OPENSHIFT

Training Manual

OpenShift Enterprise 1.1

Installation, Configuration, Administration, & Usage

redhat.

Version 1.0
Author: Grant Shipley
gshipley@redhat.com

Contents

1.0 Overview of OpenShift Enterprise

1.1 Assumptions

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

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

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

1.2 What you can expect to learn from this training class

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

1.3 Overview of OpenShift Enterprise PaaS

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

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

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

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

1.4 Overview of IaaS

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

What does this mean? This means that in order to take advantage of OpenShift Enterprise, you can use any existing resources that you have in your hardware pool today. It doesn't matter if your infrastructure is based on EC2, VMware, RHEV, Rackspace, OpenStack, CloudStack, or even bare metal as we run on top of any Red Hat Enterprise Linux operating system as long as the architecture is x86_64.

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

1.5 Using the *kickstart* script

In this training class, we are going to go into the details of installing and configuring all of the components required for OpenShift Enterprise. We will be installing and configuring BIND, MongoDB, DHCP, ActiveMQ, MCollective, and other vital pieces to OpenShift. Doing this manually will give you a better understanding of how all of the components of OpenShift Enterprise work together to create a complete solution.

That being said, once you have a solid understanding of all of the moving pieces, you will probably want to take advantage of our kickstart script that performs all the functions in the administration portion of this training on your behalf. This script will allow you to create complete OpenShift Enterprise environments in a matter of minutes. It is not intended for you to use the kickstart as part of this training class.

The kickstart script is located at:
https://mirror.openshift.com/pub/enterprise-server/scripts/1.0/

When using the kickstart script, be sure to edit it to use the correct Red Hat subscriptions. Take a look at the script header for full instructions.

1.6 Electronic version of this document

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

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

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

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

Lab 1: Registering and updating the operating system (Estimated time: 10 minutes)

Server used:

- broker host

Tools used:

- SSH
- subscription-manager
- ntpdate
- yum

Registering the system and adding subscriptions

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

- Red Hat Enterprise Linux Employee Subscription
- OpenShift Enterprise Employee Subscription

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

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

From the list provided, subscribe to Red Hat Enterprise Linux.

```
# subscription-manager subscribe --pool [POOL IID from previous command]
```

Once you have subscribed to Red Hat Enterprise Linux, the next step is subscribe to the OpenShift Enterprise Employee subscription. Complete that step and then verify that you are subscribed to both RHEL and OpenShift Enterprise.

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

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

```
# yum repolist
```

Updating the operating system to the latest packages

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

```
# yum update
```

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

Configuring the clock to avoid clock skew

OpenShift Enterprise requires NTP to synchronize the system and hardware clocks. This synchronization is necessary for communication between the broker and node hosts; if the clocks are too far out of synchronization, MCollective will drop messages. Every MCollective request (discussed in a later lab) includes a time stamp, provided by the sending host's clock. If a sender's clock is substantially behind a recipient's clock, the recipient drops the message. This is often referred to as clock skew and is a common problem that users encounter when they fail to sync all of the system clocks.

```
# ntpdate clock.redhat.com
# chkconfig ntpd on
# service ntpd start
```

Lab 1 Complete!

Lab 2: Installing and configuring DNS (Estimated time: 20 minutes)

Server used:

- broker host

Tools used:

- SSH
- BIND
- text editor (vi, emacs, nano, etc.)
- environment variables
- SELinux
- Commands: cat, echo, chown, dnssec-keygen, rndc-confgen, restorecon, chmod, lokkit, chkconfig, service, nsupdate, ping, dig

Note: For this lab, use the 192.x.x.x IP address when setting up your nameserver

Installing the BIND DNS Server

In order for OpenShift Enterprise to work correctly, you will need to configure BIND so that you have a DNS server setup. At a typical customer site, they will have an existing DNS infrastructure in place. However, for the purpose of this training class, we need to install and configure our own server so that name resolution works properly. Primarily, we will be using name resolution for communication between our broker and node hosts as well as dynamically updating our DNS server to resolve gear application names when we start creating application gears.

This lab starts off by requiring the installation of both *bind* and *bind-utils* packages.

```
# yum install bind bind-utils
```

Creating environment variables and a DNSSEC key file

The official OpenShift documentation suggests that you set an environment variable for the domain name that you will be using to facilitate faster configuration of BIND. Let's follow the suggested route for this training class by issuing the following command:

```
# domain=example.com
```

DNSSEC, which stands for DNS Security Extensions, is a method by which DNS servers can verify that DNS data is coming from the correct place. You create a private/public key pair to determine to authenticity of the source domain name server. In order to implement DNSSEC on our new PaaS, we need to create a key file, which we will store in /var/named. For convenience, set the $keyfile variable now to the location of the this key file:

```
# keyfile=/var/named/${domain}.key
```

Create a DNSSEC key pair and store the private key in a variable named $key by using the following commands:

```
# cd /var/named
# dnssec-keygen -a HMAC-MD5 -b 512 -n USER -r /dev/urandom ${domain}
# KEY="$(grep Key: K${domain}*.private | cut -d ' ' -f 2)"
# cd -
# rndc-confgen -a -r /dev/urandom
```

Verify that the key was created properly by viewing the contents of the key variable:

```
# echo $KEY
```

Configure the ownership, permissions, and SELinux context for the key we created:

```
# restorecon -v /etc/rndc.* /etc/named.*
# chown -v root:named /etc/rndc.key
# chmod -v 640 /etc/rndc.key
```

Creating the *fowarders.conf* configuration file for host name resolution

The DNS forwarding facility of BIND can be used to create a large site-wide cache on a few servers, reducing traffic over links to external nameservers. It can also be used to allow queries by servers that do not have direct access to the Internet, but wish to look up exterior names anyway. Forwarding occurs only on those queries for which the server is not authoritative and does not have the answer in its cache.

Create a forwards.conf file with the following commands:

The forwarders.conf file is available on the lab support website

```
# echo "forwarders { 8.8.8.8; 8.8.4.4; } ;" >> /var/named/forwarders.conf
# restorecon -v /var/named/forwarders.conf
# chmod -v 755 /var/named/forwarders.conf
```

Configuring subdomain resolution and creating an initial DNS database

To ensure that we are starting with a clean */var/named/dynamic* directory, let's remove this directory if it exists:

```
# rm -rvf /var/named/dynamic
# mkdir -vp /var/named/dynamic
```

Issue the following command to create the *${domain}.db* file (before running this command, verify that the domain variable you set earlier in this lab is available to your current session):

The example.com.db file is available on the lab support website

```
cat <<EOF > /var/named/dynamic/${domain}.db
\$ORIGIN .
\$TTL 1 ; 1 seconds (for testing only)
${domain}      IN SOA  ns1.${domain}. hostmaster.${domain}. (
        2011112904 ; serial
        60        ; refresh (1 minute)
        15        ; retry (15 seconds)
        1800      ; expire (30 minutes)
        10        ; minimum (10 seconds)
        )
      NS  ns1.${domain}.
      MX  10 mail.${domain}.
\$ORIGIN ${domain}.
ns1       A   127.0.0.1
EOF
```

Once you have entered the above echo command, cat the contents of the file to ensure that the command was successful:

```
# cat /var/named/dynamic/${domain}.db
```

You should see the following output:

```
$ORIGIN .
$TTL 1  ; 1 second
example.com          IN SOA  ns1.example.com. hostmaster.example.com. (
                     2011112916 ; serial
                     60         ; refresh (1 minute)
                     15         ; retry (15 seconds)
                     1800       ; expire (30 minutes)
                     10         ; minimum (10 seconds)
                     )
                NS     ns1.example.com.
                MX     10 mail.example.com.
$ORIGIN example.com.
ns1                 A      127.0.0.1
```

Now we need to install the DNSSEC key for our domain:

```
cat <<EOF > /var/named/${domain}.key
key ${domain} {
 algorithm HMAC-MD5;
 secret "${KEY}";
};
EOF
```

Set the correct permissions and context:

```
# chown -Rv named:named /var/named
# restorecon -rv /var/named
```

Creating the *named* configuration file

We also need to create our *named.conf* file, Before running the following command, verify that the domain variable you set earlier in this lab is available to your current session.

The */etc/named.conf* file is available on the lab support website

```
cat <<EOF > /etc/named.conf
```

```
// named.conf
//
// Provided by Red Hat bind package to configure the ISC BIND named(8) DNS
// server as a caching only nameserver (as a localhost DNS resolver only).
//
// See /usr/share/doc/bind*/sample/ for example named configuration files.
//

options {
   listen-on port 53 { any; };
   directory   "/var/named";
   dump-file   "/var/named/data/cache_dump.db";
      statistics-file "/var/named/data/named_stats.txt";
      memstatistics-file "/var/named/data/named_mem_stats.txt";
   allow-query     { any; };
   recursion yes;

   /* Path to ISC DLV key */
   bindkeys-file "/etc/named.iscdlv.key";

   // set forwarding to the next nearest server (from DHCP response
   forward only;
      include "forwarders.conf";
};

logging {
      channel default_debug {
            file "data/named.run";
            severity dynamic;
      };
};

// use the default rndc key
include "/etc/rndc.key";

controls {
```

```
    inet 127.0.0.1 port 953
    allow { 127.0.0.1; } keys { "rndc-key"; };
};

include "/etc/named.rfc1912.zones";

include "${domain}.key";

zone "${domain}" IN {
    type master;
    file "dynamic/${domain}.db";
    allow-update { key ${domain} ; } ;
};
EOF
```

And finally, set the permissions for the new configuration file that we just created:

```
# chown -v root:named /etc/named.conf
# restorecon /etc/named.conf
```

Configuring host name resolution to use new the *BIND* server

We need to update our resolv.conf file to use our local *named* service that we just installed and configured. Open up your */etc/resolv.conf* file and add the following entry **as the first nameserver entry in the file**:

```
nameserver 127.0.0.1
```

We also need to make sure that *named* starts on boot and that the firewall is configured to pass through DNS traffic:

```
# lokkit --service=dns
# chkconfig named on
```

Starting the *named* service

We are finally ready to start up our new DNS server and add some updates.

```
# service named start
```

You should see a confirmation message that the service was started correctly. If you do not see an OK message, I would suggest running through the above steps again and ensuring that the output of each command matches the contents of this exercise. If you are still having trouble after trying the steps again, ask the instructor for help.

Adding entries using *nsupdate*

Now that our BIND server is configured and started, we need to add a record for our broker node to BIND's database. To accomplish this task, we will use the nsupdate command, which opens an interactive shell where we can perform commands, **using the 10.x.x.x address provided to you**:

```
# nsupdate -k ${keyfile}
> server 127.0.0.1
> update delete broker.example.com A
> update add broker.example.com 180 A ${your system ip address}
> send
```

Press control-D to exit from the interactive session.

In order to verify that you have successfully added broker.example.com to your DNS server, you can perform

```
# ping broker.example.com
```

and it should resolve to the local machine that you are working on. You can also perform a dig request using the following command:

```
# dig @127.0.0.1 broker.example.com
```

Lab 2 Complete!

Lab 3: Configuring the DHCP client and hostname (Estimated time: 5 minutes)

Server used:

- broker host

Tools used:

- text editor
- Commands: hostname

Creating *dhclient-eth0.conf*

In order to configure your broker host to use a DNS server that we installed in a previous lab, you will need to edit the */etc/dhcp/dhclient-{$network device}.conf file* or create the file if it does not exist. Without this step, the DNS server information in */etc/resolv.conf* would default back the server returned from your DHCP server on the next boot of the server. For example, if you are using eth0 as your default ethernet device, you would need to edit the following file:

```
/etc/dhcp/dhclient-eth0.conf
```

If you are unsure of which network device that your system is using, you can issue the *ifconfig* command to list all available network devices for your machine. Note, the *lo* device is the loopback device and is not the one you are looking for.

Once you have the correct file opened, add the following information making sure to substitute the correct IP Address in place of 10.10.10.10

```
prepend domain-name-servers 10.10.10.10;
supersede host-name "broker";
supersede domain-name "example.com";
```

Set the host name for your server

You need to set the hostname of your broker host. In order to accomplish this task, edit the */etc/sysconfig/network* file and locate the section labeled *HOSTNAME*. The line that you want to replace should look like this:

```
HOSTNAME=localhost.localdomain
```

We need to change this to reflect the new hostname that we are going to apply for this server. For this lab, we will be using broker.example.com. Change the */etc/sysconfig/network* file to reflect the following change:

```
HOSTNAME=broker.example.com
```

Now that we have configured our hostname, we also need to set it for our current session by using the following command:

```
# hostname broker.example.com
```

Lab 3 Complete!

Lab 4: Installing and configuring MongoDB (Estimated time: 10 minutes)

Server used:

- broker host

Tools used:

- text editor
- yum
- mongo
- chkconfig
- service

OpenShift Enterprise makes heavy use of MongoDB for storing internal information about users, gears, and other necessary items. If you are not familiar with MongoDB, I suggest that you head over the official MongoDB site (http://www.mongodb.org) to read up on this great NoSQL database. For the purpose of this lab, you need to know that MongoDB is a document data storage system and uses JavaScript for the command syntax and stores all documents in a JSON format.

Install *mongod* server

In order to use MongoDB, we will need to install the mongod server. To accomplish these tasks on Red Hat Enterprise Linux with an OpenShift Enterprise subscription, simply issue the following command:

```
# yum install mongodb-server
```

At the time of this writing, you should see the following packages being installed:

Packages installed from mongodb-server

Package Name	Arch	Package Version	Repo	Size
mongodb-server	x86_64	2.0.2–2.el6op	rhel-server-ose-infra–6-rpms	3.8 M
boost-program-options	x86_64	1.41.0–11.el6_1.2	rhel–6-server-rpms	105 k
boost-thread	x86_64	1.41.0–11.el6_1.2	rhel–6-server-rpms	105 k
libmongodb	x86_64	1.41.0–11.el6_1.2	rhel–6-server-rpms	41 k
boost-program-options	x86_64	2.0.2–2.el6op	rhel-server-ose-infra–6-rpms	531 k
mongodb	x86_64	2.0.2–2.el6op	rhel-server-ose-infra–6-rpms	21 M

Configuring *mongod*

MongoDB uses a configuration file for its settings. This file can be found at */etc/mongodb.conf*. We need to make a few changes to this file to ensure that we handle authentication correctly and that we enable the ability to use small files. Go ahead and edit the configuration file and ensure the two following conditions are set correctly:

The */etc/mongodb.conf* file is available on the lab support website

```
auth=true
```

By default, this line is commented out so just remove the hash mark *(#)*at the beginning of the line to enable the setting. We also need to enable smallfiles, so add the following line:

```
smallfiles=true
```

Setting *smallfiles=true* configures MongoDB not to pre-allocate a huge database, which wastes a surprising amount of time and disk space and is unnecessary for the comparatively small amount of data that the broker will store in it. It wouldn't hurt anything not to set *smallfiles=true*, except that it would take a minute or two to initialize a larger database and waste about half a gigabyte of disk space.

Configuring *mongod* to start on boot

MongoDB is an essential part of the OpenShift Enterprise platform. Because of this, we need to ensure that mongod is configured to start on system boot:

```
# chkconfig mongod on
```

By default, when you install *mongod* via the yum command, the service is not started. You can verify this with the following:

```
# service mongod status
```

This should return - *mongod is stopped*. In order to start the service, simply issue

```
# service mongod start
```

We need to verify that mongod was installed and configured correctly. In order to do this, we are going to make use of the mongo shell client tool. If you are more familiar with MySQL or Postgres, this is similar to the mysql client where you are dropped into an interactive SQL shell. Remember, MongoDB is a NoSQL database, so the notion of entering SQL commands is nonexistent. In order to start the mongo shell, enter the following command:

```
# mongo
```

You should see a confirmation message that you are using MongoDB shell version: 2.0.2 and that you are connecting to the test database. To verify even further, let's list all of the available databases that we currently have.

```
> show dbs
```

You will then be presented with a list of valid databases that are currently available to the mongod service.

```
admin   (empty)
local   (empty)
```

Lab 4 Complete!

Lab 5: Installing and configuring ActiveMQ (Estimated time: 10 minutes)

Server used:

- broker host

Tools used:

- text editor
- yum
- wget
- lokkit
- chkconfig
- service

ActiveMQ is a fully open source messenger service that is available for use across many different programming languages and environments. OpenShift Enterprise makes use of this technology to handle communications between the broker host and the node host in our deployment. In order to make use of this messaging service, we need to install and configure ActiveMQ for use on our broker node.

Installing ActiveMQ

Installing ActiveMQ on Red Hat Enterprise Linux 6 is a fairly easy and straightforward process as the packages are included in the rpm repositories that are already configured on your broker node. We want to install both the server and client packages by using the following command:

```
# yum install activemq activemq-client
```

You will notice that this will also install any of the dependencies required for the packages if you don't already have them. Notably, Java 1.6 and the libraries for use with the Ruby programming language.

Configuring ActiveMQ

ActiveMQ uses an XML configuration file that is located at */etc/activemq/activemq.xml*.

```
# cd /etc/activemq
# mv activemq.xml activemq.orig
```

The above command will backup the default configuration file that ships with ActiveMQ and replace it with one configured for use with OpenShift Enterprise. Now that we have the configuration template, we need to make a few minor changes to the configuration.

The first change we need to make is to replace the hostname provided (activemq.example.com) to the FQDN of your broker host. For example, the following line:

The activemq.xml file is available on the lab support website

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="activemq.example.com"
dataDirectory="${activemq.data}">
```

Should become:

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="broker.example.com"
dataDirectory="${activemq.data}">
```

Note: The *${activemq.data}* text should be entered as stated as it does not refer to a shell variable

The second change is to provide your own credentials for authentication. The authentication information is stored inside of the block of code. Make the changes that you desire to the following code block:

```
<simpleAuthenticationPlugin>
  <users>
   <authenticationUser username="mcollective" password="marionette"
groups="mcollective,everyone"/>
   <authenticationUser username="admin" password="secret" groups="mcollective,admin,everyone"/>
  </users>
 </simpleAuthenticationPlugin>
```

Updating the firewall rules and configuring ActiveMQ to start on boot

We need to modify the firewall rules to allow MCollective to communicate on port 61613.

```
# lokkit --port=61613:tcp
```

Finally, we need to enable the ActiveMQ service to start on boot as well as start the service for the first time.

```
# chkconfig activemq on
# service activemq start
```

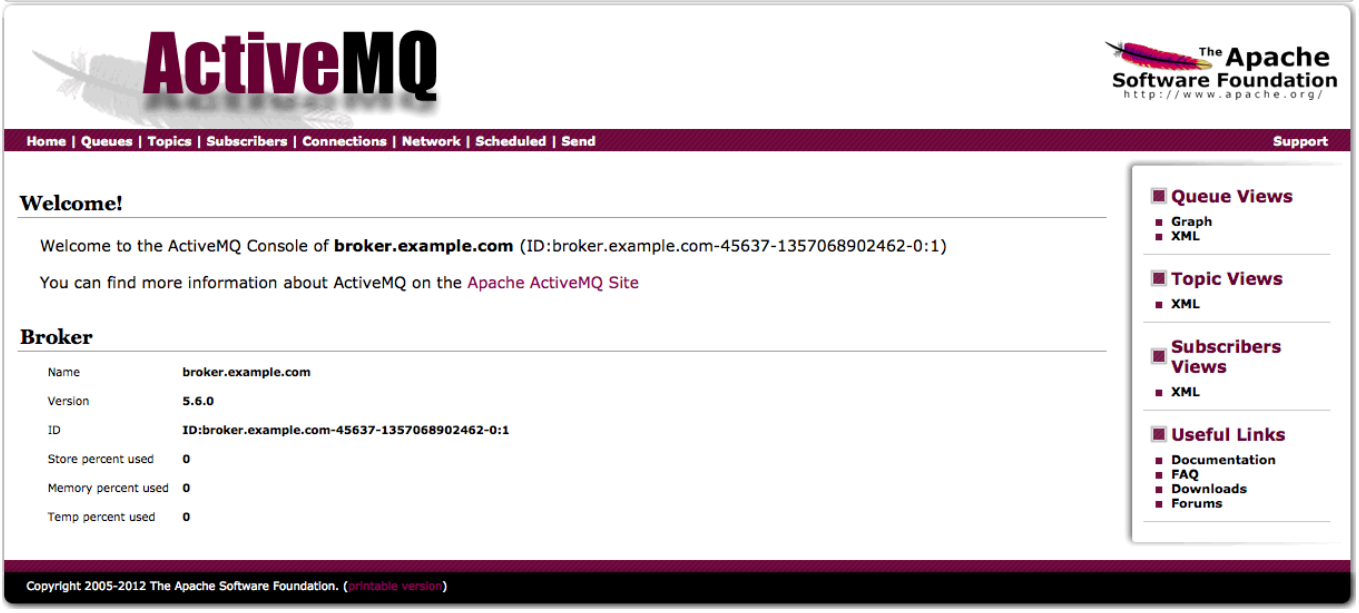
Verifying ActiveMQ is working

Now that ActiveMQ has been installed, configured, and started, let's verify that the web console is working as expected. ActiveMQ web console should be running and listening on port 8161. In order to verify that everything worked correctly, load the following URL in a web browser:

```
http://localhost:8161
```

Note: Given the current configuration, ActiveMQ is only available on the localhost. If you want to be able to connect to it via HTTP remotely, you will need to either enable a SSH port forwarding tunnel or you will need to add a rule to your firewall configuration:

```
# lokkit --port=8161:tcp
# ssh -f -N -L 8161:broker.example.com:8161 root@10.10.10.10
```



Note: While we changed the authentication credentials for the ActiveMQ service itself, the above configuration requires no authentication for accessing the activemq console, which can still be accessed via a Web interface with default the credentials. For a production deployment, you would want to restrict access to localhost (127.0.0.1) and require authentication. The authentication information is stored in the */etc/activemq/jetty.xml* configuration file as well as the */etc/activemq/jetty-realm.properties* file.

Lab 5 Complete!

Lab 6: Installing and configuring the MCollective client (Estimated time: 10 minutes)

Server used:

- broker host

Tools used:

- text editor
- yum

For communication between the broker host and the gear nodes, OpenShift Enterprise uses MCollective. You may be wondering how MCollective is different from ActiveMQ, that we installed in a previous lab. ActiveMQ is the messenger server that provides a queue of transport messages. You can think of MCollective as the client that actually sends and receives those messages. For example, if we want to create a new gear on an OpenShift Enterprise node, MCollective would receive the create gear message from ActiveMQ and perform the operation.

Installing the MCollective client

In order to use MCollective, we need to install and configure it.

```
# yum install mcollective-client
```

Configuring the MCollective client

Replace the contents of the */etc/mcollective/client.cfg* with the following information:

The */etc/mcollective/client.cfg* file is available on the lab support website

```
topicprefix = /topic/
main_collective = mcollective
collectives = mcollective
libdir = /usr/libexec/mcollective
logfile = /var/log/mcollective-client.log
loglevel = debug

# Plugins
securityprovider = psk
plugin.psk = unset

connector = stomp
plugin.stomp.host = localhost
plugin.stomp.port = 61613
plugin.stomp.user = mcollective
plugin.stomp.password = marionette
```

Here, we have configured the MCollective client to connect to ActiveMQ running on the local host. In a typical deployment, you will configure MCollective to connect to ActiveMQ running on a remote server by putting the appropriate hostname for the plugin.stomp.host setting.

Lab 6 Complete!

Lab 7: Installing and configuring the broker application (Estimated time: 15 minutes)

Server used:

- broker host

Tools used:

- text editor
- yum
- sed
- chkconfig
- lokkit
- openssl
- ssh-keygen
- fixfiles
- restorecon

Installing necessary packages for the broker application

In order for users to interact with the OpenShift Enterprise platform, they will typically use client tools or the web console. These tools communicate with the broker via a REST API that is also accessible for writing third party applications and tools. In order to use the broker application, we need to install several packages from the OpenShift Enterprise repository.

```
# yum install openshift-origin-broker openshift-origin-broker-util rubygem-openshift-origin-auth-remote-user rubygem-openshift-origin-msg-broker-mcollective rubygem-openshift-origin-dns-bind
```

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

Modifying the broker proxy server name

The default value of the ServerName property is localhost, and you need to change this to accurately reflect your broker's host name. Run the following command to update your broker's host name using sed:

```
# sed -i -e "s/ServerName .*$/ServerName `hostname`/" /etc/httpd/conf.d/000000_openshift_origin_broker_proxy.conf
```

Note: You can also manually update the */etc/httpd/conf.d/000000_openshift_origin_broker_proxy.conf* and modify the ServerName attribute to reflect the correct hostname.

Configuring the firewall and setting service to start on boot

The broker application requires a number of services to be running in order to function properly. Instead of having to start these services each time the server boots, we can add them to startup at boot time.

```
# chkconfig httpd on
# chkconfig network on
# chkconfig ntpd on
# chkconfig sshd on
```

We also need to modify the firewall rules to ensure that the traffic for these services is accepted:

```
# lokkit --service=ssh
# lokkit --service=https
# lokkit --service=http
```

Generating access keys

We now need to generate access keys that will allow some of the services, Jenkins for example, to communicate to the broker.

```
# openssl genrsa -out /etc/openshift/server_priv.pem 2048
# openssl rsa -in /etc/openshift/server_priv.pem -pubout > /etc/openshift/server_pub.pem
```

We also need to generate a ssh key pair that will allow communication between the broker host and any nodes that you have configured. Remember, the broker host is the director of communications and the node hosts actually contain all of the application gears that your users create. In order to generate this SSH keypair, perform the following commands:

```
# ssh-keygen -t rsa -b 2048 -f ~/.ssh/rsync_id_rsa
```

Just press enter for the password.

```
# cp ~/.ssh/rsync_id_rsa* /etc/openshift/
```

In a later lab that covers configuration of the node hosts, we will copy this newly created key to each node host.

Configuring SELinux Boolean variables and setting file contexts

SELinux has several variables that we want to ensure are set correctly. These variables include the following:

SELinux Boolean Values

Variable Name	Description
httpd_unified	Allow the broker to write files in the “http” file context
httpd_can_network_connect	Allow the broker application to access the network
httpd_can_network_relay	Allow the SSL termination Apache instance to access the backend Broker application
httpd_run_stickshift	Enable passenger-related permissions
named_write_master_zones	Allow the broker application to configure DNS
allow_ypbind	Allow the broker application to use ypbind to communicate directly with the name server

In order to set all of these variables correctly, enter the following:

```
# setsebool -P httpd_unified=on httpd_can_network_connect=on httpd_can_network_relay=on
httpd_run_stickshift=on named_write_master_zones=on allow_ypbind=on
```

We also need to set several files and directories with the proper SELinux contexts. Issue the following commands:

```
# fixfiles -R rubygem-passenger restore
# fixfiles -R mod_passenger restore
# restorecon -rv /var/run
# restorecon -rv /usr/share/rubygems/gems/passenger-*
```

Understanding and changing the broker configuration

The OpenShift Enterprise broker uses a configuration file to define several of the attributes for controlling how the platform as a service works. This configuration file is located at */etc/openshift/broker.conf*. For instance, the valid gear types that a user can create are defined using the *VALID_GEAR_SIZES* variable.

```
# Comma separated list of valid gear sizes
VALID_GEAR_SIZES="small,medium"
```

Edit this file and ensure that the *CLOUD_DOMAIN* variable is set to correctly reflect the domain that you are using to configure this deployment of OpenShift Enterprise.

```
# Domain suffix to use for applications (Must match node config)
CLOUD_DOMAIN="example.com"
```

While you are in this file, you can change any other settings that need to be configured for your specific installation.

Lab 7 Complete!

Lab 8: Configuring the broker plugins and MongoDB user accounts (Estimated time: 15 minutes)

Server used:

- broker host

Tools used:

- text editor
- cat
- echo
- environment variables
- pushd
- semodule
- htpasswd
- mongo
- bundler
- chkconfig
- service

OpenShift Enterprise uses a plugin system for core system components such as DNS, authentication, and messaging. In order to make use of these plugins, we need to configure them and provide the correct configuration items to ensure that they work correctly. The plugin configuration files are located in the */etc/openshift/plugins.d* directory. We will be working with several of these files so it is suggested that you change to that directory to complete the steps in this lab.

```
# cd /etc/openshift/plugins.d
```

Once you are in this directory, you will see that OpenShift Enterprise provides several example configuration files for you to use to speed up the process of configuring these plugins. You should see three example files.

- openshift-origin-auth-remote-user.conf.example
- openshift-origin-dns-bind.conf.example
- openshift-origin-msg-broker-mcollective.conf.example

Creating configuration files from examples

Let's begin by copying the .example files to actual configuration files that will be used by OpenShift Enterprise.

```
# cp openshift-origin-auth-remote-user.conf.example openshift-origin-auth-remote-user.conf
# cp openshift-origin-msg-broker-mcollective.conf.example openshift-origin-msg-broker-mcollective.-conf
```

The broker application will check the plugins.d directory for files ending in .conf. The presence of .conf file enables the corresponding plug-in. Thus, for example, copying the openshift-origin-auth-remote-user.conf.example file to openshift-origin-auth-remote-user.conf enables the auth-remote-user plug-in.

