



# OPENSIFT

## OpenShift Enterprise 2.0 Installation, Configuration, Administration and Usage

# Contents

---

1. Overview of OpenShift Enterprise 2.0
2. Using the OpenShift Enterprise Subscription
3. Installing OpenShift Enterprise 2.0
4. Verifying the installation
5. Configuring local machine for DNS resolution
6. Adding cartridges
7. Managing resources
8. Managing districts
9. Installing the RHC client tools
10. Using *rhc setup*
11. Creating a PHP application
12. Managing an application
13. Managing an application
14. Using the web console to create applications
15. Scaling an application
16. The DIY application type
17. Developing Java EE applications using JBoss EAP
18. Using Jenkins continuous integration
19. Using JBoss Tools
20. Using quickstarts
21. Creating a quick start

# **Lab 1: Overview of OpenShift Enterprise 2.0**

---

## **1.1 Assumptions**

This lab manual assumes that you are attending an instructor led training class and that you will be using this lab manual in conjunction with the lecture.

I also assume that you have been granted access to two Red Hat Enterprise Linux servers with which to perform the exercises in this lab manual. If you do not have access to your servers, please notify the instructor.

A working knowledge of SSH, git, and yum, and familiarity with a Linux-based text editor are assumed. If you do not have an understanding of any of these technologies, please let the instructor know.

## **1.2 What you can expect to learn from this training class**

At the conclusion of this training class, you should have a solid understanding of how to install and configure OpenShift Enterprise 2.0. You should also feel comfortable in the usage of creating and deploying applications using the OpenShift Enterprise web console, OpenShift Enterprise administration console, command line tools, and managing the application lifecycle.

## **1.3 Overview of OpenShift Enterprise PaaS**

Platform as a Service is changing the way developers approach developing software. Developers typically use a local sandbox with their preferred application server and only deploy locally on that instance. Developers typically start JBoss locally using the startup.sh command and drop their .war or .ear file in the deployment directory and they are done. Developers have a hard time understanding why deploying to the production infrastructure is such a time consuming process.

System Administrators understand the complexity of not only deploying the code, but procuring, provisioning and maintaining a production level system. They need to stay up to date on the latest security patches and errata, ensure the firewall is properly configured, maintain a consistent and reliable backup and restore plan, monitor the application and servers for CPU load, disk IO, HTTP requests, etc.

OpenShift Enterprise provides developers and IT organizations an auto-scaling cloud application platform for quickly deploying new applications on secure and scalable resources with minimal configuration and management headaches. This means increased developer productivity and a faster pace in which IT can support innovation.

This manual will walk you through the process of installing and configuring an OpenShift Enterprise 2.0 environment as part of this training class that you are attending.

## 1.4 Overview of IaaS

The great thing about OpenShift Enterprise is that we are infrastructure agnostic. You can run OpenShift on bare metal, virtualized instances, or on public/private cloud instances. The only thing that is required is Red Hat Enterprise Linux as the underlying operating system. We require this in order to take advantage of SELinux and other enterprise features so that you can ensure your installation is stable and secure.

What does this mean? This means that in order to take advantage of OpenShift Enterprise, you can use any existing resources that you have in your hardware pool today. It doesn't matter if your infrastructure is based on EC2, VMware, RHEV, Rackspace, OpenStack, CloudStack, or even bare metal as we run on top of any Red Hat Enterprise Linux operating system as long as the architecture is x86\_64.

For this training class will be using OpenStack as our infrastructure as a service layer.

## 1.5 Using the *oo-install* tool

In this training class, we are going to take advantage of the *oo-install* tool which automates the deployment and initial configuration of OpenShift Enterprise Platform. However, for a deeper understanding of the internals of the platform, it is suggested that the student read through the official deployment guide for OpenShift Enterprise.

## 1.6 Electronic version of this document

This lab manual contains many configuration items that will need to be performed on your broker and node hosts. Manually typing in all of these values would be a tedious and error prone effort. To alleviate the risk of errors, and to let you concentrate on learning the material instead of typing tedious configuration items, an electronic version of the document is available at the following URL:

```
http://training.runcloudrun.com
```

In order to download all of the sample configuration files for the lab, enter the following command on your host:

```
# wget -rnp --reject index.* http://training.runcloudrun.com/labs/
```

**Lab 1 Complete!**

# Lab 2: Using the OpenShift Enterprise Subscription

## (Estimated time: XXX minutes)

---

### Servers used:

- broker host
- node host

### Tools used:

- SSH
- subscription-manager
- yum
- ssh-keygen
- ssh-copy-id
- sh

## Registering the system and adding subscriptions

In order to be able to update to newer packages, and to download the OpenShift Enterprise software, your system will need to be registered with Red Hat to allow your system access to appropriate software channels. You will need the following subscriptions at a minimum for this class.

- OpenShift Enterprise Employee Subscription

The machines provided to you in this lab have already been registered with the production Red Hat Network. However, they have not been enabled for the above subscriptions. List all of the available subscriptions for the account that has been registered for you:

**Note:** Execute the following on the broker host

```
# subscription-manager list --available
```

From the list provided, subscribe to the OpenShift Enterprise Employee subscription.

```
# subscription-manager attach --pool {pool id from the above command}
```

Once you have attached to correct pool, verify that you are subscribed to OpenShift Enterprise.

```
# subscription-manager list --consumed
```

Also, take note of the yum repositories that you are now able to install packages from.

```
# yum repolist
```

Perform the subscription process on the node host as well.

## Updating the operating system to the latest packages

We need to update the operating system to have all of the latest packages that may be in the yum repository for RHEL Server. This is important to ensure that you have a recent update to the SELinux packages that OpenShift Enterprise relies on. In order to update your system, issue the following command:

```
# yum update
```

**Note:** Depending on your connection and speed of your broker host, this installation make take several minutes.

## Creating SSH keys

In order for the *oo-install* tool to work correctly, we need to have SSH keys available on the both the broker and the node host to allow for authentication without a password. In order to accomplish this task, we will need to generate a ssh keypair and then copy those keys to both the broker and node hosts. Perform the following steps:

**Note:** Execute the following on the broker host

```
# ssh-keygen -t rsa
```

Use the defaults and do not use a password for the key

```
# ssh-copy-id -i ~/.ssh/id_rsa root@{yourBrokerIPAddress}
```

After entering in the above command, you will be prompted for the root password. This password has been provided to you by the instructor of this course. Enter the password and ensure that no errors occurred. If everything worked correctly, let's copy the key to the node host as well.

```
# ssh-copy-id -i ~/.ssh/id_rsa root@{yourNodeIPAddress}
```

In order to verify that the SSH keys were copied correctly, SSH into the node host from the broker and ensure that no password was required:

**Note:** Execute the following on the broker host

```
# ssh root@{yourNodeIPAddress}
```

You should have been authenticated to the node host without requiring a password. If this is not the case, please alert your instructor so you can troubleshoot the issue.

Disconnect from the node host by exiting your SSH session to the node host.

```
# exit
```

**Lab 2 Complete!**

# Lab 3: Installing OpenShift Enterprise 2.0 (Estimated time: XXX minutes)

---

## Server used:

- broker host

## Tools used:

- oo-install

**Note:** For this lab, use the 209.x.x.x IP address when defining your hosts

## Overview of *oo-install*

The OpenShift team have developed a new interactive installer for the platform that simplifies the installation and configuration of the PaaS. Using the *oo-install* tool is the ideal way to get an environment up and running quickly without having to worry about manually configuring all of the required services. For a better understanding of how the tool works, it is suggested that the user view the source code of the installer to become familiar with the configuration involved in setting up the platform. For this training class, once the installation of the PaaS has started, the instructor will go over the architecture of the PaaS so that you are familiar with all of the components and their purpose.

## Installing the Ruby runtime

The *oo-install* tool requires the use of a Ruby language runtime in order to execute. However, Red Hat Enterprise Linux may not have this installed by default. To check if your system has ruby installed, issue the following command:

**Note:** Execute the following on the broker host

```
# ruby -v
```

If you have ruby installed you should see output similar to the following:

```
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
```

If you get an error message stating that the ruby command can not be found, we will need to install the RPM package on the broker host. In order to install the runtime, enter in the following command:

**Note:** Execute the following on the broker host

```
# yum install ruby
```

Accept any certificates, if prompted, and ensure that the install was successful by running the *ruby -v* command again.

## Download and Execute *oo-install*

In order to use the *oo-install* tool, we will need to download and execute the command. This can be performed in a single action from the command line:

**Note:** Execute the following on the broker host

```
# sh <(curl -s https://install.openshift.com/ose-2.0/)
```

Once the download has completed, the installer will automatically start and you will be presented with the following information:

```
OpenShift Installer
```

```
-----
```

Welcome to OpenShift.

This installer will guide you through a basic system deployment, based on one of the scenarios below.

Select from the following installation scenarios.

You can also type '?' for Help or 'q' to Quit.:"

1. Install OpenShift Enterprise
2. Add a Node to OpenShift Enterprise

Type a selection and press <return>:

Since this is a new installation of OpenShift Enterprise 2.0, select option 1 and press the enter key.

The next section of the installation wizard will begin the configuration process for the OpenShift Enterprise Deployment. The first question you will be asked is if you have an existing broker that is running:

```
It looks like you are running oo-install for the first time on a new system. The installer will guide you through the process of defining your OpenShift deployment.
```

```
First things first: do you already have a running Broker? (y/n/q)
```

Since this is a new installation, enter in *n* and hit enter. The next step is to gather the domain name that you would like to use your applications that will be hosted on the OpenShift Enterprise Platform.

The default is `example.com` but it makes more sense to from an architecture view point to separate these out to their own domain.

Okay. We will gather information to install a Broker in a moment. Before we do that, we need to collect some information about your OpenShift DNS configuration.

What domain do you want to use for applications that are hosted by this OpenShift deployment? `|example.com|`

Enter in `apps.example.com` and hit the enter key. At this point, the installation program will ask you if you want to register the DNS entries for the hosts that we will be creating. Since we are going to be using the BIND server that ships with OpenShift Enterprise, we can select `y` here.

Do you want to register DNS entries for your OpenShift hosts with the same OpenShift DNS service that will be managing DNS records for the hosted applications? `(y/n/q)`

The next step of the installation is to configure the domain that we want to use for our OpenShift Enterprise Hosts. It is good practice to put your hosts (broker and nodes) on a separate domain.

What domain do you want to use for the OpenShift hosts?

Enter in `hosts.example.com` and press the enter key.

## Adding a Broker host

Now that we have our DNS configuration set for our hosts, we can continue with the installation by specifying the fully qualified domain name for our broker.

-----  
Broker Configuration  
-----

Okay. I'm going to need you to tell me about the host where you want to install the Broker.

Hostname (the FQDN that other OpenShift hosts will use to connect to the host that you are describing):

Enter in `broker.hosts.example.com` and press the enter key.

```
Hostname / IP address for SSH access to broker.hosts.example.com from  
the host where you are running oo-install. You can say 'localhost' if  
you are running oo-install from the system that you are describing:  
|broker.hosts.example.com|
```

The next step of the process is to tell the installer the IP address of our broker.hosts.example.com server. For this question, use the 209.x.x.x address that was provided to you by your instructor. Please ensure that you are using the IP address for the broker host here.

You will then be prompted for the username to connect to the broker host. Use the default of root by pressing enter and you should see the following information on the screen:

```
Validating root@209.x.x.x... looks good.  
  
Detected IP address 192.168.59.X at interface eth0 for this host.  
Do you want Nodes to use this IP information to reach this host?  
(y/n/q/?)
```

If you were not able to connect to your broker host with the provided IP address, verify that you copied over the ssh key as mentioned in a previous lab. If you are still having problems with this step, please alert your instructor.

If you were able to successfully connect by seeing the above message, select *y* to use this ip address for the node to connect to the broker host.

## Adding a Node host

We can now begin the process of installing and configuring the node host. Since we don't already have a node host configured, select *n* at the following prompt:

```
-----  
Node Configuration  
-----  
Do you already have a running Node? (y/n/q)
```

Once you select *n*, the installer will ask you if you want to add the *Node* role to the existing broker host that we previously configured. This is useful if you want to spin up an OpenShift Enterprise Environment and colocate both the broker, node, and all gears on a single operating system instance. However, for this lab we will be installing a more production like system by having our broker and node hosts on their own hardware.

Okay. I'm going to need you to tell me about the host where you want to install the Node.

You have already described the following host system(s):

\* broker.hosts.example.com (Broker)

Do you want to assign the Node role to broker.hosts.example.com?  
(y/n/q/?)

Select *n* and press enter to allow us to enter in the information for our node host.

Okay, please provide information about the Node host.

Hostname (the FQDN that other OpenShift hosts will use to connect to the host that you are describing):

To follow the pattern that we set earlier in this lab by having all of our hosts located on their own domain, enter in *node1.hosts.example.com* and press enter. You can then enter in the 192.x.x.x address of your node host that was provided to you by the instructor of this class.

Hostname / IP address for SSH access to node1.hosts.example.com from the host where you are running oo-install. You can say 'localhost' if you are running oo-install from the system that you are describing:  
|node1.hosts.example.com|

Once you enter in the IP address of your node host, ensuring that it is the correct 209.x.x.x address, provide the authentication details just as you did for the broker host. If the installer was able to connect via SSH to your node host, you will be presented with the following information:

Username for SSH access to 192.168.59.x: |root|

Validating root@192.168.59.x... looks good.

Detected IP address 192.168.59.x at interface eth0 for this host.  
Do you want to use this as the public IP information for this Node?  
(y/n/q/?)

Just as we did with the broker host, select *y* to use the IP address as the public information.

After completing that step, you will be presented with an overview of the installation information provided. Ensure that the details of the configuration are correct.

Here are the details of your current deployment.

Note: ActiveMQ, MongoDB and named will all be installed on the Broker.  
For more flexibility, rerun the installer in advanced mode (-a).

#### DNS Settings

- \* App Domain: example.com
- \* Register OpenShift components with OpenShift DNS? Yes
- \* Component Domain: hosts.example.com

#### Role Assignments

+-----+-----+	
Broker	broker.hosts.example.com
Node	node1.hosts.example.com
+-----+-----+	

#### Host Information

+-----+-----+	
Host	broker.hosts.example.com
Roles	Broker
SSH Host	192.168.59.X
User	root
IP Addr	192.168.59.X
Install Status	new
+-----+-----+	
+-----+-----+	
Host	node1.hosts.example.com
Roles	Node
SSH Host	192.168.59.X
User	root
IP Addr	192.168.59.X
IP Interface	eth0
Install Status	new
+-----+-----+	

Do you want to change the deployment info? (y/n/q/?)

If everything looks correct, enter *n* to move the subscription management section of the installation.

## Performing the installation

Now that we have both the broker and node host installation configuration complete, we can move on to the next step of the deployment configuration: subscription management.

If you performed the subscription steps correctly in Lab 2, you will not need to enter in any subscription information when presented with this information:

```
Here is subscription configuration that the installer will use for  
this deployment.
```

Setting	Value
type	none

```
Do you want to make any changes to the subscription info in the  
configuration file? (y/n/q/?)
```

Select *n* and press enter and then select to not set any temporary subscription settings to begin the installation.

The install will now run and may take a while depending on the speed of the connection at your location and the number of RPM packages that need to be installed. During this time, the instructor will lecture about the architecture of OpenShift Enterprise.

If you want to watch the progress of the install, you can tail the log file(s) on the broker and the host:

```
$ tail -f /tmp/open*.log
```

**Lab 3 Complete!**

## #Lab 4: Verifying the installation (Estimated time: 15 minutes)

### Server used:

- broker host

### Tools used:

- cat
- ping
- oo-diagnositcs
- oo-mco
- mongo
- shutdown

## Rebooting the hosts

Congratulations, you have just installed OpenShift Enterprise 2.0! However, before preceding any further in the lab manual we need to verify that the installation was successful. There are a couple of utilities that we can use to help with this task. The first thing we want to check is that everything was installed and configured correctly to ensure that all services will be present after a reboot of the hosts. To ensure that all services are started in the correct order, reboot the broker host first. SSH to your broker host and enter in the following command.

**Note:** Perform the following command on the broker host.

```
# shutdown -r now
```

After issuing this command, it is normal and expected that your SSH session to the broker host will be closed. Once your sessions has closed, wait a minute to allow time for the system to restart and then SSH in to the broker host again. After you have verified that the broker host has been rebooted and able to accept SSH connections, it is safe to reboot the node host.

SSH to your node host and enter in the following command:

**Note:** Perform the following command on the node host.

```
# shutdown -r now
```

Wait a minute or two for your node host to come back online. You can verify that it has restarted when you can SSH into the node host again.

# Verifying the installation

## Verifying DNS

Now that our broker and node hosts have been restarted, we can verify that some of the core services have started upon system boot. The first one want to test is *bind*. In order to test that *bind* was started successfully, enter in the following command:

```
# service named status
```

You should see information similar to the following:

```
version: 9.8.2rc1-RedHat-9.8.2-0.17.rc1.el6_4.6
CPUs found: 2
worker threads: 2
number of zones: 8
debug level: 0
xfers running: 0
xfers deferred: 0
soa queries in progress: 0
query logging is OFF
recursive clients: 0/0/1000
tcp clients: 0/100
server is up and running
named (pid 1106) is running...
```

If *bind* is up and running we can be assured that there are no syntax errors in our zone file(s).

The next aspect of DNS that we can verify is the actual database for our DNS records. There should be two database files that contain the domain information for our hosts. Let's verify that the DNS information for our *hosts.example.com* has been setup correctly with the following command:

```
# cat /var/named/dynamic/hosts.example.com.db
```

The output should look similar to the following.

**Note:** The IP address information for your domains will be different than the 192 information in the following sample.

```

$ORIGIN .
$TTL 1 ; 1 seconds (for testing only)
hosts.example.com      IN SOA broker.hosts.example.com. hostmaster.hosts.example.com.
                           2011112904 ; serial
                           60          ; refresh (1 minute)
                           15          ; retry (15 seconds)
                           1800        ; expire (30 minutes)
                           10          ; minimum (10 seconds)
                           )
NS broker.hosts.example.com.
MX 10 mail.hosts.example.com.

$ORIGIN hosts.example.com.
broker           A    209.132.178.67
node1           A    209.132.178.69

```

Verify that you have entries for both the broker and the node host and listed above. Once you have verified this, take a look at the database for `apps.example.com.db` to verify that it has an `NS` entry for `broker.hosts.example.com`:

```
# cat /var/named/dynamic/apps.example.com.db
```

## Verifying DNS resolution

One of the most common problems that students encounter is improper host name resolution for OpenShift Enterprise. If this error occurs the broker will not be able to contact the node hosts and vice versa. Both the broker and node host should be using your `broker.hosts.example.com` IP address for DNS resolution. To verify this, view the `/etc/resolv.conf` file both the broker and node host to verify it is pointed to the correct `BIND` server.

```
# cat /etc/resolv.conf
```

If everything with DNS is setup correctly, you should be able to ping `node1.hosts.example.com` from the broker and receive a response as shown below:

```
# ping node1.hosts.example.com

PING node1.hosts.example.com (209.132.178.69) 56(84) bytes of data.
64 bytes from node.osop-local (209.132.178.69): icmp_seq=1 ttl=64 time=0.502 ms
```

## Verifying ActiveMQ and MCollective

ActiveMQ is a fully open source messenger service that is available for use across many different

programming languages and environments. OpenShift Enterprise makes use of this technology to handle communications between the broker host and the node host in our deployment.

The first thing we want to ensure is that the daemon is running. Perform the following command on the broker host:

**Note:** Execute this command on the broker host

```
# service activemq status
```

By default, the ActiveMQ console is only available on the broker host and is protected with a username and password. The default password for the *admin* user is a randomly generated string and is located in the */etc/activemq/jetty-realm.properties* file. To view the password for the *admin* user, enter in the following command:

**Note:** Perform the following on the broker host.

```
# grep admin: /etc/activemq/jetty-realm.properties
```

The output will list the authentication information for the user as shown below:

```
admin: 214f110b48089e2c7a6e800a9678ae8a6cd7c9a317fed3a5943764232a9bb2ea5c847
```

The long random string is the password for the *admin* user. Let's verify the topics are up and running by using the curl command on the broker host:

```
# curl --user admin:YOURPASSWORDHERE --silent http://localhost:8161/admin/xml/tc
```

You should see a list of topics scroll by on your screen.

Now that we know that ActiveMQ is up and running and has topics, let's verify the MCollective is able to communicate between the broker and the node hosts. To verify this, use the *oo-mco-ping* command.

**Note:** Execute the following on the broker host

```
# oo-mco ping
```

If MCollective is working correctly, you should see the following output:

```
node1.hosts.example.com           time=114.73 ms
```

```
---- ping statistics ----  
1 replies max: 114.73 min: 114.73 avg: 114.73
```

## Using the *oo-diagnostics* utility

OpenShift Enterprise ships with a diagnostics utility that can check the health and state of your installation. The utility may take a few minutes to run as it inspects your hosts. In order to run this tool, simple enter the following command:

**Note:** Execute the following on the broker host

```
# oo-diagnostics
```

When running the *oo-diagnostics* script at this time, it is expected to see some warning as we have not yet setup districts for our installation. You may also see an error regarding the verification of the security certificate.

**Lab 4 Complete!**

# Lab 5: Configuring local machine for DNS resolution

## (Estimated time: 10 minutes)

### Server used:

- local machine

### Tools used:

- text editor
- networking tools

At this point, we should have a complete OpenShift Enterprise installation working correctly on the lab machines that were provided to you by the instructor. During the next portion of the training, we will be focussing on administration and usage of the OpenShift Enterprise PaaS. To make performing these tasks easier, it is suggested that you add the DNS server that we created in a previous to be the first nameserver that your local machine uses to resolve hostnames. The process for this varies depending on the operating system. This lab manual will cover the configuration for both the Linux and Mac operating systems. If you are using a Microsoft Windows operating system, consult the instructor for instructions on how to perform this lab.

## Configuring example.com resolution for Linux

If you are using Linux, the process for updating your name server is straightforward. Simply edit the */etc/resolv.conf* configuration file and add the IP address of your broker node as the first entry. For example, add the following at the top of the file, replacing the 209.x.x.x IP address with the correct address of your broker node.

```
nameserver 209.x.x.x
```

Once you have added the above nameserver, you should be able to communicate with your OpenShift Enterprise PaaS by using the server hostname. To test this out, ping the broker and node hosts from your local machine:

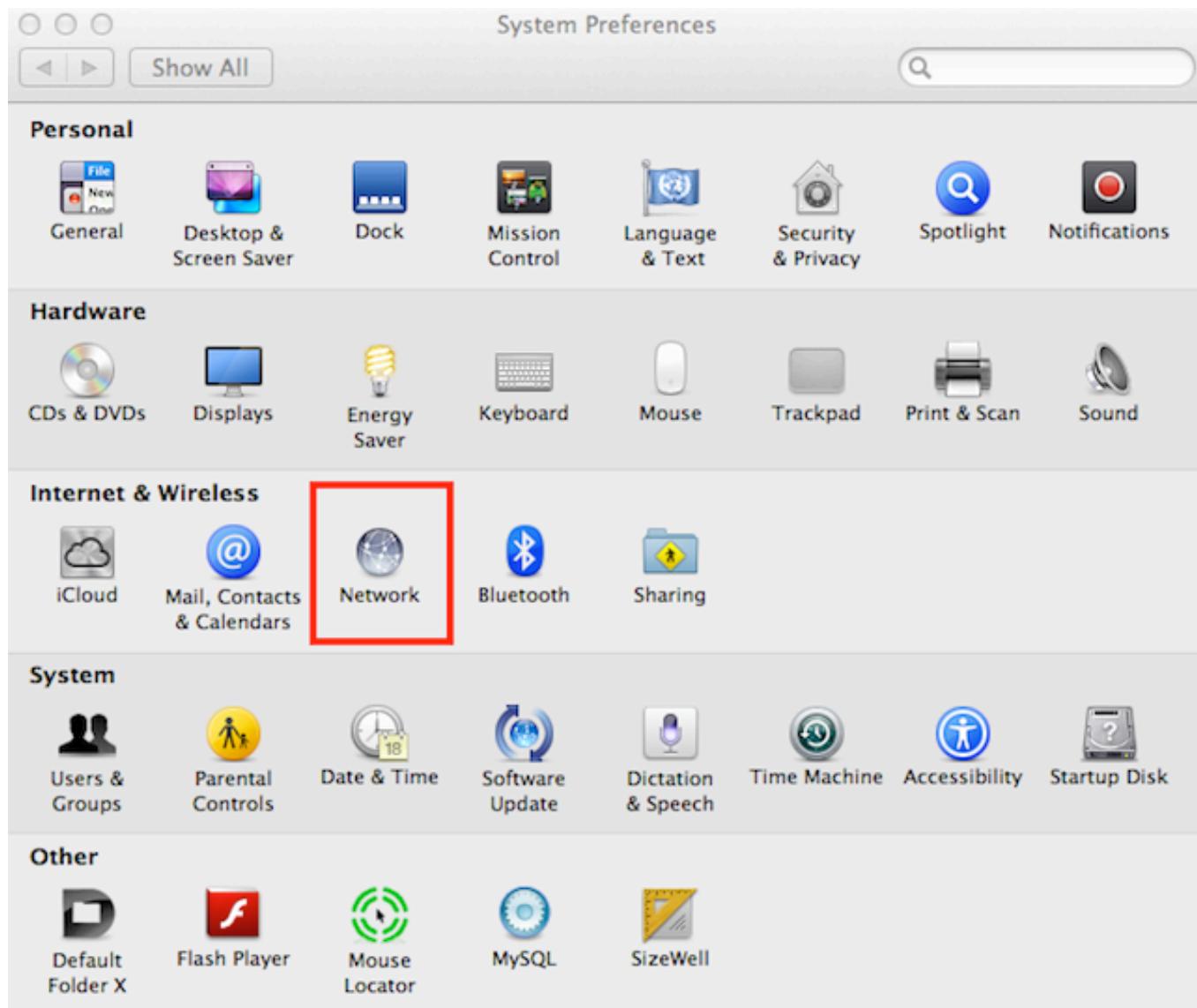
```
$ ping broker.hosts.example.com  
$ ping node.hosts.example.com
```

## Configuring example.com resolution for OS X version 10.6 and below

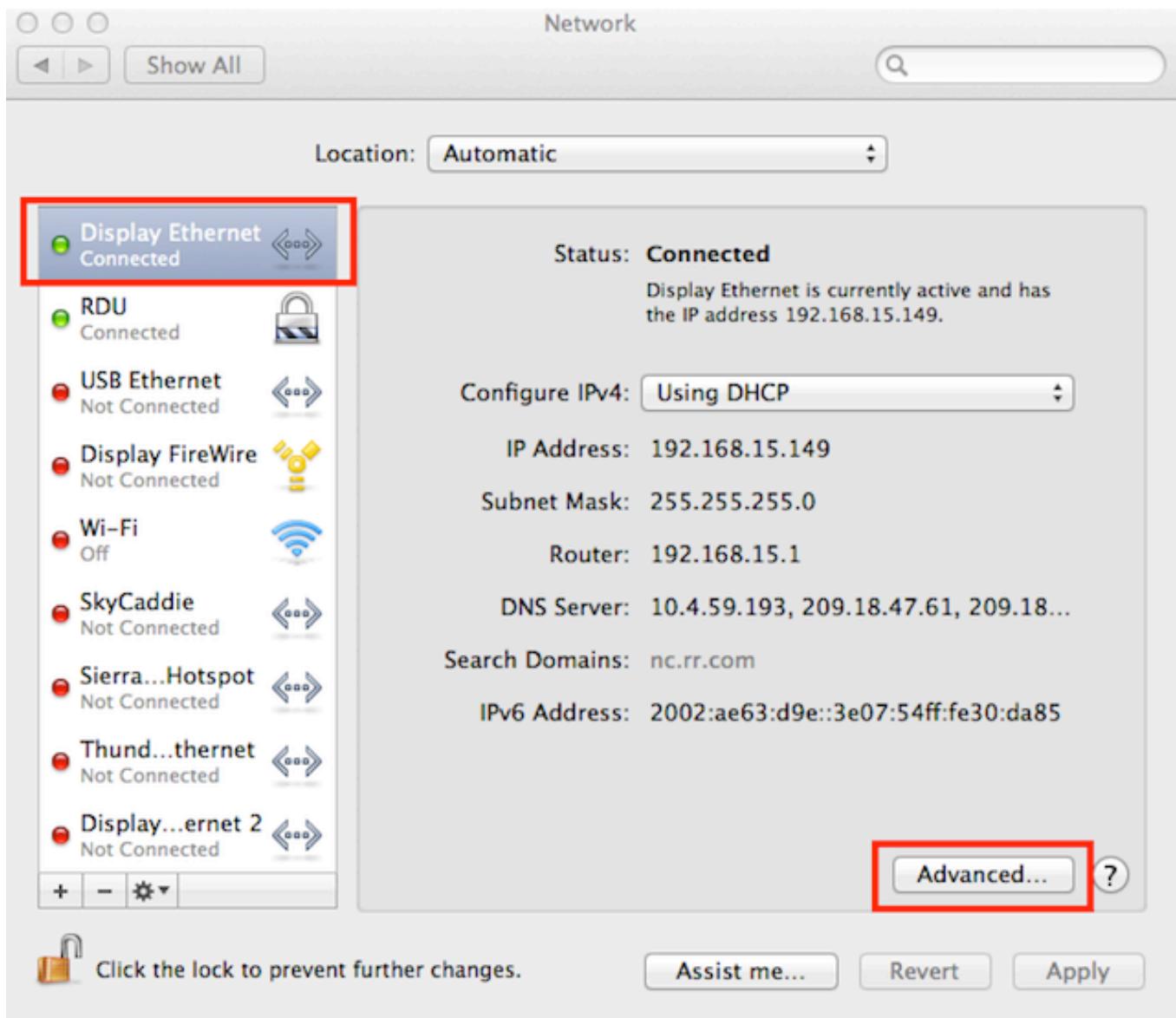
If you are using OSX, you will notice that the operating has a */etc/resolv.conf* configuration file.

