Contents

Lab 1: Overview of OpenShift Enterprise

1.0 Overview of OpenShift Enterprise

1.1 Assumptions

This lab manual assumes that you are attending the Red Hat Summit Lab Session and that you will be using this lab manual in conjunction with the workshop.

It is also assumed 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 session leaders.

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 session leaders know.

1.2 What you can expect to learn from this lab

At the conclusion of this 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, and command line tools.

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 workshop 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.

Lab 2: Installing OpenShift Enterprise

Server used:

- broker host
- node host

Tools used:

- TBD

Lab Environment

Login to Workstation

Workstation User: **lab4** Workstation Password: **lab4**

Start 3 VMs

Use *Virtual Machine Manager (virt-manager)*

VM names: * ose-broker-working * ose-node-working * ose-client-working

Login into VMs

VM's root passwords are **redhat**

The IP details for the VMs: * broker.example.com - *192.169.122.251* * node.example.com - *192.169.122.252* * client.example.com - *192.169.122.253*

Lab 3: Managing districts (Estimated time: 10 minutes)

Server used:

- node host
- broker host

Tools used:

- text editor
- oo-admin-ctl-district

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

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

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

Enabling districts

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

Note: Confirm the following on the broker host.

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

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.

Exit the Mongo shell by using the exit command:

```
> exit
```

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

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

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. Simply 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 3 Complete!

Lab 4: Managing resources

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:

Note: Execute the following on the broker host.

```
# 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:

Note: Execute the following on the broker host.

```
# 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:

Note: Execute the following on the broker host.

```
# 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 system administrator 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.

Note: Execute the following on the broker host.

```
# 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 4 Complete!

Lab 5: Adding Cartridges

Server used:

- node host
- broker host

Tools used:

- text editor
- yum
- lokkit
- chkconfig

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.

Note: Execute the following on the node host.

```
# 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 install the required cartridges:

Note: Execute the following on the node host.

```
# 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.

Note: Execute the following on the node host.

```
# 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.

Note: Execute the following on the node host.

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

Clearing the cartridge cache

If a newly installed cartridge is not immediately available, it may be due to an outdated, cached cartridge list. The first time the REST API is accessed, the broker host uses MCollective to retrieve the list of available cartridges from a node host. By default, this list is cached for six hours in a production environment. If the installed cartridges are modified, the cache must be cleared either manually or by waiting until the cache expires before developers can access the updated list. Use the commands shown below to manually clear the cache on a broker host.

Note: Execute the following on the broker host.

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

Lab 5 Complete!

Lab 6: Installing the RHC client tools

Server used:

- localhost

Tools used:

- ruby
- sudo

- git
- yum
- gem
- rhc

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

OpenShift Enterprise client tools can be installed on any operating system with Ruby 1.8.7 or higher.

Red Hat Enterprise Linux 6.2, 6.3 or 6.4

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

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

```
$ sudo yum install rhc
```

Lab 6 Complete!

Lab 7: Using *rhc setup*

Server used:

- localhost

Tools used:

- rhc

Configuring RHC setup

By default, the RHC command line tool will default to use the publicly hosted OpenShift environment. Since we are using our own enterprise environment, we need to tell *rhc* to use our broker.example.com server instead of openshift.com. In order to accomplish this, run the *rhc setup* command:

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

Once you enter in that command, you will be prompted for the username that you would like to authenticate with. For this workshop, 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. This is required for pushing your source code, via git, up to the OpenShift Enterprise server.

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

