

WA2753 Custom TD DevOps bootcamp

Student Labs

Web Age Solutions Inc.

Table of Contents

Lab 1 - Configuration Management.....	3
Lab 2 - Version Control - Git.....	13
Lab 3 - Configure Tools in Jenkins.....	19
Lab 4 - Create a Jenkins Job.....	23
Lab 5 - Create a Pipeline.....	40
Lab 6 - Install Prerequisites.....	51
Lab 7 - Continuous Code Quality - SonarQube.....	58
Lab 8 - Continuous Monitoring - Nagios.....	62
Lab 9 - Getting Started with Docker.....	71
Lab 10 - Getting Started with Kubernetes.....	82
Lab 11 - Getting Started with OpenShift.....	91
Lab 12 - CI/CD with Jenkins, Docker, and OpenShift.....	99

Lab 1 - Configuration Management

Note: This document can be found on the Desktop of your computer to allow you copy and paste the code during the exercises.

In this chapter you will explore Chef basics. You will create recipes that utilizes various Chef resources.

Part 1 - Launch terminal

In this Lab you will work with the WA2543 VM Image. Check with your instructor if you cannot find this VM.

In this part you will launch the Linux terminal.

__1. Open the **Terminal** window.



Alternatively, you can press Ctrl+Alt+T to access the terminal

or

In the task bar, click Search button. In search text box, type in terminal and hit enter key on the keyboard.



__2. Run following command to switch to **Documents** directory.

```
cd ~/Documents
```

Part 2 - Copy lab scripts

In this part you will copy lab scripts to Documents directory.

__1. Copy scripts.

```
cp -r ~/Downloads/labs ~/Documents/
```

__2. Switch to **chef** directory.

```
cd ~/Documents/labs/chef
```

Part 3 - Install Chef server core, Chef manage, and Chef Development Kit

In this part you will install Chef server, Chef manage, and Chef development kit.

Note: If prompted to enter password during the labs, enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

__1. View chef_install.sh file contents.

```
cat chef_install.sh | more
```

__2. Run the install script.

```
bash chef_install.sh
```

Note: Installation will take a few minutes complete.

Notice it executes a script and shows various messages, such as, action create, action run, action delete, action create_if_missing, and action restart. Actions are part of a cookbook recipe and will be covered later in the course.

Syntax for creating Chef admin user is as follows:

```
chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME  
EMAIL_ADDRESS PASSWORD --filename PRIVATE_KEY
```

Each chef server must belong to an organization. For complete docs on all the chef server control commands and syntax reference https://docs.chef.io/ctl_chef_server.html

Part 4 - Verify Chef server installation

In this part you will verify Chef server installation.

__1. Run following command to view the chef configuration. This dumps out the chef JSON configuration object.

```
sudo chef-server-ctl show-config
```

__2. Run following command to view the services running for chef. This dumps out the chef JSON configuration object.

```
sudo chef-server-ctl service-list
```

__3. Run following command to view status of each running chef service.

```
sudo chef-server-ctl status
```

Note: The status command can also be invoked with a specific service, e.g. chef-server-ctl status nginx.

__4. Run following command to view the service logs for all running chef services.

```
sudo chef-server-ctl tail
```

__ 5. Hit Ctrl + Z to exit.

__ 6. Similar to the service-list command you can view only the last entries for a specific service log.

```
sudo chef-server-ctl tail erchef
```

Note: The output may not show the result if there are no error messages in the log file. This is the expected message:

```
wasadmin@ubuntu:~/Downloads$ sudo chef-server-ctl tail erchef
find: `/var/log/opscode/erchef': No such file or directory
tail: cannot follow '-' by name
```

The chef server control has a built in process supervisor that ensures all of the required services are in the appropriate state at any given time. The supervisor starts two processes per service and provides the following sub-commands for managing services: hup, int, kill, once, restart, service-list, start, status, stop, tail, and term.

__ 7. Run following command to stop the embedded nginx service.

```
sudo chef-server-ctl stop nginx
```

__ 8. Run following command to verify the embedded nginx service is stopped.

```
netstat -an | grep 0.0.0.0:443
```

__ 9. Run following command to start the embedded nginx service.

```
sudo chef-server-ctl restart nginx
```

Part 5 - Verify Chef Manage installation

In this part you will verify chef manage installation. Chef manage is used for various reasons, such as, viewing nodes, editing run list, viewing reports and managing policies, and administration.

__ 1. To verify chef manage is installed successfully launch Firefox from the Linux task bar.

__ 2. In the URL bar enter:

```
https://localhost
```

Note: The SSL certificate will not be recognized and we will have to provide a confirmation of the security exception

__ 3. Click **Advanced** button.

__ 4. Click **Add Exception** button.

__ 5. Click **Get Certificate** button.

__ 6. Click **Confirm Security Exception** button.

__ 7. In user name enter "wasadmin" and in password enter "wasadmin".

Note: enter credentials without quotes. If you are running this command in your organization environment then enter the user/password that your instructor provided to you.

__ 8. Click close (x) if it prompts you to remember credentials.

Note: There are currently no nodes connected to our chef server. You will add a node later in the lab.

__ 9. Keep the browser open and switch back to the terminal.

Part 6 - Verify Chef Development Kit installation

In this part you will verify Chef development kit installation.

__ 1. Run following command to verify chef development kit is installed.

```
chef verify
```

Notice it shows status of various chef components

__ 2. Run following command to get version of various chef development kit components.

```
chef --version
```

__ 3. Run following command to output the chef server version and verify the "knife" tool is available.

```
knife --version
```

Part 7 - Create a simple standalone chef recipe

In this part you will create a standalone recipe that downloads and installs cowsay, fortune, and tree packages. It also creates a hello.txt file.

__ 1. Open a new **Terminal** window.

__ 2. Run following commands and notice they all display error message that corresponding package is not installed.

```
cowsay  
fortune  
tree
```

__ 3. Run "cat /tmp/hello.txt" and verify it displays a message saying the file doesn't exist.

__ 4. Run following command to switch to **Documents** directory.

```
cd ~/Documents
```

__5. Run following command to create a directory named **recipes**.

```
mkdir recipes
```

__6. Run following command to switch to **recipes** directory.

```
cd recipes
```

__7. Run following command to launch nano text editor.

```
sudo nano setup.rb
```

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

__8. Type following text.

```
package "cowsay" do
  action :install
end

package 'fortune'

package 'tree'

file '/tmp/hello.txt' do
  content 'Hello World!'
end
```

__9. Press Ctrl+O to save the file and press Enter (do this every time you need to save a file using nano).

__10. Press Ctrl+X to exit to the terminal.

__11. Run the following command to cook the recipe in local mode.

```
sudo chef-client -z /home/wasadmin/Documents/recipes/setup.rb
```

Alternatively, you can use `sudo chef-client --local-mode setup.rb`

Notice it downloads cowsay, fortune, and tree packages and installs it. It also creates hello.txt file.

If the package installation fails then run following command:

```
sudo apt-get update
```

__12. Run following shell command to verify cowsay is installed.

```
cowsay 'Hello World'
```

The following should be displayed



Note. If you get a path error then include /usr/games/ before the command in the following steps, for example:

```
/usr/games/cowsay 'Hello World'
```

__13. Run following command to verify "fortune" package is installed.

```
fortune
```

Notice it displays a random message each time you run it.

__14. Run following command to utilize fortune and cowsay together.

```
fortune | cowsay
```

__15. Run following command to verify "tree" package is installed.

```
tree /home
```

Notice it displays directory tree.

__16. Run following command to verify hello.txt file exists.

```
cat /tmp/hello.txt
```

It shows display Hello World!

Notice chef created /tmp/hello.txt file. Chef recipes can be used for creating configuration files, such as, app.config, web.config, settings.xml, and other similar files.

__17. Run following command to create launch nano text editor.

```
sudo nano remove.rb
```

__18. Type following code.


```
package 'fortune' do
  action :remove
end

package 'cowsay' do
  action :remove
end

file '/tmp/hello.txt' do
  action :delete
end
```

Notice for packages you use remove action and for file you use delete action. You are purposefully not removing the "tree" package since it will be used later in the lab

__19. Press Ctrl+O to save the file and press Enter.

__20. Press Ctrl+X to exit to the terminal.

__21. Run following command to run the recipe in local mode.

```
sudo chef-client -z remove.rb
```

__22. Run following commands and notice packages do not exist.

```
fortune
cowsay
```

__23. Run "cat /tmp/hello.txt" and verify the file no longer exists.

Part 8 - Create a chef cookbook and organize recipes

In this part you will create a cookbook and organize recipes.

__1. Launch a new Linux terminal or keep in the same Terminal.

__2. Switch to Documents directory.

```
cd ~/Documents
```

__3. Create a directory for storing cookbooks.

```
mkdir cookbooks
```

__4. Review what **chef** tool can do.

```
sudo chef --help
sudo chef generate --help
```

__5. Generate a cookbook.

```
sudo chef generate cookbook cookbooks/my_cookbook
```

__6. Switch to cookbooks directory.

```
cd cookbooks
```

__7. Get directory list.

```
ls
```

Notice there's my_cookbook directory

__8. Switch to my_cookbook directory.

```
cd my_cookbook
```

__9. Get directory list.

```
ls
```

__10. Notice there are various files and folders.

__11. View Berksfile.

```
cat Berksfile
```

Notice it contains source which points to <https://supermarket.chef.io> by default. That's where all the packages are downloaded by default.

__12. View cheffignore.

```
cat cheffignore
```

Notice it contains various files and directories. These are ignored by chef client. You can add more entries to the file, if required.

__13. View metadata.rb.

```
cat metadata.rb
```

Notice it contains author name, cookbook name, description, license, and version.

__14. View README.md.

```
cat README.md
```

Notice you can add description of your cookbook here.

__15. Switch to recipes directory.

```
cd recipes
```

__16. Get directory list.

```
ls
```

__17. Notice there's default.rb. You can edit this file and define custom recipe, if required.

__18. View default.rb.

```
cat default.rb
```

Notice it contains comments by default.

__19. Switch to recipes directory.

```
cd ~/Documents/cookbooks/my_cookbook/recipes
```

__20. Copy previously created standalone recipes to the recipes directory.

```
sudo cp ~/Documents/recipes/*.rb  
~/Documents/cookbooks/my_cookbook/recipes
```

__21. Get directory list.

```
ls
```

Notice there are five ruby files: default.rb, remove.rb, fortune.rb, cowsay.rb, setup.rb, and tree.rb

__22. Switch to Documents directory.

```
cd ~/Documents
```

__23. Run setup.rb cookbook recipe.

```
sudo chef-client -z -r "recipe[my_cookbook::setup]"
```

Notice it installs fortune, cowsay, and tree packages. It also creates hello.txt file.

You can also use --recipe instead of -r

__24. Run tree command.

```
tree cookbooks/my_cookbook
```

Notice it displays my_cookbook's directory tree.

__25. Run remove.rb cookbook recipe.

```
sudo chef-client -z -r "recipe[my_cookbook::remove]"
```

Notice it removes fortune, cowsay, and tree packages. It also deletes hello.txt file.

__26. Run default.rb cookbook recipe.

```
sudo chef-client -z -r "recipe[my_cookbook]"
```

Notice it doesn't do anything since it's an empty recipe.

__27. Close all Terminal windows.

Part 9 - Review

In this lab you installed and configured chef server, chef development kit, and chef manage web user interface. You also created simple recipes to utilize various resources, such as, package and file.

Lab 2 - Version Control - Git

In this chapter you will install, configure, and use Git.

Part 1 - Launch terminal

In this Lab you will work with the WA2543 VM Image. Check with your instructor if you cannot find this VM.

In this part you will launch the Linux terminal.

__1. Open the **Terminal** window.

Alternatively, you can press Ctrl+Alt+T to access the terminal.

__2. Run the following command:

```
sudo apt-get update
```

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

__3. Run following command to switch to **Downloads** directory.

```
cd ~/Downloads
```

Part 2 - Install Git by creating a chef cookbook

In this part you will remove and install Subversion and git by creating a chef cookbook.

__1. Switch to cookbooks directory.

```
cd /home/wasadmin/Documents/cookbooks
```

__2. Create a cookbook.

```
sudo chef generate cookbook git_cookbook
```

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

__3. Edit default.rb recipe.

```
sudo nano git_cookbook/recipes/default.rb
```

__4. Append following text.

```
package "git" do
  action :install
end
```

__5. Press Ctrl+O to save the file and hit enter.

__6. Press Ctrl+X to exit to the terminal.

__7. Run the recipe.

```
sudo chef-client -z -r "recipe[git_cookbook]"
```

Part 3 - Verify Git is installed

In this part you will see how to verify Git installation.

__1. In the terminal run following command to find git version number.

```
git --version
git help -g
git help -a
```

Part 4 - Using Git

In this part you will use Git to perform various operations, such as, check in, check status etc.

__1. Switch to the "Documents" directory.

```
cd ~/Documents
```

__2. Create a directory.

```
mkdir -p workspace/git
```

__3. Switch to the "git" directory.

```
cd workspace/git
```

__4. Initialize repository.

```
git init
```

__ 5. Tell Git who you are.

```
git config --global user.name "Alice Smith"
git config --global user.email alice@smith.com
```

Note: One interesting aspect of Git is that it separates user identity in the repository from any sort of authentication or authorization. Because a distributed repository will generally be maintained by many separate individuals or systems, the identity of the committer must be contained in the repository – it can't just be supplied as a user id when we do the commit. So, even if we're not connected to any central repository, we need to tell Git who we are. The identity that we supply will be recorded whenever we commit to a repository.

__ 6. Create a text file.

```
sudo nano sample.txt
```

__ 7. Enter following text.

```
First Version!
```

__ 8. Press Ctrl+O to save the file and hit enter.

__ 9. Press Ctrl+X to exit to the terminal.

__ 10. Get Git status.

```
git status
```

Notice sample.txt is listed under untracked files.

__ 11. Add the files to tracked.

```
git add .
```

Note: Here you are adding the current directory. You could also add the file using "git sample.txt".

__12. Get Git status again.

```
git status
```

Notice sample.txt is tracked.

__13. Commit changes.

```
sudo git commit
```

Notice it launched text editor automatically which lists operations that will get performed when files are committed. Here you can add detailed description that will get saved when you commit the changes.

__14. Add following text in the first line.

```
Added sample.txt
```

__15. Press Ctrl+O to save the file and hit enter.

__16. Press Ctrl+X to exit to the terminal.

__17. Get Git status.

```
git status
```

Notice it says there's nothing to commit since you have already committed all changes.

__18. Modify sample.txt.

```
sudo nano sample.txt
```

__19. Change "First Version!" to "Second Version!"

__20. Press Ctrl+O to save the file and hit enter.

__21. Press Ctrl+X to exit to the terminal.

__22. Get Git status.

```
git status
```

Notice it says the file is modified.

__23. View changes.

```
git diff
```

Notice it shows old text in red and new text in green.

__24. Create another file.

```
sudo nano another.txt
```

__25. Add the following text.

```
Hello World!
```

__26. Press Ctrl+O to save the file and hit enter.

__27. Press Ctrl+X to exit to the terminal.

__28. Add all files.

```
git add .
```

__29. Get Git status.

```
git status
```

Notice it's showing 1 file as modified and 1 file as a newly added file.

__30. Commit changes.

```
sudo git commit -m "Made 2 changes"
```

Notice when you pass -m switch, you can store a simple single line comment.

__31. Get Git status.

```
git status
```

Notice there's nothing to commit.

__32. Delete sample.txt

```
sudo rm sample.txt
```

__33. Recover file.

```
git checkout sample.txt
```

__34. View sample.txt

```
cat sample.txt
```

Note: It restored latest version by default.

__35. Delete file again.

```
sudo rm sample.txt
```

__36. View all versions of a file.

```
git log
```

Notice it shows user, commit id, date time, and comment.

__37. Copy the **commit id** for the first version.

__38. View changes between current and the first version.

```
git diff <commit_id>
```

__39. Restore the older version.

```
git checkout <commit_id> sample.txt
```

__40. View file content.

```
cat sample.txt
```

Notice it's the first version.

__41. Close the terminal.

Part 5 - Review

In this chapter you installed and used Git.

Lab 3 - Configure Tools in Jenkins

In this lab you will verify that Jenkins Continuous Integration is already installed and you will configure it.

At the end of this lab you will be able to:

1. Verify Jenkins is running
2. Configure tools in Jenkins

Part 1 - Configure Jenkins

After the Jenkins installation, you can configure few other settings to complete the installation before creating jobs.

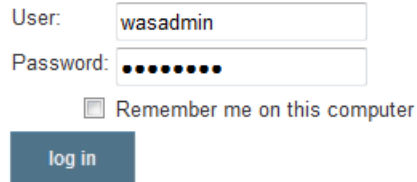
You will be setting JDK HOME and Maven Installation directory.

In this Lab you will work with the WA2271 VM Image. Check with your instructor if you cannot find this VM.

__1. To connect to Jenkins, open Firefox and enter the following URL.

`http://localhost:8080/`

__2. Enter **wasadmin** as user and password and click **Log in**.

A screenshot of the Jenkins login page. It features a 'User:' label next to a text input field containing 'wasadmin'. Below it is a 'Password:' label next to a password input field with masked characters. To the right of the password field is a checkbox labeled 'Remember me on this computer'. At the bottom is a blue 'log in' button.

User:

Password:

☐ Remember me on this computer

__3. Don't save the password if prompt or select Never Remember password for this site.

__4. Click on the **Manage Jenkins** link.

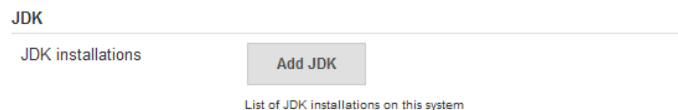


You will see some warnings.

__5. Click **Global Tool Configuration**.



__6. Scroll down and find the JDK section, Click **Add JDK**.



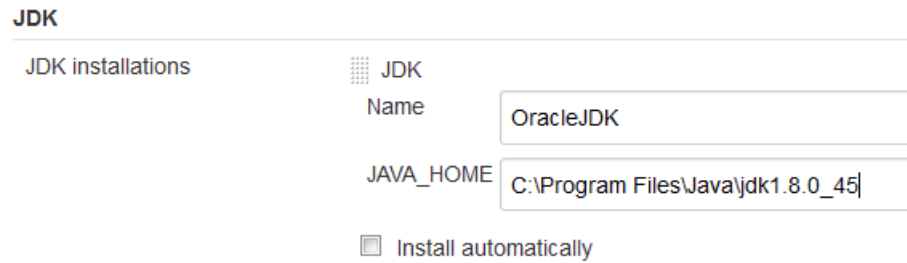
__7. Enter **OracleJDK** for JDK name.

__8. Don't check the 'Install automatically' option. Uncheck if already checked.

__9. Enter JAVA_HOME value as **C:\Program Files\Java\jdk1.8.0_45**

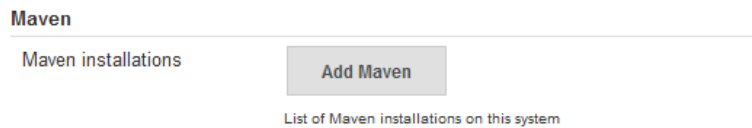
Note. You may need to use another path if Java was installed in a different folder, contact your instructor or search for the right path and use it as JAVA_HOME, it may be installed on **C:\Program Files (x86)\Java\jdk1.8.0_45**

__10. Verify your settings look as below (Java path maybe different):



JDK installations									
<table><tr><td>JDK</td><td>Name</td><td>OracleJDK</td></tr><tr><td></td><td>JAVA_HOME</td><td>C:\Program Files\Java\jdk1.8.0_45</td></tr><tr><td></td><td><input type="checkbox"/> Install automatically</td><td></td></tr></table>	JDK	Name	OracleJDK		JAVA_HOME	C:\Program Files\Java\jdk1.8.0_45		<input type="checkbox"/> Install automatically	
JDK	Name	OracleJDK							
	JAVA_HOME	C:\Program Files\Java\jdk1.8.0_45							
	<input type="checkbox"/> Install automatically								

__11. In the Maven section, click **Add Maven**.



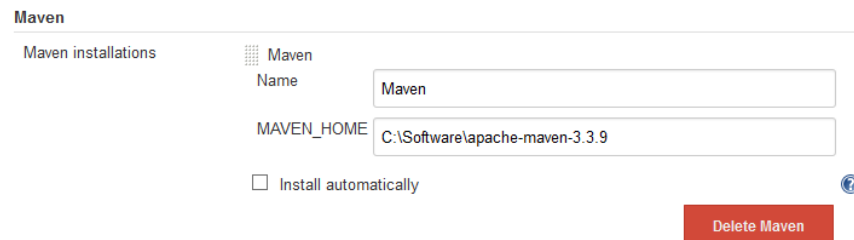
Maven				
<table><tr><td>Maven installations</td><td>Add Maven</td></tr><tr><td colspan="2">List of Maven installations on this system</td></tr></table>	Maven installations	Add Maven	List of Maven installations on this system	
Maven installations	Add Maven			
List of Maven installations on this system				

__12. Enter **Maven** for Maven name.

__13. Uncheck the 'Install automatically' option.

__14. Enter **C:\Software\apache-maven-3.3.9** for MAVEN_HOME. Make sure this folder is correct.

__15. Verify your settings look as below:

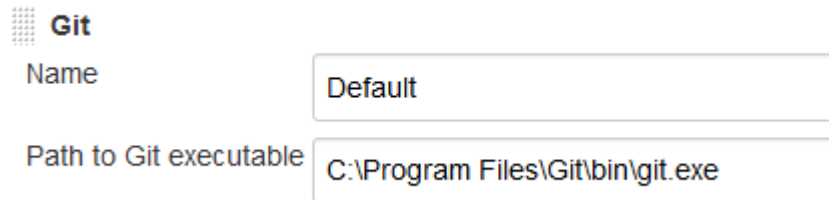


Maven												
<table><tr><td>Maven installations</td><td><table><tr><td>Maven</td><td>Name</td><td>Maven</td></tr><tr><td></td><td>MAVEN_HOME</td><td>C:\Software\apache-maven-3.3.9</td></tr><tr><td></td><td><input type="checkbox"/> Install automatically</td><td></td></tr></table></td><td><div>Delete Maven</div></td></tr></table>	Maven installations	<table><tr><td>Maven</td><td>Name</td><td>Maven</td></tr><tr><td></td><td>MAVEN_HOME</td><td>C:\Software\apache-maven-3.3.9</td></tr><tr><td></td><td><input type="checkbox"/> Install automatically</td><td></td></tr></table>	Maven	Name	Maven		MAVEN_HOME	C:\Software\apache-maven-3.3.9		<input type="checkbox"/> Install automatically		<div>Delete Maven</div>
Maven installations	<table><tr><td>Maven</td><td>Name</td><td>Maven</td></tr><tr><td></td><td>MAVEN_HOME</td><td>C:\Software\apache-maven-3.3.9</td></tr><tr><td></td><td><input type="checkbox"/> Install automatically</td><td></td></tr></table>	Maven	Name	Maven		MAVEN_HOME	C:\Software\apache-maven-3.3.9		<input type="checkbox"/> Install automatically		<div>Delete Maven</div>	
Maven	Name	Maven										
	MAVEN_HOME	C:\Software\apache-maven-3.3.9										
	<input type="checkbox"/> Install automatically											

__16. Scroll and find **Git**, you may see an error regarding the git path. Enter the following path and then hit the tab key:

```
C:\Program Files\Git\bin\git.exe
```

If this folder doesn't exist then look for Git in your computer, it maybe installed under **C:\Program Files (x86)\Git\bin\git.exe**



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

__17. Click **Save**.

Part 2 - Review

In this lab you configured the Jenkins Continuous Integration Server.

