

You are in: [DevOps Technical Workshop Wiki](#) > [Lab 1 - DevOps for Basic Environment provisioning](#) > Lab 1 - Steps 2A, 2B, 2C

## Lab 1 - Steps 2A, 2B, 2C

[Like](#) | Updated 27 January 2016 by [Sunil K. Jha](#) | Tags: *None*

### DevOps Technical Workshop

Course  
Prerequisites

Course  
Introduction

**Lab 1 -  
DevOps for  
Basic  
Environment  
Provisioning**

Lab 2 -  
DevOps for  
Scaling  
Environment  
Provisioning

Lab 3 -  
DevOps  
for  
Automated  
Functional  
Testing

Lab 4 -  
DevOps for  
Automated  
Non-  
Functional  
Testing

Lab 5 -  
DevOps  
for  
Automated  
UI Testing


Lab 6 -  
DevOps for  
Continuous  
Monitoring -  
Building  
Information  
Radiators

### Step 2: Perform this lab exercise

In this course, you will use Liberty Profile as the run-time environment. At its heart, the Liberty profile is a dynamic profile of Websphere® Application Server that enables the WAS server to provision only the features required by the application (or set of applications) deployed to the server. It's relatively light-touch and, therefore, suitable for this course; it will work within the constraint of our VM infrastructure. Of course, there are many alternative runtime environments that you'll find yourself using on different client projects. There is nothing specific about Liberty that links it with DevOps or continuous delivery practices; it's just what has been chosen for this course.

We're going to use Jenkins given that it is the leading CI solution out there and is fully open-source and freely available for you to use (subject to any risk management procedures your project might have). Jenkins is also light-touch. You will set it up from scratch, exactly as you would if you wanted to get Jenkins to do CI for you on your projects.

#### (A) Installing Liberty and Jenkins:

1. Open the terminal  and make a new directory for the installation using the following command (*in Linux,*

*the tilde is shorthand for your "home" directory*):

```
mkdir ~/Documents/Jenkins
```

2. Copy the Liberty jar file from the lab directory to the `jenkins` dir using the following command. *jar files are installable java programs*

```
cp ~/Documents/DevOps/lab1/jenkins/wlp* ~/Documents/Jenkins
```

3. Unpack and accept the license for liberty profile by running from the Jenkins directory. You will be asked to specify a target when executing this cmd. Just select return to accept the default location.

```
cd ~/Documents/Jenkins
java -jar wlp-developers-runtime-8.5.5.3.jar
```

4. Running an `ls` command shows the directory now contains the wlp which is where liberty is installed however we are not done. We need to modify the file permission on Liberty's bin directory files.

```
chmod 755 wlp/bin/*
```

5. Now, create a liberty server to which Jenkins will be deployed. Move into the `wlp` folder and execute the `server create` command (note we are calling our Liberty server `Jenkins` below - this doesn't have to be the case; we can name the server anything (for example, `appserver`) but are using `Jenkins` for convenience):

```
cd wlp
```

```
bin/server create jenkins
```

6. This has created your Jenkins server with the default Liberty profile template. However we are going to change this and expose our server on another port, go to the Jenkins server directory.

```
cd ~/Documents/Jenkins/wlp/usr/servers/jenkins
```

7. Open the server.xml file in a text editor, such as vim or getit, and change the HTTP port to 8080 and the HTTPS port to 8443.



The tool, getit, is a text editor that comes packaged with Ubuntu. It is pinned to the Unity bar on the left hand side for convenience.

8. Add another attribute to the httpEndPoint tag. This is done to allow the server bind to any host, that is, the machine name or localhost or the IP.

```
host="*"
```

9. Next, you will move your jenkins.war file into the dropins directory. Verbose logging is not needed but is included below for validation of command. You can validate the command executed successfully by the colour of the arrow on the terminal. It will be red if the cmd has failed and green if successful. Output of the command will just show the file moving (->) from one location to another.

```
cp -v ~/Documents/DevOps/lab1/jenkins/jenkins.war
~/Documents/Jenkins/wlp/usr/servers/jenkins/dropins
```

10. Finally, we can start the Jenkins server. Move into the wlp folder (cd ~/Documents/Jenkins/wlp) and execute the following command:

```
bin/server start jenkins
```

11. Go to <http://localhost:8080/jenkins> in your VM's browser (Firefox is pinned to the Unity bar) to see Jenkins loading up.

Take some time to explore Jenkins and investigate the settings by going to "manage Jenkins" on the left-hand side.

To stop Jenkins app from running, you must stop the Liberty server it runs on. The application server you are running is called 'Jenkins' for simplicity, but it could be called anything else. This command is actually you stopping the Liberty server, upon which the Jenkins app is running.

```
bin/server stop jenkins
```

Note that the Jenkins home directory is ~/.jenkins.