```
ose
```

Under the covers

The *rhc setup* tool is a convenient command line utility to ensure that the user's operating system is configured properly to create and manage applications from the command line. After this command has been executed, a *.openshift* directory was

created in the users home directory with some basic configuration items specified in the *express.conf* file. The contents of that file are as follows:

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

# Server API
libra_server = 'broker.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 7 Complete!

Lab 8: Creating a PHP application

Server used:

- localhost
- node host

Tools used:

- rhc

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

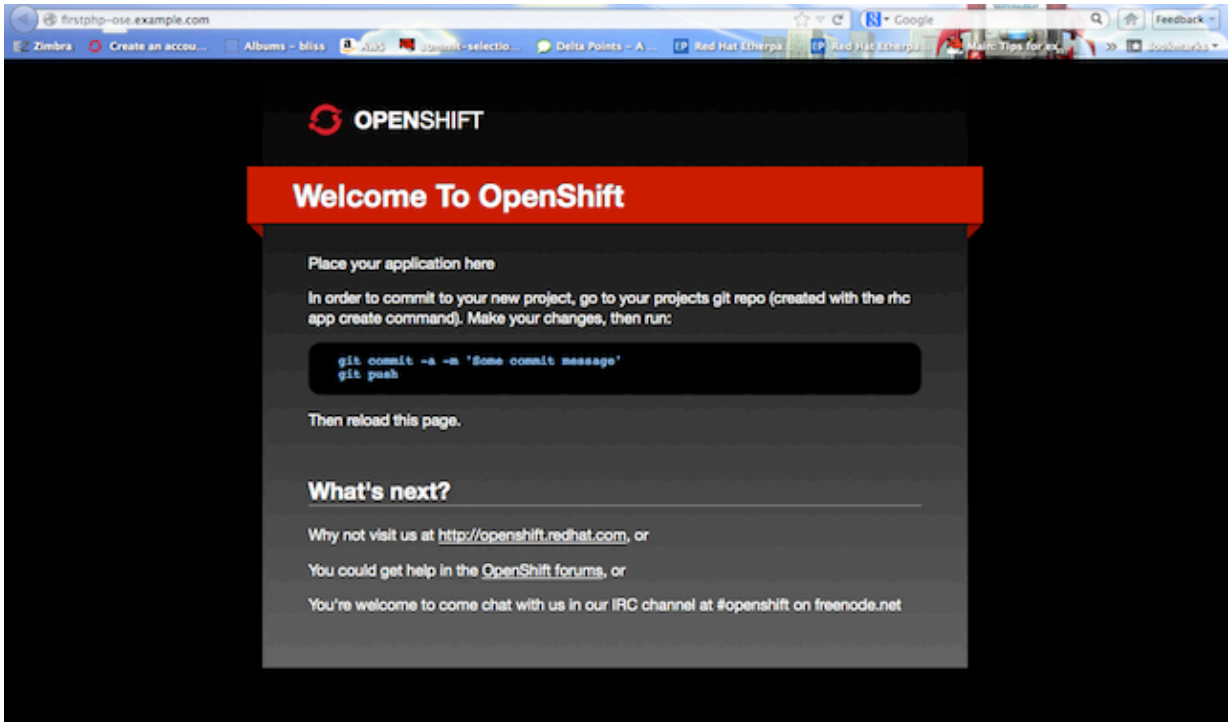
```
Password: ****

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/firstphp.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_keys* file. To verify this, use the following command to view the contents of the file:

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

You will also notice the following three directories:

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

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

For a more human readable format, you can execute the following command on the node host:

```
$ cd /var/lib/openshift/.httpd.d
$ ls
```

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

- daily
- hourly
- minutely
- monthly
- weekly

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

libs directory

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

misc directory

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

php directory

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

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

Make a change to the PHP application and deploy updated code

To get a good understanding of the development workflow for a user, let's change the contents of the *index.php* template that is provided on the newly created gear.

```
$ cd ~/ose/firstphp/php
```

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.”
```

If you see an error message stating that you need to set global.{username,email}, you can ignore the 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
```