Lab 4 - Create a Jenkins Job

In this lab you will create and build a job in Jenkins.

Jenkins supports several different types of build jobs. The two most commonly-used are the freestyle builds and the Maven 2/3 builds. The freestyle projects allow you to configure just about any sort of build job, they are highly flexible and very configurable. The Maven 2/3 builds understand the Maven project structure, and can use this to let you set up Maven build jobs with less effort and a few extra features.

At the end of this lab you will be able to:

1. Create a Jenkins Job that accesses a Git repository.

Part 1 - Enable Jenkins' Maven Plugin

- __ 1. Go to the Jenkins console:

`http://localhost:8080`

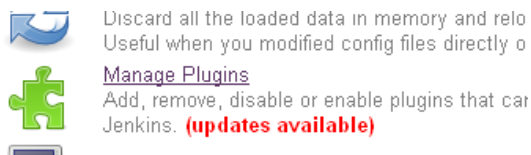
- __ 2. Login as **wasadmin** for user and password.

- __ 3. Click on the **Manage Jenkins** link



- __ 4. Ignore any message regarding an update.

- __ 5. Click on **Manage Plugins**

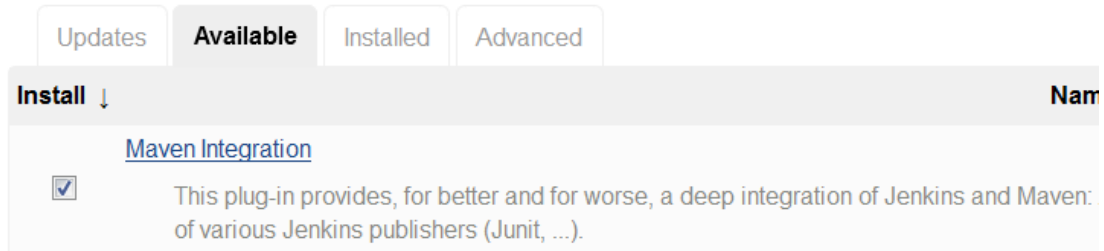


- __ 6. Click on the **Available** tab.



__7. In the **filter** box at the top-right of the window, enter '**Maven Integration**'. Note: Don't hit **Return**!

__8. Entering the filter text will narrow down the available plugins a little. Scroll down to find the '**Maven Integration**' listing, and click on the checkbox next to it.



__9. Click on **Install Without Restart**.




__10. Click on **Go back to the top page**.

Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

Javadoc Plugin  Success

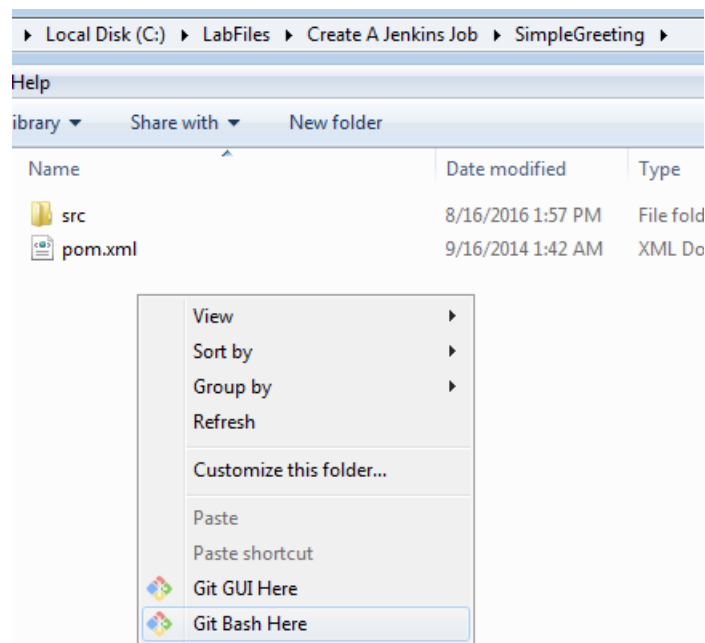
Maven Integration plugin  Success

➡ [Go back to the top page](#)
(you can start using the installed plugins right away)

Part 2 - Create a Git Repository

As a distributed version control system, Git works by moving changes between different repositories. Any repository apart from the one you're currently working in is called a "remote" repository. Git doesn't differentiate between remote repositories that reside on different machines and remote repositories on the same machine. They're all remote repositories as far as Git is concerned. In this lab, we're going to start from a source tree, create a local repository in the source tree, and then clone it to a local repository. Then we'll create a Jenkins job that pulls the source files from that remote repository. Finally, we'll make some changes to the original files, commit them and push them to the repository, showing that Jenkins automatically picks up the changes.

- ___1. Open to the folder **C:\LabFiles\Create A Jenkins Job\SimpleGreeting**.
- ___2. Right click in the empty area and select **Git Bash Here**. The Git command prompt will open.



- ___3. Enter the following command:

ls

```
wasadmin@WIN-GRFMN8J26PD /C:/LabFiles/Create A Jenkins Job/SimpleGreeting
$ ls
pom.xml  src
wasadmin@WIN-GRFMN8J26PD /C:/LabFiles/Create A Jenkins Job/SimpleGreeting
$ _
```

__4. Enter the following lines. Press enter after each line:

```
git config --global user.email "wasadmin@webagesolutions.com"
git config --global user.name "Bob Smith"
```

The lines above are actually part of the initial configuration of Git. Because of Git's distributed nature, the user's identity is included with every commit as part of the commit data. So we have to tell Git who we are before we'll be able to commit any code.

__5. Enter the following lines to actually create the Git repository:

```
git init
git add .
git commit -m "Initial Commit"
```

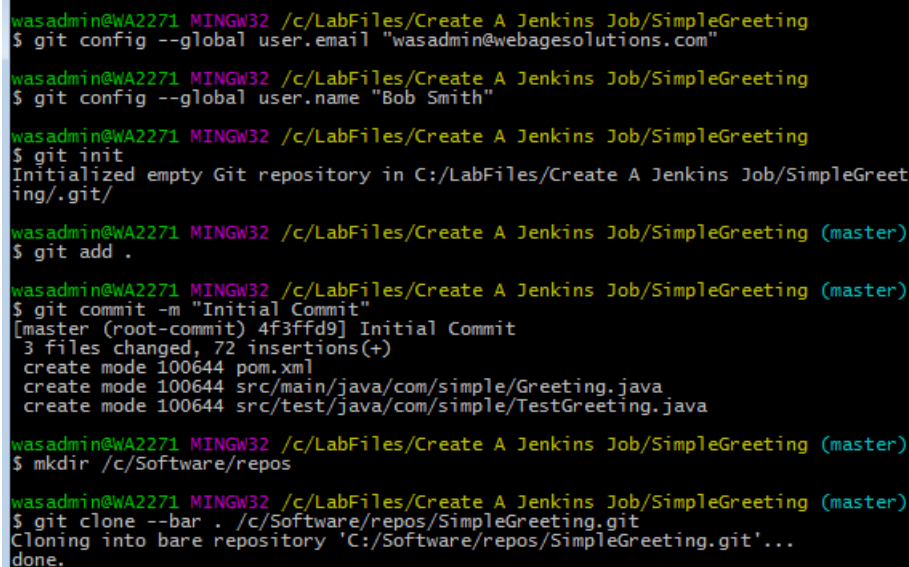
The above lines create a git repository in the current directory (which will be **C:\LabFiles\Create a Jenkins Job\SimpleGreeting**), add all the files to the current commit set (or 'index' in git parlance), then actually performs the commit.

__6. Enter the following, to create a folder called **repos** under the C:\Software folder.

```
mkdir /c/Software/repos
```

__7. Enter the following to clone the current Git repository into a new remote repository.

```
git clone --bar . /c/Software/repos/SimpleGreeting.git
```



```
wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting
$ git config --global user.email "wasadmin@webagesolutions.com"

wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting
$ git config --global user.name "Bob Smith"

wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting
$ git init
Initialized empty Git repository in C:/LabFiles/Create A Jenkins Job/SimpleGreeting/.git/

wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting (master)
$ git add .

wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting (master)
$ git commit -m "Initial Commit"
[master (root-commit) 4f3ffd9] Initial Commit
3 files changed, 72 insertions(+)
create mode 100644 pom.xml
create mode 100644 src/main/java/com/simple/Greeting.java
create mode 100644 src/test/java/com/simple/TestGreeting.java

wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting (master)
$ mkdir /c/Software/repos

wasadmin@WA2271 MINGW32 /c/LabFiles/Create A Jenkins Job/SimpleGreeting (master)
$ git clone --bar . /c/Software/repos/SimpleGreeting.git
Cloning into bare repository 'C:/Software/repos/SimpleGreeting.git'...
done.
```

At this point, we have a "remote" Git repository in the folder **C:\Software\repos\SimpleGreeting.git**. Jenkins will be quite happy to pull the source files for a job from this repo.

Part 3 - Create the Jenkins Job

- __1. Go to the Jenkins home.
- __2. Click on the **New Item** link.




- __3. Enter **SimpleGreeting** for the project name.
- __4. Select **Maven Project** as the project type.

Enter an item name


SimpleGreeting

» Required field



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system other than software build.



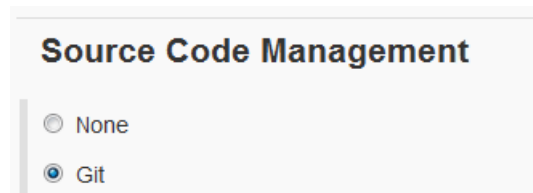
Maven project

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

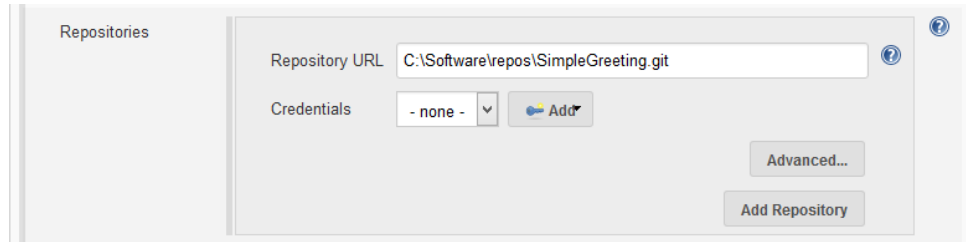
- __5. Click **OK**, to add a new job.

After the job is created, you will be on the job configuration page.

___6. Scroll down to the **Source Code Management** section and then select **Git**.



___7. Under Repositories, enter **C:\Software\repos\SimpleGreeting.git**

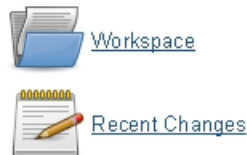


___8. Click the Tab key in your keyboard.

___9. Click **Save**.

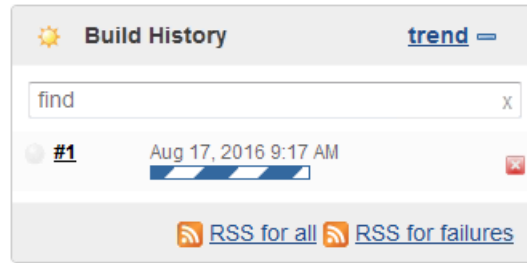
___10. You will see the Job screen.

Maven project SimpleGreeting

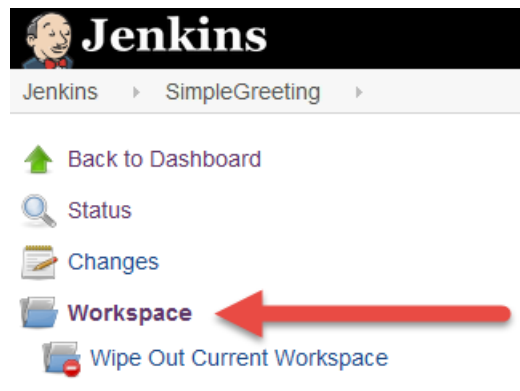


___11. Click **Build Now**.

You should see the build in progress in the **Build History** area.

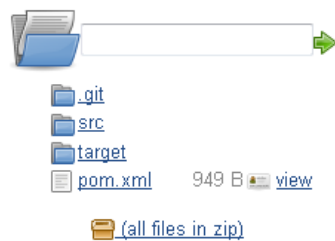


12. After a few seconds the build will complete, the progress bar will stop. Click on **Workspace**.



You will see that the directory is populated with the source code for our project.

Workspace of SimpleGreeting on master



__13. Find the **Build History** box, and click on the 'time' value for the most recent build.
You should see that the build was successful.

Build #1 (Aug 17, 2016 9:17:22 AM)



No changes.



Started by user [Administrator](#)



Revision: 4f3ffd9329c0bb74c5355effc53800df4ab2575a

- [refs/remotes/origin/master](#)

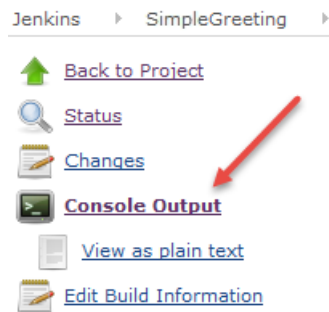


[Test Result](#) (no failures)

Module Builds

 [SimpleGreeting](#) 20 sec

__14. Click the **Console Output** from the left menu.



___15. At the end of the console you will also see the build success and successful build finish.

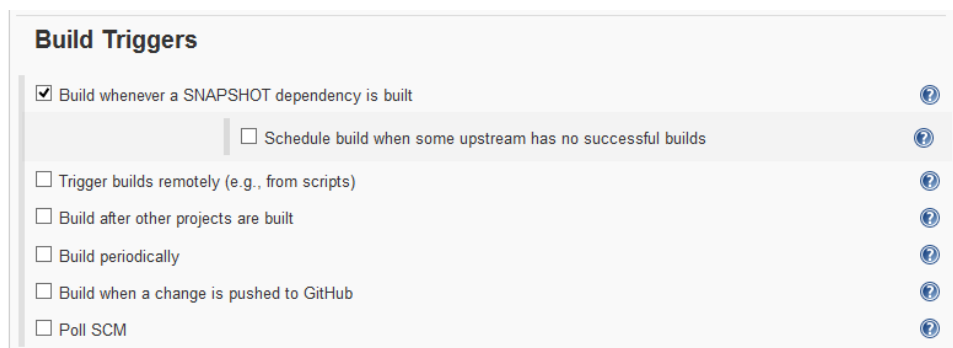
```
[INFO] Connected: http://localhost:8080/jenkins/
[INFO] Installing C:\Program Files\Jenkins\workspace\SimpleGreeting\target\S
\SimpleGreeting\1.0-SNAPSHOT\SimpleGreeting-1.0-SNAPSHOT.jar
[INFO] Installing C:\Program Files\Jenkins\workspace\SimpleGreeting\pom.xml
1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 20.554 s
[INFO] Finished at: 2016-08-17T09:17:51-04:00
[INFO] Final Memory: 12M/46M
[INFO] -----
[JENKINS] Archiving C:\Program Files\Jenkins\workspace\SimpleGreeting\pom.xml
[JENKINS] Archiving C:\Program Files\Jenkins\workspace\SimpleGreeting\target
channel stopped
Finished: SUCCESS
```

You have created a project and built it successfully.

Part 4 - Enable Polling on the Repository

So far, we have created a Jenkins job that pulls a fresh copy of the source tree prior to building. But we triggered the build manually. In most cases, we would like to have the build triggered automatically whenever a developer makes changes to the source code in the version control system.

- ___1. In the Jenkins web application, navigate to the **SimpleGreeting** project. You can probably find the project in the breadcrumb trail near the top of the window. Alternately, go to the Jenkins home page and then click on the project.
- ___2. Click the **Configure** link.
- ___3. Scroll down to find the **Build Triggers** section.



__4. Click on the checkbox next to **Poll SCM**, and then enter '* * * * *' into the **Schedule** text box. [Make sure there is a space between each *]



Note: The above schedule sets up a poll every minute. In a production scenario, that's a higher frequency than we need, and it can cause unnecessary load on the repository server and on the Jenkins server. You'll probably want to use a more reasonable schedule - perhaps every 15 minutes. That would be 'H/15 * * * *' in the schedule box.

__5. Click **Save**.

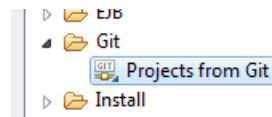
Part 5 - Import the Project into Eclipse

In order to make changes to the source code, we'll clone a copy of the Git repository into an Eclipse project.

__1. Start Eclipse by running C:\Software\eclipse\eclipse.exe and use C:\Workspace as Workspace. Close the Welcome screen.

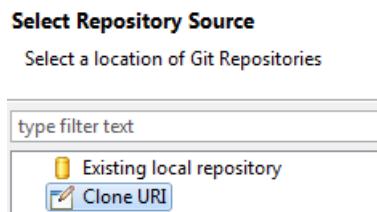
__2. From the main menu, select **File** → **Import...**

__3. Select **Git** → **Projects from Git**.



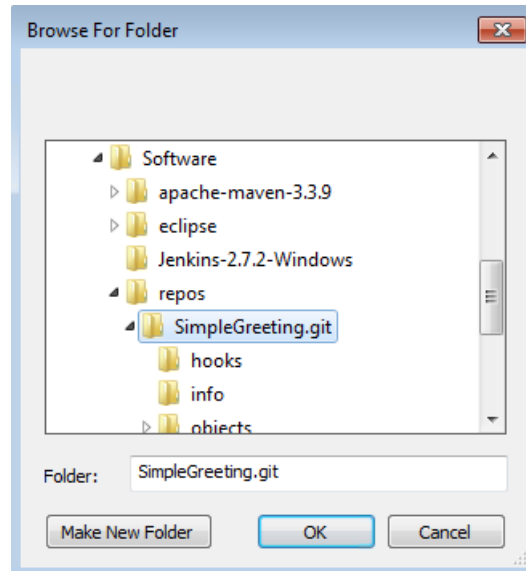
__4. Click **Next**.

__5. Select **Clone URI** and then click **Next**.



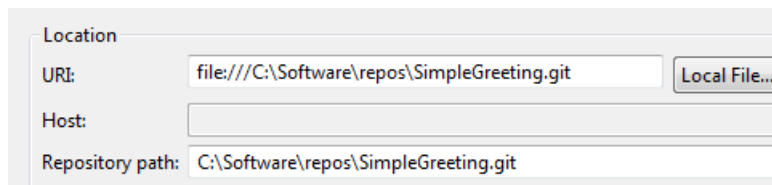
You might think that 'Existing local repository' would be the right choice, since we're cloning from a folder on the same machine. Eclipse, however, expects a "local repository" to be a working directory, not a bare repository. On the other hand, Jenkins will complain if we try to get source code from a repository with a working copy. So the correct thing is to have Jenkins pull from a bare repository, and use **Clone URI** to have Eclipse import the project from the bare repository.