Configuring the DNS plugin

Instead of copying the example DNS configuration file, we are going to create a new one by issuing an echo command. We are doing this to take advantage of the $domain and $keyfile environment variables that we created in a previous lab. If you are no longer have these variables set, you can recreate them with the following commands:

```
# domain=example.com
# keyfile=/var/named/${domain}.key
# cd /var/named
# KEY="$(grep Key: K${domain}*.private | cut -d ' ' -f 2)"
```

To verify that your variables were recreated correctly, echo the contents of your keyfile and verify your $KEY variable is set correctly:

```
# cat $keyfile
# echo $KEY
```

If you performed the above steps correctly, you should see output similar to this:

```
key example.com {
  algorithm HMAC-MD5;
  secret "3RH8tLp6fvX4RVV9ny2lmotZpTjXhB62ieC6CN1Fh/2468Z1+6lX4wpCJ6sfYH6u2+//gbDDSt-DX+aPMtSiNFw==";
};
```

and

```
3RH8tLp6fvX4RVV9ny2lmotZpTjXhB62ieC6CN1Fh/2468Z1+6lX4wpCJ6sfYH6u2+//gbDDStDX+aPMt
SiNFw==
```

Now that we have our variables setup correctly, we can create our *openshift-origin-dns-bind.conf* file. **Ensure that you are still in the */etc/openshift/plugins.d* directory** and issue the following command:

```
# cd /etc/openshift/plugins.d
# cat << EOF > openshift-origin-dns-bind.conf
BIND_SERVER="127.0.0.1"
BIND_PORT=53
BIND_KEYNAME="${domain}"
BIND_KEYVALUE="${KEY}"
BIND_ZONE="${domain}"
EOF
```

After running this command, cat the contents of the file and ensure they look similar to the following:

```
BIND_SERVER="127.0.0.1"
BIND_PORT=53
BIND_KEYNAME="example.com"
BIND_KEYVALUE="3RH8tLp6fvX4RVV9ny2lmotZpTjXhB62ieC6CN1Fh/2468Z1+6lX4wpCJ6sfY-
H6u2+//gbDDStDX+aPMtSiNFw=="
BIND_ZONE="example.com"
```

The last step to configure DNS is to compile and install the SELinux policy for the plugin.

```
# pushd /usr/share/selinux/packages/rubygem-openshift-origin-dns-bind/ && make -f
/usr/share/selinux/devel/Makefile ; popd
# semodule -i /usr/share/selinux/packages/rubygem-openshift-origin-dns-bind/dhcpnamedforward.pp
```

Configuring an authentication plugin

OpenShift Enterprise supports various different authentication systems for authorizing a user. In a production environment, you will probably want to use LDAP, kerberos, or some other enterprise class authorization and authentication system. However, for this lab we will use a system called Basic Auth which relies on a *htpasswd* file to configure authentication. OpenShift Enterprise provides three example authentication configuration files in the */var/www/openshift/broker/httpd/conf.d/* directory.

Authentication Sample Files

Authentication Type	Description
Basic Auth	openshift-origin-auth-remote-user-basic.conf.sample
Kerberos	openshift-origin-auth-remote-user-kerberos.conf.sample
LDAP	openshift-origin-auth-remote-user-ldap.conf.sample

Since we will be using Basic Auth, we need to copy the sample configuration file to the actual configuration file:

```
# cp /var/www/openshift/broker/httpd/conf.d/openshift-origin-auth-remote-user-basic.conf.sample
/var/www/openshift/broker/httpd/conf.d/openshift-origin-auth-remote-user.conf
```

This configuration file specifies that the *AuthUserFile* is located at */etc/openshift/htpasswd*. At this point, that file doesn't exist, so we need to create it and add a user named *demo*.

```
# htpasswd -c /etc/openshift/htpasswd demo
```

Note: The -c option to htpasswd creates a new file, overwriting any existing htpasswd file. If your intention is to add a new user to an existing htpasswd file, simply drop the -c option.

After entering the above command, you will be prompted for a password for the user *demo*. Once you have provided that password, view the contents of the htpasswd file to ensure that the user was added correctly. Make a note of the password as we will using it during later labs.

```
# cat /etc/openshift/htpasswd
```

If the operation was a success, you should see output similar to the following:

```
demo:$apr1$Q7yO3MF7$rmSZ7SI.vITfEiLtkKSMZ/
```

Adding a MongoDB account

As previously discussed in this training class, OpenShift Enterprise makes heavy use of the MongoDB NOSQL database. In a previous lab, we installed and configured MongoDB but now we need to add a user for the broker application. If you take a look at the broker configuration file

```
# cat /etc/openshift/broker.conf |grep MONGO
```

you will see that by default, the broker application is expecting a MongoDB user to be created called

openshift with a password of *mooo*. At this point, you can either create a user with those credentials or create a separate user. If you create a separate user, ensure that you modify the broker.conf file to reflect the correct credentials.

```
# mongo openshift_broker_dev --eval 'db.addUser("openshift", "mooo")'
```

Once you have entered the above command, you should see the following output:

```
MongoDB shell version: 2.0.2
connecting to: openshift_broker_dev
{ "n" : 0, "connectionId" : 2, "err" : null, "ok" : 1 }
{
   "user" : "openshift",
   "readOnly" : false,
   "pwd" : "8f5b96dbda3a3cd0120d6de44d8811a7",
   "_id" : ObjectId("50e4665e60ce1894d530e1f1")
}
```

Configuring Bundler

The broker rails application depends on several gem files in order to operate correctly. We need to ensure that bundler can find the appropriate gem files.

```
# cd /var/www/openshift/broker
# bundle --local
```

You should see a lot of information scroll by letting you know what gem files the system is actually using. The last line of output should be:

```
Your bundle is complete! Use `bundle show [gemname]` to see where a bundled gem is installed.
```

Setting services to start on boot

The last step in configuring our broker application is to ensure that all of our services are started and that they are configured to start upon system boot.

```
# chkconfig openshift-broker on
```

This will ensure that the broker starts upon next system boot. However, we also need to start the

broker application to run now.

```
# service httpd start
# service openshift-broker start
```

Lab 8 Complete!

Lab 9: Installing the OpenShift Enterprise Web Console (Estimated time: 10 minutes)

Server used:

- broker host

Tools used:

- text editor
- yum
- service
- chkconfig

In this lab we want to install the OpenShift Enterprise Web Console. The console is written in ruby and will provide a great way for users of the system to create and manage application gears that are deployed on the gear hosts.

Installing the Web Console RPMs

The installation of the web console can be performed with a simple *yum install* command but will pull in many dependencies from the ruby programming language. At the time of this writing, executing the following command installed 77 additional packages.

```
# yum install openshift-console
```

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

Configuring authentication for the console

In a previous lab, we configured the broker application for Basic Auth. When performing that lab, you actually configured authentication for the REST based API that the broker application provides. One of the great things about OpenShift Enterprise is that the console application uses a separate authenticate scheme for authenticating users to the web console. This will allow you to restrict which users you want to have access to the REST API and keep that authentication separate from the web based user console.

The openshift-console package that we installed previously in this lab created some sample

authentication files for us. These files are located in the */var/www/openshift/console/httpd/conf.d* directory. For this lab, we are going to use the same htpasswd file that we created when setting up the Broker Application authentication. In order to do this, simply issue the following commands:

```
# cd /var/www/openshift/console/httpd/conf.d
# cp openshift-origin-auth-remote-user-basic.conf.sample openshift-origin-auth-remote-user-basic.conf
```

Now that we have the openshift-console packages installed, we need to start the service and ensure it starts on boot.

```
# service openshift-console start
# chkconfig openshift-console on
```

Once completed, the console will now prompt the user to provide their login credentials as specified in the */etc/openshift/htpasswd* file.

Note: Seeing an error page after authenticating to the console is expected at this point. The web console will not be fully active until we add a node host in a later lab.

Lab 9 Complete!

Lab 10: Configuring DNS resolution for the node host (Estimated time: 20 minutes)

Servers used:

- Node host
- Broker host

Tools used:

- text editor
- yum
- ntpdate
- dig
- oo-register-dns
- cat
- scp
- ssh

Before proceeding with this lab, ensure that you are connected, via SSH, to your node host and subscribe to RHEL and OpenShift Enterprise repositories using *subscription-manager*.

Updating the operating system to the latest packages

During the training class, you were provided with credentials for two servers, a broker host and a node host. This lab begins with the configuration of your node / gear host.

Once you have successfully subscribed to the correct products, ensure that your operating system is updated to the latest packages.

```
# yum update
```

Configuring the clock to avoid clock skew

OpenShift Enterprise requires NTP to synchronize the system and hardware clocks. This synchronization is necessary for communication between the broker and node hosts; if the clocks are too far out of synchronization, MCollective will drop messages. Every MCollective request includes a time stamp, provided by the sending host's clock. If a sender's clock is substantially behind a

recipient's clock, the recipient drops the message. This is often referred to as clock skew as is a common problem that users encounter when they fail to sync all of the system clocks.

```
# ntpdate clock.redhat.com
# chkconfig ntpd on
# service ntpd start
```

Registering a DNS entry for the node host

SSH to your broker application host that we configured in the previous labs and set a variable that points to your keyfile. If you have been using example.com, as stated in this lab manual, the following command should work.

```
# keyfile=/var/named/example.com.key
```

If you did not use example.com, replace the above command with the correct location of your keyfile.

In order to configure your DNS to resolve your node host, we need to tell our BIND server about the host. Run the following command and **replace the IP address with the correct IP address of your node**.

```
# oo-register-dns -h node -d example.com -n 10.10.10.11 -k ${keyfile}
```

Now that we have added node.example.com to our DNS server, the broker application host should be able to resolve the node host by referring to it by name. Let's test this:

```
# dig @127.0.0.1 node.example.com
```

This should resolve to 10.10.10.11, or the IP address that you specified when you ran the command.

Configuring SSH key authentication between broker and node

While on the broker application host, we need to copy the SSH key that we previously created to the node. This will allow operations to work from inside of OpenShift Enterprise without requiring a password. Once you connect to the broker host, copy the key with the following command:

Note: Execute the following on the broker host.

```
# scp /etc/openshift/rsync_id_rsa.pub root@node.example.com:/root/.ssh
```

Once you enter that command, you will be prompted to authenticate to the node host.

At this point, we need to login to our node host to add the newly copied key to our authorized_keys. SSH into your node host and run the following:

Note: Execute the following on the node host.

```
# cat /root/.ssh/rsync_id_rsa.pub >> /root/.ssh/authorized_keys
# rm -f /root/.ssh/rsync_id_rsa.pub
```

Now that our key has been copied from our broker application host to our node host, let's verify that is copied correctly and was added to the authorized_keys file. Once you issue the following command, you should be authenticated to the node host without having to specify the root user password.

Note: Execute the following on the broker host.

```
# ssh -i /root/.ssh/rsync_id_rsa root@node.example.com
```

Configuring DNS resolution on the node

We need to configure the node host to use the BIND server that we have installed and configured on the broker application host. This is a fairly straight forward process by adding the IP address of the broker application host to our */etc/resolv.conf* on the node host. Edit this file and the following line making sure to use the correct IP address of your broker application server.

Note: Execute the following on the node host.

```
nameserver 10.10.10.10
```

Configuring the DHCP client and hostname

On the node host, we need to configure our settings to prepend the DNS server we created in a previous lab to our resolv.conf file on system boot. This will allow the node host to resolve references to broker.example.com to ensure that all pieces of OpenShift Enterprise can communicate with one another. This process is similar to setting up the *dhclient-eht0.conf* configuration file for the broker application.

Note: This step assumes that your node host is using the eth0 device for network connectivity. If that is not the case, replace eth0 with the correct Ethernet device for you host.

Edit the */etc/dhcp/dhclient-eth0.conf* file, or add it if it doesn't exist, and add the following information ensuring that you replace the IP address with the correct IP of your broker application host.

```
prepend domain-name-servers 10.10.10.10;
supersede host-name "node";
supersede domain-name "example.com";
```

We now need to set the hostname for node host to correctly reflect node.example.com. Edit the */etc/sysconfig/network* file and change the *HOSTNAME* entry to the following:

```
HOSTNAME=node.example.com
```

We also need to set the hostname for our current session but issuing the hostname command at the command prompt.

```
# hostname node.example.com
```

Verify that the hostname was set correctly by running the hostname command. If the hostname was set correctly, you should *node.example.com* as the output of the hostname command.

```
# hostname
```

Lab 10 Complete!

Lab 11: Setting up MCollective on the node host (Estimated time: 10 minutes)

Server used:

- node host

Tools used:

- text editor
- yum
- chkconfig
- service
- mco ping

If you recall, MCollective is the tool that OpenShift Enterprise uses to send and receive messages via the ActiveMQ messaging server. In order for the node host, the client, to send and receive messages with the broker application, we need to install and configure MCollective on the node host to communicate with the broker application.

Installing MCollective on the node host

In order to install MCollective on the node host, we will need to install the *openshift-origin-msg-node-mcollective* package that is provided with your OpenShift Enterprise subscription.

```
# yum install openshift-origin-msg-node-mcollective
```

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

Configuring MCollective on the node host

Now that we have MCollective installed on the node host, we need to configure the package to be able to communicate with the broker application service. In order to accomplish this, we want to replace the contents of the MCollective server.cfg configuration file to point to our correct stomp host. Edit the */etc/mcollective/server.cfg* file and add the following information. If you used a different hostname for your broker application host, ensure that you provide the correct stomp host. You also need to ensure that you use the same username and password that you specified in the ActiveMQ configuration on the broker host.

The */etc/mcollective/server.cfg* file is available on the lab support website

```
topicprefix = /topic/
main_collective = mcollective
collectives = mcollective
libdir = /usr/libexec/mcollective
logfile = /var/log/mcollective.log
loglevel = debug
daemonize = 1
direct_addressing = n
registerinterval = 30

# Plugins
securityprovider = psk
plugin.psk = unset

connector = stomp
plugin.stomp.host = broker.example.com
plugin.stomp.port = 61613
plugin.stomp.user = mcollective
plugin.stomp.password = marionette

# Facts
factsource = yaml
plugin.yaml = /etc/mcollective/facts.yaml
```

Note: STOMP, or the Simple (or Streaming) Text Orientated Messaging Protocol, is the protocol used to encode MCollective messages for transport over ActiveMQ.

We need to ensure that MCollective is set to start on boot and also start the service for our current session.

```
# chkconfig mcollective on
# service mcollective start
```

At this point, MCollective on the node host should be able to communicate with the broker application host. We can verify this by running the *mco ping* command on the broker.example.com host.

```
# mco ping
```

If MCollective was installed and configured correctly, you should see node.example.com in the output from the previous command.

Lab 11 Complete!

Lab 12: Installing and configuring the OpenShift Enterprise node packages (Estimated time: 10 minutes)

Server used:

- node host

Tools used:

- text editor
- yum
- lokkit
- chkconfig

Just as we installed specific packages that provide the source code and functionality for the broker application to work correctly, the node host also has a set of packages that need to be installed to properly identify the host as a node that will contain application gears.

Installing the core packages

The following packages are required for your node host to work correctly:

- rubygem-openshift-origin-node
- rubygem-passenger-native
- openshift-origin-port-proxy
- openshift-origin-node-util

Installing these packages can be performed in one yum install command.

```
# yum install rubygem-openshift-origin-node rubygem-passenger-native openshift-origin-port-proxy openshift-origin-node-util
```

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

Installing cartridges that the node host will support

OpenShift Enterprise gears can be created based upon a cartridge that exists in the system. The

cartridge provides the functionality that a consumer of the PaaS can use to create specific application types, databases, or other functionality. OpenShift Enterprise also provides an extensive cartridge API that will allow you to create your own custom cartridge types for your specific deployment needs. At the time of this writing, the following optional application cartridges are available for consumption on the node host.

- openshift-origin-cartridge-diy–0.1 diy ("do it yourself") application type
- openshift-origin-cartridge-haproxy–1.4 haproxy–1.4 support
- openshift-origin-cartridge-jbossews–1.0 JBoss EWS Support
- openshift-origin-cartridge-jbosseap–6.0 JBossEAP 6.0 support
- openshift-origin-cartridge-jenkins–1.4 Jenkins server for continuous integration
- openshift-origin-cartridge-ruby–1.9-scl Ruby 1.9 support
- openshift-origin-cartridge-perl–5.10 mod_perl support
- openshift-origin-cartridge-php–5.3 PHP 5.3 support
- openshift-origin-cartridge-python–2.6 Python 2.6 support
- openshift-origin-cartridge-ruby–1.8 Ruby Rack support running on Phusion Passenger (Ruby 1.8)

If you want to provide scalable PHP applications for your consumers, you would want to install the openshift-origin-cartridge-haproxy–1.4 and the openshift-origin-cartridge-php–5.3 cartridges.

For database and other system related functionality, OpenShift Enterprise provides the following:

- openshift-origin-cartridge-cron–1.4 Embedded crond support
- openshift-origin-cartridge-jenkins-client–1.4 Embedded jenkins client
- openshift-origin-cartridge-mysql–5.1 Embedded MySQL server
- openshift-origin-cartridge-postgresql–8.4 Embedded PostgreSQL server

The only required cartridge is the openshift-origin-cartridge-cron–1.4 package.

Note: If you are installing a multi-node configuration, it is important to remember that each node host *must* have the same cartridges installed.

Let's start by installing the cron package, which is required for all OpenShift Enterprise deployments.

```
# yum install openshift-origin-cartridge-cron-1.4
```

For this lab, let's also assume that we want to only allow scalable PHP applications that can connect to MySQL on our OpenShift Enterprise deployment. Issue the following command to installed the

required cartridges:

```
# yum install openshift-origin-cartridge-haproxy-1.4 openshift-origin-cartridge-php-5.3 openshift-origin-cartridge-mysql-5.1
```

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.

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

Starting required services on the node host

The node host will need to allow HTTP, HTTPS, and SSH traffic to flow through the firewall. We also want to ensure that the httpd, network, and sshd services are set to start on boot.

```
# lokkit --service=ssh
# lokkit --service=https
# lokkit --service=http
# chkconfig httpd on
# chkconfig network on
# chkconfig sshd on
```

Lab 12 Complete!

Lab 13: Configuring PAM namespace module, Linux control groups (cgroups), and user quotas (Estimated time: 10 minutes)

Server used:

- node host

Tools used:

- text editor
- sed
- restorecon
- chkconfig
- service
- mount
- quotacheck

Configuring PAM to use the OpenShift Enterprise configuration

The pam_namespace PAM module sets up a private namespace for a session with polyinstantiated directories. A polyinstantiated directory provides a different instance of itself based on user name, or when using SELinux, user name, security context or both. OpenShift Enterprise ships with its own PAM configuration and we need to configure the node to use the configuration.

```
# sed -i -e 's|pam_selinux|pam_openshift|g' /etc/pam.d/sshd
```

You also need to enter the following script on the command line:

This script is available for download from the lab support website.

```
for f in "runuser" "runuser-l" "sshd" "su" "system-auth-ac"; \
do t="/etc/pam.d/$f"; \
if ! grep -q "pam_namespace.so" "$t"; \
then echo -e "session\t\trequired\tpam_namespace.so no_unmount_on_close" >> "$t" ; \
fi; \
done;
```

Configuring Linux Control Groups (cgroups)

Cgroups allows you to allocate resources—such as CPU time, system memory, network bandwidth, or combinations of these resources—among user-defined groups of tasks (processes) running on a system. You can monitor the cgroups you configure, deny cgroups access to certain resources, and even reconfigure your cgroups dynamically on a running system.

Run the following command to configure cgroups for OpenShift Enterprise.

```
# cp -f /usr/share/doc/rubygem-openshift-origin-node-*/cgconfig.conf /etc/cgconfig.conf
# restorecon -v /etc/cgconfig.conf
# mkdir /cgroup
# restorecon -v /cgroup
# chkconfig cgconfig on
# chkconfig cgred on
# chkconfig openshift-cgroups on
# service cgconfig restart
# service cgred restart
# service openshift-cgroups start
```

In order for cgroups to work correctly, you need to ensure that services are started in the correct order.

- service cgconfig start
- service cgcred start
- service openshift-cgroups start

To verify that your cgroups configuration is correct, let’s check a few security contexts:

```
# ls -alZ /etc/cgconfig.conf
```

Output should be:

```
-rw-r--r--. root root system_u:object_r:cgconfig_etc_t:s0 /etc/cgconfig.conf
```

The context of the /cgroups directory:

```
ls -alZ /|grep cgroup
```

Output should be:

```
drwxr-xr-x. root root system_u:object_r:cgroup_t:s0    cgroup
```

Setting up disk quotas

When a consumer of OpenShift Enterprise creates an application gear, we need to be able to control and set the amount of disk space that the gear can consume. This configuration is located in the */etc/openshift/resource_limits.conf* file. The two settings of interest are the qouta_files and the quota_blocks. The quota_files setting specifies the total number of files that a gear / user is allowed to own. The quota_blocks is the actual amount of disk storage that the gear is allowed to consume — where 1 block is equal to 1024 bytes.

In order to enable *usrqouta* on the filesystem, you will need to add the *usrquote* option in the */etc/fstab* for the mount of /var/lib/openshift. In this lab, the /var/lib/openshift directory is mounted as part of the root filesystem. The corresponding line in the /etc/fstab file looks like

```
/dev/mapper/VolGroup-lv_root /                       ext4    defaults        1 1
```

In order to add the usrquote option to this mount point, change the entry to the following:

```
/dev/mapper/VolGroup-lv_root /                       ext4    defaults,usrquota        1 1
```

This file is available for download from the lab support website.

For the usrquota option to take effect, we need to reboot the node host or simply remount the filesystem.

```
# mount -o remount /
```

And then generate user quota info for the mount point:

```
# quotacheck -cmug /
```

Lab 13 Complete!

Lab 14: Configuring SELinux and System Control Settings (Estimated time: 5 minutes)

Server used:

- node host

Tools used:

- text editor
- setsebool
- fixfiles
- restorecon
- sysctl

Configuring SELinux

The OpenShift Enterprise node requires several SELinux boolean values to be set in order to operate correctly.

SELinux Boolean Values

Variable Name	Description
httpd_unified	Allow the broker to write files in the “http” file context
httpd_can_network_connect	Allow the broker application to access the network
httpd_can_network_relay	Allow the SSL termination Apache instance to access the backend Broker application
httpd_run_stickshift	Enable passenger-related permissions
httpd_read_user_content	Allow the node to read application data
httpd_enable_homedirs	Allow the node to read application data
allow_polyinstantiation	Allow polyinstantiation for gear containment

To set these values and then relabel files to the correct context, issue the following commands:

```
# setsebool -P httpd_unified=on httpd_can_network_connect=on httpd_can_network_relay=on
httpd_read_user_content=on httpd_enable_homedirs=on httpd_run_stickshift=on allow_polyinstanti-
ation=on
# restorecon -rv /var/run
# restorecon -rv /usr/sbin/mcollectived /var/log/mcollective.log /var/run/mcollectived.pid
# restorecon -rv /var/lib/openshift /etc/openshift/node.conf /etc/httpd/conf.d/openshift
```

Configuring System Control Settings

We need to modify the */etc/sysctl.conf* configuration file to increase the number of kernel semaphores (to allow many httpd processes), increase the number ephemeral ports, and to also increase the connection tracking table size. Edit the file in your favorite text editor and add the following lines to the bottom of the file:

```
# Added for OpenShift Enterprise
kernel.sem = 250  32000 32  4096
net.ipv4.ip_local_port_range = 15000 35530
net.netfilter.nf_conntrack_max = 1048576
```

Once the changes have been made, we need to reload the configuration file.

```
# sysctl -p /etc/sysctl.conf
```

Lab 14 Complete!

Lab 15: Configuring SSH, OpenShift Port Proxy, and node configuration (Estimated time: 20 minutes)

Server used:

- node host

Tools used:

- text editor
- perl
- lokkit
- chkconfig
- service
- openshift-facts

Configuring SSH to pass through the *GIT_SSH* environment variable

Edit the */etc/ssh/sshd_config* file and add the following lines

```
# Added to pass the GIT_SSH environment variable
AcceptEnv GIT_SSH
```

When a developer pushes a change up to their OpenShift Enterprise gear, an SSH connection is created. Because this may result in a high number of connections, we need to increase the limit of the number of connections allowed to the node host.

```
# perl -p -i -e "s/^#MaxSessions .*$/MaxSessions 40/" /etc/ssh/sshd_config
# perl -p -i -e "s/^#MaxStartups .*$/MaxStartups 40/" /etc/ssh/sshd_config
```

Configuring the port proxy

Multiple application gears can and will reside on the same node host. In order for these applications to receive HTTP requests to the node, we need to configure a proxy that will pass traffic to the gear application that is listening for connections on the loopback address. We need to open up a range of ports that the node can accept traffic on as well as ensure the port-proxy is started on boot.

```
# lokkit --port=35531-65535:tcp
# chkconfig openshift-port-proxy on
# service openshift-port-proxy start
```

If a node is restarted, we want to ensure that the gear applications are also restarted. OpenShift Enterprise provides a script to accomplish this task, but we need to configure the service to start on boot.

```
# chkconfig openshift-gears on
```

Configuring node settings for domain name

Edit the */etc/openshift/node.conf* file and specify the correct settings for your *CLOUD_DOMAIN, PUBLIC_HOSTNAME, and BROKER_HOST* IP address. For example, given the information in this lab, my settings are as follows:

```
PUBLIC_HOSTNAME="node.example.com"     # The node host's public hostname
PUBLIC_IP=“10.10.10.10”                # The node host's public IP address
BROKER_HOST="broker.example.com"       # IP or DNS name of broker host for REST API
```

Updating the *facter* database

Facter generates metadata files for MCollective and is normally run by cron. Run the following command to execute facter immediately to create the initial database and ensure that it runs properly:

```
# /etc/cron.minutely/openshift-facts
```

Rebooting the node

In order to verify that all services were installed and configured correctly, I suggest that you restart the node to ensure that all services start on boot as described in this post.

Testing the configuration

If everything in the current, and all previous labs were completed successfully, we can now test our deployment of OpenShift Enterprise. During this lab, we will setup an SSH tunnel to allow us to communicate with the broker and node hosts. The will allow us to connect to localhost on our lab machine and all traffic will be forwarded to your OpenShift Enterprise installation. In the next lab, we will update our local machines to point directly to the DNS server that we installed in Lab 2, but for now, an SSH tunnel will allow us to test the installation.

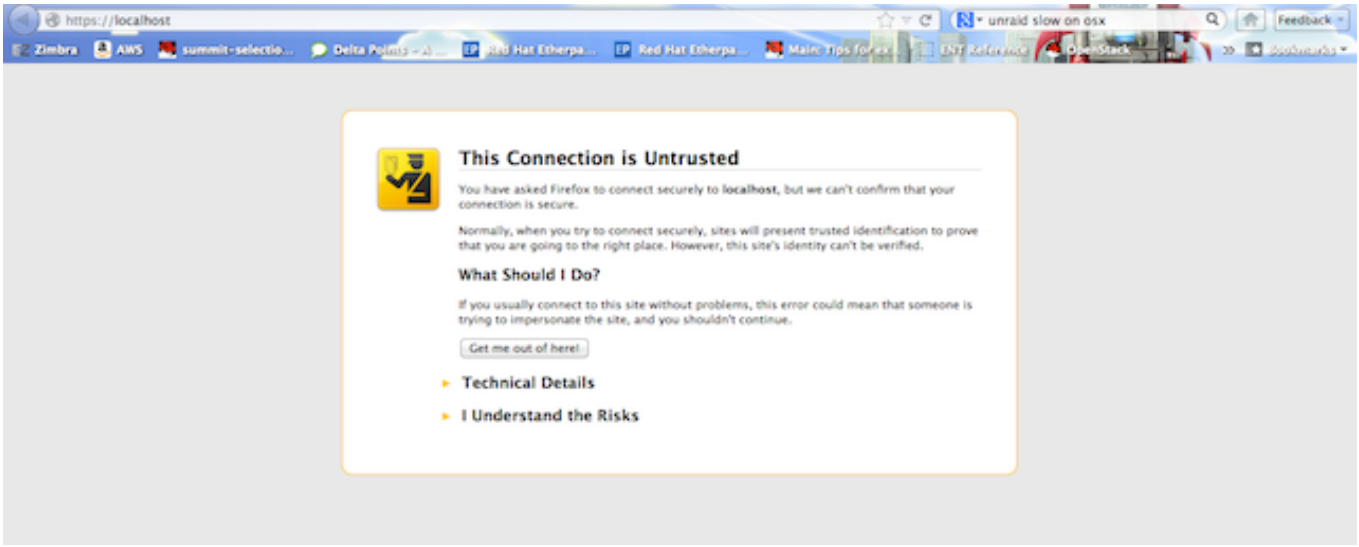
On your local machine, issue the following command, replacing the IP address with the IP address of your broker node:

```
# sudo ssh -f -N -L 80:broker.example.com:80 -L 8161:broker.example.com:8161 -L 443:broker.example.com:443 root@10.4.59.x
```

We have to use the sudo command in order to allow forwarding of low range ports. Once, you have entered the above command, and authenticated correctly, you should be able to view the web console by pointing your local browser to:

```
http://127.0.0.1
```

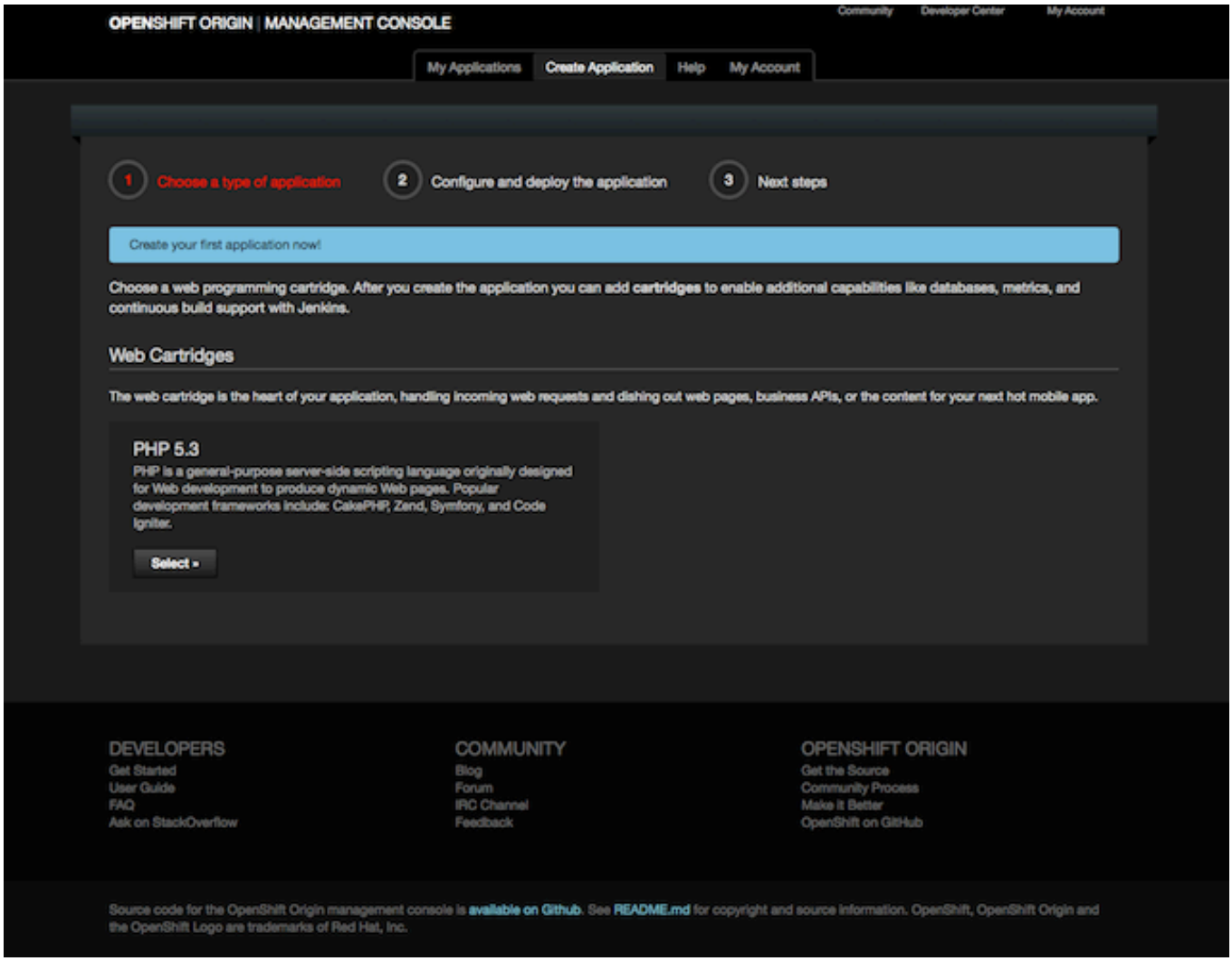
You will notice that you may, depending on your browser settings, have to accept the SSL certificate. In Firefox, the page will look similar to this:



Once you have accepted and added the SSL certificate, you will prompted to authenticate to the OpenShift console. Use the credentials that we created in a previous lab, which should be:

- Username: demo
- Password: demo

After you have authenticated, you should be presented with the OpenShift web console as shown below:



If you do not see the expected content, consult the troubleshooting section at the end of this manual.

