TEKsystems
Our people make IT possible.

TEKsystems Education Services

presents

Continuous Integration with Jenkins
Lab Guide

20750 Civic Center Drive
Suite 400, Oakland Commons II
Southfield, MI 48076
800.294.9360

IN1502-LG / 5-2-16

Our people make IT possible.

Contents

1: Install Jenkins 5

2: Create a Jenkins Job 11

3. Schedule a Jenkins Job 17

4. Secure Jenkins 23

5. Complex Jenkins Projects 25

6. Using Plugins 29

7. Distributed Builds in Jenkins 31

8. Run NodeJS Plugin in Jenkins 35

9. Create a Jenkins Plugin 39

10. Build a Jenkins Pipeline 43

1: Install Jenkins

Overview

In this lab, you will install Jenkins and perform some initial setup to enable a job to run. The purpose of the lab is to demonstrate the broad range of options available when building and testing an application in Jenkins.

Jenkins, like other continuous integration servers, supports many types of build and test processes and needs access to many different technology pieces to run.

You will perform these general tasks:

- Install Jenkins and test the installation
- Customize the Jenkins environment
- Configure Jenkins to run in Tomcat and test the installation

1. If you performed the setup for this course, you will have downloaded the **jenkins.war**. If not, it can be downloaded from: http://mirrors.jenkins-ci.org/war-stable/1.651.1/jenkins.war. Once the war file is available to run, Jenkins can be installed. Because it is an executable war, place the war in the folder where you want to install Jenkins.
   - Windows Location: **`C:\PF\jenkins`**
   - Mac Location: **`/Users/yourUserName/jenkins`**

2. Open a command window or terminal. Change to the folder where you just copied the war. Set the JENKINS_HOME environment variable to the folder.
   - For Windows: **`set JENKINS_HOME=C:\PF\jenkins`**
   - For Mac: **`export JENKINS_HOME=/Users/yourUserName/jenkins`**

3. The first time Jenkins is run, it will create all the necessary content in the JENKINS_HOME folder. To perform the initial installation and configuration, run the command below:

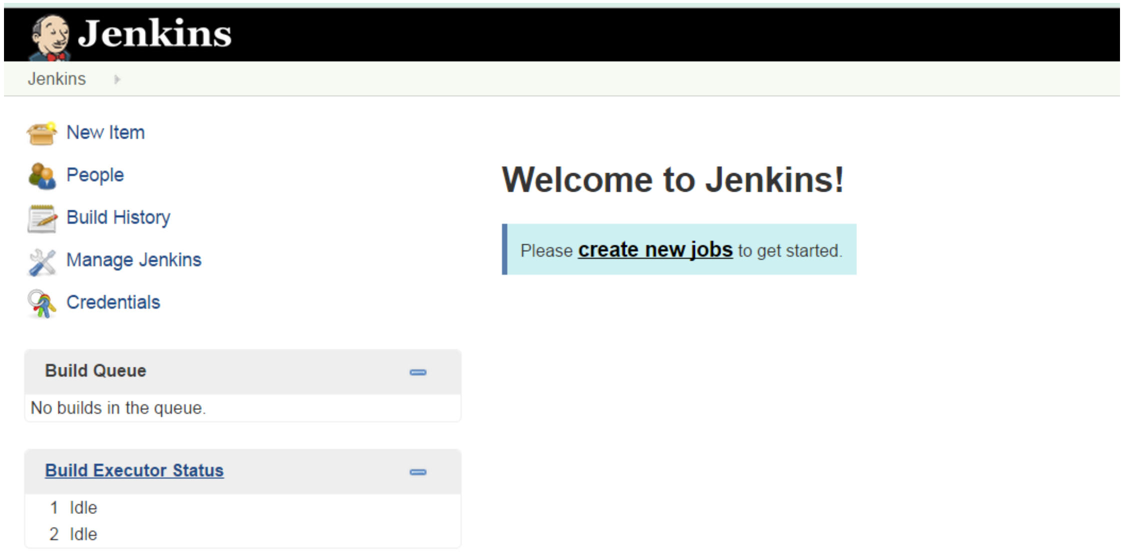
   ```
   java -jar jenkins.war
   ```

   This will install Jenkins and allow Jenkins to run as a standalone application using Jetty as the servlet engine. The following traces will eventually show in the terminal/command window.

   ```
   INFO: Jenkins is fully up and running
   Apr 13, 2016 6:21:46 PM hudson.model.UpdateSite updateData
   INFO: Obtained the latest update center data file for UpdateSource default
   Apr 13, 2016 6:21:47 PM hudson.model.DownloadService$Downloadable load
   INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
   Apr 13, 2016 6:21:48 PM hudson.model.DownloadService$Downloadable load
   INFO: Obtained the updated data file for hudson.tasks.Ant.AntInstaller
   Apr 13, 2016 6:21:53 PM hudson.model.DownloadService$Downloadable load
   INFO: Obtained the updated data file for hudson.tools.JDKInstaller
   ```

```
Apr 13, 2016 6:21:53 PM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 17,923 ms
```

4. Test that Jenkins is running by opening a web browser to `http://localhost:8080` and you will see:



Welcome to Jenkins!

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

No builds in the queue.

1 Idle
2 Idle

5. To customize the Jenkins environment, click the **`Manage Jenkins`** link. Notice the message that informs the user that the system is not secured and that it is not a good idea to run this way. We will address this in a later exercise.

6. Click the **`Configure System`** link. Let's change our Jenkins from the default settings and customize the **`System Message`**. Change it to something different like **`My Jenkins`**.

7. It is always a very important practice to set the URL to the real hostname if it didn't get configured with a real hostname. Scroll down and find **`Jenkins Location`** and set the **`Jenkins URL`** to something other than localhost. To get the correct host name, you can find the computer name in the **System Properties** of the **Control Panel** on Windows/Linux machines. Another way is to open a Command Window or Terminal and execute the **`hostname`** command.

8. This would also be a good time to set the **`admin email address`**. That is used to notify someone when the system is encountering issues. In order for Jenkins to send notification emails, it is necessary to set the **`SMTP Server information`**. This setting is nearly to the bottom. If it is not set, then Jenkins will use the default server, **`localhost`**, if available.

9. The **`CVS`** and **`Subversion`** support is installed with the initial Jenkins installation. The settings to connect to a repository need to be provided. We will not be using **`CVS`** or **`Subversion`** for this class, but this is where that definition would be done. We will

be using **`Git`,** so we will come back to this screen in a later exercise after the **`Git plugin`** is installed.

10. Save the settings by clicking the **`Save`** button. After changing system settings, it is a good idea to restart Jenkins. The easiest process is to go to the terminal window where Jenkins was started and to stop it by hitting **`Ctrl-C`**. Jenkins will stop. Next, start it again by running the same command that started it earlier. Finally, using a cursor up key, scroll up to the previous command and hit **`Enter`**.

11. Go back to the web browser and verify that Jenkins did start up by opening the page using the new hostname in the URL **`http://{the.new.host.name}:8080/`** and see if the page loads.

12. Stop Jenkins again. We will now configure Jenkins to run in a servlet container. We will be using Tomcat 8. It is very easy to get the Jenkins running in Tomcat.

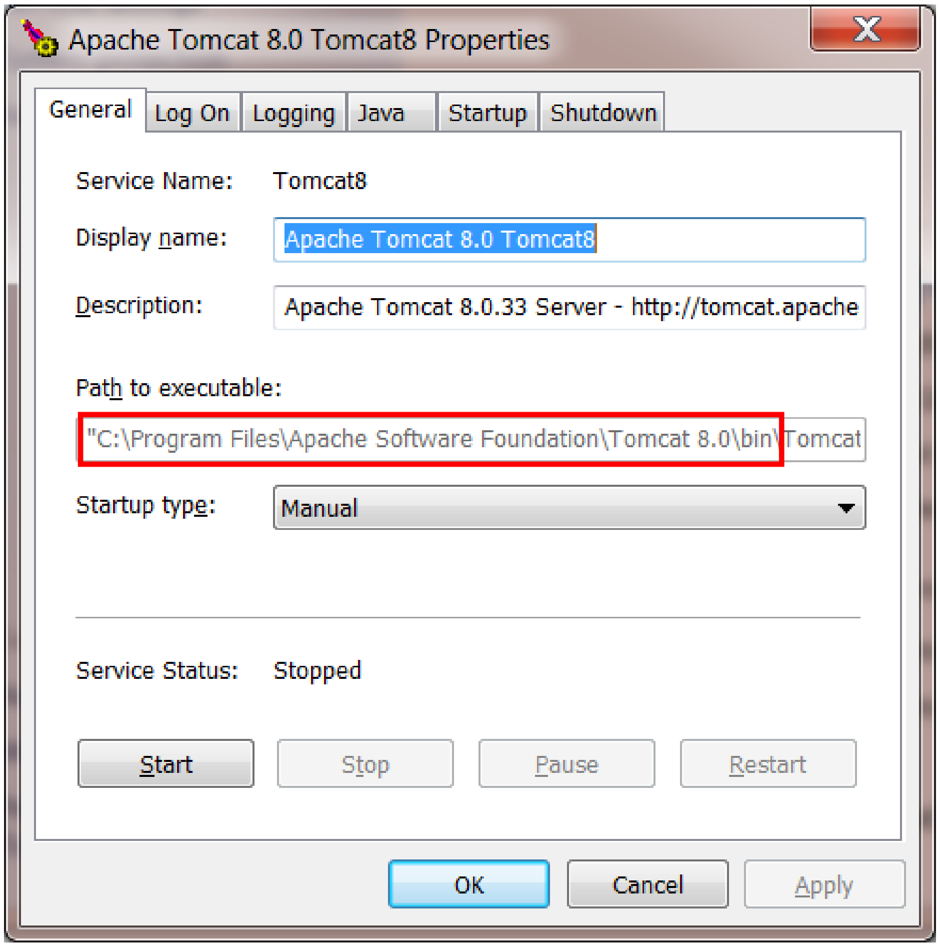
    Set a system environment variable to **`JENKINS_HOME`**.

    – For Windows: Open **`System`** in **`Control Panel`** and click **`Advanced System Settings`**. Click the **`Environment Variables`** button and click **`New`** to add a variable named **`JENKINS_HOME`** with a value of the location you installed Jenkins to in step 2 (**`C:\PF\Jenkins`**). Click **`OK`** to close all the Control Panel dialog boxes that popped up.

    – For Mac: Run the following command.
    ```
    echo `export JENKINS_HOME=/Users/yourUserName/jenkins`
    >>~/.bash_profile
    ```
    Or, you can use vi to enter the environment variable manually in the **`.bash_profile (vi .bash_profile`**).

13. Find the Apache Tomcat 8.0 installation location and deploy Jenkins.

    - For Windows: On the command line type: **`Set CATALINA_HOME`** or find the program **`Apache Tomcat 8.0 → Monitor Tomcat`** and **`Start Tomcat 8.0`**. Copy the **`jenkins.war`** file from the location you used in step 2 to the **webapps** folder of the Tomcat 8.0 installation location. You can see it in the **`Monitor Configuration`** popup.



Service Name: Tomcat8

Display name: Apache Tomcat 8.0 Tomcat8

Description: Apache Tomcat 8.0.33 Server - http://tomcat.apache

Path to executable:

"C:\Program Files\Apache Software Foundation\Tomcat 8.0\bin\Tomcat

Startup type: Manual

Service Status: Stopped

- For Mac: type **`echo $CATALINA_HOME`** in a terminal window

14. It will take a moment to deploy Jenkins. Once Jenkins has deployed to Tomcat, open it by using the URL **`http://yourHostName:8080/jenkins`** and you will see the Jenkins home page.

15. Verify that the settings were preserved by looking at the system message and verifying the message you set in **`System Message`** is displayed.

16. Since we deployed Jenkins to a servlet container, the **`URL`** is now changed. From the main Jenkins page, click on **`Manage Jenkins`**. Click on **`Configure System`**. Scroll down to the **`Jenkins Location`** and change the **`URL`** to include **`/Jenkins`** at the end. It should look something like: **`http://yourHostName:8080/jenkins`** which will be very important when users try to respond to emails or links generated by Jenkins which will include a URL.