__6. Click on **Local File...** and then navigate to **C:\Software\repos\SimpleGreeting.git**

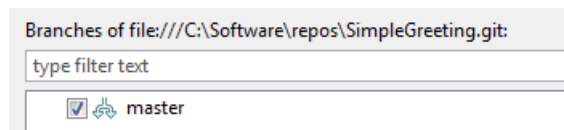


__7. Click **OK**.

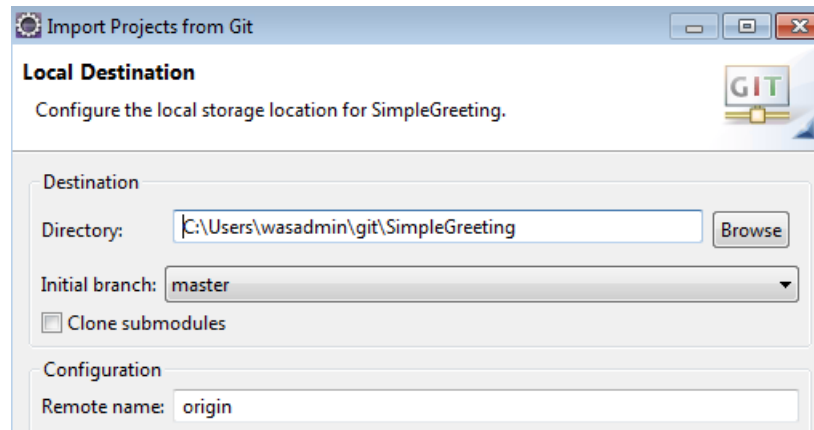
__8. Back in the **Import Projects** dialog, click **Next**.



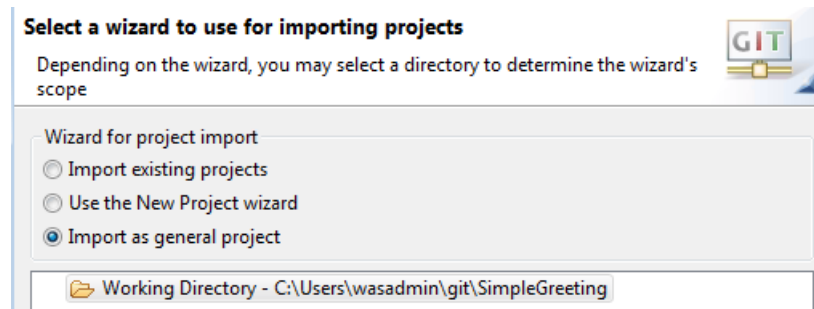
__9. Click **Next** to accept the default 'master' branch.



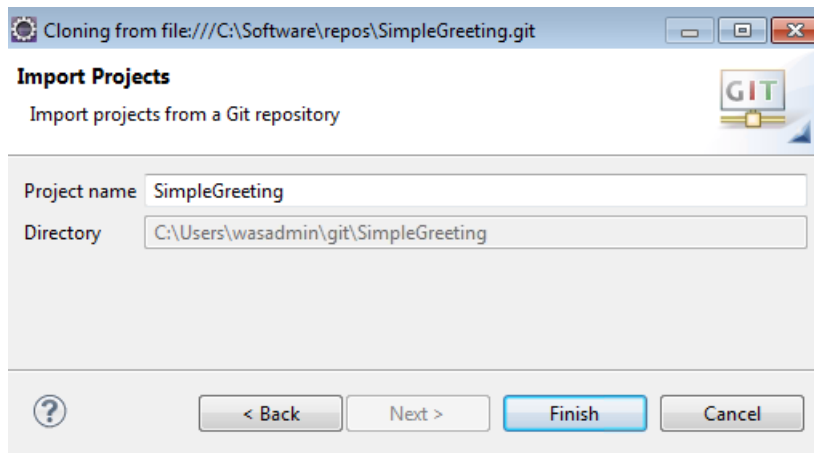
__10. In the **Local Destination** pane, leave the defaults and click **Next**.



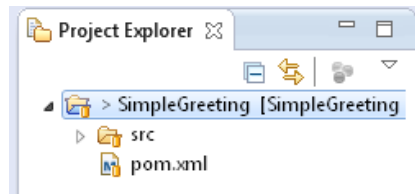
__11. Select **Import as a General Project** and click **Next**.



__12. Click **Finish**.

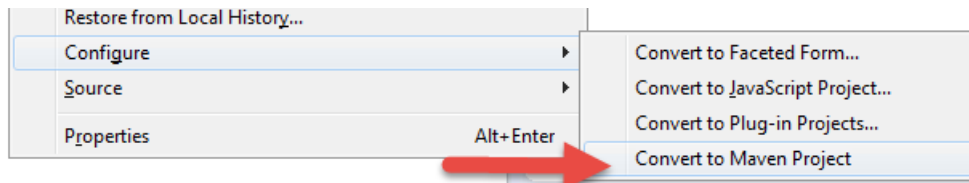


__13. You should see the new project in the **Project Explorer**, expand it.

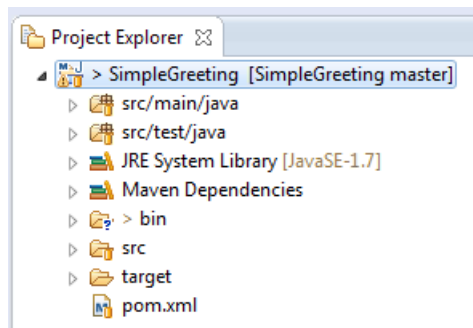


At this point, we could go ahead and edit the files, but Eclipse doesn't understand the project's layout. Let's tell Eclipse what we know - that this project is built using Apache Maven.

__14. Right-click on the **SimpleGreeting** project in the **Project Explorer**, and then select **Configure** → **Convert to Maven Project**.



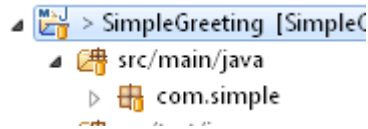
__15. After few seconds, you should now see the project represented as a Maven project in the **Project Explorer**.



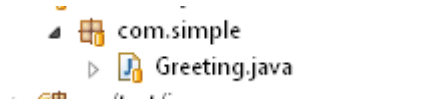
Part 6 - Make Changes and Trigger a Build

The project that we used as a sample consists of a basic "Hello World" style application, and a unit test for that application. In this section, we'll alter the core application so it fails the test, and then we'll see how that failure appears in Jenkins.

__1. In the **Project Explorer**, expand the **src/main/java** tree node.



__2. Expand the **com.simple** package to reveal the **Greeting.java** file.



__3. Double-click on **Greeting.java** to open the file.

__4. Find the line that says 'return "GOOD";'. Edit the line to read 'return "BAD";'

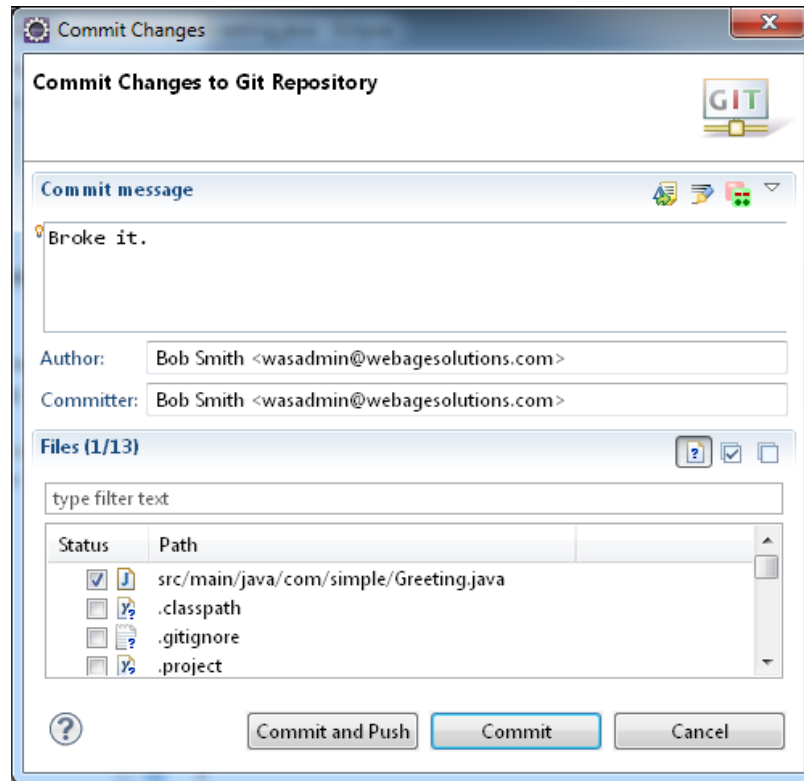
```
public String getStatus(){  
    return "BAD";  
}
```

__5. Save the file by pressing Ctrl-S or selecting **File** → **Save**.

Now we've edited the local file. The way Git works is that we'll first 'commit' the file to the local workspace repository, and then we'll 'push' the changes to the upstream repository. That's the same repository that Jenkins is reading from. Eclipse has a short-cut button that will commit and push at the same time.

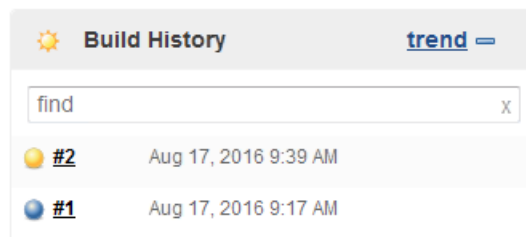
__6. Right-click on **SimpleGreeting** in the **Project Explorer** and then select **Team** → **Commit...**

__7. Enter a few words as a commit message, and then click **Commit and Push**.



__8. Click **OK** in the status dialog that pops up.

__9. Now, flip back to the web browser window that we had Jenkins running in. If you happen to have closed it, open a new browser window and navigate to <http://localhost:8080/SimpleGreeting>. After a few seconds, you should see a new build start up. You can launch a new build if it's taking too long.



__10. If you refresh the page, you should see that there is now a 'Test Result Trend' graph that shows we have a new test failure.



What happened is that we pushed the source code change to the Git repository that Jenkins is reading from. Jenkins is continually polling the repository to look for changes. When it saw that a new commit had been performed, Jenkins checked out a fresh copy of the source code and performed a build. Since Maven automatically runs the unit tests as part of a build, the unit test was run. It failed, and the failure results were logged.

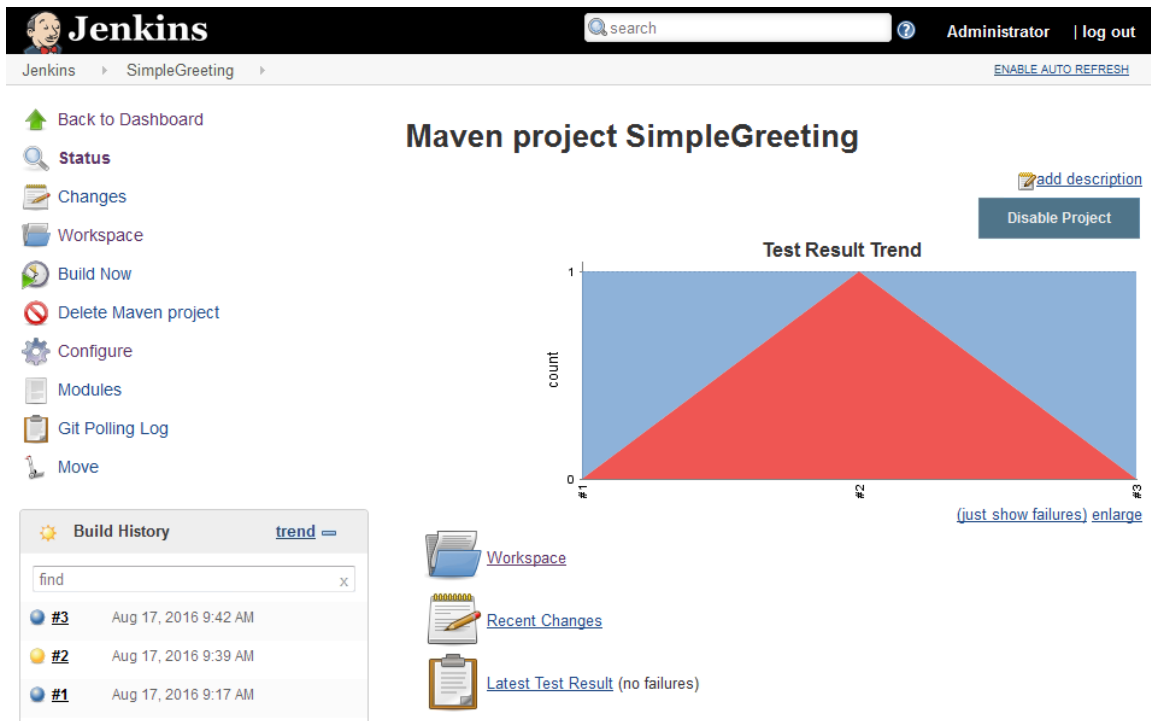
Part 7 - Fix the Unit Test Failure

__1. Back in eclipse , edit the file **Greeting.java** so that the class once again returns 'GOOD'.

```
public String getStatus(){  
    return "GOOD";  
}
```

__2. As above, save, commit and push the change.

3. Watch the Jenkins web browser window. After a minute or two you should see the build start automatically, when the build is done then refresh the page. You can launch a new build if it's taking too long.



4. Close all.

Part 8 - Review

In this lab you learned

- How to Set-up a set of distributed Git repositories
- How to create a Jenkins Job that reads from a Git repository
- How to configure Jenkins to build automatically on source code changes.

Lab 5 - Create a Pipeline

In this lab you will explore the Pipeline functionality.

At the end of this lab you will be able to:

1. Create a simple pipeline
2. Use a 'Jenkinsfile' in your project
3. Use manual input steps in a pipeline

Part 1 - Create a Simple Pipeline

We can create a pipeline job that includes the pipeline script in the job configuration, or the pipeline script can be put into a 'Jenkinsfile' that's checked-in to version control.

To get a taste of the pipeline, we'll start off with a very simple pipeline defined in the job configuration.

Prerequisite: We need to have a project in source control to check-out and build. For this example, we're using the 'SimpleGreeting' project that we previously cloned to a 'Git' repository at 'C:\Software\repos\SimpleGreeting.git'

In this Lab you will work with the WA2271 VM Image. Check with your instructor if you cannot find this VM.

- __ 1. To connect to Jenkins, open Firefox and enter the following URL .

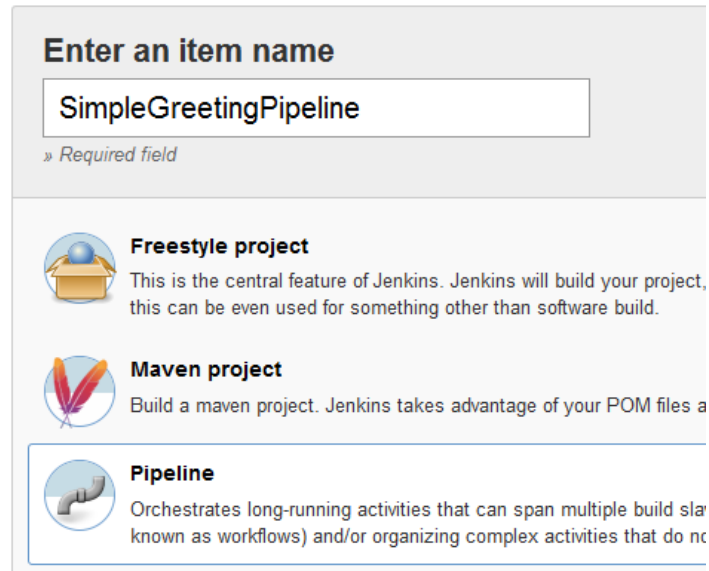
`http://localhost:8080/`

- __ 2. Enter **wasadmin** as user and password and click **Log in**.

- __ 3. Click on the **New Item** link.



___4. Enter '**SimpleGreetingPipeline**' as the new item name, and select '**Pipeline**' as the item type.



___5. When the input looks as above, click on **OK** to create the new item.

___6. Scroll down to the **Pipeline** section and enter the following in the **Script** text window.

```
node {  
  stage 'Checkout'  
    git url: 'C:\\Software\\repos\\SimpleGreeting.git'  
  
  stage 'Maven build'  
    bat 'mvn install'  
  
  stage 'Archive Test Results'  
    step([$class: 'JUnitResultArchiver',  
      testResults: '**/target/surefire-reports/TEST-*.xml'])  
}
```

This pipeline is divided into three stages. First, we checkout the project from our 'git' repository. Then we use the 'bat' command to run 'mvn install' as a Windows batch file. Finally, we use the 'step' command to utilize a step from a standard Jenkins plugin - in this case, the JUnitResultArchiver, to save and display the results of the unit tests.

All of the above is wrapped inside the 'node' command, to indicate that we want to run these commands in the context of a workspace running on one of Jenkins execution agents (or the master node if no agents are available).

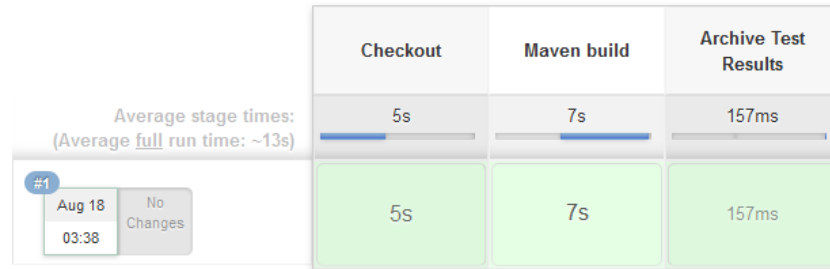
___7. Click on **Save** to save the changes and return to the project page.

__8. Click on **Build Now** to startup a pipeline instance.



__9. After a few moments, you should see the **Stage View** appear, and successive stages will appear as the build proceeds, until all three stages are completed.

Stage View



Part 2 - Pipeline Definition in a 'Jenkinsfile'

For simple pipelines or experimentation, it's convenient to define the pipeline script in the web interface. But one of the common themes of modern software development is "If it isn't in version control, it didn't happen". The pipeline definition is no different, especially as you build more and more complex pipelines.

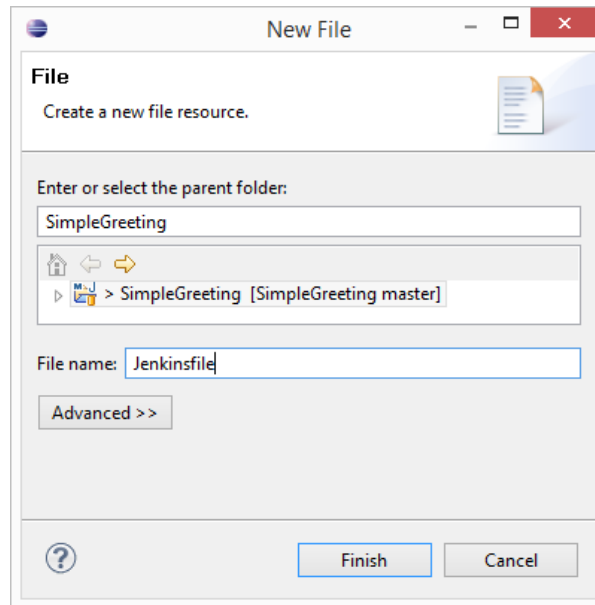
You can define the pipeline in a special file that is checked out from version control. There are several advantages to doing this. First, of course, is that the script is version-controlled. Second, we can edit the script with the editor or IDE of our choice before checking it in to version control. In addition, we can employ the same kind of "SCM Polling" that we would use in a more traditional Jenkins job.

In the following steps, we'll create a Jenkinsfile and create a pipeline job that uses it.

__1. Open the Eclipse editor. If this lab is completed in the normal sequence, you should have the 'SimpleGreeting' project already in Eclipse's workspace. If not, check out the project from version control (consult your instructor for directions if necessary).

__2. In the **Project Explorer**, right-click on the root node of the **SimpleGreeting** project, and then select **New** → **File**.

__3. Enter '**Jenkinsfile**' as the file name.



__4. Click **Finish** to create the new file.

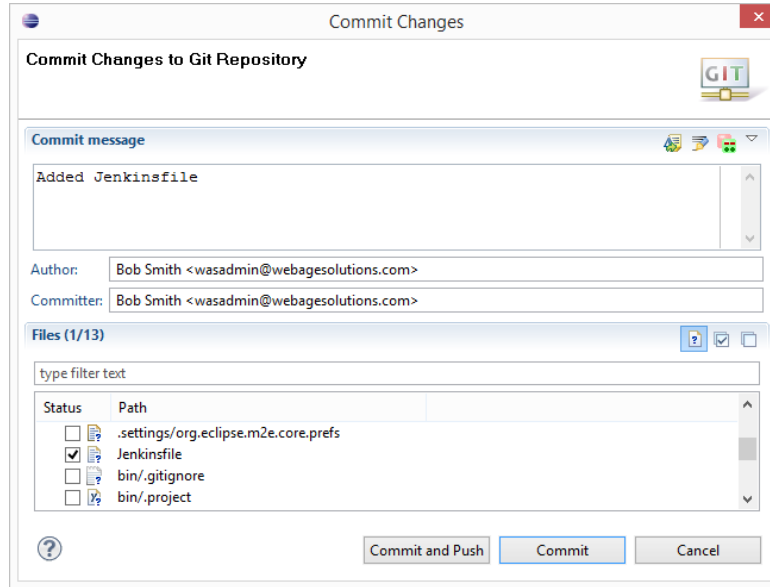
__5. Enter the following text into the new file (Note: this is the same script that we used above, so you could copy/paste it from the Jenkins Web UI if you want to avoid some typing):

```
node {  
    stage 'Checkout'  
    git url: 'C:\\Software\\repos\\SimpleGreeting.git'  
  
    stage 'Maven build'  
    bat 'mvn install'  
  
    stage 'Archive Test Results'  
    step([$class: 'JUnitResultArchiver',  
        testResults: '**/target/surefire-reports/TEST-*.xml'])  
}
```

__6. Save the Jenkinsfile by selecting **File** → **Save** from the main menu, or by hitting Ctrl-S.

__7. In the **Project Explorer**, right-click on the **SimpleGreeting** node, and then select **Team** → **Commit...**

__8. Eclipse will display the **Commit Changes** dialog. Click the checkbox next to **Jenkinsfile** (to include that file in the commit) and enter a commit message.