However, the operating system does not respect this file and requires users to edit the DNS servers via the *System Preferences* tool.

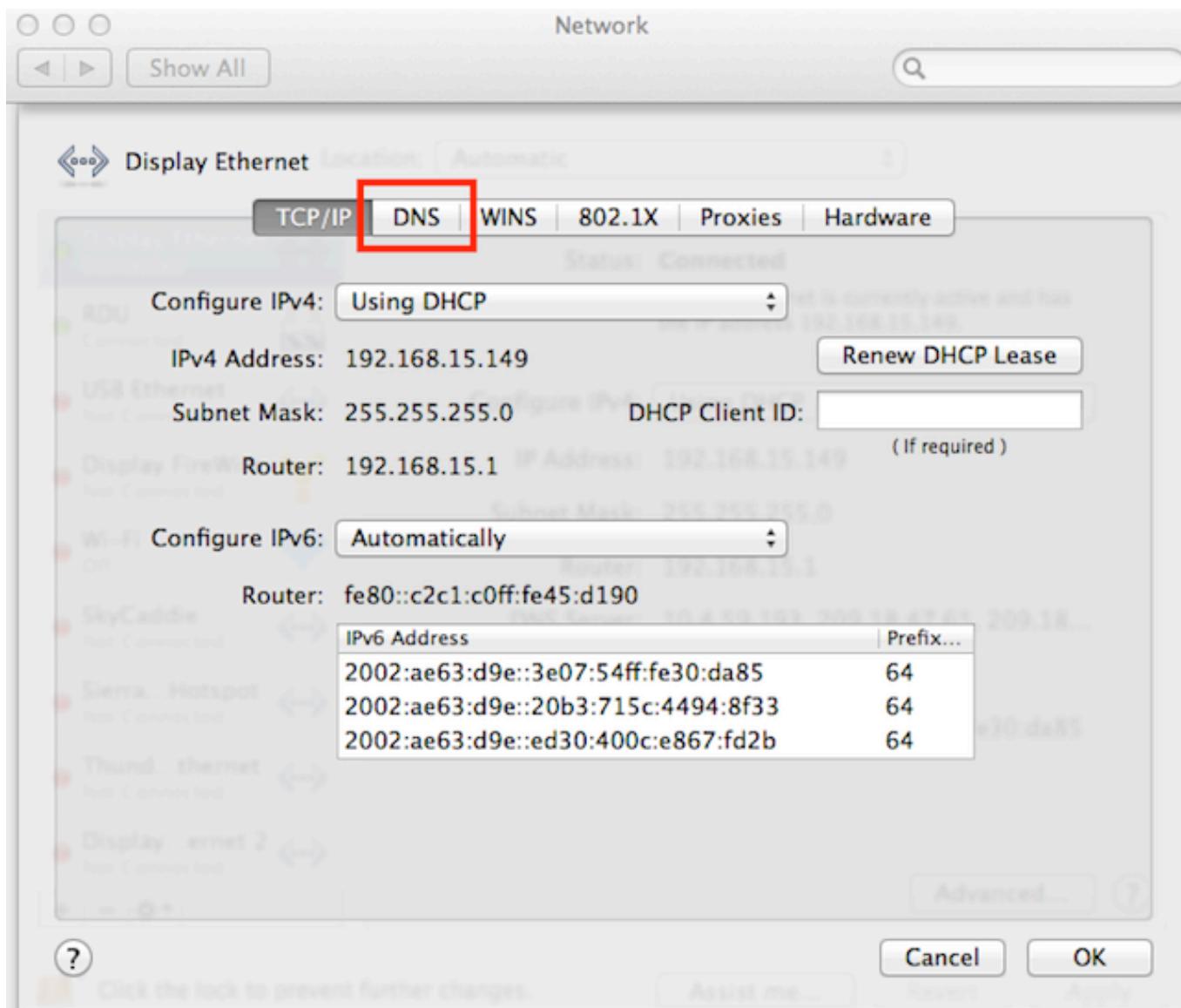
Open up the *System Preferences* tool and select the *Network* utility:



On the bottom left hand corner of the *Network* utility, ensure that the lock button is unlocked to enable user modifications to the DNS configuration. Once you have unlocked the system for changes, locate the ethernet device that is providing connectivity for your machine and click the advanced button:

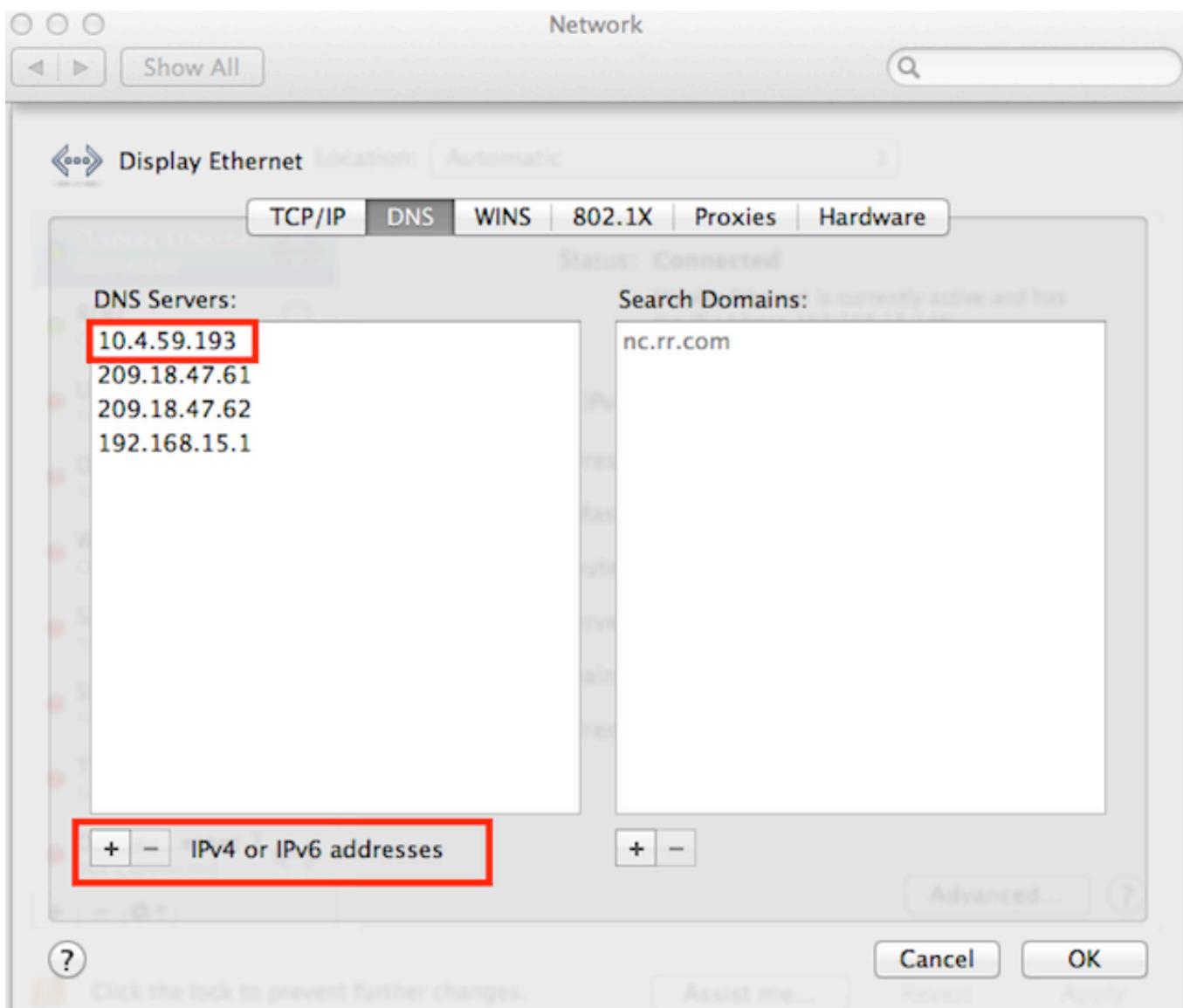


Select the DNS tab at the top of the window:



**Note:** Make a list of the current DNS servers that you have configured for your operating system. When you add a new one, OSX removes the existing servers forcing you to add them back.

Click the + button to add a new DNS server and enter the 10.4.59.x IP address of your broker host.



**Note:** Add your existing nameservers back that you made a note of above.

**Note:** After this training class, remember to remove the DNS server for your broker host.

After you have applied the changes, we can now test that name resolution is working correctly. To test this out, ping the broker and node hosts from your local machine:

```
$ ping broker.hosts.example.com  
$ ping node.hosts.example.com
```

## Configuring example.com resolution for OS X version 10.7 and above

With newer versions of OS X, the operating system will remove a DNS server from the resolution system if the server fails to return a proper address and the next server resolves it. During the OpenShift Enterprise 2.0 installation, the *oo-install* tool does not enable forwarding on the bind instance which will result in domain names other than \*.exmaple.com not resolving. Fortunately we can solve this using the */etc/resolver/* system in place for OS X.

In order to enable this functionality, we need to create the `/etc/resolver` directory if it doesn't exist. Open up a terminal on your OS X machine and enter in the following commands:

```
$ sudo mkdir /etc/resolver  
$ sudo vi /etc/resolver/example.com
```

Once the file has been created, edit the contents to include the following:

```
domain example.com  
nameserver 209.x.x.x
```

Ensure to replace the `209.x.x.x` IP address with the correct one of your broker host.

After you have applied the changes, we can now test that name resolution is working correctly. To test this out, ping the broker and node hosts from your local machine:

```
$ ping broker.hosts.example.com  
$ ping node.hosts.example.com
```

**Lab 5 Complete!**

## #Lab 6: Adding cartridges (Estimated time: 10 minutes)

### Server used:

- node host
- broker host

### Tools used:

- yum
- bundle

By default, OpenShift Enterprise caches certain values for faster retrieval. Clearing this cache allows the retrieval of updated settings.

For example, the first time MCollective retrieves the list of cartridges available on your nodes, the list is cached so that subsequent requests for this information are processed more quickly. If you install a new cartridge, it is unavailable to users until the cache is cleared and MCollective retrieves a new list of cartridges.

This lab will focus on installing cartridges to allow OpenShift Enterprise to create JBoss gears.

## Listing available cartridges for your subscription

For a complete list of all cartridges that you are entitled to install, you can perform a search using the yum command that will output all OpenShift Enterprise cartridges.

**Note: Run the following command on the node host.**

```
# yum search origin-cartridge
```

During this lab, you should see the following cartridges available to install:

```
openshift-origin-cartridge-cron.noarch : Embedded cron support for OpenShift
openshift-origin-cartridge-diy.noarch : DIY cartridge
openshift-origin-cartridge-haproxy.noarch : Provides HA Proxy
openshift-origin-cartridge-jbosseap.noarch : Provides JBossEAP6.0 support
openshift-origin-cartridge-jbossews.noarch : Provides JBossEWS2.0 support
openshift-origin-cartridge-jenkins.noarch : Provides jenkins-1.x support
openshift-origin-cartridge-jenkins-client.noarch : Embedded jenkins client support for OpenShift
openshift-origin-cartridge-mysql.noarch : Provides embedded mysql support
openshift-origin-cartridge-nodejs.noarch : Provides Node.js support
openshift-origin-cartridge-perl.noarch : Perl cartridge
```

openshift-origin-cartridge-php.noarch : Php cartridge  
openshift-origin-cartridge-postgresql.noarch : Provides embedded PostgreSQL support  
openshift-origin-cartridge-python.noarch : Python cartridge  
openshift-origin-cartridge-ruby.noarch : Ruby cartridge

## Installing JBoss support

In order to enable consumers of the PaaS to create JBoss gears, we will need to install all of the necessary cartridges for the application server and supporting build systems. Perform the following command to install the required cartridges:

**Note:** Execute the following on the node host.

```
# yum install openshift-origin-cartridge-jbosseap-6.0.noarch openshift-origin-ca
```

The above command will allow users to create JBoss EAP and JBoss EWS gears. We also installed support for the Jenkins continuous integration environment which we will cover in a later lab. At the time of this writing, the above command will download and install an additional 285 packages on your node host.

**Note:** Depending on your connection and speed of your node host, this installation make take several minutes.

## Clearing the broker application cache

At this point, you will notice that if you try to create a JBoss based application via the web console, that the application type is not available. This is because the broker host creates a cache of available gear types to increase performance. After adding a new cartridge, you need to clear this cache in order for the new gear type to be available to users.

Caching is performed in multiple components:

- Each node maintains a database of facts about itself, including a list of installed cartridges.
- Using MCollective, a broker queries a node's facts database for the list of cartridges and caches the node's response.
- Using the broker's REST API, the developer console queries the broker for the list of cartridges and caches the broker's response.

The cartridge lists are updated automatically at the following intervals:

- The node's database is refreshed every minute.

- The broker's cache is refreshed every six hours.
- The Console's cache is refreshed every five minutes.

In order to clear the cache for both the broker and web console at the same time, enter in the following command:

**Note:** Execute the following on the broker host.

```
# oo-admin-broker-cache --clear --console
```

You should see the following confirmation message:

```
Clearing broker cache.  
Clearing console cache.
```

It may take several minutes before you see the new cartridges available on the web console as it takes a few minutes for the cache to completely clear.

## Testing new cartridges

Given the steps in lab 16 of this training, you should be able to access the web console from a web browser using your local machine. Open up your preferred browser and enter the following URL:

```
http://broker.example.com
```

You will be prompted to authenticate and then be presented with an application creation screen. After the cache has been cleared, and assuming you have added the new cartridges correctly, you should see a screen similar to the following:

The screenshot shows the OpenShift Enterprise application creation interface. At the top, there is a navigation bar with 'OPENSHIFT ENTERPRISE' and tabs for 'Applications', 'Settings', and 'Help'. Below the navigation bar, there are three numbered steps: '1 Choose a type of application', '2 Configure the application', and '3 Next steps'. Step 1 is currently active. On the left, there is a search bar with 'Search by keyword or tag' and a 'Browse by tag...' button. To the right of the search bar, there is a section titled 'Cartridge' with a description: 'A managed runtime that receives security updates and upgrades automatically.' Below this, there is a section titled 'QuickStart' with a description: 'A quick way to try out a new technology with code and libraries preconfigured. You are responsible for updating core libraries for security updates.' A red box highlights a 'Featured' cartridge card for 'JBoss Enterprise Application Platform 6.1.0'. The card includes a thumbnail, the cartridge name, a brief description: 'Market-leading open source enterprise platform for next-generation, highly transactional enterprise Java applications. Build and deploy enterprise Java in the cloud.', a link: 'http://www.redhat.com/products/jbossenterprisemiddleware/application-platform/', and a note: '★ OpenShift maintained'. Below the cartridge card, there are two tabs: 'Java' and 'Ruby', each with a 'see all' link and a list of cartridges. The 'Java' tab shows 'Tomcat 6 (JBoss EWS 1.0)' under the 'TOMCATS' category, and the 'Ruby' tab shows 'Ruby 1.9'.

If you do not see the new cartridges available on the web console, check that the new cartridges are available by viewing the contents of the `/usr/libexec/openshift/cartridges` directory:

```
# cd /usr/libexec/openshift/cartridges  
# ls
```

## Installing the PostgreSQL and DIY cartridges

Using the knowledge that you have gained during in this lab, perform the necessary commands to install both the PostgreSQL and DIY cartridges on your node host. Verify the success of the installation by ensuring that the DIY application type is available on the web console:

The screenshot shows the OpenShift Origin Management Console at the URL `https://broker.example.com/console/application_types`. The page is titled "OPENSHIFT ORIGIN | MANAGEMENT CONSOLE". It displays three steps: "1 Choose a type of application", "2 Configure and deploy the application", and "3 Next steps". Step 1 is active. A blue banner says "Create your first application now!". Below it, a message encourages adding cartridges for databases, metrics, and Jenkins support. The "Web Cartridges" section lists several options: "JBoss Enterprise Application Platform 6.0" (Recently Added), "PHP 5.3", "Tomcat (JBoss Enterprise Web Server 1.0)" (Recently Added), and "Do-It-Yourself". The "Do-It-Yourself" card is highlighted with a red border. It is labeled "EXPERIMENTAL" and describes it as a blank slate for unsupported languages, frameworks, and middleware. A "Select" button is visible next to each cartridge entry.

**Lab 6 Complete!**

# Lab 7: Managing resources (Estimated time: 10 minutes)

---

## Server used:

- node host
- broker host

## Tools used:

- text editor
- oo-admin-ctl-user

## Setting default gear quotas and sizes

A users default gear size and quota is specified in the */etc/openshift/broker.conf* configuration file located on the broker host.

The *VALID\_GEAR\_SIZES* setting is not applied to users but specifies the gear sizes that the current OpenShift Enterprise PaaS installation supports.

The *DEFAULT\_MAX\_GEAR*s setting specifies the number of gears to assign to all users upon user creation. This is the total number of gears that a user can create by default.

The *DEFAULT\_GEAR\_SIZE* setting is the size of gear that a newly created user has access to.

The *\*DEFAULT\_MAX\_DOMAINS* setting specifies the number of domains that one user can create.

Take a look at the */etc/openshift/broker.conf* configuration file to determine the current settings for your installation:

**Note:** Execute the following on the broker host.

```
# cat /etc/openshift/broker.conf
```

By default, OpenShift Enterprise sets the default gear size to small and the number of gears a user can create to 100 and the maximum number of domains to 10.

When changing the */etc/openshift/broker.conf* configuration file, keep in mind that the existing settings are cached until you restart the *openshift-broker* service.

## Setting the number of gears a specific user can create

There are often times when you want to increase or decrease the number of gears a particular user can consume without modifying the setting for all existing users. OpenShift Enterprise provides a command that will allow the administrator to configure settings for an individual user. To see all of the available options that can be performed on a specific user, enter the following command:

```
# oo-admin-ctl-user
```

To see how many gears that our *demo* user has consumed as well as how many gears the *demo* user has access to create, you can provide the following switches to the *oo-admin-ctl-user* command:

```
# oo-admin-ctl-user -l demo
```

Given the current state of our configuration for this training class, you should see the following output:

```
User demo:  
    consumed gears: 0  
    max gears: 100  
    gear sizes: small
```

In order to change the number of gears that our *demo* user has permission to create, you can pass the *--setmaxgears* switch to the command. For instance, if we only want to allow the *demo* user to be able to create 25 gears, we would use the following command:

```
# oo-admin-ctl-user -l demo --setmaxgears 25
```

After entering the above command, you should see the following output:

```
Setting max_gears to 25... Done.  
User demo:  
    consumed gears: 0  
    max gears: 25  
    gear sizes: small
```

## Creating new gear types

In order to add new gear types to your OpenShift Enterprise 2.0 installation, you will need to do two things:

- Create and define the new gear profile on the node host
- Update the list of valid gear sizes on the broker host

Each node can only have one gear size associated with it. That being said, in a multi-node setup you would edit the above file and specify the gear name and then modify the parameters to match the

sizing you would like.

## Setting the type of gears a specific user can create

**Note:** The below information is for informational purposes only. During this lab, we are only working with one node host and can therefore not add additional gear sizes.

In a production environment, a customer will typically have different gear sizes that are available for developers to consume. For this lab, we will only create small gears. However, to add the ability to create medium size gears for the *demo* user, you can pass the *--addgearsize* switch to the *oo-admin-ctl-user* command.

```
# oo-admin-ctl-user -l demo --addgearsize medium
```

After entering the above command, you would see the following output:

```
Adding gear size medium for user demo... Done.  
User demo:  
consumed gears: 0  
max gears: 25  
gear sizes: small, medium
```

In order to remove the ability for a user to create a specific gear size, you can use the *--removegearsize* switch:

```
# oo-admin-ctl-user -l demo --removegearsize medium
```

**Lab 7 Complete!**

# Lab 8: Managing districts (Estimated time: 10 minutes)

---

## Server used:

- node host
- broker host

## Tools used:

- text editor
- oo-admin-ctl-district

Districts define a set of node hosts within which gears can be easily moved to load-balance the resource usage of those nodes. While not required for a basic OpenShift Enterprise installation, districts provide several administrative benefits and their use is recommended.

Districts allow a gear to maintain the same UUID (and related IP addresses, MCS levels and ports) across any node within the district, so that applications continue to function normally when moved between nodes on the same district. All nodes within a district have the same profile, meaning that all the gears on those nodes are the same size (for example small or medium). There is a hard limit of 6000 gears per district.

This means, for example, that developers who hard-code environment settings into their applications instead of using environment variables will not experience problems due to gear migrations between nodes. The application continues to function normally because exactly the same environment is reserved for the gear on every node in the district. This saves developers and administrators time and effort.

## Enabling districts

To use districts, the broker's MCollective plugin must be configured to enable districts. Edit the `/etc/openshift/plugins.d/openshift-origin-msg-broker-mcollective.conf` configuration file and confirm the following parameters are set:

**Note: Confirm the following on the broker host.**

```
DISTRICTS_ENABLED=true  
NODE_PROFILE_ENABLED=true
```

While you are viewing this file, you may notice the `DISTRICTS_REQUIRE_FOR_APP_CREATE=false` setting. This will not allow users to create new application if there are no districts defined or if all

districts are at max capacity.

## Creating and populating districts

To create a district that will support a gear type of small, we will use the *oo-admin-ctl-district* command. After defining the district, we can add our node host (node.example.com) as the only node in that district. Execute the following commands to create a district named small\_district which can only hold *small* gear types:

**Note: Execute the following on the broker host.**

```
# oo-admin-ctl-district -c create -n small_district -p small
```

If the command was successful, you should see output similar to the following:

```
Successfully created district: 513b50508f9f44aeb90090f19d2fd940
```

```
{ "name"=>"small_district",
  "externally_reserved_uids_size"=>0,
  "active_server_identities_size"=>0,
  "node_profile"=>"small",
  "max_uid"=>6999,
  "creation_time"=>"2013-01-15T17:18:28-05:00",
  "max_capacity"=>6000,
  "server_identities"=>{},
  "uuid"=>"513b50508f9f44aeb90090f19d2fd940",
  "available_uids"=>"<6000 uids hidden>",
  "available_capacity"=>6000}
```

If you are familiar with JSON, you will understand the format of this output. What actually happened is a new document was created in the MongoDB database that we installed in while using the *oo-install* tool. To view this document inside of the database, execute the following:

```
# mongo localhost/openshift_broker -u openshift -p mongopass
```

**Note:** The default mongodb username and password is specified in the */etc/openshift/broker.conf* file.

This will drop you into the mongo shell where you can perform commands against the database. The first thing we need to do is list all of the available collections in the *openshift\_broker* database, you can issue the following command:

```
> db.getCollectionNames()
```

You should see the following collections returned:

```
[  
    "applications",  
    "authorizations",  
    "cloud_users",  
    "districts",  
    "domains",  
    "locks",  
    "system.indexes",  
    "system.users",  
    "usage",  
    "usage_records"  
]
```

We can now query the *district* collection to verify the creation of our small district:

```
> db.districts.find()
```

The output should be:

```
{ "_id" : "513b50508f9f44aeb90090f19d2fd940", "name" : "small_district", "extern  
"active_server_identities_size" : 0, "node_profile" : "small", "max_uid" : 6999,  
"2013-01-15T17:18:28-05:00", "max_capacity" : 6000, "server_identities" : [ ], "  
"513b50508f9f44aeb90090f19d2fd940", "available_uids" : [      1000, .....], , ,
```

**Note:** The *server\_identities* array does not contain any data yet.

**Note:** The above output is an abbreviated version. You will see more information returned from the command. The order is not set as this is a JSON document.

Exit the Mongo shell by using the exit command:

```
> exit
```

Now we can add our node host, node.example.com, to the *small\_district* that we created above:

```
# oo-admin-ctl-district -c add-node -n small_district -i node.hosts.example.com
```

You should see the following output:

```
Success!
```

```
{ "available_capacity"=>6000,  
  "creation_time"=>"2013-01-15T17:18:28-05:00",  
  "available_uids"=>"<6000 uids hidden>",  
  "node_profile"=>"small",  
  "uuid"=>"513b50508f9f44aeb90090f19d2fd940",  
  "externally_reserved_uids_size"=>0,  
  "server_identities"=>{ "node.example.com"=>{ "active"=>true} },  
  "name"=>"small_district",  
  "max_capacity"=>6000,  
  "max_uid"=>6999,  
  "active_server_identities_size"=>1}
```

**Note:** If you see an error message indicating that you can't add this node to the district because the node already has applications on it, congratulations – you worked ahead and already created an application. To clean this up, you will need to delete both your application and domain that you created. If you don't know how to do this, ask the instructor.

In order to verify that the information was added to the MongoDB document, enter in the following commands:

```
# mongo localhost/openshift_broker -u openshift -p mongopass  
> db.district.find()
```

You should see the following information in the *server\_identities* array.

```
"server_identities" : [ { "name" : "node.hosts.example.com", "active" : true } ]
```

If you continued to add additional nodes to this district, the *server\_identities* array would show all the node hosts that are assigned to the district.

