OPENSHIFT

OpenShift Enterprise 2.0

Installation, Configuration, Administration and Usage

redhat.

Contents

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.

This manual also assumes that you have been granted access to two Red Hat Enterprise Linux servers with which to perform the exercises in it. 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 creating and deploying applications using the OpenShift Enterprise management console, using the OpenShift Enterprise administration console, using 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 with 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

One 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 running on x86_64 architecture. We require Red Hat Enterprise Linux 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 running on x86_64.

For this training class, we will be using OpenStack as our Infrastructure as a Service layer.

1.5 Using the *openshift.sh* installation script

In this training class, we are going to take advantage of the *openshift.sh* installation script, 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 you 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
```

Lab 1 Complete!

Lab 2: Using the OpenShift Enterprise Subscription

Servers used:

- broker host
- node host

Tools used:

- SSH
- rhn-channel
- yum
- ssh-keygen
- ssh-copy-id
- sh

Verifying Red Hat Network Channels

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 grant your system access to appropriate software channels. The machines provided to you in this lab have already been registered with the production Red Hat Network, and the channels you will need have already been added.

Please verify now that your machines have the required channels enabled:

Note: Execute the following on both of the hosts that have been provided to you

```
# rhn-channel --list
```

You should see the following list of channels:

```
jb-ews-2-x86_64-server-6-rpm
jbappplatform-6-x86_64-server-6-rpm
rhel-x86_64-server-6
rhel-x86_64-server-6-ose-2.0-infrastructure
rhel-x86_64-server-6-ose-2.0-jbosseap
rhel-x86_64-server-6-ose-2.0-node
rhel-x86_64-server-6-ose-2.0-rhc
rhel-x86_64-server-6-rhscl-1
```

You can also see the list of available channels using Yum:

```
# yum repolist
```

You should see the same channels listed. If you do not see the expected output, ask the instructor for assistance.

Note that if you were registered using Red Hat Subscription Manager, the channels would be named using a different format.

Lab 2 Complete!

Lab 3: Installing OpenShift Enterprise 2.0

Server used:

- broker host

Tools used:

- openshift.sh

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

Overview of *openshift.sh*

The OpenShift team has developed a installation script for the platform that simplifies the installation and configuration of the PaaS. The *openshift.sh* script is a flexible tool 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 *openshift.sh* 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.

A copy of *openshift.sh* has already been loaded on each system provided to you. This script is also available in the enterprise–2.0 branch of the openshift-extras Github repository. In that repository, you can find a kickstart version of the script as well as other versions. The script we will be using is the generic openshift.sh script.

Installing and Configuring the OpenShift Broker host using *openshift.sh*

The *openshift.sh* script takes arguments in the form of environment variables or command-line arguments. All of the recognized arguments are documented extensively in comments at the top of the script. For our purposes, we will want to specify the following arguments when we use this script to install the broker host:

- *install_components=broker,named,activemq,datastore*
- *domain=apps.example.com*
- *hosts_domain=hosts.example.com*
- *broker_hostname=broker.hosts.example.com*
- *named_ip_addr={host1 IP address}*
- *named_entries=named_entries=broker:{host1 IP address},activemq:{host1 IP*

address},datastore:{host1 IP address},node:{host2 IP address}

- *install_method=rhsm*

Let's go over each of these options in more detail.

install_components

The *install_components* setting specifies which components the script will install and configure. In this training session, we will install the OpenShift broker and supporting services on which OpenShift depends on one host, and we will install the OpenShift node component on a second host.

In more complex installations, you will want to install each component on a separate host (in fact, you will want to install most components on several hosts each for redundancy). For testing or POCs, you may want to install all components (including both the OpenShift broker and node) on a single host, which is the default if you do not specify a setting for install_components.

domain, *host_domain*, *broker_hostname*, *named_ip_addr*, and *named_entries*

The *domain* setting specifies the domain name that you would like to use for your applications that will be hosted on the OpenShift Enterprise Platform. The default is example.com, but it makes more sense from an architecture view point to separate these out to their own domain.

The *hosts_domain* setting specifies the domain name that you would like the OpenShift infrastructure hosts to use, which includes the OpenShift broker and node hosts. This domain also includes hosts running supporting services such as ActiveMQ and MongoDB, although in our case we are running these services on the OpenShift broker host as well.

While it is not required to do so, it is good practice to put your infrastructure hosts (broker and nodes) under a separate domain from applications.

The *broker_hostname* setting specifies the fully-qualified hostname that the installation script will configure for the OpenShift broker host. In a more complex configuration with redundant brokers, you will want to use this setting to specify a unique hostname for each host that you install (e.g.,* broker_hostname=broker01.hosts.example.com* on the first host, *broker_hostname=broker02.hosts.example.com* on the second host, and so on). For this training session, we will only be installing one broker host.

For the *named_ip_addr* setting, use the 209.x.x.x address of *host1* that was provided to you by your instructor. We will be installing our nameserver alongside the OpenShift broker on this host, so we want to make sure that we configure the host to use itself as its own nameserver. The *named_entries* is used when the host installs the nameserver to add DNS records for the various hosts and services we will be installing. We tell the installation script to create records with the public-facing IP addresses for these hosts (as opposed to the private, internal IP addresses).

install_method

The installation script supports several installation methods. For this training session, we are using the Red Hat Network Classic, which we specify using the *install_method* setting. The host is already registered with Red Hat Network, and the required channels are already in place, so we do not need the installation script to perform these tasks for us; however, we specify this option so that the installation script will verify that the required channels are enabled and configure appropriate priorities and excludes so that Yum downloads the correct packages.

Executing *openshift.sh*

Let's go ahead and execute the command to run installation script.

For own use, set the *host1* and *host2* environment variables:

Note: Execute the following command on the broker host and ensure that you replace {host1 IP address} with the broker IP address and {host2 IP address} with the correct node IP address provided to you by the instructor.

host1={host1 IP address}; host2={host2 IP address}

For example, if the instructor gave me the following information: * Broker IP Address = 209.132.179.41 * Node IP Address = 209.132.179.75

I would enter in the following command:

```
# host1=209.132.179.42; host2=209.132.179.75
```

Now let's execute *openshift.sh*.

Note: Perform the following command on the broker host.

```
# sh openshift.sh install_components=broker,named,activemq,datastore domain=apps
```

The installation script will take a while depending on the speed of the connection at your location. While the installation script runs on the OpenShift broker host, open a new terminal window or tab and continue on to the next section to begin the installation and configuration of your second host which will be the node host.

Installing and Configuring the OpenShift Node host

To install and configure the OpenShift node host, we will want to specify the following arguments to *openshift.sh*:

- *install_components=node*
- *cartridges=all,-jboss,-jenkins,-postgres,-diy*
- *domain=apps.example.com*
- *named_ip_addr={host1 IP address}*
- *node_hostname=node.hosts.example.com*
- *node_ip_addr={host2 IP address}*
- *install_method=rhsm*

Following is an explanation for each of these arguments.

install_components **and** ***cartridges***

We are configuring this host as an OpenShift node host. As the instructor will explain shortly in the lecture, a node has several "cartridges" installed, which provide language runtimes, Web frameworks, databases, and other features for application developers to use. For now, we will install all available cartridges except for JBoss Web frameworks, the Jenkins continuous integration environment, the PostgreSQL DBMS, and the DIY cartridge.

domain, ***hosts_domain***, ***named_ip_addr***, ***node_hostname***, **and** ***node_ip_addr***

We described the *domain* and *hosts_domain* settings earlier in this lab while installing and configuring the OpenShift broker host.

For the *named_ip_addr* setting, use the 209.x.x.x address for *host1* that was provided to you by your instructor. We use the *named_ip_addr* setting to configure name resolution on the OpenShift node host to use our own nameserver.

The *node_hostname* setting specifies the fully-qualified hostname that the installation script will configure for the OpenShift node host.

For the *named_ip_addr* setting, use the 209.x.x.x address for *host2* that was provided to you by your instructor. We use the *node_ip_addr* setting to tell the installation script to configure this OpenShift node host to use its public-facing IP address when it configures routing rules for user applications.

install_method

Just as when we installed the OpenShift broker host, we must specify the *install_method* setting in order for *openshift.sh* to configure Yum channels correctly.

Executing ***openshift.sh***

Before we execute the command to run installation script, let's set the *host1* and *host2* environment

variables as we did on the broker host:

Note: Perform the following command on the node host.

host1={host1 IP address}; host2={host2 IP address}

For example, if the instructor gave me the following information: * Broker IP Address = 209.132.179.41 * Node IP Address = 209.132.179.75

Note: Perform the following command on the node host.

I would enter in the following command:

```
# host1=209.132.179.41; host2=209.132.179.75
```

Now launch the installation script:

```
# sh openshift.sh install_components=node cartridges=all,-jboss,-jenkins,-postgr
```

The installation script will 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.

Lab 3 Complete!

Lab 4: Verifying the installation

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 session 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 is 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 the named service, provided by *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.examp
                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 listed in the output of the *cat* command. Once you have verified this, take a look at the database for apps.example.com.db to verify that is 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 host for DNS resolution. To verify this, view the */etc/resolv.conf* file on 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 node.hosts.example.com from the broker and receive a response as shown below:

```
# ping node.hosts.example.com

PING node.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. MCollective is a higher-level framework for remote job execution that communicates over ActiveMQ. OpenShift Enterprise makes use of these technologies to handle communications between the broker host and the node host in our deployment.

The first thing we want to ensure is that the ActiveMQ 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 that MCollective is able to communicate between the broker and the node hosts. To very 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 (the time values will vary):

```
node.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

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 lab 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
```

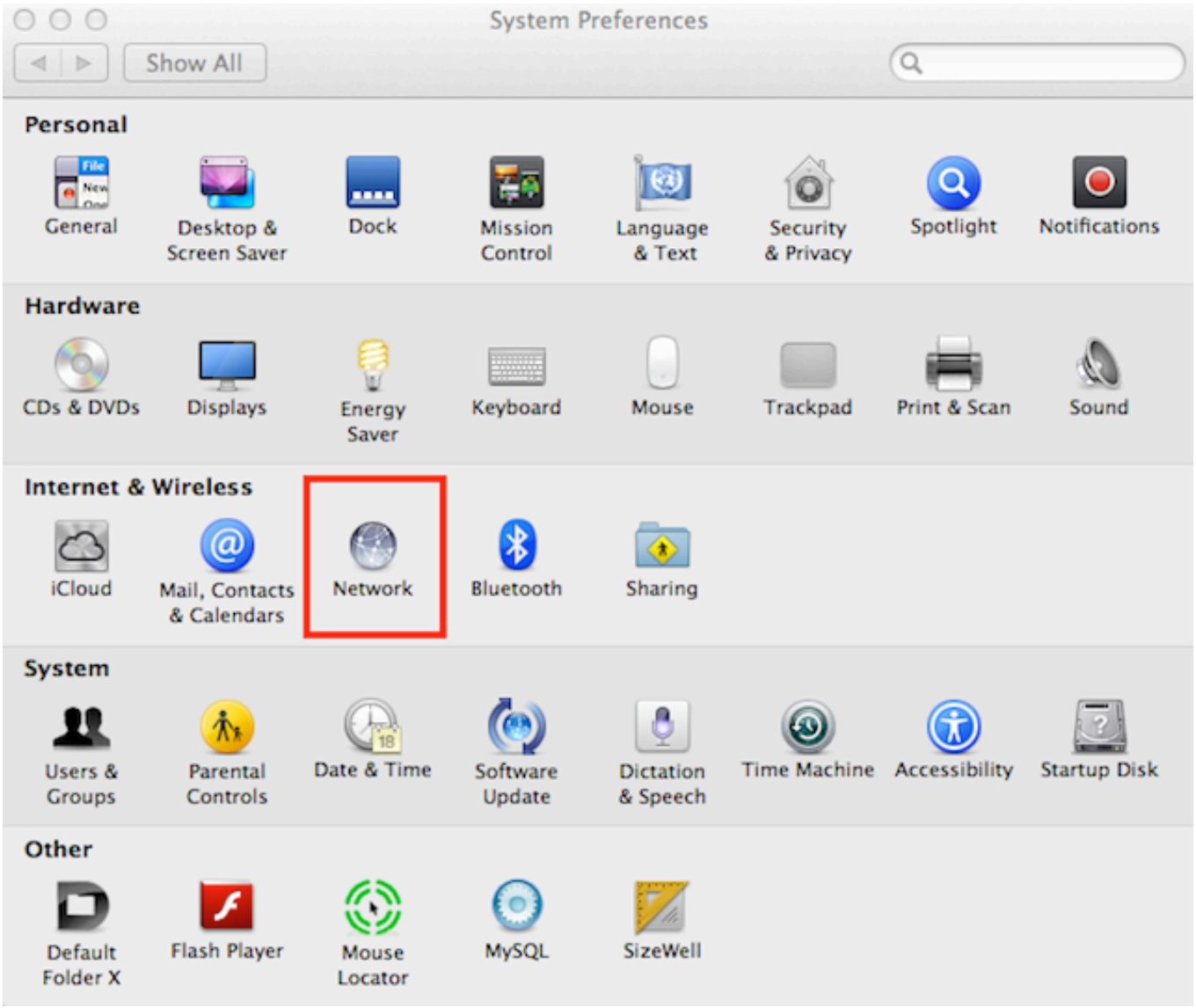
Once you have added the above nameserver, you should be able to communicate with your OpenShift Enterprise PaaS by using the server hostname. To test this out, ping the broker and node hosts from your local machine:

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

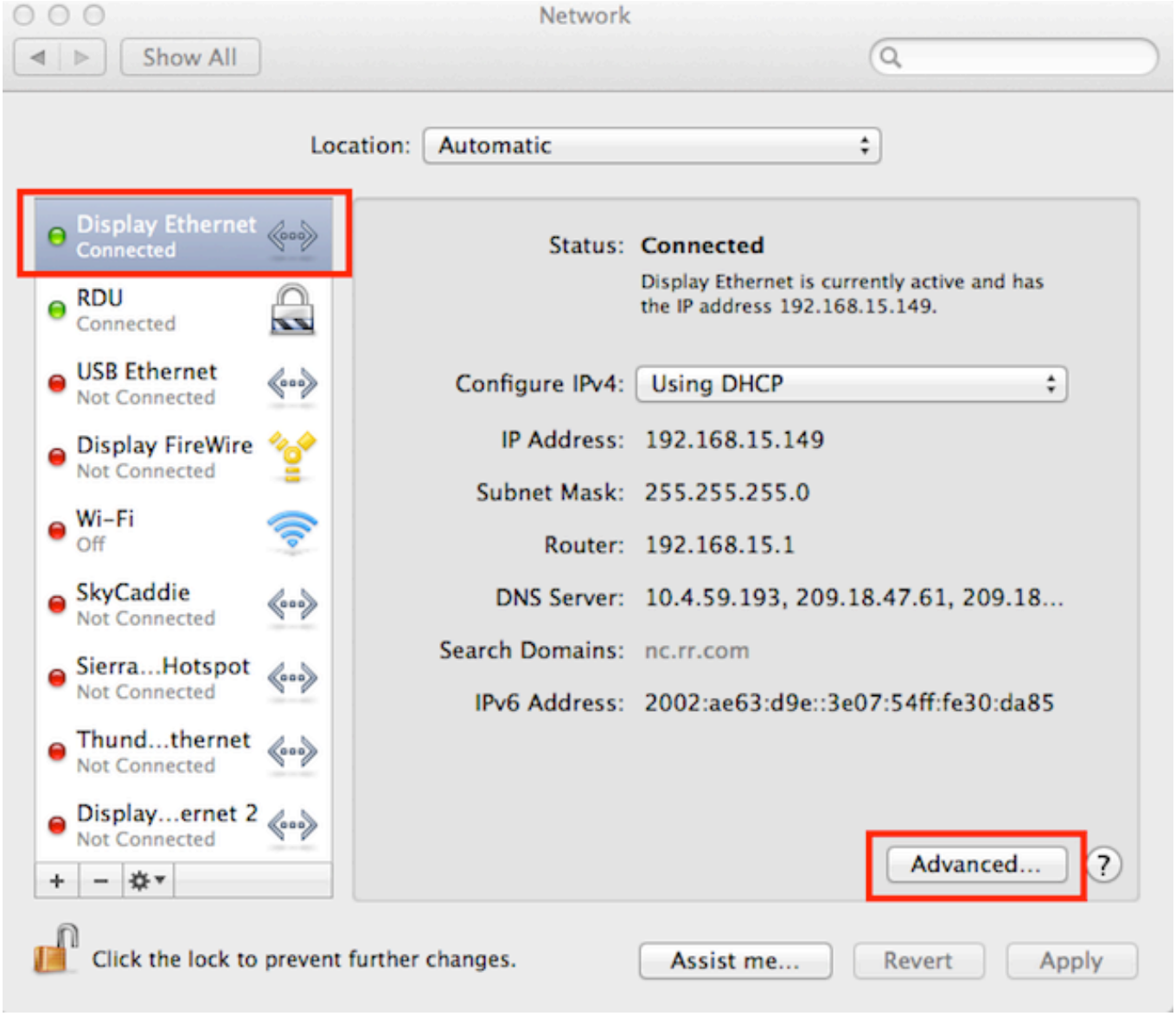
Configuring example.com resolution for OS X version 10.6 and below

If you are using OS X, 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.

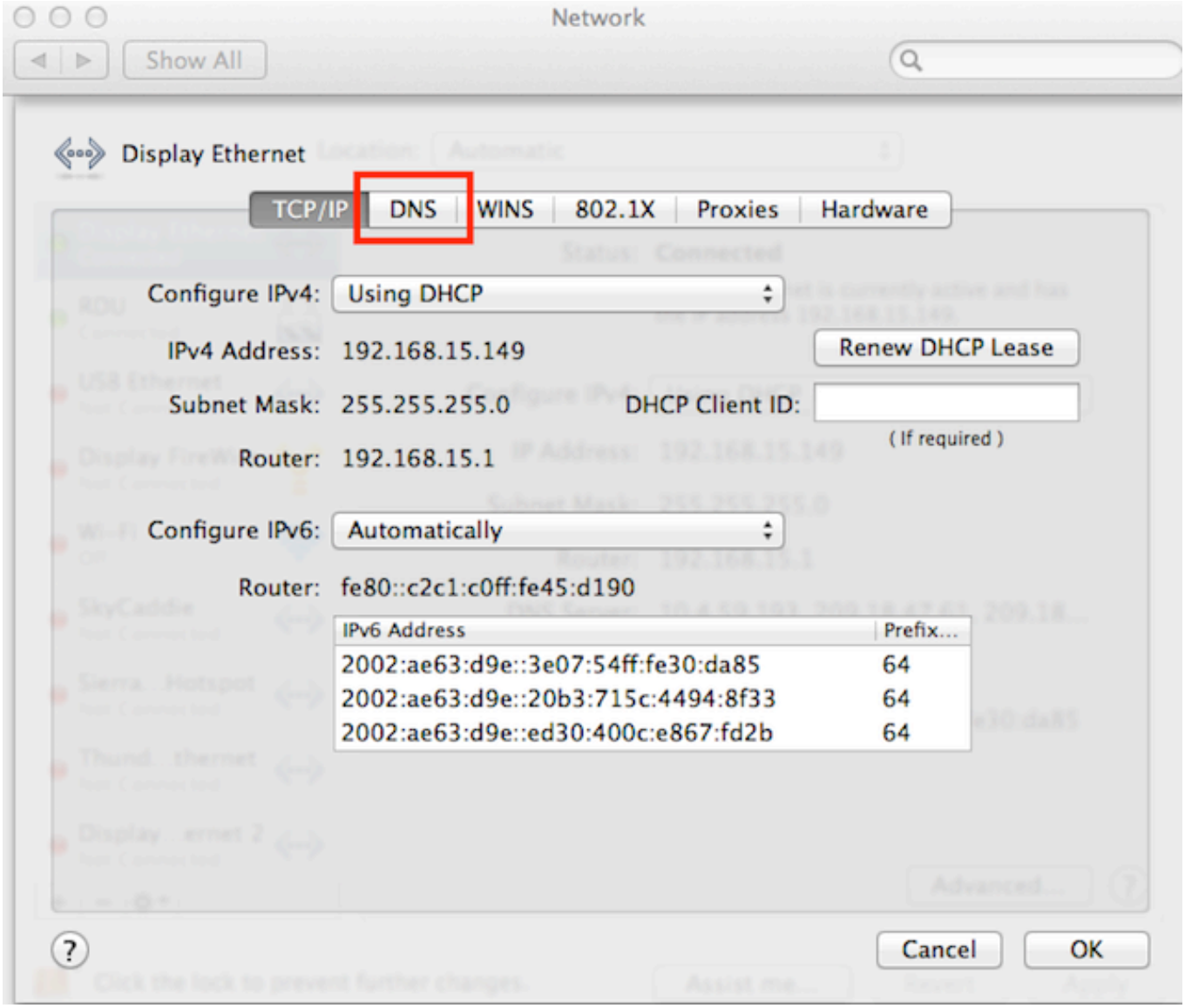
Open up the *System Preferences* tool and select the *Network* utility:



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:

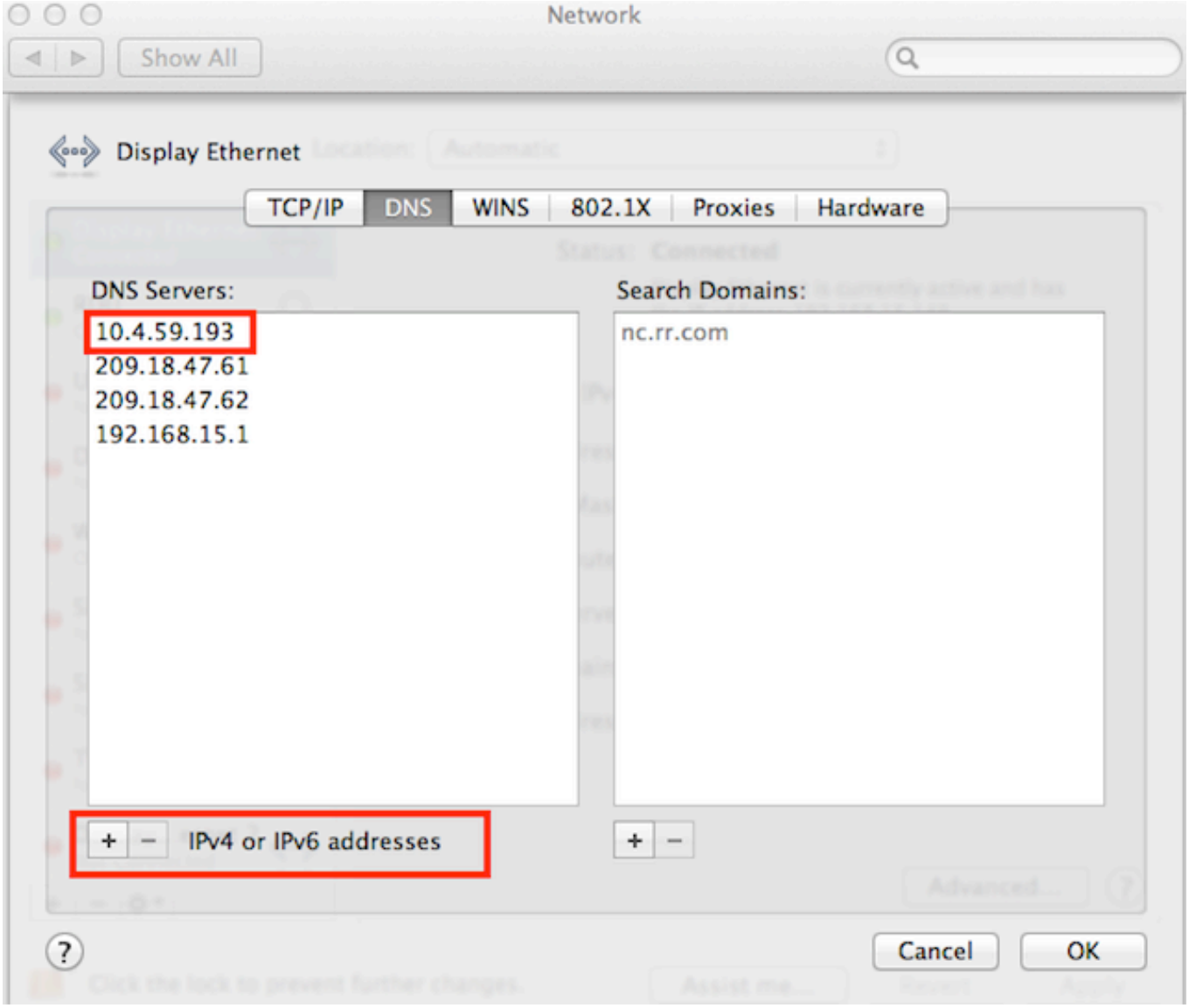


Select the DNS tab at the top of the window:



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 209.x.x.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 *openshift.sh* installation script does not enable forwarding on the BIND instance, which will result in domain names other than **.example.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
```

Be sure 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

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 openshift-origin-cartridge-jbo
```

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 management console, the application type is not available. This is because the broker host creates a cache of available cartridges to increase performance. After adding a new cartridge, you need to clear this cache in order for the new cartridge 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 management 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 management 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.
```

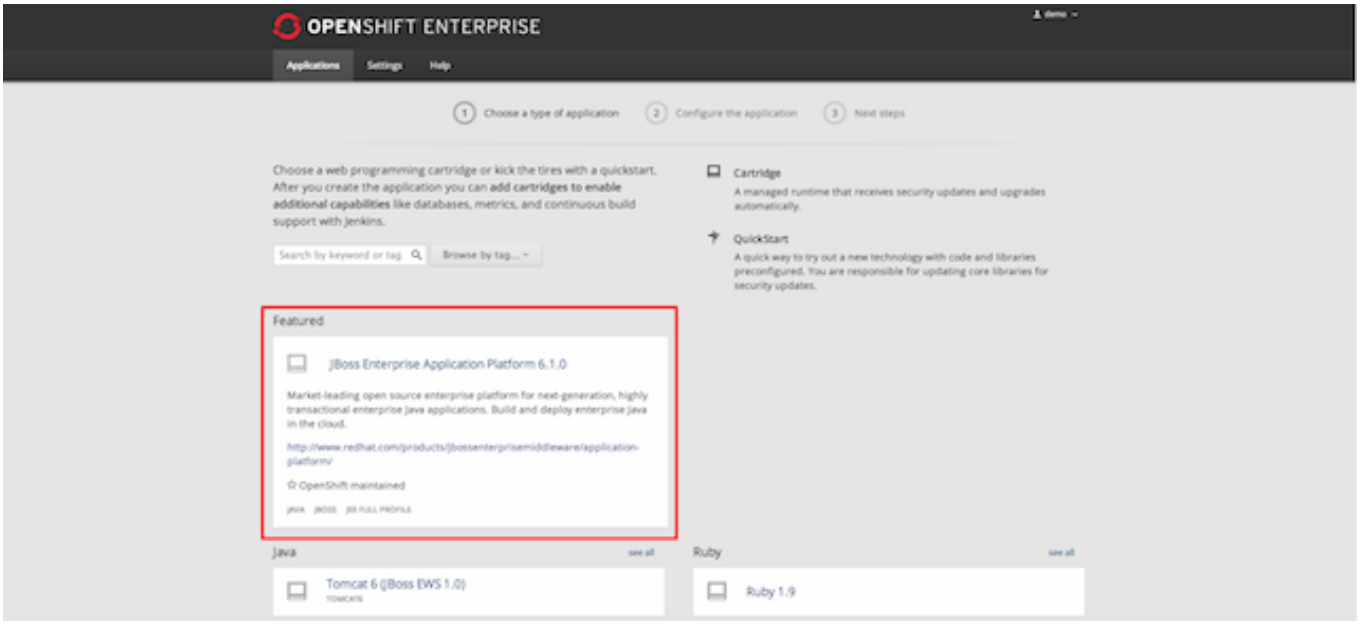
It may take several minutes before you see the new cartridges available on the management console as it takes a few minutes for the cache to completely clear.

Testing new cartridges

Given the steps in Lab 5 of this training, you should be able to access the management console from a web browser using your local machine. Open up your preferred browser and enter the following URL:

```
http://broker.hosts.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:

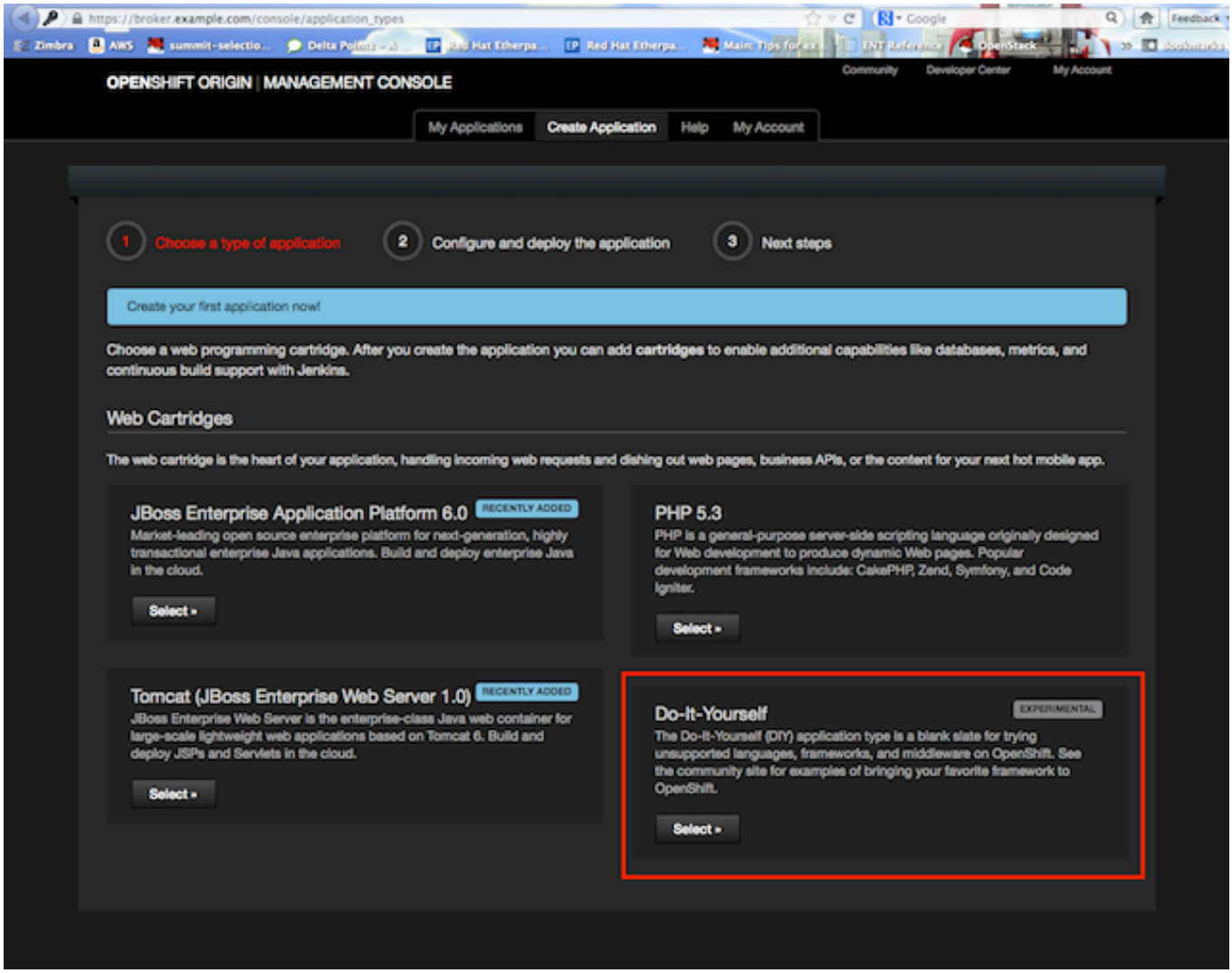


If you do not see the new cartridges available on the management console, check that the new cartridges are available by viewing the contents of the */usr/libexec/openshift/cartridges* directory:

```
# 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 management console:



Lab 6 Complete!

Lab 7: Managing resources

Server used:

- node host
- broker host

Tools used:

- text editor
- oo-admin-ctl-user

Setting default gear quotas and sizes

A user's default gear size and quota are 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_GEARS* settings 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 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 */etc/openshift/broker.conf* file on each broker host to specify the gear name and then modify the */etc/openshift/resource_limits.conf* file on each node that you would like to that you would

like to host that gear size to match the name and 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

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 in 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. Enabling this option prevents users from creating a 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.hosts.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 using the *openshift.sh* installation script. To view this document inside of the database, execute the following command on the broker host:

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

Note: The default mongodb username and password can be found 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. To do so, 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.hosts.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.hosts.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 have 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.districts.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 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

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 runing:

```
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. 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 rubygems 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*

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 using the 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 ca
'broker.hosts.example.com'.

You may bypass this check, but any data you send to the server could be intercep
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 user's 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 will have been created in the user's 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 read by 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

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 alpha-numeric 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
```

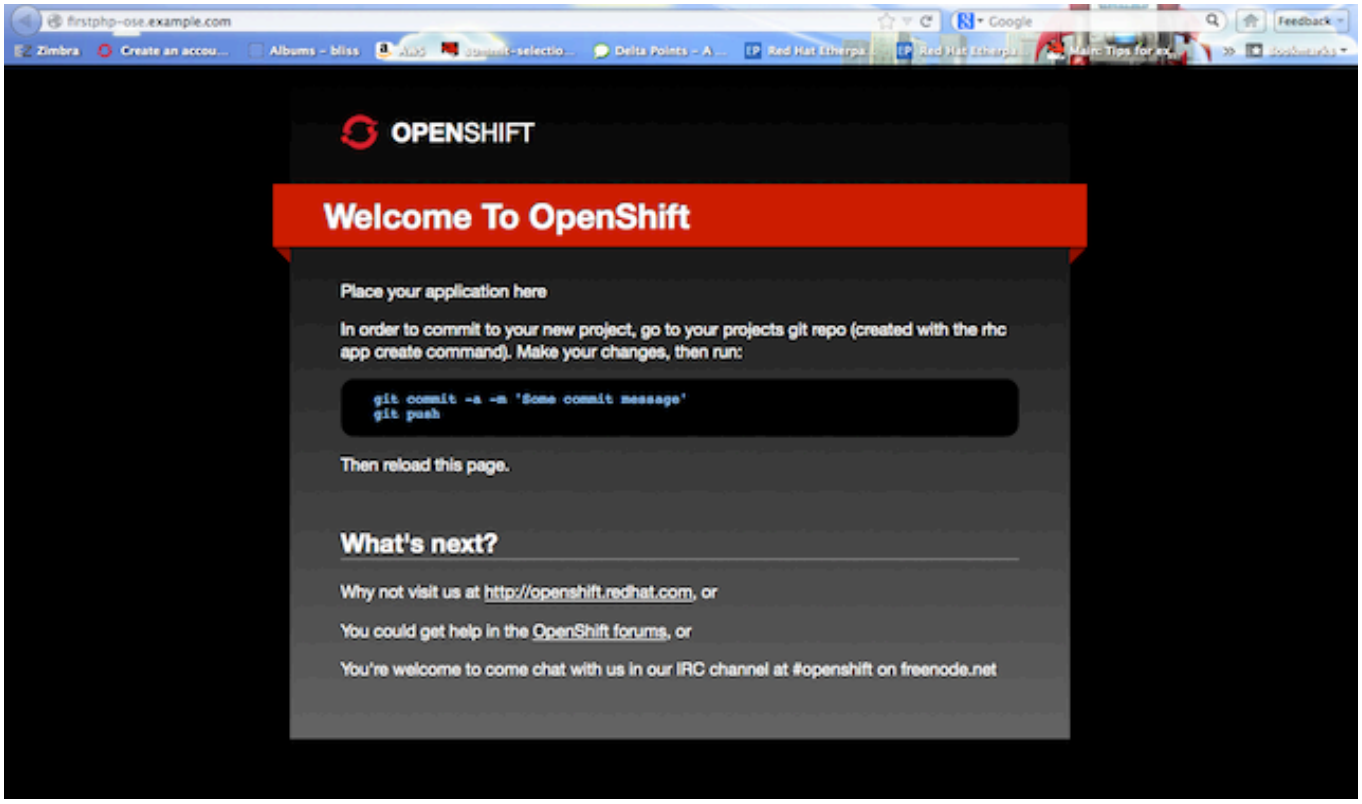
After entering that command, you should see the following output:

```
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 broadcast 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.
- 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 onto 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 .gitconfi
-rw-r--r--.  1 root root                             1352 Jan 21 13:47 .pearrc
drwxr-xr-x. 10 root root                             4096 Jan 21 13:47 php
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.

Let's close our SSH session and return to your local machine:

```
# exit
```

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 they 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. To enable this functionality, you need to add the cron cartridge to your application. Once you have done so, any scripts or jobs added to the minutely, hourly, daily, weekly or monthly directories will be run 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 deplist.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
```

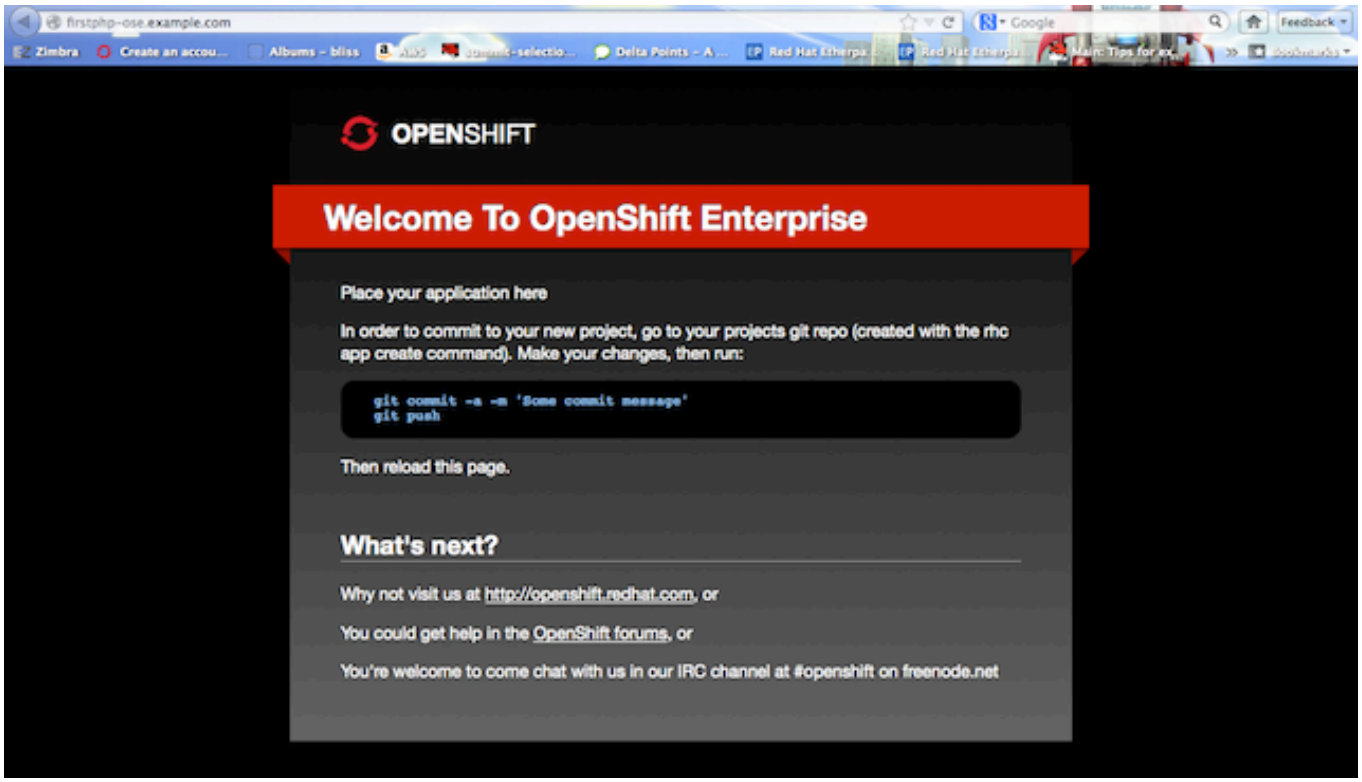
Notice that we stop the application runtime (Apache), deploy the code, and then run any action hooks that may have been specified in the .openshift directory.

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 involves adding the new file to your git repository, committing the change, and then pushing the code to your OpenShift Enterprise gear:

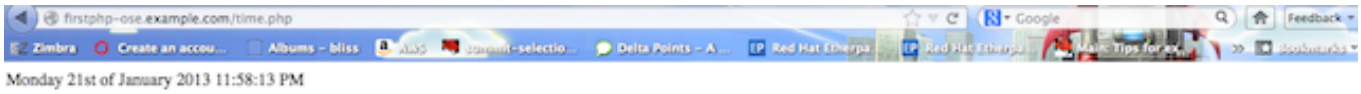
```
$ git add .
$ git commit -am "Adding time.php"
$ git push
```

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

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, the 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: 52afd7bc3a0fb277cf000070
----------------------------------------------------------------------------
  Domain:     ose
  Created:    Dec 16  9:49 PM
  Gears:      1 (defaults to small)
  Git URL:    ssh://52afd7bc3a0fb277cf000070@firstphp-ose.apps.example.com/~/git
  SSH:        52afd7bc3a0fb277cf000070@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 a user starts 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, temporary 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 *OPENSHIFT_LOG_DIR* and *OPENSHIFT_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
```

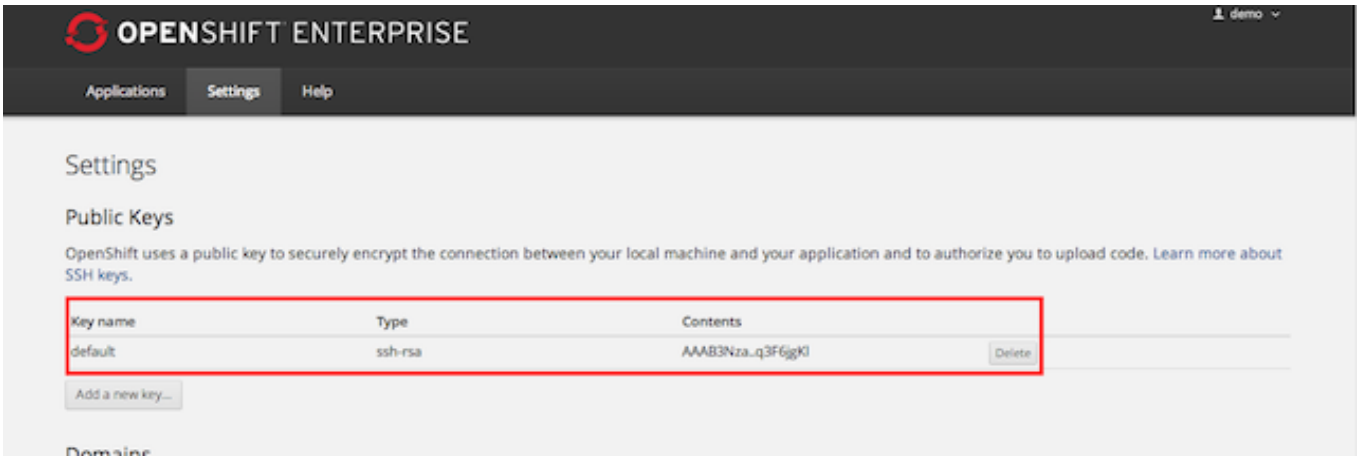
SSH to application gear

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 management console and clicking on the “Settings” tab, as shown below.