Lab 15 Complete!

Lab 16: Configuring local machine for DNS resolution (Estimated time: 10 minutes)

Server used:

- local machine

Tools used:

- text editor
- networking tools

At this point, we should have a complete OpenShift Enterprise installation working correctly on the lab machines that were provided to you by the instructor. During the next portion of the training, we will be focussing on administration and usage of the OpenShift Enterprise PaaS. To make performing these tasks easier, it is suggested that you add the DNS server that we created in lab 2 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 10.x.x.x IP address with the correct address of your broker node.

```
nameserver 10.4.59.x
```

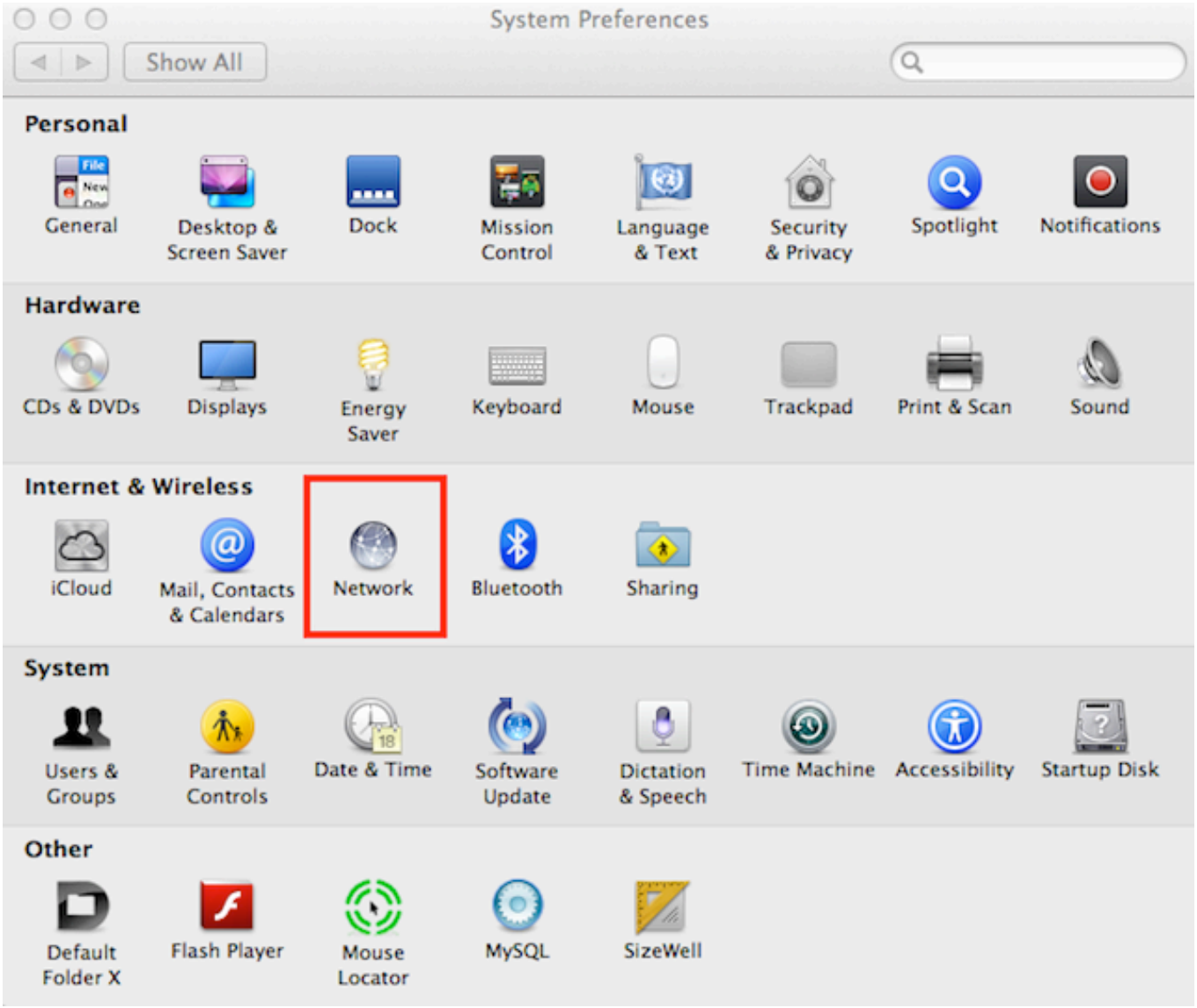
Once you have added the above nameserver, you should be able to communicate with your OpenShift Enterprise PaaS by using the server hostname. To test this out, ping the broker and node hosts from your local machine:

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

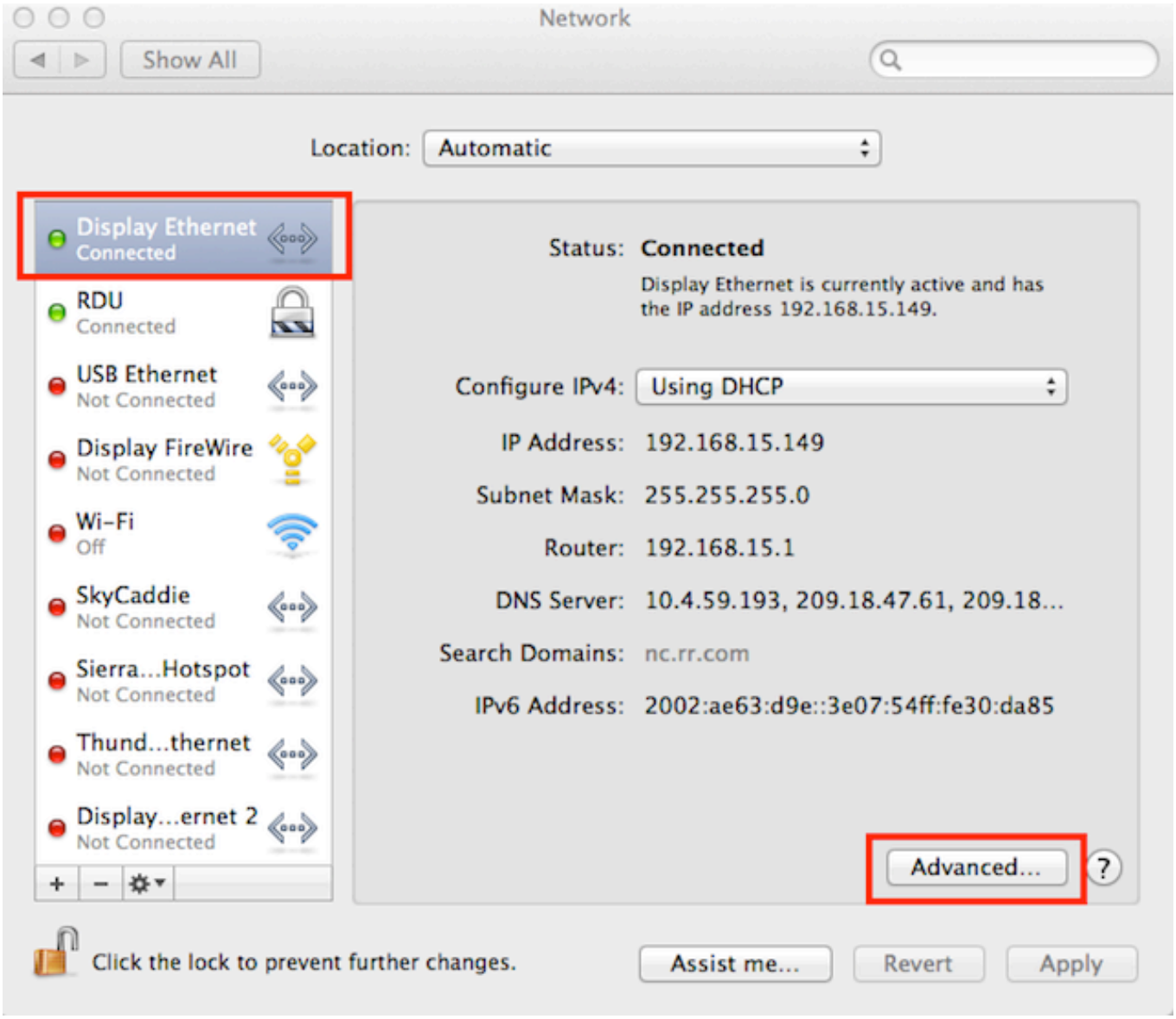
Configuring example.com resolution for OS X

If you are using OSX, you will notice that the operating has a */etc/resolv.conf* configuration file. However, the operating system does not respect this file and requires users to edit the DNS servers via the *System Preferences* tool.

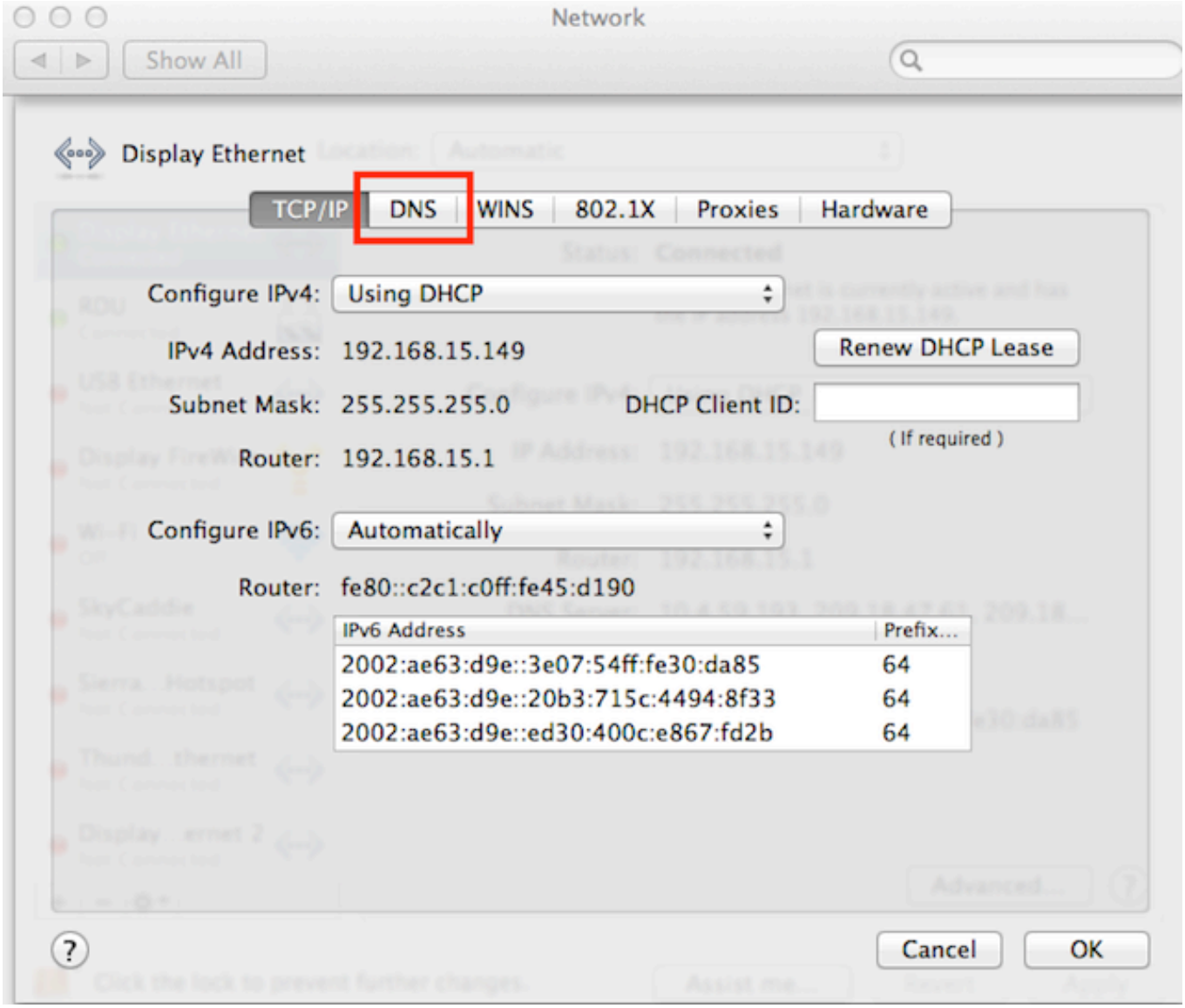
Open up the *System Preferences* tool and select the *Network* utility:



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

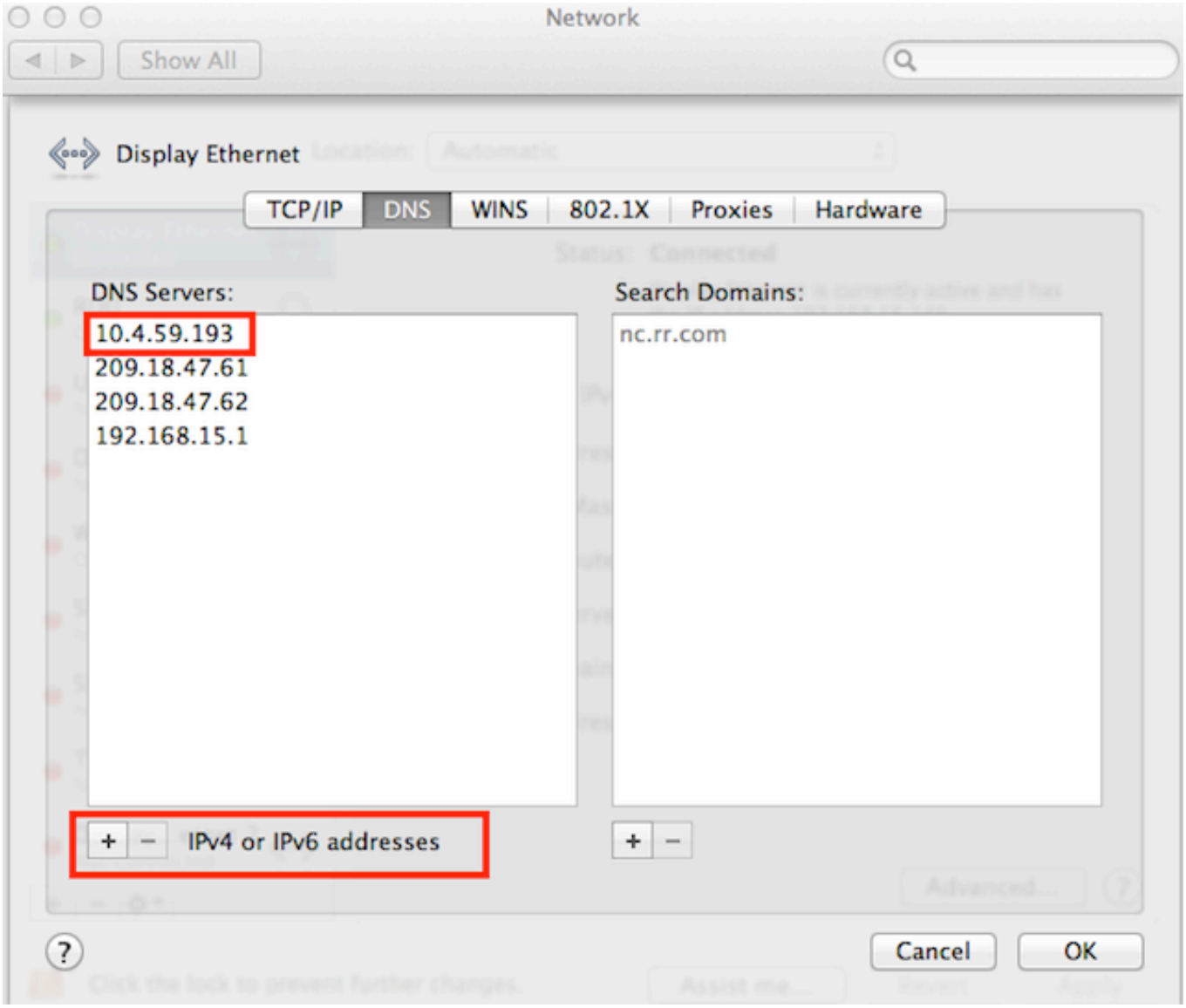


Select the DNS tab at the top of the window:



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

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



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

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

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

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

Lab 16 Complete!

Lab 17: Adding cartridges (Estimated time: 10 minutes)

Server used:

- node host
- broker host

Tools used:

- yum
- bundle

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

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

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

Listing available cartridges for your subscription

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

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

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

- openshift-origin-cartridge-abstract.noarch : OpenShift common cartridge components
- openshift-origin-cartridge-cron–1.4.noarch : Embedded cron support for express
- openshift-origin-cartridge-diy–0.1.noarch : Provides diy support
- openshift-origin-cartridge-haproxy–1.4.noarch : Provides embedded haproxy–1.4 support
- openshift-origin-cartridge-jbosseap–6.0.noarch : Provides JBossEAP6.0 support
- openshift-origin-cartridge-jbossews–1.0.noarch : Provides JBossEWS1.0 support
- openshift-origin-cartridge-jenkins–1.4.noarch : Provides jenkins–1.4 support
- openshift-origin-cartridge-jenkins-client–1.4.noarch : Embedded jenkins client support for

express

- openshift-origin-cartridge-mysql–5.1.noarch : Provides embedded mysql support
- openshift-origin-cartridge-perl–5.10.noarch : Provides mod_perl support
- openshift-origin-cartridge-php–5.3.noarch : Provides php–5.3 support
- openshift-origin-cartridge-postgresql–8.4.noarch : Provides embedded PostgreSQL support
- openshift-origin-cartridge-python–2.6.noarch : Provides python–2.6 support
- openshift-origin-cartridge-ruby–1.8.noarch : Provides ruby rack support running on Phusion Passenger
- openshift-origin-cartridge-ruby–1.9-scl.noarch : Provides ruby rack support running on Phusion Passenger

Installing JBoss support

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

Note: Execute the following on the node host.

```
# yum install openshift-origin-cartridge-jbosseap-6.0.noarch openshift-origin-cartridge-jbossews-1.0.noarch openshift-origin-cartridge-jenkins–1.4.noarch openshift-origin-cartridge-jenkins-client–1.4.noarch
```

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

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

Clearing the broker application cache

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

Note: Execute the following on the broker host.

```
# cd /var/www/openshift/broker
# bundle exec rake tmp:clear
```

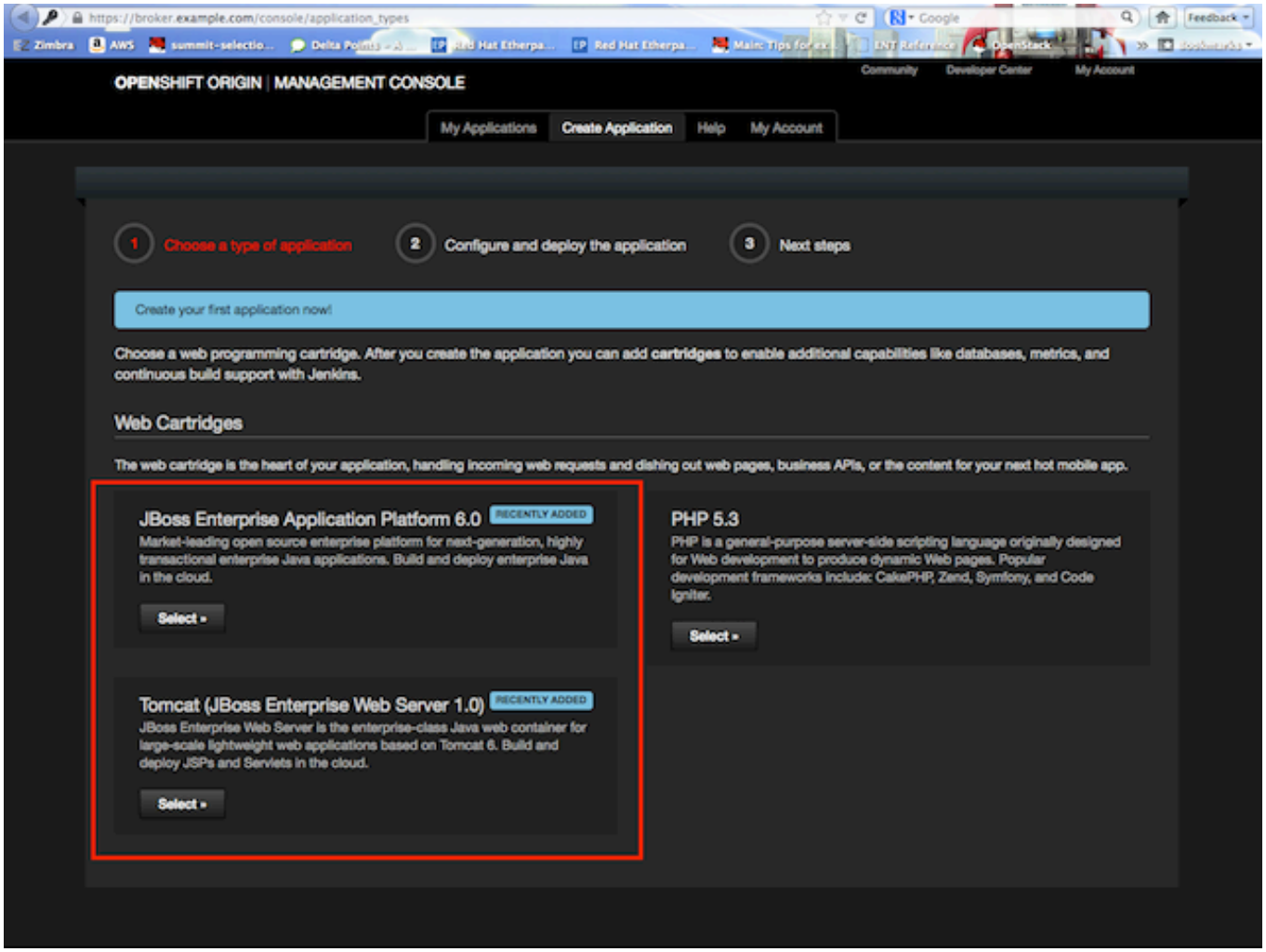
It may take several minutes before you see the new cartridges available on the web console as it takes a few minutes for the cache to completely clear.

Testing new cartridges

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

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

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



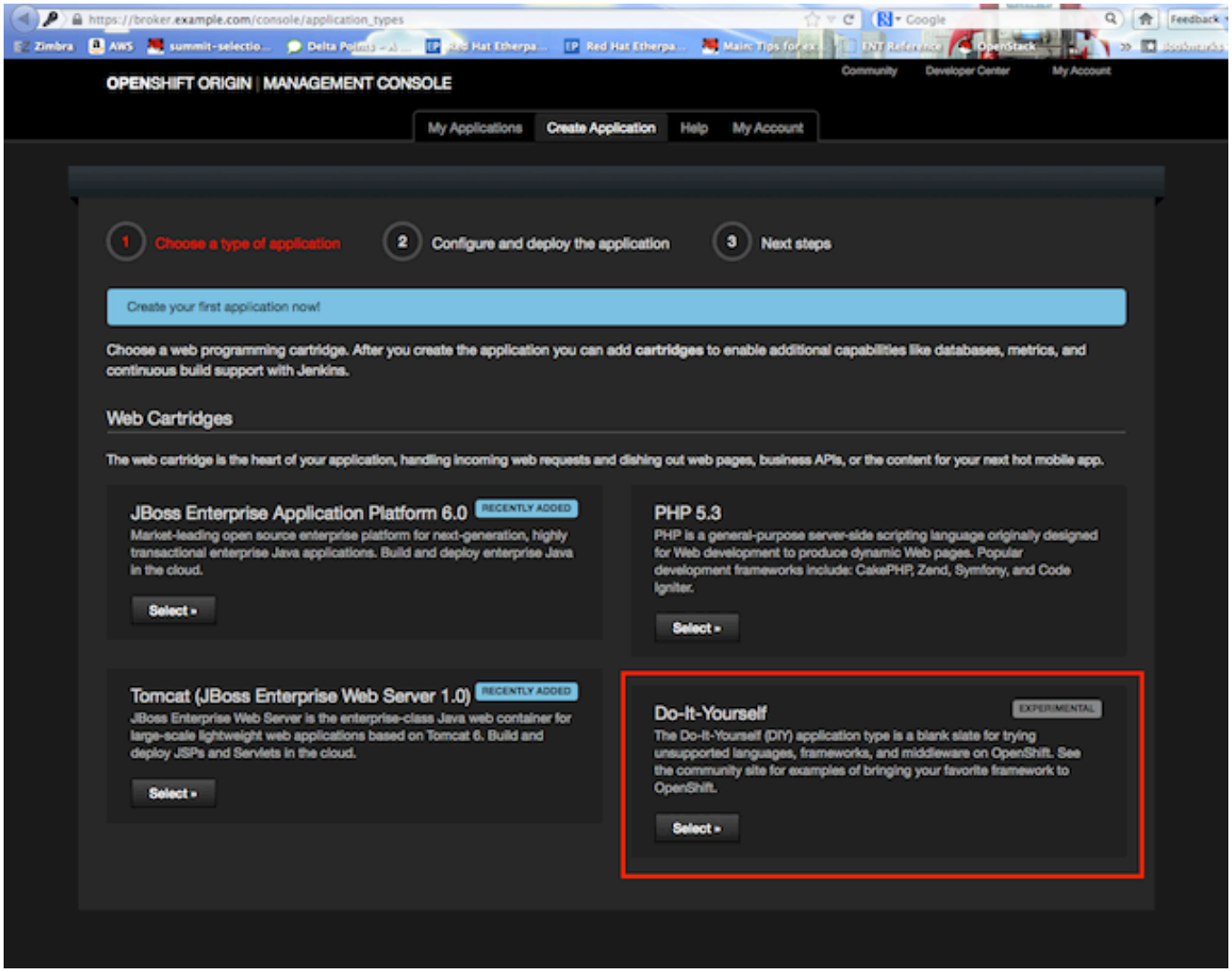
If you do not see the new cartridges available on the web console, check that the new cartridges are

available by viewing the contents of the */usr/libexec/openshift/cartridges* directory:

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

Installing the PostgreSQL and DIY cartridges

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



Lab 17 Complete!

Lab 18: Managing resources (Estimated time: 10 minutes)

Server used:

- node host
- broker host

Tools used:

- text editor
- oo-admin-ctl-user

Setting default gear quotas and sizes

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

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

The *DEFAULT_MAX_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.

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.

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

Setting the number of gears a specific user can create

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

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

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

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

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

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

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

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

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

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

Setting the type of gears a specific user can create

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 should 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 18 Complete!

Lab 19: Managing districts (Estimated time: 10 minutes)

Server used:

- node host
- broker host

Tools used:

- text editor
- oo-admin-ctl-district

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

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

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

Enabling districts

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

Note: Execute the following on the broker host.

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

Creating and populating districts

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

Note: Execute the following on the broker host.

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

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

```
Successfully created district: 513b50508f9f44aeb90090f19d2fd940

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

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

```
# mongo
```

This will drop you into the mongo shell where you can perform commands against the database. The first thing we need to do is let MongoDB know which database we want to use:

```
> show dbs
> use openshift_broker_dev
```

To list all of the available collections in the *openshift_broker_dev* database, you can issue the following command:

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

You should see the following collections returned:

```
[ "district", "system.indexes", "system.users", "user" ]
```

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

```
> db.district.find()
```

The output should be:

```
{ "_id" : "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" : [    1000, .....], , "available_capacity" :
6000 }
```

Note: The *server_identities* array does not contain any data yet.

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

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

You should see the following output:

```
Success!

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

Note: If you see an error message indicating that you can't add this node to the district because the node already has applications on it, consult the troubleshooting section.

Repeat the steps above to query the database for information about districts. Notice that the *server_identities* array now contains the following information:

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

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

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

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

Managing district capacity

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

- Maximum number of application per node : 100
- Maximum number of active applications per node : 100

Lab 19 Complete!

Lab 20: Installing the RHC client tools (Estimated time: 15 minutes)

Server used:

- localhost

Tools used:

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

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

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

Microsoft Windows

Installing Ruby for Windows

RubyInstaller 1.9 provides the best experience for installing Ruby on Windows XP, Vista, and Windows 7. Download the newest 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 and 17

To install from yum on Fedora 16 and 17, 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 6.2 and 6.3

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.