OpenShift Enterprise also provides a command line tool to display information about a district. Simply enter the following command to view the JSON information that is stored in the MongoDB database:

```
# oo-admin-ctl-district
```

Once you enter the above command, you should a more human readable version of the document:

```
{ "_id"=>"52afc6ed3a0fb2e386000001",
  "active_server_identities_size"=>1,
  "available_capacity"=>6000,
  "available_uids"=>"<6000 uids hidden>",
  "created_at"=>2013-12-17 03:37:17 UTC,
  "gear_size"=>"small",
  "max_capacity"=>6000,
  "max_uid"=>6999,
  "name"=>"small_district",
  "server_identities"=>[{"name"=>"broker.hosts.example.com", "active"=>true}],
  "updated_at"=>2013-12-17 03:45:35 UTC,
  "uuid"=>"52afc6ed3a0fb2e386000001"}
```

## Managing district capacity

Districts and node hosts have a configured capacity for the number of gears allowed. For a node host, the default value configured in `/etc/openshift/resource_limits.conf` is:

- Maximum number of active gears per node : 100

Use the `max_active_gears` parameter in the `/etc/openshift/resource_limits.conf` file to specify the maximum number of active gears allowed per node. By default, this value is set to 100, but most administrators will need to modify this value over time. Stopped or idled gears do not count toward this limit; a node can have any number of inactive gears, constrained only by storage. However, starting inactive gears after the `max_active_gears` limit has been reached may exceed the limit, which cannot be prevented or corrected. Reaching the limit exempts the node from future gear placement by the broker.

## Viewing district capacity statistics

In order view usage information for your installation, you run use the `oo-stats` command. Let's view the current state of our district by entering in the following command:

```
# oo-stats
```

You should see information similar to the following:

```
-----  
Profile 'small' summary:  
-----  
    District count : 1  
    District capacity : 6,000  
    Dist avail capacity : 6,000  
    Dist avail uids : 6,000  
    Lowest dist usage pct : 0.0  
    Highest dist usage pct : 0.0  
    Avg dist usage pct : 0.0  
        Nodes count : 1  
        Nodes active : 1  
        Gears total count : 0  
        Gears active count : 0  
        Available active gears : 100  
        Effective available gears : 100
```

Districts:

Name	Nodes	DistAvailCap	GearsActv	EffAvailGears	LoActvUsgPct	AvgActvUs
small_district	1	6,000	0	100	0.0	

```
-----
```

Summary for all systems:

```
-----
```

Districts : 1  
 Nodes : 1  
 Profiles : 1

**Lab 8 Complete!**

# Lab 9: Installing the RHC client tools (Estimated time: 15 minutes)

---

## Server used:

- localhost

## Tools used:

- ruby
- sudo
- git
- yum
- gem
- rhc

The OpenShift Client tools, known as **rhc**, are built and packaged using the Ruby programming language. OpenShift Enterprise integrates with the Git version control system to provide powerful, decentralized version control for your application source code.

OpenShift Enterprise client tools can be installed on any operating system with Ruby 1.8.7 or higher. Instructions for specific operating systems are provided below. It is assumed that you are running the commands from a command line window, such as Command Prompt, or Terminal. If you are using Ruby Version Manager (rvm) see the instructions below.

## Microsoft Windows

### Installing Ruby for Windows

---

[RubyInstaller 1.9](#) provides the best experience for installing Ruby on Windows XP, Vista, and Windows 7. Download the latest 1.9 version from the [download page](#) and launch the installer.

**Important:** During the installation you should accept all of the defaults, it is mandatory that you select the “Add Ruby executables to your PATH” check box in order to run Ruby from the command line.

After the installation is complete, to verify that the installation is working run:

```
C:\Program Files> ruby -e 'puts "Welcome to Ruby"'
Welcome to Ruby
```

If the ‘Welcome to Ruby’ message does not display, the Ruby executable may not have been added to the path. Restart the installation process and ensure the “Add Ruby executables to your PATH” check box is selected.

## Installing Git for Windows

---

The next step is to install [Git for Windows](#) so that you can synchronize your local application source and your OpenShift application. Git for Windows offers the easiest Git experience on the Windows operating system and is the recommended default - if you use another version of Git please ensure it can be executed from the command line and continue to the next section.

Download and install the [latest version of Git for Windows](#). Ensure that Git is added to your PATH so that it can be run from the command line. After the installation has completed, verify that Git is correctly configured by running:

```
C:\Program Files> git --version  
git version 1.7.11.msysgit.1
```

## Installing RHC for Windows

---

After Ruby and Git are correctly installed, use the RubyGems package manager (included in Ruby) to install the OpenShift Enterprise client tools. Run:

```
C:\Program Files> gem install rhc
```

RubyGems downloads and installs the rhc gem from [www.rubygems.org/gems/rhc](http://www.rubygems.org/gems/rhc). The installation typically proceeds without errors. After the installation has completed, run:

```
C:\Program Files> rhc
```

## Mac OS X

### Installing Ruby for OS X

---

From OS X Lion onwards, Ruby 1.8.7 is installed by default. On older Mac systems, Ruby is shipped as part of the [Xcode development suite](#) and can be installed from your installation CD. If you are familiar with Mac development, you can also use [MacRuby](#) or see the Ruby installation page for [help installing with homebrew](#).

To verify that Ruby is correctly installed run:

```
$ ruby -e 'puts "Welcome to Ruby"'  
Welcome to Ruby
```

## Installing Git for OS X

---

There are a number of options on Mac OS X for Git. We recommend the Git for OS X installer - download and run the latest version of the dmg file on your system. To verify the [Git for OS X installation](#), run:

```
$ git --version  
git version 1.7.11.1
```

## Installing RHC for OS X

---

With Ruby and Git installed, use the RubyGems library system to install and run the OpenShift Enterprise gem. Run:

```
$ sudo gem install rhc
```

After the installation has completed, run:

```
$ rhc -v
```

## Fedora 16 or later

To install from yum on Fedora, run:

```
$ sudo yum install rubygem-rhc
```

This installs Ruby, Git, and the other dependencies required to run the OpenShift Enterprise client tools.

After the OpenShift Enterprise client tools have been installed, run:

```
$ rhc -v
```

## Red Hat Enterprise Linux 6 with OpenShift entitlement

The most recent version of the OpenShift Enterprise client tools are available as a RPM from the OpenShift Enterprise hosted YUM repository. We recommend this version to remain up to date, although a version of the OpenShift Enterprise client tools RPM is also available through EPEL.

With the correct entitlements in place, you can now install the OpenShift Enterprise 2.0 client tools by running the following command:

```
$ sudo yum install rubygem-rhc
```

If you do not have an OpenShift Enterprise on the system you want to install the client tools on, you can install ruby and ruby-gems and then issue the following command:

```
$ sudo gem install rhc
```

## Ubuntu

Use the apt-get command line package manager to install Ruby and Git before you install the OpenShift Enterprise command line tools. Run:

```
$ sudo apt-get install ruby-full rubygems git-core
```

After you install both Ruby and Git, verify they can be accessed via the command line:

```
$ ruby -e 'puts "Welcome to Ruby"'
$ git --version
```

If either program is not available from the command line, please add them to your PATH environment variable.

With Ruby and Git correctly installed, you can now use the RubyGems package manager to install the OpenShift Enterprise client tools. From a command line, run:

```
$ sudo gem install rhc
```

**Lab 9 Complete!**

# Lab 10: Using *rhc setup* (Estimated time: 10 minutes)

**Server used:**

- localhost

**Tools used:**

- rhc

## Configuring RHC setup

By default, the RHC command line tool will default to use the publicly hosted OpenShift environment. Since we are using our own enterprise environment, we need to tell *rhc* to use our broker.hosts.example.com server instead of openshift.com. In order to accomplish this, the first thing we need to do is run the *rhc setup* command and pass in an optional *--server* parameter.

```
$ rhc setup --server broker.hosts.example.com
```

Once you enter in that command, you will be prompted for the username that you would like to authenticate with. For this training class, use the *demo* user account.

The first thing that you will be prompted with will look like the following:

```
The server's certificate is self-signed, which means that a secure connection can't be established with 'broker.hosts.example.com'.
```

```
You may bypass this check, but any data you send to the server could be intercepted. Connect without checking the certificate? (yes|no):
```

Since we are using a self signed certificate, go ahead and select yes here and press the enter key.

At this point, you will be prompted for the username, enter in demo and specify the password for the demo user.

After authenticating, OpenShift Enterprise will prompt if you want to create a authentication token for your system. This will allow you to execute command on the PaaS as a developer without having to authenticate. It is suggested that you generate a token to speed up the other labs in this training class.

The next step in the setup process is to create and upload our SSH key to the broker server. This is required for pushing your source code, via git, up to the OpenShift Enterprise server.

Finally, you will be asked to create a namespace for the provided user account. The namespace is a unique name which becomes part of your application URL. It is also commonly referred to as the users domain. The namespace can be at most 16 characters long and can only contain alphanumeric characters. There is currently a 1:1 relationship between usernames and namespaces. For this lab, create the following namespace:

```
ose
```

## Under the covers

The *rhc setup* tool is a convenient command line utility to ensure that the user's operating system is configured properly to create and manage applications from the command line. After this command has been executed, a *.openshift* directory was created in the users home directory with some basic configuration items specified in the *express.conf* file. The contents of that file are as follows:

```
# Default user login
default_rhlogin='demo'

# Server API
libra_server = 'broker.hosts.example.com'
```

This information will be provided to the *rhc* command line tool for every future command that is issued. If you want to run commands as a different user than the one listed above, you can either change the default login in this file or provide the *-l* switch to the *rhc* command.

## Lab 10 Complete!

# Lab 11: Creating a PHP application (Estimated time: 30 minutes)

## Server used:

- localhost
- node host

## Tools used:

- rhc

In this lab, we are ready to start using OpenShift Enterprise to create our first application. To create an application, we will be using the *rhc app* command. In order to view all of the switches available for the *rhc app* command, enter the following command:

```
$ rhc app -h
```

This will provide you with the following output:

### List of Actions

configure	Configure several properties that apply to an application
create	Create an application
delete	Delete an application from the server
deploy	Deploy a git reference or binary file of an application
force-stop	Stops all application processes
reload	Reload the application's configuration
restart	Restart the application
show	Show information about an application
start	Start the application
stop	Stop the application
tidy	Clean out the application's logs and tmp directories and tidy up server

## Create a new application

It is very easy to create an OpenShift Enterprise application using *rhc*. The command to create an application is *rhc app create* and it requires two mandatory arguments:

- **Application Name** : The name of the application. The application name can only contain alphanumeric characters and at max contain only 32 characters.

- **Type:** The type is used to specify which language runtime to use.

Create a directory to hold your OpenShift Enterprise code projects:

```
$ cd ~  
$ mkdir ose  
$ cd ose
```

To create an application that uses the *php* runtime, issue the following command:

```
$ rhc app create firstphp php-5.3
```

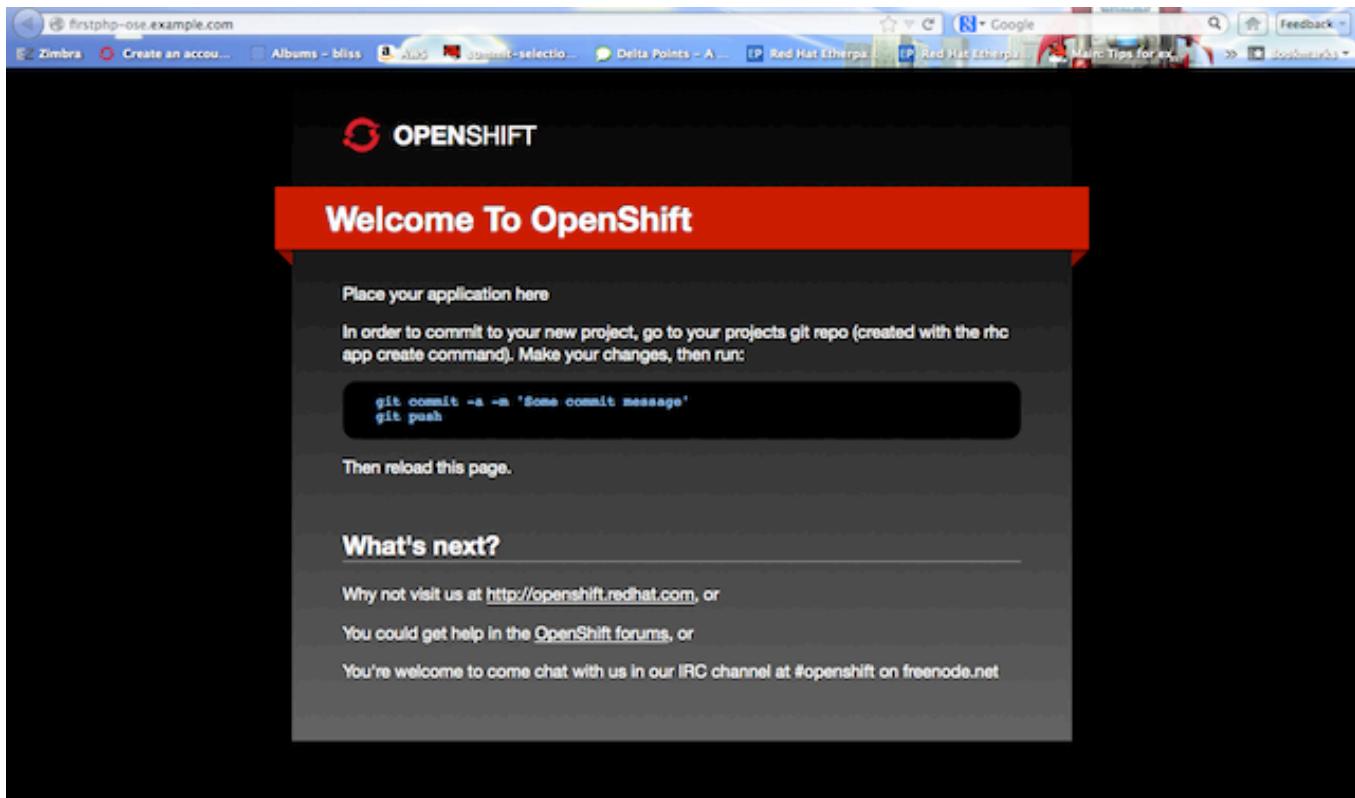
After entering that command, you should see the following output:

```
Application Options  
-----  
Domain: gshipley  
Cartridges: php-5.3  
Gear Size: default  
Scaling: no  
  
Creating application 'firstphp' ... done  
  
Waiting for your DNS name to be available ... done  
  
Cloning into 'firstphp'...  
The authenticity of host 'firstphp-ose.apps.example.com (209.132.178.87)' can't  
RSA key fingerprint is e8:e2:6b:9d:77:e2:ed:a2:94:54:17:72:af:71:28:04.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'firstphp-ose.apps.example.com' (RSA) to the list of  
Checking connectivity... done  
  
Your application 'firstphp' is now available.  
  
URL: http://firstphp-ose.apps.example.com/  
SSH to: 52afd7043a0fb277cf000036@firstphp-ose.apps.example.com  
Git remote: ssh://52afd7043a0fb277cf000036@firstphp-ose.apps.example.com/~/git  
Cloned to: /Users/gshipley/code/ose/firstphp  
  
Run 'rhc show-app firstphp' for more details about your app.
```

If you completed all of the steps in the DNS setup lab correctly, you should be able to verify that your application was created correctly by opening up a web browser and entering the following URL:

<http://firstphp-ose.apps.example.com>

You should see the default template that OpenShift Enterprise uses for a new application.



## What just happened?

After you entered the command to create a new PHP application, a lot of things happened under the covers:

- A request was made to the broker application host to create a new php application
- A message was dropped using MCollective and ActiveMQ to find a node host to handle the application creation request
- A node host responded to the request and created an application / gear for you
- All SELinux and cgroup policies were enabled for your application gear
- A userid was created for your application gear
- A private git repository was created for your gear on the node host
- The git repository was cloned on your local machine
- BIND was updated on the broker host to include an entry for your application

## Understanding the directory structure on the node host

It is important to understand the directory structure of each OpenShift Enterprise application gear. For the PHP application that we just created, we can verify and examine the layout of the gear on the node host. SSH to your node host and execute the following commands:

```
# cd /var/lib/openshift  
# ls
```

You will see output similar to the following:

```
e9e92282a16b49e7b78d69822ac53e1d
```

The above is the unique user id that was created for your application gear. Lets examine the contents of this gear by using the following commands:

```
# cd e9e92282a16b49e7b78d69822ac53e1d  
# ls -al
```

You should see the following directories:

```
total 44  
drwxr-x---. 9 root e9e92282a16b49e7b78d69822ac53e1d 4096 Jan 21 13:47 .  
drwxr-xr-x. 5 root root 4096 Jan 21 13:47 ..  
drwxr-xr-x. 4 root e9e92282a16b49e7b78d69822ac53e1d 4096 Jan 21 13:47 app-root  
drwxr-x---. 3 root e9e92282a16b49e7b78d69822ac53e1d 4096 Jan 21 13:47 .env  
drwxr-xr-x. 3 root root 4096 Jan 21 13:47 git  
-rw-r--r--. 1 root root 56 Jan 21 13:47 .gitconfig  
-rw-r--r--. 1 root root 1352 Jan 21 13:47 .pearrc  
drwxr-xr-x. 10 root root 4096 Jan 21 13:47 php-5.3  
d-----. 3 root root 4096 Jan 21 13:47 .sandbox  
drwxr-x---. 2 root e9e92282a16b49e7b78d69822ac53e1d 4096 Jan 21 13:47 .ssh  
d-----. 3 root root 4096 Jan 21 13:47 .tmp  
[root@node e9e92282a16b49e7b78d69822ac53e1d]#
```

During a previous lab, where we setup the *rhc* tools, our SSH key was uploaded to the server to enable us to authenticate to the system without having to provide a password. The SSH key we provided was actually appended to the *authorized\_keys* file. To verify this, use the following command to view the contents of the file:

```
# cat .ssh/authorized_keys
```

You will also notice the following three directories:

- app-root - Contains your core application code as well as your data directory where persistent

data is stored.

- git - Your private git repository that was created upon gear creation.
- php-5.3 - The core PHP runtime and associated configuration files. Your application is served from this directory.

## Understanding directory structure on the localhost

When you created the PHP application using the *rhc app create* command, the private git repository that was created on your node host was cloned to your local machine.

```
$ cd firstphp  
$ ls -al
```

You should see the following information:

```
total 8  
drwxr-xr-x 9 gshipley staff 306 Jan 21 13:48 .  
drwxr-xr-x 3 gshipley staff 102 Jan 21 13:48 ..  
drwxr-xr-x 13 gshipley staff 442 Jan 21 13:48 .git  
drwxr-xr-x 5 gshipley staff 170 Jan 21 13:48 .openshift  
-rw-r--r-- 1 gshipley staff 2715 Jan 21 13:48 README  
-rw-r--r-- 1 gshipley staff 0 Jan 21 13:48 deplist.txt  
drwxr-xr-x 3 gshipley staff 102 Jan 21 13:48 libs  
drwxr-xr-x 3 gshipley staff 102 Jan 21 13:48 misc  
drwxr-xr-x 4 gshipley staff 136 Jan 21 13:48 php
```

### .git directory

If you are not familiar with the git revision control system, this is where information about the git repositories that you will be interacting with is stored. For instance, to list all of the repositories that you are currently setup to use for this project, issue the following command:

```
$ cat .git/config
```

You should see the following information which specifies the URL for our repository that is hosted on the OpenShift Enterprise node host:

```

[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
ignorecase = true

[remote "origin"]
fetch = +refs/heads/*:refs/remotes/origin/*
url = ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.apps.example.com/~

[branch "master"]
remote = origin
merge = refs/heads/master

[rhc]
app-uuid = e9e92282a16b49e7b78d69822ac53e1d

```

**Note:** You are also able to add other remote repositories. This is useful for developers who also use github or have private git repositories for an existing code base.

## .openshift directory

The .openshift directory is a hidden directory where a user can create action hooks, set markers, and create cron jobs.

Action hooks are scripts that are executed directly so can be written in Python, PHP, Ruby, shell, etc. OpenShift Enterprise supports the following action hooks:

Action Hooks

Action Hook	Description
build	Executed on your CI system if available. Otherwise, executed before the deploy step
deploy	Executed after dependencies are resolved but before application has started
post_deploy	Executed after application has been deployed and started
pre_build	Executed on your CI system if available. Otherwise, executed before the build step

OpenShift Enterprise also supports the ability for a user to schedule jobs to be ran based upon the familiar cron functionality of linux. Any scripts or jobs added to the minutely, hourly, daily, weekly or monthly directories will be ran on a scheduled basis (frequency is as indicated by the name of the directory) using run-parts. OpenShift supports the following schedule for cron jobs:

- daily
- hourly
- minutely
- monthly
- weekly

The markers directory will allow the user to specify settings such as enabling hot deployments or which version of Java to use.

## libs directory

---

The libs directory is a location where the developer can provide any dependencies that are not able to be deployed using the standard dependency resolution system for the selected runtime. In the case of PHP, the standard convention that OpenShift Enterprise uses is providing *PEAR* modules in the deptlist.txt file.

## misc directory

---

The misc directory is a location provided to the developer to store any application code that they do not want exposed publicly.

## php directory

---

The php directory is where all of the application code that the developer writes should be created. By default, two files are created in this directory:

- health\_check.php - A simple file to determine if the application is responding to requests
- index.php - The OpenShift template that we saw after application creation in the web browser.

## Make a change to the PHP application and deploy updated code

To get a good understanding of the development workflow for a user, let's change the contents of the *index.php* template that is provided on the newly created gear. Edit the file and look for the following code block:

```
<h1>
    Welcome to OpenShift
</h1>
```

Update this code block to the following and then save your changes:

```
<h1>
    Welcome to OpenShift Enterprise
</h1>
```

**Note:** Make sure you are updating the `<h1>` tag and not the `<title>` tag.

Once the code has been changed, we need to commit our change to the local git repository. This is accomplished with the `git commit` command:

```
$ git commit -am "Changed welcome message."
```

Now that our code has been committed to our local repository, we need to push those changes up to our repository that is located on the node host.

```
$ git push
```

You should see the following output:

```
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 395 bytes, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: restart_on_add=false
remote: httpd: Could not reliably determine the server's fully qualified domain
remote: Waiting for stop to finish
remote: Done
remote: restart_on_add=false
remote: ~/git/firstphp.git ~/git/firstphp.git
remote: ~/git/firstphp.git
remote: Running .openshift/action_hooks/pre_build
remote: Running .openshift/action_hooks/build
remote: Running .openshift/action_hooks/deploy
remote: hot_deploy_added=false
remote: httpd: Could not reliably determine the server's fully qualified domain
remote: Done
remote: Running .openshift/action_hooks/post_deploy
To ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.example.com/~/git/firstph
 3edf63b..edc0805  master -> master
```

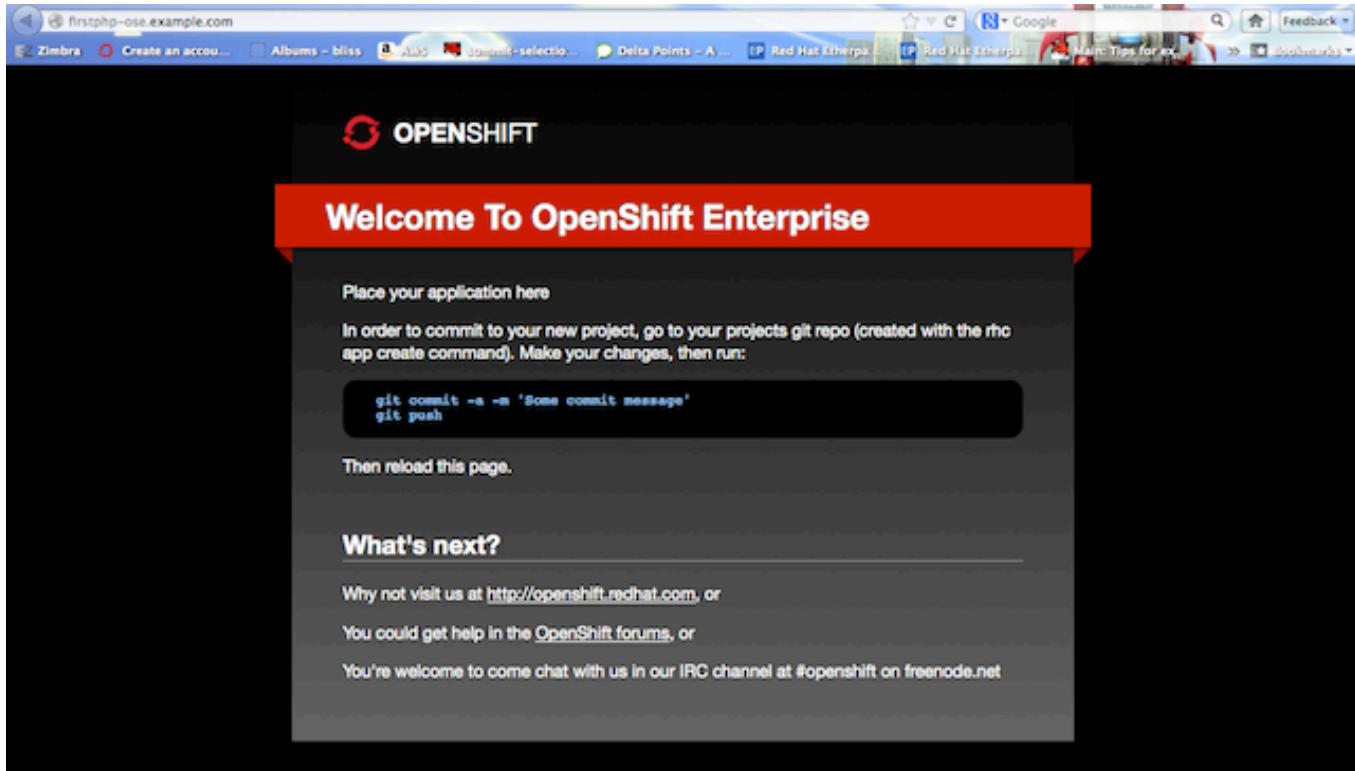
Notice that we stop the application runtime (Apache), deploy the code, and then run any action hooks that may have been specified in the `.openshift` directory.

## Verify code change

If you completed all of the steps in lab 16 correctly, you should be able to verify that your application was deployed correctly by opening up a web browser and entering the following URL:

```
http://firstphp-ose.apps.example.com
```

You should see the updated code for the application.



## Adding a new PHP file

Adding a new source code file to your OpenShift Enterprise application is an easy and straightforward process. For instance, to create a PHP source code file that displays the server date and time, create a new file located in *php* directory and name it *time.php*. After creating this file, add the following contents:

```
<?php  
// Print the date and time  
echo date('l JS \of F Y h:i:s A');  
?>
```

Once you have saved this file, the process for pushing the changes involve adding the new file to your git repository, committing the change, and then pushing the code to your OpenShift Enterprise gear:

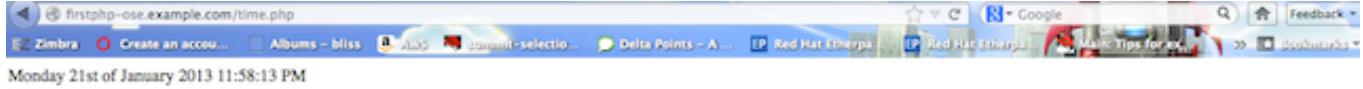
```
$ git add .  
$ git commit -am "Adding time.php"  
$ git push
```

## Verify code change

To verify that we have created and deployed the new PHP source file correctly, open up a web browser and enter the following URL:

```
http://firstphp-ose.example.com/time.php
```

You should see the updated code for the application.



**Lab 11 Complete!**

# Lab 12: Managing an application (Estimated time: 30 minutes)

## Server used:

- localhost
- node host

## Tools used:

- rhc

## Start/Stop/Restart OpenShift Enterprise application

OpenShift Enterprise provides commands to start,stop, and restart an application. If at any point in the future you decide that an application should be stopped for some maintenance, you can stop the application using the *rhc app stop* command. After making necessary maintenance tasks you can start the application again using the *rhc app start* command.