Troubleshooting Notes:

When you try to start Jenkins in Tomcat, you could encounter an issue where Tomcat will start but is unable to deploy Jenkins. The symptoms are that you see the Tomcat home screen at `http://yourHostName:8080` but get a 404-error at `http://yourHostName:8080/jenkins`.

- Make sure you have Tomcat and Jenkins shut down **before** you copy the **jenkins.war** to the **webapps** directory.

- You may have to change the environment variables to an 8dot3 style, such as: **C:\progra~1\apache~2** instead of **C:\program files\apache-tomcat-8.0.33**.

2: Create a Jenkins Job

Overview

In this lab, you will create an instance of a job in Jenkins. The purpose of the lab is to demonstrate the broad range of challenges that are presented when building and testing an application.

Jenkins, like other continuous integration servers supports many types of build and test processes and organizes them under `Jobs`. Jenkins uses `Plugins` to support the many additional features needed to support different technologies and environments required for building applications of all sorts.

This job will be a very simple job that:

- Copies a set of Java classes and `JUnit` tests to the Jenkins workspace
- Builds the project using `Maven`

To accomplish this, you will perform these tasks:

- Configure and install Maven and its dependencies
- Define and configure a new Maven Job
- Copy the provided `simple-app` project consisting of a Java class and Junit test
- Run the Build and view the results
  - Compiles the Java classes and the Test classes
  - Executes the unit tests
  - Installs the artifact into the Maven local repository

1. The Jenkins server should already be running. Open the browser of your choice and navigate to the Jenkins URL. It should be `http://yourHostName:8080/jenkins` by default, unless it was configured for another port.

2. We are going to build a Maven project. Since this is our first job and Maven isn't initially configured, that is our first task. Click on `Manage Jenkins` link. Then, click on `Configure System` to open the system configuration settings screen.

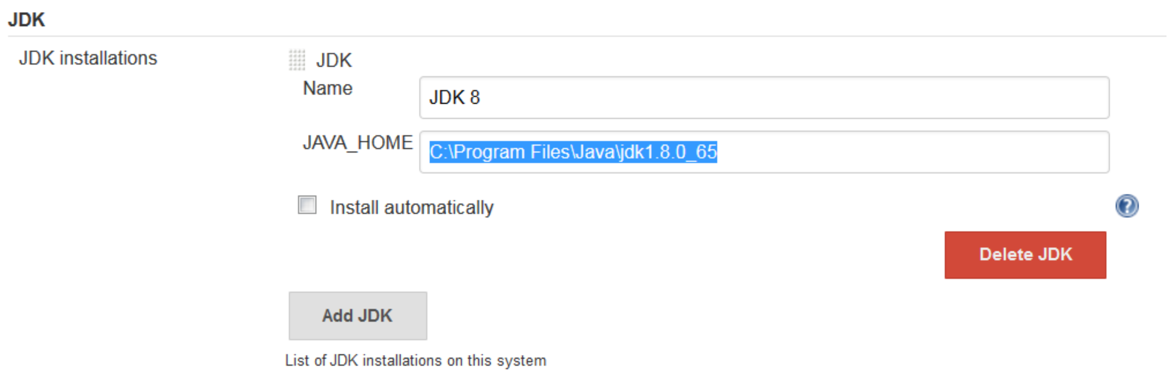
3. Scroll down to the `JDK settings` and add a new JDK by clicking the `Add` button.

   A. Name it `JDK 8`.
   B. Uncheck the **Install automatically** checkbox.
   C. Set `JAVA_HOME` to the location of your installed JDK. It must be a **JDK** not a **JRE**.

   – For Windows: `C:\Program Files\Java\jdk1.8.0_NN`
   – For Mac: `/Library/Java/JavaVirtualMachines/jdk1.8.0_NN.jdk/Contents/Home`
   (Replace `NN` with the version number of Java installed on the machine.)

If your JDK is somewhere else, you can find the JAVA_HOME by the following commands:

– For Windows in a Command window: **`set JAVA_HOME`**
– For Mac in the Terminal: **`echo $JAVA_HOME`**


JDK

JDK installations

JDK

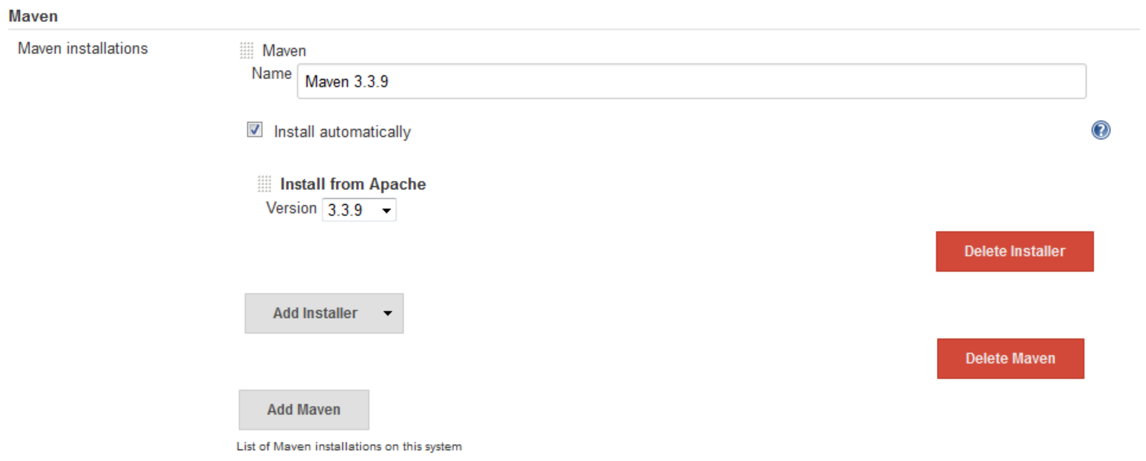
Name: JDK 8

JAVA_HOME: C:\Program Files\Java\jdk1.8.0_65

☐ Install automatically

List of JDK installations on this system

4. Next configure Maven. Click the **`Add`** button to add a new Maven installation. Name it **`Maven 3.3.9`** and leave **`3.3.9`** selected in the dropdown. Also keep the **`Install automatically`** checked so that when the build requires a Maven instance, it will be installed in the **tools** folder of Jenkins and also in the nodes when it runs in a node.


Maven

Maven installations

Maven

Name: Maven 3.3.9

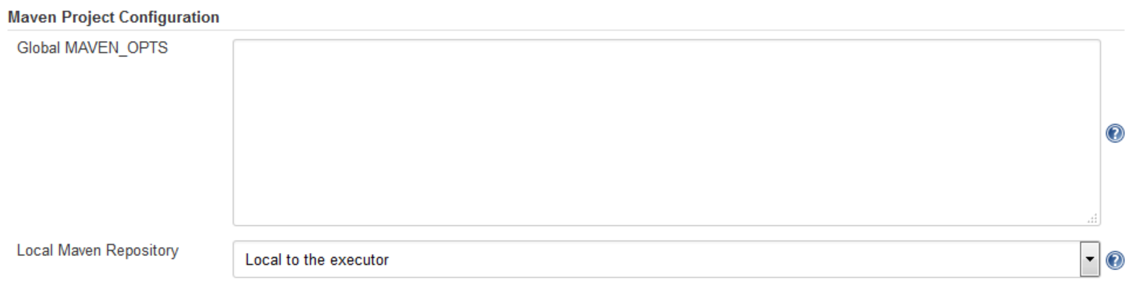
☑ Install automatically

Install from Apache

Version: 3.3.9

List of Maven installations on this system

5. Change the location of the **`Local Maven Repository`** to be **`local to the executor`**. This allows the nodes to have access to a Maven repository no matter which user the node runs under.

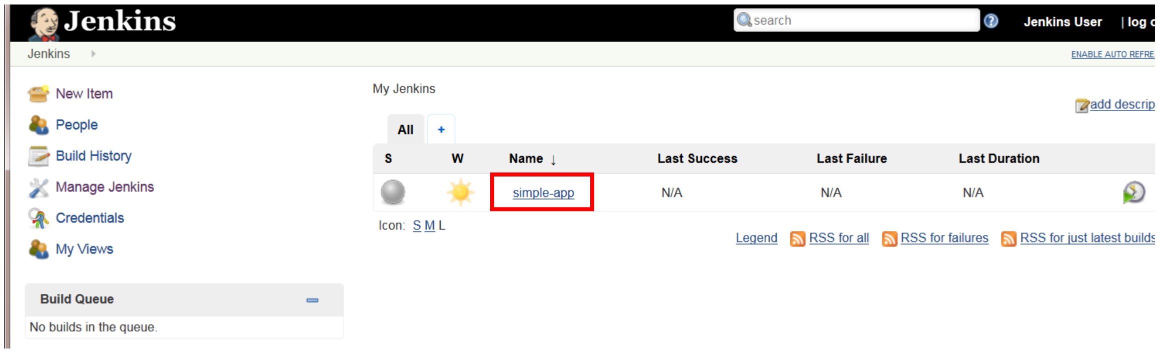

Maven Project Configuration

Global MAVEN_OPTS

Local Maven Repository: Local to the executor

6. Save the changes.

7. Define a new Maven job to build a simple Maven archetype job for a Java application. It was created using Eclipse. This job will use the Maven project, which is widely used in the industry.
   A. Select **`Create new Jobs`** link to begin defining the new job.
   B. Name the project **`simple-app`**. It is a good practice <u>not</u> to use spaces or non-alphanumeric characters in the project name. Some plugins have been known to have issues with projects that have a space in the name in the past. It doesn't happen as frequently anymore.
   C. Click the **`Save`** button.
   D. Click the **`Back to Dashboard`** link.



S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	simple-app	N/A	N/A	N/A

8. Click the link to open **<u>`simple-app`</u>**. Click the **`Configure`** link to open up the configuration settings for the **`simple-app`** job.

9. Select the **`Add pre-build step`** dropdown and select the **`execute`** command appropriate for your environment.
   – For Windows: **`Execute Windows batch command`**
   – For Mac: **`Execute shell`**

10. A **`simple-app`** project is provided in the **Setup.zip** archive that was provided with the course materials. This project has a Java class that is simply a Hello World application. It also contains a JUnit Test Case.

Enter the command to copy the folder **`simple-app`** from the **`Setup`** folder, which should be **`C:\Setup`** for Windows or the users **home folder/Setup** for Mac. The syntax to do this is below:

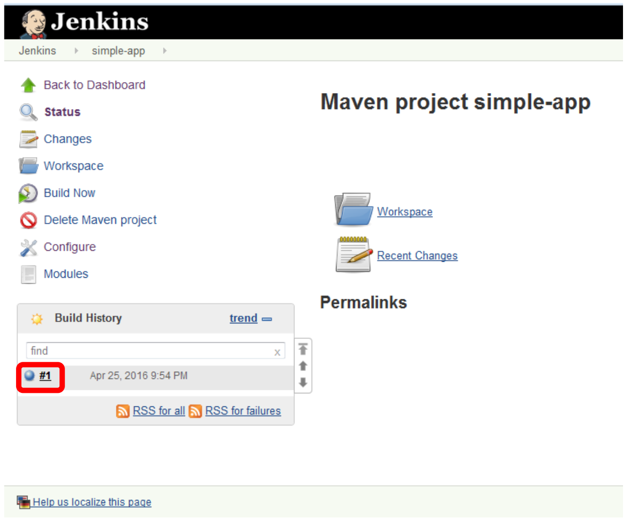
- For Windows: **`xcopy /S /Y c:\setup\simple-app\* %WORKSPACE%\*`**
- For Mac: **`cp -r ~/setup/simple-app/ .`**

  <u>Make sure there is a space between `~/setup/simple-app/` and the dot</u>

11. Scroll down to the **`Maven Goals:`** Enter **`clean install`**.

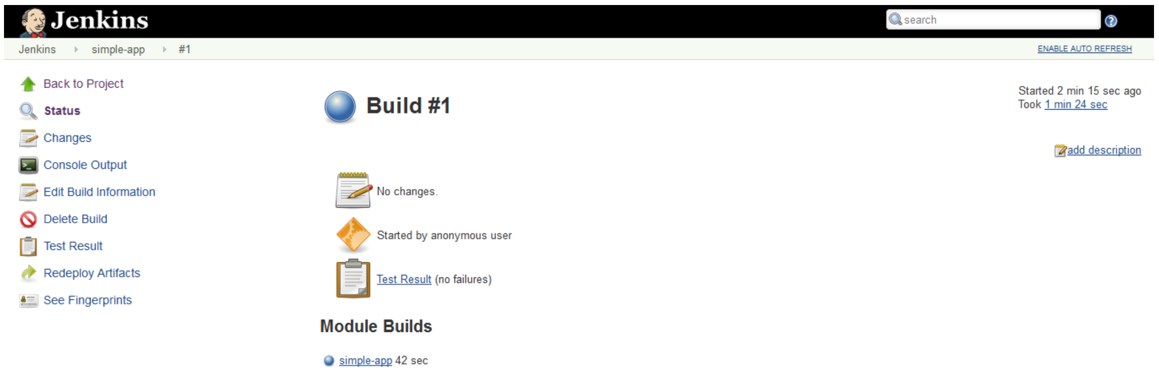
12. Click the **`Save`** button. Build the project by clicking the **`Build Now`** link.