To add the OpenShift Enterprise YUM repository, download the repository file openshift.repo and move it to your **/etc/yum.repos.d/** directory.

```
$ sudo mv ~/openshift.repo /etc/yum.repos.d/
```

In order to install the rubygems package, the *RHEL Optional* channel must be enabled. There are two ways of doing this from the command line. If you are using the Certificate-Based RHN tooling, enter the following command:

```
$ sudo yum-config-manager --enable rhel-6-server-optional-rpms
```

If you are using RHN-Classic, enter the following command:

```
$ sudo rhn-channel --add -- channel=rhel-x86rhel-x86_64-server-optional-6
```

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

```
$ sudo yum 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 20 Complete!

Lab 21: Using *rhc setup* (Estimated time: 10 minutes)

Server used:

- localhost

Tools used:

- rhc

Configuring RHC setup

By default, the RHC command line tool will default to use the publicly hosted OpenShift environment. Since we are using our own enterprise environment, we need to tell *rhc* to use our broker.example.com server instead of openshift.com. In order to accomplish this, the first thing we need to do is set our *LIBRA_SERVER* variable to point to our broker host and then run the *rhc setup* command:

```
$ LIBRA_SERVER=broker.example.com rhc setup
```

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 that we created in a previous lab. After providing the username that you would like to connect with, you will be prompted for the password of the user account.

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

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

```
ose
```

Under the covers

The *rhc setup* tool is a convenient command line utility to ensure that the users operating system is configured properly to create and manage applications from the command line. After this command has been executed, a *.openshift* directory was created in the users home directory with some basic

configuration items specified in the *express.conf* file. The contents of that file are as follows:

```
# Default user login
default_rhlogin=‘demo’

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

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

Lab 21 Complete!

Lab 22: Creating a PHP application (Estimated time: 30 minutes)

Server used:

- localhost
- node host

Tools used:

- rhc

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

```
$ rhc app -h
```

This will provide you with the following output:

```
List of Actions
create          Create an application and adds it to a domain
git-clone       Clone and configure an application's repository locally
delete          Delete an application from the server
start           Start the application
stop            Stop the application
force-stop      Stops all application processes
restart         Restart the application
reload          Reload the application's configuration
tidy            Clean out the application's logs and tmp directories and tidy up the git repo on the server
show            Show information about an application
status          Show status of an application's gears

Global Options
-l, --rhlogin login      OpenShift login
-p, --password password  OpenShift password
-d, --debug              Turn on debugging
--timeout seconds        Set the timeout in seconds for network commands
--noprompt               Suppress the interactive setup wizard from running before a command
--config FILE            Path of a different config file
-h, --help               Display help documentation
-v, --version            Display version information
```

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 (-a or –app)** : The name of the application. The application name can only contain alpha-numeric characters and at max contain only 32 characters.
- **Type (-t or –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 -a firstphp -t php
```

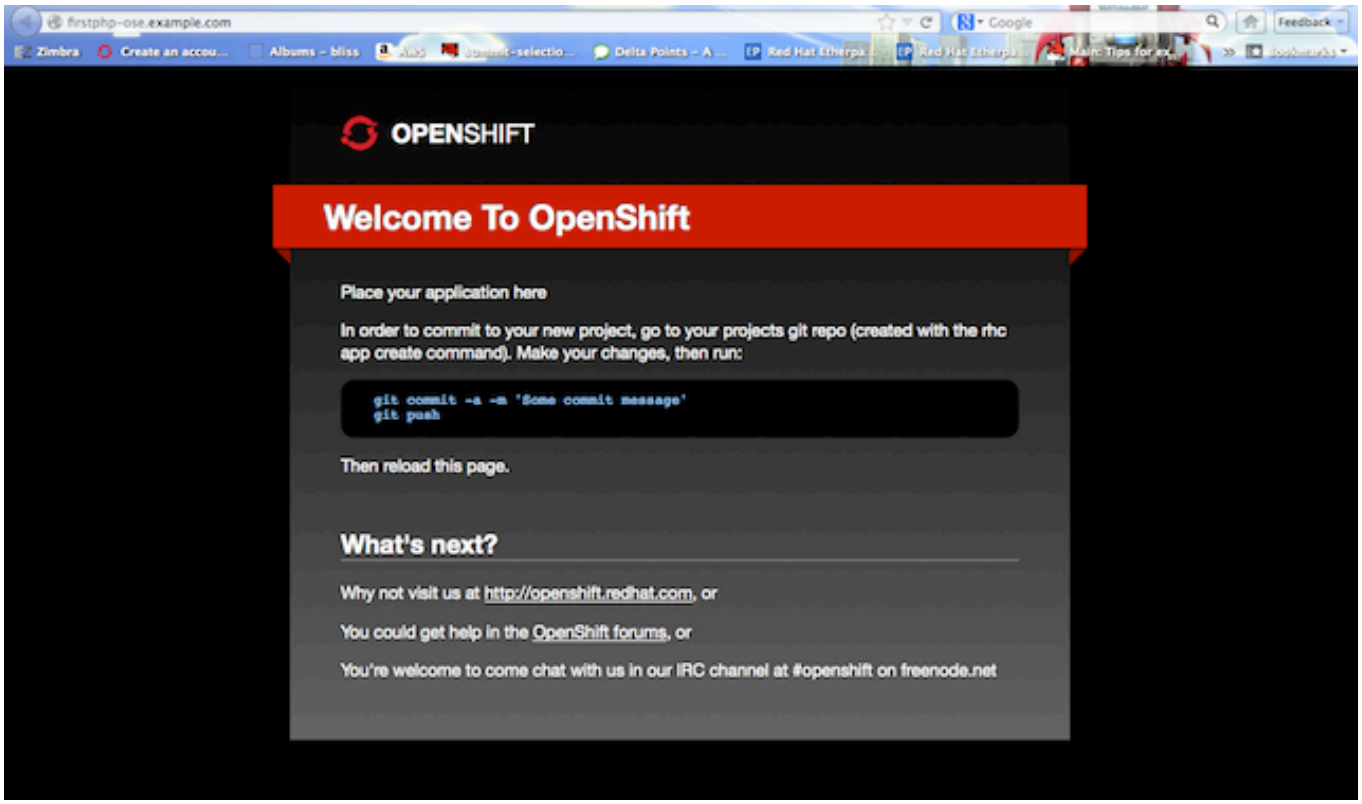
After entering that command, you should see the following output:

```
Password: ****

Creating application 'firstphp'
==============================

  Namespace: ose
  Scaling:   no
  Cartridge: php
  Gear Size: default

Your application's domain name is being propagated worldwide (this might take a minute)...
The authenticity of host 'firstphp-ose.example.com (10.4.59.221)' can't be established.
RSA key fingerprint is 6c:a5:e5:fa:75:db:5a:7f:dc:a2:44:ed:e4:97:af:3c.
Are you sure you want to continue connecting (yes/no)? yes
Cloning into 'firstphp'...
done

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

RESULT:
Application firstphp was created.
```

If you completed all of the steps in lab 16 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.example.com

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



What just happened?

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

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

Understanding the directory structure on the node host

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

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

You will see output similar to the following:

```
e9e92282a16b49e7b78d69822ac53e1d
```

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

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

You should see the following directories:

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

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

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

You will also notice the following three directories:

- app-root - Contains your core application code as well as your data directory where persistent data is stored.
- git - Your private git repository that was created upon gear creation.
- php–5.3 - The core PHP runtime and associated configuration files. Your application is served from this directory.

Understanding directory structure on the localhost

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

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

You should see the following information:

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

.git directory

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

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

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

```
[core]
   repositoryformatversion = 0
   filemode = true
   bare = false
   logallrefupdates = true
   ignorecase = true
[remote "origin"]
   fetch = +refs/heads/*:refs/remotes/origin/*
   url = ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.example.com/~/git/firstphp.git/
[branch "master"]
   remote = origin
   merge = refs/heads/master
[rhc]
   app-uuid = e9e92282a16b49e7b78d69822ac53e1d
```

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

.openshift directory

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

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

Action Hooks

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

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

- daily
- hourly
- minutely
- monthly
- weekly

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

libs directory

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

misc directory

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

php directory

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

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

Make a change to the PHP application and deploy updated code

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

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

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

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

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 name, using node.example.com for ServerName
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 name, using node.example.com for ServerName
remote: Done
remote: Running .openshift/action_hooks/post_deploy
To ssh://e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.example.com/~/git/firstphp.git/
   3edf63b..edc0805  master -> master
```

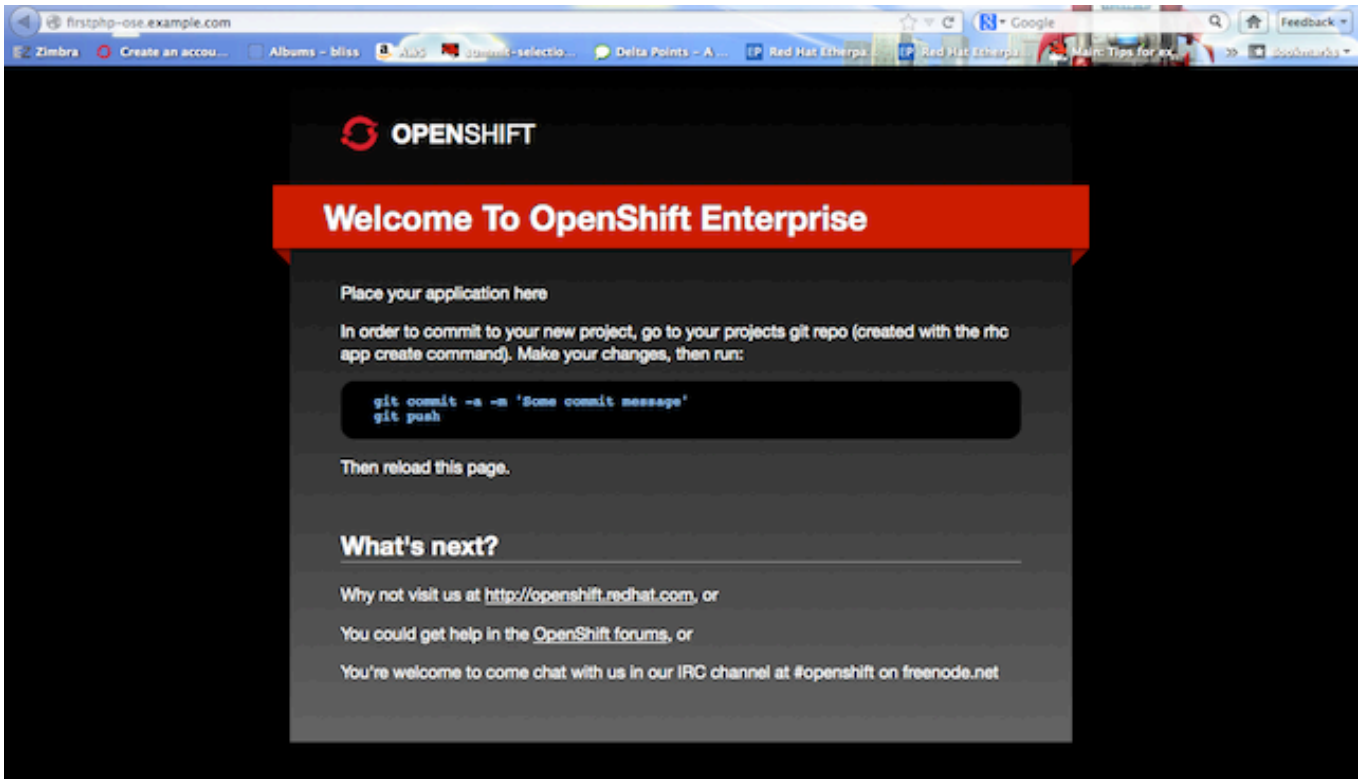
Notice that we stop the application runtime (Apache), deploy the code, and then run any action hooks that may have been specified in the .openshift directory.

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.example.com
```

You should see the updated code for the application.



Adding a new PHP file

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

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

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

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

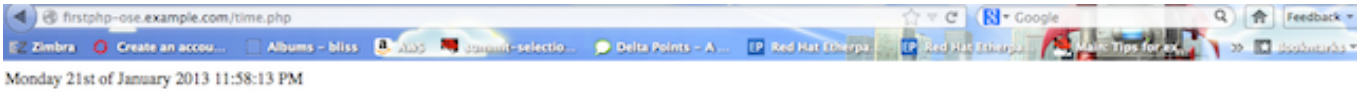
Verify code change

To verify that we have created and deployed the new PHP source file correctly, open up a web

browser and enter the following URL:

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

You should see the updated code for the application.



Enable *hot_deploy*

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

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

Pay attention to the output:

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

Lab 23: Managing an application (Estimated time: 30 minutes)

Server used:

- localhost
- node host

Tools used:

- rhc

Start/Stop/Restart OpenShift Enterprise application

OpenShift Enterprise provides commands to start,stop, and restart an application. If at any point in the future you decided 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 -a firstphp

RESULT:
firstphp stopped
```

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

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

<!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</address>
</body></html>
```

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

```
$ rhc app start -a firstphp

RESULT:
firstphp started
```

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

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

1
```

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

```
$ rhc app restart -a firstphp

RESULT:
firstphp restarted
```

Viewing application details

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

other details as shown below:

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



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

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 -a firstphp
Password: ****



RESULT:
Geargroup php-5.3 is started
```

Cleaning up an application

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

command is also available under *rhc app* and performs the following functions:

- Runs the *git gc* command on the application's remote git repository
- Clears the application's /tmp and log file directories. These are specified by the application's *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:

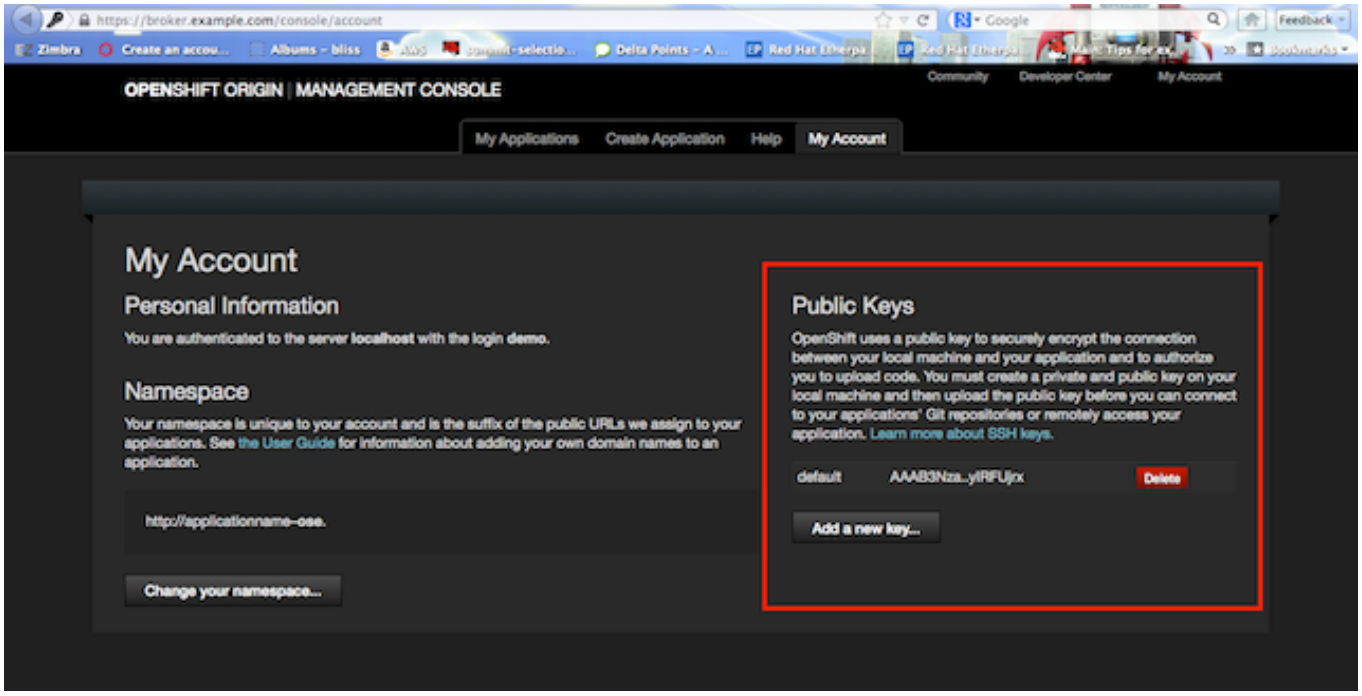
```
$ rhc app tidy -a firstphp
```

SSH to application gear

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

- Performing Git operations
- Remote access your application gear

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



Note: If you don't see an entry under "Public Keys" then you can either upload the SSH keys 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 OS X and Linux. For windows, you can use Putty. Instructions for installing Putty can be found on the OpenShift website.

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

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

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


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

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

```
$ ssh e9e92282a16b49e7b78d69822ac53e1d@firstphp-ose.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.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:

```
$ rhc tail -h
Usage: rhc tail <application>

Tail the logs of an application

Options for tail
  -n, --namespace namespace Namespace of your application
  -o, --opts options        Options to pass to the server-side (linux based) tail command (applicable to tail
command only) (-f is implicit.  See the linux tail man page full
list of options.) (Ex: --opts '-n 100')
  -f, --files files         File glob relative to app (default <application_name>/logs/*) (optional)
  -a, --app app             Name of application you wish to view the logs of
```

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 -a 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.example.com*. You should see a new entry in the log files that looks similar to this:

```
10.10.56.204 - - [22/Jan/2013:18:39:27 -0500] "GET / HTTP/1.1" 200 5242 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:19.0) Gecko/20100101 Firefox/19.0"
```

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

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 -u
```

Adding a custom domain to an application

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

After buying the domain name, you have to add a CName record for the custom domain name. Once

you have created the CName record, you can let OpenShift Enterprise know about the CName by using the *rhc alias* command.

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

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

Backing up an application

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

```
$ rhc snapshot save -a 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.

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 -a 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 -a someAppToDelete

Deleting application 'someAppToDelete'

RESULT:
Application 'someAppToDelete' successfully deleted
```

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 -a firstphp -f firstphp.tar.gz
```

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.example.com
```

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

before the delete operation that you performed earlier in this lab.

Viewing a thread dump of an application

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

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

```
$ rhc threaddump -a 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 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 250'
```

Lab 23 Complete!

Lab 24: Using cartridges (Estimated time: 30 minutes)

Server used:

- localhost
- node host

Tools used:

- rhc
- mysql
- tail
- git
- PHP

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

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:

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

Add the MySQL cartridge

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

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

You should see the following output:

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

Adding 'mysql-5.1' to application 'firstphp'
Success
mysql-5.1
=========
 Properties
 ==========
  Username      = admin
  Password      = aFh_GsHP63fV
  Connection URL = mysql://127.1.244.1:3306/
  Database Name  = firstphp
```

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(200), PRIMARY
KEY(user_id));
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, simple 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 node host.

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

You should see the following information return from the command:

```
OPENSHIFT_MYSQL_DB_PORT=3306
OPENSHIFT_MYSQL_DB_HOST=127.1.244.1
OPENSHIFT_MYSQL_DB_PASSWORD=aFh_GsHP63fV
OPENSHIFT_MYSQL_DB_USERNAME=admin
OPENSHIFT_MYSQL_DB_SOCKET=/var/lib/openshift/e9e92282a16b49e7b78d69822ac53e1d//mysql-5.1/socket/mysql.sock
OPENSHIFT_MYSQL_DB_URL=mysql://admin:aFh_GsHP63fV@127.1.244.1:3306/
OPENSHIFT_MYSQL_DB_LOG_DIR=/var/lib/openshift/e9e92282a16b49e7b78d69822ac53e1d//mysql-5.1/log
```

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.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 -a firstphp -c mysql
```

To stop the cartridge, enter the following command:

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

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

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

You should see the following message returned to your browser:

```
Could not connect to database
```

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

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

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

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

The user will then be presented with the following output:

```
Password: ****

mysql-5.1
=========
 Properties
 ==========
  Username      = admin
  Password      = aFh_GsHP63fV
  Database Name  = firstphp
  Connection URL = mysql://127.1.244.1:3306/
```

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 -a firstphp

Checking available ports...
Binding httpd -> 127.11.144.1:8080...
Binding mysqld -> 127.11.144.1:3306...
Forwarding ports, use ctl + c to stop
```

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://127.11.144.1:8080 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 loo alias 127.11.144.1
```

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 127.11.144.1
```

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

Lab 24 Complete!

Lab 25: Using the web console to create applications (Estimated time: 10 minutes)

Server used:

- localhost

Tools used:

- OpenShift Enterprise web console
- git

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

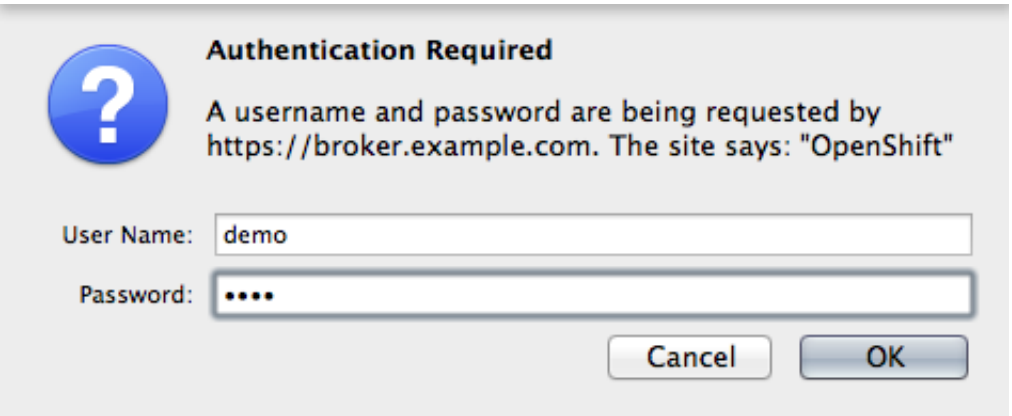
Having DNS resolution setup on your local machine, as discussed in lab 16, is crucial in order to complete this lab.