To stop an application execute the following command:

```
$ rhc app stop firstphp
```

RESULT:

```
firstphp stopped
```

Verify that your application has been stopped with the following *curl* command:

```
$ curl http://firstphp-ose.apps.example.com/health_check.php

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>503 Service Temporarily Unavailable</title>
</head><body>
<h1>Service Temporarily Unavailable</h1>
<p>The server is temporarily unable to service your
request due to maintenance downtime or capacity
problems. Please try again later.</p>
<hr>
<address>Apache/2.2.15 (Red Hat) Server at myfirstapp-ose.example.com Port 80</a
</body></html>
```

To start the application back up, execute the following command:

```
$ rhc app start firstphp
```

RESULT:

```
firstphp started
```

Verify that your application has been started with the following *curl* command:

```
$ curl http://firstphp-ose.apps.example.com/health
```

```
1
```

You can also stop and start the application in one command as shown below.

```
$ rhc app restart firstphp
```

RESULT:

```
firstphp restarted
```

## Viewing application details

All of the details about an application can be viewed by the *rhc app show* command. This command will list when the application was created, unique identifier of the application, git URL, SSH URL, and other details as shown below:

```
$ rhc app show firstphp
```

```
Password: ****
```

```
firstphp @ http://firstphp-ose.apps.example.com/ (uuid: 52af7bc3a0fb277cf000070)
```

```
-----
```

```
Domain: ose
```

```
Created: Dec 16 9:49 PM
```

```
Gears: 1 (defaults to small)
```

```
Git URL: ssh://52af7bc3a0fb277cf000070@firstphp-ose.apps.example.com/~/git
```

```
SSH: 52af7bc3a0fb277cf000070@firstphp-ose.apps.example.com
```

```
Deployment: auto (on git push)
```

```
-----
```

```
php-5.3 (PHP 5.3)
```

```
-----
```

```
Gears: 1 small
```

## Viewing application status

The state of application gears can be viewed by passing the *state* switch to the *rhc app show* command as shown below:

```
rhc app show --state firstphp  
Password: ****
```

```
RESULT:  
Cartridge php-5.3 is started
```

## Cleaning up an application

As users start developing an application and deploying changes to OpenShift Enterprise, the application will start consuming some of the available disk space that is part of their quota. This space is consumed by the git repository, log files, temp files, and unused application libraries. OpenShift Enterprise provides a disk space cleanup tool to help users manage the application disk space. This command is also available under *rhc app* and performs the following functions:

- Runs the *git gc* command on the application's remote git repository
- Clears the application's /tmp and log file directories. These are specified by the application's *OPENSOURCE\_LOG\_DIR\** and *OPENSOURCE\_TMP\_DIR* environment variables.
- Clears unused application libraries. This means that any library files previously installed by a *git push* command are removed.

To clean up the disk space on your application gear, run the following command:

```
$ rhc app tidy firstphp
```

After running this command you should see the following output:

```
RESULT:  
firstphp cleaned up
```

## SSH to application gear

OpenShift allows remote access to the application gear by using the Secure Shell protocol (SSH). [Secure Shell \(SSH\)](#) is a network protocol for securely getting access to a remote computer. SSH uses RSA public key cryptography for both the connection and authentication. SSH provides direct access to the command line of your application gear on the remote server. After you are logged in on the remote server, you can use the command line to directly manage the server, check logs and test quick changes. OpenShift Enterprise uses SSH for:

- Performing Git operations
- Remote access your application gear

The SSH keys were generated and uploaded to OpenShift Enterprise by `rhc setup` command we executed in a previous lab. You can verify that SSH keys are uploaded by logging into the OpenShift Enterprise web console and clicking on the “Settings” tab as shown below.

Key name	Type	Contents	
default	ssh-rsa	AAAB3Nza...q3F6ggKl	<a href="#">Delete</a>

**Note:** If you don't see an entry under “Public Keys” then you can either upload the SSH key by clicking on “Add a new key” or run the `rhc setup` command again. This will create a SSH key pair in `<User.Home>/ssh` folder and upload the public key to the OpenShift Enterprise server.

After the SSH keys are uploaded, you can SSH into the application gear as shown below. SSH is installed by default on most UNIX like platforms such as Mac OSX and Linux. For windows, you can use [PuTTY](#). Instructions for installing PuTTY can be found [on the OpenShift website](#).

Although you can SSH in by using the standard ssh command line utility, the OpenShift client tools includes a ssh utility that makes the process of logging in to your application even easier. To SSH to your gear, execute the following command:

```
$ rhc app ssh firstphp
```

If you want to use the ssh command line utility, execute the following command:

```
$ ssh UUID@appname-namespace.apps.example.com
```

You can get the SSH URL by running `rhc app show` command as shown below:

```
$ rhc app show firstphp
Password: ****

firstphp @ http://firstphp-ose.apps.example.com/
=====
Application Info
=====
Created      = 1:47 PM
UUID         = e9e92282a16b49e7b78d69822ac53e1d
SSH URL     = ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.apps.example
Gear Size   = small
Git URL     = ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.apps.example
Cartridges
=====
php-5.3````
```

Now you can ssh into the application gear using the SSH URL shown above:

```
$ ssh e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.apps.example.com
*****
You are accessing a service that is for use only by authorized users.
If you do not have authorization, discontinue use at once.
Any use of the services is subject to the applicable terms of the
agreement which can be found at:
https://openshift.redhat.com/app/legal
*****
Welcome to OpenShift shell

This shell will assist you in managing OpenShift applications.

!!! IMPORTANT !!! IMPORTANT !!! IMPORTANT !!!
Shell access is quite powerful and it is possible for you to
accidentally damage your application. Proceed with care!
If worse comes to worst, destroy your application with 'rhc app destroy'
and recreate it
!!! IMPORTANT !!! IMPORTANT !!! IMPORTANT !!!
Type "help" for more info.
```

You can also view all of the commands available on the application gear shell by running the help command as shown below:

```
[firstphp-ose.apps.example.com ~]\> help
Help menu: The following commands are available to help control your openshift
application and environment.

ctl_app          control your application (start, stop, restart, etc)
ctl_all          control application and deps like mysql in one command
tail_all         tail all log files
export           list available environment variables
rm               remove files / directories
ls               list files / directories
ps               list running applications
kill             kill running applications
mysql            interactive MySQL shell
mongo            interactive MongoDB shell
pgsql            interactive PostgreSQL shell
quota            list disk usage
```

## Viewing log files for an application

Logs are very important when you want to find out why an error is happening or if you want to check the health of your application. OpenShift Enterprise provides the *rhc tail* command to display the contents of your log files. To view all the options available for the *rhc tail* command, issue the following:

```
Usage: rhc tail <application>
```

Tail the logs of an application

#### Options

-n, --namespace NAME	Name of a domain
-o, --opts options	Options to pass to the server-side (linux based) tail command (e.g. -o "-f --since 100")
-f, --files files	File glob relative to app (default <application_name>.log)
-g, --gear ID	Tail only a specific gear
-a, --app NAME	Name of an application

#### Global Options

-l, --rhlogin LOGIN	OpenShift login
-p, --password PASSWORD	OpenShift password
--token TOKEN	An authorization token for accessing your account.
--server NAME	An OpenShift server hostname (default: openshift.redhat.com)
--timeout SECONDS	The timeout for operations

See 'rhc help options' for a full list of global options.

The rhc tail command requires that you provide the application name of the logs you would like to view. To view the log files of our *firstphp* application, use the following command:

```
$ rhc tail firstphp
```

You should see information for both the access and error logs. While you have the *rhc tail* command open, issue a HTTP get request by pointing your web browser to *http://firstphp-ose.apps.example.com*. You should see a new entry in the log files that looks similar to this:

```
10.10.56.204 -- [22/Jan/2013:18:39:27 -0500] "GET / HTTP/1.1" 200 5242 "-" "Mozilla/5.0 (Windows NT 6.1; rv:22.0) Gecko/20100101 Firefox/22.0"
```

The log files are also available on the gear node host in the *php-5.3/logs* directory.

Now that you know how to view log files by using the *rhc tail* command it is also important to know how to view logs during an SSH session on the gear. SSH into the gear and use the knowledge you have learned in the lab to tail the logs files on the gear.

## Viewing disk quota for an application

In a previous lab, we configured the application gears to have a disk usage quota. You can view the quota of your currently running gear by connecting to the gear node host via SSH as discussed

previously in this lab. Once you are connected to your application gear, enter the following command:

```
$ quota -s
```

If the quota information that we configured earlier is correct, you should see the following information:

```
Disk quotas for user e9e92282a16b49e7b78d69822ac53e1d (uid 1000):
  Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
/dev/mapper/VolGroup-lv_root
                22540      0    1024M            338      0   40000
```

To view how much disk space your gear is actually using, you can also enter in the following:

```
$ du -h
```

## Adding a custom domain to an application using the command line

OpenShift Enterprise supports the use of custom domain names for an application. For example, suppose we want to use `http://www.somesupercooldomain.com` domain name for the application `firstphp` we created in a previous lab. The first thing you need to do before setting up a custom domain name is to buy the domain name from domain registration provider.

After buying the domain name, you have to add a [CName record](#) for the custom domain name. Once you have created the CName record, you can let OpenShift Enterprise know about the CName by using the `rhc alias` command.

```
$ rhc alias add firstphp www.mycustomdomainname.com
```

Technically, what OpenShift Enterprise has done under the hood is set up a Vhost in Apache to handle the custom URL.

## Adding a custom domain to an application using the web console

The OpenShift web console now allows you to customize your application's domain host url without having to use the command line tools. Point your browser to `broker.hosts.example.com` and authenticate. After you have authenticated to the web console, click on the *Applications* tab at the top of the screen.

You should see the `firstphp` application listed:

The screenshot shows the OpenShift Enterprise application console. At the top, there's a navigation bar with 'OPENSHIFT ENTERPRISE' and links for 'Applications', 'Settings', and 'Help'. A dropdown menu 'demo' is open. Below the navigation, the application details for 'firstphp-ose.apps.example.com' are shown. The application was 'Created 1 day ago in domain ose'. It has a status of 'Started' with 1 gear and 1 GB storage. It uses a PHP 5.3 cartridge. There are sections for 'Source Code' (ssh://52af7bc3a0fb277cf00) and 'Remote Access'. A 'Delete this application...' button is also present. The 'Cartridges' section lists 'PHP 5.3'. The 'Databases' section includes options to add MySQL 5.1 or PostgreSQL 9.2. A note says 'Or, see the entire list of cartridges you can add'.

Click on the *change* link next to your application name. On the following page, you can specify the custom domain name for your application. Go ahead and add a custom domain name of [www.openshiftrocks.com](http://www.openshiftrocks.com) and click the save button.

The screenshot shows the 'New Alias for firstphp' configuration page. The title is 'New Alias for firstphp'. It explains that aliases allow using your own domain names for your apps on OpenShift. Step 1: Set up the CNAME record with your DNS provider. Step 2: Configure OpenShift to use your alias. The 'Application URL' is 'firstphp-ose.apps.example.com'. The 'Domain name \*' field is highlighted with a red box and contains 'www.openshiftrocks.com'. A note below says 'Enter your custom domain name, e.g. www.example.com or something.example.com'. Under 'SSL Certificate', it says 'Your account does not allow custom SSL certificates.' A warning message in a yellow box says 'Your account does not allow custom SSL certificates. [View your account...](#)'. There are fields for 'SSL Certificate', 'SSL Certificate Chain', 'Certificate Private Key', and 'Private Key Pass Phrase', each with a 'Choose File' button. At the bottom are 'Cancel' and 'Save' buttons, with 'Save' highlighted by a red box.

You should now see the new domain name listed in the console. You can also verify the alias was added by running the following command at your terminal prompt:

```
$ rhc app show firstphp
```

If the alias was added correctly, you should see the following output:

```
firstphp @ http://firstphp-ose.apps.example.com/ (uuid: 52af7bc3a0fb277cf000070
-----
Domain:      ose
Created:     Dec 16  9:49 PM
Gears:       1 (defaults to small)
Git URL:    ssh://52af7bc3a0fb277cf000070@firstphp-ose.apps.example.com/~/.git
SSH:        52af7bc3a0fb277cf000070@firstphp-ose.apps.example.com
Deployment:  auto (on git push)
Aliases:    www.openshiftrocks.com

php-5.3 (PHP 5.3)
-----
Gears: 1 small
```

If you point your web browser to [www.openshiftrocks.com](http://www.openshiftrocks.com), you will notice that it does not work. This is because the domain name has not been setup with a dns registry. In order to verify that the vhost was added, add an entry in your `/etc/hosts` file on your local machine.

```
$ sudo vi /etc/hosts
```

Add the following entry, replacing the IP address with the address of your node host.

```
209.132.178.87  www.openshiftrocks.com
```

Once you have edited and saved the file, open your browser and go to the [www.openshiftrocks.com](http://www.openshiftrocks.com) URL. You should see the application.

Once you have verified that the vhost was added correctly by viewing the site in your web browser, delete the line from the `/etc/hosts` file.

## Backing up an application

Use the `rhc snapshot save` command to create backups of your OpenShift Enterprise application. This command creates a gzipped tar file of your application and of any locally-created log and data files. This snapshot is downloaded to your local machine and the directory structure that exists on the server is maintained in the downloaded archive.

```
$ rhc snapshot save firstphp  
Password: ****  
  
Pulling down a snapshot to firstphp.tar.gz...  
Waiting for stop to finish  
Done  
Creating and sending tar.gz  
Done  
  
RESULT:  
Success
```

After the command successfully finishes you will see a file named firstphp.tar.gz in the directory where you executed the command. The default filename for the snapshot is \$Application\_Name.tar.gz. You can override this path and filename with the -f or –filepath option.

**NOTE:** This command will stop your application for the duration of the backup process.

Now that we have our application snapshot saved, edit the *index.php* file in your firstphp application and change the *Welcome to OpenShift Enterprise* <h1> tag to say *Welcome to OpenShift Enterprise before restore*.

Once you have made this change, perform the following to push your changes to your application gear:

```
$ git commit -am "Added message"  
$ git push
```

Verify that changes are reflected in your web browser.

## Restoring a backup

Not only you can take a backup of an application but you can also restore a previously saved snapshot. This form of the *rhc* command restores the git repository, as well as the application data directories and the log files found in the specified archive. When the restoration is complete, OpenShift Enterprise runs the deployment script on the newly restored repository. To restore an application snapshot, run the following command:

```
$ rhc snapshot restore firstphp -f firstphp.tar.gz
```

You will see the following confirmation message:

```
Restoring from snapshot firstphp.tar.gz...
Removing old data dir: ~/app-root/data/*
Restoring ~/app-root/data

RESULT:
Success
```

**NOTE:** This command will stop your application for the duration of the restore process.

## Verify application has been restored

Open up a web browser and point to the following URL:

```
http://firstphp-ose.apps.example.com
```

If the restore process worked correctly, you should see the restored application running just as it was before.

## Deleting an application

You can delete an OpenShift Enterprise application by executing the *rhc app delete* command. This command deletes your application and all of its data on the OpenShift Enterprise server but leaves your local directory intact. This operation can not be undone so use it with caution.

```
$ rhc app delete someAppToDelete

Are you sure you wish to delete the 'someAppToDelete' application? (yes/no)
yes

Deleting application 'someAppToDelete'

RESULT:
Application 'someAppToDelete' successfully deleted
```

There is another variant of this command which does not require the user to confirm the delete operation. To use this variant, pass the *--confirm* flag.

```
$ rhc app delete --confirm someAppToDelete

Deleting application 'someAppToDelete'

RESULT:
Application 'someAppToDelete' successfully deleted
```

## Viewing a thread dump of an application

**Note:** The following sections requires a Ruby or JBoss application type. Since we have not created one yet in this class, read through the material below but don't actually perform the commands at this time.

You can trigger a thread dump for Ruby and JBoss applications using the *rhc threaddump* command. A thread dump is a snapshot of the state of all threads that are part of the runtime process. If an application appears to have stalled or is running out of resources, a thread dump can help reveal the state of the runtime, identify what might be causing any issues and ultimately to help resolve the problem. To trigger a thread dump execute the following command:

```
$ rhc threaddump ApplicationName
```

After running this command for a JBoss or Ruby application, you will be given a log file that you can view in order to see the details of the thread dump. Issue the following command, substituting the correct log file:

```
$ rhc tail ApplicationName -f ruby-1.9/logs/error_log-20130104-000000-EST -o '-n
```

**Lab 12 Complete!**

# Lab 13: Using cartridges (Estimated time: 30 minutes)

## Server used:

- localhost
- node host

## Tools used:

- rhc
- mysql
- tail
- git
- PHP

Cartridges provide the actual functionality necessary to run applications. Currently, there are several cartridges available to support different programming languages, databases, monitoring and management. Cartridges are designed to be extensible so the community can add support for any programming language, database or any management tool not officially supported by OpenShift Enterprise. Please refer to the official OpenShift Enterprise documentation for how you can [write your own cartridge](#).

```
https://www.openshift.com/wiki/introduction-to-cartridge-building
```

## Viewing available cartridges

To view all of the available commands for working with cartridges on OpenShift Enterprise, enter the following command:

```
$ rhc cartridge -h
```

## List available cartridges

To see a list of all available cartridges to users of this OpenShift Enterprise deployment, issue the following command:

```
$ rhc cartridge list
```

You should see the following output depending on which cartridges you have installed:

```
RESULT:  
cron-1.4, mysql-5.1, haproxy-1.4, postgresql-8.4
```

## Add the MySQL cartridge

In order to use a cartridge, we need to embed it into our existing application. OpenShift Enterprise provides support for version 5.1 of this popular open source database. To enable MySQL support for the *firstphp* application, issue the following command:

```
$ rhc cartridge-add mysql-5.1 -a firstphp
```

You should see the following output:

```
dding mysql-5.1 to application 'firstphp' ... done
```

```
mysql-5.1 (MySQL 5.1)
```

```
-----  
Gears:          Located with php-5.3  
Connection URL: mysql://$OPENSHIFT_MYSQL_DB_HOST:$OPENSHIFT_MYSQL_DB_PORT/  
Database Name:  firstphp  
Password:       9svQXLVtv89Y  
Username:       adminxzGaLVm
```

MySQL 5.1 database added. Please make note of these credentials:

```
Root User: adminxzGaLVm  
Root Password: 9svQXLVtv89Y  
Database Name: firstphp
```

```
Connection URL: mysql://$OPENSHIFT_MYSQL_DB_HOST:$OPENSHIFT_MYSQL_DB_PORT/
```

## Using MySQL

Developers will typically interact with MySQL by using the mysql shell command on OpenShift Enterprise. In order to use the mysql shell, use the information you gained in a previous lab in order to SSH to your application gear. Once you have been authenticated, issue the following command:

```
[firstphp-ose.example.com ~]\> mysql
```

You will notice that you did not have to authenticate to the MySQL database. This is because OpenShift Enterprise sets environment variables that contains the connection information for the database.

When embedding the MySQL database, OpenShift Enterprise creates a default database based upon

the application name. That being said, the user has full permissions to create new databases inside of MySQL. Let's use the default database that was created for us and create a *users* table:

```
mysql> use firstphp;
Database changed

mysql> create table users (user_id int not null auto_increment, username varchar(255));
Query OK, 0 rows affected (0.01 sec)

mysql> insert into users values (null, 'gshipley@redhat.com');
Query OK, 1 row affected (0.00 sec)
```

Verify that the user record has been added by selecting all rows from the *users* table:

```
mysql> select * from users;
+-----+-----+
| user_id | username      |
+-----+-----+
|       1 | gshipley@redhat.com |
+-----+-----+
1 row in set (0.00 sec)
```

To exit out of the MySQL session, simply enter the *exit* command:

```
mysql> exit
```

## MySQL environment variables

As mentioned earlier in this lab, OpenShift Enterprise creates environment variables that contain the connection information for your MySQL database. If a user forgets their connection information, they can always retrieve the authentication information by viewing these environment variables:

**Note: Execute the following on the application gear**

```
[firstphp-ose.example.com ~]\> env | grep MYSQL
```

You should see the following information return from the command:

```
OPENSHIFT_MYSQL_DIR=/var/lib/openshift/52af7bc3a0fb277cf000070/mysql/
OPENSHIFT_MYSQL_DB_PORT=3306
OPENSHIFT_MYSQL_DB_HOST=127.10.134.130
OPENSHIFT_MYSQL_DB_PASSWORD=9svQXLVtv89Y
OPENSHIFT_MYSQL_IDENT=redhat:mysql:5.1:0.2.6
OPENSHIFT_MYSQL_DB_USERNAME=adminxzGaLVm
OPENSHIFT_MYSQL_DB_SOCKET=/var/lib/openshift/52af7bc3a0fb277cf000070/mysql//soc
OPENSHIFT_MYSQL_DB_URL=mysql://adminxzGaLVm:9svQXLVtv89Y@127.10.134.130:3306/
OPENSHIFT_MYSQL_DB_LOG_DIR=/var/lib/openshift/52af7bc3a0fb277cf000070/mysql//lc
```

To view a list of all *OPENSHIFT* environment variables, you can use the following command:

```
[firstphp-ose.example.com ~]\> env | grep OPENSHIFT
```

## Viewing MySQL logs

Given the above information, you can see that the log file directory for MySQL is specified with the *OPENSHIFT\_MYSQL\_DB\_LOG\_DIR* environment variable. To view these log files, simply use the tail command:

```
[firstphp-ose.example.com ~]\> tail -f $OPENSHIFT_MYSQL_DB_LOG_DIR/*
```

## Connecting to the MySQL cartridge from PHP

Now that we have verified that our MySQL database has been created correctly, and have created a database table with some user information, let's connect to the database from PHP in order to verify that our application code can communicate to the newly embedded MySQL cartridge. Create a new file in the *php* directory of your *firstphp* application named *dbtest.php*. Add the following source code to the *dbtest.php* file:

```

<?php

$dbhost = getenv("OPENSHIFT_MYSQL_DB_HOST");
$dbport = getenv("OPENSHIFT_MYSQL_DB_PORT");
$dbuser = getenv("OPENSHIFT_MYSQL_DB_USERNAME");
$dbpwd = getenv("OPENSHIFT_MYSQL_DB_PASSWORD");
$dbname = getenv("OPENSHIFT_APP_NAME");

$connection = mysql_connect($dbhost, $dbuser, $dbpwd);

if (!$connection) {
    echo "Could not connect to database";
} else {
    echo "Connected to database.<br>";
}

$dbconnection = mysql_select_db($dbname);

$query = "SELECT * from users";

$rs = mysql_query($query);
while ($row = mysql_fetch_assoc($rs)) {
    echo $row['user_id'] . " " . $row['username'] . "\n";
}

mysql_close();

?>

```

Once you have created the source file, add the file to your git repository, commit the change, and push the change to your OpenShift Enterprise gear.

```

$ git add .
$ git commit -am "Adding dbtest.php"
$ git push

```

After the code has been deployed to your application gear, open up a web browser and enter the following URL:

```
http://firstphp-ose.apps.example.com/dbtest.php
```

You should see a screen with the following information:

```
Connected to database.
1 gshipley@redhat.com
```

## Managing cartridges

OpenShift Enterprise provides the ability to embed multiple cartridges in an application. For instance, even though we are using MySQL for our *firstphp* application, we could also embed the cron cartridge as well. It may be useful to stop, restart, or even check the status of a cartridge. To check the status of our MySQL database, use the following command:

```
$ rhc cartridge-status mysql -a firstphp
```

To stop the cartridge, enter the following command:

```
$ rhc cartridge-stop mysql -a firstphp
```

Verify that the MySQL database has been stopped by either checking the status again or viewing the following URL in your browser:

```
http://firstphp-ose.example.com/dbtest.php
```

You should see the following message returned to your browser:

```
Could not connect to database
```

Start the database back up using the *start* switch.

```
$ rhc cartridge-start mysql -a firstphp
```

Verify that the database has been restarted by opening up a web browser and entering in the following URL:

```
http://firstphp-ose.apps.example.com/dbtest.php
```

You should see a screen with the following information:

```
Connected to database.  
1 gshipley@redhat.com
```

OpenShift Enterprise also provides the ability to list important information about a cartridge by using the *show* switch. For example, if a user has forgotten their MySQL connection information, they can display this information with the following command:

```
$ rhc cartridge-show mysql -a firstphp
```

The user will then be presented with the following output:

```
Password: ****

mysql-5.1 (MySQL 5.1)
-----
Gears:          Located with php-5.3
Connection URL: mysql://$OPENSHIFT_MYSQL_DB_HOST:$OPENSHIFT_MYSQL_DB_PORT/
Database Name: firstphp
Password:       9svQXLVtv89Y
Username:       adminxzGaLVm
```

## Using port forwarding

At this point, you may have noticed that the database cartridge is only accessible via a 127.x.x.x private address. This ensures that only the application gear can communicate with the database.

With OpenShift Enterprise port forwarding, developers can connect to remote services with local client tools. This allows the developer to focus on code without having to worry about the details of configuring complicated firewall rules or SSH tunnels. To connect to the MySQL database running on our OpenShift Enterprise gear, you have to first forward all the ports to your local machine. This can be done using the *rhc port-forward* command. This command is a wrapper that configures SSH port forwarding. Once the command is executed, you should see a list of services that are being forwarded and the associated IP address and port to use for connections as shown below:

```
$ rhc port-forward firstphp

To connect to a service running on OpenShift, use the Local address

Service Local           OpenShift
-----
httpd    127.0.0.1:8080  =>  127.10.134.129:8080
mysql    127.0.0.1:3307  =>  127.10.134.130:3306

Press CTRL-C to terminate port forwarding
```

In the above snippet, you can see that mysql database, which we added to the *firstphp* gear, is forwarded to our local machine. If you open <http://127.0.0.1:8080> in your browser you will see the application.

**Note:** At the time of this writing, there is an extra step to enable port forwarding on Mac OS X based systems. You will need to create an alias on your loopback device for the IP address listed in output shown above.

```
sudo ifconfig lo0 alias 127.0.0.1
```

Now that you have your services forward, you can connect to them using local client tools. To connect to the MySQL database running on the OpenShift Enterprise gear, run the *mysql* command as shown below:

```
$ mysql -uadmin -p -h 127.0.0.1
```

**Note:** The above command assumes that you have the MySQL client installed locally.

## Enable *hot\_deploy*

If you are familiar with PHP, you will probably be wondering why we stop and start apache on each code deployment. Fortunately, we provide a way for developers to signal to OpenShift Enterprise that they do not want us to restart the application runtime for each deployment. This is accomplished by creating a *hot\_deploy* marker in the correct directory. Change to your application root directory, for example `~/code/ose/firstphp` and issue the following commands:

```
$ touch .openshift/markers/hot_deploy
$ git add .
$ git commit -am "Adding hot_deploy marker"
$ git push
```

Pay attention to the output:

```
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 403 bytes, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: restart_on_add=false
remote: Will add new hot deploy marker
remote: App will not be stopped due to presence of hot_deploy marker
remote: restart_on_add=false
remote: ~/git/firstphp.git ~/git/firstphp.git
remote: ~/git/firstphp.git
remote: Running .openshift/action_hooks/pre_build
remote: Running .openshift/action_hooks/build
remote: Running .openshift/action_hooks/deploy
remote: hot_deploy_added=false
remote: App will not be started due to presence of hot_deploy marker
remote: Running .openshift/action_hooks/post_deploy
To ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.apps.example.com/~/git/firstphp
  4fbda99..fdbd056  master -> master
```

The two lines of importance are:

```
remote: Will add new hot deploy marker  
remote: App will not be stopped due to presence of hot_deploy marker
```

Adding a hot\_deploy marker will significantly increase the speed of application deployments while developing an application.