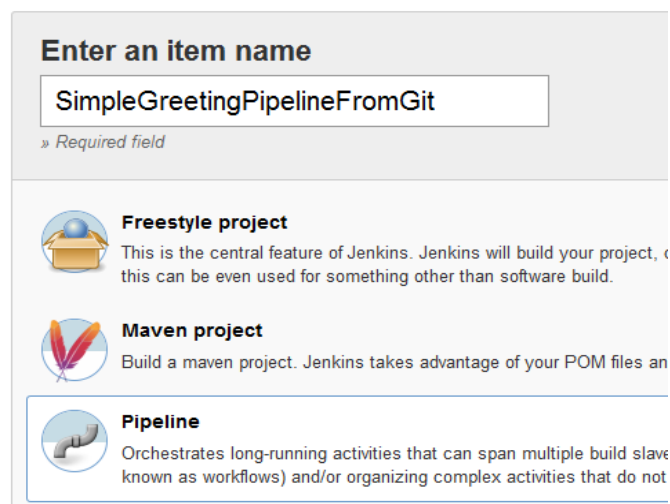


__9. Click **Commit and Push**, and then click **OK** to dismiss the status dialog.

Now we have a Jenkinsfile in our project, to define the pipeline. Next, we need to create a Jenkins job to use that pipeline.

__10. In the Jenkins user interface, navigate to the root page, and then click on **New Item**.

__11. Enter '**SimpleGreetingPipelineFromGit**' as the name of the new item, and select **Pipeline** as the item type.

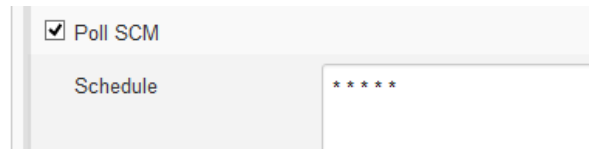


__12. Click **OK** to create the new item.

__13. Scroll down to the **Build Triggers** section.

__14. Click on **Poll SCM** and enter '* * * * *' as the polling schedule. This entry will

cause Jenkins to poll once per minute.



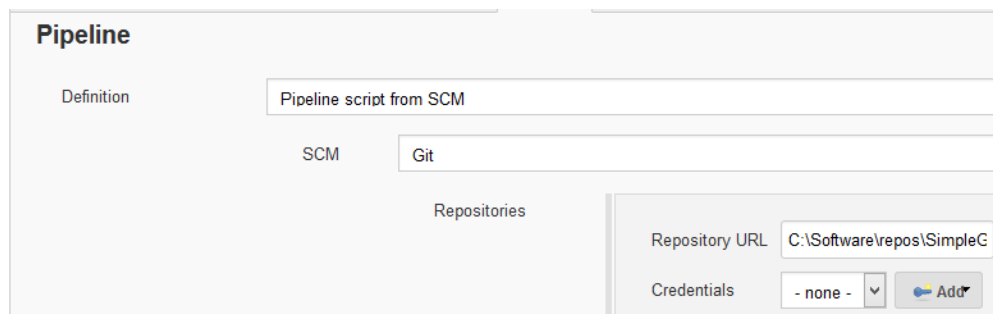
A screenshot of a Jenkins configuration page. It shows a checkbox labeled 'Poll SCM' which is checked. Below it is a 'Schedule' field containing six asterisks '*****'.

__15. Scroll down to the **Pipeline** section, and change the **Definition** entry to 'Pipeline Script from SCM'

__16. Enter the following:

SCM:	Git
Repository URL:	C:\Software\repos\SimpleGreeting.git

__17. Press tab. The **Pipeline** section should look similar to:



A screenshot of the Jenkins 'Pipeline' configuration section. The 'Definition' field is set to 'Pipeline script from SCM'. The 'SCM' dropdown is set to 'Git'. Under the 'Repositories' section, the 'Repository URL' is 'C:\Software\repos\SimpleC' and the 'Credentials' dropdown is set to '- none -' with an 'Add' button next to it.

__18. Click **Save** to save the new configuration.

__19. Click **Build Now** to launch the pipeline.

__20. You should see the pipeline execute, similar to the previous section.

Part 3 - Try out a Failing Build

The pipeline that we've defined so far appears to work perfectly. But we haven't tested it with a build that fails. In the following steps, we'll insert a test failure and see what happens to our pipeline.

__1. In Eclipse, go to the **Project Explorer** and locate the file 'Greeting.java'. It will be under **src/main/java** in the package 'com.simple'. Open 'Greeting.java'.

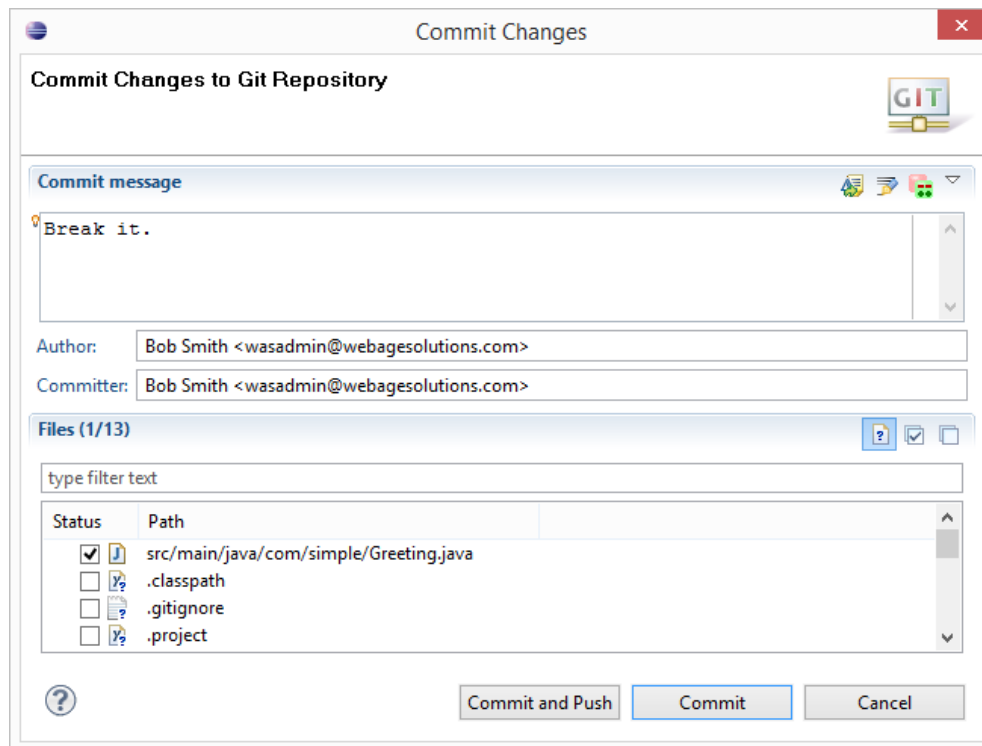
__2. Locate the line that reads 'return "GOOD";'. Change it to read 'return "BAD";'

```
public String getStatus() {  
  
    return "BAD";  
  
}
```

__3. Save the file.

__4. In the **Project Explorer**, right-click on **Greeting.java** and then select **Team** → **Commit...** (This is a shortcut for committing a single file).

__5. Enter an appropriate commit message and then click **Commit and Push**.

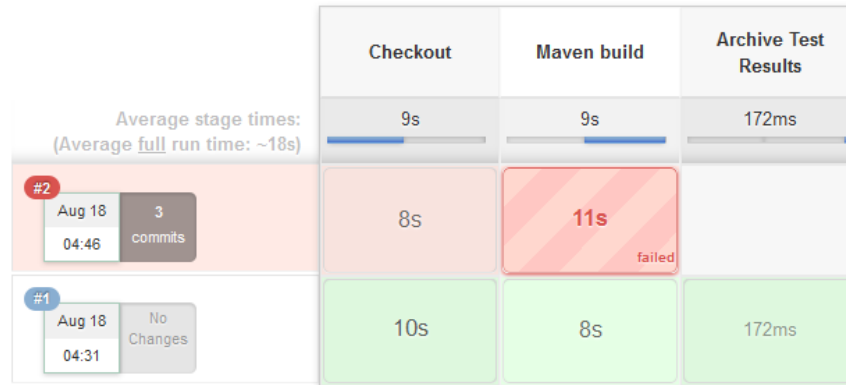


__6. Click **OK** in the results box, to close it.

__7. Switch back to Jenkins.

__ 8. In a minute or so, you should see a build launched automatically. Jenkins has picked up the change in the 'Git' repository and initiated a build. If nothing happens then click **Build Now**.

Stage View



This time, the results are a little different. The 'Maven Build' stage is showing a failure, and the 'Archive Test Results' stage was never executed.

What's happened is that the unit tests have failed, and Maven exited with a non-zero result code because of the failure. As a result, the rest of the pipeline was canceled. This behavior probably isn't what you want or what you expect in most cases. We'd like to go ahead and archive the test results, even when there's a failure. That way, we can see the trend including failed tests.

The solution here is to add a command-line parameter to the Maven invocation. If we add '-Dmaven.test.failure.ignore' to the Maven command line, then Maven will continue with the build even if the tests fail.

__ 9. Go back to Eclipse and open the 'Jenkinsfile' if necessary.

__ 10. Alter the 'bat "mvn..." line to read as follows:

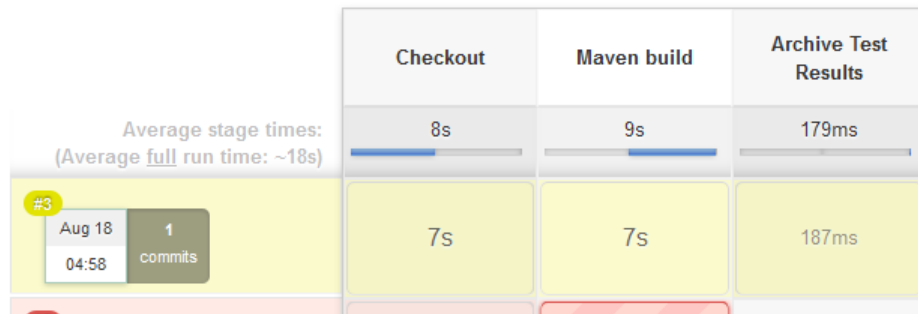
```
bat 'mvn -Dmaven.test.failure.ignore install'
```

__ 11. Save the 'Jenkinsfile'.

__ 12. Commit and push the changes using the same technique as above.

__13. After a minute or so, you should see a new Pipeline instance launched. If nothing happens then click **Build Now**.

Stage View



This time, the pipeline runs to completion, and the test results are archived as expected. Notice that the build is now flagged as 'unstable' (indicated by the yellow color and the icon). The JUnit archiver noticed the failures and flagged the build unstable, even though Maven exited normally.

Part 4 - Add a Manual Approval Step

One of the interesting features of the Pipeline functionality is that we can include manual steps. This is very useful when we're implementing a continuous deployment pipeline. For example, we can include a manual approval step (or any other data collection) for cases like 'User Acceptance Testing' that might not be fully automated.

In the steps below, we'll add a manual step before a simulated deployment.

- __1. Go to 'Eclipse' and open the 'Jenkinsfile' if necessary.
- __2. Add the following to the end of the file:

```
stage 'User Acceptance Test'

def response= input message: 'Is this build good to go?',
    parameters: [choice(choices: 'Yes\nNo',
        description: '', name: 'Pass')]

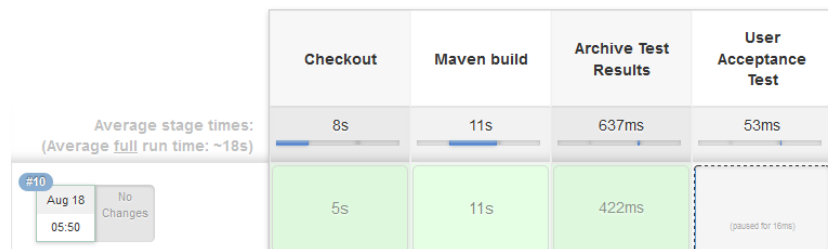
if(response=="Yes") {
    node {
        stage 'Deploy'
        bat 'mvn -Dmaven.test.failure.ignore install'
    }
}
```


This portion of the script creates a new stage called 'User Acceptance Test', then executes an 'input' operation to gather input from the user. If the result is 'Yes', the script executes a deploy operation in a new 'node' step. (In this case, we're repeating the 'mvn install' that we did previously. Only because we don't actually have a deployment repository setup)

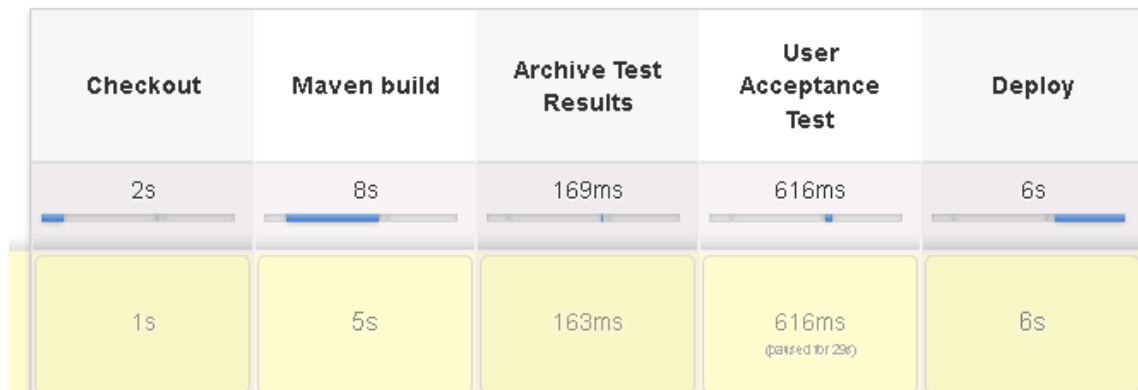
__3. Save and commit 'Jenkinsfile' as previously.

__4. When the pipeline executes, watch for a "paused" stage called 'User Acceptance Test'. If you move your mouse over this step, you'll be able to select "Yes" or "No", and the button Proceed.

Stage View



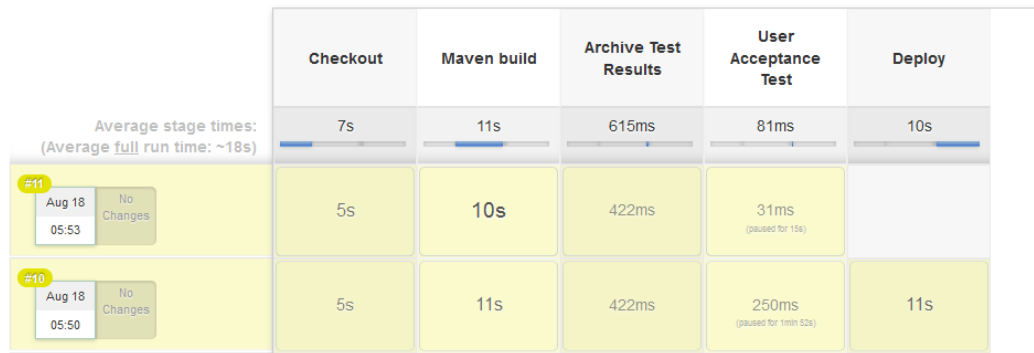
__5. Select **Yes** and click **Proceed**, you should see the job run to completion.



___6. Run the pipeline again (click **Build Now**), but this time, select **No** on the 'User Acceptance Test' and click **Proceed**.

You'll see that the pipeline exits early and doesn't run the 'deploy' stage.

Stage View



What's happened is that the final 'Deploy' stage was only executed when we indicated that the 'User Acceptance Test' had passed.

___7. Close all.

Part 5 - Review

In this lab, we explored the Pipeline functionality in Jenkins. We built a simple pipeline in the Jenkins web UI, and then used a 'Jenkinsfile' in the project. Lastly, we explored how to gather user input, and then take different build actions based on that user input.

Lab 6 - Install Prerequisites

In this lab you will install the Apache web server, MySQL, and PHP. They are required by Continuous Code Quality (SonarQube) and Continuous Application Monitoring (Nagios) tools.

Part 1 - Launch terminal

In this Lab you will work with the WA2543 VM Image. Check with your instructor if you cannot find this VM.

In this part you will launch the Linux terminal.

- ___ 1. In the task bar, click **Search** button.
- ___ 2. In search text box, type in **terminal** and hit enter key on the keyboard.

Alternatively, you can press Ctrl+Alt+T to access the terminal.

- ___ 3. Run following command to switch to **Downloads** directory.

```
cd ~/Downloads
```

Part 2 - Install Java other prerequisites.

In this part you will install Java and other prerequisites.

- ___ 1. View the script that will install Java and other prerequisites.

```
cat ~/Documents/labs/prereqs/java.sh
```

- ___ 2. Run the script.

```
bash ~/Documents/labs/prereqs/java.sh
```

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

Note: Press **Y** if it prompts you to do so.

- ___ 3. Verify Java is installed.

```
java -version
```

Part 3 - Install Apache web server

In this part you will install Apache web server.

- ___ 1. Update APT repository.

```
sudo apt-get update
```

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

__ 2. Install Apache web server.

```
sudo apt-get install apache2
```

Note: Press Y to continue installation.

Since we have a service running on port 80, it will show errors. If this were unexpected, you could run `netstat -an | grep 80` to verify what is running and take appropriate action.

__ 3. Edit ports.conf file.

```
sudo nano /etc/apache2/ports.conf
```

You will modify the default HTTP and HTTPS ports.

__ 4. Change "80" to "1080" and "443" to "10443"

__ 5. Press Ctrl+O to save the file and hit Enter.

__ 6. Press Ctrl+X.

__ 7. Edit the virtualhost file.

```
sudo nano /etc/apache2/sites-enabled/000-default.conf
```

Next you will modify the default http port.

__ 8. Change 80 to 1080.

__ 9. Press Ctrl+O to save the file and hit Enter.

__ 10. Press Ctrl+X to exit to the terminal.

__ 11. Edit apache2.conf.

```
sudo nano /etc/apache2/apache2.conf
```

__ 12. Append following text to the bottom of the file to allow the server to identify itself and eliminate DNS binding errors for hostname.

```
ServerName localhost
```

__ 13. Press Ctrl+O to save the file and hit Enter.

__ 14. Press Ctrl+X to exit to the terminal.

__15. Restart Apache web server.

```
sudo service apache2 restart
```

__16. Launch nano text editor and create a test page.

```
sudo nano /var/www/html/test.html
```

__17. Type in following text.

```
<h1>Hello world!</h1>
```

__18. Press Ctrl+O to save the file and hit Enter.

__19. Press Ctrl+X to exit to the terminal.

__20. Launch Firefox web browser from the task bar and type in following URL.

```
http://localhost:1080/test.html
```

Notice the test page shows up.

If you want to install Apache web server using a chef cookbook then use following recipe.

```
# install Apache web server
package "apache2" do
  action :install
end

# start the web server
service "apache2" do
  action [:enable, :start]
end

# create a test file
file "/var/www/test.html" do
  content "<html><head><title>Test
Page</title></head><body><h1>Hello World!
</h1></body></html>"
end
```

Part 4 - Install MySQL

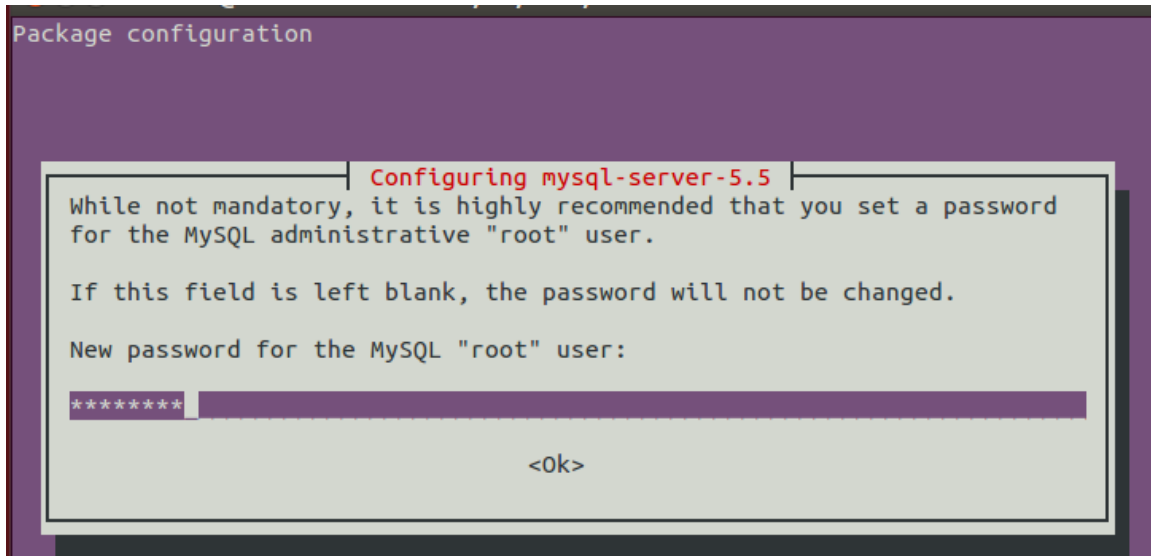
In this part you will install MySQL.