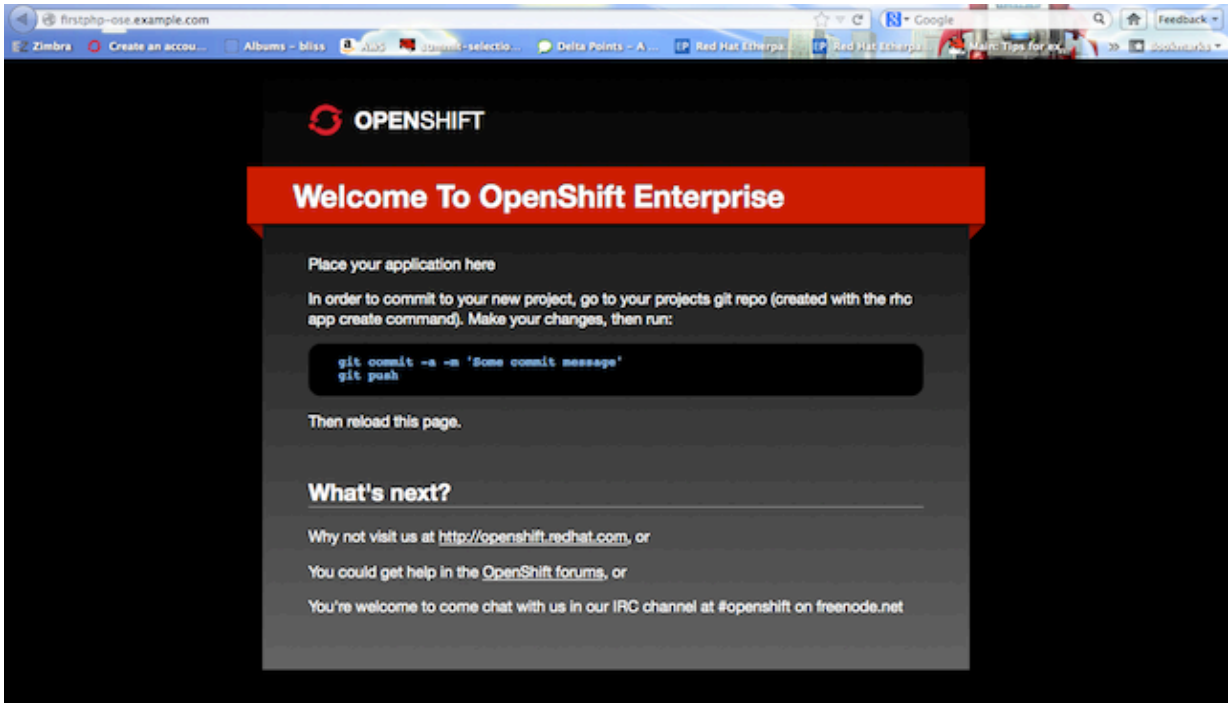
Notice that we stop the application runtime (Apache), deploy the code, and then run any action hooks that may have been specified in the .openshift directory.

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:

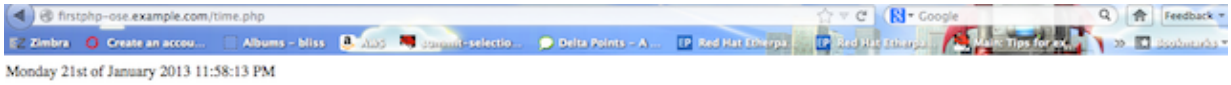
```
$ git add .
$ git commit -am “Adding time.php”
$ git push
```

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.



Lab 8 Complete!

Lab 9: Scaling an application

Server used:

- localhost
- node host

Tools used:

- rhc
- ssh

- git
- touch
- pwd

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

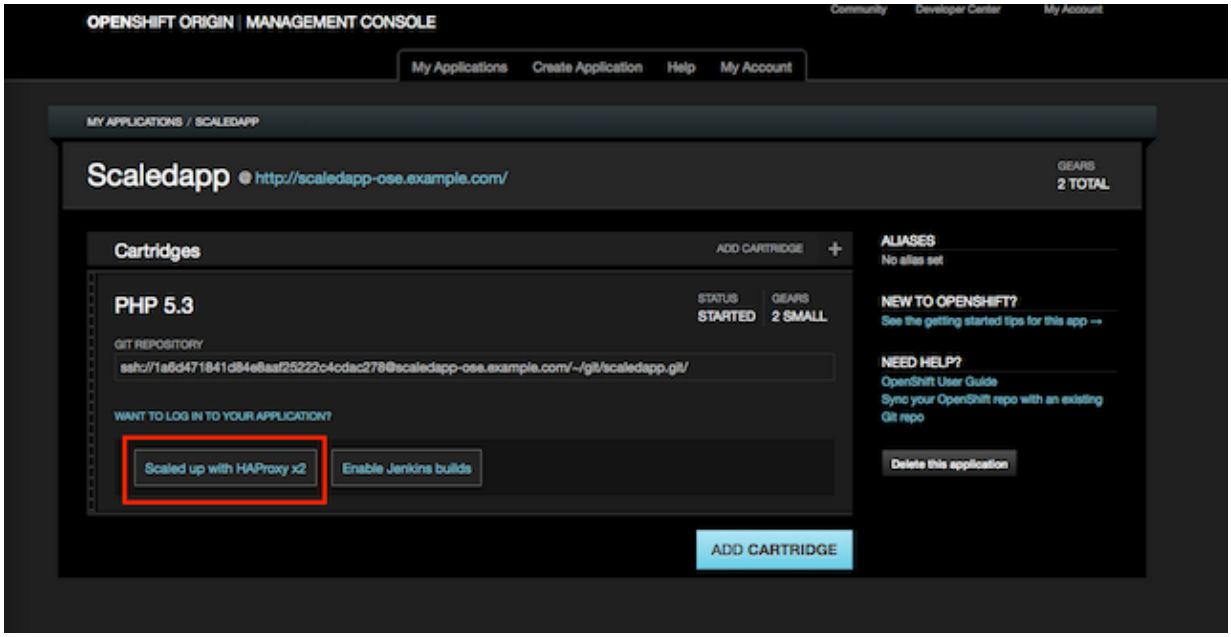
In order to create a scaled application using the *rhc* command line tools, you need to specify the *-s* switch to the command. Let's create a scaled PHP application with the following command:

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

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

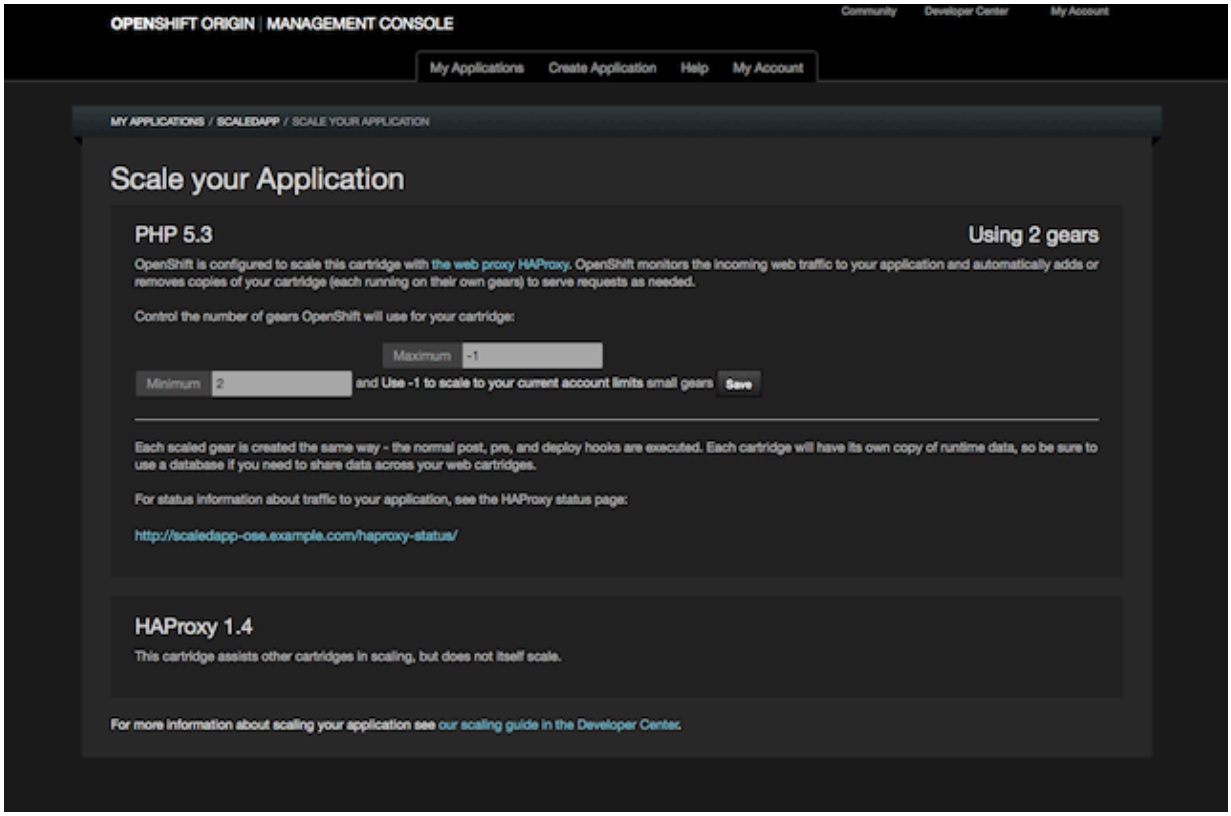
```
Password: ****

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.
```