Authenticate to the web console

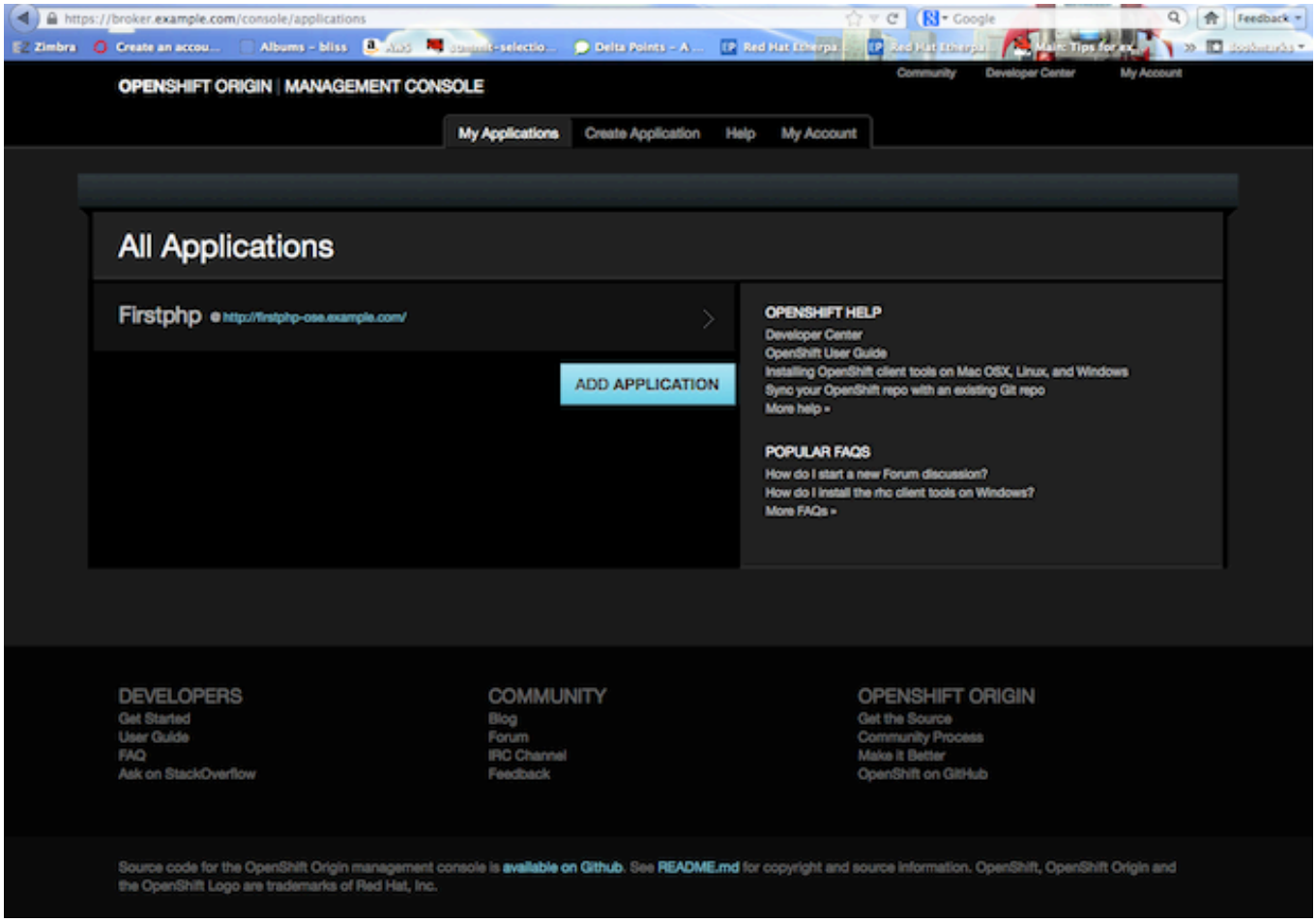
Open your favorite web browser and go to the following URL:

```
http://broker.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 you created in a previous lab.

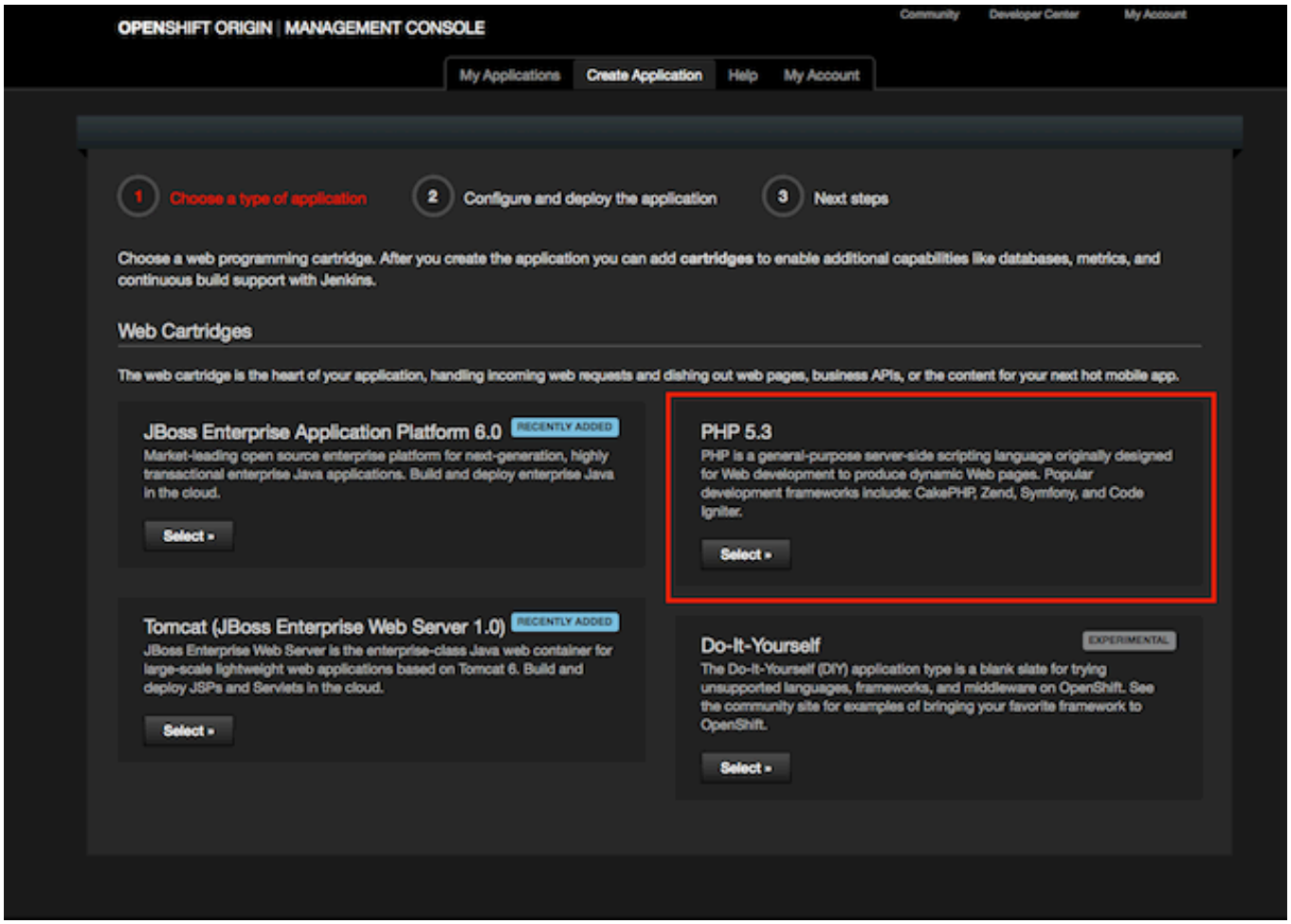


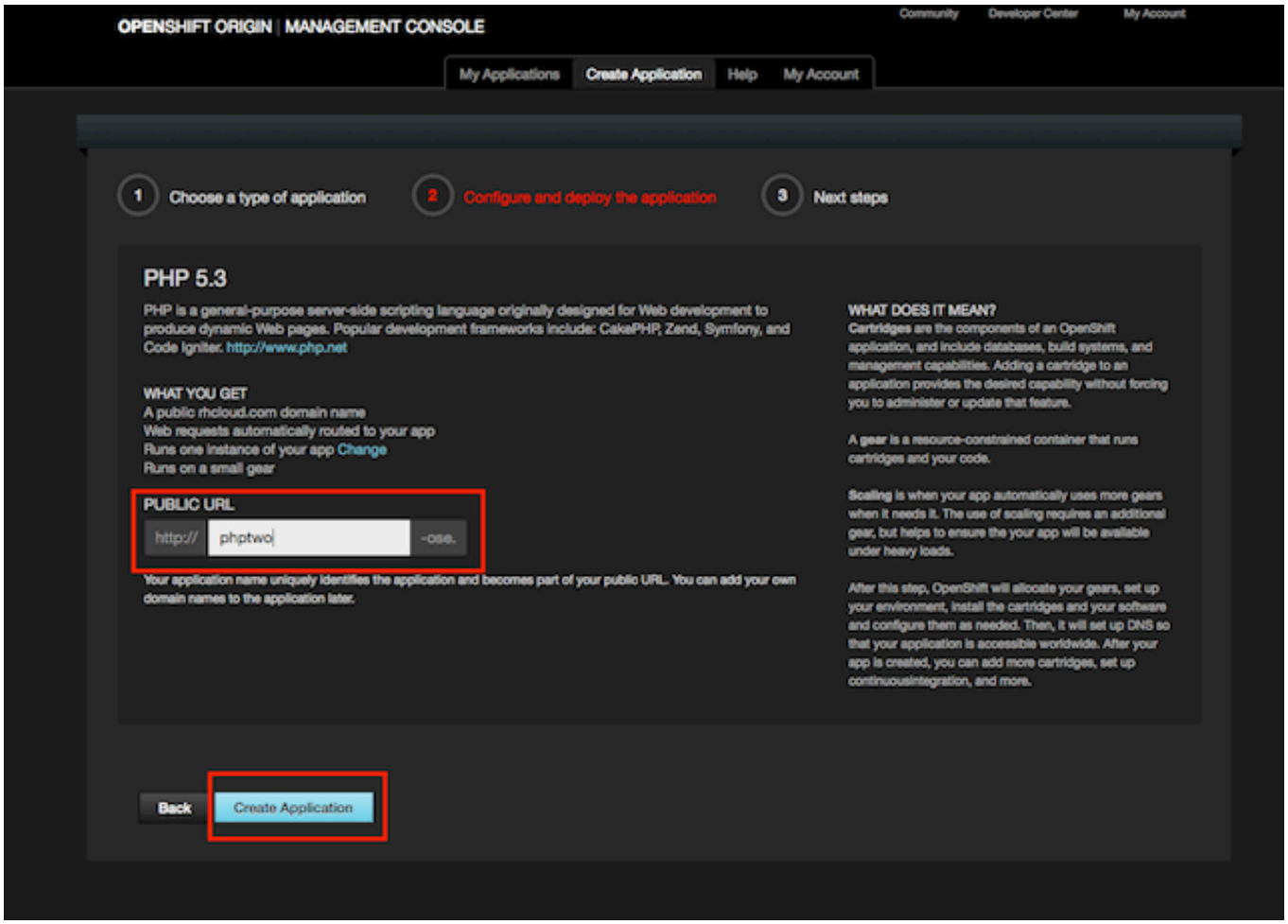
After entering in valid credentials, you will see the OpenShift Enterprise web console dashboard:



Creating a new application

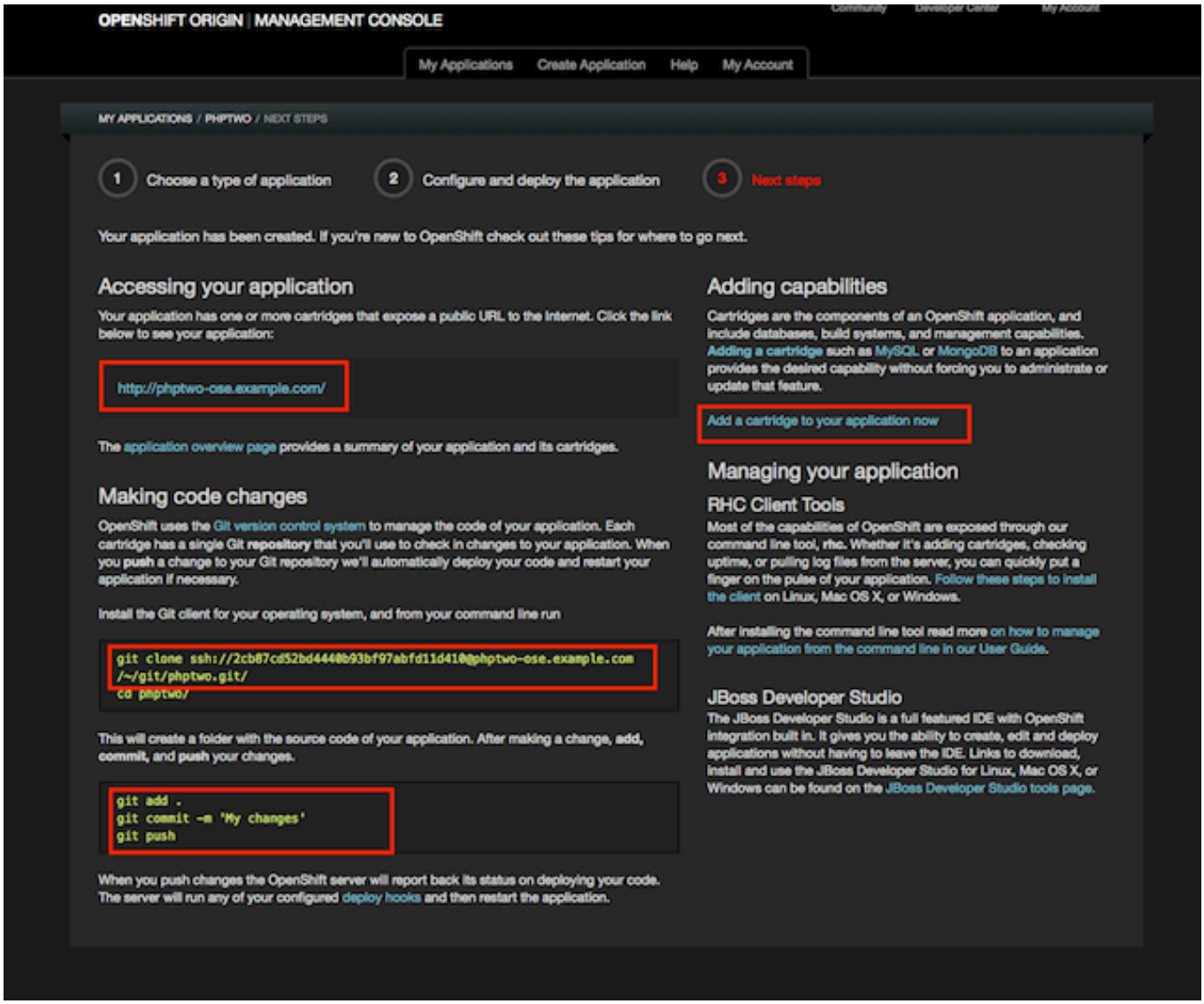
In order to create a new application using the web console, click on the *ADD APPLICATION* button. You will then be presented with a list of available run times that you can choose from. To follow along with our PHP examples above, let's create a new PHP application and name it *phptwo*.


OPENSHIFT ORIGIN | MANAGEMENT CONSOLE
Community
Developer Center
My Account
My Applications
Create Application
Help
My Account
1 Choose a type of application
2 Configure and deploy the application
3 Next steps
Choose a web programming cartridge. After you create the application you can add cartridges to enable additional capabilities like databases, metrics, and continuous build support with Jenkins.
Web Cartridges
The web cartridge is the heart of your application, handling incoming web requests and dishing out web pages, business APIs, or the content for your next hot mobile app.
JBoss Enterprise Application Platform 6.0
RECENTLY ADDED
Market-leading open source enterprise platform for next-generation, highly transactional enterprise Java applications. Build and deploy enterprise Java in the cloud.
Select »
PHP 5.3
PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. Popular development frameworks include: CakePHP, Zend, Symfony, and Code Igniter.
Select »
Tomcat (JBoss Enterprise Web Server 1.0)
RECENTLY ADDED
JBoss Enterprise Web Server is the enterprise-class Java web container for large-scale lightweight web applications based on Tomcat 6. Build and deploy JSPs and Servlets in the cloud.
Select »
Do-It-Yourself
EXPERIMENTAL
The Do-It-Yourself (DIY) application type is a blank slate for trying unsupported languages, frameworks, and middleware on OpenShift. See the community site for examples of bringing your favorite framework to OpenShift.
Select »



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

- URL for your application
- GIT repository URL
- Instructions for making code changes
- Link to add a cartridge



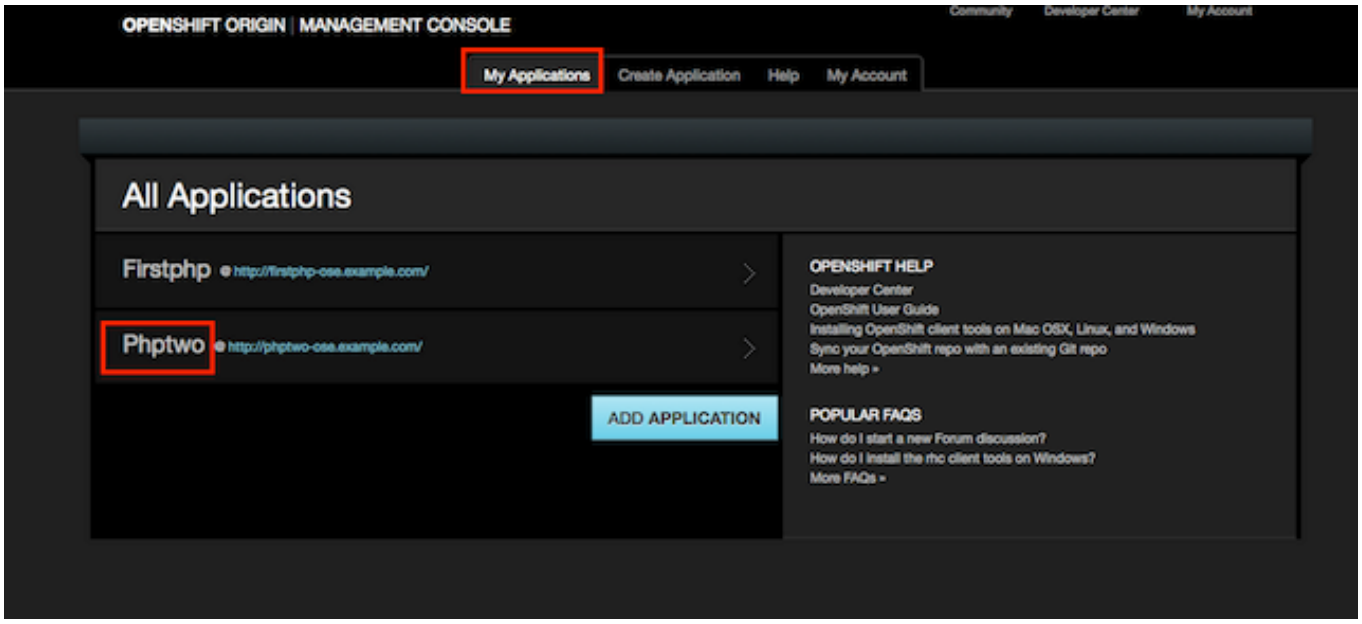
Clone your application repository

Open up a command prompt and clone your application repository with the instructions provided on the web console. Once you have a local copy of your application, make a small code change to the *index.php* and push your changes to your OpenShift Enterprise gear.

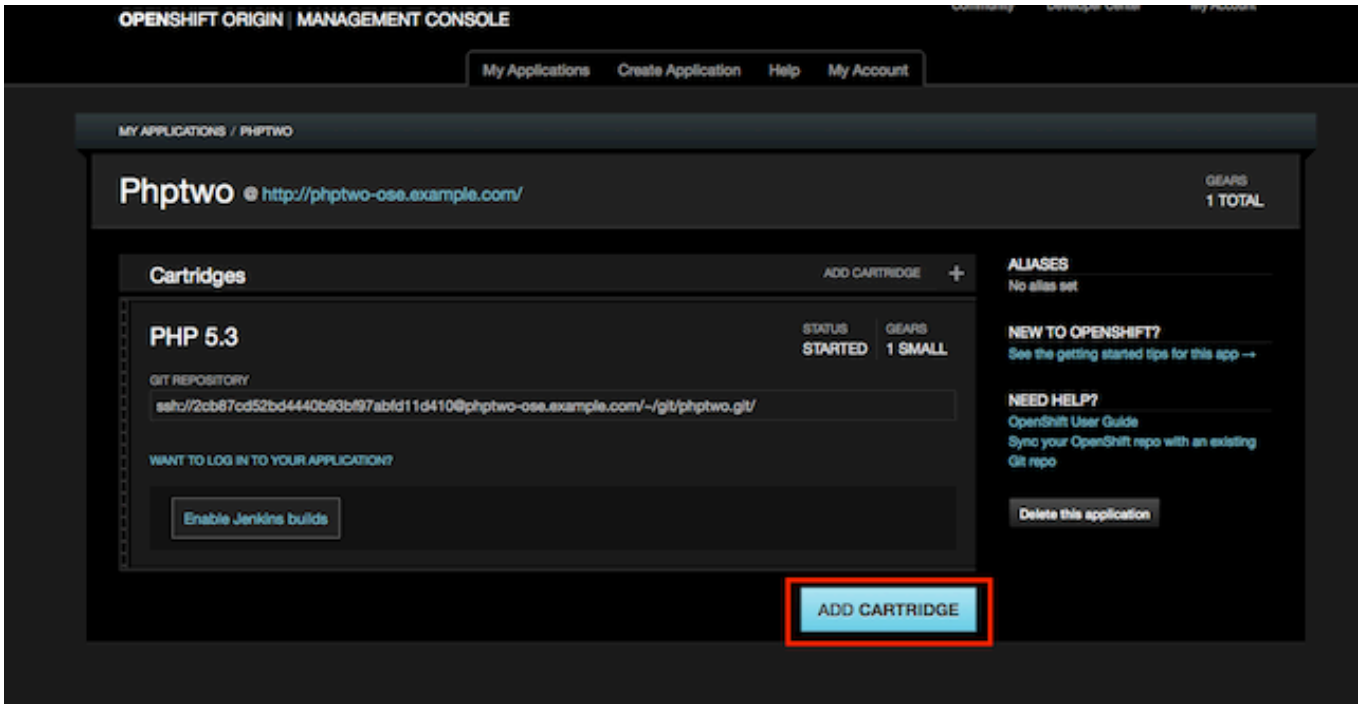
Once you have made a code change, view your application in a web browser to ensure that the code was deployed correctly to your server.

Adding a cartridge with the web console

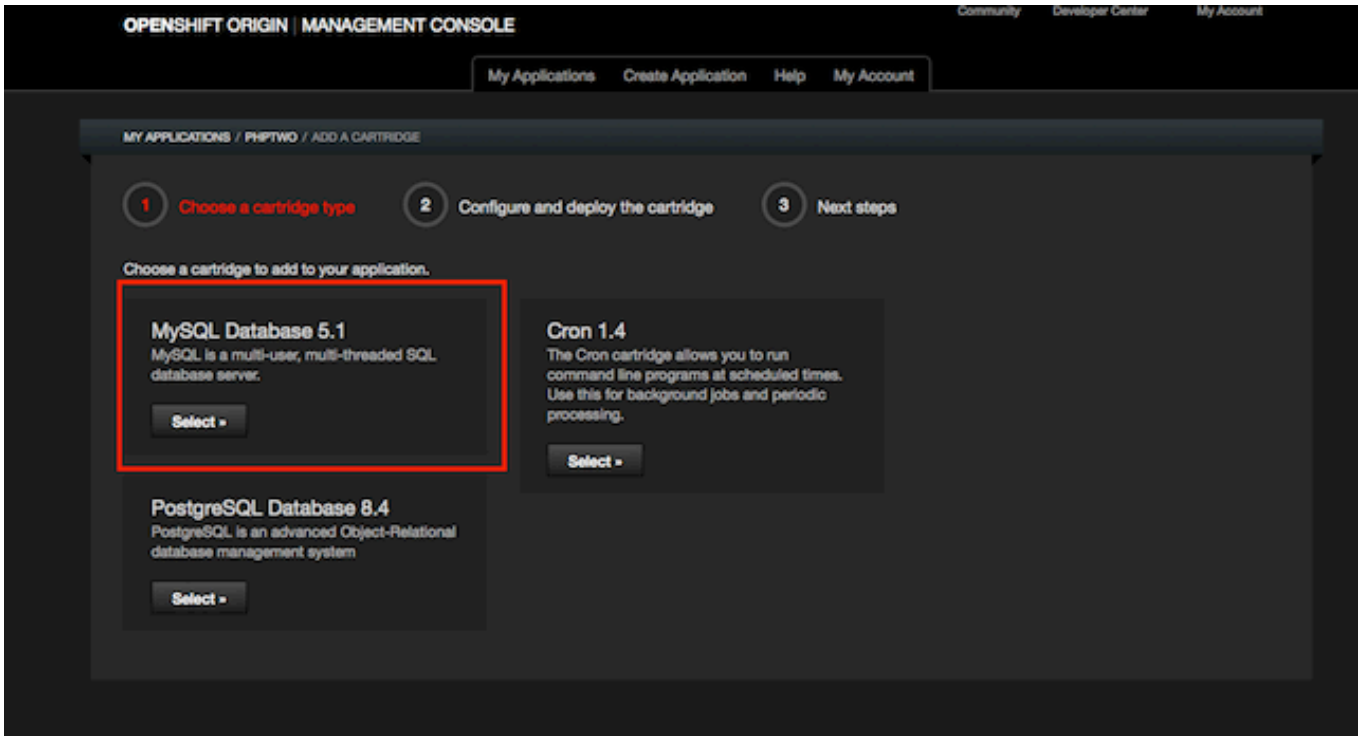
Click on the *My Applications* tab at the top of the screen and then select the *Phptwo* application by clicking on it.



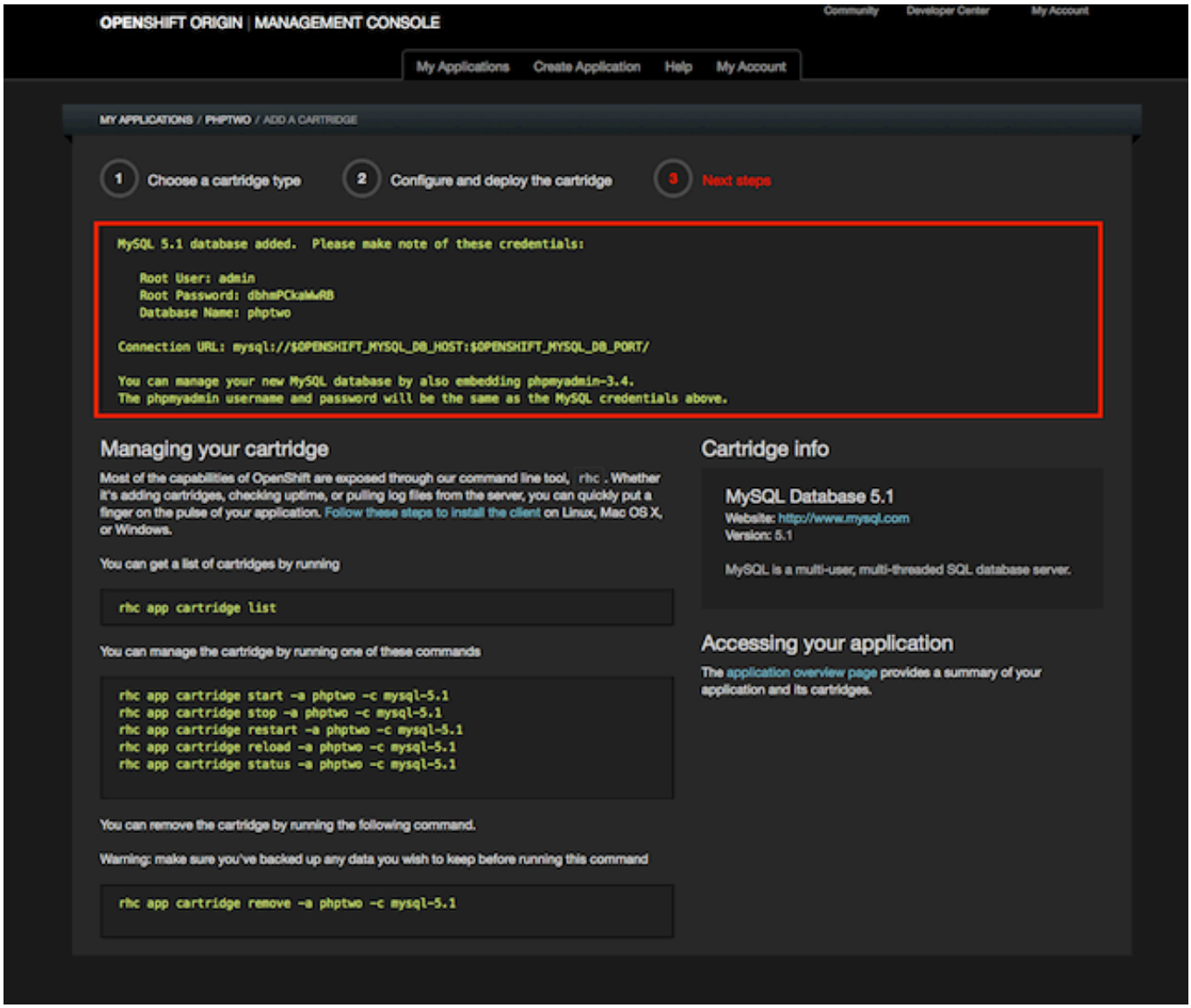
After clicking on the *Phptwo* application link, you will be presented with the management dashboard for that application. On this page, you can view the GIT repository URL, add a cartridge, or delete the application. We want to add the MySQL database to our application. To do this, click on the *ADD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



On the next screen, select the MySQL 5.1 cartridge.



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



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

Verify database connection

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

Lab 25 Complete!

Lab 26: Scaling an application (Estimated time: 15 minutes)

Server used:

- localhost
- node host

Tools used:

- rhc
- ssh
- git
- touch
- pwd

Application scaling enables your application to react to changes in HTTP traffic and automatically allocate the necessary resources to handle the current demand. The OpenShift Enterprise infrastructure monitors incoming web traffic and automatically adds additional gear of your web cartridge online to handle requests.

How scaling works

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

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

The algorithm for scaling up and scaling down is based on the number of concurrent requests to your application. OpenShift Enterprise allocates 10 connections per gear - if HAProxy sees that you're sustaining 90% of your peak capacity, it adds another gear. If your demand falls to 50% of your peak

capacity for several minutes, HAProxy removes that gear. Simple!

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

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

Create a scaled application

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

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

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

```
Password: ****

Creating application 'scaledapp'
================================

  Scaling:   yes
  Namespace: ose
  Cartridge: php
  Gear Size: default

Your application's domain name is being propagated worldwide (this might take a minute)...
The authenticity of host 'scaledapp-ose.example.com (10.4.59.221)' can't be established.
RSA key fingerprint is 6c:a5:e5:fa:75:db:5a:7f:dc:a2:44:ed:e4:97:af:3c.
Are you sure you want to continue connecting (yes/no)? yes
Cloning into 'scaledapp'...
done

scaledapp @ http://scaledapp-ose.example.com/
=============================================
```

Application Info

==========

UUID = 1a6d471841d84e8aaf25222c4cdac278

Gear Size = small

Git URL = ssh://1a6d471841d84e8aaf25222c4cdac278@scaledapp-ose.example.com/~/git/scaledapp.git/

SSH URL = ssh://1a6d471841d84e8aaf25222c4cdac278@scaledapp-ose.example.com

Created = 4:20 PM

Cartridges

==========

php-5.3

haproxy-1.4

Scaling Info

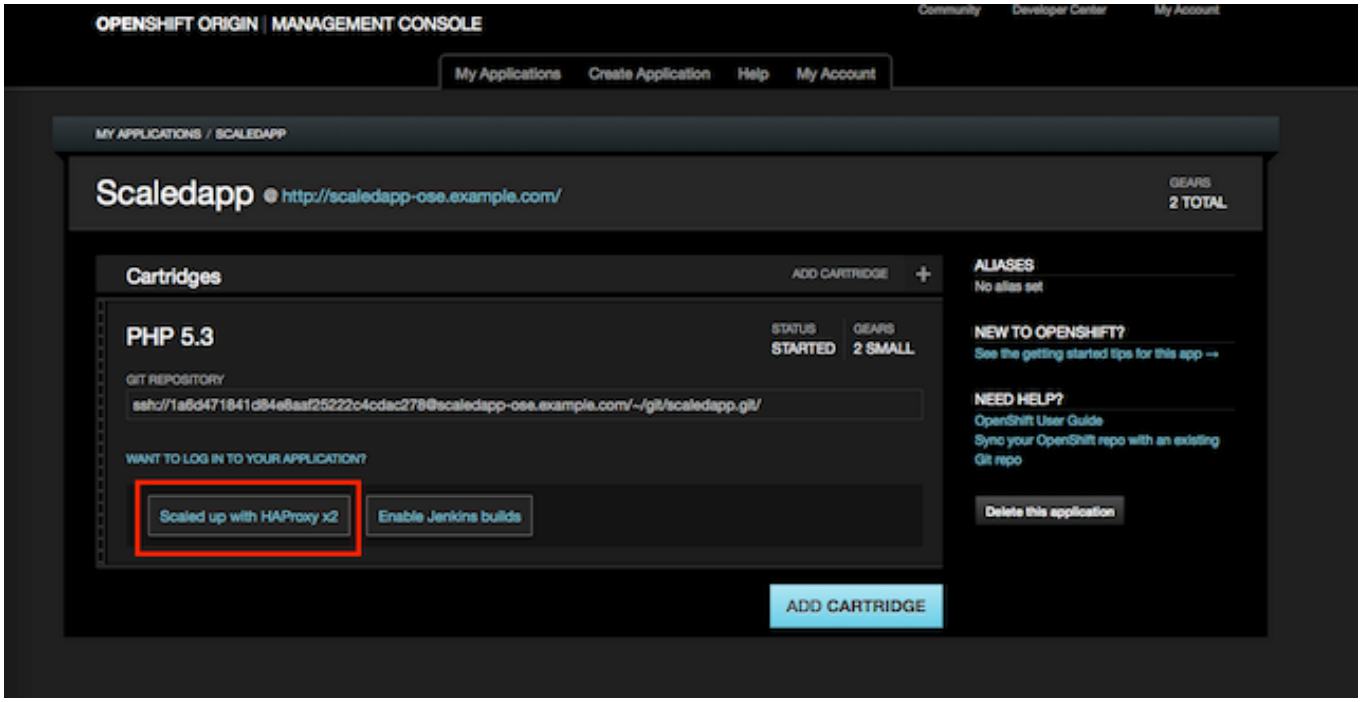
==========

Scaled x2 (minimum: 2, maximum: available gears) with haproxy-1.4 on small gears

RESULT:

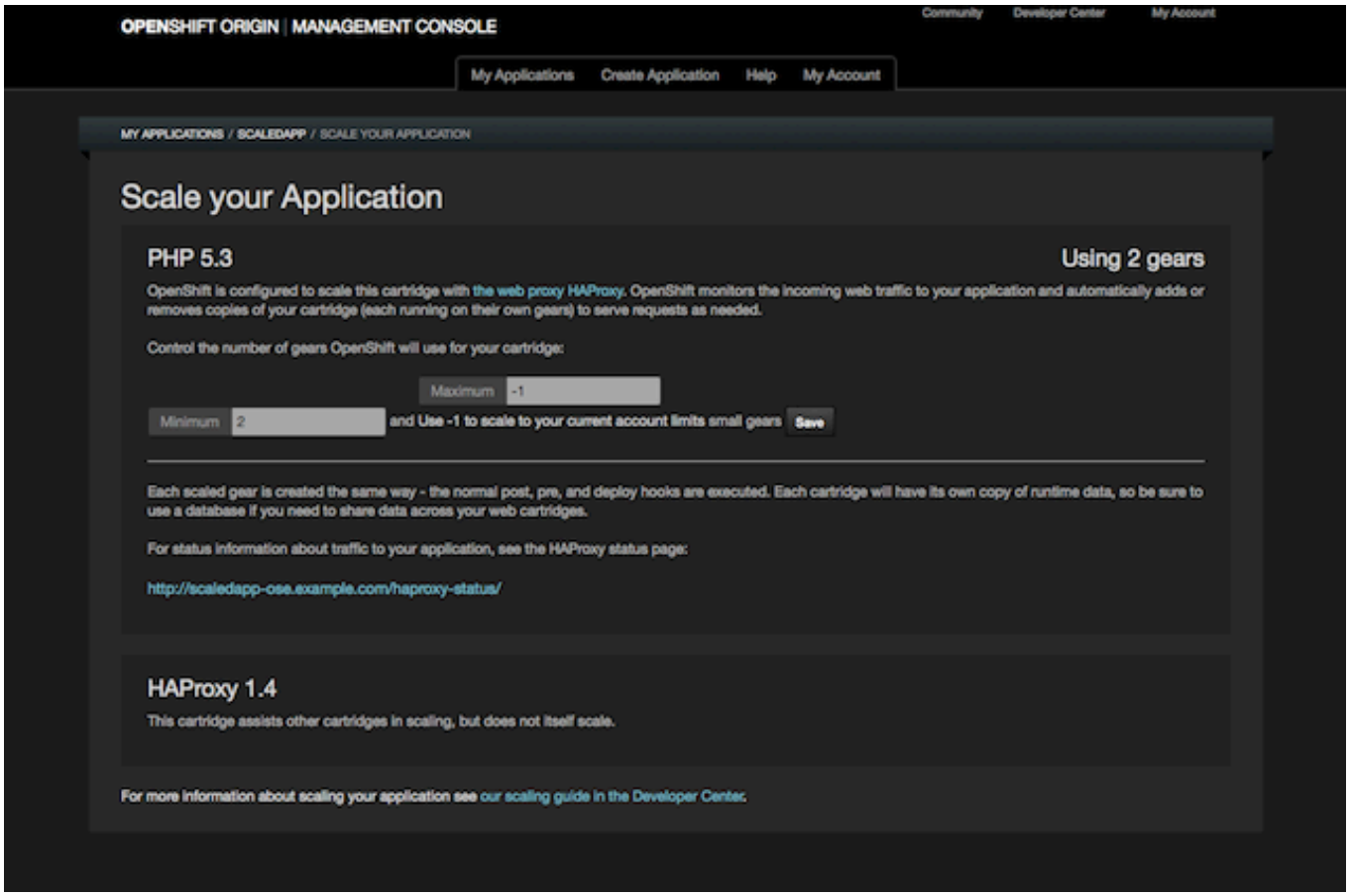
Application scaledapp was created.

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



Setting the scaling strategy

OpenShift Enterprise allows users the ability to set the minimum and maximum numbers of gears that an application can use to handle increased HTTP traffic. This scaling strategy is exposed via the web console. While on the application details screen, click the *Scaled up with HAProxy x2* button to change the default scaling rules.



Manual scaling

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

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

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

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