__1. Install MySQL.

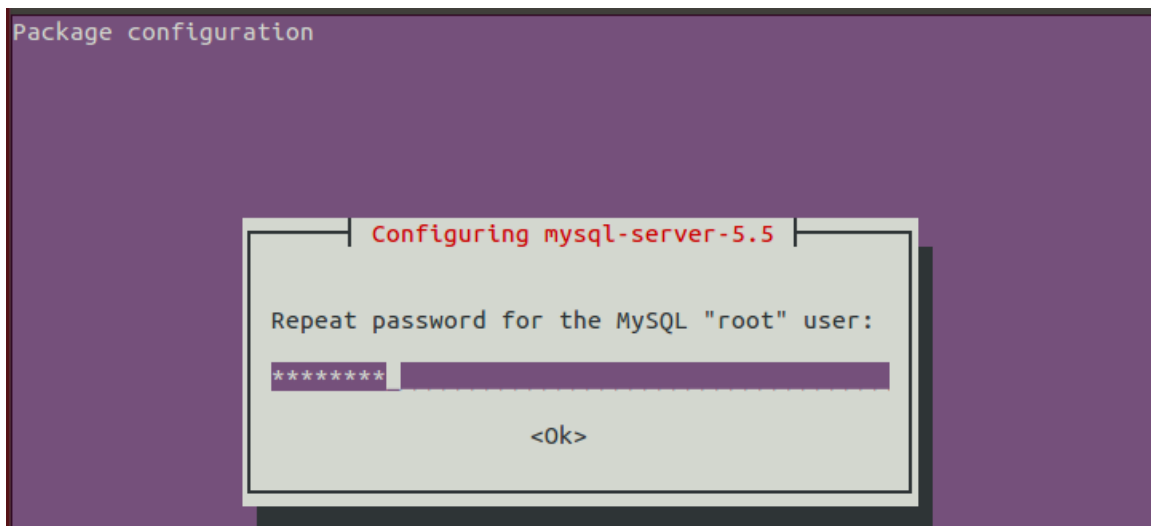
```
sudo apt-get install mysql-server-5.6 php5-mysql
```

__2. Press Y to continue installation.

__3. Enter **wasadmin** when prompted to enter the "root" password.



__4. Enter **wasadmin** again when prompted to reenter the "root" password.



__5. Secure the installation by running following command.

```
sudo mysql_secure_installation
```

__6. Enter **"wasadmin"** (without quotes) when prompted to enter the "root" password.

__7. Press "n" when prompted to change the "root" password.

- __ 8. Press "Y" when prompted to remove anonymous users.
- __ 9. Press "Y" when prompted to disallow remote root login.
- __ 10. Press "Y" when prompted to remove test database.

Ignore error messages. You don't have a test database.

- __ 11. Press "Y" when prompted to reload privileges table.

- __ 12. Restart MySQL service.

```
sudo service mysql restart
```

- __ 13. Connect to the MySQL server.

```
sudo mysql -u root -p
```

- __ 14. Enter "wasadmin" (without quotes) when prompted to enter the password.

- __ 15. At "mysql" prompt, run following command to obtain MySQL version.

```
SELECT VERSION();
```

Note: It should 5.6

- __ 16. Run following command to get list of databases.

```
show databases;
```

Notice it shows result like this.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
+-----+
3 rows in set (0.00 sec)
```

- __ 17. Type "quit" to exit to the terminal.

Part 5 - Install PHP

In this part you will install and configure PHP.

- __ 1. Install PHP and helper packages.

```
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

Press Y to continue installation.

__ 2. Edit Apache web server's `dir.conf` file to change default file.

```
sudo nano /etc/apache2/mods-enabled/dir.conf
```

__ 3. Change the line that has "DirectoryIndex" so that `index.php` appears before `index.html`

__ 4. Press Ctrl+O to save the file and hit Enter.

__ 5. Press Ctrl+X to exit to the terminal.

__ 6. Restart Apache web server.

```
sudo service apache2 restart
```

__ 7. Launch nano text editor to create a same PHP file.

```
sudo nano /var/www/html/index.php
```

__ 8. Type following text.

```
<?php phpinfo(); ?>
```


__ 9. Press Ctrl+O to save the file and hit Enter.

__ 10. Press Ctrl+X to exit to the terminal.

__ 11. Launch Firefox from the task bar and enter following URL.

```
http://localhost:1080/index.php
```

Notice it shows a page like this.

<div> <div>PHP Version 5.5.9-1ubuntu4.17</div>  </div>	
System	Linux student-VirtualBox 4.2.0-27-generic #32~14.04.1-Ubuntu SMP Fri Jan 22 15:32:26 UTC 2016 x86_64
Build Date	May 19 2016 19:05:33
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-json.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-readline.ini
PHP API	20121113

__12. Remove the test file.

```
sudo rm /var/www/html/index.php
```

__13. Close the Terminal and web browser.

Part 6 - Review

In this lab you installed and configured Apache web server, MySQL, and PHP.

Lab 7 - Continuous Code Quality - SonarQube

In this lab you will install, configure, and use SonarQube server, SonarQube Scanner, and Maven to check code quality.

Part 1 - Launch terminal

In this Lab you will work with the WA2543 VM Image. Check with your instructor if you cannot find this VM.

In this part you will launch the Linux terminal.

- ___ 1. In the task bar, click **Search** button.
- ___ 2. In search text box, type in **terminal** and hit enter key on the keyboard.

Alternatively, you can press Ctrl+Alt+T to access the terminal.

- ___ 3. Run following command to switch to **Downloads** directory.

```
cd ~/Downloads
```

Part 2 - Create SonarQube database and user

In this part you will create SonarQube database and user.

- ___ 1. Connect to MySQL.

```
sudo mysql -u root -p
```

- ___ 2. Enter twice "wasadmin" (without quotes) when prompted to enter the "root" password.

- ___ 3. Create database.

```
CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8_general_ci;
```

- ___ 4. Create a user.

```
CREATE USER 'wasadmin' IDENTIFIED BY 'wasadmin';
```

- ___ 5. Grant permission on the database to the user.

```
GRANT ALL ON sonar.* TO 'wasadmin'@'%' IDENTIFIED BY 'wasadmin';  
GRANT ALL ON sonar.* TO 'wasadmin'@'localhost' IDENTIFIED BY 'wasadmin';
```

- ___ 6. Reload privileges.

```
FLUSH PRIVILEGES;
```

__7. Type the following command and press enter to exit to the terminal.

```
exit
```

Part 3 - Extract SonarQube archive

In this part you will extract SonarQube archive.

__1. Switch to the home directory.

```
cd ~
```

__2. Unzip the archive.

```
sudo unzip /home/wasadmin/Downloads/sonarqube-5.5.zip
```

__3. Move the archive to "opt" directory.

```
sudo mv sonarqube-5.5 /opt/sonar
```

Part 4 - Configure SonarQube

In this part you will configure SonarQube.

__1. Edit sonar.properties.

```
sudo nano /opt/sonar/conf/sonar.properties
```

__2. Find sonar.jdbc.username and uncomment it by removing # in front of the property. Also set user name. It should read as follows.

```
sonar.jdbc.username=wasadmin
```

__3. Find sonar.jdbc.password and uncomment it by removing # in front of the property. Also set user name. It should read as follows.

```
sonar.jdbc.password=wasadmin
```

__4. In MySQL section, uncomment sonar.jdbc.url by removing # in front of the property.

It should read as follows.

```
sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?  
useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=true&useConfig  
s=maxPerformance
```

__5. In "Web Server" of the configuration file find sonar.web.host then uncomment and

set it as follows.

```
sonar.web.host=127.0.0.1
```

__ 6. In "Web Server" of the configuration file find sonar.web.context then uncomment and set it as follows.

```
sonar.web.context=/sonar
```

Note: /sonar will act as the virtual directory for accessing SonarQube's web user interface.

__ 7. In "Web Server" of the configuration file find sonar.web.port then uncomment and set it as follows.

```
sonar.web.port=1090
```

__ 8. Press CTRL+O to save the file and hit Enter.

__ 9. Press CTRL+X to exit.

Part 5 - Run SonarQube server and connect to the web user interface

In this part you will start SonarQube server and connect its web user interface.

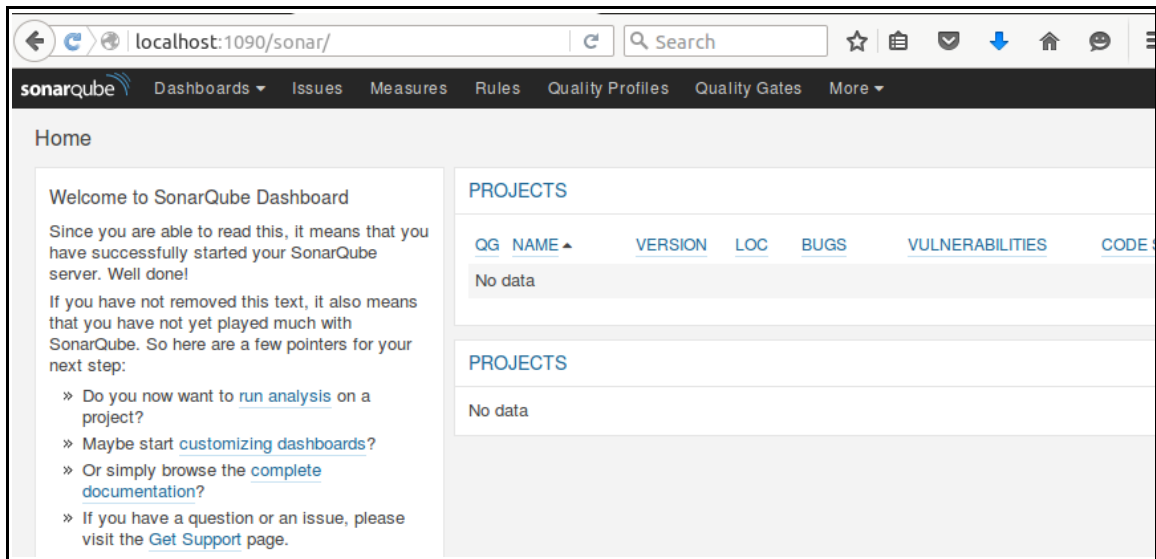
__ 1. Start SonarQube server.

```
sudo /opt/sonar/bin/linux-x86-64/sonar.sh start
```

__ 2. Start Firefox from the task bar and enter following URL (it may take a while to load the page).

```
http://localhost:1090/sonar
```

Notice the web page shows up like this.



___3. Close the web browser and Terminal.

Part 6 - Review

In this lab you installed, configured, and used SonarQube Serve, SonarQube Scanner, and Maven to check code quality.

Lab 8 - Continuous Monitoring - Nagios

In this lab you will install, configure, and use Nagios to monitor servers and services

Part 1 - Launch terminal

In this Lab you will work with the WA2543 VM Image. Check with your instructor if you cannot find this VM.

In this part you will launch the Linux terminal.

- __ 1. In the task bar, click **Search** button.
- __ 2. In search text box, type in **terminal** and hit enter key on the keyboard.

Alternatively, you can press Ctrl+Alt+T to access the terminal.

- __ 3. Run following command to switch to **Downloads** directory.

```
cd ~/Downloads
```

Part 2 - Execute script for installing Nagios and Nagios plugins.

In this part you will execute a script that will create user account for using Nagios, install Nagios, and install Nagios plugins.

- __ 1. View installation script.

```
cat ~/Documents/labs/nagios/install.sh | more
```

- __ 2. Run the script.

```
bash ~/Documents/labs/nagios/install.sh
```

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

Enter **Y** if prompted to do so.

Part 3 - Configure Nagios

In this part you will configure Nagios.

- __ 1. Open nagios.cfg file in text editor.

```
sudo nano /usr/local/nagios/etc/nagios.cfg
```

- __ 2. Uncomment the line `#cfg_dir=/usr/local/nagios/etc/servers` by removing `#` symbol.

Note: This is the directory where you will create a configuration file for each host you want to monitor. You will use this directory later in this lab.

- __ 3. Press Ctrl+O to save the file and hit enter.
- __ 4. Press Ctrl+X to exit to the terminal.
- __ 5. Create directory where you will store configuration file for each server that you will monitor.

```
sudo mkdir /usr/local/nagios/etc/servers
```

- __ 6. Review contacts.cfg.

```
sudo nano /usr/local/nagios/etc/objects/contacts.cfg
```

Scroll down and notice there's define_contact statement that defines email address and other properties.

- __ 7. Press Ctrl+X to exit to the terminal.

Part 4 - Configure Apache

In this part you will execute a script which will configure Apache web server so you can access Nagio's web interface.

- __ 1. View script that you will execute for configuring Apache web server.

```
cat ~/Documents/labs/nagios/configure.sh
```

- __ 2. Execute the script.

```
bash ~/Documents/labs/nagios/configure.sh
```

If it prompts you to enter password, enter "wasadmin" (without quotes).

Note: Default admin user is nagiosadmin. Since you have used a non-default admin name so you need to edit cgi.cfg file as well. You will do that in the next step.

- __ 3. Edit cgi.cfg file.

```
sudo nano /usr/local/nagios/etc/cgi.cfg
```

- __ 4. Press Ctrl+\

- __ 5. In "Search (to replace)" type "nagiosadmin" (without quotes) and press the Enter key on the keyboard.

- __ 6. In "replace with" type "wasadmin" (without quotes) and press the Enter key on the keyboard.

- __ 7. Press "A" to replace all instances of nagiosadmin with wasadmin.

- __ 8. Press Ctrl+O to save the file and hit enter.

- __ 9. Press Ctrl+X to exit to the terminal.

__10. View script that you will execute for configuring Apache web server.

```
cat ~/Documents/labs/nagios/restart.sh
```

__11. Execute the script.

```
bash ~/Documents/labs/nagios/restart.sh
```

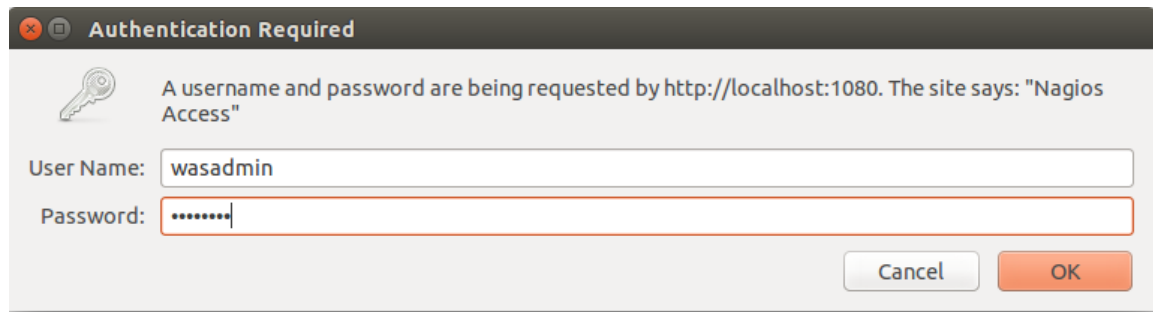
Part 5 - Access the Nagios Web Interface

In this part you will access the Nagios web interface.

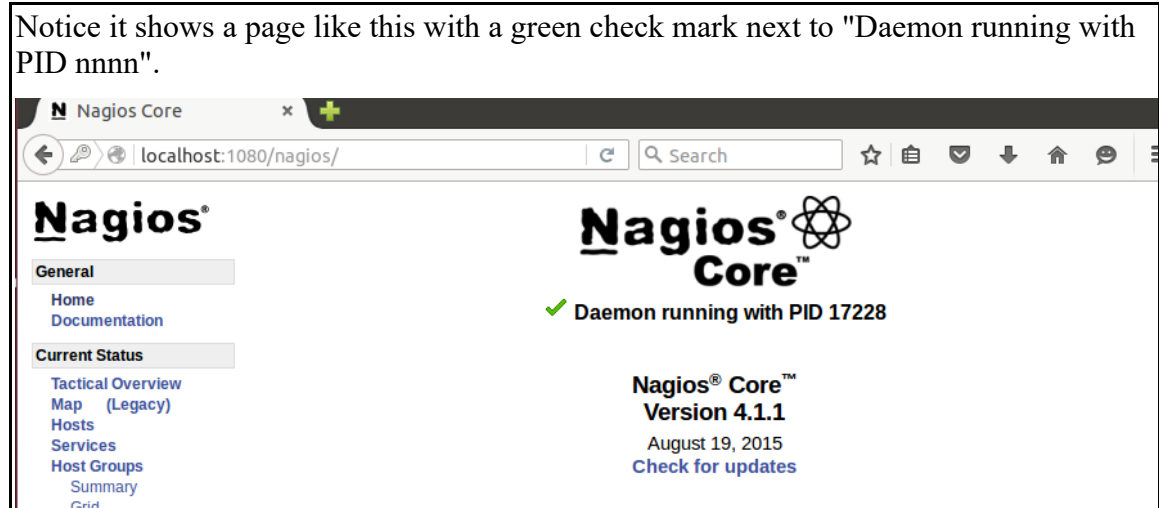
__1. Launch firefox from the task bar and enter following URL.

```
http://localhost:1080/nagios
```

__2. Enter "wasadmin" as user name and password.



__3. Press x when it prompts you to save the credentials.



__4. Under "Current Status", click "Services".

Notice it shows a page like this

General

Home

Documentation

Current Status

Tactical Overview

Map (Legacy)

Hosts

Services

Host Groups

Summary

Grid

Service Groups

Summary

Grid

Problems

Services (Unhandled)

Hosts (Unhandled)

Network Outages

Quick Search:

Reports

Availability

Trends (Legacy)

Alerts

History

Summary

Current Network Status

Last Updated: Tue Jul 19 07:23:12 MDT 2016
Updated every 90 seconds
Nagios® Core™ 4.1.1 - www.nagios.org
Logged in as wasadmin

View History For all hosts
View Notifications For All Hosts
View Host Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
1	0	0	0

All Problems All Types

0	1
---	---

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
7	0	0	1	0

All Problems All Types

1	8
---	---

Service Status Details For All Hosts

Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
localhost	Current Load	OK	07-19-2016 07:21:33	0d 1h 1m 39s	1/4	OK - load average: 0.44, 0.50, 0.50
	Current Users	OK	07-19-2016 07:22:11	0d 1h 1m 1s	1/4	USERS OK - 2 users currently logged in
	HTTP	OK	07-19-2016 07:22:48	0d 1h 0m 24s	1/4	HTTP OK: HTTP/1.1 301 Moved Permanently - 374 bytes in 0.000 second response time
	PING	OK	07-19-2016 07:18:26	0d 0h 59m 46s	1/4	PING OK - Packet loss = 0%, RTA = 0.04 ms
	Root Partition	OK	07-19-2016 07:19:03	0d 0h 59m 9s	1/4	DISK OK - free space: / 20854 MB (67% inode=80%):
	SSH	CRITICAL	07-19-2016 07:18:18	0d 0h 58m 31s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	07-19-2016 07:20:18	0d 0h 57m 54s	1/4	SWAP OK - 100% free (8676 MB out of 8676 MB)
	Total Processes	OK	07-19-2016 07:20:56	0d 0h 57m 16s	1/4	PROCS OK: 110 processes with STATE = RSZDT

Part 6 - Using Nagios Plugins

In this part you will utilize some of the Nagios plugins you installed in Part 6 of this lab.

__ 1. Switch to the plugins directory.

```
cd /usr/local/nagios/libexec
```

__ 2. Get a list of files.

```
ls
```

Notice there are various check_* files. These plugins can be used to check disk storage, checking whether ftp/http is up and running or not, ping a host etc.

__ 3. Check Google's http status.

```
./check_http -H www.google.ca
```

__ 4. Check Apache web server's http status on local machine.

```
./check_http -H localhost -p 1080
```

If you have internet access then notice it returns HTTP OK: HTTP/1.1 200 OK message.

__ 5. Try http status for an invalid host.

```
./check_http -H invalidtest
```

Notice it displays "Name or service not known" message. It could display Socket time out.

__6. Get host up-time.

```
./check_uptime
```

Notice it displays the uptime.

__7. Get disk storage.

```
df -k
```

Make a note of "Use%" for /. Subtract the value from 100%.

__8. Get disk storage and set threshold using Nagios plugin.

```
./check_disk -w 10% -c 5%
```

Note: -w will display a warning if disk storage is less than 10% free. -c shows critical message if disk storage is less than 5% free.

To -w and -c pass a value greater than the number you noticed in step 10 to see if it shows warning and critical message.

Part 7 - Adding Hosts to Monitor

In this part you will review how to create a configuration file for each of the remote hosts that you want to monitor.