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
psql            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) tai
                            100')
  -f, --files files         File glob relative to app (default <application_name
  -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.red
  --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 "-" "Moz
```

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 for the application *firstphp* that 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.

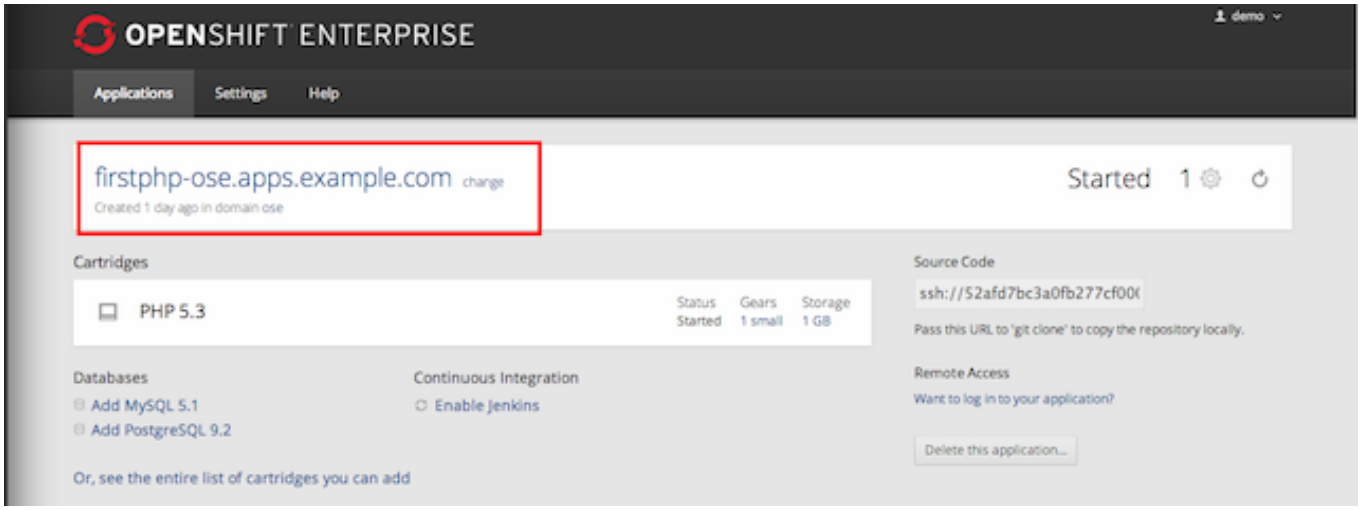
```
$ rhc alias add firstphp www.mycustomdomainname.com
```

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

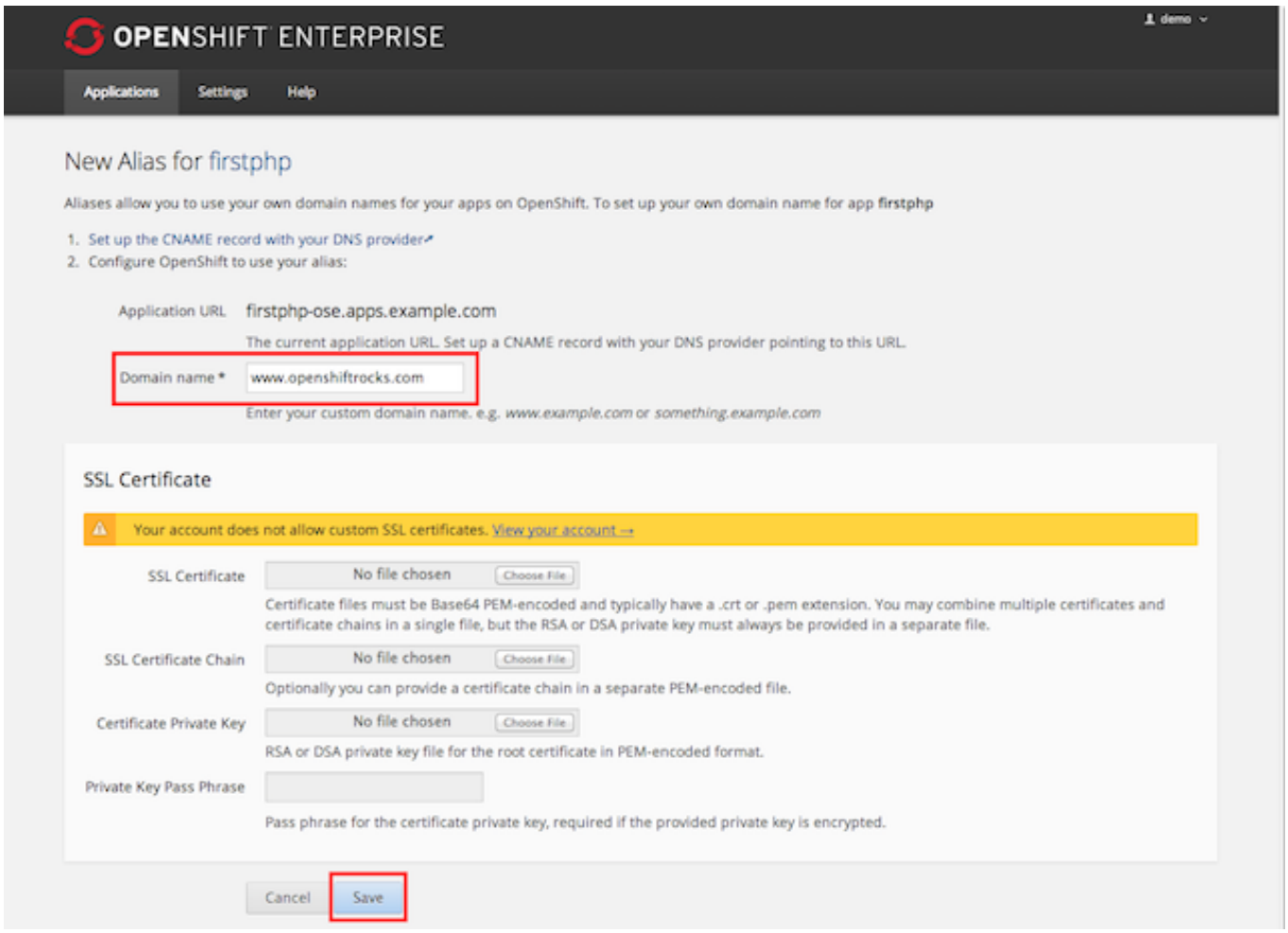
Adding a custom domain to an application using the management console

The OpenShift management 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 management console, click on the *Applications* tab at the top of the screen.

You should see the firstphp application listed:



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* and click the save button.



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: 52afd7bc3a0fb277cf000070
------------------------------------------------------------------------------
  Domain:     ose
  Created:    Dec 16  9:49 PM
  Gears:      1 (defaults to small)
  Git URL:    ssh://52afd7bc3a0fb277cf000070@firstphp-ose.apps.example.com/~/git
  SSH:        52afd7bc3a0fb277cf000070@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, 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 your custom domain name. 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. 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 command 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 opeartion. 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 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

Server used:

- localhost
- node host

Tools used:

- rhc
- mysql
- tail
- git
- PHP

Cartridges provide the actual functionality necessary to run applications. 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:

jbosseap–6 JBoss Enterprise Application Platform 6.1.0 web

jenkins–1 Jenkins Server web
nodejs–0.10 Node.js 0.10 web
perl–5.10 Perl 5.10 web
php–5.3 PHP 5.3 web
python–2.6 Python 2.6 web
python–2.7 Python 2.7 web
ruby–1.8 Ruby 1.8 web
ruby–1.9 Ruby 1.9 web
jbossews–1.0 Tomcat 6 (JBoss EWS 1.0) web
jbossews–2.0 Tomcat 7 (JBoss EWS 2.0) web
diy–0.1 Do-It-Yourself 0.1 web
cron–1.4 Cron 1.4 addon
jenkins-client–1 Jenkins Client addon
mysql–5.1 MySQL 5.1 addon
postgresql–8.4 PostgreSQL 8.4 addon
postgresql–9.2 PostgreSQL 9.2 addon
haproxy–1.4 Web Load Balancer addon

Note: Web cartridges can only be added to new applications.

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:

Adding 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
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/52afd7bc3a0fb277cf000070/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/52afd7bc3a0fb277cf000070/mysql//soc
OPENSHIFT_MYSQL_DB_URL=mysql://adminxzGaLVm:9svQXLVtv89Y@127.10.134.130:3306/
OPENSHIFT_MYSQL_DB_LOG_DIR=/var/lib/openshift/52afd7bc3a0fb277cf000070/mysql//lo
```

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 *cartridge-start* command.

```
$ 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 *cartridge-show* command. 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 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/fi
   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 management console to create applications

Server used:

- localhost

Tools used:

- OpenShift Enterprise Management 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-based management console. During this lab, we will explore the creation and management of application using the management console.

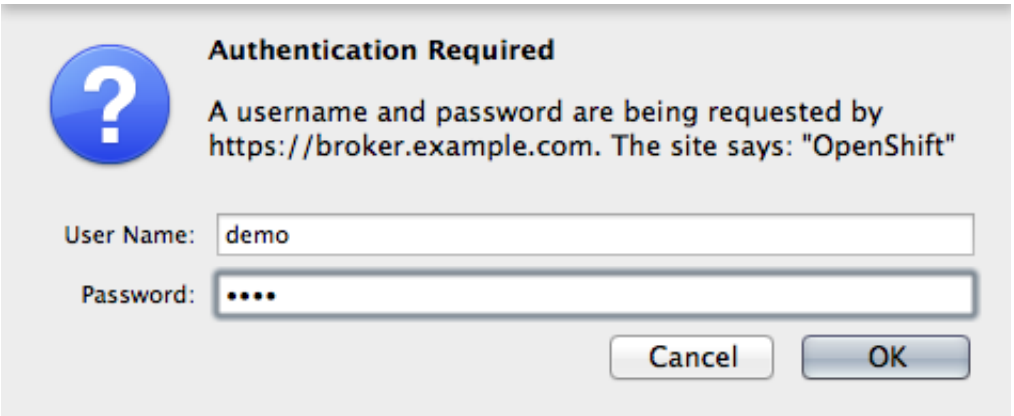
Having DNS resolution setup on your local machine, as discussed in a previous lab, is crucial in order to complete this lab.

Authenticate to the management console

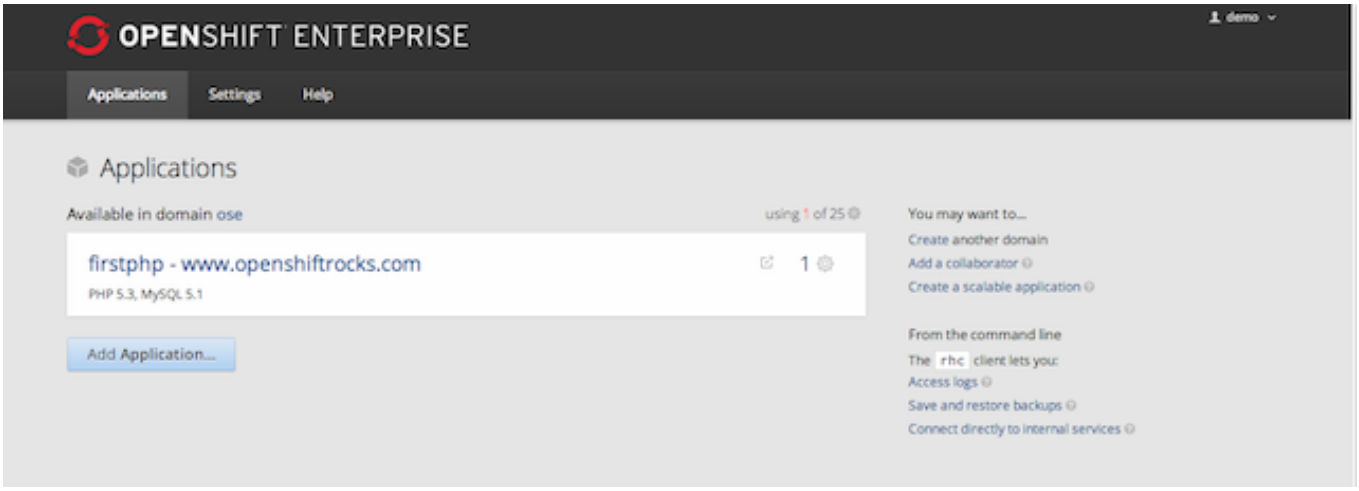
Open your favorite web browser and go to the following URL:

```
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.

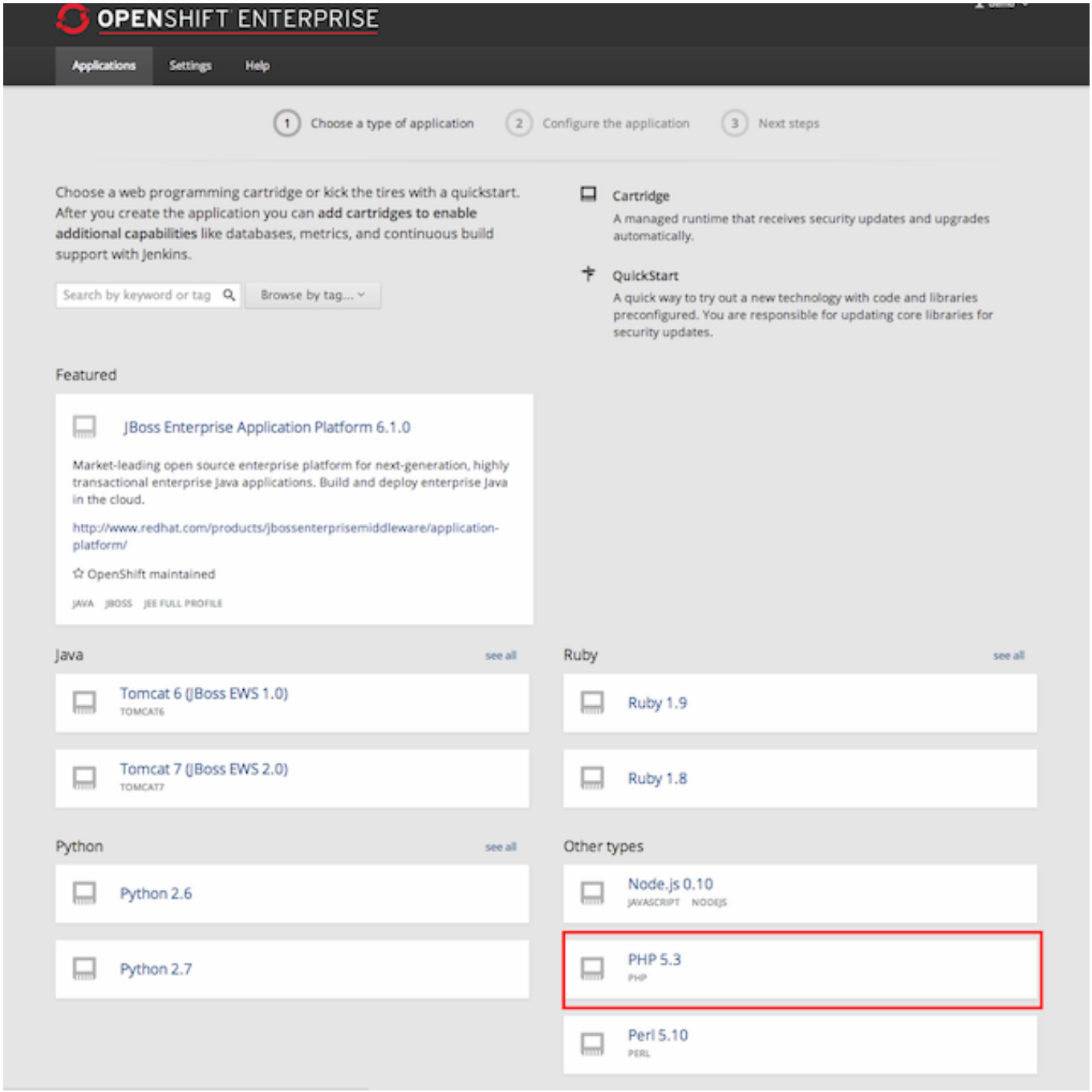


After entering in valid credentials, you will see the OpenShift Enterprise management console dashboard:

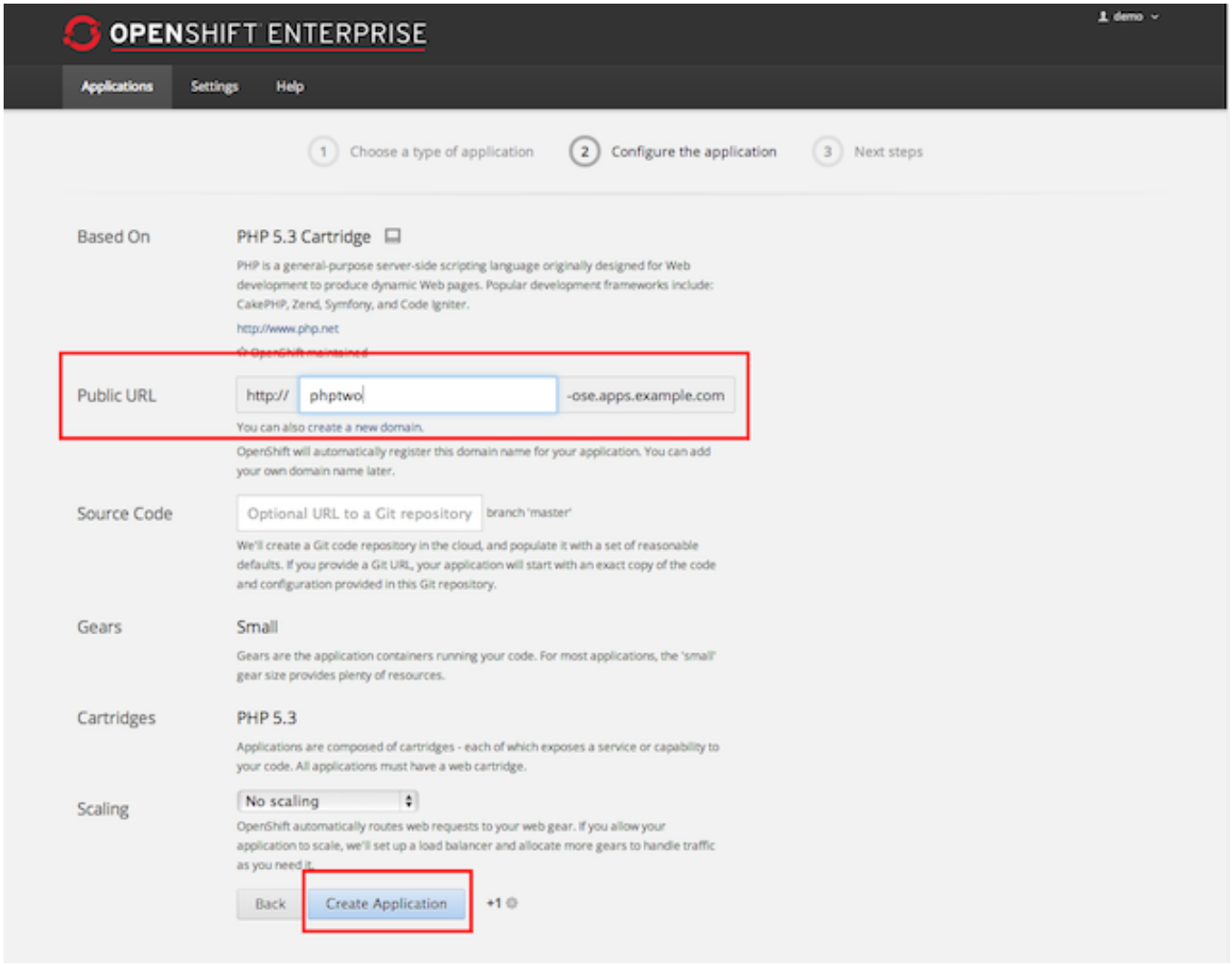


Creating a new application

In order to create a new application using the management 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*.