13. The build will start and run for some time. Eventually the build will finish and there will be a blue ball by the **`#1`** under the **build history**. Each build will increment and be available in the build history. Click the **`#1`**.



Apr 25, 2016 9:54 PM

The **`Build Details`** screen will appear.



Started 2 min 15 sec ago
Took 1 min 24 sec

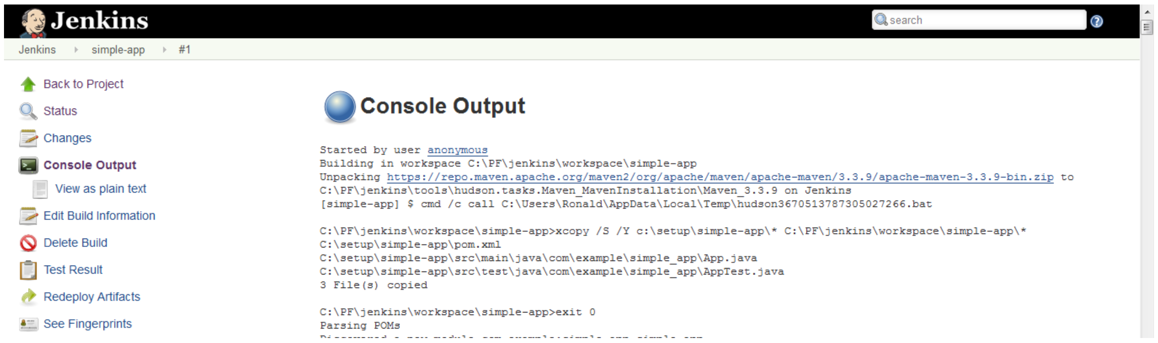
No changes.

Started by anonymous user

Test Result (no failures)

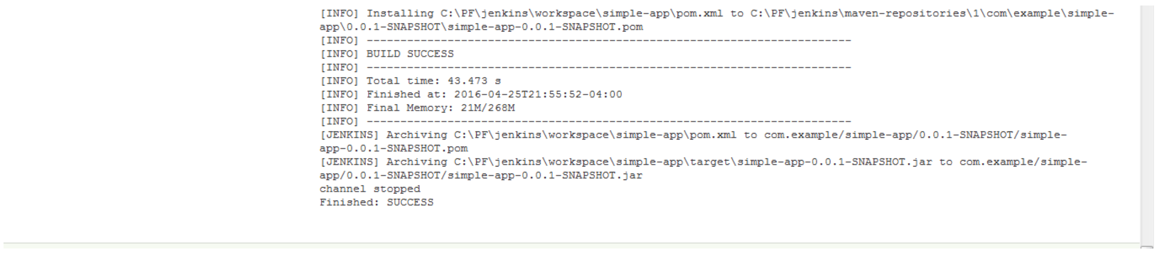
simple-app 42 sec

14. Click the **`Console Output`**. This displays the log of how the build was accomplished.



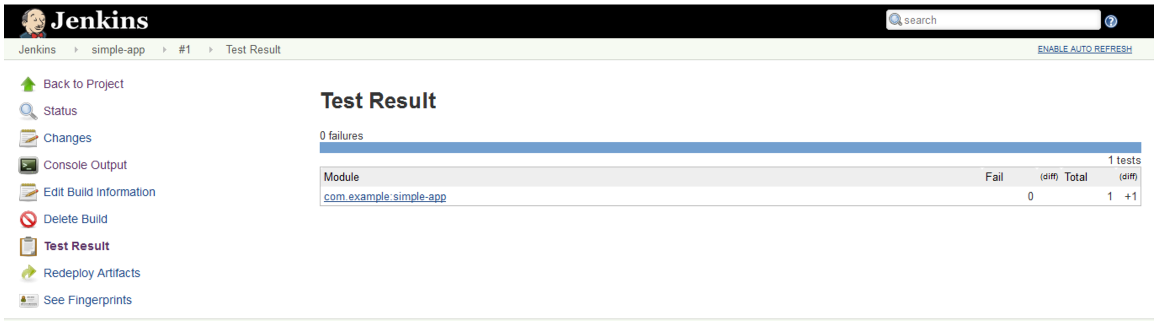
```
Started by user anonymous
Building in workspace C:\PF\jenkins\workspace\simple-app
Unpacking https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.3.9/apache-maven-3.3.9-bin.zip to
C:\PF\jenkins\tools\hudson.tasks.Maven_MavenInstallation\Maven_3.3.9 on Jenkins
[simple-app] $ cmd /c call C:\Users\Ronald\AppData\Local\Temp\hudson3670513787305027266.bat

C:\PF\jenkins\workspace\simple-app>xcopy /S /Y c:\setup\simple-app\* C:\PF\jenkins\workspace\simple-app\*
C:\setup\simple-app\pom.xml
C:\setup\simple-app\src\main\java\com\example\simple_app\App.java
C:\setup\simple-app\src\test\java\com\example\simple_app\AppTest.java
3 File(s) copied

C:\PF\jenkins\workspace\simple-app>exit 0
Parsing POMs
```

When you scroll all the way down, you should see a **`Build SUCCESS`** message.



```
[INFO] Installing C:\PF\jenkins\workspace\simple-app\pom.xml to C:\PF\jenkins\maven-repositories\1\com\example\simple-
app\0.0.1-SNAPSHOT\simple-app-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 43.473 s
[INFO] Finished at: 2016-04-25T21:55:52-04:00
[INFO] Final Memory: 21M/268M
[INFO] ------------------------------------------------------------------------
[JENKINS] Archiving C:\PF\jenkins\workspace\simple-app\pom.xml to com.example/simple-app/0.0.1-SNAPSHOT/simple-
app-0.0.1-SNAPSHOT.pom
[JENKINS] Archiving C:\PF\jenkins\workspace\simple-app\target\simple-app-0.0.1-SNAPSHOT.jar to com.example/simple-
app/0.0.1-SNAPSHOT/simple-app-0.0.1-SNAPSHOT.jar
channel stopped
Finished: SUCCESS
```

15. Click the **`Test Result`** link. This displays the results of executing the test cases.



Test Result

0 failures

1 tests

Module	Fail	(diff)	Total	(diff)
com.example:simple-app	0		1	+1

16. Click the **`Changes`** link. This will display **`No Changes`** because this project is not associated with any Version Control System.

3: Schedule a Jenkins Job

Overview

In this lab, you will schedule a job to run when various events happen.

To accomplish this, you will perform these tasks:

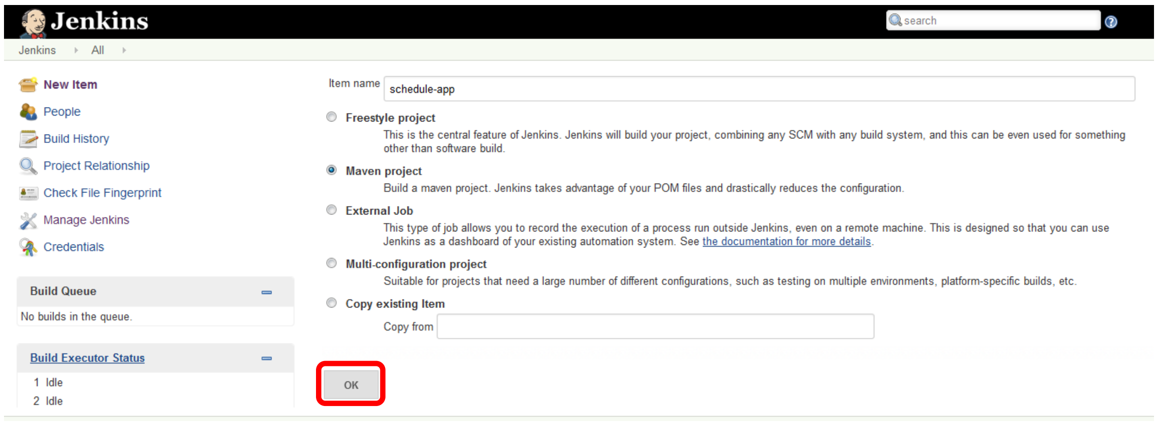
- Use Eclipse to import the provided **Cucumber** project from a **Git** repository
- Use Jenkins to define a new Maven job named **schedule-app**
- Configure the **schedule-app** job to specify the **schedule** and the **repository location**
- Run the job and examine the results
- Make a code change in Eclipse and watch the build run on schedule

1. The Jenkins server should already be running. Open the browser of your choice and navigate to the Jenkins URL. It should be **`http://yourHostName:8080/jenkins`** by default, unless it was configured for another port.

2. First, using Eclipse, you will import a project from Git.
   A. Open **`Eclipse`** and click **`File`** → **`Import…`**
   B. When the dialog pops up select **`Existing Maven Projects`** under **`Maven`** and click **`Next`**.
   C. Click **Browse…** and navigate to:
      – For Windows: **`C:\setup\git\cuctest`**.
      – For Mac: **`/Users/yourUserName/Setup/git/cuctest`**.
   D. Click **`OK`** or **`Open`**. Check the Project and click **`Finish`**.

   The project will be imported. On the file system, navigate to the project and examine the contents of the **`cuctest`** project.

3. Back in Jenkins, you will define a new Maven job to build a simple Maven archetype job for a Java application. It was created using Eclipse.
   A. Select **`Create new Jobs`** link to begin defining the new job.
   B. Name the project **`schedule-app`**. Again, it is a good practice not to use spaces or non-alphanumeric characters in the project name.
   C. Click the **`OK`** button.


Jenkins
Jenkins > All >

- New Item
- People
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- Credentials

Build Queue
No builds in the queue.

Build Executor Status
1 Idle
2 Idle

Item name: schedule-app

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.

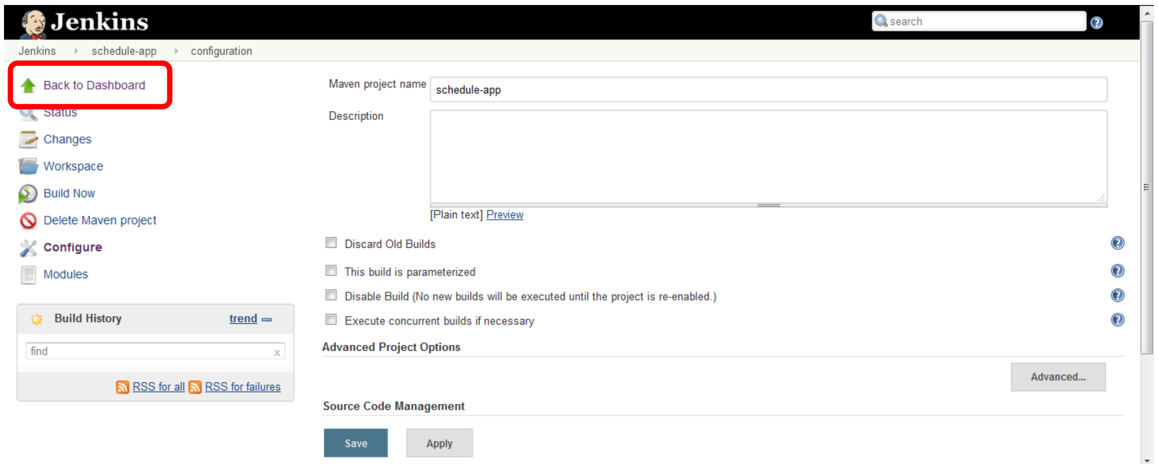
External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

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

OK

D. Click the **`Back to Dashboard`** link.


Jenkins
Jenkins > schedule-app > configuration

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Maven project
- Configure
- Modules

Build History trend
find
RSS for all RSS for failures

Maven project name: schedule-app
Description:
[Plain text] Preview

- [ ] Discard Old Builds
- [ ] This build is parameterized
- [ ] Disable Build (No new builds will be executed until the project is re-enabled.)
- [ ] Execute concurrent builds if necessary

Advanced Project Options
Advanced...