__1. Create a configuration file for a new host.

```
sudo nano /usr/local/nagios/etc/servers/server2.cfg
```

__2. Add following text.

```
define host {
    use                linux-server
    host_name          yourhost
    alias              My Apache server
    address            10.132.234.52
    max_check_attempts 5
    check_period       24x7
    notification_interval 30
    notification_period 24x7
}
```

}

Notice you can specify host_name, alias, and address. You can also specify other properties, such as, max_check_attempts, check_period, notification_interval, and notification_period.

You have a single server setup provided as part of the course so you won't be able to add more hosts.. Here you are just seeing how a configuration file for additional hosts will look like.

__ 3. Press Ctrl+O to save the file and hit enter.

__ 4. Press Ctrl+X to exit to the terminal.

__ 5. Reload the configuration.

```
sudo service nagios reload
```

__ 6. Launch firefox from the task bar and access Nagios web user interface by entering following URL.

```
http://localhost:1080/nagios
```

__ 7. Enter "wasadmin" both in user name and password if prompt.

__ 8. On left side of the page, click "Hosts".

Notice it shows a page like this:

The screenshot shows the Nagios web interface at <http://localhost:1080/nagios/>. The interface includes a sidebar with navigation links, a top status bar, and a main content area with summary tables and a detailed host list.

Current Network Status
Last Updated: Thu Jul 21 05:30:26 MDT 2016
Updated every 90 seconds
Nagios® Core™ 4.1.1 - www.nagios.org
Logged in as wasadmin

Host Status Totals

Up	Down	Unreachable	Pending
1	1	0	0

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
7	0	0	1	0

Host Status Details For All Host Groups

Host	Status	Last Check	Duration	Status Information
localhost	UP	07-21-2016 05:27:23	1d 3h 27m 10s	PING OK - Packet loss = 0%, RTA = 0.05 ms
yourhost	DOWN	07-21-2016 05:29:52	0d 0h 0m 4s	PING CRITICAL - Packet loss = 100%

On the page you can see "yourhost" server is down. Refresh the page until the status is updated.

Part 8 - Monitoring Services on Host

In this part you will mimic a second host by reusing 127.0.0.1 address as a different host.

__ 1. In the web browser type in following URL.

```
http://localhost:1080/nagios
```

__ 2. On left side of the page, click "Services".

Notice it shows up as follows

Limit Results:

Host	Service	Status	Last Check	Duration	Attempt	Status Information
localhost	Current Load	OK	07-21-2016 07:33:36	1d 5h 34m 34s	1/4	OK - load average: 0.27, 0.35, 0.43
	Current Users	OK	07-21-2016 07:36:42	1d 5h 33m 56s	1/4	USERS OK - 4 users currently logged in
	HTTP	OK	07-21-2016 07:36:06	1d 5h 33m 19s	1/4	HTTP OK: HTTP/1.1 301 Moved Permanently - 374 bytes in 0.001 second response time
	PING	OK	07-21-2016 07:37:21	1d 5h 32m 41s	1/4	PING OK - Packet loss = 0%, RTA = 0.04 ms
	Root Partition	OK	07-21-2016 07:34:14	1d 5h 32m 4s	1/4	DISK OK - free space: / 19300 MB (62% inode=80%):
	SSH	CRITICAL	07-21-2016 07:35:28	1d 5h 31m 26s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	07-21-2016 07:36:43	1d 5h 30m 49s	1/4	SWAP OK - 100% free (8676 MB out of 8676 MB)
	Total Processes	OK	07-21-2016 07:34:45	1d 5h 30m 11s	1/4	PROCS OK: 117 processes with STATE = RSZDT

Services are Nagios plugins and the status is displayed in Status Information column

__3. Review localhost configuration file.

```
sudo nano /usr/local/nagios/etc/objects/localhost.cfg
```

Notice it defines the "localhost" as host and it also contains "define service" entries where it's executing the plugins.

__4. Press Ctrl+X to exit to the terminal.

__5. Review commands configuration file.

```
sudo nano /usr/local/nagios/etc/objects/commands.cfg
```

Notice it defines various plugins that can be executed as commands

__6. Press Ctrl+X to exit to the terminal.

__7. Duplicate localhost configuration file to mimic different servers.

```
sudo cp /usr/local/nagios/etc/objects/localhost.cfg
/usr/local/nagios/etc/servers/server2.cfg
```

```
sudo cp /usr/local/nagios/etc/objects/localhost.cfg
/usr/local/nagios/etc/servers/server3.cfg
```

__8. Edit server2.cfg.

```
sudo nano /usr/local/nagios/etc/servers/server2.cfg
```

__9. In "define host" section, change host_name and alias to "Server2" (without quotes).

- __10. Remove the entire "define hostgroup" section.
- __11. Press Ctrl+\ to access search & replace option.
- __12. In "Search (to replace)" enter "localhost" (without quotes).
- __13. In "replace with" enter "Server2" (without quotes).
- __14. Press "A" to replace all occurrences.
- __15. Press Ctrl+O to save the file and hit enter.
- __16. Press Ctrl+X to exit to the terminal.
- __17. Edit server3.cfg.

```
sudo nano /usr/local/nagios/etc/servers/server3.cfg
```

- __18. In "define host" section, change host_name and alias to "Server3" (without quotes).
- __19. Remove the entire "define hostgroup" section.
- __20. Press Ctrl+\ to access search & replace option.
- __21. In "Search (to replace)" enter "localhost" (without quotes).
- __22. In "replace with" enter "Server3" (without quotes).
- __23. Press "A" to replace all occurrences.
- __24. Press Ctrl+O to save the file and hit enter.
- __25. Press Ctrl+X to exit to the terminal.
- __26. Reload Nagios configuration.







```
sudo service nagios reload
```

- __27. In the web browser type in following URL.

```
http://localhost:1080/nagios
```

- __28. On left side of the page, click "Services".

Notice it shows localhost, Server2, and Server3 as shown below. If the status shows "PENDING" then wait for 2-3 minutes and refresh the page.

Host ★★	Service ★★	Status ★★	Last Check ★★	Duration ★★	Attempt ★★	Status Information
Server2	Current Load	OK	07-21-2016 07:55:03	0d 0h 7m 50s	1/4	OK - load average: 0.13, 0.21, 0.28
	Current Users	OK	07-21-2016 07:56:36	0d 0h 6m 17s	1/4	USERS OK - 4 users currently logged in
	HTTP 	OK	07-21-2016 07:53:10	0d 0h 4m 43s	1/4	HTTP OK: HTTP/1.1 301 Moved Permanently - 374 bytes in 0.000 second response time
	PING	OK	07-21-2016 07:53:08	0d 0h 4m 45s	1/4	PING OK - Packet loss = 0%, RTA = 0.05 ms
	Root Partition	OK	07-21-2016 07:55:22	0d 0h 7m 31s	1/4	DISK OK - free space: / 19378 MB (62% inode=80%):
	SSH 	CRITICAL	07-21-2016 07:54:55	0d 0h 5m 58s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	07-21-2016 07:53:28	0d 0h 4m 25s	1/4	SWAP OK - 100% free (8676 MB out of 8676 MB)
	Total Processes	OK	07-21-2016 07:54:13	0d 0h 8m 40s	1/4	PROCS OK: 114 processes with STATE = RSZDT
Server3	Current Load	OK	07-21-2016 07:56:13	0d 0h 1m 40s	1/4	OK - load average: 0.15, 0.22, 0.28
	Current Users	OK	07-21-2016 07:57:15	0d 0h 0m 38s	1/4	USERS OK - 4 users currently logged in
	HTTP 	PENDING	N/A	0d 0h 2m 42s+	1/4	Service check scheduled for Thu Jul 21 07:58:18 MDT 2016
	PING	PENDING	N/A	0d 0h 2m 42s+	1/4	Service check scheduled for Thu Jul 21 07:59:20 MDT 2016
	Root Partition	OK	07-21-2016 07:56:37	0d 0h 1m 16s	1/4	DISK OK - free space: / 19378 MB (62% inode=80%):
	SSH 	CRITICAL	07-21-2016 07:57:25	0d 0h 1m 28s	2/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	07-21-2016 07:57:28	0d 0h 0m 25s	1/4	SWAP OK - 100% free (8676 MB out of 8676 MB)
	Total Processes	PENDING	N/A	0d 0h 2m 42s+	1/4	Service check scheduled for Thu Jul 21 07:58:30 MDT 2016
localhost	Current Load	OK	07-21-2016 07:53:36	1d 5h 54m 37s	1/4	OK - load average: 0.10, 0.24, 0.30
	Current Users	OK	07-21-2016 07:56:42	1d 5h 53m 59s	1/4	USERS OK - 4 users currently logged in
	HTTP 	OK	07-21-2016 07:56:06	1d 5h 53m 22s	1/4	HTTP OK: HTTP/1.1 301 Moved Permanently - 374 bytes in 0.000 second response time
	PING	OK	07-21-2016 07:57:21	1d 5h 52m 44s	1/4	PING OK - Packet loss = 0%, RTA = 0.04 ms
	Root Partition	OK	07-21-2016 07:54:14	1d 5h 52m 7s	1/4	DISK OK - free space: / 19378 MB (62% inode=80%):
	SSH 	CRITICAL	07-21-2016 07:55:28	1d 5h 51m 29s	4/4	connect to address 127.0.0.1 and port 22: Connection refused

29. Close the Terminal and web browser.

Part 9 - Review

In this lab you installed, configured, and used Nagios to monitor servers and services.

Lab 9 - Getting Started with Docker

Docker is an open IT automation platform widely used by DevOps, and in this lab, we will review the main Docker commands. In this lab, you will install Docker and use its basic commands. You will also create a custom image by creating a Dockerfile.

Part 1 - Setting the Stage

In this Lab you will work with the WA2675 VM Image. Check with your instructor if you cannot find this VM.

- ___ 1. Open a new Terminal window by clicking **Applications > Terminal**.
- ___ 2. Switch the logged-in user to **root**:

```
sudo -i
```

When prompted for the logged-in user's password, enter **wasadmin**

- ___ 3. Enter the following command:

```
whoami
```

You should see that you are **root** now.

```
root
```

Note: Now that you are **root** on your Lab server, beware of system-wrecking consequences of issuing a wrong command.

- ___ 4. Enter the following command:

```
cd /home/wasadmin/Works
```

You should see that your system prompt has changed to

```
[root@LabServer Works]#
```

- ___ 5. Get directory listing:

```
ls
```

Notice there is SimpleGreeting-1.0-SNAPSHOT.jar file. You will use this file later in this lab. It will be deployed in a custom Docker image and then you will create a container based on that image.

Part 2 - Learning the Docker Command-line

Get quick information about Docker by running it without any arguments.

__1. Run the following command:

```
docker | less
```

__2. Navigate through the scrollable output using your arrow keys and review Docker's commands.

__3. Enter q to exit.

The commands list is shown below for you reference.

attach	Attach to a running container
build	Build an image from a Dockerfile
commit	Create a new image from a container's changes
cp	Copy files/folders from a container's filesystem to the host path
create	Create a new container
diff	Inspect changes on a container's filesystem
events	Get real time events from the server
exec	Run a command in a running container
export	Stream the contents of a container as a tar archive
history	Show the history of an image
images	List images
import	Create a new filesystem image from the contents of a tarball
info	Display system-wide information
inspect	Return low-level information on a container or image
kill	Kill a running container
load	Load an image from a tar archive
login	Register or log in to a Docker registry server
logout	Log out from a Docker registry server
logs	Fetch the logs of a container
port	Lookup the public-facing port that is NAT-ed to PRIVATE_PORT
pause	Pause all processes within a container
ps	List containers
pull	Pull an image or a repository from a Docker registry server
push	Push an image or a repository to a Docker registry server
rename	Rename an existing container
restart	Restart a running container
rm	Remove one or more containers
rmi	Remove one or more images
run	Run a command in a new container
save	Save an image to a tar archive
search	Search for an image on the Docker Hub
start	Start a stopped container
stats	Display a live stream of one or more containers' resource usage statistics
stop	Stop a running container
tag	Tag an image into a repository
top	Lookup the running processes of a container
unpause	Unpause a paused container
version	Show the Docker version information
wait	Block until a container stops, then print its exit code

__4. You can get command-specific help by using the **--help** flag added to the commands invocation line, e.g. to list containers created by Docker (the command is called **ps**), use the following command:

```
docker ps --help
```

More information on Docker's command-line tools can be obtained at

Part 3 - Run the "Hello World!" Command on Docker

Let's check out what OS images are currently installed on your Lab Server.

__1. Enter the following command:

```
docker images
```

Notice there are a few images in the VM. You will use them in various labs.

One of the images is ubuntu:12.04.

__2. Enter the following command:

```
docker run ubuntu:12.04 echo 'Yo Docker!'
```

Notice it displays Yo Docker! message

So, what happened?

docker run executes the command that follows after it on a container provisioned on-the-fly for this occasion.

When the Docker command completes, you get back the command prompt, the container gets stopped and Docker creates an entry for that session in the transaction history table for your reference.

Part 4 - List and Delete Container

In this lab part, we will need a second terminal also running a shell as **root**.

__1. Open a new terminal, expand it width wise (horizontally) to capture the output of the *docker ps* command we are going to run into it. Also make sure it does not completely overlap the original terminal window.

We will be referring to this new terminal as **T2**; the original terminal will be referred to as **T1**.

__2. In the new terminal (**T2**) become **root** (use the **sudo -i** command).

__3. Change to the ~/Works directory:

```
cd /home/wasadmin/Works
```

__4. Enter the following command:

```
docker ps
```

You should see an empty container table listing only the captions part of the table.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

__5. To see a list of all containers, whether active or inactive, run the following command:

```
docker ps -a
```

__6. Switch to the original terminal window (**T1**).

__7. Enter the following command and replace the container id with the one from T2:

```
docker rm <container id>
```

Note: You may only need to type the first two or three characters of the container id and then press the **Tab** key - Docker will go ahead and auto-complete it.

The container id shown to the user is, actually, the first 12 bytes of a 64 hexadecimal character long hash used by Docker to create unique ids for images and containers.

The **docker ps** command only shows the running containers; if you repeat the **docker ps** command now after you have killed the container process, it will show the empty table.

In order to view all the containers created by docker with stopped ones stashed in the history table, use the **-a** flag with this command.

__8. Switch to the **T2** terminal.

__9. Enter the following command:

```
docker ps
```

Now it should show an empty table.

Part 5 - Working with Containers

- __1. Switch to the **T1** terminal.
- __2. Enter the following command:

```
docker run -it --hostname basic_host ubuntu:12.04 /bin/bash
```

This command will create a container from the *ubuntu* OS image, it will give the instance of the container the hostname of **basic_host**, launch the *bash* program on it and will make the container instance available for interactive mode (**i**) over allocated pseudo TTY (**t**).

After running the above command, you should be dropped at the prompt of the **basic_host** container.

```
root@basic_host:/#
```

As you can see, you are **root** there (symbolized by the '#' prompt sign).

- __3. Enter the following command to stop the container and exit out to the host OS:

```
exit
```

You should be placed back in the root's *Works* folder.

- __4. Enter the following command to see the containers:

```
docker ps -a
```

You should see the status as Exited.

- __5. Enter the following command to restart the container:

```
docker start <container id>
```

__6. Connect to the container:

```
docker exec -it <container id> /bin/bash
```

__7. Enter the following command:

```
exit
```

You should be logged off and placed back in the root's *Works* folder.

__8. Enter the following command:

```
docker stop <container id>
```

You should see that the container is not active any more.

__9. Enter the following command:

```
docker ps -a
```

You should see that the container is listed in the transaction history.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
aed47e954acd	ubuntu:12.04	"/bin/bash"	23 minutes ago	Exited
(0) 4 minutes ago	prickly_nobel			
...				

Part 6 - Create a Custom Image

Now that we have ourselves a container (which is currently in the **exited / stopped** status), let's create a custom image based on it.

__1. Enter the following command providing your container id (*aed47e954acd*, in our case):

```
docker commit <container id> my_server:v1.0
```

You should get back the OS image id generated by Docker.

__2. Enter the following command:

```
docker images
```

You should see the new image listed on top of the available images in our local image repository.

__3. Enter the following command:

```
docker run -it my_server:v1.0 /bin/bash
```

This command will start a new container from the custom image we created in our local image repository.

__ 4. Enter the following command at the container prompt:

```
exit
```

You will be dropped back at the Lab Server's prompt.

Part 7 - Workspace Clean-Up

__ 1. Enter the following command:

```
docker ps -a
```

This command will show all the containers and not only the running ones.

It is always a good idea to clean-up after yourself, so we will remove all the containers we created so far.

__ 2. Enter the following command for every listed container id:

NOTE: Do NOT delete any image or container other than the ones you have created in this lab. Leave OpenShift and other images / containers as is.

```
docker rm <container id>
```

__ 3. Verify there are no docker containers:

```
docker ps -a
```

__ 4. Remove the *my_server:v1.0* image we created and persisted in our local image repository:

```
docker rmi my_server:v1.0
```

__ 5. Verify your image has gone:

```
docker images
```

Part 8 - Create a Dockerfile for Building a Custom Image

In this part you will download Ubuntu docker image and create a Dockerfile which will build a custom Ubuntu based image. It will automatically update the APT repository, install JDK, and copy the SimpleGreeting.jar file from host machine to the docker image.

You will also build a custom image by using the Dockerfile.

__1. In the terminal, type following command to create Dockerfile:

```
gedit Dockerfile
```

__2. Type in following code:

```
# lets use ubuntu docker image
FROM dongjoon/ubuntu12.04-jdk8

# RUN apt-get update -y
# RUN apt-get install openjdk-7-jdk -y

# deploy the jar file to the container
COPY SimpleGreeting-1.0-SNAPSHOT.jar /root/SimpleGreeting-1.0-
SNAPSHOT.jar
```

Note: The Dockerfile creates a new image based on Ubuntu 12.04, updates the APT repository, installs JDK, and copies host's /home/wasadmin/Works/SimpleGreeting-1.0-SNAPSHOT.jar to the image under the root directory.

__3. Click **Save** button.

__4. Close gedit. You will see some errors on the Terminal, ignore them.

__5. Type **ls** and verify your new file is there.

__6. Run following command to build a custom image:

```
docker build -t dev-ubuntu:v1.0 .
```

This command builds / updates a custom image named dev-ubuntu:v1.0. Don't forget to add the period at the end of docker build command.

Notice, Docker created the custom image with JDK installed in it and the jar file deployed in the image. The first time you build the job, it will be slow since the image will get built for the first time. Subsequent runs will be faster since image will just get updated, not rebuilt from scratch.

Part 9 - Verify the Custom Image

In this part you will create a container based on the custom image you created in the previous part. You will also connect to the container, verify the jar file exists, and execute the jar file.

__ 1. In the terminal, run following command to verify the custom image exists:

```
docker images
```

Notice dev-ubuntu:v1.0 is there

__ 2. Create a container based on the above image and connect to it:

```
docker run --name dev --hostname dev -it dev-ubuntu:v1.0 /bin/bash
```

Note: You are naming the container dev, hostname is also dev, and it's based on your custom image dev-ubuntu:v1.0

__ 3. Switch to the root directory in the container:

```
cd /root
```

__ 4. Get the directory list:

```
ls
```

Notice SimpleGreeting*.jar file is there.

__ 5. Execute the jar file:

```
java -cp SimpleGreeting-1.0-SNAPSHOT.jar com.simple.Greeting
```

Notice it displays the following message:

root@dev:~# java -cp SimpleGreeting-1.0-SNAPSHOT.jar com.simple.Greeting
GOOD

__ 6. Exit out from the container to the command prompt:

```
exit
```

__7. Get active docker container list:

```
docker ps
```

Notice there's no active container.

__8. Get list of all docker containers:

```
docker ps -a
```

Notice there's one inactive container named dev.

Part 10 - Cleanup Docker

In this part you will clean up docker by removing containers and images.

__1. In the terminal, run following to get list of all containers:

```
docker ps -a
```

__2. In case, if there are any containers, stop (if running) and remove them by running following commands. Don't remove the ones you haven't created yourself in this lab:

```
docker stop <container>  
docker rm <container>
```

__3. Get list of all docker containers. It should be empty:

```
docker ps -a
```

__4. Get docker image list:

```
docker images
```

__5. Remove all images that you created by executing following command:

```
docker rmi <REPOSITORY:TAG>
```

e.g. `docker rmi dev-ubuntu:v1.0`

__6. Make sure your image has been deleted:

`docker images`

__7. In each Terminal type **exit** and then close the Terminal window.

Part 11 - Review

In this lab, we reviewed the main Docker command-line operations.

Lab 10 - Getting Started with Kubernetes

Kubernetes is a an open-source container orchestration solution. It is used for automating deployment, scaling, and management of containerized applications. In this lab, you will explore the basics of Kubernetes. You will use minikube, which allows you to create a Kubernetes environment with ease. In later labs, you will use a more full blown solution by utilizing OpenShift Origin.

This Lab requires Internet connection to work. If you don't have internet then watch the video.

Part 1 - Setting the Stage

In this Lab you will work with the WA2675 VM Image. Check with your instructor if you cannot find this VM.

__ 1. Open a new Terminal window by clicking **Applications > Terminal**.

__ 2. Switch the logged-in user to **root**:

```
sudo -i
```

When prompted for the logged-in user's password, enter **wasadmin**

__ 3. Enter the following command:

```
whoami
```

You should see that you are **root** now.

```
root
```

Note: Now that you are **root** on your Lab server, beware of system-wrecking consequences of issuing a wrong command.

Part 2 - Ensure you can run minikube and kubectl

In this part you will ensure you can run minikube and kubectl.