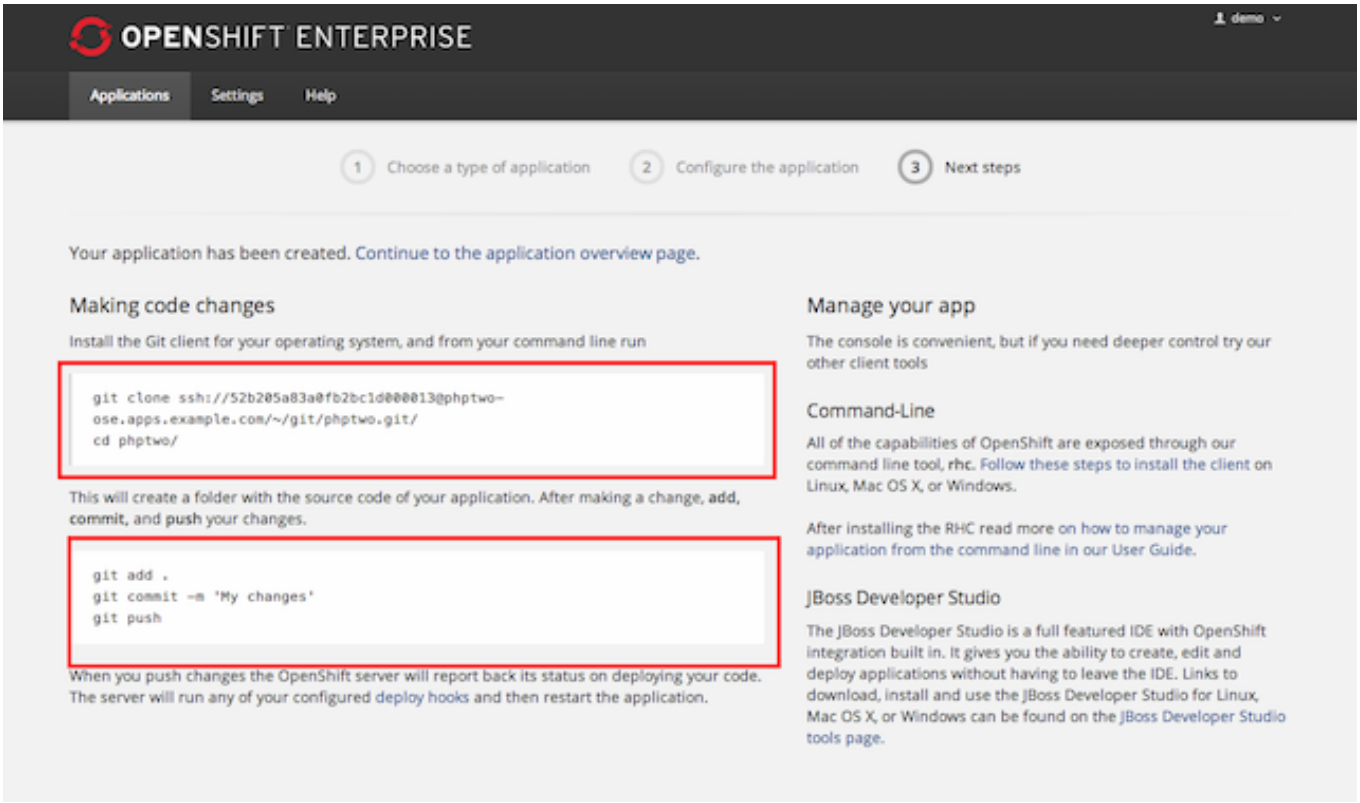


After selecting to create a new PHP application, specify the name of your application:



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

- Git repository URL
- Instructions for making code changes



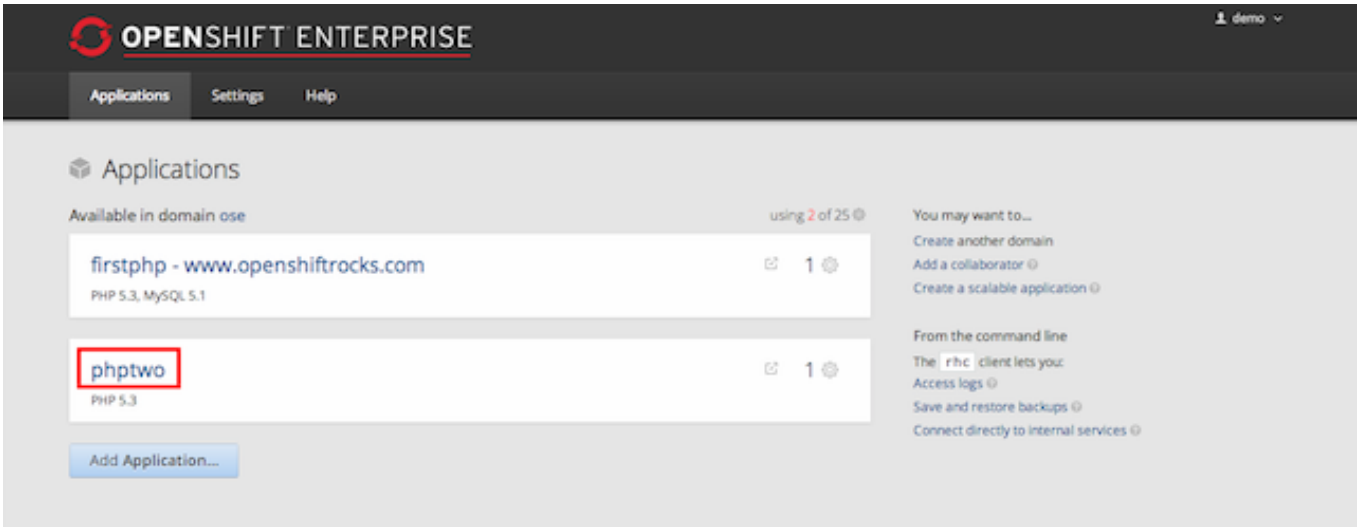
Clone your application repository

Open up a command prompt and clone your application repository with the instructions provided on the management 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.

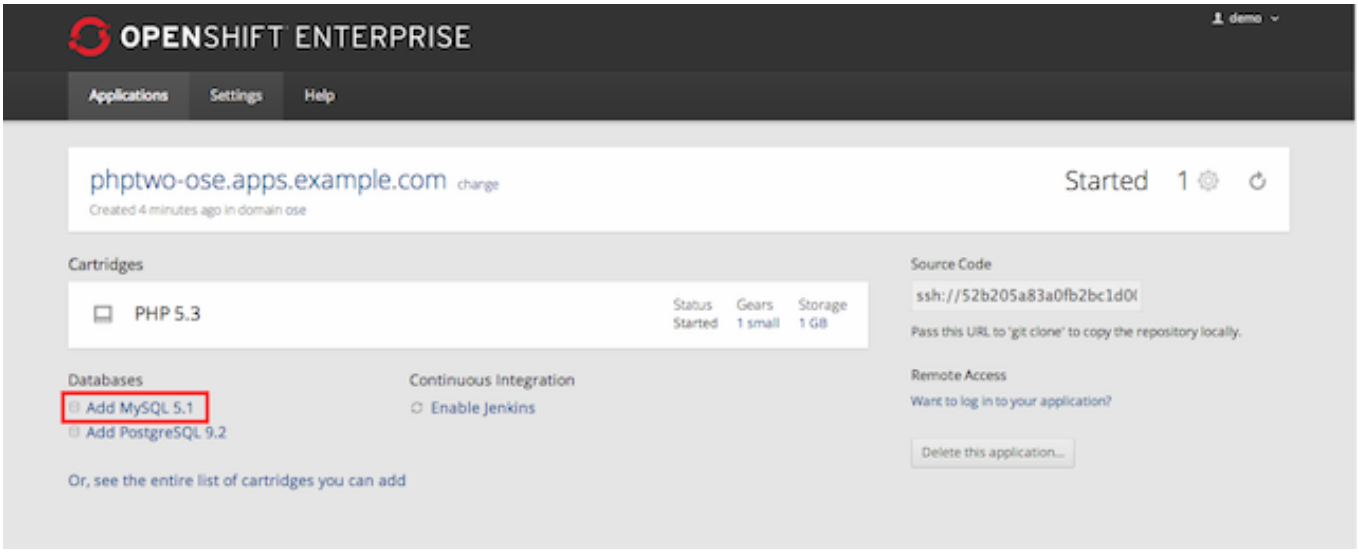
Once you have made a code change, view your application in a web browser to ensure that the code was deployed correctly to your server.

Adding a cartridge with the management console

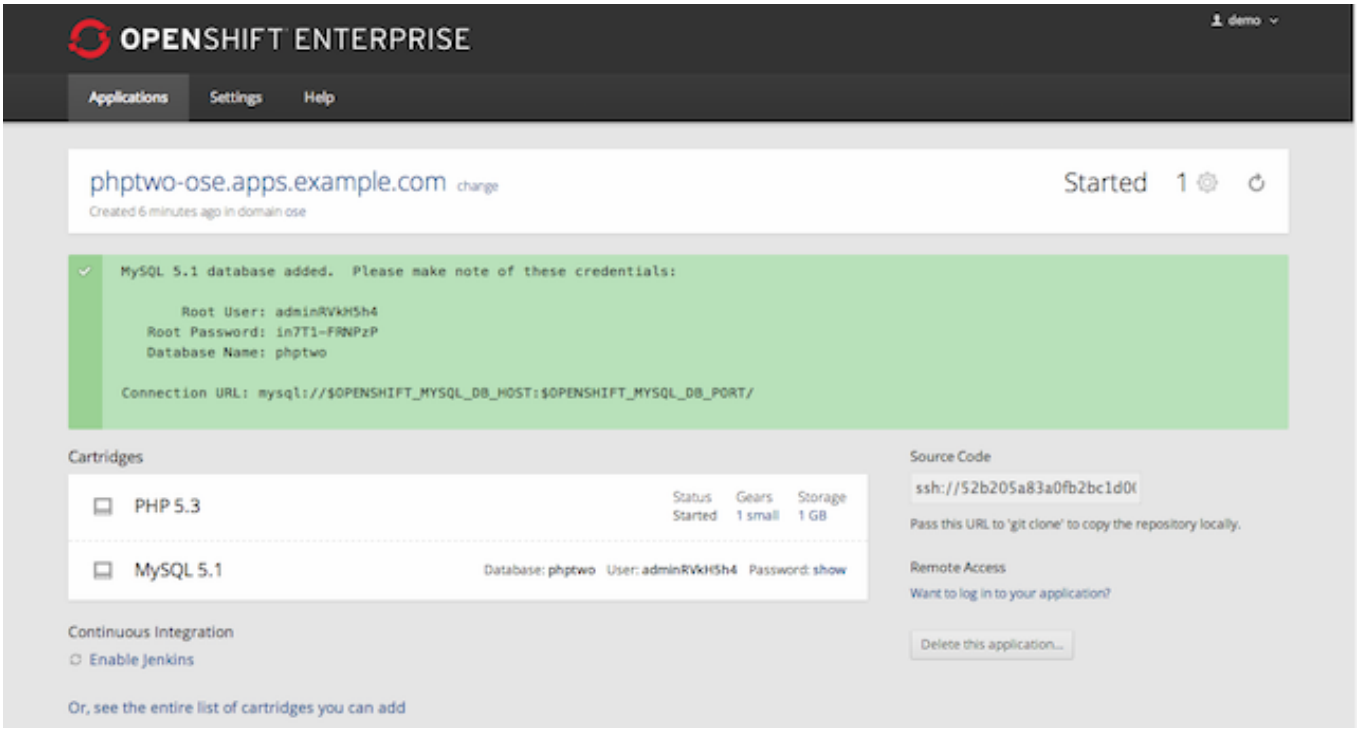
Click on the *My Applications* tab at the top of the screen and then select the *Phptwo* application by clicking on it.



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.



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



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 whether 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: Using the admin console

Server used:

- localhost

Tools used:

- OpenShift Enterprise Admin Console

OpenShift Enterprise 2.0 includes the first version of an Admin Console that will allow an Admistrator to gain valuable insights into the usage of the platform. Having this visibility will allow the adminstrator to perform capacity planning as well as view metrics of the platform as a whole.

Note: The current iteration of the Admin Console is set to read-only.

Enable external access to the admin console

The default location for the admin console is at the following URL:

```
http://brokers.hosts.example.com/admin-console
```

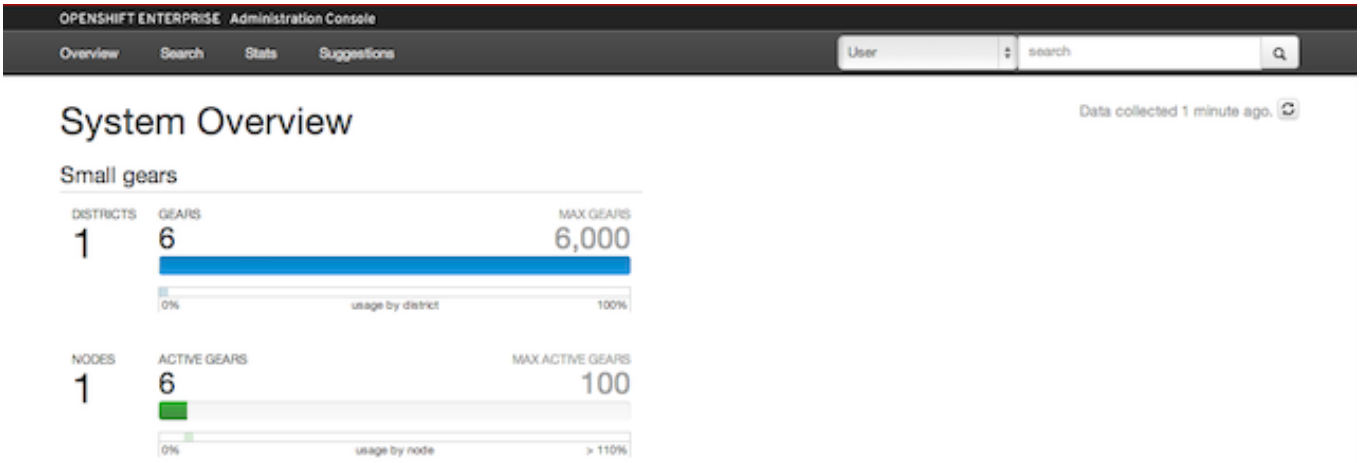
However, by default the admin console is restricted so that it will not allow external traffic. In order to enable access to the admin console, you will need to create an SSH tunnel to the broker host.

System Overview

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

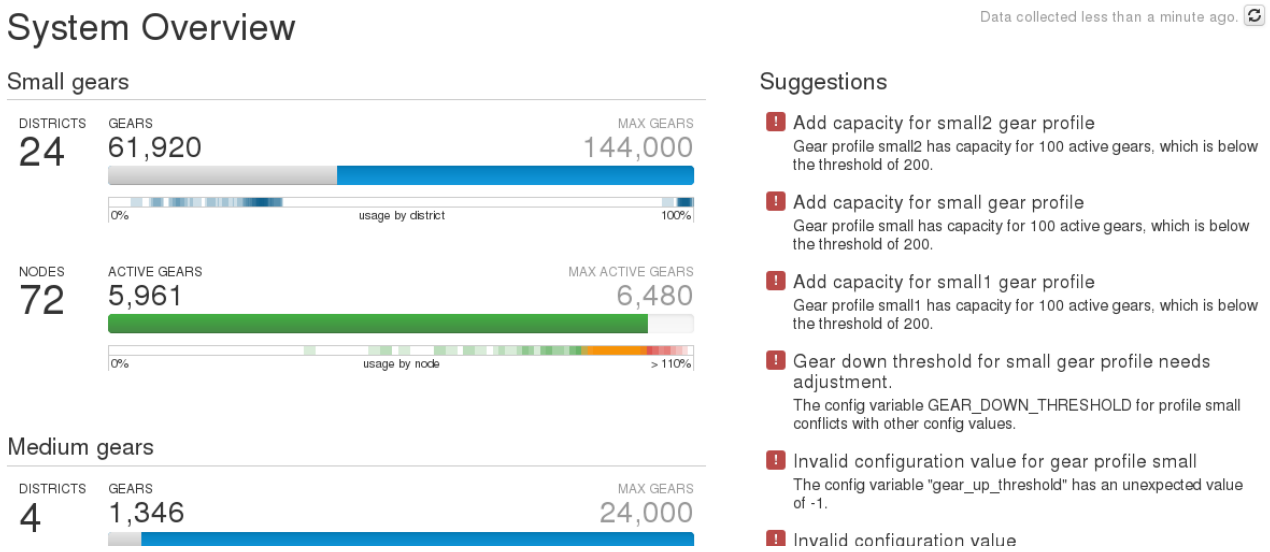
```
http://broker.hosts.example.com/admin-console
```

Once you enter the above URL, you will the admin console *System Overview* dashboard.



On this page you can see the information for the number of disticts and nodes that you have deployed in your environment for each available gear type. In our lab, you will only see a single district with small gears. However, in a production deployment of OpenShift Enterprise 2.0, the administrator will be able to view information for all available gear types.

For reference, a production environment may look look like the following image:



Viewing Gear Profiles

Click on a gear profile's *Details* link from the System Overview page to view more information about it. Each gear profile page provides the same summary for the respective gear profile as seen on the System Overview page and allows you to toggle between viewing the relevant districts or nodes. The DISTRICTS view shows all of the districts in that gear profile, and by default sorts by the fewest total gears remaining, or the most full districts. Each district displays a progress bar of the total gears and a link to view the nodes for that district.

The DISTRICTS view also displays a threshold indicator. The threshold is a configurable value for the target number of active gears available on a node. Each node for the district appears as either over

(displayed in red) or under (displayed in green) the threshold. Each bar is slightly opaque to allow for multiple bars of the same type to show through. Therefore, if there is an intense red or green color, then several nodes are either over or under the threshold.

At any point in time you can refresh the statistics collected by clicking the refresh button in the upper right hand corner of the detail page.

Viewing Suggestions

The admin console also provides a suggestion system that will make reccomendations on the current deployment. In order to view any suggestions click on the *Suggestions* tab at the top of the screen. During this training lab, our install if a fairly basic deployment with a minimal number of applications created and deployed on the platform. Because of this, you may not see any suggestions offered from the console.

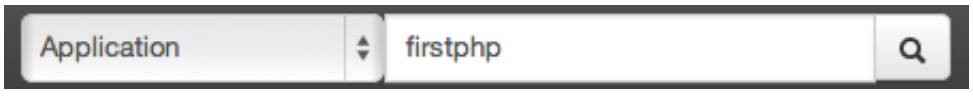
Any suggestions will also be displayed on the *System Overview* page on the right hand side of the screen.

Searching for Application Entities

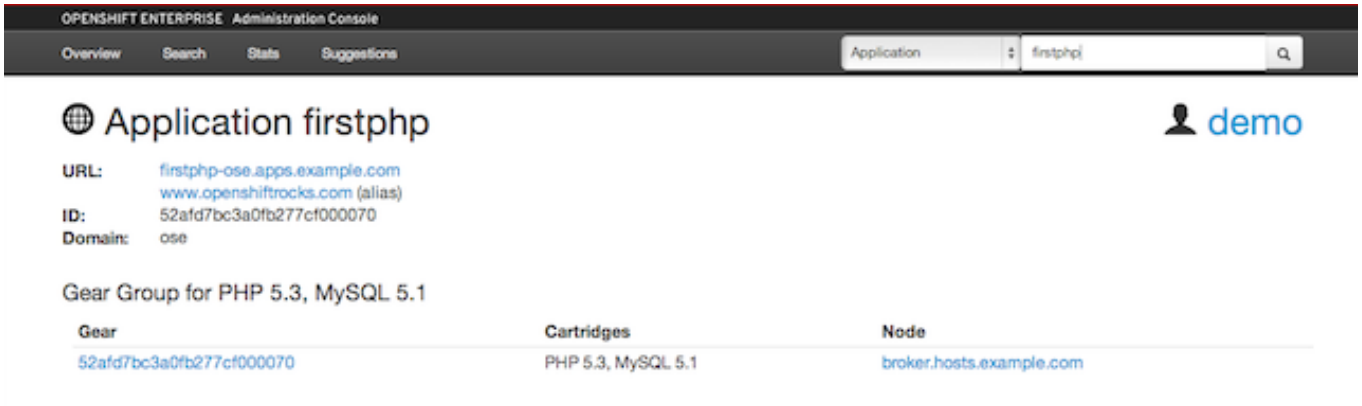
The upper right section of every page of the Administration Console contains a search box, providing a quick way to find OpenShift Enterprise entities. Additionally, the dedicated Search page provides more information on the expected search queries for the different entities, such as Applications, Users, Gears, and Nodes.

The search does not intend to provide a list of possible matches; it is a quick access method that attempts to directly match the search query. Applications, User, Gear, and Node pages link to each other where possible. For example, a User page links to all of the user's applications, and vice versa.

Let's use the search feature to find the *firstphp* application that we created in a previous lab. In the upper right hand corner of the admin console, select application from the dropbox box and then enter in *firstphp* in the search field.



After clicking the search button, you will the details for the application displayed in the main browser window as shown below:

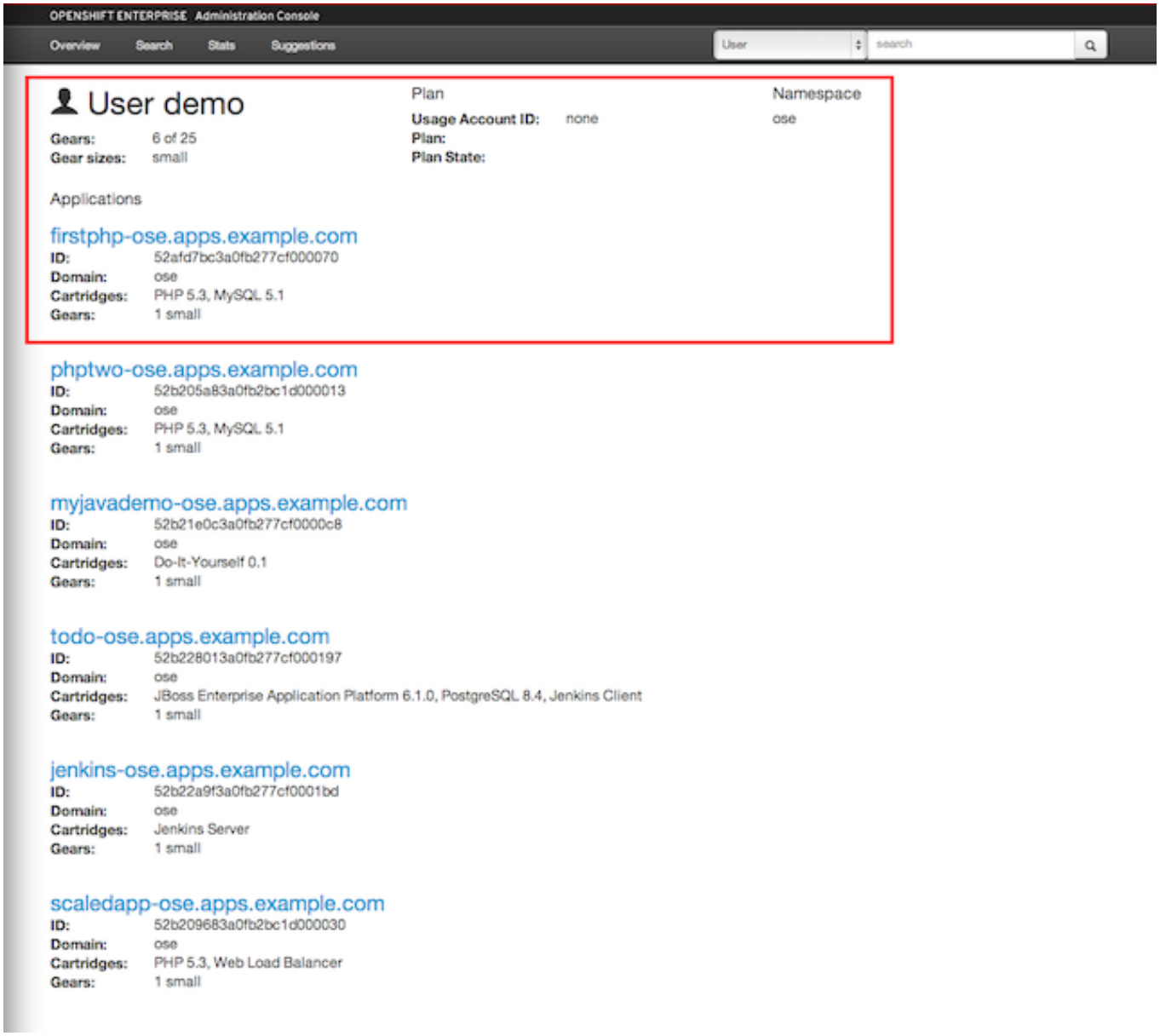


On this application detail page, you can view all of the information about the application including the URL, any aliases assigned to the application, the Linux user id that was created for the gear, the domain, which node the gear resides on, and any cartridges that application is using. You can drill down even further by clicking on any of the link presented on the application detail page.