```
$ touch .openshift/markers/disable_auto_scaling
$ git add .
$ git commit -a “remove automatic scaling”
$ git push
```

To add a new gear to your application, SSH to your node host and enter 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:

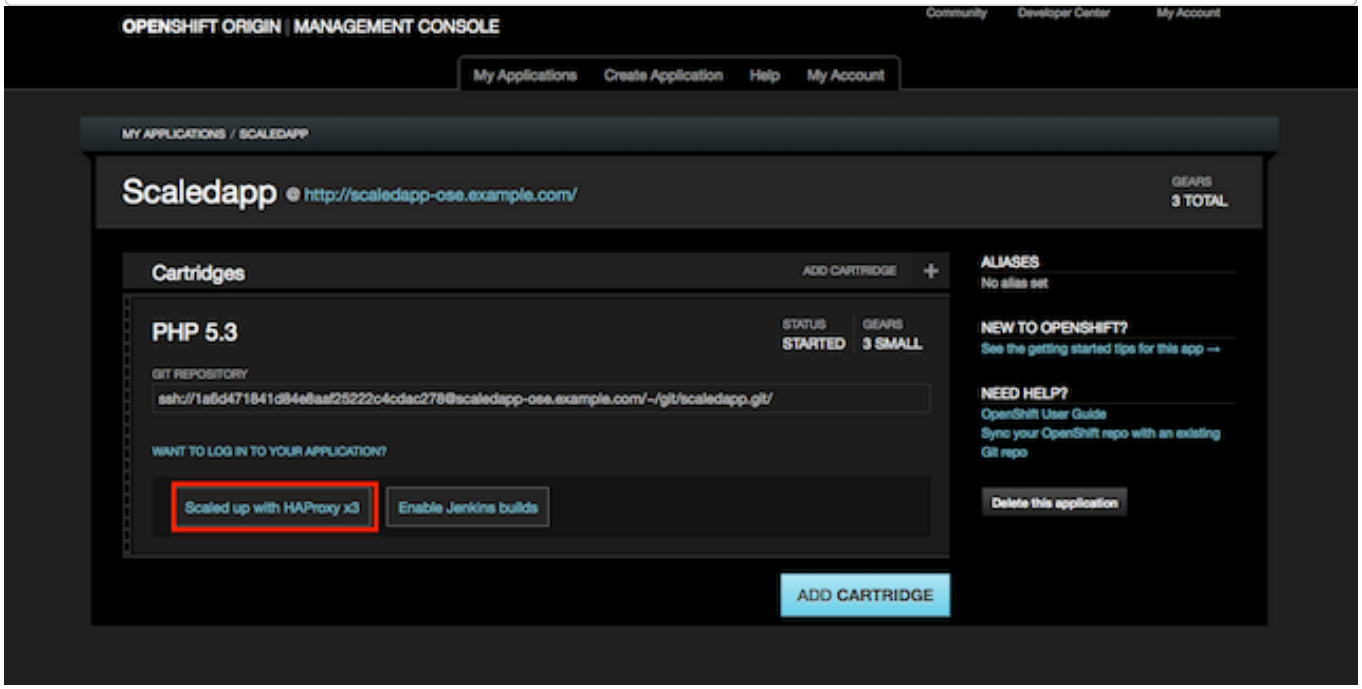
```
[scaledapp-ose.example.com ~]\> add-gear -a scaledapp -u 1a6d471841d84e8aaf25222c4cdac278 -n ose
```

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

```
$ rhc app show scaledapp
```

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

```
scaledapp @ http://scaledapp-ose.example.com/
============================================
 Application Info
 ================
  SSH URL   = ssh://1a6d471841d84e8aaf25222c4cdac278@scaledapp-ose.example.com
  Gear Size = small
  Git URL   = ssh://1a6d471841d84e8aaf25222c4cdac278@scaledapp-ose.example.com/~/git/scaledap-
p.git/
  Created   = 4:20 PM
  UUID      = 1a6d471841d84e8aaf25222c4cdac278
 Cartridges
 ==========
  php-5.3
  haproxy-1.4
 Scaling Info
 ============
  Scaled x3 (minimum: 2, maximum: available gears) with haproxy-1.4 on small gears
```



Just as we scaled up with the *add-gear* command, we can manually scale down with the *remove-gear* command. Remove the third 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 1a6d471841d84e8aaf25222c4cdac278 -n ose
```

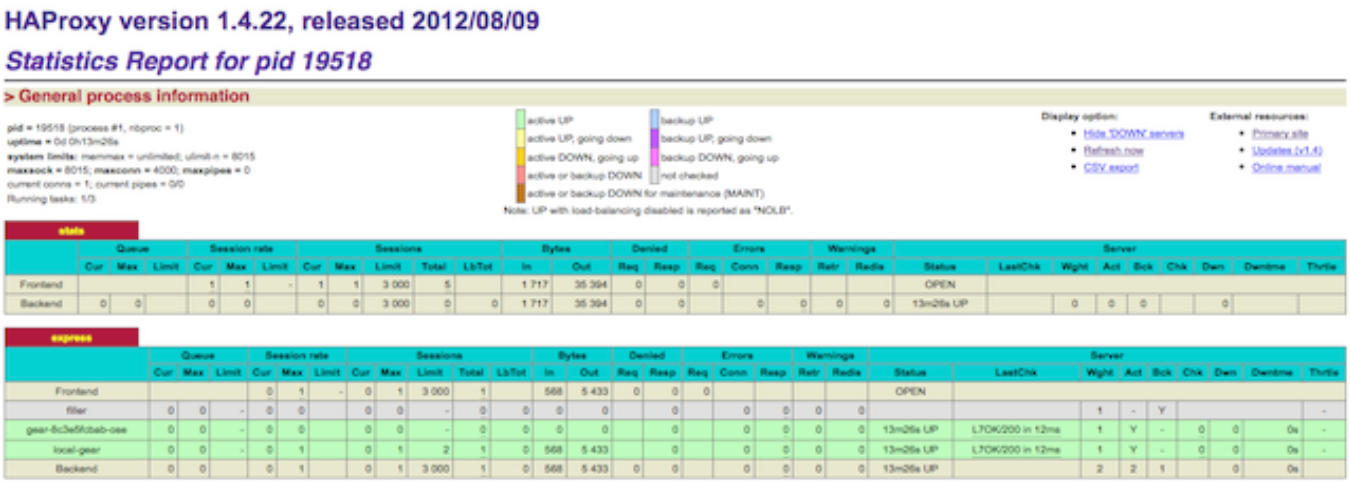
After removing the gear with the *remove-gear* command, verify that the application only contains two gears, HAProxy and a single runtime gear:

```
$  rhc app show scaledapp

scaledapp @ http://scaledapp-ose.example.com/
=============================================
 Application Info
 ================
  Created   = 4:20 PM
  Gear Size = small
  SSH URL   = ssh://1a6d471841d84e8aaf25222c4cdac278@scaledapp-ose.example.com
  Git URL   = ssh://1a6d471841d84e8aaf25222c4cdac278@scaledapp-ose.example.com/~/git/scaledap-p.git/
  UUID      = 1a6d471841d84e8aaf25222c4cdac278
 Cartridges
 ==========
  php-5.3
  haproxy-1.4
 Scaling Info
 ============
  Scaled x2 (minimum: 2, maximum: available gears) with haproxy-1.4 on small gears
```

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.example.com/haproxy-status/
```



HAProxy version 1.4.22, released 2012/08/09

Statistics Report for pid 19518

> General process information

pid = 19518 (process #1, nbproc = 1)
uptime = 0d 0h13m26s
system limits: memmax = unlimited; ulimit-n = 8015
maxsock = 8015; maxconn = 4000; maxpipes = 0
current conns = 1; current pipes = 0/0
Running tasks: 1/3

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

Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.4)
- Online manual

stats

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	1	-	1	1	3 000	5		1 717	35 394	0	0	0					OPEN								
Backend	0	0		0	0		0	0	3 000	0	0	1 717	35 394	0	0		0	0	0	0	13m26s UP		0	0	0		0		

express

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	1	3 000	1		568	5 433	0	0	0					OPEN								
filler	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0			1	-	Y				-
gear-8c3e5fcbab-cee	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	13m26s UP	L7OK/200 in 12ms	1	Y	-	0	0	0s	-
local-gear	0	0	-	0	1		0	1	2	1	0	568	5 433		0		0	0	0	0	13m26s UP	L7OK/200 in 12ms	1	Y	-	0	0	0s	-
Backend	0	0		0	1		0	1	3 000	1	0	568	5 433	0	0		0	0	0	0	13m26s UP		2	2	1		0	0s	

Lab 26 Complete!

Lab 27: The DIY application type (Estimated time: 10 minutes)

Server used:

- localhost

Tools used:

- rhc
- git

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

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

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

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

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

Creating a DIY application type

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

```
$ rhc app create -a myjavademo -t diy
$ cd diy
```

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.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 2>&1 &

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

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

Lab 27 Complete!

Lab 28: Developing Java EE applications using JBoss EAP (Estimated time: 30 minutes)

Server used:

- localhost

Tools used:

- rhc
- git
- curl

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

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

Create a JBoss EAP application

Note: Before starting this lab, it is suggested that you delete any existing applications that you have deployed to your OpenShift Enterprise installation. The OpenStack virtual machines that have been provided for this training class only contain 2gb of memory.

```
$ rhc app create -a todo -t jbosseap
```

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

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

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

Additional marker files for JBoss EAP

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

- enable_jpda - Will enable the JPDA socket based transport on the JVM running the JBoss EAP application server. This enables you to remotely debug code running inside of the JBoss application server.
- skip_maven_build - Maven build step will be skipped
- force_clean_build - Will start the build process by removing all non essential Maven dependencies. Any current dependencies specified in your pom.xml file will then be re-downloaded.
- hot_deploy - Will prevent a JBoss container restart during build/deployment. Newly built archives will be re-deployed automatically by the JBoss 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 applications root directory in order to build the code remotely.

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

Embed PostgreSQL cartridge

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

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.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    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-source>
        <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://java.sun.-
com/xml/ns/javaee http://jboss.org/schema/cdi/beans_1_0.xsd"/>
```

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 of OpenShift Enterprise","tags":["javascript","ui"]}' https://todo-ose.example.com/rest/todos
```

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

```
$ curl -k -i -H "Accept: application/json" https://todo-ose.example.com/rest/todos/1
```

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"],"createdOn":1359086546955}
```

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.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 28 Complete!

Lab 29: Using Jenkins continuous integration (Estimated time: 30 minutes)

Server used:

- localhost
- node host

Tools used:

- rhc
- git
- yum

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

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

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

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

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

Verify Jenkins cartridges are installed

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

```
# rpm -qa |grep jenkins
```

You should see the following four packages installed:

- openshift-origin-cartridge-jenkins–1.4–1.0.1–1.el6op.noarch
- jenkins–1.488–2.el6op.noarch
- openshift-origin-cartridge-jenkins-client–1.4–1.0.1–1.el6op.noarch
- jenkins-plugin-openshift–0.6.5–0.el6op.x86_64

If you do now have the above RPM packages installed on your node host, follow the directions in lab 17 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 to contains the Jenkins application. This is done using the *rhc app create* command line tool, or you can use the web console to create the application. The syntax for using the command line tool is as follows:

```
$ rhc app create -a jenkins -t jenkins
```

You should see the following output from this command:

```
Creating application 'jenkins'
==============================

  Gear Size: default
  Scaling:   no
  Cartridge: jenkins
  Namespace: ose

Your application's domain name is being propagated worldwide (this might take a minute)...

Cloning into 'jenkins'...
done

jenkins @ http://jenkins-ose.example.com/
=========================================
  Application Info
```

```
  ================
    SSH URL   = ssh://4437d81168c94baf9268f0592bbe31a9@jenkins-ose.example.com
    Git URL   =
ssh://4437d81168c94baf9268f0592bbe31a9@jenkins-ose.example.com/~/git/jenkins.git/
    UUID      = 4437d81168c94baf9268f0592bbe31a9
    Gear Size = small
    Created   = 2:05 PM
  Cartridges
  ==========
    jenkins-1.4

RESULT:
Application jenkins was created.

Jenkins created successfully.  Please make note of these credentials:

   User: admin
   Password: QKVn_1ZlQ7T_

Note:  You can change your password at: https://jenkins-ose.example.com/me/configure
```

Make a note of the user username and password that was created for you by OpenShift Enterprise.

Adding Jenkins support to your application

Now that we have a Jenkins server setup and running, we can add support to our *todo* application which will allow all futures builds to compile on the Jenkins server. To embed the Jenkins support cartridge in your application, use the following command:

```
$ rhc cartridge add -a todo -c jenkins-client
```

The output should be the following:

```
Adding 'jenkins-client-1.4' to application 'todo'
Success
jenkins-client-1.4
==================
 Properties
 ==========
  Job URL = https://jenkins-ose.example.com/job/todo-build/
```

Verify that the Jenkins client was added to your application by running the following command:

```
$ rhc app show todo
```

At the bottom of the output, you should see the following information:

```
 Cartridges
 ==========
  jenkins-client-1.4 = https://jenkins-ose.example.com/job/todo-build/
  postgresql-8.4     = postgresql://127.1.248.129:5432/
  jbosseap-6.0
```

Configuring Jenkins

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

```
https://jenkins-ose.example.com/job/todo-build/
```

Authenticate to the Jenkins environment by providing the username and password that was displayed after adding the Jenkins application.

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

A 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 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 -DskipTests

# Deploy new build

# Stop app
jenkins_stop_app 4d1b096e414243e9833dad55d774de73@todo-ose.example.com

# Push content back to application
jenkins_sync_jboss 4d1b096e414243e9833dad55d774de73@todo-ose.example.com

# Configure / start app
$GIT_SSH 4d1b096e414243e9833dad55d774de73@todo-ose.example.com deploy.sh

jenkins_start_app 4d1b096e414243e9833dad55d774de73@todo-ose.example.com

$GIT_SSH 4d1b096e414243e9833dad55d774de73@todo-ose.example.com post_deploy.sh
```

Deploying code to Jenkins

Now that you have the Jenkins client embedded into your *todo* application gear, any future *git push* commands will send the code to the Jenkins server for building. To test this out, edit the

src/main/webapp/todo.xhtml source file and change the title of the page. If you do not have this file, just create a new file instead. Look for the following code block:

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

Change the above code to the following:

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

Commit and push your change:

```
$ git commit -am “changed h2”
$ git push
```

After you push your changes to the Jenkins server, you should see the following output:

```
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 282 bytes, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: restart_on_add=false
remote: Executing Jenkins build.
remote:
remote: You can track your build at https://jenkins-ose.example.com/job/todo-build
remote:
remote: Waiting for build to schedule....Done
remote: Waiting for job to complete.....Done
remote: SUCCESS
remote: New build has been deployed.
To ssh://4d1b096e414243e9833dad55d774de73@todo-ose.example.com/~/git/todo.git/
  eb5f9dc..8cee826  master -> master
```

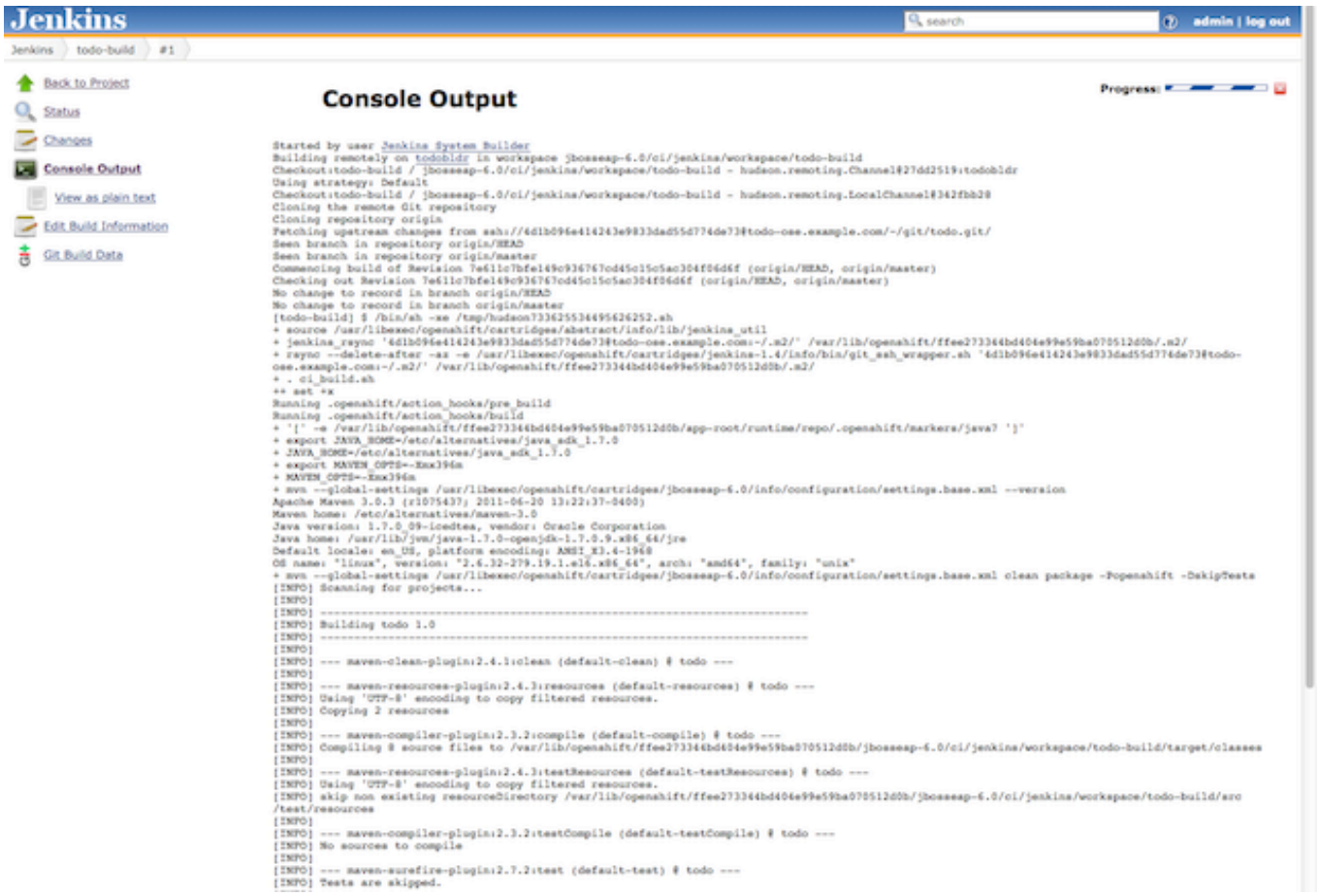
While the build is happening, open up a new terminal window and run the following command:

```
$ 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.example.com/
==========================================
  Application Info
  ================
    UUID      = ffee273344bd404e99e59ba070512d0b
    Git URL   = ssh://ffee273344bd404e99e59ba070512d0b@todobldr-ose.example.com/~/git/todobldr.git/
    SSH URL   = ssh://ffee273344bd404e99e59ba070512d0b@todobldr-ose.example.com
    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.

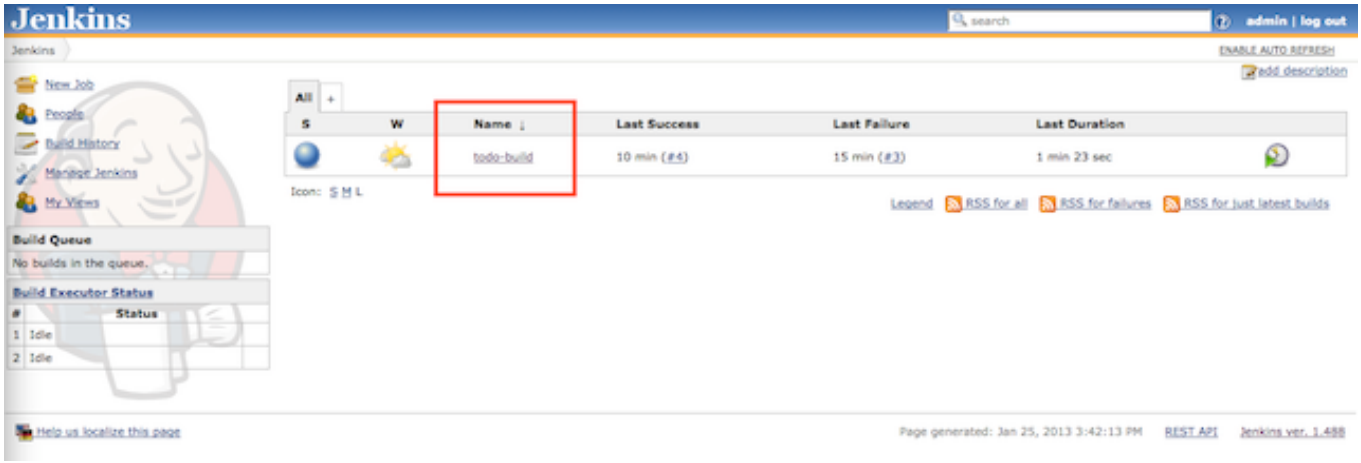


Starting a new build

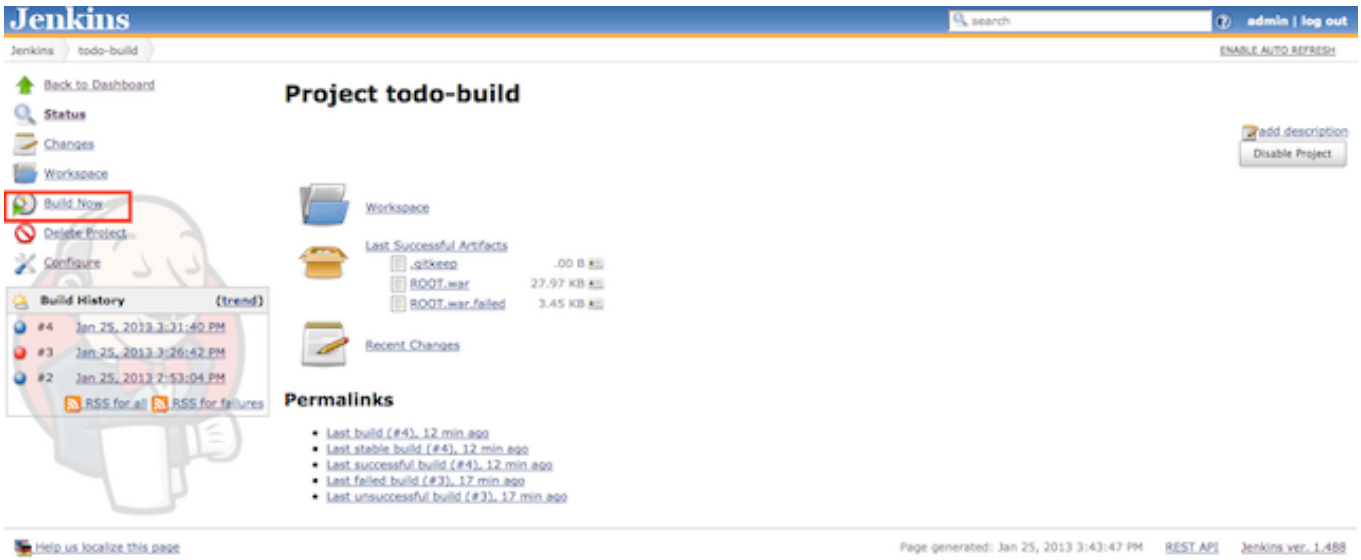
One of the great things about integrating your application with the Jenkins CI environment is the ability to start a new build without having to modify and push your source code. To initiate a new build, log in to the Jenkins dashboard and select the *todo* builder. Point your browser to:

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

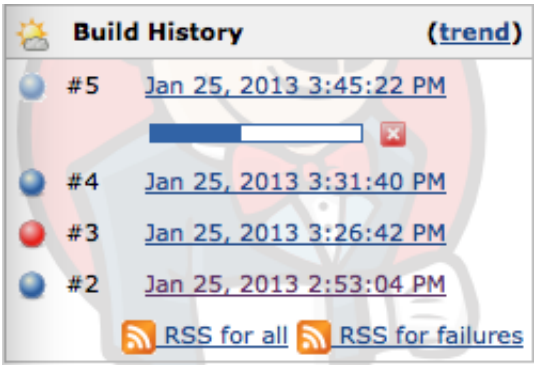
Once you have been authenticated, click the *todo-build* link:



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 29 Complete!

Lab 30: Using JBoss Tools (Estimated time: 30 minutes)

Server used:

- localhost

Tools used:

- eclipse

JBoss Tools is an umbrella project for a set of Eclipse plugins that supports JBoss and related technologies; there is support for OpenShift, Hibernate, JBoss AS, Drools, jBPM, JSF, (X)HTML, Seam, Maven, JBoss ESB, JBoss Portal and more…

Download and install Eclipse - Juno

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 Juno 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.

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 4.0 (Juno).

Using JBoss Tools and OpenShift Enterprise

By default, JBoss Tools OpenShift integration will default to use the OpenShift Online service that is hosted by Red Hat. In order for us to use JBoss Tools to communicate with our OpenShift Enterprise installation, we need to configure Eclipse with our *LIBRA_SERVER* setting. This is a straight forward process but one that may catch people off guard. To make the change, edit the *eclipse.ini* file located in the root directory of your Eclipse deployment and add *-Dlibra_server=broker.example.com* directly after the the following line:

```
-vmargs
```

When finished, it should look like this:

```
-vmargs
-Dlibra_server=broker.example.com
```

Note: If you are using the Mac OS X operating system, the process is a little more complicated. In order to pass arguments to Eclipse, you'll have to edit the *eclipse.ini* file inside the Eclipse application bundle:

- Select the Eclipse application bundle icon while holding down the Control Key
- This will present you with a popup menu. Select "Show Package Contents" in the popup menu.
- Locate eclipse.ini file in the Contents/MacOS sub-folder and open it with your favorite text editor to edit the command line options.