__ 1. Add /usr/local/bin to the path environment variable:

```
export PATH=$PATH:/usr/local/bin
```

__ 2. Verify you can execute minikube:

```
minikube version
```

__ 3. Get minikube help:

```
minikube -help
```

__4. Verify you can execute kubectl and also obtain Kubernetes version:

```
kubectl version
```

Notice it lists major and minor version in JSON format. Don't worry about the connection message.

__5. Find out if this version of minikube supports the installed version of Kubernetes:

```
minikube get-k8s-versions
```

Verify the Kubernetes version is listed. Don't worry about the connection message.

__6. Get Kubernetes cluster information:

```
kubectl cluster-info
```

Notice it displays the IP address and port where the Kubernetes master is running. Don't worry about the connection message.

Part 3 - Start the Cluster

In this part you will start the Kubernetes cluster and interact with it in various ways.

__1. Run following command to start a cluster:

```
minikube start
```

Note: By default it runs a node in a VirtualBox VM. If you want to use a different virtualization solution, then you can pass it as an argument to the command like this:

```
minikube start --vm-driver="<driver>"
```

e.g. virtualbox, hyper-v, ...

__2. Wait until it switches back to the terminal.

Note: The above command performs following operations:

1. Generates the certificates and then proceeds to provision a local Docker host. This results in a VM created inside of VirtualBox.
2. The host is provisioned with the boot2Docker ISO image.
3. IP address is assigned to it.
4. Displays a message that kubectl is configured to talk to your local Kubernetes cluster.

__3. Switch back to the terminal and run following command:

```
minikube status
```

Notice it shows messages like this:

```
[root@localhost ~]# minikube status
minikube: Running
localkube: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100
```

__4. Run following command to obtain the cluster IP address:

```
minikube ip
```

Notice it shows the IP address of our cluster.

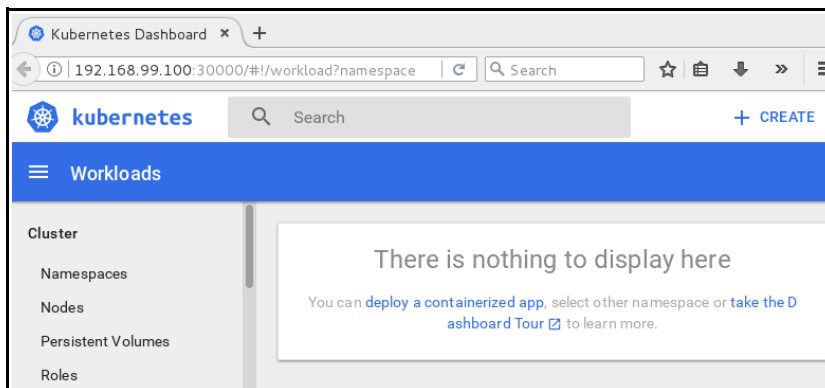
Part 4 - View Kubernetes Dashboard

In this part you will view Kubernetes dashboard and interact with it.

__1. In the terminal, run following command to view the dashboard URL:

```
minikube dashboard
```

__2. It will open Firefox web browser and show the Kubernetes page.



Notice that the Terminal is running, if you close the Terminal or terminate the command the browser will close, so keep it open.

__3. On left side of the dashboard, click **Nodes**.

Notice the page looks like this: (minikube is the name of your node. You also noticed this name in VirtualBox as a VM instance).

kubernetes		Nodes			+ CREATE
Admin					
Namespaces					
Nodes					
Persistent Volumes					
		Name	Labels	Ready	Age
		minikube	beta.kubernetes.io/arch=amd64 beta.kubernetes.io/os=linux kubernetes.io/hardware-id=...	True	42 minutes

__ 4. Open a new terminal by clicking **Applications > Terminal**.

For the next steps you will use this Terminal.

__ 5. Switch the logged-in user to **root**:

```
sudo -i
```

Enter wasadmin as password.

__ 6. In the terminal, run following commands to view nodes:

```
export PATH=$PATH:/usr/local/bin
kubectl get nodes
```

Notice you can see the node in the terminal. It's the same information you saw in the dashboard earlier in this part of the lab.

In case if you encounter any error message while obtaining the nodes, run following command to fix it.

```
kubectl config use-context minikube
```

This command tells kubectl what cluster to connect to.

Part 5 - Create a Container

In this part you will create a container and run it in the node / cluster i.e. you will run a workload in the cluster.

__ 1. Run following command:

```
kubectl run my-web-server --image=nginx --port=80
```

It uses nginx image to create a container named my-web-server and exposes the service on port 80. If the image is not available on your local machine, it connects to the Docker hub registry. You can specify other registries by specifying the full URL to the image.

You may get an AlreadyExist server error, if so continue with the Lab.

In case if it container's image doesn't get downloaded properly, delete the cached data located under `~/minikube/cache` and retry.

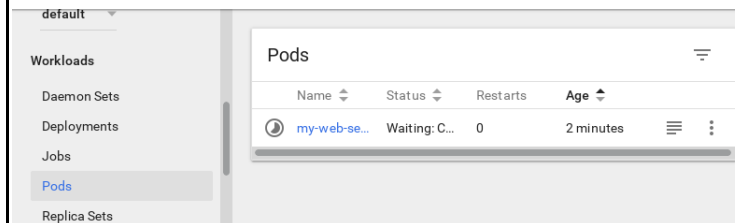
__ 2. Run following command to view pod list:

```
kubectl get pods
```

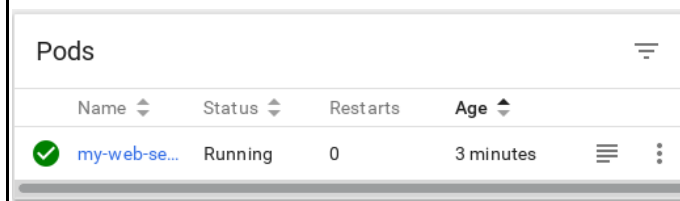
Notice it shows my-web-server pod. Under READY, 0/1 means there's one container running the nginx service and it's still created. You might have to wait for a minute or two to see the "running" status.

__3. Switch to the Firefox window, where dashboard is open, and click **Pods** on left side of the page.

Notice the pod is listed like this:



When the pod is created completely, it would show up like this:



Part 6 - View Logs and Details

In this part you will view logs and details in various ways.

__1. On the dashboard, click **my-web-server** pod.

Notice it shows various pod details, such as creation date, status, and other events.

__2. Make a note of the pod name under **Details** on the dashboard.

It would be something like this: my-web-server-3913049308-qmcwt

__3. In the terminal run following command to view pod details:

```
kubectl describe pod <pod_name>
```

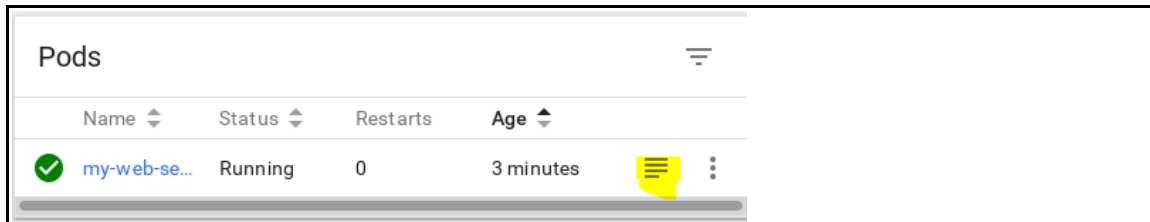
Notice you get to view similar details in the terminal as you saw in the previous steps of this part.

__4. Run following command to view node details:

```
kubectl describe node minikube
```

Notice it shows the node details.

5. On the dashboard, click **Pods** on left side of the page, then click the **Logs** icon next to the pod.



Name	Status	Restarts	Age
my-web-se...	Running	0	3 minutes

Notice it opens another tab and shows the pod log. Currently, the log is empty. It shows up like this:

Logs from my-web-server in my-web-server-3913049308-qmcwt

The selected container has not logged any messages yet.

You will view the log again, after accessing the nginx service.

6. Close the tab where the log is displayed and go back to the tab where the dashboard is open.

Part 7 - Expose a Service

In this part you will expose nginx service, which you deployed in the previous parts of this lab.

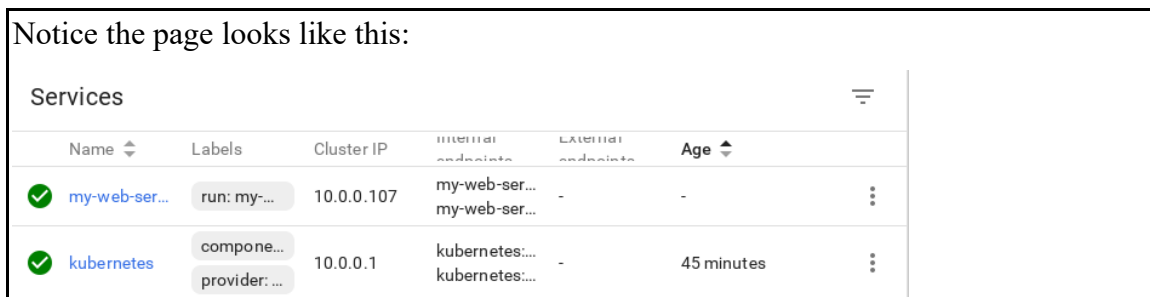
1. In the terminal, run following command:

```
kubectl expose deployment my-web-server --type=NodePort
```

Notice it shows a message that service has been exposed.

2. In the Firefox web browser tab where dashboard is open, click **Services** on left side of the page.

Notice the page looks like this:



Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
my-web-ser...	run: my-...	10.0.0.107	my-web-ser... my-web-ser...	-	-
kubernetes	compon... provider: ...	10.0.0.1	kubernetes:... kubernetes:...	-	45 minutes

Note: The IP address is the internal IP address. You will access the public IP address later in the lab.

3. In the terminal run following command to view the exposed services:

```
kubectl get services
```

__ 4. View my-web-server service details:

```
kubectl describe service my-web-server
```

__ 5. In the terminal, run following command to access the service's public IP address:

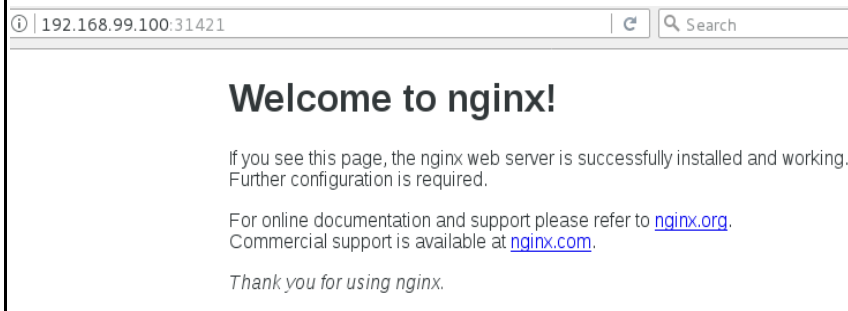
```
minikube service my-web-server --url=true
```

Notice it shows IP address like this: `http://192.168.99.100:31421`

__ 6. Ctrl+Click the URL in the terminal. It will launch the page in Firefox web browser.

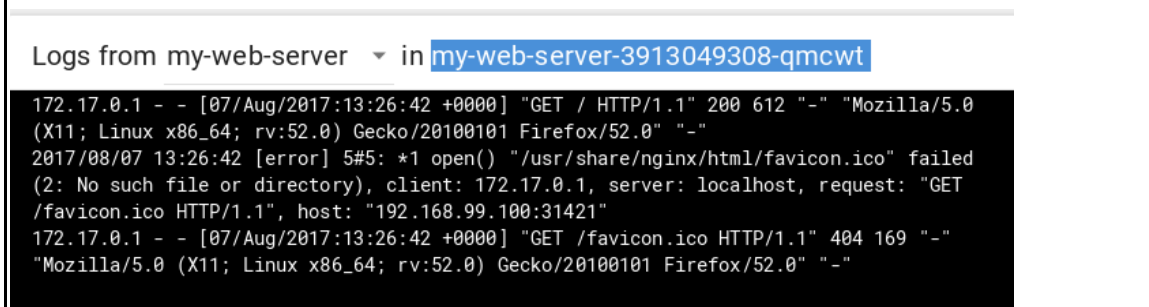
Alternatively, you can type in the URL manually in Firefox.

Notice it shows the familiar looking Nginx home page.



__ 7. On the dashboard, click **Services**, then click your service, and click Log icon next to the service.

Notice it opens another tab in the web browser and shows the service access log like this:



__ 8. Make a note of the full service name, as shown in the above picture.

__ 9. In the terminal run following command:

```
kubectl logs <full_service_name>
```

Notice it shows the service access log in the terminal like this:


```
[root@localhost ~]# kubectl logs my-web-server-3913049308-qmcwt
172.17.0.1 - - [07/Aug/2017:13:26:42 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
2017/08/07 13:26:42 [error] 5#5: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 172.17.0.1, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "192.168.99.100:31421"
172.17.0.1 - - [07/Aug/2017:13:26:42 +0000] "GET /favicon.ico HTTP/1.1" 404 169 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
```

Part 8 - Scaling the Services

In the previous parts of the lab, you exposed a service. There was a single instance of the service, with one Pod that was provisioned on a single node. In this part you will scale the service by having 3 Pods.

__1. In the terminal, run following command:

```
kubectl get deployment
```

Notice it shows result like this:

```
[root@localhost ~]# kubectl get deployment
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-web-server 1         1         1            1           1h
```

Notice there's a single instance running right now.

__2. Run following command to scale the service:

```
kubectl scale --replicas=3 deployment/my-web-server
```

__3. Run following command to get the service instance count:

```
kubectl get deployment
```


Notice now it shows result like this:

```
[root@localhost ~]# kubectl get deployment
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-web-server 3         3         3            3           1h
```

__4. On the dashboard page, in the web browser, click **Deployments** on left side of the page.

Notice it shows 3/3 Pods.

Deployments

Name	Labels	Pods
 my-web-server	run: my-web-...	3 / 3

__5. On the dashboard page, click **Services** on left side of the page, then click **my-web-server** service.

Notice it shows Pods like these:

Pods					
Name	Status	Restarts	Age		
✓ my-web-server-3913049308-6rq...	Running	0	-	☰	⋮
✓ my-web-server-3913049308-qm...	Running	0	an hour	☰	⋮
✓ my-web-server-3913049308-rjcw4	Running	0	-	☰	⋮

___ 6. On left side of the page, click **Pods**.

Notice it shows Pod list like this

Pods					
Name ↕	Status ↕	Restarts	Age ↕		
✓ my-web-server-3913049308-6rqn7	Running	0	-	☰	⋮
✓ my-web-server-3913049308-rjcw4	Running	0	-	☰	⋮
✓ my-web-server-3913049308-qmcwt	Running	0	an hour	☰	⋮

Part 9 - Stop and Delete the Cluster

In this part you will stop and delete the cluster you created in this lab.

___ 1. In the terminal, run following command to stop the cluster:

```
minikube stop
```

___ 2. Run following command to delete the cluster:

```
minikube delete
```

___ 3. Close all open browsers.

___ 4. Type exit in each Terminal and then close it.

Part 10 - Review

In this lab, you learned the basics of Kubernetes with minikube and kubectl.

Lab 11 - Getting Started with OpenShift

RedHat: OpenShift Origin is a container application platform that brings docker and Kubernetes to the enterprise. In this lab, you will utilize OpenShift CLI tools and also use the web UI to create a simple project. You will deploy a Docker image which contains "Hello world" service application. You will also see how to manage pods, service applications, and scale the application. Finally, you will delete the service application and the deployment configuration.

This Lab requires Internet connection to work. If you don't have internet then the instructor will provide a recorded video.

Part 1 - Setting the Stage

In this Lab you will work with the WA2675 VM Image. Check with your instructor if you cannot find this VM.

- __1. Open a new Terminal window by clicking **Applications > Terminal**.
- __2. Switch the logged-in user to **root**:

```
sudo -i
```

When prompted for the logged-in user's password, enter **wasadmin**

- __3. Enter the following command:

```
whoami
```

You should see that you are **root** now.

```
root
```

Note: Now that you are **root** on your Lab server, beware of system-wrecking consequences of issuing a wrong command.

- __4. Switch to the home directory:

```
cd ~
```

Part 2 - Connect to OpenShift

In this part you will use "oc" command line tool to connect to OpenShift and perform various operations.

__1. Enter these commands:

```
setenforce 0
systemctl stop firewalld
systemctl disable firewalld
```

__2. Enter these commands:

```
dhclient -r
dhclient -v
```

__3. Start OpenShift.

```
oc cluster up
```

Note: In case, if you encounter IP address / port based error, you might have to renew the IP address by using following commands.

```
dhclient -r && dhclient -v
```

Then run oc cluster up again.

__4. Find user you are connected to OpenShift Origin as:

```
oc whoami
```

It will most likely show you **developer**.

If you get an error that is enable to connect then run these commands:

```
dhclient -r
dhclient -v
oc whoami
```

__5. Let's connect as system:admin account

```
oc login -u system:admin
```

__6. Verify you are connected as system:admin

```
oc whoami
```

__7. Get projects list:

```
oc projects
```

__8. Make myproject the active project:

```
oc project myproject
```

__9. Get status:

```
oc status
```

Notice it shows the project name along with server URL. It also tells you that no services or apps are currently deployed.

Part 3 - Deploy a new Application

In this part you will deploy a simple hello-world style application. The container image is already available in your VM.

__1. In the terminal, run following command to view existing docker images:

```
docker images
```

Notice: It shows docker.io/openshift/hello-openshift as one of the images.

Note: openshift/hello-openshift is a Docker image hosted at <http://hub.docker.com>. It has already been pulled into the VM. It's a simple image of a Docker container which when deployed and executed, displays a simple greeting message. In the next lab, you will see how to create your custom images hosting services.

__2. Create a new OpenShift app from the docker image:

```
oc new-app --docker-image="docker.io/openshift/hello-openshift"
```

The output of above command looks like this:

```
[root@localhost ~]# oc new-app --docker-image="docker.io/openshift/hello-openshift"
--> Found Docker image 2055816 (2 hours old) from docker.io for "docker.io/openshift/hello-openshift"

* An image stream will be created as "hello-openshift:latest" that will track this image
* This image will be deployed in deployment config "hello-openshift"
* Ports 8080/tcp, 8888/tcp will be load balanced by service "hello-openshift"
* Other containers can access this service through the hostname "hello-openshift"
* WARNING: Image "docker.io/openshift/hello-openshift" runs as the 'root' user which may not be p

--> Creating resources ...
    imagestream "hello-openshift" created
    deploymentconfig "hello-openshift" created
    service "hello-openshift" created
--> Success
    Run 'oc status' to view your app.
```

You can also specify a full URL to some other Docker registry, such as Google. You can also specify the image details in yaml format.

___ 3. Get status:

```
oc status
```

Notice hello-openshift service and deployment configuration are displayed.

___ 4. Get pod list:

```
oc get pods
```

It might take a while before the pod READY status shows 1/1. Wait for a few seconds and run the command again until it shows 1/1.

___ 5. Open a web browser and enter following URL:

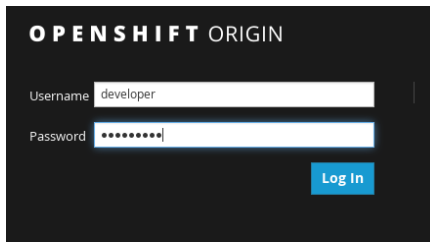
```
https://localhost:8443
```

Note: It might give you certificate error that your connect is not secure. If you encounter this error then perform following steps:

1. Click Advanced.
2. Click Add Exception.
3. Click Confirm Security Exception.

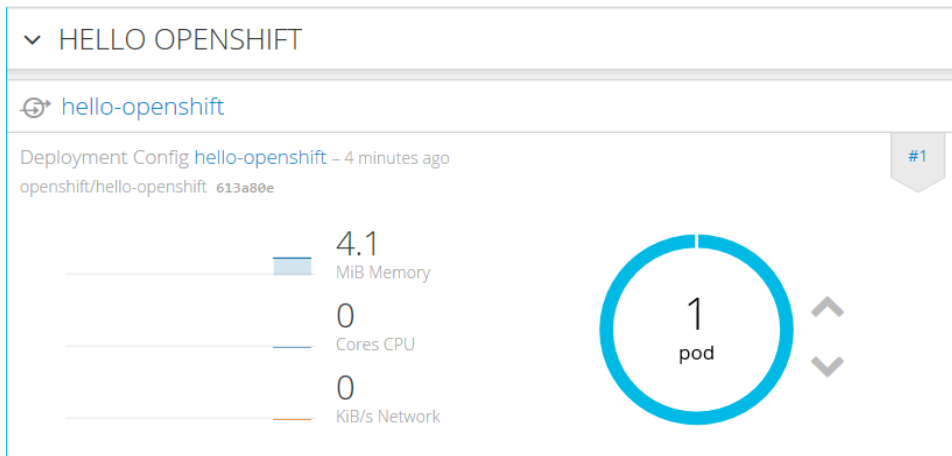
You may have to perform these steps 2 times until the login is shown.

__6. Login with developer/developer credentials.



__7. Click **My Project** project, then click **Overview** on left side of the page.

Notice, if the pod is created and running properly, it should show up like this:



Part 4 - Expose Service and Access it

In this part you will expose the "hello" service and access it via the web browser.