Searching for User Entities

In the previous section, we searched for our application that we created as part of this training class. We can also use the search funtionality to view detailed information for a user of the platform. Using the skills you learned in the previous section, search for the *demo* user.

You should see a detail screen listing the applications the user has deployed.



Using Data with External Tools

The Administration Console exposes OpenShift Enterprise 2.0 system data for use by external tools. In the current iteration of the Administration Console, you can retrieve the raw data from some of the application controllers in JSON format. This is not a long-term API however, and is likely to change in future releases. You can access the following URLs by appending them to the appropriate host name:

Exposed Data Points*

- /admin-console/capacity/profiles.json returns all profile summaries from the Admin Stats library (the same library used by the oo-stats command). Add the ?reload=1 parameter to ensure the data is current rather than cached.
- /admin-console/stats/gears_per_user.json returns frequency data for gears owned by a user.
- /admin-console/stats/apps_per_domain.json returns frequency data for applications belonging to

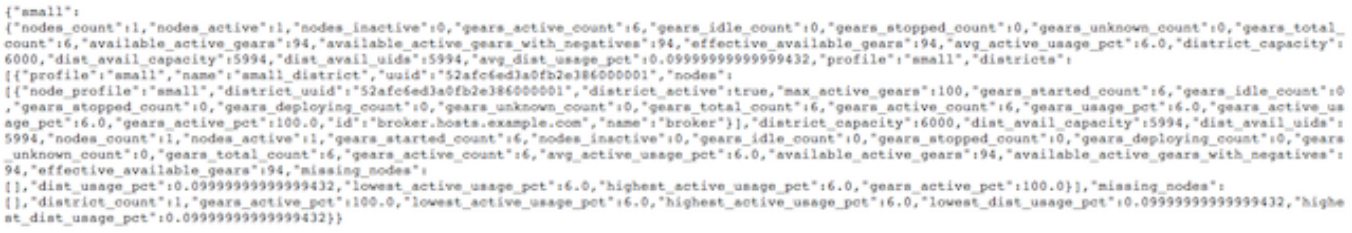
a domain.

- /admin-console/stats/domains_per_user.json returns frequency data for domains owned by a user.

To verify this, you can enter in the following URL in your web browser:

```
http://broker.hosts.example.com/admin-console/capacity/profiles.json
```

You should see output similar to the following:



Having these data points avaiable in a consumable JSON fashion will allow administrators to use external tools to monitor the OpenShift Enterprise 2.0 environment.

Lab 15 Complete!

Lab 16: Scaling an application

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 can consume multiple gears: one for the high-availability proxy (HAProxy) itself, and one or more for your actual application. 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 management 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

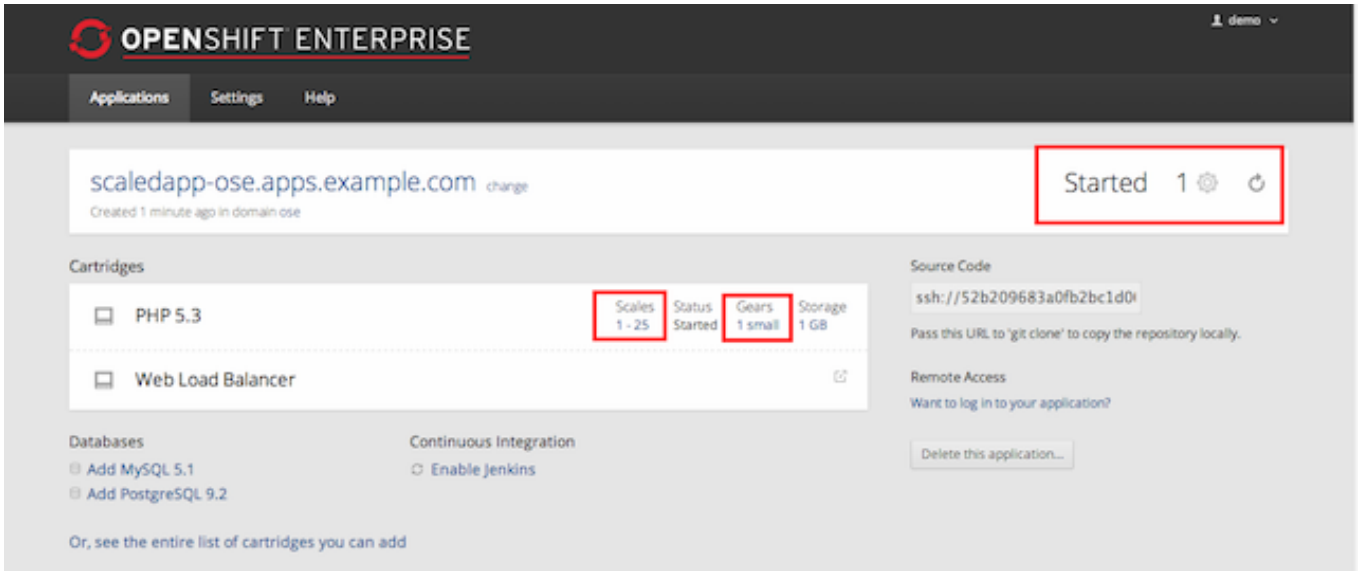
In order to create a scaled application using the *rhc* command line tools, you need to specify the *-s* switch to the command. Let’s create a scaled PHP application with the following command:

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

After executing the above command, you should see output that specifies that scaling is enabled:

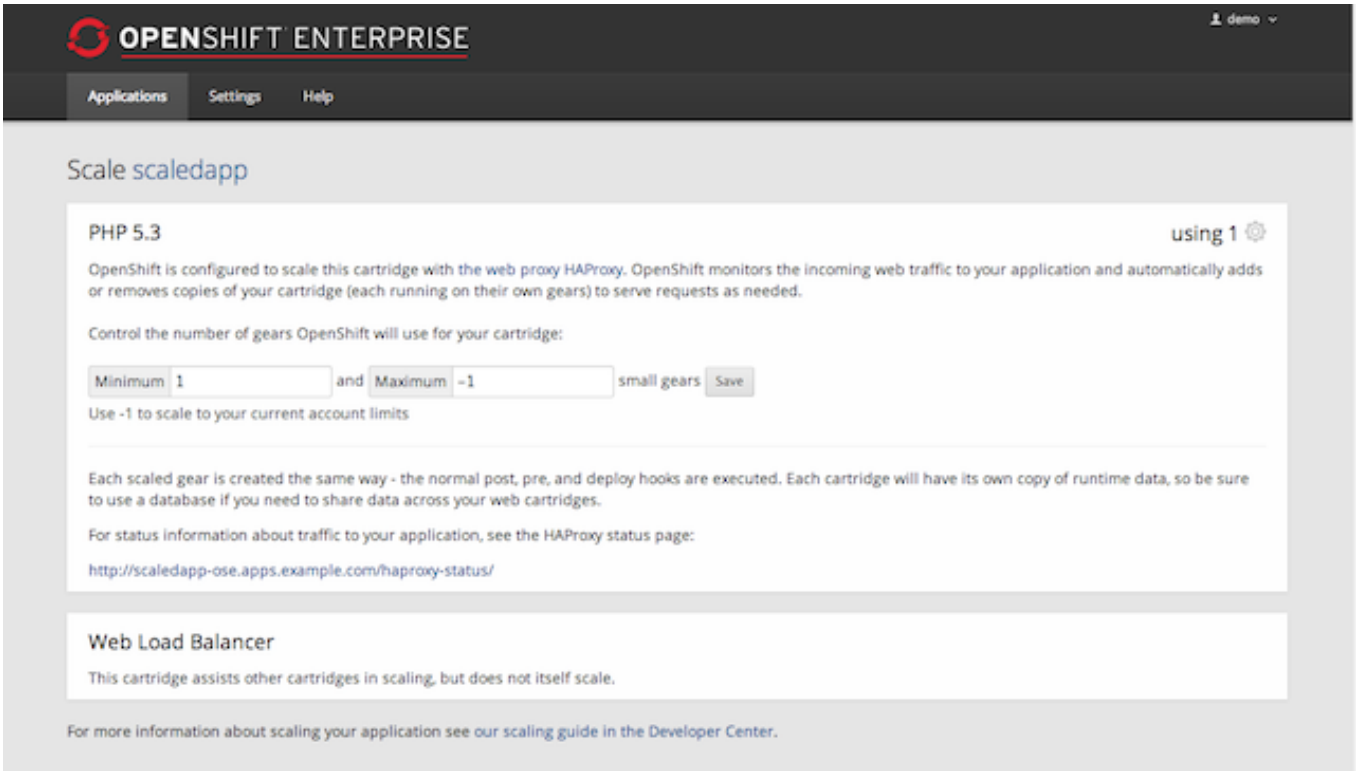
```
Application Options
-------------------
  Domain:     ose
  Cartridges: php-5.3
  Gear Size:  default
  Scaling:    yes

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
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
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/~/gi
  Cloned to:  /Users/gshipley/code/ose/scaledapp

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

Log in to the management 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.

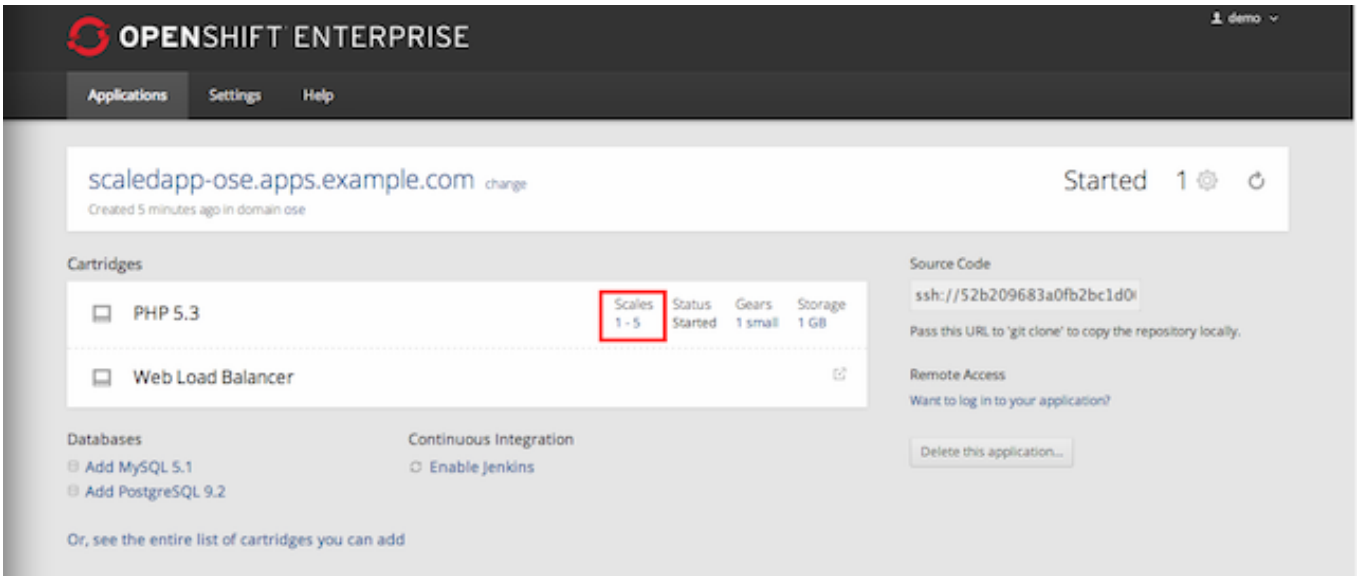


Setting the scaling strategy

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 management console. While on the application details screen, click the *Scales 1 - 25* link to change the default scaling rules.



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



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 the following:

- 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 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 looking similar to the following:

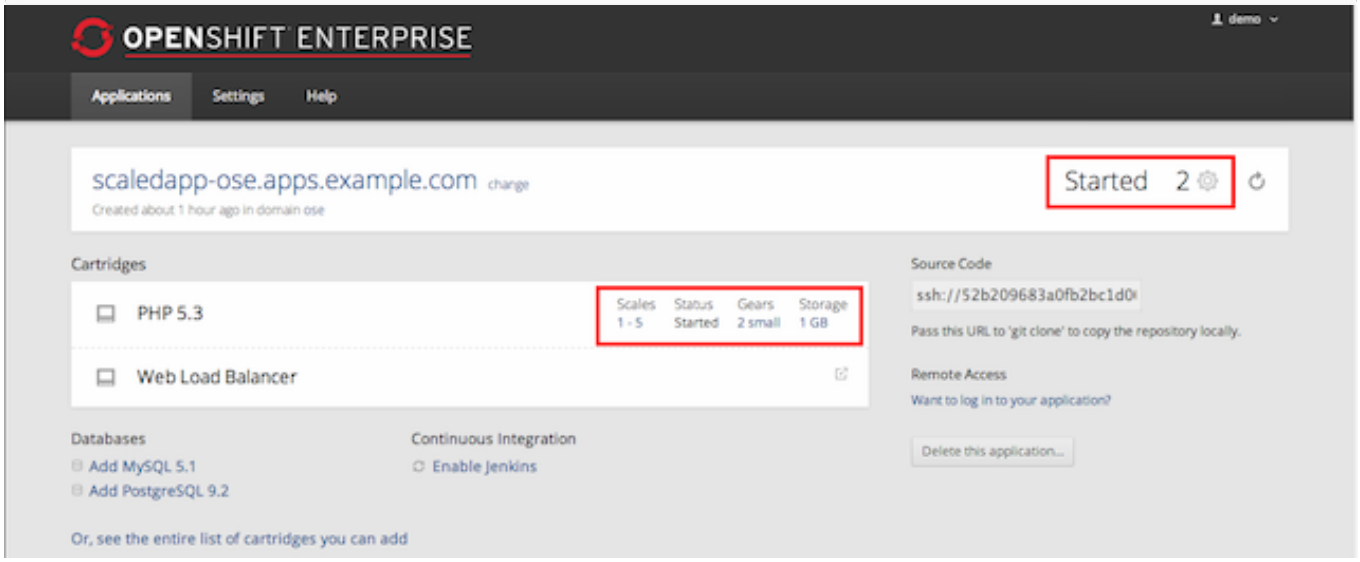
```
[scaledapp-ose.example.com ~]\> add-gear -a scaledapp -u 1a6d471841d84e8aaf25222
```

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 management 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
```



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 1a6d471841d84e8aaf25
```

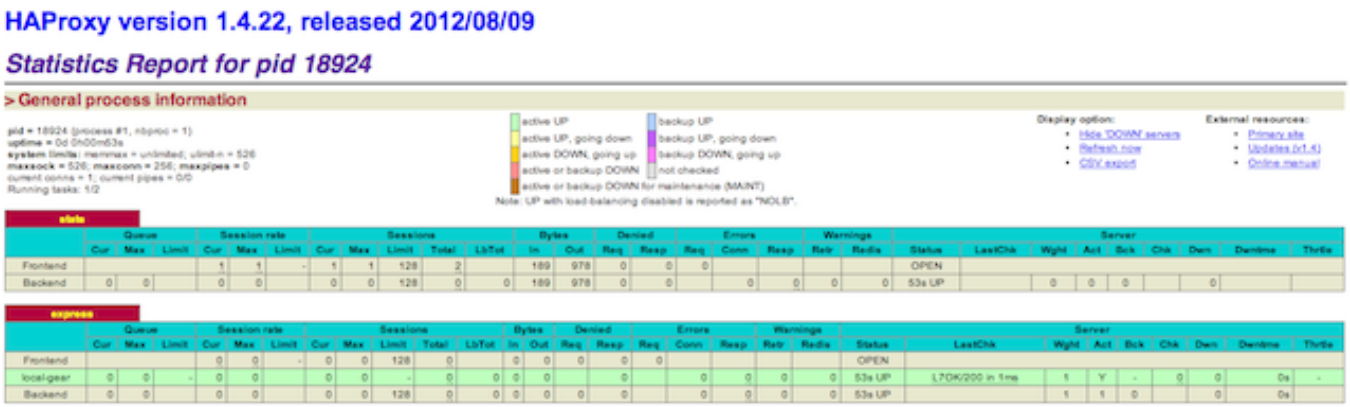
After removing the gear with the *remove-gear* command, verify that the application only contains one gear, HAProxy and a single runtime gear:

```
$  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/
```



Lab 16 Complete!

Lab 17: The DIY application type

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 x86_64 systems. Thus, OpenShift Enterprise can run any binary that will run on RHEL 6.3 x86_64.

The way that the OpenShift Enterprise DIY runtime 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 are a great way for developers to try out unsupported languages, frameworks, or middleware that don’t ship as official OpenShift Enterprise cartridges.

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'
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 c
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/~/g
  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 perform 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

$ 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 17 Complete!

Lab 18: Developing Java EE applications using JBoss EAP

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 appication 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:

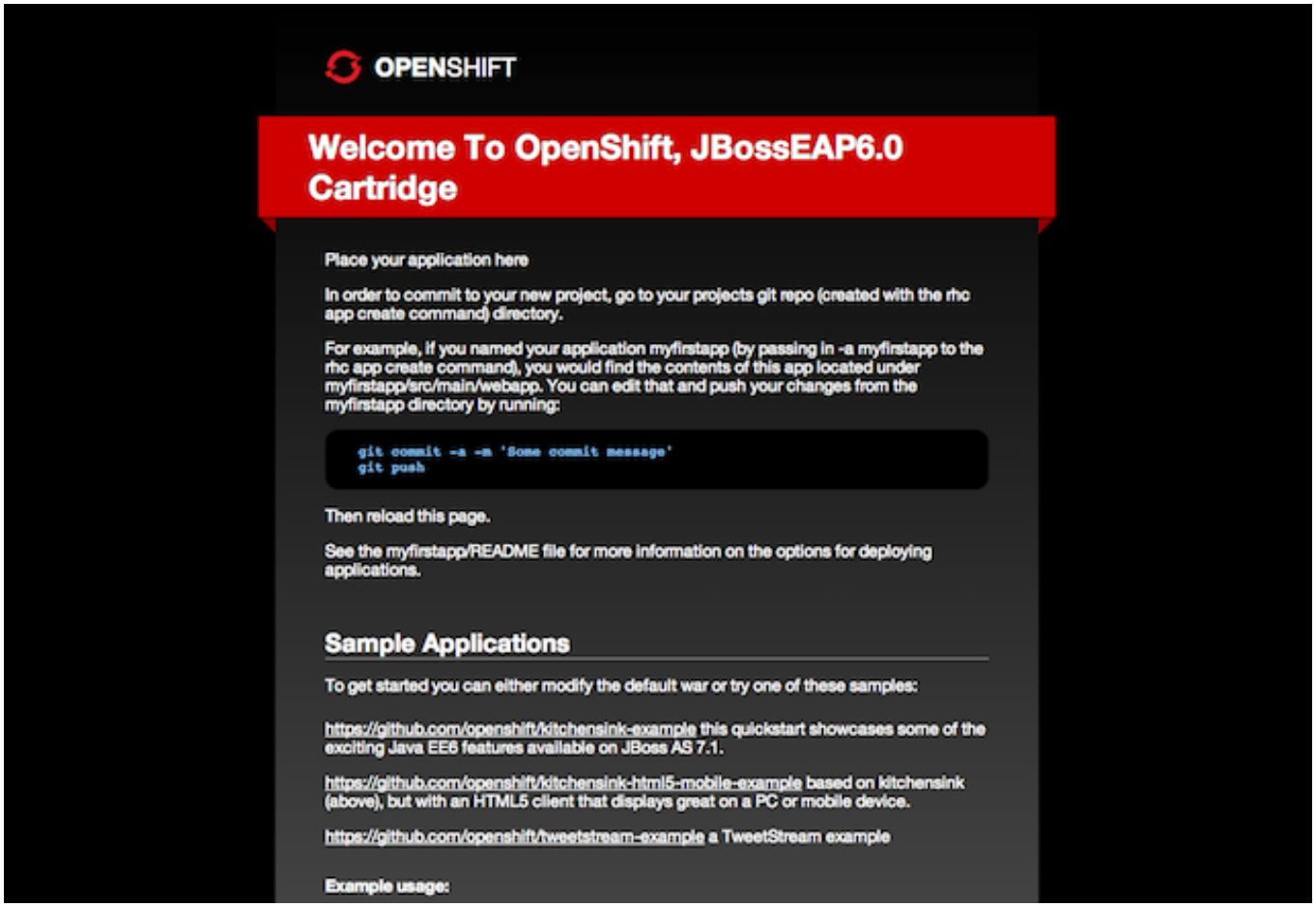
```
http://todo-ose.apps.example.com
```

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 know
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/tod
  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
```



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 nonessential 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 HDScanner 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 application's 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. Then skip ahead to the section on deploying the application.