**Lab 13 Complete!**

# Lab 14: Using the web console to create applications

## (Estimated time: 15 minutes)

**Server used:**

- localhost

**Tools used:**

- OpenShift Enterprise Web Console
- git

OpenShift Enterprise provides users with multiple ways to create and manage applications. The platform provides command line tools, IDE integration, REST APIs, and a web console. During this lab, we will explore the creation and management of application using the web console.

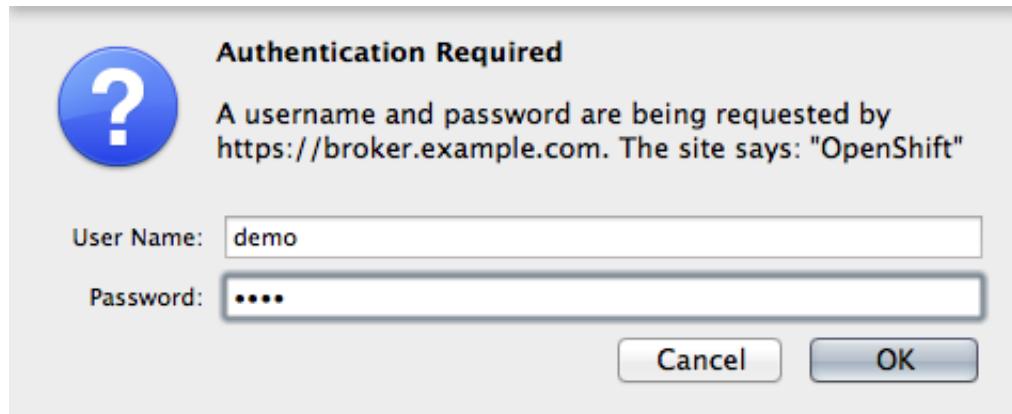
Having DNS resolution setup on your local machine, as discussed in a previous lab, is crucial in order to complete this lab.

### Authenticate to the web console

Open your favorite web browser and go to the following URL:

```
http://broker.hosts.example.com
```

Once you enter the above URL, you will be asked to authenticate using basic auth. For this training class, you can use the demo account that has been provided for you.



After entering in valid credentials, you will see the OpenShift Enterprise web console dashboard:

The screenshot shows the OpenShift Enterprise web interface. At the top, there's a navigation bar with the OpenShift logo and the text "OPENShift ENTERPRISE". On the right of the bar, there's a dropdown menu labeled "demo". Below the bar, there are three tabs: "Applications" (which is selected), "Settings", and "Help".

The main content area is titled "Applications". It displays a list of applications available in the domain. There is one application listed:

- firstphp - www.openshiftrocks.com** (using 1 of 25)

Below this list is a button labeled "Add Application...". To the right of the application list, there are several links and options:

- You may want to...
  - Create another domain
  - Add a collaborator
  - Create a scalable application
- From the command line
  - The `rhc` client lets you:
    - Access logs
    - Save and restore backups
    - Connect directly to internal services

## Creating a new application

In order to create a new application using the web console, click on the *ADD APPLICATION* button. You will then be presented with a list of available runtimes that you can choose from. To follow along with our PHP examples above, let's create a new PHP application and name it *phptwo*.

 **OPENShift ENTERPRISE**

Applications Settings Help

1 Choose a type of application    2 Configure the application    3 Next steps

Choose a web programming cartridge or kick the tires with a quickstart. After you create the application you can add cartridges to enable additional capabilities like databases, metrics, and continuous build support with Jenkins.

Search by keyword or tag  Browse by tag...

**Featured**

 JBoss Enterprise Application Platform 6.1.0

Market-leading open source enterprise platform for next-generation, highly transactional enterprise Java applications. Build and deploy enterprise Java in the cloud.

<http://www.redhat.com/products/jbossenterprisemiddleware/application-platform/>

☆ OpenShift maintained

JAVA JBOSS JEE FULL PROFILE

**Java** see all

 Tomcat 6 (JBoss EWS 1.0)  
TOMCAT6

 Tomcat 7 (JBoss EWS 2.0)  
TOMCAT7

**Ruby** see all

 Ruby 1.9

 Ruby 1.8

**Python** see all

 Python 2.6

 Python 2.7

**Other types**

 Node.js 0.10  
JAVASCRIPT NODEJS

 PHP 5.3  
PHP

 Perl 5.10  
PERL

After selecting to create a new PHP application, specify the name of your application:

Based On **PHP 5.3 Cartridge**

PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. Popular development frameworks include: CakePHP, Zend, Symfony, and Code Igniter.

<http://www.php.net>

OpenShift automatically registered this domain name for your application. You can add your own domain name later.

Public URL  You can also create a new domain.

Optional URL to a Git repository

We'll create a Git code repository in the cloud, and populate it with a set of reasonable defaults. If you provide a Git URL, your application will start with an exact copy of the code and configuration provided in this Git repository.

Gears **Small**

Gears are the application containers running your code. For most applications, the 'small' gear size provides plenty of resources.

Cartridges **PHP 5.3**

Applications are composed of cartridges - each of which exposes a service or capability to your code. All applications must have a web cartridge.

Scaling

OpenShift automatically routes web requests to your web gear. If you allow your application to scale, we'll set up a load balancer and allocate more gears to handle traffic as you need it.

+1

Once you have created the application, you will see a confirmation screen with some important information:

- GIT repository URL
- Instructions for making code changes

The screenshot shows the OpenShift Enterprise web interface. At the top, there's a navigation bar with 'OPENSHIFT ENTERPRISE' and links for 'Applications', 'Settings', and 'Help'. A dropdown menu 'demo' is open. Below the navigation, a progress bar indicates three steps: 'Choose a type of application', 'Configure the application', and 'Next steps'. A message says 'Your application has been created. Continue to the application overview page.' On the left, under 'Making code changes', there's a section for cloning the repository with the command: 

```
git clone ssh://52b205a83a0fb2bc1d000013@phptwo-ose.apps.example.com/~/git/phptwo.git/
```

This will create a folder with the source code of your application. After making a change, add, commit, and push your changes.

```
git add .
git commit -m 'My changes'
git push
```

When you push changes the OpenShift server will report back its status on deploying your code. The server will run any of your configured deploy hooks and then restart the application.

On the right, under 'Manage your app', there's a section for the Command-Line tool with instructions to install it on Linux, Mac OS X, or Windows. It also links to the User Guide for managing applications from the command line. Another section for JBoss Developer Studio provides a link to download and install the IDE.

## Clone your application repository

Open up a command prompt and clone your application repository with the instructions provided on the web console. When executing the *git clone* command, a new directory will be created in your current working directory. Once you have a local copy of your application, make a small code change to the *index.php* and push your changes to your OpenShift Enterprise gear.

Once you have made a code change, view your application in a web browser to ensure that the code was deployed correctly to your server.

## Adding a cartridge with the web console

Click on the *My Applications* tab at the top of the screen and then select the *Phptwo* application by clicking on it.

The screenshot shows the OpenShift Enterprise web console's Applications dashboard. At the top, there are tabs for Applications, Settings, and Help. A dropdown menu labeled "demo" is visible in the top right corner. Below the tabs, the title "Applications" is displayed next to a gear icon. A message "Available in domain ose" is shown, along with a note that 2 of 25 cartridges are being used. Two application cards are listed: "firstphp - www.openshiftrocks.com" (using PHP 5.3, MySQL 5.1) and "phptwo" (using PHP 5.3). Both cards show a status of "1" with a gear icon. To the right of the cards, there are links for "Create another domain", "Add a collaborator", and "Create a scalable application". Below the cards, a section titled "From the command line" provides links for the rhc client, access logs, save and restore backups, and connect to internal services. A blue "Add Application..." button is located at the bottom left.

After clicking on the *Phptwo* application link, you will be presented with the management dashboard for that application. On this page, you can view the GIT repository URL, add a cartridge, or delete the application. We want to add the MySQL database to our application. To do this, click on the *Add MySQL 5.1* link.

The screenshot shows the OpenShift Enterprise web console's application dashboard for "phptwo-ose.apps.example.com". The top header includes the OpenShift logo, the application name, and a dropdown menu. The main content area shows the application's status as "Started" with 1 instance. It lists the "PHP 5.3" cartridge, which is currently running. The "Source Code" URL is provided as ssh://52b205a83a0fb2bc1d01. Below the cartridge information, there are sections for "Databases" (with options to "Add MySQL 5.1" and "Add PostgreSQL 9.2", where "Add MySQL 5.1" is highlighted with a red box), "Continuous Integration" (with an option to "Enable Jenkins"), and "Remote Access" (with a link to log in). A link to "Delete this application..." is also present. A note at the bottom encourages users to see the entire list of cartridges they can add.

Once the MySQL database cartridge has been added to your application, the web console will display a confirmation screen which contains the connection information for your database.

The screenshot shows the OpenShift Enterprise application dashboard. At the top, there's a navigation bar with links for Applications, Settings, and Help. The main content area displays the details for the application "phptwo-ose.apps.example.com". It shows the application was created 6 minutes ago in domain ose, is currently Started with 1 gear, and has a status of OK. A green box contains a message about a MySQL 5.1 database being added, providing credentials: Root User: adminRVkH5h4, Root Password: in7T1-FRNPzP, and Database Name: phptwo. It also provides a Connection URL: mysql://\$OPENSHIFT\_MYSQL\_DB\_HOST:\$OPENSHIFT\_MYSQL\_DB\_PORT/. Below this, there's a section for Cartridges, listing PHP 5.3 and MySQL 5.1. The PHP cartridge is shown as Started, with 1 small gear and 1 GB storage. To the right, there are links for Source Code (ssh://52b205a83a0fb2bc1d0), Remote Access (Want to log in to your application?), and a Delete this application... button. At the bottom left, there's a link to see the entire list of cartridges you can add.

If you recall from a previous lab, the connection information is always available via environment variables on your OpenShift Enterprise gear.

## Verify database connection

Using information you learned in a previous lab, add a PHP file that tests the connection to the database. You will need to modify the previously used PHP code block to only display if the connection was successful as we have not created a schema for this new database instance.

Once you have completed this lab and your application, *phptwo*, is connected to the database, raise your hand to show your instructor.

**Lab 14 Complete!**

# Lab 15: Scaling an application (Estimated time: 15 minutes)

---

## Server used:

- localhost
- node host

## Tools used:

- rhc
- ssh
- git
- touch
- pwd

Application scaling enables your application to react to changes in HTTP traffic and automatically allocate the necessary resources to handle the current demand. The OpenShift Enterprise infrastructure monitors incoming web traffic and automatically adds additional gears to satisfy the demand your application is receiving.

## How scaling works

If you create a non-scaled application, the web cartridge occupies only a single gear and all traffic is sent to that gear. When you create a scaled application, it consumes two gears; one for the high-availability proxy (HAProxy) itself, and one for your actual application (after three scale events). If you add other cartridges like PostgreSQL or MySQL to your application, they are installed on their own dedicated gears.

The HAProxy cartridge sits between your application and the network and routes web traffic to your web cartridges. When traffic increases, HAProxy notifies the OpenShift Enterprise servers that it needs additional capacity. OpenShift checks that you have a free gear (out of your max number of gears) and then creates another copy of your web cartridge on that new gear. The code in the git repository is copied to each new gear, but the data directory begins empty. When the new cartridge copy starts it will invoke your build hooks and then HAProxy will begin routing web requests to it. If you push a code change to your web application all of the running gears will get that update.

The algorithm for scaling up and scaling down is based on the number of concurrent requests to your application. OpenShift Enterprise allocates 10 connections per gear - if HAProxy sees that you're

sustaining 90% of your peak capacity, it adds another gear. If your demand falls to 50% of your peak capacity for several minutes, HAProxy removes that gear. Simple!

Because each cartridge is “share-nothing”, if you want to share data between web cartridges you can use a database cartridge. Each of the gears created during scaling has access to the database and can read and write consistent data. As OpenShift Enterprise grows we anticipate adding more capabilities like shared storage, scaled databases, and shared caching.

The OpenShift Enterprise web console shows you how many gears are currently being consumed by your application. We have lots of great things coming for web application scaling, so stay tuned.

## Create a scaled application

In order to create a scaled application using the *rhc* command line tools, you need to specify the *-s* switch to the command. Let’s create a scaled PHP application with the following command:

```
$ rhc app create scaledapp php-5.3 -s
```

After executing the above command, you should see output that specifies that you are using both the PHP and HAProxy cartridges:

## Application Options

---

Domain: ose  
Cartridges: php-5.3  
Gear Size: default  
Scaling: yes

Creating application 'scaledapp' ... done

Waiting for your DNS name to be available ... done

Cloning into 'scaledapp'...

The authenticity of host 'scaledapp-ose.apps.example.com (209.132.178.87)' can't be established.  
RSA key fingerprint is e8:e2:6b:9d:77:e2:ed:a2:94:54:17:72:af:71:28:04.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'scaledapp-ose.apps.example.com' (RSA) to the list of known hosts.  
Checking connectivity... done

Your application 'scaledapp' is now available.

URL: http://scaledapp-ose.apps.example.com/  
SSH to: 52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com  
Git remote: ssh://52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com/~/.git  
Cloned to: /Users/gshipley/code/ose/scaledapp

Run 'rhc show-app scaledapp' for more details about your app.

Log in to the web console with your browser and click on the *scaledapp* application. You will notice while looking at the gear details that it lists the number of gears that your application is currently using.

The screenshot shows the OpenShift Enterprise web interface. At the top, there's a navigation bar with 'OPENSHIFT ENTERPRISE' and dropdown menus for 'demo' and other options. Below the navigation, the application details for 'scaledapp-ose.apps.example.com' are displayed. The application was created 1 minute ago in domain 'ose'. It has one gear, which is currently started. The gear details show 1 small gear and 1 scale. The cartridge information includes 'PHP 5.3' and 'Web Load Balancer'. Under 'Source Code', there's a URL 'ssh://52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com/~/.git'. The 'Remote Access' section asks if you want to log in to the application. At the bottom, there are links for 'Delete this application...' and 'Continuous Integration' (with Jenkins enabled). A note at the bottom says 'Or, see the entire list of cartridges you can add'.

## Setting the scaling strategy

OpenShift Enterprise allows users the ability to set the minimum and maximum numbers of gears that an application can use to handle increased HTTP traffic. This scaling strategy is exposed via the web console. While on the application details screen, click the *Scales 1 - 25* link to change the default scaling rules.

The screenshot shows the OpenShift Enterprise web console with the application details for 'scaledapp'. The top navigation bar includes 'demo' and tabs for 'Applications', 'Settings', and 'Help'. The main content area is titled 'Scale scaledapp' and shows the cartridge 'PHP 5.3' with 'using 1' gear. It includes instructions for scaling, a control panel with 'Minimum 1' and 'Maximum -1' dropdowns, and a 'Save' button. Below this, there's information about the Web Load Balancer and a link to the HAProxy status page. A note at the bottom encourages reading the scaling guide in the Developer Center.

Change this setting to scale to 5 nodes and click the save button. Verify that the change is reflected in the web console by clicking on your application under the *Applications* tab.

The screenshot shows the OpenShift Enterprise web console with the application details for 'scaledapp-ose.apps.example.com'. The top navigation bar includes 'demo' and tabs for 'Applications', 'Settings', and 'Help'. The main content area shows the application URL 'scaledapp-ose.apps.example.com' with a 'change' link, a creation timestamp, and a status of 'Started 1'. A red box highlights the 'Scales 1 - 5' link. To the right, there are sections for 'Source Code' (SSH URL), 'Remote Access' (log in link), and a 'Delete this application...' button. At the bottom, there are links for 'Databases' (MySQL 5.1, PostgreSQL 9.2) and 'Continuous Integration' (Enable Jenkins).

## Manual scaling

There are often times when a developer will want to disable automatic scaling in order to manually control when a new gear is added to an application. Some examples of when manual scaling may be preferred over automatic scaling could include:

- If you are anticipating a certain load on your application and wish to scale it accordingly.
- You have a fixed set of resources for your application.

OpenShift Enterprise supports this workflow by allowing users to manually add and remove gears for an application. The instructions below describe how to disable the automatic scaling feature. It is assumed you have already created your scaled application as detailed in this lab and are at the root level directory for the application.

From your locally cloned Git repository, create a *disable autoscaling* marker, as shown in the example below:

```
$ touch .openshift/markers/disable_auto_scaling  
$ git add .  
$ git commit -am "remove automatic scaling"  
$ git push
```

To add a new gear to your application, SSH to your application gear with the following command replacing the contents with the correct information for your application. Alternatively, you can use the *rhc app ssh* command.

```
$ ssh [AppUUID]@[AppName]-[DomainName].example.com
```

Once you have been authenticated to your application gear, you can add a new gear with the following command:

```
$ add-gear -a [AppName] -u [AppUUID] -n [DomainName]
```

In this lab, the application name is *scaledapp*, the application UUID is the username that you used to SSH to the node host, and the domain name is *ose*. Given that information, your command should look similar to the following:

```
[scaledapp-ose.example.com ~]\> add-gear -a scaledapp -u 1a6d471841d84e8aaaf25222
```

Verify that your new gear was added to the application by running the *rhc app show* command or by looking at the application details on the web console:

```
$ rhc app show scaledapp
```

After executing this command, you should see the application is now using two gears.

```
scaledapp @ http://scaledapp-ose.apps.example.com/ (uuid: 52b209683a0fb2bc1d0000
-----
Domain: ose
Created: 1:45 PM
Gears: 2 (defaults to small)
Git URL: ssh://52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com/~/.gi
SSH: 52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com
Deployment: auto (on git push)

php-5.3 (PHP 5.3)
-----
Scaling: x2 (minimum: 1, maximum: 5) on small gears

haproxy-1.4 (Web Load Balancer)
-----
Gears: Located with php-5.3
```

The screenshot shows the OpenShift Enterprise application management interface. At the top, there's a navigation bar with tabs for Applications, Settings, and Help. The Applications tab is selected. Below the navigation, the application name 'scaledapp-ose.apps.example.com' is displayed, along with a 'change' link and a timestamp indicating it was created about 1 hour ago in domain 'ose'. To the right of the application name, there's a status indicator showing 'Started' with a value of '2' and a gear icon, which is highlighted with a red box. Further down, under the 'Cartridges' section, there are two entries: 'PHP 5.3' and 'Web Load Balancer'. The 'PHP 5.3' entry has a table with columns: Scales (1-5), Status (Started), Gears (2 small), and Storage (1 GB). This table is also highlighted with a red box. On the right side of the application details, there are sections for 'Source Code' (containing a SSH URL) and 'Remote Access' (with a link to log in). There's also a 'Delete this application...' button. At the bottom left, there's a note: 'Or, see the entire list of cartridges you can add'.

Just as we scaled up with the *add-gear* command, we can manually scale down with the *remove-gear* command. Remove the second gear from your application with the following command making sure to substitute the correct application UUID:

```
[scaledapp-ose.example.com ~]\> remove-gear -a scaledapp -u 1a6d471841d84e8aaaf25
```

After removing the gear with the *remove-gear* command, verify that the application only contains one gear, HAProxy and a single runtime gear:

```
$ rhc app show scaledapp

scaledapp @ http://scaledapp-ose.apps.example.com/ (uuid: 52b209683a0fb2bc1d0000
-----
Domain: ose
Created: 1:45 PM
Gears: 1 (defaults to small)
Git URL: ssh://52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com/~/gi
SSH: 52b209683a0fb2bc1d000030@scaledapp-ose.apps.example.com
Deployment: auto (on git push)

php-5.3 (PHP 5.3)
-----
Scaling: x1 (minimum: 1, maximum: 5) on small gears

haproxy-1.4 (Web Load Balancer)
-----
Gears: Located with php-5.3
```

## Viewing HAProxy information

OpenShift Enterprise provides a dashboard that will give users relevant information about the status of the HAProxy gear that is balancing and managing load between the application gears. This dashboard provides visibility into metrics such as process id, uptime, system limits, current connections, and running tasks. To view the HAProxy dashboard, open your web browser and enter the following URL:

```
http://scaledapp-ose.apps.example.com/haproxy-status/
```

**HAProxy version 1.4.22, released 2012/08/09**

**Statistics Report for pid 18924**

> General process information

```
pid = 18924 (process #1, haproxy = 1)
uptime = 0d 00m03s
system limits: memmax = unlimited, ulimit = 526
maxsock = 526, maxconn = 256, maxpipes = 0
current conn = 0, current pipes = 0/0
Running tasks: 1/2
```

Legend:  
■ active UP  
■ active UP, going down  
■ active DOWN, going up  
■ active or backup DOWN  
■ backup UP  
■ backup UP, going down  
■ backup DOWN, going up  
■ not checked  
■ active or backup DOWN for maintenance (MAINT)

Display option: [Hide DOWN servers](#) [Refresh now](#) [Updates \(x1,4\)](#)  
[OK](#) [Cancel](#) [Online manual](#)

stats												Server																													
Queue			Session rate			Sessions			Bytes			Denied			Errors			Warnings			LastChk			Wght			Act			Sick			Chk			Dens			DenTime		
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LtTot	In	Out	Req	Resp	Req	Resp	Conn	Resp	Reir	Recls	Status	LastChk	Wght	Act	Sick	Chk	Dens	DenTime	Throttle												
Frontend			1	1	-	1	1	1	128	2	189	978	0	0	0	0	0	0	0	0	OPEN																				
Backend	0	0	0	0	0	0	0	0	128	0	189	978	0	0	0	0	0	0	0	0	53s UP		0	0	0	0	0	0	0	0	0	0									

express												Server																													
Queue			Session rate			Sessions			Bytes			Denied			Errors			Warnings			LastChk			Wght			Act			Sick			Chk			Dens			DenTime		
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LtTot	In	Out	Req	Resp	Req	Resp	Conn	Resp	Reir	Recls	Status	LastChk	Wght	Act	Sick	Chk	Dens	DenTime	Throttle												
Frontend	0	0	-	0	0	0	0	0	128	0	0	0	0	0	0	0	0	0	0	0	OPEN																				
local-gear	0	0	-	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	53s UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-	0s	0s	0s									
Backend	0	0	0	0	0	0	0	0	128	0	0	0	0	0	0	0	0	0	0	0	53s UP		1	1	0	0	0	0	0	0	0	0									

**Lab 15 Complete!**

# Lab 16: The DIY application type (Estimated time: 10 minutes)

---

## Server used:

- localhost

## Tools used:

- rhc
- git

In addition to supporting Ruby, PHP, Perl, Python, and Java EE6, the OpenShift Enterprise environment supports the “Do it Yourself” or “DIY” application type. Using this application type, users can run just about any program that speaks HTTP.

How this works is remarkably straightforward. The OpenShift Enterprise execution environment is a carefully secured Red Hat Enterprise Linux operating system on x64 systems. Thus, OpenShift Enterprise can run any binary that will run on RHEL 6.3 x64.

The way that OpenShift Enterprise DIY runtimes interfaces your application to the outside world is by creating an HTTP proxy specified by the environment variables *OPENSHIFT\_DIY\_IP* and *OPENSHIFT\_DIY\_PORT*. All your application has to do is bind and listen on that address and port. HTTP requests will come into the OpenShift Enterprise environment, which will proxy those requests to your application. Your application will reply with HTTP responses, and the OpenShift Enterprise environment will relay those responses back to your users.

Your application will be executed by the `.openshift/action_hooks/start` script, and will be stopped by the `.openshift/action_hooks/stop` script.

**Note:** DIY applications are unsupported but is a great way for developers to try out unsupported languages, frameworks, or middleware that doesn’t ship as an official OpenShift Enterprise cartridge.

## Creating a DIY application type

To create an application gear that will use the DIY application type, use the *rhc app create* command:

```
$ rhc app create myjavademo diy
```

After executing this command, you should see the following output:

```
Using diy-0.1 (Do-It-Yourself 0.1) for 'diy'
```

#### Application Options

---

```
Domain:      ose
Cartridges:  diy-0.1
Gear Size:   default
Scaling:     no
```

```
Creating application 'myjavademo' ... done
```

```
Disclaimer: This is an experimental cartridge that provides a way to try unsup-
```

```
Waiting for your DNS name to be available ... done
```

```
Cloning into 'myjavademo'...
```

```
The authenticity of host 'myjavademo-ose.apps.example.com (209.132.178.87)' can't be established.
RSA key fingerprint is e8:e2:6b:9d:77:e2:ed:a2:94:54:17:72:af:71:28:04.
```

```
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'myjavademo-ose.apps.example.com' (RSA) to the list of known hosts.
Checking connectivity... done
```

```
Your application 'myjavademo' is now available.
```

```
URL:          http://myjavademo-ose.apps.example.com/
SSH to:       52b21e0c3a0fb277cf0000c8@myjavademo-ose.apps.example.com
Git remote:   ssh://52b21e0c3a0fb277cf0000c8@myjavademo-ose.apps.example.com/~/git/myjavademo.git
Cloned to:    /Users/gshipley/code/ose/scaledapp/myjavademo
```

```
Run 'rhc show-app myjavademo' for more details about your app.
```

Now that we have our application created, we can begin the deployment of our custom runtime. Let's start by changing to the application directory:

```
$ cd myjavademo
```

## Deploying application code

Instead of spending time in this lab with writing a server runtime, we are going to use an existing one that is available on the OpenShift github page. This application code is written in Java and consists of a single MyHttpServer main class. Since this source code lives on the github OpenShift project page, we need to add the remote github repository and then pull the remote source code while at the same time overwriting the existing source code we have in our DIY application directory.

```
$ git remote add upstream git://github.com/openshift/openshift-diy-java-demo.git
$ git pull -s recursive -X theirs upstream master
$ git push
```

## Verify the DIY application is working

Once the java example has been pushed to your OpenShift Enterprise gear, open up a web browser and point to the following URL:

```
http://myjavademo-ose.apps.example.com/index.html
```

**Note:** Make sure to include the index.html file at the end of the URL.

If the application was deployed correctly, you should see a *Hello DIY World!* message. This little http java server will serve any files found in your application's html directory, so you can add files or make changes to them, push the contents and see those reflected in your browser.

## Under the covers

The DIY cartridge provides a number of hooks that are called during the lifecycle actions of the application. The hooks available to you for customization are found in the .openshift/action\_hooks directory of your application repository.

For this application, all that has been customized are the start and stop scripts. They simply launch the MyHttpServer class using Java, and performs a *wget* call to have the MyHttpServer stop itself:

```
cat .openshift/action_hooks/start
#!/bin/bash
# The logic to start up your application should be put in this
# script. The application will work only if it binds to
# $OPENSHIFT_INTERNAL_IP:8080

cd $OPENSHIFT_REPO_DIR
nohup java -cp bin test.MyHttpServer >${OPENSHIFT_DIY_LOG_DIR}/MyHttpServer.log

[24](ironmaiden:diy) > cat .openshift/action_hooks/stop
#!/bin/bash
# The logic to stop your application should be put in this script.
wget http://${OPENSHIFT_INTERNAL_IP}:${OPENSHIFT_INTERNAL_PORT}?action=stop
```

See the *src/test/MyHttpServer.java* source to understand how the Java application is making use of the OpenShift Enterprise environment variables to interact with the server environment.

Add another source file to your application and verify that it works. From inside of your *myjavademo* directory, create a new file called *test.html*.

```
$ cd html  
$ echo "Hello from DIY on Enterprise" > test.html  
$ git add .  
$ git commit -am "Adding new HTML file"  
$ push
```

Verify that you can view this file by going to the following URL:

```
http://myjavademo-ose.apps.example.com/test.html
```

**Lab 16 Complete!**

# Lab 17: Developing Java EE applications using JBoss EAP (Estimated time: 30 minutes)

---

## Server used:

- localhost

## Tools used:

- rhc
- git
- curl

OpenShift Enterprise provides the JBoss EAP runtime to facilitate the development and deployment of Java EE 6 applications.