__1. In the terminal, run following command to see service details:

```
oc describe svc/hello-openshift
```

__2. In the web browser, on left side of the page, click **Applications > Services**.

Notice hello-openshift service is available. You can see similar details which you noticed in the previous step of this part.

__3. Click the **hello-openshift** service.

On the page you should be able to see service details and the pod which is running the service. Notice that there's no route defined for the service on port 8080, which means you can't access the service, yet.

Route		Service Port		Target Port	Hostname	TLS Termination
none	→	8080/TCP (8080-tcp)	→	8080	none	none

__4. In the terminal, run following command to expose the service:

```
oc expose svc/hello-openshift --name=hello-openshift --hostname=hello-openshift.localhost
```

Notice in the web browser, now hostname is available. It means the service the now exposed.

Route		Service Port		Target Port	Hostname
hello-openshift	→	8080/TCP (8080-tcp)	→	8080	http://hello-openshift.localhost

__5. Right click the URL under **Hostname** and click **Open Link in new tab**.

Notice Hello OpenShift! message is displayed on the page.

__6. Close the tab where Hello OpenShift! message is displayed and go back to the OpenShift management console.

__7. In the terminal run following command:

```
oc status
```

Notice it shows the hostname on the terminal like this:

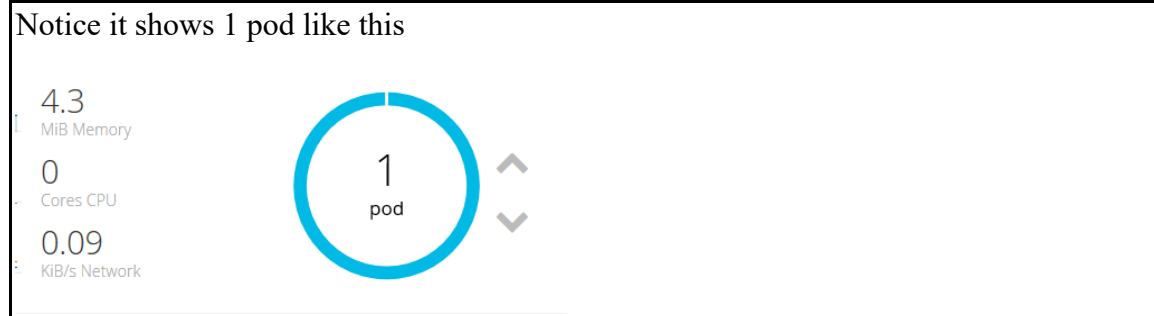
```
http://hello-openshift.localhost to pod port 8080-tcp (svc/hello-openshift)
```

It means the service is exposed.

Part 5 - Scale the Service / Application

In this part you will scale the service / application by creating multiple instances. By default, 1 instance is created.

__ 1. In the web browser, click **Overview** on left side of the page.



__ 2. Click the up arrow, next to 1 pod circle.

Notice it shows 2 pods now. It might take a while, depending on container size, to bring the second pod up and running.

__ 3. On left side of the page, click **Applications > Pods**.

Notice 2 pods are listed.

__ 4. In the terminal run following command:

```
oc get pods
```

Notice 2 pods are running.

__ 5. In the terminal, run following command:

```
oc status
```

__ 6. Make a note of deployment configuration name `dc/hello-openshift`

__ 7. Run following command to scale down back to 1 pod:

```
oc scale dc/hello-openshift --replicas=1
```

__ 8. Run following command to get pod list:

```
oc get pods
```

Notice there's 1 pod running.

__ 9. In the web browser, click **Overview**.

Notice there's 1 pod running.

Part 6 - Delete Service / Application and Deployment

In this part you will delete the hello-openshift deployment and the service / application.

__ 1. In the terminal, run following command to delete the service / application:

```
oc delete svc/hello-openshift
```

__ 2. Run following command to delete the deployment configuration:

```
oc delete dc/hello-openshift
```

__ 3. Get pod list:

```
oc get pods
```

Notice no pods are running.

__ 4. Get status:

```
oc status
```

Notice no pods are running.

__ 5. In the web browser, click **Overview**.

Notice no pods are running.

__ 6. Keep the terminal and the web browser open for the next lab.

Part 7 - Review

In this lab, you accessed OpenShift Origin by utilizing the CLI tools and the web UI. You deployed a Docker container image to it, managed the pods and service applications, scaled the service application by running multiple instances, and then cleaned up the environment by deleting the deployment configuration and the service application.

Lab 12 - CI/CD with Jenkins, Docker, and OpenShift

In this part you will implement CI/CD with Jenkins, Docker, and OpenShift. You will create a custom Docker image with a Node.js application, then create a Jenkins job which automatically deploys the image and creates a service application in OpenShift.

This Lab currently does not work with a proxy server, if you are using a proxy server then your instructor is going to demonstrate the Lab.

Part 1 - Create a Node.js Application

In this part you will create a Node.js application. Later in the lab, you will create a Docker image hosting the Node.js service.

In this Lab you will work with the WA2675 VM Image. Check with your instructor if you cannot find this VM.

__1. In the terminal, run following command:

```
mkdir -p /var/lib/jenkins/repos/hello-node
```

__2. Switch to the directory:

```
cd /var/lib/jenkins/repos/hello-node
```

__3. Initialize a new node application:

```
npm init .
```

__4. Press **Enter** key to use default value for each option, at the end enter y and hit Enter.

__5. Download express node module:

```
npm install express --save
```

Note: You will use express node module to create a simple Node.js service.

__6. Create index.js file:

```
gedit index.js
```

__7. Enter following code:

```
'use strict';

const express = require('express');

// Constants
const PORT = 9090;
const HOST = '0.0.0.0';
```

```
// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello Node.js v1.0\n');
});

app.listen(PORT, HOST);
```

__ 8. Click the **Save** button to save the file. Then close gedit.

__ 9. Run the service:

```
node index.js
```

Note. Do not stop the command.

__ 10. Open Firefox web browser and enter following URL in a new tab:

```
http://localhost:9090
```

Notice it shows Hello Node.js v1.0 message.

__ 11. Now stop the command by pressing CTRL+C.

__ 12. Close the tab in the browser.

Part 2 - Create a Docker Image

In this part you will create a docker image of your Node.js service. You already have the latest version of node image available in the VM. You will use this node image as the base image and deploy your Node.js service into it and save it as a custom image.

__ 1. Ensure you are under /var/lib/jenkins/repos/hello-node directory:

```
pwd
```

__ 2. Create Dockerfile:

```
gedit Dockerfile
```

__3. Enter following text (Note: To save time, you can skip the comments with # in front of them. They do contain some important instructions which you must perform):

```
FROM node:boron

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies. Note: Don't forget the period at the end
COPY package.json .

RUN npm install

# Bundle app source. Note: There's period space period at the end
COPY . .

EXPOSE 9090
CMD [ "node", "index.js" ]
```

__4. Save the file and close gedit.

__5. Create a .dockerignore file:

```
gedit .dockerignore
```

Note: There's a period in front of dockerignore. This file specifies files and directories which shouldn't become part of a docker image.

__6. Enter following text:

```
node_modules
npm-debug.log
```

__7. Save the file and close gedit.

__8. Run following command to build a custom image:

```
docker build -t node-app:v1.0 .
```

Make sure you add the dot at the end of the command.

__9. After the above command completes, verify the custom image exists:

```
docker images
```

Part 3 - Check in the code into Git repository

In this part you will check in the code into the Git repository. You will utilize the repository in Jenkins, later in the lab.

__1. Ensure you are under /var/lib/jenkins/repos/hello-node directory:

```
pwd
```

__2. Create .gitignore file:

```
gedit .gitignore
```

Note: There's a period in front of gitignore. In a bit, you will check in your app source code into a git repository, so you can utilize it in Jenkins. .gitignore specifies files and directories which shouldn't get checked into the repository.

__3. Enter following text:

```
node_modules  
npm-debug.log
```

__4. Save the file and close gedit.

__5. Initialize a Git repository:

```
git init .
```

__6. Stage changed files:

```
git add .
```

__7. Commit the changes:

```
git commit -m "nodejs microservice v1.0"
```

Part 4 - Manually Deploy the Custom Image and Verify the Service

In this part you will manually deploy the custom docker image and verify your custom service is working in OpenShift.

__1. Set myproject as the active project:

```
oc project myproject
```

__2. Create a new application by deploying your custom docker image:

```
oc new-app --docker-image="node-app:v1.0"
```

__3. Get status:

```
oc status
```

Notice node-app service and deployment configuration are displayed.

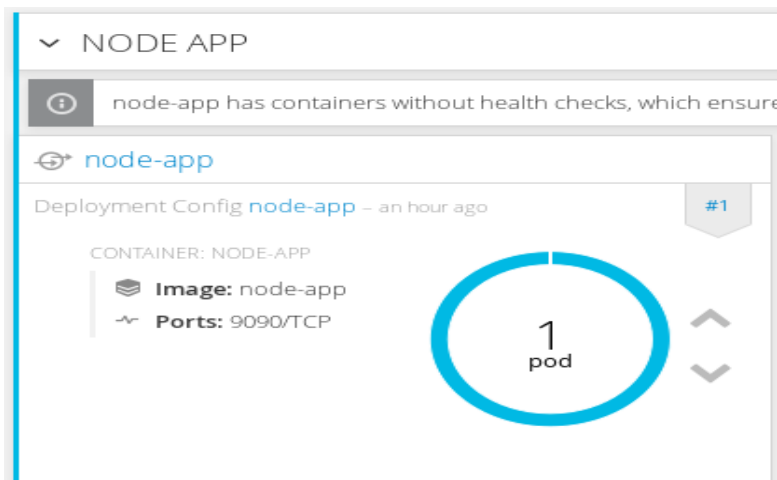
__4. Get pod list:

```
oc get pods
```

It might take a while before the pod READY status shows 1/1. Wait for a few seconds and run the command again until it shows 1/1.

__5. In the web browser, click **My Project** (if it's not already selected), then click **Overview** on left side of the page.

Notice, if the pod is created and running properly, it shows up like this:



Part 5 - Expose node-app Service and Access it

In this part you will expose node-app service and access it via the web browser.

__1. In the terminal, run following command to see service details.

```
oc describe svc/node-app
```

__2. In the web browser, on left side of the page, click **Applications > Services**.

Notice node-app service is available. You can see similar details which you noticed in the previous step of this part.

__3. Click the **node-app** service.

On the page you should be able to see service details and the pod which is running the service. Notice that there's no route defined for the service on port 9090, which means you can't access the service, yet.

__4. In the terminal, run following command to expose the service:

```
oc expose svc/node-app --name=node-app --hostname=node-app.localhost
```

Notice in the web browser, now hostname is available. It means the service is now exposed.

Route		Service Port		Target Port	Hostname
node-app	→	9090/TCP (9090-tcp)	→	9090	http://node-app.localhost

__5. Right click the URL under **HostName** and click **Open in new tab**.

Notice it shows the message Hello Node.js v.10.

__6. Keep the tab open. Later lab instructions will refer to it as "Hello Node Tab".

Part 6 - Connect to Jenkins and Configure it

In this part you will connect to Jenkins and configure it so you can create jobs which can deploy to OpenShift origin.

__1. Open a new tab in the web browser and enter following URL:

```
http://localhost:8080
```

Notice Jenkins login page shows up.

User:

Password:

☐ Remember me on this computer

__2. Enter following credentials to log in:

```
User Name: wasadmin
```


Password: wasadmin

__ 3. On left side of the page, click **Manage Jenkins**.

__ 4. Click **Configure System**.

Ignore warnings on the page.

__ 5. Under **Global properties**, select check box in front of **Environment variables**.

__ 6. Click **Add** button.

__ 7. Add a new environment variable with following values:

Name: KUBECONFIG

Value: /var/lib/origin/openshift.local.config/master/admin.kubeconfig

Note: admin.kubeconfig contains OpenShift configuration. It's required so you can execute the "oc" commands from a Jenkins job.

__ 8. Click **Save** button.

Part 7 - Create a New Jenkins Job

In this part you will implement CI/CD by creating a Jenkins job. It will checkout the Node.js code from the Git repository, build a new Docker image, delete the existing OpenShift service and deployment configuration, and redeploy the app to OpenShift.

__ 1. On left side of the Jenkins home page, click **New Item**.

__ 2. Enter **CI-CD** as the job name.

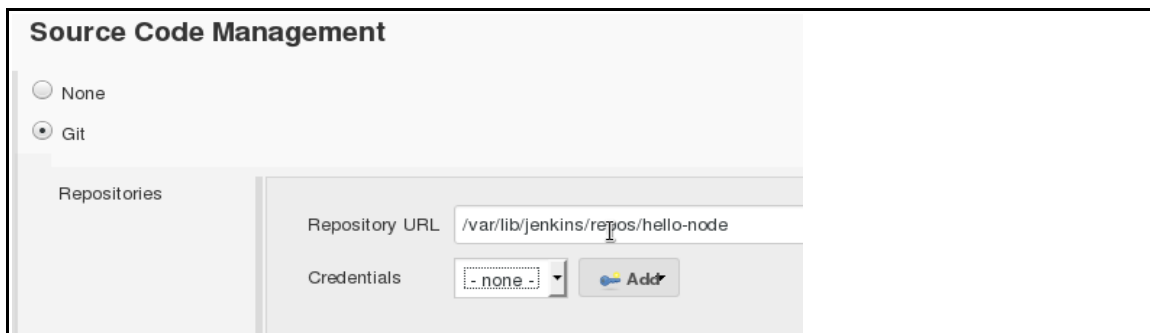
__ 3. Select **Freestyle project** type.

__ 4. Click **OK** button.

__ 5. Under **Source Code Management** select **Git**.

__ 6. In Repository URL enter following location and then hit Tab.

/var/lib/jenkins/repos/hello-node

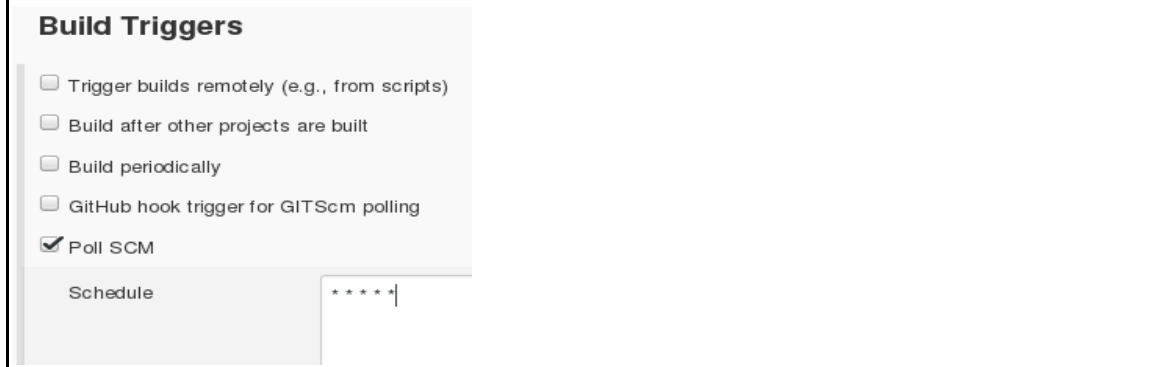


__ 7. Click **Apply** button.

__ 8. Under **Build Triggers**, select **Poll SCM** and enter following Schedule:

* * * * *

Note: There are 5 asterisks with a space in between. It checks the SCM every 1 minute and triggers a build if changes are detected in the repository.




Build Triggers

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

Schedule: *****|

__ 9. Click **Apply** button.

__ 10. Under **Build**, click **Add build step** and select **Execute shell**.



Build

Add build step ▼

- Execute Windows batch command
- Execute shell
- Invoke Ant

__ 11. Enter following in **Command** text box:

```
docker build -t node-app:v1.0 .
oc project myproject
oc delete svc/node-app
oc delete dc/node-app
oc new-app --docker-image="node-app:v1.0"
```

__ 12. Click **Save** button.

Part 8 - Test Jenkins CI/CD Job

In this part you will test Jenkins CI-CD job you created in the previous part. You will modify the Node.js code, commit changes, and verify the result.

__ 1. In the terminal you are located under /var/lib/jenkins/repos/hello-node directory.

```
pwd
```

__ 2. Edit index.js file:

```
gedit index.js
```

__ 3. Change v1.0 to v2.0

__ 4. Save the file and close gedit.

__ 5. Stage the changed files.

```
git add .
```

__ 6. Commit changes.

```
git commit -m "v2.0"
```

__ 7. In the web browser, go to the page where Jenkins CI-CD job is available.

__ 8. Wait for about a minute or so. Under **Build History** you should be able to see a new build.

__ 9. While the build is getting performed, click the build and go to **Console Output**.

In the Console Output, it should show all the operations the job is performing.

__ 10. After the Job finishes, in the web browser, go to the tab where **OpenShift** page is open and click **Overview** tab.

__ 11. If you time it properly, you should be able to see the node-app pod is getting reinitialized.

__ 12. When the pod is up and running, in the web browser, go to the Hello Node Tab and refresh the page.

Note: It should show Hello Node.js v2.0 message.

__13. In the terminal, execute following command.

```
docker images
```

Notice node-app image has been updated.

__14. Keep the terminal and the web browser open for the next part.

Part 9 - Create a New Jenkins Job - Pipeline

In this part you will implement CI/CD by creating another Jenkins job, but this time you will use the pipeline template. It will do exact same things as the previous job.

__1. On left side of the Jenkins home page, go home and then click **New Item**.

__2. Enter **CI-CD Pipeline** as the job name.

__3. Select **Pipeline** type.

__4. Click **OK** button.

__5. Under **Pipeline**, in Script text box, type in following script.

```
node {
  stage 'Checkout'
    git url: '/var/lib/jenkins/repos/hello-node'

  stage 'Build Docker Image'
    sh 'docker build -t node-app:v1.0 .'

  stage 'Activate myproject'
    sh 'oc project myproject'

  stage 'Delete Service'
    sh 'oc delete svc/node-app'

  stage 'Delete Deployment Configuration'
    sh 'oc delete dc/node-app'

  stage 'Redeploy App'
    sh 'oc new-app --docker-image="node-app:v1.0"'
}
```

Note: This script can also be typed into a file named Jenkinsfile and checked into the same repository where you have Node.js application code along with Dockerfile.

__6. Click **Save** button.

When you click Save button it will run the first build automatically. Wait for it to complete before continue.

Part 10 - Test Jenkins CI/CD Pipeline

In this part you will test Jenkins CI-CD Pipeline you created in the previous part. You will modify the Node.js code, commit changes, and verify the result.

__1. In the terminal you are located under `/var/lib/jenkins/repos/hello-node` directory:

```
pwd
```

__2. Edit `index.js` file:

```
gedit index.js
```

__3. Change `v2.0` to `v3.0`

__4. Save the file and close `gedit`.

__5. Stage the changed files:

```
git add .
```

__6. Commit changes:

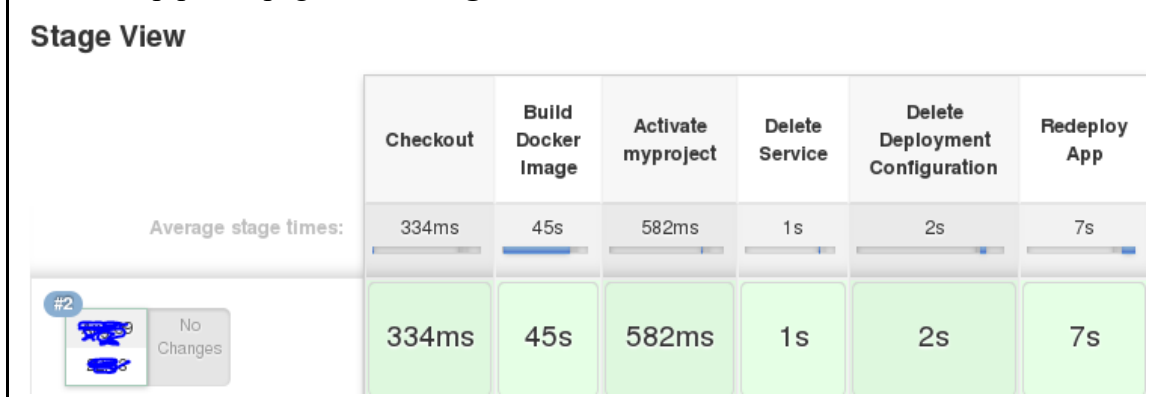
```
git commit -m "v3.0"
```

__7. In the web browser, go to the page where Jenkins **CI-CD Pipeline** is available.

__8. Click **Build Now**.

__9. Under **Build History** you should be able to see a new build.

Notice the pipeline page shows **Stage View** like this:



__10. After the Job finishes, in the web browser, go to the tab where **OpenShift** page is open and click **Overview** tab.

__11. If you time it properly, you should be able to see the node-app pod is getting reinitialized.

__12. When the pod is up and running, in the web browser, go to the Hello Node Tab and refresh the page.

Note: It should show Hello Node.js v3.0 message.

__13. In the terminal, execute following command:

```
docker images
```

Notice node-app image has been updated.

__14. Keep the terminal and the web browser open for the next lab.

Part 11 - Review

In this lab, you installed & configured OpenShift CLI tools. You used both the CLI and web UI to manage OpenShift. You created a project, deployed a Docker container image to it, managed the pods and service applications, scaled the service application by running multiple instances, and then cleaned up the environment by deleting the deployment configuration and the service application.