---

## References:

- <http://jenkins-ci.org/>
  - <https://www.cloudbees.com/>
  - <https://wiki.jenkins-ci.org/display/JENKINS/Home>
  - <https://developer.ibm.com/wasdev/downloads/liberty-profile-using-non-eclipse-environments/>
  - [http://www-01.ibm.com/support/knowledgecenter/SSD28V\\_8.5.5/com.ibm.websphere.wlp.core.doc/ae/cwlp\\_core\\_welcome.html](http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/cwlp_core_welcome.html)
- 

## (B) Installing a Jenkins plugin





So, you now have a vanilla instance of Jenkins up and running..... great! However, Jenkins alone doesn't do that much. It's a glorified job scheduler. The real power of Jenkins lies within its plug-ins, and being such a highly reputable open-source product, thousands of plug-ins have been written by the developer for developers. It should be noted that because of this, not all plugins are necessarily secure. It is advised that you check the latest time any work has been done on a plugin to verify it's still in active development/improvement.

As installing plug-ins should become routine to developers who are constantly inspecting and adapting to newer and better ways of doing things, you will install the plug-ins yourself. First is the Git plug-in. If you refer to the diagram at the top of this lab, you'll see that Git is a popular modern source-code management tool. We want Jenkins to integrate it so it can automatically build code immediately following a Git check-in.

There are three ways to install a plug-in in Jenkins, all of which will be explored:


- Download and install the plug-in using the Internet
- Upload a compiled plugin (.jpi file) manually
- Copy existing .jpi files into the \$JENKINS\_HOME/plugins directory

To install the plugin, go to the Jenkins console (<http://localhost:8080/jenkins>) and press  **Manage Jenkins**, followed




by  **Manage Plugins** Add, remove, disable. This list is not sorted alphabetically. The Available (  **Available**   ) tab will display all the plug-ins you can get for Jenkins (over 1000 available now). It may take some time when loading this page, as it can appear blank. Try refreshing the page if it fails to load.

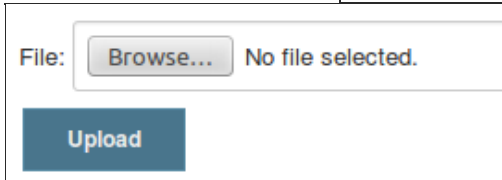
**Note:** Sometimes, Jenkins will fail to load all the plug-ins and the page will remain blank. In this case, try checking your internet connection with the VM. If there is no connection, you will not see the plug-in list. If there is Internet connectivity but still you are unable to view the list, restart Jenkins by going to the following URL:

<http://localhost:8080/jenkins/restart> and click **OK** on restart.

To install one, do not click the link on of the plug-in name but select the **checkbox**, and install or download and install after restart. If you press the link, you will be brought to the information page for that plugin. When deciding on what plug-ins to use, it's best to check how frequent the releases are or the plug-ins popularity, as it will indicate the level of support and commitment to the plug-in. If you are online, try installing the plot build data plug-in:  [Plot Plugin](#) This plugin is

Some plug-ins are dependent on other ones. Try installing the github plug-in. It will install and download several other plugins too such as git plug-in and ssh agent. Jenkins manages these dependencies for you. When looking for plug-ins available, there is a filter box on the Jenkins UI or you can use the search feature in the browser. If the git plug-in fails to install, it's best to restart Jenkins and try again. This can be done by navigating to: <http://localhost:8080/jenkins/restart> and selecting **OK** to restart. Sometimes the Jenkins UI is slow to update so restarting may actually show the plug-ins have been installed so it is best to check there first before trying again.

Alternatively, you can manually install a plug-in (.jpi file) from the `~/Documents/DevOps/lab1/jenkins` directory by going to the advanced column on this page:    **Advanced** . Move to the Upload

plug-in section  and browse to the directory

(`~/Documents/DevOps/lab1/jenkins`), select a file and upload it.

The VM already has copies of the necessary plugins available to install this way. This will cause you to move to another page where you will see the plugin both upload and install at the same time. Some Jenkins plugins require a restart.

Upload manually or install using Jenkins both build-with-parameters and show-build-parameters, as they will be needed in later stages too.

**Before continuing, ensure that you install all the remaining plug-ins that will be used the remaining lessons.**

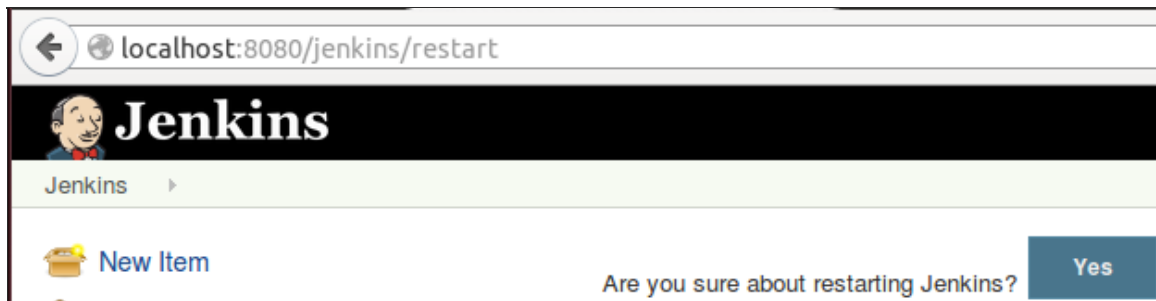
For the purposes of this course, all the plug-ins that will be used are available in the Lab1 directory.