JBoss Enterprise Application Platform 6 (JBoss EAP 6) is a fully compliant Java EE 6 platform which includes a subscription model with long-term support, platform certification, service packs and SLA(s). In this lab we will build a simple todo application using Java EE 6 deployed on the JBoss EAP platform. The application will have a single entity called Todo and will persist todos to PostgreSQL using JPA. The application will also use EJB 3.1 Stateless session beans, Context and Dependency Injection (or CDI), and JAX RS for exposing RESTful web services.

## Create a JBoss EAP application

In order to create a JBoss EAP application runtime, enter in the following command:

```
$ rhc app create todo jbosseap
```

Just as we saw in previous labs, a template has been deployed for you at the following URL:

```
http://todo-ose.apps.example.com
```

You should see the following output:

```
Using jbosseap-6 (JBoss Enterprise Application Platform 6.1.0) for 'jbosseap'
```

#### Application Options

---

```
Domain:      ose
Cartridges:  jbosseap-6
Gear Size:   default
Scaling:     no
```

```
Creating application 'todo' ... done
```

```
Waiting for your DNS name to be available ... done
```

```
Cloning into 'todo'...
```

```
The authenticity of host 'todo-ose.apps.example.com (209.132.178.87)' can't be e
RSA key fingerprint is e8:e2:6b:9d:77:e2:ed:a2:94:54:17:72:af:71:28:04.
```

```
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'todo-ose.apps.example.com' (RSA) to the list of known hosts.
```

```
Checking connectivity... done
```

```
Your application 'todo' is now available.
```

```
URL:          http://todo-ose.apps.example.com/
SSH to:       52b220d23a0fb277cf0000e9@todo-ose.apps.example.com
Git remote:   ssh://52b220d23a0fb277cf0000e9@todo-ose.apps.example.com/~/git/todo
Cloned to:   /Users/gshipley/code/ose/scaledapp/myjavademo/src/test/todo
```

```
Run 'rhc show-app todo' for more details about your app.
```

Verify that the application has been deployed and the template is displaying correctly in your web browser.

```
http://todo-ose.apps.example.com
```



## Welcome To OpenShift, JBossEAP6.0 Cartridge

Place your application here

In order to commit to your new project, go to your projects git repo (created with the rhc app create command) directory.

For example, if you named your application myfirstapp (by passing in -a myfirstapp to the rhc app create command), you would find the contents of this app located under myfirstapp/src/main/webapp. You can edit that and push your changes from the myfirstapp directory by running:

```
git commit -a -m 'Some commit message'  
git push
```

Then reload this page.

See the myfirstapp/README file for more information on the options for deploying applications.

### Sample Applications

To get started you can either modify the default war or try one of these samples:

<https://github.com/openshift/kitchensink-example> this quickstart showcases some of the exciting Java EE6 features available on JBoss AS 7.1.

<https://github.com/openshift/kitchensink-html5-mobile-example> based on kitchensink (above), but with an HTML5 client that displays great on a PC or mobile device.

<https://github.com/openshift/tweetstream-example> a TweetStream example

Example usage:

## Additional marker files for JBoss EAP

If you recall from a previous lab, we discussed the way that OpenShift Enterprise allows the developer to control and manage some of the runtime features using marker files. For Java based deployments, there are additional marker files that a developer needs to be aware of:

- enable\_jpda - Will enable the JPDA socket based transport on the JVM running the JBoss EAP application server. This enables you to remotely debug code running inside of the JBoss application server.
- skip\_maven\_build - Maven build step will be skipped
- force\_clean\_build - Will start the build process by removing all non essential Maven dependencies. Any current dependencies specified in your pom.xml file will then be re-downloaded.
- hot\_deploy - Will prevent a JBoss container restart during build/deployment. Newly built archives will be re-deployed automatically by the JBoss HDSscanner component.
- java7 - Will run JBoss EAP with Java7 if present. If no marker is present then the baseline Java version will be used (currently Java6)

## Deployment directory

If you list the contents of the application repository that was cloned to your local machine, you will notice a deployments directory. This directory is a location where a developer can place binary archive files, .ear files for example, for deployment. If you want to deploy a .war file rather than pushing source code, copy the .war file to deployments directory, add the .war file to your git repository, commit the change, and then push the content to your OpenShift Enterprise server.

## Maven

OpenShift Enterprise uses the Maven build system for all Java projects. Once you add new source code following the standard Maven directory structure, OpenShift Enterprise will recognize the existing *pom.xml* in your applications root directory in order to build the code remotely.

The most important thing specified in the *pom.xml* file is a Maven profile named *openshift*. This is the profile which is invoked when you do deploy the code to OpenShift Enterprise.

## Embed PostgreSQL cartridge

The *todo* sample application that we are going to write as part of this lab will make use of the PostgreSQL 8.4 database. Using the information that you have learned from previous labs, add the PostgreSQL 8.4 cartridge to the *todo* application.

## Building the *todo* application

At this point, we should have an application named *todo* created as well as having PostgreSQL embedded in the application to use as our datastore. Now we can begin working on the application.

### Creating Domain Model

---

**Note:** The source code for this application is available on github at the following URL:

```
https://github.com/gshipley/todo-javaee6
```

If you want the easy way out, use the information you have learned from a previous lab to add the above repository as a remote repository and then pull in the source code while overwriting the existing template. (Hint: The previous lab)

The first thing that we have to do is to create the domain model for the *todo application*. The application will have a single entity named *Todo* as shown below. The entity shown below is a simple JPA entity with JPA and bean validation annotations. Create a source file named *Todo.java* in the *todo/src/main/java/com/todo/domain* directory with the following contents:

```
package com.todo.domain;
```

```
import java.util.Date;
import java.util.List;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

@Entity
public class Todo {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotNull
    @Size(min = 10, max = 40)
    private String todo;

    @ElementCollection(fetch=FetchType.EAGER)
    @CollectionTable(name = "Tags", joinColumns = @JoinColumn(name = "todo_id"))
    @Column(name = "tag")
    @NotNull
    private List<String> tags;

    @NotNull
    private Date createdOn = new Date();

    public Todo(String todo) {
        this.todo = todo;
    }

    public Todo() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

```
}

public String getTodo() {
    return todo;
}

public void setTodo(String todo) {
    this.todo = todo;
}

public Date getCreatedOn() {
    return createdOn;
}

public void setCreatedOn(Date createdOn) {
    this.createdOn = createdOn;
}

public void setTags(List<String> tags) {
    this.tags = tags;
}

public List<String> getTags() {
    return tags;
}

@Override
public String toString() {
    return "Todo [id=" + id + ", todo=" + todo + ", tags=" + tags
           + ", createdOn=" + createdOn + "]";
}

}
```

## Create the `persistence.xml` file

The `persistence.xml` file is a standard configuration file in JPA that defines your data source. It has to be included in the `META-INF` directory inside of the JAR file that contains the entity beans. The `persistence.xml` file must define a persistence-unit with a unique name. Create a `META-INF` directory under `src/main/resources` and then create the `persistence.xml` file with the contents below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.s
version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

    <persistence-unit name="todos" transaction-type="JTA">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>java:jboss/datasources/PostgreSQLDS</jta-data-s
        <class>com.todo.domain.Todo</class>
        <properties>
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="create" />
        </properties>

    </persistence-unit>
</persistence>
```

The *jta-data-source* refers to JNDI name preconfigured by OpenShift Enterprise in the standalone.xml file located in the *.openshift/config* directory.

### Create the TodoService EJB bean

---

Next we will create a stateless EJB bean named *TodoService* in the *com.todo.service* package. This bean will perform basic CRUD operations using *javax.persistence.EntityManager*. Create a file named *TodoService* in the *src/main/java/com/todo/service* directory and add the following contents:

```

package com.todo.service;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import com.todo.domain.Todo;

@Stateless
public class TodoService {

    @PersistenceContext
    private EntityManager entityManager;

    public Todo create(Todo todo) {
        entityManager.persist(todo);
        return todo;
    }

    public Todo find(Long id) {
        Todo todo = entityManager.find(Todo.class, id);
        List<String> tags = todo.getTags();
        System.out.println("Tags : " + tags);
        return todo;
    }
}

```

## Enable CDI

CDI or Context and Dependency Injection is a Java EE 6 specification which enables dependency injection in a Java EE 6 project. To enable CDI in the *todo* project, create a *beans.xml* file in *src/main/webapp/WEB-INF* directory with the following contents:

```

<?xml version="1.0"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http:

```

In order to use the *@Inject* annotation instead of the *@Ejb* annotation to inject an EJB, you will have to write a producer which will expose the *EntityManager*. Create a source file in the *src/main/java/com/todo/utils* directory named *Resources* and add the following source code:

```
package com.todo.utils;

import javax.enterprise.inject.Produces;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

public class Resources {

    @Produces
    @PersistenceContext
    private EntityManager em;

}
```

## Creating a RESTful web service

---

Before exposing a RESTful web service for the *Todo* entity, we need to enable JAX-RS in our application. To enable JAX-RS, create a class which extends *javax.ws.rs.core.Application* and specify the application path using a *javax.ws.rs.ApplicationPath* annotation. Create a source file named *JaxRsActivator* in the *src/main/java/com/todo/rest* directory and add the following source code:

```
package com.todo.rest;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest")
public class JaxRsActivator extends Application {
    /* class body intentionally left blank */
}
```

Next we will create a *TodoRestService* class which will expose two methods that will create and read a *Todo* object. The service will consume and produce JSON. Create a source file named *TodoRestService* in the *src/main/java/com/todo/rest* directory and add the following source code:

```

package com.todo.rest;

import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;
import com.todo.domain.Todo;
import com.todo.service.TodoService;

@Path("/todos")
public class TodoRestService {

    @Inject
    private TodoService todoService;

    @POST
    @Consumes("application/json")
    public Response create(Todo entity) {
        todoService.create(entity);
        return Response.created(
            UriBuilder.fromResource(TodoRestService.class)
                .path(String.valueOf(entity.getId())).build()).build();
    }

    @GET
    @Path("/{id:[0-9][0-9]*}")
    @Produces(MediaType.APPLICATION_JSON)
    public Todo lookupTodoById(@PathParam("id") long id) {
        Todo todo = todoService.find(id);
        if (todo == null) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        return todo;
    }
}

```

## Deploy the *todo* application to OpenShift Enterprise

Now that we have our application created, we need to push our changes to the OpenShift Enterprise

gear that we created earlier in this lab. From the application root directory, issue the following commands:

```
$ git add .
$ git commit -am "Adding source code"
$ git push
```

Once you execute the *git push* command, the application will begin building on the OpenShift Enterprise node host. During this training class, the OpenStack virtual machines we have created are not production grade environments. Because of this, the build process will take some time to complete. Sit back, be patient, and help your fellow classmates who may be having problems.

## Testing the *todo* application

In order to test out the RESTful web service that we created in this lab, we can add and retrieve todo items using the *curl* command line utility. To add a new item, enter the following command:

```
$ curl -k -i -X POST -H "Content-Type: application/json" -d '{"todo": "Sell a lot'...
```

To list all available todo items, run the following command:

```
$ curl -k -i -H "Accept: application/json" https://todo-ose.apps.example.com/res...
```

You should see the following output:

```
HTTP/1.1 200 OK
Date: Fri, 25 Jan 2013 04:05:51 GMT
Server: Apache-Coyote/1.1
Content-Type: application/json
Connection: close
Transfer-Encoding: chunked

{"id":1,"todo": "Sell a lot of OpenShift Enterprise","tags":["javascript","ui"],..."
```

If you downloaded and deployed the source code from the git repository, the project contains a JSF UI component which will allow you to test the application using your web browser. Simply point your browser to

```
http://todo-ose.apps.example.com
```

to verify that the application was deployed correctly.

The screenshot shows a web application titled "Start creating your Todos". At the top, it says "Todo Application Running on OpenShift Enterprise". Below this, there's a "Todo List Creation" section with fields for "Todo" and "Tags", and a "Create New Todo" button. Underneath, a table titled "Created Todos" lists one item: "Sell a lot of OpenShift Enterprise" with ID 1, created on 2013-12-18 18:01:22.505, and tags [javascript, ui]. The entire "Created Todos" table is highlighted with a red border.

ID	Name	Created On	Tags
1	Sell a lot of OpenShift Enterprise	2013-12-18 18:01:22.505	[javascript, ui]

Powered By OpenShift

## Extra Credit

SSH into the application gear and verify the todo item was added to the PostgreSQL database.

**Lab 17 Complete!**

# Lab 18: Using Jenkins continuous integration

## (Estimated time: 30 minutes)

---

### Server used:

- localhost
- node host

### Tools used:

- rhc
- git
- yum

Jenkins (<https://wiki.jenkins-ci.org>) is a full featured continuous integration (CI) server that can run builds, tests, and other scheduled tasks. OpenShift Enterprise allows you to integrate Jenkins with your OpenShift Enterprise applications.

With Jenkins, you have access to a full library of plugins (<https://wiki.jenkins-ci.org/display/JENKINS/Plugins>) and a vibrant, thriving community of users who have discovered a new way to do development.

There are many reasons why you would want to leverage Jenkins as a continuous integration server. In the context of OpenShift Enterprise, some of the benefits are:

- Archived build information
- No application downtime during the build process
- Failed builds do not get deployed (leaving the previous working version in place)
- More resources to build your application as each Jenkins build spins up a new gear for short lived period of time

Jenkins includes a feature-rich web user interface that provides the ability to trigger builds, customize builds, manage resources, manage plugins, and many other features.

### Verify Jenkins cartridges are installed

SSH to your node host and verify that you have the Jenkins cartridges installed:

**Note:** Execute the following on the node host

```
# rpm -qa |grep jenkins
```

You should see the following four packages installed:

- jenkins-1.509.1-1.el6op.noarch
- openshift-origin-cartridge-jenkins-1.16.1-2.el6op.noarch
- jenkins-plugin-openshift-0.6.25-1.el6op.x86\_64
- openshift-origin-cartridge-jenkins-client-1.17.1-2.el6op.noarch

If you do now have the above RPM packages installed on your node host, follow the directions in a previous to install the Jenkins packages. Make sure to clear the cache on the broker host after installing the new packages.

## Create a Jenkins gear

In order to use Jenkins on OpenShift Enterprise, you will need to create an application gear that contains the Jenkins application. This is done using the *rhc app create* command line tool, or you can use the web console to create the application. The syntax for using the command line tool is as follows:

```
$ rhc app create jenkins jenkins
```

You should see the following output from this command:

```
Using jenkins-1 (Jenkins Server) for 'jenkins'
```

#### Application Options

---

```
Domain:      ose
Cartridges:  jenkins-1
Gear Size:   default
Scaling:     no
```

```
Creating application 'jenkins' ... done
```

```
Jenkins created successfully. Please make note of these credentials:
```

```
User: admin
Password: H8j5TBjglT8x
```

```
Note: You can change your password at: https://jenkins-ose.apps.example.com/me/
```

```
Waiting for your DNS name to be available ... done
```

```
Cloning into 'jenkins'...
```

```
The authenticity of host 'jenkins-ose.apps.example.com (209.132.178.87)' can't be
RSA key fingerprint is e8:e2:6b:9d:77:e2:ed:a2:94:54:17:72:af:71:28:04.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'jenkins-ose.apps.example.com' (RSA) to the list of known hosts.
Checking connectivity... done
```

```
Your application 'jenkins' is now available.
```

```
URL:          http://jenkins-ose.apps.example.com/
SSH to:       52b22a9f3a0fb277cf0001bd@jenkins-ose.apps.example.com
Git remote:   ssh://52b22a9f3a0fb277cf0001bd@jenkins-ose.apps.example.com/~/git/
Cloned to:   /Users/gshipley/code/oso/jenkins
```

```
Run 'rhc show-app jenkins' for more details about your app.
```

Make a note of the user username and password that was created for you by OpenShift Enterprise.

## Adding Jenkins support to your application

Now that we have a Jenkins server setup and running, we can add support to our *todo* application which will allow all futures builds to compile on the Jenkins server. To embed the Jenkins support cartridge in your application, use the following command:

```
$ rhc cartridge-add jenkins-client -a todo
```

The output should be the following:

```
Using jenkins-client-1 (Jenkins Client) for 'jenkins-client'
Adding jenkins-client-1 to application 'todo' ... done

jenkins-client-1 (Jenkins Client)
-----
Gears: Located with jbosseap-6, postgresql-8.4
Job URL: https://jenkins-ose.apps.example.com/job/todo-build/

Associated with job 'todo-build' in Jenkins server.
```

Verify that the Jenkins client was added to your application by running the following command:

```
$ rhc app show todo
```

You should see the following information indicating that Jenkins has been enabled for the *todo* application.

```
todo @ http://todo-ose.apps.example.com/ (uuid: 52b228013a0fb277cf000197)
-----
Domain: ose
Created: 3:56 PM
Gears: 1 (defaults to small)
Git URL: ssh://52b228013a0fb277cf000197@todo-ose.apps.example.com/~/git/todo
SSH: 52b228013a0fb277cf000197@todo-ose.apps.example.com
Deployment: auto (on git push)

jbosseap-6 (JBoss Enterprise Application Platform 6.1.0)
-----
Gears: Located with postgresql-8.4, jenkins-client-1

postgresql-8.4 (PostgreSQL 8.4)
-----
Gears: Located with jbosseap-6, jenkins-client-1
Connection URL: postgresql://$OPENSHIFT_POSTGRESQL_DB_HOST:$OPENSHIFT_POSTGR...
Database Name: todo
Password: 5uyxIvYNd5s6
Username: admindrgdkpd

jenkins-client-1 (Jenkins Client)
-----
Gears: Located with jbosseap-6, postgresql-8.4
Job URL: https://jenkins-ose.apps.example.com/job/todo-build/
```

# Configuring Jenkins

Open up a web browser and point to the following URL:

```
https://jenkins-ose.apps.example.com/job/todo-build/
```

Authenticate to the Jenkins environment by providing the username and password that was displayed after adding the Jenkins application.



Once you are authenticated to the Jenkins dashboard, click on the configure link:

A screenshot of the Jenkins project configuration page for 'todo-build'. The top navigation bar shows 'Jenkins &gt; todo-build'. The main content area is titled 'Project todo-build' and contains sections for 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', and 'Configure'. The 'Configure' link is highlighted with a red box. Below these are links for 'Workspace' and 'Recent Changes', and a section for 'Permalinks'. On the right side, there are buttons for 'add description' and 'Disable Project'. The bottom of the page includes links for 'Help us localize this page', 'Page generated: Jan 25, 2013 2:19:56 PM', 'REST API', and 'Jenkins ver. 1.488'.

A few interesting configuration items exist that may come in handy in the future:

**Builder Configuration:** The first interesting configuration is concerned with the builder. The configuration below states that Jenkins should create a builder with a small size gear using the JBoss EAP cartridge and that the Jenkins master will wait for 5 minutes for the slave to come online.

Builder Size	Small	(?)
Builder Timeout	300000	(?)
Builder Type	jbosseap-6.0	(?)

**Git Configuration:** The next configuration item of interest is the git SCM URL. It specifies the URL of the git repository to use, the branch to use, etc. This section is important if you want to use Jenkins to build a project which exists outside of OpenShift Enterprise. This would be useful for developers who have an internal repo for their source code that they would prefer to build from.

**Build Configuration:** The last configuration item which is interesting is under the *build* section. Here you can specify a shell script for building the project. For our current builder it does the following:

- Specify if the project should be built using Java 6 or Java 7
- Specify XMX memory configuration for maven and build the maven project. The memory it configures is 396M.
- Deploying the application which includes stopping the application, pushing the content back from Jenkins to the application gear(s), and finally deploying the artifacts.

The source code for the default build script is as follows:

```

source /usr/libexec/openshift/cartridges/abstract/info/lib/jenkins_util

jenkins_rsync 4d1b096e414243e9833dad55d774de73@todo-ose.example.com:~/m2/ ~/m2

# Build setup and run user pre_build and build
. ci_build.sh

if [ -e ${OPENSHIFT_REPO_DIR}.openshift/markers/java7 ];
then
    export JAVA_HOME=/etc/alternatives/java_sdk_1.7.0
else
    export JAVA_HOME=/etc/alternatives/java_sdk_1.6.0
fi

export MAVEN_OPTS="$OPENSHIFT_MAVEN_XMX"
mvn --global-settings $OPENSHIFT_MAVEN_MIRROR --version
mvn --global-settings $OPENSHIFT_MAVEN_MIRROR clean package -Popenshift -DskipTests

# Deploy new build

# Stop app
jenkins_stop_app 4d1b096e414243e9833dad55d774de73@todo-ose.example.com

# Push content back to application
jenkins_sync_jboss 4d1b096e414243e9833dad55d774de73@todo-ose.example.com

# Configure / start app
$GIT_SSH 4d1b096e414243e9833dad55d774de73@todo-ose.example.com deploy.sh

jenkins_start_app 4d1b096e414243e9833dad55d774de73@todo-ose.example.com

$GIT_SSH 4d1b096e414243e9833dad55d774de73@todo-ose.example.com post_deploy.sh

```

## Deploying code to Jenkins

Now that you have the Jenkins client embedded into your *todo* application gear, any future *git push* commands will send the code to the Jenkins server for building. To test this out, edit the *src/main/webapp/todo.xhtml* source file and change the title of the page. If you do not have this file, just create a new file instead. Look for the following code block:

```
<h2>Todo List Creation</h2>
```

Change the above code to the following:

```
<h2>Todo List Creation using Jenkins</h2>
```

Commit and push your change:

```
$ git commit -am "changed h2"  
$ git push
```

After you push your changes to the Jenkins server, you should see the following output:

```
Counting objects: 5, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 282 bytes, done.  
Total 3 (delta 2), reused 0 (delta 0)  
remote: restart_on_add=false  
remote: Executing Jenkins build.  
remote:  
remote: You can track your build at https://jenkins-ose.apps.example.com/job/todo...  
remote:  
remote: Waiting for build to schedule....Done  
remote: Waiting for job to complete.....  
remote: SUCCESS  
remote: New build has been deployed.  
To ssh://4d1b096e414243e9833dad55d774de73@todo-ose.apps.example.com/~/git/todo.g...  
eb5f9dc..8cee826 master -> master
```

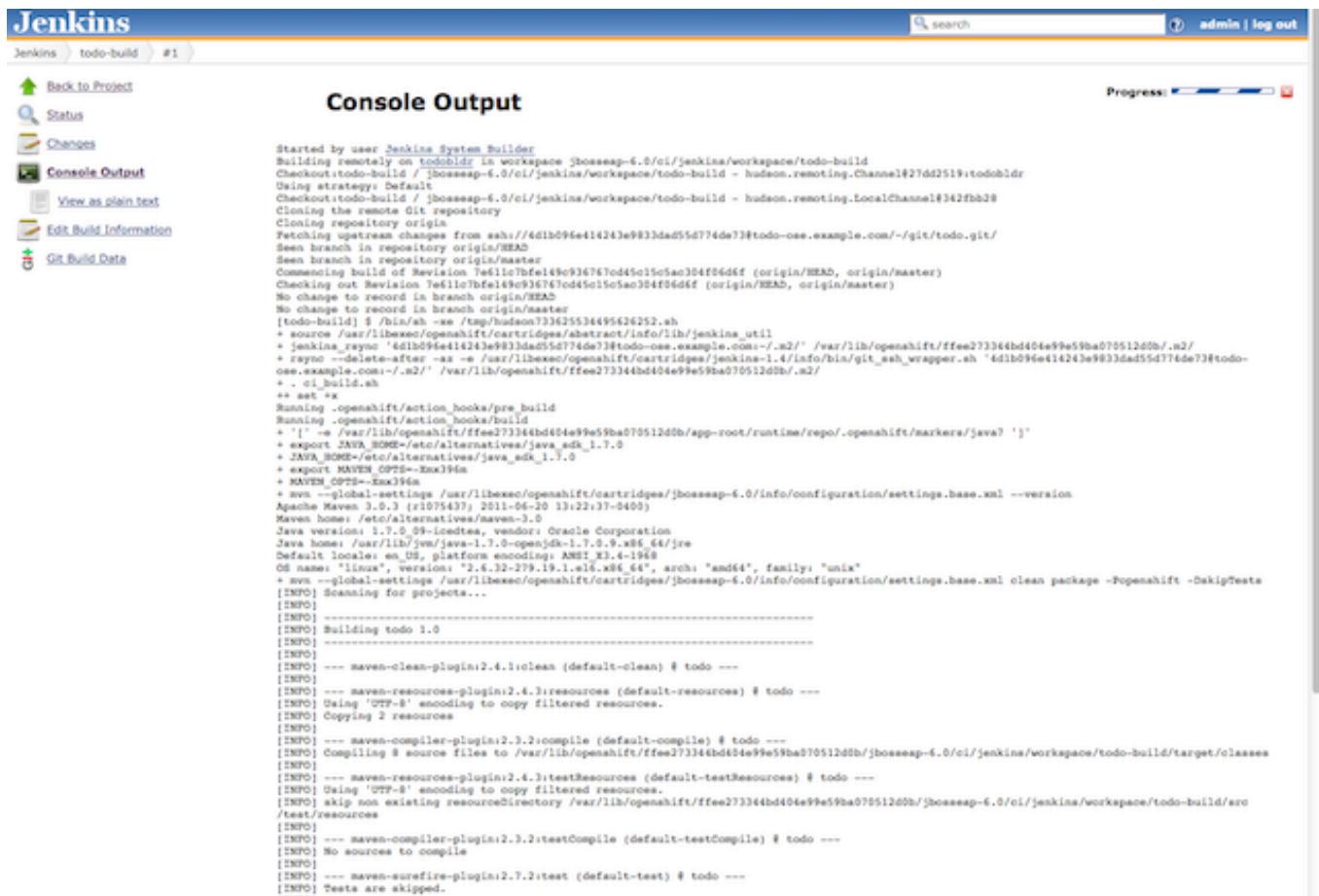
While the build is happening, open up a new terminal window and run the following command:

```
$ rhc domain show
```

You will see a new gear that was created by the Jenkins application. This new gear is a temporary gear that OpenShift Enterprise creates in order to build your application code.

```
todobldr @ http://todobldr-ose.apps.example.com/  
=====  
Application Info  
=====  
UUID      = ffee273344bd404e99e59ba070512d0b  
Git URL   =  
ssh://ffee273344bd404e99e59ba070512d0b@todobldr-ose.apps.example.com/~/git/todob...  
SSH URL   = ssh://ffee273344bd404e99e59ba070512d0b@todobldr-ose.apps.example.com/~/gi...  
Gear Size = small  
Created   = 2:48 PM  
Cartridges  
=====  
jbossseap-6.0
```

If the build fails, or if you just want to see the output of the Maven build process, you can log in to your Jenkins application, click on the build, and then click the link to view the console output. Log in to your Jenkins application and view the contents of the last build.