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 -am “remove automatic scaling”
$ git push
```

To add a new gear to your application, SSH to your application gear with the following command replacing the contents with the correct information for your application.

Note: To get a list of all your application that you can ssh into, you can run the *rhc domain show* command.

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

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

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

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

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

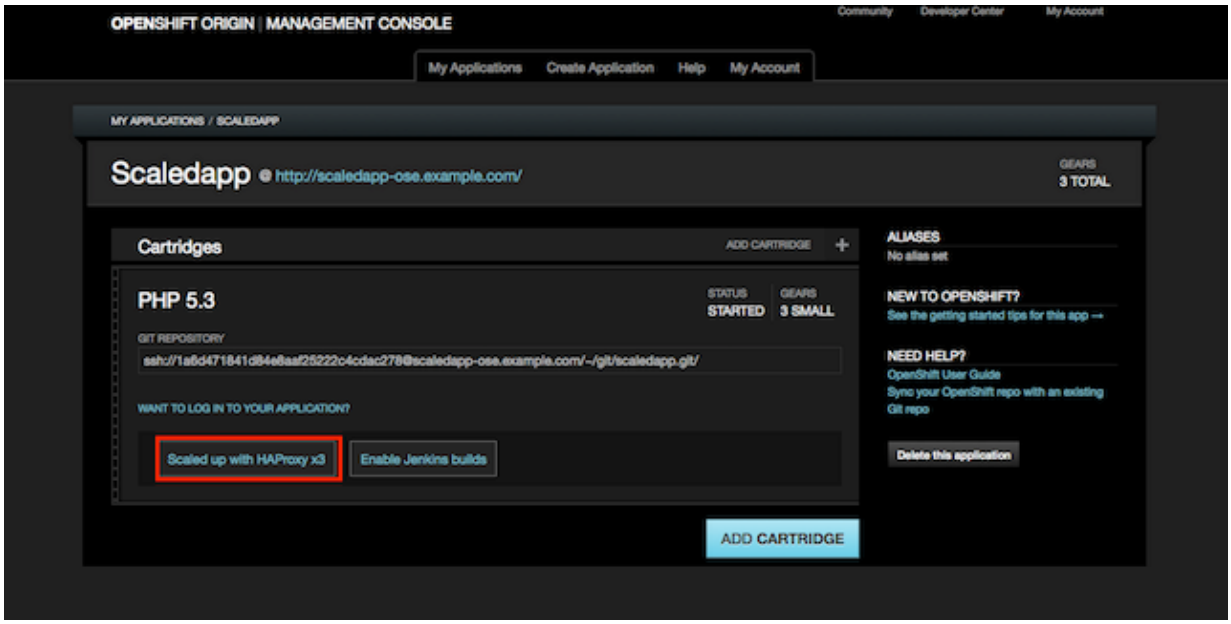
Verify that your new gear was added to the application by running the *rhc app show* command or by looking at the application details on the web console:

```
$ 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/scaledapp.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:

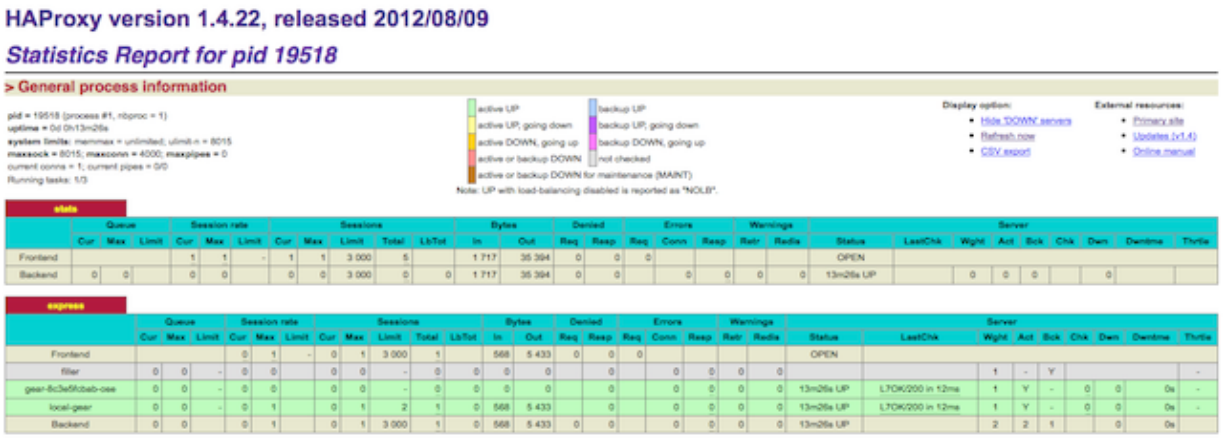
```
$  rhc app show scaledapp

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/scaledapp.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/
```



Lab 9 Complete!

Lab 10: Managing an application

Server used:

- rhc client host
- 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_check.php

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

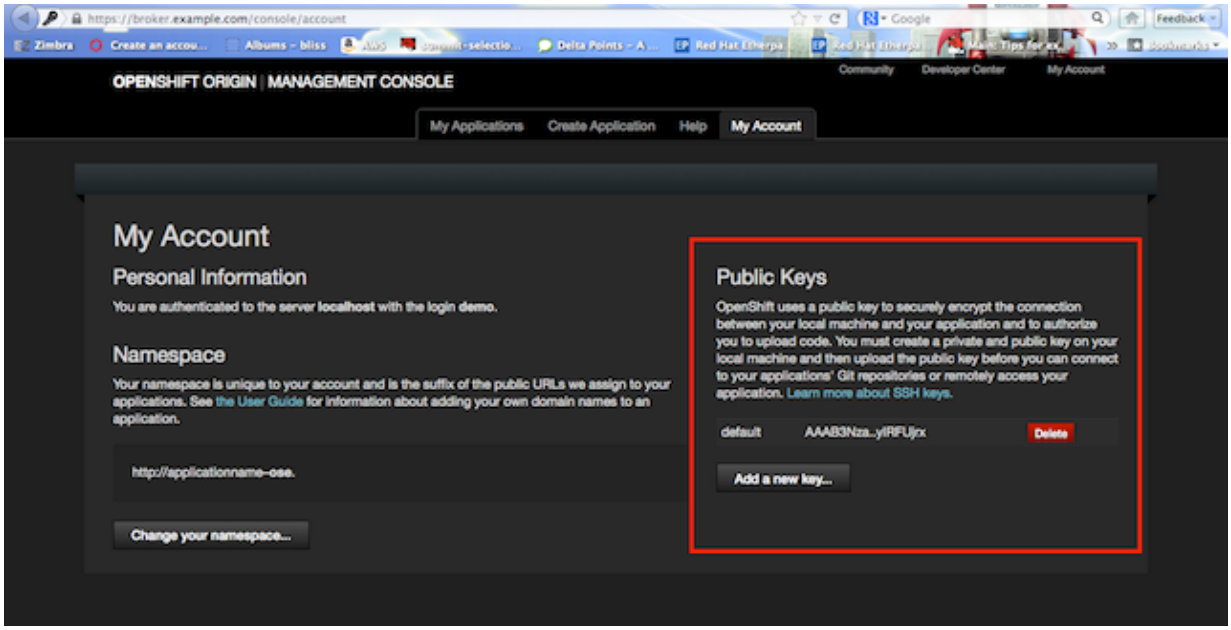
```
$ rhc app tidy -a firstphp
```

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/firstphp.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

You can view the quota of your currently running gear by connecting to the gear node host via SSH as discussed previously in this lab. Once you are connected to your application gear, enter the following command:

```
$ quota -s
```

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

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

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

```
$ du -h
```

Adding a custom domain to an application

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.

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.

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
```

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 see the details of the thread dump. Issue the following command, substituting the correct log file:

```
$ rhc tail ApplicationName -f ruby-1.9/logs/error_log-20130104-000000-EST -o '-n 250'
```

Lab 10 Complete!