Copy all the pre-downloaded plugins into the plugins directory.

```
cp -v ~/Documents/DevOps/lab1/jenkins/*.jpi ~/.jenkins/plugins/
```

In a real world scenario, you wouldn't have this luxury; you would have to manually download them. If you were to work in an environment where you would be provisioning multiple Jenkins instances to distribute to various project teams, you could use Puppet to ensure that the plug-ins expected for use on a project are there. This has its drawbacks, though, as you would want to make certain that your stock Jenkins is up to date with the latest fixes and changes to the plug-ins.

In order for these plugins to be made available to the Jenkins runtime, you must restart Jenkins. Go to <http://localhost:8080/jenkins/restart>, and select **Yes** to restart Jenkins, as shown below:



To restart Jenkins, you can also restart the Liberty server container it is served from (From Liberty install directory:

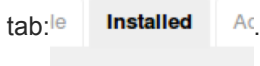
```
~/Documents/Jenkins/wlp)
```

```
bin/server stop jenkins
```

```
bin/server start jenkins
```

It does not matter too much which method you choose to restart Jenkins (via the Webserver or the Apps' ul) but it is handy to know both. Sometimes Jenkins can run out of memory / crash and in these instances it's handy to know how to really pull the plug on the app.

Verify that the plugins have been installed by navigating to Manage Jenkins > Manage Plugins and clicking the Installed



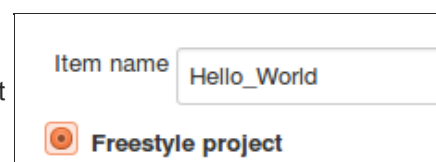
Those plug-ins that have an "Uninstall" button are additional plug-ins to those required by the default Jenkins installed and include the new plugins you just added. Verify that there are approximately 14 plug-ins with the Uninstall button next to them, including the "Build Monitor View," and "Build With Parameters Plug-in".

### (C) Creating your first job

You now have Jenkins up and running with some plug-ins set up that's going to allow you to progress toward Continuous Integration and Continuous Delivery. But your console looks pretty blank right now, doesn't it? To start getting Jenkins doing useful stuff, you need to give him some "jobs" - processes that Jenkins can repeat over and over again and execute either by a button push or some other pre-condition triggering it. Building up jobs is what makes Continuous Delivery possible, so let's get some practice at creating them.


On the Jenkins home screen, press  **New Item**.

Type the name of the item: Hello\_World, and select freestyle project



We are using this type of project because Jenkins does not provide a template for the type of project we will be building. The next screen that loads is the Jenkins job configuration. It is broadly separated into three components: Pre-build tasks,

build steps, and post-build tasks.

Note: Click the  **Configure** button to return to the Settings page at any time. For our Hello World job, we will have no build triggers or source code management.

On the **Build** divider, select **Add build step** (  ), and then select **Execute shell**.


Type the following command into the **Command** box:

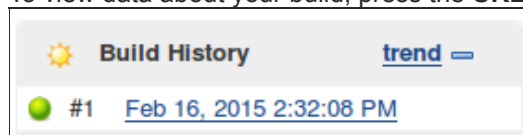
```
echo "Hello World, I am jenkins from ${JOB_NAME}.${BUILD_NUMBER}" >> hello.txt
mv hello.txt ~/Desktop
```

Jenkins exposes various Environment Variables (the items denoted with `${}`), which are made available to each job as it is executing. They are available and can be set at <http://localhost:8080/jenkins/env-vars.html/> Environment variables allow common things to be set centrally.

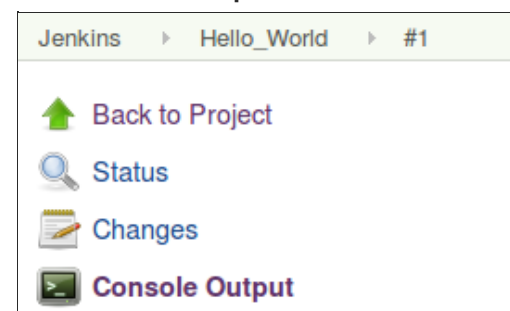
We will not add any post build actions yet. so just click **Apply**, and click **Save** at the end of the page.



To build your job, press the  **Build Now** button on the left-hand side. Jenkins was originally created for jobs to build software, but it can run all kinds of jobs. When Jenkins says "Build," you can think of it as "Run." To view data about your build, press the **URL** on the **build history container** on the bottom left of the screen.



Select **console output** to show a breakdown of the build.



If the build was successful, the icon will be displayed as green circle and the sun will be shining on the home screen.



Return to your desktop to view the file you had Jenkins create. Jenkins exposes a lot of environment variables, which are useful, such as the ones we printed to our file. They can also be used in .sh scripts executed by Jenkins.

Build this job again. Remember by Building we essentially mean *"run this task."*

**Question for your consideration:** Why does the file on your desktop only contain one line with the latest build info when you are using the append (`>>`) bash function?

[Back](#)[Next](#)

## Comments

~~There are no comments.~~

[Back to Top](#)