Source Code Management

Save Apply

4. Next, we will configure the **`schedule-app`** job. Click the link to open **`schedule-app`**. Click the **`Configure`** link to open up the configuration settings. There are many settings that can be set to control the build process. The ones that we are going to set are the **`schedule`** and the **`repository location.`**

5. Now, you will associate the job with the Git repository. To do this, scroll down to the **`Source Code Management`** section. Select the **`Git`** radio button. (If you don't see the Git radio button look at the end of this lab under troubleshooting tips on how to solve this.) This will open up settings to allow mapping the job to a Git repository. Set the location to:
   – For Windows: **`C:\Setup\git\cuctest`**
   – For Mac: **`/Users/yourUserName/setup/git/cuctest`**

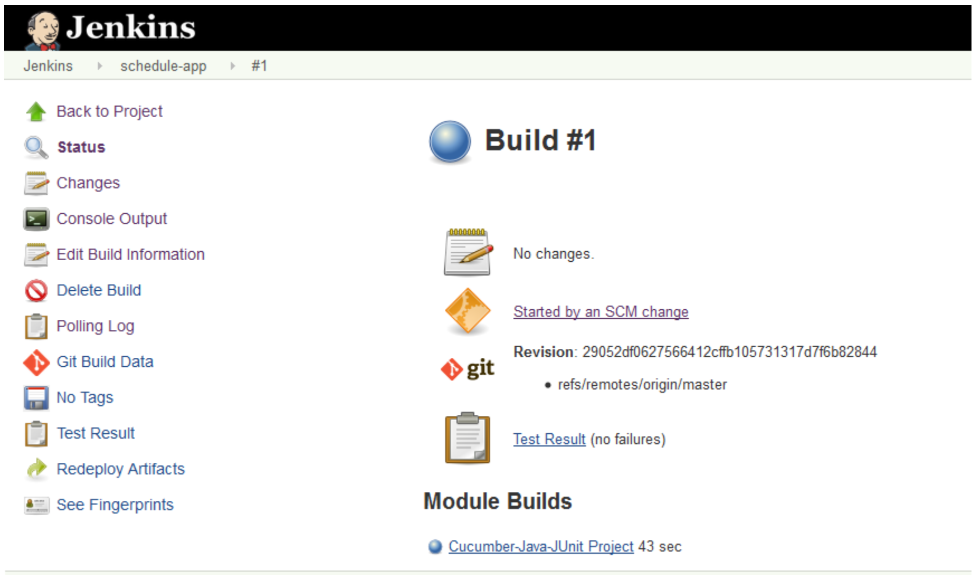
   Clicking the **Advanced** button will allow specifying more detailed control over how Git repository mapping is managed. The Advanced settings are not applicable to this exercise since this is a local repository with no remote.

6. Select the `Poll SCM` radio button to enable the schedule field. This is where the frequency of polling is defined. It is a `cron` style format. To find out appropriate values click the `?` to the right of the field.

   We will use `H/5 * * * *` to specify that it will be every 5 minutes hashed randomly based on the job name. The `H` on the front will make jobs with the same 5 minute interval not start at the same minute.

7. Save the configuration.

8. Navigate to the main page. Keep refreshing the page for about 5 minutes and eventually the `Executor` will poll the `SCM` and find that there was a change. When the blue ball appears indicating a build has completed, select the `#1` link to see the results of the build.



Build #1

No changes.

Started by an SCM change

Revision: 29052df0627566412cffb105731317d7f6b82844

- refs/remotes/origin/master

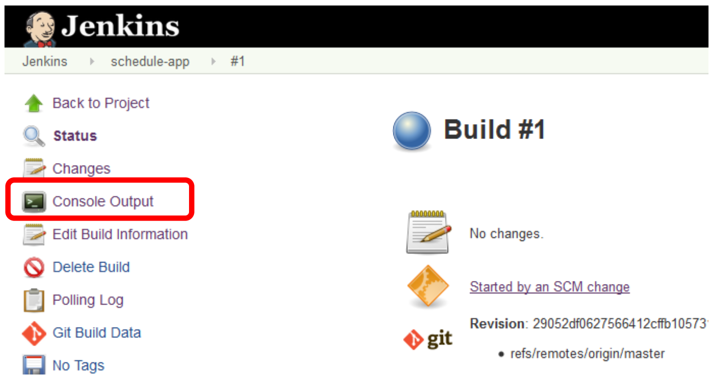
Test Result (no failures)

Module Builds

Cucumber-Java-JUnit Project 43 sec

9. Things to notice in the build:
   - Triggered by SCM change
   - Git retrieved content
   - Maven was discovered as missing, so it was downloaded and installed
   - Maven parsed pom.xml
   - Downloads for all the dependencies
   - Cucumber tests run 5 total 3 skipped (Click on `Test Result` to see this)

10. Now we will update the **Stepdefs.java** in Eclipse.
    A. In Jenkins, click on the **schedule-app** and on the **#1** next.
    B. On the left, click on **Console Output**.



Build #1

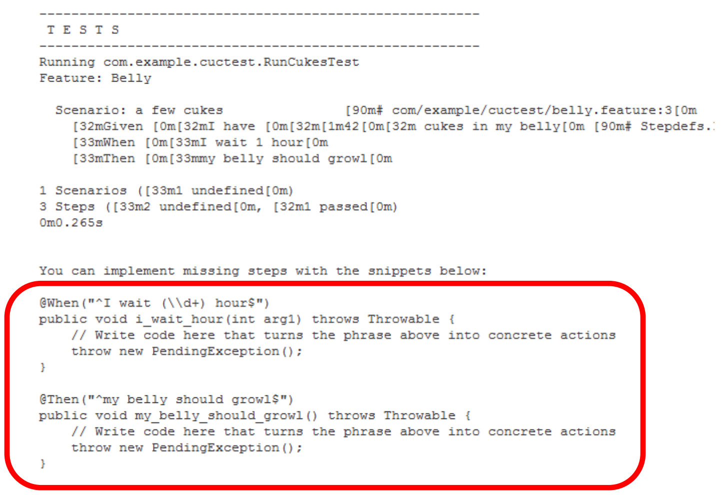
No changes.

Started by an SCM change

Revision: 29052df0627566412cffb10573

- refs/remotes/origin/master

C. When you scroll down you will see the following. Copy the snippets from the browser into Eclipse to **Stepdefs.java**.



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.example.cuctest.RunCukesTest
Feature: Belly

  Scenario: a few cukes               [90m# com/example/cuctest/belly.feature:3[0m
    [32mGiven [0m[32mI have [0m[32m[1m42[0m[32m cukes in my belly[0m [90m# Stepdefs.
    [33mWhen [0m[33mI wait 1 hour[0m
    [33mThen [0m[33mmy belly should growl[0m

1 Scenarios ([33m1 undefined[0m)
3 Steps ([33m2 undefined[0m, [32m1 passed[0m)
0m0.265s


You can implement missing steps with the snippets below:

@When("^I wait (\\d+) hour$")
public void i_wait_hour(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^my belly should growl$")
public void my_belly_should_growl() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

D. Eclipse will show some code errors because some import statements are missing. To fix that you will need to add (organize) the imports.
   - For Windows: ctrl+shift+O
   - For Mac: ⌘+shift+O

E. Save the file and commit.
   - Access the project's context menu in Eclipse. (Right-click the project on Windows or Control+Click the project on Mac.) From the project's context menu, select **`Team`** → **`Commit.`**
   - Add the comment **`Add the missing test cases.`**

11. Wait for a few minutes until the interval triggers a build.

12. Notice that the build triggered and the missing test cases were run. Since there was not a valid implementation of the methods, there will be a message indicating that there is still work to be done. However, the goal of this exercise was to configure a job that builds on a schedule and automatically integrates code changes. You have accomplished the continuous integration for the build.

Troubleshooting Tips:

- If you don’t see the **`Git`** radio button you, do not have the **`Git plugin`** installed.
  - Click on the **`Jenkins`** link in the left top corner
  - Click **`Manage Jenkins`** → **`Manage Plugins`**
  - Click the **`Available`** tab and type **`Git plugin`** and check the box to the left of it.
  - And click **`Download now and install after restart`**
  - Restart Jenkins

4: Secure Jenkins

Overview

In this lab, you will configure the security settings in Jenkins to prevent unauthorized access to the Jenkins build machine. Enabling security is an industry best practice.

First, you will start with a Jenkins environment which has not been secured and will explore the various options for securing Jenkins. Then, you will secure the environment using the built-in Jenkins database. Initially you will enable authentication, and later you will add authorization for full or limited access.

The general steps are:

- Enable security
- Self-register a user
- Assign user privileges
- Self-register a different user
- Give users different privileges
- Switch users and notice what is permitted

1. Open the Jenkins URL **`http://yourHostName:8080/jenkins`** to see the main Jenkins page. Notice that there is no link for adding a user, or a place to log in. That is because security is not enabled yet. We will now configure security. From the main Jenkins page, click **`Manage Jenkins`**. Click on **`Configure Global Security`**.

2. The **`Configure Global Security`** page has only a couple of items to choose until the **`Enable Security`** checkbox is checked, so check it now. That will enable options for securing Jenkins. Click on **`Jenkins own user database`** radio button. Click on **`Allow users to sign up`** to enable that feature.

3. Under **`Authorization`**, click the radio button to select **`Logged in users can do anything`**. Later we will change the security settings, but for now this is a good starting point.

4. Save the configuration and Jenkins will reload the main Jenkins screen, but this time with security features enabled. **`Log in`** and **`sign up`** links are now available and there is a place to enter login credentials. However, before anyone can log in as a user, they must sign up.

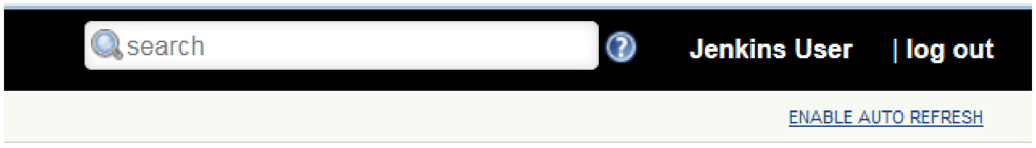
5. To enable credentials to log in, first sign up. Click the **`sign up`** link.

   Fill in the fields as shown below:

Username:	**`admin`**
Password:	**`secret`**

Confirm Password:	**`secret`**
Full Name:	**`Jenkins User`**
E-mail address:	**`jenkins.user@example.com`**

6. Click the **`sign up`** button and Jenkins will redirect to a page with confirmation that you are now logged in. The user name displays in the top right corner. Next to that is a **`log out`** link to allow the user to log out. Clicking the link **`to the top page`** will take the user back to the main Jenkins page.



7. Click the **`Manage Jenkins`** link and the **`Configure Global Security`** link to reconfigure security now that there is a user.

   Click the radio button for **`matrix-based security`** and add the user just created with all permissions. To do that, enter **`admin`** in the **`User/group to add`** field and click the **`add`** button. The user will be added to the matrix.

   Click the icon next to the red box with an X on it that resembles a notebook page. All the permissions are now checked. For **`anonymous`**, select **`Read`** in the **`Overall`** section, **`Read`** under the **`View`** section, and **`Read`** under the **`Job`** section. Hovering over the permission with a mouse will pop up the details regarding the permission. Click **`Save`** to make the changes active.