The screenshot shows the Jenkins interface with the 'Console Output' tab selected. The output window displays the command-line logs of the Maven build process. The logs show the checkout of the 'todo-build' repository from 'jbosseap-6.0.0-rc1/jenkins/workspace/todo-build'. It then fetches upstream changes from 'ash://4d1b096e414243e9833dad55d774de73#todo-ose.example.com/-/git/todo.git'. The build continues with various Maven commands like 'mvn clean', 'mvn compile', and 'mvn test', along with other system and Java environment details. The progress bar at the top right indicates the build is complete.

```
Started by user Jenkins System Builder
Building remotely on todoBuilder in workspace /jenkins/workspace/todo-build
Checkout:todo-build / jbosseap-6.0.0-rc1/jenkins/workspace/todo-build - hudson.remoting.Channel@27dd2519:todoBuilder
Using strategy: Default
Checkout:todo-build / jbosseap-6.0.0-rc1/jenkins/workspace/todo-build - hudson.remoting.LocalChannel@342fbba2
Cloning the remote Git repository
Cloning repository origin
Fetching upstream changes from ash://4d1b096e414243e9833dad55d774de73#todo-ose.example.com/-/git/todo.git
Seen branch in repository origin/HEAD
Seen branch in repository origin/master
Comparing build of Revision 7ef1c7fe149c934767od45a15c5ac304f04d6ff (origin/HEAD, origin/master)
Checking out Revision 7ef1c7fe149c934767od45a15c5ac304f04d6ff (origin/HEAD, origin/master)
No change to record in branch origin/HEAD
No change to record in branch origin/master
[todo-build] $ /bin/sh -xe /tmp/hudson733625534495626252.sh
+ source /usr/libexec/openshift/cartridges/abstract/info/lib/jenkins_util
+ jenkins_resync '4d1b096e414243e9833dad55d774de73#todo-ose.example.com/-/m2/' '/var/lib/openshift/ffee273344bd404e99e59ba075512d0b/.m2/'
+ rayon--delete-after -ax -e /var/libexec/openshift/cartridges/jenkins-1.4/info/bin/git_ssh_wrapper.sh '4d1b096e414243e9833dad55d774de73#todo-ose.example.com/-/m2/' '/var/lib/openshift/ffee273344bd404e99e59ba075512d0b/.m2/'
+ . ci_build.sh
+ set -x
Starting openshift/action_hooks/pre-build
Running openshift/action_hooks/build
+ '[' -e /var/lib/openshift/ffee273344bd404e99e59ba075512d0b/app-root/runtime/repo/.openshift/markers/java7 ']'
+ export JAVA_HOME=/etc/alternatives/java_sdk-1.7.0
+ export MAVEN_OPTS=-Xmx394m
+ mvn --global-settings /usr/libexec/openshift/cartridges/jbosseap-6.0/info/configuration/settings.base.xml --version
Apache Maven 3.0.3 (r1075437; 2011-06-20 13:22:37-0400)
Java version: 1.7.0_09-b13, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.9.x86_64/jre
Default locale: en_US, platform encoding: ASCII_X3.4-1998
OS name: "linux", version: "2.6.32-279.19.1.el6.x86_64", arch: "amd64", family: "unix"
+ mvn --global-settings /usr/libexec/openshift/cartridges/jbosseap-6.0/info/configuration/settings.base.xml clean package -Popenshift -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building todo 1.0
[INFO] -----
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) # todo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) # todo ---
[INFO] Compiling 8 source files to /var/lib/openshift/ffee273344bd404e99e59ba075512d0b/jbosseap-6.0/ci/jenkins/workspace/todo-build/target/classes
[INFO] --- maven-resources-plugin:2.4.3:resources (default-resources) # todo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/openshift/ffee273344bd404e99e59ba075512d0b/jbosseap-6.0/ci/jenkins/workspace/todo-build/src/test/resources
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) # todo ---
[INFO] No sources to compile
[INFO] --- maven-surefire-plugin:2.7.2:test (default-test) # todo ---
[INFO] Tests are skipped.
```

## Starting a new build

One of the great things about integrating your application with the Jenkins CI environment is the ability to start a new build without having to modify and push your source code. To initiate a new build, log in to the Jenkins dashboard and select the *todo* builder. Point your browser to:

```
https://jenkins-ose.apps.example.com/
```

Once you have been authenticated, click the *todo-build* link:

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Job', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Below that is a 'Build Queue' section stating 'No builds in the queue.' To the right is a main panel titled 'todo-build'. It has a table with columns: 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. A row for 'todo-build' is shown with 'Name' highlighted by a red box. Below the table is a legend with three RSS feed icons: 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'. At the bottom, there's a note 'Help us localize this page' and footer text 'Page generated: Jan 25, 2013 3:42:13 PM REST API Jenkins ver. 1.488'.

This will place you on the *todo* application builder dashboard. Click the *Build Now* link on the left hand side of the screen to initiate a new build:

The screenshot shows the 'Project todo-build' dashboard. On the left, there's a sidebar with links like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now' (which is highlighted with a red box), 'Delete Project', 'Configure', and 'Build History' (with a '(trend)' link). Below that is a 'Recent Changes' section. The main area is titled 'Project todo-build' and contains sections for 'Workspace', 'Last Successful Artifacts' (listing 'gitkeep', 'ROOT.war', and 'ROOT.war.failed'), and 'Recent Changes'. At the bottom, there's a 'Permalinks' section with a list of build links and 'RSS for all' and 'RSS for failures' links. The footer includes 'Help us localize this page' and 'Page generated: Jan 25, 2013 3:43:47 PM REST API Jenkins ver. 1.488'.

After you click the *Build Now* link, a new build will show up under the links on the left hand side of the screen.

The screenshot shows the 'Build History' section. It lists five builds: #5 (Jan 25, 2013 3:45:22 PM), #4 (Jan 25, 2013 3:31:40 PM), #3 (Jan 25, 2013 3:26:42 PM), and #2 (Jan 25, 2013 2:53:04 PM). Each build entry has a blue circle icon. At the bottom, there are 'RSS for all' and 'RSS for failures' links.

For more information about the current build, you can click on the build to manage and view details, including the console output, for the build.

**Lab 18 Complete!**

# Lab 19: Using JBoss Tools (Estimated time: 30 minutes)

---

## Server used:

- localhost

## Tools used:

- eclipse

JBoss Tools is an umbrella project for a set of Eclipse plugins that supports JBoss and related technologies; there is support for OpenShift, Hibernate, JBoss AS, Drools, jBPM, JSF, (X)HTML, Seam, Maven, JBoss ESB, JBoss Portal and more...

## Download and install Eclipse - Kepler

In this lab, we are going to use the latest version of JBoss Tools. In order to make use of this version, we will need to use the Kepler version of the popular Eclipse IDE. Head on over to the [eclipse.org](http://eclipse.org) website and download the latest version of Eclipse for Java EE developers that is the correct distribution for your operating system.

```
http://www.eclipse.org/downloads/
```

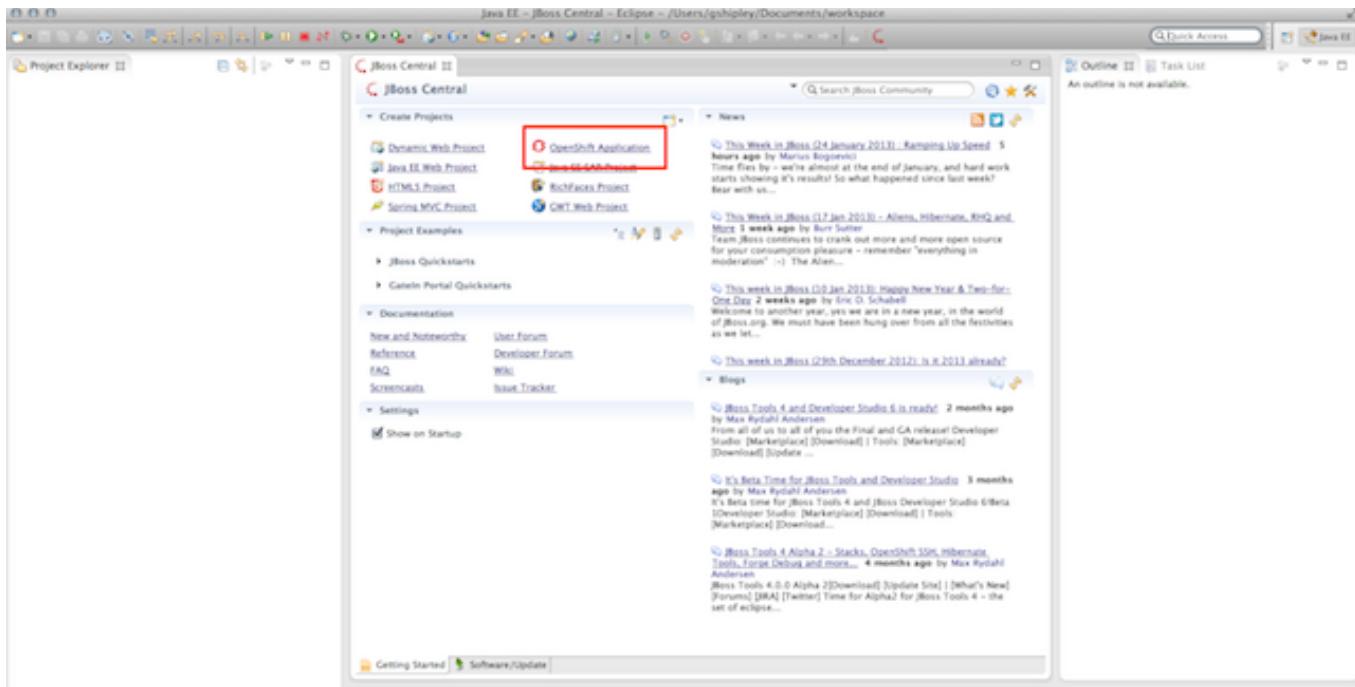
Once you have Eclipse installed, go to the JBoss Tools page located at

```
http://www.jboss.org/tools
```

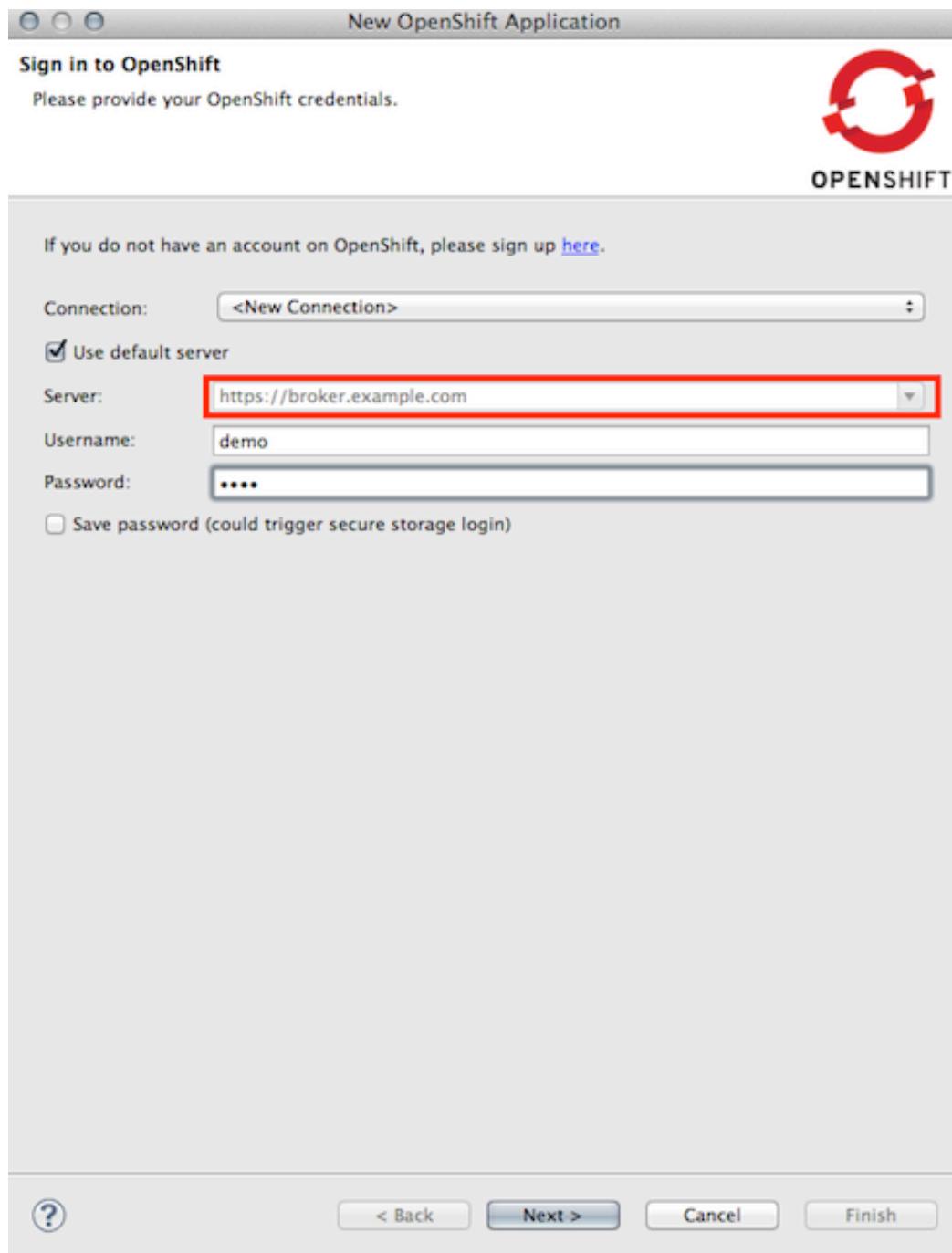
and follow the instructions to install JBoss Tools 4.1 (Kepler). While you are welcome to install all of the JBoss Tools, we will only be using the JBoss OpenShift Tools for this lab.

## Create an OpenShift Enterprise application

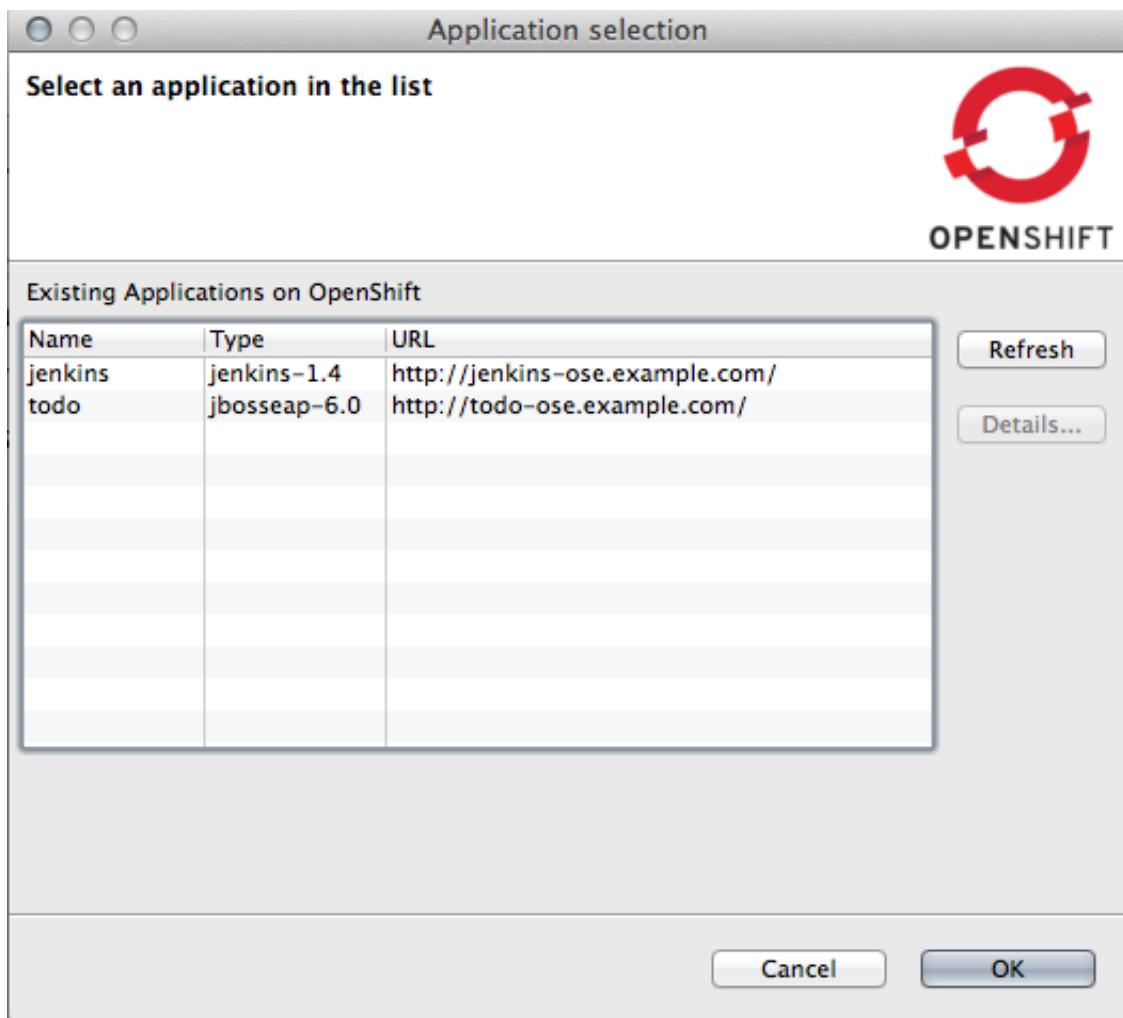
Now that we have Eclipse Juno and JBoss Tools 4.0 installed, we can create an OpenShift Enterprise application without having to leave the comfort of our favorite IDE. Click on the *OpenShift Application* link that is provided on the JBoss Central screen. If you do not have this link, you can also select *file->new->other->OpenShift Application*.



Once you click on the link to create a new OpenShift Enterprise application, you will be presented with a dialog to authenticate to OpenShift Enterprise. Now is also a good time to validate the *Server* setting is correctly set to *broker.hosts.example.com*. If your server does not reflect this, you have not configured your *express.conf* file correctly. If you are unable to configure your *express.conf* file as specified in this lab, inform the instructor so that he/she may help you.



After clicking *next*, the JBoss Tools plugin will authenticate you to the broker host and present another dialog box to you. On this dialog box, you have the option of creating a new application, or to use an existing one. Since we already have a JBoss EAP application deployed, let's select to *Use existing application* and click the *Browse* button. After clicking the *Browse* button, a REST API call be made to the broker host to retrieve the existing applications that you already have deployed.



Highlight the *todo* application and click on the *Details...* button. This will display all of the necessary information about the application, including any cartridges that may be embedded.

Application Details

Details of Application todo

 OPENSHIFT

Property	Value
Name	todo
Public URL	<a href="http://todo-ose.example.com/">http://todo-ose.example.com/</a>
Type	jbosseap-6.0
Created on	2013/01/25 at 13:05:52
UUID	4d1b096e414243e9833dad55d774de73
Git URL	<a href="ssh://4d1b096e414243e9833dad55d774de73@todo-ose.example.com:22/">ssh://4d1b096e414243e9833dad55d774de73@todo-ose.example.com:22/</a>
▼ Cartridges	
postgresql-8.4	<a href="postgresql://127.1.248.129:5432/">postgresql://127.1.248.129:5432/</a>
jenkins-client-1.4	<a href="https://jenkins-ose.example.com/job/todo-build/">https://jenkins-ose.example.com/job/todo-build/</a>

OK

After clicking *Next* to use the existing *todo* application, Eclipse will ask you to create a new project or to use an existing one. Let's create a new one and set the correct location where we want to store the project files.

New OpenShift Application

**Import an existing OpenShift application**

Configure the cloning settings by specifying the clone destination if you create a new project, and the git remote name if you're using an existing project.

  
OPENSIFT

**Cloning settings**

Use default location  
Location: /Users/gshipley/code

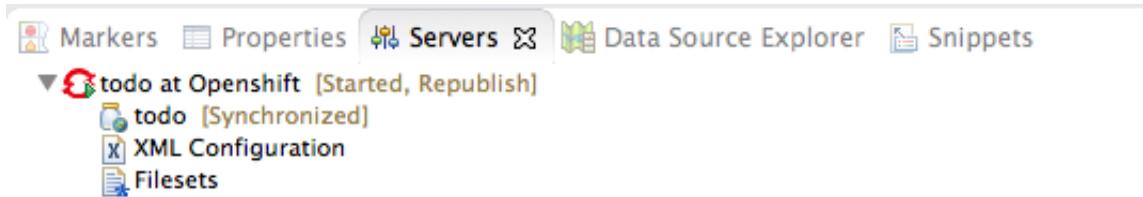
Use default remote name  
Remote name: origin

Make sure that you have SSH keys added to your OpenShift account demo via [SSH Keys wizard](#) and that the private keys are listed in [SSH2 Preferences](#)

Once you click the *Finish* button, the existing application will be cloned to your local project.

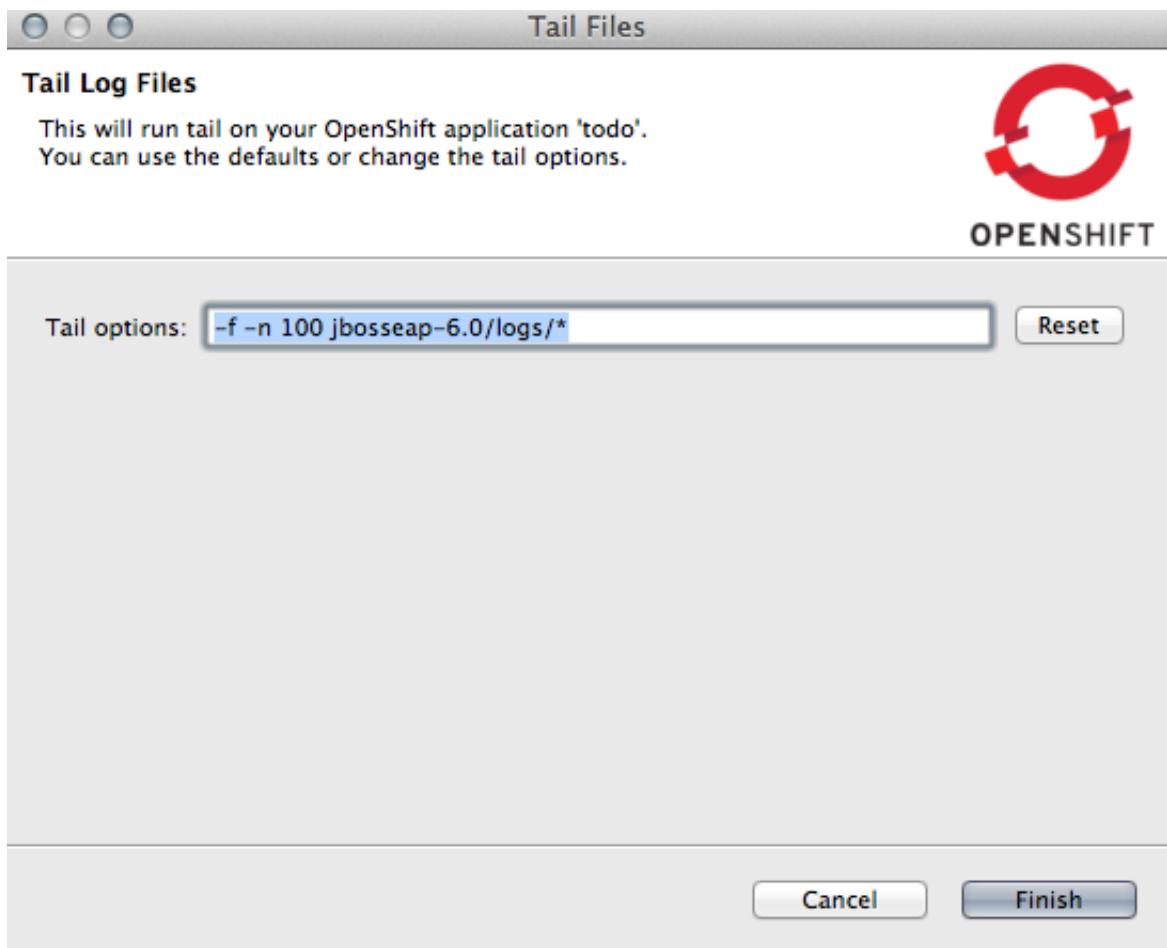
## Managing OpenShift Enterprise application with JBoss Tools

JBoss Tools provide many features to allow a developer to manage their application from directly inside of the Eclipse IDE. This includes features such as viewing log files, publishing the application, and port-forwarding. Click on the servers tab at the bottom on the Eclipse IDE to see your OpenShift Enterprise server.



### Tailing log files

After clicking on the *servers* tab, right click on your OpenShift Enterprise server and then select *OpenShift* and finally select *tail files*.



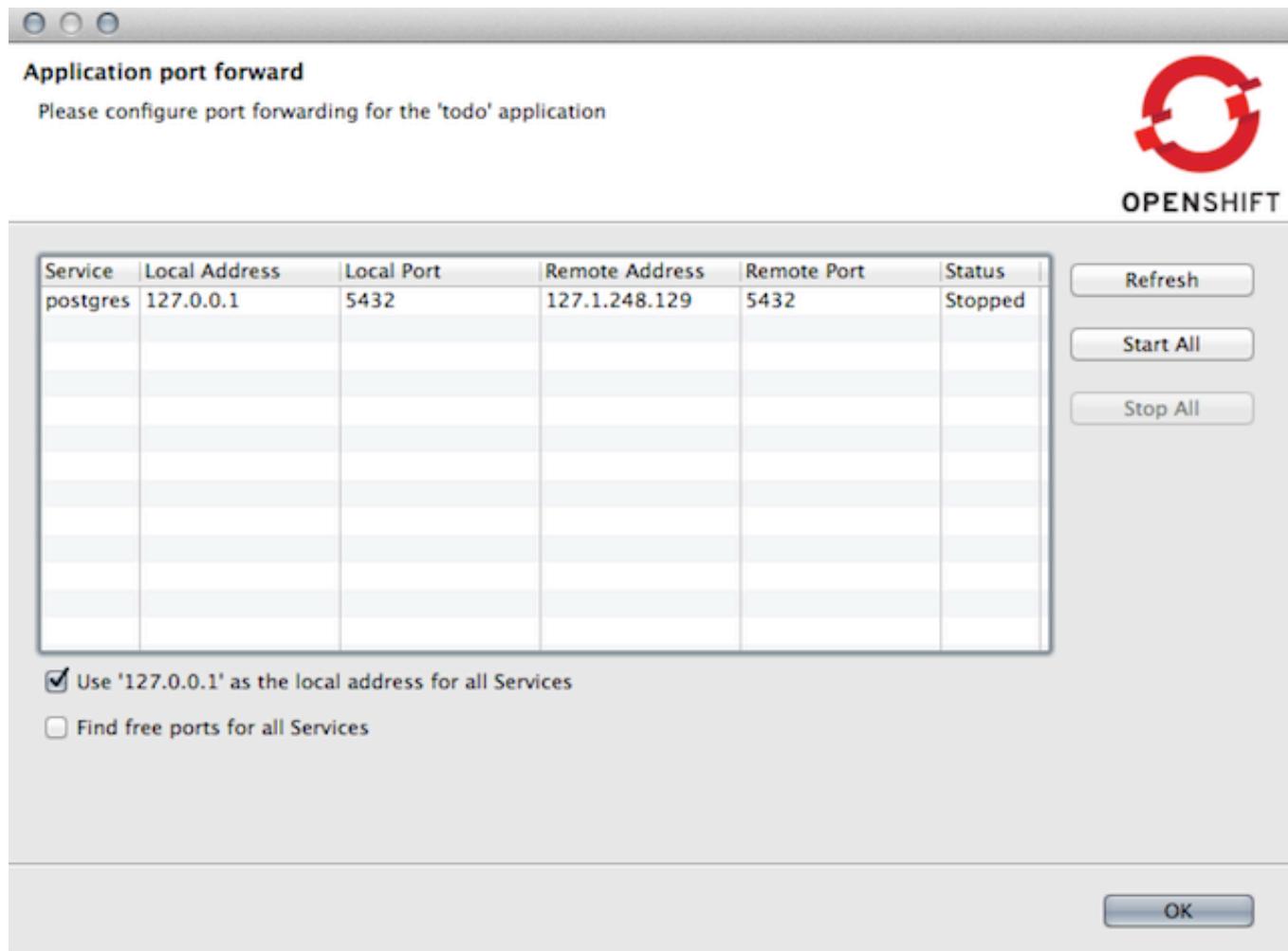
You will now be able to view the log files in the console tab that has been opened for you inside of Eclipse.

### **Viewing environment variables**

After clicking on the *servers* tab, right click on your OpenShift Enterprise server and then select *OpenShift* and finally select *Environment Variables*. Once you select this option, all of the system environment variables, including database connections, will be displayed in the console window of Eclipse.