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.

Extra Credit

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

Lab 18 Complete!

Lab 19: Using Jenkins continuous integration

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 include the following:

- 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 lab 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 management 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 b
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 k
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 username and password that were 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 future 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/tod
  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/
```

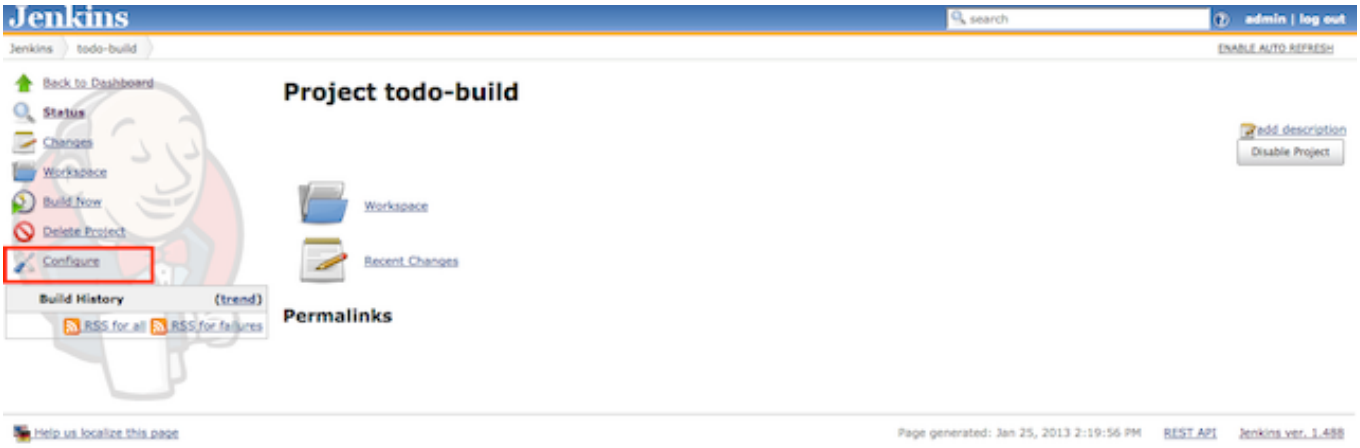
Configuring Jenkins

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 were displayed after adding the Jenkins application.

Once you are authenticated to the Jenkins dashboard, click on the configure link:



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.

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 -DskipTe

# 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/tod
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
```

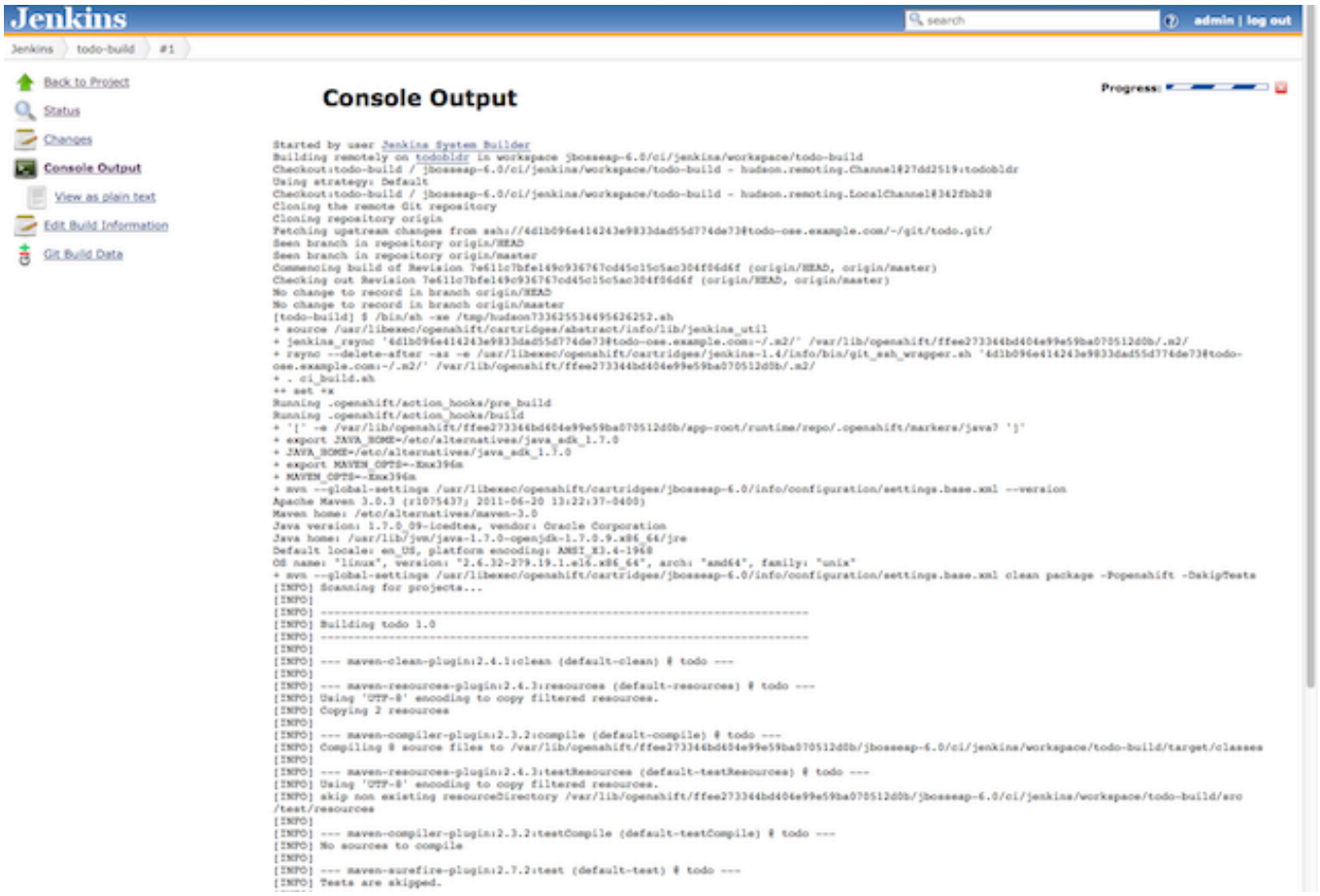
While the build is happening, open up a new terminal window and run the following command:

```
$ 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.examp
      Gear Size = small
      Created   = 2:48 PM
    Cartridges
    ==========
      jbosseap-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.

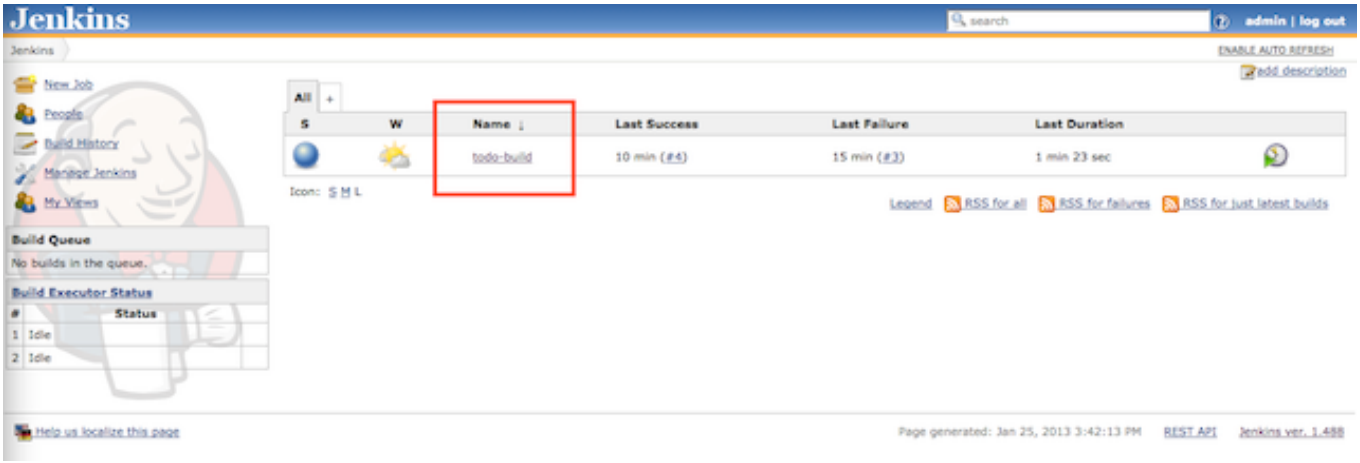


Starting a new build

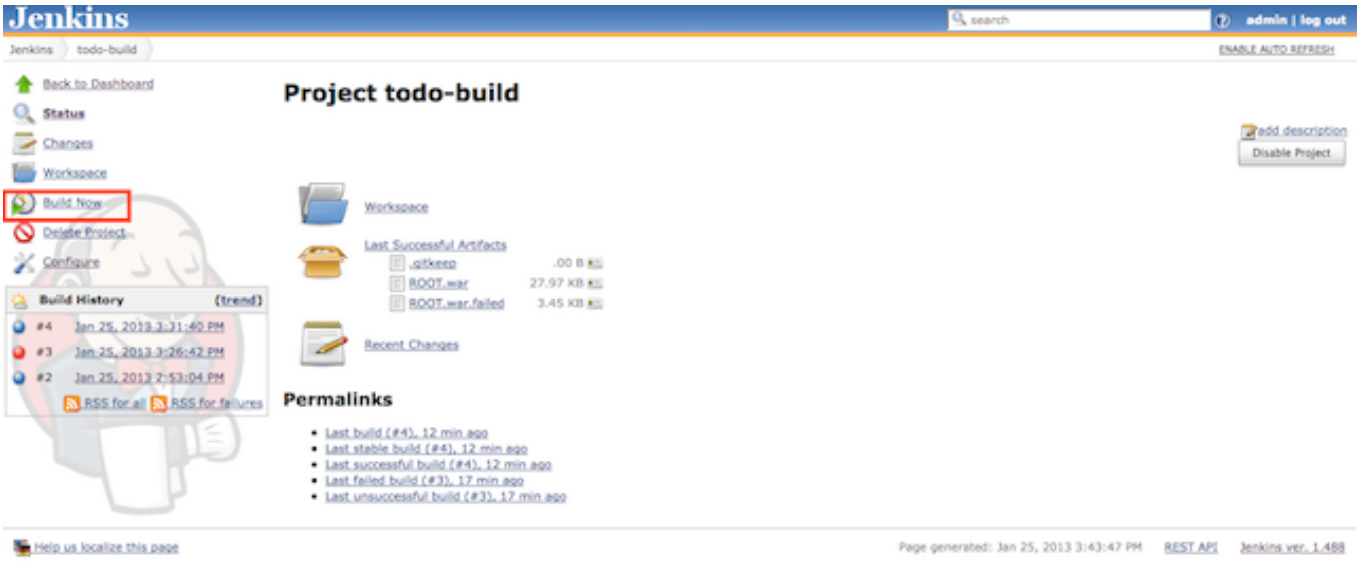
One of the great things about integrating your application with the Jenkins CI environment is the ability to start a new build without having to modify and push your source code. To initiate a new build, log in to the Jenkins dashboard and select the *todo* builder. Point your browser to:

```
https://jenkins-ose.apps.example.com/
```

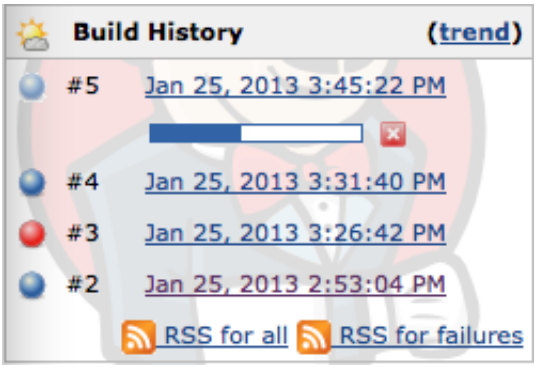
Once you have been authenticated, click the *todo-build* link:



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:



After you click the *Build Now* link, a new build will show up under the links on the left hand side of the screen.



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 19 Complete!

Lab 20: Using JBoss Tools

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 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/
```

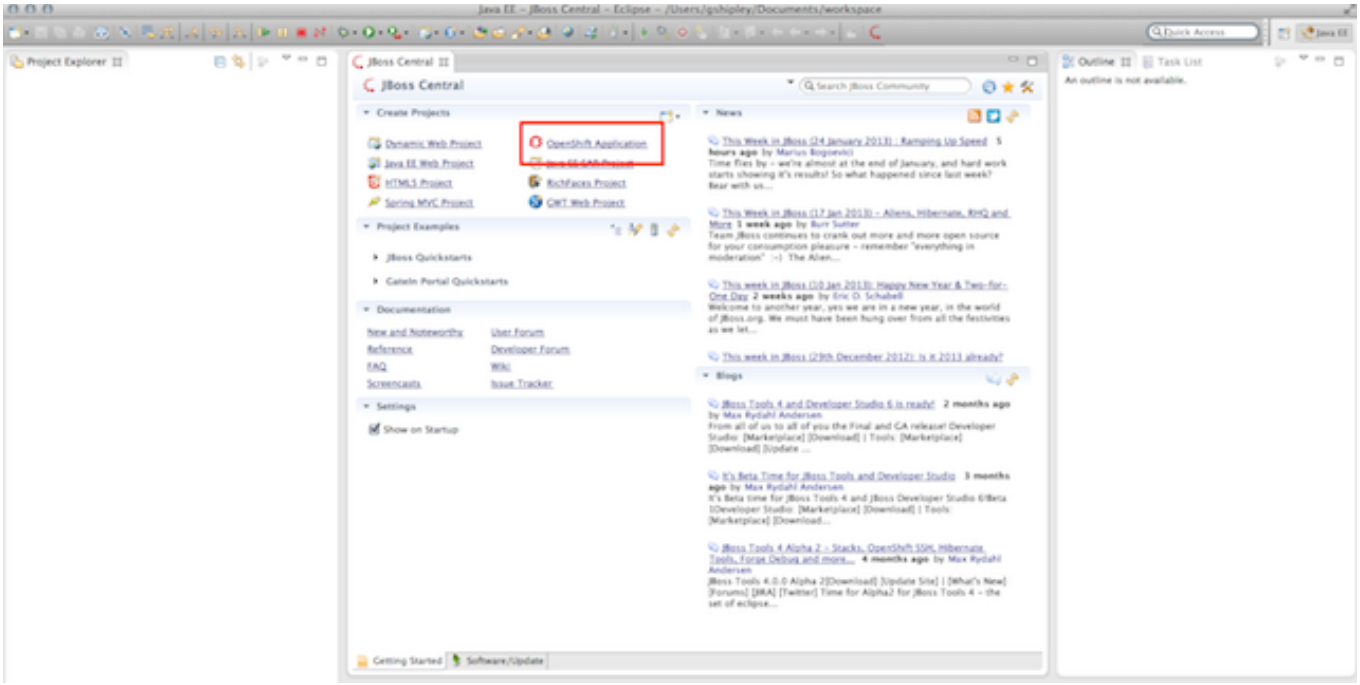
Once you have Eclipse installed, go to the JBoss Tools page located at

```
http://www.jboss.org/tools
```

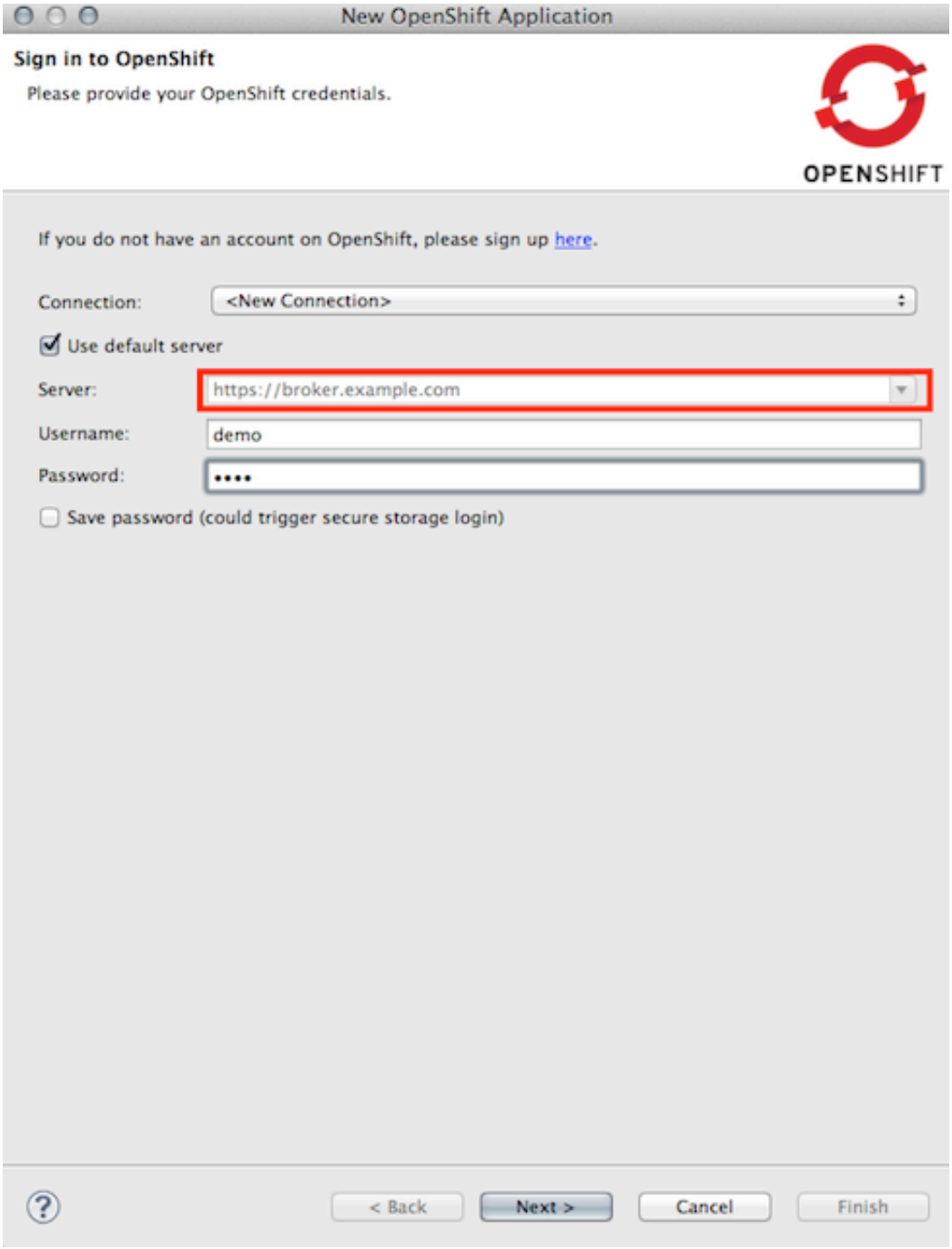
and follow the instructions to install JBoss Tools 4.1 (Kepler). While you are welcome to install all of the JBoss Tools, we will only be using the JBoss OpenShift Tools for this lab.

Create an OpenShift Enterprise application

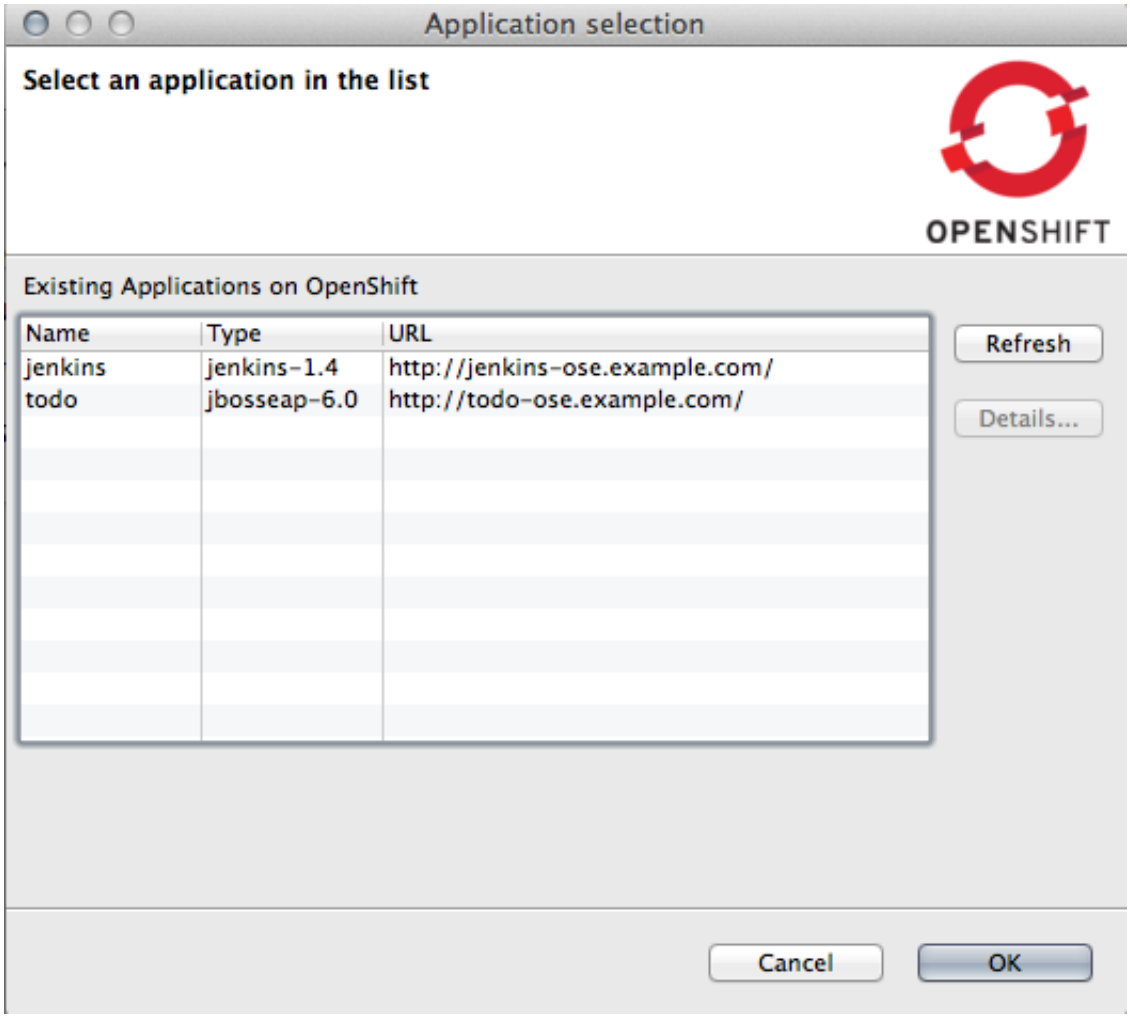
Now that we have Eclipse Juno and JBoss Tools 4.0 installed, we can create an OpenShift Enterprise application without having the leave the comfort of our favorite IDE. Click on the *OpenShift Application* link that is provided on the JBoss Central screen. If you do not have this link, you can also select *file->new->other->OpenShift Application*.