8. Log out of Jenkins. Register a new user:

Username:	**`ouser`**
Password:	**`secret`**
Confirm Password:	**`secret`**
Full Name: Ordinary	**`User`**
E-mail address:	**`ordinary.user@example.com`**

9. Save the user. This ordinary user does not have the ability to create jobs, but can view jobs.

10. Log out and log back in as **`ouser`** and notice that the user has only limited functionality.

11. Log back in as **`admin`** and notice that full functionality is available.

You have now enabled authentication and authorization in Jenkins.

5: Complex Jenkins Projects

Overview

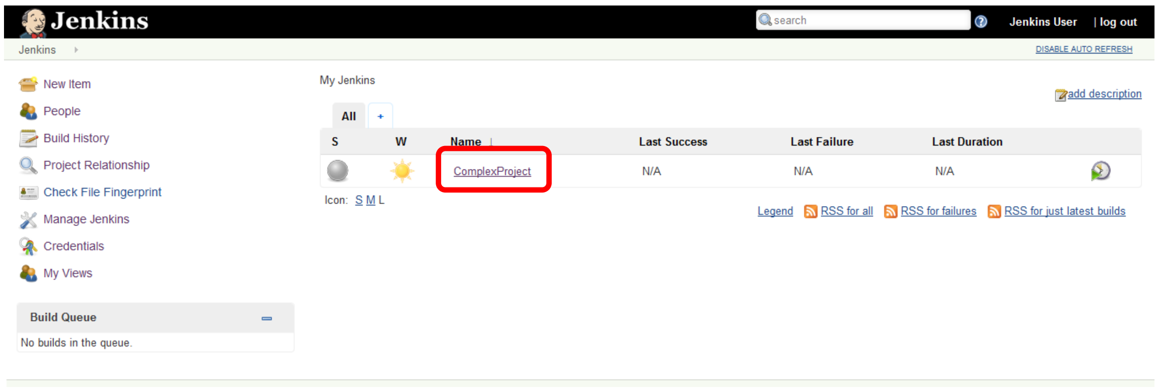
In this lab, you will create an instance of a job in Jenkins that includes more than just simple Java projects. You will use Jenkins to configure a job with a configuration matrix to build a complex project.

The general steps are:

- Create a Multi-configuration project
- Define the configuration matrix
- Add a build step to execute a Windows batch command or (Mac shell)
    - Copies a file from one folder to another
- Build the project
- View the results

1. The Jenkins server should already be running. Open the browser of your choice and navigate to the Jenkins URL. It should be `http://yourHostName:8080/jenkins` by default, unless it was configured for another port.

2. Define a new job to copy a file from one folder to another. This job will use the **`Multi-configuration project`**. There are other choices available but to begin with this is the most common project type for generic build processes. (Note: If your organization is already using Maven, then that would be used by your team.)
   A. Select the **`New Item`** link to begin defining the new job.
   B. Name the project **`ComplexProject`**. Select the **`Multi-configuration project`** radio button and click **`OK`**
   C. Click the **`Back to Dashboard`** link.



Jenkins | search | Jenkins User | log out

Jenkins ›

DISABLE AUTO REFRESH

New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
Credentials
My Views

Build Queue
No builds in the queue.

My Jenkins

add description

All +

S	W	Name	Last Success	Last Failure	Last Duration
●	☀	ComplexProject	N/A	N/A	N/A

Icon: S M L

Legend | RSS for all | RSS for failures | RSS for just latest builds

3. Click the **`ComplexProject`** link and then the **`Configure`** button.

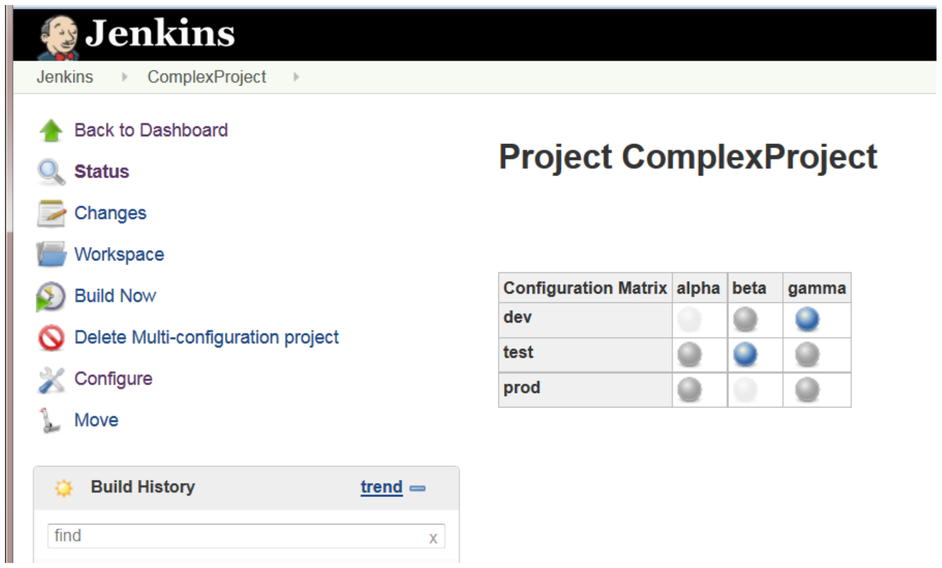
4. Scroll down to the **`Configuration Matrix.`** Click **`Add Axis`** and select **`User-defined axis`**. Enter **`target`** for the name. For the values enter **`alpha beta gamma`**.

5. Add another axis named **releaseenv** with values **dev test prod**.

6. Scroll down to the **Build** and add a **build step**.
   - For Windows: **execute windows batch command** and add the lines:
     - **echo %target%**
     - **echo %releaseenv%**
   - For Mac: **execute shell** and add the lines:
     - **echo ${target}**
     - **echo ${releaseenv}**

   Click the **Save** button.

7. Click the **ComplexProject** project link.
   A. Click the **Build Now** link. The build will run for a few seconds and the **Build History** will update with the timestamp of the build and a blue icon indicating a successful build.
   B. The matrix will update as the jobs are executed and the balls will turn from grey to blue:



Project ComplexProject

Configuration Matrix	alpha	beta	gamma
dev	○	●	🔵
test	●	🔵	●
prod	●	○	●

   C. Click the **#1** build link in the **Build History**.
   D. Click the arrow next to the blue ball at the intersection of **dev** and **alpha**. Click the **Console Output** link.

   The console output should look similar to this:

```
Started by upstream project "ComplexProject" build number 3
originally caused by:
 Started by user Jenkins User
Building on master in workspace
    C:\PF\jenkins\workspace\ComplexProject\releaseenv\dev\target\alpha
[alpha] $ cmd /c call "C:\Program Files\Apache Software Foundation\Tomcat
    8.0\temp\hudson2190539730012835759.bat"

C:\PF\jenkins\workspace\ComplexProject\releaseenv\dev\target\alpha>echo alpha
alpha

C:\PF\jenkins\workspace\ComplexProject\releaseenv\dev\target\alpha>echo dev
dev

C:\PF\jenkins\workspace\ComplexProject\releaseenv\dev\target\alpha>exit 0
Finished: SUCCESS
```

E. Notice in the console that the user is now set on build. That is because security has been enabled for Jenkins. That will be important to track which users have contributed code that caused the builds to fail.
F. Click the **`Back to Project`** link twice.
G. Click the **`Back to Dashboard`** link.

6: Using Plugins

Overview

In this lab, you will enhance the Jenkins build process by installing Plugins and adding them to jobs.

Some very useful plugins for a continuous integration improvement process are **FindBugs** and **JaCoCo**. It is easy to improve the quality of code by adding these to the build process to get a better understanding of the overall quality of the code. This lab will show how easy it is to add that functionality to the build process.

You will perform these tasks:

- Install several plugins for code quality
- Create a new Maven project named **`simple-app2`**
- Add a pre-build step to copy the provided application from the setup folder
- Configure the build to use the plugins
- Add post-build steps to produce the code quality reports
- Build the project
- View the code quality reports produced

1. The Jenkins server should already be running. Open the browser of your choice and navigate to the Jenkins URL. It should be **`http://yourHostName:8080/jenkins`** by default, unless it was configured for another port.

2. Click on **`Manage Jenkins`** → **`Manage Plugins`**.

3. Click on the tab **`Available`**. In the search field, enter **`Checkstyle`**. Check the checkbox to the left of the **Checkstyle** plugin.

4. Repeat the above step for the following plugins.
   - **`FindBugs`**
   - **`PMD`**
   - **`JaCoCo`**
   - **`Static Analysis Collector`**

5. After checking the checkbox by all the plugins listed above, click the button labeled **`Download now and install after restart.`** While none of these have been installed before and it is safe to install without restart, it is almost always better to restart after installing plugins. Wait for the plugins to download.

6. Restart Tomcat.

7. Go back to the Jenkins main page.

8. Create a **New Item**. Name it **simple-app2**. Use the **Maven project**. Click **OK**.

9. Configure the new item just created by clicking the link **simple-app2** and then clicking **Configure**.

10. Select **Add pre-build step** dropdown and select the execute command appropriate for your environment.

For Windows:	**Execute Windows batch command**
For Mac:	**Execute shell**

11. Enter the command to copy the folder **simple-app2** from the course setup folder, which should be **C:\Setup** for Windows or the user's **home folder/Setup**. Syntax below:

For Windows: `xcopy /S /Y c:\setup\simple-app\* %WORKSPACE%\*`

For Mac: `cp -r ~/setup/simple-app/ .` (← **Note the space and dot**)

12. Scroll down to the **Build** and under **Goals and options** enter:

```
clean install pmd:pmd findbugs:findbugs checkstyle:checkstyle site
```

13. Scroll down to the **Build Settings.** Check the checkboxes by the **Publish the Checkstyle analysis results**. Do the same for **FindBugs** and **PMD.** Checking the checkboxes triggers the plugins to be run on the next build.

14. Add a **Post-build Action** to **Publish combined analysis report**. This plugin will aggregate the results from the PMD, FindBugs and CheckStyle plugins into a single report.

15. Add a **Post-build Action** to add **Record JaCoCo coverage report**. The JaCoCo coverage report will use the results of the unit tests of the application to determine the percentage of code being covered by test cases.

16. Save the configuration.

17. Click the **Build Now** button. The build will take a bit longer this time as there is much more to the build process.

18. Eventually the build will finish. Click the **Status** link or click the **simple-app2** link near the top of the page (like a header with breadcrumbs).

19. Click the **PMD Warnings**. There are duplicate imports and unused imports. Both are yellow as they are not considered terribly bad. Click the **Details** tab and the code will be displayed.

20. Click the **FindBugs Warnings**. There are three warnings regarding useless code. Click on the **Details** tab and see where the useless code is.

7: Distributed Builds in Jenkins

Overview

In this lab, you will create a distributed build system and execute jobs on the node.

To do this, you will perform these tasks:

- Ensure Tomcat is running as administrator
- Define a **`Dumb Slave`** node
- Define a new Maven job named **`distributed-app`**
- Configure the build **`schedule`** and **`repository location`**
- Restrict the job to run only on the node you defined earlier
- Configure additional job settings
- Run the job and review the results

1. Test that Jenkins is running by opening a web browser to **`http://yourHostName:8080/jenkins`** and you will see the Jenkins home page with the Welcome screen.

2. For this exercise, ensure Tomcat is started as administrator.
   For Windows:
   - In the Start Menu find the **`Command Prompt`**
   - Right click → **`Run as administrator`**
   - Type **`C:\Program Files\apache-tomcat-8.0.33\bin\startup`**

   For Mac:
   - Open the Terminal
   - Type: **`sudo /library/tomcat/bin/startup.sh`**

3. In order to configure jobs to run on nodes, the nodes need to be defined. Go back to the Jenkins main Page. Click the **`Manage Jenkins`** link. Click the **`Manage Nodes`** link.

4. This screen is where master and slave configuration is managed. Clicking on the **`master`** shows there has been no activity on the master yet. Click the **`back`** button on the browser and click the **`New Node`** link. Name the new node **`JenkinsNode1.`** Select the **`Dumb Slave`** radio button and click **`OK`**.

