it has the name of the branch.)

Jenkins will be waiting for you to update the input parameters (if you want) or just tell it to **Proceed**.

You won't see the prompt for input by default. But there are two ways to see it and interact with it.

- a. In the **Stage View**, you can hover over the **Parameters** cell, and the prompt will pop up to the side. (Note: You may need to **Disable Auto Refresh** to have the pop-up stay up long enough to interact with it.)



- b. Or, in the **Console Log**, you will see a link for the input.

```
> git checkout -f fef356a42c8a98212c88c92d3d3ca3c0bdb6d3ab
> git rev-list fef356a42c8a98212c88c92d3d3ca3c0bdb6d3ab # timeout=10
[Pipeline] node
Running on worker_node1 in /home/jenkins2/worker_node1/workspace/mbdemo_jfile-V
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Parameters)
[Pipeline] input
Input requested
🌀
```

Clicking on that link will bring up a more familiar prompt.

Jenkins > mbdemo > jfile > #15 > Paused for Input

[Back to Project](#)  
[Status](#)  
[Changes](#)  
[Console Output](#)  
[Edit Build Information](#)  
[Git Build Data](#)  
[No Tags](#)  
[Paused for Input](#)  
[Thread Dump](#)

### Enter version changes (if any):

MAJOR_VERSION	<input type="text" value="1"/>
MINOR_VERSION	<input type="text" value="1"/>
PATCH_VERSION	<input type="text" value="15"/>
BUILD_STAGE	<input type="text" value="SNAPSHOT"/>

After you **respond** to the prompt and tell it to **Proceed**, your pipeline should finish building completely. At this point, you can explore the running app in the Docker image (info in the console log), look at the artifacts in Artifactory, the analysis in Sonar, etc.

enlarge

	Parameters	Source	Compile	Unit Test	Integration Test	Analysis	Assemble	Publish Artifacts	Retrieve Latest Artifact	Deploy To Docker
Average stage times:	152ms	551ms	10s	54s	13s	16s	8s	10s	2s	9s
#15 Apr 13 14:01 3 commits	93ms <small>(paused for 7 min 43s)</small>	463ms	6s	56s	13s	19s <small>(paused for 2s)</small>	9s	14s	3s	10s

## Bonus Lab 2 - Updating source and running it through the pipeline.

**Purpose:** In this optional lab, we run through an example of making a simple source update and watching Jenkins build it for you.

1. 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).

2. In the terminal session, if not already there, cd to the **workshop** directory AND switch to the **jfile** branch.

```
cd workshop
```

```
git checkout jfile
```

3. 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
```

Add a new line after

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

The new line can have the form

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

4. Save your file and quit/exit the editor.
5. From the workshop directory, do a local build to make sure things compile.

## gradle build

This should finish with a “BUILD SUCCESSFUL” message.

6. 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.pipeline
```

<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.

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

9. Now push the change to Git.

**git push origin jfile**

10. After a minute or two, Jenkins should detect the change and start building the **jfile** project in **mbdemo**. Here again, you'll need to respond to the **input prompt** to tell it to **Proceed**.

Once the build is complete, you can explore the running app in the Docker image (info in the console log), look at the artifacts in Artifactory, the analysis in Sonar, etc.

=====

**END OF BONUS LABS**

=====