

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

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

[Like](#) | Updated 25 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

#### (A) Creating a System-Integration environment using existing puppet script.

Now that we have generated one environment using the Puppet script, let's scale up and make another.

Edit the Puppet script (params.pp) in the /home/devops/Documents/DevOps/lab1/puppet directory to reflect the values below [lab1

directory is not a typo]. Gedit  is pinned to the Unity bar for convenience. The important thing to note in these changes is

the swap of 'ci' for 'si'.

```
$LIBERTY_TARGET = "/home/devops/si/Liberty"
$SERVER_NAME = "sidevopsworklight"
$defaultHttpPort = "2080"
$defaultHttpsPort = "2443"
$CONTEXT_ROOT = "/sidevopsworklight"
```

#### Note:

In the real world, you would not modify the config file to build a new environment. For the purpose of this VM, each different environment living on another piece of hardware is simulated by being put in the CI (Continuous Integration), SI (Systems Integration) or UAT (User Acceptance Test) directory inside the home of the user. But, as you are using the same machine for all the environments, you have to ensure that there is no conflict for ports or other resources. Puppet can be run as an agent that can be set up to monitor multiple environments. If you have provisioned several virtual devices and want to ensure that no one changes configs on them (that is, the port of the MFP server), Puppet will actively monitor files and change values to what they should be at all times as defined by the manifest file.

As you did in the previous lab, apply these changes by moving to the following directory on the terminal, and execute the apply command:

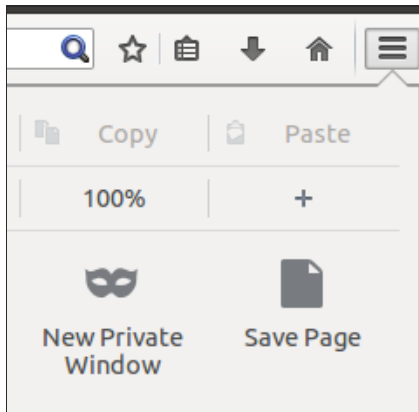
```
cd /home/devops/Documents/DevOps/lab1/puppet
puppet apply puppetmaster.pp
```

Use `puppet apply clean.pp` to reset your puppet scripts if necessary.

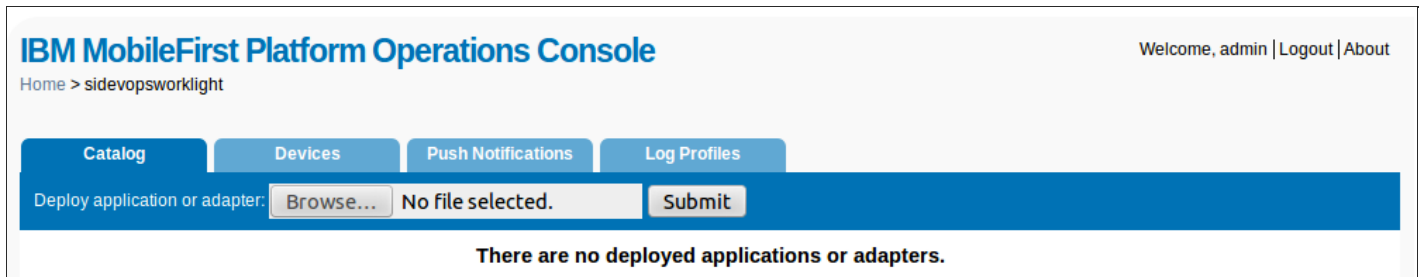
**Note:** This will undo your change to the params.pp file, as it resets it to the values for CI (port 1080, and so on), so if you edited it to create your SI server you will have to make your changes again. If you open it after you have applied your changes, you will see that the values are reset to CI prefixes, and ports will reflect 1xxx.

To check that your new MFP server has loaded correctly, open the console in the browser using the <http://localhost:2080/worklightconsole/index.html>. Ensure that you use the **username** as **admin** and **password** as **admin** to log in to the console.

**Note:** Opening two MFP consoles will cause you to log out of the other one. It is easier to open a private browser window if you want to keep both sessions alive concurrently. To do this, click the **Open** menu on Firefox as shown below, followed by **New Private** window.



If deployment of the server was successful, you should see the following message displayed in your browser:

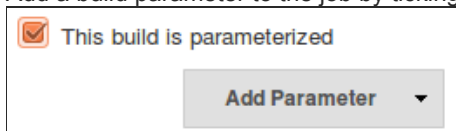


### (B) Creating a second (SI) Jenkins build job

Remember: Jenkins is at <http://localhost:8080/jenkins>

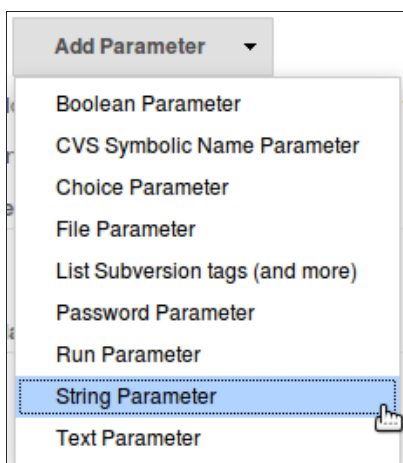
Create a new Jenkins item on the console: **New Item**. Build a freestyle project once again and give it a sensible name once more, such as System-Integration. **It is important that there are no spaces in the job name.**

Add a build parameter to the job by ticking the **This build is parameterized** icon on the configure page that loads.