5. This screen is where the details of the slave are defined.
   A. Set the **`remote root directory.`**
   - For Windows: **`C:\PF\JenkinsNode1`**
   - For Mac: **`/Users/YourUserName/JenkinsNode1`**

   Set the Launch Method: **`Launch Slave agent via Java Web Start`**

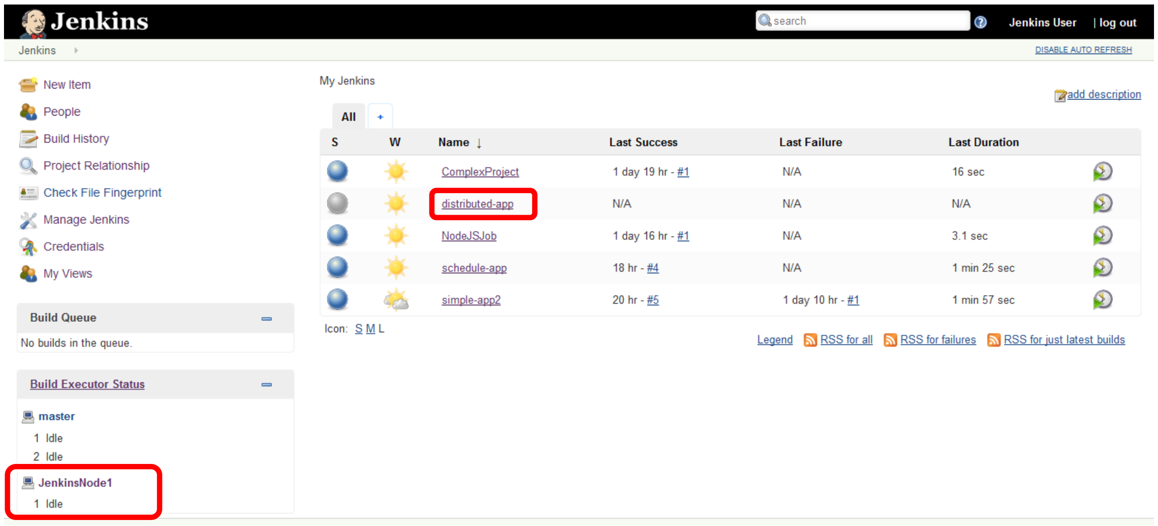
B. Click **save** and on the next screen click the **`Launch`** button.
NOTE: If prompted to install a later version of Java, skip the install if there is already Java installed.

C. This will download a **`slave-agent.jnpl`** application.
   – For Windows: This application should start automatically. If not, start it manually.
   – For Mac: This application will not start automatically.
      – Control-click on the download and select **Show in Finder**
      – Control-click and select **Open**

D. Click **`Run`** and eventually an agent will pop up. This is running a Jenkins node agent. Keep it running so Jenkins can route jobs to it.

6. If that fails, the service can be run from the command line or launched from Jenkins when needed. To run from the command line, go to the directory of the node and enter the command displayed in Jenkins for the node.

7. Define a new Maven job to build a simple Maven archetype job for a Java application. It was created using Eclipse. This job will use the Maven project.
A. Select the **`Create new Jobs`** link to begin defining the new job.
B. Name the project **`distributed-app`**. Again, we are avoiding spaces or non-alphanumeric characters in the project name.
C. Click the **`Create`** button.
D. Click the **`Back to Dashboard`** link.



Jenkins | search | Jenkins User | log out
Jenkins
DISABLE AUTO REFRESH

New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
Credentials
My Views

Build Queue
No builds in the queue.

Build Executor Status
master
1 Idle
2 Idle
JenkinsNode1
1 Idle

My Jenkins
add description
All +

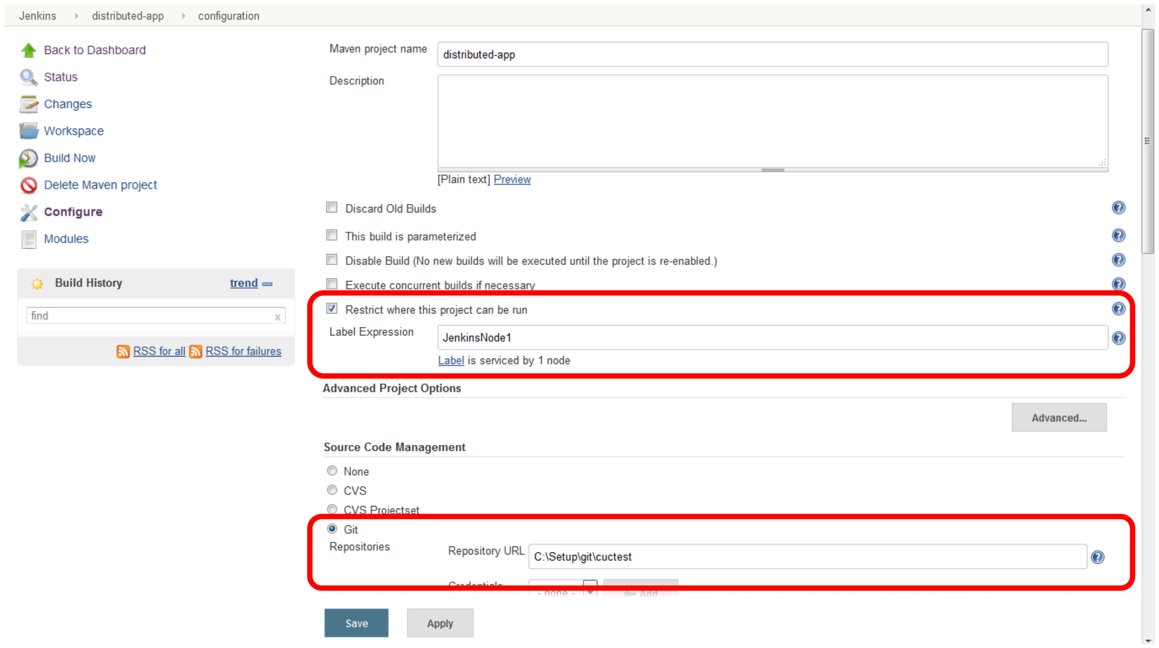
S	W	Name ↓	Last Success	Last Failure	Last Duration
		ComplexProject	1 day 19 hr - #1	N/A	16 sec
		distributed-app	N/A	N/A	N/A
		NodeJSJob	1 day 16 hr - #1	N/A	3.1 sec
		schedule-app	18 hr - #4	N/A	1 min 25 sec
		simple-app2	20 hr - #5	1 day 10 hr - #1	1 min 57 sec

Icon: S M L
Legend RSS for all RSS for failures RSS for just latest builds

8. Click the link to open **`distributed-app`**. Click the **`Configure`** link to open up the configuration settings. There are many settings that can be set to control the build process. The ones that are going to be set are the **`schedule`** and the **`repository location`**, as we did in an earlier exercise. The next step will be familiar.

9. First, you will associate the job with Git repository.
   A. Scroll down to the `Source Code Management` section.
   B. Select the `Git` radio button. This will open up settings to allow mapping the job to a Git repository.
   C. Enter the Repository URL
      – For Windows: `C:\Setup\git\cuctest`
      – For Mac: `/Users/YourUserName/setup/git/cuctest/`

10. Find the setting for `Restrict where this project can be run` and check the box. In the entry field, type the name of the node you created earlier: `JenkinsNode1`.


Jenkins › distributed-app › configuration

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Maven project
- Configure
- Modules

Build History trend
find
RSS for all RSS for failures

Maven project name: distributed-app
Description:
[Plain text] Preview
☐ Discard Old Builds
☐ This build is parameterized
☐ Disable Build (No new builds will be executed until the project is re-enabled.)
☐ Execute concurrent builds if necessary
☑ Restrict where this project can be run
Label Expression: JenkinsNode1
Label is serviced by 1 node
Advanced Project Options
Advanced...
Source Code Management
○ None
○ CVS
○ CVS Projectset
● Git
Repositories
Repository URL: C:\Setup\git\cuctest
Save Apply

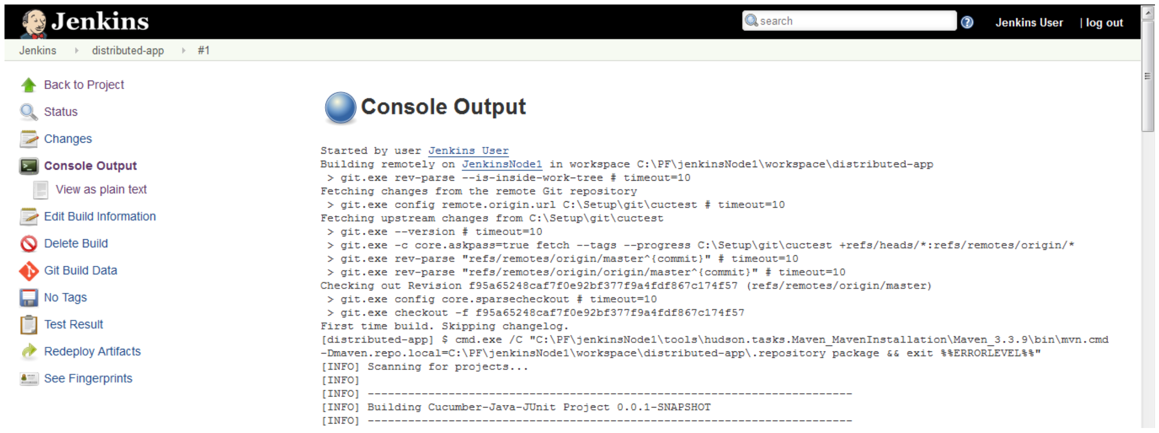
11. Set the **Maven version** to use to **Maven 3.3.9** in the dropdown. Set the **goal** to **package**.
   – Scroll to `Pre Steps` and click `Add pre-build step`
   – Select `Invoke top-level Maven targets`
   – Set `Maven Version` to `Maven 3.3.9`
   – Set `Goals` to `package`

12. Save the changes.

13. Run the job. Notice that it did not run on the master. It ran on the node. If the master is not busy it would normally have picked up the job and run it.

14. Review the logs. Notice that Maven was downloaded for use by the node. Even though the master and the slave are on the same machine they do not share the same tool folders. They also do not share the same repository.

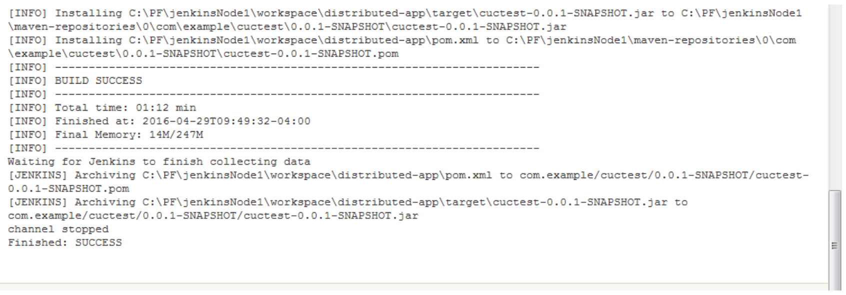
    Beginning of the log:



Console Output

```
Started by user Jenkins User
Building remotely on JenkinsNode1 in workspace C:\PF\jenkinsNode1\workspace\distributed-app
 > git.exe rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git.exe config remote.origin.url C:\Setup\git\cuctest # timeout=10
Fetching upstream changes from C:\Setup\git\cuctest
 > git.exe --version # timeout=10
 > git.exe -c core.askpass=true fetch --tags --progress C:\Setup\git\cuctest +refs/heads/*:refs/remotes/origin/*
 > git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
 > git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision f95a65248caf7f0e92bf377f9a4fdf867c174f57 (refs/remotes/origin/master)
 > git.exe config core.sparsecheckout # timeout=10
 > git.exe checkout -f f95a65248caf7f0e92bf377f9a4fdf867c174f57
First time build. Skipping changelog.
[distributed-app] $ cmd.exe /C "C:\PF\jenkinsNode1\tools\hudson.tasks.Maven_MavenInstallation\Maven_3.3.9\bin\mvn.cmd
-Dmaven.repo.local=C:\PF\jenkinsNode1\workspace\distributed-app\.repository package && exit %%ERRORLEVEL%%"
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Cucumber-Java-JUnit Project 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
```

End of the log:



```
[INFO] Installing C:\PF\jenkinsNode1\workspace\distributed-app\target\cuctest-0.0.1-SNAPSHOT.jar to C:\PF\jenkinsNode1
\maven-repositories\0\com\example\cuctest\0.0.1-SNAPSHOT\cuctest-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\PF\jenkinsNode1\workspace\distributed-app\pom.xml to C:\PF\jenkinsNode1\maven-repositories\0\com
\example\cuctest\0.0.1-SNAPSHOT\cuctest-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:12 min
[INFO] Finished at: 2016-04-29T09:49:32-04:00
[INFO] Final Memory: 14M/247M
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving C:\PF\jenkinsNode1\workspace\distributed-app\pom.xml to com.example/cuctest/0.0.1-SNAPSHOT/cuctest-
0.0.1-SNAPSHOT.pom
[JENKINS] Archiving C:\PF\jenkinsNode1\workspace\distributed-app\target\cuctest-0.0.1-SNAPSHOT.jar to
com.example/cuctest/0.0.1-SNAPSHOT/cuctest-0.0.1-SNAPSHOT.jar
channel stopped
Finished: SUCCESS
```

8: Run NodeJS Plugin in Jenkins

Overview

In this lab, you will use Jenkins to perform a build using the NodeJS environment.