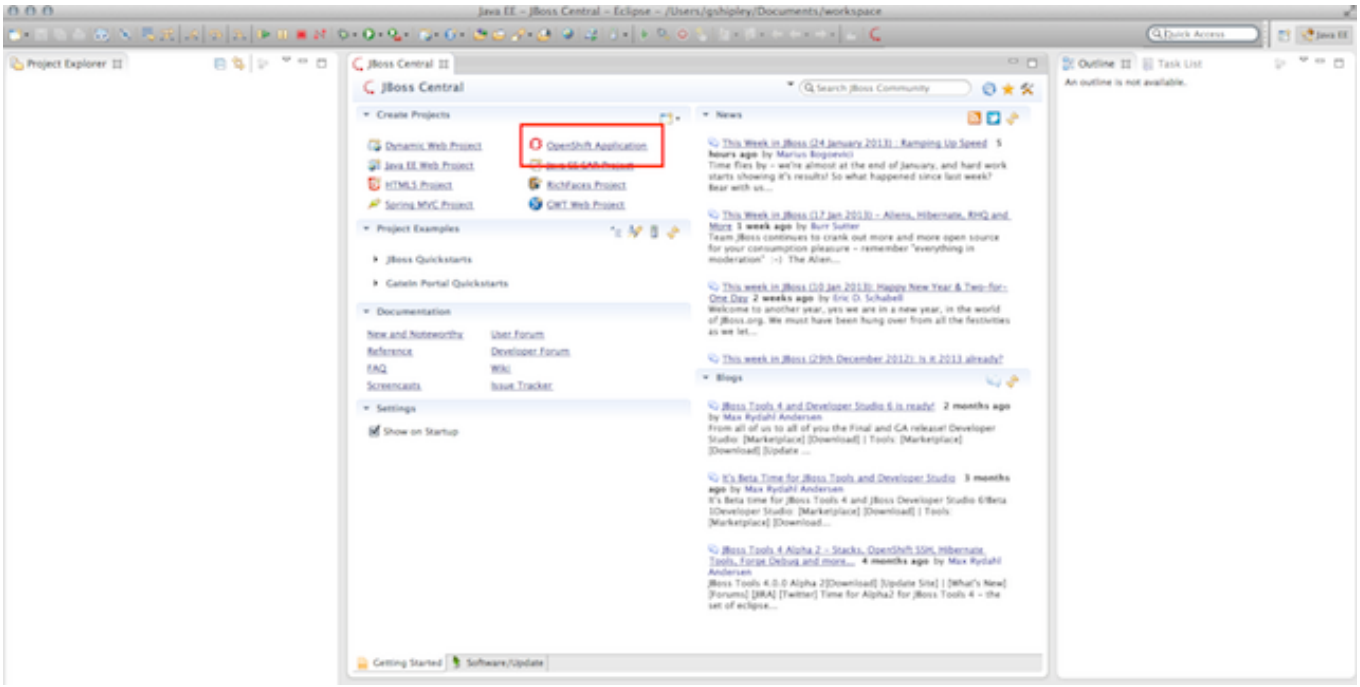
On my OS X operating system, after editing the *eclipse.ini* file it looks like following:

```
-startup
../../../plugins/org.eclipse.equinox.launcher_1.3.0.v20120522-1813.jar
--launcher.library
../../../plugins/org.eclipse.equinox.launcher.cocoa.macosx.x86_64_1.1.200.v20120522-1813
-product
org.eclipse.epp.package.jee.product
--launcher.defaultAction
openFile
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
-Dlibra_server=broker.example.com
-Dosgi.requiredJavaVersion=1.5
-Dhelp.lucene.tokenizer=standard
-XstartOnFirstThread
-Dorg.eclipse.swt.internal.carbon.smallFonts
-XX:MaxPermSize=256m
-Xms40m
-Xmx512m
-Xdock:icon=../Resources/Eclipse.icns
-XstartOnFirstThread
-Dorg.eclipse.swt.internal.carbon.smallFonts
```

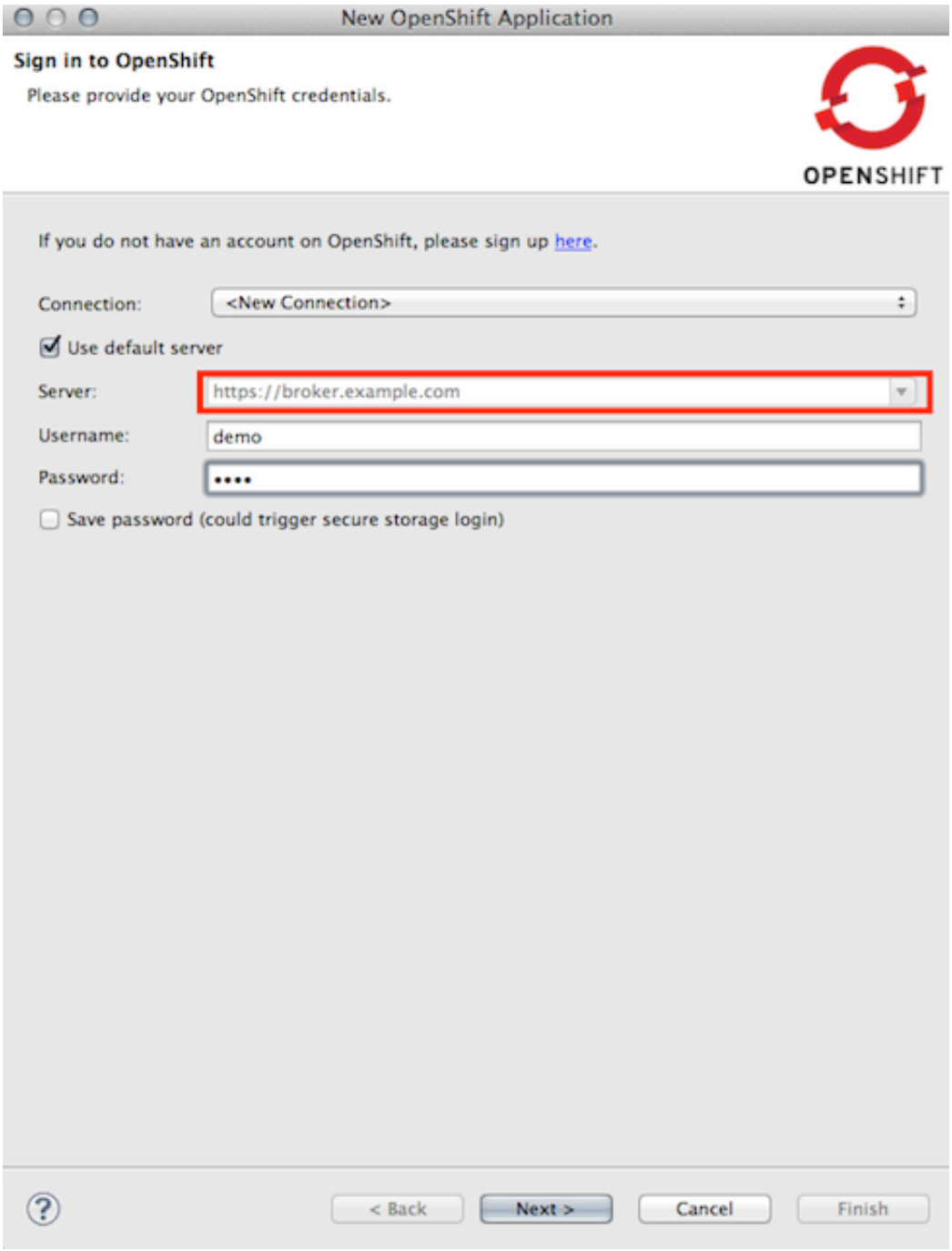
After adding the correct arguments to the *eclipse.ini* file, restart the Eclipse IDE.

Create an OpenShift Enterprise application

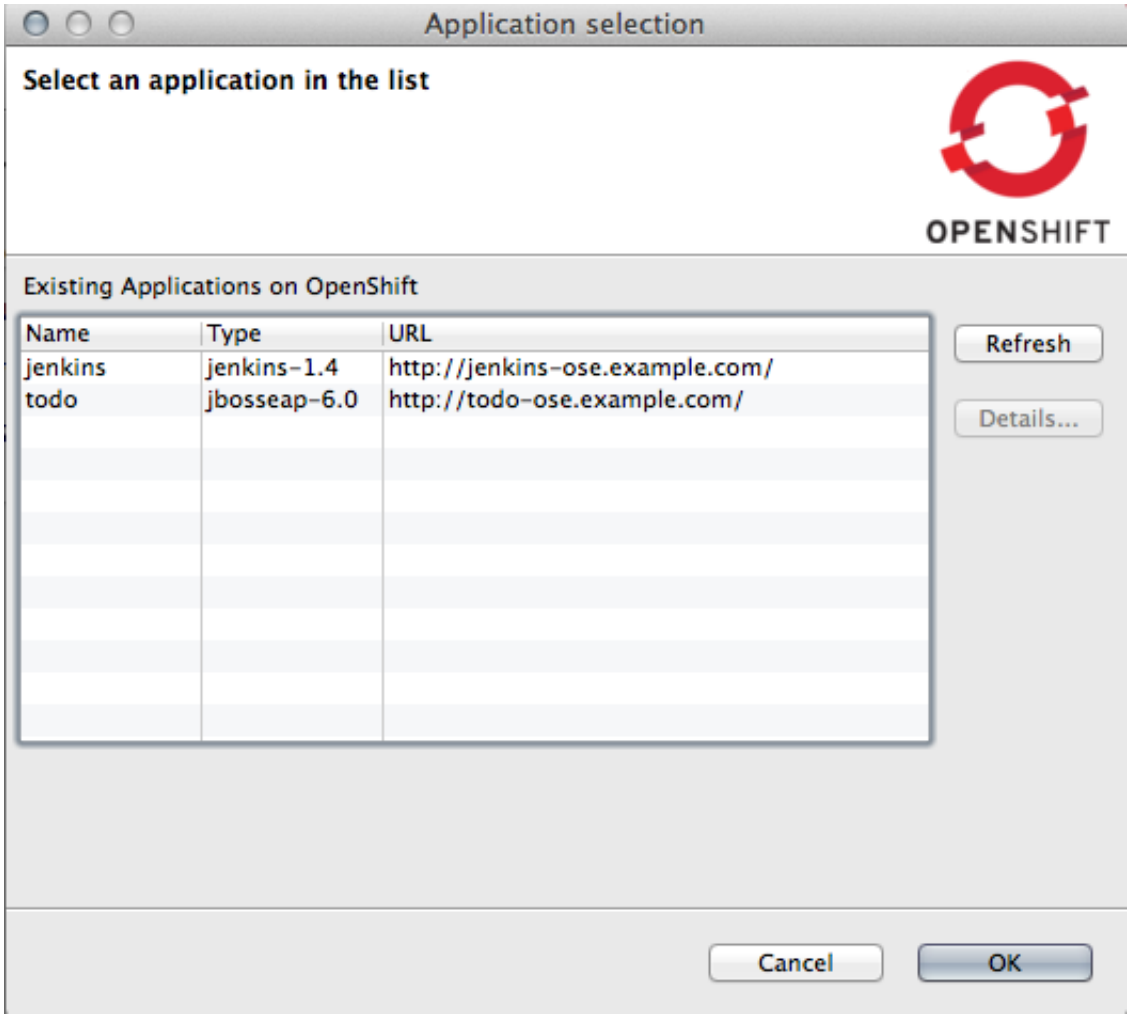
Now that we have Eclipse Juno and JBoss Tools 4.0 installed, we can create an OpenShift Enterprise application without having the leave the comfort of our favorite IDE. Click on the *OpenShift Applicaiton* link that is provided on the JBoss Central screen.



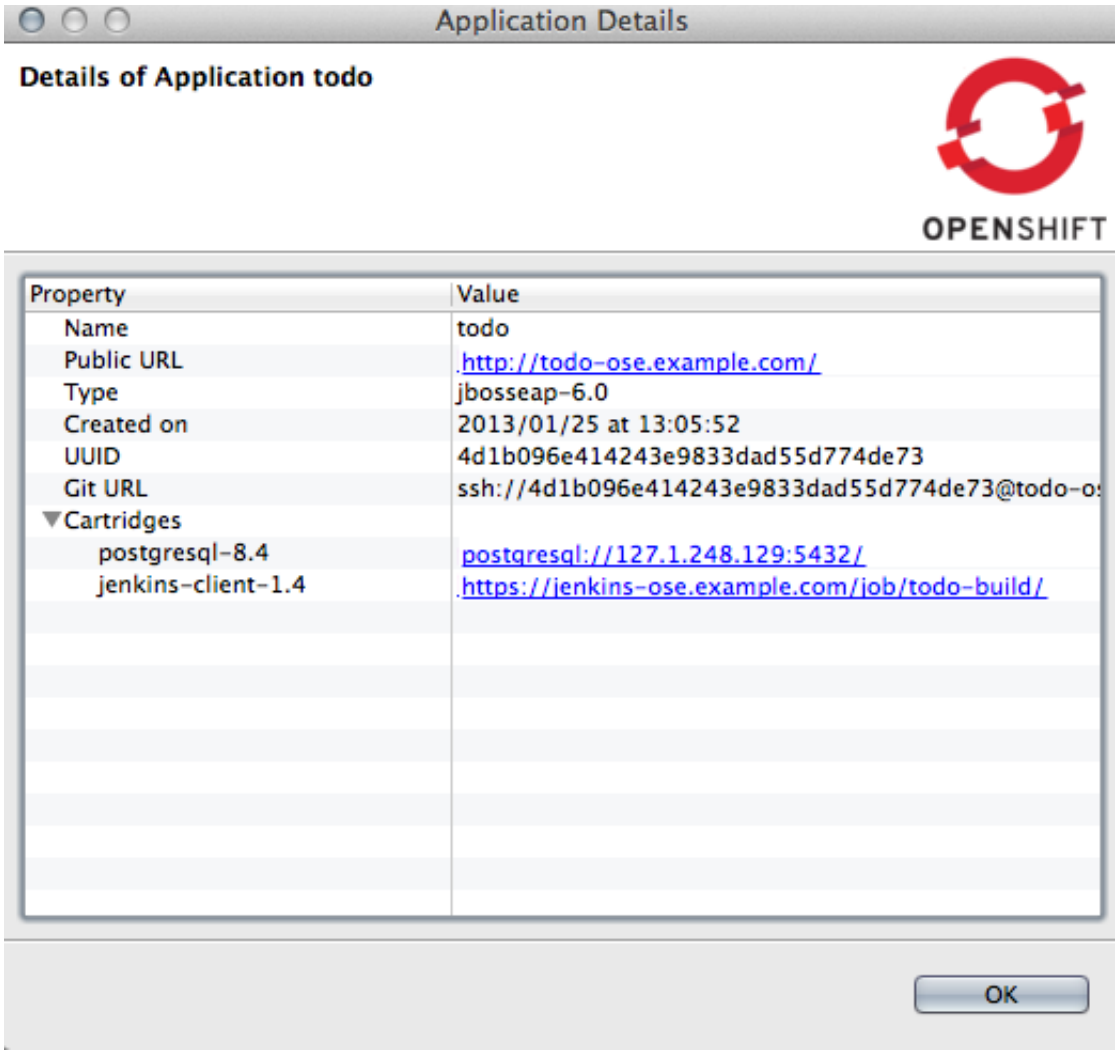
Once you click on the link to create a new OpenShift Enterprise application, you will be presented with a dialog to authenticate to OpenShift Enterprise. Now is also a good time to validate the *Server* setting is correctly set to *broker.example.com*. If your server does not reflect this, you have not configured your *eclipse.ini* file correctly. If you are unable to configure your *eclipse.ini* file as specified in this lab, inform the instructor so that he/she may help you.



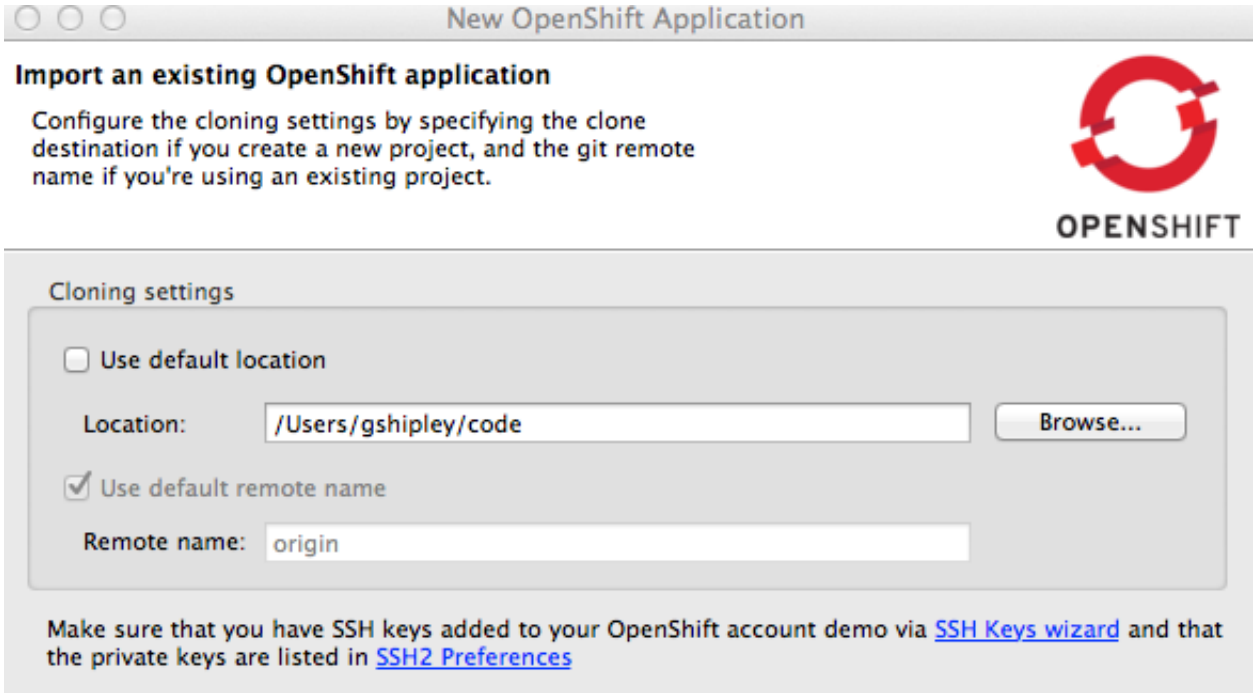
After clicking *next*, the JBoss Tools plugin will authenticate you to the broker host and present another dialog box to you. On this dialog box, you have the option of creating a new application, or to use an existing one. Since we already have a JBoss EAP application deployed, let's select to *Use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



Highlight the *todo* application and click on the *Details…* button. This will display all of the necessary information about the application, including any cartridges that may be embedded.



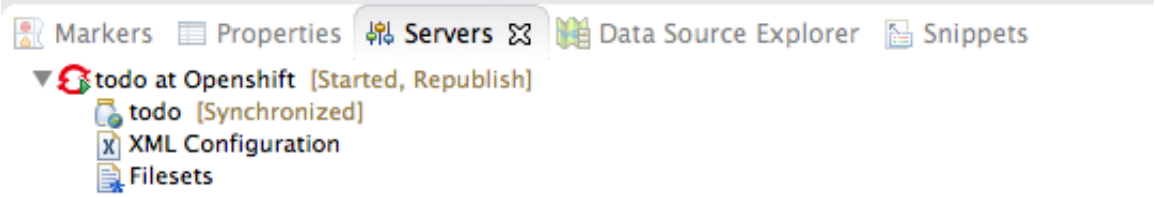
After clicking *Next*, Eclipse will ask you to create a new project or to use an existing one. Let's create a new one and set the correct location where we want to store the project files.



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

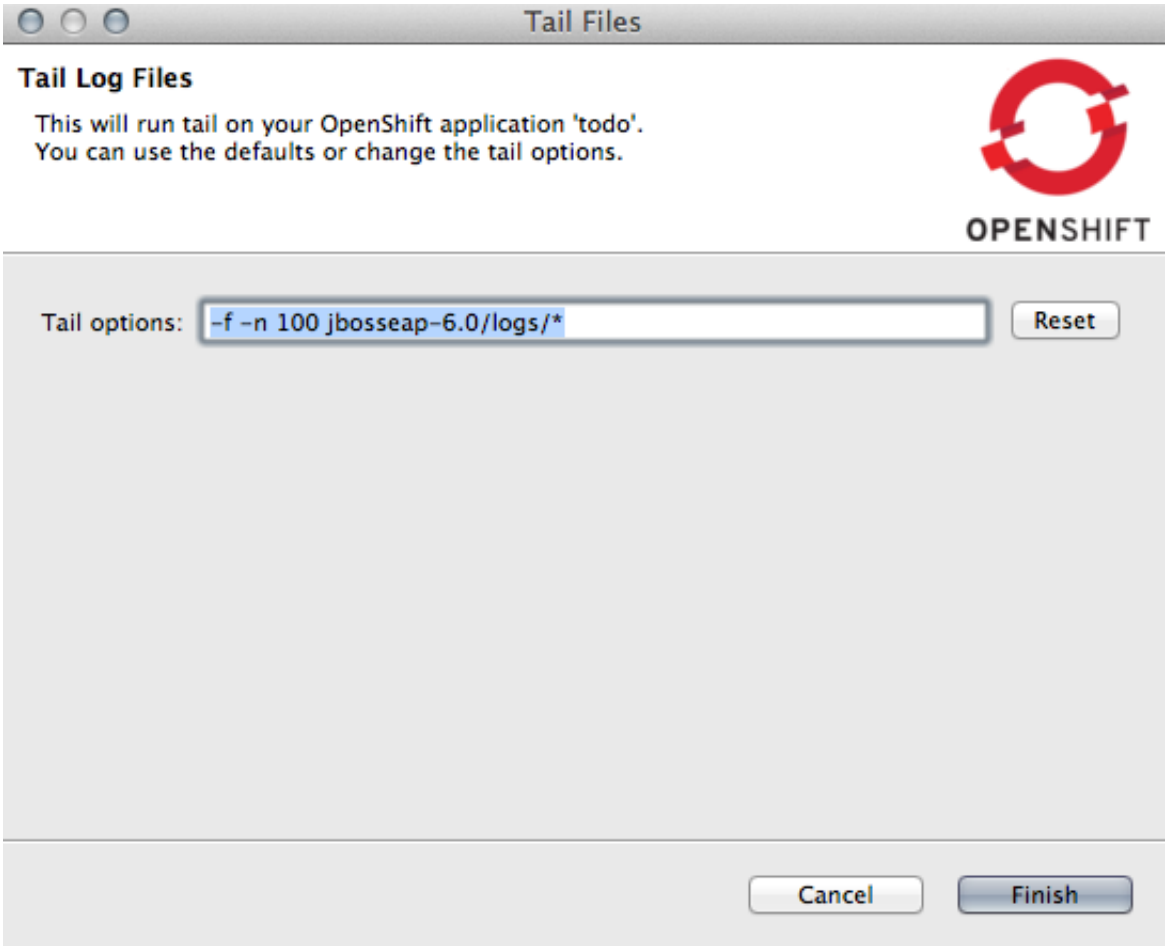
Managing OpenShift Enterprise application with JBoss Tools

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

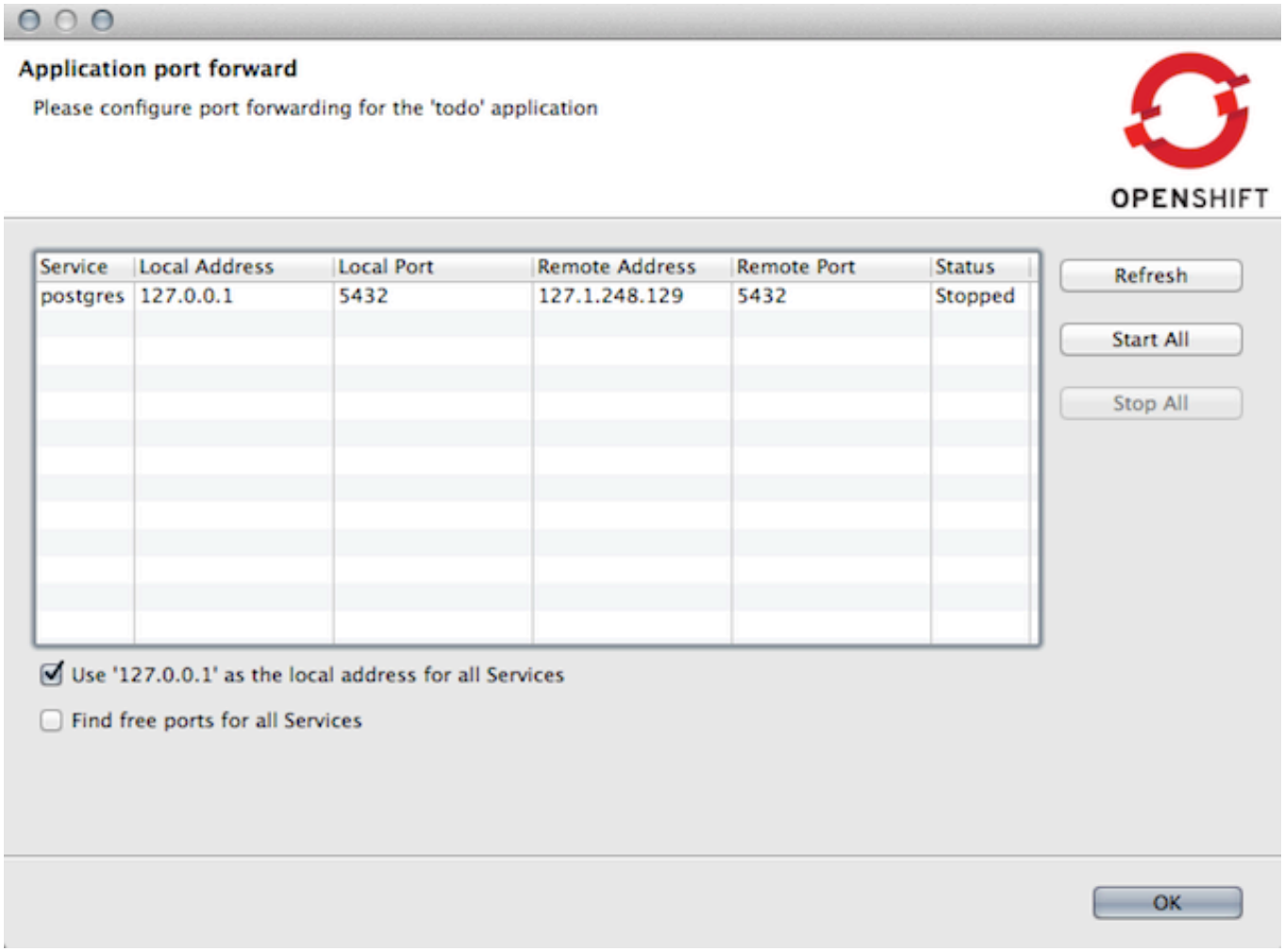
Eclipse.

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.



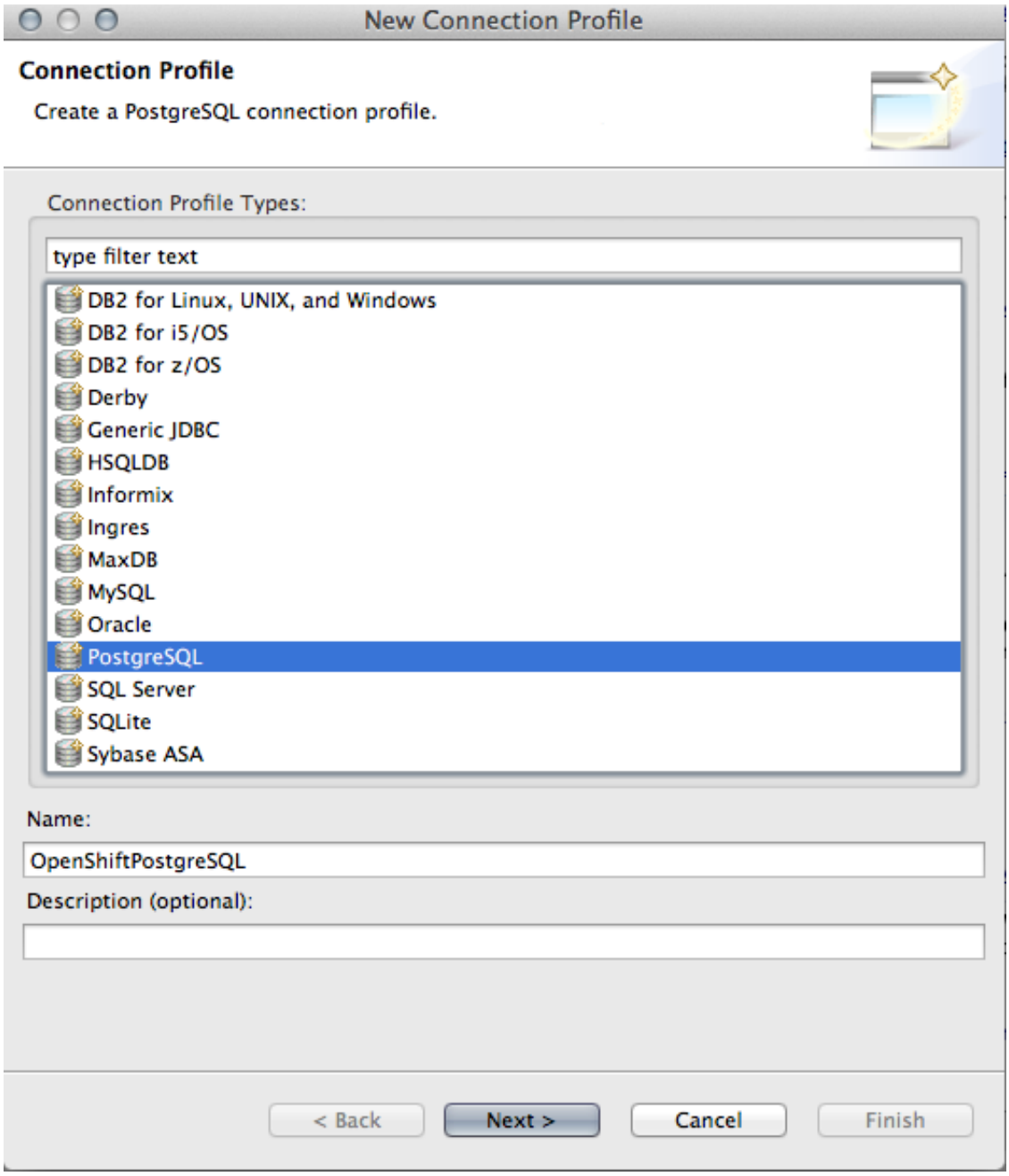
For the next section of this lab, ensure that you click on *Start Forwarding* so that we will be able to connect to PostgreSQL from our local machine.

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 is setup correctly, press the *test connection* button. If your connection is failing, make sure that you have the correct authorization credentials and that port-fowarding is started via JBoss Tools.

Once you have correctly added the database connection, you should now see the remote database from the OpenShift Enterprise node host available for use in your Eclipse IDE.

At this point, you should be able to use any of the database tools provided by Eclipse to communicate with and manage your OpenShift Enterprise PostgreSQL database.