We are going to use *git tags* as the parameters for our build, so select **Add Parameter** followed by **String**.

**Note:** You are hosting the source code locally to the VM but, in the real world, this would be hosted externally, probably in the cloud. Each developer would have his or her own full copy of the code stored locally, though, as you have in the WorklightProject directory. What you're doing is tagging the source code with build numbers so that you know exactly what version of each file was used in a given build. When you execute a build and 'tag' the code, it is the local repository you are tagging. In reality, you would tag the version hosted elsewhere. To view the current tags in the repository, run *git tag* inside any repo.



The name of your string will be TAG. This will expose a variable (\$) called TAG to our build scripts. In your existing job (Continuous-

Integration), you tag the repository with the name (\$JOB\_NAME) and number (\$BUILD\_NUMBER) of the item when it builds. For example, you should have a build tag Continuous-Integration.1 from your very first job execution stored on the source code for that level that Jenkins has built. This tag is used to fuel the next job (the System-Integration one or, further down the line, User-Acceptance ). On the description field, add a hint such as this to the description for the variable. There should be no Default Value. By feeding one job into another; it allows you to take a stable cut of code and promote it up through the system for further integration and testing while also allowing developers to continue working. Demoing the app using stable code, deployed to a stable environment (that is, one that isn't being trashed every 15 minutes) can allow product owners get a feel for the application they are having built.

String Parameter	
Name	TAG
Default Value	
Description	Git-tag of level of code to be built. eg Continuous-Integration.123

Add File System to the Source Code Management while configuring the job. Select **File System** and add the path: /home/devops/Documents/WorklightProject. Next, select **Clear workspace** and **copy hidden files**.

Source Code Management	
<input type="radio"/> None	
<input type="radio"/> CVS	
<input type="radio"/> CVS Projectset	
<input checked="" type="radio"/> File System	
Path	/home/devops/Documents/WorklightProject
Clear Workspace	<input checked="" type="checkbox"/>
Copy Hidden Files/Folders	<input checked="" type="checkbox"/>

Now, add a build step to Execute Shell, as you have done previously. In this instance, add the following text.

```
export DEPLOYER_ENV="SI"
/home/devops/Documents/WL_Build_Deploy/build-main.sh
```

The DEPLOYER\_ENV="SI" sets the configuration for this job to use the SI environment variables, such as port 22, and not to use the latest code level but the tagged version instead.

**Note:** We do not add a periodic Build Trigger because this build requires a parameter, and we will not supply a default. Builds without a parameter would fail.

Execute a number of builds by passing in a tag from previous **\*Successful\*** Continuous-Integration build to begin generating some historical data (for example, Continuous-Integration.3). To do this, move to the project page and select **Build** and pass in a tag, as shown in the screen shot below.

Project System-Integration	
This build requires parameters:	
TAG	Continuous-Integration.4
Git-tag of level of code to be built. eg Continuous-Integration.123	
<b>Build</b>	

NOTE : Ensure that your SI server is running. If it is not, consult the cheat sheet for instructions on how to start the server.

**(C) Creating an Ops Jenkins Job**

Jenkins can be used to drive utility functions and maintenance tasks; for example, getting server logs, rotating logs, and restarting servers.

Create a new Jenkins item called “Generic-Server-Restart”. *NOTE: Jeob name is important here as the script called by this job looks for this name.*

Again, build a freestyle project.

This job will be parameterized so select the **This build is parameterized checkbox**, and add parameter. The parameter type you will add is a String. This will export whatever value you assign to the variable and make it available to your shell script. For the string name, enter `DEPLOYER_ENV`. The default value can be `CI`. For description, enter **Set the environment you would like to restart** (for example: `CI`, `SI` or `UAT`).

The screenshot shows the Jenkins configuration page for a new project named "Generic-Server-Restart". The "Project name" field is filled with "Generic-Server-Restart". The "Description" field is empty. Below the description, there is a checkbox for "Discard Old Builds" which is unchecked. The "GitHub project" field is empty. The checkbox "This build is parameterized" is checked. Under this checkbox, a "String Parameter" is configured with the name "DEPLOYER\_ENV", a default value of "CI", and a description "Set the environment you would like to restart. For example : CI, SI or UAT".

As you did previously, select **Add build step** followed by **Execute Shell**. The script you are going to run is located here, so enter the following in the command box: `/home/devops/Documents/WL_Build_Deploy/generic-restart.sh`


Take the time to open the script file and understand what it is doing.

Click **Apply** and **Save**.

Execute the job by passing in `CI` to the Build when prompted.

The screenshot shows the Jenkins "Build with parameters" dialog for the project "Project Generic-Server-Restart". It states "This build requires parameters:". There is a parameter "DEPLOYER\_ENV" with a value of "CI". Below the parameter, the description "Set the environment you would like to restart. For example : CI, SI or UAT" is visible. A blue "Build" button is at the bottom left.

Create another job as described above, but change the job name to “Generic-Server-Stop.” You can copy the existing job configuration from **Generic-Server-Restart**.

 **Copy existing Item**

Copy from

This uses the same script and same parameters. Once you have executed this Generic-Server-Stop job, use Generic-Server-Restart to bring your MFP server back to life.

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

## Comments

There are no comments.

[Back to Top](#)