NodeJS works much better on Linux than on Windows or Mac. They really only support one version of NodeJS per build. It is necessary to change the PATH for each build to support multiple versions of NodeJS. For developers, this should not be a problem.

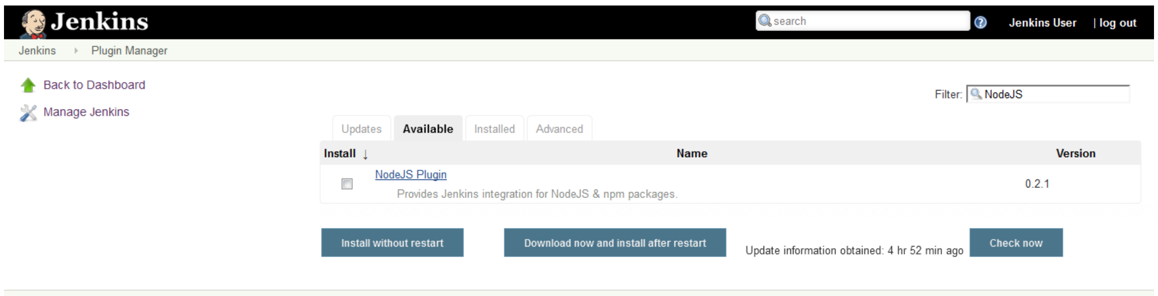
The general steps that you will perform are:

- Install the **NodeJS plugin** and configure it
- Create a **`Freestyle`** job named **`NodeJSJob`**
- Add a build step to display the versions in use
- Add a **`NodeJS script`** build step to write to the console the build number and current working directory path
- Build the job and view the output to verify NodeJS is working in Jenkins

1. Test that Jenkins is running by opening a web browser to `http://yourHostName:8080/jenkins` and you will see the Jenkins home page.

2. To enable Jenkins to use NodeJS, the plugin must be installed. Click on **`Manage Jenkins`** to get to the management screens. Click the **`Manage Plugins`** link to view the page that allows plugins to be added and updated.

3. Click on the **`Available`** tab, search for **`NodeJS`** and then check the box by **`NodeJS`**. Click the **`Install without Restart`** button to install it.



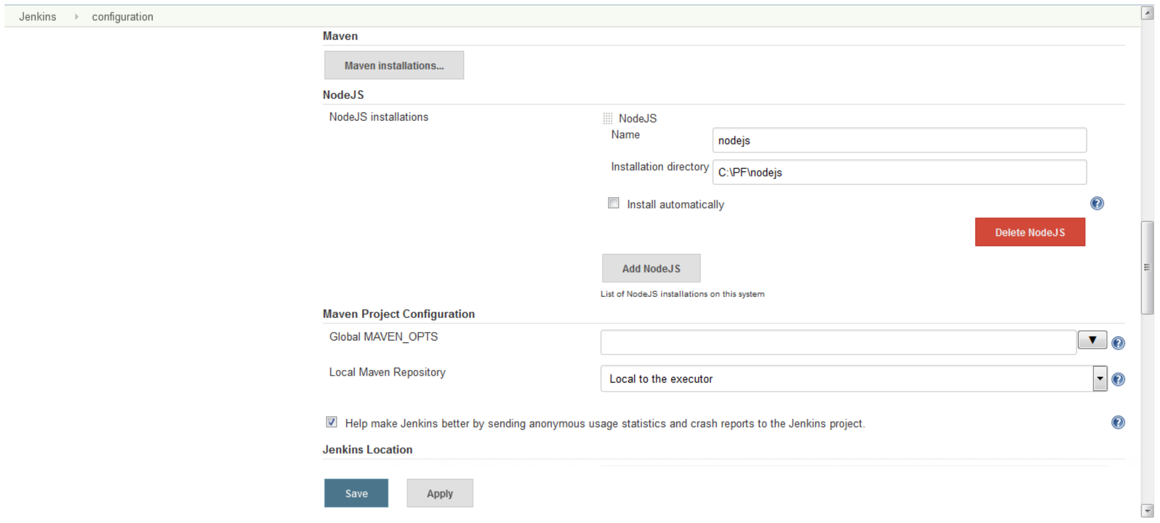
Install ↓	Name	Version
☐	NodeJS Plugin Provides Jenkins integration for NodeJS & npm packages.	0.2.1

Update information obtained: 4 hr 52 min ago

4. After the install completes go back to the **`Manage Jenkins`** page.

5. In this step, you will configure the plugin to point to the installed NodeJS. To do this:
   A. Click the **`Configure System`** link.
   B. Scroll down to the **`NodeJS.`**
   C. Click **`Add NodeJS`**.
   D. Set the **`name`** to **`NodeJS`** as there is only one for now. Set the **`installation directory`** to the location used during setup.

– For Windows: **C:\PF\nodejs**
– For Mac: **/usr/local**



E. Save the configuration changes.

6. Now, we want to add a step to display the versions in use so that we know NodeJS is working in Jenkins. To do this, create a new job named **NodeJSJob** and set it to a **Freestyle** job.

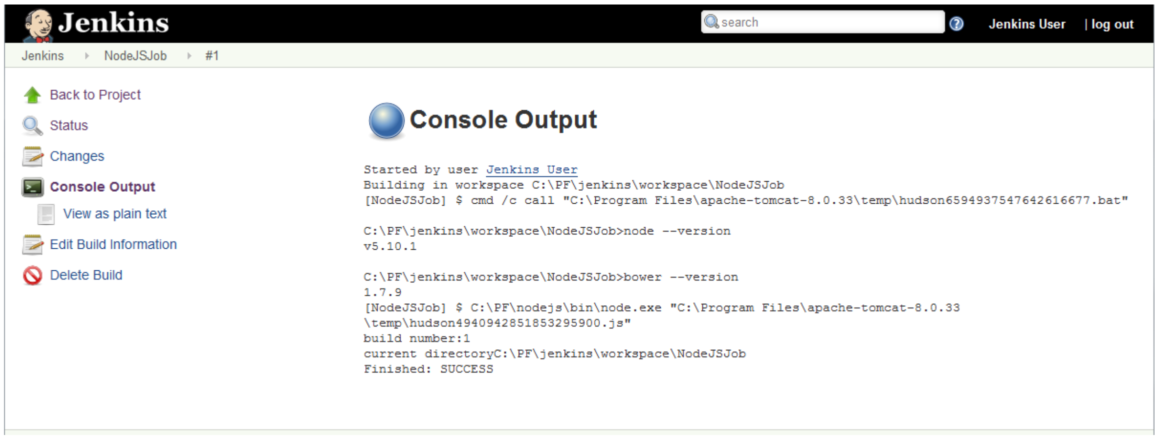
   Then, add a build step for **Execute Windows batch command** or a **Mac execute shell** and add the following lines:

```
node --version
bower --version
grunt --version
```

7. Add another build step. Select **NodeJS script** and add the following lines:

```
var console=require('console')
console.log('build number:' + process.env['BUILD_NUMBER']);
console.log('current directory' + process.cwd());
```

8. Save the changes and test the job. You should get the **Finished: Success** preceded by a build number and the current working directory path. This confirmed that the NodeJS plugin is working in Jenkins.



Console Output

```
Started by user Jenkins User
Building in workspace C:\PF\jenkins\workspace\NodeJSJob
[NodeJSJob] $ cmd /c call "C:\Program Files\apache-tomcat-8.0.33\temp\hudson6594937547642616677.bat"

C:\PF\jenkins\workspace\NodeJSJob>node --version
v5.10.1

C:\PF\jenkins\workspace\NodeJSJob>bower --version
1.7.9
[NodeJSJob] $ C:\PF\nodejs\bin\node.exe "C:\Program Files\apache-tomcat-8.0.33
\temp\hudson4940942851853295900.js"
build number:1
current directoryC:\PF\jenkins\workspace\NodeJSJob
Finished: SUCCESS
```

9: Create a Jenkins Plugin

Overview

In this lab, you will extend Jenkins by writing a Jenkins plugin. Jenkins plugins are like Ant and Maven plugins. They are implementation classes that attach to extension points in the Jenkins environment. They have access to contexts of Jenkins and the job currently running as well as other plugins running within the job.

To accomplish this, you will perform these tasks:

- Configure and install Maven and Jenkins connection to repositories
- Create a Jenkins plugin Maven project
- Build the Jenkins plugin Maven project
- Test/use the new Jenkins plugin
    - Copy the Jenkins plugin to enable it in Jenkins
    - Restart Tomcat to enable reloading configurations
    - Create a new job and set the property on the build
    - Configure the builder global setting

1. Add the Jenkins **profile** to your **`settings.xml`** file, typically in the Apache Maven **conf** folder or in the **`user folder\.m2\.`** There is a snippets file in the provided setup folder with the profile below, if you wish to copy and paste.

```
    <profile>
      <id>jenkins</id>
      <activation>
        <activeByDefault>true</activeByDefault> <!-- change this to false, if
you don't like to have it on per default -->
      </activation>
      <repositories>
        <repository>
          <id>repo.jenkins-ci.org</id>
          <url>http://repo.jenkins-ci.org/public/</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>repo.jenkins-ci.org</id>
          <url>http://repo.jenkins-ci.org/public/</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
```

2. Add the following **mirror** to the `settings.xml` file (also in the snippets file). Note: There is usually a commented out section in the XML file where the mirror goes.

```
<mirror>
    <id>repo.jenkins-ci.org</id>
    <url>http://repo.jenkins-ci.org/public/</url>
    <mirrorOf>m.g.o-public</mirrorOf>
</mirror>
```

3. To develop the Jenkins plugin, we will create a plugin project using the Maven archetype generator. In the workspace location of Eclipse, likely user folder\workspace, execute the following command in a command window:

```
mvn -U org.jenkins-ci.tools:maven-hpi-plugin:create
```

   Enter the values below when prompted:

```
groupId: MyPlugin
artifactId: MyArtifactId
```

4. The plugin project will be created and it will take quite a while to download all the jars needed for the generation process. Once that finishes, run the following command to build the plugin **.hpi** file that will be copied into the plugins folder of Jenkins.

   Change to the **`MyArtifactId`** folder and run:

```
mvn install
```

5. This project can be imported as a Maven project into Eclipse if you would like to do so. Eclipse is not the best tool for this though. The **m2e** has several issues with these projects.

6. This will also take quite some time to build. Once it completes there will be a **`.hpi`** file in the target folder of the project. Copy the **`.hpi`** file to the **`Jenkins\plugins`** folder.
   - For Windows: **`C:\PF\Jenkins\plugins`**
   - For Mac as regular user: **`~/Jenkins/plugins`**
   - For Mac as super user: **`/var/root/.jenkins/plugins`**

7. Test that Jenkins is running by opening a web browser to **`http://yourHostName:8080/jenkins`**.

   Note: If you had Jenkins running before you began this exercise, you need to restart Jenkins here.

