OpenShift Enterprise Administration Training - Lab Manual

Author: Grant Shipley

Revision: 1.0

1.0 An 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, yum, and familiarity with a linux based text editor. 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 An 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 An 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.

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.

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 the 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.onopenshift.com
```

Lab 1: Register and update your Operating system (Estimated time: xx minutes)

Server Used:

Broker host

Tools used

- SSH
- subscription-manager
- ntpdate
- yum

Register system and apply subscription

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 subscription 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
```

Update operating system to 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
```

Depending on the network connectivity at this location, this update may take several minutes.

Configuration of 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 as is a common problem that users encounter when they fail to sync all of the system clocks.

```
# ntpdate clock.redhat.com
```

Lab 1 Complete!

Lab 2: Installation and configuration of DNS (Estimated time: xx minutes)

Server Used:

Broker host

Tools used

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

Install DNS Server

In order for OpenShift Enterprise to work correctly, you will need 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 servers 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
```

Create environment variables and DNSSEC key file

The official OpenShift documentation suggests that you set an environment variable for the domain name that you will be using which allow 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. In order to implement DNSSEC on our OpenShift Enterprise broker node, we need to create a key file.

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

Create a DNSSEC key pair and store the 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
```

Create fowards.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:

```
# 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
```

Configure subdomain resolution and create database

To ensure that we are starting with a clean */var/named/dynamic* directory, lets remove this directory if it exists.

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

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

```
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
```

Create our named configuration file

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

```
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
```

Configure host name resolution to use new bind server

We need to update our resolve.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:

```
nameserver 127.0.0.1
```

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

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

Start *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 ensure 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.

Add entries using nsupdate

Now that our bind server is configured and started, we need to add our broker node to our DNS entries. To accomplish this task, we will use the nsupdate command, which opens an interactive shell where we can perform commands.

```
# 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: Configure dhclient-eth0.conf and setting the hostname (Estimated time: xx minutes)

Server Used:

Broker host

Tools used

- text editor
- Commands: hostname

Create 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*. Without this step, the DNS server information in resolve.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 node. 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 but using the following command:

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

Lab 3 Complete!

Lab 4: Installation and Configuration of MongoDB (Estimated time: xx 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 had 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 this 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 package 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

Configure *mongod* server

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:

```
auth=true
```

By default, this line is commented out so just remove the comment. We also need to enable smallfiles, so add the following line.

```
smallfiles=true
```

Configure *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 and we also need to ensure the database is started for our current use.

```
# 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 are 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: Installation and Configuration of ActiveMQ (Estimated time: xx 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 hosts 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.

Install 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 is 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.

Configure ActiveMQ

ActiveMQ uses an xml configuration file that is located at */etc/activemq/activemq.xml*. Instead of creating a new configuration file from scratch, I suggest that you download a sample configuration file and make any necessary changes to the sample file.

```
# cd /etc/activemq
# mv activemq.xml activemq.orig
# wget https://mirror.openshift.com/pub/enterprise-server/scripts/1.0/activemq.xml
```

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:

```
<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}">
```

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>
```

##**Modify firewall and configure 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
```

##**Verify 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
```

title

Note: The above configuration required no authentication for accessing the activemq console. 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: Installation and Configuration of the MCollective client (Estimated time: xx 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.

##**Installation of MCollective client**

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

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

##**Configuration of MCollective client**

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

```
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
```

Note: Provide the correct information for your installation for the host, password, etc.

Lab 6 Complete!

Lab 7: Installation and Configuration of the Broker Application (Estimated time: xx minutes)

Server Used:

Broker host

Tools used

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

Install 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 server, this installation make take several minutes.

Modify 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.

Configure start on boot and firewall for services

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 are accepted:

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

Generate 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
# 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.

Configure SELinux Variables and set proper contexts

SELinux has several variables that we want to ensure is 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_stick
shift=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: xx 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

Create conf 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
```

Configure 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 "3RH8tLp6fvX4RVV9ny2lm0tZpTjXhB62ieC6CN1Fh/2468Z1+6lX4wpCJ6sfYH6u2+//gbDDStDX+aPMtSiNFw==";
};
```

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:

```
# 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="3RH8tLp6fvX4RVV9ny2lm0tZpTjXhB62ieC6CN1Fh/2468Z1+6lX4wpCJ6sfYH6u2+//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/de
vel/Makefile ; popd
# semodule -i /usr/share/selinux/packages/rubygem-openshift-origin-dns-bind/dhcpnamedforward.pp
```

Configure Authentication

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/ww
w/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
```

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/
```

Add 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")
}
```

Configure 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

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: xx 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 nodes.

Install 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 76 additional packages.

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

Configure authentication for the console

In a preview 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 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
# mv 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 on
# 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 server in a later lab.

Lab 9 Complete!

Lab 10: Configure DNS resolution for node host (Estimated time: xx minutes)

Server Used:

node host broker host

Tools used

- text editor
- yum
- 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*.

Update node host with subscriptions and latest packages

During the training class, you were provided with credentials for two servers, a broker host and a node host. This lab begin 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
```

Configure broker application host

SSH to your broker application node 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.

Configure 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:

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

One you enter that command, you will be prompted to authenticate to the node system.

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:

```
# 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.

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

Configure node host for DNS resolution

We need to configure the node host to use the bind server that we have installed and configured on the broker application server. 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 server. Edit this file and the following line making sure to use the correct IP address of your broker application server.

```
nameserver 10.10.10.10
```

Configure nameserver placement 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 resolve.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.cfg 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: xx 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 from 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.

Install MCollective on 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
```

Configure MCollective on 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.

```
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
```

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: Installation and configuration of core OpenShift Enterprise node packages (Estimated time: xx 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.

Install 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 opensh
ift-origin-node-util
```

Install 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, lets also assume that we want to only allow scaleable 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-origi
n-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 httpd, 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: Configure PAM namespace, linux control groups (cgroups), and user quotas (Estimated time: xx minutes)

Server Used:

node host

Tools used

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

Configure PAM to use the OpenShift 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:

```
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;
```

Configure Linux Control Groups (cgroups)

cgroups allow 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.

- cgconfig
- cgcred
- openshift-cgroups

To verify that your cgroup configuration is correct, lets 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 euqal 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
```

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: Configure SELinux and System Control Settings (Estimated time: xx minutes)

Server Used:

node host

Tools used

- text editor
- setbool
- fixfiles
- restorecon
- sysctl

Configure 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 command:

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

Configure System Control Settings

We need to modify the */etc/sysctl.conf* configuration file to increase the number of kernel semaphores (allows many httpds 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: Configure SSH, OpenShift Port Proxy, and node configuration (Estimated time: xx minutes)

Server Used:

node host

Tools used

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

Configure SSH to pass 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
```

Configure Proxy for node

Multiple application gears can and will reside on the same node host. In order for these application 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
```

Configure 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 server for REST API
```

Update 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
```

Reboot 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.

Lab 15 Complete!

Appendix A - Installation of Red Hat Enterprise Linux

Appendix B - RHC command line reference