Making a code change and deploying the application

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

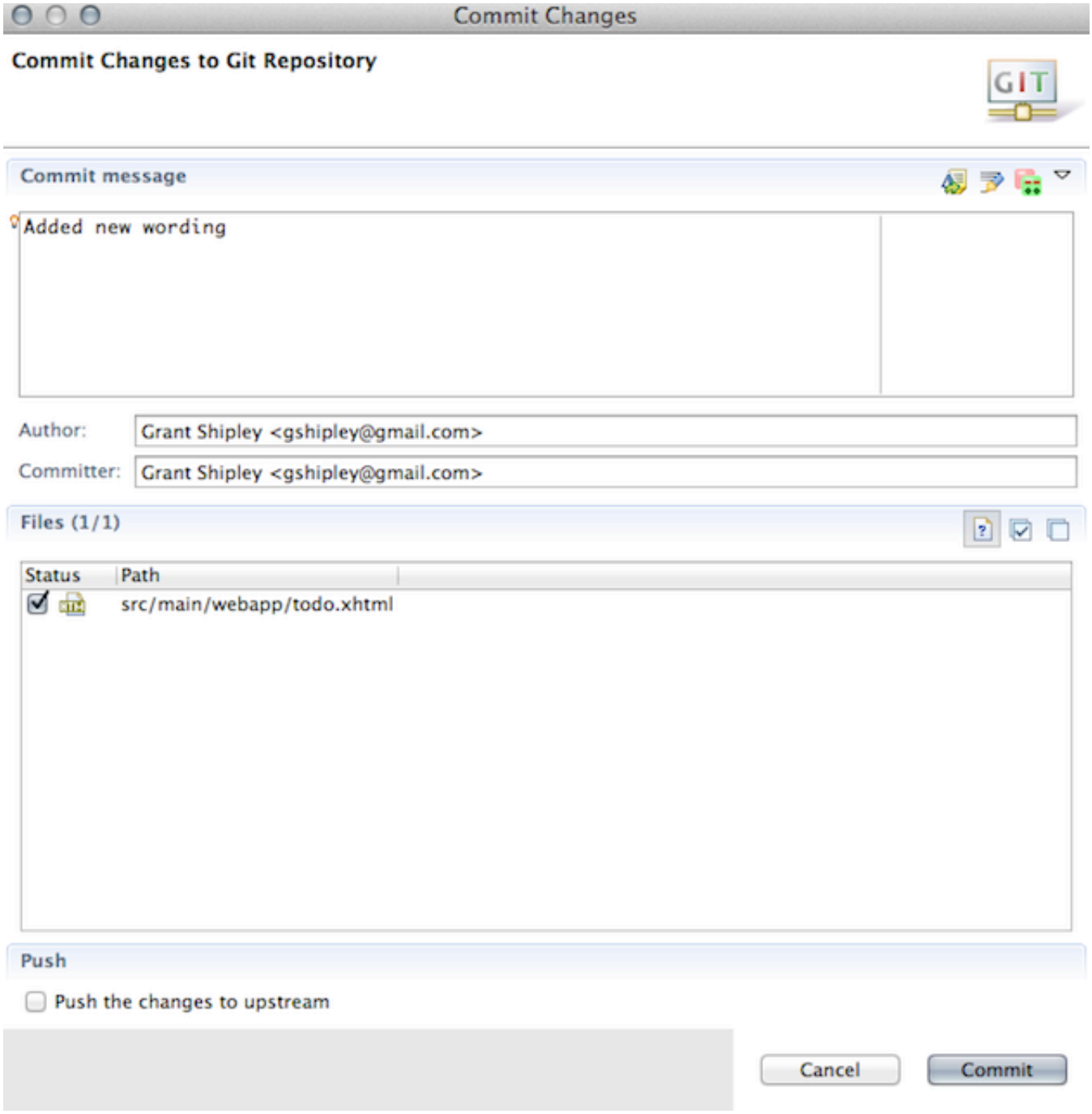
```
<h2>Todo List Creation using Jenkins</h2>
```

to the include JBoss Tools

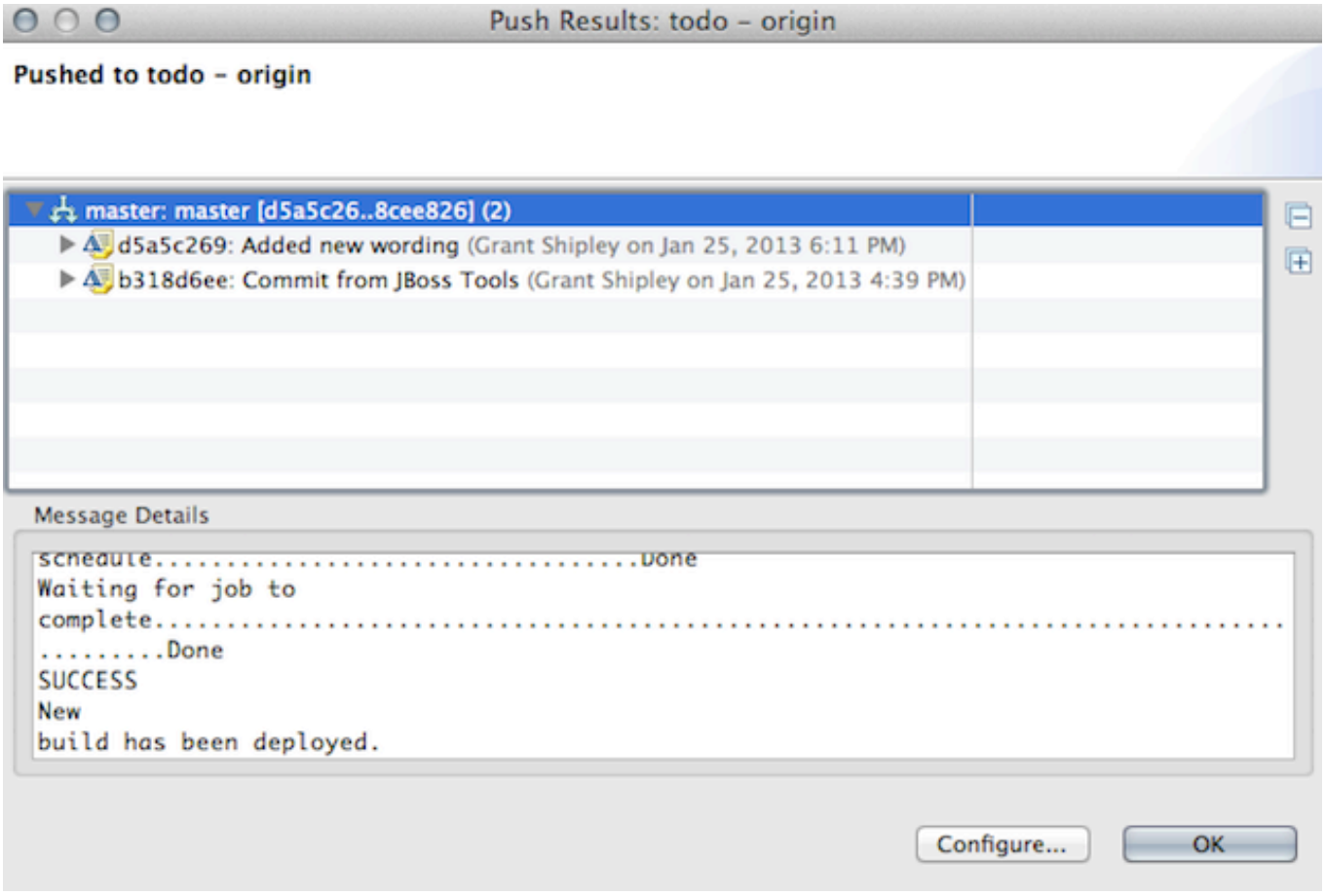
```
<h2>Todo List Creation using Jenkins and JBoss Tools</h2>
```

Once you have made the source code change, save the contents of the file and then use the *Team* functionality by right clicking on your project. Commit and push the changes to your OpenShift

Enterprise server. This push will follow the same workflow used previously by initiating a build on your Jenkins server.

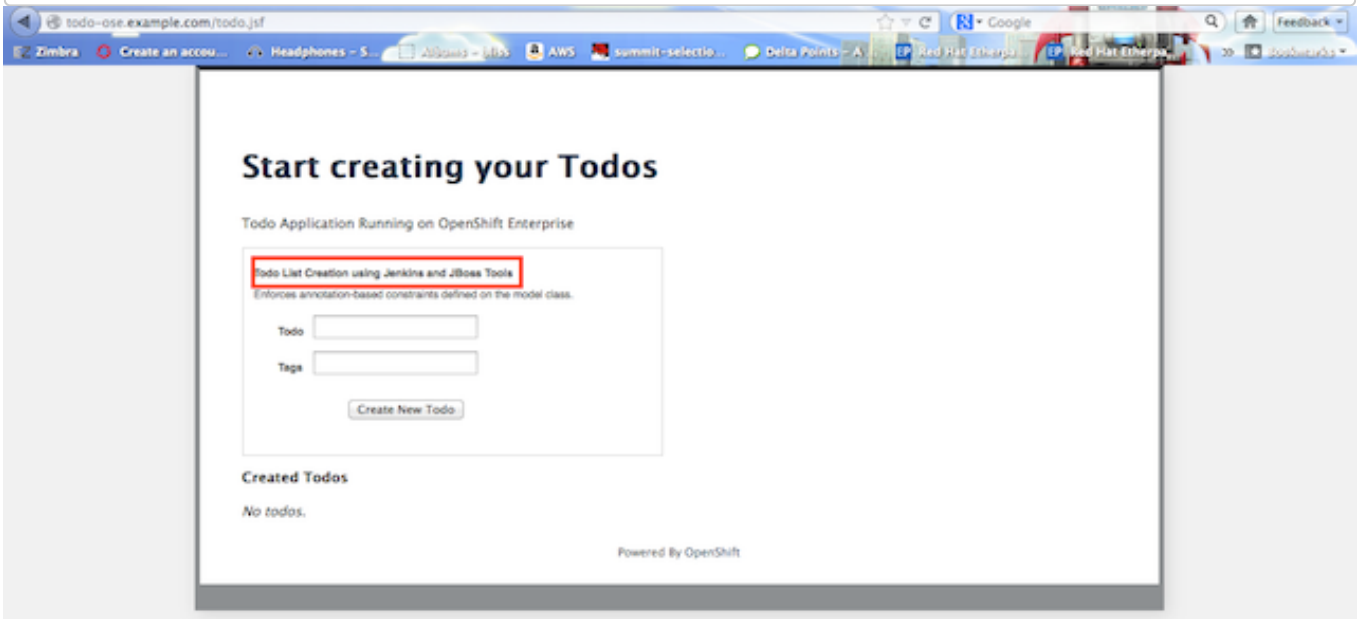


After you push your changes, open up your Jenkins dashboard and open the *Console Output* screen to see the build progress. Once your build has completed, Eclipse will display a dialog box with a summary of the deployment:



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

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



Lab 30 Complete!

Lab 31: Using quickstarts (Estimated time: 10 minutes)

Server used:

- localhost

Tools used:

- rhc
- git

A key tenant when Red Hat was designing OpenShift Enterprise was the ability for developers to be able to run their source code and application as is, without having to use proprietary API(s). To illustrate how easy it is for developers to get their existing application deployed on OpenShift Enterprise, the team has created a github space where they provide numerous quick start projects that make deploying common open source applications to the platform a painless task. Some of the popular open source projects the team provides a quick start for are:

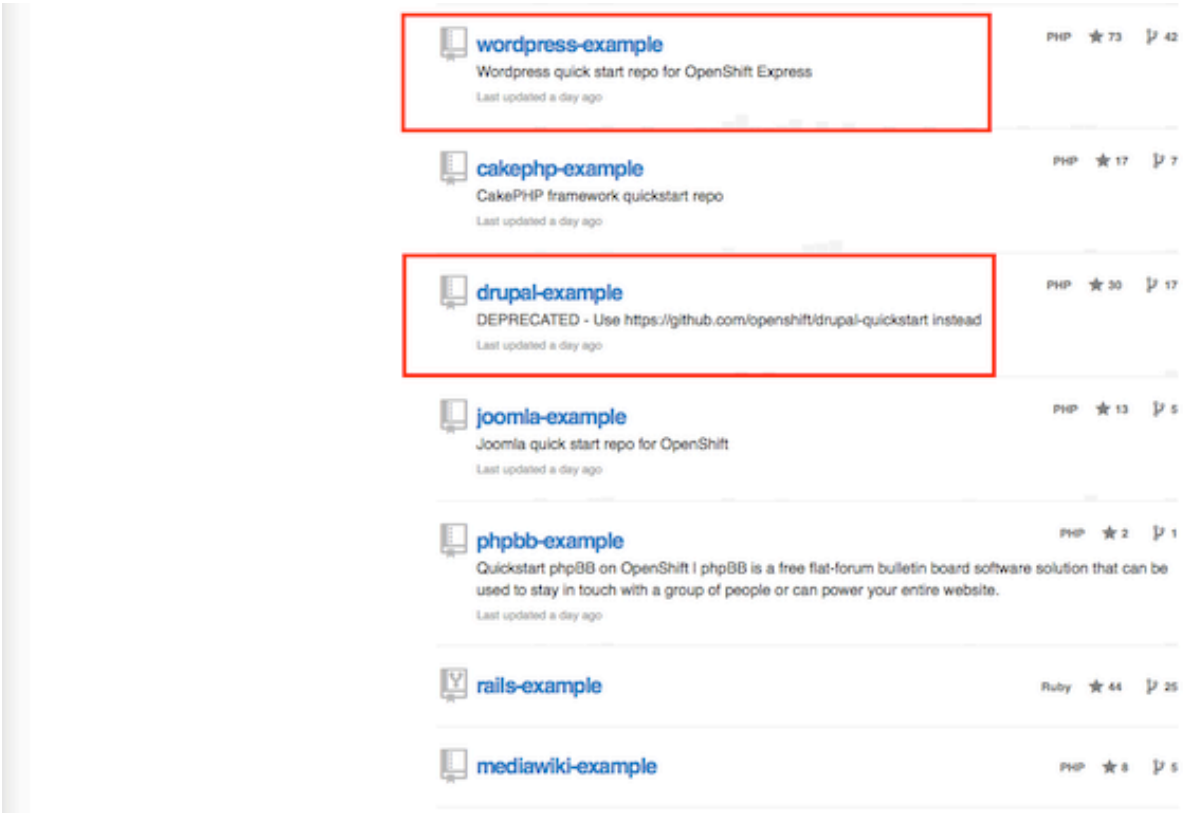
- Drupal
- Review Board
- Wordpress
- Frog CMS
- Sugar CRM
- Redmine
- MediaWiki

Install a quickstart

Point your browser to the following URL:

```
http://www.github.com/openshift
```

Given the number of available quick starts, you may have to use the search functionality of your browser to locate the quick start that you would like to install. For this lab, choose either the Wordpress or Drupal quick start and follow the instructions provided to install the application.



Lab 31 Complete!

Lab 32: Creating a quick start (Estimated time: 30 minutes)

Server used:

- localhost
- node host

Tools used:

- rhc
- git
- github

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

```
$ cd ~
$ mkdir code
$ mkdir piwikstage
$ unzip latest.zip
```

This will create a piwik directory under the ~/code/piwikstage directory.

Create an OpenShift Enterprise application

We need to create an OpenShift Enterprise application to hold the source code as well as embed the MySQL database:

```
$ cd ~/code
$ rhc app create-a piwik -t php
$ rhc cartridge add -a piwik -c mysql
```

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 applciation:

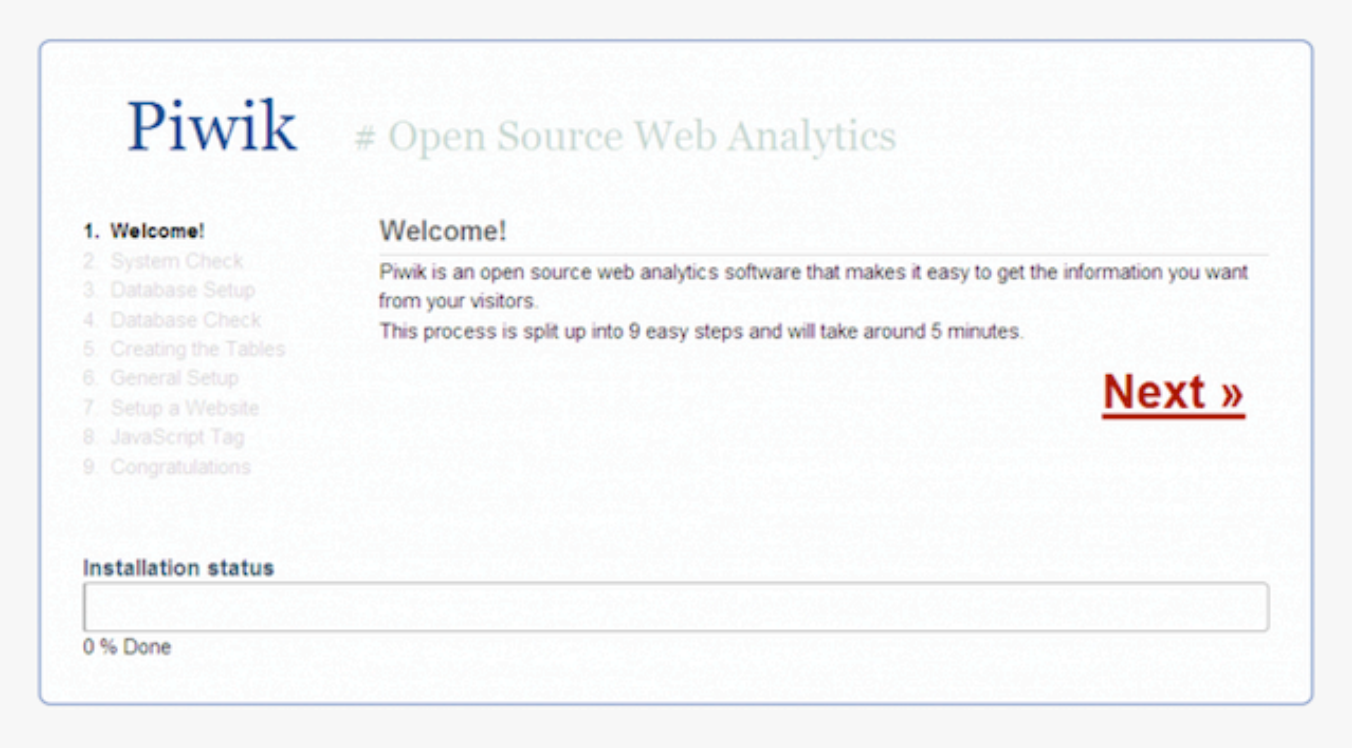
```
$ cp –av ~/code/piwikstage/piwik/* ~/code/piwik/php
```

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

```
$ cd ~/code/piwik/php
$ git add .
$ git commit –am “Initial commit for Piwik”
$ git push
```

Assuming everything went as expected, you should be able to verify Piwik is running by opening up your web browser and pointing to the following URL:

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



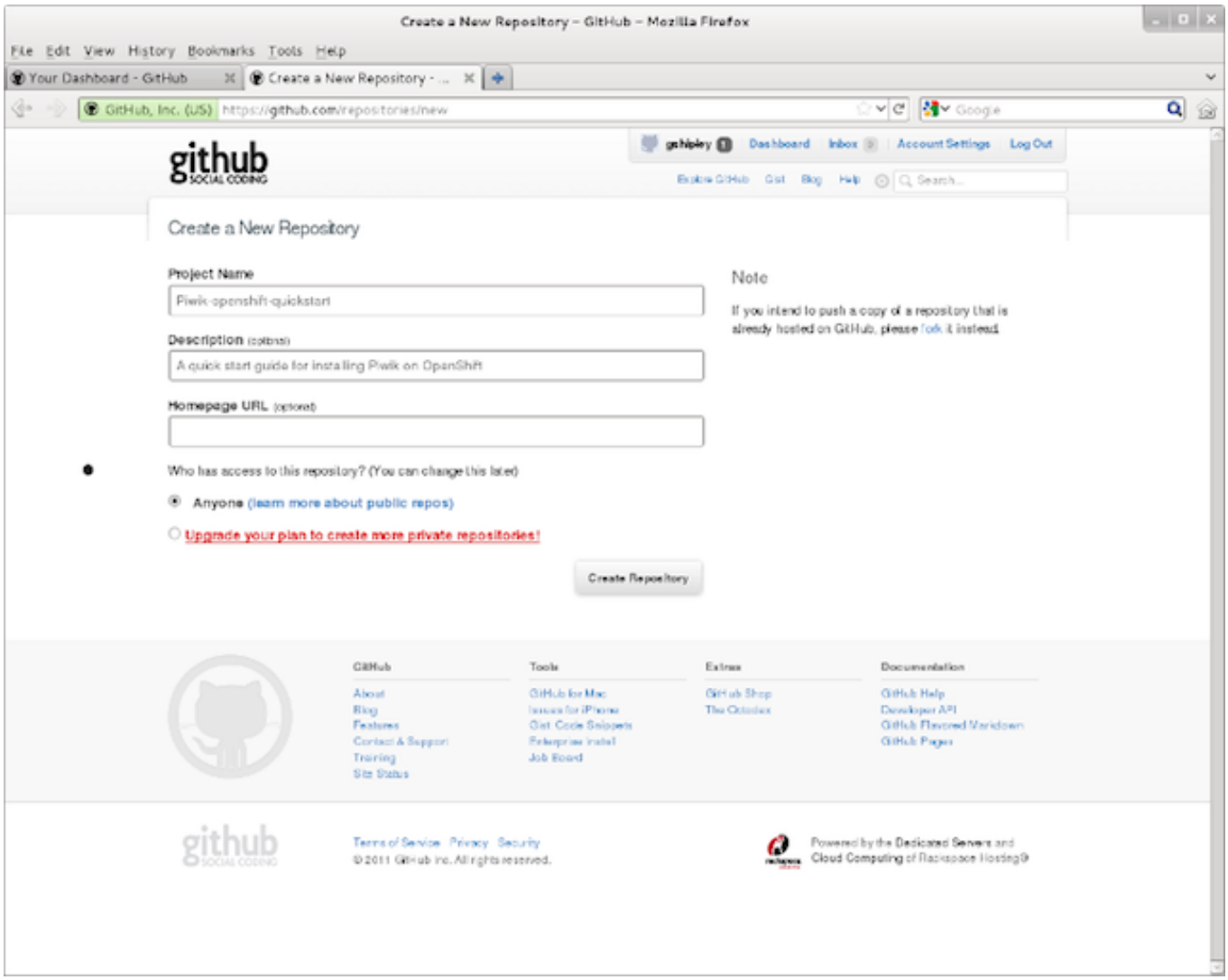
Creating a github repository

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).

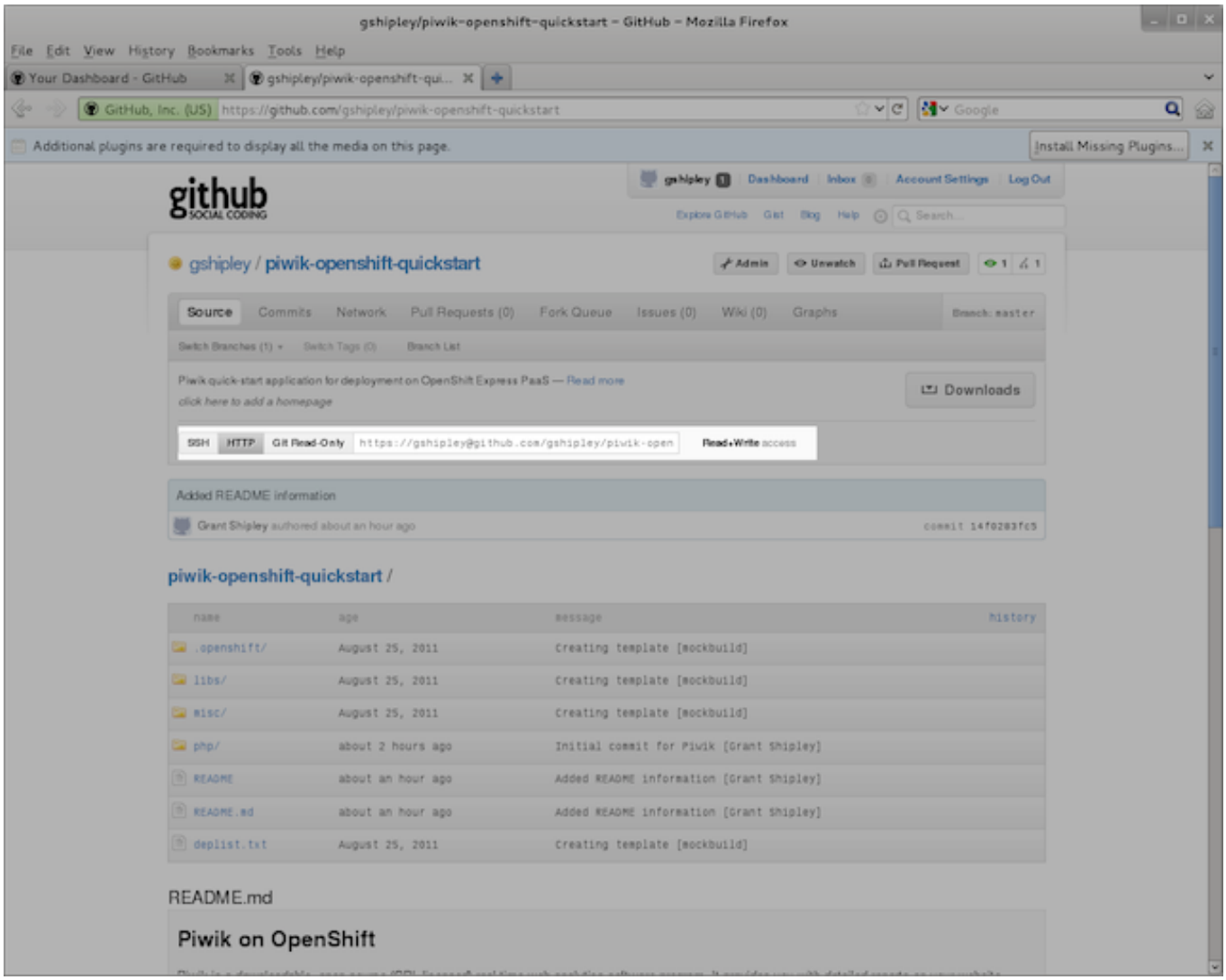
Log in to the github website and create a new repository for our quick start. The direct link, after you are logged in, to create a new repository is:

```
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 software program. It provides you with detailed reports on your website visitors: the search engines and keywords they used, the language they speak, your popular pages, and so much more.
```

Piwik aims to be an open source alternative to Google Analytics, and is already used on more than 150,000 websites.

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 these to complete the Piwik installation on OpenShift

Add this upstream Piwik quickstart repo

```
cd piwik/php
rm -rf *
git remote add upstream -m master git://github.com/gshipley/piwik-openshift-quickstart.git
git pull -s recursive -X theirs upstream master
```

Then push the repo upstream to OpenShift

```
git push
```

That's it, you can now checkout your application at:

```
http://piwik-$yourlogin.rhcloud.com
```

Create the *README* and *README.md* in the *~/code/piwik* directory and add the contents provided above. Once you have created these files, add and commit them to your repository:

```
$ cd ~/code/piwik
$ git add .
$ git commit -am “Add installation instructions”
```

Now we need to push these changes to the github repository we created:

```
$ git push -u github master
```

Verify your quick start works

Delete the *piwik* OpenShift Enterprise application and follow the instruction you created for your Piwik quick start to verify that everything works as expected.

Note: If your application requires an existing populated database, the way to accomplish this is by using the .openshift/action_hooks/build script located in your application directory. Once you have your database created locally, do a *mysqldump* on the table and store the .sql file in the action_hooks directory. You can then modify an existing build file to import the schema on application deployment. For an example, take a look at the action_hooks directory of the Wordpress quick start.

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 lab support website and is called *oo-diagnostics*. For this lab, running the version provided on the support website should suit your needs but when helping customers out, I suggest you pull the script from the official github repository to ensure that you have most updated version. The github script can is located at:

```
https://raw.github.com/openshift/origin-server/master/util/oo-diagnostics
```

This script can be run on any OpenShift Enterprise broker or node host. Once you have the script downloaded, change the permission to enable execution of the script:

```
# chmod +x oo-diagnostics
```

To run the command and check for errors, issue the following command:

```
# ./oo-diagnostics -v
```

Note: Sometimes the script will fail at the first error and not continue processing. In order to run a full check on your node, add the *–abortok* switch

```
# ./oo-diagnostics -v --abortok
```

Under the covers, this script performs a lot of checks on your host as well as executing the existing *oo-accept-broker* and *oo-accept-node* commands on the respective host.

Recovering failed nodes

A node host that fails catastrophically can be recovered if the gear directory /var/lib/openshift has been stored in a fault-tolerant way and can be recovered. In practice this scenario occurs rarely, especially when node hosts are virtual machines in a fault tolerant infrastructure rather than physical machines.

Note: Do not start the MCollective service until you have completed the following steps.

Install a node host with the same hostname and IP address as the one that failed. The hostname DNS A record can be adjusted if the IP address must be different, but note that the application CNAME and database records all point to the hostname, and cannot be easily changed.

Duplicate the old node host's configuration on the new node host, ensuring in particular that the gear profile is the same.

Mount */var/lib/openshift* from the original, failed node host. SELinux contexts must have been stored on the */var/lib/openshift* volume and should be maintained.

Recreate /etc/passwd entries for all the gears using the following steps:

- Get the list of UUIDs from the directories in /var/lib/openshift.
- Get the Unix UID and GID values from the group value of /var/lib/openshift/UUID.
- Create the corresponding entries in /etc/passwd, using another node's /etc/passwd file for reference.

Reboot the new node host to activate all changes, start the gears, and allow MCollective and other services to run.

Removing applications from a node

While trying to add a node to a district, a common error is that the node already has user applications on it. In order to be able to add this node to a district, you will either need to move these applications to another node or delete the applications. In this training class, it is suggested that you simply delete the application as we are working in a single node host configuration. In order to remove a users application, issue the following commands:

```
# oo-admin-ctl-app -l username -a appname -c stop
# oo-admin-ctl-app -l username -a appname -c destroy
```

The above commands will stop the users application and them remove the application from the node. If you want to preserve the application data, you should backup the application first using the snapshot tool that is part of the RHC command line tools.

Lab 32 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 lab support website and is called *oo-diagnostics*. For this lab, running the version provided on the support website should suit your needs but when helping customers out, I suggest you pull the script from the official github repository to ensure that you have most updated version. The github script can is located at:

```
https://raw.github.com/openshift/origin-server/master/util/oo-diagnostics
```

This script can be run on any OpenShift Enterprise broker or node host. Once you have the script downloaded, change the permission to enable execution of the script:

```
# chmod +x oo-diagnostics
```

To run the command and check for errors, issue the following command:

```
# ./oo-diagnostics -v
```

Note: Sometimes the script will fail at the first error and not continue processing. In order to run a full check on your node, add the *–abortok* switch

```
# ./oo-diagnostics -v --abortok
```

Under the covers, this script performs a lot of checks on your host as well as executing the existing *oo-accept-broker* and *oo-accept-node* commands on the respective host.

Recovering failed nodes

A node host that fails catastrophically can be recovered if the gear directory /var/lib/openshift has been stored in a fault-tolerant way and can be recovered. In practice this scenario occurs rarely, especially when node hosts are virtual machines in a fault tolerant infrastructure rather than physical machines.

Note: Do not start the MCollective service until you have completed the following steps.

Install a node host with the same hostname and IP address as the one that failed. The hostname DNS

A record can be adjusted if the IP address must be different, but note that the application CNAME and database records all point to the hostname, and cannot be easily changed.

Duplicate the old node host's configuration on the new node host, ensuring in particular that the gear profile is the same.

Mount */var/lib/openshift* from the original, failed node host. SELinux contexts must have been stored on the */var/lib/openshift* volume and should be maintained.

Recreate /etc/passwd entries for all the gears using the following steps:

- Get the list of UUIDs from the directories in /var/lib/openshift.
- Get the Unix UID and GID values from the group value of /var/lib/openshift/UUID.
- Create the corresponding entries in /etc/passwd, using another node's /etc/passwd file for reference.

Reboot the new node host to activate all changes, start the gears, and allow MCollective and other services to run.

Removing applications from a node

While trying to add a node to a district, a common error is that the node already has user applications on it. In order to be able to add this node to a district, you will either need to move these applications to another node or delete the applications. In this training class, it is suggested that you simply delete the application as we are working in a single node host configuration. In order to remove a users application, issue the following commands:

```
# oo-admin-ctl-app -l username -a appname -c stop
# oo-admin-ctl-app -l username -a appname -c destroy
```

The above commands will stop the users application and them remove the application from the node. If you want to preserve the application data, you should backup the application first using the snapshot tool that is part of the RHC command line tools.