### **Using port-forwarding**

After clicking on the *servers* tab, right click on your OpenShift Enterprise server and then select *OpenShift* and finally select *Port forwarding*. This will open up a new dialog that displays which services and what IP address will be used for the forwarded services.



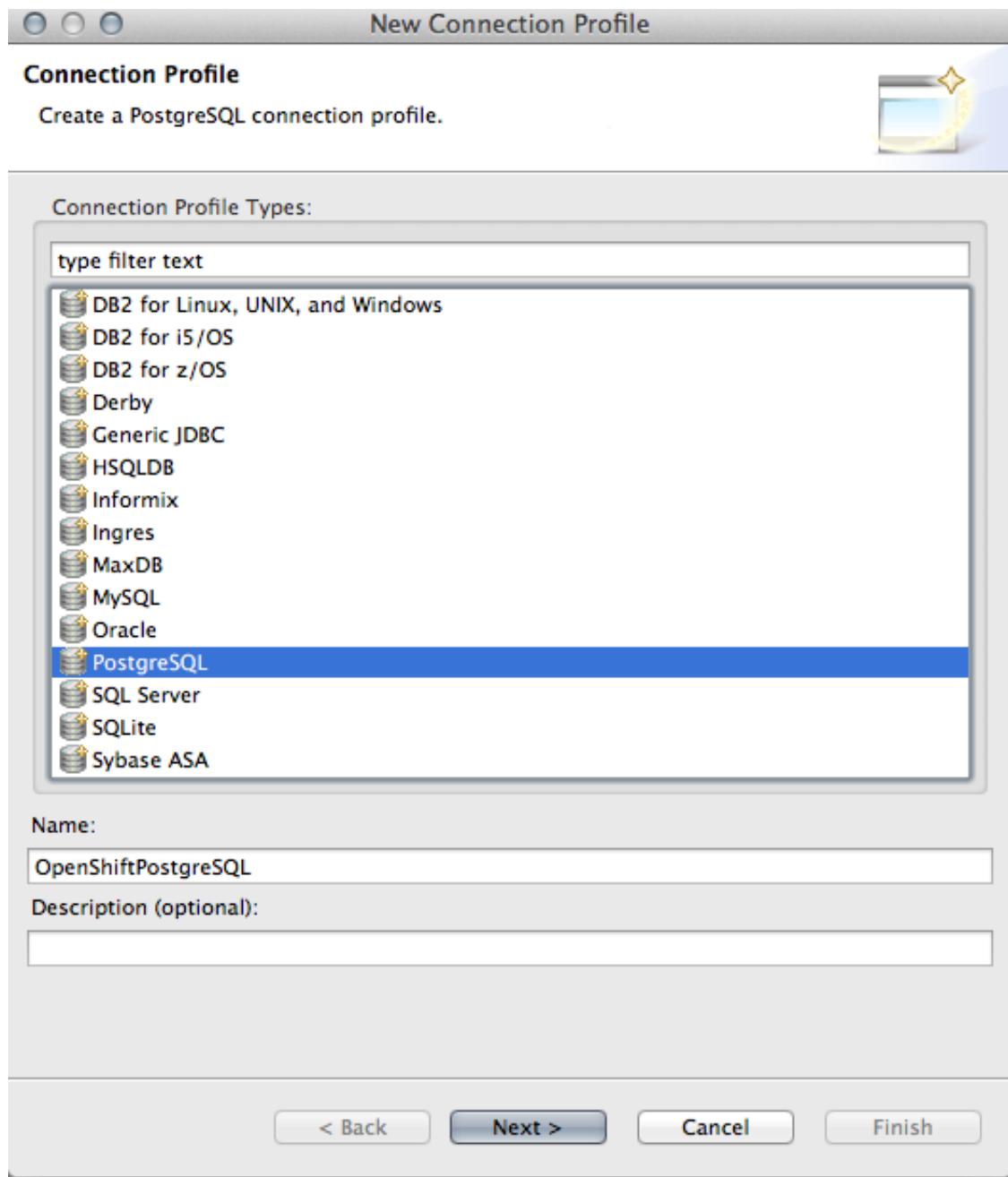
For the next section of this lab, ensure that you click on *Start Forwarding* so that we will be able to connect to PostgreSQL from our local machine.

### **Adding PostgreSQL as an Eclipse data source**

Download the latest PostgreSQL driver from the following location

<http://jdbc.postgresql.org/download.html>

and save it to your local computer. Once you have the file downloaded, click on the *Data Source Explorer* tab, right click on *Database Connection* and select *New*. This will open the following dialog where you will want to select PostgreSQL:



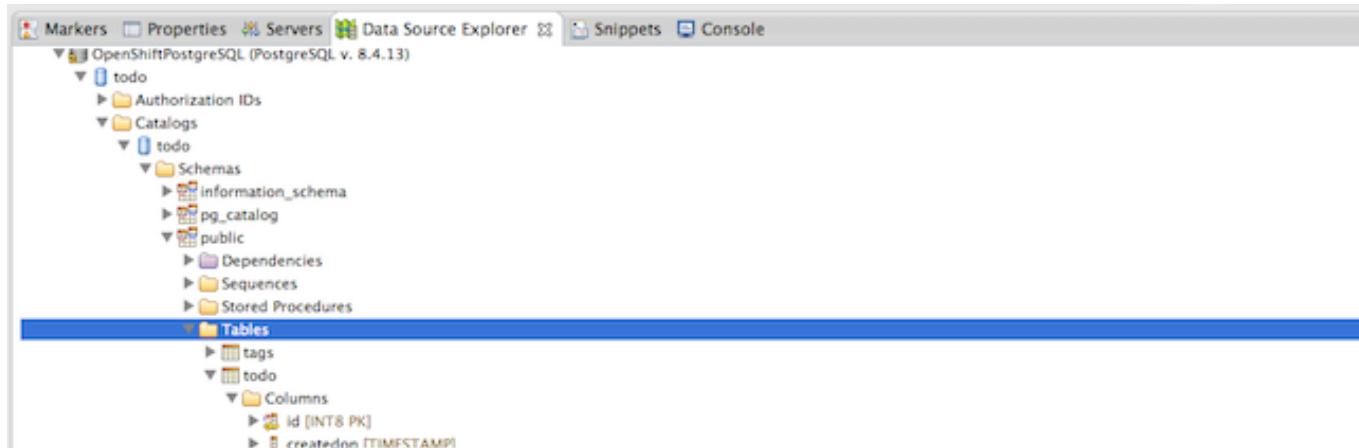
Initially, the *Drivers* pull down box will be empty. In order to add our PostgreSQL driver, click the plug sign next to the drop down, highlight *PostgreSQL JDBC Driver* and then click on *JAR List*. Click on *Add JAR/Zip* and browse to the location of the JDBC4 driver that you downloaded.

Now that you have added the driver, the dialog box will display the available driver and allow you to specify your connection details. Enter the following information:

- Database: todo
- URL: jdbc:postgresql://127.0.0.1:5432/todo
- User name: admin
- Password: The password supplied by OpenShift. If you forgot this, use the *Environment Variables* utility provided by JBoss Tools.

In order to verify that your port-forwarding and database connection is setup correctly, press the *test connection* button. If your connection is failing, make sure that you have the correct authorization credentials and that port-fowarding is started via JBoss Tools.

Once you have correctly added the database connection, you should now see the remote database from the OpenShift Enterprise node host available for use in your Eclipse IDE.



At this point, you should be able to use any of the database tools provided by Eclipse to communicate with and manage your OpenShift Enterprise PostgreSQL database.

## Making a code change and deploying the application

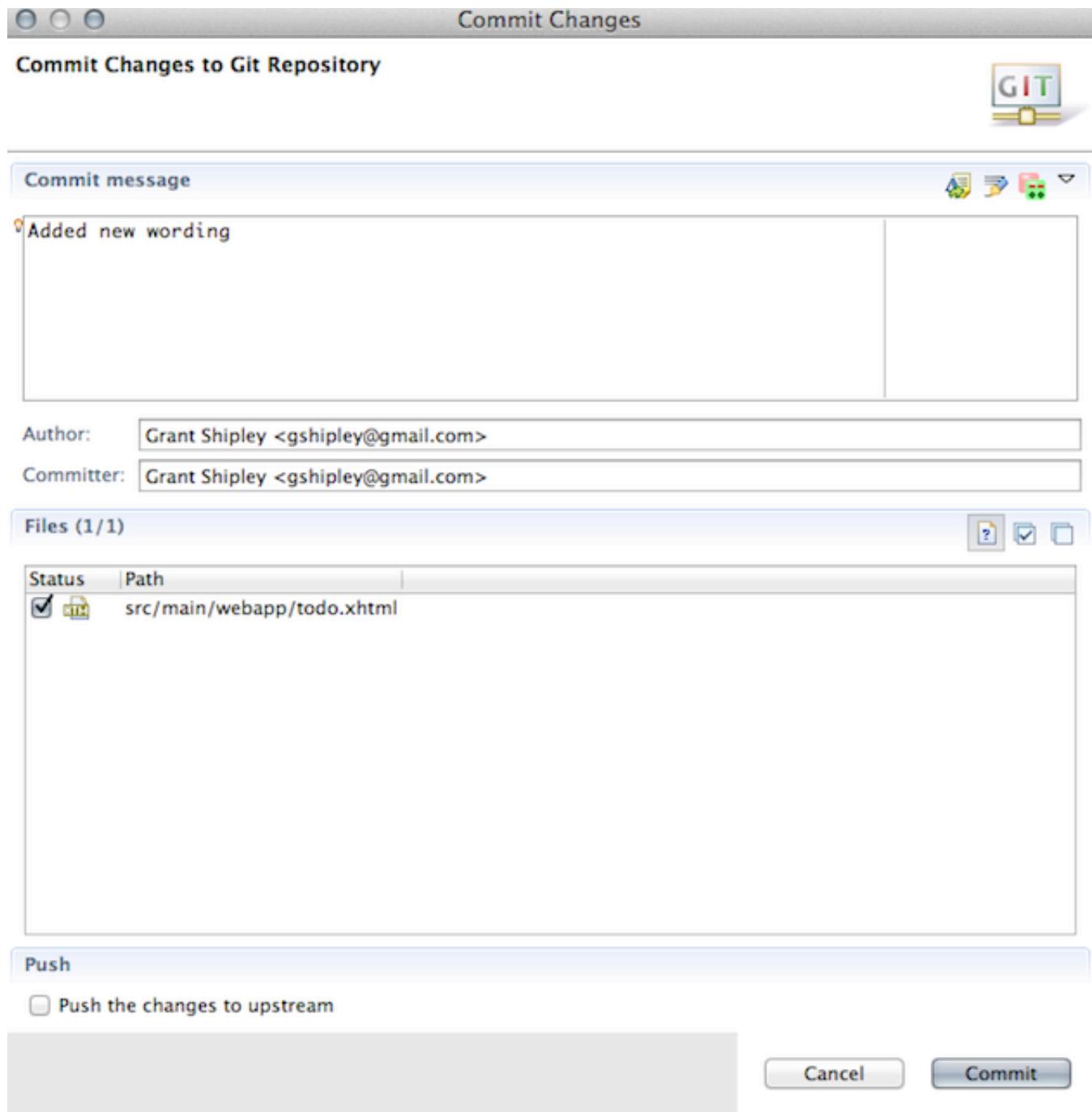
In the project view, expand the source files for the *src/main/webapp* directory and edit the *todo.xhtml* source file. Change the following line

```
<h2>Todo List Creation using Jenkins</h2>
```

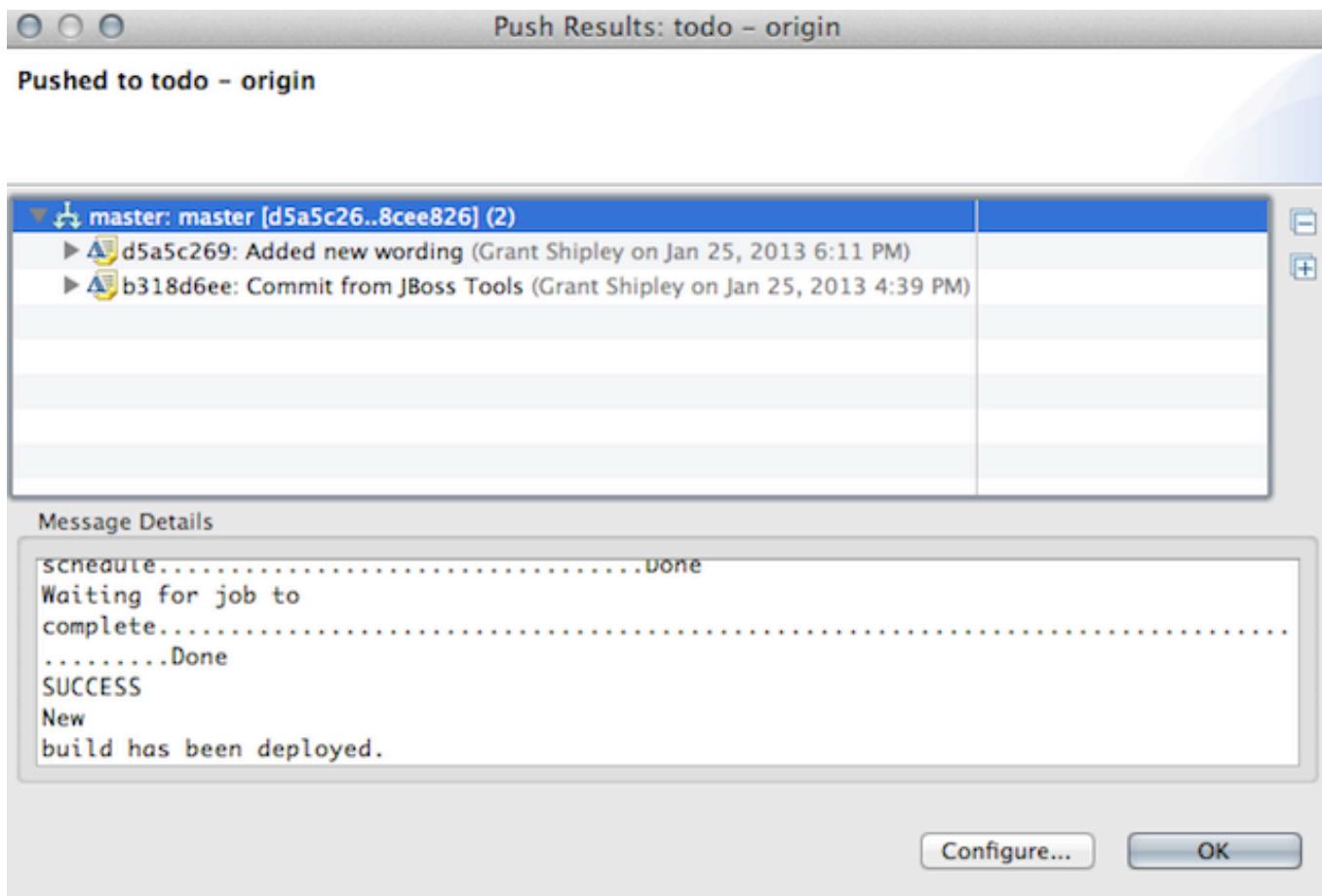
to the include JBoss Tools

```
<h2>Todo List Creation using Jenkins and JBoss Tools</h2>
```

Once you have made the source code change, save the contents of the file and then use the *Team* functionality by right clicking on your project. Commit and push the changes to your OpenShift Enterprise server. This push will follow the same workflow used previously by initiating a build on your Jenkins server.



After you push your changes, open up your Jenkins dashboard and open the *Console Output* screen to see the build progress. Once your build has completed, Eclipse will display a dialog box with a summary of the deployment:



Verify that your changes were deployed correctly by opening up a web browser and going to the following URL:

The screenshot shows a web browser window with the URL <http://todo-ose.example.com/>. The page title is "Start creating your Todos". The sub-header says "Todo Application Running on OpenShift Enterprise". A red box highlights the text "Todo List Creation using Jenkins and JBoss Tools". Below it, a note states "Enforces annotation-based constraints defined on the model class." There are input fields for "Todo" and "Tags", and a "Create New Todo" button. Under the heading "Created Todos", it says "No todos." At the bottom, it says "Powered By OpenShift".

**Lab 19 Complete!**

## #Lab 20: Using quickstarts (Estimated time: 10 minutes)

### Server used:

- localhost

### Tools used:

- rhc
- git

A key tenet when Red Hat was designing OpenShift Enterprise was the ability for developers to be able to run their source code and application as is, without having to use proprietary API(s). To illustrate how easy it is for developers to get their existing application deployed on OpenShift Enterprise, the team has created a github space where they provide numerous quick start projects that make deploying common open source applications to the platform a painless task. Some of the popular open source projects the team provides a quick start for are:

- Drupal
- Review Board
- Wordpress
- Frog CMS
- Sugar CRM
- Redmine
- MediaWiki

## Install a quickstart

Point your browser to the following URL:

```
http://www.github.com/openshift
```

Given the number of available quick starts, you may have to use the search functionality of your browser to locate the quick start that you would like to install. For this lab, choose either the Wordpress or Drupal quick start and follow the instructions provided to install the application.

 **wordpress-example**  
Wordpress quick start repo for OpenShift Express  
Last updated a day ago

 **cakephp-example**  
CakePHP framework quickstart repo  
Last updated a day ago

 **drupal-example**  
DEPRECATED - Use <https://github.com/openshift/drupal-quickstart> instead  
Last updated a day ago

 **joomla-example**  
Joomla quick start repo for OpenShift  
Last updated a day ago

 **phpbb-example**  
Quickstart phpBB on OpenShift | phpBB is a free flat-forum bulletin board software solution that can be used to stay in touch with a group of people or can power your entire website.  
Last updated a day ago

 **rails-example**  
Ruby ★ 44 ⚡ 25

 **mediawiki-example**  
PHP ★ 8 ⚡ 5

**Lab 20 Complete!**

## Lab 21: Creating a quick start (Estimated time: 30 minutes)

### Server used:

- localhost
- node host

### Tools used:

- rhc
- git
- github

A common task that you will be asked to do is make a software developer's development environment easily deployable on OpenShift Enterprise. Development teams desire a quick and repeatable way to spin up an environment with their application code already deployed and integrated with various data stores. In the previous lab, we saw how easy it was to install applications via our quick start process. During this lab, we will focus on the ability for you to create your own quick starts using the popular open source project Piwik as an example.

### Download the Piwik source code

At the time of this writing, you can obtain the code directly from the Piwik website at:  
<http://piwik.org/latest.zip>. Once downloaded, save the file to `~/code/piwikstage`.

After you have downloaded the source code, extract the contents of the zip archive with the following command:

```
$ cd ~  
$ mkdir code  
$ mkdir piwikstage  
$ unzip latest.zip
```

This will create a `piwik` directory under the `~/code/piwikstage` directory.

### Create an OpenShift Enterprise application

We need to create an OpenShift Enterprise application to hold the source code as well as embed the MySQL database:

```
$ cd ~/code  
$ rhc app create piwik php-5.3  
$ rhc cartridge add -a piwik -c mysql-5.1
```

OpenShift Enterprise, as you know, creates a default *index* file for your application. Because we are going to be using the source code from our Piwik applicaiton, we need to remove the existing template.

```
$ rm -rf ~/code/piwik/php/*
```

At this point, we need to copy over the source code that we extracted from the zip archive to our *piwik* OpenShift Enterprise applicaiton:

```
$ cp -av ~/code/piwikstage/piwik/* ~/code/piwik/php
```

Now we need to add and commit our changes to our *piwik* applicaiton:

```
$ cd ~/code/piwik/php  
$ git add .  
$ git commit -am "Initial commit for Piwik"  
$ git push
```

Assuming everything went as expected, you should be able to verify Piwik is running by opening up your web browser and pointing to the following URL:

```
http://piwik-ose.apps.example.com
```

The screenshot shows the first step of the Piwik setup wizard. The title 'Piwik # Open Source Web Analytics' is at the top. On the left, a sidebar lists steps: 1. Welcome!, 2. System Check, 3. Database Setup, 4. Database Check, 5. Creating the Tables, 6. General Setup, 7. Setup a Website, 8. JavaScript Tag, and 9. Congratulations. The main content area has a heading 'Welcome!' and text explaining Piwik is an open source web analytics software. It also states the process is split into 9 steps and will take around 5 minutes. A red 'Next »' button is on the right. At the bottom, there's a 'Installation status' section with a progress bar showing '0 % Done'.

## Creating a github repository

**Note:** This step assumes that you already have a github account. If you don't, head on over to

[www.github.com](http://www.github.com) and sign up (It's free).

Log in to the github website and create a new repository for our quick start. The direct link, after you are logged in, to create a new repository is:

<https://github.com/repositories/new>

Enter a project name and a description for your quick start. I suggest a name that identifies the project as a OpenShift Enterprise quick start. For example, a good name would be *Piwik-openshift-quickstart*.

The screenshot shows the 'Create a New Repository' form on the GitHub website. The 'Project Name' field contains 'Piwik-openshift-quickstart'. The 'Description' field contains 'A quick start guide for installing Piwik on OpenShift'. The 'Homepage URL' field is empty. Under 'Who has access to this repository?', the radio button for 'Anyone' is selected. A note on the right says, 'If you intend to push a copy of a repository that is already hosted on GitHub, please [fork it](#) instead.' At the bottom is a 'Create Repository' button.

GitHub

Create a New Repository

Project Name  
Piwik-openshift-quickstart

Description (optional)  
A quick start guide for installing Piwik on OpenShift

Homepage URL (optional)

Who has access to this repository? (You can change this later)

Anyone ([Learn more about public repos](#))  
 [Upgrade your plan to create more private repositories!](#)

Create Repository

GitHub

About  
Blog  
Features  
Contact & Support  
Training  
Site Status

Tools  
GitHub for Mac  
Issues for iPhone  
Git Code Snippets  
Enterprise Install  
Job Board

Extra  
GitHub Shop  
The Octocat

Documentation  
GitHub Help  
Developer API  
GitHub Flavored Markdown  
GitHub Pages

github SOCIAL CODING

Terms of Service Privacy Security  
© 2011 GitHub Inc. All rights reserved.

Powered by the Dedicated Servers and  
Cloud Computing of Rackspace Hosting

On your newly created project space, grab the HTTP Git URL and add the github repository as a remote to your existing *piwik* OpenShift Enterprise application.

The screenshot shows a GitHub repository page for 'gshipley/piwik Openshift Quickstart'. The repository has 1 commit and 1 pull request. The commit history shows several initial commits from 'Grant Shipley' creating the repository structure. The README section contains a link to 'Piwik on OpenShift'.

name	age	message	history
.openshift/	August 25, 2011	Creating template [mockbuild]	
libs/	August 25, 2011	Creating template [mockbuild]	
misc/	August 25, 2011	Creating template [mockbuild]	
php/	about 2 hours ago	Initial commit for Piwik [Grant Shipley]	
README	about an hour ago	Added README information [Grant Shipley]	
README.ad	about an hour ago	Added README information [Grant Shipley]	
depist.txt	August 25, 2011	Creating template [mockbuild]	

```
$ cd ~/code/piwik
$ git remote add github ${github http URL from github}
```

## Create deployment instructions

In order for developers to be able to use the quick start that you have created, you need to provide instructions on how to install the application. These instructions need to be in the *README* and *README.md* files. By default, github will display the contents of this file, using the markdown version if it exists, on the repository page. For example, a proper *README* file would contain the following contents:

```
Piwik on OpenShift
```

```
=====
Piwik is a downloadable, open source (GPL licensed) real time web analytics soft
```

```
Piwik aims to be an open source alternative to Google Analytics, and is already
```

```
More information can be found on the official Piwik website at http://piwik.org
```

```
Running on OpenShift
```

```
-----
Create an account at http://openshift.redhat.com/
```

```
Create a PHP application
```

```
rhc app create -a piwik -t php-5.3 -l $USERNAME
```

```
Add mysql support to your application
```

```
rhc cartridge add -a piwik -c mysql -l $USERNAME
```

```
Make a note of the username, password, and host name as you will need to use
```

```
Add this upstream Piwik quickstart repo
```

```
cd piwik/php
rm -rf *
git remote add upstream -m master git://github.com/gshipley/piwik-openshift
git pull -s recursive -X theirs upstream master
```

```
Then push the repo upstream to OpenShift
```

```
git push
```

```
That's it, you can now checkout your application at:
```

```
http://piwik-\$yourlogin.rhcloud.com
```

```
Create the README and README.md in the ~/code/piwik directory and add the contents provided above. Once you have created these files, add and commit them to your repository:
```

```
$ cd ~/code/piwik
$ git add .
$ git commit -am "Add installation instructions"
```

```
Now we need to push these changes to the github repository we created:
```

```
$ git push -u github master
```

## Verify your quick start works

Delete the *piwik* OpenShift Enterprise application and follow the instruction you created for your Piwik quick start to verify that everything works as expected.

**Note:** If your application requires an existing populated database, the way to accomplish this is by using the `.openshift/action_hooks/build` script located in your application directory. Once you have your database created locally, do a `mysqldump` on the table and store the `.sql` file in the `action_hooks` directory. You can then modify an existing build file to import the schema on application deployment. For an example, take a look at the `action_hooks` directory of the Wordpress quick start.

**Lab 20 Complete!**

# Appendix A - Troubleshooting

---

## Diagnostics script

Installing and configuring an OpenShift Enterprise PaaS can often fail due to simple mistakes in configuration files. Fortunately, the team provides an unsupported troubleshooting script that can diagnose most problems with an installation. This script is located on the OpenShift Enterprise hosts is called *oo-diagnostics*.

To run the command and check for errors, issue the following command:

```
# oo-diagnostics -v
```

**Note:** Sometimes the script will fail at the first error and not continue processing. In order to run a full check on your node, add the *--abortok* switch

```
# oo-diagnostics -v --abortok
```

Under the covers, this script performs a lot of checks on your host as well as executing the existing *oo-accept-broker* and *oo-accept-node* commands on the respective host.

## Unknown locale error when running *lokkit*

If you get unknown locale error when running *lokkit*, run the following command:

```
# export LC_CTYPE="en_US.UTF-8"
```

## Recovering failed nodes

A node host that fails catastrophically can be recovered if the gear directory `/var/lib/openshift` has been stored in a fault-tolerant way and can be recovered. In practice this scenario occurs rarely, especially when node hosts are virtual machines in a fault tolerant infrastructure rather than physical machines.

**Note: Do not start the MCollective service until you have completed the following steps.**

Install a node host with the same hostname and IP address as the one that failed. The hostname DNS A record can be adjusted if the IP address must be different, but note that the application CNAME and database records all point to the hostname, and cannot be easily changed.

Duplicate the old node host's configuration on the new node host, ensuring in particular that the gear profile is the same.

Mount `/var/lib/openshift` from the original, failed node host. SELinux contexts must have been stored on the `/var/lib/openshift` volume and should be maintained.

Recreate `/etc/passwd` entries for all the gears using the following steps:

- Get the list of UUIDs from the directories in `/var/lib/openshift`.
- Get the Unix UID and GID values from the group value of `/var/lib/openshift/UUID`.
- Create the corresponding entries in `/etc/passwd`, using another node's `/etc/passwd` file for reference.

Reboot the new node host to activate all changes, start the gears, and allow MCollective and other services to run.

## Removing applications from a node

While trying to add a node to a district, a common error is that the node already has user applications on it. In order to be able to add this node to a district, you will either need to move these applications to another node or delete the applications. In this training class, it is suggested that you simply delete the application as we are working in a single node host configuration. In order to remove a users application, issue the following commands:

```
# oo-admin-ctl-app -l username -a appname -c stop  
# oo-admin-ctl-app -l username -a appname -c destroy
```

The above commands will stop the users application and them remove the application from the node. If you want to preserve the application data, you should backup the application first using the snapshot tool that is part of the RHC command line tools.