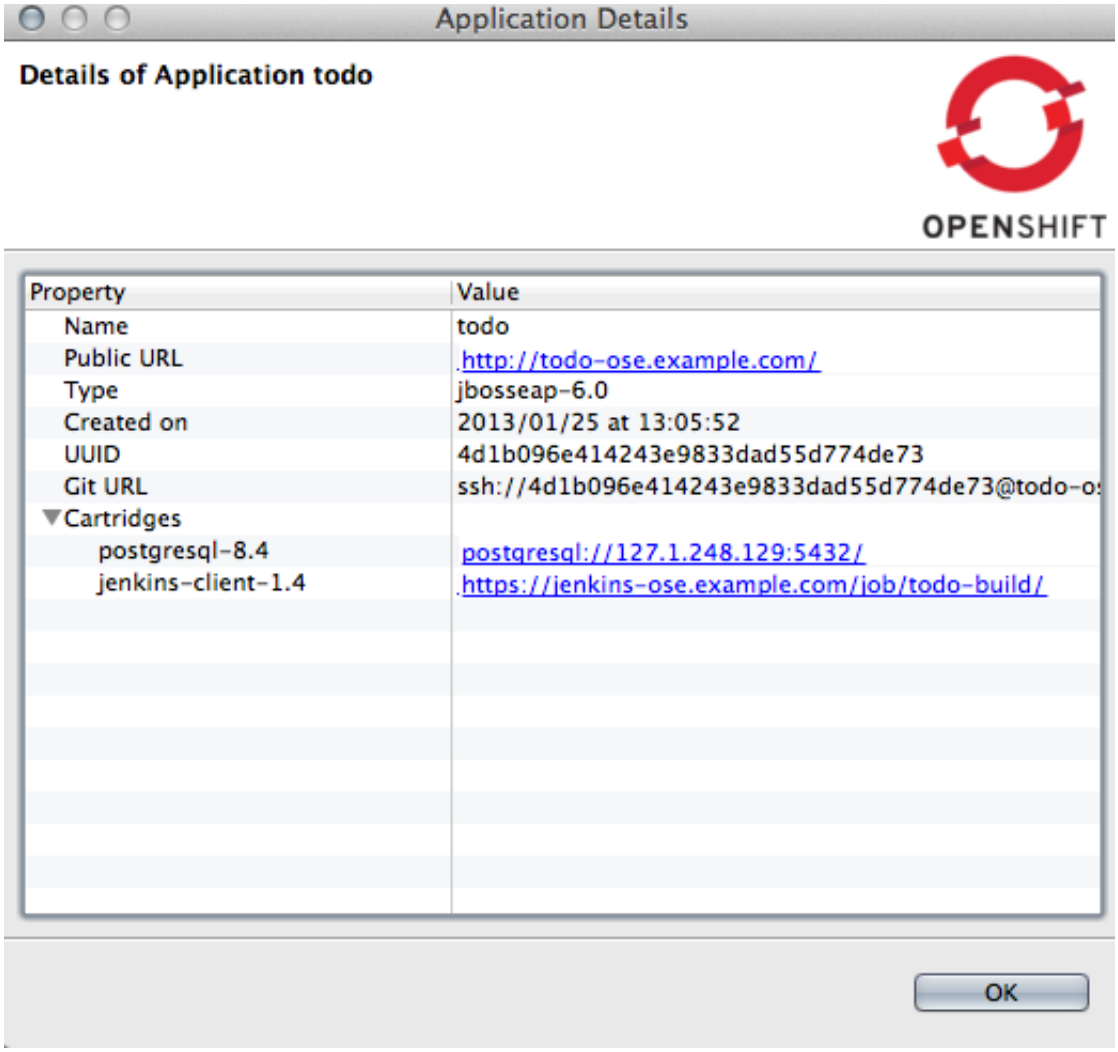
Once you click on the link to create a new OpenShift Enterprise application, you will be presented with a dialog to authenticate to OpenShift Enterprise. Now is also a good time to validate the *Server* setting is correctly set to *broker.hosts.example.com*. If your server does not reflect this, you 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.



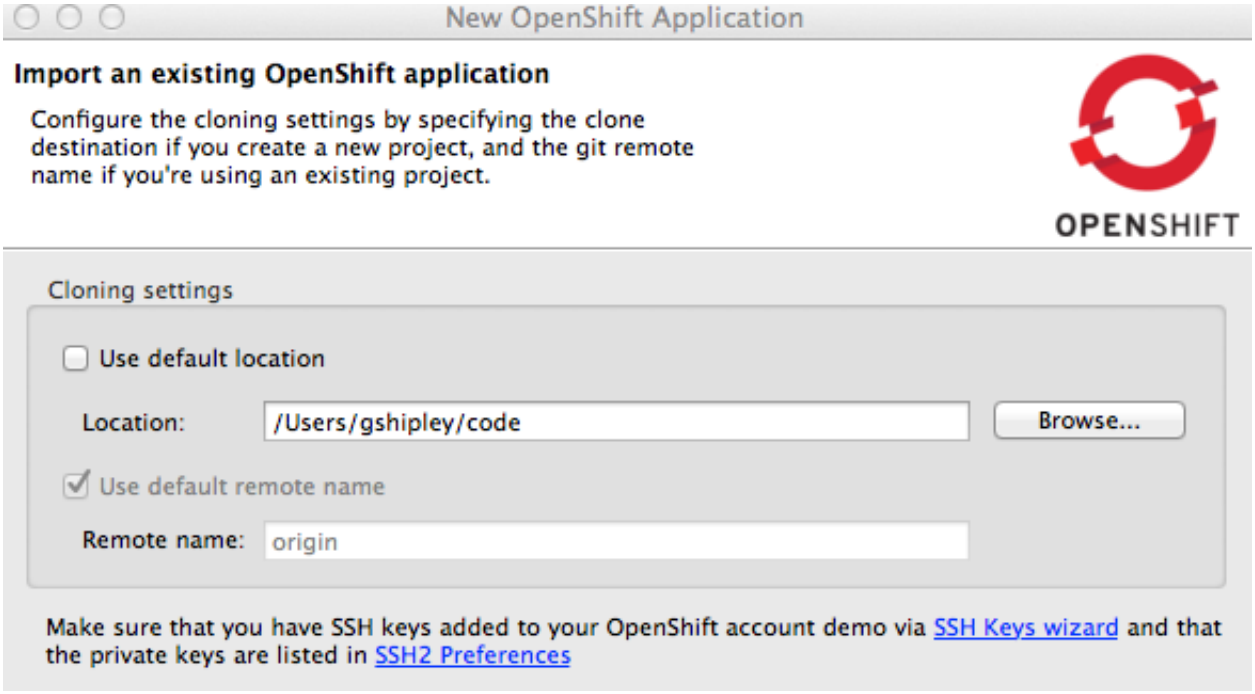
After clicking *next*, the JBoss Tools plugin will authenticate you to the broker host and present another dialog box to you. On this dialog box, you have the option of creating a new application, or to use an existing one. Since we already have a JBoss EAP application deployed, let's select to *Use existing application* and click the *Browse* button. After clicking the *Browse* button, a REST API call will 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.



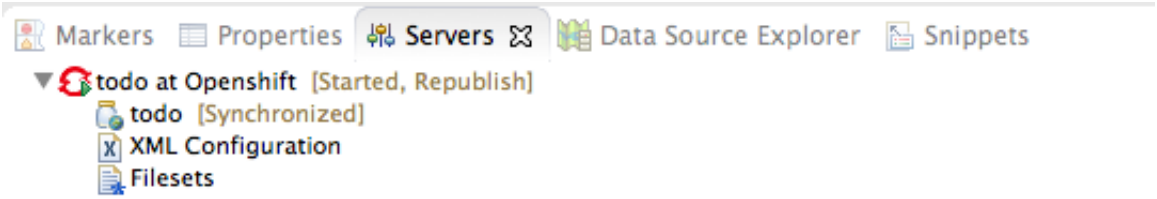
After clicking *Next* to use the existing *todo* application, Eclipse will ask you to create a new project or to use an existing one. Let's create a new one and set the correct location where we want to store the project files.



Once you click the *Finish* button, the existing application will be cloned to your local project.

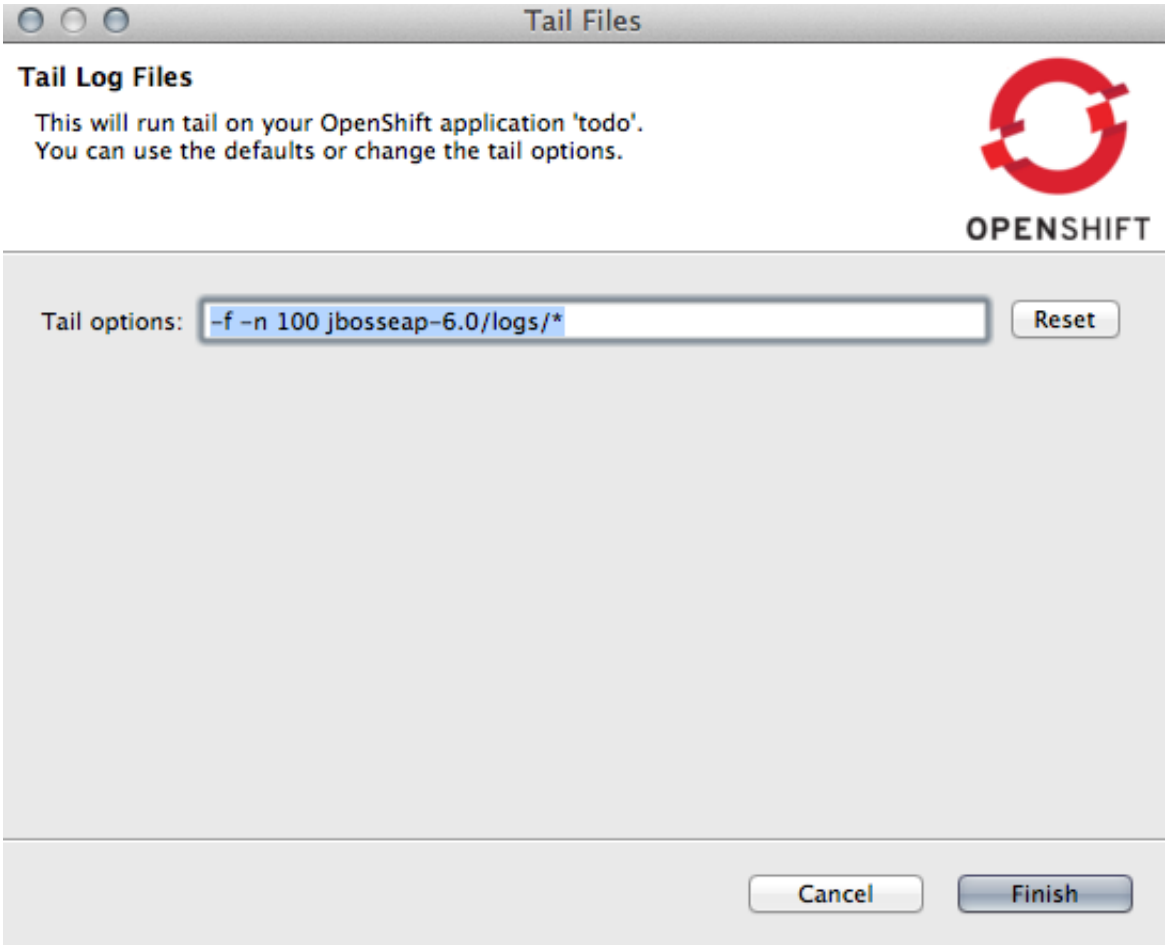
Managing OpenShift Enterprise application with JBoss Tools

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*.



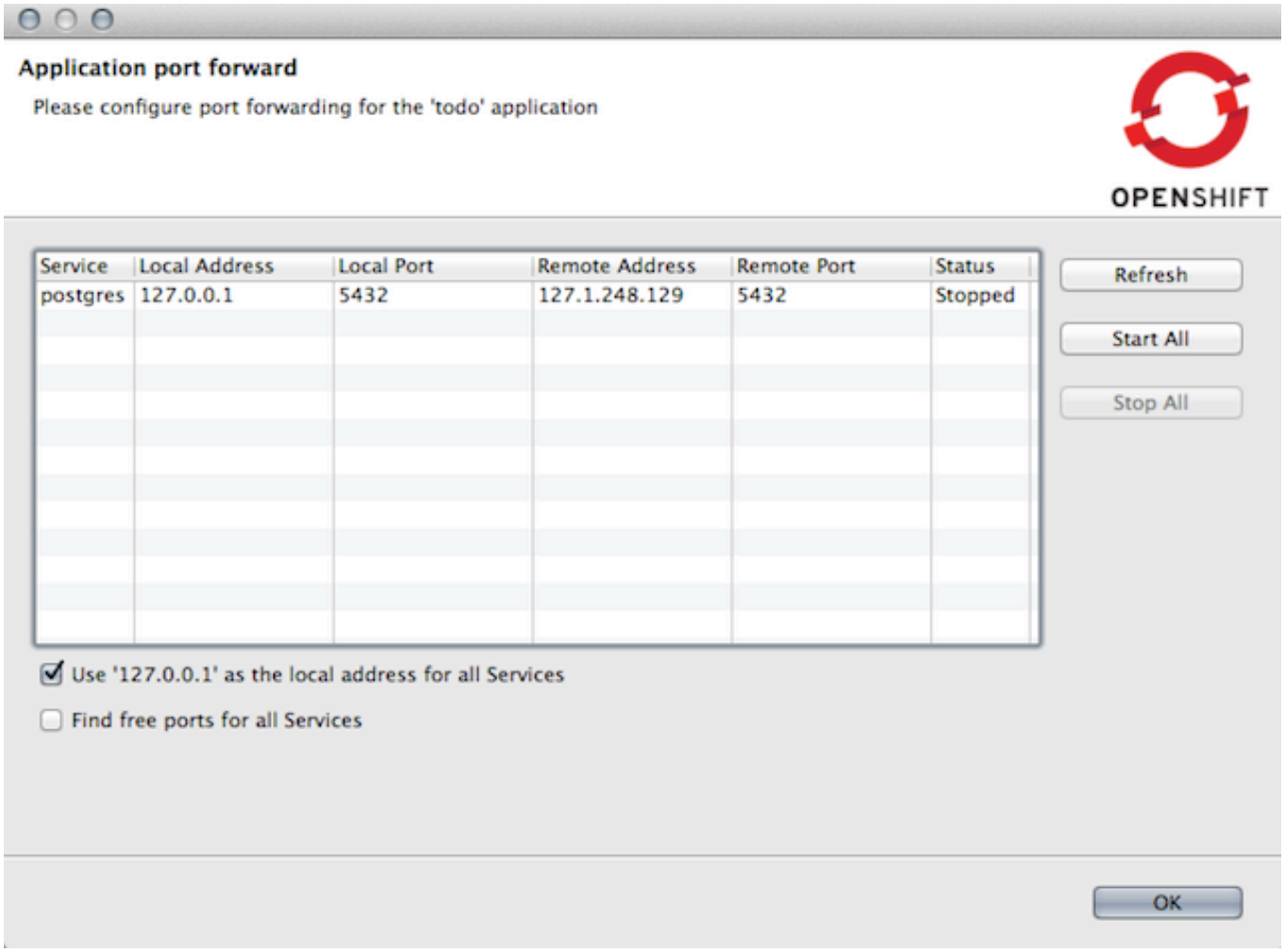
You will now be able to view the log files in the console tab that has been opened for you inside of Eclipse.

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.



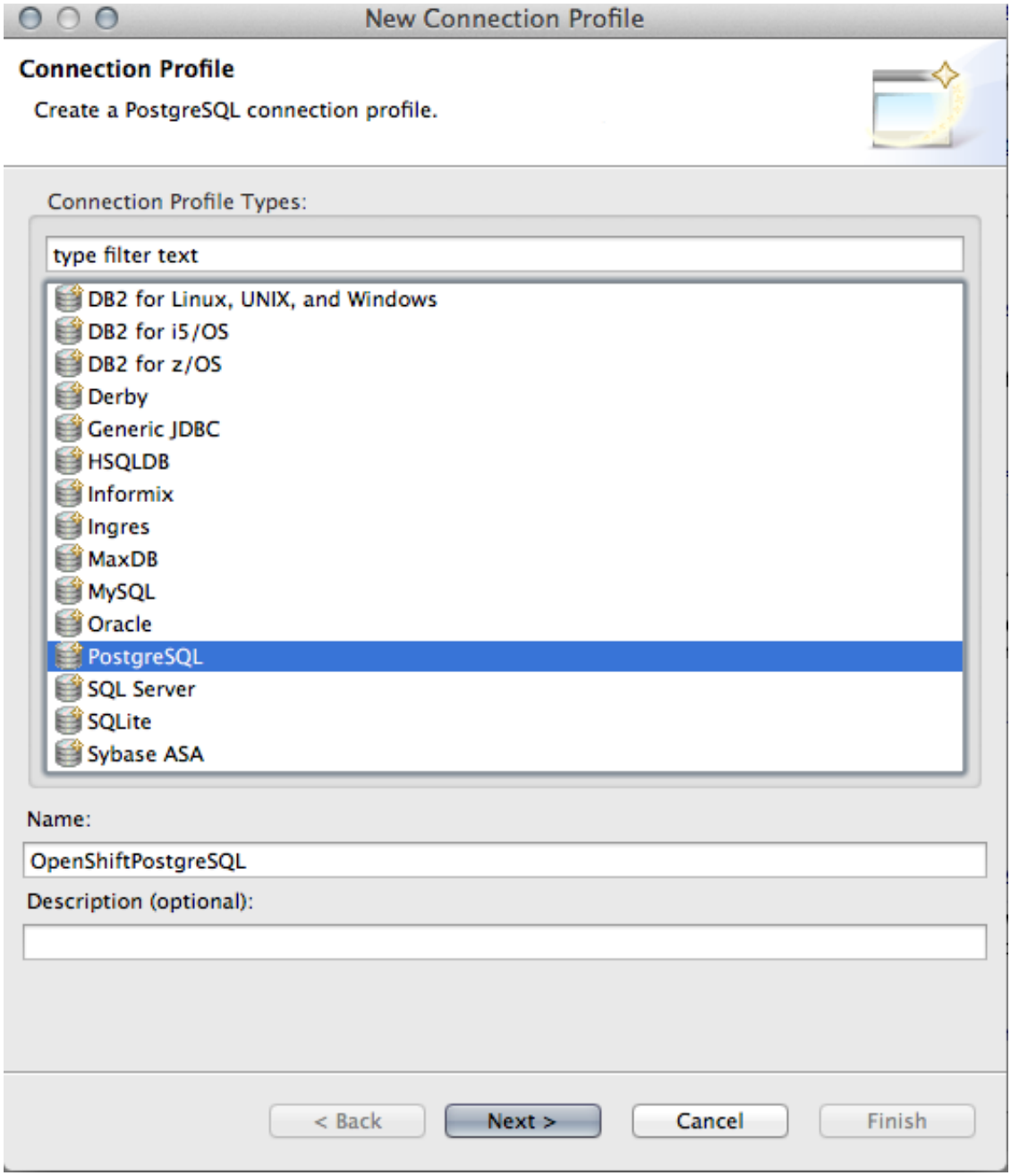
For the next section of this lab, ensure that you click on *Start Forwarding* so that we will be able to connect to PostgreSQL from our local machine.