8. To customize the Jenkins environment, click the **`Manage Jenkins`** link. Click **`Configure System`**. Scroll down to the **`Hello World Builder`**. You can change the language to French with the checkbox.

9. Create a **New Item** named **`hello-example`** as a **`Freestyle`** project. Click **`OK`**.

10. Scroll down to the **`build steps`**. You can **`add a build step`** of **`Say hello world`**. Set the **`name`** to your name. **`Save`** and click **`Build Now`**.

11. When the build completes click the **`#1`** link. Click the **`Console Output`** link. The console will have a line **Hello, your name!**

12. To customize the Jenkins environment, click the **`Manage Jenkins`** link. Click **`Configure System`**. Scroll down to the **`Hello World Builder`**. You can change the language to French with the checkbox. Then, rerun the build. The console will now show **Bonjour, your name!**

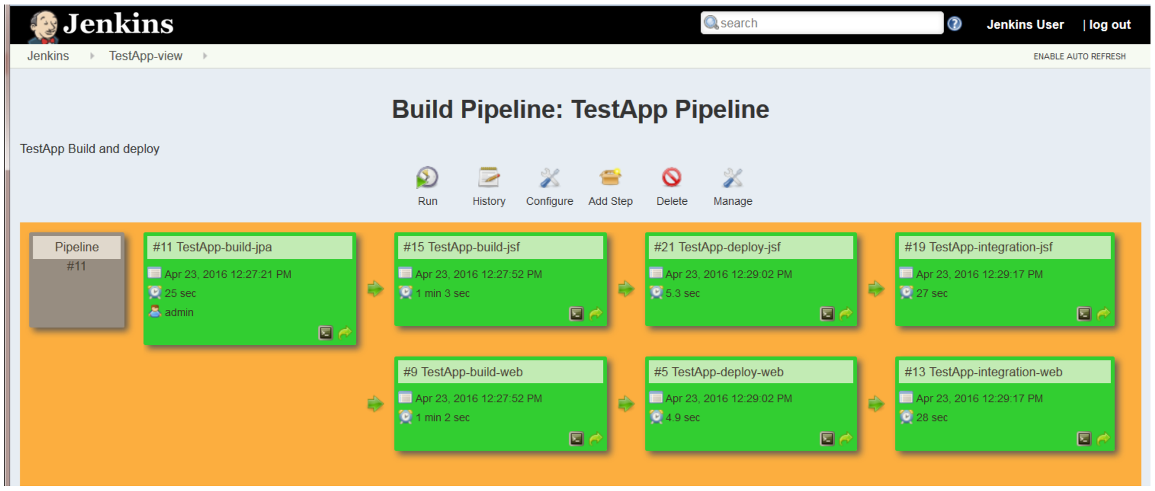
13. For those who feel truly adventurous, as a challenge, add a property to the **HelloWorldBuilder** of country as a String. **Hints:** provide getters and setters, change the constructor, add a validation method for the country, then redeploy.

10: Build a Jenkins Pipeline

Overview

In this lab, you will install the **Jenkins Pipeline Plugin** and configure a Pipeline. Pipelines are a way of viewing projects that shows the dependencies in a graphical context.

TestApp-build-jpa is a utility jar that is used by the web applications. Successfully building **TestApp-build-jpa** will trigger **TestApp-jsf** and **TestApp-web**. If the unit tests and other checks pass, then **TestApp-deploy-jsf** and **TestApp-deploy-web** will be triggered. Their completion will trigger **TestApp-integration-jsf** and **TestApp-integration-web** to run the integration tests on the deployed application. If all goes well, every job will be green.



The general steps to accomplish this are:

- Create a build pipeline view
- Create a job for a shared library
- Create jobs to build two web applications
- Create jobs to deploy the applications to Tomcat
- Create jobs to perform integration tests on both apps
- Add to the build pipeline view
- Run the initial job and watch the pipeline progress

1. Test that Jenkins is running by opening a web browser to `http://yourHostName:8080/jenkins`.

2. Click the `Manage Plugins` link. Click the **Available** tab and search for `Pipeline`.

3. Select the **`Pipeline`** plugin. Also select the **`Build Pipeline`** plugin. Search for **`Copy Artifact Plugin`** and select it as well. Click the **`download now and install after restart`** button.

4. After some time the download of the plugins will finish. Stop and restart Tomcat which will reload Jenkins. Now, it is possible to define a Pipeline job.

5. Return to the main Jenkins page and add five New Items. You can see these items in the pipeline graphic in the introduction. These are **Maven projects** because they will run a build from the **pom.xml** so we won't have to add that as a build step. We will need to specify the goal and will do so in a later step.
   - Select **`Maven project`**. Click **`OK`** → **`Back to Dashboard`**.
   - Use these names:
     - **`TestApp-build-jpa`**
     - **`TestApp-build-jsf`**,
     - **`TestApp-build-web`**
     - **`TestApp-integration-jsf`**
     - **`TestApp-integration-web`**

6. Return to the main Jenkins Page and add two more New Items. These are configured as **Freestyle** because we don't need to do a Maven build step. This is a simple copy step.
   - Select **`Freestyle project`**
   - Use these names:
     - **`TestApp-deploy-jsf`**
     - **`TestApp-deploy-web`**.

7. Return again to the main Jenkins page and select **`TestApp-build-jpa`** and select **`Configure`**. This is where we will get the build started and build our first artifact.
   A. Under **`Source Code Management`** select **`Git.`**
      - Set **`Repository URL`** to:
        - For Windows: **`C:\Setup\git\course-jpa`**
        - For Mac: **`~/setup/git/course-jpa`**

   B. Uncheck **`Build whenever a SNAPSHOT dependency is built`**. If this is left checked then the pipeline will attempt to build it twice and it will show up as two objects in the pipeline view.

   C. The builds will be manual so don't define a schedule.

   D. Under **`Pre Steps`** click **`Add pre-build step.`**
      1. Select **`Invoke top-level Maven targets`**
         - Set **`Maven Version`** to **`Maven 3.3.9`**
         - Set **`Goals`** to **`clean install`**

E. Under **Post Steps** click **Add post-build step.**
   - Select **Trigger/call builds other project**.
     - Enter **TestApp-build-jsf, TestApp-build-web**

F. Save the changes.

8. Configure **TestApp-build-jsf**. This will build our second artifact, the web application using JSF technologies.

A. Under **Source Code Management** select **Git.**
   - Set **Repository URL** to:
     - For Windows: **C:\Setup\git\course-jsf**
     - For Mac: **~/setup/git/course-jsf**

B. Uncheck **Build whenever a SNAPSHOT dependency is built**.

C. Under **Pre Steps** click **Add pre-build step.**
   1. Select **Invoke top-level Maven targets.**
      - Set **Maven Version** to **Maven 3.3.9**
      - Set **Goals** to **clean install**

D. Under **Post Steps** click **Add post-build step.**
   1. Select **Trigger/call builds other project**.
      - Enter **TestApp-deploy-jsf**

E. Save the changes.

9. Configure **TestApp-build-web.** This is where we build our web project that uses the artifact built in the first project.

A. Under **Source Code Management** select **Git.**
   - Set **Repository URL** to:
     - For Windows: **C:\Setup\git\course-web**
     - For Mac: **~/setup/git/course-web**

B. Uncheck **Build whenever a SNAPSHOT dependency is built**.

C. Under **Pre Steps** click **Add pre-build step.**
   1. Select **Invoke top-level Maven targets**
      - Set **Maven Version** to **Maven 3.3.9**
      - Set **Goals** to **clean install**

D. Under **Post Steps** click **Add post-build step.**
   1. Select **Trigger/call builds other project**.
      - Enter **TestApp-deploy-web**

E. Save the changes.

10. Configure **TestApp-integration-jsf**.
    A. Under **Source Code Management** select **Git.**
        – Set **Repository URL** to:
            – For Windows: **C:\Setup\git\course-jsf**
            – For Mac: **~/setup/git/course-jsf**

    B. Uncheck **Build whenever a SNAPSHOT dependency is built**.

    C. Under **Pre Steps** click **Add pre-build step.**
        1. Select **Invoke top-level Maven targets**
            – Set **Maven Version** to **Maven 3.3.9**
            – Set **Goals** to: **compiler:testCompile failsafe:integration-test**

    Note: The reason these goals are used is that there is no need to build and install a new version of the project since that was done in previous steps. All this step is going to do is execute the integration tests for the project which require the application already deployed into a running application server. To run the test, the test classes need to be available at runtime.

    D. No **Post Steps** need to be defined for this project.

    E. Save the changes.

11. Configure **TestApp-integration-web**.
    A. Under **Source Code Management** select **Git.**
        – Set **Repository URL** to:
            – For Windows: **C:\Setup\git\course-web**
            – For Mac: **~/setup/git/course-web**

    B. Uncheck **Build whenever a SNAPSHOT dependency is built**.

    C. Under **Pre Steps** click **Add pre-build step.**
        1. Select **Invoke top-level Maven targets**
            – Set **Maven Version** to **Maven 3.3.9**
            – Set **Goals** to: **compiler:testCompile failsafe:integration-test**

    D. No **Post Steps** need to be defined for this project.

    E. Save the changes.

12. Configure **TestApp-deploy-jsf**.
    A. Under **Build**, click **Add build step** to **Copy artifacts from another project**.
       – Set the project name to **TestApp-build-jsf**.

    B. Click **Add build step** again.
       – For Windows: select **Execute Windows batch command**
       ```
       copy %WORKSPACE%\com.example\course-jsf\0.0.1-SNAPSHOT\course-
       jsf-0.0.1-SNAPSHOT.war "C:\Program Files\Apache Software
       Foundation\Tomcat 8.0\webapps\course-jsf.war"
       ```

       – For Mac: select **Execute shell**
       ```
       cp ${WORKSPACE}/com.example/course-jsf/0.0.1-SNAPSHOT/course-
       jsf-0.0.1-SNAPSHOT.war /usr/local/tomcat/webapps/course-
       jsf.war
       ```

    This will copy the artifact just copied in step A to the running instance of Tomcat to deploy it.

    C. Click **Add build step** again.
       1. Select **Trigger/call builds on other projects**
       2. Set **Projects to build** to: **TestApp-integration-jsf**

13. Configure **TestApp-deploy-web**.
    A. Under **Build**, click **Add build step** to **Copy artifacts from another project**.
       – Set the project name to **TestApp-build-web**.

    B. Click **Add build step** again.
       – For Windows: select **Execute Windows batch command**
       ```
       copy %WORKSPACE%\com.example\course-web\0.0.1-SNAPSHOT\course-
       web-0.0.1-SNAPSHOT.war "C:\Program Files\Apache Software
       Foundation\Tomcat 8.0\webapps\course-web.war"
       ```

       – For Mac: select **Execute shell**
       ```
       cp ${WORKSPACE}/com.example/course-web/0.0.1-SNAPSHOT/course-
       web-0.0.1-SNAPSHOT.war /usr/local/tomcat/webapps/course-
       web.war
       ```

    C. Click **Add build step** again.
       1. Select **Trigger/call builds on other projects**
       2. Set **Projects to build** to: **TestApp-integration-web**

14. Above the list of projects click the **+** sign next to the **`All`** tab.
    A. Enter **`TestApp-Pipeline`** for the name and select the **`Build Pipeline View`** radio button.
    B. Click **`OK`** to create the pipeline view.
    C. Click the **`Configure`** button and select **`TestApp-build-jpa`** for the **`Select Initial Job`** dropdown.
    D. Click **`Save`** and the screen will refresh to show the job hierarchy shown on the first page of this exercise.
    E. Compare the jobs to the picture at the beginning of the lab. Everything should be blue initially as none of the jobs have run yet.

15. Click the **`Run`** button. As each job is executed, they will turn from blue to yellow to green. Yellow indicates the job is in progress. If the job turns red, then there was an error. Fix the error and rerun the pipeline.

16. Eventually all the jobs will turn green as the build cascades.

Note: The first time you run this, two build jobs will stay yellow. When you look at the console output it is because the build is Unstable. This happens because the build jobs test a deploy that is being done by the deploy jobs which at that point have not run yet. When you run it again, all the files are in place and it will run successfully.