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 are 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

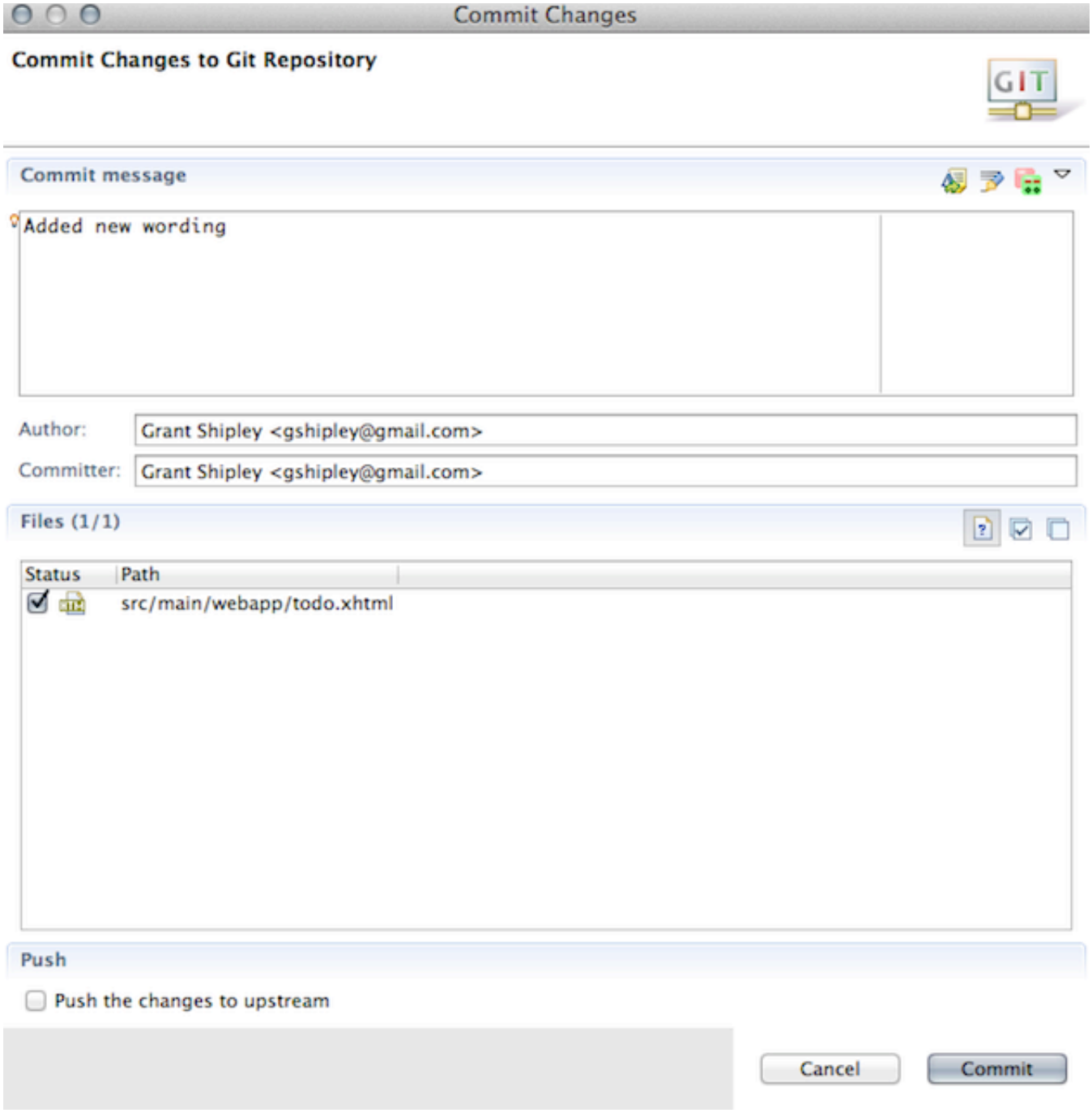
In the project view, expand the source files for the *src/main/webapp* directory and edit the *todo.xhtml* source file. Change the following line

```
<h2>Todo List Creation using Jenkins</h2>
```

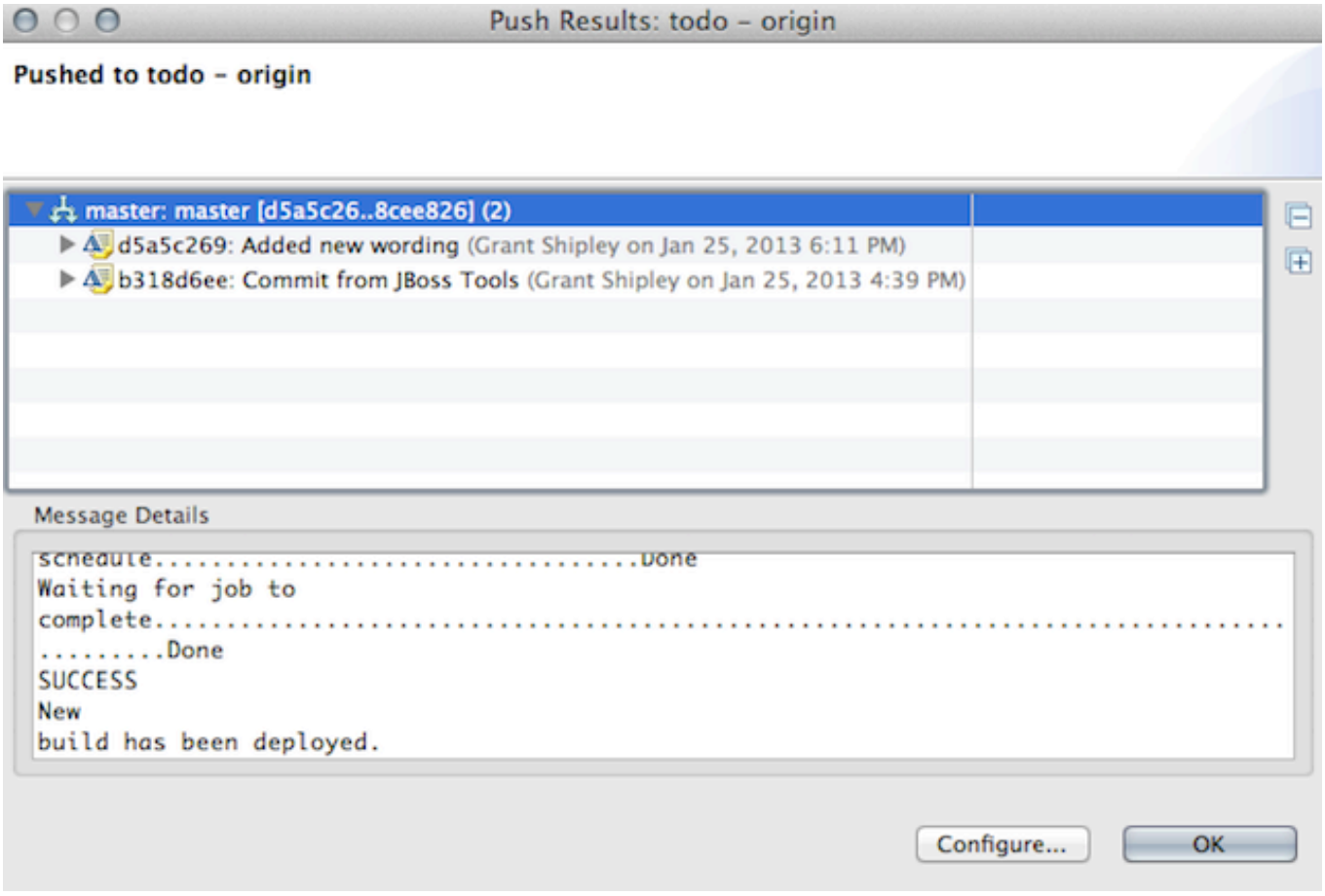
to the include JBoss Tools

```
<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.

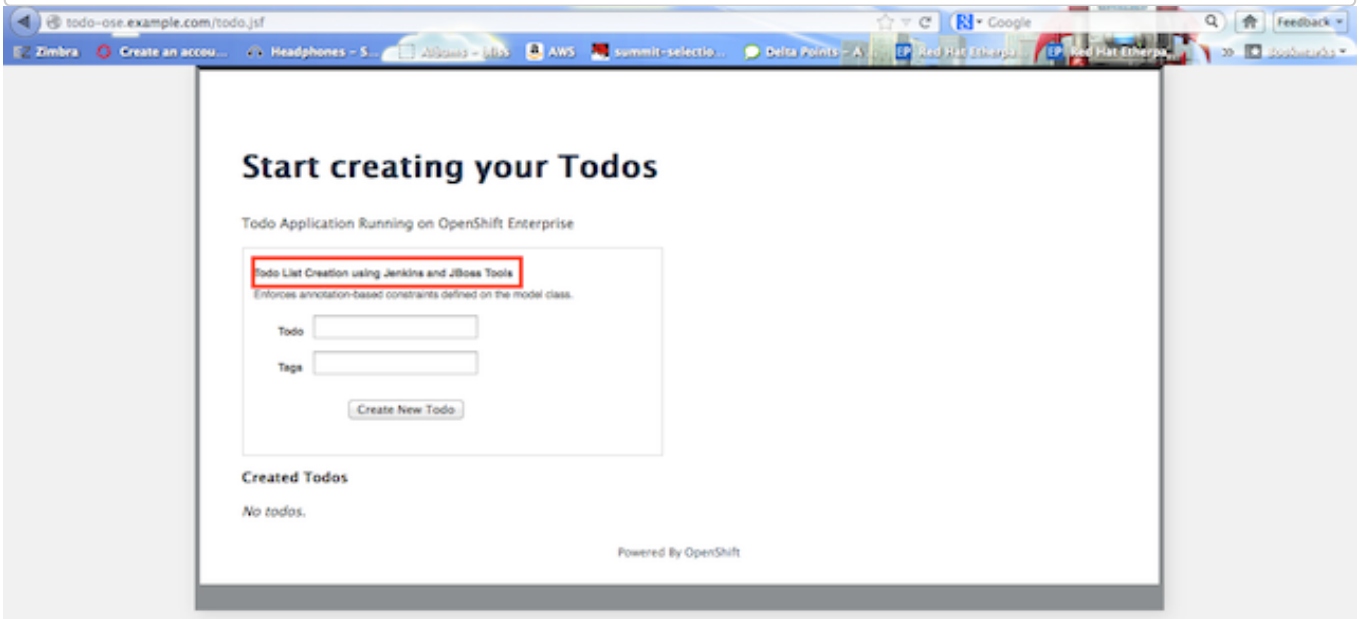


After you push your changes, open up your Jenkins dashboard and open the *Console Output* screen to see the build progress. Once your build has completed, Eclipse will display a dialog box with a summary of the deployment:



Verify that your changes were deployed correctly by opening up a web browser and going to the following URL:

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



Lab 20 Complete!

Lab 21: Using quickstarts

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 an 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 quick starts for are:

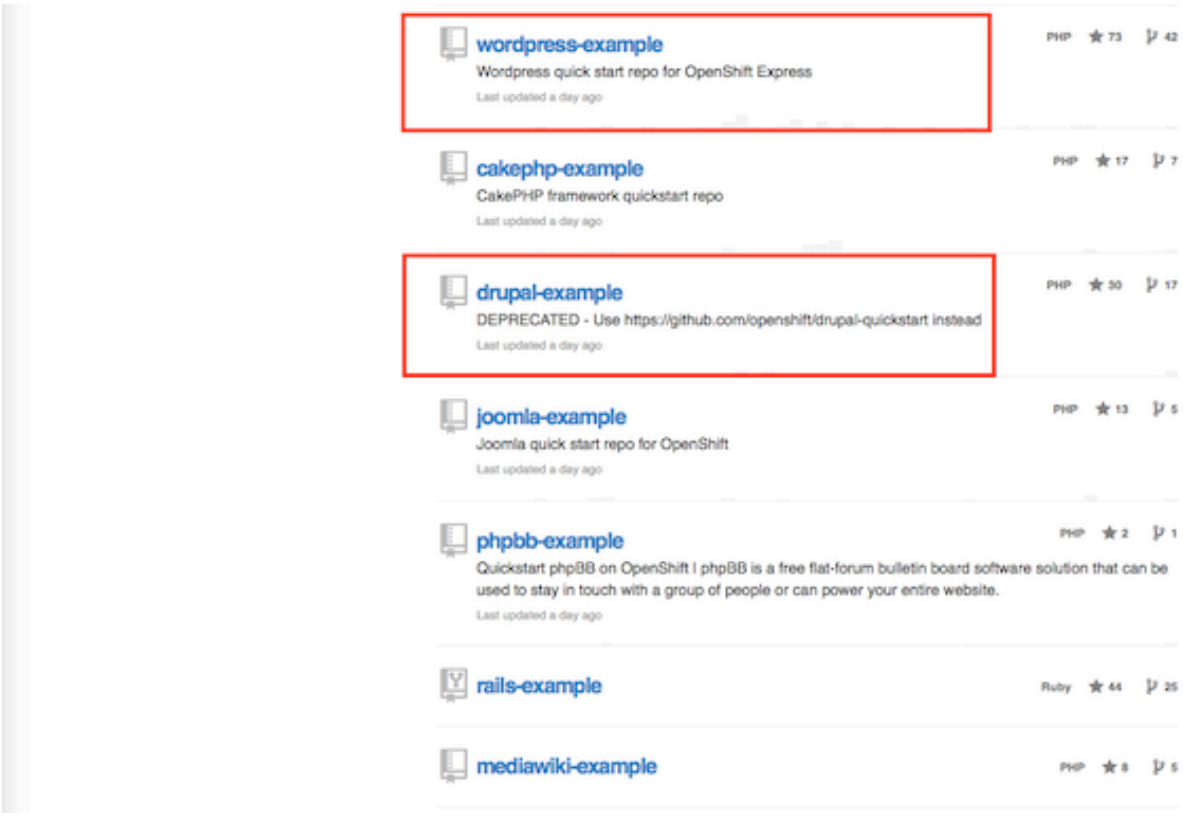
- Drupal
- Review Board
- Wordpress
- Frog CMS
- Sugar CRM
- Redmine
- MediaWiki

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.



Lab 21 Complete!

Lab 22: Creating a quick start

Server used:

- localhost
- node host

Tools used:

- rhc
- git
- Github

A common task that you will be asked to do is make a software developers 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 commands:

```
$ cd ~
$ mkdir code
$ cd code
$ mkdir piwikstage
$ cd 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 application:

```
$ cp –av ~/code/piwikstage/piwik/* ~/code/piwik/php
```

Now we need to add and commit our changes to our *piwik* application:

```
$ 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
```

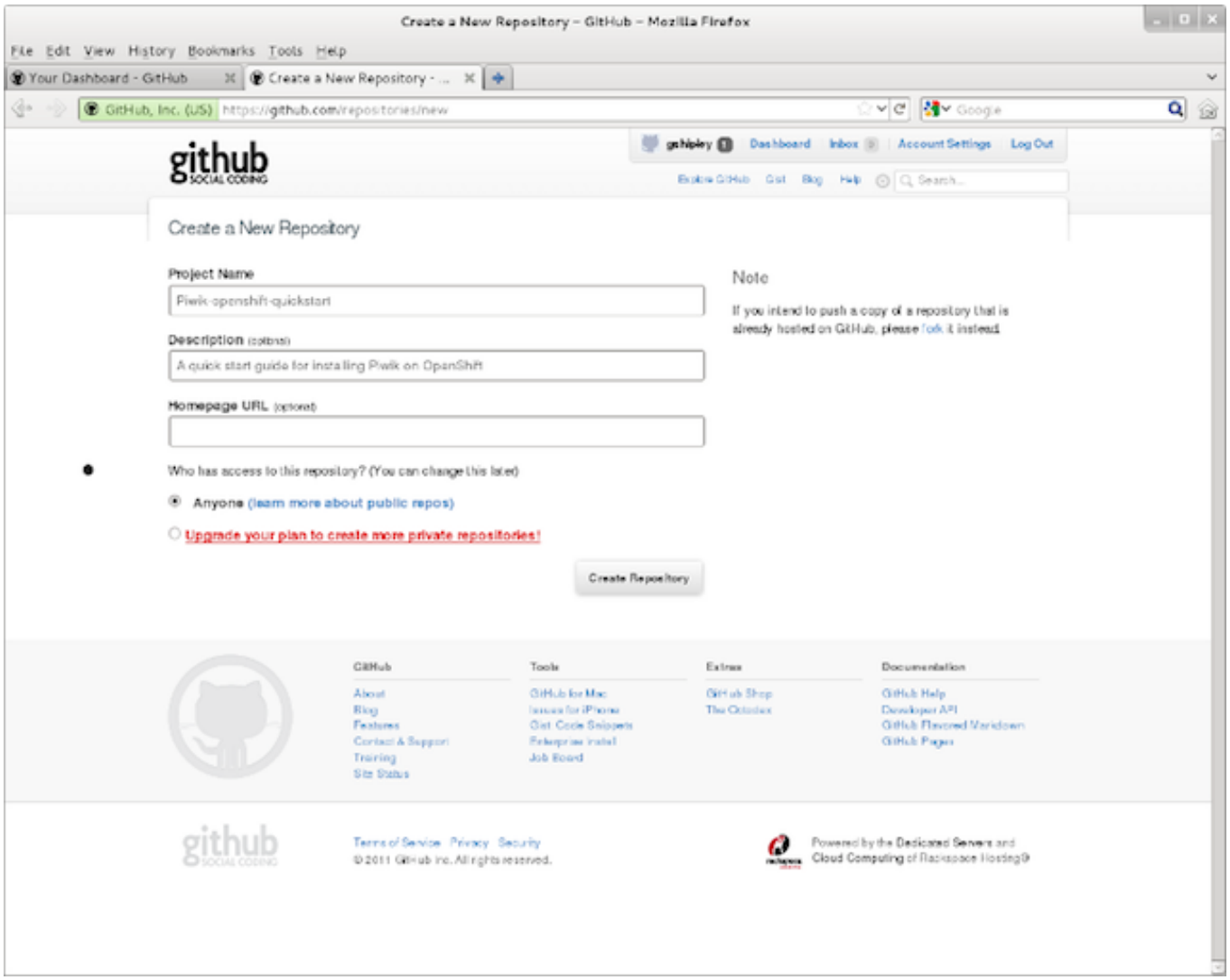
Creating a Github repository

Note: This step assumes that you already have a Github account. If you don't, head on over to www.github.com and sign up (it's free).

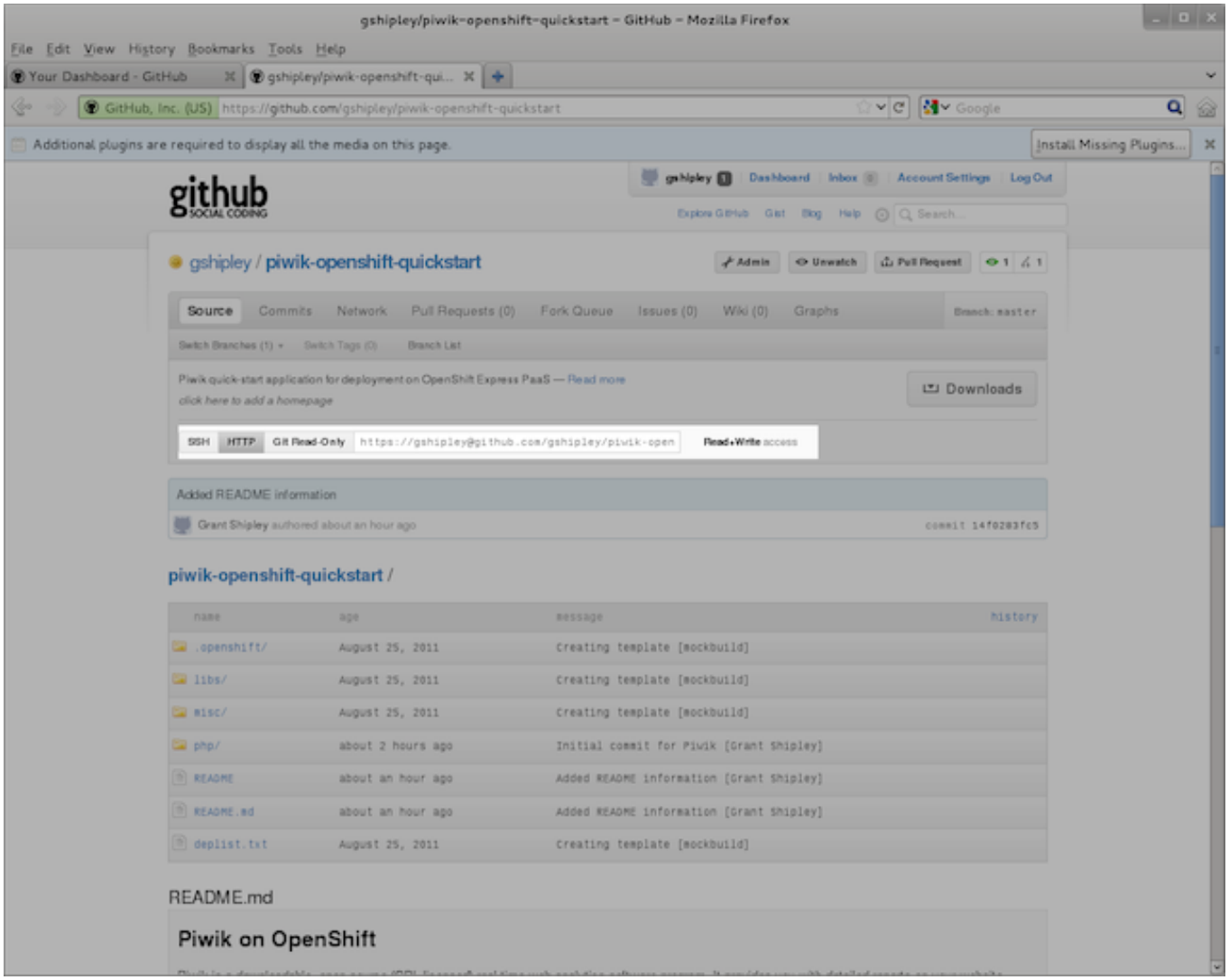
Log in to the Github website and create a new repository for our quick start. The direct link, after you are logged in, to create a new repository is:

```
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*.



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.




```
$ 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 exits, 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-opensh
        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 instructions 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 22 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 and 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 *oo-accept-broker* and *oo-accept-node* diagnostics commands, as appropriate for the type of 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 user's application as an administrator, 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 user's 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.